Нейронные сети

основные модели

Воронеж
1999

УДК 612.8: 681.5
И. В. Заенцев
Нейронные сети: основные модели

Учебное пособие к курсу "Нейронные сети"
для студентов 5 курса магистратуры
к. электроники
физического ф-та
Воронежского Государственного университета

© Все права защищены.
Разрешается воспроизведение любой части данного пособия с обязательным уведомлением автора. Иван Витальевич Заенцев, 2:5025/2000@fidonet, e-mail: ivz@ivz.vrn.ru
394000, г. Воронеж, ул. Фр. Энгельса, д. 24, кв. 48. Тел. (0732) 52-71-37.

Если Вы заметили ошибки или опечатки, пожалуйста, сообщите автору и они будут исправлены.

Введение

Теория нейронных сетей включают широкий круг вопросов из разных областей науки: биофизики, математики, информатики, схемотехники и технологии. Поэтому понятие "нейронные сети" детально определить сложно.

Искусственные нейронные сети (НС) — совокупность моделей биологических нейронных сетей. Представляют собой сеть элементов — искусственных нейронов — связанных между собой синаптическими соединениями. Сеть обрабатывает входную информацию и в процессе изменения своего состояния во времени формирует совокупность выходных сигналов.

Работа сети состоит в преобразовании входных сигналов во времени, в результате чего меняется внутреннее состояние сети и формируются выходные воздействия. Обычно НС оперирует цифровыми, а не символьными величинами.

Большинство моделей НС требуют обучения. В общем случае, *обучение* — такой выбор параметров сети, при котором сеть лучше всего справляется с поставленной проблемой. Обучение — это задача многомерной оптимизации, и для ее решения существует множество алгоритмов.

Искусственные нейронные сети — набор математических и алгоритмических методов для решения широкого круга задач. Выделим характерные черты искусственных нейросетей как универсального инструмента для решения задач:

1. НС дают возможность лучше понять организацию нервной системы человека и животных на средних уровнях: память, обработка сенсорной информации, моторика.
2. НС — средство обработки информации:
   а) гибкая модель для нелинейной аппроксимации многомерных функций;
   б) средство прогнозирования во времени для процессов, зависящих от многих переменных;
   в) классификатор по многим признакам, дающий разбиение входного пространства на области;
   г) средство распознавания образов;
   д) инструмент для поиска по ассоциациям;
   г) модель для поиска закономерностей в массивах данных.
3. НС свободны от ограничений обычных компьютеров благодаря параллельной обработке и сильной связанности нейронов.
4. В перспективе НС должны помочь понять принципы, на которых построены высшие функции нервной системы: сознание, эмоции, мышление.

Существенную часть в теории нейронных сетей занимают биофизические проблемы. Для построения адекватной математической модели необходимо детально изучить работу биологических нервных клеток и сетей с точки зрения химии, физики, теории информации и синергетики. Должны быть известны ответы на основные вопросы, касающиеся

1. Как работает нервная клетка — биологический нейрон? Необходимо иметь математическую модель, адекватно описывающую информационные процессы в нейроне. Какие свойства нейрона важны при моделировании, а какие — нет?
2. Как передается информация через соединения между нейронами - синапсы? Как меняется проводимость синапса в зависимости от проходящих по нему сигналов?
3. По каким законам нейроны связаны друг с другом в сеть? Откуда нервная клетка знает, с какими соседями должно быть установлено соединение?

4. Как биологические нейронные сети обучаются решать задачи? Как выбираются параметры сети, чтобы давать правильные выходные сигналы? Какой выходной сигнал считается "правильным", а какой — ошибочным?

Важнейшие свойства биологических нейросетей:

1. Параллельность обработки информации. Каждый нейрон формирует свой выход только на основе своих входов и собственного внутреннего состояния под воздействием общих механизмов регуляции нервной системы.
2. Способность к полной обработке информации. Все известные человеку задачи решаются нейронными сетями. К этой группе свойств относятся ассоциативность (сеть может восстанавливать полный образ по его части), способность к классификации, обобщению, абстрагированию и множество других. Они до конца не систематизированы.
3. Самоорганизация. В процессе работы биологические НС самостоятельно, под воздействием внешней среды, обучаются решению разнообразных задач. Неизвестно никаких принципиальных ограничений на сложность задач, решаемых биологическими нейронными сетями. Нервная система сама формирует алгоритмы своей деятельности, уточняя и усложняя их в течение жизни. Человек пока не сумел создать систем, обладающих самоорганизацией и самоусложнением. Это свойство НС рождает множество вопросов. Ведь каждая замкнутая система в процессе развития упрощается, деградирует. Следовательно, подвод энергии к нейронной сети имеет принципиальное значение. Почему же среди всех диссипативных (рассеивающих энергию) нелинейных динамических систем только у живых существ, и, в частности, биологических нейросетей проявляется способность к усложнению? Какое принципиальное условие упущено человеком в попытках создать самоусложняющиеся системы?
4. Биологические НС являются аналоговыми системами. Информация поступает в сеть по большому количеству каналов и кодируется по пространственному принципу: вид информации определяется номером нервного волокна, по которому она передается. Амплитуда входного воздействия кодируется плотностью нервных импульсов, передаваемых по волокну.
5. Надежность. Биологические НС обладают фантастической надежностью: выход из строя даже 10% нейронов в нервной системе не прерывает ее работы. По сравнению с последовательными ЭВМ, основанными на принципах фон-Неймана, где сбой одной ячейки памяти или одного узла в аппаратуре приводит к краху системы.

Современные искусственные НС по сложности и "интеллекту" приближаются к нервной системе таракана, но уже сейчас демонстрируют ценные свойства:

1. Обучаемость. Выбрав одну из моделей НС, создав сеть и выполнив алгоритм обучения, мы можем обучить сеть решению задачи, которая ей по силам. Нет никаких гарантий, что это удастся сделать при выбранных сети, алгоритме и задаче, но если все сделано правильно, то обучение бывает успешным.
2. Способность к обобщению. После обучения сеть становится нечувствительной к малым изменениям входных сигналов (шуму или вариациям входных образов) и дает правильный результат на выходе.
3. Способность к абстрагированию. Если предъявить сети несколько искаженных вариантов входного образа, то сеть сама может создать на выходе идеальный образ, с которым она никогда не встречалась.

Параллельность обработки и реализуемость НС

Быстродействие современных ЭВМ составляет около 100 Mflops (flops - операция с плавающей запятой в секунду). В мозгу содержится примерно 10^11 нейронов. Время прохождения одного нервного импульса около 1 мс, и можно считать, что производительность одного нейрона порядка 10 flops. Эквивалентное быстродействие мозга составит 10^11 * 10 = 10^12 flops. Если рассмотреть задачи, решаемые мозгом, и подсчитать требуемое количество операций для их решения на обычных ЭВМ, то получим оценку быстродействия до 10^12..10^14 flops. Разница в производительности между обычной ЭВМ и мозгом — 4..6 порядков! Чем это объясняется?

Во многом этот выигрыш обусловлен параллельностью обработки информации в мозге. Следовательно, для повышения производительности ЭВМ необходимо перейти от принципов фон-Неймана к параллельной обработке информации. Тем не менее, параллельные компьютеры пока не получили распространения по нескольким причинам:

1. Тирания межсоединений. Каждый процессор в параллельной системе связан с большим количеством других. Количество связей занимает намного больший объем, чем сами процессоры. Такая плотность связей не реализуется в обычных интегральных схемах.
2. Трехмерность структуры связей между процессорами. Существуют различные типы связности процессоров в параллельной системе. Обычно требуются трехмерные связи. Технологически такие связи тоже пока невыполнимы.
3. Сложность программирования. Пока не создано единых способов программирования параллельных ЭВМ и средств для написания программ.

Несмотря на перспективность параллельных ЭВМ и, в частности, нейронных сетей, для их создания нет элементной базы. Поэтому, вместо моделирования НС на параллельных машинах, большая часть исследований проводится двумя способами:

1) моделирование НС на обычных последовательных ЭВМ;
2) создание специализированных нейроплат и нейропроцессоров для ускорения работы ЭВМ с нейронными сетями.

Первый способ дает проигрыш в быстродействии даже по сравнению с обычной ЭВМ, а второй способ не позволяет переходить от одной модели нейросети к другой, т.к. модель определяется используемой нейроплатой или нейропроцессором, и требуется сменить нейропроцессор, чтобы сменить модель.

Попытки использовать оптические, химические, биологические и другие технологии для создания НС, несмотря на перспективность, пока не имеют практического применения.

Место нейронных сетей среди других методов решения задач

Нейронные сети превосходят последовательные машины в решении тех же задач, в которых машину превосходит человек. Задачи, требующие большого объема вычислений или высокой точности лучше выполняются обычной ЭВМ.

К задачам, успешно решаемым НС на данном этапе их развития относятся:

— распознавание зрительных, слуховых образов; огромная область применения: от распознавания текста и целей на экране радара до систем голосового управления;
— ассоциативный поиск информации и создание ассоциативных моделей; синтез речи; формирование естественного языка;
— формирование моделей и различных нелинейных и трудно описываемых математически систем, прогнозирование развития этих систем во времени:
применение на производстве; прогнозирование развития циклонов и других природных процессов, прогнозирование изменений курсов валют и других финансовых процессов;

— системы управления и регулирования с предсказанием; управление роботами, другими сложными устройствами

— разнообразные конечные автоматы: системы массового обслуживания и коммутации, телекоммуникационные системы;

— принятие решений и диагностика, исключающие логический вывод; особенно в областях, где отсутствуют четкие математические модели: в медицине, криминалистике, финансовой сфере;

Уникальное свойство нейросетей — универсальность. Хотя почти для всех перечисленных задач существуют эффективные математические методы решения и несмотря на то, что НС проигрывают специализированным методам для конкретных задач, благодаря универсальности и перспективности для решения глобальных задач, например, построения ИИ и моделирования процесса мышления, они являются важным направлением исследования, требующим тщательного изучения.

Биологический нейрон

Как видно из предыдущего параграфа, моделирование биологических нейронных сетей обосновано и перспективно. Но для исследования НС необходимо иметь математическую модель биологического нейрона и биологической нейронной сети.

Центральная нервная система имеет клеточное строение. Единица – нервная клетка, нейрон. Нейрон имеет следующие основные свойства:

1. Участвует в обмене веществ и рассеивает энергию. Меняет внутреннее состояние с течением времени, реагирует на входные сигналы и формирует выходные воздействия и поэтому является активной динамической системой.

2. Имеет множество синапсов – контактов для передачи информации.

3. Нейрон взаимодействует путем обмена электрохимическими сигналами двух видов: электротоническими (с затуханием) и нервными импульсами (спайками), распространяющимися без затухания.

Существуют два подхода к созданию искусственных нейронных сетей. *Информационный подход*: безразлично, какие механизмы лежат в основе работы искусственных нейронных сетей, важно лишь, чтобы при решении задач информационные процессы в НС были подобны биологическим. *Биологический*: при моделировании важно полное биоподобие, и необходимо детально изучать работу биологического нейрона.

Крупные работы в исследованиях биологических нейронных сетей принадлежат Эндрю Хаксли, Алану Ходжкину, Бернарду Катцу, Джону Экклзу, Стивену Куффлеру.

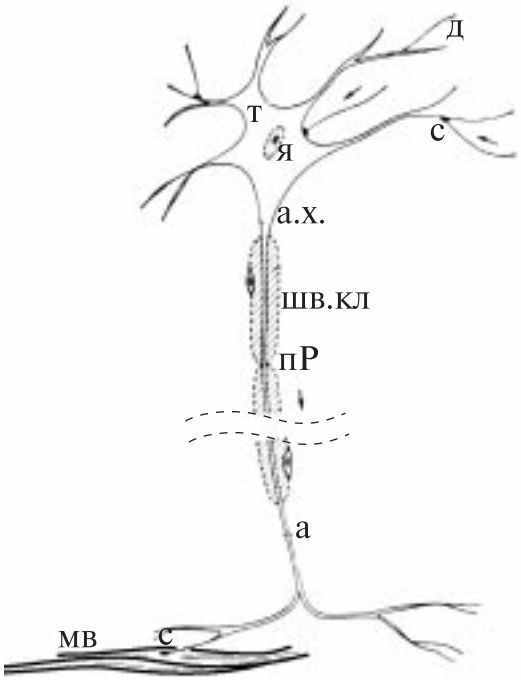


Рис. . Биологический нейрон.

Биологический нейрон содержит следующие структурные единицы:

Тело клетки **(т)** — *сома:* содержит ядро **(я),** митохондрии (обеспечивают клетку энергией), другие органеллы, поддерживающие жизнедеятельность клетки.

Дендриты **(д)** – входные волокна, собирают информацию от других нейронов. Активность в дендритах меняется плавно. Длина их обычно не больше 1 мм.

Мембрана – поддерживает постоянный состав цитоплазмы внутри клетки, обеспечивает проведение нервных импульсов.

Цитоплазма — внутренняя среда клетки. Отличается концентрацией ионов K^+, Na^+, Ca^{++} и других веществ по сравнению с внеклеточной средой.

Аксон **(а)**, один или ни одного у каждой клетки, – длинное, иногда больше метра, выходное нервное волокно клетки. Импульс генерируется в аксонном холмике (**а.х.**). Аксон обеспечивает проведение импульса и передачу воздействия на другие нейроны или мышечные волокна **(мв)**. Ближе к концу аксон часто ветвится.

Си́напс **(с)** – место контакта нервных волокон — передает возбуждение от клетки к клетке. Передача через синапс почти всегда однонаправленная. Различают пресинаптические и постсинаптические клетки — по направлению передачи импульса.

Шванновские клетки **(шв.кл).** Специфические клетки, почти целиком состоящие из миелина, органического изолирующего вещества. Плотно "обматывают" нервное волокно 250 слоями миелина. Неизолированные места нервного волокна между шванновскими клетками называются *перехватами Ранвье* **(пР)**. За счет миелиновой изоляции скорость распространения нервных импульсов возрастает в 5-10 раз и уменьшаются затраты энергии на проведение импульсов. Миелинизированные волокна встречаются только у высших животных.

В центральной нервной системе человека насчитывается от 100 до 1000 типов нервных клеток, в зависимости выбранной степени детализации. Они отличаются картиной дендритов, наличием и длиной аксона и распределением синапсов около клетки.

Клетки сильно связаны между собой. У нейрона может быть больше 1000 синапсов. Близкие по функциям клетки образуют скопления, шаровидные или параллельные слоистые. В мозгу выделены сотни скоплений. Кора головного мозга – тоже скопление. Толщина коры — 2 мм, площадь — около квадратного фута.

Нервный импульс

Нервный импульс (спайк) – процесс распространения возбуждения по аксону от тела клетки (аксонного холмика) до окончания аксона. Это основная единица информации, передаваемая по волокну, поэтому модель генерации и распространения нервных импульсов (НИ) — одна из важнейших в теории НС.

Импульсы по волокну передаются в виде скачков потенциала внутриклеточной среды по отношению к внешней среде, окружающей клетку. Скорость передачи – от 1 до 100 м/с. Для миелинизированных волокон скорость передачи примерно в 5 – 10 раз выше, чем для немиелинизированных.

При распространении форма спайка не меняется. Импульсы не затухают. Форма спайка фиксирована, определяется свойствами волокна и не зависит от того, каким способом создан импульс.

На рис. показаны нервные импульсы, возникающие в зрительном нервном волокне при воздействии вспышки света постоянной длительности для различных интенсивностей вспышки. Свет воздействовал на фоторецепторы и вызывал импульсацию в соответствующем зрительном волокне.

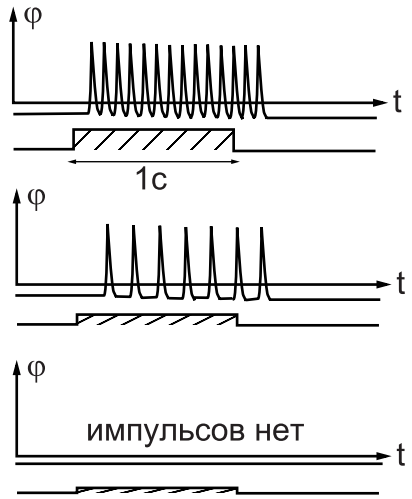


Рис. . Потенциал цитоплазмы относительно межклеточной среды в зрительном волокне при воздействии световой вспышки различной интенсивности.

Видно, что от интенсивности света зависит не амплитуда импульсов и их форма, а плотность и общее количество.

Для возбуждения и регистрации НИ можно использовать такую схему:

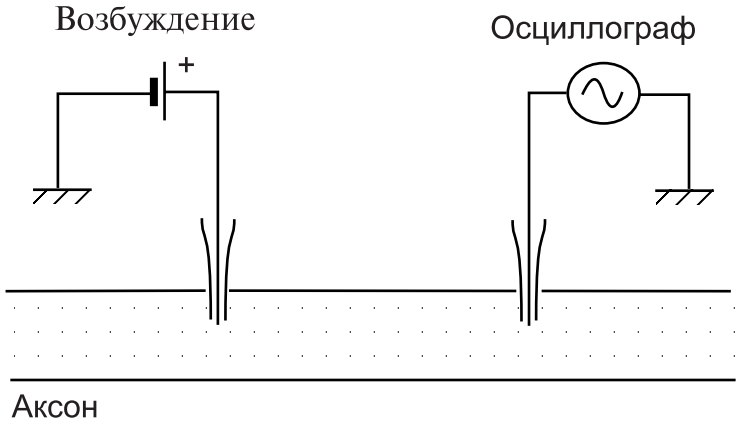


Рис. . Возбуждение и регистрация нервных импульсов.

Подавая на возбуждающий электрод электрические импульсы различной длительности и амплитуды, можно фиксировать возникающие при этом импульсы.

Оказалось, что зависимость минимального возбуждающего тока, при котором возникают нервные импульсы, от времени возбуждения имеет вид гиперболы:

$$I_{П} = \frac{a}{\Delta t} + b\,,$$

где $I_{П}$ — минимальный ток, при котором возникает импульсация при данной длительности Δt возбуждения. Данная формула — эмпирическая.

Параметры *a* и *b* имеют следующий смысл. *b* называется *реобазой* и задает минимальный ток возбуждения, при котором вообще может возникнуть импульсация. Если возбуждающий ток $I < b$, то импульсов не возникнет при любом Δt. Количество электричества, необходимое для возбуждения импульсов, при малых Δt примерно постоянно: $I_{П}\Delta t \approx a$.

Экспериментально было открыто свойство *рефрактерности*: после того, как по волокну прошел нервный импульс, в течение нескольких миллисекунд новые импульсы не возбуждаются при любом $I_{П}$. Поэтому между нервными импульсами всегда есть минимальный интервал времени — период рефрактерности.

Мембрана. Мембранный потенциал.

Клеточная мембрана создает и поддерживает постоянную концентрацию веществ внутри клетки, механическую прочность клетки и транспортировку молекул и ионов в обоих направлениях. Для нервных волокон она обеспечивает проведение нервных импульсов по волокну.

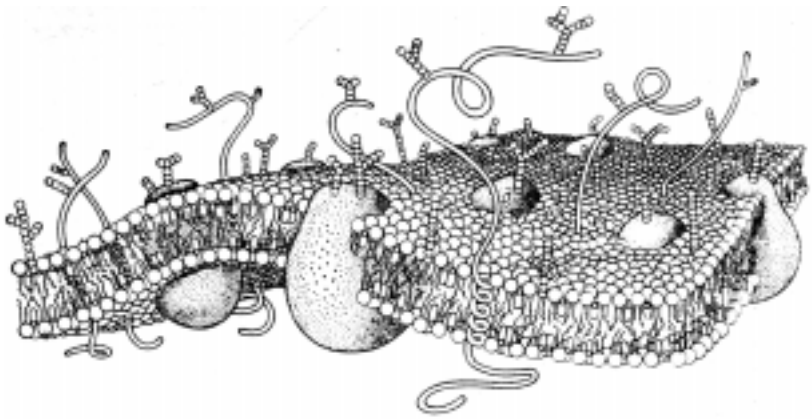


Рис. Клеточная мембрана.

Мембрана состоит из двух слоев молекул липидов (жиров). Молекулы липидов содержат полярную "голову" и два гидрофобных углеводородных "хвоста". Молекулы ориентированы "хвостами" внутрь мембраны. Такая конструкция оказывается стабильной и энергетически выгодной (по структуре она аналогична пленке мыльного пузыря). Молекулы белков взаимодействуют с такой двухслойной структурой. Белки с гидрофобными поверхностями взаимодействуют с "хвостами" липидов и встраиваются в саму мембрану, а с гидрофильными — соединяются с поверхностью мембраны. По химическому составу мембраны весьма разнообразны. Толщина мембраны около 10 нм.

В состав мембраны входит множество механизмов, необходимых для работы клетки. Мы будем рассматривать только те, что имеют отношение к передаче нервных сигналов.

Натриевый насос

Внутри клетки, в цитоплазме, преобладают ионы K^+, снаружи — Na^+. Активный мембранный транспорт, *натриевый насос*, переносит K^+ внутрь, а Na^+ — наружу. Такой перенос происходит в направлении роста электрохимического потенциала и требует затрат энергии. Для работы натриевого насоса используется энергия гидролиза АТФ (аденозинтрифосфата) — основного энергетического аккумулятора клетки. Механизм переноса использует белок-носитель, обозначим его С, образующий комплекс с ионами на одной стороне и отщепляющий эти ионы на противоположной стороне мембраны. Ни носитель, ни комплексы СК, СФNa с ионами Na^+, K^+: не покидают мембрану. В итоге через мембрану проходят потоки ионов K+ и Na+, направленные: K^+ — внутрь клетки, Na^+ — наружу.

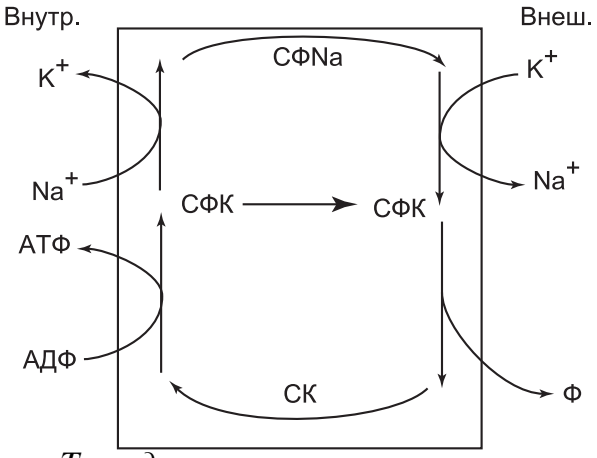


Рис. . Термодинамическая схема натриевого насоса.

В результате работы натриевого насоса концентрация K^+ и Na^+ становится неравновесной, но электрический потенциал мембраны не меняется, так как количество ионов K^+, перенесенных внутрь клетки, совпадает с количеством ионов Na^+, перенесенных наружу. Поэтому суммарный перенос заряда равен нулю и электрический потенциал цитоплазмы не меняется.

Калиевые каналы

В мембране существуют каналы, пропускающие только ионы K^+ в обоих направлениях. Каналы являются управляемыми, и могут открываться или закрываться в зависимости от разности потенциалов между цитоплазмой и внешней средой, а также обладают инерционными свойствами.

В состоянии покоя часть K-каналов открыта. Из-за разницы концентраций K^+ и Na^+ на разных сторонах мембраны ионы K начинают диффундировать через K-каналы из цитоплазмы наружу. Этот процесс приводит к оттоку положительного заряда из клетки, и цитоплазма заряжается отрицательно. Возникают кулоновские силы, препятствующие дальнейшей диффузии ионов K^+. Как только эти процессы уравновешивают друг друга, диффузия ионов K+ через K-каналы прекращается, и потенциал цитоплазмы достигает равновесного состояния -70 мВ.

В состоянии покоя разница потенциалов на мембране определяется формулой Ходжкина-Катца:

$$\varphi = \frac{RT}{F} \ln \frac{P_K c_K^i + P_{Na} c_{Na}^i + P_{Cl} c_{Cl}^e}{P_K c_K^e + P_{Na} c_{Na}^e + P_{Cl} c_{Cl}^i},$$

где P_K — проницаемость для мембраны для ионов K^+, P_K — концентрация K^+ внутри клетки, c_K^e — снаружи, аналогично для других ионов. Формула выводится, исходя из однородности поля на мембране, если диффузионные токи I_K и I_{Na} известны. Значение потенциала покоя -70..-80 мВ, полученное по этой формуле, согласуется с экспериментальными значениями.

Калиевые каналы открываются при изменении потенциала в положительную сторону и остаются открытыми, пока потенциал сохраняет свое значение.

Натриевые каналы

Na-каналы аналогичны калиевым, но пропускают только ионы Na^+. Отличаются также уровнем потенциала, открывающим канал, и инерционными характеристиками. В состоянии покоя Na-каналы закрыты. Натриевые каналы также открываются при изменении потенциала протоплазмы в положительную сторону. Закрываются сами по себе, через некоторое время после открытия. Закрывшись, находятся в состоянии рефрактерности в течение примерно 1 мс и не могут открыться снова до окончания рефрактерности.

Возникновение нервных импульсов

Механизмы мембраны, ответственные за возникновение нервных импульсов приведены на рис. .

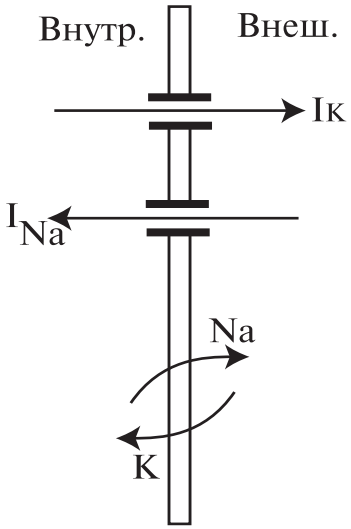


Рис. . Механизмы мембраны, участвующие в возникновении нервного импульса.

1. Возбуждение импульсов происходит по принципу "все или ничего". Если потенциал цитоплазмы станет положительнее, чем -50 мВ, то произойдет генерация импульса. Изменение потенциала можно проводить микроэлектродом, введенным в цитоплазму и подключенным к источнику напряжения.

Деполяризацией мембраны называется изменение ее разности потенциалов от состояния равновесия в положительную сторону, например, от -70 до -50 мВ.

2. Если мембрана деполяризуется до уровня -50 мВ, то открываются натриевые каналы, и поток ионов Na^+ начинает поступать в клетку. Возникает ток I_{Na}, направленный внутрь клетки. Это приводит к дальнейшей деполяризации мембраны. Т.к. Na-каналы управляются разностью потенциалов на мембране, дальнейшая деполяризация открывает все новые Na-каналы, что приводит к росту I_{Na}. Положительная обратная связь в данном процессе дает резкий скачок потенциала в сторону деполяризации.

3. Натриевые каналы закрываются самопроизвольно, через некоторое время после открытия. Закрывшись, находятся в состоянии рефрактерности около 1 мс и не могут быть открыты вновь, несмотря на деполяризованность мембраны. Ток I_{Na} падает до нуля, деполяризация прекращается. В этот момент потенциал достигает значения +40 мВ.

4. Калиевые каналы тоже управляются потенциалом, но более инертны и открываются с задержкой по сравнению с натриевыми. Т.к. потенциал смещен в сторону деполяризации, открытие калиевых каналов приводит в диффузионному калиевому току, направленному из клетки. Ток I_K нарастает медленно и направлен противоположно I_{Na}. К моменту, когда I_K достигает максимума, ток I_{Na} уменьшается, и потенциал начинает меняться в противоположную сторону: мембрана *реполяризуется*.

5. За счет тока I_K мембрана реполяризуется до исходного потенциала -70 мВ. За счет инерционности К-каналов мембрана *гиперполяризуется* до -90 мВ. Калиевый ток прекращается.

6. По окончании импульса натриевые каналы пребывают в состоянии рефрактерности около 1 мс, когда возникновение нового импульса невозможно. Происходит деполяризация до –70..-80мВ — равновесного мембранного потенциала.

7. Под действием изменения потенциала открываются натриевые каналы на соседнем участке, и процесс возбуждения распространяется вдоль волокна.

Зависимости токов I_K, I_{Na} и мембранного потенциала φ от времени приведены на рис. .

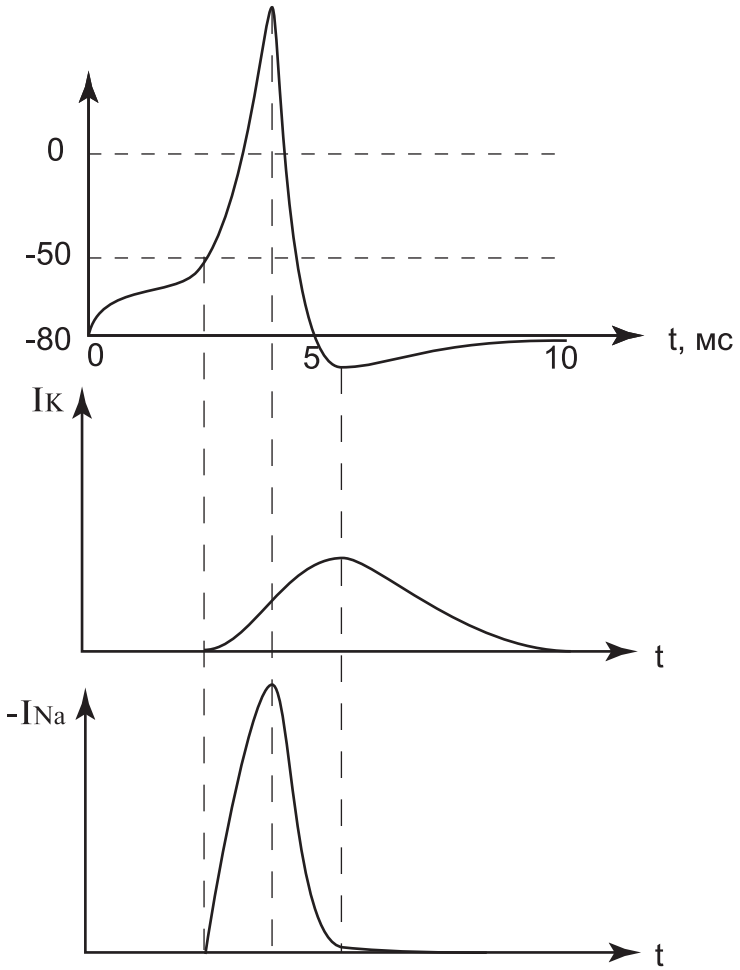


Рис. . Зависимость ионных токов и мембранного потенциала от времени.

