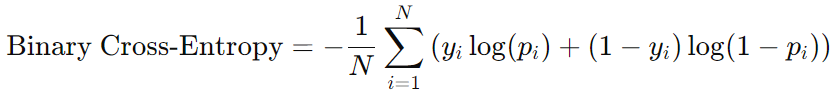
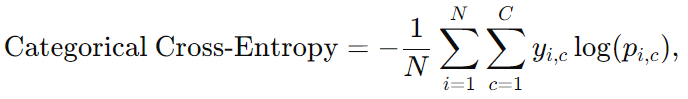
**Подготовка к РК по предмету Нейронные сети в ml**

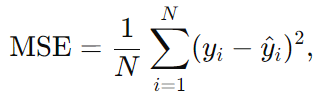
**Теоретический минимум**

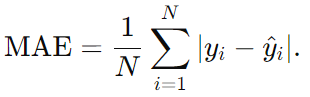
1. **Произношение слов accuracy, bias, variable, function, Байесовский**
2. **Функции потерь для классификации и регрессии**

**Binary Cross-Entropy** (бинарная кросс-энтропия): используется для бинарной классификации, когда есть два класса. Где yi— истинная метка, а pi — предсказанная вероятность для положительного класса.

**Categorical Cross-Entropy** (категориальная кросс-энтропия): используется для много классовой классификации, когда классов больше двух.

где yi,c ​ — бинарный индикатор того, принадлежит ли образец i классу c, а pi,c​ — предсказанная вероятность для класса c.

**Mean Squared Error (MSE)** (среднеквадратичная ошибка): часто используется для задач регрессии, так как усиливает значение больших ошибок. где yi​ — истинное значение, а y^​i​ — предсказанное значение.

**Mean Absolute Error (MAE)** (средняя абсолютная ошибка): хорошо справляется с выбросами, так как не возводит разницу в квадрат.

1. **Проблемы обучения глубинных нейронных сетей**

**Взрыв и затухание градиентов**: При обратном распространении ошибки в глубоких сетях градиенты на начальных слоях могут становиться очень маленькими (затухание) или, наоборот, очень большими (взрыв градиентов). Это мешает корректному обновлению весов, так как на затухающих слоях веса обновляются крайне медленно, а на слоях с взрывом градиентов они изменяются слишком резко. **Решение**: методы нормализации, такие как Batch Normalization, правильная инициализация весов и градиентный спуск с отсечением градиентов.

**Переобучение**: Глубокие нейронные сети с большим количеством параметров могут подстраиваться под тренировочные данные слишком точно. Решение: регуляризация, Dropout, увеличение данных и использование больших и разнообразных наборов данных.

**Недостаток данных**: Для успешного обучения глубоких сетей требуется большое количество данных. Недостаток данных приводит к переобучению и снижению качества предсказаний. **Решение**: методы увеличения данных (data augmentation) и transfer learning.

**Долгое время обучения**: Глубокие сети требуют значительных вычислительных ресурсов и времени для обучения, особенно при работе с большими наборами данных и сложными архитектурами. **Решение**: оптимизация архитектуры модели, использование параллельных вычислений на GPU и эффективные алгоритмы оптимизации, такие как Adam или RMSProp.

**Необходимость настройки гиперпараметров**: Глубокие сети имеют множество гиперпараметров (размеры слоев, скорость обучения, коэффициенты регуляризации и др.), и их неправильная настройка может сильно снизить производительность модели. **Решение**: автоматическая настройка гиперпараметров с использованием методов, таких как grid search, random search и байесовская оптимизация.

**Отсутствие интерпретируемости**: Глубокие сети действуют как "черный ящик", и трудно понять, какие именно признаки или паттерны они используют для принятия решений. **Решение**: интерпретация моделей с помощью Grad-CAM, LIME и SHAP.

**Зависимость от начальной инициализации весов**: Плохая инициализация весов может сильно замедлить обучение, а иногда и привести к застреванию в локальных минимумах. **Решение**: использование специальных методов инициализации, таких как Xavier и He, для улучшения сходимости и качества модели.

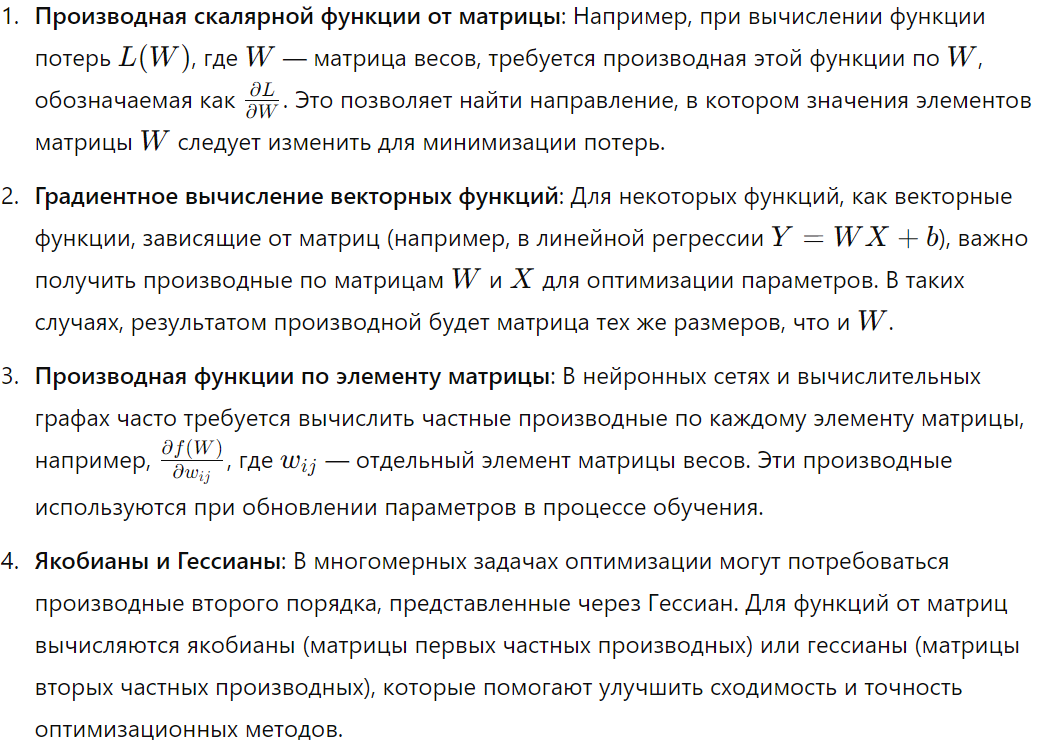
1. **Граф вычислений**

Граф вычислений — это способ представления вычислительных операций, необходимых для выполнения алгоритма, в виде направленного графа. Узлы (вершины) в графе соответствуют переменным и операциям, а ребра — зависимостям между ними. Каждое ребро показывает, как данные "передаются" от одной операции к другой, создавая зависимость и последовательность вычислений.

В глубоких нейронных сетях граф вычислений помогает структурировать все этапы, от ввода данных до расчета потерь и градиентов, и визуализировать взаимодействие между слоями. Важной функцией графа является возможность эффективного вычисления градиентов для обновления весов через метод обратного распространения ошибки (backpropagation). Когда граф завершен, сеть может получить градиенты по каждому узлу, что позволяет вычислить производные от ошибки по параметрам и корректировать их.

Графы вычислений бывают **динамическими** и **статическими**. Динамические графы (например, в PyTorch) позволяют создавать и изменять граф структуры на каждом этапе вычислений, что гибко и удобно для рекурсивных сетей или изменяемых операций. Статические графы (например, в TensorFlow до 2.0) создаются заранее и не изменяются в ходе выполнения, что обеспечивает высокую оптимизацию и скорость, но снижает гибкость.

1. **Задачи на подсчет матричных производных**

Задачи на подсчет матричных производных включают вычисление производных функций, в которых аргументы или значения представлены в виде матриц. В машинном обучении и статистике матричные производные широко используются для оптимизации параметров моделей, таких как нейронные сети, где необходимо рассчитать градиенты для матриц весов и параметров.

