# 1 Первая домашка

В математике матрица не может быть 3 и более мерной. В ML многомерные объекты называются **тензорами**. Частным случаем тензора является скаляр, вектор или матрица.

Нейрон выполняет взвешенную сумму входов:

$$\sum \text{weight} \cdot \text{input} + \text{bias},$$

где bias — пороговое значение или смещение.

**Функция активации** отвечает за активацию нейрона и определяет выходное значение. Нейрон игнориуется в сети, когда функция возвращает 0, иначе нейрон считается активированным.

Простейшая фукнция активации — ступенька:

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0. \end{cases}$$

Хорошо подходит для бинарной классификации.

Линейная функция:

$$f(x) = cx.$$

Имеет построянную производную, что мешает выполнять градиентный спуск по эмпирическому риску. Позволяет несколько слоев представить в виде одного.

Логистическая фукнция (она же сигмоидная):

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Ограничена 0 и 1, что позволяет использовать ее на последнем слое для вероятностного представления. Производная этой функции мала во всех точках, кроме небольшого промежутка, что порождает проблему исчезающего градиента. Производная:

$$\begin{aligned} \sigma'(x) &= -1(1 + e^{-x})^{-2} e^{-x}(-1) \\ &= \frac{e^{-x}}{(1 + e^{-x})^2} \end{aligned}$$

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

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Гиперболический тангенс:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 = 2\sigma(2x) - 1$$

Является скорректированной сигмоидой, имеет более крутую производную, но ограничена $-1$ и 1.

ReLU (Rectified Linear Unit):

$$f(x) = \max(0, x)$$

Позволяет полностью отключить часть нейронов и облегчить сеть.

Leaky ReLU:

$$f(x) = \begin{cases} 0.01x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

Утечка позволяет избавиться от нулевого градиента при отрицательных значениях.

Слой softmax позволяет получить вероятностное распределение по выходу предыдущего слоя. Функция активации для каждого $i$-нейрона в слое:

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Нейронная сеть с прямой связью. Feedforward networks. Нейросеть, в которой соединения между узлами не образуют цикл. Первый и самый простой тип нейросети.

Выход слоя называют **локальным полем**, сам слой называют вычислительным блоком сети.

ANN (artifical neural network) — искуссвтенная нейронная сеть.

- входы сети: $x = a^{[0]}$
- локальное поле: $z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$, где $l$ — номер слоя, $W$ — веса, $b$ — смещение.
- активация: $a^{[l]} = g^{[l]}(z^{[l]})$, где $g$ — функция активации,
- выходы сети: $\hat{y} = a^{[L]}$, где $L$ — количество слоев сети
- скалярная функция потерь (loss): $J(\hat{y}, y)$.

Для обновления весов и смещения, нам нужно выполнять градиентный спуск (gradient descent):

$$W^{[l]} = W^{[l]} - \eta \frac{\partial J}{\partial W^{[l]}}$$

$$b^{[l]} = b^{[l]} - \eta \frac{\partial J}{\partial b^{[l]}}$$

где $\eta$ — шаг (learning rate parameter).

На выходе должен получиться градиент той же размерности, что и тензор, который мы обновляем.

Для нахождения производной будем пользоваться цепным правилом дифференцирования. Пример для двух слоев:

$$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

Локальный градиент — частная производная определенного слоя:

$$\delta^{[l]} = \frac{\partial J}{\partial z^{[l]}}$$

Используя это обозначение тот же пример:

$$\frac{\partial J}{\partial W^{[1]}} = \delta^{[1]} \frac{\partial z^{[1]}}{\partial W^{[1]}}.$$

Рассмотрим каждый элемент локального поля:

$$z_i = \sum_{k=1}^{n} W_{ik} a_k + b_i$$

Производная локального поля по весам будет матрицей Якоби, каждой элемент которой:

$$\frac{\partial z_i}{\partial a_j} = W_{ij}$$

Таким образом:

$$\frac{\partial W a}{\partial a} = W$$

Для активации:

$$a_i = g(z_i)$$

$$\frac{\partial a_i}{\partial z_j} = \frac{\partial g(z_i)}{\partial z_j} = g' I_{\{i=j\}}$$

Тогда матрица Якоби:

$$\frac{\partial a}{\partial z} = diag(g')$$

Иногда эту диагональ можно записать в виде вектора. Обозначается так: $\circ g'(x)$

Аналогично для смещения:

$$/\frac{\partial J}{\partial b^{[l]}} = \delta^{[l]}$$

Для веса мы не можем записать производную в матричной форме, но если выразить через локальный градиент, то все получится. Для этого нам придется использовать локальный градиент.

$$z_i(W) = \sum_{k} W_{ik} a_k + b_i$$

$$\frac{\partial z_i(W)}{\partial W} = \nabla z_i(W) = \left( \frac{\partial z_i(W)}{\partial W_{js}} = a_s I_{\{j=i\}} \right)_{js}$$

Получили матрицу, где в одной строке $a$.

$$\frac{\partial z}{\partial W_{js}} = \left( \frac{\partial z_i}{\partial W_{js}} = a_s I_{j=i} \right)_i$$

Получили столбец с одним элементом.

$$\frac{\partial J}{\partial W_{js}} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial W_{js}} = \delta\frac{\partial z}{\partial W_{js}} = \delta_j a_s$$

Здесь матричное умножение двух столбцов волшебным образом дало скалярное произведение, что похоже на ошибку.

