ЛЕКЦИЯ 6. МАШИННОЕ ОБУЧЕНИЕ В МАТЕРИАЛОВЕДЕНИИ

Одна из основных целей всей нашей области деятельности – это так называемый компьютерный дизайн материалов. На протяжении многих веков материалы создавались экспериментальным методом, то есть на основании опыта и интуиции предпринимались попытки угадать и синтезировать материал с нужными свойствами. Если попытка была успешной, материал оставляли и пытались его дальше как-то модифицировать, чтобы улучшить свойства. В целом, это долгий метод проб и ошибок.

Цель компьютерного метода в том, чтобы существенно сократить время на поиск таких материалов. Как это в идеале должно работать? Мы находим нужное положение атомов (на компьютере), которое должно быть, чтобы обеспечить нужные свойства, и предсказываем, как нам синтезировать данный материал. Потом проводим эксперимент и используем полученный материал там, где хотим.

Соответственно машинное обучение – важный шаг на пути к этому идеальному сценарию. Сейчас в основном работаем с двумя методами. Первый – машинное обучение, второй - из первых принципов (ab initio). Из первых принципов – это точные физические расчеты, где используется какая-то теоретическая модель. Соответственно эти расчеты связаны с квантовой механикой. То, что связано с квантовой механикой, обычно долго и дорого. Нужны целые кластеры, чтобы посчитать даже не очень сложные свойства.

Если мы хотим действительно заняться поиском материалов, обычно нужно перебрать на компьютере большое число материалов. Или если как-то хотим оптимизировать положение атомов, мы должны понимать, куда их двигать, чтобы это приводило нас в итоге к нужным свойствам, то есть нам нужно уметь быстро считать эти свойства.

Смысл машинного обучения - решить проблему скорости расчетов. Просто берем данные расчетов, которые у нас есть, и обучаем какой-то метод

машинного обучения, чтобы она предсказывал эти свойства. Обычно это получается менее точно, чем с помощью теоретических методов. Но не всегда нужна эта точность. Нам нужно быстро оценить свойства для большого количества материалов. Те материалы, которые мы считаем хорошими, потом можно пересчитать, используя теоретические методы. Машинное обучение дает возможность быстро оценить свойства.

Бывает такое, что для каких-то свойств нет теоретической модели. Не придумали, как ее можно в принципе нормально считать. Типичным примером является сверхпроводимость. Соответственно, если у нас есть какие-то данные, полученные, например, из эксперимента, мы можем обучить какую-то модель машинного обучения, которая будет делать предсказания. В этом случае, мы сможем оценить свойства материала.

Что из себя представляет модель машинного обучения? Есть два класса моделей: параметрические и непараметрические. Про параметрические модели можно думать, как об очень сложной функции. У нас есть простые функции, например ax^2+bx+c, где три коэффициента. Модель машинного обучения – это тоже функция, но в ней может быть 100 тысяч параметров, и у нее сложное поведение.

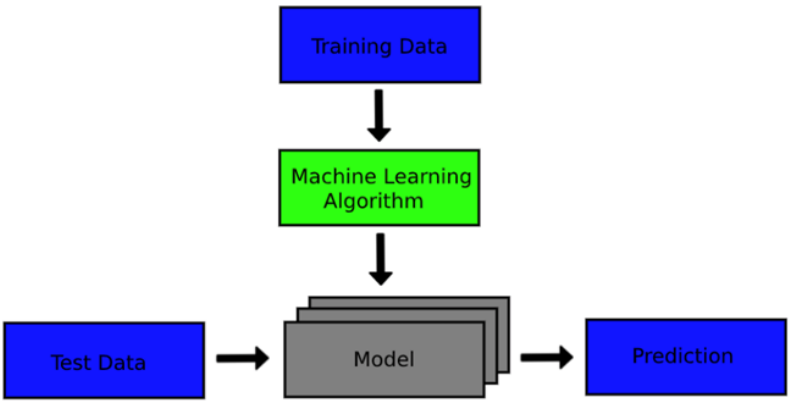
Соответственно, параметрическая модель – это такая сложная функция, такой черный ящик из коэффициентов, которые подстраиваются под данные.

Есть непараметрические модели, где нет какой-то заданной функции. При этом они тоже могут быть достаточно сложными. Например, метод ближайших соседей (KNN).

Например, вы гуляете в лесу, видите интересное дерево, хотите предсказать тип этого дерева: береза это или осина. Смотрите на 5 ближайших к нему деревьев, если три из них – осина, предсказываете, что ваше дерево это тоже осина (то есть определяете по типу ближайших соседей). Лес – такой простой пример. Можно располагать признаки в каких-то более сложных пространствах, но в целом здесь нет никаких параметров, то есть мы работаем непосредственно с данными.

Основной принцип заключается в том, что мы берем данные из более тяжелых теоретических расчетов, либо из экспериментов и берем модель машинного обучения, которую пытаемся настроить так, чтобы она по данным, которые у нас есть, предсказывала имеющиеся результаты. После этого мы сможем ее использовать на части данных, для которой мы не знаем результатов. Берем новую часть данных, подаем на вход этой сложной функции и получаем интересующие нас результаты (рис. 1).

- Мы можем использовать данные, которые у нас есть (экспериментальные или полученные теоретическим путем) и обучить модель получать ответы для других данных



- Лучшие результаты модели могут быть затем перепроверены более точными методами

Рисунок 1. Основные принципы

Сейчас время, в которое машинное обучение сильно развивается, потому что:

1. Во-первых, повышается производительность работы с данными. Если раньше данные часто были частью каких-то статей, особо не структурированные, лаборатории неохотно делились этими данными друг с

другом, то сейчас налажен какой-то процесс, где данные собираются в одном месте, проверяются, обрабатываются.

Уже сейчас есть большие базы данных, в том числе в открытом доступе, из которых можно брать какие-то материалы, использовать их для новых моделей. Среди материалов можно производить поиск по интересующим нас свойствам. Такие базы данных позволяют обучать модели машинного обучения.

2. Сами алгоритмы тоже развились очень сильно. За последние пять лет все очень сильно изменилось, появились новые методы, новые типы нейронных сетей, новый тип алгоритмов обучения этих нейронных сетей.

При том же количестве данных обучение моделей стало гораздо более эффективно.

3. Физическая сторона вопроса тоже очень сильно развилась. В частности, можно считать какие-то свойства, которые раньше нельзя было считать, можно понимать, как связаны разные свойства. Например, из этой связи получать сначала одно свойство, допустим, предсказывать сперва одно свойство и из него получать другое свойство. То есть появилось много разных типов, как можно с этими данными работать с физической стороны.

4. Также эти методы стали интересны индустрии. Раньше у науки не получалось дойти до практического применения, потому что не было хороших методов машинного обучения, поэтому придумать нужный материал было достаточно сложно. Сейчас то время, когда появляются достаточно уверенные результаты. Например, компании Huawei и Sony, заказывали расчеты по поиску материалов с заданными характеристиками.

Модель машинного обучения - это очень большая функция, для которой мы пытаемся подобрать коэффициенты. Здесь встает классическая проблема, представленная на рисунке 2.

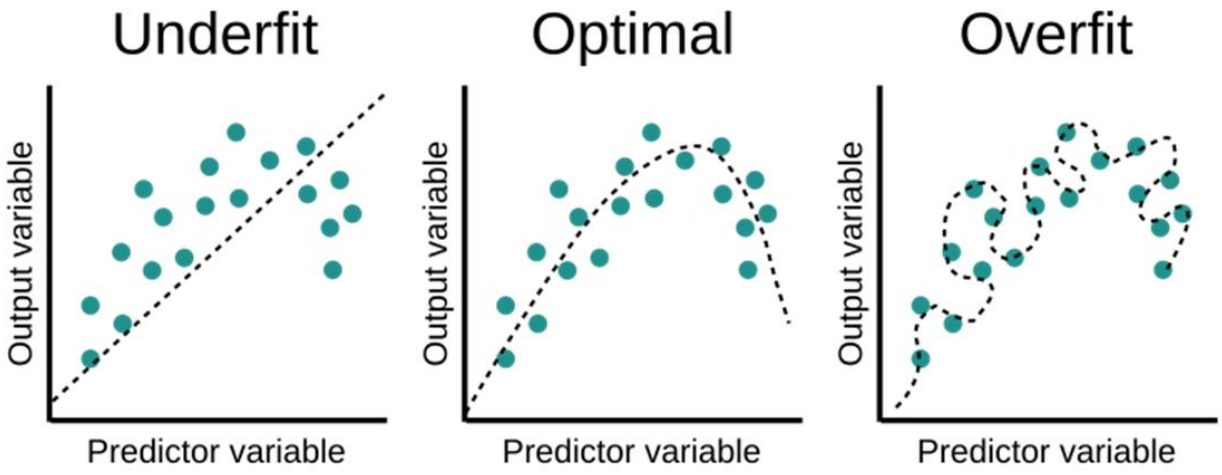


Рисунок 2. Машинное обучение: overfitting&underfitting

Мы можем «недофитить» - недообучить (если мы используем слишком грубую модель, обычно это модель с небольшим количеством параметров), «зафитить» оптимально или «перефитить» - переобучить (если мы используем очень гибкую модель). Если мы хотим, чтобы модель обучения описывала данные за пределами известных нам точек, мы должны оптимально обучить ее.

Об этом можно также думать с точки зрения дисперсии. Смещение, то есть насколько то, что предсказывает модель, отличается от того, что на самом деле, это и есть дисперсия. Здесь можно в двух плоскостях об этом говорить. Во-первых, насколько при небольшом сдвиге данных отличается. А во-вторых, можно думать о том, что мы взяли какую-то часть данных, обучили модель, потом другую часть данных, снова обучили модель, и нашли разница между ними - это и будет дисперсия.

На рис. 3 по серединке у нас пример, высокого смещения (синяя функция, которые предсказана, сильно отличаются от красной функции, которая на самом деле была использована), но при этом низкая дисперсия (если мы возьмем какие-то другие точки, и будем искать то же самое, у нас примерно те же линии и получатся).