В результате прохождения импульса часть ионов калия выходит наружу, а часть ионов натрия входит в клетку, причем в итоге потенциал возвращается к исходному значению –80 мВ. Следовательно, количество ионов Na, вошедших в клетку, в точности равно количеству ионов K, вышедших из клетки.

Натриевый насос восстанавливает разницу концентраций очень быстро. Так как емкость мембраны мала (C=1 мкФ/кв.см),то суммарное количество ионов, прошедших через мембрану, невелико. Выход ионов каждого вида при прохождении импульса составляет $3{,}7 \cdot 10^{-12}$ моль/см^2. Восстановление концентраций требует затрат энергии, которая берется из реакции $АТФ \rightarrow АДФ$.

Сальтаторный механизм распространения импульса

Сальтаторный механизм встречается для миелинизированных волокон. В этом случае участки волокна, покрытые миелином (шванновскими клетками) не проводят ток, зато в неизолированных местах, перехватах Ранвье, плотность ионного тока возрастает в 10 раз по сравнению с волокнами, лишенными миелиновой оболочки (рис.).

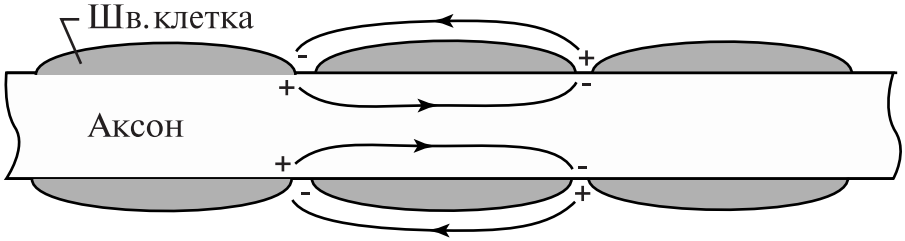


Рис. . Сальтаторный механизм распространения

Распространение импульса состоит в последовательном возбуждении перехватов Ранвье. Импульс передается "скачками" от одного перехвата к следующему. Так как большая часть волокна изолирована, то суммарный обмен ионами намного меньше, чем для немиелинизированных волокон. Скорость передачи возрастает в 10-50 раз, а энергии расходуется меньше. Для немиелинизированных

волокон кошки скорость передачи — 0,7-2,3 м/с, а для миелинизированных — 50-100 м/с. Поэтому сальтаторный механизм часто оказывается более эффективным. Такой вид распространения НИ встречается только у высших животных.

Эквивалентная схема волокна

Нервное волокно можно представить в виде непрерывного кабеля, составленного из элементов:

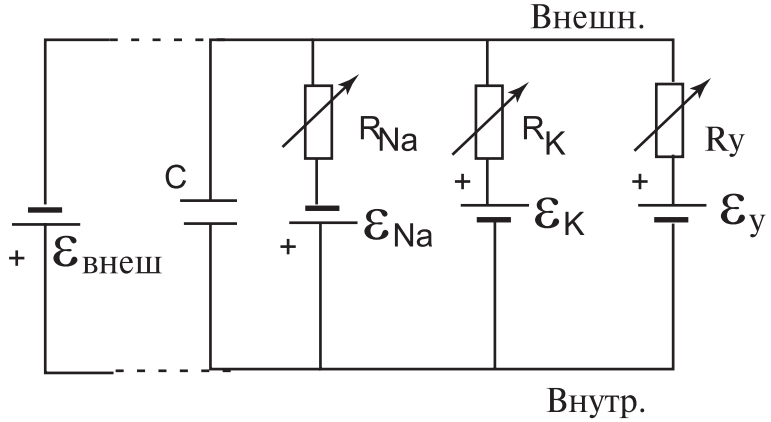


Рис. . Эквивалентная схема участка нервного волокна.

C — емкость единицы площади мембраны;

ε_{Na} — ЭДС, возникающая за счет разности концентраций ионов Na снаружи и внутри клетки;

$R_{Na}(\varphi,t)$ — сопротивление натриевых каналов в зависимости от потенциала мембраны и времени, приходящееся на единицу площади мембраны.

Аналогично вводятся величины для калиевых каналов с индексом K. Величины с индексом "y" соответствуют ЭДС и проводимости мембраны для прочих ионов (Cl, Ca).

Распространение нервных импульсов. Уравнение Ходжкина-Хаксли

Рассмотрим эквивалентную схему участка волокна. Общая формула для плотности мембранного тока:

$$I_{внеш} = C\dot{\varphi} + I_i \qquad (1)$$

где $C\dot{\varphi}$ — плотность тока за счет изменения концентрации ионов по обеим сторонам мембраны; дальше под "током" будем понимать поверхностную плотность тока;

$I_i = I_K + I_{Na} + I_y$ — ток за счет диффузии ионов трех видов (K^+, Na^+ и прочие ионы) через мембрану;

Компоненты ионного тока можно записать через проводимости g_K, g_{Na}, g_y и потенциалы покоя, создаваемые ионами соответствующего вида, φ_K, φ_{Na}, φ_y:

$$I_K = g_K(\varphi - \varphi_K), \quad I_{Na} = g_{Na}(\varphi - \varphi_{Na}), \quad I_y = g_y(\varphi - \varphi_y) \qquad (2)$$

Экспериментально были получены значения потенциалов $\varphi_K = -12\,мВ$, $\varphi_{Na} = +115\,мВ$.

В модели Ходжкина-Хаксли считается, что K-канал открывается, если к данному участку мембраны подходит четыре управляющих частицы одного типа. Пусть n — вероятность подхода одной частицы, тогда n^4 — вероятность подхода четырех частиц одновременно. Тогда удельную проводимость K-канала можно записать в виде:

$$g_K = \bar{g}_K n^4 \qquad (3)$$

Здесь $\bar{g}_K$ — максимальная проводимость K-канала. Аналогично, Na-канал открывается, если подходит три активирующих частицы одного типа и уходит одна блокирующая. Пусть m — вероятность подхода одной активирующей частицы, h — удаления одной блокирующей. Тогда:

$$g_{Na} = \bar{g}_{Na} m^3 h \qquad (4)$$

Вероятности n, m, h удовлетворяют кинетическим уравнениям:

$$\begin{cases} \dot{n} = \alpha_n (1-n) - \beta_n n \\ \dot{m} = \alpha_m (1-m) - \beta_m m \\ \dot{h} = \alpha_h (1-h) - \beta_h h \end{cases} \qquad (5)$$

Коэффициенты α, β зависят от φ. С ростом φ в сторону деполяризации $\alpha_n, \alpha_m, \beta_h$ — растут, $\beta_n, \beta_m, \alpha_h$ — убывают. Если потенциал φ не меняется, то α, β не меняются и n, m, h экспоненциально зависят от времени.

Подставим (3), (4) в (2):

$$\begin{aligned} I_{Na} &= \bar{g}_{Na} m^3 h (\varphi - \varphi_{Na}) \\ I_K &= \bar{g}_K n^4 (\varphi - \varphi_K) \\ I_y &= g_y (\varphi - \varphi_y) \end{aligned} \qquad (6)$$

Подставим (6) в (1) и получим уравнение для полного мембранного тока:

$$I_{внеш} = C\dot{\varphi} + \bar{g}_K n^4 (\varphi - \varphi_K) + \bar{g}_{Na} m^3 h (\varphi - \varphi_{Na}) + g_y \qquad (7)$$

По уравнениям (5) и (7) можно провести численное моделирование и получить зависимость потенциала в данной точке аксона от времени. Все константы в формулах должны быть предварительно определены из опытов над реальными волокнами.

Деполяризуем мембрану подключением внешнего источника ЭДС $\varepsilon_{внеш}$ (см. схему). После отключения внешнего источника в момент t=0 $I_{внеш}$=0. Система (5) + (7), представляет собой систему 4-х дифференциальных уравнений 1-й степени. Она решается численно. Находим зависимости $n(t)$, $m(t)$, $h(t)$, $\varphi(t)$. Полученные решения хорошо согласуются с опытом над биологическими нервными волокнами. Система (5) + (7) называется *канонической моделью электрогенеза нервной клетки*.

Пространственное описание нервного импульса

Рассмотрим участок волокна длиной l и радиусом a (рис.).

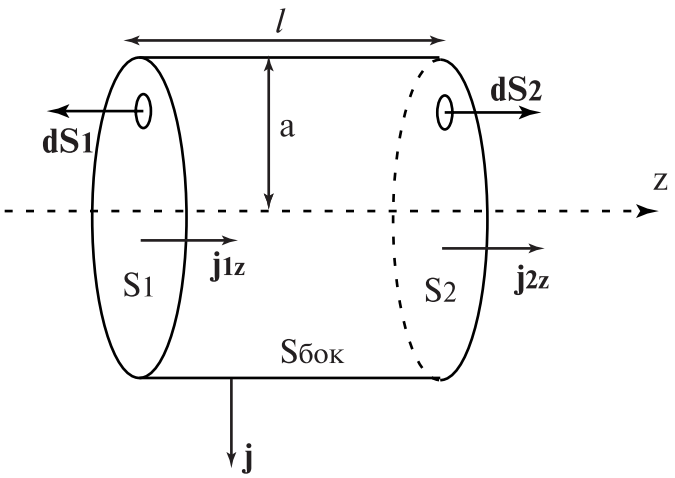


Рис. . Цилиндрический участок волокна

Вектор плотности тока связан с потенциалом:

$$\mathbf{j} = -g \operatorname{grad} \varphi$$

В цилиндрических координатах компоненты градиента:

$$\operatorname{grad} \varphi = \left\{ \frac{\partial \varphi}{\partial \rho}, \frac{1}{\rho} \frac{\partial \varphi}{\partial \Phi}, \frac{\partial \varphi}{\partial z} \right\}$$

Полный ток через участок в отсутствие $\mathbf{j}_{внеш}$ равен нулю: $\oint \mathbf{j} d\mathbf{S} = 0$.

Распишем компоненты интеграла: $-S_1 j_{1z} + S_2 j_{2z} + S_{бок} j = 0$, $S_1 j_{1z} - S_2 j_{2z} - S_{бок} j = 0$.

Площадь боковой поверхности цилиндра: $S_{бок} = 2\pi a l$, торцевой: $S_1 = \pi a^2$. Будем считать нормальную составляющую плотности тока на торцевой поверхности постоянной: $j_{1z} \approx const = -\left.\frac{\partial \varphi}{\partial z}\right|_{S_1}$. Ее значение на второй торцевой поверхности получим, разложив плотность тока в ряд Тейлора:

$$j_{2z} = j_{1z} + l\frac{\partial j_z}{\partial z} = -\frac{\partial \varphi}{\partial z} - l\frac{\partial^2 \varphi}{\partial z^2}$$

В итоге получим:

$$\pi a^2 g_a \left(-\frac{\partial \varphi}{\partial z} + \left(\frac{\partial \varphi}{\partial z} + \frac{\partial^2 \varphi}{\partial z^2} \right) l \right) = 2\pi a l j$$

После сокращения двух слагаемых:

$$\pi a^2 g_a l \frac{\partial^2 \varphi}{\partial z^2} = 2\pi a l j \qquad (8)$$

Нервный импульс распространяется в виде плоской волны с постоянной скоростью, поэтому справедливо волновое уравнение:

$$\frac{\partial^2 \varphi}{\partial z^2} = \frac{1}{v^2}\frac{\partial^2 \varphi}{\partial t^2} \qquad (9)$$

Итак, для плотности тока справедливо уравнение электрогенеза:

$$\frac{a g_a}{2v^2}\ddot{\varphi} = C\dot{\varphi} + (\varphi - \varphi_K)\bar{g}_K n^4 + (\varphi - \varphi_{Na})\bar{g}_{Na} m^3 h + \qquad (10)$$

— это уравнение называется *уравнением Ходжкина-Хаксли.*

Уравнения (10) и (5) образуют систему 5-го порядка, решаемую численно. Существуют упрощенные модели для решения этой системы, которые дают хорошую точность. Числовое решение имеет вид

Рис. . Зависимость потенциала от времени для уравнения Ходжкина-Хаксли.

Синаптическая передача

Си́напс – соединение для передачи нервного импульса от нейрона к нейрону или от нейрона к мышечному волокну. Синапсы бывают химические и электрические, в центральной нервной системе преобладают химические. В месте контакта мембраны клеток не сливаются, между ними всегда существует небольшой промежуток — *синаптическая щель.*

Электрический синапс: ширина щели — 2-4 нм (при толщине мембраны 7-10нм). Между контактирующими мембранами образуются две системы каналов: цитоплазма – внешняя среда и цитоплазма – цитоплазма. Первый тип каналов регулирует обмен ионов калия , натрия и хлора. Межклеточные каналы второго типа имеют низкую утечку во внешнюю среду и передают импульсы с использованием того же механизма, что и при передаче импульса по волокну. Задержка передачи для

электрических синапсов очень мала. Недостаток электрических синапсов — нерегулируемость: они не реагируют на биологически активные вещества и не меняют свою проводимость. Встречаются у беспозвоночных и низших позвоночных.

В мозгу человека и высших животных преобладают *химические синапсы*. У одного нейрона бывает от 300 до 20 000 синапсов между аксонами, аксонами – дендритами, дендритами – дендритами, аксонами и мышечными волокнами, аксонами и телом клетки и т.п.

Схема химического синапса представлена на рис. .

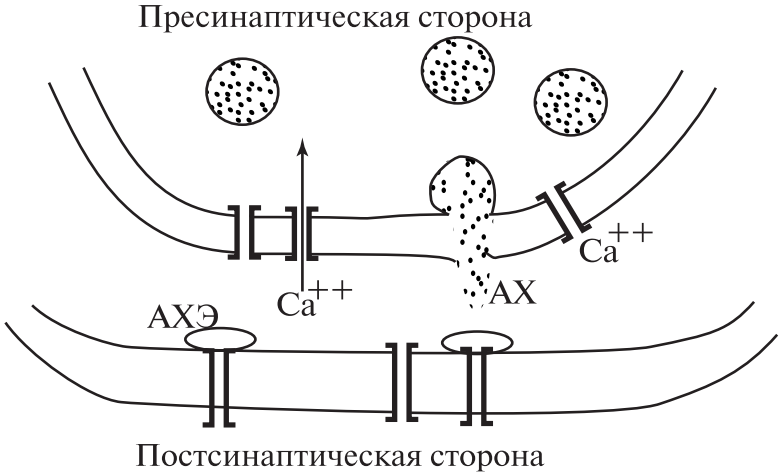


Рис. . Химический синапс.

Порядок синаптической передачи.

1. Импульс подходит к пресинаптической части волокна.
2. Открываются кальциевые каналы, ионы Ca^{++} поступают в пресинаптическую цитоплазму.
3. В пресинаптической цитоплазме постоянно находится большое количество *синаптических пузырьков* — образований, содержащих молекулы вещества — *медиатора*. Самым распространенным медиатором является ацетилхолин (АХ), кроме него существует около 20 других видов медиаторов.

Вследствие появления ионов Ca^{++} в пресинаптической цитоплазме посредством неизвестного пока механизма пузырьки подходят к мембране и лопаются, сливая медиатор в синаптическую щель. Пузырьки выбрасываются не по одному, а квантами по несколько пузырьков. За один квант в щель попадает $10^3 – 10^4$ молекул медиатора. За один импульс проходит 100 – 200 квантов медиатора. Даже в отсутствие нервных импульсов каждую секунду выбрасывается несколько квантов медиатора, и синапс поддерживается в состоянии готовности к передаче.

4. Попавший в щель медиатор диффундирует на постсинаптическую сторону щели. Этот процесс требует около 0,5мс и вносит существенный вклад в задержку передачи.
5. Молекулы медиатора улавливаются рецепторами на постсинаптической стороне:

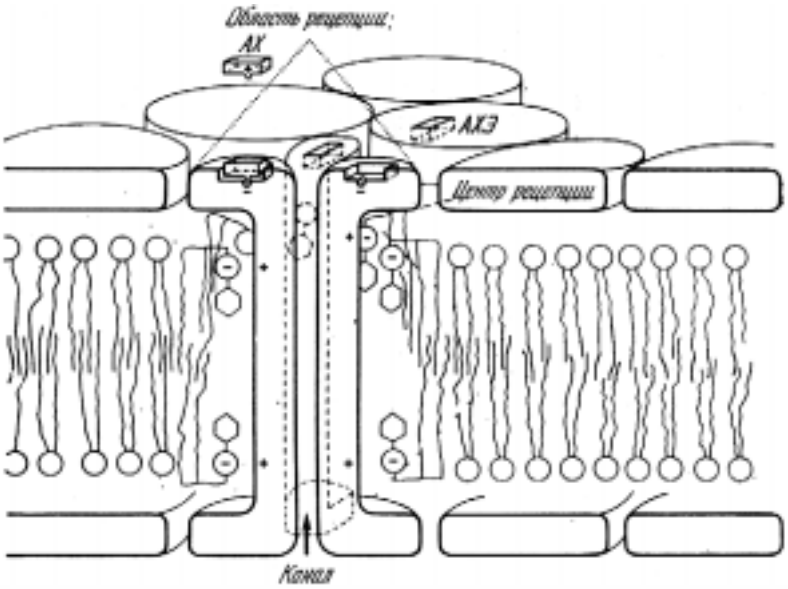


Рис. . Постсинаптическая мембрана.

6. Попадание ацетилхолина на рецептор увеличивает проводимость канала в обоих направлениях для ионов Na^+ и K^+ в равной степени. Это эквивалентно небольшому кратковременному "проколу" мембраны. Ацетилхолин, попавший в рецепторы, гидролизуется ацетилхолинэстеразой (АХЭ), во избежание забивания областей рецепции молекулами АХ. Яд кураре и подобные ему вещества попадают в молекулы АХЭ, предотвращают гидролиз АХ и останавливают работу синапсов.

7. Открытие каналов на постсинаптической стороне вызывает поток ионов натрия внутрь, а калия наружу. Возникший ионный ток возбуждает нервный импульс, который распространяется дальше по постсинаптическому волокну.

Существуют медиаторы, которые работают по-другому. Так как контакты бывают дендро-дендритные, дендро-аксональные, то синапсы оказываются разнообразными по структуре.

Каналы являются специфическими для различных ионов (калия, натрия, хлора). В зависимости от вида ионов, постсинаптическая мембрана или гиперполяризуется при прохождении соответствующих ионов через канал (тормозный синапс), или деполяризуется (возбуждающий синапс).

Синапсы образуются в различных участках клетки. Информационные функции клетки определяются расположением этих участков и их влиянием на мембрану и могут быть очень разными. Например, в зоне аксонного холмика, где обычно генерируется импульс, деполяризация мембраны с наибольшей вероятностью вызовет возникновение нервного импульса.

Отдельный нервный импульс слабо влияет на постсинаптическую клетку и не является самостоятельным носителем информации. Для существенного возбуждения необходимо поступление пачки импульсов. Плотность импульсов в пачке и распределение плотности во времени является носителем информации в нервной системе. Пачка нервных импульсов является аналоговым сигналом, т.к. плотность импульсов в пачке может меняться непрерывно.

Подпороговое возбуждение может распространяться градуально, с затуханием. В коротких дендритах используется этот механизм. Градуальные синаптические потенциалы тоже являются аналоговыми сигналами. Поэтому нервная система человека и животных оказывается *аналоговой* информационной системой.

Генерация нервных импульсов

Вольтамперная характеристика нервного волокна нелинейна и имеет участок с отрицательным сопротивлением. Рассмотрим кусочно-линейную аппроксимацию ВАХ волокна и покажем, что даже в такой простой модели могут возникать импульсы, похожие на реальные.

При возбуждении нервного импульса калиевый ток I_K растет заметно медленнее, чем I_{Na}. Будем рассматривать малую длительность возбуждения, когда можно пренебречь током I_K.

Экспериментальная вольтамперная характеристика нервного волокна приведена на рис. . (пунктирная линия). Будем аппроксимировать ее тремя линейными участками с различной дифференциальной проводимостью.

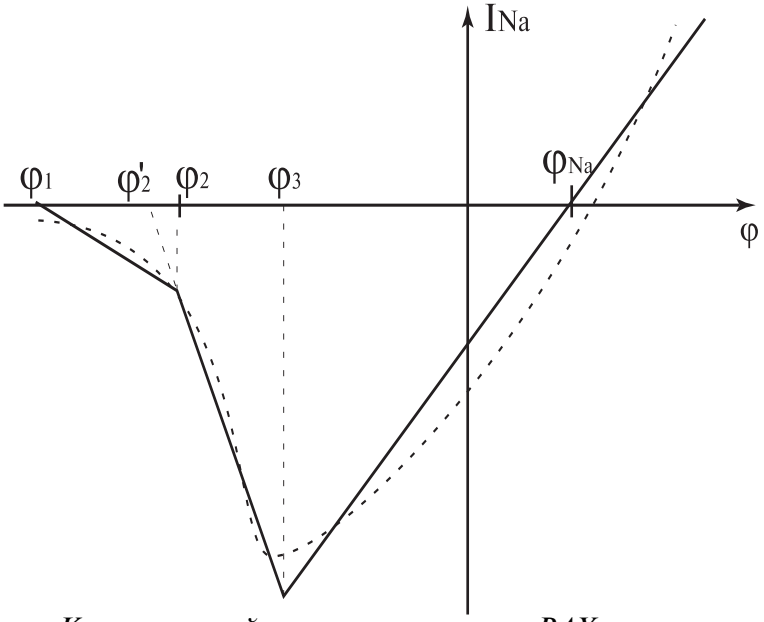


Рис. . Кусочно-линейная аппроксимация ВАХ нервного волокна.

Так как калиевый ток растет медленно, то на начальном участке ток через мембрану $I \approx I_{Na} + I_y$, где I_y — плотность тока утечки.

По закону Ома $I_y = (\varphi - \varphi_0) g$, где g — проводимость мембраны, φ_0 — потенциал покоя.

Аппроксимируем I_{Na} прямоугольной ямой: $I_{Na} = \text{const}$ от точки максимума до спада, в остальное время $I_{Na} = 0$.

Для кусочно-линейной аппроксимации ВАХ:

$$\frac{dI_{Na}}{d\varphi} = \begin{cases} -g_1, & \varphi < \varphi_2 \\ -g_2, & \varphi_2 \le \varphi < \varphi_{Na} \\ g_2, & \varphi > \varphi_{Na} \end{cases} \qquad I_{Na} = \begin{cases} -g_1(\varphi - \varphi_1), & \varphi < \varphi_2 \\ -g_2(\varphi - \varphi_2'), & \varphi_2 \le \varphi < \varphi_3 \\ g_2(\varphi - \varphi_{Na}), & \varphi > \varphi_3 \end{cases}$$

Пусть проводимость лежит в пределах: $g_1 < g < g_2$. Приложим внешнее возбуждение $\tilde{\varphi}$. Его ток составит $I = g\tilde{\varphi}$. Рассмотрим различные случаи возбуждения $\tilde{\varphi}$.

1. Случай $\tilde{\varphi} < \varphi_1$. Здесь $I_{Na} = 0$. Мембранный ток: $I_m = C\frac{d\varphi}{dt} + I_y$.

Система будет описываться уравнением:

$$C\frac{d\varphi}{dt} = g(\tilde{\varphi} + \varphi_0 - \varphi).$$

Решение: $\varphi - \varphi_0 = \tilde{\varphi}\left(1 - \exp\left(-\frac{gt}{C}\right)\right)$ (1)

— потенциал экспоненциально нарастает до $\tilde{\varphi}$.

Пусть возбуждение отключается при $t = t_0$. Имеем уравнение:

$$C\frac{d\varphi}{dt} = g(\varphi_0 - \varphi)$$

С начальным условием: $\varphi(t_0) = \varphi_0 + \tilde{\varphi}\left(1 - \exp\left(-\frac{gt_0}{C}\right)\right)$.

Решение экспоненциально убывает с течением времени:

$$\varphi - \varphi_0 = \tilde{\varphi}\left[\exp\left(\frac{gt_0}{C}\right) - 1\right]\exp\left(-\frac{gt}{C}\right)$$

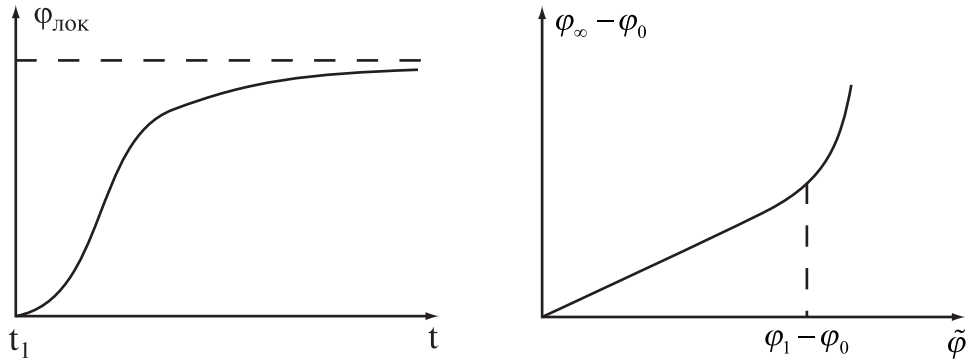
2. Случай $\tilde{\varphi} > \varphi_1$. Наступает момент, когда потенциал достигает значения $\varphi = \varphi_1$. Появляется ток I_{Na}. Дифференциальное уравнение, описывающее систему:

$$C\frac{d\varphi}{dt} = g\tilde{\varphi} - g(\varphi - \varphi_0) + g_1(\varphi - \varphi_1)$$

Решение: $\varphi - \varphi_0 = \tilde{\varphi} + \frac{\tilde{\varphi} - (\varphi_1 - \varphi_0)}{g - g_1}\left(g_1 - g\exp\left(-\frac{g - g_1}{C}(t -\right.\right.$ (2)

Наблюдается экспоненциальный рост $\varphi - \varphi_0$ до значения $\varphi_\infty - \varphi_0 = \tilde{\varphi} + \frac{g_1}{g - g_1}\left(\tilde{\varphi} - (\varphi_1 - \varphi_0)\right)$.

Вклад, вносимый натриевым током I_{Na} в потенциал φ (*локальный ответ*) определяется разностью (2) и (1):



а) б)

Рис. . а) Зависимость локального ответа от времени.
б) Сдвиг мембранного потенциала в зависимости от уровня возбуждения.

Благодаря отрицательной проводимости появляется ток через мембрану. Но при выполнении условия $g > g_1$ импульсов не возникает.

3. Случай $\varphi_\infty - \varphi_0 > \varphi_2$. Здесь при некотором значении $t = t_2$ потенциал достигает значения φ_2 и натриевый ток определяется проводимостью $-g_2$, $g_2 > g$. При $t \geq t_2$ поведение мембраны будет определяться уравнением:

$$C\frac{d\varphi}{dt} = g\tilde{\varphi} - g(\varphi - \varphi_0) + g_2(\varphi - \varphi_2)$$

Решение имеет вид:

$$\varphi = -\frac{g}{g_2 - g}(\tilde{\varphi} + \varphi_0) + \frac{g_2}{g_2 - g}\varphi_2 + \left(\varphi(t_2) + \frac{g}{g_2 - g}\right.$$

Показатель экспоненты больше нуля. Если $\tilde{\varphi} > \varphi_2 - \varphi_0$, то перед экспонентой стоит положительное число, и ток нарастает экспоненциально то некоторого момента t_3, когда натриевый ток отключается самопроизвольно из-за свойств мембраны. Рис. .

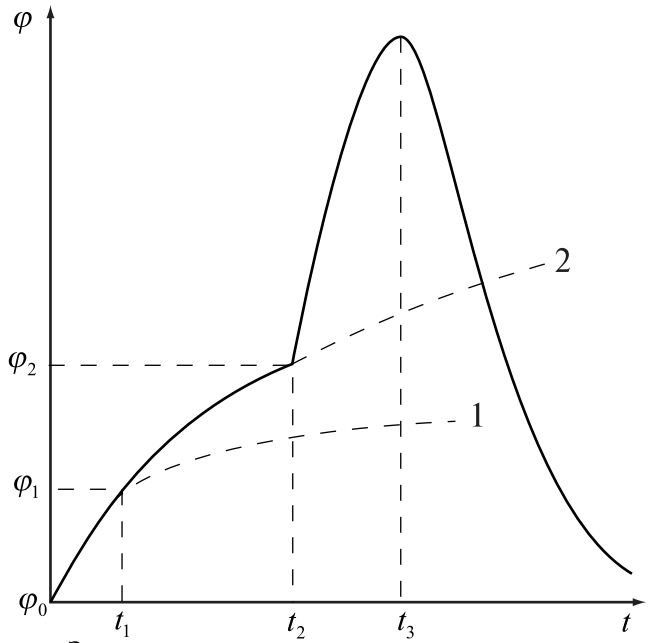


Рис. . Зависимость потенциала от времени при надпороговом возбуждении.

В случае, когда отсутствует локальный ответ и потенциал $\varphi < \varphi_1$, зависимость потенциала от времени представляет собой кривую **1**. В момент $t = t_1$, $\varphi = \varphi_1$ появляется натриевый ток и зависимость принимает вид кривой **2**. В момент $t = t_2$ потенциал превышает значение φ_2 и включается отрицательная проводимость g_2, что приводит к экспоненциальному росту потенциала. В момент t_3 натриевый ток отключается и потенциал экспоненциально падает до значения $\varphi = \varphi_0$ за счет тока утечки.

Мы рассмотрели случай, когда проводимость g_2 включается мгновенно, сразу после достижения потенциалом значения φ_2. Если же каналы открываются не сразу, а с некоторой задержкой Δt, как в реальном волокне, то картина немного меняется. Предположим также, что внешнее возбуждение прекращается в некоторый момент $t = t_0$, когда проводимость g_2 еще не включилась. Рис. .