1. **Операция свертки**

Операция свертки (convolution) — это ключевая операция в свёрточных нейронных сетях (CNN), используемая для обработки данных, имеющих пространственную структуру, таких как изображения, видео и временные ряды. С её помощью можно обнаруживать и обрабатывать определённые признаки, например, контуры, текстуры или специфические формы, выделяя важную информацию в изображении.

В свёртке над изображением используется небольшой фильтр (ядро свёртки), который представляет собой матрицу, например, 3×3 или 5×5. Этот фильтр последовательно «прокатывается» по всему изображению, перемещаясь с заданным шагом (stride), и на каждом шаге вычисляется значение путём умножения каждого элемента фильтра на соответствующий элемент изображения с последующим суммированием. Полученные значения составляют новое изображение меньшего размера, называемое картой признаков (feature map).

Операция свёртки позволяет сократить размерность входных данных, сохранив при этом наиболее важные пространственные признаки. Для управления размерностью иногда добавляются отступы (padding), что позволяет получать карты признаков того же размера, что и исходное изображение.

Особенность свёртки — это её способность захватывать локальные зависимости и выявлять повторяющиеся паттерны, что позволяет выявлять различные признаки на изображении независимо от их позиции. Различные фильтры в сети могут обучаться для распознавания определённых признаков (например, горизонтальные или вертикальные границы, текстуры и формы).

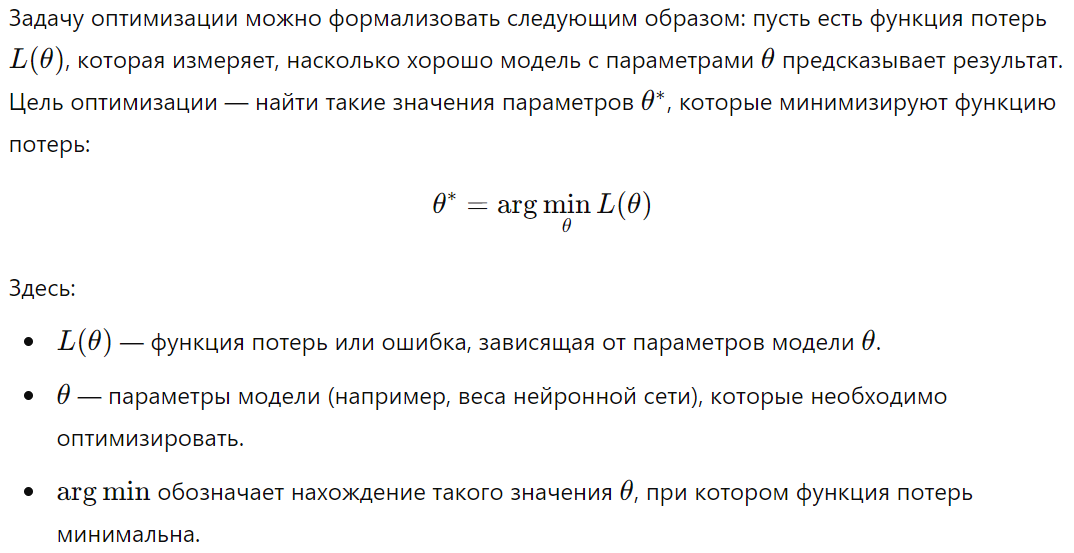
Свёртка является вычислительно эффективной операцией, так как требует меньше параметров по сравнению с традиционными полносвязными слоями.

1. **Операция пулинга**

Операция пулинга (pooling) — это операция, используемая в свёрточных нейронных сетях (CNN) для уменьшения размерности данных и сокращения количества параметров, сохраняя при этом ключевые признаки. Пулинг помогает выделить наиболее значимую информацию, делая сеть менее чувствительной к небольшим сдвигам и искажениям в данных.

1. **Max Pooling**: Наиболее распространённый тип пулинга, который выделяет максимальное значение из каждой области данных. Например, если применяется окно 2×2, то max pooling выбирает максимальное значение из каждого 2×2 блока изображения и сохраняет его в новой карте признаков. Эта операция позволяет эффективно сохранять самые сильные признаки.
2. **Average Pooling**: Вместо максимального значения берётся среднее значение в каждом окне. Average pooling уменьшает чувствительность к небольшим шумам и более равномерно усредняет значения, что может быть полезно, когда важно сохранять общие характеристики изображения, а не только самые сильные признаки.
3. **Global Pooling**: Пулинг проводится один раз на всей карте признаков, сокращая её размер до одного значения на карту (например, один максимум или одно среднее). Global pooling часто используется на последних слоях сети перед полносвязными слоями, чтобы уменьшить размерность перед финальной классификацией.

Операция пулинга снижает размер карты признаков и помогает сократить вычислительные затраты, делая сеть более устойчивой к вариациям данных.

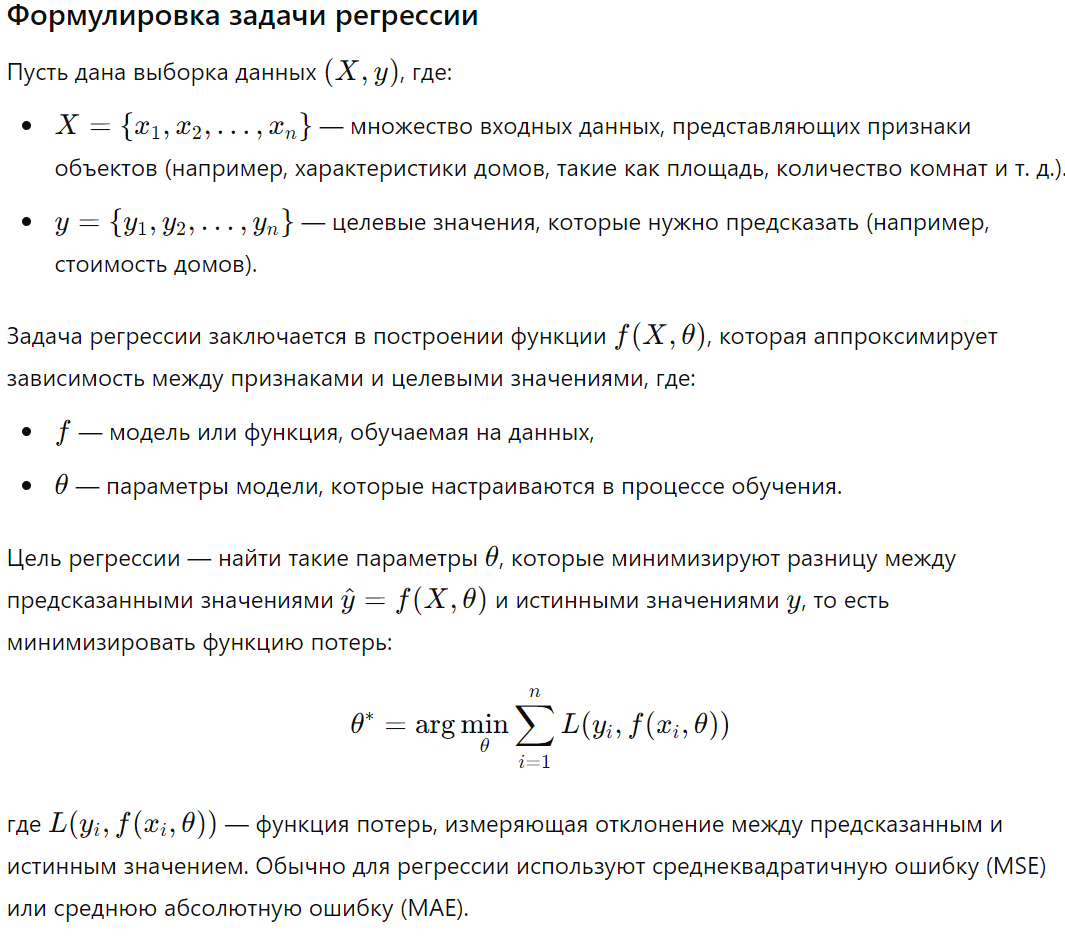
1. **Постановка задачи оптимизации**

Постановка задачи оптимизации в машинном обучении и искусственном интеллекте заключается в минимизации или максимизации функции ошибки (или целевой функции) путём подбора оптимальных параметров модели.

