Глава 4. Анализ данных и машинное обучение

§ 1. Тема 28. Нейрон, персептрон и нейронная сеть

Ключевые слова:

- искусственный нейрон
- вес
- функция потерь
- слой
- смещение
- персептрон

Искусственный нейрон

В машинном обучении исследуемый объект, как правило, представляется в виде набора числовых признаков. Например, электронное письмо может быть описано длиной текста, количеством гиперссылок, наличием определённых подозрительных слов и характеристиками адреса отправителя. Изображение представляется совокупностью значений пикселей, отражающих яркость отдельных точек и интенсивность цветовых каналов.

Искусственный нейрон — это математическая модель, которая принимает на вход числовые признаки, обрабатывает их и формирует выходное значение.

Идея описывать нейрон как вычислительный элемент появилась ещё до современных компьютеров. В 1943 году Уоррен Маккаллок и Уолтер Питтс предложили математическую модель нейрона, который получает несколько сигналов и выдаёт ответ по принципу «сработал» или «не сработал». Их модель была очень упрощённой, но важной: она показала, что сеть таких элементов можно рассматривать как систему логических вычислений. Поэтому слово «нейрон» в искусственных нейронных сетях исторически связано не с точной копией мозга, а с идеей простого вычислительного блока, который принимает входы и

выдаёт результат. В учебной модели такое вычисление удобно разложить на несколько шагов.

Рассмотрим пример работы искусственного нейрона. Пусть он оценивает абитуриента по нескольким числовым признакам и принимает решение о его поступлении. Для каждого абитуриента известны числовые оценки его способностей в следующих областях:

- *математике (x_{math}),*
- *пении (x_{sing}),*
- *знании латинского языка (x_{latin}),*
- *дзюдо (x_{judo}),*
- *умении работать с компьютером (x_{comp}).*

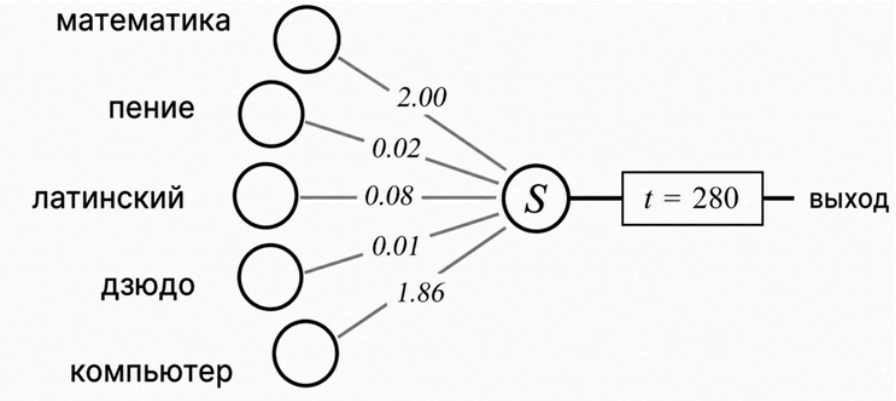


Рис. 28.1. Иллюстрация к задаче о поступлении в университет.

Все способности оцениваются от 0 до 100. С помощью весов мы можем показать модели, какие способности для поступления более важны, а какие менее.

Рассмотрим технический университет (рис. 28.1); веса зададим от 0 до 2. Для него математика важнее пения, поэтому пению присвоим небольшой вес ($w_{sing} = 0.02$), а математике большой ($w_{math} = 2.00$).

Теперь, если у абитуриента способности в пении на высоте (например, $x_{sing} = 100$), то на поступление это не сильно повлияет, поскольку вес у способности петь небольшой: $100 * 0.02 = 2$.

Вместе с тем, если у абитуриента отличные способности в математике (например, $x_{math} = 90$), то этот факт окажет существенное влияние на решение приёмной комиссии: $90 * 2 = 180$.

Аналогичным образом необходимо умножить способность студента в каждой сфере на соответствующий вес, а затем получившиеся числа сложить:

$$S = w_{\text{math}}x_{\text{math}} + w_{\text{sing}}x_{\text{sing}} + w_{\text{latin}}x_{\text{latin}} + w_{\text{judo}}x_{\text{judo}} + w_{\text{comp}}x_{\text{comp}}$$

Такая сумма, в которой сигналы перед сложением умножаются на веса, называется взвешенной.