Таким образом, производная потерь по весам:

$$\frac{\partial J}{\partial W} = \nabla_W J = \left(\delta_i a_j\right)_{ij} = \delta a^T$$

Последнее равенство определяет матрицу в виде внешнего произведения (outer product).

Локальное поле через другое:

$$\delta^{[l]} = \delta^{[l+1]}\frac{\partial z^{[l+1]}}{\partial a^{[l]}}\frac{\partial a^{[l]}}{\partial z^{[l]}} = W^{T\,[l+1]}\delta^{[l+1]} \circ g'^{\,[l]}(z^{[l]})$$

Здесь опять произошла магия с перестановкой матрицы весов. Препод объяснил это тем, что размерности тензоров могут не совпадать.

Вручную считать производные функций не очень хорошо. **Автоматическое дифференцирование (automatic differentiation)**. Строим граф вычислений. Для $f(x_1, x_2) = sin(x_1) + x_1 \cdot x_2$:

$w_1 = x_1,$
$w_2 = x_2,$
$w_3 = \sin(w_1)$
$w_4 = w_1 \cdot w_2$
$w_5 = w_3 + w_4$

Можно дифференцировать в обоих направлениях, но на практике как правило удобней использовать обратное направление (от $y$ к $x$).

$Dw_5 = \frac{\partial y}{\partial w_5} = 1$

$Dw_4 = \frac{\partial w_5}{\partial w_4} = 1$

$Dw_3 = \frac{\partial w_5}{w_3} = 1$

$Dw_2 = \frac{\partial w_5}{\partial w_2} = \frac{\partial w_5}{\partial w_3}\frac{\partial w_3}{\partial w_2} + \frac{\partial w_5}{\partial w_4}\frac{\partial w_4}{\partial w_2} = 1 \cdot 0 + 1 \cdot w_1 = w_1$

$Dw_1 = \frac{\partial w_5}{\partial w_1} = \frac{\partial w_5}{\partial w_3}\frac{\partial w_3}{\partial w_1} + \frac{\partial w_5}{\partial w_4}\frac{\partial w_4}{\partial w_1} = 1 \cdot \cos(w_1) + 1 \cdot w_2 = \cos(w_1) + w_2$

Мы использовали цепное правило дифференцирования для многомерного случая (multivariable chain rule), обозначение $D$.

## 1.1 Описание фреймворка

Наш фреймворк должен уметь вычислять градиент на $l$ слое и передавать его дальше (по основному стволу дерева). И градиенты для $W$ и $b$ (листы). Это для полносвязных сетей.

### 1.2 Соглашения по дифференцированию сложных функций

Вектор определяется как матрица с одним столбцом:

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Для векторной функции $f : \mathbb{R}^n \to \mathbb{R}^m$ производная является матрицей Якоби:

$$\frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

Вектор-функция (vector function) отображает аргумент (скаляр или вектор) в векторное пространство.

Для функций $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ производная называется гардиентом. Размерность градиента определяется размерностью аргумента:

$$\nabla_x f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_{11}} & \cdots & \frac{\partial f(x)}{\partial x_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(x)}{\partial x_{m1}} & \cdots & \frac{\partial f(x)}{\partial x_{mn}} \end{pmatrix}$$

Когда ищем производную, рассматриваем поэлементно.

# 2 Вторая домашка

# 3 RNN

Рекурентные нейронные сети (). Базовая архитектура для работы с данными, которые представляются в виде какой-то последовательности (временной ряд температуры, например). Могут быть использованы для работы с естественным текстом.

Sequence Model.

Последовательность можно представить в виде вектора $x^{(t)}$. $t$ — время или просто порядковый номер элемента последовательности. $x$ — последовательность векторов (возможно из одного элемента) $x^{(i)}$.

Мы хотим получить условное распределение, которое по по истории позволит получить следующий элемент последовательности.

$$x^{(t)} \sim p\left(x^{(t)} \mid x^{(t-1)}, \cdots, x^{(1)}\right)$$

Символ читается как is distributed as

Обчно учитываются не все исторические данные, а их часть в окне:

$$x^{(t)} \sim p\left(x^{(t)} \mid x^{(t-1)}, \cdots, x^{(t-n+1)}\right)$$

Чтобы не хранить большое количество данных, можем определить вектор $h^{(t)}$, который будет каким-то образом инкапсулировать предыдущие данные:

$$x^{(t)} \sim p\left(x^{(t)} \mid x^{(t-1)}, h^{(t-1)}\right)$$

чтобы получить этот вектор $h^{(t)} = f\left(x^{(t-1)}, h^{(t-1)}\right)$, $f$ — нейронная сеть.

Мы мерили температуру за окном, потом решили предсказать температуру в комната. Или обучались на текстах одного жанра, потом предстказываем другой. Мы предполагаем, что во время обучения модели и инференса распределения не поменялись (стационарность в статистике).

