Занятие 2. Глубокие нейронные сети и метод обратного распространения ошибок

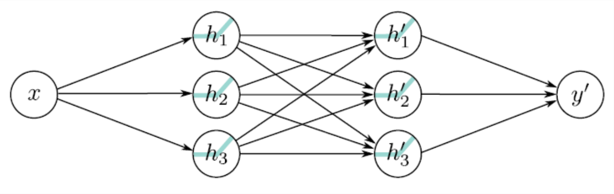
Гирдюк Дмитрий Викторович

22 февраля 2025

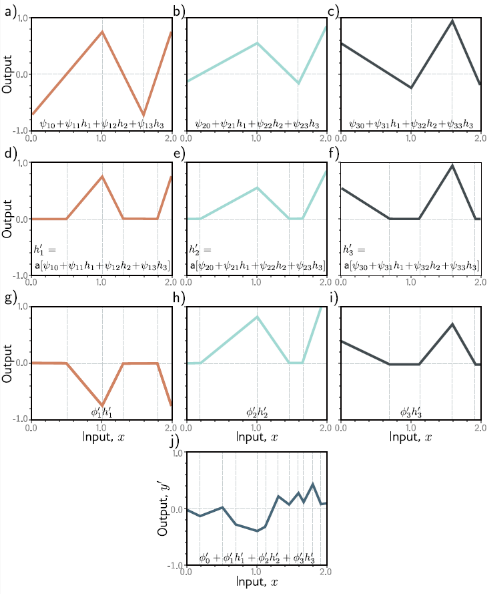
СПбГУ, ПМ–ПУ, ДФС

Многослойные нейронные сети

- Двигаемся дальше, рассмотрим многослойные сети
- Несмотря на универсальные теоремы аппроксимации, для многих задач число скрытых нейронов в однослойной сети, необходимых для хорошей аппроксимации зависимости, может быть очень велико
- Многослойные сети позволяют создавать куда большее число линейных регионов для фиксированного числа параметров



Визуализация двухслойной нейронной сети [1]



Зависимость числа линейных регионов от параметров [1]

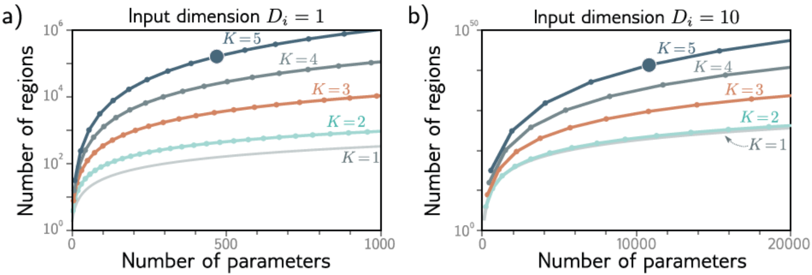


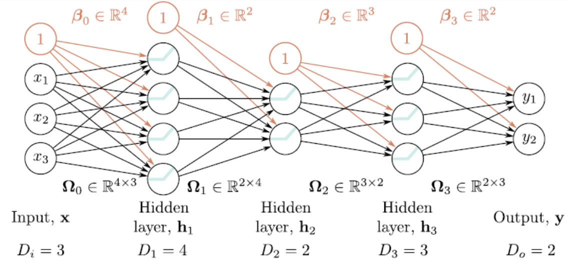
Figure 4.7 The maximum number of linear regions for neural networks increases rapidly with the network depth. a) Network with $D_i = 1$ input. Each curve represents a fixed number of hidden layers K, as we vary the number of hidden units D per layer. For a fixed parameter budget (horizontal position), deeper networks produce more linear regions than shallower ones. A network with $K = 5$ layers and $D = 10$ hidden units per layer has 471 parameters (highlighted point) and can produce 161,051 regions. b) Network with $D_i = 10$ inputs. Each subsequent point along a curve represents ten hidden units. Here, a model with $K = 5$ layers and $D = 50$ hidden units per layer has 10,801 parameters (highlighted point) and can create more than 10^{40} linear regions.