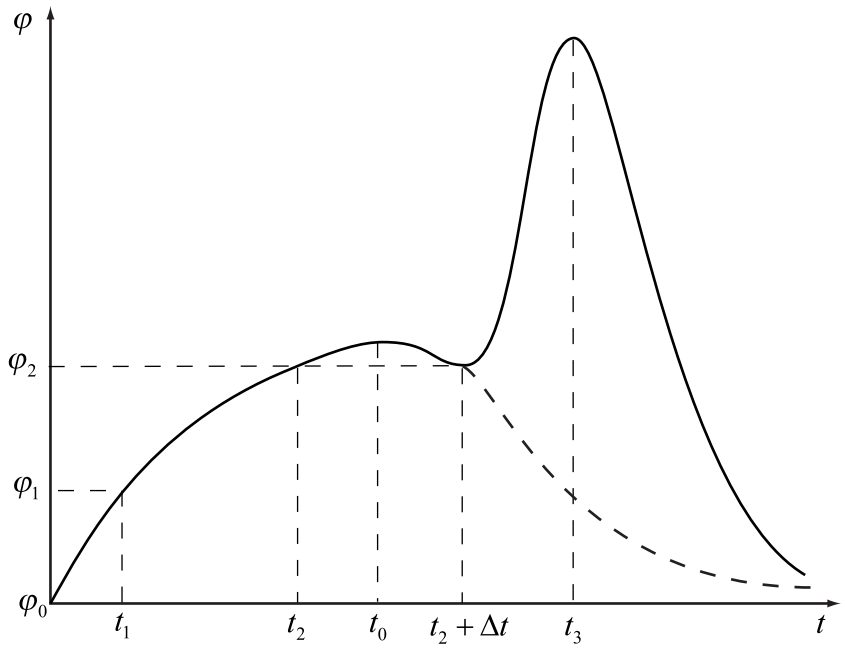


Рис. . Зависимость потенциала от времени при наличии задержки Δt включения проводимости g_2.

В момент $t = t_2$ потенциал превышает пороговое значение, но проводимость g_2 не включается. Спустя некоторое время, в момент $t = t_0$, прекращается внешнее возбуждение $\tilde{\varphi}$, и потенциал начинает уменьшаться по экспоненте, к значению φ_0. Если к моменту $t_2 + \Delta t$ потенциал не уменьшится ниже φ_2, то проводимость g_2 активируется и развивается импульс. Иначе происходит разряд емкости мембраны через проводимость утечки, и тогда импульса не возникает.

Итак, даже в простой системе с вольтамперной характеристикой, аппроксимируемой ломаной линией, может возникать импульсация, сходная с биологическими нервными импульсами.

Искусственные нейронные сети

Формальный нейрон

Биологический нейрон — сложная система, математическая модель которого до сих пор полностью не построена. Введено множество моделей, различающихся вычислительной сложностью и сходством с реальным нейроном. Одна из важнейших — *формальный нейрон* (ФН, рис. .). Несмотря на простоту ФН, сети, построенные из таких нейронов, могут сформировать произвольную многомерную функцию на выходе.

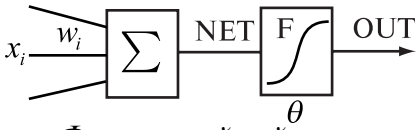


Рис. . Формальный нейрон

Нейрон состоит из взвешенного сумматора и нелинейного элемента. Функционирование нейрона определяется формулами:

$$NET = \sum_i w_i x_i \qquad (1)$$

$$OUT = F\left(NET - \theta\right) \qquad (2)$$

где x_i — входные сигналы, совокупность всех входных сигналов нейрона образует вектор **x**;

w_i — *весовые коэффициенты*, совокупность весовых коэффициентов образует вектор весов **w**;

NET — взвешенная сумма входных сигналов, значение NET передается на нелинейный элемент;

θ — *пороговый уровень* данного нейрона;

F — нелинейная функция, называемая *функцией активации.*

Нейрон имеет несколько входных сигналов **x** и один выходной сигнал OUT. Параметрами нейрона, определяющими его работу, являются: вектор весов **w**, пороговый уровень θ и вид функции активации F.

Виды функций активации

Рассмотрим основные виды функций активации, получившие распространение в искусственных НС.

1. **Жесткая ступенька** (рис. .):

$$OUT = \begin{cases} 0, & NET < \theta \\ 1, & NET \geq \theta \end{cases}$$

Используется в классическом формальном нейроне. Развита полная теория [Мкртчян71], позволяющая синтезировать произвольные логические схемы на основе ФН с такой нелинейностью. Функция вычисляется двумя-тремя машинными инструкциями, поэтому нейроны с такой нелинейностью требуют малых вычислительных затрат.

Эта функция чрезмерно упрощена и не позволяет моделировать схемы с непрерывными сигналами. Отсутствие первой производной затрудняет применение градиентных методов для обучения таких нейронов. Сети на классических ФН чаще всего формируются, синтезируются, т.е. их парамет-

ры рассчитываются по формулам, в противоположность обучению, когда параметры подстраиваются итеративно.

2. Логистическая функция (сигмоида, функция Ферми, рис. .):

$$OUT = \frac{1}{1+e^{-NET}}$$

Применяется очень часто для многослойных перцептронов и других сетей с непрерывными сигналами. Гладкость, непрерывность функции — важные положительные качества. Непрерывность первой производной позволяет обучать сеть градиентными методами (например, метод обратного распространения ошибки).

Функция симметрична относительно точки (NET=0, OUT=1/2), это делает равноправными значения OUT=0 и OUT=1, что существенно в работе сети. Тем не менее, диапазон выходных значений от 0 до 1 несимметричен, из-за этого обучение значительно замедляется.

Данная функция — сжимающая, т.е. для малых значений NET коэффициент передачи K=OUT/NET велик, для больших значений он снижается. Поэтому диапазон сигналов, с которыми нейрон работает без насыщения, оказывается широким.

Значение производной легко выражается через саму функцию. Быстрый расчет производной ускоряет обучение.

3. Гиперболический тангенс (рис.):

$$OUT = \text{th}(NET) = \frac{e^{NET} - e^{-NET}}{e^{NET} + e^{-NET}}$$

Тоже применяется часто для сетей с непрерывными сигналами. Функция симметрична относительно точки (0,0), это преимущество по сравнению с сигмоидой.

Производная также непрерывна и выражается через саму функцию.

4. Пологая ступенька (рис.):

$$OUT = \begin{cases} 0, & NET \le \theta \\ \dfrac{(NET - \theta)}{\Delta}, & \theta \le NET < \theta + \Delta \\ 1, & NET \ge \theta + \Delta \end{cases}$$

Рассчитывается легко, но имеет разрывную первую производную в точках $NET = \theta$, $NET = \theta + \Delta$, что усложняет алгоритм обучения.

5. Экспонента: $OUT = e^{-NET}$. Применяется в специальных случаях.

6. SOFTMAX-функция:

$$OUT = \frac{e^{NET}}{\sum_i e^{NET_i}}$$

Здесь суммирование производится по всем нейронам данного слоя сети. Такой выбор функции обеспечивает сумму выходов слоя, равную единице при любых значениях сигналов NET_i данного слоя. Это позволяет трактовать OUT_i как вероятности событий, совокупность которых (все выходы слоя) образует полную группу. Это полезное свойство позволяет применить SOFTMAX-функцию в задачах классификации, проверки гипотез, распознавания образов и во всех других, где требуются выходы-вероятности.

7. Участки синусоиды:

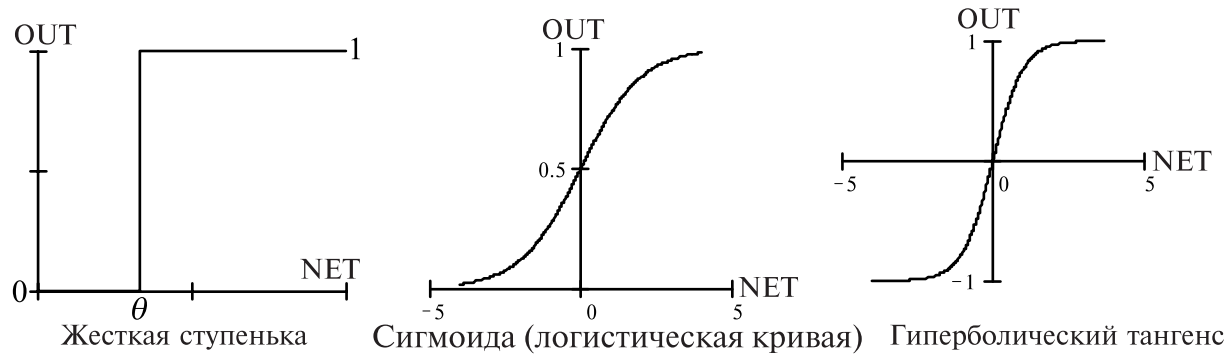
$$OUT = \sin(NET) \quad \text{для } NET = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \text{ или } NET = [-\pi, \pi]$$

8. Гауссова кривая (рис.):

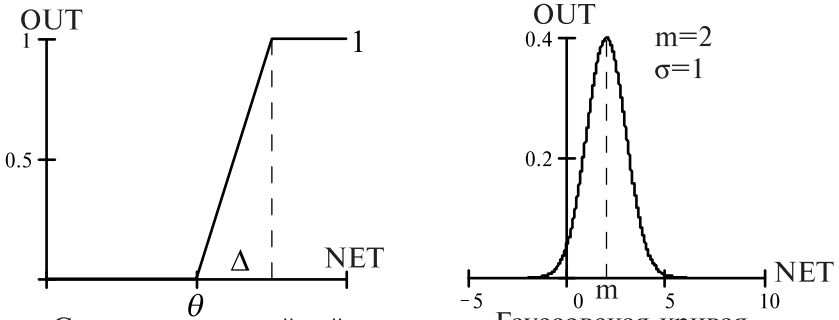
$$OUT = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(NET-m)}{2\sigma^2}}$$

Применяется в случаях, когда реакция нейрона должна быть максимальной для некоторого определенного значения *NET*.

9. Линейная функция, $OUT = K\,NET$, K=const. Применяется для тех моделей сетей, где не требуется последовательное соединение слоев нейронов друг за другом.



Жесткая ступенька Сигмоида (логистическая кривая) Гиперболический тангенс



Ступенька с линейной частью Гауссовская кривая

Рис. . Виды функций активации.

Выбор функции активации определяется:

1. Спецификой задачи.
2. Удобством реализации на ЭВМ, в виде электрической схемы или другим способом.
3. Алгоритмом обучения: некоторые алгоритмы накладывают ограничения на вид функции активации, их нужно учитывать.

Чаще всего вид нелинейности не оказывает принципиального влияния на решение задачи. Однако удачный выбор может сократить время обучения в несколько раз.

Ограничения модели нейрона

1. Вычисления выхода нейрона предполагаются мгновенными, не вносящими задержки. Непосредственно моделировать динамические системы, имеющие "внутреннее состояние", с помощью таких нейронов нельзя.
2. В модели отсутствуют нервные импульсы. Нет модуляции уровня сигнала плотностью импульсов, как в нервной системе. Не появляются эффекты синхронизации, когда скопления нейронов обрабатывают информацию синхронно, под управлением периодических волн возбуждения-торможения.
3. Нет четких алгоритмов для выбора функции активации.
4. Нет механизмов, регулирующих работу сети в целом (пример - гормональная регуляция активности в биологических нервных сетях).

5. Чрезмерная формализация понятий: "порог", "весовые коэффициенты". В реальных нейронах нет числового порога, он динамически меняется в зависимости от активности нейрона и общего состояния сети. Весовые коэффициенты синапсов тоже не постоянны. "Живые" синапсы обладают пластичностью и стабильностью: весовые коэффициенты настраиваются в зависимости от сигналов, проходящих через синапс.
6. Существует большое разнообразие биологических синапсов. Они встречаются в различных частях клетки и выполняют различные функции. Тóрмозные и возбуждающие синапсы реализуются в данной модели в виде весовых коэффициентов противоположного знака, но разнообразие синапсов этим не ограничивается. Дендро-дендритные, аксо-аксональные синапсы не реализуются в модели ФН.
7. В модели не прослеживается различие между градуальными потенциалами и нервными импульсами. Любой сигнал представляется в виде одного числа.

Итак, модель формального нейрона не является биоподобной и скорее похожа на математическую абстракцию, чем на живой нейрон. Тем удивительнее оказывается многообразие задач, решаемых с помощью таких нейронов и универсальность получаемых алгоритмов.

Многослойный перцептрон

Формальные нейроны могут объединяться в сети различным образом. Самым распространенным видом сети стал *многослойный перцептрон* (рис.).

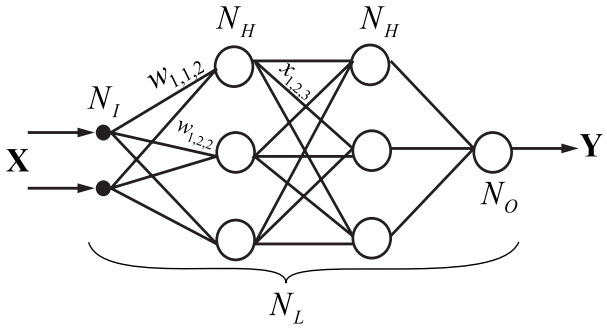


Рис. . Многослойный перцептрон.

Сеть состоит из произвольного количества слоев нейронов. Нейроны каждого слоя соединяются с нейронами предыдущего и последующего слоев по принципу "каждый с каждым". Первый слой (слева) называется *сенсорным* или *входным*, внутренние слои называются *скрытыми* или *ассоциативными*, последний (самый правый, на рисунке состоит из одного нейрона) — *выходным* или *результативным*. Количество нейронов в слоях может быть произвольным. Обычно во всех скрытых слоях одинаковое количество нейронов.

Обозначим количество слоев и нейронов в слое. Входной слой: N_I нейронов; N_H нейронов в каждом скрытом слое; N_O выходных нейронов. **x** — вектор входных сигналы сети, **y** - вектор выходных сигналов.

Существует путаница с подсчетом количества слоев в сети. Входной слой не выполняет никаких вычислений, а лишь распределяет входные сигналы, поэтому иногда его считают, иногда — нет. Обозначим через N_L полное количество слоев в сети, считая входной.

Работа многослойного перцептрона (МСП) описывается формулами:

$$NET_{jl} = \sum_i w_{ijl} x_{ijl} \qquad (1)$$

$$OUT_{jl} = F\left(NET_{jl} - \theta_{jl}\right) \qquad (2)$$

$$x_{ij(l+1)} = OUT_{il} \qquad (3)$$

где индексом i всегда будем обозначать номер входа, j — номер нейрона в слое, l — номер слоя.

x_{ijl} — i-й входной сигнал j-го нейрона в слое l;

w_{ijl} — весовой коэффициент i-го входа нейрона номер j в слое l;

NET_{jl} — сигнал NET j-го нейрона в слое l;

OUT_{jl} — выходной сигнал нейрона;

θ_{jl} — пороговый уровень нейрона j в слое l;

Введем обозначения: $\mathbf{w}_{jl}$ — вектор-столбец весов для всех входов нейрона j в слое l; $\mathbf{W}_l$ — матрица весов всех нейронов в слоя l. В столбцах матрицы расположены вектора $\mathbf{w}_{jl}$. Аналогично $\mathbf{x}_{jl}$ — входной вектор-столбец слоя l.

Каждый слой рассчитывает нелинейное преобразование от линейной комбинации сигналов предыдущего слоя. Отсюда видно, что линейная функция активации может применяется только для тех моделей сетей, где не требуется последовательное соединение слоев нейронов друг за другом. Для многослойных сетей функция активации должна быть нелинейной, иначе можно построить эквивалентную однослойную сеть, и многослойность оказывается ненужной. Если применена линейная функция активации, то каждый слой будет давать на выходе линейную комбинацию входов. Следующий слой даст линейную комбинацию выходов предыдущего, а это эквивалентно одной линейной комбианции с другими коэффициентами, и может быть реализовано в виде одного слоя нейронов.

Многослойная сеть может формировать на выходе произвольную многомерную функцию при соответствующем выборе количества слоев, диапазона изменения сигналов и параметров нейронов. Как и ряды, многослойные сети оказываются универсальным инструментом аппроксимации функций. Видно отличие работы нейронной сети от разложения функции в ряд:

Ряд: $f(x) = \sum_i c_i f_i(x)$

Нейронная сеть: $f(x) = \underbrace{F\left(\sum_{i_N} w_{i_N j_N N} \ldots \underbrace{\sum_{i_2} w_{i_2 j_2 2} \underbrace{F\left(\sum_{i_1} w_{i_1 j_1 1} x_{i_1 j_1 1} - \theta_{j_1}\right)}_{\text{слой } 1}}_{\text{слой } 2}\right)}_{\text{слой } N}$

За счет поочередного расчета линейных комбинаций и нелинейных преобразований достигается аппроксимация произвольной многомерной функции при соответствующем выборе параметров сети.

В многослойном перцептроне нет обратных связей. Такие модели называются *сетями прямого распространения*. Они не обладают внутренним состоянием и не позволяют без дополнительных приемов моделировать развитие динамических систем.

Алгоритм решения задач с помощью МСП

Чтобы построить МСП, необходимо выбрать его параметры. Чаще всего выбор значений весов и порогов требует *обучения*, т.е. пошаговых изменений весовых коэффициентов и пороговых уровней.

Общий алгоритм решения:

1. Определить, какой смысл вкладывается в компоненты входного вектора **x**. Входной вектор должен содержать формализованное условие задачи, т.е. всю информацию, необходимую для получения ответа.
2. Выбрать выходной вектор **y** таким образом, чтобы его компоненты содержали полный ответ поставленной задачи.
3. Выбрать вид нелинейности в нейронах (функцию активации). При этом желательно учесть специфику задачи, т.к. удачный выбор сократит время обучения.

4. Выбрать число слоев и нейронов в слое.
5. Задать диапазон изменения входов, выходов, весов и пороговых уровней, учитывая множество значений выбранной функции активации.
6. Присвоить начальные значения весовым коэффициентам и пороговым уровням и дополнительным параметрам (например, крутизне функции активации, если она будет настраиваться при обучении). Начальные значения не должны быть большими, чтобы нейроны не оказались в насыщении (на горизонтальном участке функции активации), иначе обучение будет очень медленным. Начальные значения не должны быть и слишком малыми, чтобы выходы большей части нейронов не были равны нулю, иначе обучение также замедлится.
7. Провести обучение, т.е. подобрать параметры сети так, чтобы задача решалась наилучшим образом. По окончании обучения сеть готова решить задачи того типа, которым она обучена.
8. Подать на вход сети условия задачи в виде вектора **x**. Рассчитать выходной вектор **y**, который и даст формализованное решение задачи.

Формализация задачи

Многослойный перцептрон может рассчитывать выходной вектор $\mathbf{y}$ для любого входного вектора $\mathbf{x}$, т.е. давать значение некоторой векторной функции $\mathbf{y} = \mathbf{f}(\mathbf{x})$. Следовательно, условие любой задачи, которая может быть поставлена перцептрону, должно являться множеством векторов $\{\mathbf{x}^1 \ldots \mathbf{x}^S\}$ с N_I компонентами каждый: $\mathbf{x}^s = \begin{pmatrix} x_1^s \\ \ldots \\ x_{N_I}^s \end{pmatrix}$. Решением задачи будет множество векторов $\{\mathbf{y}^1 \ldots \mathbf{y}^S\}$, каждый вектор $\mathbf{y}^s$ с N_O компонентами; $\mathbf{y}^s = \mathbf{f}(\mathbf{x}^s)$, где $s = 1..S$ — номер предъявленного образа.

Все, что способен сделать перцептрон — это сформировать отображение $X \to Y$ для $\forall \mathbf{x} \in X$. Данное отображение мы не можем "извлечь" полностью из перцептрона, а можем только посчитать отображение произвольного количества точек:

$$\begin{pmatrix} \mathbf{x}^1 \to \mathbf{y}^1 \\ \ldots \\ \mathbf{x}^S \to \mathbf{y}^S \end{pmatrix}$$

здесь множество векторов $\mathbf{x}^1 \ldots \mathbf{x}^S$ — *формализованное условие задачи*, а множество $\mathbf{y}^1 \ldots \mathbf{y}^S$ — *формализованное решение*. Задача формализации, т.е. выбора смысла, которым наделяются компоненты входного и выходного векторов, пока решается только человеком на основе практического опыта. Жестких рецептов формализации для нейронных сетей пока не создано. Рассмотрим, как выбирается смысл входных и выходных данных в наиболее распространенных случаях.

Примеры формализации задач

1. Задача классификации.

Пусть есть некоторый объект, который характеризуется несколькими параметрами $p_1..p_N$. Пусть также есть M классов объектов, $C_1..C_M$. Мы наблюдаем объект и можем рассчитать или измерить его параметры. Вектор **p** характеризует наблюдаемый объект:

$$\mathbf{p} = \begin{pmatrix} p_1 \\ \ldots \\ p_N \end{pmatrix}$$

На основании вектора **p** мы должны решить, к какому классу отнести объект, т.е. выбрать C_i, к которому принадлежит объект, характеризуемый набором параметров **p**.

Решение задачи можно представить в виде вектора:

$$\mathbf{c} = \begin{pmatrix} c_1 \\ \dots \\ c_M \end{pmatrix}$$

и выполняются условия:

$$0 \le c_m \le 1 \quad \text{и} \quad \sum_{m=1}^{M} c_m = 1 \qquad (1)$$

Здесь c_m — вероятность, с которой объект относится к классу C_m. Если рассматривать c_m как вероятности, то должны выполняться условия (1). К примеру, $c_1 = 0{,}9, \quad c_2 = 0{,}1$ означает, что объект с данным набором параметров **p** с вероятностью 0,9 относится к классу C_1 и с вероятностью 0,1 — к классу C_2.

Если создать МСП с N входами и M выходами и обучить его давать на выходе вектор **c**, когда на вход подается **p**, то мы решим поставленную задачу.

Сеть строит отображение $P \to C$ в процессе обучения. Целиком извлечь это отображение сеть не позволяет, но можно получить произвольное количество пар $(\mathbf{p} \to \mathbf{c})$, связанных отображением. Для произвольного вектора **p** на входе мы можем получить вероятности принадлежности к классам на выходе.

Почему на выходе будут получены именно вероятности c_m и будут ли выполняться условия (1)?

Если обучение прошло успешно, то мы наверняка получим на выходе что-то похожее на вероятности. Это определяется алгоритмом обучения. Но чаще всего оказывается, что компоненты выходного вектора могут быть меньше 0 или больше 1, а второе условие (1) выполняется лишь приблизительно: $\sum_{m=1}^{M} c_m \approx 1$. Неточность — следствие аналоговости нейронных сетей. Большинство результатов, даваемых нейросетями, неточно. Кроме того, при обучении сети указанные условия, накладываемые на вероятности, не вводятся в сеть непосредственно, а неявно содержатся во множестве данных, на которых обучается сеть. Это — вторая причина некорректности результата.

Такой способ формализации — не единственный, но один из удачных. Можно обучить сеть и по-другому. Пусть у сети только один выход, и пусть его смысл — номер класса *m* для вектора **p**, предъявленного на входе. Следовательно, сеть обучается зависимости *m*(**p**).

Если обучение прошло успешно, то когда на вход сети подан вектор **p**, характеризующий объект, на выходе будет получено число *m*, и нами принимается решение о принадлежности **p** к классу C_m.

На первый взгляд такой способ формализации более экономичен: используется всего один выход. Но существует важный недостаток. Рассмотрим пример классификации (рис.).

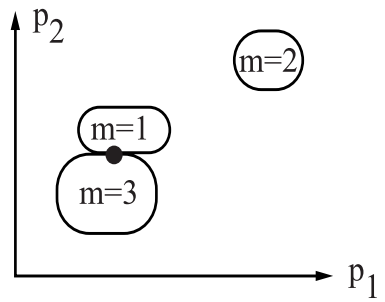


Рис. . Пример некорректной классификации.

Пусть требуется разделить объекты по двум признакам, p1, p2, на три класса, m=1, m=2, m=3. Если входной вектор **p** примет значение, обозначенное жирной точкой, то выход сети, при правильном обучении, примет значение m=2, т.е. объект будет отнесен к классу 2, совершенно неподходящему.

Данное явление возникает, т.к. сеть склонна интерполировать входные и выходные данные. Если функции активации плавные, весовые коэффициенты не слишком большие, и количество слоев не слишком велико, то выход сети тоже будет гладким и непрерывным. Для близких **p** будут получены близкие m на выходе. Но при решении задачи классификации такое допущение бывает неверным. Отсюда неправильное решение.

Чтобы избежать ошибок, можно применить другие способы формализации или упорядочить номера классов m так, чтобы близким m соответствовали близкие в пространстве P классы.

2. Распознавание букв алфавита.

Будем представлять буквы в виде точечных изображений (рис.).

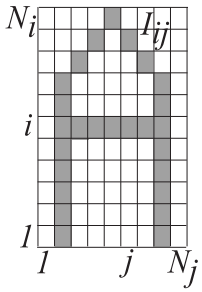


Рис. . Точечное изображение.

Темной клетке-пикселу на изображении соответствует $I_{ij}=1$, светлому — $I_{ij}=0$. Задача состоит в том, чтобы определить по изображению букву, которая была предъявлена.

Построим МСП с $N_i \cdot N_j$ входами, каждому входу соответствует один пиксел: $x_k = I_{ij}$, $k = 1..N_i \cdot N_j$. Яркости пикселов будут компонентами входного вектора.

В качестве выходных сигналов выберем вероятности того, что предъявленное изображение соответствует данной букве: $\mathbf{y} = (c_1 ... c_M)^T$. Сеть рассчитывает выход:

$$(I_{ij}) \to \begin{pmatrix} c_1 \\ ... \\ c_M \end{pmatrix}$$

где выход $c_1 = 0,9$ означает, к примеру, что предъявлено изображение буквы "А", и сеть уверена в этом на 90%, выход $c_2 = 0,1$ — что изображение соответствовало букве "Б" с вероятностью 10% и т.д.

Другой способ: входы сети выбираются так же, а выход — только один, номер m предъявленной буквы. Сеть учится давать значение m по предъявленному изображению I:

$$(I_{ij}) \to m$$

Недостаток, рассмотренный в примере **1**, сохраняется и здесь: буквы, имеющие близкие номера m, но непохожие изображения, могут быть перепутаны сетью при распознавании.

3. Прогнозирование одномерной функции

Пусть задана функция W, Θ, определенная на интервале времени $[0, t_0]$, где t_0 — текущее значение времени. Требуется предсказать значение функции при $t > t_0$. Чтобы применить многослойный перцептрон для прогнозирования, время придется дискретизировать. Будем считать известными значения функции в моменты времени:

$$\begin{pmatrix} x_0 = f(t_0) \\ x_1 = f(t_0 - \delta_1) \\ x_2 = f(t_0 - \delta_1 - \delta_2) \\ \dots \\ x_n = f(t_0 - \delta_1 - \dots \delta_n) \end{pmatrix} = \mathbf{x}, \quad \delta_i > 0.$$

Будем предсказывать значение функции в момент времени $(t_0 + \delta_0)$ для $\forall \delta_0 > 0$. δ_0 называется *интервалом прогнозирования*. Решением задачи будем считать значение $f(t_0 + \delta_0) = y$.

Построим сеть, имеющую *n* входов и 1 выход. В качестве входного вектора возьмем вектор **x**, а выходного — один сигнал y.

Такая сеть предсказывает значение функции в одной точке *y* по (*n*+1) известным значениям функции, заданным вектором **x**. Выбрав при обучении сети набор интервалов δ_i, его нельзя изменить после обучения. Сеть с данными параметрами W, Θ, полученными при обучении, может прогнозировать только с одним набором δ_i.

Можно ли прогнозировать функцию в виде дискретного процесса во времени? Как предсказать несколько значений функции в разных точках?

Для этого найден интересный способ. Выберем все интервалы одинаковыми: $\delta_i = \delta = \text{const}, \quad i = 0 \dots n$. Построим и обучим сеть. Подадим на вход вектор **x** со значениями функции в известных точках. Рассчитав выход сети, получим прогнозируемое значение функции в точке $f(t_0 + \delta_0) = y$. Теперь "сдвинем" компоненты входных и выходных векторов следующим образом (знак равенства означает "присвоить значение"):

$x_n = x_{n-1}$

...

$x_1 = x_0$

$x_0 = y$

Теперь выходной вектор стал одной из компонент входного. Снова рассчитываем выход, и получаем значение функции в точке $(t_0 + 2\delta)$. Повторив эти операции, можно прогнозировать функцию в любом количестве точек с дискретным шагом по времени, равным δ.

4. Аппроксимация многомерной функции.

Рассмотрим многомерную функцию $\mathbf{y} = \mathbf{f}(\mathbf{x})$, где вектор **y** имеет N_O компонент, а вектор **x** — N_I компонент. Самый простой способ формализации — использовать сеть с N_I входами и N_O выходами. Компоненты вектора **x** подаются на вход сети, **y** — снимаются на выходе. Сеть обучается на известных значениях функции **f**.

Выбор количества нейронов и слоев

Нет строго определенной процедуры для выбора количества нейронов и количества слоев в сети. Чем больше количество нейронов и слоев, тем шире возможности сети, тем медленнее она обучается и работает и тем более нелинейной может быть зависимость вход-выход.

Количество нейронов и слоев связано:

1) со сложностью задачи;
2) с количеством данных для обучения;
3) с требуемым количеством входов и выходов сети;

4) с имеющимися ресурсами: памятью и быстродействием машины, на которой моделируется сеть;

Были попытки записать эмпирические формулы для числа слоев и нейронов, но применимость формул оказалась очень ограниченной.

Если в сети слишком мало нейронов или слоев:

1) сеть не обучится и ошибка при работе сети останется большой;

2) на выходе сети не будут передаваться резкие колебания аппроксимируемой функции **y**(**x**).

Превышение требуемого количества нейронов тоже мешает работе сети.

