Нейронные сети

Сергей Николенко

Академический Университет, весна 2011

Outline

Почему мы лучше?

- Компьютер считает быстрее человека
- Но гораздо хуже может:
  - понимать естественный язык
  - узнавать людей
  - обучаться в широком смысле этого слова
  - ...
- Почему так?

Строение мозга

Как человек всего этого добивается?

- В мозге много нейронов
- Но цепочка нейронов, которые успевают поучаствовать в принятии решения, не может быть длиннее нескольких сот штук!
- Значит, мозг очень хорошо структурирован в этом смысле

Искусственные нейронные сети

- Основная мысль позаимствована у природы: есть связанные между собой нейроны, которые передают друг другу сигналы.
- Есть нейронные сети, которые стараются максимально точно моделировать головной мозг; это уже не AI и не наша тема.
- John Denker: «neural networks are the second best way of doing just about anything».

История ANN

- Warren McCulloch & Walter Pitts, 1943: идея.
- Marvin Minsky, 1951: первая реализация.
- 1960-е годы: долго изучали перцептрон, но ничего существенного одним перцептроном не сделать.
- 1980-е: появились многоуровневые ANN, была разработана современная теория.

Общая структура

- Есть сеть из нейронов, соединённых между собой.
- Нейроны возбуждаются под действием входов и передают возбуждение (либо как один бит, либо с каким-то значением) дальше.
- В результате последний нейрон на выход подаёт ответ.

Как построить один нейрон?

Общие требования к модели

Суть модели:

- Нейрон возбуждается, если выполнено некоторое условие на входах;
- Затем он передаёт свой импульс дальше.

Нейрон возбуждается под действием какой-то функции от входов. Такая конструкция называется *перцептроном*. Это простейшая модель нейрона в искусственной нейронной сети.

Линейный перцептрон

У линейного перцептрона заданы:

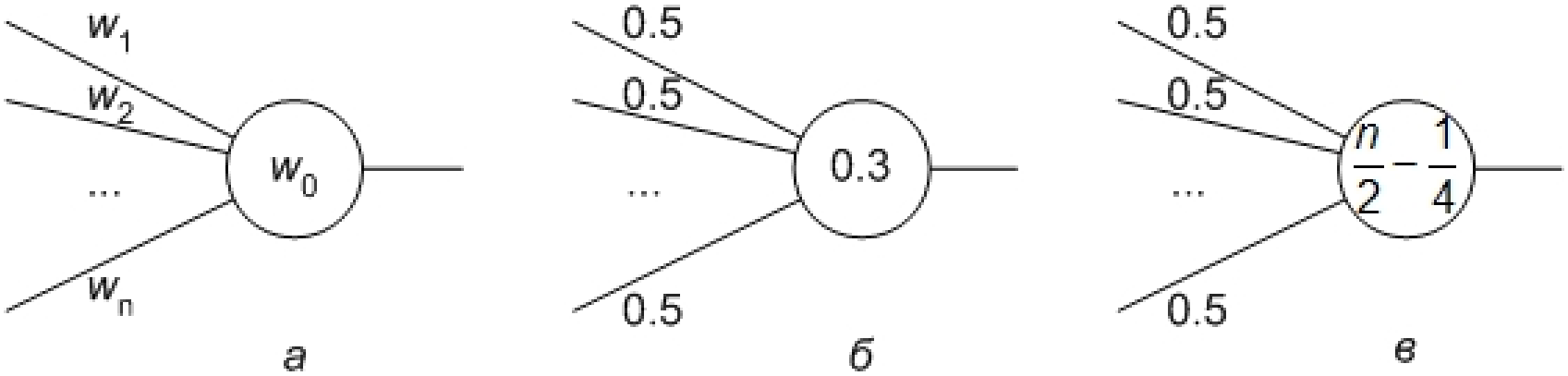
- n весов $w_1, w_2, \ldots, w_n$;
- лимит активации w_0;
- выход перцептрона $o(x_1, \ldots, x_n)$ вычисляется так:

$$o(x_1, \ldots, x_n) = \begin{cases} 1, \text{ если } w_0 + w_1 x_1 + \ldots + w_n x_n > 0, \\ -1 \text{ в противном случае.} \end{cases}$$

- или запишем иначе, введя переменную $x_0 = 1$:

$$o(x_1, \ldots, x_n) = \begin{cases} 1, \text{ если } \sum_i w_i x_i > 0, \\ -1 \text{ в противном случае.} \end{cases}$$

Примеры перцептронов



а *б* *в*

- *а* — общий вид перцептрона
- *б* — дизъюнкция
- *в* — конъюнкция

Сила перцептронов

- Один перцептрон может реализовать любую гиперплоскость, рассекающую пространство возможных решений. Иначе говоря, если прообразы 0 и 1 у целевой функции линейно отделимы, то *одного перцептрона достаточно*.
- Но он не может реализовать линейно неотделимое множество решений, например, XOR.
- А вот сеть из нескольких уровней перцептронов уже и XOR может.

Упражнение. Докажите, что любая булевская функция представима в виде построенной из перцептронов искусственной нейронной сети глубины 2.

Общие принципы

Как обучать перцептрон?

- Всё, что может у перцептрона меняться — это веса w_i, $i = 0..n$.
- Их мы и будем подправлять при обучении.
- Если перцептрон отработал правильно, веса не меняются.
- Если неправильно — сдвигаются в нужную сторону.

Perceptron training rule

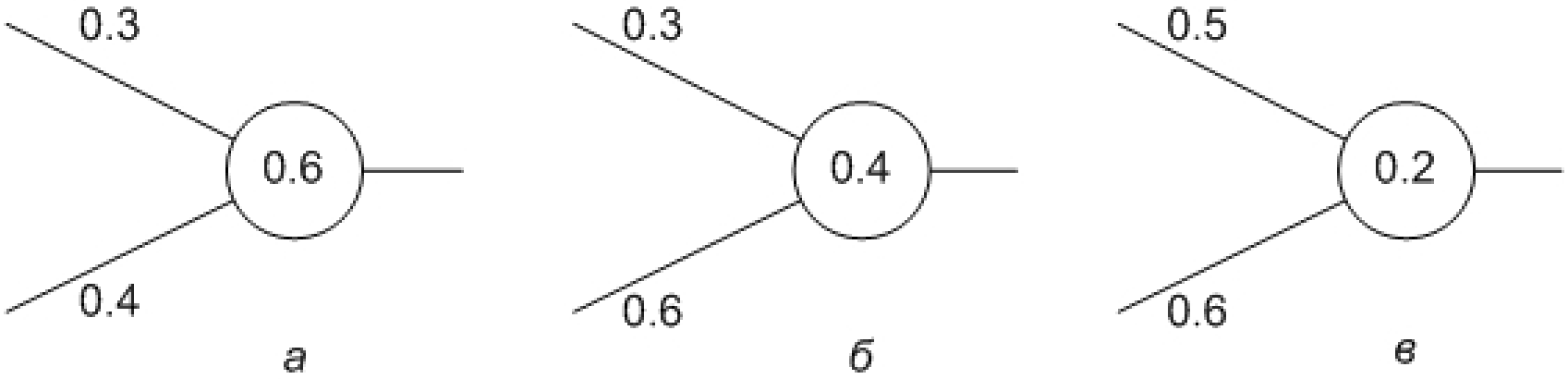
Простейшее правило:

$$w_i \leftarrow w_i + \eta(t - o)x_i,$$

где:

- t — значение целевой функции
- o — выход перцептрона
- $\eta > 0$ — небольшая константа (обычно 0.05–0.2), которая задаёт скорость обучения

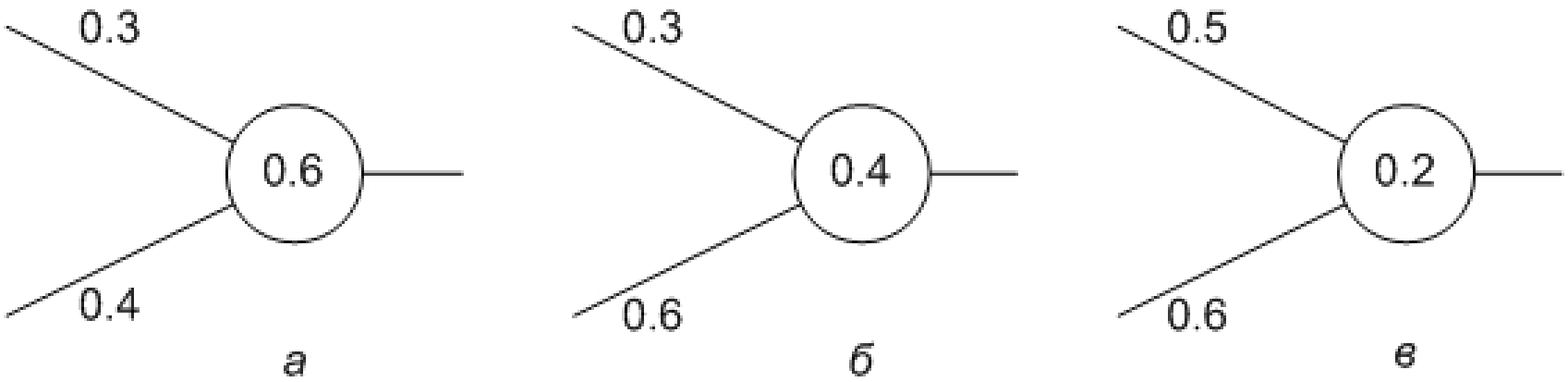
Пример обучения перцептрона



Мы хотим научить перцептрон распознавать дизъюнкцию.

- Рисунок *а* — перед началом обучения.
- Первый тест: $x_1 = 0$, $x_2 = 1 \Rightarrow t = 1$.
- Перцептрон тест не проходит.

Пример обучения перцептрона

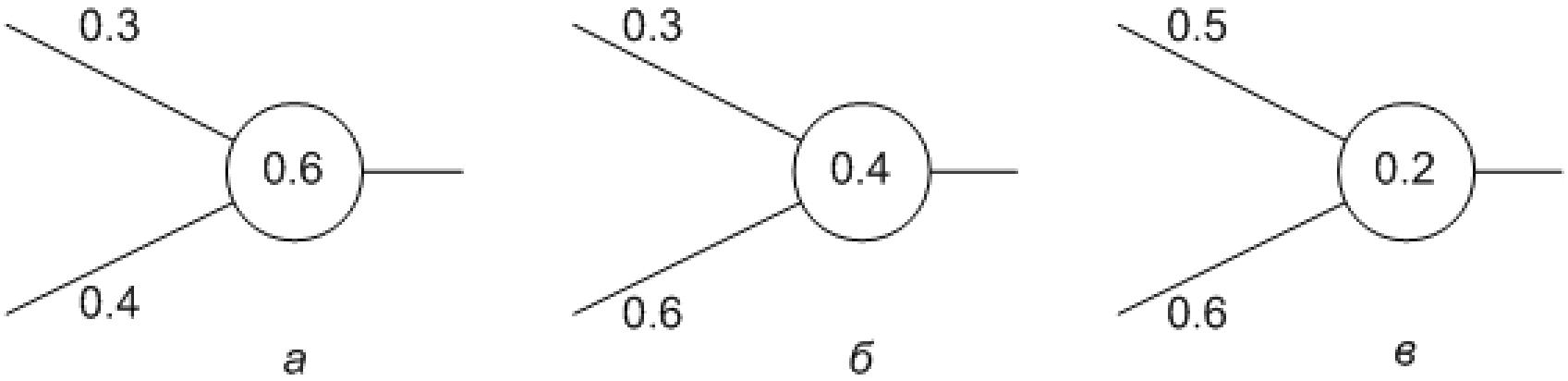


- Поправки на первом шаге:

$$
\begin{aligned}
w_0 \leftarrow w_0 + \eta(t-0)x_0 &= -0.6 + 0.1 \cdot (1-(-1)) \cdot 1 = -0.4,\\
w_1 \leftarrow w_1 + \eta(t-0)x_1 &= 0.3 + 0.1 \cdot (1-(-1)) \cdot 0 = 0.3,\\
w_2 \leftarrow w_2 + \eta(t-0)x_2 &= 0.4 + 0.1 \cdot (1-(-1)) \cdot 1 = 0.6.
\end{aligned}
$$

- После первого шага получаем перцептрон на рисунке *б*
- Второй тест: $x_1 = 1$, $x_2 = 0 \Rightarrow t = 1$.
- Перцептрон опять тест не проходит.

