Нейронные сети: под капотом

Математическая модель нейросетей и их обучения

Никита Ковалев

2 апреля 2022

Примерный план

1. Базовые определения
2. Математическая основа обучения
   - Градиентный спуск
   - Backpropagation - обратное распространение ошибки
3. Обзор CNN

Данные, сети, их параметры и целевая функция

Будем обозначать входные данные как $x \in \mathbb{R}^{n_1}$, соответствующий им идеальный результат — $y \in \mathbb{R}^{n_L}$ (в случае датасетов - $x^{\{i\}}$ и $y^{\{i\}}$), а применение нейронной сети к данным — $F(x)$. Поскольку нейронные сети состоят из некоторого количества слоев, обозначим это число через L, веса и свободные члены i-го слоя $W^{[i]}$ и $b^{[i]}$, а результат действия i-го слоя — $a^{[i]}$, тогда

$$a^{[1]} = x,$$

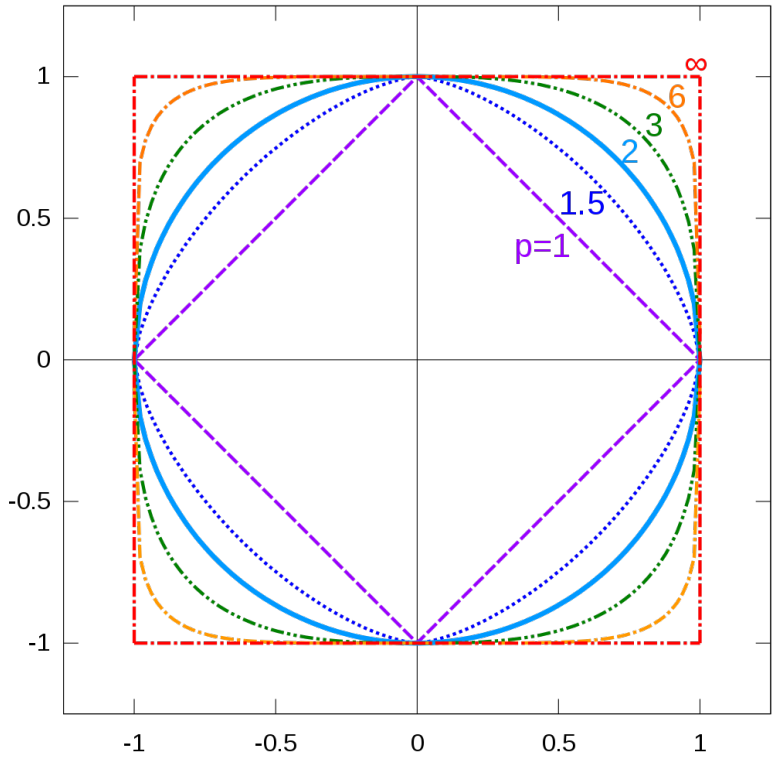
$$a^{[\ell]} = \sigma(W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}) \quad \forall \ell \in \overline{2, L},$$

где $\sigma(x)$ - функция активации. Отсюда также получаем $F(x) = a^{[L]}$.

Обозначем все веса и свободные члены в сети как $\boldsymbol{W}, \boldsymbol{b}$. Тогда, если суммарное число параметров сети обозначить S, можно обозначить их множество как $\theta := (\boldsymbol{W}, \boldsymbol{b}) \in \mathbb{R}^S$. Добавим также множества входных данных и эталонов датасета — X, Y. В таких условиях мы можем ввести обозначение целевой функции $C(\theta|X, Y) : \mathbb{R}^S \to \mathbb{R}$, или просто $C(\theta)$, памятуя о необходимости вычислять её на некотором множестве данных. Тогда задача обучения нейронной сети на датасете сводится к задаче оптимизации $\min_{\boldsymbol{W}, \boldsymbol{b}} C(\boldsymbol{W}, \boldsymbol{b}|X, Y)$.

Норма

Введем понятие нормы. Для этого сначала зададим векторное пространство — в нашем случае это $\mathbb{R}^n$ для некоторого n — и его различные подпространства ℓ_p для $0 < p < \infty$. Такие подпростанства состоят из векторов $x \in \mathbb{R}^n$ $s.t.$ $\sum_{i=1}^{n} |x_i|^p < \infty$. Для случая, когда $p \geq 1$, можно ввести соответствующую норму: $||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$, иначе будем иметь только метрику: $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|^p$.



