Проектирование интеллектуальных систем

Лекция № 5. Многослойные нейронные сети

Содержание

Возможности двух- и трёхслойных сетей

Так как один нейрон (или сеть, составленная из независимых нейронов) имеет существенные ограничения, большее распространение получили *многослойные нейронные сети*, которые могут решать более сложные задачи.

Идея послойной организации нейронов в сети состоит в следующем. Нейроны делятся на группы, и каждая группа нейронов образует так называемый **слой**. Каждый слой имеет определённую позицию (порядок) в нейронной сети. Нейроны соседних слоёв связаны между собой таким образом, что выходные значения одних нейронов являются входными значениями других нейронов. Связи между нейронами одного слоя, как правило, отсутствуют.

Если каждый нейрон последующего слоя имеет связь со всеми нейронами предыдущего слоя, то такая сеть называется **полносвязанной** (или полносвязной). Если сигналы передаются в сети строго от входа к выходу (то есть нет обратных связей), такая сеть называется **сетью прямого распространения**. В данном разделе рассматриваются полносвязные нейронные сети прямого распространения.

В зависимости от положения, слои можно разделить на *входной*, *выходной* и *скрытые*. **Выходной слой** – это последний слой нейронной сети, выходные значения которого считаются выходными значениями нейронной сети (решением задачи). **Входной слой** образуют сигналы, которые подаются на вход нейронной сети. **Скрытые слои** – это все остальные слои, расположенные «внутри» нейронной сети.

Существуют некоторые расхождения во мнениях между исследователями в том, что считать «слоем» и каким образом проводить подсчёт слоёв. Одни исследователи считают, что входные сигналы нельзя считать отдельным слоем, так как в этом «слое» не происходит никаких вычислений (то есть в

нём нет нейронов). Однако, на графах, отображающих структуру нейронной сети, входные значения обычно изображают такими же вершинами графа, как и нейроны, поэтому некоторые исследователи рассматривают входные сигналы как отдельный слой. В последующем изложении будем считать, что входные сигналы не являются отдельным слоем.

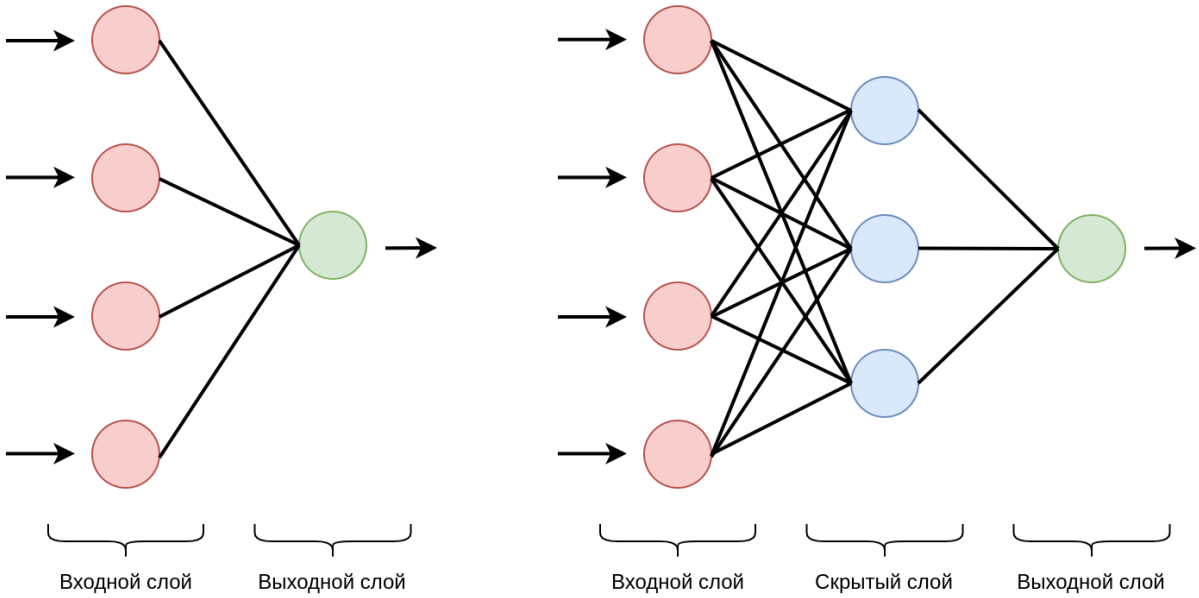

Рис. 1. Пример однослойной и многослойной нейронных сетей.

Рассмотрим возможности двух- и трёхслойных нейронных сетей. Лучше всего это сделать на примере задачи бинарной классификации в двумерном пространстве (т. е. на плоскости). В этом случае можно легко визуализировать разделяющую границу, которую строит нейронная сеть.

Примером ограниченных возможностей однослойной сети является проблема XOR. Ниже представлена архитектура двухслойной нейронной сети, которая может решить эту задачу (рис. 2). Эта нейронная сеть состоит из трёх нейронов: 2 в первом слое и 1 – во втором. Функция активации – пороговая. Каждый из нейронов первого слоя задаёт разделяющую прямую. Нейрон 3 задаёт прямую с нормалью (w_{31}, w_{32}), а нейрон 4 – с нормалью (w_{41}, w_{42}). От

направления нормали зависит, с какой стороны от прямой будет область, для точек из которой соответствующий нейрон выдаёт значение 1. Пересечением этих областей является заштрихованная на графике область. Чтобы реализовать пересечение областей, нейрон 5 должен воспроизвести функцию логического И. Этого можно добиться, например, с использованием значений $w_{53} = 0{,}5;\ w_{54} = 0{,}5$ и $b_5 = -0{,}6$.

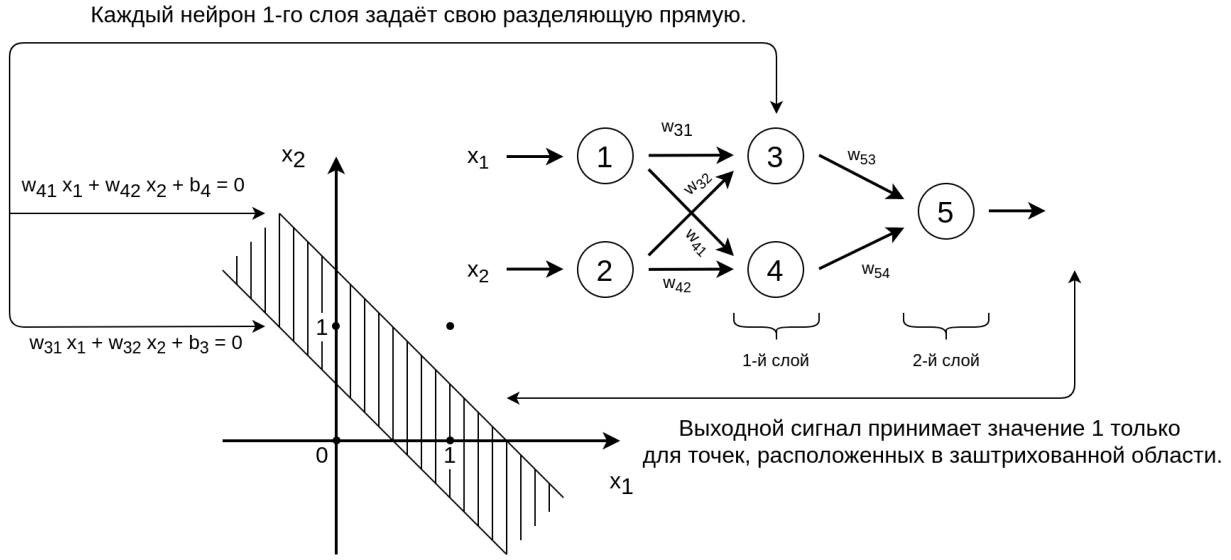

Рис. 2. Пример двухслойной нейронной сети, которая может воспроизвести функцию XOR.

Обобщая рассмотренный пример, можно сделать вывод, что с помощью двухслойной нейронной сети можно задать произвольную выпуклую область решения в пространстве признаков [6]. Область называется *выпуклой*, если отрезок, соединяющий две произвольные точки из этой области также полностью принадлежит ей.

Чтобы преодолеть ограничение выпуклости, можно добавить в нейронную сеть третий слой. Каждый нейрон третьего слоя может реализовывать логические операции над выпуклыми областями, которые получаются на вы-

ходе нейронов второго слоя. Например, если объединить две выпуклые области, результат не обязательно будет выпуклой областью. Пример такой операции представлен на рисунке ниже.

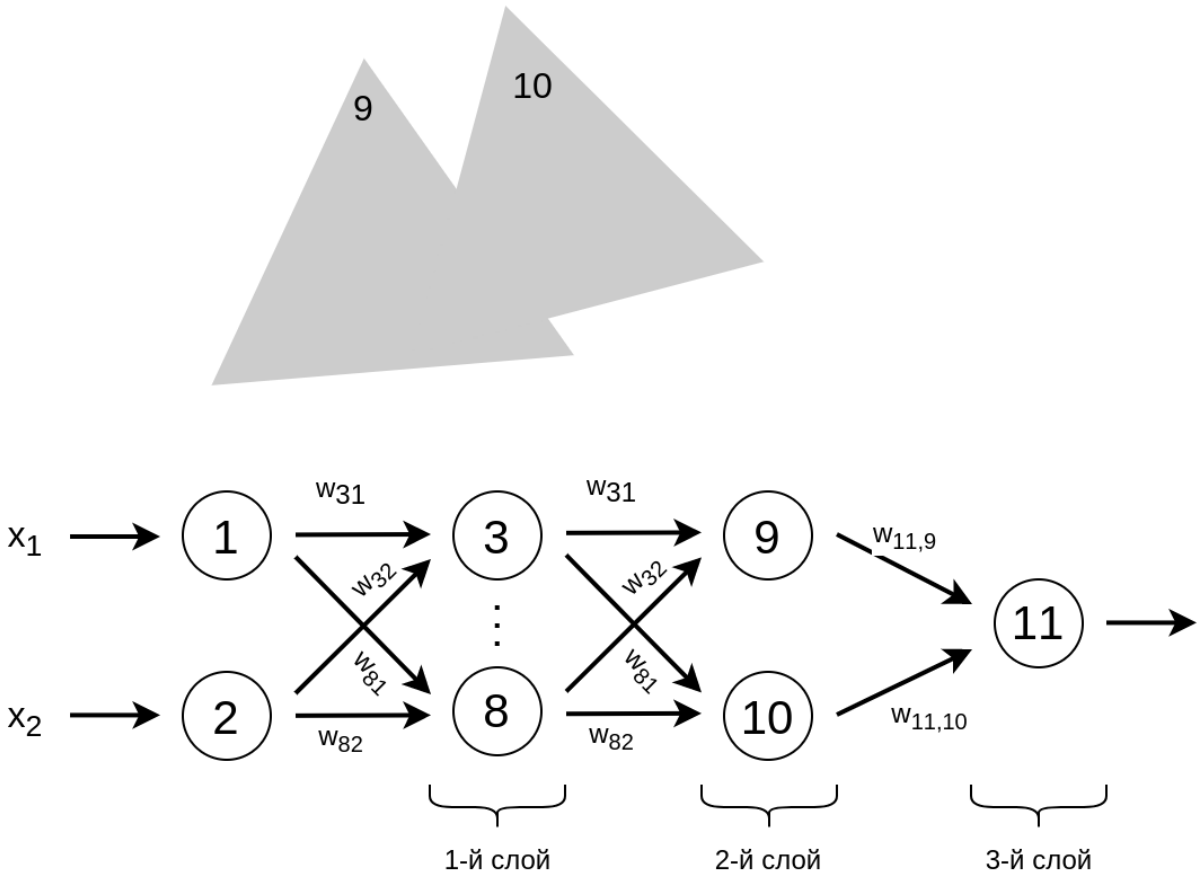

Рис. 3. Пример задания невыпуклой области с помощью трёхслойной нейронной сети.

Предположим, что нейроны 9 и 10 задают треугольные выпуклые области. Чтобы задать треугольную область требуется построить три грани, поэтому в первом слое 6 нейронов – по три на каждый треугольник. Чтобы объединить треугольные области, нейрон 11 должен воспроизвести функцию логического ИЛИ. Этого можно добиться, например, с использованием значе-

ний $w_{11,9} = 0{,}5$; $w_{11,10} = 0{,}5$ и $b_5 = -0{,}4$. По приведённому рисунку видно, что полученная область не является выпуклой.

Выбор функции активации

Основные требования, которым должна удовлетворять функция активации, которая используется в многослойных сетях, – это нелинейность и дифференцируемость. Современные нейронные сети чаще всего обучаются при помощи градиентного спуска, отсюда требование дифференцируемости функции активации. Нелинейность является менее очевидным требованием. В некоторых случаях без линейной функции активации не обойтись. Например, при решении задачи регрессии в качестве функции активации нейронов выходного слоя используется именно линейная функция. Однако, использование линейной функции в скрытых слоях многослойной нейронной сети значительно ограничивает возможности этой нейронной сети.

Под линейной функцией активации понимается функция

$$\varphi(v) = \alpha v + \beta,$$

где α и β – это некоторые действительные числа.

В простейшем случае, когда $\alpha = 1$ и $\beta = 0$, имеем тождественную функцию $\varphi(v) = v$.

Предположим, что эта функция активации используется в двухслойной нейронной сети (рис. 4).

Если обозначить

$$\mathbf{X} = [x_1 \quad x_2], \quad \mathbf{W^{(1)}} = \begin{bmatrix} w_{31} & w_{41} \\ w_{32} & w_{42} \end{bmatrix}, \quad \mathbf{W^{(2)}} = \begin{bmatrix} w_{53} \\ w_{54} \end{bmatrix},$$

то можно записать следующее

$$\hat{y}_5 = \varphi(\varphi(\mathbf{X}\mathbf{W^{(1)}})\mathbf{W^{(2)}}),$$

где $\mathbf{X}$ – матрица (в данном случае матрица-строка) входных значений, $\mathbf{W^{(1)}}$ – матрица весов первого слоя, $\mathbf{W^{(2)}}$ – матрица весов второго слоя. Функция активации φ применяется поэлементно.

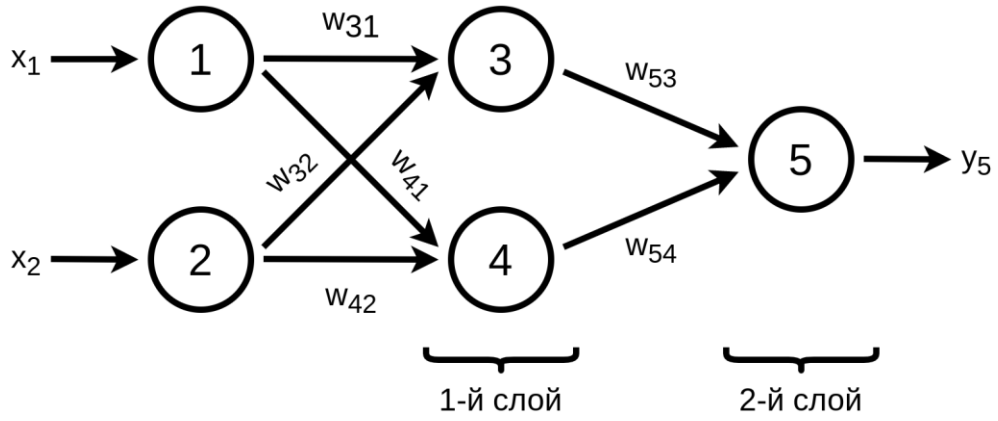

Рис. 4. Пример двухслойной нейронной сети.

Так как $\varphi(v) = v$, можно переписать формулу следующим образом

$$\hat{y}_5 = \mathbf{X}\mathbf{W^{(1)}}\mathbf{W^{(2)}}.$$

Введем обозначение

$$\mathbf{W} = \mathbf{W^{(1)}}\mathbf{W^{(2)}} = \begin{bmatrix} w_{31} & w_{41} \\ w_{32} & w_{42} \end{bmatrix}\begin{bmatrix} w_{53} \\ w_{54} \end{bmatrix} = \begin{bmatrix} w_{31}w_{53} + w_{41}w_{54} \\ w_{32}w_{53} + w_{42}w_{54} \end{bmatrix}.$$

Тогда получим, что $\hat{y}_5 = \mathbf{XW}$, то есть при использовании линейной функции активации двухслойная сеть эквивалентна однослойной сети с матрицей весов $\mathbf{W}$. Можно доказать, что это будет справедливо и для большего количества слоёв. Таким образом, использование линейной функции активации в многослойной сети приводит к её вырождению в однослойную сеть с присущими ей ограничениями.

Помимо сигмоиды, в нейронных сетях также используется функция **гиперболический тангенс**, которая задаётся выражением

$$th(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}}.$$

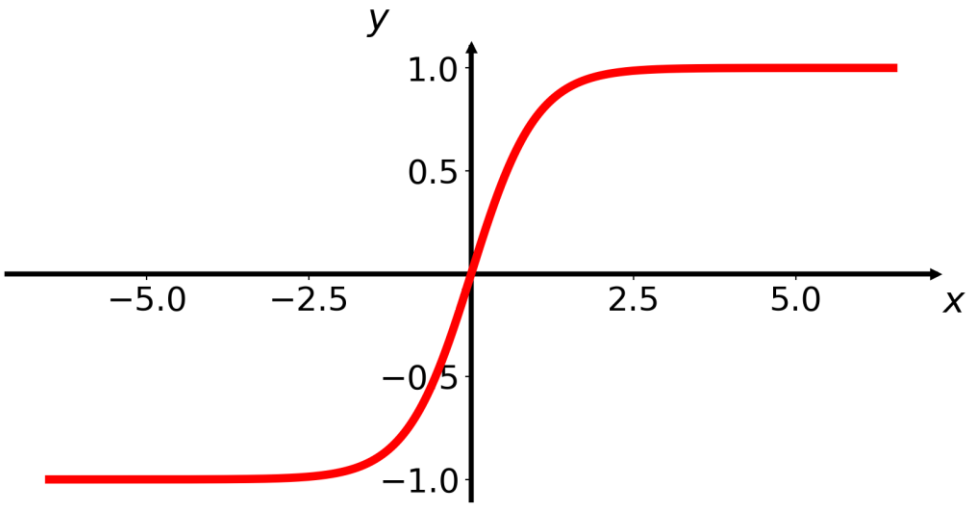

Рис. 5. Гиперболический тангенс.

Гиперболический тангенс имеет тесную связь с сигмоидой, и даже может быть выражен через неё

$$th(v) = 2\sigma(2v) - 1.$$

Производная гиперболического тангенса имеет вид

$$th'(v) = 1 - th(v)^2.$$

Сигмоида и гиперболический тангенс подвержены проблеме насыщения. Она заключается в том, что при очень больших или очень меленьких значениях аргумента эти функции выходят на плато, и значение производной в таких точках близко к 0. Это приводит к занулению градиента, и нейронная сеть перестаёт обучаться. В контексте обучения нейронных сетей это называется *проблемой исчезновения градиента*. В результате исчезновения градиента наступает так называемый «паралич сети».

Чтобы избежать проблемы исчезновения градиента, используется функция ReLU (англ. Rectified Linear Unit), которая имеет следующий вид

$$ReLU(v) = max(0, v).$$

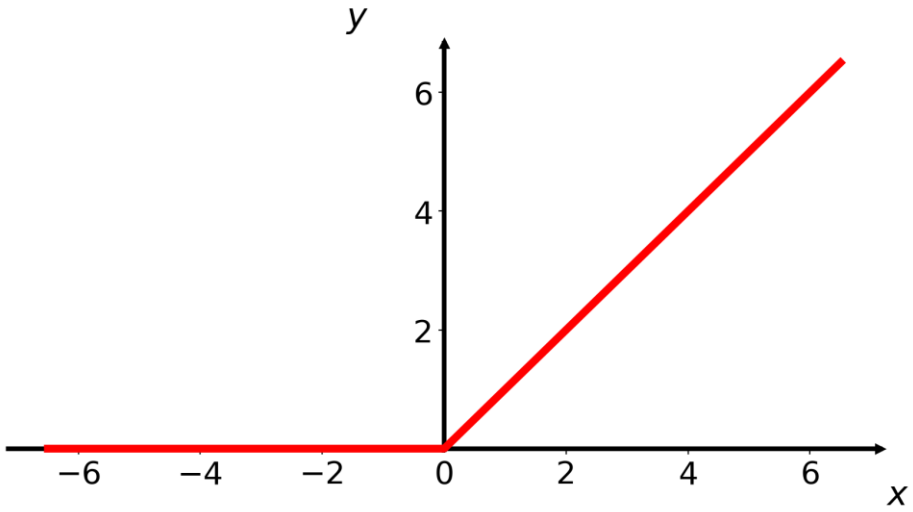

Рис. 6. ReLU.

Функция ReLU не является непрерывно дифференцируемой, так как в точке $v = 0$ её производная не определена. Однако, на практике, чтобы можно было использовать эту функцию в нейронной сети, используется субградиент, чтобы доопределить значение производной в этой точке. Таким образом

$$ReLU'(v) = \begin{cases} 0, & \text{если } v \leq 0, \\ 1, & \text{иначе.} \end{cases}$$

Достоинством функции ReLU является отсутствие проблем с насыщением, а также простота вычисления.

Чтобы не прибегать к субградиенту, можно воспользоваться «гладким» аналогом ReLU – функцией Softplus

$$Softplus(v) = \log(e^v + 1).$$

Производной от функции Softplus по её аргументу является сигмоида

$$Softplus'(v) = \sigma(v).$$

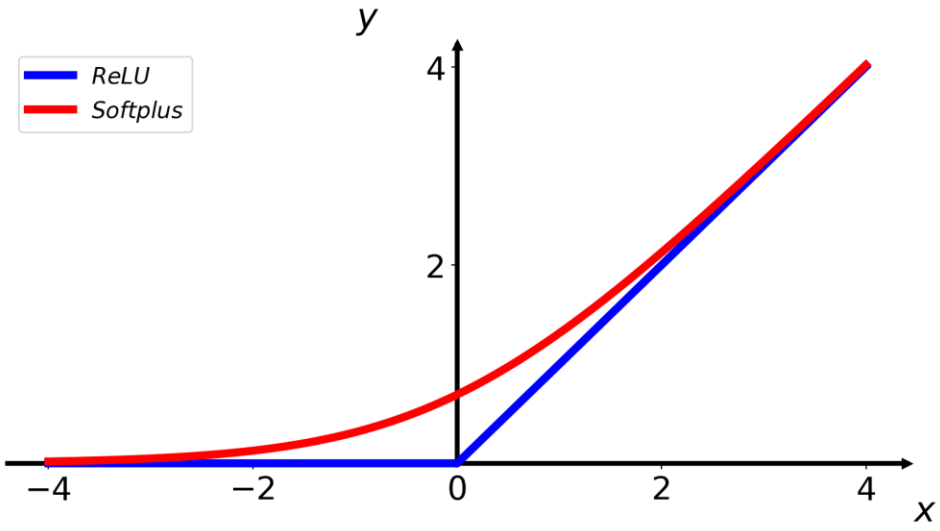

Рис. 7. Softplus.

Алгоритм обратного распространения ошибки

Так как в многослойной нейронной сети заранее неизвестно, чему должны равняться выходные значения нейронов скрытых слоёв, нельзя подобрать эвристический алгоритм обучения, основанный на коррекции ошибки. Вместо этого для обучения многослойных сетей используется градиентный спуск. Для применения градиентного спуска необходимо вычислять частные производные от целевой функции по параметрам нейронной сети (весам и смещениям). Для этого используется **алгоритм обратного распространения ошибки** (англ. backpropagation), который основан на правиле дифференцирования сложной функции и принципах динамического программирования.

Выведем алгоритм обратного распространения ошибки на примере небольшой нейронной сети с 3-мя слоями (рис. 8).

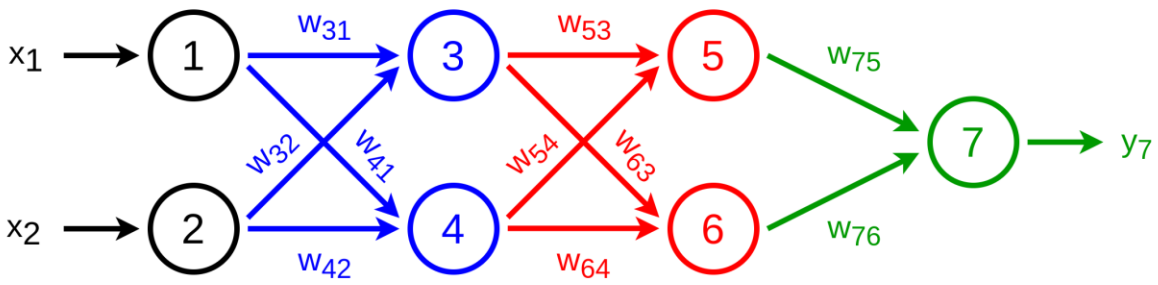

Рис. 8. Пример нейронной сети с 3-мя слоями.

Для того, чтобы вычислить выходное значение $\hat{y}_7$, нужно воспользоваться следующими формулами

$$\begin{aligned}
\hat{y}_7 &= \varphi(v_7), & v_7 &= w_{75}\,\hat{y}_5 + w_{76}\,\hat{y}_6,\\
\hat{y}_6 &= \varphi(v_6), & v_6 &= w_{63}\,\hat{y}_3 + w_{64}\,\hat{y}_4,\\
\hat{y}_5 &= \varphi(v_5), & v_5 &= w_{53}\,\hat{y}_3 + w_{54}\,\hat{y}_4,\\
\hat{y}_4 &= \varphi(v_4), & v_4 &= w_{41}\,\hat{y}_1 + w_{42}\,\hat{y}_2,\\
\hat{y}_3 &= \varphi(v_3), & v_3 &= w_{31}\,\hat{y}_1 + w_{32}\,\hat{y}_2,\\
\hat{y}_2 & & &= x_2,\\
\hat{y}_1 & & &= x_1.
\end{aligned}$$

Здесь для простоты изложения опущены смещения.

Пусть в качестве функции активации используется сигмоида $\varphi(v) = \frac{1}{1+e^{-v}}$. Определим функцию потерь следующим образом

$$L(\mathbf{w}) = \frac{1}{2}(\hat{y}_7 - y)^2,$$

где $\hat{y}_7$ – выходное значение нейрона 7, y – желаемое выходное значение, $\mathbf{w}$ – вектор параметров нейронной сети.

$$\mathbf{w} = (w_{31}, w_{32}, w_{41}, w_{42}, w_{53}, w_{54}, w_{63}, w_{64}, w_{75}, w_{76}).$$

Воспользовавшись правилом дифференцирования сложной функции вычислим $\frac{\partial L}{\partial w_{75}}$:

$$\frac{\partial L}{\partial w_{75}} = \frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}\,\frac{\partial v_7}{\partial w_{75}} = (\hat{y}_7 - y)\;\hat{y}_7(1-\hat{y}_7)\;\hat{y}_5 = \delta_7\;\hat{y}_7.$$

Обратите внимание, что $\frac{\partial \hat{y}_7}{\partial v_7}$ есть ничто иное как производная от сигмоиды по её аргументу

$$\frac{\mathrm{d}\sigma}{\mathrm{d}v} = \sigma(v)(1-\sigma(v)).$$

Теперь аналогично вычислим $\frac{\partial L}{\partial w_{76}}$

$$\frac{\partial L}{\partial w_{76}} = \frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}\,\frac{\partial v_7}{\partial w_{76}} = \delta_7\;\hat{y}_6.$$

В этой формуле присутствует подвыражение $\frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}$, которое мы обозначили за δ_7 при рассчёте предыдущей производной. Мы уже вычислили, чему оно равно, поэтому нет необходимости заново производить вычисления.

Теперь вычислим производные по весам нейрона 5:

$$\frac{\partial L}{\partial w_{54}} = \frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}\,\frac{\partial v_7}{\partial \hat{y}_5}\,\frac{\partial \hat{y}_5}{\partial v_5}\,\frac{\partial v_5}{\partial w_{54}} = \delta_7\;w_{75}\;\hat{y}_5(1-\hat{y}_5)\;\hat{y}_4 = \delta_5\;\hat{y}_4,$$

$$\frac{\partial L}{\partial w_{53}} = \frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}\,\frac{\partial v_7}{\partial \hat{y}_5}\,\frac{\partial \hat{y}_5}{\partial v_5}\,\frac{\partial v_5}{\partial w_{53}} = \delta_5\;\hat{y}_3.$$

По приведённым формулам видно, что при вычислении производных по параметрам w_{54} и w_{53} изменяется лишь последний множитель. Общую часть мы обозначили δ_5 – её можно вычислить лишь один раз. Также обратите внимание, что при вычислении δ_5 мы воспользовались полученным ранее значением δ_7.

Производные по весам нейрона 6 вычисляются аналогично:

$$\frac{\partial L}{\partial w_{64}} = \frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}\,\frac{\partial v_7}{\partial \hat{y}_6}\,\frac{\partial \hat{y}_6}{\partial v_6}\,\frac{\partial v_6}{\partial w_{64}} = \delta_7\;w_{76}\;\hat{y}_6(1-\hat{y}_6)\;\hat{y}_4 = \delta_6\;\hat{y}_4,$$

$$\frac{\partial L}{\partial w_{63}} = \frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}\,\frac{\partial v_7}{\partial \hat{y}_6}\,\frac{\partial \hat{y}_6}{\partial v_6}\,\frac{\partial v_6}{\partial w_{63}} = \delta_6\;\hat{y}_3.$$

Формулы для вычисления производных по параметрам нейронов 3 и 4 будут несколько сложнее, так как выходные значения этих нейронов оказывают влияние как на $\hat{y}_5$, так и на $\hat{y}_6$ (то есть более чем на один нейрон следующего слоя).

Для веса w_{31} получим формулу

$$\frac{\partial L}{\partial w_{31}} = \frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}\,\frac{\partial v_7}{\partial w_{31}}.$$

Заметим, что

$$\frac{\partial v_7}{\partial w_{31}} = \frac{\partial v_7}{\partial \hat{y}_5}\,\frac{\partial \hat{y}_5}{\partial w_{31}} + \frac{\partial v_7}{\partial \hat{y}_6}\,\frac{\partial \hat{y}_6}{\partial w_{31}}.$$

Тогда можно написать

$$\begin{aligned}\frac{\partial L}{\partial w_{31}} &= \frac{\partial L}{\partial \hat{y}_7}\,\frac{\partial \hat{y}_7}{\partial v_7}\left(\frac{\partial v_7}{\partial \hat{y}_5}\,\frac{\partial \hat{y}_5}{\partial v_5}\,\frac{\partial v_5}{\partial \hat{y}_3}\,\frac{\partial \hat{y}_3}{\partial v_3}\,\frac{\partial v_3}{\partial w_{31}} + \frac{\partial v_7}{\partial \hat{y}_6}\,\frac{\partial \hat{y}_6}{\partial v_6}\,\frac{\partial v_6}{\partial \hat{y}_3}\,\frac{\partial \hat{y}_3}{\partial v_3}\,\frac{\partial v_3}{\partial w_{31}}\right) = \\ &= \frac{\partial \hat{y}_3}{\partial v_3}\,\frac{\partial v_3}{\partial w_{31}}\left(\delta_7\,\frac{\partial v_7}{\partial \hat{y}_5}\,\frac{\partial \hat{y}_5}{\partial v_5}\,\frac{\partial v_5}{\partial \hat{y}_3} + \delta_7\frac{\partial v_7}{\partial \hat{y}_6}\,\frac{\partial \hat{y}_6}{\partial v_6}\,\frac{\partial v_6}{\partial \hat{y}_3}\right) = \\ &= \hat{y}_3(1-\hat{y}_3)\,(\delta_5\,w_{53} + \delta_6\,w_{63})\,\hat{y}_1 = \delta_3\,\hat{y}_1.\end{aligned}$$

Для весов w_{32}, w_{41} и w_{42} формулы можно получить аналогично:

$$\frac{\partial L}{\partial w_{32}} = \delta_3\,\hat{y}_2, \quad \frac{\partial L}{\partial w_{41}} = \delta_4\,\hat{y}_1, \quad \frac{\partial L}{\partial w_{42}} = \delta_4\,\hat{y}_2.$$

Таким образом, универсальная формула для вычисления производной по любому синаптическому весу выглядит следующим образом

$$\frac{\partial L}{\partial w_{qj}} = \delta_q\,\hat{y}_j,$$

где $\delta_q = \frac{\partial L}{\partial v_q}$.

Значение δ_q будет вычисляться по-разному в зависимости от того, находится ли нейрон q в выходном слое. Если это так, то

$$\delta_q = \frac{\partial L}{\partial \hat{y}_q}\,\frac{\partial \hat{y}_q}{\partial v_q},$$

то есть производную можно вычислить непосредственно воспользовавшись правилом дифференцирования сложной функции.

Если же нейрон q находится в скрытом слое, то

$$\delta_q = \frac{\partial \hat{y}_q}{\partial v_q} \sum_{i \in O_q} \delta_i \, w_{iq},$$

где O_q – множество номеров нейронов следующего слоя, с которыми связан нейрон q.

Таким образом, суть алгоритма обратного распространения ошибки состоит в том, что производные целевой функции по параметрам нейронной сети вычисляются последовательно, начиная с параметров выходного слоя и заканчивая параметрами первого слоя (то есть в обратном порядке). Закономерности, обусловленные структурой нейронной сети, позволяют при вычислении производных более ранних слоёв использовать значения, вычисленные на предыдущем шаге алгоритма. Это позволяет значительно ускорить вычисления.

Формулы, приведённые выше, будут справедливы не для любой целевой функции. В приведённом выше примере в целевой функции учитывается лишь один обучающий пример. В реальных задачах объём обучающей выборки может достигать десятков тысяч примеров, и все они должны учитываться в целевой функции. Чтобы можно было применить алгоритм обратного распространения ошибки, целевая функция должна удовлетворять следующим условиям [5].

- Целевая функция должна быть представима в виде суммы значений функции потерь по каждому примеру обучающей выборки

$$L(\mathbf{w}) = \sum_{i=1}^{n} L_i,$$

- где n – размер обучающей выборки. Это условие необходимо, так как приведённые выше формулы можно использовать, чтобы вычислить производные лишь по одному примеру. Если выполнено указанное выше условие, то

$$\frac{\partial L}{\partial w_{qj}} = \sum_{i=1}^{n} \frac{\partial L_i}{\partial w_{qj}},$$

- то есть производная от целевой функции по какому-то параметру w_{qj} вычисляется как сумма производных по каждому примеру. Так как производную по каждому слагаемому можно вычислить независимо от остальных, то чтобы вычислить искомую производную, достаточно применить алгоритм обратного распространения ошибки к каждому слагаемому и просуммировать полученные результаты.

- Целевая функция должна зависеть от выходных значений нейронной сети. Если это не так, то целевая функция не зависит от параметров нейронной сети, и вычислять от неё производные по параметрам сети не имеет смысла.

- Целевая функция (и функции активации нейронов) должны быть дифференцируемыми.

«Паралич сети» – проблема затухания градиента

Обратите внимание, что в формуле для вычисления δ_q присутствует множитель $\partial y_q / \partial v_q$, то есть производная от функции активации. Это объясняет явление затухания градиента в глубоких сетях.

Рассмотрим упрощённый пример, наглядно демонстрирующий эту проблему. Предположим, что имеется нейронная сеть, состоящая из 4 нейронов,

расположенных последовательно друг за другом. Предположим также, что веса нейронов равны 1, а смещения – 0. В качестве функции активации нейронов используется сигмоида. Отметим, что наибольшее значение производная сигмоиды достигает в точке 0. Это значение $1/4$. Вычислим производную $\partial \mathcal{Y}_4/\partial x$:

$$\frac{\partial \mathcal{Y}_4}{\partial x} = \frac{\partial \sigma(\mathcal{Y}_3)}{\partial \mathcal{Y}_3}\frac{\partial \sigma(\mathcal{Y}_2)}{\partial \mathcal{Y}_2}\frac{\partial \sigma(\mathcal{Y}_1)}{\partial \mathcal{Y}_1}\frac{\partial \sigma(\mathcal{Y}_1)}{\partial x} \leq \left(\frac{1}{4}\right)^4.$$

Величина этой производной ограничена сверху значением $(1/4)^4$. Если бы в этой сети было t слоёв по одному нейрону, мы бы получили верхнюю оценку равную $(1/4)^t$.

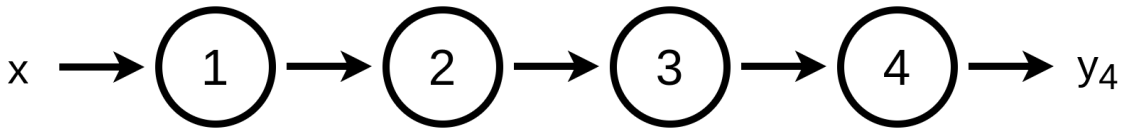

Рис. 9. Нейронная сеть для иллюстрации проблемы затухания градиента.

Аналогичная ситуация возникает при обучении глубоких нейронных сетей. Конечно, там при вычислении производных присутствуют и другие множители, но чтобы градиент «затух» достаточно, чтобы хотя бы один из этих множителей был близок к нулю. С ростом числа слоёв такая ситуация становится практически неизбежной.

Для борьбы с этой проблемой используют функции активации, не подверженные насыщению (такие как ReLU), а также специальные приёмы, такие как «пропущенные соединения» (англ. skip connections) в свёрточных нейронных сетях.

Модификации градиентного спуска

Алгоритм обратного распространения ошибки используется для вычисления градиента целевой функции, а для минимизации целевой функции обычно используется градиентный спуск, который на основе вычисленного градиента обновляет параметры нейронной сети.

В случае, когда размер обучающей выборки велик, вычисление градиента может быть затратной операцией как с точки зрения времени вычисления, так и с точки зрения необходимого объёма памяти. Так, например, если в обучающей выборке присутствует 1000 примеров, то чтобы рассчитать градиент, нужно произвести прямой и обратный проходы по нейронной сети 1000 раз (т. е. для каждого примера), и только после этого можно будет обновить значения параметров. Это приводит к тому, что обучение нейронной сети происходит очень медленно. Однако, как правило, использование всех примеров из обучающей выборки для вычисления градиента *избыточно*, и можно получить достаточно точное значение на основе некоторой подвыборки. Эта идея лежит в основе некоторых модификаций классического пакетного градиентного спуска.

Одна из этих модификаций – **стохастический градиентный спуск** (англ. Stochastic Gradient Descent, SGD). В стохастическом градиентном спуске вместо вычисления градиента по всем примерам обучающей выборки, градиент вычисляется по *одному* случайно выбранному примеру. Таким образом, значение градиента целевой функции заменяется его *грубой аппроксимацией*

$$\frac{\partial L}{\partial w_{qj}} \approx \frac{\partial L_c}{\partial w_{qj}}, \quad 1 \leq c \leq n.$$

Время обучения нейронной сети часто измеряется в эпохах. **Эпоха** – один «проход» по всем примерам из обучающей выборки. В классическом

градиентном спуске обновление весов происходит один раз за эпоху. В стохастическом градиентном спуске за одну эпоху происходит столько шагов градиентного спуска, сколько примеров в обучающей выборке. За счёт этого обучение нейронной сети значительно ускоряется при использовании стохастического градиентного спуска.

Необходимо учитывать, что направления градиента, вычисленные по разным примерам, могут значительно отличаться, а если в выборке присутствуют выбросы, то направление градиента, вычисленное по эти выбросам, будет и вовсе смотреть «не туда». Поэтому если проследить траекторию движения от начального приближения к локальному минимуму, к которому сошёлся алгоритм стохастического градиентного спуска, можно увидеть, что эта траектория будет представлять собой ломанную линию, в отличие от траектории, получаемой классическим градиентным спуском (рис. 10) [4].

В качестве компромисса между пакетным и стохастическим градиентным спуском был предложен так называемый **«mini batch» градиентный спуск**. В этой модификации градиент вычисляется по некоторому случайному подмножеству примеров обучающей выборки. Таким образом с одной стороны получается более точное значение градиента, чем в стохастическом градиентном спуске, а с другой – более быстрая сходимость, чем в пакетном. Так как при использовании этого метода получается более точная аппроксимация градиента, и его направление более стабильно, этот метод более устойчив к выбору больших значений скорости обучения (α), чем стохастический градиентный спуск. Подмножество примеров, на основе которых вычисляется градиент, называется **пакетом** (англ. batch). Размер пакета определяется эмпирически исходя из доступного объёма оперативной памяти и особенностей решаемой задачи.

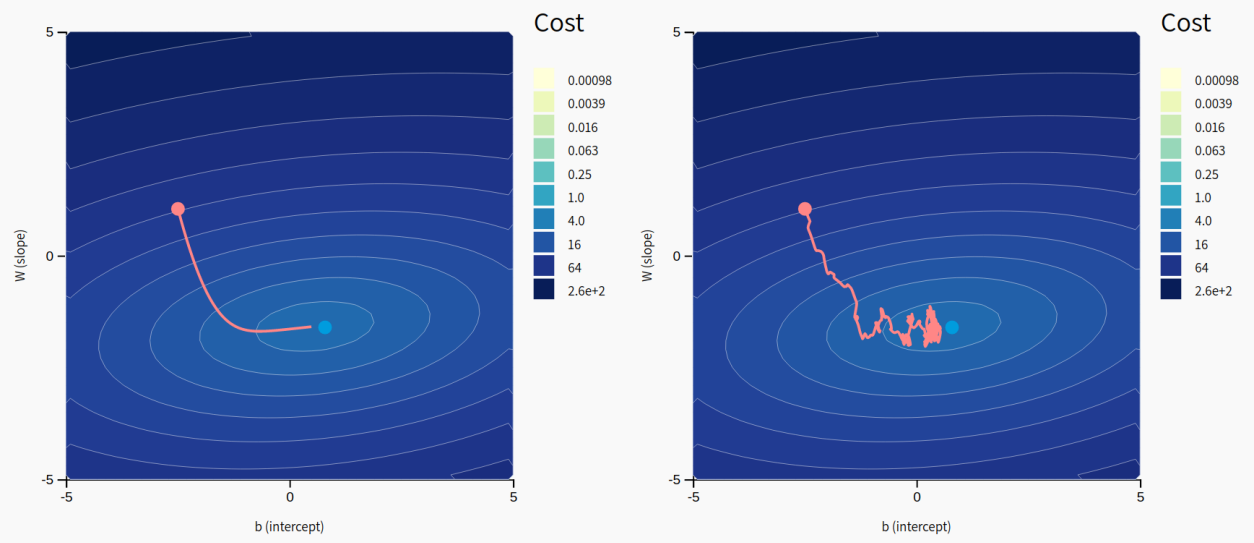

Рис. 10. Сравнение пакетного и стохастического градиентного спуска на примере задачи поиска оптимальных параметров линейной регрессии.[1]

Вопросы для самопроверки

1. Что такое полносвязная нейронная сеть прямого распространения?
2. Какими возможностями обладают двух- и трёхслойные сети?
3. Какими свойствами должна обладать функция активации? Перечислите основные функции активации, используемые в современных многослойных нейронных сетях.

---

[1] Katanforoosh, Kunin et al. «Parameter optimization in neural networks», 2019. URL: http://deeplearning.ai/ainotes/optimization/ (дата обращения: 17.09.2020)

4. В чём заключается суть алгоритма обратного распространения ошибки? Выпишите общие формулы для нахождения $\frac{\partial L}{\partial w_{qj}}$ и δ_q для некоторого нейрона q и входа j.
5. Какие требования предъявляются к целевой функции, чтобы к ней было возможно применить алгоритм обратного распространения ошибки?
6. В чём заключается проблема затухания градиента? Как можно её решить?
7. В чём заключается суть стохастического градиентного спуска и «mini batch» градиентного спуска? Назовите их преимущества и недостатки.

Список литературы

1. Bishop C. Pattern Recognition and Machine Learning. Springer, 2006.
2. Goodfellow I. J., Bengio Y., Courville A. Deep Learning. MIT Press, 2016.
3. Géron A. A Short Introduction to Entropy, Cross-Entropy and KL-Divergence // YouTube. 2018. URL: https://www.youtube.com/watch?v=ErfnhcEV1O8 (дата обращения: 17.09.2020)
4. Katanforoosh, Kunin et al. «Parameter optimization in neural networks», 2019. URL: http://deeplearning.ai/ai-notes/optimization/ (дата обращения: 17.09.2020)
5. Nielsen M. A. Neural Networks and Deep Learning. Determination Press, 2015.
6. Уоссермен Ф. Нейрокомпьютерная техника : Теория и практика. М.: Мир, 1992. 240 с.

7. Хайкин С. Нейронные сети: полный курс, 2-е издание.: Пер. с англ. М.: Издательский дом «Вильямс», 2006. 1104 с.