Обзорная статья
по теме:

Введение в МО. Интуитивное обучение в нейронных сетях

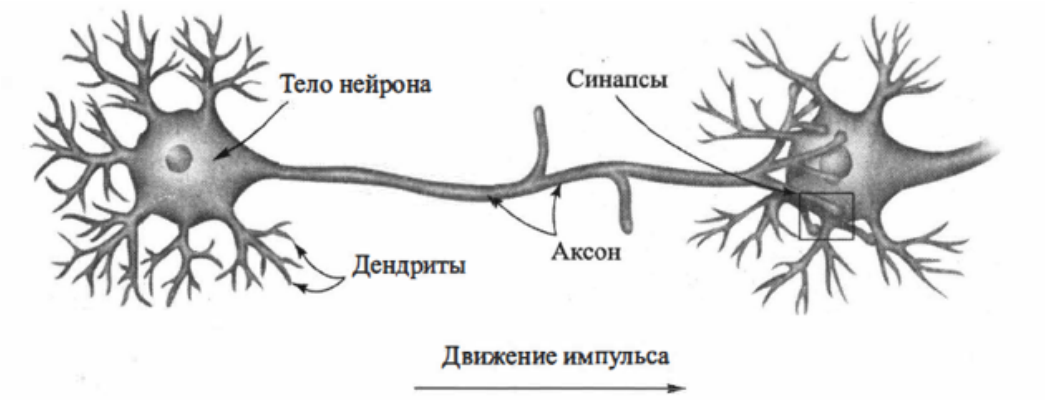
Выполнил:
https://github.com/enlacroix

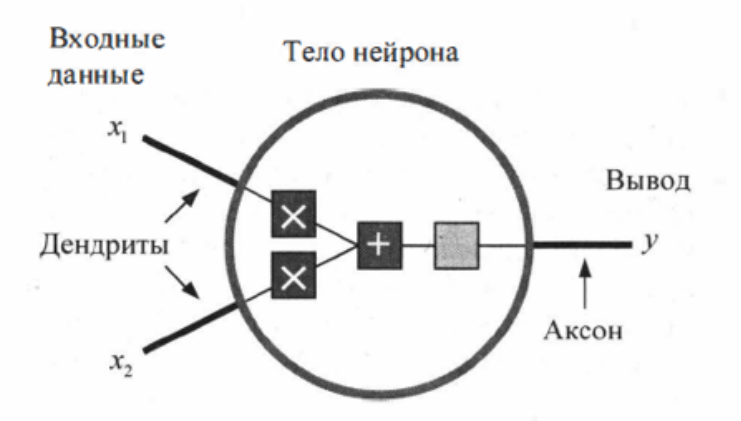
Содержание

1 Введение в машинное обучение

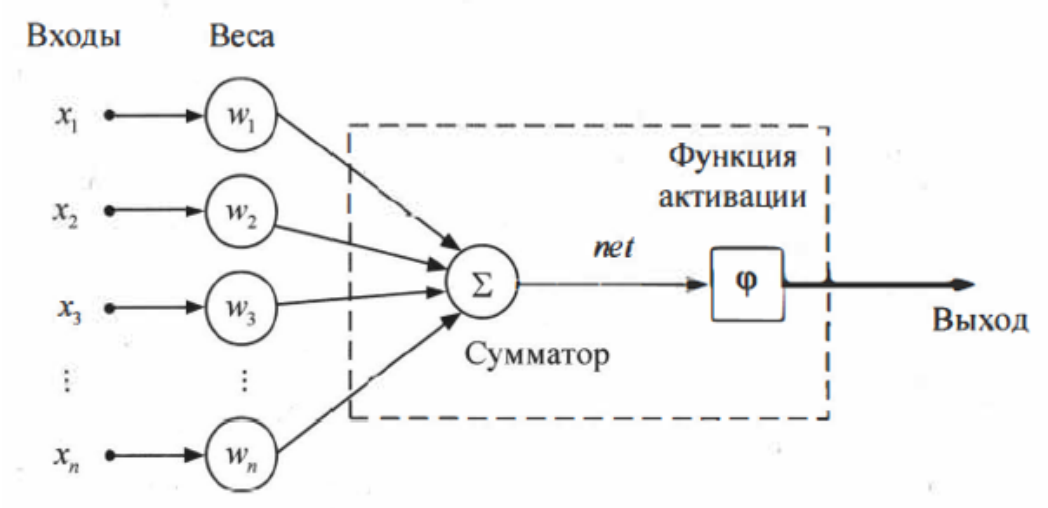
1.1 Математическая модель нейрона

Одна из основных задач искусственной нейронной сети - смоделировать реальное человеческое мышление, которое способно выполнять нетривиальные задачи, на которые не способны классические программы. Поэтому разумно было бы обратиться к нейробиологии и почерпнуть идеи из настоящих нейронов человеческого мозга. Рисунок ниже более подробно отражает эту аналогию.



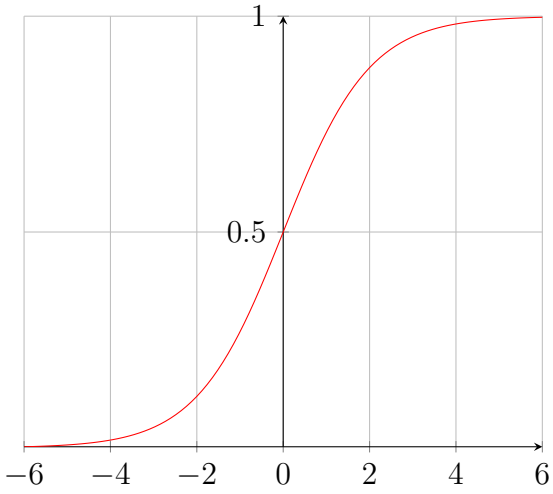


Входные данные поступают в нейрон, который можно считать 'черным ящиком', выдающим конкретный результат y. Попробуем увеличить уровень абстракции и построить уже математическую модель нейрона:



Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. На вход математического нейрона поступает некоторое количество входных параметров $x_1, x_2, \ldots, x_n$, т. е. дендритов, каждый входной параметр имеет свой вес - $w_1, w_2, \ldots, w_n$. В теле искусственного нейрона имеется сумматор, где каждый входной сигнал умножается на некоторый действительный весовой коэффициент и формируется итоговая сумма. Полученное значение передается в *функцию активации*, которая и определяет активируется нейрон или нет. К тому же она нормализует входные данные. То есть, если на входе у вас будет большое число, передав его функцию активации, будет получено число в нужном диапазоне (обычно [0, 1] или [-1, 1]).

$$\boxed{sigmoid(x) = \frac{1}{1 + e^{-x}}}$$



Это сигмоид – самая распространенная функция активации, ее диапазон значений $E(y) = [0,1]$.
Разобравшись с устройством одного нейрона, перейдем к рассмотрению их совокупностей - *слоев*. Различают входной (распределение данных по последующим нейронам), скрытый (осуществляет промежуточные вычисления) и выходной (выдача окончательного результата).

1.2 Основные понятия

Построенная матмодель формализирует человеческое мышление, что позволяет решить множество ключевых задач через *обучение*:
Обучение нейросети это поиск таких весовых коэффициентов, при котором входные данные после прохода по сети преобразуются в нужную нам выходную информацию.
Процесс обучения называется **глубоким**, если структура искусственных нейронных сетей состоит из нескольких входных, выходных и скрытых слоев.
Однако задача нейросети состоит не в том, чтобы в результате подбора коэффициентов выдать идеально правильный ответ. Ожидается получить гораздо больший результат – алгоритм, способный к обобщению данных. Выделяют две выборки – *обучающую* (на которой обучается алгоритм) и *тестовую* (на которой оценивается эффективность алгоритма). Более подробно эта проблема будет рассмотрена в разделе 3.3. **Эпоха** – одна итерация в процессе обучения, включающая предъявление всех примеров из обучающего множества. ([2])
Гиперпараметры —- это настраиваемые параметры, позволяющие управлять процессом обучения модели.