Порог — это числовой параметр искусственного нейрона, задающий для взвешенной суммы входных сигналов границу, при превышении которой нейрон формирует единичный выходной сигнал.

При пороге t и взвешенной сумме входных сигналов S функция активации нейрона задаётся следующим правилом:

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

Здесь f — функция активации, а $f(S)$ — конкретное выходное значение нейрона.

Функция активации — это математическая функция, которая преобразует взвешенную сумму входных сигналов нейрона в его выходное значение.

В примере с абитуриентом порог равен $t = 280$. Поэтому при $S>280$ модель формирует положительное решение о поступлении, а при $S \leq 280$ отрицательное.

Качество модели определяется точностью её предсказаний. Чем лучше подобраны параметры модели, тем в большей степени предсказанные значения соответствуют целевым, а величина ошибки уменьшается. Процесс автоматической настройки параметров модели на основе данных называется обучением модели. Разработкой методов такого обучения занимается машинное обучение — область искусственного интеллекта, охватывающая нейронные сети и другие подходы к анализу данных и построению моделей.

Веса нейрона можно записать в вектор (см. параграф 24) $\mathbf{w}$:

$$\mathbf{w} = (w_{\text{math}}, w_{\text{sing}}, w_{\text{latin}}, w_{\text{judo}}, w_{\text{comp}})$$

а входные значения — в вектор $\mathbf{x}$:

$$\mathbf{x} = \left(x_{\text{math}}, x_{\text{sing}}, x_{\text{latin}}, x_{\text{judo}}, x_{\text{comp}}\right).$$

тогда взвешенную сумму S можно записать в виде скалярного произведения (см. параграф 24):

$$S = \mathbf{w} \cdot \mathbf{x}.$$

Для получения окончательного выхода нейрона, то есть его активации, к вычисленной сумме применяют пороговую функцию:

$$A(\mathbf{x}) = f(S) = f(\mathbf{w} \cdot \mathbf{x})$$

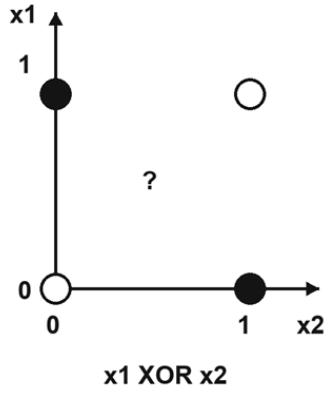
Итак, вся модель представляет собой функцию A(**x**). Её вид зависит от вектора весов **w** и порога t; это записывают так (слева от черты аргумент, справа параметры):

$$A(\mathbf{x}|\mathbf{w}, t) = f(\mathbf{w} \cdot \mathbf{x}).$$

Проблема XOR

Классическим примером ограничения работы искусственного нейрона служит операция XOR, исключающее ИЛИ. Для пар (0,0) и (1,1) ответ равен 0, а для пар (0,1) и (1,0) ответ равен 1. Никакая одна прямая на плоскости не может отделить две пары одного класса от двух пар другого класса.

Это называется линейной неразделимостью. Ограничения таких моделей подробно обсуждали Марвин Минский и Сеймур Пейперт в книге «Perceptrons» 1969 года. Они показали, что простые однослойные модели не решают некоторые наглядные задачи, связанные с линейной неразделимостью.



x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Рис. 28.2. Точки задачи XOR нельзя разделить одной прямой.

Слой нейронной сети — это совокупность нейронов, которые получают один и тот же вектор входных значений и формируют вектор выходных значений. Выход одного слоя может передаваться на вход следующего слоя.

Один из способов обойти это ограничение состоит в добавлении промежуточного слоя. Тогда несколько нейронов могут выделить промежуточные признаки, а следующий слой объединит их и построит более сложную границу решения. Однако долгое время главная трудность была не в самой идее многослойной сети, а в её обучении.

Построение нейронной сети

В нейронных сетях у каждого нейрона есть свой вектор весов **w**. Вектор весов нейрона скалярно перемножается с вектором выхода предыдущего слоя (или вектором входа, если предыдущего слоя нет).

