Лекция 1. Введение

Вспоминаем ML

Логистическая регрессия

Пусть имеется выборка $\{x_i, y_i\}_{i=1}^{\ell}$, где $x_i \in \mathbb{R}^d$ — признаковое описание i-го объекта, $y_i \in \{-1, +1\}$ — метка класса. Мы имеем дело с задачей бинарной классификации. В классическом машинном обучении существует алгоритм линейной классификации: $y(x) = \text{sign}(w^T x)$, где $w \in \mathbb{R}^d$ — обучаемые веса. Обучение происходит за счет минимизации эмпирического риска Q, который представляет собой сумму функций потерь по всем объектам в обучающей выборке: $Q = \sum_{i=1}^{\ell} \mathcal{L}(x_i, y_i, w)$. Линейный классификатор называют логистической регрессией, если в качестве $\mathcal{L}$ выбрано следующее:

$$\mathcal{L}(x, y, w) = -\log \sigma(yw^T x), \quad \sigma(z) = \frac{1}{1 + \exp(-z)}.$$

Функционал Q в таком случае называют логлоссом. Задачу оптимизации логлосса решают градиентными методами (например, SGD, LBFGS). Они требуют знание аналитических формул для подсчета градиента. Градиент функции потерь $\mathcal{L}$ можно вычислить с помощью стандартных правил матрично-векторного дифференцирования и правила цепочки:

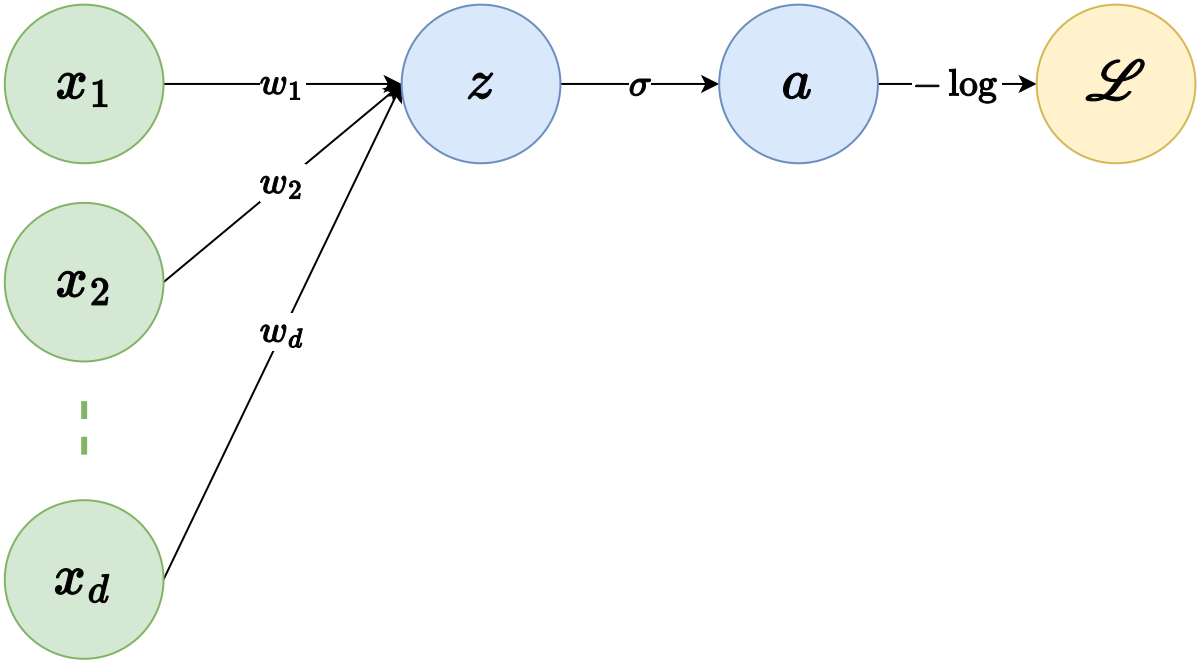
$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial yw^T x}{\partial w} \frac{\partial \sigma(yw^T x)}{\partial yw^T x} \frac{\partial(-\log \sigma)}{\partial \sigma} \\
&= yx \cdot \sigma(yw^T x)(1 - \sigma(yw^T x)) \cdot \frac{-1}{\sigma(yw^T x)} \\
&= -yx(1 - \sigma(yw^T x)).
\end{aligned}$$

