ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №1. Реализация многослойного персептрона с алгоритмом обратного распространения ошибки для задачи глубокого обучения

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ:

1. Внимательно ознакомиться с заданием на практическую работу;
2. Изучить теоретический материал по а) лекции, б) непосредственно к данной работе.
3. Разработать план реализации задания
4. Подготовить входную выборку (если необходимо)
5. Разработать математическую модель алгоритма решения поставленной задачи
6. Программно на любом доступном языке программирования реализовать поставленную задачу
7. Провести эксперименты (если необходимо)
8. Написать отчет о проделанной работе
9. Защитить практическую работу преподавателю
10. Сдать отчет по работе преподавателю (в установленном формате)

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Выполненная практическая работа должна сопровождаться отчетом. Отчет должен содержать не менее 8 страниц текста в формате MS Word (*.doc или *.docx) или LibreOficce (*.odt). Форматирование страницы – все поля по 2 см, межстрочный интервал - полуторный, шрифт Times New Roman 14, нумерация внизу страницы справа. В отчете должны присутствовать: архитектура нейронной сети, результаты экспериментов (таблицы, графики), если разработан уникальный алгоритм построения модели, то необходима блок – схема работы алгоритма. Таблицы и рисунки обозначаются в стандартном варианте.

Весь исходный код разработанного программного обеспечения приводить не надо.

Отчет сдается только в электронном виде (распечатывать не надо).

В данной работе мы не будет собственно разрабатывать алгоритм глубокого обучения, пока остановимся только на реализации многослойного персептрона с расчетом выходов (а данный многослойный персептрон далее будет использован в сверточной нейронной сети).

Теоретическая часть

Формальные персептроны (или, лучше говорить - нейроны) могут объединяться в сети различными способами. Рассмотрим простейшее объединение – многослойный персептрон прямого распространения сигнала (связи соотвественно идут направлено от i-го слоя к i+1 - му).

Различают:

Входной слой - это начальный уровень сети, который принимает входные данные, которые будут использоваться для создания выходных данных. Входной слой не производит никаких вычислений – просто передает сигнал на первый скрытый слой (в нейронах входного слоя нет функций активации).

Скрытые слои (может быть только один скрытый слой) - в сети должен быть хотя бы один скрытый слой. Скрытые слои выполняют вычисления и операции с входными данными, чтобы получить результат на выходе. Скрытые слои связаны между собой синаптическими связями – каждый нейрон предыдущего с слоя с каждым нейроном последующего слоя.

Выходной слой – собственно выдает результат вычисления сети. В нейронах выходного слоя также есть функции активации. Обычно функции активации одни и те же в нейронах скрытых слоев и выходном слое. Но могут быть и исключения (в данном случае нужен «хитрый» способ обучения).

В то время как входные нейроны берут свои значения из окружения, значения всех других нейронов вычисляются с помощью математической функции, включающей веса и значения предшествующего слоя (см. предыдущую лекцию).

Работа многослойного персептрона описывается следующими формулами:

$$NET_{jl} = \sum_i w_{ijl} x_{ijl}$$

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

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

где индексом i обозначается 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 слоя l;

NET_{jl} - сигнал NET нейрона номер j слоя l;

Введем обозначения: w_{jl} — вектор-столбец весов для всех входов нейрона j в слое i. W_j — матрица весов всех нейронов в слоя j. В столбцах матрицы расположены вектора w_{jl}. Аналогично x_{jl} — входной вектор-столбец слоя l.

Каждый слой рассчитывает нелинейное преобразование от линейной комбинации сигналов предыдущего слоя. Многослойная нейронная сеть может формировать на выходе произвольную многомерную функцию при соответствующем выборе количества слоев, диапазона изменения сигналов и параметров нейронов. Многослойные нейронные сети являются универсальным аппроксиматором функций [4]. Общая формула, описывающая многослойный персептрон:

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

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

В некоторых источниках указывается, что способность выявлять статистические закономерности высокого порядка зависит от большого количества нейронов в скрытых слоях. Но не всегда сеть должна быть большого размера – все зависит от задачи, плюс необходима оптимизация работы сети. Нередко нейронная сеть может быть меньшего размера и может решать задачу с тем же показателем качества.