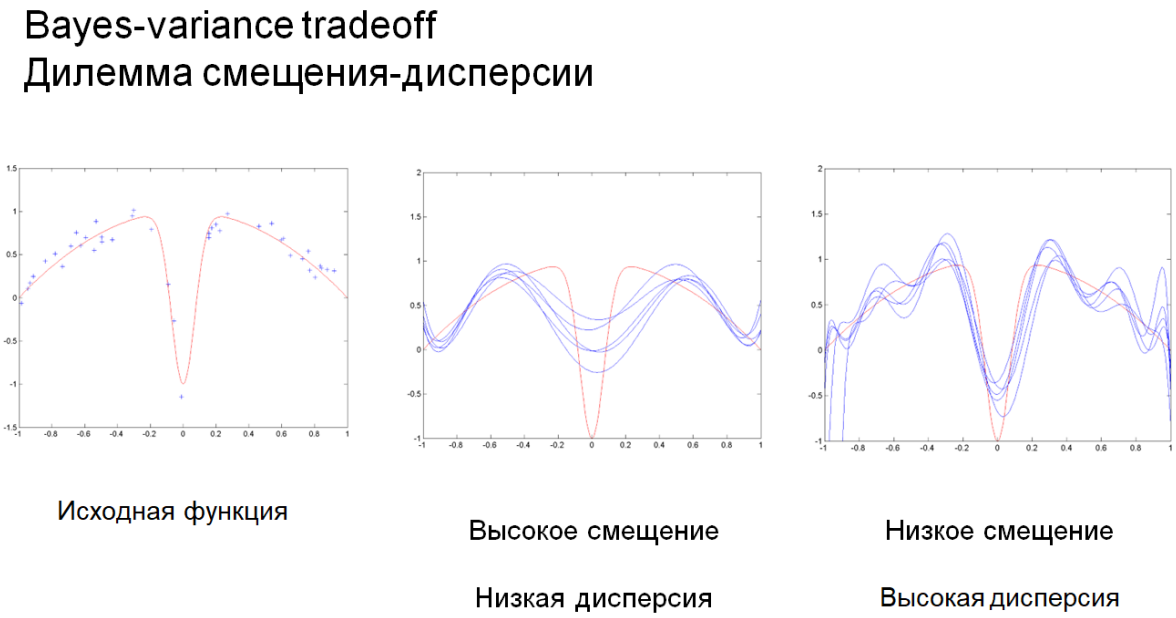


Рисунок 3. Дилемма смещения-дисперсии

На правой картинке у нас, наоборот, низкое смещение (синие функции почти совпадают с красной), однако видно, что они идут волной - это обусловлено высокой дисперсией (функция подстраивается под точки). По сути, эти точки отличаются от красной кривой, потому что есть дополнительный шум, который мы не можем предсказывать.

В идеале, должна быть модель, у которой одновременно низкие и смещение, и дисперсия, но на практике такое редко удается достичь, обычно приходится чем-то жертвовать. Искусство машинного обучения заключается в подборе оптимума.

Соответственно есть данные, на которых мы хотим обучить модель. Мы делим их на две части. Одна называется train, другая – test, то есть тренировочные и тестовые части. Мы можем делить их в разных пропорциях, например, 80 % - train, 20% - test. Мы используем train, чтобы обучить модель, а на части данных test мы проверяем результат (рис. 4).

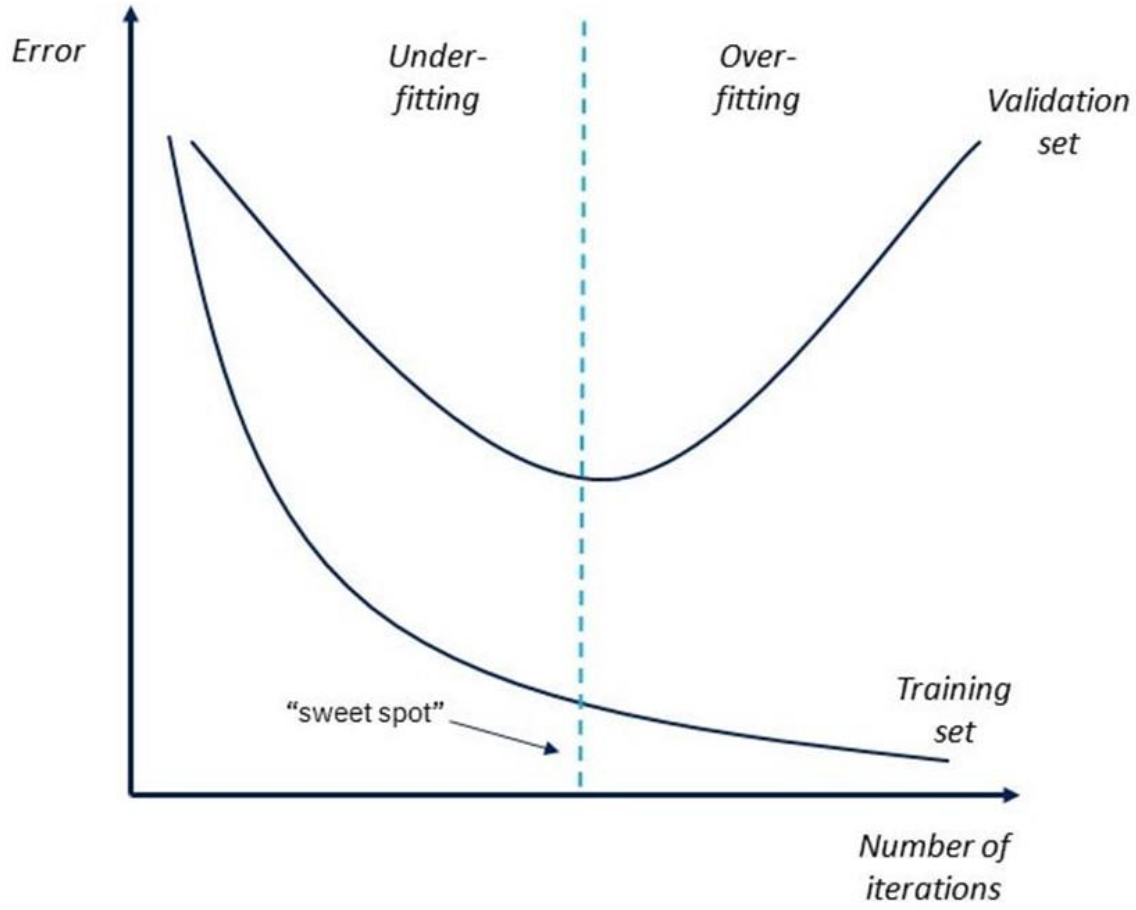


Рисунок 4. Обучение модели

error - ошибка
number of iterations - число итераций
under - "недообучение"
over - "переобучение"
sweet spot - оптимум
training set - тренировочный набор данных
validation set - тестовый набор данных

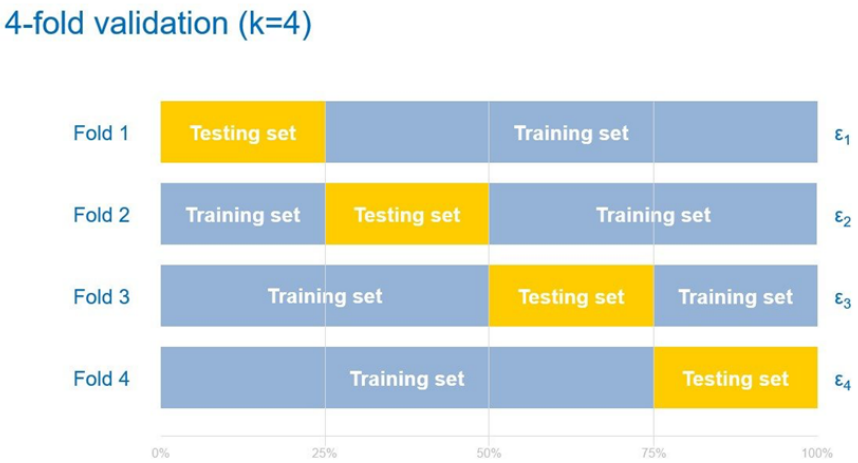
На рисунке 4 показана типично встречающаяся картинка. Нижняя кривая соответствует тренировочному набору, у нее ошибка уменьшается (данные все лучше и лучше подгоняются). Validation - синоним test - сначала

видим, что результат улучшается, потом ухудшается, то есть в какой-то момент ошибка начинает расти (когда ошибка на тесте растет - это типичный оверфиттинг). В идеале, мы должны выбрать оптимальный вариант с минимальной ошибкой.

Минус такого метода в том, что если мы делим на train и test, у нас в test могут случайно попасть какие-то специфические данные. Например, мы обучаем какую-то модель, где есть натрий, а в test попадают данные только с углеродом. Достаточно странно оценивать качество модели, которая обучалась на натрии, а тестировалась на углероде. Немного утрировано, но смысл такой. В test могут попасть данные, которые не совсем представляют общее распределение, поэтому был создан метод кросс-валидация.

Метод кросс-валидации работает несколько точнее. Разделим все данные на четыре части - на трех из них мы обучаем, а четвертую используем для теста. Потом выбираем в качестве тестовой другую часть данных, и проводим то же самое, и так несколько раз. Для каждого случая получаются разные результаты. Мы их усредняем и получаем общую картину. Минус такого подхода достаточно очевиден - вместо одного раза, как в предыдущем случае, модель нужно обучать четыре раза. Это часто на практике бывает затратно, потому что эти модели не так уж легко учатся (рис. 5).

Деление на train и test



1. Кросс-валидация является более точным подходом к оценке точности модели, но на практике часто затратна.
2. Иногда данные делятся на 3 части: train/validation/test

Рисунок 5. Деление на train/test

Иногда данные делится на три части: train/validation/test. То есть сначала мы обучаем модель на train, затем проверяем на validation и, возможно, используем, чтобы улучшить какие-то параметры нейронной сети.

Потом проводим финальное предсказание на test, чтобы понимать, какая у нас ошибка предсказания.

Что такое нейронная сеть? Наверное, это один из наиболее часто используемых методов машинного обучения. У нас есть какие-то данные на входе, и нейронная сеть - это набор весов. Сначала эти данные просто входят в каждый из весов и там на него умножаются, потом суммируются, и дальше к ним применяется какая-то нелинейная функция (рис. 6).