Если нейронов или слоев слишком много:

1) быстродействие будет низким, а памяти потребуется много — на фон-неймановских ЭВМ;

2) сеть *переобучится*: выходной вектор будет передавать незначительные и несущественные детали в изучаемой зависимости **y**(**x**), например, шум или ошибочные данные;

3) зависимость выхода от входа окажется резко нелинейной: выходной вектор будет существенно и непредсказуемо меняться при малом изменении входного вектора **x**;

4) сеть будет неспособна к *обобщению*: в области, где нет или мало известных точек функции **y**(**x**) выходной вектор будет случаен и непредсказуем, не будет адекватен решамой задаче.

Подготовка входных и выходных данных

Данные, подаваемые на вход сети и снимаемые с выхода, должны быть правильно подготовлены. Один из распространенных способов — масштабирование:

$$\mathbf{x} = (\mathbf{x}' - \mathbf{m})c$$

где $\mathbf{x}'$ — исходный вектор, $\mathbf{x}$ — масштабированный. Вектор **m** — усредненное значение совокупности входных данных. c — масштабный коэффициент.

Масштабирование желательно, чтобы привести данные в допустимый диапазон. Если этого не сделать, то возможно несколько проблем:

1) нейроны входного слоя или окажутся в постоянном насыщении (|**m**| велик, дисперсия входных данных мала) или будут все время заторможены (|**m**| мал, дисперсия мала);

2) весовые коэффициенты примут очень большие или очень малые значения при обучении (в зависимости от дисперсии), и, как следствие, растянется процесс обучения и снизится точность.

Рассмотрим набор входных данных для сети с одним входом:

$$\left\{x'^{(s)}\right\} = \{10 \quad 10,5 \quad 10,2 \quad 10,3 \quad 10,1 \quad 10,4\}$$

Если функция активации — гиперболический тангенс с множеством значений, то при весовых коэффициентах около единицы нейроны входного слоя окажутся в насыщении для всех $x'^{(s)}$. Применим масштабирование с m=10,2 и c=4. Это даст значения в допустимом диапазоне $(-1;\ 1)$.

Выходы сети масштабируются так же. Т.к. мы сами выбираем смысл выходного вектора при создании сети, то мы должны подготовить данные так, чтобы диапазон изменения выходных сигналов лежал на рабочем участке функции активации.

Другие способы подготовки данных

Исходя из условий конкретной задачи, можно выбрать другой способ подготовки данных. Можно использовать нелинейные преобразования. Рассмотрим задачу прогнозирования курса доллара в следующий день на основе курсов в предыдущие дни. Хорошие результаты были получены при таком выборе выхода сети:

$$y = \operatorname{sgn}\left(f(t_{i+1}) - f(t_i)\right)$$

или $\quad y = c\left(f(t_{i+1}) - f(t_i)\right)$

где $f(t_i)$ — значение курса в i-й день.

От сети требуется предсказать или только направление изменения курса (первая формула), или само изменение. Оказалось, что точность предсказания в первом случае выше, чем когда предсказывается абсолютное значение курса. Направление изменения предсказывается, когда для прогнозирования точного значения недостаточно данных.

Если диапазон изменения входных данных очень велик (например, при обработке яркостной информации о реальных объектах), можно использовать логарифмическую шкалу для данных. Другие нелинейные преобразования при подготовке данных тоже находят применение.

Методы обучения

Алгоритмы обучения бывают с учителем и без. Алгоритм называется *алгоритмом с учителем,* если при обучении известны и входные, и выходные вектора сети. Имеются пары вход + выход — известные условия задачи и решение. В процессе обучения сеть меняет свои параметры и учится давать нужное отображение $X \to Y$. Сеть учится давать результаты, которые нам уже известны. За счет способности к обобщению сетью могут быть получены новые результаты, если подать на вход вектор, который не встречался при обучении.

Алгоритм относится к обучению *без учителя*, если известны только входные вектора, и на их основе сеть учится давать *наилучшие* значения выходов. Что понимается под “наилучшими” — определяется алгоритмом обучения.

Перцептрон обучается с учителем. Это означает, что должно быть задано множество пар векторов $\{\mathbf{x}^s, \mathbf{d}^s\}$, $s = 1...S$, где $\{\mathbf{x}^s\} = \{\mathbf{x}^1, \ldots, \mathbf{x}^S\}$ — формализованное условие задачи, а $\{\mathbf{d}^s\} = \{\mathbf{d}^1, \ldots, \mathbf{d}^S\}$ — известное решение для этого условия. Совокупность пар $\{\mathbf{x}^s, \mathbf{d}^s\}$ составляет *обучающее множество*. S — количество элементов в обучающем множестве — должно быть достаточным для обучения сети, чтобы под управлением алгоритма сформировать набор параметров сети, дающий нужное отображение $X \to Y$.

Количество пар в обучающем множестве не регламентируется. Если элементов слишком много или мало, сеть не обучится и не решит поставленную задачу.

Выберем один из векторов $\mathbf{x}^s$ и подадим его на вход сети. На выходе получится некоторый вектор $\mathbf{y}^s$. Тогда ошибкой сети можно считать $E^s = \|\mathbf{d}^s - \mathbf{y}^s\|$ для каждой пары $(\mathbf{x}^s, \mathbf{d}^s)$. Чаще всего для оценки качества обучения выбирают суммарную квадратическую ошибку: $E = \frac{1}{2}\sum_s \sum_j \left(d_j^s - y_j^s\right)^2$. Реже применяется средняя относительная ошибка: $\sigma = \frac{1}{S\,N_O}\sum_s \sum_j \left(\frac{\left|d_j^s - y_j^s\right| + 1}{\left|d_j^s\right| + 1} - 1 \right) \cdot 100\%$. Ее преимущество в том, что она дает значение, не зависящее напрямую ни от количества примеров в обучающем множестве, ни от размерности выходного вектора, и имеет удобное для восприятия человеком значение в интервале от 0 до 100%.

Задача обучения перцептрона ставится так: подобрать такие значения параметров сети, чтобы ошибка была минимальна для данного обучающего множества $\{\mathbf{x}^s, \mathbf{d}^s\}$.

Большая часть методов обучения — итерационные. Параметрам сети (весовым коэффициентам и пороговым уровням) присваиваются малые начальные значения. Затем параметры изменяются так, чтобы ошибка E убывала. Изменения продолжаются до тех пор, пока ошибка не станет достаточно малой.

Общая схема обучения перцептрона:

1. Инициализировать веса и параметры функции активации в малые ненулевые значения;
2. Подать на вход один образ и рассчитать выход;

3. Посчитать ошибку E^s, сравнив $\mathbf{d}^s$ и $\mathbf{y}^s$.
4. Изменить веса и параметры функции активации так, чтобы ошибка E^s уменьшилась.
5. Повторить шаги 2-4 до тех пор, пока ошибка не перестанет убывать или не станет достаточно малой.

Здесь веса меняются так, что убывает не E, а E^s, относящаяся к образу s, а не ко всему обучающему множеству. Шаги в данном варианте алгоритма делаются не в направлении убывания E, а в направлении убыли E^s, таким образом, E не обязательно должна убывать. Какие условия необходимы для существенной убыли E? Опыт показывает, что для этого необходимо отсутствие упорядоченности в предъявлении образов, т.е. в выборе s на каждой итерации. Если образы выбираются случайно из обучающего множества, то ошибка E чаще всего убывает. Если же есть упорядоченность (например, образы предъявляются циклически: 1-й, 2-й, ..., S-й, 1-й,...) то чаще всего $E(t)$, где t — время обучения, не имеет предела при $t \to \infty$, т.е. алгоритм расходится. В этом случае E^s тоже убывает при каждом изменении параметров, но при следующей коррекции для образа $(s + 1)$ ошибка E^{s+1} убывает, а E^s, относящаяся к предыдущему образу, возрастает сильнее, так что E может увеличиться. Сеть "забывает" текущий образ при предъявлении следующего.

Чтобы шаг по параметрам на каждой итерации делался в правильном направлении, надо провести усреднение по S, т.е. предъявить все образы, и коррекции вычислять по всем образам сразу. Такие алгоритмы называются *алгоритмами с пакетной коррекцией* (batch update). Они требуют больших затрат вычислительного времени и памяти, но сходятся за меньшее число итераций.

В большинстве случаев $E(W,\Theta)$ при таком методе обучения сходится и достигает локального минимума. Для каждой конкретной задачи нет гарантий, что E сойдется к приемлемому значению за конечное число шагов.

Задача поиска $\min_{W,\Theta} E(W,\Theta)$ является задачей безусловной оптимизации. Для ее решения известно множество методов [Гилл85]. В теории оптимизации функция ошибки $E(W,\Theta)$ называется целевой функцией. Для нейронных сетей хорошо работают многие методы безусловной оптимизации, часто лучше, чем узкоспециальные, придуманные для обучения нейронных сетей.

Применяются следующие методы теории оптимизации:

1) для небольшого количества параметров — стабилизированные методы Ньютона, Гаусса-Ньютона, Левенберга-Маркардта;
2) для среднего количества параметров — квазиньютоновские методы;
3) для большого количества параметров — метод сопряженных градиентов.

Для сходимости алгоритма достаточно, чтобы на каждой итерации обеспечивалась существенная убыль Е. Формулы, которым должно удовлетворять изменение Е, можно найти в [Гилл85]. Была также доказана теорема об обучаемости перцептрона [Розенблат65], что перцептрон способен изучить любое отображение $X \to Y$, которое он способен дать на выходе. Если существует набор параметров с минимальным значением $E_0 = E(W,\Theta)$, то этот набор может быть найден в результате работы алгоритма обучения.

На практике сходимость обычно определяется методом проб и ошибок. Теория не дает точных данных о количестве итераций, требуемом для обучения перцептрона конкретной задаче. Неизвестно также, сможет ли вообще нейронная сеть обучиться ее решению.

Обучение однослойного перцептрона

Задача обучения однослойного перцептрона решается просто. Его работа определяется выражением:

$$NET_{j1} = \sum_i w_{ij1} x_{ij1}, \quad y_j = F\left(NET_{j1} - \theta_{j1}\right) \quad \text{или:}$$

$y_j = F\left(\sum_i w_{ij} x_{ij} - \theta_j\right)$, если отбросить ненужный индекс l=1, обозначающий слой.

Подадим на вход один вектор $\mathbf{x}^s$ из обучающего множества. Рассчитаем выход и сравним полученный выходной вектор $\mathbf{y}^s$ с эталоном: $\mathbf{d}^s$. Зная разницу между ними, можно ввести коррекции для весовых коэффициентов и пороговых уровней:

$$\begin{cases} \Delta w_{ij} = \varepsilon\left(d_j^s - y_j^s\right) x_{ij} \\ \Delta\theta_j = -\varepsilon\left(d_j^s - y_j^s\right) \end{cases} \qquad ()$$

где ε — небольшое положительное число, характеризующее скорость обучения. Разница между выходом и эталоном, $\left(d_j^s - y_j^s\right)$ и умножение на текущее значение входа x_{ij} обеспечивают правильное направление коррекций: если $y_j^s < d_j^s$, то выход должен увеличиться, и вес увеличивается, если $x_{ij} > 0$ и уменьшается, если $x_{ij} < 0$. Если $x_{ij} = 0$, то вес менять нельзя, т.к. он не влияет на выход. Абсолютное значение x_{ij} также учитывается при обучении. Если значение входа велико, то небольшое изменение веса сильно меняет выход. Чем меньше меняются веса, тем меньше вероятность искажения уже запомненных образов. Поэтому множитель x_{ij} оправдан. Можно использовать $\mathrm{sgn}\left(d_j^s - y_j^s\right)$ вместо $\left(d_j^s - y_j^s\right)$. Результаты при этом остаются похожими.

Чем больше дисперсия параметров сети W, Θ, тем больше количество информации, запомненное сетью. Такой выбор коррекций () способствует росту дисперсии, и, следовательно, увеличивает количество запомненной информации.

Расписание обучения

Веса и пороговые уровни инициализируются случайными значениями. Созданная таким образом сеть абсолютно неадекватна решаемой задаче и может генерировать на выходе только шум. Поэтому ошибка в начале обучения очень велика, и есть смысл вводить большие коррекции параметров. Ближе к концу обучения ошибка значительно снижается, и коррекции должны быть малыми. Чтобы менять длину шагов по параметрам, используют *расписание обучения* (learning schedule). Выберем скорость обучения зависящей от времени обучения: $\varepsilon(t)$. Обычно скорость монотонно убывает с ростом времени. Для сходимости алгоритма необходимо:

$$\varepsilon(t) \underset{t\to\infty}{\to} 0 \quad \text{и} \quad \int_0^\infty \varepsilon(t) = +\infty$$

Часто выбирают $\varepsilon(t) = 1/\alpha t, \quad \alpha > 0$ или аналогичные функции.

Алгоритмы с расписанием обучения сходятся быстрее, т.к. в начале используются большие коррекции, и дают более точные результаты за счет точной настройки параметров в конце обучения.

Уменьшение шагов к концу алгоритма сходно с методом имитации отжига, который рассматривается далее. Сходство проявляется еще и в том, что преодолеваются локальные минимумы на начальном этапе обучения. Коррекции настолько велики, что параметры "проскакивают" оптимальное значение и сеть попадает в область притяжения другого минимума, а не задерживается в первом найденном минимуме.

Перцептронная представляемость

Сложность задач, решаемых перцептроном, существенно зависит от количества слоев. Рассмотрим эти различия.

1. Однослойный перцептрон.

Круг проблем, которые под силу однослойному перцептрону, очень ограничен. Рассмотрим однослойную сеть из одного нейрона (рис).

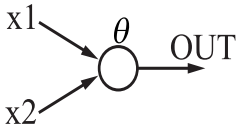


Рис. . Сеть из одного нейрона.

Выход сети: $y = F\left(w_1 x_1 + w_2 x_2 - \theta\right)$. Если F имеет вид жесткой ступеньки (рис.) с двумя возможными значениями, 0 и 1, то выход сети будет иметь вид (рис.). Гиперплоскость (в случае многих входов), разделяющая различные значения выхода, называется *решающей поверхностью.* Для жесткой ступеньки решающая поверхность задается уравнением:

$$w_1 x_1 + w_2 x_2 = \theta \qquad ()$$

Для двухвходового нейрона она имеет вид прямой, произвольно повернутой и смещенной из начала координат. Угол поворота определяется коэффициентами w_1, w_2, а смещение из начала координат — порогом θ.

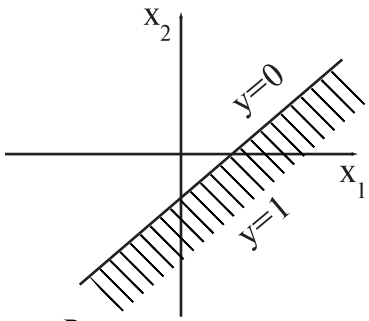


Рис. . Решающая поверхность однослойного перцептрона.

Если выбрана гладкая функция активации, то выход сети будет плавно меняться от нуля до единицы в направлении, перпендикулярном прямой (). Зависимость выходного сигнала от входов удобно представлять полутоновой картой: черному цвету соответствует значение выхода, равное 0, белому — равное 1. По осям отложены значения входов. Реальная зависимость для функции активации в виде гиперболического тангенса — на рис. .

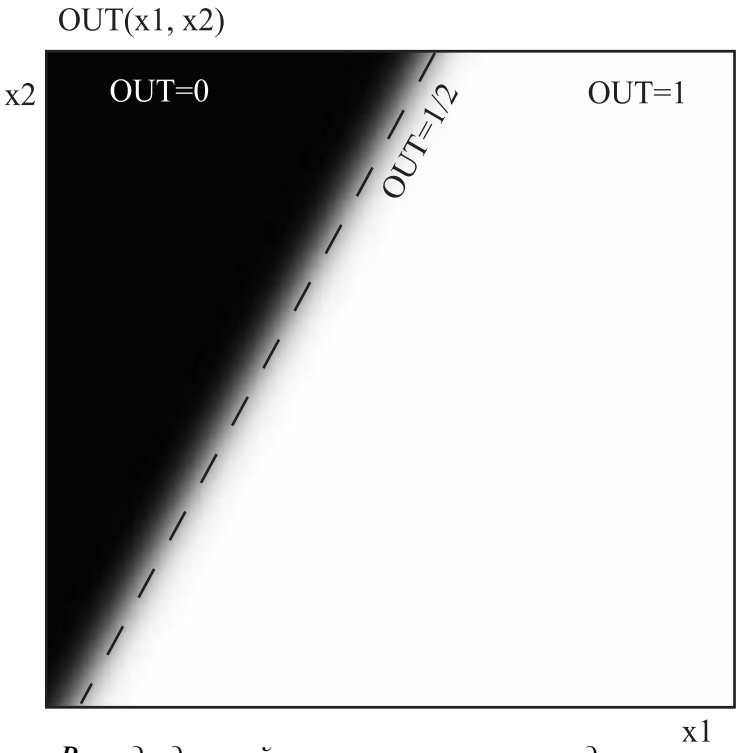


Рис. . Выход однослойного перцептрона в виде полутоновой карты.

Для искусственного нейрона с гладкой функцией активации поверхности *y*=const — гиперплоскости $\sum_{i=1}^{N_I} w_i x_i = \theta$. Ориентация плоскостей в пространстве может быть произвольной.

2. Двухслойный перцептрон.

Двухслойный перцептрон с двумя входами и одним выходом представлен на рис. . Пусть функция активации — жесткая ступенька с двумя возможными значениями. Решающая поверхность является пересечением, объединением, инверсией или их комбинацией от областей, создаваемых тремя нейронами в первом слое. Вид функции (пересечение, объединение, инверсия или их комбинация) определяется параметрами нейрона второго слоя (порогом и весами). Если моделируется пересечение, то такая двухслойная сеть может сформировать *произвольную выпуклую многоугольную односвязную решающую область*. Число сторон в многоугольнике совпадает с количеством нейронов в слое 1. В зависимости от вида комбинирования областей, а также от положения гиперплоскостей область может быть открытой или закрытой. Если область закрытая, то она *всегда* имеет вид выпуклого многоугольника. Пример открытой области, полученной объединением областей первого слоя — на рис. . Первый слой содержит 4 нейрона, их выходы обозначены $y_1 ... y_4$. Выход сети получается объединением выходов первого слоя операцией "ИЛИ". Серым цветом показана область, в которой выход сети равен единице.

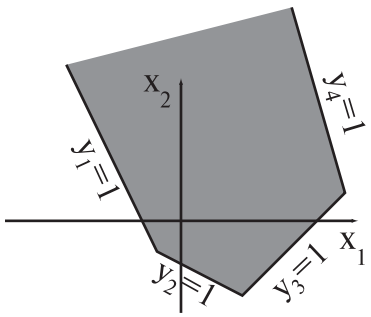


Рис. . Открытая область, сформированная двухслойным перцептроном.

Для гладкой непрерывной функции активации результаты аналогичные, но нет четких линий, отделяющих области друг от друга. Картина "расплывается". Реальные области, полученные двухслойным перцептроном с функцией активации — гиперболическим тангенсом — на рис. . в виде полутоновой карты.

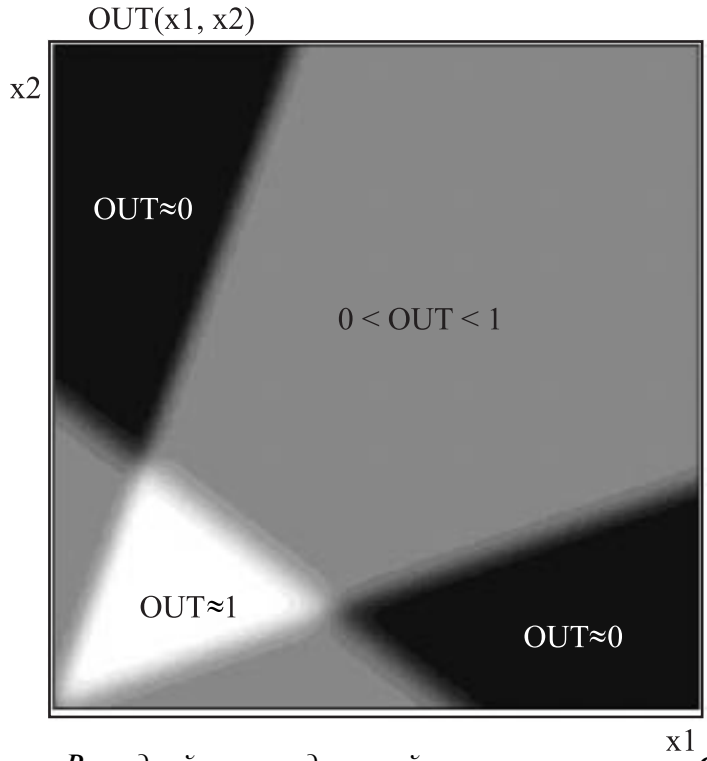
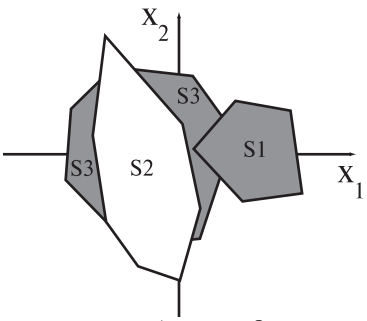


Рис. . Выходной сигнал двухслойного перцептрона с ФА — гиперболическим тангенсом.

3. Трехслойный перцептрон

Трехслойный перцептрон является наиболее общим в своем классе сетей и способен формировать произвольные многоугольные невыпуклые многосвязные области. Нейроны первых двух слоев создают произвольные независимые многоугольные решающие области в нужном количестве и в нужных измерениях входного пространства X. Эти области объединяются нейронами третьего слоя в нужной комбинации. Как и для двухслойного перцептрона, допускаются операции пересечения и объединения. Весовые коэффициенты могут быть отрицательными, и соответствующая область может входить со знаком минус, что реализует операцию инверсии. Результирующая область может быть открытой.

Пример решающей области для жесткой нелинейности в нейронах показан на рис. . Первый и второй слой формируют независимые подобласти S1, S2, S3. Единственный нейрон третьего слоя объединяет их по закону $(S1 \vee S3) \wedge \neg S2$. Весовой коэффициент нейрона в третьем слое, соответствующий подобласти S2, имеет знак "-", и поэтому перед областью S2 стоит знак отрицания.



Решающая область для трехслойного перцептрона с жесткой нелинейностью.

Проблема "исключающего ИЛИ"

Наглядной и уже ставшей классической [Минский71] иллюстрацией ограниченности для однослойного перцептрона является функция "исключающее ИЛИ". Эта булева функция от двух переменных принимает значение "истина", когда значения входных переменных различны, и "ложь" — в противном случае. Попробуем применить однослойную сеть, состоящую из одного нейрона (рис.), для построения этой функции.

Для сохранения симметрии будем сопоставим значению "ложь" сигнал нейросети, равный -1, а значению "истина" — равный 1. Значения входов и выходов дискретны, и есть смысл использовать жесткую ступеньку в качестве функции активации. Требуемая таблица истинности и соответствующие уравнения для весов и порогов для функции "исключающее или":

x_1	x_2	y	*условие*
1	1	−1	$w_1 + w_2 - \theta < 0$
1	−1	1	$w_1 - w_2 - \theta > 0$
−1	1	1	$-w_1 + w_2 - \theta > 0$
−1	−1	−1	$-w_1 - w_2 - \theta < 0$

(a)

Табл. . Система неравенств для построения функции XOR.

Вычтем из первого уравнения второе, а из третьего — четвертое. Получим несовместную систему:

$$\begin{cases} 2w_2 < 0 \\ 2w_2 > 0 \end{cases} \qquad ()$$

Хотя система (a) — лишь система неравенств, пусть содержащая три переменных и четыре уравнения, она оказывается несовместной. Следовательно, *функция XOR не реализуется на однослойном перцептроне*. Тот же результат можно получить графически. Возможные значения входов сети — на рис. .

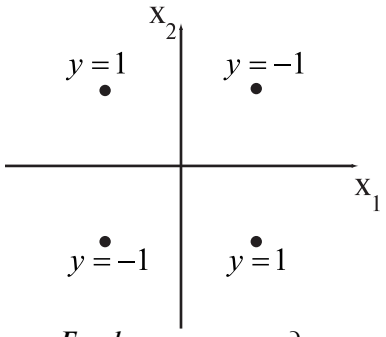


Рис. . Графическое представление функции XOR.

Один нейрон с двумя входами может сформировать решающую поверхность в виде произвольной прямой. Требуется провести ее так, чтобы отделить значения y=1 от значений y=-1. Очевидно, что это невозможно. Задачи, для которых не существует решающей поверхности в виде гиперплоскости, называются *линейно неразделимыми*.

Хотя данный пример нагляден, он не является серьезным ограничением нейросетей. Функция XOR легко формируется уже двухслойной сетью, причем многими способами.

Решение проблемы XOR

Вообще говоря, для построения булевых функций с помощью нейросетей есть завершенные математические методы [Мкртчян71]. Рассмотрим простейший пример и построим нейронную сеть без предварительного обучения.

Запишем функцию XOR в виде: $y = x_1 \text{ XOR } x_2 = (x_1 \text{ AND NOT } x_2) \text{OR} (\text{NOT } x_1 \text{ AI}$. Алгоритмические обозначения операторов использованы для наглядности. Легко построить двухслойную сеть для реализации такой функции. Инверсию можно осуществить отрицательным весовым коэффициентом, а операции AND и OR — каждую отдельным нейроном с различными значениями порогов. Схема сети представлена на рис. .

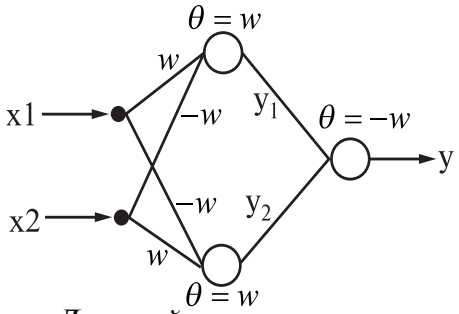


Рис. . Двухслойная сеть, реализующая функцию XOR.

Таблица истинности для такой сети: рис. .

x_1	x_2	NET_1	NET_2	OUT_1	OUT_2	NET
−1	−1	0	0	−1	−1	−2w
−1	1	−w	w	−1	1	w
1	−1	w	−w	1	−1	w
1	1	0	0	−1	−1	−2w

Рис. . Таблица истинности для нейронной сети.

Видно, что функция реализована правильно. Общие методы синтеза сетей для булевых функций изложены в [Мкртчян71].

Обучение многослойного перцептрона
Алгоритм обратного распространения ошибки

Для однослойного перцептрона алгоритм обучения очевиден. Как обобщить этот простой алгоритм на случай многослойной сети? Эту задачу решает алгоритм Румельхарта-Хинтона-Вильямся

(алгоритм обратного распространения ошибки). Он был предложен в различных вариациях в нескольких научных работах, существует также множество улучшенных версий алгоритма.

Пусть задан многослойный перцептрон с гладкой функцией активации (рис.).

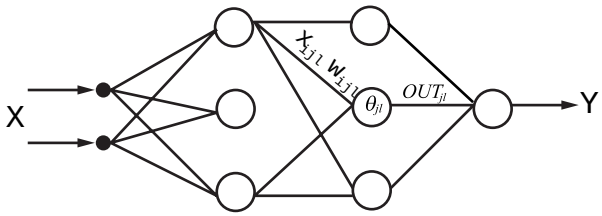


Рис. . Многослойный перцептрон.

Его работа задается:

$$NET_{jl} = \sum_i w_{ijl} x_{ijl} - \theta \qquad (1)$$

$$OUT_{jl} = F\left(NET_{jl}\right) \qquad (2)$$

$$x_{ij(l+1)} = OUT_{il} \qquad (3)$$

Выберем суммарную квадратическую ошибку в качестве целевой функции:

$$E = \frac{1}{2}\sum_j \sum_s \left(y_j^s - d_j^s\right)^2 \qquad (4)$$

Сеть задается своим вектором параметров — набором весовых коэффициентов и пороговых уровней:

$$\mathbf{P} = \begin{pmatrix} \mathbf{W} \\ \mathbf{\Theta} \end{pmatrix}$$

где $\mathbf{W}$ — вектор, компоненты которого — все весовые коэффициенты сети; $\mathbf{\Theta}$ — вектор пороговых уровней сети. Таким образом, если считать обучающее множество заданным, то ошибка сети зависит только от вектора параметров: $E = E\left(\mathbf{P}\right)$.

При обучении на каждой итерации будем корректировать параметры в направлении антиградиента E:

$$\Delta\mathbf{P} = -\varepsilon \nabla E\left(\mathbf{P}\right) \qquad (5)$$

В теории оптимизации доказано, что такой алгоритм обеспечивает сходимость к одному из локальных минимумов функции ошибки, при условии правильного выбора $\varepsilon > 0$ на каждой итерации. Такой метод оптимизации называется *методом наискорейшего спуска*.

Коррекции необходимо рассчитывать на каждой итерации. Поэтому каждая итерация требует расчета компонент градиента ∇E и выбора оптимального шага ε. Как рассчитать градиент ∇E с наименьшими вычислительными затратами?

Самое простое, но не лучшее решение для этого — воспользоваться определением градиента:

$$\frac{\partial E\left(\mathbf{P}\right)}{\partial p_i} = \lim_{\Delta \mathbf{p}_i \to \mathbf{0}} \frac{E\left(\mathbf{P} + \Delta \mathbf{p}_i\right) - E\left(\mathbf{P}\right)}{\Delta \mathbf{p}_i} \qquad (6)$$

где $\Delta \mathbf{p}_i$ — приращение *i*-й компоненты вектора параметров **P**.

Однако чтобы рассчитать каждое значение функции *E*, требуется подать входной вектор и просчитать выходные значения для всех нейронов в сети. Это — очень большой объем вычислений. Если учесть, что требуется рассчитать *все* компоненты градиента таким образом, неэффективность метода становится очевидной.

Алгоритм обратного распространения — способ ускоренного расчета компонент градиента. Идея метода в том, чтобы представить *E* в виде сложной функции и последовательно рассчитать частные производные по формуле для сложной функции.