Граф вычислений логистической регрессии

Давайте взглянем на алгоритм логистической регрессии с точки зрения входов, выходов и промежуточных результатов. Введем обозначения:

- $z := w^T x$ — логит,
- $a := \sigma(z)$ — вероятность,
- $\mathcal{L} := -\log a$ — значение функции потерь.

Справедлива следующая схема:



Вершины в графе обозначают численные значения, ребра — операции. Зеленым цветом обозначены входы алгоритма (признаковое описание), оранжевым цветом обозначен выход, синим цветом обозначены промежуточные вычисления.

Мультиномиальная регрессия

Пусть имеется выборка $\{x_i, y_i\}_{i=1}^{\ell}$, где $x_i \in \mathbb{R}^d$ — признаковое описание i-го объекта, $y_i \in \{1, \dots, K\}$ — метка класса. Мы имеем дело с задачей многоклассовой классификации. В классическом машинном обучении существует алгоритм линейной классификации:

$$y(x) = \arg \max_{1 \leqslant k \leqslant K} (w_k^T x),$$

где $W \in \mathbb{R}^{K \times d}$ — обучаемые веса. Линейный классификатор называют мультиномиальной регрессией, если в качестве $\mathcal{L}$ выбрано следующее:

$$\mathcal{L}(x, y, w) = -\log \frac{\exp(w_y^T x)}{\sum_{k=1}^{K} \exp(w_k^T x)}.$$

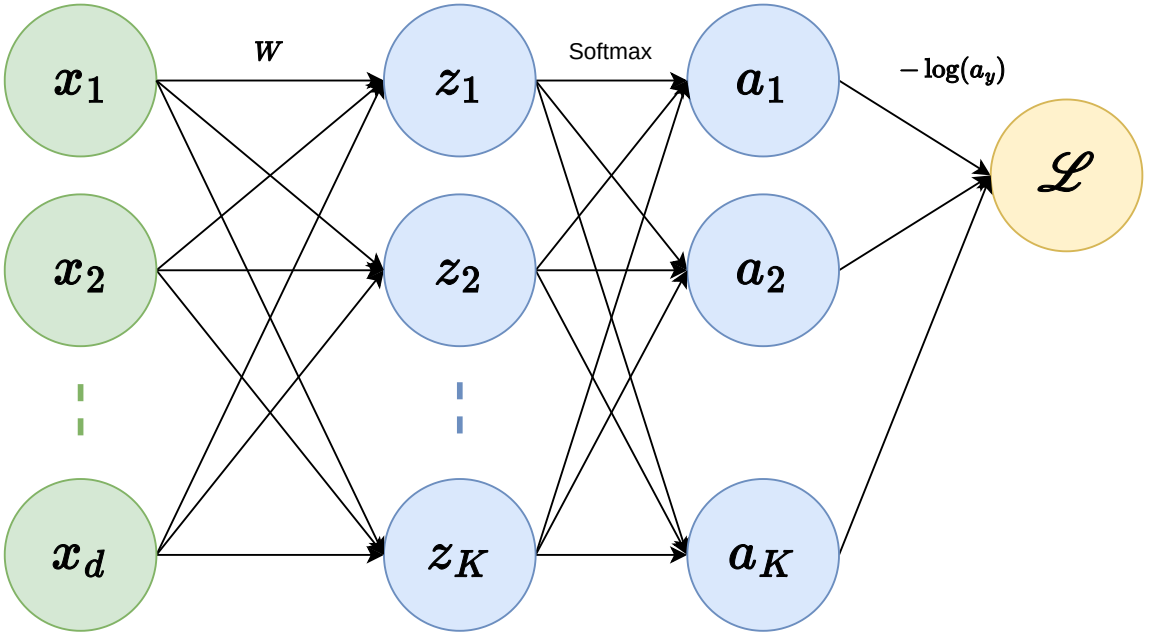
Функционал Q в таком случае называют кросс-энтропией. Заметим, что в некотором смысле алгоритм мультиномиальной регрессии состоит из K логистических регрессий.