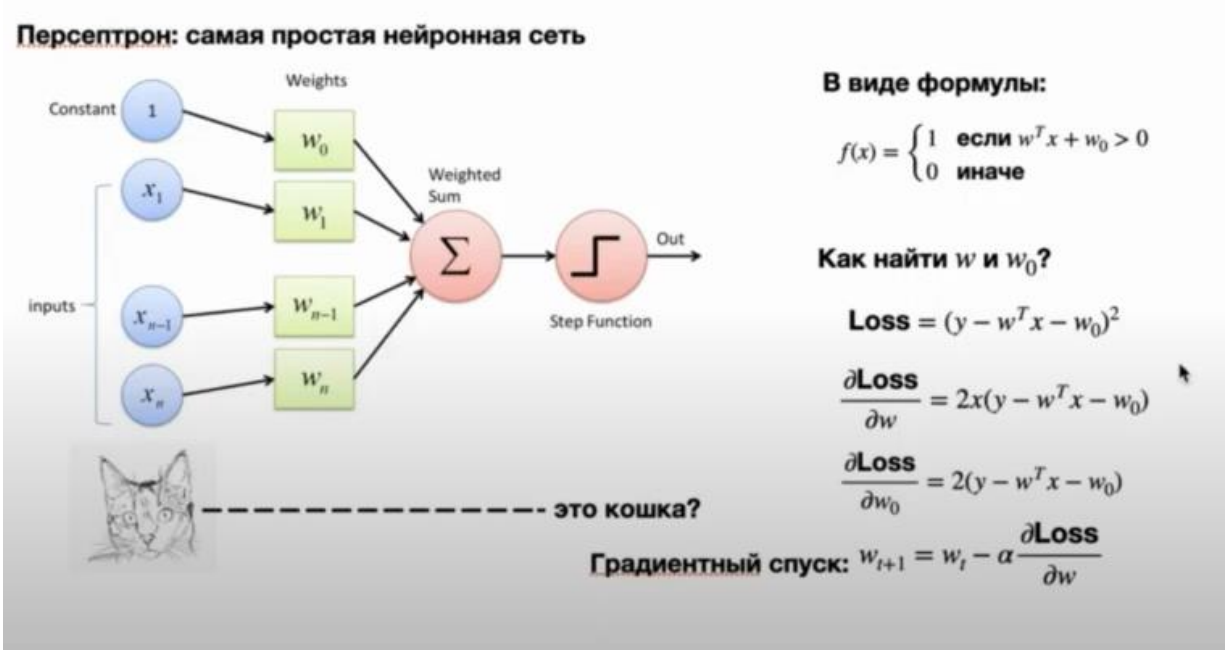


Рисунок 6. Персептрон: самая простая нейронная сеть

Если представить, что, данные представляют закодированную картинку кошки, то они входят просто числами, которые кодируют эту картинку кошки, умножаются на веса, складываются, применяется это нелинейная функция и получается какой-то результат. И результат мы можем представить как некую нелинейную функцию. Если результат больше нуля, мы считаем его равным 1, что будет означать, что мы предсказали кошку; если нет, то считаем, что результат равен 0, и мы предсказали что-то другое.

Вопрос в том, как в таком случае найти веса, которые бы предсказывали, где кошка, а где нет. Для этого нужно сначала нужно записать ошибку, которую мы хотим минимизировать. И ошибка (формула 1) - это разница между правильным классом (мы же знаем ответы для этих картинок) и тем, что выдает нейронная сеть.

$$\text{Loss} = (y - w^T x - w_0)^2$$

Соответственно мы можем взять производные этой ошибки по весам и обновлять эти веса в соответствии с их производной. Это называется градиентным спуском. То есть производная показывает нам степень

изменения этого веса, если мы постепенно вычитаем, в итоге приходим к минимуму. Большинство методов обучения нейронных сетей основано на этом градиентом способе.

Например, в примере, где у нас ответы только 1 и 0, можно использовать формулу бинарной кросс-энтропии.

$$H = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \tilde{y}_i + (1 - y_i)\log(1 - \tilde{y}_i)$$

y_i - это настоящие классы: 0 либо 1, который мы знаем у картины.

$\tilde{y}_i$ - то, что показывает сеть.

Соответственно можем видеть, если у нас y_i равно 1, например, у нас правая часть исчезает вообще, потому что 1 минус 1 равно 0. И слева просто видим логарифм $\tilde{y}_i$. Чем больше он, тем лучше, потому что тем меньше кросс-энтропия. Таким образом, вводя такую ошибку, мы тоже можем достигать того, чтобы предсказания стремились к правильным ответам.

Если сравнить его с предыдущей функцией, то в большинстве практических случаев между ними будет не очень большая разница. Формула кросс-энтропии считается правильнее со статистической точки зрения. Такой способ вычисления ошибки должен давать более стабильные результаты. Но в целом, если применять предыдущую формулу, разницы большой не будет.

Мы рассмотрели случай, когда у нас есть один персептрон, то есть одна нейронная сеть, один слой весов. Теперь можем представить случай, когда у нас нескольких таких слоев из коэффициентов (рис. 7).

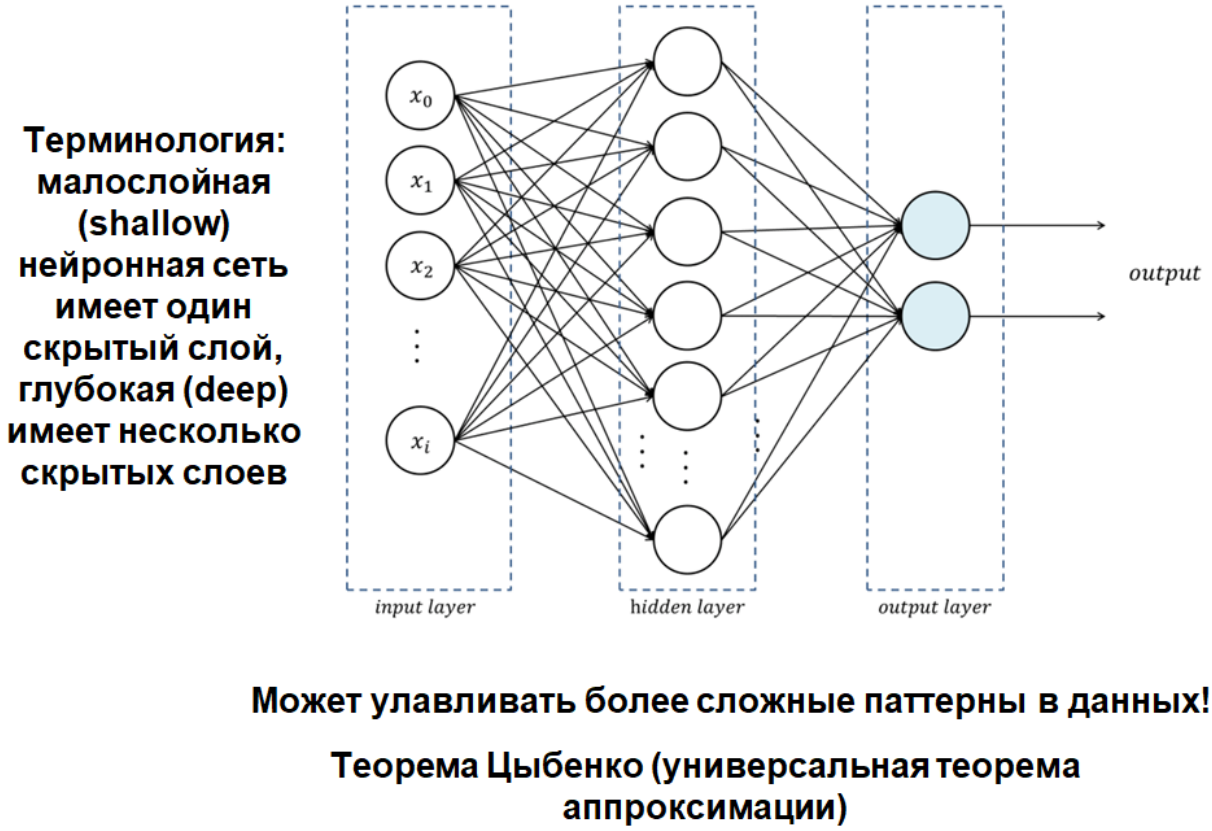


Рисунок 7. Случай множества персептронов

Дополнительный слой, который появляется, называется скрытым. На левой стороне становятся данные, умножаются на веса, дальше применяется какая-то нелинейная функция, эти результаты передаются следующему слою, где будет то же самое (может быть не один, а много таких слоев). В итоге, мы получаем ответ, который нам нужно оптимизировать под данные, которые у нас есть.

Есть доказанная теорема Цыбенко, она может описывать вообще любые непрерывные функции. Например, есть функция, которую мы хотим предсказать, этот метод, в принципе, может описать при достаточном количестве этих весов.

Возникает вопрос, зачем вообще тогда такое большое количество слоев у этих нейронных сетей обычно используется? Не 2 слоя, как здесь, а 10 или 50 слоев? Потому что такие системы легче обучать, за то же время получаются более точные результаты. Много причин, почему это так, не всегда понятных.

Но на практике получается, что с добавлением слоев мы будем получать лучшие результаты.

Можно по-разному выбирать нелинейные функции. Есть некоторые стандартные функции, допустим, sigmoid, есть очень простой пример ReLU- где если на вход поступает отрицательное значение, мы выдаем 0, поступает положительное значение, мы просто их оставляем (рис. 8).

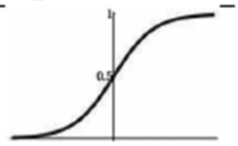
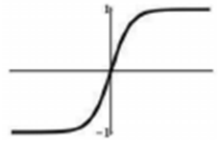
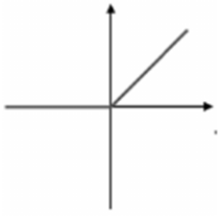
Name	Function	Derivative	Figure
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))^2$	
tanh	$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) \begin{cases} 0 & if\ x < 0 \\ x & if\ x \geq 0. \end{cases}$	$f(x) \begin{cases} 0 & if\ x < 0 \\ x & if\ x \geq 0. \end{cases}$	
Softmax	$f(x) = \frac{e^x}{\sum_i e^x}$	$f'(x) = \frac{e^x}{\sum_i e^x} - \frac{(e^x)^2}{(\sum_i e^x)^2}$	

Table 2.1: Non-linear activation functions.

Рисунок 8. Виды нелинейностей

Казалось бы, не очень сложная идея, но она используется в огромном количестве нейронных сетей, которые используются во многих современных приложениях. Именно такая функция хорошо работает для нейронных сетей.

Еще одна особенность нейронных сетей состоит в том, что слои идут друг за другом, и мы можем использовать обратное распространение ошибки.

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial C} \frac{\partial C}{\partial w}$$

Если мы идем с конца, мы можем пересчитывать градиенты, то есть сначала рассчитать градиенты первого слоя после выхода, потом второго слоя после выхода, третьего слоя после выхода и так далее. Уже считая эти градиенты, мы используем градиентный спуск, чтобы находить правильные веса.

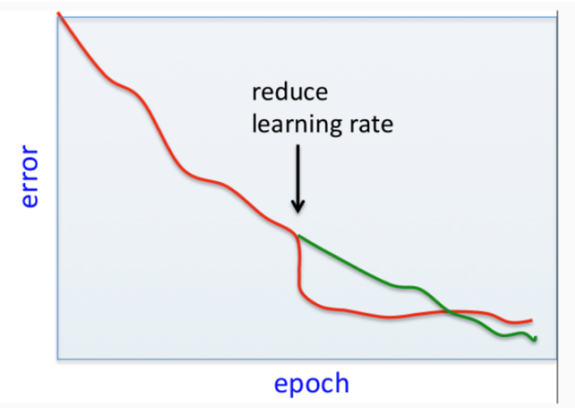
На практике часто используются адаптивные методы градиентного спуска для обучения этих сетей: Adam, Adadelta, Adagrad и др..

Еще важно отметить, что этот градиентный спуск гарантирует сходимость только к локальному минимуму. В целом функция ошибок от весов, которые у нас есть в этой нейронной сети, может очень сложно выглядеть: есть пологие участки, множество точек, где производная равна нулю - локальный минимум или максимум; и мы почти всегда сходимся к какому-то локальному минимуму, но этого достаточно для того, чтобы модель хорошо работала. Локальный минимум будет каким-то небольшим значением ошибок.