Inference (вывод) Инференс — это выводы обученной нейронной сети для конкретных примеров.

Можем использовать цепное правило вероятностей (probability chain rule), чтобы определить вероятность вектора:

$$p\left(x^{(1)}, \cdots, x^{(t)}\right) = \prod_{\tau=1}^{t} p\left(x^{(\tau)} \mid x^{(\tau-1)}, \cdots, x^{(1)}\right)$$

Probability chain rule:

$$P(A, B, C) = P(A \mid B, C)P(B, C) = P(A \mid B, C)P(B \mid C)P(C)$$

Если $x^{(t)}$ является непрерывным (цена, температура), то задача является регрессионной, иначе если объекты дискретные (слова) получаем классификацию. В задаче классификации могут быть тысячи классов, несмотря на то, что ощущается такая зада как регрессионная.

Далее мы будем рассматривать задачу классификации. Классификация от регрессии будет отличаться только функцией потерь.

Примеры.

Марковский процесс первого порядка:

$$p\left(x^{(1)}, \cdots, x^{(t)}\right) = \prod_{\tau=1}^{t} p\left(x^{(\tau)} \mid x^{(\tau-1)}\right)$$

Простая рекурентная сеть:

Представим себе текст или символы входящей строки: $x^{(t)}$. Эти символы берутся из словаря.

Наша цель — построить модель для условной вероятности для текущего токена на основании предыдущих $n$ токенов. Полученную вероятность будум обозначать $\hat{y}^{(t)}$.

$$p\left(x^{(t)} \mid x^{(t-1)}, \cdots, x^{(t-n+1)}\right) = \hat{y}^{(t)}$$

Если мы хотим хранить такую длинную историю, то нм понадобится $|V|^n$ параметров.

Конвертируем историю в скрытое состояние $h^{(t)}$.

$X^{(t)} \in R^{|V|}$ — унитарный (one-hot) вектор соответствующий $x^{(t)}$. Кажая $x^{(t)}$ — токен, каждому токену отвечает порядковый номер в словаре, единичка будет в компоненте с этим номером.

$E \in R^{d \times |V|}$ — **матрица представлений** (embedding matrix). Работать с унитарными векторми не очень удобно, так как они разряженные. d — размерность представления, $|V|$ — число токенов в словаре. Т. е. мы каждому токену сопоставляем $d$-мерный вектор.

Определим модель:

Получаем соответствующее представление.

$$e^{(t)} = E \cdot X^{(t)}$$

Строим модель, которая дает представления для вектора состояния $h$. Добавили нелинейность гиперболическим тангенсом.

$$h^{(t)} = \tanh(z_1^{(t)})$$

Наша финальная вероятность (является one-hot-вектором):

$$\hat{y}^{(t)} = softmax(z_2^{(t)})$$

Полносвязная — это значит, что каждый нейрон связан со всеми нейронами предыдущего слоя

$z_1$ и $z_2$ — обычные плоносвязные нейронные сети

$$z_1^{(t)} = W_h h^{(t-1)} + W_e e^{(t)} + b_1$$

или можем рассмотреть аналогичную модель с конкатенацией векторов (объединяются в более длинный вектор):

$$z_1 = W_1 \cdot [h^{(t-1)}; e^{(t)}] + b_1$$

где $W_h \in R^{d_h \times d_h}$, $W_e \in R^{d_h \times d}$, $b_1 \in R^{d_h}$. Вектор состояиня и представление преобразуется матрицами $W$ и добавляется смещение.

$$z_2^{(t)} = W_2 h^{(t)} + b_2$$

где $W_2 \in R^{|V| \times d_h}$, $b_2 \in R^{|V|}$.

Для начального момента времени (1) нам нужно определить $h^{(0)}$. Обычно это нулевой вектор.

Функция потерь является кросс-энтрпией:

$$J^{(t)} = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}.$$

Когда распределения совпадут, будет равняться 0, иначе будет увеличиваться.

Получилась сеть из двух полносвязных слоев. Второй слой нужен чтобы прийти к правильной размерности, чтобы на выходе получилась вероятность над всем словарем.

Отличие этой модели от просто двуслойной полносвязной сети заключается в том, что имеется циклический блок $z_1$, через который входной сигнал проходит $t$ раз при вычислении $h^{(t)}$.

Общая функция потерь для модели:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

Backpropagation Through Time (BTT):

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{\tau=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\bigg|_{(\tau)} \frac{\partial W_h|_{(\tau)}}{\partial W_h} = \sum_{\tau=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\bigg|_{(\tau)}$$

Нам нужно продифференцировать нашу функцию потерь для всех промежуточных моментов времени и просуммировать.

We backpropagete over time steps $\tau = t, t-1, \cdots, 0$ and sum gradients.

Сумма появляется из правила полной производной для функции $f'(x_1(t), x_2(t))$.

Реализация без дерева вычислений. Будем кэшировать выходное значение каждого слоя (блока). Когда будем возвращаться в методе обратного распространения ошибки (метод backward), будем из кэша брать все в обратном порядке.

Реализация: