1. Что такое машинное обучение? Кто такой Data Scientist? Как машинное обучение и наука о данных связаны с искусственным интеллектом?
2. Уровни развития искусственного интеллекта (слабый, сильный, ANI, AGI, ASI).
3. История развития ИИ, МО и глубокого обучения.
4. В каких областях применяется машинное обучение? Приведите примеры решения прикладных задач с помощью МО.
5. Постановка задачи обучения на примерах.
6. Описание объектов и ответов. Типы задач машинного обучения.
7. Обучение с учителем, предсказательные модели. Приведите не менее 4-х примеров описания прикладных задач.
8. Алгоритм обучения. Сведение задачи обучения к задаче оптимизации.
9. Оценивание моделей. Эмпирический риск и функция потерь.
10. Что такое переобучение (overfitting) и недообучение (underfitting)? Как их можно избежать?
11. Одномерная и многомерная линейная регрессия.
12. Конструирование признаков.
13. Нормализация признаков.
14. Метод наименьших квадратов.
15. Алгоритм градиентного спуска.
16. Стохастический градиентный спуск.
17. Варианты инициализации весов и выбора скорости обучения в алгоритме градиентного спуска.
18. Использование регуляризации для борьбы с переобучением в алгоритме градиентного спуска.
19. Повышение производительности с помощью векторизации.
20. Постановка задачи бинарной и многоклассовой классификации.
21. Классификаторы на основе разделяющей поверхности (margin-based).
22. Логистическая регрессия.
23. Принцип максимизации правдоподобия, его связь с эмпирическим риском.
24. L1- и L2-регуляризация. Вероятностный смысл регуляризации.
25. Понятие расстояния между объектами. Метрика Минковского.
26. Обобщенный метрический классификатор. Метод k ближайших соседей.
27. Метод k взвешенных ближайших соседей. Метод окна Парзена.
28. Оптимальная разделяющая гиперплоскость, ее геометрическая интерпретация.
29. Применение условий Каруша-Куна-Такера к задаче построения оптимальной разделяющей гиперплоскости
30. Понятие опорного вектора и типизация объектов.
31. Нелинейное обобщение метода опорных векторов с помощью функции ядра. Виды ядер.
32. Интерпретируемость алгоритмов машинного обучения.
33. Деревья принятия решений. Определение, алгоритмы построения.
34. Критерий Джинни, энтропийный критерий.

35. Проблема переобучения деревьев принятия решений. Регулирование глубины дерева (обрезка ветвей).
36. Объясните понятие ошибок первого и второго рода, их связь с машинным обучением.
37. Объясните понятия accuracy, полноты (recall), точности (precision) и F1-меры.
38. Кривые ROC и Precision-Recall, площадь под ними.
39. Метрики оценки качества регрессии.
40. Задача кластеризации. Типы кластерных структур, чувствительность к выбору признаков.
41. Задача частичного обучения.
42. Оценка качества решения задачи кластеризации.
43. Метод k-средних.
44. Алгоритм DBSCAN.
45. Иерархическая кластеризация.
46. Карты Кохонена.
47. Ансамбль моделей машинного обучения.
48. Методы стохастического ансамблирования.
49. Случайный лес.
50. Отличие между бэггингом и бустингом.
51. Алгоритм AdaBoost.
52. Градиентный бустинг.
53. Алгоритмы CatBoost, XGBoost.
54. Работа с большими данным. Экосистема Apache Hadoop.
55. Файловая система HDFS.
56. Алгоритм Map-Reduce.
57. Apache Spark и распределенные наборы данных (RDD).
58. Принципы работы рекомендательных систем.
59. Коллаборативная и контентная фильтрация.
60. Техники коллаборативной фильтрации: memory-based и model-based.
61. Корреляционные модели в коллаборативной фильтрации. Непараметрическая регрессия, функции сходства.
62. Понятие латентной модели.
63. Матричные разложения. Сингулярное разложение.
64. Измерение качества рекомендаций.
65. Понятие искусственного нейрона.
66. Сеть Хопфилда.
67. Понятие стохастического нейрона. Машина Больцмана.
68. Алгоритм имитации отжига.
69. Ограниченная машина Больцмана. Алгоритм контрастного расхождения (contrastive divergence).
70. Глубокие сети на основе машины Больцмана.