При обучении часто возникают какие-то проблемы. Например, паралич сети. Если мы обратно пересчитываем градиенты из предыдущих, часть из них внезапно становятся очень маленькими и, соответственно, все последующие тоже будут очень малыми. Если градиент маленький, то веса перестают обновляться и наступает паралич сети. Бывают такие нестабильности в обучении сложных нейронных сетей.

Здесь важно отметить, что для обновления весов используется GPU/TPU. Они отличаются от CPU тем, что позволяют делать много мелких операций, в частности, перемножать матрицы, что нужно для подсчета градиентов и обновления весов. Таким образом, на процессорах GPU/TPU обучить сеть получится гораздо быстрее (рис. 9).

- Что если уменьшать шаг градиентного спуска?



$$\theta_t = \theta_{t-1} - \eta \nabla E(\theta)$$

$$\eta = \eta_0(1 - T/T_0), \eta = \eta_0 \exp(-T/T_0)$$

Рисунок 9. Адаптивные методы

Кроме того, что можно использовать градиентный спуск, где мы просто вычитаем градиент, умноженный на какой-то коэффициент, мы можем этот коэффициент как-то подбирать. То есть, допустим, мы как-то медленно оптимизировали веса, потом мы берем и меняем их в какой-то момент - сначала коэффициенты меняются довольно быстро, а когда модель дает уже хороший результат, вносятся лишь небольшие изменения параметров. Это позволяет улучшить результаты обучения.

Можно отслеживать изменение отдельных параметров методами Adam, Adadelta, Adagrad.

В принципе, есть идея использования не градиентного спуска, а оценки производной второго порядка (у производной более точные шаги). Для серьезных моделей гессиан посчитать сложно, никакого выигрыша в итоге от этого нет.

Как вообще строится это обучение?

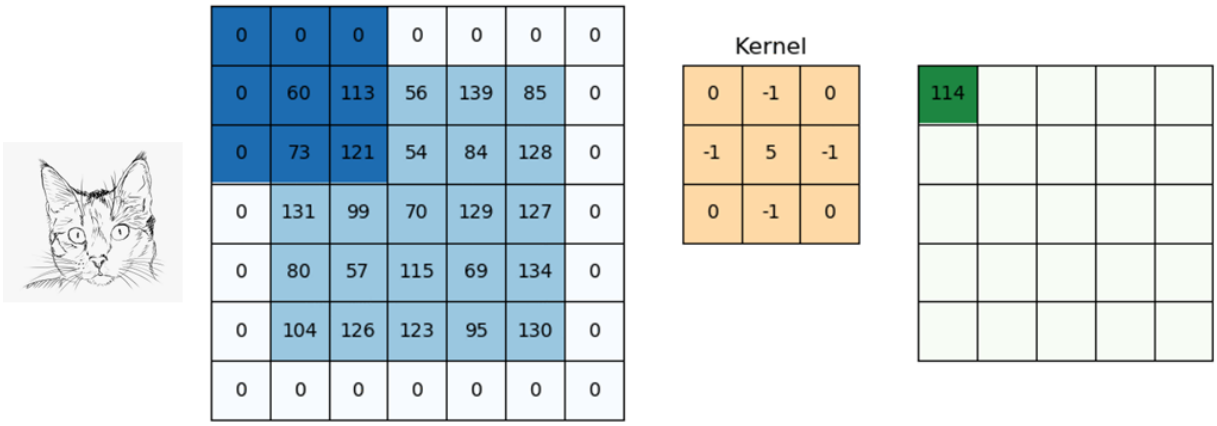
У нас есть dataset, который мы называем train. Мы берем из него маленькие кусочки данных. Например, если у нас 1000 картинок, мы берем 100 из них и подаем на вход сети, затем считаем выход и ошибку, дальше используем обратное распространение ошибок и обновляем веса. Потом мы берем следующие 100 картинок и проделываем то же самое, и т.д.. Когда мы пройдем по 100 картинок весь dataset, мы сможем сказать, что случилась эпоха. Нейронная сеть обучается в несколько таких эпох.

Зачем вообще нужно выбирать по 100 картинок? Почему нельзя сразу отправить весь этот train dataset и обновить веса, снова отправить, снова обновить, и получить такой градиентный спуск? Причина в том, что вычислительно проще считать градиенты для 100 картинок, чем сразу для всех, тем более что обычно dataset большие. Вторая причина, тоже немаловажная, заключается в том, что мы получаем стохастичность. То есть если бы мы брали все картинки, то мы получали бы более точное значение градиента. Раз мы берем 100 картинок, мы получаем грубое приближение к нему, то есть у нас постоянно будет какая-то разница между настоящим градиентом и рассчитанным. Из-за этого у нас шаги идут не строго к минимуму, в них как будто добавляется шум. Это приводит к тому, что если у нас встречается какой-то локальный минимум, мы можем его просто проскочить. Часто оказывается, что это приводит к моделям, которые лучше обучаются.

Существует также свертка. Соответственно нейронные сети в таком составе называются сверточными сетями. Идея в том, что у нас есть

изображение, записанное какими-то числами, и у нас есть так называемое ядро (рис. 10).

Рисунок 10. Свертка



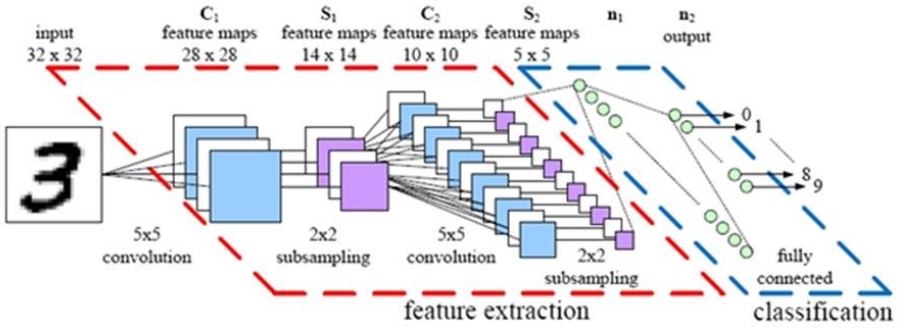
В данном случае это желтый квадратик. Если мы представим, что мы это ядро наложили на это синее место, можем просто посчитать произведение чисел, которые будут стоять друг над другом. Потом просто сложить. Соответственно то, что получилось, мы записываем в таблицу справа. Мы начинаем двигать это ядро на клетку вправо и снова записываем число, потом снова на одну клетку вправо, снова записываем число. Таким образом у нас заполняется правая зелёная таблица.

Сейчас очень много качественных нейронных сетей, которые работают с изображениями, в том числе за счет сверточных слоев, использующих некую локальность признаков. Например, посмотрим на кошку, у нее есть ухо. Когда квадратик проходит через ухо, он может это зафиксировать, смотрит на какие-то детали, которые рядом друг с другом находятся. Если мы возьмем этого кошку, возьмем все числа и выпрямим в прямоугольник, то есть сделаем из них один вектор, вся информация теряется. Поэтому нейронной сети будет сложнее детектировать локальные признаки. Но благодаря сверткам это получается, причем они могут находиться на разных слоях, в зависимости от слоя они могут фиксировать разные признаки.

Есть также операция pooling. Если мы работаем с картинкой, нам в какой-то момент нужно получить предсказание - перейти от этой картинки к вектору. Один из способов это сделать – постепенно уменьшать размеры картинки. Например, есть MaxPool, и он может брать какой-то участок этой картинки, вычислять максимальное значение и только его оставлять. Есть также AvgPool, который оставляет среднее значение и MinPool, сохраняющий минимальные значения.

Можно с каким-то шагом делать делать свертки квадратиком, и тогда у нас естественным образом уменьшается размер изображения.

Также часто применяют каналы, то есть исходная картинка - это три слоя на самом деле. И мы можем таким образом, используя свертки, делать много таких каналов. Они будут уже представлять не цвета, а что-то более сложное. Математически получается конструкция, где из трех каналов получается много каналов, потом еще больше. Часто они каким-то образом регулируются (рис. 11).



Нейронная сеть может состоять из нескольких параллельных слоев, которые называются каналами (channels)

Рисунок 11. Каналы (channels)

Сверточные сети делают из картинки одномерный вектор, просто набор чисел. Его можно назвать дескриптором (вектор), он описывает эту картинку. Часто этот вектор можно использовать в других целях (рис. 12).

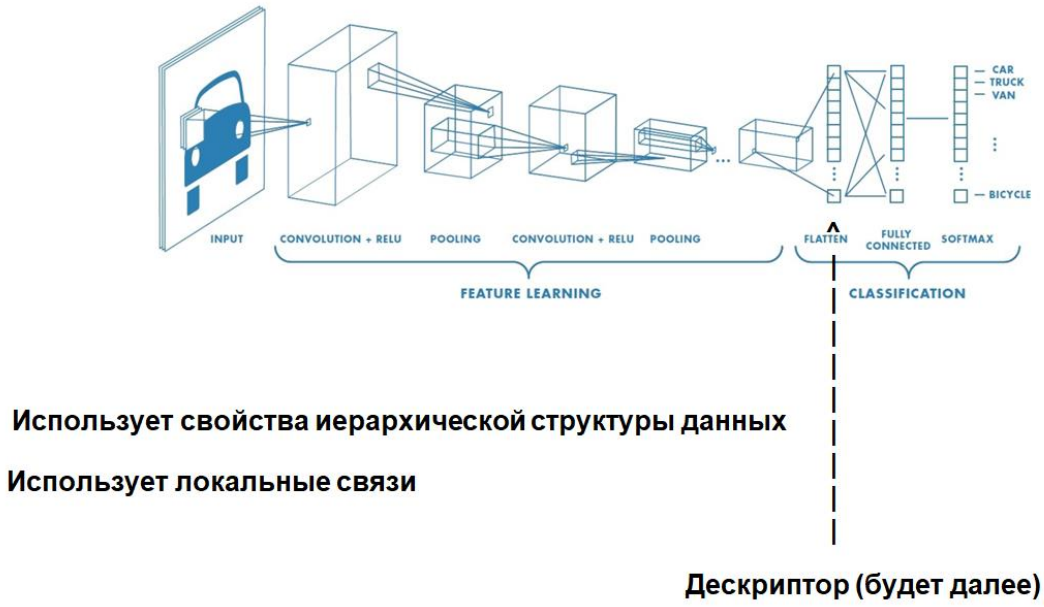


Рисунок 12. Сверточные сети

Часто этот вектор можно использовать в каких-то других целях. Есть понятие BatchNorm. Мы можем регулировать веса внутри сети, чтобы они не могли принимать произвольные значения, и за счет этого мы регулируем нашу модель, делаем ее более гладкой, т. е. снижаем тенденцию к overfitting.

Есть такое правило, что нелинейность ставится после BatchNorm. Обычно это приводит к более хорошим результатом. И можно примерно понять, почему так (рис. 13).