2.2 Парадигмы обучения нейронных сетей

Обычно выделяют две вида обучения нейронных сетей:

- с учителем – есть и входы и выходы;

- без учителя – есть только входы.

Но, есть еще промежуточная форма – обучение с подкреплением (о ней будет говориться в 3-х последних лекциях).

Остановимся на стандартных парадигмах. В первом случае есть и входные данные нейронной сети и соответствующие им выходы. Во втором случае есть только входные данные и нейронная сеть, обучающаяся без учителя, соответствующим образом интерпретирует входные данные на

выходы сети. Обычно это используется в задачах кластеризации и классификации. Также обучение без учителя нередко используется в задачах сжатия данных.

Обучение с учителем. Пусть мы имеем входной вектор $X = \{x_1, x_{2,..,} x_n\}$ и ответ, то есть выходной вектор $Y = \{y_1, y_2, ..., y_n\}$. Тогда задача обучения нейросети сводится к тому, чтобы сеть на X^i выдавала на выходе Y^i или значение близкое к нему. Сеть обучается путем подстраивания весов синаптических связей w_{ij}^k. Для подстройки весов связей обычно используется один из градиентных методов.

В обычном многослойном персептроне всем весам связей присваиваются случайные веса. Эти случайные веса распространяют значения по сети для получения фактического результата (за счет их и функций активации сеть и функционирует, т.е. решает поставленную задачу). Естественно, этот результат будет отличаться от ожидаемого, т.к. веса связей «забиваются случайными значениями». Разница между двумя значениями на выходе – полученным и желаемым, называется ошибкой. Обратное распространение относится к процессу отправки этой ошибки обратно по сети с автоматической корректировкой весов, так что в конечном итоге ошибка между фактическим и ожидаемым выходом сводится к минимуму. Таким образом, выход текущей итерации становится входом и влияет на следующий выход. Это повторяется до тех пор, пока не будет получен правильный результат, ну или … необходимо заново переформировать нейронную сеть, произвести коррекцию входных временных рядов. Веса в конце процесса завершившегося с устраивающей разработчика ошибкой будут теми, на которых нейронная сеть работает правильно.

Обобщенный алгоритм обучения нейронных сетей с учителем следующий:

- Шаг 1. Подготовить обучающую выборку (входы - выходы);
- Шаг 2. Предварительно рассчитать структуру сети;
- Шаг 3. Инициализировать случайным образом матрицы весов синаптических связей нейронной сети;
- Шаг 4. Подать выбранный случайным образом пример из обучающей выборки и рассчитать выходы сети (прямой проход);
- Шаг 5. Рассчитать ошибку на выходе сети и с помощью специальных формул скорретировать веса синаптических связей. Если ошибка сети меньше заданной, то вернуться на шаг 3.
- (Шаг 6) Проверить на валидационной выборке, если ошибка удовлетворяет разработчика, то сеть – в реальный «боевой» режим;
- (Шаг 7) Оптимизация работы сети – сокращение числа слоев и нейронов;

- (Шаг 8). В режиме эксплуатации зачатую необходимо переобучение, т.к. появляются новые данные.

В скобках указаны не обязательные, но на практике необходимые шаги.

Немного про предварительный расчет структуры сети. Существует несколько подходов для выбора количества связей в нейронной сети. Например, использование формулы, основанной на теореме Колмогорова – Арнольда:

$$\frac{N_Y \cdot N_P}{1+\log_2(N_P)} \le N_W \le N_Y \cdot (\frac{N_P}{N_X}+1) \cdot (N_X + N_Y + 1) + N_Y$$

Здесь необходимо упомянуть о процедуре Робинсона – Монро, датированной еще 1951 годом [**Ошибка! Источник ссылки не найден.**]!