35. Проблема переобучения деревьев принятия решений. Регулирование глубины дерева (обрезка ветвей).

Переобучение (overfitting) деревьев принятия решений представляет собой ситуацию, при которой построенная модель слишком точно отражает особенности обучающей выборки, включая шум и случайные выбросы, в ущерб обобщающей способности. Такое дерево, как правило, имеет большую глубину, множество ветвей и разделяет данные до уровня полной чистоты узлов, что не требуется, для сохранения обобщающей способности.

Для борьбы с переобучением применяется регулирование сложности дерева, одним из основных методов которого является обрезка дерева (pruning). Обрезка направлена на удаление ненужных или малозначимых ветвей, которые не дают ощутимого вклада в точность модели на тестовых данных.

Предварительная обрезка (pre-pruning). Она включает в себя установку ограничений на процесс построения дерева, таких как: максимальная глубина дерева, минимальное количество объектов в узле для дальнейшего разбиения, минимальное уменьшение критерия чистоты (значение критерия Джини или Энтропийного критерия) для выполнения разбиения. Эти параметры ограничивают рост дерева на этапе построения, предотвращая появление переобобщающих ветвей. Однако при слишком агрессивной предварительной обрезке возможна потеря информации, что приводит к недообучению.

Построенная обрезка (post-pruning). Она применяется после построения полного дерева и заключается в удалении или упрощении уже существующих ветвей. Основные методы post-pruning: сокращение ветвей, если они не улучшают метрику качества на тестовой выборке, обрезка на основе оценки статистической значимости разбиений и критериев чистоты. Построенная обрезка считается более точным методом, поскольку она основана на оценке

уже построенной структуры, однако требует дополнительного этапа валидации и усложняет вычисления.

36. Объясните понятие ошибок первого и второго рода, их связь с машинным обучением.

Рассмотрим понятие ошибки первого и второго рода в контексте задачи бинарной классификации. Существует два класса, Положительный (True) и Отрицательный (False) классы. Также существует две гипотезы, принадлежность отрицательному и положительному классу, соответственно гипотезы 0 и 1.

Ошибка первого рода - это ложноположительное решение. Она возникает в ситуации, когда модель отвергает нулевую гипотезу, хотя на самом деле она истинна. В контексте бинарной классификации это означает, что модель предсказала положительный результат, хотя истинная метка отрицательная. В классификационных задачах эта ошибка связана с метрикой False Positive (FP) — количество отрицательных примеров, ошибочно классифицированных как положительные.

Ошибка второго рода — это ложноотрицательное решение. Она возникает, когда модель не отвергает нулевую гипотезу, хотя она ложна. В терминах бинарной классификации это означает, что объект ошибочно отнесен к отрицательному классу, в то время как он должен быть классифицирован как положительный. Соответствует метрике False Negative (FN) — количество положительных объектов, ошибочно классифицированных как отрицательные.

В машинном обучении, на основе ошибок первого и второго рода, формируется матрица ошибок (confusion matrix), которая отражает соотношения между предсказанными и истинными метками классов:

		PREDICTED	
		Positive	Negative
ACTUAL	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

43. Метод k-средних.

Цель метода — минимизировать внутрикластерную дисперсию, то есть сумму квадратов расстояний между объектами и центрами соответствующих кластеров. Алгоритм относится к методам жёсткой кластеризации: каждый объект однозначно относится к одному кластеру.

Минимизация суммы квадратов внутрикластерных расстояний:

$$\sum_{i=1}^{\ell} \|x_i - \mu_{a_i}\|^2 \to \min_{\{a_i\},\{\mu_a\}}, \quad \|x_i - \mu_a\|^2 = \sum_{j=1}^{n} \left(f_j(x_i) - \mu_{aj}\right)^2$$

Алгоритм метода k-средних (алгоритм Лойда):

1. Случайным образом выбираются k начальных центров кластеров $\mu_1, ..., \mu_k$.
2. Каждый объект x_i назначается тому кластеру, чей центр ближе всего.
3. Центры кластеров пересчитываются как средние значения объектов, вошедших в кластер.
4. Повторение шагов 2–3 до сходимости: алгоритм считается сошедшимся, если центры масс не изменяются или изменения становятся меньше заданного порога.