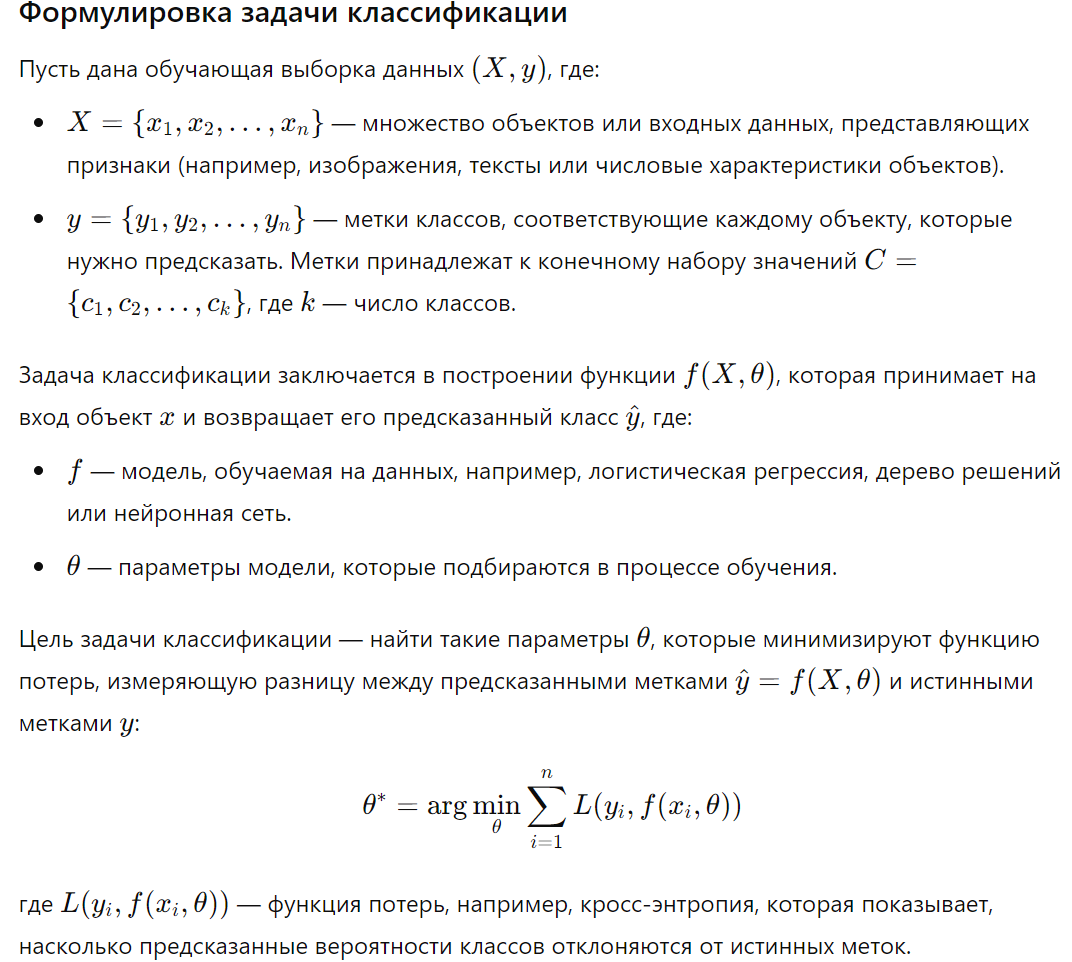
В машинном обучении применяются различные методы оптимизации:

**Градиентный спуск** (Gradient Descent): один из основных методов для минимизации функции потерь, при котором параметры модели обновляются в направлении отрицательного градиента функции потерь.

**Стохастический градиентный спуск (SGD)** и его вариации (например, Adam, RMSProp): модификации градиентного спуска, позволяющие ускорить и стабилизировать обучение модели.

1. **Постановка задачи регрессии**

Постановка задачи регрессии заключается в построении модели, которая предсказывает непрерывные значения на основе входных данных.

1.  **Постановка задачи классификации**

Постановка задачи классификации заключается в построении модели, способной предсказывать категориальные (дискретные) метки для объектов на основе их признаков .

1. **Способы валидации модели**

Способы валидации модели — это методы, которые позволяют оценить её качество на различных данных и избежать проблемы переобучения. Валидация важна для объективной оценки модели, так как она показывает, насколько хорошо модель будет работать на новых, невиданных данных, а не только на данных, на которых она обучалась.

**Hold-out (или простое разделение на обучающую и тестовую выборки)**

Метод Hold-out заключается в простом разбиении данных на две (или три) части: обучающую, валидационную и тестовую. Обычно 60-80% данных используют для обучения, а 20-40% — для тестирования. Иногда дополнительно выделяют валидационную выборку (например, 10-20% данных) для настройки гиперпараметров.

**K-fold cross-validation (K-кратная кросс-валидация)**

В K-кратной кросс-валидации данные разбиваются на K равных частей (фолдов). Модель обучается K раз, каждый раз используя одну из частей как тестовую, а остальные K−1K-1K−1 частей — как обучающие. В итоге модель тестируется K раз на разных фолдах, и финальная оценка рассчитывается как среднее значение всех тестовых результатов.

**Leave-One-Out Cross-Validation (LOOCV)**

В LOOCV каждый объект данных по очереди используется как тестовая выборка, а все остальные объекты — как обучающая. В итоге модель обучается столько раз, сколько объектов в наборе данных, и среднее значение по всем итерациям дает оценку качества модели.

**Stratified K-fold (стратифицированная K-кратная кросс-валидация)**

Stratified K-fold — это вариация K-кратной кросс-валидации, в которой каждый фолд сохраняет пропорции классов, аналогичные всему набору данных.

**Time Series Split (валидация для временных рядов)**

Для временных рядов данные нельзя перемешивать из-за временной зависимости между точками. Поэтому для валидации временных рядов данные делят так, чтобы каждый фолд содержал последовательные временные отрезки.

**Bootstrapping**

Bootstrapping — это метод, при котором из исходных данных берутся выборки с возвращением, то есть некоторые объекты могут попадать в выборку несколько раз, а некоторые — не попадать вовсе.

1. **Особенности работы с изображениями**

**Высокая размерность данных**. Изображения состоят из большого количества пикселей, каждый из которых обычно имеет значения для трёх цветовых каналов (RGB), что приводит к высокой размерности входных данных. Это требует значительных вычислительных ресурсов и применения мощных алгоритмов для обработки и анализа.

**Пространственные взаимосвязи**. В изображениях важен контекст, то есть пространственные зависимости между соседними пикселями. Например, объект можно распознать только с учетом его формы и текстуры, что требует применения методов, учитывающих локальные особенности. Глубинные сверточные нейронные сети (CNN) учитывают такие связи через свёртки, что делает их популярным выбором для задач обработки изображений.

**Предобработка данных**. Изображения необходимо масштабировать, нормализовать и, возможно, обрабатывать фильтрами для улучшения качества данных и ускорения обучения. К предобработке также относят изменение размера изображения, перевод в оттенки серого (для некоторых задач) и устранение шумов.

**Аугментация данных**. Чтобы избежать переобучения и улучшить устойчивость модели, часто применяют аугментацию изображений — искусственное увеличение объема данных с помощью таких трансформаций, как повороты, масштабирование, сдвиги, отражения, добавление шумов и изменение яркости. Это позволяет модели учиться на большем количестве вариаций изображения одного и того же объекта.

**Снижение размерности и извлечение признаков**. Для обработки изображений часто используют методы снижения размерности, такие как Principal Component Analysis (PCA), или глубокие архитектуры нейронных сетей, которые сами извлекают важные признаки. Это помогает сократить размер данных, сохраняя при этом значимую информацию.

**Необходимость в больших данных**. Для успешного обучения моделей, особенно глубоких нейронных сетей, требуется большое количество размеченных изображений. Это связано с высокой сложностью признаков, которые необходимо извлечь для точного распознавания или классификации объектов.