Пример обучения перцептрона

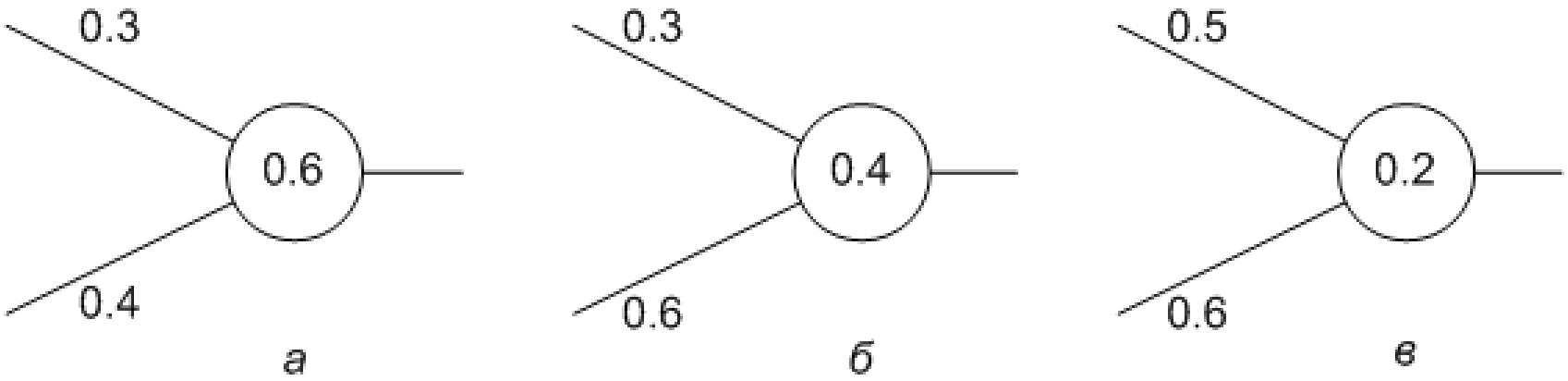


- Поправки на втором шаге:

$$
\begin{aligned}
w_0 &\leftarrow w_0 + \eta(t-0)x_0 = -0.4 + 0.1 \cdot (1-(-1)) \cdot 1 = -0.2,\\
w_1 &\leftarrow w_1 + \eta(t-0)x_1 = 0.3 + 0.1 \cdot (1-(-1)) \cdot 1 = 0.5,\\
w_2 &\leftarrow w_2 + \eta(t-0)x_2 = 0.6 + 0.1 \cdot (1-(-1)) \cdot 0 = 0.6.
\end{aligned}
$$

- Итого получается перцептрон с рисунка *в*. Он уже реализует дизъюнкцию правильно.

Пример обучения перцептрона



Если бы $\eta = 0.01$, веса были бы исправлены в нужную сторону, но недостаточно. Поэтому нужно запускать алгоритм по имеющимся тестовым примерам до тех пор, пока очередной прогон алгоритма по всем тестам не оставит все веса на месте.

Алгоритм обучения перцептрона

$\texttt{PerceptronTraining}(\eta, \{x_i^j, t^j\}_{i=1,j=1}^{n,m})$

- Инициализировать $\{w_i\}_{i=0}^n$ маленькими случайными значениями.
- $\texttt{WeightChanged} = \texttt{true}$.
- Пока $\texttt{WeightChanged} = \texttt{true}$:
  - $\texttt{WeightChanged} = \texttt{false}$.
  - Для всех j от 1 до m:
    - Вычислить

$$o^j = \begin{cases} 1, \text{ если } w_0 + w_1 x_1^j + \ldots + w_n x_n^j > 0, \\ -1 \text{ в противном случае.} \end{cases}$$

    - Если $o^j \neq t^j$:
      — $\texttt{WeightChanged} = \texttt{true}$.
      — Для каждого i от 0 до n изменить значение w_i по правилу

$$w_i \leftarrow w_i + \eta(t^j - o^j)x_i^j.$$

- Выдать значения $w_0, w_1, \ldots, w_n$.

Сходимость

Этот алгоритм сходится всегда, когда это возможно.

Теорема

Если конечное множество точек $C_1 \subset \{0,1\}^n$ можно в $\{0,1\}^n$ отделить гиперплоскостью от конечного множества точек $C_2 \subset \{0,1\}^n$, то алгоритм обучения перцептрона за конечное количество шагов выдаёт параметры перцептрона, который успешно разделяет множества C_1 и C_2.

Доказательство

- Входы принадлежат к отделимым гиперплоскостью множествам; это значит, что существует такой вектор u (нормаль), что

$$\begin{aligned} \forall x \in C_1 \quad & u^\top x > 0, \\ \forall x \in C_2 \quad & u^\top x < 0. \end{aligned}$$

- Цель обучения — сделать так, чтобы веса перцептрона w образовывали такой вектор u.

Доказательство

- Заменим C_2 на $-C_2 = \{-x \mid x \in C_2\}$. Это позволит нам объединить два неравенства в одно:

$$\forall x \in C \; w^\top x > 0, \quad C = C_1 \cup (-C_2).$$

- Итак, мы предполагаем, что входы принадлежат к отделимым гиперплоскостью множествам; это означает, что существует такой вектор u (нормаль к той самой гиперплоскости), что

$$\begin{aligned} \forall x \in C_1 \quad & u^\top x > 0, \\ \forall x \in C_2 \quad & u^\top x < 0. \end{aligned}$$

Цель обучения перцептрона — сделать так, чтобы веса перцептрона w образовывали такой вектор u.

Доказательство

- Геометрически: хотим построить вектор w, который образует острые углы со всеми тестовыми примерами x.
- Обучение: если вдруг обнаружится вектор, с которым w образует тупой угол, мы просто прибавим его к w (с константой η, конечно).
- Осталось понять, почему этот процесс закончится. Для этого рассмотрим вектор, который *действительно* образует с ними тупые углы (он существует), и будем доказывать, что последовательность длин проекций w на этот вектор не может быть бесконечной.

Доказательство

- $w^0, w^1, \ldots$ — веса перцептрона по мере обучения, $x^0, x^1, \ldots$ — векторы тестовых примеров.
- W.l.o.g. $w^0 = 0$, все тестовые примеры принадлежат C, и на всех тестовых примерах $(w^k)^\top x^k < 0$, т.е. перцептрон не справляется с тестом (если он справляется с тестом, то веса перцептрона никак не меняются, поэтому такие тесты можно просто исключить из последовательности).
- Тогда правило обучения выглядит как

$$w_i^{k+1} = w_i^k + \eta x_i^k,$$

и в наших предположениях

$$w^k = \eta \sum_{j=0}^{k-1} x^j.$$

Доказательство

- Идея: получить две серии противоположных оценок на длину векторов $\|w^i\|$ и показать, что они не могут обе иметь место до бесконечности.
- Это и будет означать, что любая конечная последовательность тестов из двух фиксированных линейно отделимых множеств рано или поздно закончится, т.е. все последующие тесты перцептрон будет проходить успешно.

Доказательство

- Рассмотрим решение u и обозначим через α минимальную проекцию любого из x^j на u:

$$\alpha = \min_j u^\top x^j$$

(поскольку разных тестов конечное число, $\alpha > 0$).

- Тогда

$$u^\top w^{k+1} = \eta u^\top \sum_{j=0}^{k} x^j \geq \eta \alpha k.$$

Доказательство

- $u^\top w^{k+1} = \eta u^\top \sum_{j=0}^{k} x^j \geq \eta \alpha k.$
- Вспомним неравенство Коши–Буняковского:

$$\|x\|^2 \|y\|^2 \geq (x \cdot y)^2.$$

- Применительно к нашей задаче:

$$\|u\|^2 \|w^{k+1}\|^2 \geq (\eta \alpha k)^2, \quad \|w^{k+1}\|^2 \geq \frac{(\eta \alpha k)^2}{\|u\|^2}.$$

Доказательство

- $\|u\|^2\|w^{k+1}\|^2 \geq (\eta\alpha k)^2, \quad \|w^{k+1}\|^2 \geq \frac{(\eta\alpha k)^2}{\|u\|^2}.$
- Т.е. на каждой итерации длина вектора w^k возрастает линейно.

Доказательство

- С другой стороны, $w^{k+1} = w^k + \eta x^k$. Поскольку $(w^k)^\top x^k < 0$,

$$\|w^{k+1}\|^2 = \|w^k\|^2 + 2\eta(w^k)^\top x^k + \eta^2\|x^k\|^2 \leq \|w^k\|^2 + \eta^2\|x^k\|^2.$$

- Следовательно, $\|w^{k+1}\|^2 - \|w^k\|^2 \leq \eta^2\|x^k\|^2$.
- Просуммируем по k:

$$\|w^{k+1}\|^2 \leq \eta^2 \sum_{j=0}^{k} \|x^k\|^2 \leq \eta^2 \beta k,$$

где $\beta = \max_j \|x^j\|^2$.

Доказательство

- Итак, получились две оценки:

$$\frac{(\eta\alpha)^2}{\|u\|^2}k^2 \leq \|w^{k+1}\|^2 \leq \eta^2\beta k.$$

- Рано или поздно с ростом k эти оценки войдут в противоречие друг с другом.
- Это и значит, что последовательность применения одних и тех же тестовых примеров не может быть бесконечной.

Outline

Перцептрон без лимита активации

Здесь мы будем рассматривать перцептроны, у которых нет лимита активации. Они просто выдают линейную форму от своих входов:

$$o(x_0, \ldots, x_n) = \sum_0^n w_i x_i.$$

Т.е. у такого перцептрона не два, а континуально много возможных значений.

В остальном это всё те же линейные перцептроны.

Функция ошибки

Мы хотим найти перцептрон, который минимизирует ошибку.
Пусть есть m тестовых примеров x_i^j с верными ответами t^j, $j = 1..m$.
Ошибка — из статистики, среднеквадратичное отклонение:

$$E(w_0, \ldots, w_n) = \frac{1}{2} \sum_{j=1}^{m} (t_j - o(x_0^j, \ldots, x_n^j))^2.$$

Задача: минимизировать функцию E на пространстве возможных весов $\{w_i\}$.

Градиент

Как минимизировать нелинейную функцию от нескольких аргументов?

Градиент

Как минимизировать нелинейную функцию от нескольких аргументов?

Нужно двигаться в сторону, обратную *градиенту*. Градиент — направление, в котором достигается наибольший прирост значений:

$$\nabla E(w_0, \ldots, w_n) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right].$$

Градиент

Как минимизировать нелинейную функцию от нескольких аргументов?
Нужно двигаться в сторону, обратную *градиенту*. Градиент — направление, в котором достигается наибольший прирост значений:

$$\nabla E(w_0, \ldots, w_n) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right].$$

Значит, веса нужно подправлять так:

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}.$$

Градиент от функции ошибки

В нашем случае подсчитать градиент совсем просто:

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\sum_{j=1}^{m}\frac{\partial}{\partial w_i}\left(t^j - \sum_{0}^{n} w_i x_i^j\right)^2 =$$

$$= \sum_{j=1}^{m}\left(t^j - \sum_{0}^{n} w_i x_i^j\right)(-x_i^j).$$

Изменения весов примут вид:

$$w_i \leftarrow w_i + \eta\sum_{j}\left(t^j - \sum_{0}^{n} w_i x_i^j\right)x_i^j.$$

Алгоритм градиентного спуска: мелочи

- Алгоритм — в каждой эпохе подправлять веса на значения градиента.
- Непонятно, когда останавливаться; мы будем останавливаться после фиксированного количества шагов.
- Если η маленькая, мы не успеем дойти до минимума; если η большая — будем через него «перескакивать». Что делать?

Алгоритм градиентного спуска: мелочи

- Алгоритм — в каждой эпохе подправлять веса на значения градиента.
- Непонятно, когда останавливаться; мы будем останавливаться после фиксированного количества шагов.
- Если η маленькая, мы не успеем дойти до минимума; если η большая — будем через него «перескакивать». Что делать?
- Решение: начать с достаточно большой η, а потом её постепенно уменьшать.

Алгоритм градиентного спуска

`GradientDescent`$(\eta, \{x_i^j, t^j\}_{i=1,j=1}^{n,m})$

- Инициализировать $\{w_i\}_{i=0}^n$ маленькими случайными значениями.
- Повторить `NUMBER_OF_STEPS` раз:
  - Для всех i от 1 до n $\Delta w_i = 0$.
  - Для всех j от 1 до m:
    - Для всех i от 1 до n

$$\Delta w_i = \Delta w_i + \eta \left(t^j - \sum_0^n w_i x_i^j \right) x_i^j.$$

  - $w_i = w_i + \Delta w_i$.
- Выдать значения $w_0, w_1, \ldots, w_n$.

Стохастический градиентный спуск

- Предыдущий алгоритм не мог преодолевать локальные минимумы.
  - Почему это пока не проблема?

Стохастический градиентный спуск

- Предыдущий алгоритм не мог преодолевать локальные минимумы.
  - Почему это пока не проблема?
  - Потому что у параболоида локальных минимумов не бывает. Но в более сложных случаях. . .

Стохастический градиентный спуск

- Предыдущий алгоритм не мог преодолевать локальные минимумы.
  - Почему это пока не проблема?
  - Потому что у параболоида локальных минимумов не бывает. Но в более сложных случаях...
- Модификация — изменять веса после каждого тестового примера, а не накапливать Δw_i.

Алгоритм стохастического градиентного спуска

$\texttt{GradientDescent}(\eta, \{x_i^j, t^j\}_{i=1,j=1}^{n,m})$

- Инициализировать $\{w_i\}_{i=0}^n$ маленькими случайными значениями.
- Повторить `NUMBER_OF_STEPS` раз:
  - Для всех j от 1 до m:
    - Для всех i от 1 до n

$$w_i = w_i + \eta \left(t^j - \sum_0^n w_i x_i^j \right) x_i^j.$$

- Выдать значения $w_0, w_1, \ldots, w_n$.

Главная проблема

- Какое главное ограничение у перцептронов без лимита активации?

Главная проблема

- Какое главное ограничение у перцептронов без лимита активации?
- Они реализуют только линейные функции.
- А какая проблема у перцептронов с лимитом активации?

Главная проблема

- Какое главное ограничение у перцептронов без лимита активации?
- Они реализуют только линейные функции.
- А какая проблема у перцептронов с лимитом активации?
- У них негладкий выход, и градиент от него не подсчитать. Соответственно, и алгоритм не сработает.
- Что же делать?

Главная проблема

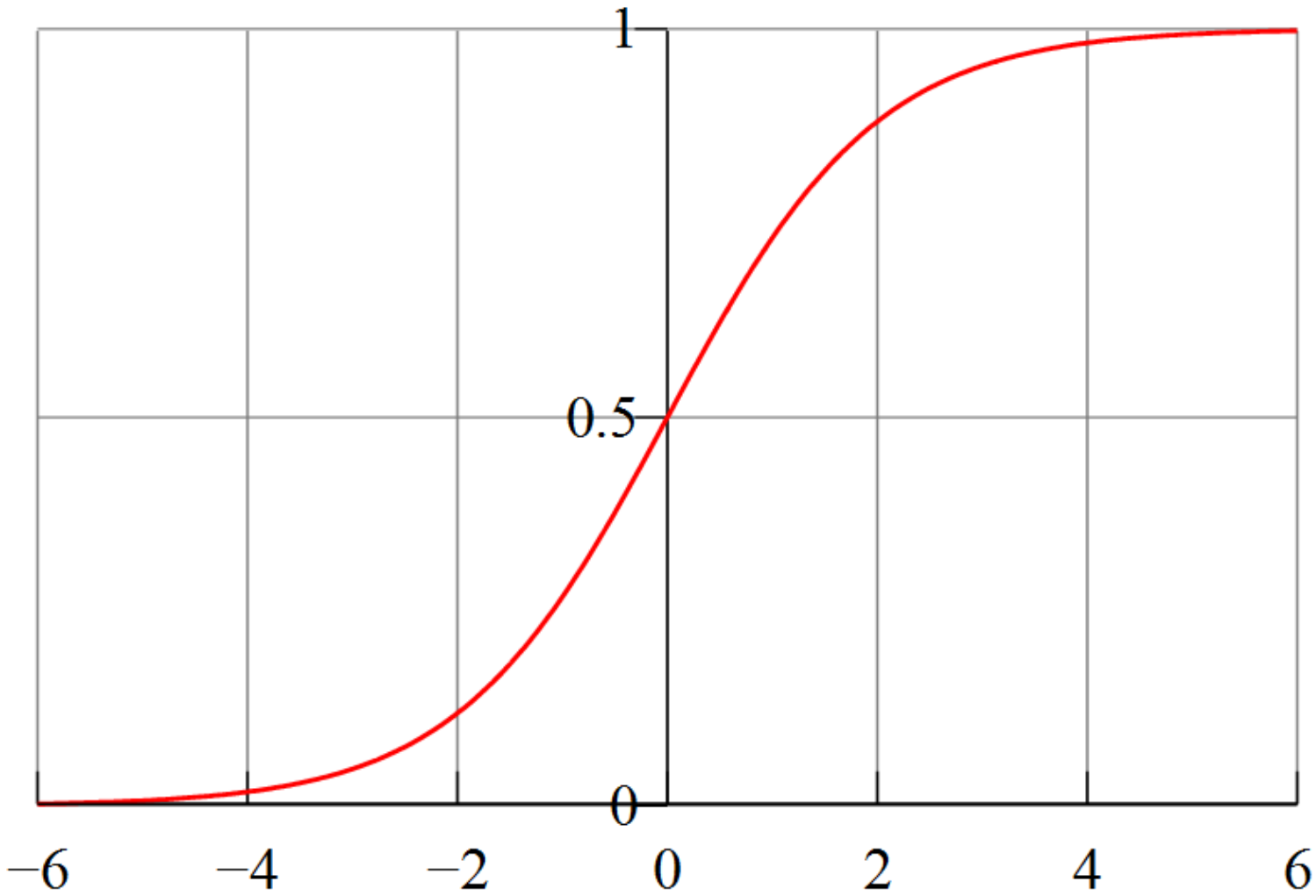
- Какое главное ограничение у перцептронов без лимита активации?
- Они реализуют только линейные функции.
- А какая проблема у перцептронов с лимитом активации?
- У них негладкий выход, и градиент от него не подсчитать. Соответственно, и алгоритм не сработает.
- Что же делать?
- Строить гладкие, но не линейные перцептроны.

Нелинейные перцептроны

- Всё то же самое, но функция выхода уже не линейная.
- Но от линейной формы всё же не отказываемся, а берём её результат и сглаживаем; получается *сигмоид*-функция (похожая на букву S).
- Популярный сигмоид — логистическая функция $\sigma(x) = \frac{1}{1+e^{-x}}$.
- То есть выход перцептрона подсчитывается по формуле:

$$o(x_1, \ldots, x_n) = \frac{1}{1 + e^{-\sum_i w_i x_i}}.$$

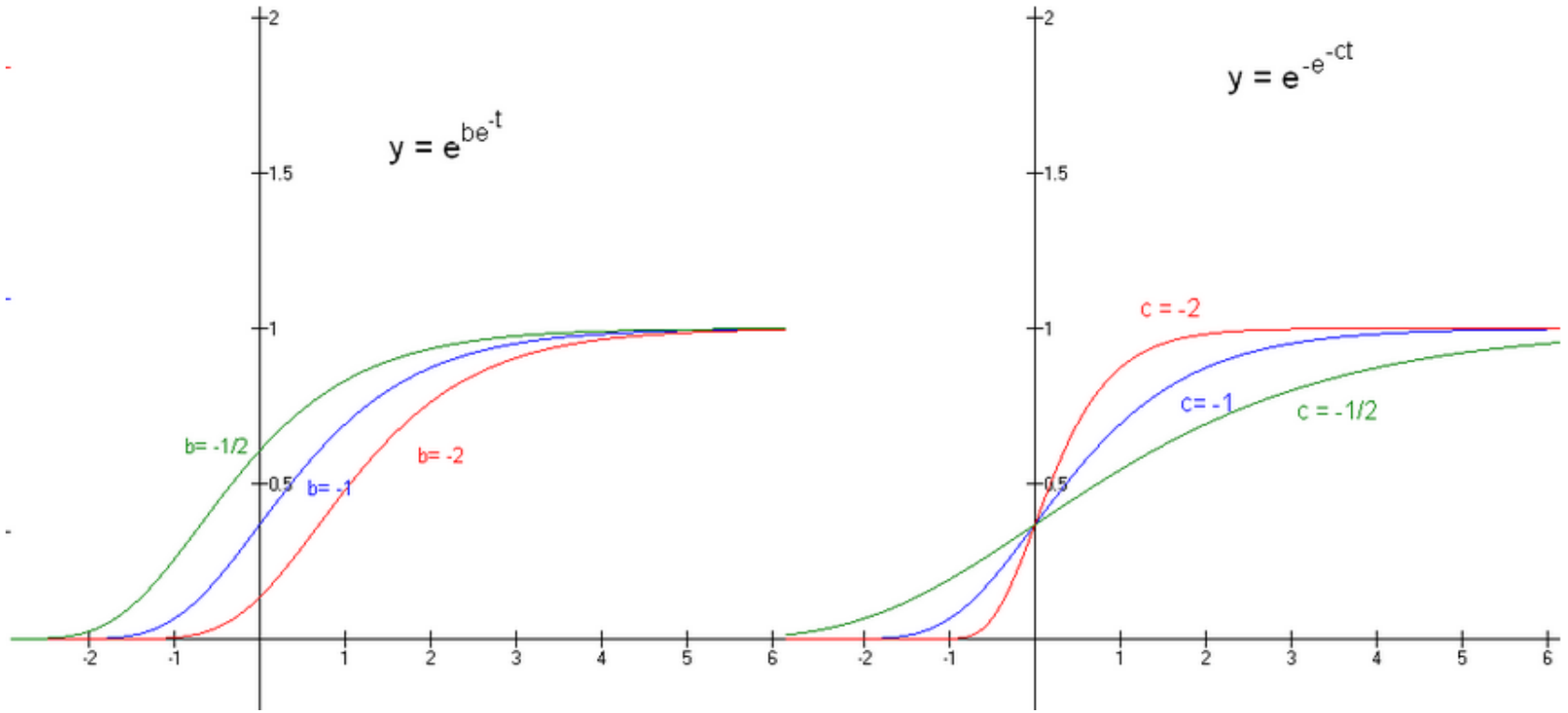
Логистическая функция



Функция Гомперца

- Другой популярный сигмоид — функция Гомперца

$$y(x) = ae^{be^{cx}}.$$



Градиент

- От логистической функции легко считать производную:

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

- Соответственно, правило модификации веса для одного перцептрона выглядит как

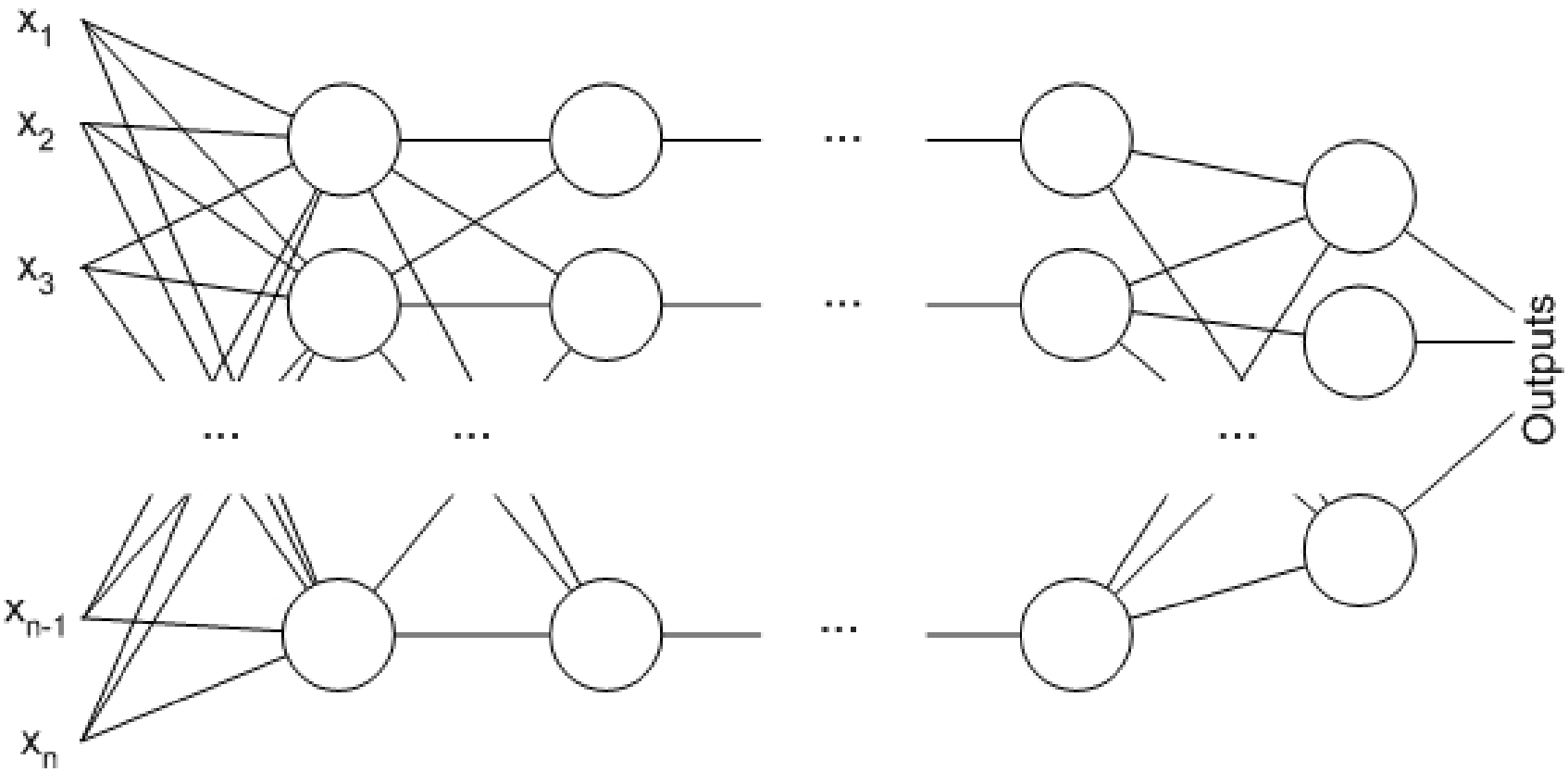
$$w_i \leftarrow w_i + \eta o(x)(1 - o(x))(t(x) - o(x))x_i,$$

где $t(x)$ — целевое значение функции.

Outline

Нейронная сеть

Теперь у нас не один перцептрон, а целая их сеть.



Обозначения

- У сети есть входы $x_1, \ldots, x_n$, выходы Outputs и внутренние узлы.
- Перенумеруем все узлы (включая входы и выходы) числами от 1 до N.
- w_{ij} — вес, стоящий на ребре (i, j).
- o_i — выход i-го узла.
- Даны m тестовых примеров $\{x_1^d, \ldots, x_n^d)\}_{d=1..m}$ с целевыми значениями выходов $\{t_k^d\}_{,d=1..m,k\in\text{Outputs}}$.

Функция ошибки:

$$E(\{w_{ij}\}) = \frac{1}{2}\sum_{d=1}^{m}\sum_{k\in\text{Outputs}}\left(t_k^d - o_k(x_1^d, \ldots, x_n^d)\right)^2.$$

Градиент

Как подсчитать градиент функции ошибки для каждого тестового примера?

$$E^d(\{w_{ij}\}) = \frac{1}{2} \sum_{k \in \text{Outputs}} \left(t_k^d - o_k^d\right)^2.$$

Случай 1. Считаем $\Delta w_{ij} = -\eta \frac{\partial E^d}{\partial w_{ij}}$, где $j \in \text{Outputs}$.

- w_{ij} влияет на выход o^d только как часть суммы $S_j = \sum_i w_{ij} x_{ij}$, поэтому

$$\frac{\partial E^d}{\partial w_{ij}} = \frac{\partial E^d}{\partial S_j} \frac{\partial S_j}{\partial w_{ij}} = x_{ij} \frac{\partial E^d}{\partial S_j}.$$

- S_j влияет на общую ошибку только в рамках выхода j-го узла o_j (это выход всей сети). Поэтому...

Градиент

Как подсчитать градиент функции ошибки для каждого тестового примера?

$$E^d(\{w_{ij}\}) = \frac{1}{2} \sum_{k \in \text{Outputs}} \left(t_k^d - o_k^d\right)^2.$$

$$\frac{\partial E^d}{\partial S_j} = \frac{\partial E^d}{\partial o_j} \frac{\partial o_j}{\partial S_j} = \left(\frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{Outputs}} (t_k - o_k)^2\right) \left(\frac{\partial \sigma(S_j)}{\partial S_j}\right) =$$

$$= \left(\frac{1}{2} \frac{\partial}{\partial o_j} (t_j - o_j)^2\right) (o_j(1 - o_j)) = -o_j(1 - o_j)(t_j - o_j).$$

Градиент

Как подсчитать градиент функции ошибки для каждого тестового примера?

$$E^d(\{w_{ij}\}) = \frac{1}{2} \sum_{k \in \text{Outputs}} \left(t_k^d - o_k^d\right)^2.$$

Случай 2. Считаем $\Delta w_{ij} = -\eta \frac{\partial E^d}{\partial w_{ij}}$, где $j \notin \text{Outputs}$.

- У узла j есть потомки $\text{Children}(j)$. Тогда

$$\frac{\partial E^d}{\partial S_j} = \sum_{k \in \text{Children}(j)} \frac{\partial E^d}{\partial S_k} \frac{\partial S_k}{\partial S_j}, \text{ и } \frac{\partial S_k}{\partial S_j} = \frac{\partial S_k}{\partial o_j} \frac{\partial o_j}{\partial S_j} = w_{jk} o_j (1 - o_j).$$

- $\frac{\partial E^d}{\partial S_k}$ — это в точности аналогичная поправка, но вычисленная для узла следующего уровня. Так мы и дойдём от выходов ко входам.

Резюме подсчёта градиента

- Для узла последнего уровня

$$\delta_j = -o_j(1 - o_j)(t_j - o_j).$$

- Для внутреннего узла сети

$$\delta_j = -o_j(1 - o_j) \sum_{\text{Outputs}(j)} \delta_k w_{jk}.$$

- Для любого узла

$$\Delta w_{ij} = -\eta \delta_j x_{ij}.$$

Алгоритм обратного распространения ошибки

$\texttt{BackPropagation}(\eta, \{x_i^d, t^d\}_{i=1,d=1}^{n,m}, \texttt{NUMBER\_OF\_STEPS})$

- Инициализировать $\{w_{ij}\}_{i,j}$ случайными значениями.
- Повторить NUMBER_OF_STEPS раз:
  - Для всех d от 1 до m:
    - Подать $\{x_i^d\}$ на вход и подсчитать выходы o_i каждого узла.
    - Для всех $k \in \text{Outputs}$ $\delta_k = o_k(1-o_k)(t_k - o_k)$.
    - Для каждого уровня l, начиная с предпоследнего:
      — Для каждого узла j уровня l вычислить

$$\delta_j = o_j(1-o_j) \sum_{k \in \text{Children}(j)} w_{jk}\delta_k.$$

  - Для каждого ребра сети $\{ij\}$

$$w_{ij} = w_{ij} + \eta \delta_j x_{ij}.$$

- Выдать значения w_{ij}.

Резюме

Чему мы научились:

- Понятие перцептрона: с лимитом активации и без, линейного и нелинейного.
- Как обучать перцептроны поодиночке, в том числе:
  - алгоритмом обучения перцептрона,
  - методом градиентного спуска.
- Как обучать сеть из перцептронов с гладким выходом алгоритмом обратного распространения ошибки.

Нейронные сети

- В нейронных сетях мы минимизировали среднеквадратичную ошибку. Почему?
- Как мы сейчас увидим, именно её нужно минимизировать для того, чтобы получить MAP — в определённых предположениях, разумеется.

Предположения

- Мы хотим выучить функцию $f : \mathcal{X} \longrightarrow \mathbb{R}$.
- Есть набор тестовых примеров

$$D = \{\langle x_1, t_1 \rangle, \ldots, \langle x_m, t_m \rangle\}.$$

- Мы предполагаем, что $t_i = f(x_i) + e_i$, где e_i — *равномерно распределённый* шум с нулевым средним. Это и есть основное предположение.
- Кроме того, предполагаем независимость тестовых примеров.

Вывод

- Мы ищем $h_{MAP} = \text{argmax}_H p(D|h) = \text{argmax}_H \prod_{i=1}^{m} p(t_i|h)$.
- $p(t_i|h)$ — нормальное распределение с вариацией σ и с центром в $\mu = f(x_i) = h(x_i)$ (т.к. при условии h).

Вывод

- Мы ищем $h_{MAP} = \text{argmax}_H p(D|h) = \text{argmax}_H \prod_{i=1}^{m} p(t_i|h)$.
-

$$h_{MAP} = \text{argmax}_H \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2} =$$

$$= \text{argmax}_H \sum_{i=1}^{m} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2 \right) =$$

$$= \text{argmin}_H \sum_{i=1}^{m} (d_i - h(x_i))^2.$$

Вывод

- Мы ищем $h_{MAP} = \operatorname{argmax}_H p(D|h) = \operatorname{argmax}_H \prod_{i=1}^{m} p(t_i|h)$.
- Вот и доказали, что минимизация среднеквадратичной ошибки ведёт к MAP. Это фактически оправдывает применение нейронных сетей.

Функция ошибки

- Начнём с одного нейрона. Пусть у него n входов $\mathbf{x} = \{x_1, \ldots, x_n\}$, веса $\mathbf{w} = \{w_0, \ldots, w_n\}$, и он вычисляет функцию $o(\mathbf{x}, \mathbf{w}) \in [0, 1]$.
- Ему дают данные $\{t^j = f(\mathbf{x}^j)\}_{j=1}^m$, причём $t^j \in \{0, 1\}$. Какова функция ошибки?
- Мы уже даже доказывали, что в случае данных с нормально распределённым шумом функцией ошибки должно быть среднеквадратичное отклонение.
- А какая должна быть функция ошибки с точки зрения теории информации?

Относительная энтропия

- Kullback–Leibler distance (divergence) — это информационно-теоретическая мера того, насколько далеки распределения друг от друга.

$$D_{KL}(p, q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = H(p) - H(p, q).$$

- Его ещё называют *относительной энтропией*.
- Известно, что это расстояние всегда неотрицательно, равно нулю iff $p \equiv q$.

Функция ошибки и информация

- Информационное содержание одного запуска — это относительная энтропия $H(p, q)$ ($H(p)$ минимизировать нельзя, поэтому его опустим) между эмпирическим распределением $(t^j, 1 - t^j)$ и распределением выхода нейрона $(o, 1 - o)$:

$$G^j(\mathbf{w}) = -t^j \ln o(\mathbf{x}^j, \mathbf{w}) - (1 - t^j) \ln \left(1 - o(\mathbf{x}^j, \mathbf{w})\right).$$

- То есть мы рассматриваем данные t и выход перцептрона $o(\mathbf{x}, \mathbf{w}) \in [0, 1]$ — как распределение вероятностей (с какой вероятностью ответ равен 1).
- Общая ошибка, естественно, получается как сумма:

$$G(\mathbf{w}) = -\sum_{j=1}^{m} \left[t^j \ln o(\mathbf{x}^j, \mathbf{w}) + (1 - t^j) \ln \left(1 - o(\mathbf{x}^j, \mathbf{w})\right)\right].$$

Минимизация относительной энтропии

- Алгоритм от этого не меняется, кстати. Если $o(\mathbf{x}, \mathbf{w}) = \frac{1}{1+e^{-\mathbf{xw}}}$, то, продифференцировав $G(\mathbf{w})$ по w_i, получим
$$\frac{\partial G}{\partial w_i} = -\sum_j (t^j - o^j) x_i^j.$$
- То есть по-прежнему градиентный спуск ходит по направлению, обратному ошибке.
- Однако относительная энтропия позволит нам потом разработать другие методы.

Оверфиттинг

- А пока поборемся с оверфиттингом. Оверфиттинг — это когда модель *слишком* хорошо отвечает данным.
- В случае нейронных сетей такое бывает, когда данным соответствуют всё лучше и лучше, но веса при этом растут до бесконечности.
- Как этого избежать? Можно, конечно, искусственно останавливаться раньше, но это слишком уж ad hoc.

Регуляризация

- Нужно внести в функцию ошибки поправку, которая будет убирать нежелательные эффекты (такой процесс называется *регуляризацией*).
- Чтобы убрать нейронный оверфиттинг, будем рассматривать функцию ошибки

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}),$$

где α — константа (*гиперпараметр*), а функция E — *регуляризатор*

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} w_i^2.$$

- Тогда

Упражнение. Как изменится алгоритм градиентного спуска?

Суть проблемы

- Во-первых, скорость обучения должна меняться со временем, становиться меньше по мере приближения к минимуму и быть достаточно большой далеко от него.
- Во-вторых, неплохо бы всё-таки научиться вылезать хотя бы из самых мелких локальных минимумов на пути к глобальному.

Момент

- Одно из решений — ввести в правило пересчёта весов так называемый *момент* (momentum). Пересчёт на шаге t имеет вид:

$$\Delta w_{ij}(t) = \delta + m\Delta w_{ij}(t-1),$$

  где δ — изменение веса по обычному алгоритму (какому угодно).
- То есть по мере продвижения к минимуму нейронная сеть как бы «разгоняется», одновременно и быстрее к нему приближаясь, и, возможно, перепрыгивая мелкие минимумы и ущелья целевой функции по ходу дела.

Адаптация скорости обучения

- Нужно менять скорость обучения, чтобы быстрее придти к минимуму и там остаться.
- Для этого есть разные методы; мы рассмотрим три из них.

Bold driver

- Модель «бесстрашного водителя» — увеличивать скорость, пока не врежешься во что-нибудь, а потом уменьшать.
- Пока ошибка уменьшается, скорость обучения η ненамного увеличивается (на 5–10%).
- Как только ошибка при очередном шаге увеличилась, скорость обучения тут же резко падает (на $\approx 50\%$).

Уменьшение скорости обучения

- Если проблема не в том, чтобы побыстрее дойти до минимума, а в том, чтобы там остаться, можно просто уменьшать скорость обучения:

$$\eta(t) = \frac{\eta(0)}{1 + \frac{t}{T}},$$

где T — новый параметр.

- В этой модели скорость падает небыстро во время первых T шагов, когда мы достигаем примерно минимума, но при этом уменьшается достаточно быстро, чтобы гарантировать сходимость.

Градиентный спуск для скорости обучения

- Более изощрённый метод — подправлять η при помощи того же алгоритма обучения (градиентного спуска). Пусть веса обновляются как

$$w_{ij}(t+1) = w_{ij}(t) + \eta_{ij}(t)\Delta w_{ij}(t).$$

- Мы хотим изменить $\eta_{ij}(t)$ так, чтобы уменьшить $E(t+1)$ (функцию ошибки):

$$\frac{\partial E(t+1)}{\partial \eta_{ij}(t)} = \frac{\partial E(t+1)}{\partial w_{ij}(t+1)} \frac{\partial w_{ij}(t+1)}{\partial \eta_{ij}(t)} = -\Delta w_{ij}(t)\Delta w_{ij}(t-1).$$

Градиентный спуск для скорости обучения

- То есть можно подправлять η_{ij} по правилу:

$$\eta_{ij}(t) = \eta_{ij}(t-1) + \xi \Delta w_{ij}(t) \Delta w_{ij}(t-1).$$

- Правда, с таким подходом η_{ij} может стать отрицательной. Кроме того, нам надо менять η_{ij} на несколько порядков. Поэтому применим то же самое не к η, а к её логарифму:

$$\log(\eta_{ij}(t)) = \log(\eta_{ij}(t-1)) + \xi \Delta w_{ij}(t) \Delta w_{ij}(t-1).$$

Проблема

- При этом алгоритме значения η_{ij} будут сильно прыгать.
- Поэтому часто заменяют $\Delta w_{ij}(t-1)$ на среднее нескольких последних значений Δw_{ij}, обычно экспоненциальное среднее, которое вычисляется как

$$\bar{u}(t) = m\bar{u}(t-1) + (1-m)u(t),$$

где m — новый параметр.

Thank you!

Спасибо за внимание!