Пример целевой функции

Теперь мы можем привести пример реализации простой целевой функции:

$$C(p|X,Y) = \frac{1}{N}\sum_{i=1}^{N} C(\boldsymbol{W},\boldsymbol{b}|x^{\{i\}},y^{\{i\}}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}||y^{\{i\}} - F(x^{\{i\}})||_2^2 \quad (1)$$

Понятно, что аргументы функции на самом деле содержатся в forward-pass нашей сети $F(x)$. Таким образом, после некоторого обучения значения параметров сети будут такими, что они в некотором (возможно, локальном) смысле минимизируют целевую функции — иначе говоря, приближают поведение нейронной сети на входных данных к тому, что от неё ожидается в идеальном случае.

Для датасетов и сетей, где ожидаемым ответом являются не просто числа из какого-то поля, а некий класс или категория, используются такие функции, как cross-entropy loss function. Почитать про разные функции можно, например, здесь.

Общий принцип обучения и градиентный спуск

Обучение модели машинного обучения — итерационный процесс. С каждый прогоном по датасету мы получаем обновленную информацию об отклонениях результатов работы сети от идеала и можем использовать это для улучшения весов. Самым простым и интуитивно понятным вариантом является алгоритм градиентного спуска.
Учитывая принцип небольших постепенных обновлений значений параметров и применив формулу Тейлора, посмотрим, как приблизительно меняется функция при малых изменениях аргумента:

$$C(p + \Delta p) = C(p) + \sum_{r=1}^{S} \frac{\partial C(p)}{\partial p_r} \Delta p_r = C(p) + (\nabla C(p))^{\tau} \Delta p$$

Отсюда можно сделать вывод, что для уменьшения значения целевой функции нам необходимо каждый раз выбирать изменения параметра, противоположные градиенту $C(p)$ при нынешних весах модели. Таким образом получаем формулу для обновления весов:

$$p_{t+1} := p_t - \eta \nabla C(p_t) = p_t - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla C(p_t | x^{\{i\}}, y^{\{i\}}), 0 < \eta \ll 1$$

Обратный проход — понятия и их мат. выражение

Для начала введем необходимые обозначения.

Пре-активации: $z^{[\ell]} := W^{[\ell]} a^{[\ell-1]} + b^{[\ell]} \in \mathbb{R}^{n_\ell}, \ell \in 2, 3, .., L$. Под этим подразумеваются вычисления очередного слоя, взятые до того, как пройти через свою функцию активации.

Полный результат прогона каждого из слоев превращается в $a^{[\ell]} = \sigma(z^{[\ell]})$

Под вектором ошибок по определению будем иметь в виду градиент нашей целевой функции относительно результатов работы сети в заданном месте прогона. Тогда на каждом слое имеем: $\delta^{[\ell]} \in \mathbb{R}^{n_\ell}, \delta_j^{[\ell]} := \frac{\partial C}{\partial z_j^{[\ell]}}, 1 \leq j \leq n_\ell$

Также нам понадобится обозначение произведения Адамара (Hadamard product): $\forall x, y \in \mathbb{R}^n \;\; x \circ y \in \mathbb{R}^n \;\; s.t. \;\; (x \circ y)_k = x_k y_k \;\; \forall k \in \overline{1, n}$

Теперь мы можем определить теорему о backpropagation (крайне рекомендую видео по ссылке и канал автора), которая в сути является следствием применения chain rule для обновления весов послойно в обратном порядке на основе градиентов.

Обратный проход — теорема

Theorem

Для обновления весов и свободных членов сети справедливы следующие утверждения:

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^{[L]} - y) \tag{2}$$

$$\delta^{[\ell]} = \sigma'(z^{[\ell]}) \circ (W^{[\ell+1]})^{\tau} \delta^{[\ell+1]}, 2 \leq \ell \leq L-1 \tag{3}$$

$$\frac{\partial C}{\partial b_j^{[\ell]}} = \delta_j^{[\ell]}, 2 \leq \ell \leq L \tag{4}$$

$$\frac{\partial C}{\partial W_{jk}^{[\ell]}} = \delta_j^{[\ell]} a_k^{[\ell-1]}, 2 \leq \ell \leq L \tag{5}$$

Обратный проход — доказательство 1

Доказательство.