**Особенности цветовых пространств**. В задачах с изображениями могут применяться разные цветовые пространства (RGB, HSV, Lab), выбор которых зависит от особенностей задачи. Например, пространство HSV позволяет выделить оттенки и яркость, что может быть полезным в задачах сегментации.

**Требования к аппаратным ресурсам**. Из-за большого объема и сложности данных обработка изображений требует высоких вычислительных мощностей, особенно при использовании глубоких нейронных сетей. Часто применяют графические процессоры (GPU) для ускорения обучения моделей.

**Специализированные модели и архитектуры**. Для работы с изображениями активно используются сверточные нейронные сети (CNN), которые оптимизированы для извлечения признаков и обработки пространственных данных. Архитектуры CNN, такие как ResNet, VGG и EfficientNet, обеспечивают высокую точность в задачах классификации, детекции и сегментации объектов на изображениях.

1. **Идея transfer learning**

Transfer learning (трансферное обучение) — это подход в машинном обучении, при котором модель, обученная на одной задаче, адаптируется для решения другой, похожей задачи. В традиционном машинном обучении и глубоких нейронных сетях модели обучаются «с нуля» (то есть инициализируются случайными весами), что требует большого количества данных и вычислительных ресурсов. Transfer learning позволяет ускорить обучение и улучшить качество на новом наборе данных, особенно если он небольшого объема.

Основная идея трансферного обучения заключается в том, что нижние слои модели, обученной на большом и универсальном наборе данных (например, ImageNet для задач обработки изображений), извлекают универсальные признаки — такие как контуры, текстуры и простые формы, — которые полезны и для других задач. Эти слои могут быть «заморожены» и не переобучаться, в то время как верхние слои, ответственные за более специализированные признаки, адаптируются под новую задачу.

Применение transfer learning включает несколько этапов:

1. Выбор предобученной модели. Сначала выбирается модель, обученная на похожей задаче или на большом наборе данных, например, ResNet, VGG или BERT, в зависимости от типа данных и задачи (изображения, текст и т.д.).
2. Адаптация модели. В нижних слоях (на ранних уровнях сети) модель сохраняет предобученные веса, так как эти слои уже хорошо научились распознавать универсальные признаки. Верхние слои, связанные с конкретной задачей (например, определенные классы или категории), могут быть изменены или переобучены на новом наборе данных.
3. Финальное обучение. Модель с адаптированными верхними слоями дообучается на целевом наборе данных, часто с пониженной скоростью обучения. Это позволяет модели освоить специфические особенности новой задачи, сохраняя уже имеющиеся универсальные признаки.

**Билеты**

1. **Алгоритм обратного распространения ошибки для графов вычислений**

**Построение графа вычислений**: Сначала создается граф, в котором каждый узел представляет собой операцию (например, сложение, умножение, применение функции активации) или параметр (например, вес или входное значение). Выходной узел представляет собой значение функции потерь, которое модель стремится минимизировать.

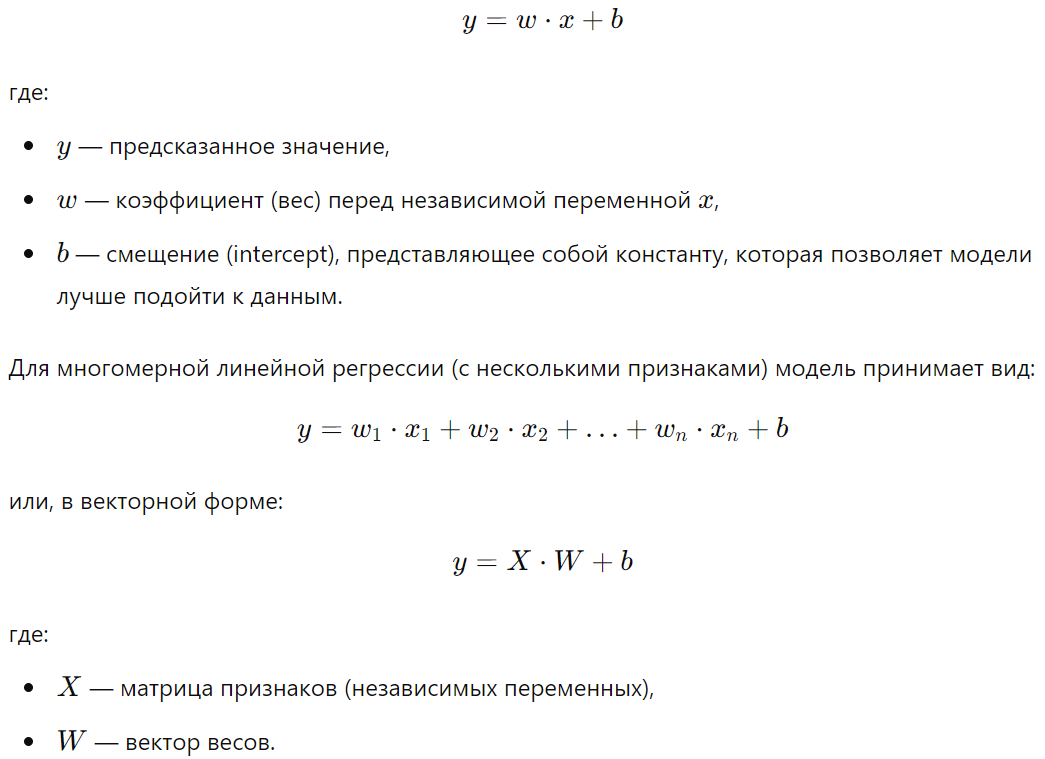
**Прямое распространение (forward pass)**: В ходе прямого прохода данные проходят через все узлы от входных данных до выхода. Это позволяет получить предсказание модели и вычислить значение функции потерь. На этом этапе сохраняются промежуточные значения для каждой операции, так как они понадобятся на этапе обратного распространения.

**Обратное распространение ошибки (backward pass)**: После вычисления значения функции потерь начинается обратное распространение, чтобы определить градиенты. Это происходит с конца графа (выходного узла) к его началу:

* В узле функции потерь вычисляется производная функции потерь по её входу.
* Затем, используя правило цепочки, градиенты «проталкиваются» через узлы графа. Каждый узел вычисляет градиент для своих входов, используя сохраненные значения и переданные градиенты, накопленные из предыдущих слоев.
* На этом этапе учитываются частные производные каждого узла по его входам, и итоговые градиенты передаются на предыдущие узлы.

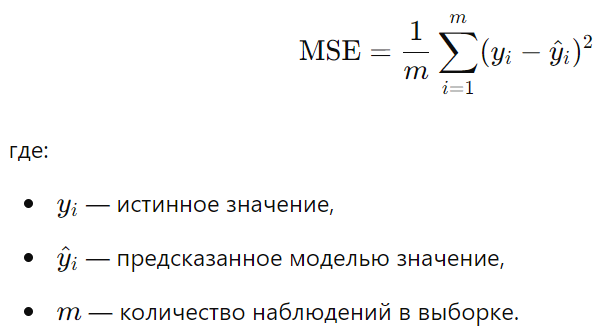
**Обновление параметров**: После вычисления градиентов для каждого параметра сети веса обновляются с использованием выбранного алгоритма оптимизации (например, стохастического градиентного спуска). Это позволяет минимизировать функцию потерь и улучшить точность модели.