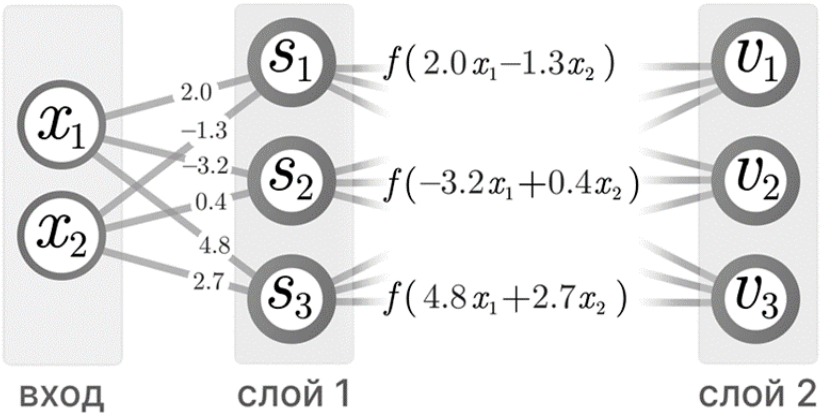


Рис. 28.3. Пример нейронной сети, состоящей из двух слоёв.

Обозначим вектор весов для нейрона s_1 за $\mathbf{w}_1$, s_2 за $\mathbf{w}_2$ и т.д. Вектор предыдущего слоя (в данном случае вектор входа) обозначим за **x**:

$$\mathbf{x} = (x_1, x_2)$$

Теперь активацию нейрона s_1 можно записать через скалярное произведение:

$$s_1 = f(\mathbf{w}_1 \cdot \mathbf{x})$$

Аналогичным образом можно записать активации и для оставшихся нейронов: $s_2 = f(\mathbf{w}_2 * \mathbf{x})$ и $s_3 = f(\mathbf{w}_3 * \mathbf{x})$. Теперь, чтобы посчитать активации следующего слоя (слоя 2), нужно построить вектор активаций текущего слоя (слоя 1):

$$\mathbf{s} = (s_1, s_2, s_3) = \big(f(\mathbf{w}_1 \cdot \mathbf{x}), f(\mathbf{w}_2 \cdot \mathbf{x}), f(\mathbf{w}_3 \cdot \mathbf{x})\big)$$

Обычно записывают вектор **s** следующим образом:

$$\mathbf{s} = f(\mathbf{w}_1 \cdot \mathbf{x}, \mathbf{w}_2 \cdot \mathbf{x}, \mathbf{w}_3 \cdot \mathbf{x})$$

то есть вместо того, чтобы независимо считать активации для каждого нейрона, сначала считают вектор скалярных произведений:

$$\mathbf{h} = (\mathbf{w}_1 \cdot \mathbf{x}, \mathbf{w}_2 \cdot \mathbf{x}, \mathbf{w}_3 \cdot \mathbf{x})$$

а затем к этому вектору применяют пороговую функцию:

$$\mathbf{s} = f(\mathbf{h})$$

Для этого пороговую функцию векторизуют — реализовывают так, чтобы пороговая функция автоматически применилась ко всем элементам вектора:

$$f(a, b, c) \mapsto \big(f(a), f(b), f(c)\big)$$

Векторизация позволяет значительно ускорить вычисления. Заметим, что **h** — это вектор скалярных произведений, а значит, можно записать матрицу весов W_1 (для слоя 1) следующим образом:

$$W_1 = \begin{bmatrix} — & \mathbf{w}_1 & — \\ — & \mathbf{w}_2 & — \\ — & \mathbf{w}_3 & —, \end{bmatrix}$$

тогда

$$\mathbf{s} = f(W_1\mathbf{x})$$

Вот и всё: мы записали прямую зависимость между s и x, то есть между предыдущим и текущим слоями. Такое преобразование *(про произведение матриц см. параграф 24)* позволяет сильно ускорить вычисления активаций.

Для примера на рис. 28.2 формула бы выглядела так:

$$\mathbf{s} = f\left(\begin{bmatrix} 2.0 & -1.3 \\ -3.2 & 0.4 \\ 4.8 & 2.7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$$

Теперь процесс обучения модели заключается в поиске оптимальной матрицы W_1 и порога t. Заметим существенную проблему: в нашей модели мы не можем для каждого нейрона

установить свой порог. Если мы установим какой-нибудь порог, он применится ко всем.

Пусть для пороговой функции установлен порог t. Тогда запишем функцию *f(x)*:

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

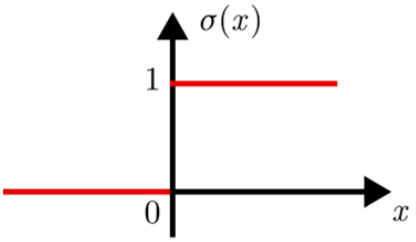


Рис. 28.4. График пороговой функции с порогом 0.

Однако такая запись равносильна следующей:

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

Пусть $\sigma(x)$ — пороговая функция с порогом 0. Тогда запись *f(x)* равносильна записи $\sigma(x - t)$. Таким образом, мы можем для каждого нейрона установить собственный порог следующим образом:

$$\mathbf{s} = \sigma\left(\begin{bmatrix} 2.0 & -1.3 \\ -3.2 & 0.4 \\ 4.8 & 2.7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -t_1 \\ -t_2 \\ -t_3 \end{bmatrix}\right)$$

где t_i — порог для i-го нейрона.

Прибавленный вектор констант называется вектором смещений. Смещение обозначим символом b_1 (от английского слова bias — «смещение»; индекс указывает на номер слоя — первый). Таким образом, выход слоя можно найти по следующей формуле:

$$\mathbf{s} = \sigma(W_1\mathbf{x} + \mathbf{b}_1),$$

где

W_1 — матрица весов первого слоя,

$\mathbf{b}_1$ — вектор смещений первого слоя,

x — выход предыдущего слоя (или вход модели),

$\sigma(x)$ — пороговая функция с порогом 0.

Аналогичным образом запишем выход слоя 2:

$$\mathbf{v} = \sigma(W_2\mathbf{s} + \mathbf{b}_2).$$

Вот мы и построили шаг за шагом прототип персептрона — многослойной нейронной сети.

Персептрон — это простейшая обучаемая модель искусственного нейрона, предназначенная для классификации объектов. Он принимает числовые входные признаки, умножает каждый на соответствующий вес, складывает произведения и сравнивает полученную взвешенную сумму с порогом. Если сумма превышает порог, персептрон выдаёт единицу, в противном случае ноль.

В 1958 году Розенблатт представил нейрокомпьютер MARK I Perceptron, в котором была реализована модель персептрона. Устройство могло распознавать некоторые английские буквы. Однако однослойный персептрон может строить только линейные границы решения и поэтому не справляется с задачами, в которых классы линейно неразделимы, например с рассмотренной выше проблемой XOR.

Градиентный спуск и метод обратного распространения ошибки.

Прежде чем описывать обучение, вернёмся к функции активации. Многослойную сеть мы записали с пороговой функцией. Она наглядна, но для обучения методом градиентного спуска не подходит: для применения этого метода необходимо вычислять производные функции потерь по параметрам сети, в том числе с учётом производной функции активации. У пороговой функции на горизонтальных участках производная равна нулю, а в точке скачка не существует. Поэтому при обучении пороговую функцию заменяют дифференцируемой функцией активации, например сигмоидой или гиперболическим тангенсом.

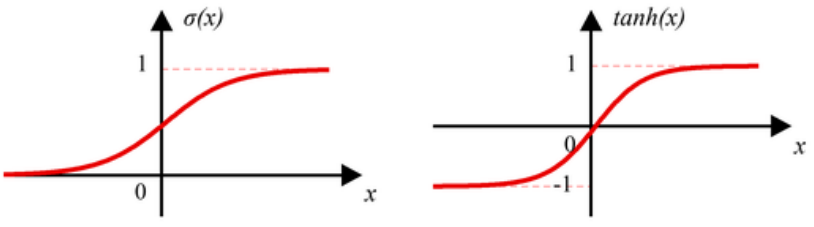


Рис. 28.5. Графики сигмоидальной функции и гиперболического тангенса.

Обучение модели заключается в поиске таких параметров, при которых она ошибается как можно реже или допускает лишь минимальные ошибки. Один из главных методов такой оптимизации, градиентный спуск (подробнее см. параграф 20), позволяет постепенно изменять параметры модели так, чтобы уменьшать значение функции потерь.

$$\boxed{\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla S(\boldsymbol{\theta}_k)}$$

Здесь Θ_k обозначает вектор параметров модели на k-й итерации, а Θ_{k+1} — вектор параметров после их обновления. Величина α определяет размер шага (скоростью обучения), а $\nabla S(\Theta_k)$ — градиент функции ошибки в текущей точке. Знак «минус» указывает, что параметры изменяются в направлении уменьшения значения функции S.

Для многослойных нейронных сетей одной идеи градиентного спуска недостаточно: значение функции потерь зависит от всех параметров сети, в том числе от параметров скрытых слоёв. Эту задачу решает метод обратного распространения ошибки, или backpropagation. Его математическая основа имеет более длинную историю и связана с эффективным вычислением производных сложных функций, заданных последовательностью вычислений. В советской математической школе близкие методы быстрого вычисления производных развивали К. В. Ким, Ю. Е. Нестеров, В. А. Скоков и Б. В. Черкасский. А в 1986 году Дэвид Румельхарт, Джеффри Хинтон и Рональд Уильямс показали, как применять этот подход к обучению многослойных нейронных сетей, после чего backpropagation стал одним из ключевых методов машинного обучения.

Метод обратного распространения ошибки, или backpropagation, — это алгоритм вычисления градиентов функции потерь по параметрам многослойной нейронной сети. Он позволяет определить, как изменение каждого веса и каждого смещения влияет на итоговую ошибку модели.

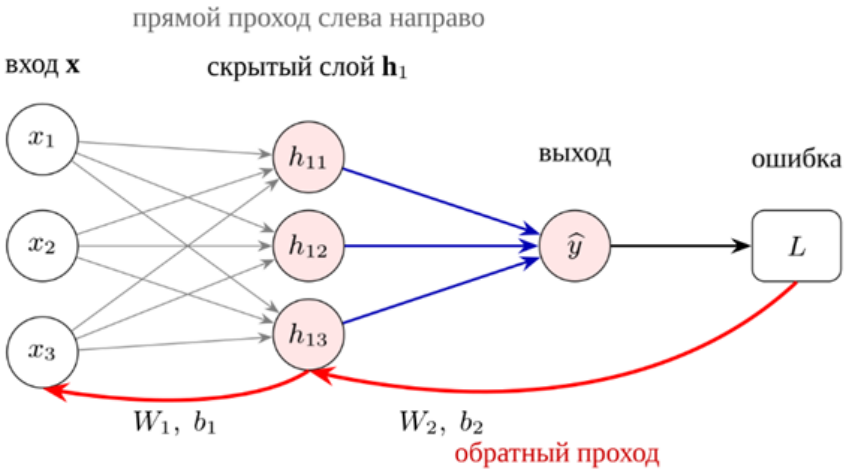


Рис. 28.6. На прямом проходе сеть вычисляет предсказание и значение функции потерь. На обратном проходе производные передаются справа налево.

Работа метода состоит из двух основных этапов. Сначала выполняется прямой проход: входные данные последовательно проходят через все слои сети, формируется предсказание модели, после чего вычисляется значение функции потерь. Затем выполняется обратный проход: производные функции потерь последовательно передаются от выходного слоя к предыдущим слоям. На каждом шаге применяется правило дифференцирования сложной функции, поэтому можно вычислить градиенты не только для последнего слоя, но и для скрытых слоёв.

Полученные градиенты затем используются алгоритмом оптимизации, например градиентным спуском. Параметры сети изменяются в направлении, в котором значение функции потерь уменьшается. Эта процедура многократно повторяется на обучающих примерах, благодаря чему модель постепенно подбирает веса и смещения, обеспечивающие более точные предсказания.

Эмбеддинг

Результатом обучения является обученная модель, способная решать поставленную задачу. При обработке входного объекта она последовательно преобразует его на разных слоях, формируя промежуточные представления. Представление, полученное на одном из последних слоёв, часто имеет вид вектора числовых значений.

Эмбеддинг — это векторное представление объекта, в котором его существенные для задачи свойства закодированы набором числовых значений.

Свёрточная нейронная сеть является разновидностью нейронных сетей, предназначенной для обработки изображений и выделения признаков в их локальных участках. При обработке изображения она постепенно преобразует значения пикселей в более компактные числовые представления. Вектор, полученный на одном из последних, часто предпоследнем, слоёв сети, называют эмбеддингом изображения. Он содержит признаки, важные для решаемой задачи, и может использоваться для поиска похожих изображений, обучения классификаторов для новых классов и сопоставления фотографий одного человека.

В обработке языка эмбеддинги используются для представления токенов, то есть слов или их частей, в виде числовых векторов. Токены, которые имеют похожее значение или встречаются в сходных контекстах, обычно получают близкие векторные представления. Такие векторы являются одним из базовых компонентов современных языковых моделей. Механизм внимания, архитектура трансформеров и применение эмбеддингов в языковых задачах будут подробнее рассмотрены в параграфе 29.

Сиамские сети и контрастивное обучение

Эмбеддинги особенно полезны в задачах сравнения объектов. Для построения векторных представлений, отражающих степень сходства между объектами, используют специальные архитектуры и методы обучения.

Классификатор определяет, к какому из заранее заданных классов относится объект. Однако во многих задачах требуется не классифицировать отдельный объект, а оценить степень сходства между двумя объектами. Например, необходимо установить, выполнены ли две подписи одним и тем же человеком, изображён ли на двух фотографиях один человек или является ли веб-страница дубликатом ранее проиндексированной страницы.

Однако прямое сведение такой задачи к классификации связано с рядом существенных ограничений. Если рассматривать каждого человека как отдельный класс, число классов может достигать миллионов. Кроме того, множество классов не является фиксированным: при добавлении нового человека классификатор пришлось бы переобучать или перестраивать. Наконец, для многих людей доступно лишь несколько фотографий, что недостаточно для формирования полноценной обучающей выборки.

Поэтому задачу ставят по-другому. Нужно построить отображение f, которое каждому объекту x ставит в соответствие вектор признаков $f(x)$, его эмбеддинг. Сеть обучают так, чтобы эмбеддинги похожих объектов оказывались близко друг к другу, а эмбеддинги непохожих далеко. Близость здесь измеряют евклидовым расстоянием, то есть длиной отрезка между двумя точками пространства:

$$d(\mathbf{x}, \mathbf{y}) = \|f(\mathbf{x}) - f(\mathbf{y})\|$$

Такой подход называют *метрическим обучением*. После обучения объект распознают, сравнивая его эмбеддинг с векторами из базы и выбирая ближайший. Чтобы добавить нового человека, достаточно записать в базу его эталонный вектор, не переобучая всю сеть.

Обучающая выборка состоит из пар объектов (x_i, y_i), каждой из которых сопоставлена бинарная метка t_i. Значение $t_i=1$ соответствует положительной паре, состоящей из похожих объектов, например двух фотографий одного человека. Значение $t_i=0$ соответствует отрицательной паре, состоящей из непохожих объектов, например фотографий разных людей.

Для обучения такой модели может быть использована сиамская нейронная сеть. Архитектура сети предложена в 1993 году в Bell Labs для автоматической проверки подписей (Бромли, Гийон, ЛеКун и др.). **Сиамская сеть** (Siamese network) состоит из двух копий одной и той же сети f с общими весами. Каждый объект пары проходит через свою копию, и по двум эмбеддингам вычисляется расстояние $d = \| f(\boldsymbol{x}) - f(\boldsymbol{y}) \|$ (рис. 28.7). Название дано по аналогии с сиамскими близнецами: копии не независимы, их связывают общие веса.

Общие веса здесь принципиальны. Разность эмбеддингов имеет смысл, только если оба вычислены одной и той же функцией; кроме того, так автоматически выполняется симметрия $d(x,y) = d(y,x)$. По существу это одна сеть f, применённая к двум входам.

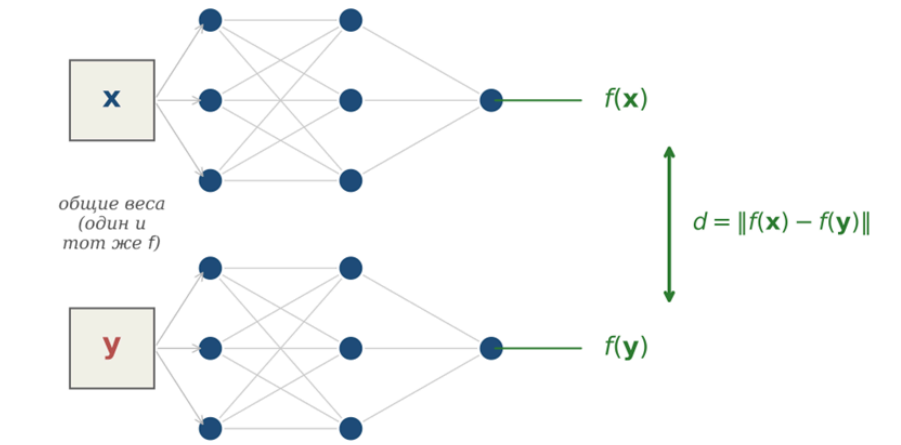


Рис. 28.7. Архитектура сиамской сети.