Граф вычислений мультиномиальной регрессии

Введем обозначения:

- $z_k := w_k^T x$ — логит для класса k,
- $a_k := \exp(z_k)$ — активация для класса k.

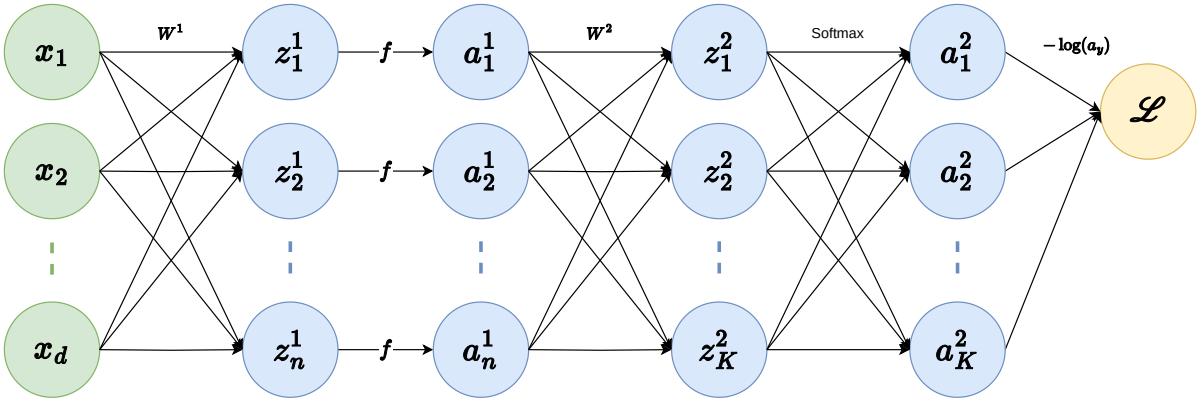
Справедлива следующая схема:



Многослойная полносвязная сеть

Заметим, что активации a_i выстраиваются в ряд, подобный тому, в который выстраиваются входы x_i. Идея многослойной полносвязной сети (прибизительно) состоит в том, чтобы подать активации a_i на вход другой мультиномиальной регрессии, а для подсчета функции потерь использовать финальные активации. Эту операцию можно повторить L раз и получится L “слоев”.

Схема для $L = 2$:



Немного терминологии. На данной схеме:

- $W^1 \in \mathbb{R}^{n \times d}$ — обучаемые веса первого слоя,
- $W^2 \in \mathbb{R}^{K \times n}$ — обучаемые веса второго слоя,
- $f : \mathbb{R} \to \mathbb{R}$ — некоторое нелинейное преобразование, т.н. функция активации (например, сигмоида),
- $z^1, a^1 \in \mathbb{R}^n$ — активации первого слоя (а не логиты и вероятности),
- $z^2, a^2 \in \mathbb{R}^K$ — активации второго слоя (но по совпадению логиты и вероятности).

Каждый z еще называют суммматором, или нейроном (по аналогии из биологии, согласно которой нейроны суммируют сигналы от других нейронов, с которыми они связаны). Активации z^1 и z^2 называют слоями нейронов.

Есть и жаргонные термины. Линейные преобразования $z^1 = W^1x$ и $z^2 = W^2a_1$ называют линейными слоями (хотя на самом деле это операции, а не ряды чисел, и никакие слои они не могут образовывать). Так же и операцию f часто называют слоем активации (хотя никаких слоев нет).

Многослойность

Какой смысл в многослойной архитектуре? Если отвечать коротко, то каждый слой преобразует признаки и делает их более абстрактными и высокоуровневыми.

