

# Машинное обучение в экономике

## Нейронные сети

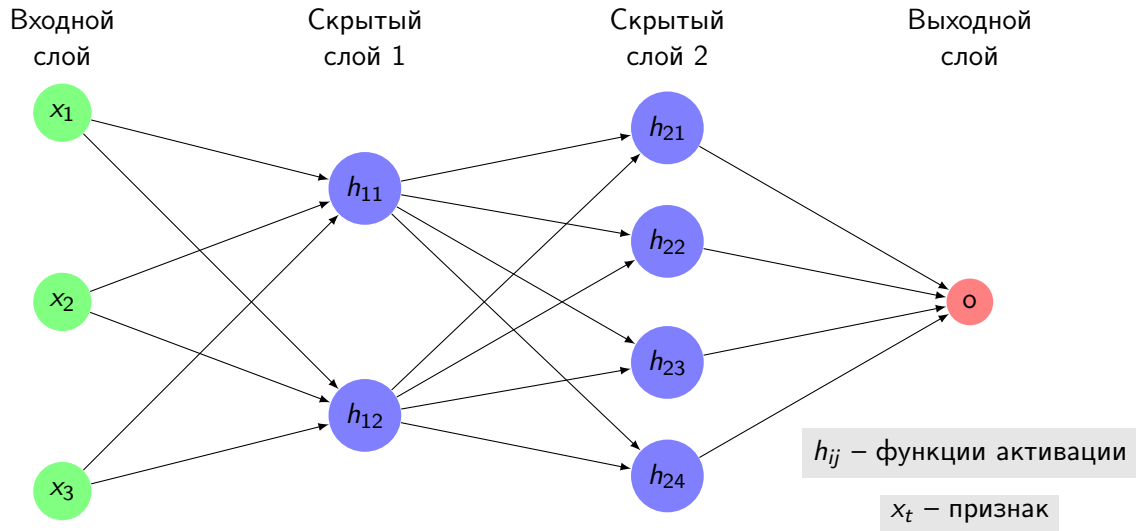
Потанин Богдан Станиславович

доцент, научный сотрудник, кандидат экономических наук

2023–2024

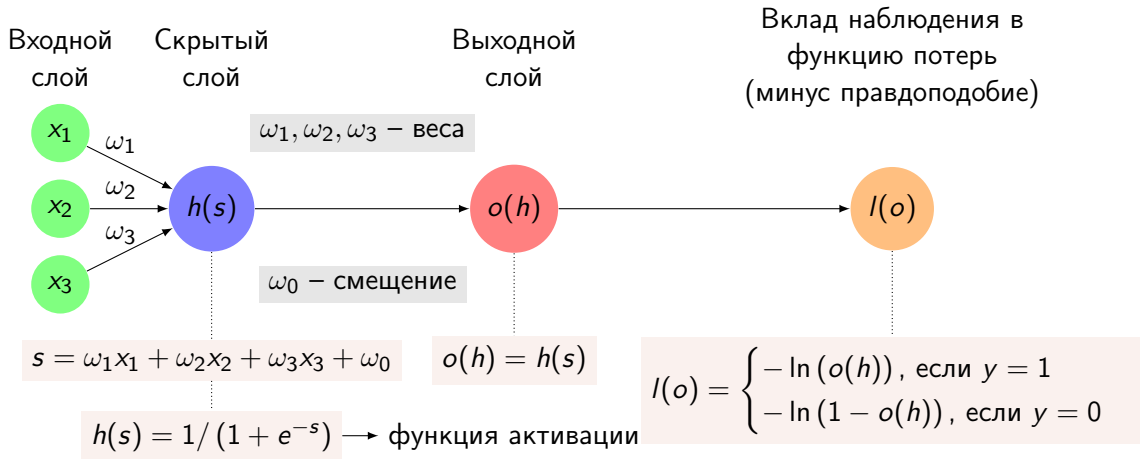
# Нейронная сеть

Графическая репрезентация идеи



# Нейронная сеть

## Логистическая регрессия как нейронная сеть



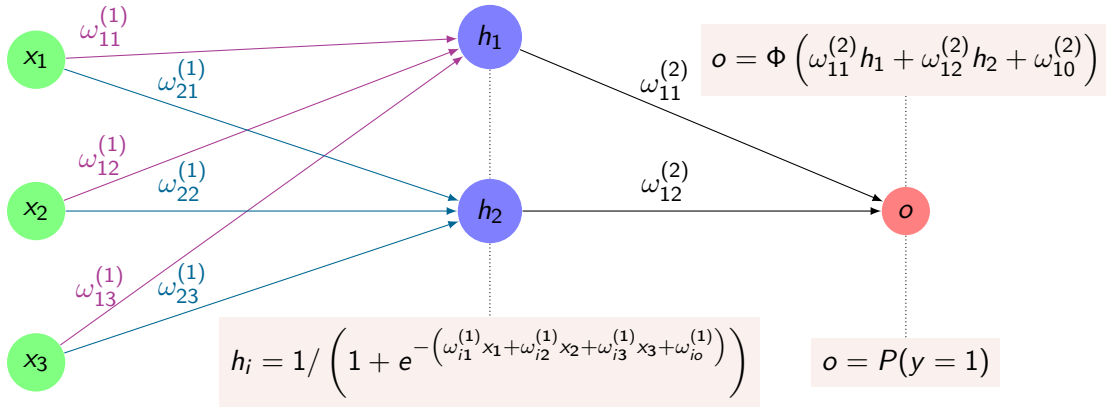
# Нейронная сеть

Пример с двумя функциями активации

Входной слой

Скрытый слой

Выходной слой



- Определяется функция потерь. Например, в задаче классификации в качестве функции потерь может выступать умноженная на минус единицу функция правдоподобия. При работе с непрерывными переменными можно рассмотреть сумму квадратов отклонений предсказанных значений от истинных.
- Сперва представим, что веса зафиксированы. В таком случае для получения значения функции потерь достаточно каждое наблюдение провести через нейросеть (от входного слоя до выходного слоя) и воспользоваться полученными значениями (из выходного слоя) для расчета функции потерь. В задаче классификации эти значения, как правило, представляют собой вероятности, а при работе с непрерывными переменными – предсказанные значения целевой переменной.
- Можно воспользоваться любым удобным алгоритмом численной оптимизации, для того чтобы найти веса  $\omega_{(ij)}^k$  и смещения  $\omega_{(i0)}^k$  нейросети, минимизирующие функцию потерь.
- Число слоев и нейронов, а также функции активации являются гиперпараметрами нейросети.

# Функции активации

## Общая идея

- Как правило во всех скрытых слоях применяется одна и та же функция активации. Однако, можно использовать разные функции активации даже в рамках одного слоя, а также включать дополнительные оцениваемые параметры этих функций активации.
- Некоторые популярные функции активации:

Сигмоида (логистическая):  $1 / (1 + e^{-s})$

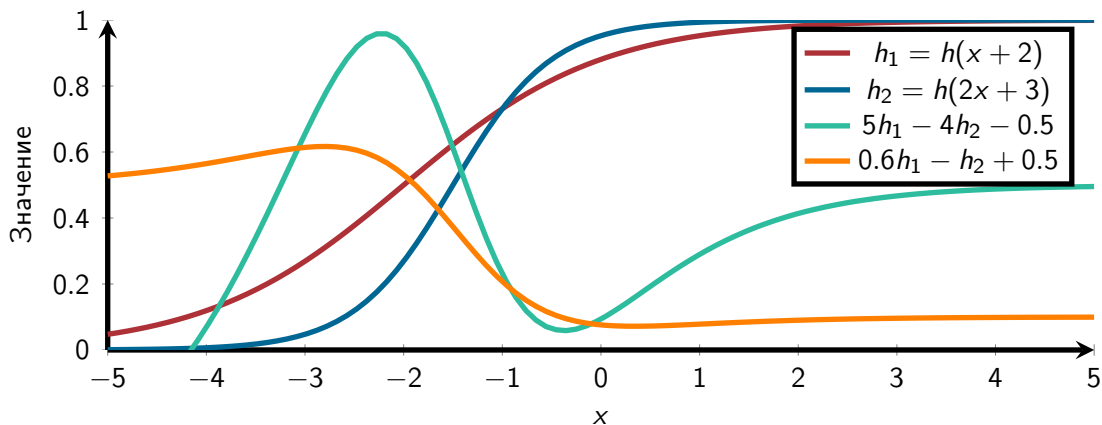
ReLU:  $\max(0, s) = 0.5(s + |s|)$

ELU: 
$$\begin{cases} \alpha(e^s - 1), & \text{если } s \leq 0 \\ s, & \text{если } s > 0 \end{cases}$$

- Форма каждой функции активации варьируется достаточно слабо, однако линейные комбинации даже одних и тех же функций могут принимать самые разнообразные формы, благодаря чему и достигается гибкость нейросети.

# Функции активации

Графический пример с одним признаком и линейной комбинацией логистических функций активации



**Вывод** –  $h_1$  и  $h_2$  похожи, но их линейные комбинации могут принимать разнообразные формы, что позволяет аппроксимировать очень сложные зависимости.

# Алгоритм обратного распространения ошибки (backpropagation)

## Основная идея

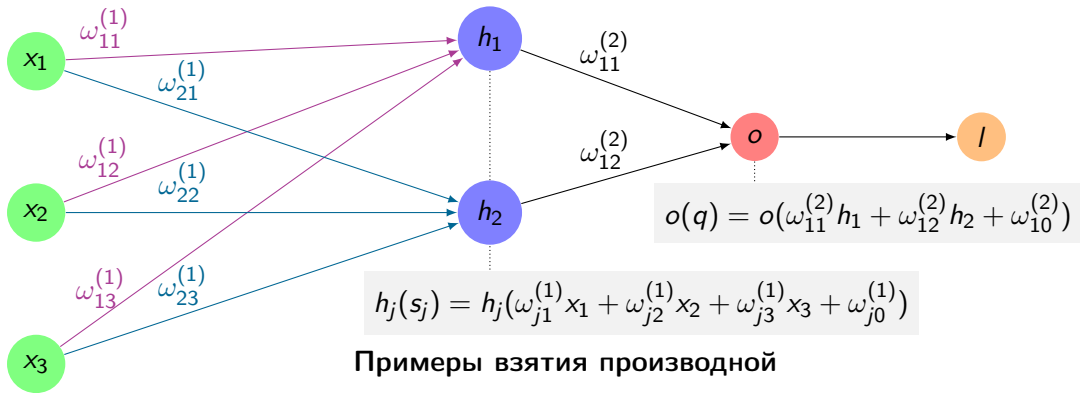
- Для повышения скорости оптимизации нейронных сетей обычно используются методы численной оптимизации, опирающиеся на информацию о градиенте, такие как, например, градиентный спуск и BFGS.
- Численный расчет градиента может привести к существенным потерям в скорости расчетов, поэтому, в качестве альтернативы часто рассматривается аналитический градиент.
- Для расчета градиента по параметрам нейросети (весам и смещению) часто применяется **алгоритм обратного распространения ошибки**. Этот алгоритм сводится к обычному применению дифференцирования по цепочке (chain rule) и его идея легко иллюстрируется графически.
- **Преимущество** - для того, чтобы запрограммировать производную функции потерь по параметру нейросети, достаточно запрограммировать частные производные функций активации, выходного слоя и функции потерь, а затем, с помощью правила цепочки, перемножить эти производные.



# Алгоритм обратного распространения ошибки (backpropagation)

Графическая иллюстрация

Входной слой      Скрытый слой      Выходной слой      Функция потерь



Примеры взятия производной

$$\frac{\partial l}{\partial \omega_{13}^{(1)}} = \frac{\partial l}{\partial o} \frac{\partial o}{\partial q} \frac{\partial q}{\partial h_1} \frac{\partial h_1}{\partial s_1} \frac{\partial s_1}{\partial \omega_{13}^{(1)}} = \frac{\partial l}{\partial o} \frac{\partial o}{\partial q} \omega_{11}^{(2)} \frac{\partial h_1}{\partial s_1} x_3$$

$$\frac{\partial l}{\partial \omega_{12}^{(2)}} = \frac{\partial l}{\partial o} \frac{\partial o}{\partial q} \frac{\partial q}{\partial \omega_{12}^{(2)}} = \frac{\partial l}{\partial o} \frac{\partial o}{\partial q} h_2$$

# Модификации градиентного спуска

## Стохастический градиентный спуск

- **Проблема** – обычно нейросети обучаются на очень большом числе наблюдений, поэтому численная оптимизация, например, с помощью градиентного спуска, оказывается очень времязатратной.
- **Решение** – воспользоваться **стохастическим градиентным спуском**.
- Обратим внимание, что во многих оптимизационных задачах функция потерь может быть представлена в аддитивном по наблюдениям виде:

$$L(Y, F(X; \theta)) = \sum_{i=1}^n L(Y_i, F(X_i; \theta))$$

- В частности, в нейросети параметры  $\theta$  будут отражать веса  $w$  (включая смещение - константу), а функция  $F(X_i, \theta)$  – выходное (output) значение для  $i$ -го наблюдения.
- Стохастический градиентный спуск (SGD) отличается от обычного градиентно спуска (GD) лишь тем, что значение параметра на  $k$ -й итерации алгоритма  $\theta^{(k)}$  рассчитывается не по всем наблюдениям, а по каждому наблюдению поорчедено:

$$\text{SGD: } \theta^{(k+1)} = \theta^{(k)} - \alpha \nabla L(Y_i, F(X_i; \theta^{(k)})) \quad \text{вместо} \quad \text{GD: } \theta^{(k+1)} = \theta^{(k)} - \alpha \nabla L(Y, F(X; \theta^{(k)}))$$

- После того, как были перебраны все наблюдения, они сортируются в случайном порядке и алгоритм повторяется, что позволяет избежать заикливания.

# Модификации градиентного спуска

## Мини-пакетный градиентный спуск

- **Проблема** – стохастический градиентный спуск аппроксимирует градиент всей функции с помощью одного наблюдения, что может приводить к большим погрешностям и, как следствие, к тому, что на некоторых итерациях функция будет не уменьшаться, а увеличиваться.
- **Решение** – повысить точность аппроксимации наблюдений за счет использования не одного, а группы наблюдений.
- Выборка делится на  $m$  выборок приблизительно равного размера. Эти подвыборки именуется **мини-пакетами** (mini-batch).
- Например, выборка из 1000 наблюдений может быть разделена на  $m = 10$  мини-пакетов по 100 наблюдений в каждом.
- **Мини-пакетный градиентный спуск** (MBGD) рассчитывает значение параметра на  $k$ -й итерации алгоритма  $\theta^{(k)}$  по выборке из мини-пакетов:

$$\theta^{(k+1)} = \theta^{(k)} - \alpha \nabla L(Y^{(b)}, F(X^{(b)}; \theta^{(k)}))$$

Где  $Y^{(b)}$  и  $X^{(b)}$  отражают мини-пакет  $b \in \{1, \dots, m\}$ .

- После того, как были использованы все мини-пакеты, перед продолжением алгоритма они сортируются в случайном порядке во избежание заикливания.

- **Проблема** - поскольку нейронные сети очень хорошо аппроксимируют данные, то они склонны к переобучению.
- **Решение** - воспользоваться регуляризацией.
- **Штрафы** - можно накладывать штрафы на большие значения параметров нейросети, например, с помощью лассо или ридж регуляризации. В частности, при лассо регуляризации наименее важные веса могут обнуляться.
- **Ранняя остановка** - вместо того, чтобы искать минимум функции потерь, можно прекратить алгоритм численной оптимизации до нахождения точного решения, что снижает степень подгонки под данные и благодаря этому иногда воспрепятствует переобучению.
- **Зашумление** - можно добавить некоторую случайную компоненту в данные или в градиент.

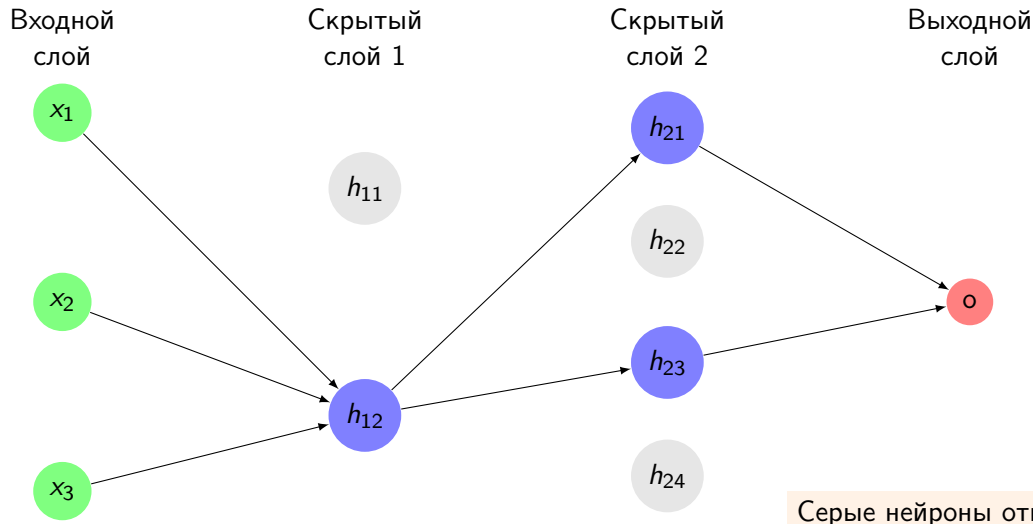
# Исключение (dropout / dilution)

## Основная идея

- **Исключение** (dropout / dilution) - модификация методов численной оптимизации параметров нейросети, при которой на каждой итерации алгоритма случайным образом исключаются некоторые из нейронов.
- Обычно эта техника используется вместе с мини-пакетным градиентным спуском, где значение и градиент функции потерь для каждого мини-пакета рассчитывается не по исходной, а по **утонченной нейросети**.
- **Утонченная нейросеть** (thinned network) обладает теми же параметрами (весами), что и исходная. Однако, каждый нейрон исходной нейросети входит в утонченную сеть лишь с некоторой вероятностью  $p$ , которая в самом простом случае одинакова для всех нейронов.
- Исключение нейрона равносильно **временному** обнулению входящих и исходящих из него весов.
- Каждый раз, в том числе при повторном использовании мини-пакета, для него генерируется новая утонченная нейросеть, с использованием которой рассчитывается градиент функции потерь для очередного шага мини-пакетного градиентного спуска.
- Для прогнозирования обычно используется исходная нейросеть с весами, обученными мини-пакетным градиентным спуском с исключением и домноженными на вероятность  $p$ .

# Исключение (dropout / dilution)

Утонченная нейронная сеть



# Исключение (dropout / dilution)

## Алгоритм

- Рассмотрим  $k$ -ю итерацию типичного алгоритма обучения параметров нейросети с помощью мини-пакетного градиентного спуска с исключением.
- **Шаг 1.** Генерируется утонченная нейросеть за счет того, что каждый из нейронов исключается с вероятностью  $1 - p$ .
- **Шаг 2.** Обновляются значения параметров нейросети  $w$  с использованием мини-пакета  $(X^{(b)}, Y^{(b)})$  и утонченной нейросети  $F_k$ , в которой используются лишь те из весов предыдущего шага алгоритма  $w^{(k)}$ , которые относятся к неисключенным нейронам.

$$w^{(k+1)} = w^{(k)} - \alpha \nabla L(Y^{(b)}, F_k(X^{(b)}; w^{(k)}))$$

**Примечание** - очевидно, что градиент функции потерь по исключенным весам равняется нулю.

- При прогнозировании вместо обученных весов  $w$  обычно используется их скорректированная на вероятность исключения нейрона версия  $pw$ .
- **Обратное исключение** – каждую итерацию обучения вместо  $F_k(X^{(b)}; w^{(k)})$  используется  $F_k(X^{(b)}; w^{(k)}/p)$ , а при прогнозировании обычные обученные веса  $w$ , что иногда снижает вычислительную нагрузку.
- **Интуиция** – деление  $w^{(k)}/p$  позволяет сохранить ожидаемые значения линейных комбинаций и поэтому помогает избежать проблем с изменениями размерности вследствие исключения нейронов.

# Исключение (dropout / dilution)

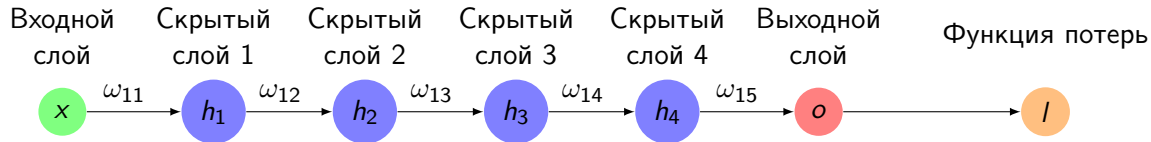
## Технические рекомендации

- Рекомендуется использовать достаточно большую скорость обучения  $\alpha$ , примерно в 10 – 100 раз больше, чем при обычном (без исключения) обучении нейросети.
- Для того, чтобы избежать чрезмерно больших значений весов при большом значении скорости обучения  $\alpha$ , рекомендуется устанавливать ограничение на L2-норму  $\|W\|_2 \leq c$ , где  $c$  выступает в качестве гиперпараметра, обычно принимающего значения в интервале (3, 4).
- Если в одном из скрытых слоев отключены все нейроны, то из него передается только константа.
- Вероятность  $p$  рекомендуется брать из интервала (0.5, 0.8) и делать ее тем большей, чем меньше общее число нейронов в нейросети.



# Проблема затухающих и взрывающихся градиентов

Иллюстрация на простом примере с одним признаком



$$s_j = \begin{cases} \omega_{11}x + \omega_{01}, & \text{если } j = 1 \\ \omega_{1j}h_{j-1} + \omega_{0j}, & \text{если } j > 1 \end{cases}$$

$$\frac{\partial l}{\partial \omega_{11}} = \frac{\partial l}{\partial o} \frac{\partial o}{\partial s_5} \frac{\partial s_5}{\partial h_4} \frac{\partial h_4}{\partial s_4} \frac{\partial s_4}{\partial h_3} \frac{\partial h_3}{\partial s_3} \frac{\partial s_3}{\partial h_2} \frac{\partial h_2}{\partial s_2} \frac{\partial s_2}{\partial h_1} \frac{\partial h_1}{\partial s_1} \frac{\partial s_1}{\partial \omega_{11}} = \frac{\partial l}{\partial o} \frac{\partial o}{\partial s_5} \omega_{15} \frac{\partial h_4}{\partial s_4} \omega_{14} \frac{\partial h_3}{\partial s_3} \omega_{13} \frac{\partial h_2}{\partial s_2} \omega_{12} \frac{\partial h_1}{\partial s_1} x$$

**Интуиция** – если произведения  $(\partial h_j / \partial s_j) \omega_{1j}$ , как правило, оказываются по модулю меньше (больше) единицы, то возникает **проблема затухающих (взрывающихся) градиентов**: при большом числе скрытых слоев производные весов быстро приближаются к нулю (большим значениям). Из-за этого при использовании методов численной оптимизации, опирающихся на информацию о градиенте, например, градиентного спуска, скорость обучения весов в первых (последних) слоях окажется гораздо ниже. Эта неравномерность в скорости обучения, в частности, серьезно повышает время численной оптимизации.

# Проблема затухающих и взрывающихся градиентов

Решение с помощью использования специальных функций активации

- Проблема затухающих градиентов чаще встречается при использовании определенных функций активации, таких как, например, сигмоида, производная которой имеет вид:

$$h(t) = \frac{1}{1 + e^{-t}} \implies h'(t) = \frac{e^{-t}}{(1 + e^{-t})^2} \in (0, 0.25)$$

- **Проблема** – производная сигмоидной функции активации принимает значения меньше 0.25 и достаточно быстро убывает, в частности, при аргументах  $|t| > 3$  оказывается меньше 0.05, что благоприятствует затуханию градиентов.
- **Решение** – воспользоваться альтернативными функциями активации, такими как, например, ReLu (rectified linear unit), производная которой принимает два значения:

$$h(t) = \max(0, t) \implies h'(t) = \begin{cases} 1, & \text{если } t > 0 \\ \text{не определена, но берет 0 или 1,} & \text{если } t = 0 \\ 0, & \text{если } t < 0 \end{cases}$$

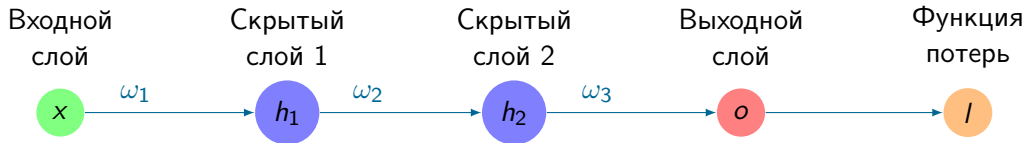
# Пакетная нормализация (batch normalization)

## Краткое описание

- Для того, чтобы повысить стабильность вычислений в нейронной сети, рекомендуется **нормализовать** данные (к нулевому выборочному среднему и единичной выборочной дисперсии) входного слоя, то есть признаки.
- **Проблема внутреннего ковариатного сдвига** – даже если данные входного слоя нормализованы, распределение значений скрытых слоев может существенно разниться и изменяться вместе с весами нейронной сети по мере численной оптимизации. В частности, это может приводить к проблеме затухающих и взрывающихся градиентов.
- **Решение** – использовать **пакетную нормализацию**, основная идея которой заключается в нормализации данных каждого скрытого слоя. При использовании мини-пакетного градиентно спуска для этого используются выборочное среднее и выборочная дисперсия данных слоя, рассчитанные лишь с использованием мини-пакета.
- Вследствие нормализации теряется часть полезной информации, с целью сохранения возможности восстановления которой нормализованные данные  $h_{kt}$ , относящиеся к  $t$ -й функции активации  $k$ -го слоя, перед передачей в следующий слой преобразуются как  $h_{kt}^* = \tau_{1kt} \times h_{kt} + \tau_{0kt}$ , где  $\tau_{1kt}$  и  $\tau_{0kt}$  обучаются вместе с весами.

# Нейросети в матричном виде

## Формулировка



- Часто нейронные сети удобно мыслить в матричном виде:

$$x = \begin{bmatrix} x_{11} & \dots & x_{1m_0} \\ x_{21} & \dots & x_{2m_0} \\ \vdots & \vdots & \vdots \\ x_{n1} & \dots & x_{nm_0} \end{bmatrix} \quad \omega_j = \begin{bmatrix} \omega_{j11} & \dots & \omega_{j1m_{(j-1)}} \\ \omega_{j21} & \dots & \omega_{j2m_{(j-1)}} \\ \vdots & \vdots & \vdots \\ \omega_{jm_j1} & \dots & \omega_{jm_jm_{(j-1)}} \end{bmatrix} \quad b_j = \begin{bmatrix} b_{j1} & \dots & b_{jm_j} \\ b_{j1} & \dots & b_{jm_j} \\ \vdots & \vdots & \vdots \\ b_{j1} & \dots & b_{jm_j} \end{bmatrix} \quad o = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_n \end{bmatrix} \quad l = \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_n \end{bmatrix}$$

$$h_1 = a_1(x\omega_1^T + b_1) \quad h_2 = a_2(h_1\omega_2^T + b_2) \quad o = g(h_2\omega_3^T + b_3) \quad L(y, o) = \sum_{i=1}^n l_i = \sum_{i=1}^n L(y_i, o_i)$$

- Важно** – в данном случае  $h_j$  обозначает не один нейрон, а вектор (сразу несколько) нейронов.
- Функции активации  $\alpha_1(\cdot)$ ,  $\alpha_2(\cdot)$  и функция  $g(\cdot)$  берутся **поэлементно** от соответствующих матриц.
- Число нейронов в скрытом слое  $j$  обозначается  $m_j$ , а через  $b_{jk}$  обозначается относящаяся к его  $k$ -й линейной комбинации константа (смещение). Через  $m_0$  обозначается число признаков.