1. **Модель линейной регрессии и ее решение**

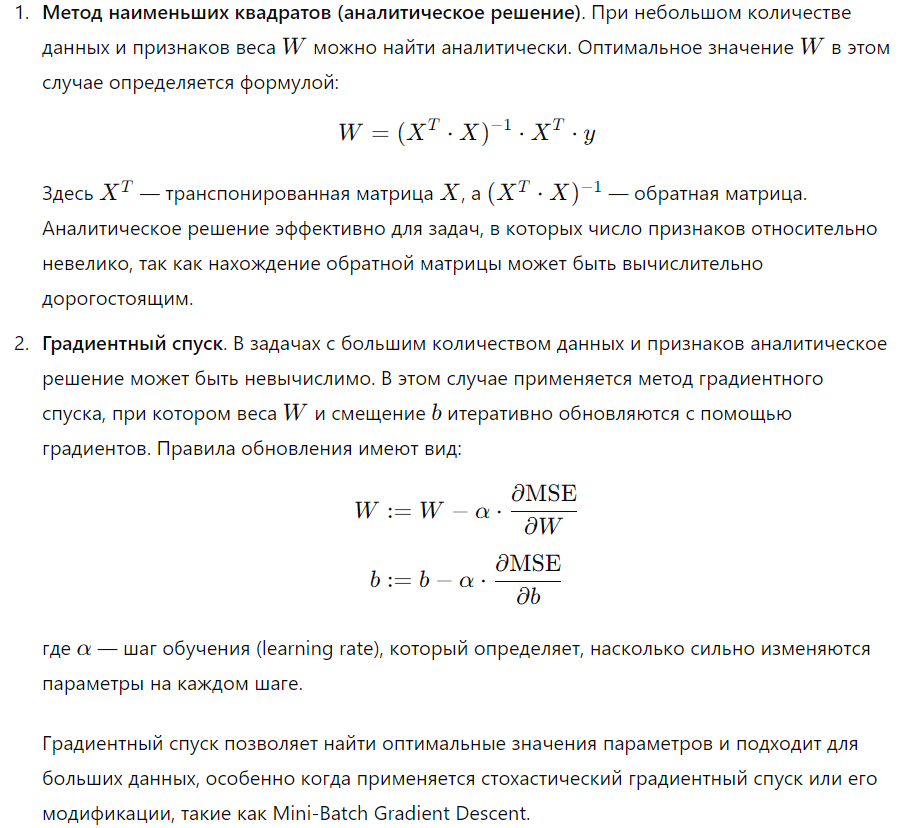
Линейная регрессия предполагает, что между целевой переменной и независимыми переменными существует линейная связь. Модель линейной регрессии для одной переменной выражается как:

**Задача обучения модели**

Цель обучения линейной регрессии — найти такие значения W и b, которые минимизируют ошибку предсказания. Эта ошибка измеряется с помощью функции потерь, обычно средней квадратичной ошибки (MSE):

**Решение задачи линейной регрессии**

Существует два основных метода для нахождения оптимальных значений весов W и смещения b:

Линейная регрессия хорошо подходит для решения задач регрессии, где есть линейная зависимость между целевой переменной и признаками. Однако при наличии нелинейных зависимостей этот метод может показать ограниченную точность, что требует применения нелинейных моделей.

1. **Сверточные нейронные сети**

Сверточные нейронные сети (Convolutional Neural Networks, CNNs) — это класс нейронных сетей, специально разработанный для работы с изображениями и данными, имеющими пространственную структуру. CNN активно применяются в задачах компьютерного зрения, таких как распознавание и классификация объектов, детекция и сегментация объектов, и даже в обработке речи. Основная особенность CNN заключается в использовании сверток, которые позволяют модели эффективно обрабатывать локальные участки изображения, извлекать признаки и выявлять зависимости.

**Основные компоненты сверточной нейронной сети**

1. **Сверточный слой**. Этот слой является основным строительным блоком CNN и выполняет операцию свертки. Свертка заключается в прохождении небольшого фильтра (ядра) по изображению, что позволяет выделить локальные признаки, такие как края, углы и текстуры. Каждый фильтр настраивается в процессе обучения и реагирует на определенные признаки в изображении, а выходное значение каждого фильтра формирует карту признаков (feature map).
2. **Слой активации**. После применения свертки к выходу применяется функция активации, например, ReLU (Rectified Linear Unit), которая добавляет нелинейность. ReLU преобразует отрицательные значения в нули, что помогает модели обучаться сложным и нелинейным зависимостям.
3. **Слой пулинга (Pooling)**. Пулинг — это операция понижения размерности, которая уменьшает размер карты признаков, сохраняя основные признаки. Чаще всего используется max pooling, при котором берется максимальное значение в небольшом окне, которое перемещается по карте признаков. Это помогает уменьшить вычислительные затраты и делает модель более устойчивой к небольшим изменениям в изображении.
4. **Полносвязный слой (Fully Connected Layer)**. После нескольких сверточных и пулинг-слоев данные передаются в один или несколько полносвязных слоев, которые отвечают за окончательное принятие решения. Полносвязные слои используются для комбинирования признаков и формирования предсказаний.
5. **Слой нормализации**. Batch Normalization или другие формы нормализации могут использоваться между слоями для ускорения обучения и улучшения устойчивости модели.

Входное изображение пропускается через серию сверточных, пулинг- и активационных слоев, где каждый слой по очереди обрабатывает и фильтрует информацию. Сначала извлекаются низкоуровневые признаки (например, края и текстуры), а затем более сложные — такие как формы и объекты. На выходе CNN получает карту признаков, которая затем передается в полносвязные слои для классификации.

Сверточные нейронные сети используются в:

* Классификации изображений
* Детекции объектов (например, YOLO, Faster R-CNN)
* Сегментации (например, U-Net, Mask R-CNN)
* Анализе медицинских снимков
* Обработке видео и потоков изображений

1. **Dropout, Batch normalization**

**Dropout** и **Batch** **Normalization** — это техники, широко используемые в нейронных сетях для улучшения обобщающей способности модели и ускорения обучения.

**Dropout** — это метод регуляризации, который помогает снизить переобучение. Основная идея заключается в случайном "отключении" (занулении) нейронов с заданной вероятностью во время каждого шага обучения. При этом структура сети изменяется на каждом этапе обучения, что заставляет нейроны в оставшейся части сети обучаться самостоятельно и не зависеть от других нейронов.

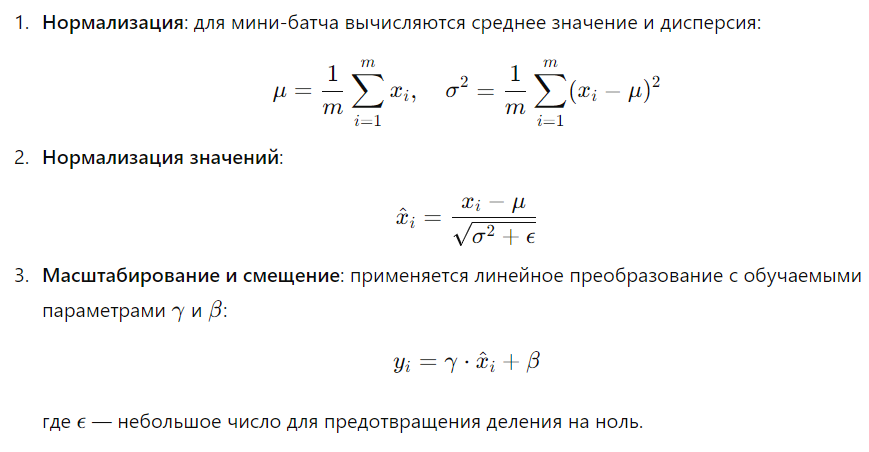
Примерно при обучении каждый нейрон "выпадает" (то есть зануляется) с вероятностью p, которая обычно составляет около 0.5 для скрытых слоев и меньше (например, 0.2) для входного слоя.

**Преимущества Dropout:**

Защита от переобучения, так как сеть не может излишне подстраиваться под обучающие данные.

Улучшение обобщающей способности, так как нейроны вынуждены обучаться работать независимо от других.

На этапе тестирования Dropout не применяется: все нейроны остаются активными, и их веса обычно масштабируются так, чтобы компенсировать "выпадение" нейронов во время обучения.

**Batch Normalization (BN)** — это метод, используемый для ускорения и стабилизации обучения нейронной сети. Его суть заключается в нормализации данных в каждом слое перед передачей на активацию. Во время обучения Batch Normalization нормализует входные данные для каждого слоя (то есть приводит их к нулевому среднему значению и единичной дисперсии), а затем масштабирует и смещает их с помощью параметров, которые также обучаются.

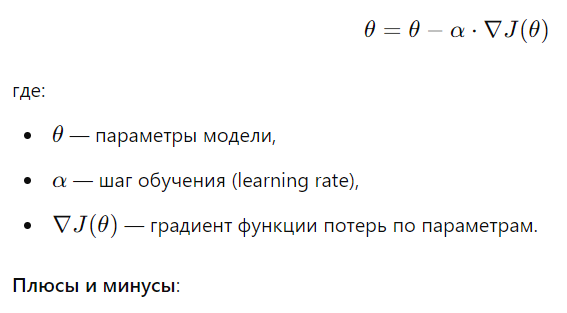
Формула Batch Normalization:

**Преимущества Batch Normalization:**

* Сглаживание и стабилизация градиентов, что позволяет использовать более высокие значения скорости обучения.
* Снижение зависимости от начальной инициализации параметров.
* Регуляризация, что помогает снижать переобучение, даже если Dropout не используется.

**Применение**: Batch Normalization можно использовать перед функцией активации или после нее, но чаще используется перед активацией, особенно с ReLU.

1. **GD, SGD, Momentum, NAG**

**GD (Gradient Descent)**, **SGD (Stochastic Gradient Descent)**, **Momentum** и **NAG (Nesterov Accelerated Gradient)** — это методы оптимизации, которые используются для нахождения минимальных значений функции потерь при обучении нейронных сетей.

**Градиентный спуск (GD)** — это базовый метод, в котором на каждом шаге вычисляется градиент функции потерь по всем данным обучающей выборки. Модель обновляет параметры в направлении отрицательного градиента, что позволяет минимизировать функцию потерь.

Формула обновления параметров:

**Плюсы и минусы**:

* Точный, так как использует все данные.
* Требует больших вычислительных ресурсов, особенно при больших наборах данных.

**Стохастический градиентный спуск (SGD)** — это модификация обычного градиентного спуска, в которой обновление параметров выполняется для каждого отдельного примера (или мини-батча) из обучающей выборки, а не для всех данных сразу.

Формула обновления параметров:

где (xi,yi)— случайно выбранный пример из данных.

**Плюсы и минусы**:

* Работает быстрее и требует меньше памяти, особенно для больших наборов данных.
* Но из-за высокой стохастичности его траектория может быть шумной, и он может сойтись медленнее, чем обычный GD.

**Momentum** — это метод, который ускоряет градиентный спуск и помогает избежать осцилляций. Вместо того чтобы просто двигаться в направлении текущего градиента, Momentum учитывает предыдущие обновления и добавляет их с весом, называемым "моментом" (β\betaβ).

Формулы:

1. Вычисление скорости v (момента) на основе градиента:
2. Обновление параметров

где β обычно выбирается из диапазона 0.9−0.99, что помогает поддерживать более стабильное направление сходимости, и скорость увеличивается в направлениях, где градиенты согласованы.

**Плюсы и минусы**:

* Ускоряет сходимость в "плоских" областях и снижает колебания.
* Может потребовать дополнительных настроек для параметра β.

**Nesterov Accelerated Gradient (NAG)** — это усовершенствование метода Momentum, которое позволяет "заглядывать вперед" в направлении движения и корректировать обновление на основе будущего значения градиента. Вместо вычисления градиента на основе текущих параметров, NAG вычисляет его на основе параметров, предварительно обновленных по направлению момента.

Формулы:

1. Вычисление скорректированных параметров для "заглядывания":
2. Вычисление момента на основе градиента в точке "заглядывания":
3. Обновление параметров:

**Плюсы и минусы**:

* Более точное и оптимизированное движение, так как градиент учитывает изменение параметров.
* Ускоряет сходимость при минимизации функции, но требует больше вычислений и настройки.

**Сравнение методов**

**GD** подходит для небольших датасетов и обеспечивает точное обновление, но медленен для больших данных.

**SGD** эффективен на больших датасетах, но шумен.

**Momentum** уменьшает колебания и ускоряет движение по "плоским" областям.

**NAG** дает более точное направление за счет "заглядывания вперед", делая обучение более оптимальным и стабильным.

1. **Adagrad, RMSprop, Adadelta, Adam**

**Adagrad**, **RMSprop**, **Adadelta**, и **Adam** — это адаптивные методы оптимизации, разработанные для улучшения сходимости в задачах машинного обучения, особенно для глубоких нейронных сетей. Все они корректируют шаг обучения для каждого параметра в зависимости от его градиента, что позволяет более эффективно обрабатывать параметры с разной скоростью обучения.

**Adagrad** адаптирует скорость обучения для каждого параметра, уменьшая её для параметров, которые получают большие градиенты, и увеличивая её для параметров с меньшими градиентами. Это достигается накоплением квадратов градиентов для каждого параметра.

Формула: Суммируем квадраты градиентов:

Обновляем параметры:

где α— начальная скорость обучения, ϵ— небольшое значение для предотвращения деления на ноль.

**Плюсы и минусы**:

* Хорош для данных с разреженностью, так как учитывает редко встречающиеся признаки.
* Сильно затухает со временем, так как сумма квадратов градиентов может стать слишком большой, что приводит к почти нулевому шагу обучения.

**RMSprop** — это модификация Adagrad, в которой вместо накопления квадратов градиентов используется скользящее среднее. Это предотвращает чрезмерное затухание шага обучения.

Формула: Скользящее среднее квадратов градиентов:

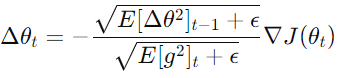
Обновляем параметры:

где β обычно составляет 0.90

**Плюсы и минусы**:

* Подходит для рекуррентных сетей и задач с шумным градиентом.
* Стабильный и более гибкий, чем Adagrad, особенно для сложных сетей.

**Adadelta** улучшает RMSprop, решая проблему с затуханием шага обучения, полностью устраняя начальный шаг обучения α. Вместо этого Adadelta динамически адаптирует шаг, опираясь на накопленные значения градиентов и изменение параметров. Формулы:

Суммируем квадрат градиентов с экспоненциальным затуханием:

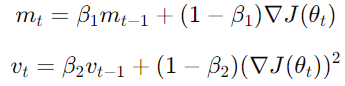
Вычисляем изменение параметров:

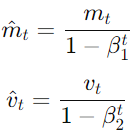
Обновляем параметры:

**Плюсы и минусы**:

* Не требует начального значения α и автоматически адаптирует шаг обучения.
* Успешно справляется с затуханием градиента, но сложен в интерпретации.

**Adam** — это оптимизатор, который объединяет преимущества RMSprop и Momentum, используя два скользящих среднего: для первого момента (градиента) и для второго момента (квадрата градиента). Adam стабилен и эффективен для большинства задач глубокого обучения.

Формулы:

Вычисляем экспоненциальные скользящие средние для градиента и его квадрата

Корректируем смещения:

Обновляем параметры:

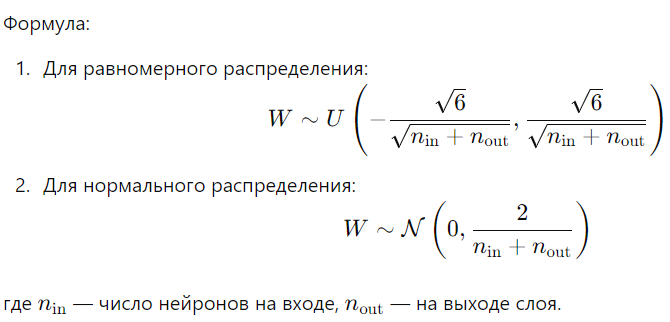
где β1=0.9 и β2=0.99 — коэффициенты для сглаживания, ϵ — параметр для предотвращения деления на ноль.

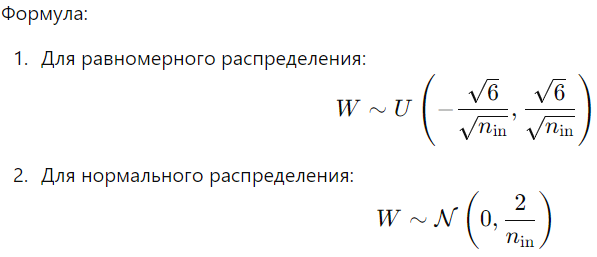
**Плюсы и минусы**:

* Работает хорошо для большинства задач, особенно с большими наборами данных и сложными архитектурами.
* Адаптируется к различным задачам, но может требовать настройки параметров для идеальной сходимости.

1. **Инициализация сетей: Xavier, He, ортогональная инициализация**

Инициализация весов — это важный этап в настройке нейронной сети, влияющий на скорость и стабильность обучения. Правильный выбор инициализации помогает избежать проблем с затуханием или взрывом градиентов.

**Xavier инициализация** (также известная как **Glorot инициализация**) была разработана для решения проблемы затухания и взрыва градиентов в сетях с сигмоидной или тангенциальной функцией активации. Смысл этой инициализации заключается в выборе начальных весов таким образом, чтобы дисперсия значений на выходе слоя была примерно равной дисперсии на входе. Веса выбираются из распределения, зависящего от числа нейронов в слое. Подходит для слоев с симметричными функциями активации, такими как sigmoid или tanh.

**Инициализация He** (или **He нормальная инициализация**) расширяет подход Xavier для сетей с активными функциями ReLU и их производными (Leaky ReLU, ELU), где большое количество выходов обнуляется. Эта инициализация масштабирует веса на основе числа входных нейронов, чтобы предотвратить затухание градиентов и сохранить активные выходы.

Эффективна для слоев с функцией активации ReLU, поскольку позволяет избежать проблем с затуханием градиентов.

**Ортогональная инициализация** основана на выборе весов так, чтобы матрицы весов были ортогональными. Это означает, что столбцы и строки матрицы весов будут линейно независимы, что помогает сохранять информацию и предотвращает корреляцию признаков. Ортогональные матрицы сохраняют норму векторов при передаче сигнала, что стабилизирует распространение градиента и предотвращает затухание и взрыв.

Процедура:

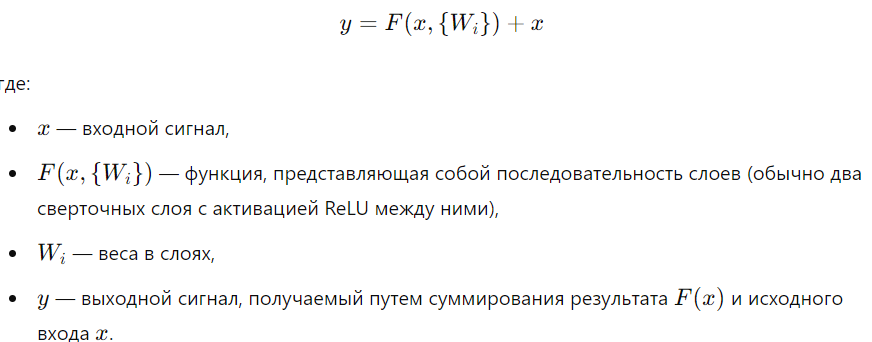
1. Инициализируем матрицу случайным образом и проводим разложение, например, с помощью QR-разложения.
2. Ортогональную матрицу Q используем в качестве весов.

Хорошо работает для рекуррентных нейронных сетей и в глубоких сетях с различными активациями, особенно на начальных этапах обучения.

1. **ResNet**

ResNet (Residual Network) — это архитектура глубоких нейронных сетей, разработанная исследователями из Microsoft Research в 2015 году для решения проблемы затухания градиентов при обучении глубоких моделей. Благодаря своей инновационной архитектуре, ResNet позволила строить чрезвычайно глубокие сети (до 1000 слоев и более), поддерживая стабильное и эффективное обучение. Эта модель продемонстрировала высокую производительность на множестве задач, таких как классификация изображений и обнаружение объектов.

**Ключевая идея: Остаточные блоки (Residual Blocks)**

Основная инновация ResNet — это **остаточные блоки** (residual blocks), которые содержат «прямые связи» или **shortcut connections** между слоями. Эти соединения позволяют пропускать сигнал через несколько слоев без изменения, добавляя его к результату преобразования, выполненного последовательностью слоев.

Формула остаточного блока:

В такой структуре **shortcut connections** позволяют передавать исходный сигнал через слой, что позволяет сохранить и улучшить передачу градиентов. Это значительно уменьшает проблему затухания градиентов, часто встречающуюся в очень глубоких сетях.

**Решение проблемы затухания градиентов**: за счет прямых связей градиенты лучше распространяются по сети.

**Повышенная точность и стабильность**: позволяет строить более глубокие сети, не теряя точности.

**Облегченная оптимизация**: остаточные блоки упрощают обучение сети, так как при необходимости могут позволить слоям выучивать тождественное преобразование (пропуская их).

С момента появления ResNet стала стандартом для многих приложений глубокого обучения:

* **Классификация изображений**: ResNet установила новый стандарт на ImageNet и в задачах классификации.
* **Обнаружение объектов** и **сегментация**: ResNet используется как базовая архитектура для многих современных моделей (например, Faster R-CNN, Mask R-CNN).
* **Transfer learning**: ResNet показала хорошие результаты в предварительном обучении (pretrained), когда обученные на ImageNet веса адаптируются под другие задачи и данные.

Таким образом, ResNet произвела революцию в обучении глубоких сетей, предоставив универсальную архитектуру, которая справляется с множеством задач, не требуя сложных механизмов обработки градиентов.

1. **RCNN, FastRCNN, FasterRCNN, SSD**

**R-CNN, Fast R-CNN, Faster R-CNN и SSD** — это популярные архитектуры для обнаружения объектов, каждая из которых поэтапно улучшала скорость и точность обнаружения.

**R-CNN** была предложена Россом Гиршиком в 2014 году и стала одним из первых значительных достижений в обнаружении объектов с использованием нейронных сетей. Она выполняет задачу в несколько этапов:

1. **Выделение предложений объектов**: Используется метод, например, Selective Search, чтобы выделить около 2000 предложений для потенциальных объектов (Region Proposals).
2. **Извлечение признаков**: Для каждого предложения создается обрезанное изображение, которое затем пропускается через сверточную сеть, чтобы извлечь признаки.
3. **Классификация и локализация**: Признаки передаются в SVM для классификации и в линейную регрессию для определения координат (bounding box regression).

**Недостатки**:

* **Медлительность**: R-CNN была очень медленной, так как для каждого предложения нужно было запускать сверточную сеть отдельно, что делало обработку изображений очень ресурсоёмкой.
* **Большой объем памяти**: Хранение отдельных признаков для каждого предложения требовало много памяти.

**Fast R-CNN**, предложенная также Гиршиком, существенно улучшила R-CNN, ускорив процесс за счёт объединения некоторых этапов:

1. **Обработка изображения**: Полностью сверточная сеть теперь обрабатывает изображение целиком один раз, а не для каждого предложения.
2. **Региональные предложения (Region of Interest, ROI)**: На этапе ROI Pooling выделяются предложения объектов и используется полносверточная карта признаков для ускорения обработки.
3. **Классификация и локализация**: Классификация и предсказание координат боксов выполняются в конце, вместо использования отдельных моделей для этих задач.

**Преимущества**:

* **Ускорение**: Однократная обработка изображения сверточной сетью ускоряет работу.
* **Повышение точности**: Использование ROI Pooling улучшает передачу информации о форме объектов и повышает точность.

**Faster R-CNN** полностью меняет этап выделения предложений, используя Region Proposal Network (RPN):

1. **Единая сверточная сеть**: RPN строится на основе той же сверточной сети, которая извлекает признаки для классификации.
2. **Region Proposal Network (RPN)**: Вместо использования сторонних методов для создания предложений объектов, RPN генерирует их как часть единой сети. RPN обучается находить потенциальные области (anchors) с объектами и передавать их на классификацию.
3. **Единая архитектура**: Faster R-CNN объединяет и RPN, и классификацию в одну сеть, что делает её эффективнее, чем Fast R-CNN.

**Преимущества**:

* **Скорость и точность**: Использование RPN сокращает время обработки, сохраняя высокую точность обнаружения.
* **Целостность сети**: Полное объединение предложений и классификации в один процесс оптимизирует ресурсы и упрощает обучение.

**SSD** (Single Shot MultiBox Detector) предлагает совершенно иной подход, существенно ускоряя процесс за счет применения архитектуры на основе одной сети:

1. **Один проход (Single Shot)**: В SSD используется одна нейронная сеть, которая сразу предсказывает bounding boxes и классы для каждого объекта на изображении.
2. **Многоуровневый детектор**: Используется несколько уровней карты признаков для поиска объектов разных размеров — от крупных на низких уровнях к более мелким на высоких уровнях.
3. **Anchor boxes**: Подобно Faster R-CNN, SSD использует anchor boxes для обработки объектов разной формы и размеров.

**Преимущества**:

* **Быстродействие**: Поскольку сеть делает всё за один проход, SSD работает значительно быстрее, чем Faster R-CNN.
* **Точность на разных размерах объектов**: Многоуровневая структура помогает эффективно обнаруживать объекты разного размера, хотя с небольшим снижением точности на мелких объектах по сравнению с Faster R-CNN.

**Сравнение и итог**

* **R-CNN**: Медленная, но значительно продвинула исследования в области детекции объектов.
* **Fast R-CNN**: Быстрее и точнее, так как обрабатывает изображение один раз.
* **Faster R-CNN**: Сочетает скорость и точность, используя RPN для генерации предложений, но требует больше ресурсов, чем SSD.
* **SSD**: Самая быстрая среди всех моделей, подходит для реального времени, хотя немного уступает в точности на мелких объектах.

1. **VGG, Network in Network**

VGG и Network in Network (NIN) — это архитектуры нейронных сетей, которые внесли значительные новшества в подходы к проектированию сверточных сетей, что позволило добиться высоких результатов в задачах распознавания и классификации изображений.

**VGG** была разработана группой исследователей из Оксфордского университета в 2014 году и представлена на конкурсе ImageNet. Главная идея VGG заключается в использовании небольших фильтров (3x3) и увеличении глубины сети. Основные особенности:

1. **Небольшие фильтры 3x3**: В отличие от предшествующих архитектур, VGG использует только фильтры размером 3x3, что позволяет глубже анализировать пространственные зависимости на изображении. Каскад таких фильтров эффективно имитирует больший размер фильтра (например, три слоя 3x3 примерно аналогичны 7x7), сохраняя при этом меньшее количество параметров.
2. **Глубокая архитектура**: VGG-16 и VGG-19 — одни из самых популярных версий модели — содержат 16 и 19 слоев соответственно. Увеличение числа слоев улучшает способность сети распознавать сложные паттерны, хотя это увеличивает вычислительную нагрузку.
3. **Слои pooling**: После нескольких сверточных слоев следуют слои max pooling (обычно размером 2x2), которые уменьшают размер пространственного представления, сохраняя важные признаки и уменьшая вычислительные затраты.
4. **Полносвязные слои**: Завершающие слои сети — это несколько полносвязных слоев, которые обрабатывают выход сверточной части сети и выполняют классификацию.

**Преимущества и недостатки**:

* **Высокая точность**: VGG показала отличные результаты на задачах классификации.
* **Много параметров**: Из-за полносвязных слоев и глубины архитектуры модель имеет большое количество параметров (до сотен миллионов), что делает её ресурсозатратной и медленной для обучения и инференса.

**Network in Network (NIN)**, предложенная в 2013 году, — это подход, направленный на улучшение представлений, создаваемых сверточными сетями, путём замены традиционных сверточных слоев на блоки, которые содержат маленькие «сети внутри сети». Основные идеи:

1. **Сверточные слои с MLP (Multi-Layer Perceptron) слоем**: В NIN вместо одного линейного фильтра для каждого канала используется многослойный перцептрон, что позволяет сверточным слоям обучаться более сложным и многоуровневым представлениям. Эта комбинация обычно включает слой, который называется **1x1 сверточным слоем**, где фильтры 1x1 работают как небольшие полносвязные сети.
2. **Меньшее количество параметров**: Благодаря использованию 1x1 фильтров, NIN снижает число параметров по сравнению с традиционными глубокими сверточными архитектурами, делая сеть более компактной и менее подверженной переобучению.
3. **Глобальный average pooling вместо полносвязных слоев**: В отличие от VGG и других моделей, в NIN используется глобальный average pooling на завершающем слое вместо полносвязных слоев. Это снижает количество параметров и делает модель более эффективной, а также снижает риск переобучения.

**Преимущества и недостатки**:

* **Улучшенное представление**: NIN позволяет обучать более сложные и адаптивные представления за счёт 1x1 фильтров.
* **Компактность**: Использование глобального pooling и отсутствие полносвязных слоев сокращает число параметров.
* **Меньше вычислительных затрат**: Модель более лёгкая и быстрая, что делает её подходящей для приложений с ограниченными вычислительными ресурсами.

1. **Inception**

Inception — это архитектура нейронной сети, предложенная в 2014 году для задачи классификации изображений, и ставшая частью семейства сетей GoogleNet. Главной целью Inception было создание эффективной по вычислительным ресурсам глубокой сети, которая бы могла захватывать более разнообразные пространственные и контекстные особенности изображения. Архитектура использует различные типы фильтров и слоев в одном блоке, что позволяет сети «решить», какие масштабы и особенности извлекать из входного изображения. Это позволило значительно улучшить точность классификации и детекции объектов, особенно в масштабных задачах, таких как ImageNet.

**Основные идеи и компоненты Inception**

1. **Inception-модули**:
   * Главная особенность сети — **Inception-модуль**, где несколько типов слоев и фильтров (1x1, 3x3, 5x5) работают параллельно на одном уровне. Это позволяет сети захватывать как локальные, так и глобальные особенности изображения.
   * Каждый модуль включает различные сверточные фильтры и слои max pooling. Выходные представления из всех слоев объединяются по глубине, создавая объемное представление данных.
   * Применение фильтров разных размеров в одном модуле позволяет улавливать черты на разных масштабах, сохраняя при этом архитектуру более эффективной.
2. **1x1 сверточные слои для уменьшения размерности**:
   * Фильтры 1x1 не только добавляют нелинейность, но и используются для уменьшения размерности перед более широкими 3x3 и 5x5 фильтрами. Это помогает снизить число параметров и вычислительную сложность, поскольку меньшее количество параметров помогает предотвратить переобучение и снижает затраты на вычисления.
3. **Эффективность и оптимизация**:
   * Благодаря параллельной обработке с фильтрами разного масштаба и использованию 1x1 слоев для снижения размерности, Inception поддерживает высокую глубину сети без избыточного увеличения числа параметров.
   * Это делает сеть подходящей для задач классификации изображений с большим количеством классов, обеспечивая баланс между точностью и вычислительной сложностью.

**Версии Inception**

* **Inception v1** (GoogleNet): Первая версия, которая включала базовую архитектуру Inception с параллельными 1x1, 3x3, 5x5 фильтрами и pooling.
* **Inception v2 и v3**: Эти версии улучшили архитектуру, добавив, например, раздельные (factorized) сверточные фильтры, которые позволяют разбивать более крупные фильтры (например, 5x5) на последовательность из нескольких более мелких (например, два 3x3), что снижает вычислительные затраты и количество параметров.
* **Inception v4 и Inception-ResNet**: В этих версиях добавились остаточные (residual) соединения, как в ResNet, что ещё больше стабилизировало обучение глубоких сетей. Эти изменения помогли улучшить сходимость и точность на сложных датасетах.

**Преимущества и недостатки**

**Преимущества**:

* **Высокая эффективность и производительность**: Сеть оптимизирована для захвата разнообразных особенностей, улучшая точность на разных масштабах.
* **Меньше параметров** по сравнению с аналогичными по мощности архитектурами, что снижает вычислительные затраты.

**Недостатки**:

* **Сложность архитектуры**: Модуль Inception имеет сложную структуру, которая затрудняет реализацию и оптимизацию.
* **Большое количество гиперпараметров**: Подбор гиперпараметров и фильтров для каждой версии сети требует внимательной настройки.