Рассмотрим на примере задачи классификации изображения. Будем подавать каждое (монохромное) изображение размера $H \times W$ в виде вектора $x \in \mathbb{R}^{HW}$, где каждое число означает яркость соответствующего пикселя. Каждый нейрон первого слоя действует как логистическая регрессия, которая “детектирует” наличие ярких точек в некоторых частях изображения. Например, вполне допустимо, что один нейрон будет “детектировать” некоторую прямую линию, как и остальные нейроны в слое. Вся эта информация затем подается во второй слой (например, в виде вероятностей из сигмоиды). И тогда нейроны (=бинарные классификаторы) второго слоя вполне вероятно смогут обучиться видеть целые геометрические фигуры. Все потому что им поступает более высокоуровневая информация нежели сырые пиксели. На вход третьему слою уже пойдет информация о том, какие геометрические фигуры и в каких местах есть на изображении, а на основании этого вполне реально произвести классификацию и понять, что за существо изображено на картинке.

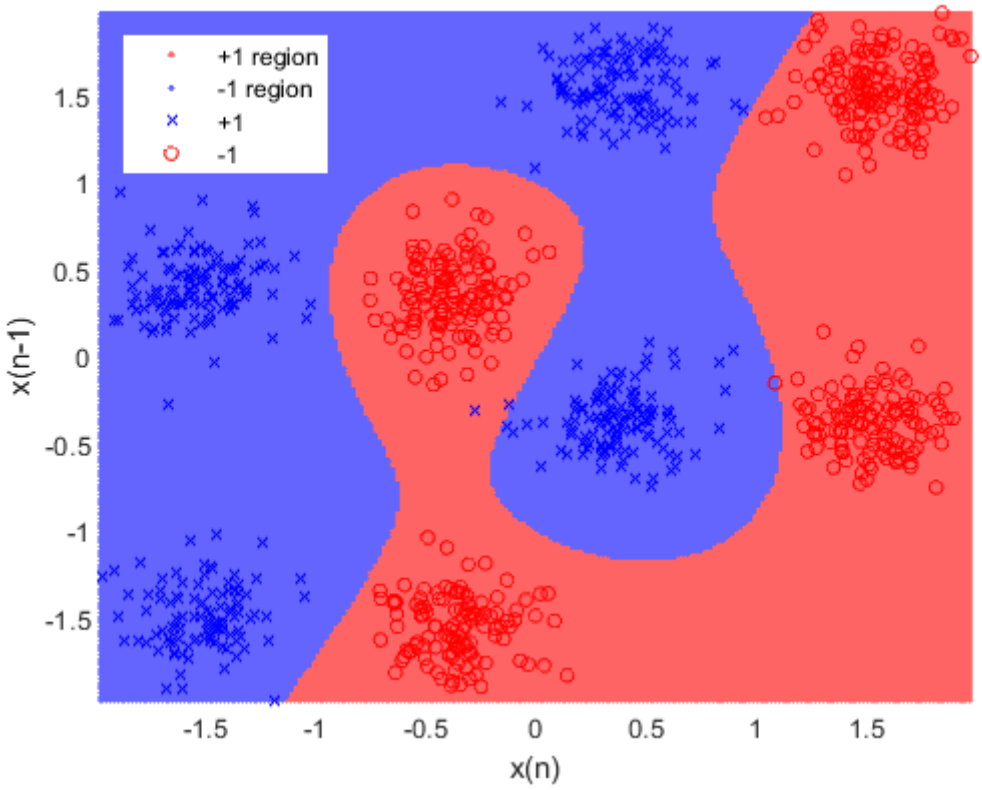
Таким образом, многослойная архитектура позволяет выучивать преобразования признаков, делать их более высокоуровневыми и абстрактными. С этим связано название глубокое обучение — так называют раздел машинного обучения, изучающего нейронные сети, — поскольку все нейросети имеют тенденцию быть многослойными (=глубокими).

Стоит заметить, что важной частью многослойной полносвязной сети являются нелинейные преобразования. Они стоят между линейными преобразованиями. Если бы функций активаций не было и все линейные преобразования шли подряд, то вся нейросеть была бы эквивалентна одной линейной модели. Таким образом, функции активации повышают репрезентативную способность нейросетей, расширяя их до класса нелинейных моделей.

Гипотеза компактности

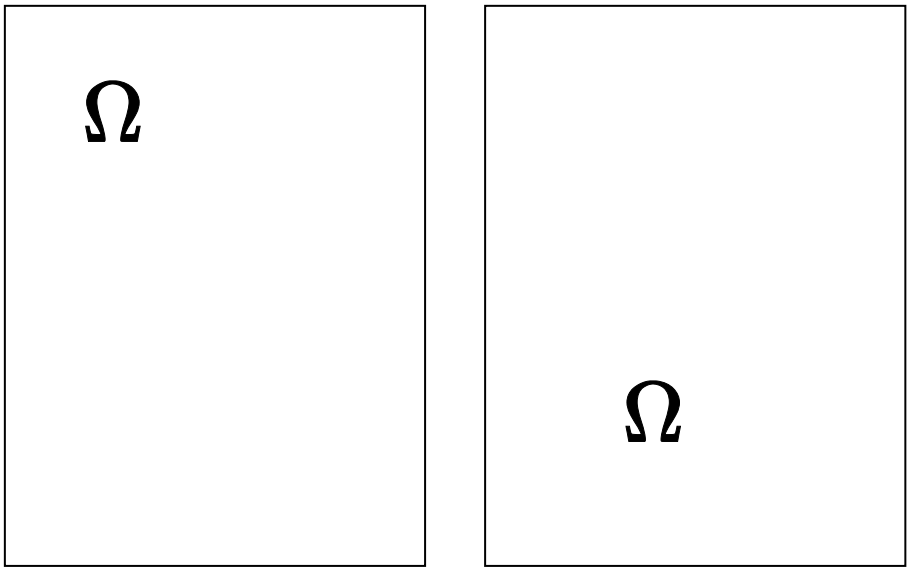
Многослойность архитектуры нейросетей связана с вопросом о том, когда применять нейронные сети. Ответ: в тех доменах, где не выполнена так называемая гипотеза компактности. Гипотезой компактности называют свойство данных, в которых семантически похожим объектам соответствуют близкие по метрике векторные представления (признаковые описания).

Вот пример данных, для которых выполнена гипотеза компактности.



Рассмотрим примеры данных, в которых гипотеза компактности не выполнена.

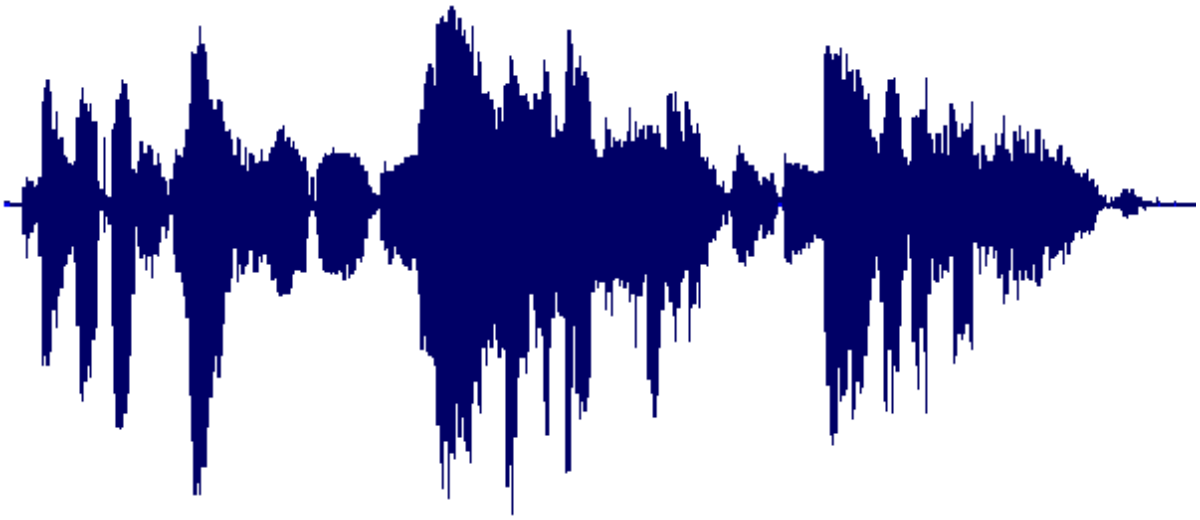
Во-первых, это изображения. Ниже представлены два изображения буквы омега. С точки зрения смысла, эти изображения являются очень похожими объектами. Однако если мы вытянем эти пиксели в вектора, то косинус между ними будет нулевой, а евклидово расстояние ожидаемо большим.



Во-вторых, это текст на естественном языке. Ниже представлены два предложения с одинаковым смыслом, но с абсолютно разными словами (за исключением стоп-слов). Векторные представления в виде мешка слов имеют нулевой косинус и большое евклидово расстояние

> 1. The cat is sleeping on the couch.
> 2. A feline rests upon the sofa.

В-третьих, это аудио-сигнал. Если одно и то же слово произнесут два человека с разной высотой голоса, то волноформа сильно поменяется, хотя смысл сообщения останется тем же.



Примеров еще очень много. Со всеми ними нейросети справляются за счет того, что каждый слой преобразует сырые признаки и получает все более абстрактные. При этом на последнем слое получается богатое признаковое описание, для которого гипотеза компактности выполнена, поскольку финальный слой — это просто мультиномиальная регрессия.

Проход вперед

Формализуем алгоритм предсказания с помощью нейронной сети в виде так называемого прохода вперед (forward pass).

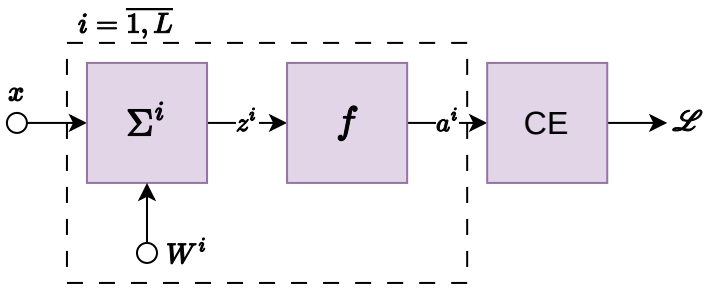
Алгоритм **MLP FORWARD PASS**

- ВХОД $x \in \mathbb{R}^d$
- ВЫХОД $\mathcal{L} \in \mathbb{R}$

---

1. $a^0 := x$
2. for $i = 1, 2, 3, \ldots, L$
    3. $z^i := W^i a^{i-1}$
    4. $a^i := f(z^i)$
3. $\mathcal{L} := \mathrm{CrossEntropy}(a^L)$

Изобразим этот алгоритм в виде схемы из функциональных элементов:



На этой схеме:

- Функциональный элемент Σ^i — это линейное преобразование с весами W^i (сигма — потому что сумматор).
- Функциональный элемент f — это функция активации.

- Функциональный элемент CE считает значение кросс-энтропии (в нее включен софтмакс и взятие логарифма от нужной компоненты)

Алгоритм называют проходом вперед, поскольку его можно представить как путь в СФЭ от входа к выходу.

СФЭ является графической репрезентацией композиции нескольких функций. Например, для случая $L = 1$ это $\mathcal{L} = \mathrm{CE}(f(Wx))$, а для $L = 2$ это $\mathcal{L} = \mathrm{CE}(f(W^2 f(W^1 x)))$.