BatchNorm

Убирает внутренний сдвиг переменных (internal covariate shift)

$$y_k = \gamma_k \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\mathrm{Var}[x_k]}} + \beta_k$$

Дополнительные переменные нужны, чтобы не пропали нелинейности

Рисунок 13. BatchNorm

Как видно из рисунка 14, график смещается, что хуже для нейронной сети, хотя на самом деле часто разница не такая большая.

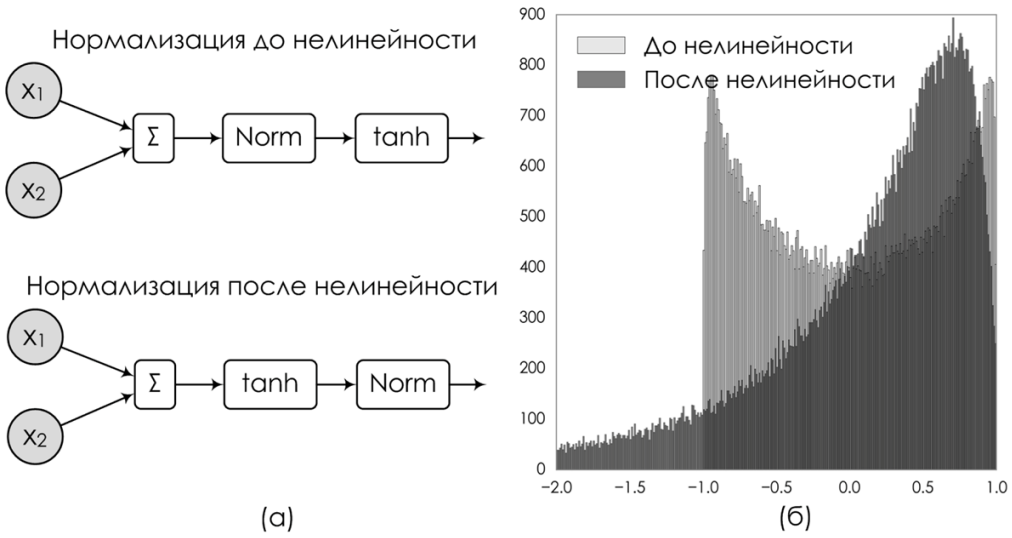
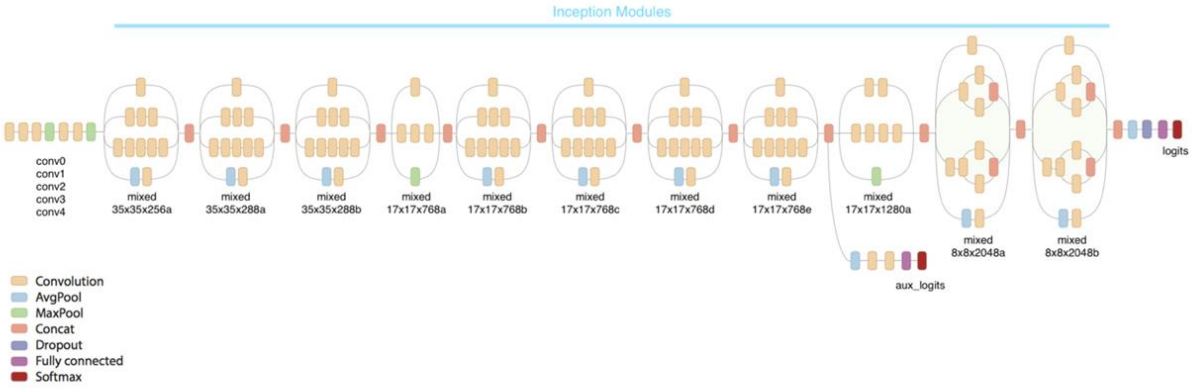


Рисунок 14. BatchNorm

Вот пример, как реальная сверточная сеть для анализа изображений может выглядеть. Это, в частности, нейронная сеть от Google (рис. 15).



Пример сверточной сети, используемый для классификации изображений с хорошей точностью

Рисунок 15. Сверточные сети (реальный пример)

Есть еще одно понятие - энкодеры. У нас есть, допустим, оригинальная картинка, к которой даже нет данных. Мы можем сделать такую нейронную сеть, которая сначала получает вектор (все, как если мы хотим предсказать какое-то свойство), потом из этого вектора обратно получает картинку (рис. 16).

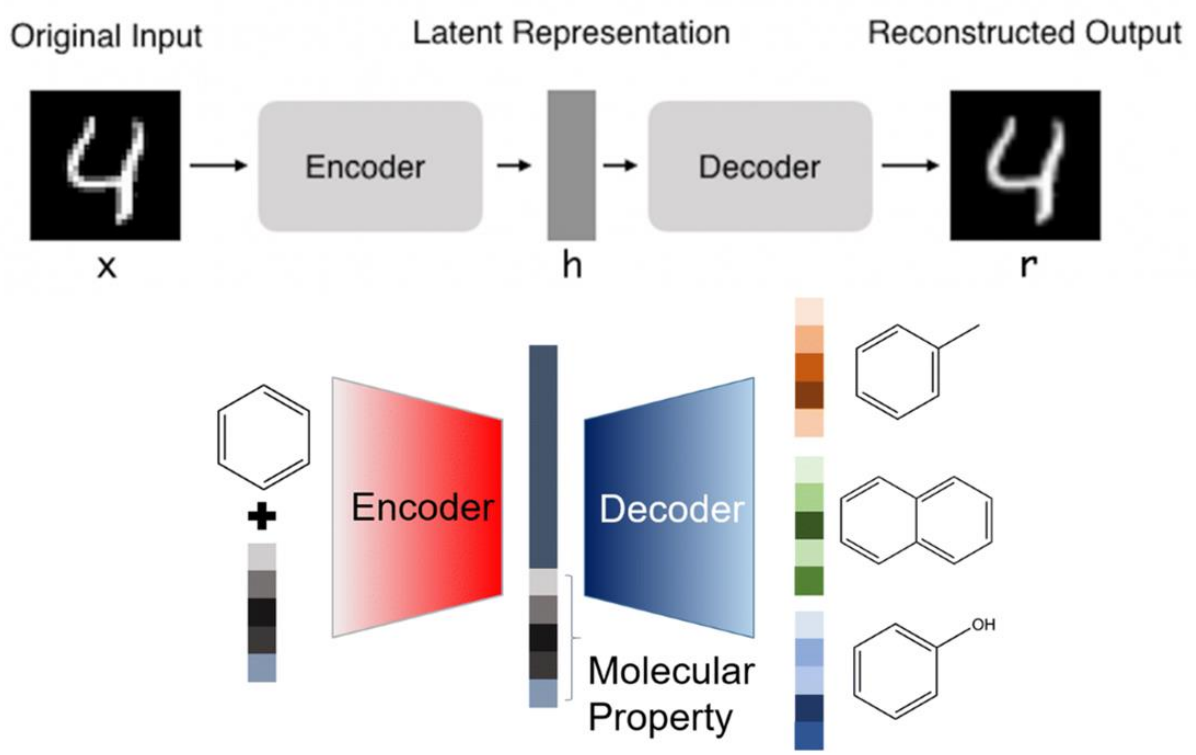


Рисунок 16. Сверточные сети (автоэнкодеры)

То есть наше предсказание - это та же картинка. Мы можем обучить нейронную сеть, чтобы она из картинки получала ту же самую картинку. На первый взгляд ничего полезного мы не получаем, но по факту наш промежуточный слой с вектором учится описывать картинку. Таким образом, мы можем получать векторы, описывающие картинку.

Это имеет прямое применение при генерации новых лиц, то есть сначала мы обучаем нейронную сеть предсказывать из лиц те же лица, потом ставим новый вектор, которого еще не было в обучающей выборке, и получаем новое лицо. Т.е. раз модель понимает соответствие между картинкой и вектором, она может сгенерировать что-то новое, чего не было в начальном dataset.

Также это может быть применено к поиску молекул с заданными свойствами. Например, у нас есть молекула и есть какое-то свойство к ней, мы можем подавать на вход молекулы вместе со свойством, и получить молекулы, которые обладают нужным свойством. Этот метод не совсем хорошо предсказывает молекулы, но в целом, при увеличении количества данных, такие сети можно будет массово использовать, чтобы предсказывать молекулы с заданными свойствами.

Рассмотрим предсказание свойств кристаллических структур. У нас есть нейронная сеть и какой-то набор математических операций, нам нужно охарактеризовать кристаллическую структуру, чтобы для нее что-то предсказывать. Это достаточно сложная задача. Какие есть варианты ее решения? Самое простое - описать структуру ее свойствами (например, такими характеристиками как масса элементов, модуль Юнга и др.) и, собственно, делать предсказания на основании этого вектора.

Здесь минус достаточно очевидный. Во-первых, часто мы просто эти свойства не знаем, чтобы дать хорошее описание структуры. Во-вторых, разные структуры будут иметь похожие свойства, и в итоге для них предсказания будут одинаковыми. Но, возможно, свойства, которые нас

интересуют, у них ожидаются разные, и мы не сможем дать корректное предсказание, используя такой метод.

Существуют два более приемлемых способа:

1. Использование дескриптора.

Мы генерируем набор чисел, который описывает кристаллическую структуру. Это сделать не так просто, потому что мы набор чисел должен как можно более полно ее описывать, чтобы не было такого, что для разных структур получается одинаковый набор чисел.

Может показаться, что достаточно взять координаты атомов структуры, и это будет набор чисел, который ее полностью описывает. Здесь возникает другая проблема. Мы, например, повернем структуру. Она останется той же самой, ничего реально не поменяется, а для модели это уже другой вход. Моделе сложнее на этих данных обучиться, потому что раз структура одинаковая, то хотелось бы, чтобы и вход был один и тот же. Соответственно, есть разные операции, которые не должны менять эти числа. Такой дескриптор не так уж и просто составить.

Координаты атомов не годятся, чтобы использовать их в качестве дескриптора. Первая идея – использование матрицы Вейля, где записаны скалярные произведения векторов, которые описывают разные соседние атомы. Можно определить, какие из них мы будем считать одинаковыми. Необходимы учесть перестановки атомов местами. Если мы переставляем

какие-то атомы, мы говорим, что матрица, которая получилась, на самом деле та же самая (рис. 17).

Рисунок 17. Дескриптор

Зато этот метод хорошо справляется с вращениями и сдвигами матрицы, потому что в нем использовано скалярное произведение.