# Нейросети в матричном виде

## Численный пример расчета

- Рассмотрим случай с двумя признаками, тремя наблюдениями, без смещений (констант), с тремя нейронами в первом скрытом слое и двумя нейронами во втором скрытом слое:

$$x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad y = [1 \quad 0 \quad 1] \quad \omega_1 = \begin{bmatrix} -0.1 & 0.2 \\ 0.3 & -0.4 \\ -0.5 & 0.6 \end{bmatrix} \quad \omega_2 = \begin{bmatrix} 0.7 & 0.8 & 0.9 \\ 1 & 1.1 & 1.2 \end{bmatrix} \quad \omega_3 = [1.3 \quad -1.4]$$

- Предположим следующие функции активации:

$$\alpha_1(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} \quad \alpha_2(s) = \max(0, s) \quad g(s) = \frac{1}{1 + e^{-s}}$$

- По результатам расчетов получаем значения нейронов и выходного слоя:

$$h_1 \approx \begin{bmatrix} 0.29 & -0.46 & 0.6 \\ 0.46 & -0.6 & 0.72 \\ 0.6 & -0.72 & 0.8 \end{bmatrix} \quad h_2 \approx \begin{bmatrix} 0.38 & 0.51 \\ 0.48 & 0.66 \\ 0.57 & 0.78 \end{bmatrix} \quad o \approx \begin{bmatrix} 0.45 \\ 0.43 \\ 0.41 \end{bmatrix}$$

- Если функция потерь является логистической, то ее значение при соответствующих весах будет:

$$l_i = -y_i \ln(o_i) - (1 - y_i) \ln(1 - o_i) \Rightarrow l \approx \begin{bmatrix} 0.81 \\ 0.56 \\ 0.88 \end{bmatrix} \Rightarrow L(y, o) = \frac{1}{n} \sum_{i=1}^n l_i \approx \frac{0.81 + 0.56 + 0.88}{3} = 0.75$$

# Нейросети в матричном виде

Обучение – поворение основных идей с матричными обозначениями

- Для удобства введем общее обозначение для вектора всех параметров нейросети:

$$W = (\text{vec}(w_1), \text{vec}(w_2), \dots, \text{vec}(w_{n_h}), b_{11}, \dots, b_{n_h m_h})$$

Через  $n_h$  обозначается количество слоев. Функция  $\text{vec}(\cdot)$  превращает матрицу в вектор, раскладывая ее по столбцам. В частности, в предыдущем примере:

$$\text{vec}(w_1) = (-0.1, 0.3, -0.5, 0.2, -0.4, 0.6)$$

- Напомним, что итоговый градиент можно посчитать методом обратного распространения ошибки. Для этого удобно использовать формулы матричного дифференцирования, например, по  $\text{vec}(w_j)$ . Однако, частные производные можно считать и отдельно по каждому параметру:

$$\nabla L = \left( \frac{\partial L}{\partial W_1}, \dots, \frac{\partial L}{\partial W_N} \right) = \frac{1}{n} \sum_{i=1}^n \left( \frac{\partial l_i}{\partial W_1}, \dots, \frac{\partial l_i}{\partial W_N} \right)$$

Где  $N$  - общее число параметров нейросети.

- Напомним, что обучение весов происходит с помощью численной оптимизации, например, в случае применения обычного градиентного спуска со скоростью обучения  $\alpha$  каждая итерация выглядит как:

$$W^{\text{new}} = W^{\text{old}} - \alpha \nabla L(y, F(x; W^{\text{old}}))$$