А еще … Многие методы обучения идут от дельта – правила (или правила Видроу – Хоффа), принцип работы которого в следующем. После расчета выхода сети на выходе k - го нейрона на n - ой итерации получается значение $y_k(n)$ и есть желаемое значение выхода $d_k(n)$. Соответственно ошибка работы сети на n-ой итерации равна:

$$e_k(n) = d_k(n) - y_k(n)$$

Сигнал ошибки инициализует механизм управления, который применяет последовательность корректировок к синаптическим весам нейрона k. Эти изменения нацелены на пошаговое приближение выходного сигнала $y_k(n)$ к желаемому $d_k(n)$. Цель достигается за счет минимизации функции стоимости (еще называется индекс производительности) $E(n)$, определяемый обычно следующим образом:

$$E(n) = 0.5e_k^2(n)$$

где $E(n)$ - текущее значение ошибки. Пошаговая корректировка синаптических весов продолжается до тех пор, пока система не достигнет устойчивого состояния, то есть пока веса не стабилизируются. При этом дельта правило изменения весов задается выражением:

$$\Delta w_{kj}(n) = \eta * e_k(n) * x_j(n)$$

где η - некоторая константа задающая скорость обучения. Дельта – правило можно сформулировать следующим образом: *«Корректировка, применяемая к синаптическому весу нейрона, пропорциональна произведению сигнала ошибки на входной сигнал, его вызвавший».* Тогда новое значение синаптической связи между нейронами:

$$\Delta w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

Для решения задач с помощью нейросетей необходимо:

- Всесторонне рассмотреть задачу – нейронные сети крайне тяжелый аппарат и, может быть, подобрать другой инструмент для ее решения;
- Собрать входную выборку и провести с ней набор необходимых процедур (шкалирование, масштабирование);
- Провести статистическую обработку входных данных. Как минимум, нужен корреляционный и факторный анализ;
- После проведенного анализа выбрать входы (входные переменные) и выходы (выходные переменные);
- Рассчитать количество скрытых слоев и нейронов в них по специальным формулам;
- Сформировать сеть, инициализировать веса связей
- Обучить нейронную сеть и протестировать на валидационной выборке

Примечания:

- сильно коррелированные входы будут только мешать нейронной сети «понимать» задачу;

- в случае прогнозирования нейронная сеть должна иметь только один вход (по опыту автора), т.к. в противном случае выходы будут «тянуть одеяло на себя»;

- количество слоев и нейронов в них проще брать либо с заведомо недостаточного количества, либо с заведомо большего и далее динамически по результатам работы сети их изменять.

Алгоритм обратного распространения ошибки

Алгоритм обратного распространения ошибки был опубликован в 1986 году в работе Руммельхарта, Хинтона и Вильямса, которые считаются первооткрывателями применения градиентного спуска при обучении нейронных сетей, но первым данный алгоритм предложил Вербос в своей кандидатской диссертации в 1974 году, а также наш, советский кибернетик Галушкин в том же году.

Искусственные нейронные сети используют обратное распространение в качестве алгоритма обучения для вычисления градиентного спуска с учетом весов [2, 3, 4]. Желаемые выходы сравниваются с достигнутыми выходами системы, а затем системы настраиваются путем регулировки веса соединений, чтобы максимально сократить разницу между ними. Алгоритм получил свое название, в связи с тем, что веса обновляются в обратном направлении, от вывода к вводу. Т.е. нам известна ошибка на выходе, далее считаем изменение весов на последнем слое, затем, через ошибку на последнем слое, пересчитываем матрицу перед предпоследним слоем и т.д.

В этом алгоритме происходит распространение ошибки от выходов нейронной сети ко входам, то есть в направлении обратном распространению сигналов в обычном режиме работы (обычный режим работы - прямой). Согласно методу наименьших квадратов, минимизируемой целевой функцией ошибки нейронной сети является величина:

$$E(w) = \frac{1}{2}\sum_{j,p}(y_{j,p}^{(N)} - d_{j,p})^2$$

где $y_{j,p}^{(N)}$ реальное выходное состояние нейрона j выходного слоя N нейронной сети при подаче на ее входы p - го образа; d_{jp} - желаемое (идеальное) выходное состояние этого нейрона. Суммирование происходит по всем нейронам выходного слоя и по всем обрабатываемым сетью образам.