Из-за проблемы с необходимостью учитывать перестановки данная

- Координаты атомов не подходят для описания структуры в качестве входа для алгоритмов машинного обучения

- Матрицы Вейля

$$\Sigma = \begin{bmatrix} \mathbf{r}_1 \cdot \mathbf{r}_1 & \mathbf{r}_1 \cdot \mathbf{r}_2 & \cdots & \mathbf{r}_1 \cdot \mathbf{r}_N \\ \mathbf{r}_2 \cdot \mathbf{r}_1 & \mathbf{r}_2 \cdot \mathbf{r}_2 & \cdots & \mathbf{r}_2 \cdot \mathbf{r}_N \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{r}_N \cdot \mathbf{r}_1 & \mathbf{r}_N \cdot \mathbf{r}_2 & \cdots & \mathbf{r}_N \cdot \mathbf{r}_N \end{bmatrix}$$

Сложно считать для большого числа атомов, плохое поведение функции

модель показывает достаточно плохие результаты, то есть это не гладкая функция, у нее перескоки в местах, где мы считаем матрицы эквивалентными. Также, если у нас большое число атомов, приходится проводить много расчетов, особенно трудно учесть все перестановки. Поэтому реально на практике ее не получается использовать.

Сейчас практически применяются 2 дескриптора. Они используют сложные построения, однако гарантируют, что если мы возьмем большое количество данных, структура будет описана (рис. 18).

- На практике используются сложные построения, гарантирующие полноту описания структур (при полном базисном наборе)

Power spectrum

$$p_0 = \frac{9}{4\pi}$$
$$p_1 = \frac{3}{4\pi}\left(\sum_{ii'}\cos\theta_{ii'} + 3\right)$$
$$p_2 = \frac{5}{4\pi}\left(\frac{3}{2}\sum_{ii'}\cos^2\theta_{ii'} + 6\right)$$
$$p_3 = \frac{7}{4\pi}\left(\frac{5}{2}\sum_{ii'}\cos^3\theta_{ii'} - \frac{3}{2}\sum_{ii'}\cos\theta_{ii'} + 3\right)$$
$$p_4 = \frac{9}{16\pi}\left(\frac{35}{2}\sum_{ii'}\cos^4\theta_{ii'} - 15\sum_{ii'}\cos^2\theta_{ii'} + 13\right)$$

Moment Tensor Potentials

$$M_{\mu,\nu}(u) := \sum_{i=1}^{n} |u_i|^{2\mu} u_i^{\otimes\nu}$$
$$B_\alpha(u) := \prod_{i=1}^{k}{}^{\alpha} M_{\alpha_{ii},\alpha'_i}(u)$$

Рисунок 18. Дескриптор

Если мы хотим идеально описать структуру, мы должны взять бесконечное количество чисел. Но даже если мы возьмем тысячу чисел, то уже получится достаточно хорошая модель для многих применений. Есть целая область науки, которая занимается составлением и оценкой (однозначно ли они описывают структуру или нет) дескрипторов.

Существует альтернатива предыдущему подходу, в которой со структурой работают просто как с графом, аналогично тому, как работали с изображениями (мы не переводили их в вектор, а применяли сверточные преобразования). Оказывается, что сверточные преобразования можно применить к графам, а кристаллическая структура это по сути дела и есть граф (рис. 19).

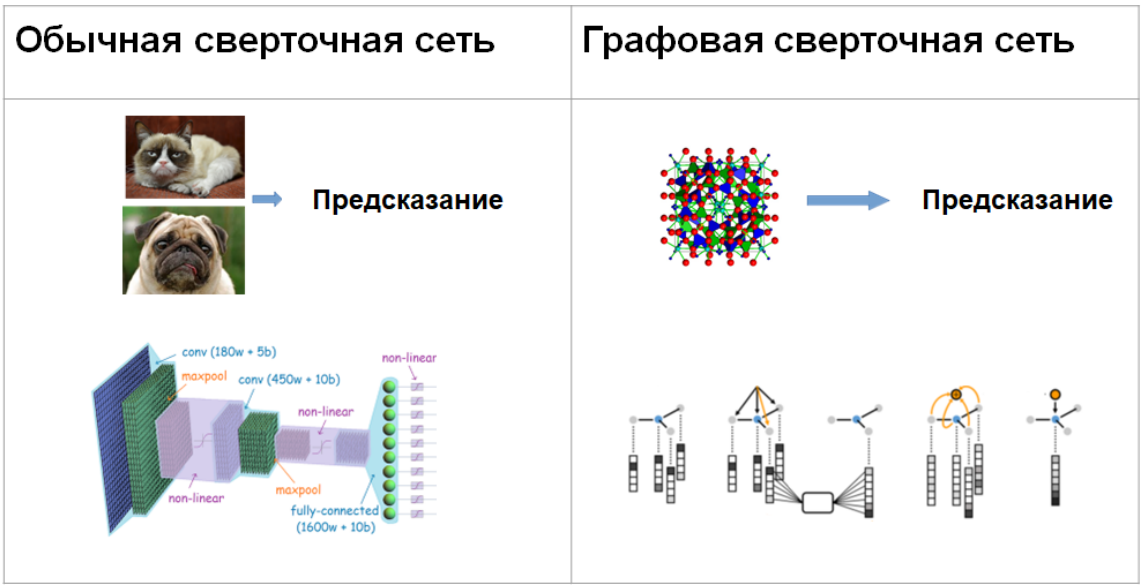
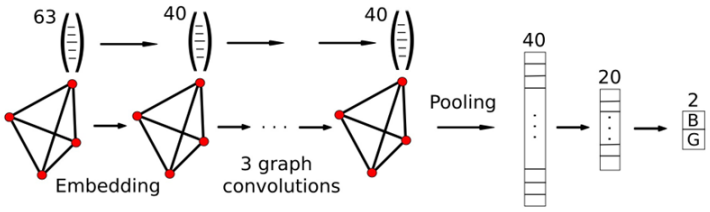


Рисунок 19. Сверточные сети на графах

У нас есть какая-то исходная кристаллическая структура, идут математические преобразования, есть pooling, где в какой-то момент получается вектор, и из этого вектора мы получаем итоговое предсказание (например, какие-то свойства этой кристаллической структуры) (рис. 20).

- Сверточные сети на графах позволяют обучать модели напрямую на кристаллических структурах



- Предсказанием является определенное свойство кристалла или атома (или много свойств)

Рисунок 20. Сверточные сети на графах

Используя сверточные сети, мы, во-первых, описываем атомы, которые в ней находятся, а во-вторых, связи, которые существуют между атомами (рис. 21).

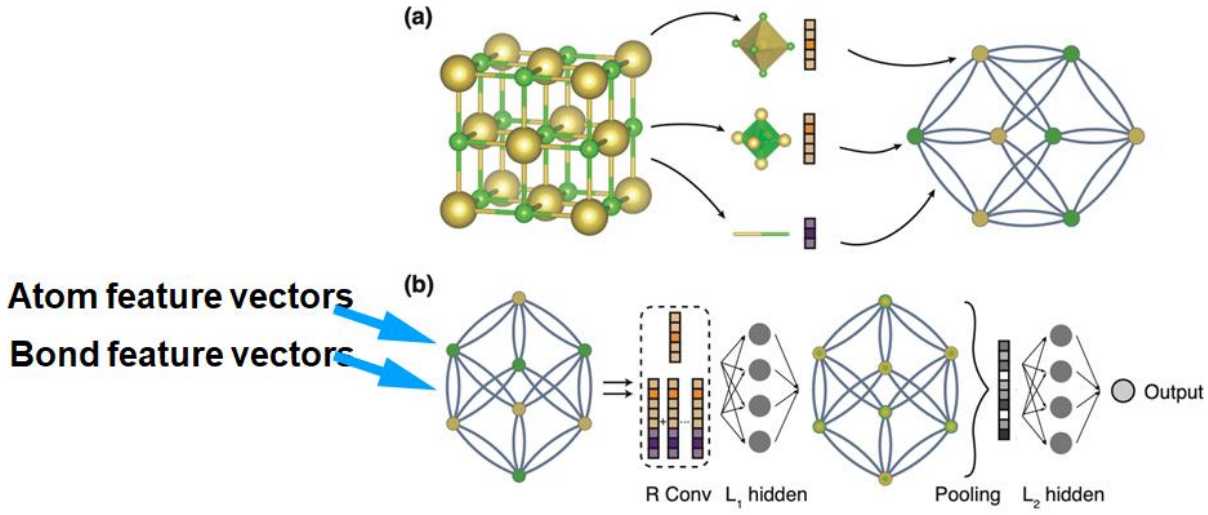


Рисунок 21. Сверточные сети на графах

Атомы могут описываться простым способом. Мы записываем вектор, где единичка стоит на том месте, где номер таблицы Менделеева, то есть

водород-10000, гелий – 01000, литий -001000, то есть делаем уникальные вектора, которые описывают атомы (рис. 22).

- Атомный признак (пример):

$$v_i = [0,0,0,1,\dots]$$

- Связевый признак (пример)

$$u_{ij} = \left[exp(-\frac{(d-0\cdot s)^2}{s^2}), exp(-\frac{(d-1\cdot s)^2}{s^2}), \dots, exp(-\frac{(d-R_{cut})^2}{s^2})\right]$$

Рисунок 22. ***Атомные и связевые признаки***

Связь мы можем описать функцией от расстояния между атомами, то есть вектором, который зависит от расстояния.

В сверточной графовой сети выделяют несколько видов слоев:

1. Линейный – берем вектор, который описывает атом, умножаем его на матрицу и прибавляем вектор. В итоге получаем новый вектор, который описывает атом.

2. Сверточный - берем атом, к нему прибавляем какую-то функцию от его соседей. Соответственно, у него меняется вектор, который его описывает. В конечном итоге у нас есть эти вектора, измененные преобразованиями, мы их усредняем и получаем дескриптор, который описывает кристаллическую структуру, которая была в начале, и из этого дескриптора получаем предсказания.

Может показаться, что чем больше слоев мы сделаем, тем лучше у нас могут получаться результаты предсказаний. В реальности это есть много факторов, нарушающих это. Первый и самый очевидный – мы используем больше параметров, получаем более сложную функцию, которая сильнее

подстроится под те данные, которые у нас есть. Они могут содержать нетипичные значения, шумы. Соответственно, добавление параметров лишь до какого-то момента дает улучшение результатов.

Есть проблема, что большие сети сложнее обучать. Одна из основных причин – убывающий градиент - когда он становится слишком маленький модель перестает обучаться.

Есть еще третий фактор, который специфичен именно для графовых сетей. Когда у нас идет переход от первого слоя к дальнейшим слоям, часто признаки перестают сильно друг с другом различаться, и результат ухудшается, то есть такая ошибка накапливается (рис . 23).

- Линейный слой:

$$v_i^{t+1} = W v_i^t + b$$

- Сверточный слой:

$$v_i^{t+1} = v_i^t + \sum_j \sigma(z_{ij} W_1 + b_1) \odot \sigma(z_{ij} W_2 + b_2)$$

$$z_{ij} = v_i \oplus v_j \oplus u_{ij}$$

- Pooling:

$$v = \frac{1}{N} \sum_i v_i$$

Рисунок 23. Сверточные графовые сети: слои

Плюсы графовых сетей:

1. Количество атомов не имеет значения, т.к. атомы описываются разными векторами. Мы можем предсказать свойства даже для тех атомов, которых вообще не было в обучающей выборке. Иногда сеть даже способна

делать для них какие-то адекватные предсказания, хотя она вообще никакой информации о них не имела.