Запишем (5) для весовых коэффициентов:

$$\Delta w_{ijl} = -\varepsilon \left(\frac{\partial E(W,\Theta)}{\partial w_{ijl}} \right)\Bigg|_{W,\Theta} \quad (7)$$

$$w'_{ijl} = w_{ijl} + \Delta w_{ijl} \quad (8)$$

— значение производной рассчитывается для текущих значений параметров W,Θ на данном шаге обучения. w'_{ijl} — значение веса на следующем шаге обучения, w_{ijl} — на данном шаге. Индексы имеют тот же смысл, что и раньше. Аналогичные коррекции вводятся для пороговых уровней.

Для выходного слоя легко записать компоненты градиента по весам как производную сложной функции:

$$\frac{\partial E}{\partial w_{ijl}} = \frac{\partial E}{\partial OUT_{jl}} \frac{\partial OUT_{jl}}{\partial NET_{jl}} \frac{\partial NET_{jl}}{\partial \theta_{jl}} \quad (9)$$

Здесь индекс l равен номеру выходного слоя.

NET_{jl} — сигнал NET для j-го нейрона в выходном слое l.

w_{ijl} — i-й вход j-го нейрона в выходном слое.

Точно так же распишем производную по порогам:

$$\frac{\partial E}{\partial \theta_{jl}} = \frac{\partial E}{\partial OUT_{jl}} \frac{\partial OUT_{jl}}{\partial NET_{jl}} \frac{\partial NET_{jl}}{\partial \theta_{jl}} \quad (10)$$

Для выходного слоя $y_j = OUT_{jl}$. Производная функции ошибки:

$$\frac{\partial E}{\partial OUT_{jl}} = \sum_s \left(y_j - d_j^s \right) \quad (11)$$

Для удобства приведем схему формального нейрона:

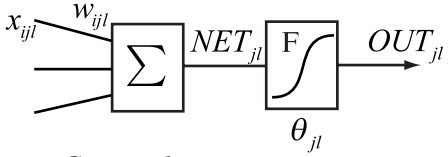


Рис. . Схема формального нейрона.

Производная от взвешенной суммы по весам:

$$\frac{\partial NET_{jl}}{\partial w_{ijl}} = \frac{\partial \left(\sum_i w_{ijl} x_{ijl} - \theta_{jl} \right)}{\partial w_{ijl}} = x_{ijl} \quad (12)$$

Производная от функции активации:

$$\frac{\partial OUT_{jl}}{\partial NET_{jl}} = \frac{\partial F\left(NET_{jl}\right)}{\partial NET_{jl}} \quad (13)$$

Производная по пороговому уровню:

$$\frac{\partial NET_{jl}}{\partial \theta_{jl}} = -1 \quad (14)$$

Если $F(NET) = \mathrm{th}(NET)$, то

$$\frac{\partial F\left(NET_{jl}\right)}{\partial NET_{jl}} = 1 - F^2\left(NET_{jl}\right) \quad (15)$$

$F(NET)$ — это текущее значение выходного сигнала нейрона. Поэтому производная рассчитывается через текущее значение сигнала OUT_{jl}:

$$\left.\frac{\partial F\left(NET_{jl}\right)}{\partial NET_{jl}}\right|_{NET_{jl}} = \left(1 - OUT_{jl}^2\right)\Big|_{OUT_{jl}} \qquad (16)$$

Аналогично рассчитывается производная сигмоиды (функции Ферми):

$$F\left(NET\right) = \sigma\left(NET\right) = \frac{1}{1+e^{-NET}} \qquad (17)$$

$$\frac{\partial F\left(NET\right)}{\partial NET} = e^{NET}\left(\frac{1}{1+e^{-NET}}\right)^2 = \frac{e^{-NET}+1}{\left(1+e^{-NET}\right)^2} - \overline{\left(1\right.} \qquad (18)$$

$$= \sigma\left(NET\right)\left(1-\sigma\left(NET\right)\right)$$

$$\left.\frac{\partial F\left(NET_{jl}\right)}{\partial NET_{jl}}\right|_{NET_{jl}} = OUT_{jl}\left(1-OUT_{jl}\right)\Big|_{OUT_{jl}} \qquad (19)$$

Благодаря тому, что для сигмоиды и гиперболического тангенса производная рассчитывается через значение выхода нейрона, эти две функции чаще всего применяются в сетях, обучаемых методом обратного распространения.

Значения производных $\frac{\partial E}{\partial w_{ijl}}$ и $\frac{\partial E}{\partial \theta_{jl}}$, рассчитанные по формулам (12)-(19), позволяют ввести коррекции Δw_{ijl} и $\Delta\theta_{jl}$ для нейронов последнего слоя. Как обобщить полученные результаты для оставшихся слоев?

Для последнего слоя:

$$\frac{\partial E}{\partial x_{ijl}} = \frac{\partial E}{\partial OUT_{jl}}\frac{\partial OUT_{jl}}{\partial NET_{jl}}\frac{\partial NET_{jl}}{\partial x_{ijl}} \qquad (20)$$

Здесь $\frac{\partial E}{\partial OUT_{jl}}$ и $\frac{\partial OUT_{jl}}{\partial NET_{jl}}$ уже рассчитаны (11), (13).

$$\frac{\partial NET_{jl}}{\partial x_{ijl}} = \frac{\partial\left(\sum_i w_{ijl}x_{ijl} - \theta_{jl}\right)}{\partial x_{ijl}} = w_{ijl} \qquad (21)$$

Но производная по входному значению x_{ijl} для последнего слоя совпадает по смыслу с производной по соответствующему выходу для предыдущего слоя:

$$\frac{\partial E}{\partial x_{ijl}} = \frac{\partial E}{\partial OUT_{j(l-1)}} \qquad (22)$$

Это выражение обеспечивает рекурсивный переход от последующего слоя к предыдущему и является аналогом (11) для предыдущих слоев.

Мы получили полный набор формул обратного распространения, который дает значения компонент градиента для всех слоев и всех нейронов в сети. Зная вектор градиента, можно проводить обучение в виде итераций по формуле (5).

Итак, *метод обратного распространения* — способ быстрого расчета градиента функции ошибки. Расчет производится от выходного слоя к входному по рекуррентным формулам и не требует пересчета выходных значений нейронов.

Обратное распространение ошибки позволяет во много раз сократить вычислительные затраты на расчет градиента по сравнению с расчетом по определению градиента. Зная градиент, можно применить множество методов теории оптимизации, использующих первую производную. Применимы также квазиньютоновские методы, в которых строится матрица вторых производных H (гессиан) на основе нескольких последовательных значений градиента. Быстрый расчет градиента необходим во

многих методах оптимизации (обучения), поэтому значение алгоритма обратного распространения в теории нейросетей велико.

Дальнейшее развитие алгоритма

В (11) суммирование по *s* обычно опускают, рассчитывают градиент ошибки для одного образа, а не всего обучающего множества. После расчета градиента можно либо сразу ввести поправки в веса и пороги по одному предъявленному образу, либо усреднить поправки по всем образам обучающего множества (*пакетная коррекция*). Нужно помнить, что усреднение требует запоминания одного вещественного числа на каждый параметр сети для хранения усредняемой коррекции, что существенно увеличивает объем памяти, занимаемый сетью.

Как и в однослойном перцептроне, можно использовать расписание обучения, $\varepsilon = \varepsilon(t)$, где t — время обучения. Это повышает скорость и точность сходимости во многих случаях.

Обратное распространение может применяться и в том случае, когда у функции активации несколько параметров, например, $F(NET) = \mathrm{th}(\beta\, NET - \theta)$. Компонента градиента $\frac{\partial E}{\partial \beta}$ рассчитывается аналогично остальным производным. Конечно, ввод дополнительных параметров может быть избыточным с точки зрения представимости выходной функции. Например, параметр β в данном примере эквивалентен умножению всех весовых коэффициентов данного нейрона на β. Но иногда избыточные параметры повышают скорость сходимости алгоритма. Например, в нашем случае может потребоваться столько коррекций весов, сколько входов у нейрона, чтобы добиться того же эффекта, который даст изменение β на одной итерации.

В 1987 г. Паркером предложен вариант алгоритма с производными второго порядка, дающий быструю сходимость, когда функция зависимость ошибки от параметров сети близка к квадратичной. Было также доказано, что использование производных высших порядков не дает выигрыша в обучении.

В 1987 г. Сторнетта и Хьюберман показали, что симметричный диапазон (например, от -1 до 1) изменения весов и сигналов в сети дает прирост скорости обучения на 30-50%. Функция активации, конечно, должна быть симметричной, подходит, например, гиперболический тангенс. Сигмоида может использоваться после симметрирования: $\sigma' = -\frac{1}{2} + \sigma(NET)$.

Было предложено множество ускоренных видов обратного распространения, но практическое применение получили в основном алгоритмы QuickProp и RProp [].

Паралич сети

Если один из весов при обучении получает слишком большое значение, то при обычных значениях этого входа выход нейрона окажется в насыщении, т.е. будет близок к предельному значению функции активации. Выход нейрона будет мало зависеть от *w*, и поэтому производная $\frac{\partial E}{\partial w} \approx 0$. Обучение по этому весу будет очень медленным, ведь изменение веса пропорционально производной. Выходной сигнал нейрона будет мало зависеть не только от веса, но и от входного сигнала *x* данного нейрона, а производная по x участвует в обратном распространении ошибки. Следовательно, предшествующие нейроны тоже будут обучаться медленно. Такое замедление обучения называется *параличом сети*.

Чтобы избежать паралича при обучении, можно:

1. Уменьшить размер шага по W *и* Θ. При этом увеличится время обучения.

2. В области больших весов отказаться от зависимости $\Delta w \sim \frac{\partial E}{\partial w}$, т.е. считать что длина шага не связана с модулем градиента.
3. Применять эвристические правила для ограничения роста весов. Эти правила пока не систематизированы, нет выкладок, оправдывающих тот или иной выбор ограничений.

Выбор длины шага

Вообще говоря, лучший выбор длины шага — точная одномерная оптимизация функции E вдоль вектора антиградиента. Требуется найти минимум функции

$$E_1(\varepsilon) = E\left(-\frac{\nabla E(\mathbf{P})\big|_{\mathbf{P}_0}}{\left|\nabla E(P)\big|_{\mathbf{P}_0}\right|}\varepsilon + \mathbf{P}_0 \right)$$

где P_0 — текущий вектор параметров, $\varepsilon > 0$ — независимая переменная. Найденное значение $\varepsilon_{ОПТ} : \left\{ \min_{\varepsilon} E_1(\varepsilon) \right\}$ даст требуемую длину шага на данной итерации. Рис. .

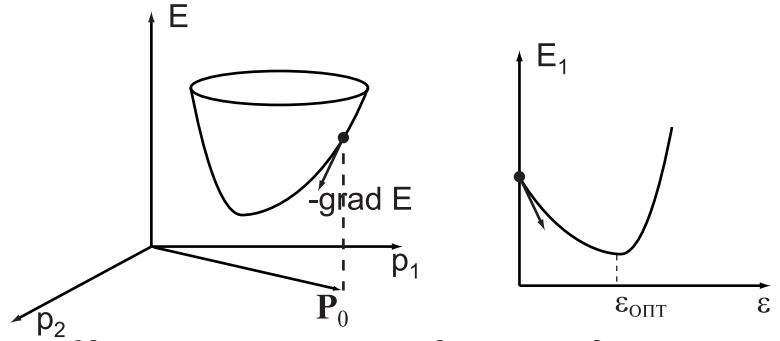


Рис. . Одномерная оптимизация для поиска длины шага.

Есть множество способов для решения задачи одномерной оптимизации. К ним относятся: метод деления пополам, метод Ньютона, чисел Фибоначчи, золотого сечения. Подробное описание, например, в [Гилл85].

Применительно к нейросетям все эти методы имеют недостаток: каждый расчет значения E требует больших затрат времени и расчета выходов всех нейронов в сети. Поэтому часто оказывается выгоднее вообще не проводить одномерного поиска, а взять шаг, пропорциональный градиенту $\varepsilon = \mathrm{const}$ в (5) или априорную зависимость от времени обучения $\varepsilon = \varepsilon(t)$ для коэффициента пропорциональности между длиной шага и градиентом. Неудачный выбор априорных данных может привести:

1) к неточности обучения: оказавшись в окрестности локального минимума, когда требуются малые длины шага для точной настройки параметров, алгоритм с большим шагом даст неточные значения параметров;
2) к медленному обучению: если шаг слишком малый, обучение может стать недопустимо медленным;
3) к отсутствию сходимости, параличу сети и другим проблемам при очень большой длине шага;

Локальные минимумы

Как и любой градиентный алгоритм, метод обратного распространения "застревает" в локальных минимумах функции ошибки, т.к. градиент вблизи локального минимума стремится к нулю. Шаг в алгоритме обратного распространения выбирается неоптимально. Точный одномерный поиск дает более высокую скорость сходимости.

Возникает интересное явление, которое оправдывает неоптимальный выбор шага. Поверхность функции $E(P)$ имеет множество долин, седел и локальных минимумов. Поэтому первый найденный минимум редко имеет малую величину E, и чем больше нейронов и синапсов в сети, тем меньше вероятность сразу найти глубокий минимум целевой функции. Если же шаг выбирается неоптимально, то он может оказаться достаточно большим, чтобы выйти из окрестности данного локального минимума и попасть в область притяжения соседнего минимума, который может оказаться глубже. Благодаря этому алгоритм способен находить более глубокие минимумы целевой функции, что повышает качество обучения.

Есть другой способ преодоления локальных минимумов — *обучение с шумом*. Будем выбирать коррекцию для весов в виде:

$$\Delta w = -\varepsilon \frac{\partial E}{\partial w} + n$$

где n — случайная величина, имеющая нулевое математическое ожидание и небольшую дисперсию. Часто используется гауссовское распределение для n. Казалось бы, добавление шума должно снижать точность обучения. Так и происходит, скорость сходимости к данному локальному минимуму (если он единственный) снижается. Но если поверхность $E(\mathbf{P})$ сложная, то шум помогает "вырваться" из данного локального минимума (выход из минимума тем вероятнее, чем меньше размеры его области притяжения), и найти более глубокий, возможно — глобальный минимум.

Обучение с шумом снижает вероятность остановки алгоритма в неглубоком локальном минимуме целевой функции.

Чувствительность к порядку предъявления образов
Временна́я неустойчивость

Алгоритм требует, чтобы *все* обучающие образы предъявлялись перед каждой коррекцией параметров. Это следует из необходимости суммировать функцию ошибки по всем образам из обучающего множества. В этом случае алгоритм всегда сходится, хотя количество итераций для сходимости может оказаться сколь угодно большим. Если же суммирования по s нет, и коррекции параметров проводятся после предъявления *каждого* образа, или даже после расчета *каждой* компоненты градиента E, то алгоритм может не сойтись вообще, если:

а) образы предъявляются не в случайном порядке. Должно быть $s = \mathrm{random}(S)$ на каждой итерации, если же есть простая закономерность в выборе s, алгоритм часто расходится;

б) обучающее множество постоянно меняется, и каждый образ предъявляется малое количество раз; это встречается для систем, работающих в реальном времени, например, систем управления сложными системами или систем прогнозирования в реальном времени; расходимость в этом случае называется *временно́й неустойчивостью*.

Проблема а) решается случайным предъявлением образов или применением алгоритма обучения с пакетной коррекцией. Проблема б) изначально присуща методу обратного распространения и устраняется выбором другого алгоритма обучения или другой модели нейронной сети.

Примеры применения перцептронов

I. Предсказание псевдослучайных последовательностей.

Существуют простые рекуррентные зависимости, генерирующие псевдослучайные последовательности чисел. Например, модель, называемая логистической картой, в которой следующее значение последовательности $x(t+1)$ связано с текущим $x(t)$:

$$x(t+1) = F(x(t)) \qquad F(x) = 4x(1-x), \quad 0 \le x \le \qquad (1)$$

График функции $F(x)$ показан на рис. . Она имеет максимум F(0,5) = 1.

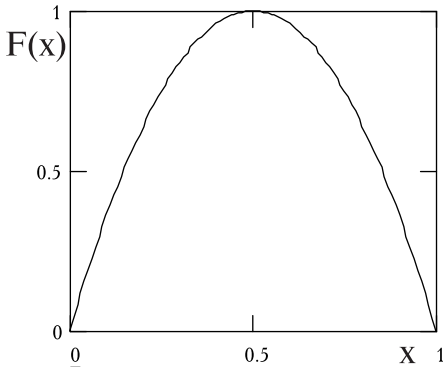


Рис. . Функция, порождающая псевдослучайную последовательность.

Если задать x(0) в интревале (0, 1), то по рекуррентной формуле (1) получим псевдослучайную последовательность, элементы которой лежат в интервале (0, 1). Пример такой последовательности, для начального значения x(0)=0,2 — на рис. .

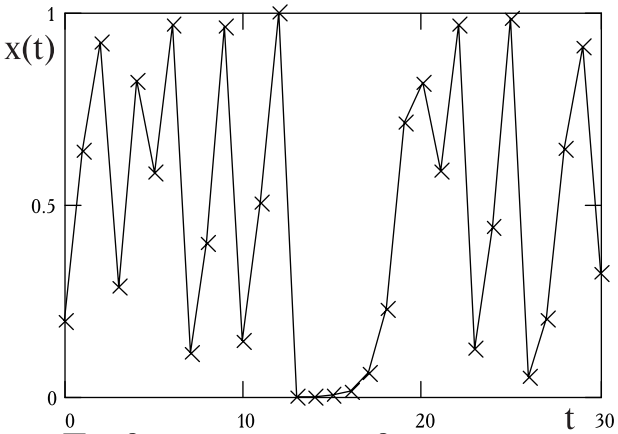


Рис. . Псевдослучайная последовательность, полученная по формуле (1).

Задача состоит в том, чтобы по конкретной реализации случайной последовательности x(t) (рис.), получить нейронную сеть, способную генерировать правильные псевдослучайные последовательности, т.е. для любого x, поданного на вход сети, давать x(t+1) на выходе.

Для решения задачи использовалась сеть с одним входом и одним выходом. Сеть содержала один скрытый слой из 5 нейронов. Применялась аналогичная сигмоидальной функция активации:

$$f\left(NET\right)=\frac{1}{1+e^{2NET}}=\frac{1}{2}\left(1+\text{th}\left(NET\right)\right)$$

Проводилось обучение методом обратного распространения с обучающим множеством, содержащим 1000 точек, т.е. 1000 известных отображений $x\left(t\right)\to x\left(t+1\right)$. Сеть также имела прямые синапсы со входа на выход, минуя скрытый слой.

В результате обучения сеть создала следующее отображение со входа на выход:

$$F^{*}\left(x\right)=-0,64\,f\left(-1,11x-0,26\right)-1,3\,f\left(2,22x-1,\right.$$
$$-2,265\,f\left(3,91x+4,82\right)-3,905\,f\left(2,46x-3,05\right)+ \qquad (2)$$
$$\left.+5,99\,f\left(1,68x+0,6\right)+0,31x-2,04\right.$$

Созданное сетью отображение аналитически совсем не похоже на исходную функцию $F(x)$. Тем не менее, отклонение F от F^{*} в рабочем интервале (0, 1) оказывается малым. Графики функций F(x) (пунктир) и $F^{*}(x)$ (непрерывная линия) представлены на рис. .

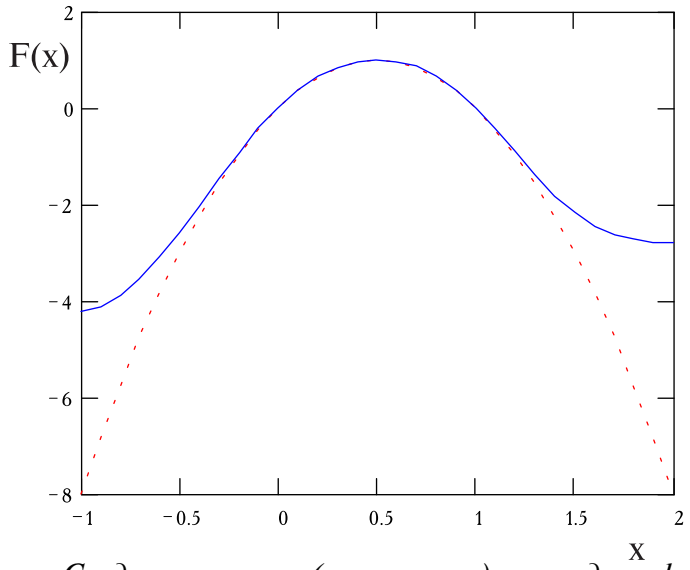


Рис. . Созданная сетью (непр. линия) и исходная функция F(x) (пунктир).

Из графиков рис. следует:

1) на рабочем участке сеть правильно *выделила закономерность* из входных данных; график F практически совпадает с F^*; разумеется, при правильно изученной функции F^* сеть способна правильно прогнозировать последовательность (1) с любым x(0).

2) на основании неявных предположений о функции F (непрерывность и дифференцируемость) сеть адекватно прогнозировала функцию F, т.е. проявила способность к *обобщению*. Эта способность не присуща сети самой по себе, а определяется выбором функции активации. Если бы в качестве функции активации была выбрана, к примеру, ступенька с линейной частью, функция F* оказалась бы недифференцируемой.

Данное исследование проведено Лапедесом и Фарбером в 1987 г.

II. Предсказание вторичной структуры белков

Молекулы белков построены из аминокислотных остатков. В живых белковых молекулах встречаются 20 различных аминокислот.

Различают три уровня структуры белковой молекулы. *Первичная структура* — порядок аминокислотных остатков в белковой молекуле, когда она растянута в линейную цепь. Аминокислотные остатки обозначаются трехбуквенными сочетаниями. Пример первичной структуры:

Лиз—Глу—Тре—Ала—Ала

—соответствует цепи, состоящей из аминокислотных остатков: лизил—глутаминил—треонил—аланил—аланил.

Структура белковой молекулы в виде линейной цепи оказывается энергетически не самой выгодной. Благодаря изменению формы молекулы близкие участки цепи формируют несколько характерных структур: α-спираль и β-формы: параллельную и антипараллельную. Эти формы образуют *вторичную* структуру белковой молекулы. Она зависит от порядка аминокислотных остатков в молекуле. *Третичная* структура — дальний порядок в молекуле, определяет, в каком порядке вторичные структуры образуют клубок или глобулу — общий вид молекулы.

Задача ставится так: предсказать участки белковой цепи, имеющие определенную вторичную структуру, в данном случае α-спираль. Для предсказания используется только информация о последовательности аминокислотных остатков в молекуле.

Для кодирования информации об одном аминокислотном остатке в нейросети используем 20 двоичных входов, например:

Глицил 00000010...0

Аланил 0.....01000

Одновременно в сеть вводится информация о 51 последовательном аминокислотном остатке, со сдвигом, как в **примере 3** формализации задач. Всего получается 51 * 20 = 1020 входов. Сеть форми-

рует единственный выход — вероятность того, что участок молекулы, заданный последовательностью из 51 входных аминокислотных остатков, имеет вид α-спирали. В экспериментах сеть состояла из 40 скрытых нейронов в одном слое, всего 40 000 весов и пороговых уровней.

Для обучения бралась известная информация о 56 белках. Для ускорения начального этапа обучения, когда ошибка очень высока, сначала обучающее множество состояло из данных лишь об одном (n=1) белке из 56. Остальные 55 использовались для контроля качества обучения. Когда ошибка немного снижалась, в обучающее множество добавлялась информация еще об одной молекуле и т.д. На завершающем этапе в обучении участвовали все 56 (n=56) белков.

Зависимость правильных предсказаний по неизвестным сети молекулам от размера n обучающего множества — на рис. .

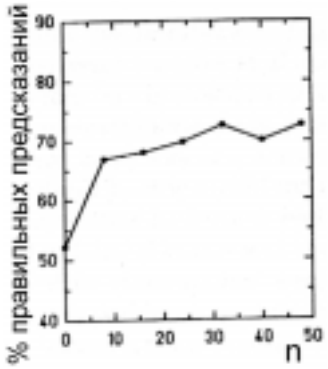


Рис. . Точность предсказания в зависимости от размера обучающего множества.

В результате обучения сеть стала давать лучшую точность, чем известные математические методы для предсказания вторичной структуры. Это редкий случай, когда нейросети превосходят математические методы по точности.

Пример предсказания структуры для молекулы родопсина (ярко-красного фоточувствительного белка сетчатки) — на рис. .

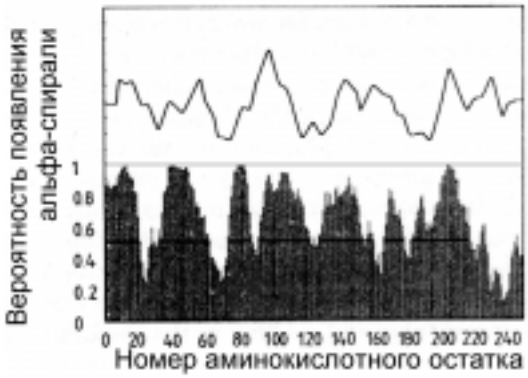


Рис. . Пример предсказания α-спиральных структур в молекуле родопсина.

Исследование проведено Боором и его коллегами в 1988 г.

III. Синтез речи: NET-talk

В 1987 г. Сейновский, Розенберг провели эксперименты по преобразованию текста из английских букв в фонемы. Задача синтеза речевого сигнала из фонем гораздо проще, чем преобразование текста в фонемы. Полученное нейронной сетью фонетическое представление затем "прочитывалось" вслух одним из многочисленных коммерческих синтезаторов речи.

Для преобразования текста в фонемы применялся многослойный перцептрон с одним скрытым слоем из 80 нейронов. На вход одновременно подавалась последовательность из 7 букв, каждая из которых кодировалась группой из 29 входов (26 букв алфавита + знаки препинания). Сеть имела 26 выходов для представления одной из 26 фонем с различной окраской звучания, сюда же относились фонемы-паузы.

Сеть прошла 50 циклов обучения по обучающему множеству из 1024 английских слов, и достигла 95% вероятности правильного произношения для слов из этого множества, и вероятности 80% на

неизвестных сети словах. Если учесть скромность затрат на реализацию эксперимента, по сравнению с коммерческими синтезаторами, то этот результат очень хорош.

Динамическое добавление нейронов

Адекватный выбор количества нейронов и слоев — серьезная и нерешенная проблема для нейронных сетей. Основным способом выбора остается прямой перебор различного количества слоев и определение лучшего. Для этого требуется каждый раз по-новому создавать сеть. Информация, накопленная в предыдущих сеансах обучения, теряется полностью. Начинать перебор количества нейронов можно как с заведомо избыточного, так и с недостаточного. Независимо от этого, новая созданная сеть с другим количеством нейронов требует полного переобучения.

Динамическое добавление нейронов состоит во включении нейронов в действующую сеть без утраты ее параметров и частично сохраняет результаты, полученные в предыдущем обучении.

Сеть начинает обучение с количеством нейронов, заведомо недостаточным для решения задачи. Для обучения используются обычные методы. Обучение происходит до тех пор, пока ошибка не перестанет убывать и не выполнится условие:

$$\begin{cases} \dfrac{E(t)-E(t-\delta)}{E(t_0)} < \Delta_T \\ t \geq t_0 + \delta \end{cases} \qquad ()$$

где t — время обучения; Δ_T — пороговое значение убыли ошибки; δ — минимальный интервал времени обучения между добавлениями новых нейронов; t_0 — момент последнего добавления;

Когда выполняются оба условия (), добавляется новый нейрон. Веса и порог нейрона инициализируются небольшими случайными числами. Обучение снова повторяется до тех пор, пока не будут выполнены условия (). Типичная зависимость ошибки от времени обучения приведена на рис. . Моменты добавления новых нейронов отмечены пунктиром. После каждого добавления ошибка сначала резко возрастает, т.к. параметры нейрона случайны, а затем быстро сходится к меньшему значению.

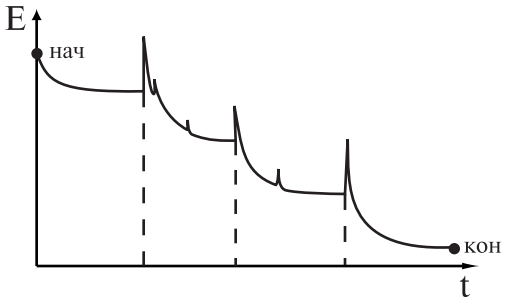


Рис. . Типичная зависимость ошибки от времени обучения при добавлении нейронов.

Интересно, что общее время обучения (от момента **нач** до **кон**) обычно оказывается лишь в ***1,4*** раза больше, чем если бы в сети *сразу* было нужное количество нейронов. Эта цифра показывает, что информация, накопленная в сети, не теряется полностью при добавлении нейрона со случайными параметрами.

Способность нейронных сетей к обобщению

Обобщение — способность сети давать близкий к правильному результат для входных векторов, которых не было в обучающем множестве. Если бы нейросети не обладали такой способностью, они были бы лишь механизмом запоминания, а не обработки информации. Но важнейшее качество нейросетей — способность дать хороший результат для векторов, с которыми сеть раньше не встречалась.

Условия и предпосылки для успешного обобщения:

1. Обобщенный выходной вектор **y**(**x**) для известных сейчас сетей никогда не бывает принципиально новым. Он является результатом неявных допущений об отображении $X \to Y$. Типичное допущение — непрерывность и дифференцируемость функции **y**(**x**). Оно вводится, когда выби-

рается функция активации в виде гиперболического тангенса или сигмоиды. Т.к. функция активации гладкая, то и **y**(**x**) будет гладкой. Кроме этого, обобщенный результат всегда оказывается простейшим в некотором смысле, в зависимости от конструкции сети.

2. Неизвестные входные вектора должны не слишком отличаться от векторов обучающего множества. Пример — график рис. . Аппроксимированная функция F*(x) совпадает с исходной F(x) на рабочем участке, но по мере удаления от исходного интервала, точность аппроксимации падает.
3. Основной закон, по которому сетью должно быть проведено обобщение, не должен быть скрыт несущественными закономерностями в обучающем множестве. Поэтому входы и выходы сети должны быть подготовлены так, чтобы максимально выявить закон, по которому они должны быть обобщены. Шум, например, полезно отфильтровать, а двоичное кодирование данных заменить кодированием по номеру канала, принятому в нейросетях.

Обучение без учителя

Обратное распространение хорошо работает во многих случаях. Но, как и во всех алгоритмах с учителем, для обратного распространения нужны известные входные и выходные вектора, а эта информация не всегда доступна.

Вопрос о биологической правдоподобности обучения с учителем также открыт. Конечно, алгоритма, подобного обратному распространению, в биологических сетях не существует. Нейроны зрительной коры, например, учатся реагировать на световые импульсы лишь под действием самих импульсов, без внешнего учителя. Однако высшие этапы обучения, например, у детей, невозможны без "учителя" в лице его родителя. Кроме того, отдельные области в мозге вполне могут выполнять роль "учителей" для других, управляя их активностью. Поэтому нельзя однозначно сказать, какой тип обучения биологически правдоподобнее, с учителем или без. В биологических сетях передача сигналов направленная, поэтому обратное распространение ошибки возможно только с помощью обратных связей. Кроме того, в сети отсутствует единый "супервизор", управляющий коррекциями параметров. Таким образом, обратное распространение биологически неправдоподбно.

Важный принцип, по которому строятся биологические нейронные сети — *локальность* при обработке информации. Выход нервной клетки определяется *только* ее внутренним состоянием и входными сигналами. Существует глобальное управление биологической нейросетью, например, гормональная регуляция. Встречается модуляция и синхронизация деятельности скоплений специальными нервными центрами. Эти механизмы не нарушают принципа локальности и непосредственно не являются "учителем".

Внешняя среда может выступать "учителем" для живой нейронной сети, хотя и косвенно. Среда обычно не дает правильного решения, к которому сеть должна прийти сама, а лишь направляет действия, "поощряя" и "наказывая" животное в зависимости от правильности реакции.

При обучении без учителя сеть имеет для обучения лишь известные входные вектора. Правильные значения выходных векторов неизвестны. Какие вектора будут сформированы на выходе, зависит от алгоритма обучения. Обучающее множество состоит из S известных входных векторов $\{\mathbf{x}^s\}$, $s=1...S$, в процессе обучения сеть учится формировать выходные вектора $\mathbf{y}^s$ для каждого вектора из обучающего множества $\{\mathbf{x}^s\}$.

Сеть с линейным поощрением

Созданы сети, промежуточные по отношению к обучению с учителем и без него. В качестве такой модели рассмотрим *сеть с линейным поощрением*. Эта модель обучается с учителем, т.е. требует знания и выходных, и входных векторов при обучении. Однако в обратном направлении распространяется ограниченный объем информации, меньший, чем при обратном распространении.

Все сигналы в сети лежат в интервале [0, 1]. Сеть послойно-полносвязная, как и многослойный перцептрон и содержит три слоя нейронов.

Последний, третий слой состоит из обычных формальных нейронов с детерминированным поведением и непрерывными выходными сигналами:

$$\begin{cases} y_{j3} = \sigma\left(NET_{j3}\right) \\ NET_{j3} = \sum_i w_{ij3} x_{ij3} \end{cases}$$

Скрытый слой состоит из *стохастических* нейронов с двумя значениями выхода, 0 и 1. Каждое из выходных значений принимается с вероятностями:

$$\begin{cases} p\left(y_{j2} = 1\right) = \sigma\left(NET_{j2}\right) \\ p\left(y_{j2} = 0\right) = \sigma\left(-NET_{j2}\right) \\ NET_{j2} = \sum_i w_{ij2} x_{ij2} \end{cases}$$

Первый слой не выполняет вычислений, а лишь распределяет входные сигналы по нейронам второго слоя.

Обучающее множество $\left\{\mathbf{x}^s, \mathbf{d}^s\right\}$ содержит известные пары выходных и входных векторов, как и в алгоритме обратного распространения, $s = 1...S$ — номер эталона в обучающем множестве. Функцию ошибки выберем нормированной и линейной, чтобы ее можно было трактовать как вероятность:

$$E^s = \frac{1}{N_O} \sum_i \left(y_{j3}^s - d_j^s\right)$$

где N_O — количество выходов сети. За счет нормирования $E^s \in [0; 1]$.

Выходной слой обучается обычным способом, коррекции весов выходного слоя:

$$\delta w_{ij3} = \varepsilon\left(d_j^s - y_j^s\right) \cdot \left.\frac{d\sigma\left(NET_{j3}\right)}{dNET_{j3}}\right|_{NET_{j3}} \cdot y_j^s$$

Второй слой обучается с поощрением и наказанием. Введем *градационный сигнал r*, характеризующий качество выходного результата. Возможны два варианта.

1. Дискретный градационный сигнал с двумя возможными значениями, 0 и 1, с вероятностями

$p\left(r = 0\right) = E^s$, $\quad p\left(r = 1\right) = 1 - E^s$

2. Непрерывный градационный сигнал, $r = 1 - E^s$.

В обратном направлении распространяется только градационный сигнал, а не полная информация об ошибке по каждому выходу, как в обратном распространении.

Коррекции весов во втором слое выбираются в виде:

$$\delta w_{ij2} = r\varepsilon\left(y_{j2} - \sigma\left(NET_{j2}\right)\right) x_{ij2} + \left(1 - r\right)\lambda\varepsilon\left(1 - y_{j2}\right.$$

где ε — скорость обучения; $\lambda << 1$ — цена ошибки (в нейронных сетях, в отличие от задач поиска сигналов, цена ошибки намного меньше цены правильного решения).

Чтобы лучше понять выражение (), запишем его в алгоритмическом виде для дискретного *r*:

Если $r = 1$ то { если $y_{j2} = 1$, то вес увеличивается, иначе уменьшается }

Если $r = 0$ то { если $y_{j2} = 1$, то вес уменьшается, иначе увеличивается }

— такое правило обучения очень напоминает уже знакомое правило Хэбба.

Существует несколько других промежуточных моделей, но сеть с линейным поощрением — одна из немногих успешных.

Задача классификации
Сети Кохонена

Задача классификации заключается в разбиении объектов на классы, когда основой разбиения служит вектор параметров объекта. Объекты в пределах одного класса считаются эквивалентными с точки зрения критерия разбиения. Сами классы часто бывают неизвестны заранее, а формируются динамически (как, например, в сетях Кохонена). Классы зависят от предъявляемых объектов, и поэтому добавление нового объекта требует корректирования системы классов.

Будем характеризовать объекты, подлежащие классификации, вектором параметров $\mathbf{x}^p \in X$, имеющим N компонент, компоненты обозначаем нижним индексом: $\mathbf{x}^p = \left(x_1^p, ..., x_N^p\right)^T$. Вектор параметров — единственная характеристика объектов при их классификации.

Введем множество классов $C^1, ..., C^M = \left\{C^m\right\}$ в пространстве классов C:

$\left(C^1 \cup C^2 ... \cup C^M\right) \subset C$

Пространство классов может не совпадать с пространством объектов X и иметь другую размерность. В простейшем случае, когда пространства классов и объектов совпадают, $X = C$, классы представляют собой области пространства X, и объект $\mathbf{x}^p$ будет отнесен к одному из классов m_0, если $\mathbf{x}^p \in C^{m_0}$. В общем случае X и C различны.

Определим ядра классов $\left\{\mathbf{c}^m\right\} = \mathbf{c}^1, ..., \mathbf{c}^m$ в пространстве классов C, как объекты, типические для своего класса. К примеру, если для классификации по национальности выбрать параметры {цвет глаз, рост, цвет волос}, то ядро класса "русский" может иметь параметры {голубоглазый, рост 185, волосы русые}, и к этому классу можно отнести объект с параметрами {светло-голубоглазый, рост 182, волосы темно-русые}, т.к. из ядер "русский", "эстонец", "киргиз" параметры объекта больше всего похожи на ядро "русский".

Очевидно, что близость объекта к ядру необходимо оценивать численно. Введем *меру близости* $d\left(\mathbf{x}^p, \mathbf{c}^m\right)$ — скалярную функцию от объекта и ядра класса, которая тем меньше, чем больше объект похож на ядро класса. Могут потребоваться вспомогательные меры близости, определенные для двух объектов, $d\left(\mathbf{x}^{p_1}, \mathbf{x}^{p_2}\right)$, и для двух ядер классов, $d\left(\mathbf{c}^{m_1}, \mathbf{c}^{m_2}\right)$.

Чаще всего применяется эвклидова мера: $d\left(\mathbf{x}, \mathbf{y}\right) = \sum_i \left(x_i - y_i\right)^2$

или "city block": $d\left(\mathbf{x}, \mathbf{y}\right) = \sum_i \left|x_i - y_i\right|$

Задавшись числом классов M, можно поставить задачу классификации: найти M ядер классов $\left\{\mathbf{c}^m\right\}$ и разбить объекты $\left\{\mathbf{x}^p\right\}$ на классы $\left\{C^m\right\}$, т.е. построить функцию $m(p)$ таким образом, чтобы минимизировать сумму мер близости:

$$\min\left\{D = \sum_p d\left(\mathbf{x}^p, \mathbf{c}^{m(p)}\right)\right\}$$

Функция $m(p)$, определяющая номер класса по индексу p множества объектов $\left\{\mathbf{x}^p\right\}$, задает разбиение на классы и является решением задачи классификации.

В простейшем случае $X = C$, пространство объектов X разбивается на области $\{C^m\}$, и если $\mathbf{x}^{p_0} \in C^{m_0}$, то $m(p_0) = m_0$, и объект относят к классу m_0.

Количество классов M может динамически меняться. При этом часто возникают ситуации, когда объекты распределены по классам неравномерно. Необходимо контролировать равномерность плотности ядер $\mathbf{c}^m$ в пространстве C и долю объектов, относящихся к каждому классу m_0. Конкретные требования должны выбираться в зависимости от задачи. При необходимости можно корректировать плотность слиянием и разделением ядер. Критерии для этого могут использоваться разные, и, в основном, эмпирические. Например, два класса могут сливаться в один, если мера близости их ядер меньше, чем средняя мера близости ядер и всех объектов в этих двух классах. Применяя аналогичные правила, надо помнить о "побочных эффектах". Приведенный критерий слияния может работать некорректно (рис.).

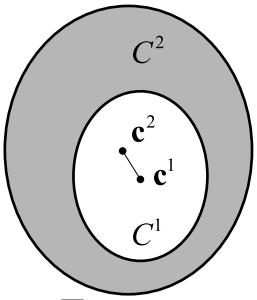


Рис. . Пример некорректной работы критерия слияния.

Мера близости ядер в этом случае много меньше, чем средняя мера близости между объектами и ядрами. В соответствии с правилом, классы будут объединены, хотя во многих задачах такое объединение необоснованно.

Полезно определить *диаметр класса* — максимальная мера близости между объектами данного класса.

Алгоритмы классификации

Большая часть алгоритмов — итерационные. Серьезный недостаток многих алгоритмов классификации — теоретическая необоснованность, отсутствие доказательств, что классификация будет правильной, не говоря уже о доказательстве оптимальности конкретного алгоритма.

Рассмотрим общий алгоритм с фиксированным количеством ядер M. Количество ядер выбирается заранее, исходя из конкретной задачи.

Начальные значения ядер $\mathbf{c}^1, ..., \mathbf{c}^m$ могут выбираться случайными, одинаковыми или по другим эвристическим правилам.

Каждая итерация алгоритма состоит из двух этапов:

1. При неизменных ядрах $\{\mathbf{c}^m\} = \text{const}$ ищем такое разбиение $m(p)$ объектов $\{\mathbf{x}^p\}$ на классы, чтобы минимизировать суммарную меру близости между объектами и ядрами их классов:

$$\min\left\{D = \sum_p d\left(\mathbf{x}^p, \mathbf{c}^{m(p)}\right)\right\}$$

Результат этапа — создание функции $m(p)$, разбивающей объекты на классы.

2. При неизменном разбиении $m(p)$ настраиваем ядра $\{\mathbf{c}^m\}$ так, чтобы в пределах каждого класса m_0 суммарная мера близости ядра этого класса и объектов, ему принадлежащих, была минимальной:

$$\min\left\{D = \sum_{p:m(p)=m_0} d\left(\mathbf{x}^p, \mathbf{c}^{m(p)}\right)\right\} \quad \text{для всех } m_0 = 1...M$$

Результат этого этапа — новый набор ядер $\{\mathbf{c}^m\}$.

Сеть Кохонена

Если для классификации применять нейронные сети, необходимо формализовать задачу. Самый очевидный способ: выберем в качестве входных данных вектор параметров единственного объекта. Результатом работы сети будет код класса, к которому принадлежит предъявленный на входе объект. В нейросетях принято кодирование номером канала. Поэтому сеть будет иметь M выходов, по числу классов, и чем большее значение принимает выход номер m_0, тем больше "уверенность" сети в том, что входной объект принадлежит к классу m_0. Полезно применить функцию активации SOFTMAX, тогда сумма выходов всегда будет равна единице. Каждый выход можно будет трактовать как вероятность того, что объект принадлежит данному классу. Все выходы образуют полную группу, т.к. сумма выходов равна единице, и объект заведомо относится к одному из классов.

Выберем евклидову меру близости (). В этом случае ядро класса, минимизирующее сумму мер близости для объектов этого класса, совпадает с центром тяжести объектов:

$$\mathbf{c}^{m_0} = \frac{1}{N(m_0)} \sum_{p:\, m(p)=m_0} \mathbf{x}^p$$

где $N(m_0)$ — число объектов $\mathbf{x}^p$ в классе m_0.

При разбиении на классы должна быть минимизирована суммарная мера близости для всего множества $\{\mathbf{x}^p\}$ входных объектов:

$$D = \sum_p \sum_i \left(\mathbf{x}_i^p - \mathbf{c}_i^{m(p)}\right)^2 =$$

$$= \sum_p \left[\left(\mathbf{x}^p, \mathbf{x}^p\right) - 2\left(\mathbf{x}^p, \mathbf{c}^{m(p)}\right) + \left(\mathbf{c}^{m(p)}, \mathbf{c}^{m(p)}\right)\right]$$

— расписано скалярное произведение. В этой сумме два слагаемых не зависят от способа разбиения и постоянны:

$$\sum_p \left(\mathbf{c}^{m(p)}, \mathbf{c}^{m(p)}\right) = \text{const}, \quad \sum_p \left(\mathbf{x}^p, \mathbf{x}^p\right) = \text{const}$$

Поэтому задача поиска минимума D эквивалентна поиску максимума выражения:

$$\min D \;\rightarrow\; \max \sum_p \sum_i x_i^p c_i^{m(p)}$$

Запишем вариант алгоритма классификации для поиска максимума этой функции:

1. Цикл: для каждого вектора $\mathbf{x}^p$ {

    2. Цикл: для каждого m {

        3. Рассчитать $\sum_i x_i^p c_i^m = D^{m,p}$.

    } // конец цикла

    4. Находим m_0, для которого $m_0 : \max_m \{D^{m,p}\}$

    5. Относим объект к классу m_0.

} // конец цикла

Такой алгоритм легко реализуется в виде нейронной сети. Для этого требуется M сумматоров, находящих все $D^{m,p}$, и интерпретатора, находящего сумматор с максимальным выходом.

Сумма $\sum_i x_i^p c_i^m$ очень напоминает взвешенную сумму $NET_{jl} = \sum_i w_{ijl} x_{ijl}$, рассчитываемую формальным нейроном. Выберем x_i^p в качестве входных сигналов (что мы, впрочем, уже сделали) и компоненты ядер c_i^m в качестве весовых коэффициентов w_{ijl}. Тогда каждый формальный нейрон с числом входов, равным числу компонент во входном векторе, будет давать на выходе одну из сумм $D^{m,p}$.

Чтобы определить класс, к которому относится объект, нужно выбрать среди всех нейронов данного слоя один с максимальным выходом — это осуществляет интерпретатор. Интерпретатор — или программа, выбирающая нейрон с максимальным выходом, или слой нейронов с латеральным торможением (описан в разделе о сети АРТ), состоящий из нейронов с обратными связями. На обычных ЭВМ программный интерпретатор эффективнее, т.к. латеральное торможение требует моделирования процесса во времени, что требует многих итераций.

Рассмотренная сеть нейронов, использующая евклидову меру близости для классификации объектов, называется *сетью Кохонена* (рис.).

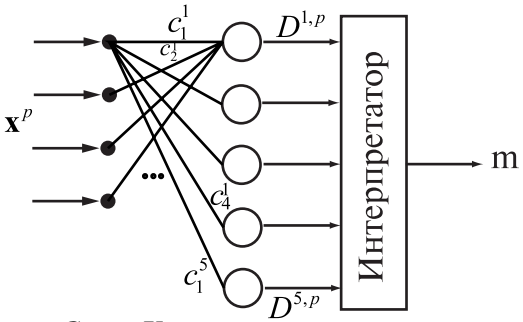


Рис. . Сеть Кохонена

Нейроны слоя Кохонена генерируют сигналы $D^{m,p}$. Интерпретатор выбирает максимальный сигнал слоя Кохонена и выдает номер класса *m*, соответствующий номеру входа, по которому интерпретатором получен максимальный сигнал. Это соответствует номеру класса объекта, который был предъявлен на входе, в виде вектора $\mathbf{x}^p$.

Ядра $\mathbf{c}^m$ являются весовыми коэффициентами нейронов. Каждый нейрон Кохонена запоминает одно ядро класса, и отвечает за определение объектов в своем классе, т.е. величина выхода нейрона тем больше, чем ближе объект к данному ядру класса.

Общее количество классов совпадает с количеством нейронов Кохонена. Меняя количество нейронов, можно динамически менять количество классов.

Нейроны Кохонена имеют линейную функцию активации. Если применить функцию SOFTMAX, то выход слоя Кохонена можно трактовать как вероятность принадлежности объекта к каждому из классов. Но применение SOFTMAX некорректно с точки зрения принципа локальности, т.к. вычисление этой функции активации требует знания *всех* выходов сети *каждым* из нейронов, а в реальной сети это не выполняется.

Входные вектора сети чаще всего нормируются:

$$\frac{\mathbf{x}^p}{\left|\mathbf{x}^p\right|} \to \mathbf{x}^p \quad \text{или} \quad \frac{\mathbf{x}^p}{\sum_p \left|\mathbf{x}^p\right|^2} \to \mathbf{x}^p \qquad ()$$

Возможны другие способы нормировки.

Обучение слоя Кохонена

Задача обучения — научить сеть активировать один и тот же нейрон для похожих векторов $\mathbf{x}^p$ на входе. Не важно, какой конкретно нейрон будет активирован.

1. Присвоение начальных значений

Обычно начальные значения в нейронных сетях выбираются малыми случайными числами. Для слоя Кохонена такой выбор возможен, но имеет недостатки. Разумеется, если ядра классов нормированы, то и начальные значения нужно нормировать.

Если веса инициализируются случайными значениями с равномерным распределением, то возникает проблема. Когда ядра распределяются равномерно, то в областях пространства X, где мало входных векторов, ядра будут использоваться редко, т.к. мало будет похожих векторов. В тех областях, где входных векторов много, плотность ядер окажется недостаточной, и непохожие объекты будут активировать один и тот же нейрон, т.к. более похожего ядра не найдется. Для устранения проблемы можно выделять ядра в соответствии с плотностью входных векторов. Но распределение входных векторов часто бывает заранее неизвестно. В этом случае помогает метод выпуклой комбинации, рассмотренный ниже.

2. Обучение сети

Если число входных векторов равно числу ядер (т.е. нейронов), то обучение не нужно. Достаточно присвоить ядрам значения входных векторов, и каждый вектор будет активировать свой нейрон Кохонена. Но чаще всего количество классов меньше числа входных векторов. В этом случае веса сети настраиваются итеративным алгоритмом.

Алгоритм аналогичен исходному алгоритму классификации, но коррекции весов проводятся после предъявления каждого входного вектора, а не после предъявления всех, как требует исходный алгоритм. Сходимость при этом сохраняется.

1. Присваиваем начальные значения весовым коэффициентам.
2. Подаем на вход один из векторов $\mathbf{x}^p$.
3. Рассчитываем выход слоя Кохонена, $D^{\mathrm{m,p}}$, и определяем номер выигравшего нейрона m_0, выход которого максимален, $m_0 : \max\limits_m D^{m,p}$.
4. Корректируем веса только выигравшего нейрона m_0:

$$\mathbf{w}_{m_0} := \mathbf{w}_{m_0} + \alpha\left(\mathbf{x}^p - \mathbf{w}_{m_0}\right) \qquad ()$$

— коррекция записана в виде векторного выражения (вектор весов $\mathbf{w}_{m_0}$ нейрона m_0 имеет столько компонент, сколько их у входного вектора $\mathbf{x}^p$). α — скорость обучения, малая положительная величина. Часто используют расписание с обучением, когда $\alpha = \alpha(t)$ монотонно убывает. Требования к $\alpha(t)$ те же, что и в случае многослойного перцептрона.

Веса корректируются так, что вектор весов приближается к текущему входному вектору. Скорость обучения управляет быстротой приближения ядра класса (вектора весов) ко входному вектору $\mathbf{x}^p$.

Алгоритм выполняется до тех пор, пока веса не перестанут меняться.

Метод выпуклой комбинации

Этот метод полезен при обучении, чтобы правильно распределить плотность ядер классов (векторов весов) в соответствии с плотностью входных векторов в пространстве *X*.

1. Присваиваем всем весам одно и то же начальное значение: $w_i^m = \dfrac{1}{\sqrt{n}}, \quad n = \dim X$. Вектора весов получают длину, равную единице, как требует нормировка. Все вектора весов одинаковы.
2. Задаем обучающее множество $\left\{\mathbf{x}^p\right\}$ и проводим обучение, но не с векторами $\mathbf{x}^p$, а с векторами $\beta(t)\mathbf{x}^p + \dfrac{1-\beta(t)}{\sqrt{n}}$, где t — время обучения, $\beta(t)$ — монотонно возрастающая функция, меняющаяся от 0 до 1 по мере обучения.

В начале обучения $\beta(t)=0$ и все обучающие вектора одинаковы и равны начальному значению весов. По мере обучения $\beta(t)$ растет и обучающие вектора расходятся из точки с координатами $\frac{1}{\sqrt{n}}$ и приближаются к своим конечным значениям $\mathbf{x}^p$, которые достигаются при $\beta(t)=1$. Каждый вектор весов "захватывает" группу или один обучающий вектор и отслеживает его по мере роста β.

Метод выпуклой комбинации дает правильное распределение плотности ядер. При этом в сети не остается "ненужных" необученных нейронов, которые бывают при обычном обучении. Когда вектор весов нейрона находится далеко от всех обучающих векторов, этот нейрон никогда не будет "выигрывать", и его веса не будут корректироваться при обучении. Выпуклая комбинация не оставляет в сети таких нейронов.

Примеры обучения

Рассмотрим примеры обучения сети Кохонена обычным методом и методом выпуклой комбинации.

В первом методе будем выбирать равномерно распределенные случайные векторы весов (ядер классов). На рис. представлен пример обучения. Точками обозначены вектора $\mathbf{x}^p$ обучающего множества, кружками — вектора весовых коэффициентов.

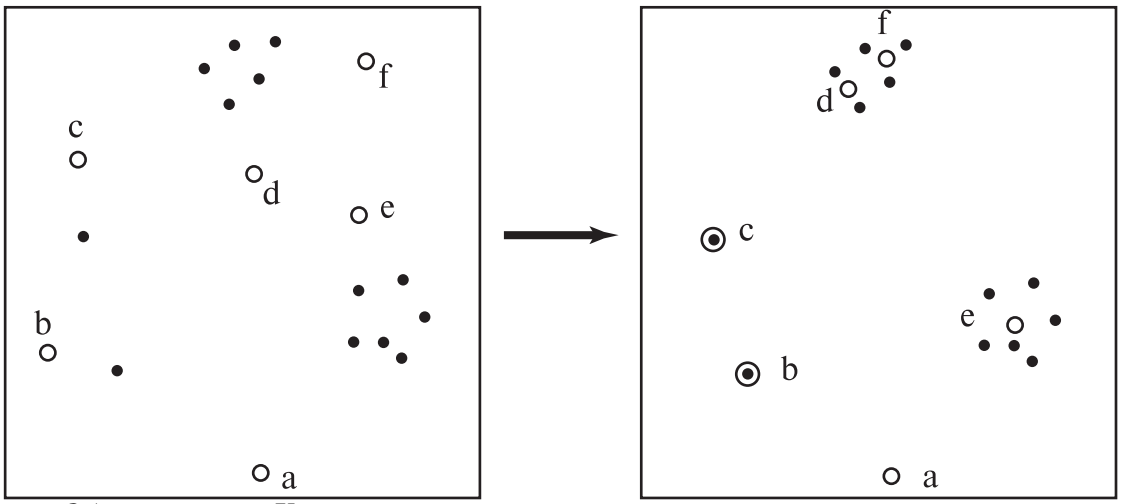


Рис. . Обучение сети Кохонена.

Вектор весов нейрона **a** не обучается, т.к. ни для одного из векторов обучающего множества этот нейрон не получает максимального выхода. Кроме того, в области из шести обучающих векторов (справа внизу) оказывается всего один вектор весов нейрона **e**. Это не соответствует высокой плотности обучающих векторов в этой области. Эти недостатки присущи обычному методу обучения сети Кохонена.

Разберем работу метода выпуклой комбинации. Последовательное изменение картины векторов и весов показано на рис. .

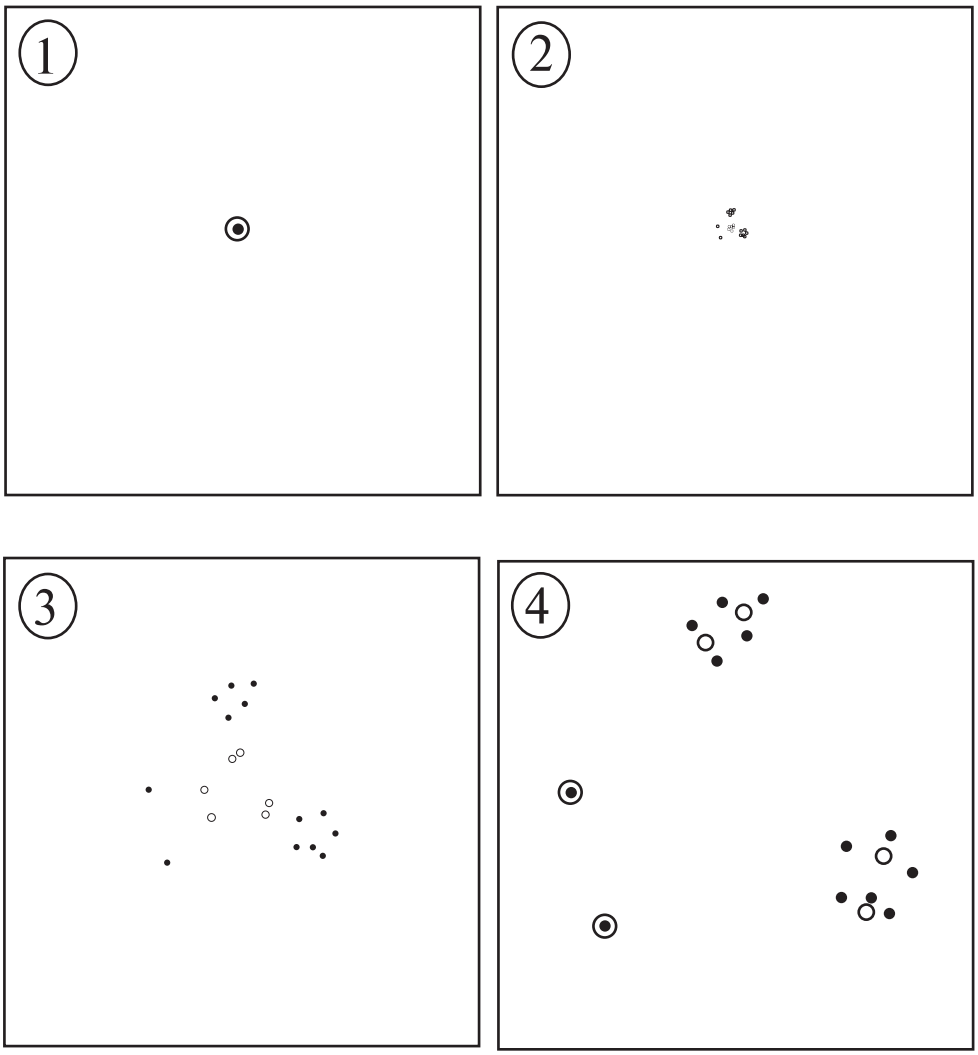


Рис. . Обучение методом выпуклой комбинации.

На первой схеме все векторы весов и обучающего множества имеют одно и то же значение. По мере обучения обучающие векторы расходятся к своим истинным значениям, а векторы весов следуют за ними. В итоге в сети не остается необученных нейронов и плотность векторов весов соответствует плотности векторов обучающего множества. Однако, процесс увеличения β требует многих итераций, и время обучения растягивается. Это существенный недостаток метода выпуклой комбинации.

Модификации алгоритма обучения

Чувство справедливости: чтобы не допустить отсутствие обучения по любому из нейронов, вводится "чувство справедливости". Если нейрон чаще других выигрывает "состязание", т.е. получает максимальный выход чаще, чем в 1 из M случаев, то его значение выхода искусственно уменьшается, чтобы дать возможность выиграть другим нейронам. Это включает все нейроны сети в процесс обучения.

Коррекция весов пропорционально выходу: в этой модификации корректируются не только веса выигравшего нейрона, но и всех остальных, пропорционально их нормированному выходу. Нормировка выполняется по максимальному значению выхода слоя или по его среднему значению. Этот метод также исключает "мертвые" нейроны и улучшает распределение плотности весов.

Режимы работы сети

Обычная сеть Кохонена работает в режиме *аккредитации*. Это означает, что активируется единственный нейрон Кохонена с максимальным значением выхода.

Можно не затормаживать остальные нейроны слоя Кохонена, а пронормировать выходные сигналы, например, функцией активации softmax:

$$OUT_j = \frac{e^{NET_j}}{\sum_i e^{NET_i}}$$

Тогда сумма всех выходов слоя будет равна единице и можно трактовать выходы, как вероятность отнесения объекта к каждому из классов.

Такой режим работы сети, когда активируется несколько нейронов одновременно, называется *режимом интерполяции*. Название режима объясняется тем, что если входной вектор $\mathbf{x}^p$ плавно меняется от одного вектора весов, $\mathbf{x}^p = \mathbf{w}^{m_1}$ к другому вектору весов $\mathbf{x}^p = \mathbf{w}^{m_2}$, то выход сети в режиме интерполяции (если применена функция softmax) будет плавно меняться от m_1 к m_2, т.е. классификация оказывается непрерывной. Если же сеть работает в режиме аккредитации, выход изменится от m_1 к m_2 скачкообразно.

Применение сети Кохонена для сжатия данных

Сеть Кохонена позволяет выделять похожие фрагменты данных в классы. Номер класса обычно занимает гораздо меньше места в памяти, чем ядро класса. Если передать получателю все ядра классов и номера классов, кодирующие каждый фрагмент данных, то данные могут быть восстановлены. При этом неизбежны потери, если число классов меньше числа различных фрагментов данных.

Применим сеть Кохонена для сжатия изображений. Представим изображение в виде прямоугольного массива точек — пикселов. Каждый пиксел имеет свою яркость. Разделим изображение на небольшие прямоугольные фрагменты, размером несколько пикселов. Создадим слой Кохонена, чтобы количество входов совпадало с количеством пикселов в одном прямоугольном фрагменте. Нейронов в слое Кохонена должно быть столько, сколько разновидностей фрагментов может встретиться в изображении. Если учесть, что различных фрагментов может быть $c^{n \cdot m}$, где с — число градаций яркости в изображении, *n* x *m* — размеры фрагмента в пикселах, и для фрагмента 8 x 8 при 16 градациях яркости может быть $7{,}9 \cdot 10^{28}$ различных фрагментов, то очевидно, что потери при классификации неизбежны.

Сеть учится активировать один и тот же нейрон для сходных фрагментов. Активируется тот нейрон, который соответствует классу, к которому отнесен данный фрагмент изображения. При обучении сеть сама формирует ядра классов, т.е. набор фрагментов, из которых строится изображение.

Пусть, например, в двух нейронах слоя Кохонена запомнены два фрагмента изображения (рис.) в виде весовых коэффициентов.

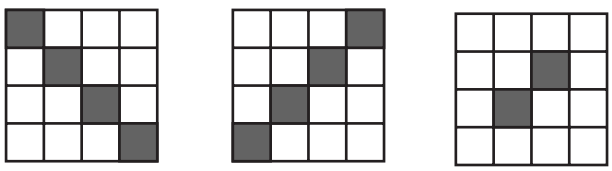


Рис. . два фрагмента изображения — ядра классов, запомненные нейронами Кохонена, и фрагмент, предъявленный на входе.

Предъявим в виде входного вектора фрагмент изображения, рис. .

Входной вектор больше похож на вектор весов, запомненный нейроном 2. Этот нейрон и будет активирован, поэтому выход сети будет равен *m*=2. Конечно, при восстановлении изображения по номеру класса, *m*=2, будет восстановлен фрагмент, запомненный нейроном 2, который не полностью совпадает с исходным фрагментом.

Для выбора оптимального количества классов и размера фрагмента необходимы дополнительные соображения. Чем больше фрагмент и чем меньше нейронов в слое Кохонена, тем выше коэффициент сжатия, и тем больше потери при восстановлении.

Этот же метод может быть применен для сжатия других типов данных, например, речевых сигналов. Но учитывая потери при сжатии, сеть Кохонена в исходной модели неприменима для данных, которые должны быть точно восстановлены. Типичные значения коэффициента сжатия для сети Кохонена — от 10 до 100.

Сеть встречного расспространения
Слой Гроссберга

Сеть встречного распространения (СВР) была предложена Робертом Хехт-Нильсеном в 1987 г. Она состоит из двух слоев нейронов: слоя Кохонена и слоя Гроссберга (рис.). Слой Кохонена работает в режиме интерполяции или аккредитации. Все слои полносвязны.

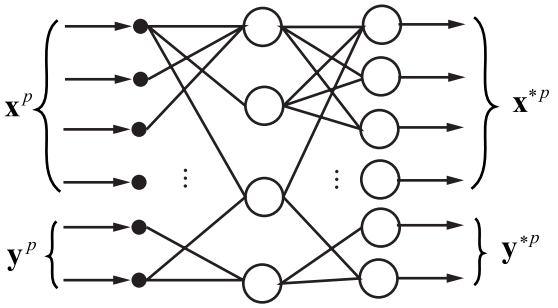


Рис. . Сеть встречного распространения.

Слой Гроссберга предназначен для совместной работы со слоем, дающим единственную единицу на выходе (как у слоя Кохонена в режиме аккредитации) или такой набор выходов, что их сумма равна единице (как слой Кохонена с функцией SOFTMAX в режиме интерполяции). Нейроны слоя Гроссберга вычисляют взвешенную сумму своих входов. Функция активации не используется (линейная). Слой Гроссберга дает на выходе линейную комбинацию своих векторов весов, коэффициенты комбинации задаются входами слоя Гроссберга.

Первый слой Кохонена функционирует так:

$$OUT_{j1} = \sum_i w_{ij1} x_{ij1} \qquad ()$$

— третий индекс обозначает слой: $l = 1$.

Второй слой Гроссберга работает по той же формуле, но смысл взвешенного суммирования другой:

$$OUT_{j2} = \sum_i w_{ij2} x_{ij2} \qquad ()$$

Учитывая, что $x_{ij2} = OUT_{i1}$:

$$OUT_{j2} = \sum_i w_{ij2} OUT_{i1}$$

— здесь суммирование проводится по всем выходам слоя Кохонена.

Если слой Кохонена в режиме аккредитации, то на слой Гроссберга поступает единичный выход только одного нейрона Кохонена: пусть i_0 — номер активного нейрона Кохонена:

$$\begin{cases} OUT_{i1} = 0, & i \neq i_0 \\ OUT_{i_0 1} = 1 \end{cases}$$

Выход слоя Гроссберга в этом случае: $OUT_{j2} = w_{i_0 j2}$, т.е. каждый нейрон Гроссберга дает на выходе один из своих весовых коэффициентов, номер которого совпадает с номером активного нейрона Кохонена. Следовательно, слой Гроссберга преобразует выход слоя Кохонена с кодированием по номеру канала в произвольный линейный код на выходе, порождающая матрица кода совпадает с матрицей весовых коээфициентов слоя Гроссберга.

Работа сети в режиме аккредитации представлена на рис. . Показаны только ненулевые выходы. Активирован второй нейрон слоя Кохонена.

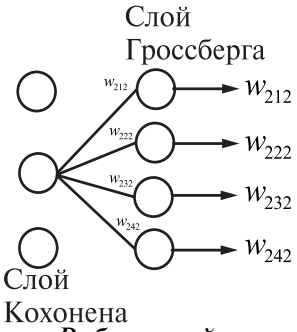


Рис. . Работа нейронов Гроссберга.

Обучение сети встречного распространения

Сеть обучается с учителем, хотя и включает в себя слой Кохонена. Для обучения необходимо обучающее множество, содержащее пары векторов $\left(\mathbf{x}^p, \mathbf{y}^p\right)$ (рис. ^СВР). Особенность СВР в том, что оба вектора $\mathbf{x}^p, \mathbf{y}^p$ подаются и на вход, и снимаются с выхода сети. Обучение происходит в следующей последовательности:

1. Подаем на вход вектор $\mathbf{D}^p = \begin{pmatrix} \mathbf{x}^p \\ \mathbf{y}^p \end{pmatrix}$. Определяем выигравший нейрон Кохонена и рассчитываем выход сети.

2. Обучаем слой Кохонена (проводим только одну итерацию обучения) по обычному алгоритму без учителя. При этом вектор $\mathbf{D}^p$ рассматривается как вектор, подлежащий классификации.

3. Зная требуемый выходной вектор, проводим коррекцию весов слоя Гроссберга так, чтобы выход приблизился к требуемому значению:

$$\delta w_{i_0 j 2} = \varepsilon\left(D_j^p - OUT_j^{p\,p}\right)$$

— где D_j^p — требуемое значение выхода *j*-го нейрона, когда на входе предъявлен вектор $\mathbf{D}^p$; OUT_j^p — текущее значение *j*-го выхода сети. При этом выход имеет значение:

$$\mathbf{OUT}^p = \begin{pmatrix} \mathbf{x}^{*p} \\ \mathbf{y}^{*p} \end{pmatrix}$$

Цель обучения — добиться точного выходного вектора с тем, который содержится в обучающем множестве:

$$\mathbf{x}^p \approx \mathbf{x}^{*p}$$
$$\mathbf{y}^p \approx \mathbf{y}^{*p} \qquad ()$$

Алгоритм завершается, когда будет достигнута хорошая точность в приближенных равенствах ().

После того, как обучение завершено, проявляются возможности СВР. Если подать оба вектора **x** и **y** на вход сети, на выходе будут получены вектора, приближенно равные им (). Такая операция, конечно, бесполезна. Но если подать на вход только один вектор, например, **x**, то на выходе будут получены *оба* вектора, $\mathbf{x}^*$ и $\mathbf{y}^*$. Если обучение прошло успешно, то () будут выполняться с хорошей точностью. Т.е. по одному из векторов **x** или **y** сеть восстанавливает второй вектор по тому закону, который изучен сетью по обучающему множеству.

Следовательно, сеть встречного распространения изучает одновременно два отображения: прямое $X \to Y$ и обратное $Y \to X$. Это важное свойство СВР.

Если на вход подавать только один вектор x, а на выходе снимать вектор y, то такая сеть способна выдавать вместо номера класса (как в случае слоя Кохонена) произвольный линейный код, соответствующий данному классу. Такая комбинация слоя Кохонена и слоя Гроссберга полезна при сжатии данных и в других приложениях.

Благодаря линейности слоя Гроссберга можно осуществлять интерполяцию кодов. Слой Кохонена работает в режиме интерполяции. В этом случае если вектор $\mathbf{x}^1$ на входе сети соответствует коду $\mathbf{c}^1$ на выходе сети, и вектор $\mathbf{x}^2$ на входе коду $\mathbf{c}^2$ на выходе, то при непрерывном изменении входного вектора $\mathbf{x}^1 \to \mathbf{x}^2$ выходной вектор (код) непрерывно меняется $\mathbf{c}^1 \to \mathbf{c}^2$ по тому же закону, что и вход.

Генетические алгоритмы для обучения НС

Генетические алгоритмы — группа алгоритмов многомерной оптимизации, основанных на моделировании развития биологической популяции.

Пусть есть целевая функция $E(\mathbf{p})$, зависящая от вектора $\mathbf{p}$ независимых переменных. В задаче оптимизации требуется найти минимум E.

Популяцией назовем набор векторов $\mathbf{P} = \{\mathbf{p}_i\} = \mathbf{p}_1..\mathbf{p}_N$. N — размер популяции. Элементы $\mathbf{p}_i$ - *особи*.

Элементы множества $\mathbf{P}$ способны эволюционировать по следующим правилам:

1. Если $E(\mathbf{p}_0)$ — мала, то особь $\mathbf{p}_0$ считается ***удачной*** и получает приоритет при размножении. Вероятность гибели этой особи снижается.

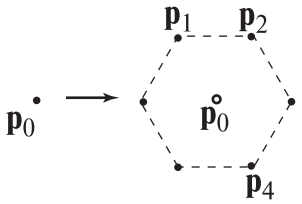
2. Если $E(\mathbf{p}_0)$ — велика, то особь $\mathbf{p}_0$ считается ***неудачной*** , вероятность размножения для этой особи снижается, и повышается вероятность гибели.

3. ***Мутации:*** любая точка имеет равную вероятность мутации, т.е. смещения на небольшую величину $\mathbf{p}_0 \to \mathbf{p}_0 + \Delta\mathbf{p}$, где $\Delta\mathbf{p}$ — небольшой по модулю вектор, характеризующий величину мутации. Закон, по которому определяется $\Delta\mathbf{p}$, зависит от реализации алгоритма. Типичный выбор — многомерное нормальное распределение с нулевым матожиданием.

4. ***Размножение:*** в соответствии с вероятностями, определенными на шагах 1, 2, каждая точка имеет вероятность размножения, тем большую, чем точка удачнее. Законы размножения могут быть различными, в зависимости от выбранной модели, например:

а) разделение на две близлежащих точки: $\mathbf{p}_0 + \mathbf{m}_1$ и $\mathbf{p}_0 + \mathbf{m}_2$, где векторы $\mathbf{m}_1$ и $\mathbf{m}_2$ определяются алгоритмом;

б) разделение на несколько близких точек построением правильного многоугольника (симплекса) вокруг точки $\mathbf{p}_0$ по известным формулам из теории оптимизации:



в) размножение со скрещиванием: две близких точки $\mathbf{p}_0$ и $\mathbf{p}_1$, таких, что $\|\mathbf{p}_0 - \mathbf{p}_1\|$ — мала, делятся со скрещиванием: $\mathbf{p}_i = c_i(\mathbf{p}_0 - \mathbf{p}_1) + \mathbf{p}_1 + \mathbf{m}_i$, где c_i — скаляр, $\mathbf{m}_i$ — вектор, зависящий только от i.

5. ***Гибель:*** в соответствии с вероятностью, определенной на шагах 1, 2, точка может погибнуть, т.е. быть бесследно удаленной из множества $\mathbf{P}$.

Текущий набор $\mathbf{p}_i$ составляет генофонд популяции. Налицо наследственность, изменчивость и естественный отбор — движущие силы биологической эволюции.

Точная количественная теория эволюции, сохранения и изменения генетической информации пока не построена (см., например: М.Эйген. Гиперцикл: принципы самоорганизации материи), поэтому остается большой произвол в выборе численных значений вероятностей гибели, размножения и мутаций и деталей алгоритма (способ размножения, мутаций). Оптимальность выбранных алгоритмов пока может быть оценена только экспериментально.

Применение генетических алгоритмов для обучения НС

Для обучения нейросетей могут применяться генетические алгоритмы. Обычно в качестве независимой переменной выбирается $\mathbf{p} = \begin{pmatrix} \mathbf{W} \\ \Theta \end{pmatrix}$ — набор весовых коэффициентов и пороговых уровней сети. Целевой функцией служит функция ошибки, или любая другая характеристика качества сети при решении поставленной задачи. Целевая функция вообще может не иметь числового выражения, а задаваться алгоритмически.

Обучение НС с помощью генетических алгоритмов состоит в следующем:

1. В памяти создается некоторое количество нейросетей (популяция) с различными значениями **p**.
2. Пошагово моделируется развитие популяции в соответствии с генетическим алгоритмом (мутации, гибель, размножение).
3. Каждые несколько итераций в популяции выбирается точка с лучшим значением $E(\mathbf{p})$. Если значение целевой функции в этой точке достаточно мало, алгоритм завершается.

В качестве целевой функции может быть выбрана любая характеристика качества при решении тестовой задачи, поэтому генетические алгоритмы применимы к обучению НС как с учителем, так и без учителя.

Положительные качества генетических алгоритмов

1. Нахождение ***глобального минимума***: неподверженность "застреванию" в локальных минимумах целевой функции.
2. ***Массовый параллелизм*** при обработке: особи в популяции функционируют независимо: расчет значений целевой функции, гибель, мутации осуществляются независимо для каждой особи. При наличии нескольких процессорных элементов быстродействие может быть очень высоким.
3. ***Биоподобность:*** генетические алгоритмы построены на тех же принципах, которые привели к возникновению человека и всего многообразия видов, и, следовательно, могут быть очень продуктивны и полезны.

Недостатки при обучении НС

1. Для каждой особи, в случае НС, требуется много памяти, т.к. количество весов и пороговых уровней обычно велико. Количество требуемой памяти пропорционально также размеру популяции. Поэтому для НС размер популяции весьма ограничен, что снижает эффективность алгоритма.
2. Склонность к параличу при обучении. Изменения параметров сети случайны, и веса могут возрастать неограниченно. Это вводит нейроны в насыщение и снижает скорость обучения. Необходимы дополнительные меры, чтобы избежать чрезмерного роста параметров.
3. Низкая эффективность на фон-неймановских ЭВМ. Мутации параметров случайны, и много ресурсов расходуется на "ненужные" вычисления. При отсутствии параллельной обработки быстродействие алгоритмов невелико.

Сети с обратными связями

Для сетей прямого распространения были приняты ограничения: все сигналы в сети распространяются только от входа к выходу, но не наоборот. Сеть также предполагалась послойно-полносвязной. Оба эти ограничения несправедливы для биологических НС и сужают возможности модели. Сеть прямого распространения не имеет внутреннего состояния: значения выходов нейронов зависят *только* от входного вектора и не меняются во времени, если вход неизменен. Моделирование динамических процессов на таких сетях возможно только искусственными приемами, например, когда сетью на каждом шаге прогнозируется малое изменение состояния для исследуемого динамического объекта.

Чтобы расширить диапазон решаемых задач, были предложены сети с обратными связями. Полное математическое описание пока создано только для простейших случаев сетей с обратными связями. Дж. Хопфилд внес вклад в разработку и теории, и моделей таких сетей.

Послойность сети и матричное умножение

Каждый слой НС из формальных нейронов выполняет взвешенное суммирование входов: $NET_{jl} = \sum_i w_{ijl} x_{ijl} - \theta_{jl}$. Это эквивалентно умножению матрицы весовых коэффициентов на вектор входных сигналов данного слоя: $NET_l = W_l^T X_l - \Theta_l$. Здесь W_l^T — матрица весовых коэффициентов слоя l, X_l — вектор входных сигналов слоя l, Θ_l — вектор пороговых уровней слоя l. Операция транспонирования появляется из-за одинакового порядка индексов в матрице W и векторе входных сигналов X: индекс i обозначает вход j-го нейрона.

Эквивалентность взвешенного суммирования умножению матрицы весов на вектор входных сигналов говорит о том, что любое устройство, умеющее перемножать матрицы, может работать в качестве слоя НС, и наоборот, операция, сводящаяся к умножению матрицы на вектор, может быть реализована в виде НС.

Расчет градиента квадратичной формы

Пусть задана квадратичная форма:

$$H = \frac{1}{2}(x, Qx) = \frac{1}{2}\left(x, \begin{pmatrix} \sum_k q_{1k} x_k \\ \dots \\ \sum_k q_{Nk} x_k \end{pmatrix} \right) = \frac{1}{2} \sum_l \sum_k q_{lk} x_l x$$

Ее градиент легко записать:

$$(\nabla H)_i = \frac{1}{2}\left(\sum_{k \neq i} q_{ik} x_k + \sum_{l \neq i} q_{il} x_l + 2 q_{ii} x_i \right) = \frac{1}{2} \sum_k (q_{ik} +$$

И если считать, что матрица Q - симметрична, $q_{ik} = q_{ki}$, то $(\nabla H)_i = \sum_{k=1}^{i} q_{ik} x_k = Qx$. Если вместо исходной формы H взят многочлен $P(x) = \frac{1}{2}(x, Qx) + (b, x)$, то его градиент $\nabla P = Qx + b$. Таким образом, расчет градиента многочлена P сводится к умножению матрицы Q на вектор и суммированию векторов. Следовательно, этот расчет может быть выполнен однослойной нейронной сетью без обратных связей и даже без нелинейных элементов в нейронах. Сеть для расчета градиента имеет количество нейронов, равное числу компонент вектора X, матрицу весовых коэффициентов $(W)_{ij} = Q_{ij}$, матрицу пороговых уровней $\Theta_j = -b_j$. Если на вход такой сети подать произвольный вектор X, то на выходе мы получим численное значение градиента P в точке, заданной численным значением вектора X. Для расчета градиента P обратные связи не требуются.

Теперь рассмотрим однослойную сеть с обратными связями, где значение выхода каждого нейрона подается обратно на входы всех нейронов того же слоя:

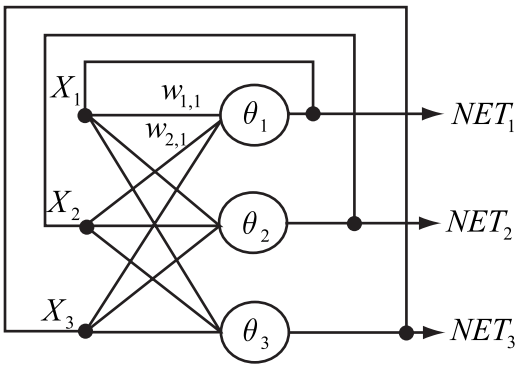


Рис. 1. Сеть с обратными связями.

Входной вектор для сети в данном случае совпадает с выходным NET, взятым на предыдущей итерации. Предполагаем, что расчет сигналов NET происходит мгновенно и одновременно во всех нейронах, а распространение по обратным связям дает задержку в одну итерацию и происходит одновременно во всех нейронах: сеть работает синхронно. В такой модели работа сети будет определяться формулами:

Пусть весовые коэффициенты и пороговые уровни заданы так, чтобы выходной сигнал NET определялся формулой:

$$NET = X - h(QX + b) = IX - hQX - hb = (I - hQ$$,

т.е. весовые коэффициенты $W = I - hQ$, пороговые уровни $\Theta = hb$. Такой выбор параметров сети позволит на каждой итерации по вектору X получать новый вектор NET, который отличается от X небольшим шагом в направлении антиградиента $-\nabla P$ в точке X. Длина шага будет пропорциональна h - небольшому положительному числу, выбранного нами при создании сети.

Сформированный сигнал NET подается обратно на вход сети, и на следующей итерации становится новым значением X. Так, на каждой итерации происходит небольшое изменение вектора X в направлении антиградиента P, т.е. в сторону минимума многочлена P(x).

Выбор начальной точки и длины шага

Чтобы выбрать начальную точку, из которой сеть начнет движение в сторону минимума P(x), мы должны подать на вход сети соответствующий вектор X_0, вместо сигнала NET, который до этого момента будет не определен. Следовательно, требуется кратковременный разрыв цепочки обратной связи, что нарушает стройность модели. В случае программной реализации сети это не представляет проблемы, а в случае аппаратной — приводит к неудобствам.

Вообще говоря, коэффициент h должен определяться точной одномерной оптимизацией, т.е. поиском минимума P(x) в направлении антиградиента на каждой итерации. НС не дают средств для решения этой задачи. Слишком большой h приведет к грубому результату (сеть будет хаотически менять свое состояние около точки минимума), а слишком малый — к медленной сходимости. Хорошие результаты дает постепенное уменьшение h на каждой итерации, но изменение h требует перестроения сети (изменения матрицы весов W), а это длительная операция.

Таким образом, простейшая сеть с обратными связями способна находить минимум квадратичной формы. Параметры сети, необходимой для поиска минимума, определяются коэффициентами многочлена P. Такая задача относится к задачам безусловной оптимизации.

Аналогичным образом НС с обратными связями могут решать системы линейных уравнений Ax = b. Задача решения системы сводится к минимизации многочлена:

$$P = \frac{1}{2}\left((Ax - b),(Ax - b)\right) = \frac{1}{2}\left(x, A^T Ax\right) - \left(A^T b, x\right)$$

Градиент $\nabla P = \frac{1}{2} A^T Ax - A^T b$. Чтобы реализовать его вычисление в виде НС, можно использовать двухслойную сеть без обратных связей с весами $W_1 = \frac{1}{2} A$, $W_2 = A^T$, $\Theta_1 = b$, $\Theta_2 = 0$. Чтобы минимизировать многочлен P, подаем выходные сигналы сети снова на вход с задержкой в одну итерацию.

Существуют также более эффективные алгоритмы решения систем линейных уравнений с помощью НС.

Сеть Хопфилда

Сети с обратными связями могут работать в качестве *ассоциативной памяти*. Это означает, что по вектору, поданному на вход, сетью будет создан на выходе один из запомненных ранее векторов, наиболее "похожий" (в некотором выбранном смысле) на данный входной вектор. Такой способ выборки данных называется *адресацией по содержимому*, в отличие от адресации по номеру ячейки памяти, принятому в ЭВМ фон-неймановского типа. Этот способ адресации широко используется в биологических НС. Например, один лишь запах жасмина может вызвать в памяти целый набор ассоциаций, причудливо связанных друг с другом и включающих в себя визуальные, звуковые и кинестетические образы. Память с такой адресацией является весьма перспективной для создания систем искусственного интеллекта, систем распознавания речевых сигналов и изображений.

Пусть задано множество векторов $\{x^k\} = x^1 .. x^K$, подлежащих запоминанию в нейросети. Критерий "похожести" векторов зависит от задачи и, вообще говоря, может быть весьма сложным. Если, к примеру, входной вектор состоит из нескольких отсчетов сигнала во времени, взятых подряд, то критерий близости векторов должен обладать инвариантностью к масштабированию отсчетов, к переносу вдоль оси времени (фазе сигнала), и к зашумленности входных данных. Так что поиск критерия близости специфичен для каждой задачи и представляет собой серьезную проблему.

Рассмотрим простейший случай, когда в качестве меры близости двух векторов используется их скалярное произведение: $\mathrm{d}(x^1, x^2) = (x^1, x^2)$. Чем более "похожи" вектора, тем больше мера близости. Тогда можно ожидать, что изменение вектора x с течением времени по закону $dx = \sum_k x^k (x^k, x)\, dt$ в конце концов приведет к тому, что x совпадет с наиболее похожим эталоном, т.е. требуемая ассоциация будет найдена. Пусть также компоненты эталонных векторов могут принимать только значения +1 и -1.

Если найти такую функцию H(x), что $\frac{dx}{dt} = -\nabla H$, то задача поиска ассоциации для вектора x будет совпадать с задачей поиска минимума функции H(x). Если выбрать функцию H, которая называется *энергией сети,* в виде $H(x) = -\frac{1}{2}\sum_k (x^k, x)^2 + \frac{1}{2}\lambda \sum_i (x_i^2 - 1)^2$, то ее градиент $\nabla H = -\sum_k x^k (x^k, x) + \lambda \sum_i e^i (x_i^2 - 1) x_i$, здесь e^i — i-й вектор базиса. Нижними индексами обозначены компоненты вектора, верхними - номер вектора в множестве. Тогда $\frac{dx}{dt} = \sum_k x^k (x^k, x) - \lambda \sum_i e^i (x_i^2 - 1) x_i$. Первое слагаемое обеспечивает стремление x к ближайшим эталонам, а второе — приближение компонент вектора x к допустимым значениям +1 или -1. Коэффициент λ соотносит интенсивности этих двух процессов. Обычно λ растет с течением времени от $\lambda < 1$ до $\lambda > 1$ с ростом времени обучения. Чтобы получить весовые коэффициенты и пороговые уровни, запишем последнее выражение покомпонентно:

$$\left(\frac{dx}{dt}\right)_i = \sum_j \sum_k x_i^k x_j^k x_j - \lambda\left(x_i^2 - 1\right)x_i = \sum_{j \neq i} \sum_k x_i^k x_j^k x_j$$

Отсюда видно, что весовые коэффициенты должны определяться:

$$W_{ij} = \begin{cases} \sum_k x_i^k x_j^k, & i \neq j \\ 0, & i = j \end{cases}$$

Веса не зависят от направления от i к j или от j к i. Данная формула аналогична формуле Хебба для обучения перцептрона, когда связь между нейронами i и j положительна, если состояния нейронов одинаковы и отрицательна, если состояния противоположны.

Каждый i-й нейрон содержит управляемый нелинейный порог со значением $\theta_i = \left(\lambda\left(x_i^2 - 1\right) - 1\right)x_i$, рассчитываемый на каждой итерации. Сеть показана на рис. ^^^. Каждый выходной сигнал подается обратно на вход сети с весом Wij и на вход того же нейрона для расчета порога. Данная модель называется *сетью Хопфилда*. Приведенный способ описания отличается от работ Хопфилда, но эквивалентность модели сохранена.

Решение задач с помощью сетей Хопфилда

Решение некоторой задачи с помощью сетей Хопфилда распадается на этапы:

1. Построить функцию энергии таким образом, чтобы точка глобального минимума этой функции совпадала с решением задачи. При этом градиент функции энергии должен допускать вычисление с помощью НС.
2. Записать формулы для расчета параметров сети (весовых коэффициентов и пороговых уровней) для расчета градиента функции энергии.
3. Разорвать цепочку обратной связи и предъявить сети входной вектор. Рассчитать значения выходов.
4. Замкнуть обратную связь и предоставить сети возможность самостоятельно менять свое состояние (релаксация). Остановить процесс релаксации после того, как выходной вектор перестанет меняться, т.е. по достижении минимума функции энергии. Полученные выходы сети дают решение задачи.

Устойчивость сети

Точка X, в которой остановится процесс релаксации, называется устойчивой, если после малого изменения вектора X из состояния равновесия сеть вернется к тому же состоянию через некоторое количество итераций.

В отличие от сетей прямого распространения, которые всегда устойчивы, сеть с обратными связями может либо вообще не сойтись к одной точке, либо дать результат, неустойчивый к малым изменениям входного вектора из полученной точки. При каких условиях процесс релаксации сети приведет к точке устойчивого равновесия? Для сети общего вида необходимое и достаточное условие устойчивости не сформулировано. Однако Кохеном и Гроссбергом было доказано достаточное условие устойчивости. Если матрица весовых коэффициентов симметрична, $W = W^T$, и на главной диагонали находятся нули, $w_{ii} = 0$, то данная сеть всегда сходится к устойчивой точке. С другой стороны, бывают устойчивые сети с несимметричной матрицей весов и с ненулевыми диагональными элементами, и сети, в которых малые отклонения от достаточного условия приводят к потере устойчивости.

Сходимость к эталонам

Если веса сети определяются по формуле $w_{ij} = \sum_k x_i^k x_j^k$, то каждый эталон будет представлять собой локальный минимум функции энергии, а градиент функции энергии в этой точке будет равен

нулю. Но сеть не всегда сходится к одному из эталонов. Сформулируем условия, повышающие вероятность правильной сходимости:

1) Количество образов M, запомненных в сети, не должно превышать емкости сети.

2) Векторы x^k, запомненные сетью, должны быть слабо коррелированы, т.е. мера близости $\mathrm{d}\left(x^k, x^l\right),\ \forall k,l < M$ должна быть мала.