Алгоритм гарантированно сходится за конечное число шагов, так как функция потерь (сумма квадратов расстояний) не увеличивается при каждом обновлении. Однако он может сходиться к локальному минимуму, что делает важным выбор начальных центров.

Изменения в алгоритме Ллойда для частичного обучения:

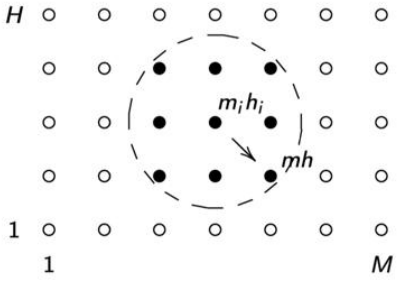
1. Если известны метки части объектов, можно использовать их для инициализации соответствующих центров кластеров: Вычислить центры кластеров по размеченным данным.
2. Размеченные объекты не переклассифицируются — их принадлежность к кластерам фиксирована в соответствии с их метками, а неразмеченные объекты назначаются в кластеры на основе минимального расстояния до центров.
3. Центры пересчитываются по всем объектам, иногда добавляется вес для уже размеченных объектов, чтобы сделать их более значимыми.
4. Повторяются шаги 2 и 3 до сходимости.

46. Карты Кохонена.

Карты Кохонена (Self-Organizing Maps, SOM) — это тип искусственной нейронной сети для обучения без учителя, которая создает упорядоченное представление многомерных данных на низкоразмерной сетке.

Изначально задаётся прямоугольная сеть нейронов-кластеров вида $Y = \{1,...,M\} \quad \{1,...,H\}$, где M и H – количество нейронов в сетке. Каждому узлу (m, h) задаётся вектор весов $\theta_{mh} \quad ^{n}$, размерность весов соответствует размерности входных данных. На сетке определяется евклидово расстояние между узлами: $r((m_i, h_i),(m,h)) = \sqrt{(m-m_i)^2 + (h-h_i)^2}$ - расстояние между i-ым нейроном и любым другим в сетке Кохонена.

Окрестность(m_i, h_i):



Алгоритм построения карты Кохонена:

1. $\theta_{mh} := random\ -\frac{1}{2MH}, \frac{1}{2MH}$ - инициализация весов для каждого нейрона.
2. Для каждого входного вектора x_i вычисляется расстояние от него до всех нейронов в сетке, выбирается нейрон с минимальным расстоянием и этот нейрон становится нейроном-победителем: $(m_i, h_i) := g(x_i) = \arg\min r(x_i, \theta_{mh})$ где r — функция расстояния между входом и весами нейрона, arg min — "аргумент минимума" (координаты, при которых достигается минимум), (m,h)∈Y — перебираем все нейроны в сетке, (m_i, h_i) — координаты нейрона-победителя. (Winner Take All или WTA).
3. Обновляются веса не только победителя, но и его соседей: для всех (m,h) ∈ Окрестность (m_i, h_i): $\theta_{mh} := \theta_{mh} + \eta(x_i - \theta_{mh})K(r(m_i, h_i), (m, h))$, где θ_{mh} — веса нейрона с координатами (m, h), η — темп обучения (learning rate), (x_i - θ_{mh}) — вектор ошибки, K(...) — функция соседства (функция, которая определяет силу обновления весов в зависимости от значения расстояния, обычно используется гауссова функция, которая со временем сужается, что позволяет сначала формировать общую структуру, а затем уточнять детали), r(...) — расстояние от нейрона до победителя.
4. Пункты 2 и 3 повторяются, пока кластеризация не стабилизируется.

Сеть автоматически организует нейроны так, что похожие входные данные активируют близко расположенные нейроны. Это создает топологическое сохранение — близкие в исходном пространстве данные остаются близкими на карте.

Существует 2 способа визуализации карт Кохонена:

1. Карта плотности: в данном случае цвет узла (m, h) показывает как часто нейрон становился победителем. Темные области — много данных попадает в эти нейроны, Светлые области — мало данных. Данный способ визуализации показывает основные кластеры в данных.