Проход назад (идейно)

Нейросети обучают градиентными методами по аналогии с тем, как обучают линейные модели классификации:

$$W^i_{\text{new}} = W^i_{\text{old}} - \eta \frac{\partial L}{\partial W^i}, \quad i = \overline{1, L}.$$

Но как считать $\partial \mathcal{L} / \partial W^i$? Воспользуемся правилами дифференцирования сложной функции. Для примера разберем случай $L = 1$, т.е. функцию $\mathcal{L} = \mathrm{CE}(f(Wx))$.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial a} &= \frac{\partial \mathrm{CE}}{\partial a} \\ \frac{\partial \mathcal{L}}{\partial z} &= \frac{\partial f}{\partial z} \frac{\partial \mathcal{L}}{\partial a} \\ \frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \Sigma}{\partial W} \frac{\partial \mathcal{L}}{\partial z} \end{aligned}$$

Таким образом, чтобы вычислить градиент лосса по весу W^i, нужно вычислить градиент лосса по всем активациям с более глубоких слоев (т.е. с номерами $i + 1, \ldots, L$). Т.е. вычисления идут от выхода схемы ($\mathcal{L}$) к ее входу (x). Алгоритм вычисления градиентов весов многослойной сети называют проходом назад (backward pass), или алгоритмом обратного распространения ошибки (bakwards propagated error, backpropagation).

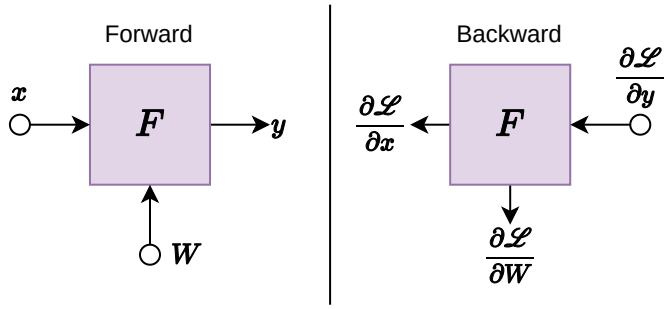
Алгоритм **MLP BACKWARD PASS**

- ВХОД $\mathcal{L} \in \mathbb{R}$
- ВЫХОД $\left\{ \frac{\partial \mathcal{L}}{\partial W^i} \right\}_{i=1}^{L}$

---

1. $\frac{\partial \mathcal{L}}{\partial a^L} := \frac{\partial \mathrm{CE}}{\partial a^L}$
2. for $i = L, L-1, \ldots, 2, 1$:
    3. $\frac{\partial \mathcal{L}}{\partial z^i} := \frac{\partial f}{\partial z^i} \frac{\partial \mathcal{L}}{\partial a^i}$
    4. $\frac{\partial \mathcal{L}}{\partial W^i} := \frac{\partial \Sigma}{\partial W^i} \frac{\partial \mathcal{L}}{\partial z^i}$
    5. $\frac{\partial \mathcal{L}}{\partial a^{i-1}} := \frac{\partial \Sigma}{\partial a^{i-1}} \frac{\partial \mathcal{L}}{\partial z^i}$

Ценность этого алгоритма в его обобщении на произвольную СФЭ с дифференцируемыми функциями. Допустим, имеется функциональный элемент F, реализующий функцию $y = F(x)$, параметризованную весами W. Схематично:



В режиме прохода вперед, элемент F работает с весами и активациями, а в режиме прохода назад — с градиентами лосса по весам и активациям. Эта абстракция находит свое применение в так называемых фреймворках автодифференцирования. Такие фреймворки содержат в себе имплементации проходов вперед и назад для большинства базовых арифметических и матрично-векторных операций. При выполнении любых вычислений такие фреймворки “под капотом” строят граф вычислений и за счет этого позволяют считать все необходимые градиенты автоматически за счет прохода назад по графу.