Многослойные нейронные сети II

- Общий случай полносвязной сети прямого распространения:

$$\boldsymbol{y} = \boldsymbol{b^d} + \boldsymbol{W^d} a\left(\boldsymbol{b^{d-1}} + \boldsymbol{W^{d-1}} a\left(\ldots \boldsymbol{b^2} + \boldsymbol{W^2} a(\boldsymbol{b^1} + \boldsymbol{W^1} \boldsymbol{x}) \ldots\right)\right)$$

- Современные архитектуры для решения прикладных задач могут содержать в себе огромное число параметров, от нескольких миллионов до сотен и даже тысяч миллиардов.
- Формальная запись нейронной сети нужна нам для выработки понимания того, как именно происходит настройка ее параметров



Обучение нейронных сетей

- Как и ранее в курсе по машинному обучению, для обучения нейронных сетей нам нужны целевая (лосс) функция и алгоритм оптимизации
- Конечно, в общем случае существенное количество параметров и сложность архитектуры ограничивают нас использованием оптимизационных методов лишь первого порядка
- Возникает вопрос, как эффективно находить производные параметров сети?
- С помощью алгоритма обратного распространения ошибки (backpropagation)

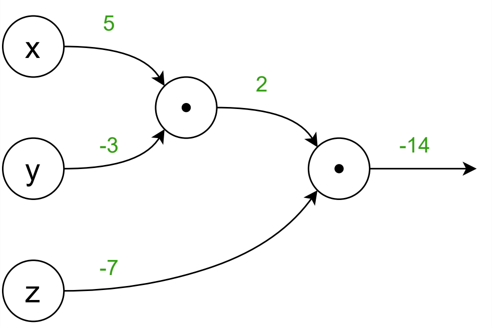
Цепное правило [2]

- Рассмотрим вычислительный граф достаточно простой функции

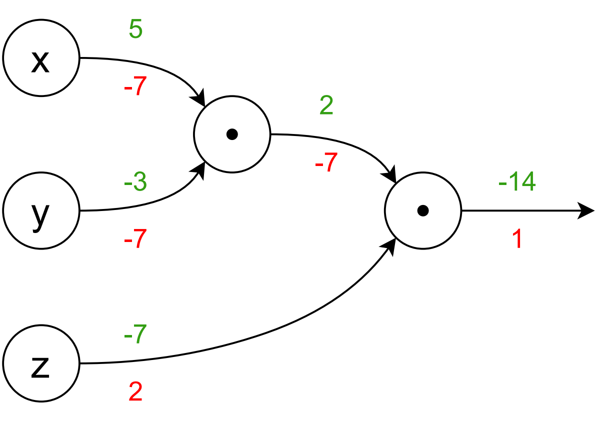
$$f(x, y, z) = (x + y)z$$

- Как по определению производится вычисление частных производных сложной функции?

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$



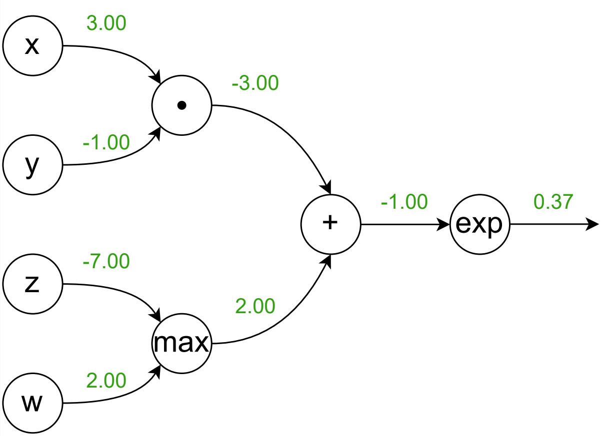
Цепное правило II [2]



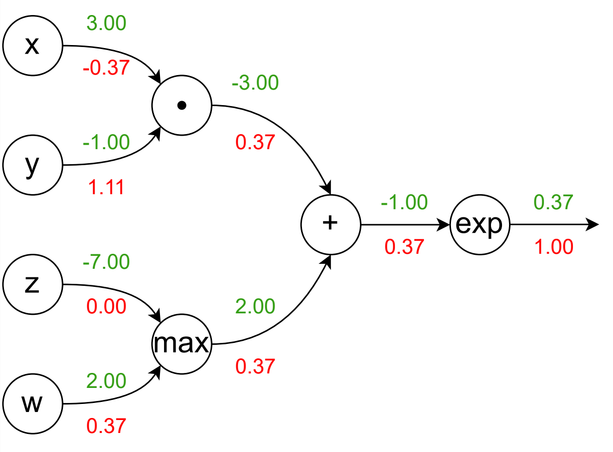
Цепное правило III

- Еще один пример

$$f(x, y, z, w) = \exp(xy + max(z, w))$$



Цепное правило III



- И еще примеры для тренировки: Samsung AI CV course link

Цепное правило IIII

- В матричном виде все не намного сложнее. Помнить все формулы наизусть не обязательно. Хватает здравого смысла и знания размерностей тензоров
- Рассмотрим матричное умножение внутри некоторой сети

$$\boldsymbol{Y}_{[n,d]} = \boldsymbol{X}_{[n,m]} \boldsymbol{W}_{[m,d]}$$

- Пусть нам известна частная производная некоторой лосс-функции L (помним, что L есть скаляр) по $\boldsymbol{Y}$
- Тогда частные производные лосс-функции по параметрам и входам $\boldsymbol{W}$ равны

$$\frac{\partial L}{\partial \boldsymbol{W}}_{[m,d]} = \boldsymbol{X}^T_{[m,n]} \frac{\partial L}{\partial \boldsymbol{Y}}_{[n,d]}$$

$$\frac{\partial L}{\partial \boldsymbol{X}}_{[n,m]} = \frac{\partial L}{\partial \boldsymbol{Y}}_{[n,d]} \boldsymbol{W}^T_{[d,m]}$$