Докажем (2). Имеем:

$$\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = \frac{\partial}{\partial z_j^{[L]}}(\sigma(z^{[L]})) = \sigma'(z_j^{[L]})$$

$$\frac{\partial C}{\partial a_j^{[L]}} \overset{(1)}{=} \frac{\partial}{\partial a_j^{[L]}}(\frac{1}{2}\sum_{k=1}^{n_L}(y_k - a_k^{[L]})^2) = -(y_j - a_j^{[L]})$$

$$\delta_j^{[L]} = \frac{\partial C}{\partial z_j^{[L]}} = \frac{\partial C}{\partial a_j^{[L]}}\frac{\partial a_j^{[L]}}{z_j^{[L]}} = (a_j^{[L]} - y_j)\sigma'(z_j^{[L]})$$

▲

Обратный проход — доказательство 2

Доказательство.

Докажем (3):

$$\delta_j^{[\ell]} = \frac{\partial C}{\partial z_j^{[\ell]}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial C}{\partial z_k^{[\ell+1]}} \frac{\partial z_k^{[\ell+1]}}{\partial z_j^{[\ell]}} = \sum_{k=1}^{n_{\ell+1}} \delta_j^{[\ell]} \frac{\partial z_k^{[\ell+1]}}{\partial z_j^{[\ell]}}$$

$$z_k^{[\ell+1]} = \sum_{s=1}^{n_\ell} W_{ks}^{[\ell+1]} \sigma(z_s^{[\ell]}) + b_k^{[\ell+1]} \implies \frac{\partial z_k^{[\ell+1]}}{\partial z_j^{[\ell]}} = W_{kj}^{[\ell+1]} \sigma'(z_j^{[\ell]})$$

$$\delta_j^{[\ell]} = \sum_{k=1}^{n_{\ell+1}} \delta_k^{[\ell+1]} W_{kj}^{[\ell+1]} \sigma'(z_j^{[\ell]}) = \sigma'(z_j^{[\ell]})(W^{[\ell+1]})^\tau \delta^{[\ell+1]}$$

▲

Обратный проход — доказательство 3

Доказательство.

Докажем (4).
Gradient w.r.t (with respect to) bias — $\frac{\partial C}{\partial b_j^{[\ell]}}$ — ?

$$z_j^{[\ell]} = (W^{[\ell]}\sigma(z^{[\ell-1]}))_j + b_j^{[\ell]} \implies \frac{\partial z_j^{[\ell]}}{\partial b_j^{[\ell]}} = 1$$

$$\frac{\partial C}{\partial b_j^{[\ell]}} = \frac{\partial C}{\partial z_j^{[\ell]}} \frac{\partial z_j^{[\ell]}}{\partial b_j^{[\ell]}} = \delta_j^{[\ell]}$$

▲

Доказательство (5) производится аналогично, проведите его самостоятельно.

Суть и плюсы свёрточных сетей

Свёрточными сетями называются такие модели, у которых в архитектуре имеются свёрточные слои. Такие слои, в свою очередь, состоят из некоторого числа т.н. фильтров, каждый из которых имеет своё, независимое от других, ядро свёртки, то есть вместо fully connected весов мы будем иметь очень разреженную матрицу переходов. Таким образом достигаются многие полезные свойства, за которые свёрточные сети и ценятся.

Главные плюшки CNN:

- Разреженность весов — малые расходы по памяти в сравнении с FC слоями
- Сюда же — общие параметры ядра в рамках каждого фильтра
- Трансляционная инвариантность
- Локальность извлекаемых свойств
- Инвариантность масштабов
- Способность выделять признаки в разных условиях/контексте

Да вы из Англии — подробнее про свойства свёрточных слоёв

- Kernel — tensor of shape $(n, m, k_1, ..., k_N)$, where $n = \#$ output maps, $m = \#$ input maps, $k_j =$ kernel size in dimension j, $N =$ dimension size ($N = 2$ in image processing)
- In standard convolution, for input size i in some dimension the output size will be $o = (i - k) + 1$, where k is the kernel size for corresponding dimension
- Padding — adding some superficial data around the input. Padding types: "same" ($p = k - 1$) to achieve $o = i$ (usually zeros or border repeat), "full" ($p = 2(k - 1)$) to achieve $o = i + (k - 1)$, "valid" — no padding
- Stride — the distance between consecutive positions where a kernel is applied. It is a way of downsampling the inputs
- General formula for output size: $o = \lfloor \frac{i+p-k}{s} \rfloor + 1$
- Pooling layers are often combined with the convolutional ones. They are used to take values from the local neighbourhoods with respect to some pattern (MaxPool, AvgPool, etc.)