Ошибка — это величина, отражающая расхождение между ожидаемым и полученным ответами.
Ошибка формируется каждую эпоху и должна идти на спад. Если этого не происходит, то необходимо либо изменить ее внутреннюю структуру, либо изменить гиперпараметры. Существует много способов оценить величину ошибки (вычислить *функцию потерь*, например MSE (mean squared error):

$$MSE = \frac{\sum_{k=1}^{n} (i_k - a_k)^2}{n}$$

Или MAE (mean absolute error):

$$MAE = \frac{\sum_{k=1}^{n} |i_k - a_k|}{n}$$

Определить в каких случаях применять ту или иную метрику – отдельное искусство. так как универсальных методов расчета ошибок не существует. Резюмируя, задача алгоритма обучения состоит в том, чтобы подобрать веса так, чтобы функция потерь(*loss - function*, обозначают иногда L) стремилась к своему минимуму.

2 Парадигмы обучения

2.1 Обучение с учителем (supervised learning)

Согласно определению, приведённому в книге [1] Обучение с учителем (supervised leaming)- вид обучения нейронной сети, при котором веса нейронных связей подбираются таким образом, чтобы ответы на выходе из сети минимально отличались от уже готовых правильных ответов. В этом случае обучение происходит на размеченном наборе данных: задается пара векторов (x, d), где x - условие задачи, d - верный ответ, известный заранее. Одна исключительная особенность этой парадигмы обучения — прямая ссылка на ошибку, которая просто сравнивает между плановым и текущим результатом. Обучение с учителем подходит для задач, которые заранее предоставляет паттерн, цель достижения. Примеры: классификация картинок, распознавание голоса, функция аппроксимации, прогнозирование.
Обучение с учителем больше всего подходит для задач, когда имеется внушительный набор достоверных данных для обучения алгоритма.

2.2 Обучение без учителя (unsupervised learning)

В обучении без учителя мы имеем дело только с данными без меток или классификации. Нейронная сеть пытается самостоятельно найти корелляции в данных, извлекая полезные признаки и анализируя их. В обучении без учителя сложно вычислить точность алгоритма, так как в данных отсутствуют «правильные ответы» или метки.
Примеры задач, где используется обучение без учителя: кластеризация(алгоритм подбирает похожие данные, находя общие признаки, и группируют их вместе.), обнаружение аномалий, статистические модели и языковые модели.

2.3 Обучение с подкреплением (reinforcement learning)

Процесс обучения с подкреплением можно смоделировать как цикл, который работает следующим образом:

- Нейросеть имеет некий параметр изначально равный S_0.
- Исходя из этого состояния S_0, алгоритм предпринимает некое действие.
- Параметр принимает новое значение S_1.
- Алгоритм дает некоторое 'поощрение' или 'наказание' за предпринятое действие.

Этот RL-цикл выдаёт последовательность состояний, действий и вознаграждений. Цель алгоритма — максимизировать ожидаемое накопленное вознаграждение. Однако на самом деле ситуация куда более сложнее, так как вознаграждения в начале и конце (если он есть) обучения не равноценны, поэтому естественно вводить поправочные коэффициенты.

2.4 Самообучение (self-supervised learning)

Мостиком между парадигмами USL и SL служит самообучение (SSL). Дана огромная неразмеченная выборка, необходимо **случайно** сформировать для каждого объекта *псевдо-метку* (*pseudolabel*) и решить полученную SL-задачу, но нас интересует не столько качество решения придуманной нами задачи (её называют *pretext task*), сколько представление (*representation*) объектов, которое будет выучено в ходе её решения. Это представление можно в дальнейшем использовать уже при решении любой задачи с метками (SL), которую называют последующей задачей (*downstream task*).
Яркой иллюстрацией самообучения являются современные методы обработки текста (NLP), которые анализируют огромные объемы неразмеченного

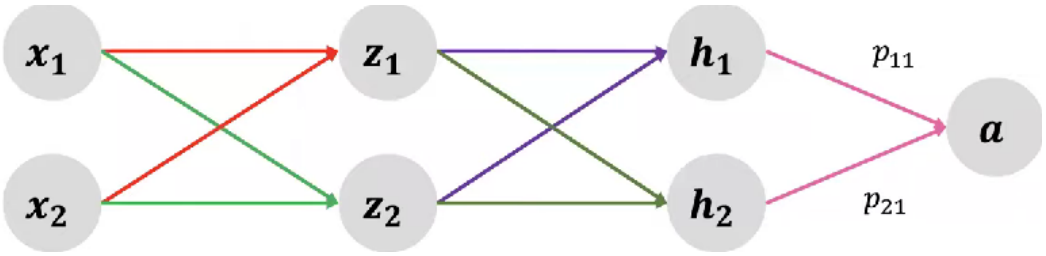
текста, строят начальное признаковое представление, на основании которого последующие алгоритмы будут решать основную задачу. Резюмируя все вышесказанное, можно дать четкое определение.
Самообучение – это направление в глубоком обучении, которое стремится сделать глубокое обучение процедурой предварительной обработки и разметки данных.

3 Алгоритмы обучения

Рассмотрим теперь, каким образом нейросеть может перестраиваться для минимизации функции потерь.

3.1 Детерминированные методы

Алгоритмы, основанные на детерминированных методах, итеративно корректируют параметры сети, учитывая ее текущие параметры, величины входа, фактических и требуемых выходы. Рассмотрим детерминированный подход на примере *метода обратного распространения ошибки.*
Цель метода состоит в том, чтобы отрегулировать каждый вес пропорционально тому, насколько он способствует общей ошибке. Таким образом, итеративно уменьшая ошибку на каждом ходу, то мы придем к оптимальному результату. Чтобы понять, как рассчитывается вклад каждого веса в финальный результат, рассмотрим простую модель нейросети с двумя скрытыми слоями:



$$a(x) = p_{11}h_1(x) + p_{21}h_2(x)$$

Возьмем от функции $a(x)$ частную производную по весу p_{11}, которая будет характризовать вклад веса p_{11} в итоговый результат (что нам и нужно)

$$\frac{\partial a(x)}{\partial p_{11}} = h_1(x)$$

Посмотрим, как влияет на итоговый результат v_{11} – вес ребра между z_1 и h_1. Для этого нужно расписать $a(x)$ и найти производную сложной функции (или

по зарубежной терминологии, воспользоваться цепным правилом):

$$a(x) = p_{11} f\left(v_{11} z_1(x) + v_{21} z_2(x)\right) + p_{21} h_2(x)$$
$$\frac{\partial a}{\partial v_{11}} = \frac{\partial a}{\partial h_1} \frac{\partial h_1}{\partial v_{11}}$$

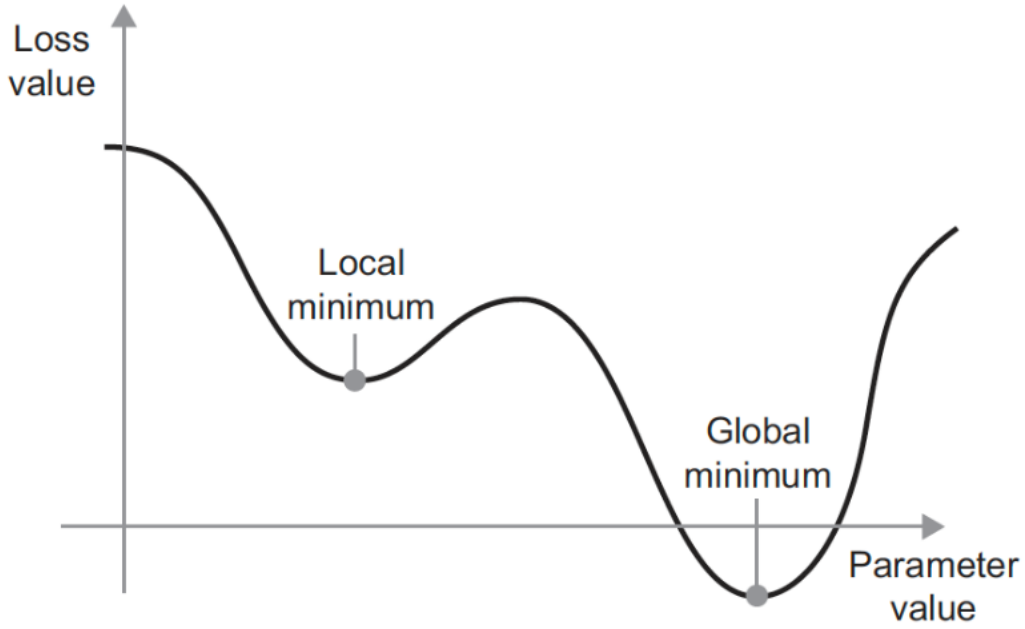
где f – это произвольная функция активации. Таким образом можно определить вклад каждого веса в финальный результат и в полученную ошибку. Для начала мы вычисляем ошибку выходного слоя и передаем результат на скрытый слой перед ним. После вычисления ошибки скрытого слоя мы передаем ее значение обратно на предыдущий скрытый слой и т.д.