Важное замечание:

- если у нас в выборке, допустим, 1000 образцов или больше, то такой подход приведет к большому числу вычислений, что сильно увеличит время обучения нейросети. Поэтому для больших выборок используют т.н. «пакетный» способ расчета ошибки – берется определенный процент от выборки (допустим 10%).

- можно считать ошибку только по одному примеру, но здесь можем сильно выиграть в скорости, но проиграть в итоге в точности обучения;

- не забываем, что примеры выбираются случайным образом, т.к. нейронная сеть будет обучаться именно последовательности! Которую мы ей предлагаем.

Алгоритм обратного распространения ошибки приводить не будем – он хорошо показан во всех учебниках по нейронным сетям (Хайкин, Заенцев, Осовский и др.).

Приведем таблицу, в которой собраны наиболее распространенные алгоритмы обучения многослойного персептрона.

Т.к. самым распространенным является метод обратного распространения ошибки, то сравнение обычно проводится именно с ним.

Таблица 1

Распространенные алгоритмы обучения многослойного персептрона

№	Название	Краткий принцип работы	Плюсы - минусы
0	Многослойный персептрон и алгоритм обратного распространения	Градиентный способ обучения	«+»: - может аппроксимировать

	ошибки (BackProp)		любую функцию; - большое кол-во успешных реализаций; «-»: - медленная скорость обучения[1]; - трудность подбора: кол – ва слоев, нейронов в них, входов и выходов. - трудность отделения локальных и глобальных минимумов.
1	МП и алгоритм Левенберга – Марквардта[2]	Градиентный способ обучения	«+»: - высокая скорость обучения; - точность прогнозирования обычно выше, чем у BackProp; «-»: - может быть только один выход; - трудность реализации.
2	Многослойный персептрон и QProp	Модернизированный алгоритм обратного распространения ошибки	«+» Скорость обучения «-»: - невысокая точность обучения;
3	Многослойный	Модифицированный	«+»:

[1] Некоторые недостатки алгоритма обратного распространения ошибки нередко можно обойти специальными методами: динамическим добавлением нейронов, метод эластичного изменения весов и т.д. [**Ошибка! Источник ссылки не найден.**, **Ошибка! Источник ссылки не найден.**]

[2] Иногда называется как «Левенберга - Маркуарда».

	персептрон и RProp	градиентный способ обучения	- очень высокая скорость обучения; - не требователен к вычислительным ресурсам; «-»: - меньшая точность обучения, чем у BackProp
4	Многослойный персептрон и алгоритм сопряженных градиентов	Направление градиентного спуска выбирается таким образом, чтобы оно было ортогонально и сопряжено ко всем предыдущим направлениям	«+»: - имеет сходимость близкую к линейной; - заметно быстрее BackProp; - Невысокие требования к памяти; - Эффективен при большом количестве переменных. «-»: - После определенного количества итераций необходим рестарт процедуры расчета; - Необходимы дополнительные расчеты при рестарте
5.	Многослойный персептрон и Quikprop	Главное отличие в «факторе момента», который адаптируется к текущему результату процесса обучения.	«+»: - Содержит элементы, предотвращающие зацикливание в локальных минимумах; - Высокая скорость. «-»: - Сильно упрощенная

			формула изменения весов; - Высокая ошибка обучения
6.	Метод Ньютона	Аналогично Ньютона решения систем уравнений	«+» «-» - Требует вычисления второй производной;
7	Метод Ньютона с регулируемым шагом	$x^{k+1} = x^k + \alpha_k h^k$, при $\alpha_k = 1$ совпадает с классическим методом Ньютона.	«+» Нередко дает лучшие результаты, чем классический метод «-» - Необходимо вычисление второй производной;
8	Метод покоординатного спуска	Стандартный покоординатный спуск	«+»: «-»: - больше вычислений, чем у BackProp
9	Метод случайного поиска	Задается начальный вектор параметров. Новый вектор ищется как начальный плюс случайный, умноженный на радиус. Если после определенного количества итераций не уменьшилась ошибка, то радиус сужается и т.д.	«+»: - возможно быстрое обучение «-»: - высокая вероятность не обучить нейронную сеть;

Есть некоторые ограничения на использование того или иного вышеперечисленного алгоритма для многослойного персептрона.

Возникает вопрос – какой задать ошибку обучения нейронной сети? Вообще говоря, все зависит от решаемой задачи. Допустим, мы прогнозируем курс доллара к рублю, тогда ошибка в 5%, если мы угадали направление движения курса, это будет существенный результат. Да даже 10%. Если мы распознаем 10 образов и сеть в итоге ошибается в одном, то это тоже не плохо (если это не система контроля управления доступом). И так далее.. все зависит от задачи.

Алгоритм RProp

Данный алгоритм является модернизацией алгоритма обратного распространения ошибки. Принцип действия алгоритма и количество шагов то же. За исключением формул корректировки. RProp хорошо работает во многих ситуациях, потому что он динамически адаптирует размер шага для каждого веса независимо. Большинство вариантов градиентного спуска используют знак и величину градиента. Градиент указывает направление наискорейшего подъема. Поскольку обычно мы хотим найти минимум, мы следуем градиенту в противоположном направлении и это направление полностью определяется знаком градиента. Формулы:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ \Delta_{ij}^{(t)}, \frac{\partial E^{(t)}}{\partial w_{ij}} \frac{\partial E^{(t-1)}}{\partial w_{ij}} > 0 \\ \eta^- \Delta_{ij}^{(t)}, \frac{\partial E^{(t)}}{\partial w_{ij}} \frac{\partial E^{(t-1)}}{\partial w_{ij}} < 0 \end{cases},$$

$$0 < \eta^- < 1 < \eta^+$$

Т.е. если знак производной не изменился, то продолжаем изменять в том же направлении для быстрой сходимости, если изменился, то в противоположном. Данный алгоритм в несколько раз быстрее, чем backprop, но менее точен. Его можно использовать для «макетирования», т.е. если с помощью RProp задача решается с приемлемой точностью, то можно для большей точности применить стандартный алгоритм обратного распространения ошибки. Алгоритм хорошо разобран на сайте www.basegroup.ru (доступен также исходный код).

Методы повышения скорости обучения

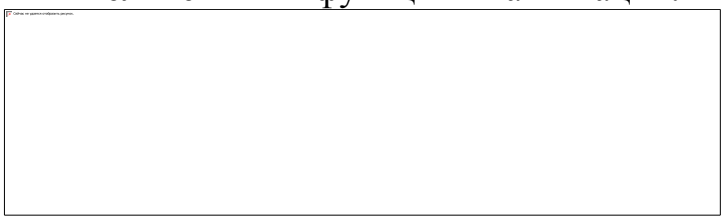
Выделяют следующие способы повышения скорости обучения многослойного персептрона с помощью градиентных методов (подробнее см. лекционный материал):

- динамическое изменение количества нейронов в скрытых слоях (обычно в сторону повышения);
- обучение по расписанию;
- использование априорных знаний об исследуемой задаче (выходит за рамки данной практической работы).

ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ

Часть 1.

1. Программно реализовать формальный нейрон с двумя входами, распознающий границу черное и белое. Зачерненные клетки соответсвуют единичному сигналу, а белые клетки - нулевому. Сигнал на входах формального нейрона устанавливается равным значениям пар примыкающих клеток рассматриваемого образа. Нейрон обучается всякий раз возбуждаться и выдавать единичный выходной сигнал, если его первый вход (на Рис. 1. - левый) соединен с белой клеткой, а второй (правый) - с черной. Таким образом, нейрон должен служить детектором границы перехода от светлого к темному тону образа. Провести эксперименты с линейными и нелинейными функциями активации.



2. Программно реализовать многослойный персептрон на любом языке программирования. Программа должна иметь удобный интерфейс. Программа должна позволять выбирать количество входов, выходов, скрытых слоев и нейронов в них. Так как разработанный в данной работе многослойный персептрон будет использовать и в следующей работе, то необходимо на панели управления оставить место под график ошибки и параметры обучения многослойного персептрона.

Часть 2.

1. Согласно последней цифре в номере зачетной книжке студента в таблице 1 выбрать метод обучения для многослойного персептрона и реализовать его программно на любом доступном языке программирования.
2. Выбрать тип задачи (прогнозирование, классификация, восстановление зависимости).
3. Выбрать предметную область, найти и подготовить обучающую выборку (возможно, самый большой набор датасетов на сайте http://www.kaggle.com). Также выборки можно найти на сайтах государственной и региональной статистики.
4. Обучить разработанную нейронную сеть на сформированной выборке и протестировать работу сети на валидационной выборке.
5. Провести эксперименты и сформировать таблицу по влиянию количества скрытых слоев и нейронов

Программа должна позволять выбирать количество входов, выходов, скрытых слоев и нейронов в них, а также обеспечивать выбор интервалов обучающей и валидационной выборки.

Допускается, что разработанное программное обеспечение будет иметь консольный интерфейс.

Примечание: примеры исходных кодов для нейронных сетей автора можно найти по адресу http://apsheronsk.bozo.ru/Neural/Neural1.htm .

Полезный совет для реализации программной реализации нейронной сети:

Существует два основных способа реализации нейронных сетей – это объектно – ориентированный подход (классы можно использовать в дальнейших лабораторных работах) и процедурно – функциональный, т.к. работа нейронной сети и ее обучение суть перемножение матриц и векторов (второй проще).

Контрольные вопросы к практической работе №1

1. Как выбирается количество слоев и нейронов в них в многослойном персептроне в зависимости от задачи и исходных данных?
2. Какие методы повышения скорости обучения многослойного персептрона Вы знаете?
3. Как влияет на скорость обучения динамическое добавление нейронов в процессе обучения?
4. Возможно ли менять структуру многослойного персептрона в процессе обучения? Если да, то как и к чему это может привести?
5. Как измеряется ошибка обучения многослойного персептрона?
6. Как определить необходимый размер обучающей выборки?
7. Как определить размер валидационной выборки?
8. Какие параметры регулируются в алгоритме обратного распространения ошибки?

Литература:

2. Рутковская Д., Пилиньский М.. Рутковский Л. «Нейронные сети, генетические алгоритмы и нечеткие системы». Пер. с польск., И.Д. Рудинского. М.: Горячая линия – Телеком, 2006. 452 с.
3. Хайкин С. Нейронные сети: полный курс. 2-е изд.: Пер. с. англ. – М.: Издательский дом «Вильямс», 2006 – 1104 с.

4. Заенцев И. В. Нейронные сети: основные модели. Учебное пособие. Воронеж: ВГУ. 1998.- 76с.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №2.

Реализация радиально – базисной сети

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ:

11. Внимательно ознакомиться с заданием на практическую работу;
12. Изучить теоретический материал по а) лекции, б) непосредственно к данной работе.
13. Разработать план реализации задания
14. Подготовить входную выборку (если необходимо)
15. Разработать математическую модель алгоритма решения поставленной задачи
16. Программно на любом доступном языке программирования реализовать поставленную задачу
17. Провести эксперименты (если необходимо)
18. Написать отчет о проделанной работе
19. Защитить практическую работу преподавателю
20. Сдать отчет по работе преподавателю (в установленном формате)

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Выполненная практическая работа должна сопровождаться отчетом. Отчет должен содержать не менее 8 страниц текста в формате MS Word (*.doc или *.docx) или LibreOficce (*.odt). Форматирование страницы – все поля по 2 см, межстрочный интервал - полуторный, шрифт Times New Roman 14, нумерация внизу страницы справа. В отчете должны присутствовать: архитектура нейронной сети, результаты экспериментов (таблицы, графики), если разработан уникальный алгоритм построения модели, то необходима блок – схема работы алгоритма. Таблицы и рисунки обозначаются в стандартном варианте.

Весь исходный код разработанного программного обеспечения приводить не надо.

Отчет сдается только в электронном виде (распечатывать не надо).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Сеть радиально – базисных функций (Radial Basis Function Neural Networks, RBFN) – это искусственная нейронная сеть, использующая радиально

базисные функции в качестве функций активации. Выходом сети является линейная комбинация радиальных базисных функций входов и параметров нейрона. Нейроны данной сети обладают выраженными локальными характеристиками. Сети радиальных базисных функций имеют множество применений, в том числе функции приближения, прогнозирования временных рядов, классификации и системы управления [Википедия]. Основная идея близка (или, можно сказать, восходит) к методу потенциальных функций и оценке по Парзену (машинное обучение или теория распознавания образов). Радиально – базисная сеть является универсальным аппроксиматором подобно многослойному персептрону.

Сети RBF аналогичны сетям кластеризации K-средних и сетям PNN / GRNN. Базовое отличие состоит в том, что сети PNN / GRNN имеют по одному нейрону для каждой точки в обучающей выборке, тогда как сети RBF имеют переменное количество нейронов, которое обычно намного меньше, чем количество обучающих точек. Для задач с обучающими наборами малого и среднего размера сети PNN / GRNN обычно более точны, чем сети RBF, но сети PNN / GRNN плохо работают на больших обучающих наборах.

Впервые сформулированы в 1988 Брумхедом и Лоу (но есть данные, что ее первыми предложили Moody и Darken также в 1988 году, но немного ранее). При этом реализуются идеи, сформированные Т.Кавером, которую можно, немного упрощенно, озвучить следующим образом: «линейно неразделимая задача в пространстве размерностью n, может стать линейно разделимой в пространстве с большей размерностью h» (очевидно, что $h>n$). Но вначале про радиально – базисные функции (РБФ).

Радиальная функция – это функция $f(x)$, зависящая только от расстояния между x и фиксированной точкой пространства X. Можно также сформулировать следующим образом: «Радиальная базисная функция – это функция, которая изменяется при удалении от местоположения». Другое важное определение – «Суперпозиция сигналов, поступающих от всех таких нейронов, которая выполняется выходным нейроном, позволяет получить отображение всего многомерного пространства»[3]. Линейные комбинации РБФ можно использовать для аппроксимации заданной функции. Их характерное свойство (радиальных функций) заключается в том, что отклик функции монотонно убывает (возрастает) с удалением от центральной точки. Типичный пример такой функции – функция Гаусса:

$$h(x) = \exp(-\frac{(x-c)^2}{r^2})$$

или

[3] https://intuit.ru/studies/courses/61/61/lecture/20450

$$h(x) = \phi(\frac{x^2}{d^2})$$

где x – вектор входных сигналов нейрона, а d – ширина окна функции. При этом h – убывающая функция.

Для $c = 0$ и $r = 1$ функция Гаусса представлена на Рисунке

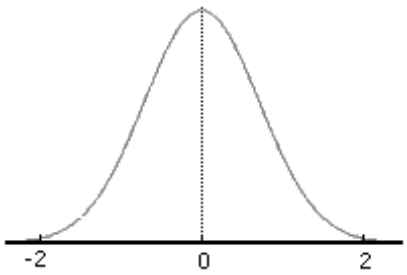


Рисунок 1. Функция Гаусса

Кроме функции Гаусса, также существуют:

- мультиквадратичная;

- обычная квадратичная;

- обратная мультиквадратичная;

- полигармонический сплайн и другие радильно – базисные функции [**Ошибка! Источник ссылки не найден.**].

(Кстати, радиальные базисные функции используются в машинах опорных векторов в машинном обучении!).

Радиально – базисный нейрон

(*иногда пишут радиальный базисный нейрон*). Пусть у радиально – базисного нейрона есть R входов. Соответственно радиально – базисный нейрон вычисляет расстояние между векторами входов радиально – базисной сети X и вектор весов W, а далее умножает его на некоторый фиксированный порог b.

Выход радиально – базисного нейрона равен максимум, т.е. «1», если его входы нулевые. Поэтому, радиально – базисный нейрон действует, как детектор и выдает на выходе 1, если его вход X равен вектору его весов W. Фиксированный порог b влияет на чувствительность радиально – базисного нейрона [1]. Радиальный нейрон представляет собой гиперсферу вокруг центральной точки.