Универсальный аппроксиматор

До сих пор мы не задавались вопросом о теоретическом обосновании работоспособности нейросетей. Оказывается, таковое имеется. Существует ряд теорем, которые утверждают, что нейронные сети являются универсальными аппроксиматорами в том смысле, что способны приближать любую функцию с любой наперед заданной точностью и при этом иметь конечное число нейронов. Однако стоит сказать, что все эти теоремы имеют ограничения на свойства функции (непрерывность / аналитичность / ограниченность и т.п.). И самое главное, эти теоремы в большинстве своем не отвечают на вопрос, как построить такую нейросеть, чтобы она приближала конкретную функцию и конкретной точностью. Эти теоремы лишь допускают существование таких архитектур и весов, удовлетворяющие условиям задачи.

Подробнее почитать об этом можно в свежем обзоре https://arxiv.org/abs/2407.12895.

Рост числа параметров

Вернемся к задаче классификации изображений на K классов. Насколько большую нейросеть с полносвязными слоями мы можем построить для решения такой задачи? Пусть изображение 100 на 100 пикселей. Входной вектор будет иметь длину 10^4. Допустим, у нас будет два скрытых слоя со 300 нейронами и выходной слой с K нейронами. Тогда веса первого полносвязного слоя это матрица $W^1 \in \mathbb{R}^{300\times10000}$, т.е. это 3×10^6 параметров. Веса второго слоя это матрица $W^2 \in \mathbb{R}^{300\times300}$, т.е. 9×10^4 параметров. Третий слой это $W^3 \in \mathbb{R}^{K\times150}$.

Этот пример показывает, как быстро растет число обучаемых параметров в полносвязной нейросети с увеличением её глубины и ширины (числа слоев и числа нейронов в каждом слое). Заметим, что для работы с изображением не обязательно назначать свой нейрон каждому пикселю входного изображения, поскольку это избыточная репрезентативная мощность. Вместо этого, мы могли бы применять одни и те же параметры к разным частям изображения. Тогда графические примитивы будут детектированы в нескольких местах с помощью одного и того же набора параметров, и не придется заводить “детектор” для каждой части изображения. Именно по этой причине в глубоком обучении существует множество других архитектур.

В нашем курсе мы изучим сверточные, рекуррентные, трансформерные архитектуры. Вся их гениальность и прозорливость состоит в том, что они спроектированы специально для некоторого домена данных (изображения, последовательности, тексты), чтобы учесть все степени свободы и все координаты входных данных таким образом, чтобы наиболее эффективно аллоцировать

обучаемые параметры.

Успех глубокого обучения

Успех глубокого обучения связывают с несколькими факторами.

- DL это автоматическая генерация признаков. Рассмотрим на примере работы с изображениями. До нейросетей все задачи решались следующим образом:
    - CV-инженер генерировал признаки с помощью преобразования Фурье или wavelet-разложения
    - детектировал конкретные геометрические фигуры с помощью сложных алгоритмов и информацию об этом тоже помещал в признаковое описание
    - придумывал еще миллион признаков
    - и все это признаковое описание подавал на вход классическим ML алгоритмам вроде логрега, SVM, random forest

  С приходом нейросетей весь этот сложный пайплайн значительно упростился за счет того, что не нужно “крафтить” признаки (hand-crafted features). Вместо этого нейросеть сама действует как фичегенератор.
- DL это матричные операции. Львиная доля операций, совершаемая при проходе вперед (и при проходе назад, на самом деле) — это перемножение матриц. Эта операция является невероятно оптимизированной для современных процессоров, в частности графических процессоров.
- DL это GPU. За последние десятилетия мощность GPU значительно выросла и они стали доступнее.
- DL это данные. Секрет успеха также кроется в огромном количестве данных. Как правило, нейросети плохо обучаются на маленьких выборках, но при этом умеют отлично генерализировать при обучении на огромных выборках. Говорят, что у нейросетей большая емкость (capacity).
- DL это автодифференцирование. За последние 10 лет фреймворки автодифференцирования стали невероятно эффективными и удобными. Использование автодифа означает, что нейросеть может состоять из совершенно любых операций, главное условие — они должны быть дифференцируемыми. Это открывает огромную гибкость при проектировании и реализации архитектур.