Пара объектов **x** и **y** обрабатывается двумя экземплярами одной и той же сети f с общими весами; на выходе вычисляется расстояние d= // *f(x)-f(y)* // между эмбеддингами. При обучении параметры f настраиваются так, чтобы d было малым для похожих пар (например, двух фотографий одного и того же человека) и большим для непохожих (фотографий разных людей).

Перейдем к контрастивной функций потерь. Укажем сразу, от чего зависит функция потерь. Пусть w — вектор параметров (весов) сети; будем писать f(x;w), подчёркивая зависимость выхода от весов. Тогда и расстояние в i-й паре зависит от w: $d_i(\mathbf{w}) = \|f(\mathbf{x}_i;\mathbf{w}) - f(\mathbf{y}_i;\mathbf{w})\|$. Функция потерь — функция именно аргумента w: выборка в ней фиксирована, а подстройке при обучении подлежат только веса.

Положительные пары нужно сближать: естественное слагаемое — квадрат расстояния d_i^2. Однако сумма $\sum_{i:t_i=1} d_i^2$ по одним положительным парам непригодна как функция потерь: у неё есть тривиальный глобальный минимум. Сеть, переводящая все объекты в одну и ту же точку, обнуляет все расстояния, а с ними и потери.

Такое вырождение называют коллапсом: формально минимум достигнут, но эмбеддинг, в котором любые два объекта неразличимы, бесполезен. Отсюда первый вывод: отрицательные пары обязаны входить в функцию потерь, и их расстояния нужно увеличивать.

Второй вывод даёт неудача прямолинейного способа. Слагаемое $-d_i^2$ для отрицательных пар делает функцию потерь неограниченной снизу: минимизация сведётся к бесконечному раздуванию расстояний. Требование к тому же избыточно: непохожим объектам незачем быть сколь угодно далёкими, достаточно отстоять друг от друга хотя бы на фиксированную величину.

Эту величину вводят как отступ (margin) — гиперпараметр $m > 0$. Отрицательная пара штрафуется, только если $d_i < m$: выражение $\max(0, m - d_i)$ положительно при $d_i < m$ и обращается в нуль при $d_i \geq m$.

Соединяя оба слагаемых, получаем контрастивную функцию потерь, предложенную Хадселл, Шопрой и ЛеКуном в 2006 году (ЛеКун — соавтор и работы 1993 года):

$$\mathcal{L}(\mathbf{w}) = \sum_i \left[t_i \cdot d_i^2(\mathbf{w}) + (1 - t_i) \cdot \left(\max\left(0, m - d_i(\mathbf{w})\right)\right)^2 \right].$$

Первое слагаемое отлично от нуля лишь при $t_i = 1$ и прижимает расстояния положительных пар к нулю; второе лишь при $t_i = 0$ и выталкивает отрицательные пары за границу m, после чего штраф исчезает. Квадраты дают гладкий штраф, растущий тем быстрее, чем сильнее нарушено требование, как в методе наименьших квадратов. Обучение — задача минимизации $\mathcal{L}(\mathbf{w}) \to \min_{\mathbf{w}}$; решается она методами типа градиентного спуска.

Функция называется контрастивной, потому что обучается на контрасте положительных и отрицательных пар, одновременно сближая первые и расталкивая вторые. Оба слагаемых необходимы: без второго минимум достигается коллапсом, без первого расстояния никак не связаны с похожестью.

Вопросы и задания

1. Из каких действий состоит вычисление искусственного нейрона?
2. Что показывает вес признака?
3. Зачем используется смещение?
4. Почему функция активации важна для нейронной сети?
5. Почему один персептрон не решает задачу XOR?
6. Почему обучение сети связано с функцией потерь?