Численные значения емкости сети и предельно допустимой близости эталонов строго не определены. Если учесть, что общее число состояний сети из n нейронов с двумя допустимыми значениями выходов +1 или -1 составляет 2^n, то общепринятая оценка емкости 0,15 n кажется совсем небольшой.

Если нарушены условия 1) или 2), то решение, полученное сетью, чаще всего представляет собой некий усредненный эталон, сочетающий в себе черты многих запомненных образов.

Адаптивная резонансная теория (АРТ)

Серьезная проблема для ИНС — правильное соотношение стабильности и пластичности при запоминании образов.

Существуют наборы эталонов (даже состоящие всего из 4-х векторов), которые при циклическом предъявлении в обучении дают никогда не сходящиеся наборы параметров сети. Предъявление всего одного нового образа в обучающем множестве часто приводит к долгому переобучению. Если сеть работает в реальном времени, например, обрабатывает сенсорную информацию, то обучающее множество может все время меняться. Для большинства моделей ИНС это приводит к отсутствию обучения вообще.

Человеческая память, напротив, эффективно хранит и корректирует запоминаемые образы. Ни предъявление нового образа, ни изменение старых не приводит к уничтожению памяти или невозможности запоминания. Даже удаление части нервной ткани чаще всего не прерывает работу сети и не стирает запомненные образы, а лишь делает их менее четкими.

Сеть АРТ — попытка приблизить механизм запоминания образов в ИНС к биологическому. Результатом работы АРТ является устойчивый набор запомненных образов и возможность выборки "похожего" вектора по произвольному предъявленному на входе вектору. Важное качество АРТ — динамическое запоминание новых образов без полного переобучения и отсутствие потерь уже запомненных образов при предъявлении новых.

АРТ-1

АРТ-1 предложена Карпентером и Гроссбергом в 1986 г. Эта сеть представляет собой векторный классификатор и обучается без учителя, лишь на основании предъявляемых входных векторов. АРТ-1 работает только с двоичными векторами, состоящими из нулей и единиц. Позже было предложено много разновидностей этой модели. АРТ-2 запоминает и классифицирует непрерывные входные векторы, FART использует нечеткую логику. Группа моделей с суффиксом "MAP" (ARTMAP и др.) классифицирует и входные, и выходные вектора, а также строит связи между ними, позволяя формировать отображения аналогично сети встречного распространения.

Архитектура и работа

Структура АРТ-1 (далее АРТ) представлена на рис .

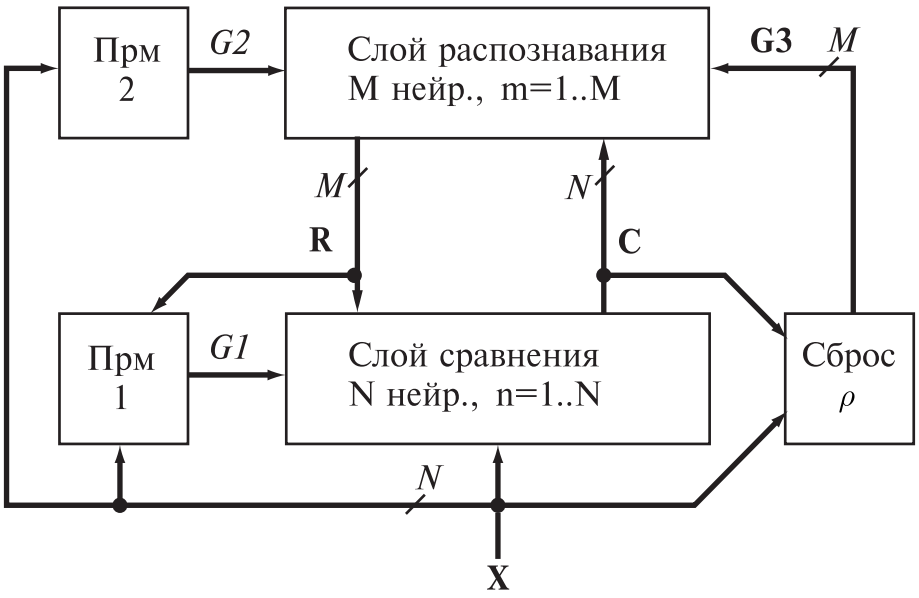


Рис. . Структурная схема АРТ.

Входной вектор сети $\mathbf{X} = X_1,..X_n,..X_N$ имеет N компонент. В слое распознавания запоминается M классов образов, по одному классу на каждый нейрон m=1..M.

Основную работу по классификации производят слой сравнения и слой распознавания. Схемы приемников (Прм1, Прм2) и схема сброса управляют режимом работы сети и могут быть реализованы в виде обычных логических схем или в виде нейронов.

Работа блоков АРТ определяется следующими формулами:

Прм 1: $G1 = \underset{n}{\text{ИЛИ}}(X_n)\ \text{И НЕ}\left(\underset{n}{\text{ИЛИ}}(R_m)\right)$. Выход прм1 обеспечивает единичный сигнал для слоя сравнения, если на вход сети подан вектор **X** (нулевой вектор на входе недопустим) и если выход слоя распознавания равен нулю.

Прм 2: $G2 = \underset{n}{\text{ИЛИ}}(X_n)$. Если на вход подан вектор **X**, то блок прм2 формирует на выходе единичный сигнал и тем самым разрешает работу слоя распознавания.

Схема сброса: $G3_m = \left(\left. \frac{\sum\limits_n C_n}{\sum\limits_n X_n} \right|_{\mathbf{X},\mathbf{C}} < \rho \right)$. Проверяет критерий сходства для векторов **X** и **C**. Критерий состоит в сравнении количества единиц в векторах **X, C.** Количества единиц сравниваются в виде отношения с некоторым пороговым уровнем сходства ρ. Если порог не превышен, то сходство считается плохим, и схема сброса вырабатывает сигнал торможения для нейрона в слое распознавания. Выход схемы сброса — двоичный вектор с M компонентами. Схема сброса является динамической и "помнит" свое состояние в течение одной классификации. Порог ρ является внешним параметром по отношению к сети и задается пользователем в интервале от 0 до 1. Чем меньше ρ, тем менее похожие вектора будут отнесены сетью к одному классу.

Слой сравнения

Структура слоя показана на рис. .

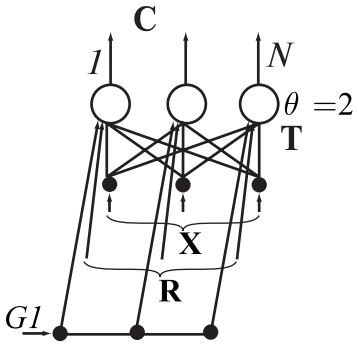


Рис. . Слой сравнения.

Каждый нейрон имеет порог, равный двум. На вход одного нейрона в слое сравнения подаются: сигнал G1 с единичным весом, одна компонента $\mathbf{X}^n$ с единичным весом и все выходы слоя распознавания, M компонент с вектором весов $\mathbf{T}^n$, где n - номер нейрона в слое сравнения. Весовые коэффициенты T — двоичные. В нейроне используется нелинейность в виде жесткой ступеньки: если сигнал NET нейрона превышает порог $\theta = 2$, то на выходе нейрона будет единица, иначе — ноль. Это "правило 2/3": для активации нейрона достаточно два сигнала из трех.

Работа слоя определяется формулами:

$$P_n = \mathbf{T}^n \mathbf{R} = \sum_m T_m^n R_m$$

$$T_n = P_n + X_n + G1$$

$$C_n = \begin{cases} 0, & NET_n < 2 \\ 1, & NET_n \geq 2 \end{cases}$$

Работой слоя управляет сигнал G1. Если G1=0, то **X** проходит без изменений на выход слоя сравнения, благодаря лишнему единичному сигналу G1 на входе нейрона. Если G1=0, то на выходе имеем $\mathbf{C} = \mathbf{X} \wedge \mathbf{P}$, т.е. вектор **C** будет логическим произведением двоичных векторов **X** и **P**.

Слой распознавания

Структура слоя распознавания показана на рис. .

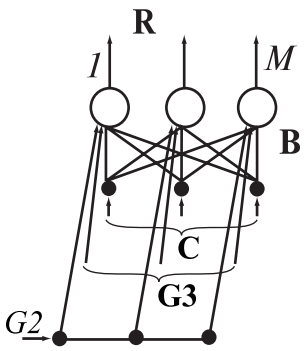


Рис. . Слой распознавания.

Каждый нейрон в слое имеет следующие входы: один сигнал G2 с единичным весом, одна компонента $G3_m$ с большим отрицательным весом (m — номер нейрона) и N сигналов со слоя сравнения с вектором весов $\mathbf{B}^m$ (у вектора $\mathbf{B}^m$ всего N компонент, $B_1^m..B_N^m$).

Нейроны слоя распознавания не содержат нелинейных элементов, но обладают следующей особенностью. Каждый нейрон в слое связан со всеми остальными нейронами этого же слоя обратными тормозящими связями и положительной обратной связью — с самим собой. Такой способ связности называется *латеральным торможением*. Это приводит к тому, что только один нейрон в слое

распознавания может быть активирован. Между нейронами существует конкуренция, и нейрон с максимальным выходом "подавляет" все остальные нейроны в слое, выигрывая "состязание". Его выход становится равным единице, остальных нейронов — нулю.

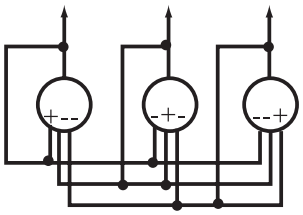


Рис. . Латеральные тормозящие связи в слое распознавания.

Веса B^m — непрерывные, в отличие от слоя сравнения. Работа слоя определяется формулой:

$$R_m = \left(\mathbf{B}^m, \mathbf{C}\right) \wedge G2 \ \wedge \ \neg\left(G3_m\right)$$

Отсюда видно, что сигнал G2 "разрешает" работу слоя распознавания, а сигнал G3 позволяет выборочно затормозить любые нейроны в слое.

Работа сети АРТ

Решение задачи классификации с помощью АРТ содержит следующие этапы: инициализация, распознавание, сравнение, поиск, обучение.

1. Инициализация.

а) выбираем параметр ρ, исходя из требуемой детальности классификации;

б) создаем сеть в памяти. Количество нейронов должно быть достаточным, чтобы запомнить все ядра классов (до M). Изначально все нейроны слоя распознавания считаются "невыделенными", их веса приравниваются одинаковым небольшим значениям: $B_n^m = B_{нач} < \dfrac{L}{L+N-1}$, где L>1 - некоторая константа (обычно L=2). Веса в слое сравнения также выбираются одинаковыми, но равными единице: $T_m^n = 1$. Такой выбор весов обеспечивает остановку поиска на невыделенном нейроне, если нет подходящих выделенных нейронов и правильное обучение.

2. Распознавание.

а) Предъявляем вектор **X** на входе. До этого момента G2=0 и выход слоя распознавания равен нулю: **R**=0.

б) У вектора **X** есть ненулевые компоненты, поэтому G1 становится равным единице, т.к. **R**=0. Сигнал G1 "подпитывает" нейроны слоя сравнения и **X** без изменений проходит через слой сравнения: **C=X.**

в) Весовые коэффициенты $\mathbf{B}^m$ имеют смысл нормированных ядер классов. В слое распознавания активируется несколько нейронов, но благодаря латеральному торможению остается один нейрон с выходом R_{m0}=1, а остальные тормозятся. m_0 — номер выигравшего нейрона.

3. Сравнение.

а) Выход $\mathbf{R} \neq \mathbf{0}$ приводит к G1=0, что снимает "подкачку" нейронов в слое сравнения. Весовые коэффициенты $\mathbf{T}^n$ имеют смысл ненормированных двоичных ядер классов. На вход слоя сравнения передается один ненулевой выход слоя распознавания, R_{m0}=1. Эта единица умножается на весовые коэффициенты, давая в сумме сигнал $NET_n = X_n + T_{m_0}^n$. Порог всех нейронов равен 2, поэтому выход слоя сравнения будет $C_n = X_n \wedge T_{m_0}^n$. Следовательно, выход слоя сравнения на этом этапе — логическое произведение входного сигнала и двоичного ядра класса из слоя сравнения.

б) Модуль сброса вычисляет второй критерий сходства (первый — максимум произведения ($\mathbf{B}^m$,**X**) в слое распознавания). Если количества единиц в векторе **C** и векторе **X** близки, то сходство считается хорошим и выносится решение о принадлежности вектора **X** к классу m_0.

4. Поиск.

а) Если критерий сходства не выполняется, схема сброса вырабатывает сигнал $G3_{m0}=1$, который тормозит нейрон m_0 в слое распознавания. Сигнал $G3_{m0}$ остается равным 1 до окончания данной классификации. Выход нейрона m_0 становится равным 0, а, следовательно, и весь вектор **R=0**. Сигнал G1 становится равным нулю и вектор **X** снова проходит через слой сравнения без изменений, вызывая новый цикл поиска (шаги 2в — 3б), пока критерий сходства не будет удовлетворен.

При соответствующем выборе начальных значений весов B поиск всегда закончится на нераспределенном нейроне слоя распознавания. Для него будет выполнен критерий сходства, т.к. все веса T равны 1. Если все нейроны выделены и критерий сходства не выполняется, следует аварийный останов, либо расширение сети введением нового нейрона в слое распознавания и новых входов в слое сравнения.

5. Обучение.

Независимо от того, найден ли на этапе поиска распределенный нейрон или нераспределенный, обучение протекает одинаково. Корректируются лишь веса выигравшего нейрона m_0 в слое распознавания, и веса $T^n_{m_0}$ для всех n в слое сравнения.

Различают быстрое и медленное обучение. При быстром обучении коррекции весов имеют вид:

$B_n^{m_0} = \dfrac{LC_n}{L+\sum_n C_n - 1} \equiv \Delta B_n^{m_0}$, где L — константа, большая 1.

Веса в слое сравнения — двоичные: $T^n_{m_0} = C_n$. В результате такого алгоритма обучения ядра T изменяются от всех компонент, равных 1, обнуляя несущественные компоненты в процессе обучения. Если какая-то компонента вектора $\mathbf{T}^n$ стала нулевой на какой-то итерации обучения, она никогда не вернется к единице. В этом проявляется асимметрия АРТ по отношению к значениям 0 и 1. Эта асимметрия имеет серьезные отрицательные последствия для модели, приводя к деградации ядер классов в случае зашумленных входных векторов.

Медленное обучение меняет ядра малыми коррекциями:

$B_n^{m_0} \to \beta\,\Delta B_n^{m_0} + (1-\beta) B_n^{m_0}, \quad T^n_{m_0} \to \beta C_n + (1-\beta) T^n_{m_0}$,

где β — мало и характеризует скорость обучения.

В результате каждой итерации обучения ядра меняются незначительно.

Видно, что веса B в любой момент времени могут быть однозначно рассчитаны через веса T, таким образом, кодирование информации о ядрах в АРТ в рассмотренной модели является избыточным в смысле расхода памяти.

Необходимость поиска

В сети АРТ используются два критерия "похожести" векторов. Первый — максимум скалярного произведения $\max_m(\mathbf{B}^m, \mathbf{X})$ при выборе "победителя" в слое распознавания. Второй — критерий сходства в блоке сброса: $\left.\dfrac{\sum_n C_n}{\sum_n X_n}\right|_{\mathbf{X,C}} \geq \rho$. Таким образом задача классификации в сети АРТ состоит в том, чтобы найти ядро с максимальным скалярным произведением $(\mathbf{B}^m, \mathbf{X})$, чтобы при этом выполнялся критерий сходства. Эти два критерия не являются эквивалентными, поэтому и фаза поиска, и фаза распознавания являются необходимыми и не могут быть опущены.

Положительные качества и недостатки АРТ

Сеть АРТ решает дилемму стабильности-пластичности и позволяет быстро запоминать новые образы без утраты старых. Как и в случае других моделей НС, на обычных машинах фон-неймановского типа сети работают медленно и неэффективно. Для решения задачи требуется найти максимум

скалярного произведения, что требует около 3NM операций с плавающей запятой, и вычислить в худшем случае M критериев сходства. Это требует существенных вычислительных затрат.

Если моделировать сеть на аналоговой параллельной машине, то результат будет получен практически мгновенно. Но такие машины — редкость. На цифровом параллельном компьютере операции расчета скалярных произведений могут быть распараллелены, но расчет критериев сходства все равно выполняется последовательно. Таким образом, даже на параллельной машине сеть АРТ является требовательной к ресурсам.

Тем не менее, одна итерация для запоминания каждого входного вектора — редкая экономичность для нейронных сетей. Вспомним, что многослойный перцептрон для запоминания нового вектора требует полного переобучения.

У сети АРТ есть несколько существенных недостатков.

1. **Чувствительность к порядку предъявления векторов.** Большинство разновидностей АРТ весьма чувствительны к порядку предъявления входных векторов **X**. Картины ядер классов, сформированные сетью, принципиально меняются при различных видах упорядочения.

2. **Невозможность классификации зашумленных векторов.** Пусть входные вектора содержат шум. Если компонента незашумленного входного вектора равна $\mathbf{x}_n$, то предъявленные сети значения будут определяться вероятностным законом:

$$\begin{cases} p(X_n = x_n) = 1 - \varepsilon \\ p(X_n = \neg x_n) = \varepsilon \end{cases}$$

где ε — малое положительное число, характеризующее уровень шума.

Если такие данные будут предъявлены АРТ, то будет наблюдаться *деградация и размножение классов*. Если сетью сформировано правильное ядро для класса, к которому относится вектор **X**, то как только компонента X_n примет нулевое значение за счет шума (если вектора предъявляются не однократно), соответствующая компонента ядра также будет обнулена. Т.к. случайное нулевое значение может принять любая компонента **X**, то с течением времени все компоненты ядра будут обнулены, запомненная информация об этом классе — утрачена. Если после этого предъявить незашумленный вариант вектора **X**, то для него будет выделен новый нейрон, т.е. сформирован новый класс. Это явление называется *размножением классов*. Через некоторое время в сети будет множество нейронов с нулевыми весами, и все нейроны будут распределены. Работа сети прекратится.

Это явление определяется исходной асимметрией алгоритмов АРТ относительно значений 0 и 1. Существуют методы для устранения асимметрии и предотвращения размножения классов, например, комплементарное кодирование.

Метод имитации отжига

При обучении НС, как и в и в других задачах многомерной оптимизации, одна из проблем — останов алгоритма в точке локального, а не глобального минимума целевой функции. При использовании градиентного алгоритма с точным выбором длины шага останов в локальном минимуме неизбежен. Обратное распространение ошибки страдает меньше, т.к. длина шага выбирается "некорректно" (без одномерной оптимизации вдоль вектора градиента), и на любой итерации возможно увеличение функции ошибки.

Метод имитации отжига позволяет преодолеть локальные минимумы и искать глобальный минимум целевой функции. "Плата" за это преимущество — медленная работа алгоритма в случае большой размерности целевой функции. Метод применим и к нейронным сетям, и к любым другим задачам многомерной оптимизации.

В 50-х годах был исследован процесс отжига металла и построена его математическая модель. Если раскалить кусок металла, то его внутренняя энергия достигнет высокого значения. Кристалличеcкая решетка при этом будет наименее упорядочена, т.к. тепловые флуктуации атомов решетки будут велики. Это соответствует начальному состоянию "необученной" нейронной сети. Если затем быстро охладить металл, то атомы будут "пойманы" в энергетически невыгодных состояниях. Энергия

системы снизится, но не достигнет глобального минимума. Кристаллическая решетка будет иметь множество дефектов, т.е. отклонений в расположении атомов от оптимального значения, а металл будет иметь высокую твердость (закаливание). Если охлаждение проводить медленно (отжиг), то с плавным уменьшением температуры тепловые колебания узлов решетки около состояния минимума энергии будут плавно уменьшаться, и в результате охлаждения решетка будет иметь высокую упорядоченность, а энергия системы достигнет глобального минимума.

В применении к задаче оптимизации модель выглядит так.

1. Выбирается смысл параметра "температуры" системы. Размерность его может быть различной и связана с размерностью целевой функции данной задачи. Целевая функция в данной модели означает энергию системы.

2. Выбирается большое начальное значение "температуры".

3. Независимые переменные, от которых зависит функция энергии, испытывают "тепловые флуктуации", т.е. случайные изменения. Обычно вероятность флуктуации независимой переменной x имеет гауссовское распределение:

$$p(x) = \exp\left(-\frac{x^2}{\theta^2}\right)$$

4. Рассчитывается изменение энергии (т.е. целевой функции), полученное за счет флуктуаций независимых переменных. Если энергия уменьшилась, то изменения шага 3 принимаются. Если энергия увеличилась, то изменения переменных сохраняются с вероятностью, зависящей от того, насколько увеличилась энергия, и каково текущее значение температуры:

$$p(\Delta E) = F(\Delta E, \theta) = \exp\left(\frac{-\Delta E}{\theta}\right)$$

где θ — значение температуры. Вероятность p убывает с ростом ΔE и с уменьшением θ.

5. Шаги 3,4 повторяются до тех пор, пока не будет достигнуто "тепловое равновесие". На практике это означает, что температура должна меняться очень медленно.

6. Температура уменьшается на малое значение, и шаги 3,4 повторяются для нового значения температуры.

7. Шаги 3-6 повторяются до тех пор, пока не будет достигнута малая температура, принятая за ноль. В этом состоянии энергия системы примет глобальное минимальное значение, если процесс охлаждения проводился бесконечно медленно. На практике скорость охлаждения конечна, и значение глобального минимума бывает неточным.

В результате такого алгоритма устанавливается тепловое равновесие, при котором вероятность обнаружить систему в состоянии с энергией E определяется распределением Больцмана:

$$p(E) = Z^{-1} \exp\left(\frac{-E}{\theta}\right)$$

Такая модель оптимизации была предложена С.Киркпатриком, С.Гелатта и М.Веччи в 1983 г. и получила название "имитации отжига". Он дает очень хорошие результаты для обучения нейронных сетей с количеством синапсов несколько сотен. Для большего размера сетей алгоритм работает слишком медленно. Имитация отжига применяется для NP-полных задач, например, задачи коммивояжера, не поддающихся точному алгоритмическому решению. Данный метод используется при автоматическом размещении компонент на печатных платах, при их автотрассировке и во многих других задачах.

Преимущество алгоритма — поиск глобального минимума и отсутствие ограничений на вид минимизируемой функции E. Недостаток — требование бесконечно медленного охлаждения, на практике означающее медленную работу алгоритма. Для нейронных сетей больших размерностей метод

трудноприменим из-за низкого быстродействия. Множество шагов по параметрам сети осуществляется в случайных, ненужных направлениях.

Список литературы

1. Суровцев И.С., Клюкин В.И., Пивоварова Р.П. Нейронные сети. — Воронеж: ВГУ, 1994. — 224 с.
2. Уоссермен Ф. Нейрокомпьютерная техника: теория и практика. — М.: Мир, 1992.
3. Горбань А.Н. и др. Нейроинформатика. — Электронная публикация.
3. Интернет: Sarle, W.S., ed. (1997), Neural Network FAQ, part 1-7: Introduction, periodic posting to the Usenet newsgroup comp.ai.neural-nets, URL ftp://ftp.sas.com/pub/neural/FAQ.html.
5. Мкртчян С.О. Нейроны и нейронные сети. (Введение в теорию формальных нейронов) — М.: Энергия, 1971. — 232 с..
6. Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. - М.: Мир, 1985. - 509 с.
7. Лоскутов А.Ю., Михайлов А.С. Введение в синергетику. — М.: Наука. Гл. ред. физ.-мат. лит., 1990. — 272 с.
8. Muller B., Reinhardt J. Neural Networks. An introduction. — Berlin: Springer-Verlag, 1991. — 266p.
9. Волькенштейн М.В. Биофизика: Учеб. руководство. — М.: Наука, Гл. ред. физ.-мат.лит., 1988. — 592 с.

Вопросы к зачету

1. Что такое нейронные сети (НС)? Что дает моделирование НС? Проблемы, возникающие при моделировании. Свойства биологических и искусственных НС. Способы реализации нейросетей.
2. Место НС среди других методов решения задач. Типы задач, решаемых нейронными сетями. Недостатки и ограничения НС.
3. Биологический нейрон. Структура, функции.
4. Нервный импульс (НИ). Возбуждение НИ, свойства НИ, примеры экспериментов.
5. Мембрана, ее структура. Мембранный потенциал. K-Na транспорт. K, Na-каналы.
6. Как возникает нервный импульс? Зависимость напряжения и токов I_K, I_{Na} от времени в импульсе. Эквивалентная схема участка волокна.
7. Сальтаторный механизм распространения НИ. Отличия от обычного механизма. Какие преимущества дает сальтаторное распространение?
8. Распространение НИ. Уравнение Ходжкина-Хаксли.
9. Пространственное описание НИ.
10. Синаптическая передача. Электрические и химические синапсы. Работа химического синапса.
11. Генерация НИ для кусочно-линейной аппроксимации ВАХ волокна.
12. Формальный нейрон. Виды функций активации. Ограниченность модели форм. нейрона.
13. Многослойный перцептрон. Структура, алгоритм работы. Этапы решения задачи с помощью НС.
14. Формализация условий задачи для НС. Примеры. Подготовка входных и выходных данных. Выбор количества слоев.
15. Обучение однослойного перцептрона. Выбор шагов по W, Theta.
16. Проблема "исключающего ИЛИ" и ее решение.
17. Перцептронная представляемость.
18. Метод обратного распространения ошибки.
19. Паралич сети. Выбор шага по параметрам. Локальные минимумы. Временна́я неустойчивость.
20. Примеры применения перцептронов.
21. Динамическое добавление нейронов. Способность НС к обобщению.
22. Обучение без учителя. Сеть с линейным поощрением.
23. Задача классификации. Сеть Кохонена.
24. Обучение слоя Кохонена. Метод выпуклой комбинации. Примеры обучения.
25. Режимы работы сети Кохонена. Применение для сжатия данных.
26. Сеть встречного распространения. Схема, обучение, свойства.
27. Генетические алгоритмы для обучения НС. Положительные качества и недостатки.
28. Послойность сети и матричное умножение. Расчет градиента квадратичной формы с помощью НС. Выбор начальной точки и длины шага.
29. Сети с обратными связями. Сеть Хопфилда. Вычислительная энергия и ее минимизация.
30. Этапы решения задачи сетью Хопфилда. Устойчивость, сходимость к эталонам.
31. Соотношение стабильности-пластичности при запоминании. Сеть АРТ-1. Структура, описание элементов сети.
32. Работа сети АРТ-1. Запоминание и классификация векторов сетью.
33. Метод имитации отжига.

Содержание

Нейронные сети 1
основные модели 1
Воронеж
1999 1
Введение 3
Параллельность обработки и реализуемость НС 5
Место нейронных сетей среди других методов решения задач 5
Биологический нейрон 6
Нервный импульс 7
Мембрана. Мембранный потенциал. 9
Натриевый насос 9
Калиевые каналы 10
Натриевые каналы 10
Возникновение нервных импульсов 10
Сальтаторный механизм распространения импульса 12
Эквивалентная схема волокна 13
Распространение нервных импульсов. Уравнение Ходжкина-Хаксли 13
Пространственное описание нервного импульса 14
Синаптическая передача 15
Генерация нервных импульсов 17
Искусственные нейронные сети 21
Формальный нейрон 21
Виды функций активации 21
Ограничения модели нейрона 23
Многослойный перцептрон 24
Алгоритм решения задач с помощью МСП 25
Формализация задачи 26
Примеры формализации задач 26
1. Задача классификации. 26
2. Распознавание букв алфавита. 28
3. Прогнозирование одномерной функции 28
4. Аппроксимация многомерной функции. 29
Выбор количества нейронов и слоев 29
Количество нейронов и слоев связано: 29
Если в сети слишком мало нейронов или слоев: 30
Если нейронов или слоев слишком много: 30
Подготовка входных и выходных данных 30
Другие способы подготовки данных 30
Методы обучения 31
Общая схема обучения перцептрона: 31
Применяются следующие методы теории оптимизации: 32
Обучение однослойного перцептрона 32
Расписание обучения 33
Перцептронная представляемость 33
1. Однослойный перцептрон. 33
2. Двухслойный перцептрон. 35
3. Трехслойный перцептрон 36
Проблема "исключающего ИЛИ" 36
Решение проблемы XOR 37
Обучение многослойного перцептрона
Алгоритм обратного распространения ошибки 37

Дальнейшее развитие алгоритма 41
Паралич сети 41
Выбор длины шага 42
Локальные минимумы 42
Чувствительность к порядку предъявления образов
Временна'я неустойчивость 43
Примеры применения перцептронов 43
I. Предсказание псевдослучайных последовательностей. 43
II. Предсказание вторичной структуры белков 45
III. Синтез речи: NET-talk 46
Динамическое добавление нейронов 47
Способность нейронных сетей к обобщению 47
Обучение без учителя 48
Сеть с линейным поощрением 48
Задача классификации
Сети Кохонена 50
Алгоритмы классификации 51
Сеть Кохонена 52
Обучение слоя Кохонена 53
1. Присвоение начальных значений 53
2. Обучение сети 54
Метод выпуклой комбинации 54
Примеры обучения 55
Модификации алгоритма обучения 56
Режимы работы сети 56
Применение сети Кохонена для сжатия данных 57
Сеть встречного расспространения
Слой Гроссберга 58
Обучение сети встречного распространения 59
Генетические алгоритмы для обучения НС 60
Применение генетических алгоритмов для обучения НС 60
Положительные качества генетических алгоритмов 61
Недостатки при обучении НС 61
Сети с обратными связями 61
Послойность сети и матричное умножение 61
Расчет градиента квадратичной формы 62
Выбор начальной точки и длины шага 63
Сеть Хопфилда 64
Решение задач с помощью сетей Хопфилда 65
Устойчивость сети 65
Сходимость к эталонам 65
Адаптивная резонансная теория (АРТ) 66
АРТ-1 66
Архитектура и работа 66
Слой сравнения 67
Слой распознавания 68
Работа сети АРТ 69
Необходимость поиска 70
Положительные качества и недостатки АРТ 70
Метод имитации отжига 71
Список литературы 73
Вопросы к зачету 74