3.2 Стохастические методы

Стохастические методы обучения изменяют параметры сети случайным образом. При этом сохраняются только те изменения, которые привели к улучшениям.

- Выбрать параметры сети случайным образом и подкорректировать их на небольшую случайную величину. Предъявить множество входов и вычислить получающиеся выходы.
- Сравнить эти выходы с желаемыми и вычислить разницу между ними. Эта разница называется ошибкой. Цель обучения состоит в том, чтобы минимизировать ошибку.
- Если ошибка уменьшилась, коррекция сохраняется, в противном случае коррекция отбрасывается и выбирается новая.

Возможны случаи, когда при стохастическом обучении нейросеть попадает в *ловушку локального минимума*, когда на определенном участке графика $e(w)$ (e - значение функции потерь, w - вес, регулируемый параметр) кажется оптимальным выбранное значение w, однако существует такое значение w, при котором функция достигает глобального минимума.



Полезная стратегия для избегания подобных проблем состоит в больших начальных шагах и постепенном уменьшении размера среднего случайного шага. Это позволяет сети вырываться из локальных минимумов и в то же время гарантирует окончательную стабилизацию сети.

Коэффициент скорости обучения – это гиперпараметр, определяющий порядок того, как мы будем корректировать наши веса. Чем ниже величина, тем медленнее мы движемся по наклонной. Хотя при использовании низкого коэффициента скорости обучения мы можем получить положительный эффект в том смысле, чтобы не пропустить ни одного локального минимума, — это также может означать, что нам придётся затратить много времени на сходимость, особенно если мы попали в область плато.

Классическим примером вероятностного метода является генетический алгоритм, который описывается в приложении **??**. Остановимся подробнее на более практичном алгоритме:

Стохастический градиентный спуск(*stochastic gradient descent*).

Суть обыкновенного градиентного спуска - минимизировать функцию потерь, каждый раз делая шаги в сторону наискорейшего убывания функции. Для это используется факт того, что вектор $\nabla f = \left(\frac{\partial f}{\partial x_1}, \dots \frac{\partial f}{\partial x_n}\right)^T$ частных производных фуннкции $f(x) = f(x_1, \dots x_n)$ задает направление наискорейшего возрастания этой функции. В таком случае движение по антиградиенту задает наибыстрейшего убывания функции.

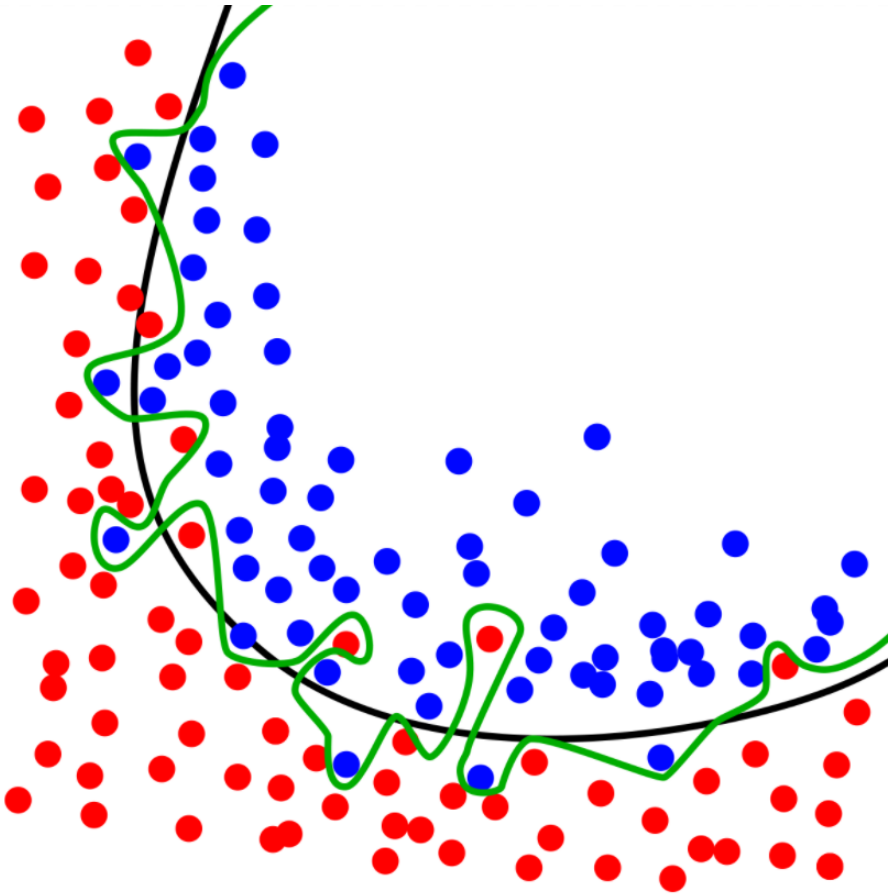
Проблема такого подхода в том, что для обсчета каждого объекта требуется суммировать некоторые величины. На практике это занимает слишком много времени, для больших задач метод необходимо подвергнуть модификации. Её суть в использовании только одного элемента, либо некоторой подвыборки, но не всего набора данных.

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \sum_{i=1}^{l} \nabla \mathscr{L}_i \left(\mathbf{w}^{(t)}\right) \quad (a : GD)$$
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla \mathscr{L}_i \left(\mathbf{w}^{(t)}\right) \quad (b : SGD)$$

$\mathbf{w}, \eta$: весовой коэффициент, гиперпараметр *градиентный шаг*, насколько сильно меняется вектор весов при смещении вдоль градиента.

3.3 Проблема переобучения. Регуляризация

Проблема *переобучения (overfitting)*, заключается в том, что модель, стремясь к минимизации затрат на обучение, теряет способность к обобщению. ([3])



В качестве примера рассмотрим задачу классификации. Необходимо построить такую модель, чтобы определить к какому классу - красный или синий принадлежит объект. Черная линия - приблизительно разграничивает объекты на два класса, но некоторые элементы были отнесены к ошибочному классу. Зеленая линия идеально разграничивает объекты, однако такое будет наблюдаться только на *обучающей выборке* - набором размеченных данных, на котором обучалась модель. На тестовой выборке переобученная модель будет показывать плохие результаты, так как в результате переобучения нейросеть запоминает данные обучающей выборки.
Для борьбы с переобучением применяют метод *регуляризации* - метод добавления дополнительных ограничений к условию. Рассмотрим L_2 регуляризацию, которую еще называют *weight-decay*:

$$\boxed{L_2 = \sum_i \left(y_i - y\left(t_i\right)\right)^2 + \lambda \sum_i a_i^2}$$

L_2-регуляризация призвана уменьшить слишком большие веса (коэффициенты) в уравнении, характеризующем модель.
Важно правильно выбрать λ. Если коэффициент слишком мал, то эффект от регуляризации будет ничтожен, если же слишком велик — модель обнулит все веса.

4 Феномен двойного спуска

Итак, изучив основы, можно плавно переходить к основной теме проекта. Настройка модели состоит в таком подборе весовых коэффициентов у признаков, чтобы функция потерь и на обучающей, и на тестовой выборке принимала своё минимальное значение. При этом могут возникнуть две разные трудности:
- недообучение (under-fitting), когда количества учитываемых признаков не хватает для полного описания модели. Функция потерь показывает высокие значения на обеих выборках.
- переобучение (over-fitting).
Классическая задача машинного обучения состоит в нахождении особой точки, минимуме функции потерь. В статье [3] такую точку назвали "sweet spot". Ее также можно заметить на рис. 1.
Долгое время считалось, что данная точка и является глобальным оптимумом, так как ели двигаться дальше, то попадаешь в зону переобучения, где функция потерь стремительно возрастает. Однако, если двигаться дальше и пройти определенный пик, то ошибка начнет падать, и мы снова будем "спускаться"к минимуму $L(\mathcal{H})$. Именно поэтому это явление назвали *двойным спуском* или *double descent*.