Цепное правило IV

- Для случая поэлементных применений функций активации с вычислением производных тоже все должно быть понятно (правда?)
- Задание на дом: как будет выглядеть производная кросс-энтропии $L(\boldsymbol{s}, \boldsymbol{y}) = -\boldsymbol{y}^T log(\boldsymbol{s})$ по параметрам $\boldsymbol{W}$ при

$$\boldsymbol{S}_{[n,d]} = \text{Softmax}\left(\boldsymbol{X}_{[n,m]}\boldsymbol{W}_{[m,d]}\right),$$

где $\text{Softmax}(\boldsymbol{y}) = \left(\frac{\exp(y_1)}{\sum_j \exp(y_j)}, \ldots, \frac{\exp(y_d)}{\sum_j \exp(y_j)}\right)$?

Метод обратного распространения ошибки

- То, что мы делали на предыдущих слайдах, и есть метод обратного распространения ошибки
- Центральная идея состоит в последовательном вычислении производных по параметрам сети в обратном порядке вычислительного графа
- Зафиксировав заранее семество функций и производных для них, которые будут использоваться для построения вычислительного графа/архитектуры сети, процесс подсчета градиентов можно автоматизировать

Подробнее про функции активации [3]

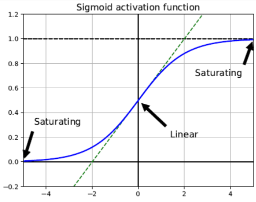
- Хотя ReLU использовалась в качеству функции активации еще в 60-х, сигмоида в то время была наиболее частым выбором
- Она является гладкой аппроксимацией функции Хевисайда

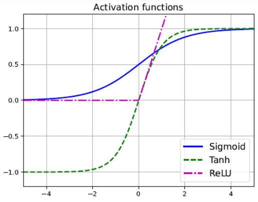
$$H(x) = \begin{cases} 1, & x \geqslant 0 \\ 0, & x < 0 \end{cases}$$

  которая, в свою очередь, использовалась в перцептроне Розенблатта
- Кстати, многослойные нейронные сети прямого распространения с произвольными нелинейными функциями активации также принято называть многослойным перцептронами

Проблема затухающих градиентов

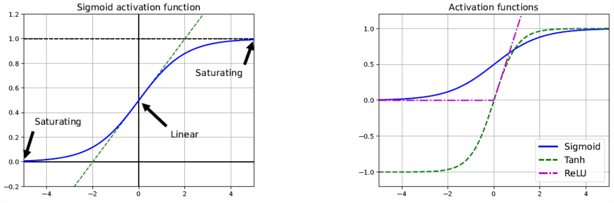
- Ключевая проблема сигмоиды состоит в том, что для больших по абсолютному значению входов ее производная стремится к 0
- Действительно, $\sigma'(x) = \sigma(x)\,(1 - \sigma(x))$
- Почему это проблема? В таком случае градиенты с поздних слоев не смогут "пробиться" до ранних слоев, что существенно усложняет тренировку параметров
- Это явление принято называть "проблема затухания градиентов" (vanishing gradient problem)





Проблема затухающих градиентов II

- От анологичной проблемы страдает другая "древняя" функция активации, гиперболический тангенс (tanh)
- Хорошо, как это побороть?
- Первое, что приходит на ум – использовать те функции активации, которые от этого не страдают!
- Например, рассмотренная ранее ReLU обладает очень приятным свойством: для неотрицательных входов ее производная равна 1



Недостатки ReLU

- Но и у ReLU есть свои недостатки
- Что произойдет, если при инициализации или в рамках обучения некоторые веса станут отрицательными и достаточно большими по абсолютному значению (например, смещения)?
- Если веса настроились таким образом, что на любом входном векторе нейрон с ReLU всегда дает отрицательные значения, то никакого обновления весов больше не предвидится, и этот нейрон по своей сути станет "мертвым"