2. Легко применить подход transfer learning - это когда мы обучаем модель на одних данных, а потом используем для улучшения предсказания на других данных. Допустим, у нас есть нейронная сеть, которая представляет собой ряд идущих друг за другом математических операций. Мы на каком-то моменте его обрезаем, и это место, где мы вырезали, можно использовать как вход другой нейронной сети, которая предсказывает другие свойства. То есть первая сеть накопила какую-то информацию, которую мы передаем дальше.

Минусы:

1. Предсказания могут быть нестабильны к небольшому изменению параметров

2. Бывают сложны в обучении.

Примеры применений моделей:

1. Поиск сверхтвердых материалов. Есть два свойства, которые определяют обычно для практики применимости материала - это твердость и трещиностойкость. Используя машинное обучение, мы можем предсказывать твердость и трещиностойкость, и выбирать те материалы, которые обладают высокими значениями обоих свойств.

Была проведена работа, в которой была использована сложная схема (напрямую обучать оценке твердости и трещиностойкости сложно, потому что данных недостаточно для обучения нейронной сети). Была использована модель, которая связывает твердость и трещиностойкость с упругими модулями (больше эмпирическая модель). Упругие модули - это модуль Юнга,

коэффициент Пуассона, модуль сжатия, модуль сдвига и т.д. И данная модель позволяет предсказывать твердость и трещиностойкость материалов (рис. 24).

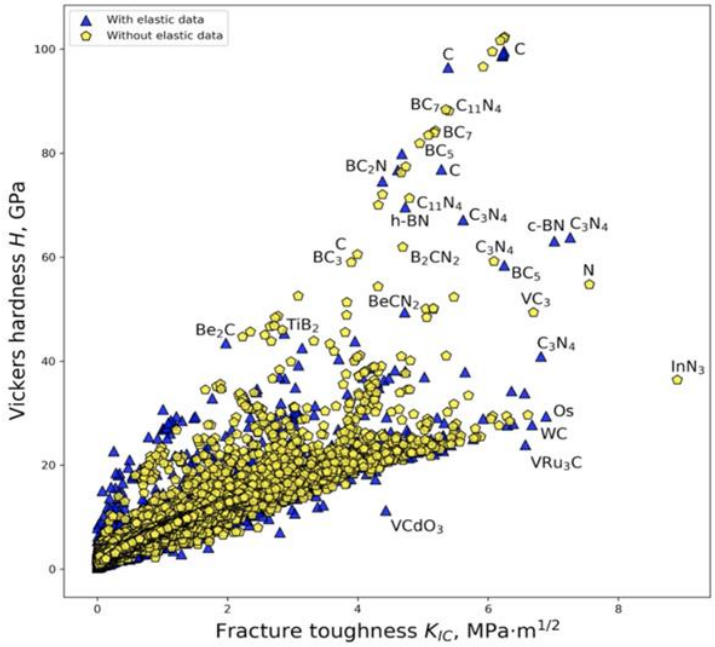


Рисунок 24. **График ~100 000 кристаллических структур с разными значениями твердости и трещиностойкости**

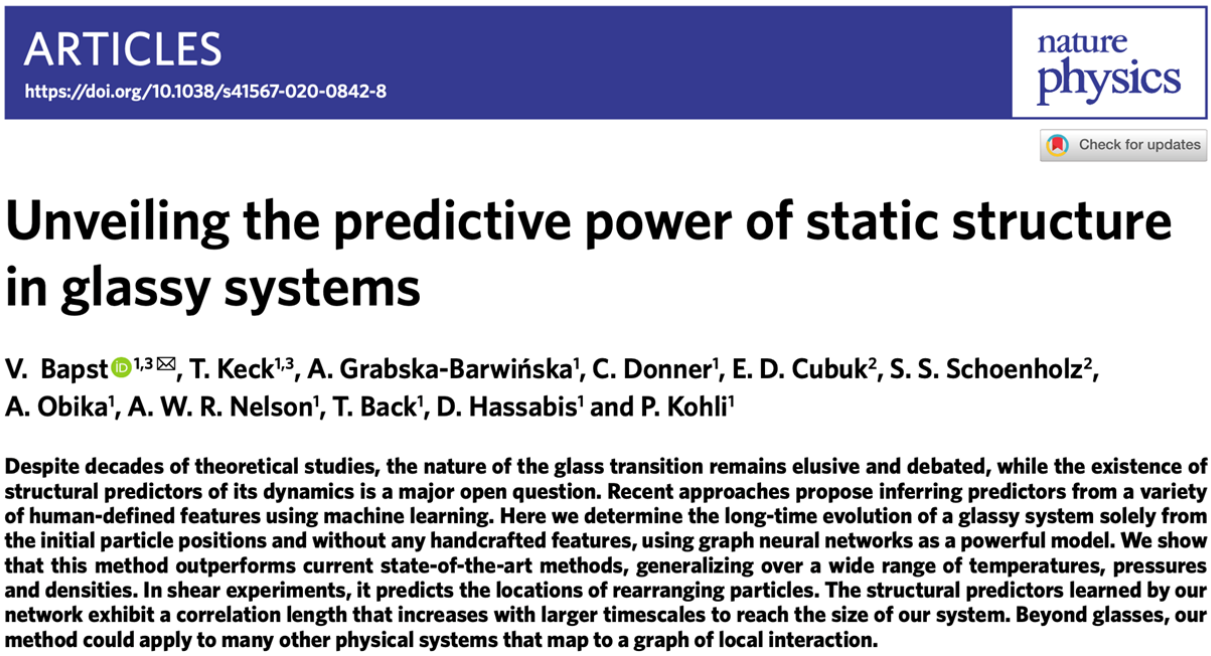
Диаграмма Эшби

Нас интересует материал, который справа и сверху. Синие – материалы, для которых уже были данные, желтым обозначены материалы, для которых не было данных, все было получено из модели. Получается, что таким методом можно гораздо большее количество материалов захватывать, которые обладают этими свойствами. Можно увидеть, что справа сверху углерод, то есть это алмаз, в целом подтверждает гипотезу о том, что под алмаз небольшие модификации эмпирические аллотропные, является, собственно, по

соотношению твердость и трещиностойкость, по разным соотношениям можно смотреть, какой именно нужен, но в целом они выделяются среди всех остальных материалов и в плане соотношения достаточно сложно какому-то материалу опередить углерод.

В целом понятно, что другие есть параметры, которые определяют практическое применение и если даже соотношение твердости и трещиностойкости не такое высокое, как у алмаза, есть другие характеристики, например, цена закупки. Этот метод позволяет нам просто большое количество материалов, которые уже примерно подходят, перебрать по этому параметру и выбрать из них наиболее подходящие.

Соответственно еще несколько примеров. В статье была выпущена компания DeepMind, которая является частью Google (рис. 25).



ARTICLES
https://doi.org/10.1038/s41567-020-0842-8

nature physics

Check for updates

Unveiling the predictive power of static structure in glassy systems

V. Bapst[1,3]✉, T. Keck[1,3], A. Grabska-Barwińska[1], C. Donner[1], E. D. Cubuk[2], S. S. Schoenholz[2], A. Obika[1], A. W. R. Nelson[1], T. Back[1], D. Hassabis[1] and P. Kohli[1]

Despite decades of theoretical studies, the nature of the glass transition remains elusive and debated, while the existence of structural predictors of its dynamics is a major open question. Recent approaches propose inferring predictors from a variety of human-defined features using machine learning. Here we determine the long-time evolution of a glassy system solely from the initial particle positions and without any handcrafted features, using graph neural networks as a powerful model. We show that this method outperforms current state-of-the-art methods, generalizing over a wide range of temperatures, pressures and densities. In shear experiments, it predicts the locations of rearranging particles. The structural predictors learned by our network exhibit a correlation length that increases with larger timescales to reach the size of our system. Beyond glasses, our method could apply to many other physical systems that map to a graph of local interaction.

Рисунок 25. Статья от Google DeepMind (AlphaGo, AlphaZero, AlphaFold)

Вот они в том числе тоже выпустили много моделей машинного обучения интересных, например, AlphaGo, AlphaZero, AlphaFold. Они применяли методы машинного обучения, чтобы обыграть самых сильных людей в игру Го (считалось, что эта игра неподвластна алгоритмам). Также у

них были научные разработки, например, они пытались создать модели для исследования белков.

В статье с использованием графовых сетей предсказывали мобильность различных атомов при образовании стекол. Идея достаточно простая. Есть жидкий материал, в котором атомы достаточно беспорядочно расположены. В зависимости от того, с какой скоростью мы охлаждаем этот материал, у нас появляется либо стекло с беспорядочным расположением атомов, либо кристаллы с каким-то порядком, если мы медленно охлаждаем (рис. 26).

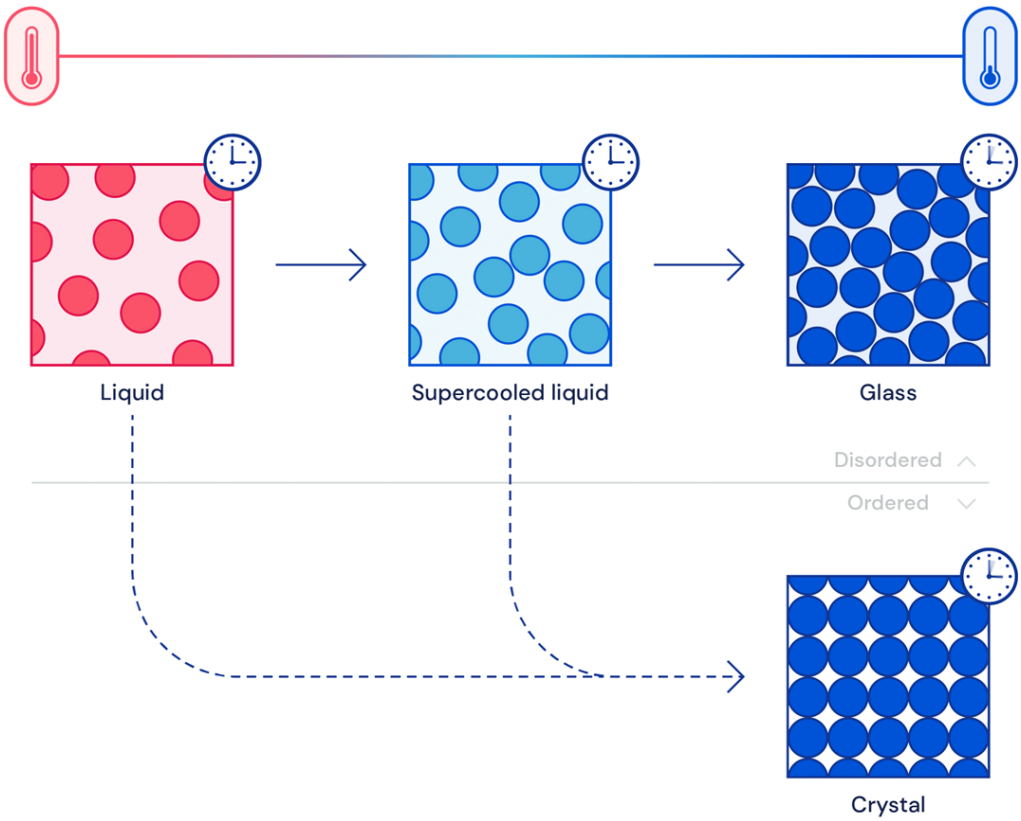


Рисунок 26. Материал в зависимости от скорости охлаждения

Можем попытаться предсказать что получится. Посчитать с помощью квантово-механических методов достаточно сложно, потому что расчеты очень тяжелые, а чтобы простимулировать их, нужна зависимость мобильности от времени, что сложно определить теоретическими методами.

Вот они взяли и обучили на них графовую сеть, которая просто для каждого атома предсказывает его мобильность. Кроме того, они также посмотрели интересные вещи с физической точки зрения. Можно представить, что есть какой-то атом (или какая-то группа атомов), и мы можем отодвигать от него остальные атомы, и смотреть, как меняется предсказание. Если мы видим, что предсказание не меняется, это означает что сеть считает, что отодвигаемые атомы не участвуют во взаимодействии с этим атомом, если предсказание меняется, значит участвуют. Таким образом, двигая в графовой нейронной сети атомы, можно смотреть дальность порядка взаимодействия между ними (рис. 27).

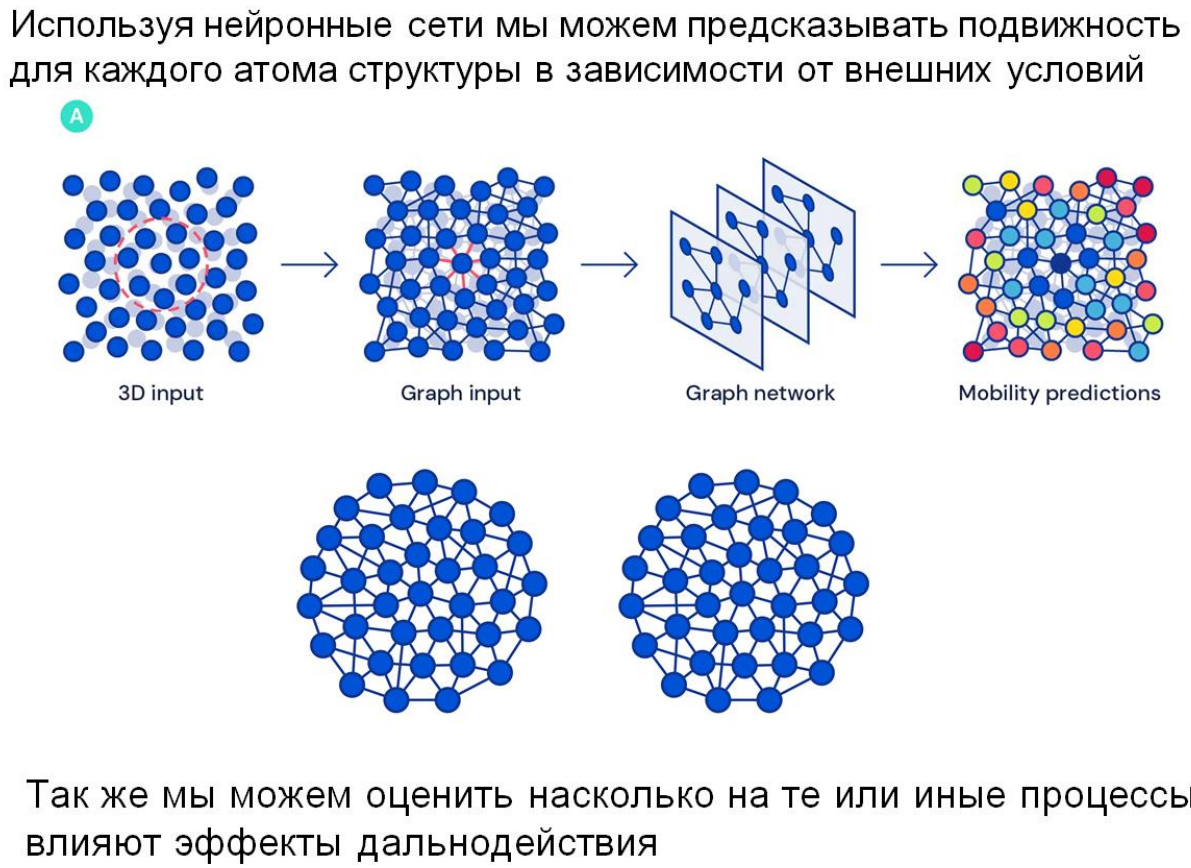
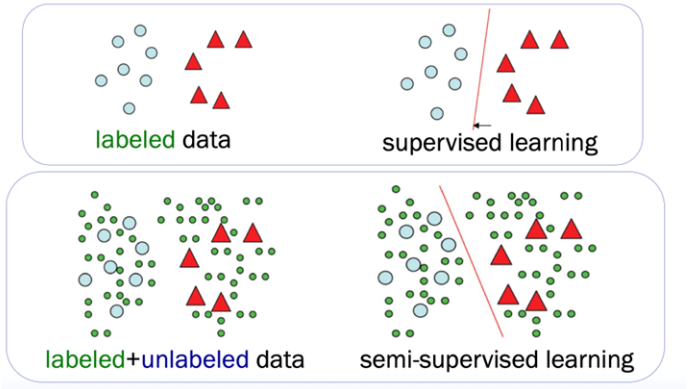


Рисунок 27. Модель предсказания подвижности атомов

С первого взгляда кажется, что если у нас есть модель машинного обучения, то ей вообще не нужны данные, для которых у нас нет ответов. Однако достаточно давно установили, что использование данных, в том числе для которых нет ответов, помогает обучать модели. Есть иллюстрация, которая поможет это понять (рис. 28).

На верхней картинке показано два типа данных - кружочки и треугольники. Когда мы обучаем модель, линию или плоскость проводим между этими данными. Теперь то, что слева, мы предсказываем как один класс, то, что справа, как другой.



Мы можем использовать semi-supervised learning когда неизвестны свойства для каких-то структур или отдельных атомов в структуре

Transfer learning: предобучение

Рисунок 28. Обучение модели

Снизу слева картинка показывает, что мы добавили данных. И где теперь провести линию? Интуитивно видно, что ее хочется провести уже не между кружочками и треугольниками условно посередине, как это было раньше. Зеленые точки образуют облака. Понятно, что слева эти облака, скорее всего,

принадлежат одному классу, те, что справа, другому. Это позволяет нам провести линию в другом месте. В целом, так и происходит.

Если мы добавляем какие-то данные, то можем реально улучшить качество модели. Это называется semi-supervised learning.

Перейдем к процессу создания так называемых потенциалов. У нас есть какое-то положение атомов, и мы предсказываем для них энергию. Зачем это нужно? Например, если мы умеем предсказывать энергию, мы можем ее минимизировать, т.е. найти то положение атомов, которое наиболее стабильно. Для этих подходов существует множество методов, в том числе метод на основе дескрипторов. Графовые сети тоже иногда применяются, но обычно они работают немного хуже, поскольку методы, основанные на дескрипторах сразу предполагают, что энергия - достаточно гладкая функция. Графовые сети также менее стабильны. В начале получается большая дисперсия, потому что данных не так много. Поэтому часто подходы, которые используют дескрипторы, выигрывают. Но такой подход использует некоторое свойства энергии, которые неверны для других свойств, которые мы хотим предсказывать, например, для тех же упругих модулей. Такие подходы применяются, их не всегда удается распространить на другие свойства, а графовые сети могут это сделать. Плюс есть подход active learning, когда мы используем какой-то изначальный набор данных. Дальше, когда мы видим новые данные, мы смотрим, насколько они похожи на предыдущие. Если похожи, мы предсказываем для них новые значения, если нет, мы добавляем их в обучающую выборку, переучиваем модель, и получаем предсказание - получается постепенное обучение на тех данных, в которых модель не уверена. Преимущество этих потенциалов в том, что легко пересчитать уверенность модели, в то время как для более сложных моделей это часто сложно сделать с вычислительной точки зрения.

Еще один пример - поиск соотношения свойств, в данном случае температурного расширение и упругих свойств (рис. 29).

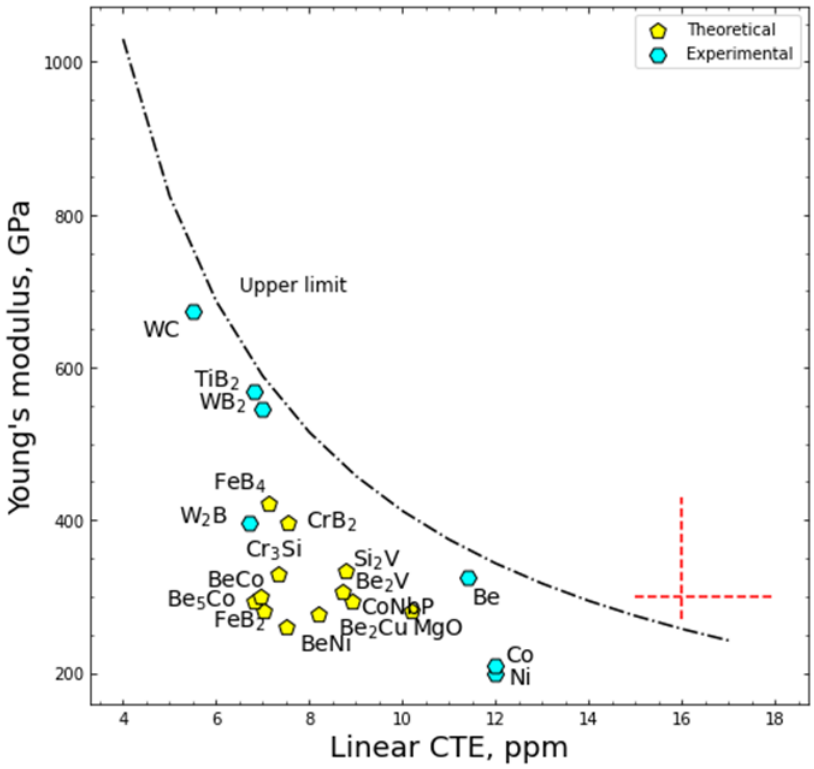


Рисунок 29. Упругие свойства и температурное расширение

Можно увидеть, что между ними есть некоторое соотношение. Сложно найти материал одновременно и с высоким модулем Юнга, и с высоким коэффициентом температурного расширения. Задача заключалась в том, чтобы найти материалы с наиболее высоким произведением этих свойств. Применяя модель машинного обучения мы находим такие материалы, определяем для них свойства, и более точными методами пересчитываем те из них, которые проявили лучшие свойства.