4.1 Анализ графика $L(\mathcal{H})$

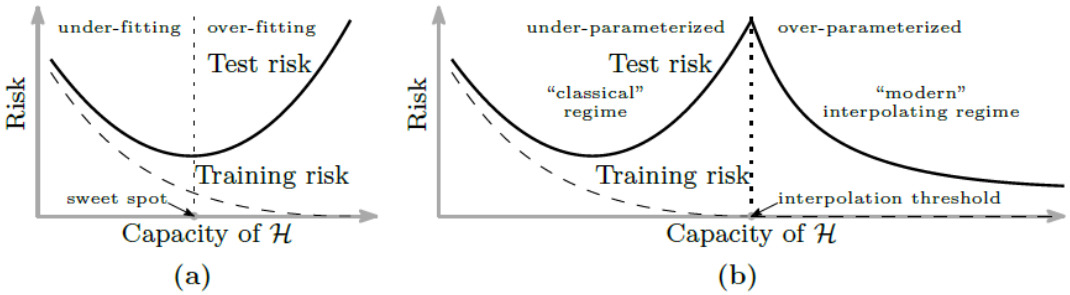


Рис. 1: (a) U - образная кривая, классическая теория машинного обучения, (b) Кривая двойного спуска

Рассмотрим точку на графике (b), которая отмечена, как interpolation threshold, порог интерполяции. Это максимум функции потерь, который достигается при равенстве количества объектов в выборке N и количеству признаков $\mathcal{H}$. Точка разделяет график на две области: under-parameterized и over-parameterized. В дальнейшем мы как раз будем второе – перепараметризированные нейронные сети.
Подчеркнём, что конкретно имеется в виду под $\mathcal{H}$. Авторы в статье [3] приводят следующее определение – количество параметров, необходимых для того чтобы задать функцию в этом классе. Это может быть так и количество признаков, так и количество скрытых нейронных слоев, или любой другой параметр, описывающий структуру сети.
Чтобы объяснить причины феномена, рассмотрим конкретную модель глубокого обучения – *Random Foirer Features*.

4.2 Двойной спуск на примере RFF

Пусть задана функция $h : \mathbb{R}^d \to \mathbb{C}$, осуществляющая отображение из пространства признаков в комплексное:

$$h(x) = \sum_{k=1}^{N} a_k \varphi\left(x; v_k\right) \quad \text{, где} \quad \varphi(x; v) := e^{\sqrt{-1}\langle v, x\rangle}$$

Функция h задает механизм работы RFF. Не вдаваясь в математические тонкости, отметим только лишь, что результат работы функции напрямую зависит от количества признаков N, то есть является моделью вида $\mathcal{H}(N)$. Именно поэтому удобно использовать RFF в исследованиях зависимости $L(\mathcal{H})$.

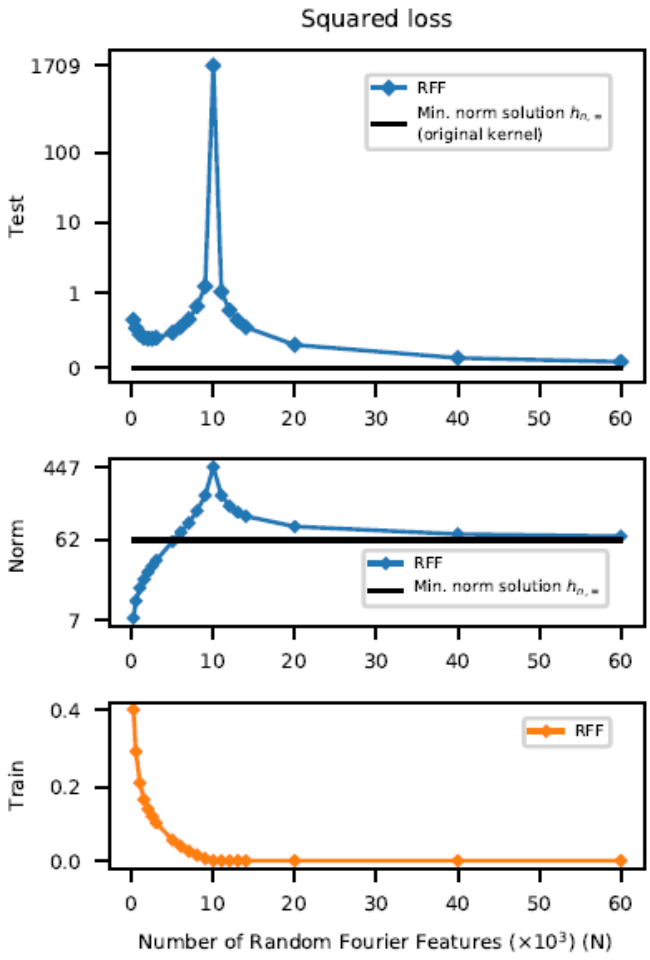


Рис. 2: Графики функции потерь на тестовых и обучающих наборах данных MNIST ($n = 10^4$).

Какие структурные механизмы объясняют форму двойного спуска? Когда количество признаков намного меньше размера выборки, то классическая статистика гарантирует, что ошибки на обучающих и тестовых выборках будут примерно равными, а добавление признаков для малых N будут улучшать работу модели. Однако по мере приближения к порогу интерполяции, весовые коэффициенты будут подбираться таким образом, чтобы идеально описать обучающую выборку, но не тестовую. Формируется пик ошибки на тестовой выборке, но далее происходит самое удивительное, что противоречит классической теории машинного обучения.

Увеличение N позволяет нам строить все более совершенные приближения к $h_{n,\infty}$ - наиболее регуляризованной, наименьшей нормировочной функции.

Таким образом, мы ожидаем, что изученные предикторы, которые также достигли свои пика на пороге интерполяции, $h_{n,N}$ будут монотонно уменьшаться по мере увеличения N, что объясняет второй сегмент спуска кривой. Это то, что мы наблюдаем на рис. 2, и действительно, $h_{n,\infty}$ имеет лучшую точность, чем все $h_{n,N}$ для любого конечного N. Существует и теорема, которая математически обосновывает нашу гипотезу, о том, что нормировочная функция h будет монотонно уменьшаться. Доказательство приводится в статье [3].

Теорема 1.
Зададим любую $h^* \in \mathcal{H}_\infty$. Пусть $(x_1, y_1), \ldots, (x_n, y_n)$ будут независимы и иметь одинаковое случайное распределение, где x_i равномерно случайно распределено в компактном кубе $\Theta^2\Omega \subset \mathbb{R}^d$, и $y_i = h^*(x_i)\ \forall\ i$. Тогда $\exists A, \exists B > 0$ такие что, для любой интерполирующей функции $h \in \mathcal{H}_\infty$ (то есть, $h(x_i) = y_i\ \forall i$), с высокой вероятностью верно следующее утверждение:

$$\sup_{x\in\Omega} |h(x) - h^*(x)| < Ae^{-B(n/\log n)^{1/d}} \left(\|h^*\|_{\mathcal{H}_\infty} + \|h\|_{\mathcal{H}_\infty}\right).$$

Минимальная нормальная интерполирующая функция $h_{n,\infty}$ имеет норму, не превышающую норму h^* (по определению), и, следовательно, достигает наименьшей границы в теореме 1. Хотя эти границы применимы только для *незашумленных* данных (в данном контексте шум — данные, не соответствующие общему правилу, по которому построен набор), они обеспечивают обоснование *индуктивного смещения* (допущения, которые нужно принять, чтобы получить неизвестный результат на новых данных), основанного на выборе решения с малой нормой. Эмпирические исследования показывают, что теорема справедлива и для частично зашумленных данных, с некоторыми поправками.

4.3 О связи нормы, гладкости, регуляризации и простоты решения

Рассмотрим следующие примеры:

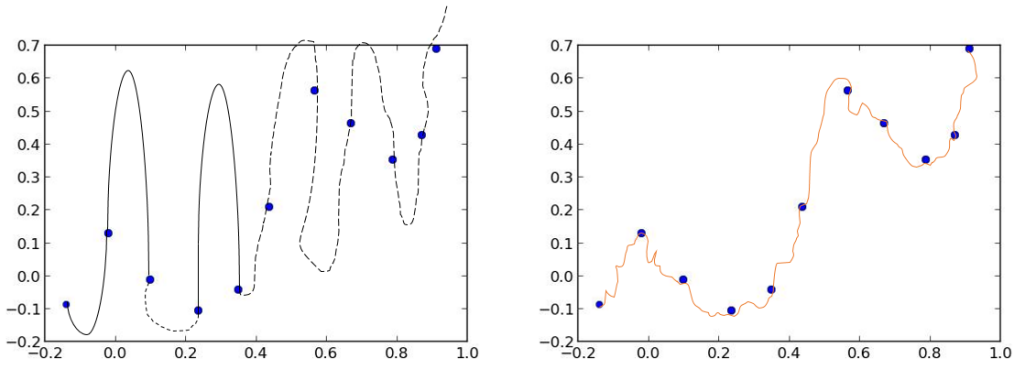


Рис. 3: (1) ненормированная неточная функция, (2) нормированная гладкая точная функция. (примеч. рисунок можно потом заменить)

На (1) схематично представлена аппроксимация полиномом высокого порядка, к которому не была применена регуляризация, поэтому он имеет пикообразную ломанную форму. Хотя это линия проходит через все точки обучающей выборки, однако совершенно не описывает тестовые точки.
Напротив, на (2), когда мы также работаем с полиномиальной регрессией, но нормализованной. Малые весовые коэффициенты обеспечивают *гладкость кривой*, которая позволяет точно предсказывать и обучающие, и тестовые значения.
Если индуктивное смещение может найти решение с меньшей нормой, то оно выберет именно его, согласно приципу *минимальной длины описания(minimum description length)*: при формировании гипотезы старайтесь минимизировать ее длину описания. Полагается, что более простые гипотезы с большей вероятностью окажутся верными. Это соотносится с еще одной научной концепцией – бритвой Оккама, которая при ранжировании гипотез рекомендует начать с наиболее простых и легких в описании.
Итак, **чем ниже норма вектора весов, тем проще функция**, но не с точки зрения количества параметров, а в том как она "передвигается"между точками тестовой выборки рис. 3(2). Обратим внимание на то, что между нормой и ошибкой существует связь (некая корреляция), рис. 2. Как только достигается пик интерполяции, то и норма, и ошибка достигают своего максимума.
Связь четырех понятий установлена, хотя не до конца ясно, как можно её строго обосновать, не ссылаясь на интуитивные субъективные представления о "простоте"решения. Это и будет посвящен следующий раздел, в котором будет дана попытка объяснить природу явления на примере стохастического градиентного спуска (SGD).

5 Плоский минимум. Формализация способности моделей к обобщению

Введём понятие *generalisation gap* (разрыв обобщения G), характеризующий разницу между ошибками $\mathscr{L}$ на обучающей и тестовой выборках.
Примечателен тот факт, что существует правило, установливающее связь между формой минимума и функцией потерь: **чем шире минимум, тем лучше модель обобщает новые данные.**

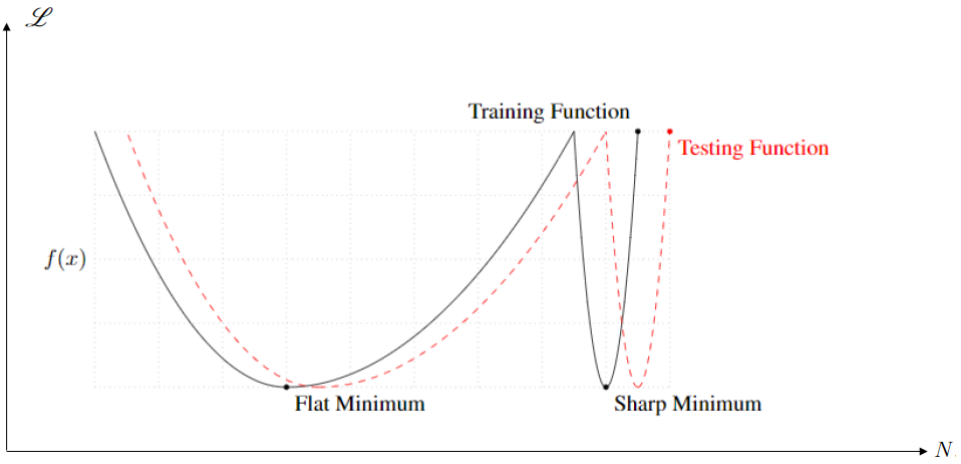


Рис. 4: Зарисовка концепции острого и полого минимума

Наглядный пример, подтверждающий справедливость этого правила, приведен в приложении А.
Как можно формализовать ширину оптимума и попробовать математически доказать правило плоского минимума (*flat minima*)?

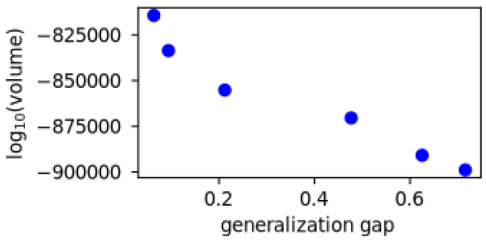


Рис. 5: Связь объема оптимума и G. Источник: [6]

Например, через объём, занимаемый минимумом в $N-$мерном простран-

стве. График, построенный на эмпирических данных, говорит о том, что существует негативная корреляция: чем больше объем, тем меньше разрыв обобщения, следовательно, тем лучше модель обобщает новые данные.

6 Двойной спуск по эпохам

Epoch-wise DD (двойной спуск по эпохам) представляет собой аналогичное явление, что мы рассматривали ранее, только по оси абцисс откладывается количество пройденных эпох, а не сложность модели. Для него справедливо всё, что было сказано выше.
Существует концепция, которое объясняет форму кривой двойного спуска, основываясь на epoch-wise модели и алгоритме SGD. Согласно ей можно выделить три этапа:

1. Начальное обобщение (simple generalisation)

Модель изучает простые закономерности. Test Error ↓

2. Запоминание (memorizing)

Если шума нет, то модель начинает концентрироваться на отдельных примерах, запоминая их и пытаясь встроить к уже изученным правилам. Если данные к тому же зашумлены, то неправильные примеры также запоминаются. Train Error ↓; Test Error ↑

3. Объединение (consolidation)

Модель объединяет успешные результаты этапов (1) и (2), путем регуляризации и упрощения функции. В результате Test Error ↓, двойной спуск завершён.

Первый этап наступает раньше второго, так как градиенты объектов, которые объединены неким неизвестным правилом, зачастую сонаправлены (коррелируют между собой), что позволяет им иметь приоритет над разрозненными частными случаями при регуляризации малых выборок в SGD – mini-batch. В исследовании [7] показывается, что в функцию потерь SGD уже встроена неявная, скрытая регуляризация (второе слагаемое) и учет всех mini-batch-ей. (на это указывает суммирование в третьем слагаемом).

$$L_{SGD}(w) = L(w) + \frac{\eta}{4}\|\nabla L(w)\|^2 + \frac{\eta}{4m}\sum_{k=0}^{m-1}\left\|\nabla \hat{L}_k(w) - \nabla L(\theta)\right\|^2$$

В таком случае становится ясным природа третьего этапа и способность к обобщению через анализ всех порций данных. Иными словами, SGD выбирает не только широкие, но и удовлетворяющие каждому mini-batch-у оптимумы (uniform optima).

7 Интуитивное обучение

Частным проявлением феномена двойного спуска на малых алгоритмических датасетах называется *гроккингом*. Слову grokking можно дать такую трактовку: понимать что-то естественно, на интуитивном уровне, но не интересуясь деталями и тонкостями. В сущности это и происходит с моделью, когда она за относительно короткий срок способна установить структуру **всего** набора данных, однако этот вывод был сделан без понимания того, *с чем* она сейчас работала: сложение по модулю, композиция перестановок или дешифровка алфавита.

7.1 Условия и скорость наступления гроккинга

Появились два существенных отличия от предыдущих разделов.
Во-первых, появилось ограничение по времени, существует некоторый лимит, после превышения которого обучение прекращается. Для нас будет важно смогла ли модель достичь понимания, или условия таковы, что она не успела или не может этого сделать.
Во-вторых, в этом разделе речь будет только о малых искусственных алгоритмических наборах данных. Для упрощения изложения назовем их **орди-нарными.**
В-третьих, будет рассматриваться только бинарные операции, вида $a \circ b = c$, где a, b, c - дискретные символы без информации об их внутренней структуре. Например, нейросеть не видит числа в десятичной системе счисления или перестановки в строчном виде. Итак, посмотрим на пример того, как в модели наступает гроккинг.

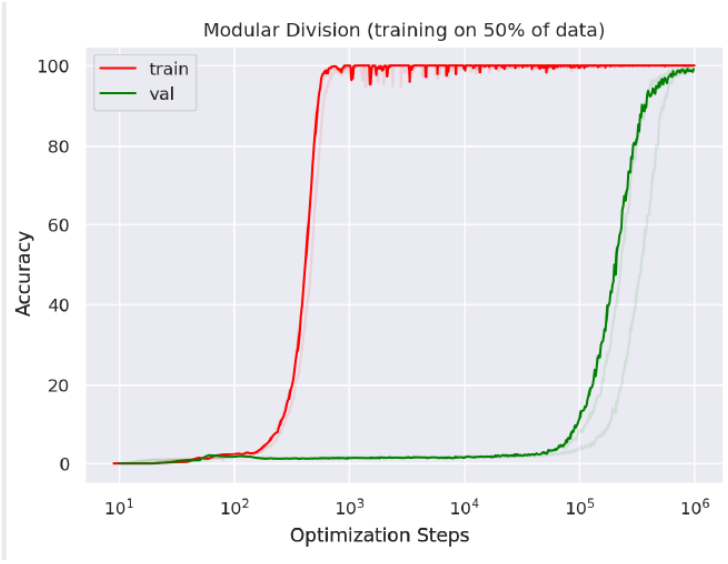


Рис. 6: Характерный пример гроккинга для операции деления с остатком на 97. Источник: [8]

7.2 Как обобщаются разные бинарные операции?

Так как операнды представляются нейросети в виде неразличимых абстрактных символов, то операции $x + y (\text{mod } p - 1)$ and $x * y (\text{mod } p)$ с простым p и ненулевым x, y неразличимы с точки зрения модели (аналогично $x - y (\text{ mod } p - 1)$, $x/y (\text{mod } p)$). Это так, поскольку каждый ненулевой остаток по модулю простого числа может быть преставлен как степень первообразного корня. Это представление показывает эквивалентность (с точностью до переименования символов) сложение по модулю $p-1$ and modular умножение по модулю p. Мы видим на рис. 7, что $x - y$ and x/y в самом деле требуют одинаковое количество обработанных данных, для того чтобы произошло обобщение.

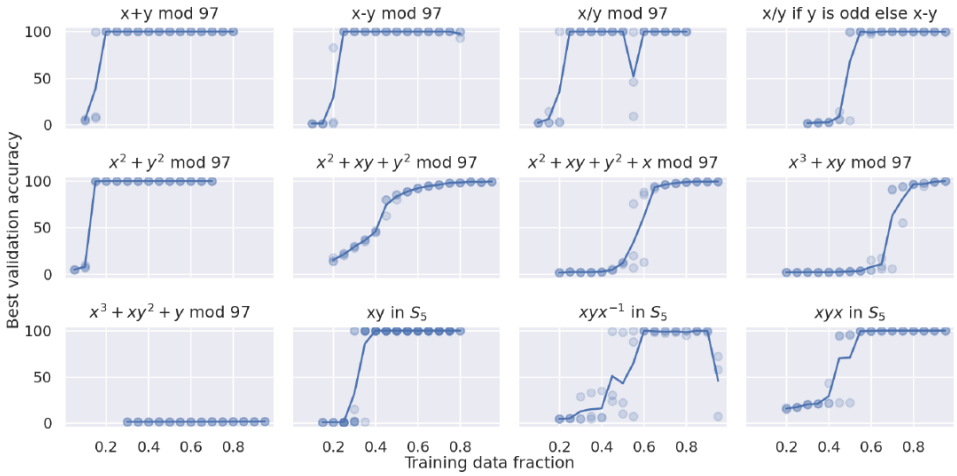


Рис. 7: Точность для тренировочных наборов для различных бинарных операций

Некоторые операции, приведенные на рис. 7 симметричны с точки зрения порядка операндов ($x+y, x*y, x^2+y^2$ и x^2+xy+y^2). Такие операции требуют меньше информации для обобщения, чем достаточно похожие, но несимметричные ($x-y, x/y, x^2+xy+y^2+x$). Это логично, так как это имеет прямое отношение к архитектуре нейросети, и понятно, что игнорирование порядка элементов позволяет ускорить работу.

Есть операции (например $x^3+xy^2+y \pmod{97}$) не приводятся к обобщению за установленный лимит по времени (95%). Нейросеть запоминает данные тренировочной выборки, не находя общий паттерн. Для такой модели данные фактически случайны.

Операция φ задана следующим образом:

$$\varphi(x,y) = \begin{cases} x/y \pmod{p} & \text{, если } y \bmod 2 = 1 \\ x-y \pmod{p} & \text{, если } y \bmod 2 = 0 \end{cases}$$

Она требует от нейросети выучить несколько простых операций - в частности, роль x должен интерпретироваться как остаток в группе по сложению, когда он в паре с четным y, и как остаток в группе по умножению, когда он в паре с нечётным y. Это показательный пример того, что модель способна обобщать комплексные модели, которые нельзя представить, как группу или кольцо.

7.3 Визуализация открытых правил. t-SNE проекции и структуры

Уточним, что такое *t-SNE проекция.* Это один из способов понизить размерность пространства признаков, сохранив при этом изначальную струкутуру и взаимосвязь между данными. Таким образом, t-SNE проекции дают двух- или трехмерное распределение информации.
В нашем случае проекция позволяет наглядно продемонстрировать, как с точки зрения модели расположены обработанные ею данные.

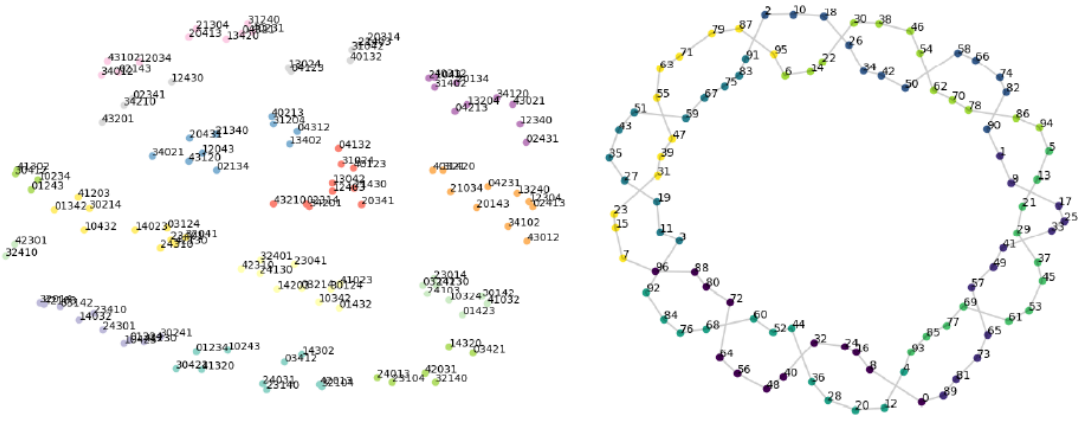


Рис. 8: **Слева:** проекция весов выходного слоя для нейросети, обученной на S_5. **Справа:** проекция весов для сложения по модулю 8 Источник: [8]

Слева мы видим, что каждый выделенный кластер есть подгруппа множества $\langle (0;3)(1;4);(1;2)(3;4) \rangle$ или сопряженные ей. Справа каждый цвет показывает каждый остаток от деления на 8, что также доказывает точность выполненного обобщения.
Хотя свойства этих математических объектов нам знакомы, можно предположить, что такие визуализации однажды могут стать полезным способом получать интуитивное представление о новых математических объектах и их внутренней структуре.
Еще одним интересным выводом ([9]) является то, что объем оптимизации, необходимый для обобщения, быстро увеличивается по мере уменьшения размера набора данных. Поэтому дальнейшие исследования в этой области могли бы ускорить работу компьютера с малыми наборами данных.

8 Практическая часть

8.1 Задание 1. Демонстрация переобучения на решающих деревьях

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot
```

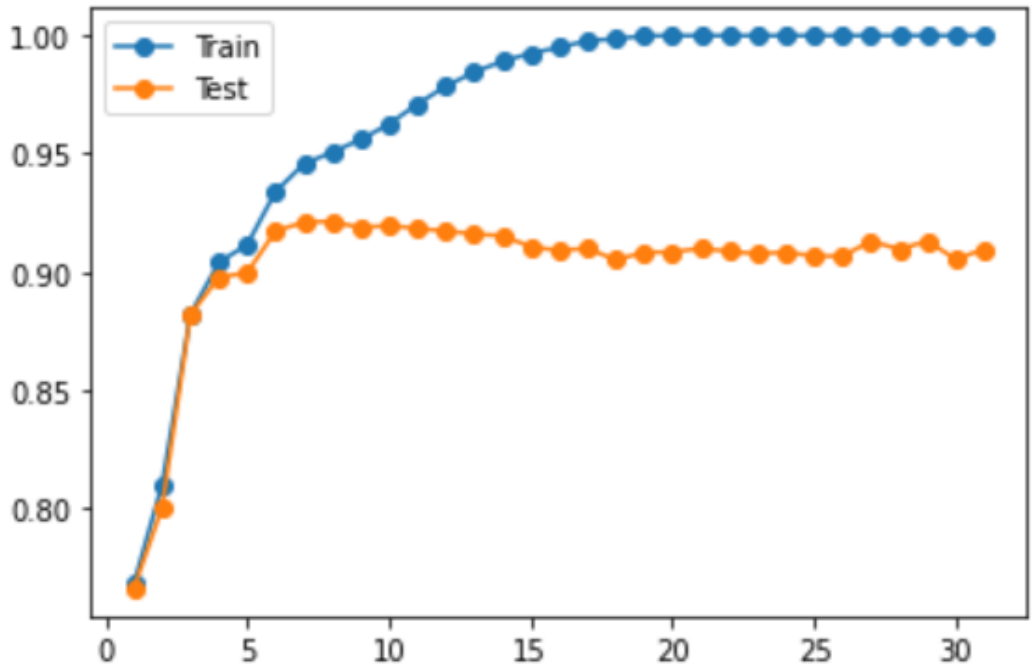


Рис. 9: **Переобучение для решающего дерева** Точность на тренировчной выборке снижается по мере увеличения глубины дерева, а на обучающей растет, демонстрируя то, что модель запоминает данные обучающей выборки.

Приведем ключевой фрагмент кода (подробнее в приложенном файле Jupyter Notebook: **Overfitting.ipynb**):

```python
# создаем искусственный набор данных
X, y = make_classification(n_samples=10000, n_features=20, n_informative=5, n_redundant=15, random_state=1)
# разделяем на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# задаем списки, в которых будут сохраняться оценки точности модели
train_scores, test_scores = list(), list()
# задаем диапазон глубины дерева
values = [i for i in range(1, 32)]
# evaluate a decision tree for each depth
for i in values:
    # вызываем модель
    model = DecisionTreeClassifier(max_depth=i)
    # обучаем ее на обучающих данных
    model.fit(X_train, y_train)
    # оцениваем качество на обучающей выборке
    train_yhat = model.predict(X_train)
    train_acc = accuracy_score(y_train, train_yhat)
    train_scores.append(train_acc)
    #  оцениваем качество на тестовой выборке
    test_yhat = model.predict(X_test)
    test_acc = accuracy_score(y_test, test_yhat)
    test_scores.append(test_acc)
    # выводим в компактном виде
    print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))
# график зависимости точности от глубины дерева для разных выборок
pyplot.plot(values, train_scores, '-o', label='Train')
pyplot.plot(values, test_scores, '-o', label='Test')
pyplot.legend()
pyplot.show()
```

8.2 Задание 2. Обучение разделению четных и нечетных перестановок (бинарная классификация)

Рассматривается группа перестановок S_5 из чисел $(1, 2, 3, 4, 5)$. В группе задана операция композиции двух перестановок: $\varphi \circ \varepsilon$. Это буквально означает, что на последовательность сначала воздействовали перестановкой φ, потом ε. Строковыми методами Python можно задать функцию перемножения перестановок:

```
def multi_permutation(a, b):
    c = [0] * len(a)
    for i in range(1, len(a) + 1):
        c[i - 1] = b[int(a[i - 1]) - 1]
    return tuple(c)
```

В дальнейшем нам потребуется цикловая запись:

Цикловая запись перестановки

$$s = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 6 & 2 & 5 & 3 \end{pmatrix}$$

Запись цикла начинается с наименьшего числа, поэтому сразу пишем (1) и смотрим куда переходит 1: в 4. Куда переходит 4? В 2. Куда переходит 2? В

1. Цикл замкнулся.
Открываем новую скобку и выбираем наименьшее число из оставшихся (3). Куда переходит 3? В 6. Куда переходит 6? В 3. Цикл замкнулся. Осталось число 5, оно переходит в себя. Итого:

$$s = (1, 4, 2)(3, 6)(5)$$

Перестановка называется четной, если число ее беспорядков четно, и нечетной в противном случае. Произведение *двух четных перестановок четно*, произведение двух нечетных - четно, произведение четной и нечетной - нечетно. **Перестановка четна в том и только том случае, когда в ее цикловой записи число циклов четной длины - четно.** Мы будем обучать модель бинарной классификации, где 0 - это четная перестановка, 1 - нечетная. Для генерации датасета нужно создать функцию определения четности перестановка. Это можно сделать через матричное представление:

```
import numpy as np
def gen_perm_matrix(s):
    n = len(s)
    A = np.zeros((n, n))
    for i in range(n):
        A[i][int(s[i])-1] = 1
    print(A)
    return np.linalg.det(A)
gen_perm_matrix('12354')
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0.]]

-1.0
```

Рис. 10: $det\ M = -1$ у нечетных перестановок, $det\ M = 1$ у четных.

Создаем набор данных и записываем его в датафрейм, который берем из библиотеки *Pandas*:

```
    index = list(range(1, 120*120+1))
x = [1, 2, 3, 4, 5]
dataset = list(permutations(x))
permutation_dict = {'a': [], 'b': [], 'result': [], 'class': []}
for a in dataset:
```

```
for b in dataset:
    permutation_dict['a'].append(tuple_to_int(a))
    permutation_dict['b'].append(tuple_to_int(b))
    permutation_dict['result'].append(tuple_to_int(multi_permutation(a, b)))
```

В файле *Permutations Classifier.ipynb* приведен код для обучения решающего дерева (подробнее про решающие деревья - в приложении **??**) на основе этих даных. Визуализация дерева выглядит следующим образом:

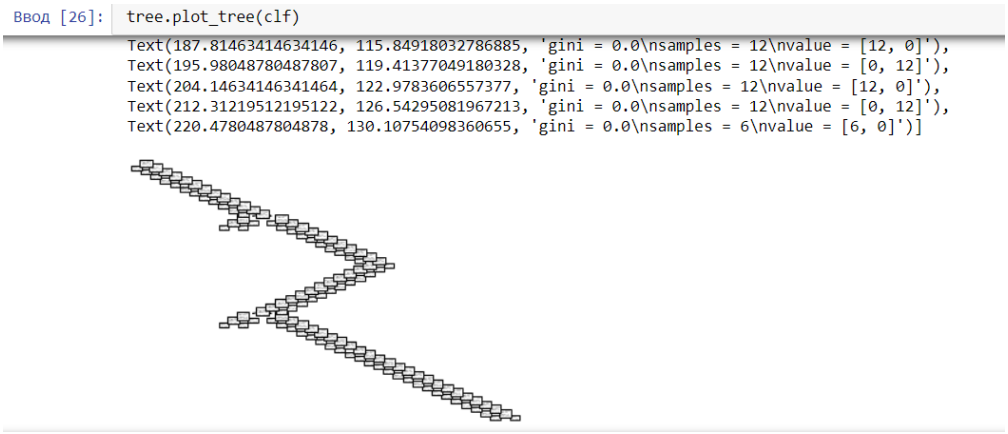


Рис. 11: Глубокое решающее дерево

Алгоритм дает идеальные ответы, безошибочно опереденяя четность и нечетнось перестановок на новых данных.

8.3 Задание 3. Исследование взаимосвязи цикловой структур перестановок и их композиции

Группа перестановок S_n может быть разделена на некоторые группы (задача многоклассовой классификации с известными ответами) по тому, как выглядят их цикловые структуры. Закодируем кластеры для группы S_5:
0 - *единичная перестановка*, **1** - *2-цикл*, **2** - *3-цикл*, **3** - *4-цикл*, **4** - *5-цикл*, **5** - *(2,2) цикл*, **6** - *(2,3) цикл*. Цель нашего исследования в том, чтобы установить связь (если она есть) между классами перемножающихся перестановок и результатом композиции.
Функция определения класса приведена в файле *Permutations Exploration.ipynb* (и проверена в приложении C).

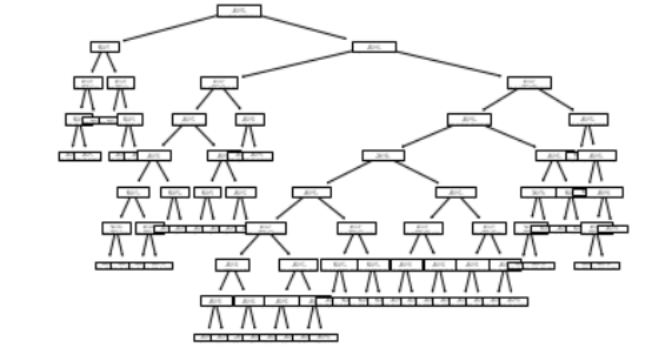


Рис. 12: Решающее дерево для задачи классификации итоговой перестановки (признаки: классы перестановок A и B)

В этой задаче мы будем использовать встроенные в библиотеку *scikit − learn* статистические метрики качества классификации.

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
0.4168771043771044
```

Используем метод $classification - report$ для характеристик модели:

	precision	recall	f1-score	support
единичная	0.00	0.00	0.00	45
2-цикл	0.80	0.01	0.02	376
3-цикл	0.40	0.50	0.44	795
4-цикл	0.51	0.77	0.62	1168
5-цикл	0.33	0.58	0.42	972
(2,2)-цикл	1.00	0.02	0.04	608
(2,3)-цикл	0.40	0.14	0.21	788
accuracy			0.42	4752
macro avg	0.49	0.29	0.25	4752
weighted avg	0.52	0.42	0.35	4752

Как видим, связь достаточно слабая, и однозначное правило не было подобрано. Иными словами, не нашлось такой связи, которая бы объясняла связь цикловых структур множителей и результата.

9 Заключение

В данной работе была дана кракая теоретическая справка по основам машинного обучения и теории нейронных сетей. Были рассмотрены ключевые

парадигмы ML - обучение с учителем и без, а также объяснены стохастические и детерминированные алгоритмы. Полученные знания позволили нам подробнее изучить главную проблему этой работы - переобучение.
Именно, то, что происходит с моделями за гранью переобучения, нас и интересовало. За точкой равенства числа признаков и числа объектов происходит феномен двойного спуска, когда ошибка на тестовом наборе вновь идет вниз. Было приведено одно из возможных объяснений феномена, которое основывается на принципе Оккама - о том, что модель всегда ищет простое решение из возможных.
Разобрались с плоским минимумом, который может формально и с точки зрения математики пролить свет на причины двойного спуска. Одним из его проявлений и ялвяется гроккинг (интуитивное обучение), когда модель, находясь в перепараметризированном состоянии, находит внутренние закономерности в данных, что доказывается методами визуализации данных, как $t - SNE$.

A Визуализация flat minima

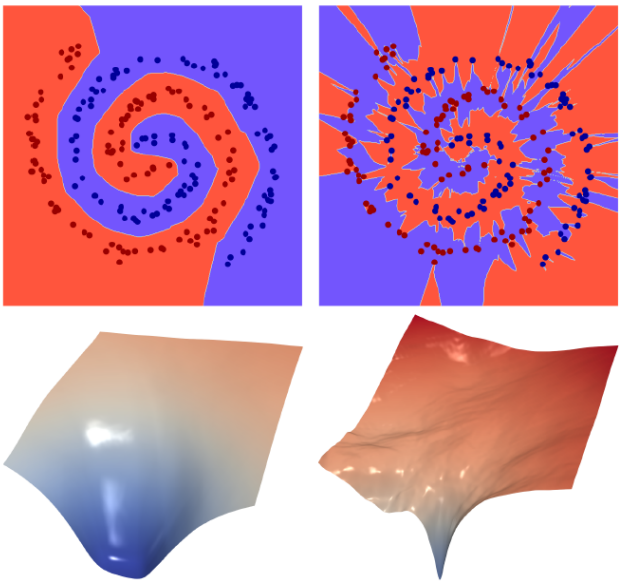


Рис. 13: Двумерные срезы функции потерь для различных классификаций

Сверху представлена схема того, как алгоритм классифицировал объекты. Понятно, что левый алгоритм справился с своей задачей лучше, чем правый. (стоит отметить, что показаны результаты на тестовых выборках, причем оба алгоритма дали хорошие результаты на обучающих). Удивительно то, что визуальные портреты $\mathscr{L}$ отличаются: минимум левой фукнции пологий, в отличие от минимума правого, который заостренный и узкий. По материалам исследования статьи [6].

B Решающие деревья

Модель, которая предсказывает значение искомой переменной, основываясь на простых правилах принятия решений, оформленных в виде конструкции "если, ..., то ...". Оформляются в виде блок-схем.

- Просто понять и интерпретировать. Деревья можно визуализировать.
- Нужна совсем небольшая подготовка данных. Нормализация и приведение к логарифмической шкале не требуется.

- Возможна проверка модели с помощью статистических тестов (accuracy score, confusion matrix, true/false positive/negative в случае бинарной классификации).

C Цикловая структура S_5

Для проверки корректности функции цикловой классификации, соотнесем результат работы программы с математически рассчитанной структурой S_5:

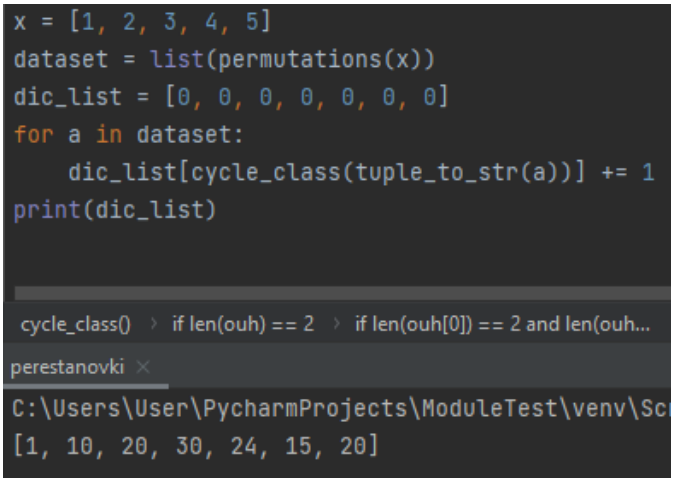


Рис. 14: Как видим, результаты даваемые программой совпадают с комбинаторным расчетом.

Таблица 1: Группа S_5

Структура	**Количество**	**Формула**
2-цикл	10	C_5^2
3-цикл	20	$C_5^3 \cdot (3-1)!$
4-цикл	30	$C_5^4 \cdot (4-1)!$
5-цикл	24	$C_5^5 \cdot (5-1)!$
(2,2) цикл	15	$\frac{1}{2} C_5^2 \cdot C_3^2$
(2,3) цикл	20	$C_5^2 \cdot (3-1)! \cdot (2-1)!$
единичная	1	-

Список используемой литературы

[1] *Анатолий Постолит* – "Основы искусственного интеллекта в примерах на Python [2021]"

[2] *Пол Дейтел, Харви Дейтел* – "Python: Искусственный интеллект, большие данные и облачные вычисления [2020]"

[3] *Андреас Мюллер, Сара Гвидо* – "Введение в машинное обучение с помощью Python [2016-2017]"

[4] *Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal* – "Reconciling modern machine learning practice and the bias-variance trade-off"[2019]

[5] *Maxim Kodryan Samsung HSE Laboratory* – "Double Descent, flat minima, and SGD"[2020].

[6] *W Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K Terry, Furong Huang, and Tom Goldstein.* – "Understanding generalization through visualizations". arXiv preprint arXiv:1906.03291, [2019].

[7] *Lei Wu, Chao Ma, et al.* – "How SGD selects the global minima in over-parameterized learning."[2018].

[8] *Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin* – "Grokking: generalisation beyond overfitting on small algorithmic datasets"[2020].

[9] *Elad Hoffer, Itay Hubara, Daniel Soudry.* "Train longer, generalize better: closing the generalization gap in large batch training of neural networks"[2018].

[10] *Anonymous authors.* "Deep Double Descent: Where Bigger Models and Big Data Hurt"[2020].