2. Компонентные карты: Цвет узла (m, h) = значение j-й компоненты вектора весов θ_{mh} Каждая карта показывает, как один конкретный признак распределен по сетке, разные цвета - разные значения признака.

Достоинства:

- Возможность визуального анализа многомерных данных
- Квантование выборки по кластерам,
  с автоматическим определением числа непустых кластеров

Недостатки:

- **Субъективность.** Карта отражает не только кластерную структуру данных, но также зависит от...
  — свойств сглаживающего ядра;
  — (случайной) инициализации;
  — (случайного) выбора x_i в ходе итераций.
- **Искажения.** Близкие объекты исходного пространства могут переходить в далёкие точки на карте, и наоборот.

Рекомендуется только для разведочного анализа данных.

47. Ансамбль моделей машинного обучения.

Ансамбль моделей машинного обучения представляет собой методологию, при которой несколько отдельных моделей (так называемых базовых алгоритмов) объединяются в единую систему с целью повышения общей точности и устойчивости предсказаний. Основная идея ансамблевого подхода заключается в том, что слабые модели, допускающие ошибки на различных подмножествах данных, при объединении в ансамбль способны компенсировать ошибки друг друга, формируя более надёжный и точный итоговый предсказатель. Это особенно актуально в случае нестабильных алгоритмов, чувствительных к колебаниям в данных, таких как деревья решений.

Обучающая выборка: $X^l = (x_i, y_i)$ - набор данных из пространства X×Y, где y_i - истинные ответы. Базовые алгоритмы: $g_t : X \to Y$, t = 1, …, T.

Вначале происходит декомпозиция базовых алгоритмов: $g_t(x) = C(b_t(x))$, где b_t - алгоритмичные операторы, которые преобразуют входные данные в промежуточное представление, C - решающее правило, которое принимает финальное решение.

В общем виде задача ансамбля описывается в следующем виде: $g_t : X \xrightarrow{b_t} R \xrightarrow{C} Y$, где R – пространство оценок. Ансамбль принимает решение на основе оценок, которые дали более слабые алгоритмы, применяя решающее правило.

$$g(x) = C\left(F\left(b_1(x),...,b_T(x)\right)\right)$$

где F – агрегирующая (корректирующая) функция.

Общие требования к агрегирующей функции:

- $F(b_1, \ldots, b_T, x) \in \left[\min_t b_t, \max_t b_t\right]$ — среднее по Коши $\forall x$
- $F(b_1, \ldots, b_T, x)$ монотонно не убывает по всем b_t

Примеры агрегирующих функций:

- простое голосование (simple voting):
  $F(b_1, \ldots, b_T) = \frac{1}{T} \sum_{t=1}^{T} b_t$
- взвешенное голосование (weighted voting):
  $F(b_1, \ldots, b_T) = \sum_{t=1}^{T} \alpha_t b_t, \quad \sum_{t=1}^{T} \alpha_t = 1, \quad \alpha_t \geqslant 0$
- смесь алгоритмов (mixture of experts)
  с функциями компетентности (gating function) $G_t: X \to \mathbb{R}$
  $F(b_1, \ldots, b_T, x) = \sum_{t=1}^{T} G_t(x) b_t(x)$

48. Методы стохастического ансамблирования.

Методы стохастического ансамблирования представляют собой класс подходов в машинном обучении, при которых разнообразие в ансамбле

достигается за счёт введения случайности в процесс обучения отдельных моделей. Основная идея стохастического ансамблирования заключается в том, что различные варианты исходных данных или их представления позволяют обучить модели, совершающие разные ошибки, и при объединении этих моделей можно получить предсказатель с более высокой обобщающей способностью, чем любая отдельная модель.