Недостатки ReLU

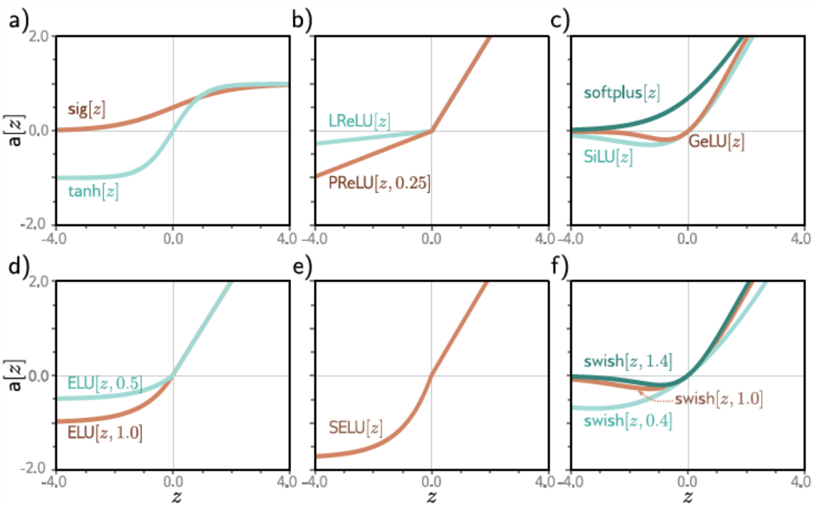
- Но и у ReLU есть свои недостатки
- Что произойдет, если при инициализации или в рамках обучения некоторые веса станут отрицательными и достаточно большими по абсолютному значению (например, смещения)?
- Если веса настроились таким образом, что на любом входном векторе нейрон с ReLU всегда дает отрицательные значения, то никакого обновления весов больше не предвидится, и этот нейрон по своей сути станет "мертвым"

Основные функции активации

Name	Definition	Range
Sigmoid	$\sigma(a) = \frac{1}{1+e^{-a}}$	$[0, 1]$
Hyperbolic tangent	$\tanh(a) = 2\sigma(2a) - 1$	$[-1, 1]$
Softplus	$\sigma_+(a) = \log(1 + e^a)$	$[0, \infty]$
Rectified linear unit	$\mathrm{ReLU}(a) = \max(a, 0)$	$[0, \infty]$
Leaky ReLU	$\max(a, 0) + \alpha \min(a, 0)$	$[-\infty, \infty]$
Exponential linear unit	$\max(a, 0) + \min(\alpha(e^a - 1), 0)$	$[-\infty, \infty]$
Swish	$a\sigma(a)$	$[-\infty, \infty]$
GELU	$a\Phi(a)$	$[-\infty, \infty]$

Рис. 1: Наиболее известные функции активации [3]

Визуализация основных функций активации [1]



Заключительные замечания по функциям активации

- Мы рассмотрели различные функции активации, предназначенные для борьбы с проблемой затухающих градиентов
- Увы, единого ответа на вопрос о том, какую функцию активации использовать, нет
- Несмотря на свои недостатки, предложенные обобщения и альтернативы, ReLU – сильный бэйзлайн и выбор по умолчанию, зачастую, с точки зрения результатов, показывает себя не хуже остальных

Заключительные замечания по затухающим градиентам

- Есть ли еще способы борьбы с затухающими градиентами?
- Да, и даже несколько
- Residual connections (остаточные связи), познакомимся чуть позже
- Использование одной и той же функции активации не только для всех нейронов в слое, но и во всей архитектуре сети
- Наконец, очень важно правильно инициализировать параметры нейронной сети (рассмотрим этот вопрос в следующей лекции)

Проблема взрывающихся градиентов

- Хорошо, мы разобрали ситуацию, когда градиенты обращаются в ноль
- А что с обратной ситуацией, когда градиенты, наоборот, экспоненциально увеличиваются?
- Тут решение достаточно простое, предлагается градиенты обрезать (gradient clipping)

$$g' = \min\left(1, \frac{c}{||g||} g\right)$$

где c есть заранее заданный порог

Итого

- Познакомились с многослойными сетями, зафиксировали их преимущества в контексте аппроксимации нелинейных зависимостей с меньшим совокупным числом параметров
- Разобрались с методом обратного распространения ошибки
- Познакомились с основными функциями активации, их преимуществами и недостатками
- На следующем занятии начнем знакомство с библиотекой глубокого обучения PyTorch и платформой Google Colab aka "Jupyter Notebook in the cloud" с доступом к вычислительным ресурсам (GPU, TPU)

Использованные источники i

1. *Prince S. J.* **Understanding Deep Learning.** The MIT Press, 2023. URL: `http://udlbook.com`.
2. *Stanford*. **CS231n: Deep Learning for Computer Vision.** URL: `https://cs231n.stanford.edu/`.
3. *Murphy K. P.* **Probabilistic Machine Learning: An introduction.** MIT Press, 2022. URL: `probml.ai`.