1. Bagging (Bootstrap Aggregating). Bagging использует бутстрэп-подвыборки обучающих данных, сформированные случайной выборкой с возвращением (возможны повторы объектов в одной подвыборке). Каждая модель обучается на своей случайной подвыборке. Совокупность таких моделей даёт устойчивый и точный результат. В задачах классификации итоговый ответ принимается путём голосования, в регрессии — усреднением.
2. Pasting. Pasting аналогичен bagging, за исключением одного ключевого различия: подвыборки формируются без возвращения, то есть каждый объект обучающей выборки включается в подвыборку только один раз.
3. Random Subspaces. Метод случайных подпространств предполагает, что каждая модель обучается на всех объектах, но на случайном подмножестве признаков. Таким образом, разнообразие ансамбля обеспечивается за счёт того, что разные модели оперируют с разными проекциями исходного признакового пространства.
4. Random Patches. Метод случайных фрагментов (random patches) объединяет идеи bagging/pasting и random subspaces: каждая модель обучается на случайной подвыборке объектов и признаков одновременно.
5. Cross-Validated Committees Метод комитетов на основе кросс-валидации (cross-validated committees) строится по следующей схеме: проводится k-кратная кросс-валидация, в ходе которой обучаются k моделей на различных разбиениях данных. После завершения обучения модели объединяются в ансамбль, и их предсказания агрегируются (усредняются или выбирается наиболее вероятный класс).

В качестве примера организации ансамблевого алгоритма реализуем bagging + random subspaces.

X^l — обучающая выборка, T — количество базовых алгоритмов в ансамбле, ℓ' — объём обучающих подвыборок (для bagging), n' — размерность признаковых подпространств, ε_1 — порог качества базовых алгоритмов на обучении, ε_2 — порог качества базовых алгоритмов на контроле. На выходе ожидаем получить базовые алгоритмы $b_t, t = 1,...,T$.

1. Определяется случайная подвыборка U_t размерности l' из X^l для bugging;
2. G_t случайное подмножество мощности n' для random subspaces;
3. $b_t = \mu(G_t, U_t)$ - обучение базового алгоритма на подвыборке U_t с использованием только признаков из G_t;
4. Если $Q(b_t, U_t) > \varepsilon_1$, то не включать b_t в ансамбль (алгоритм не прошёл порог качества на этапе обучения), если $Q(b_t, X^l \setminus U_t) > \varepsilon_2$, то не включать b_t в ансамбль (алгоритм не прошёл порог качества на этапе тестирования);
5. Шаги 1-4 повторяются, пока не сформируются T алгоритмов.

51. Алгоритм AdaBoost.

Основная идея алгоритма заключается в построении композиции слабых моделей (чаще всего простых решающих деревьев глубины 1 — т.н. решающих пней), каждая из которых обучается на данных с адаптивно скорректированными весами объектов. На каждом шаге последующая модель сосредотачивается на тех наблюдениях, которые были ошибочно классифицированы предыдущими моделями, тем самым постепенно улучшая точность всей системы.

На вход подаётся обучающая выборка X^l и параметр T.

1. Каждому объекту обучающей выборки присваивается вес $w_i := \frac{1}{l}$, это означает, что изначально все объекты одинаково важны для обучения.

2. На каждой итерации алгоритма находим слабый классификатор $b_t = \arg\min N(b;W^l)$ N - Взвешенная ошибка - ошибка классификации, где каждый неправильно классифицированный объект вносит вклад пропорционально своему весу. $N(b;W^l) = \sum_{i=1}^{l} w_i * L[y_i \quad b(x_i)]$.
3. Вычисление коэффициента важности данного классификатора $\alpha_t := \frac{1}{2}\ln\frac{1-N(b_t;W^l)}{N(b_t;W^l)}$ Этот коэффициент показывает, насколько "хорош" текущий классификатор: если $N(b_t;W^l) \to 0 \quad \alpha_t \to +$ (очень хороший алгоритм), $N(b_t;W^l) \to 0,5 \quad \alpha_t \to 0$ (случайное угадывание), $N(b_t;W^l) \to 1 \quad \alpha_t \to -$ (не правильный классификатор).
4. Далее происходит обновление весов объектов $w_i = w_i \exp\left(-\alpha_t y_i b_t\left(x_i\right)\right)$. Если объект классифицирован правильно, то вес объекта уменьшается, если неправильно – увеличивается.
5. Нормализация весов, для того, чтобы в сумме веса давали единицу $w_i = \frac{w_i}{\sum_{j=1}^{l} w_j}$
6. Пункты 2-5 повторяются, пока не получится T различных классификаторов.
7. На выходе получится T алгоритмов с их весами $\alpha_t b_t$.

Итоговый сильный классификатор строится как взвешенное голосование:

$H\left(x\right) = sign \sum_{t=1}^{T} \alpha_t b_t(x)$.

Презентация по статье о глубоком остаточном обучении (ResNet)
Deep Residual Learning for Image Recognition

1. О чём статья?

Эта статья рассказывает о лучшем способе обучения очень глубоких нейронных сетей для распознавания изображений. В ней представлена методика остаточного обучения (Residual Learning), которая решает основные проблемы, возникающие при увеличении глубины сетей.

Как предыдущие модели, такие как VGG-16, справлялись с глубиной?

Предыдущие модели, такие как VGG-16 и VGG-19 использавали только 3×3 сверточные фильтры на всех слоях.

- Если размер карты признаков остается прежним на следующем слое, количество фильтров тоже остаётся прежним.

-Если же размер карты признаков уменьшается, количество фильтров увеличивается для балансировки вычислений.

Несмотря на свою мощность, эти модели требуют очень больших вычислительных ресурсов и всё равно сталкиваются с проблемой ухудшения качества при глубине больше 19 слоев.

Простые сети с глубиной более 30 слоев показывали увеличение ошибки на обучении, что говорит о трудностях оптимизации, несмотря на использование batch normalization, предотвращающего затухание градиентов.

2. Что такое глубокие сверточные нейронные сети (CNN)?

CNN — это тип нейронных сетей, которые особенно хорошо подходят для распознавания изображений.

Они работают путем последовательного наложения слоев, которые выявляют простые паттерны — края, текстуры — и затем более сложные объекты.

Большее количество слоев позволяет изучать более сложные и абстрактные признаки.

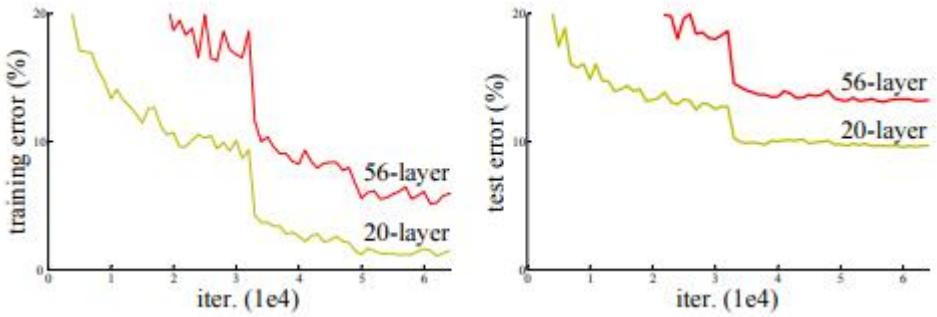
3. Зачем идти в глубину?

Глубокие сети обычно показывают лучшие результаты в задачах классификации изображений.

Например, на большом датасете ImageNet сети с 16–30 слоями работают лучше всего.

Это вдохновило исследователей попробовать ещё более глубокие сети, ожидая повышения точности.

4. В чём проблема?



Может показаться, что достаточно просто добавить больше слоев.

Но оказывается, что увеличение числа слоев не всегда помогает.

Иногда это ухудшает результаты, даже точность на обучении падает!

Например, сеть с 56 слоями показала ошибку выше, чем сеть с 20 слоями — что неожиданно.

5. Может, это переобучение?

Нет.

При переобучении обычно ошибка на обучении мала, а на тесте — велика.

Здесь же более глубокая сеть имеет большую ошибку на обучении, то есть ей сложно даже подстроиться под тренировочные данные.

Это называется проблемой деградации — с увеличением глубины сеть становится труднее обучать.

6. **Почему возникает деградация?**

Если добавить слои, которые просто ничего не делают (идентичное отображение), то глубокая сеть должна работать не хуже.

То есть хорошее решение существует.

Но на практике алгоритмы обучения (SGD, обратное распространение) не могут его найти.

Очень глубокие сети сложно оптимизировать традиционными методами.

Решение: учить разницу, а не функцию целиком

Вместо того, чтобы обучать функцию H(x) напрямую, авторы предлагают учить остаточную функцию:

$F(x)=H(x)-x$

или, эквивалентно,

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

Это значит, что сеть учится, насколько нужно изменить вход x, а не строить полное преобразование с нуля.

Почему так проще?

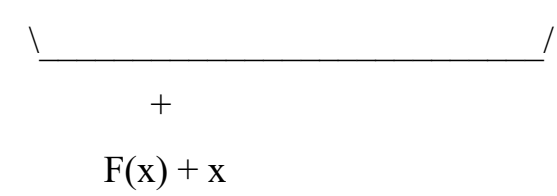
Если лучшая функция — просто оставить вход без изменений (H(x) = x), то остаточная функция F(x)=0.

Это намного проще, чем использовать несколько нелинейных уровней для аппроксимации идентичной функции.

Остаточное обучение упрощает оптимизацию, особенно для очень глубоких сетей.

Как это реализовано?

Остаточное обучение использует короткие связи (shortcut connections), которые пропускают один или несколько слоев и прибавляют вход xx напрямую к выходу слоя:



Эти короткие связи:

Не требуют дополнительных параметров.

Не добавляют вычислительной нагрузки.

Легко реализуются в популярных фреймворках (PyTorch, TensorFlow).

7. Что показали эксперименты?

Авторы протестировали остаточные сети на ImageNet и CIFAR-10 и обнаружили:

Остаточные сети проще обучать — у них меньше ошибка на обучении, чем у обычных.

Они могут быть намного глубже (50, 100, даже 152 слоя) и при этом улучшать точность.

Их 152-слойная сеть установила рекорд — 3.57% ошибка топ-5 на ImageNet, выиграв ILSVRC 2015.

Остаточные сети улучшили результаты в задачах детекции и сегментации на PASCAL VOC и MS COCO.

8. Почему это важно?

Остаточные блоки:

Решают проблему оптимизации очень глубоких сетей.

Позволяют создавать намного более глубокие модели без ухудшения результатов.

Хорошо работают не только для классификации, но и для других задач, например, распознавания речи и обработки языка.

9. Обзор связанных работ

Остаточные представления (VLAD, Fisher Vectors) использовали разницы для улучшения поиска изображений, но это были неглубокие методы.

В численных методах остатки помогают решать сложные задачи на разных масштабах.

Короткие связи использовались и раньше, например в Highway Networks, но там они были с гейтом (контролируемые параметры), что усложняло и снижало эффективность для очень глубоких сетей.

Главное отличие ResNet — это чистые идентичные связи без гейтов и параметров, которые работают даже на больших глубинах.

Математическая формулировка остаточного обучения

Цель: выучить H(x)

Переформулировка:

Цель: выучить $H(x)$.

Переформулировка:

$$F(x) = H(x) - x \quad \Rightarrow \quad H(x) = F(x) + x$$

Где $F(x)$ реализуется несколькими слоями с весами $\{W_i\}$:

$$y = F(x, \{W_i\}) + x$$

Если размеры входа и выхода не совпадают, используется проекция W_s:

$$y = F(x) + W_s x$$

Архитектуры сетей

Авторы построили простые и остаточные версии, вдохновленные VGG:

Используют только 3×3 свертки.

Если размер карты признаков не меняется — число фильтров тоже остается.

При даунсемплинге размер уменьшается, а количество фильтров удваивается.

Простая 34-слойная сеть требует 3.6 млрд FLOPs — лишь 18% от VGG-19, так что она более эффективна, но глубже.

Остаточные сети строятся на этой архитектуре с добавлением коротких связей.

10. Заключение

В работе представлен глубокий фреймворк остаточного обучения, который упрощает и улучшает обучение очень глубоких сетей.

Остаточные сети:

Последовательно превосходят простые аналоги.

Показывают рост качества с увеличением глубины.

Устраняют проблему деградации

С помощью очень глубоких остаточных сетей достигнуто:

3.57% ошибка топ-5 на ImageNet, что лучше предыдущих результатов, включая ансамбли моделей.

Значительный прирост в задачах детекции при замене VGG-16 на ResNet-101 в Faster R-CNN.

Этот метод прост, эффективен и масштабируем, позволяя обучать сети глубиной более 1000 слоев с хорошей оптимизацией и обобщающей способностью.