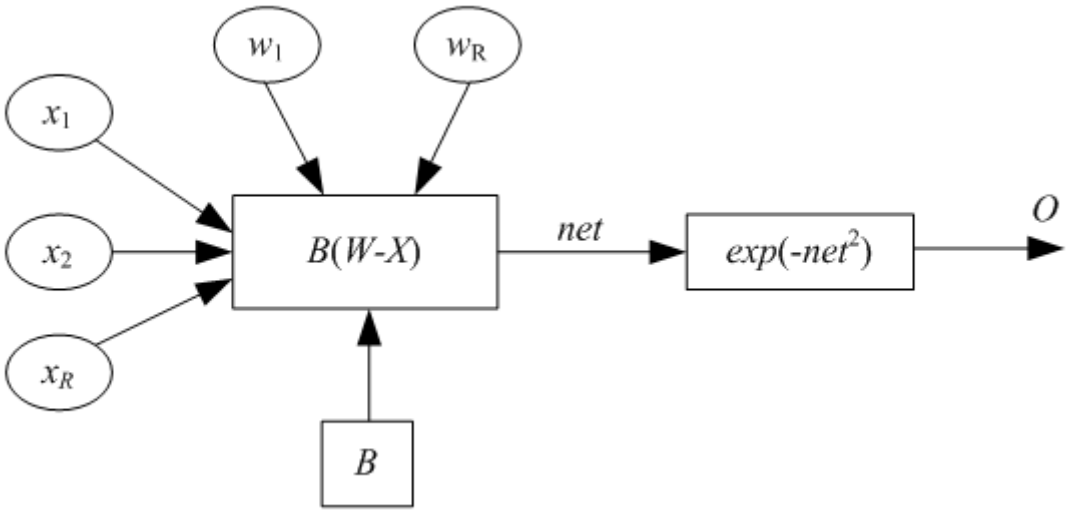


Рисунок 2. Радиально - базисный нейрон

Структура радиально – базисной сети – это нейронная сеть, которая содержит слой скрытых нейронов с радиально симметричной активационной, каждый из которых предназначен для хранения отдельного эталонного вектора (в виде вектора весов). Можно привести и другое определение – радиально – базисная нейронная сеть состоит из двух слоев нейронов – скрытого радиального базисного слоя и выходного линейного слоя. Нейроны первого слоя вычисляют расстояние между входным вектором образцом и векторами весов первого слоя, которые формируются из строк первой матрицы.

Количество нейронов во входном слое равно размерности вектора признаков. Точно так же количество узлов в выходном слое соответствует количеству классов.

Для построения RBFN необходимо выполнение следующих условий.

Во-первых, наличие эталонов, представленных в виде весовых векторов нейронов скрытого слоя. Во-вторых, наличие способа измерения расстояния входного вектора от эталона. Обычно это стандартное евклидово расстояние. В-третьих, специальная функция активации нейронов скрытого слоя, задающая выбранный способ измерения расстояния. Обычно используется функция Гаусса, существенно усиливающая малую разницу между входным и эталонным векторами.

Или, другими словами, выходной сигнал эталонного нейрона скрытого слоя y_i - это функция (гауссиан) только от расстояния p_i между входными и эталонными векторами.

Обучение слоя образцов-нейронов сети подразумевает предварительное проведение кластеризации для нахождения эталонных векторов и определенных эвристик для определения значений δ_i.

Нейроны скрытого слоя соединены по полно-связной схеме с нейронами выходного слоя, которые осуществляют взвешенное суммирование. Для нахождения значения весов от нейронов скрытого к выходному слою используется линейная регрессия.

Каждый радиально – базисный нейрон производит следующее нелинейное преобразование:

$$y_i = F_i(x) = w_{j0} + \sum_{i=1}^{h} w_{ji}\phi_i(x)$$

где $\phi_i(x)$ – радиально – базисные функции, которые определяют характер отображения из пространства входов в пространство выходов.

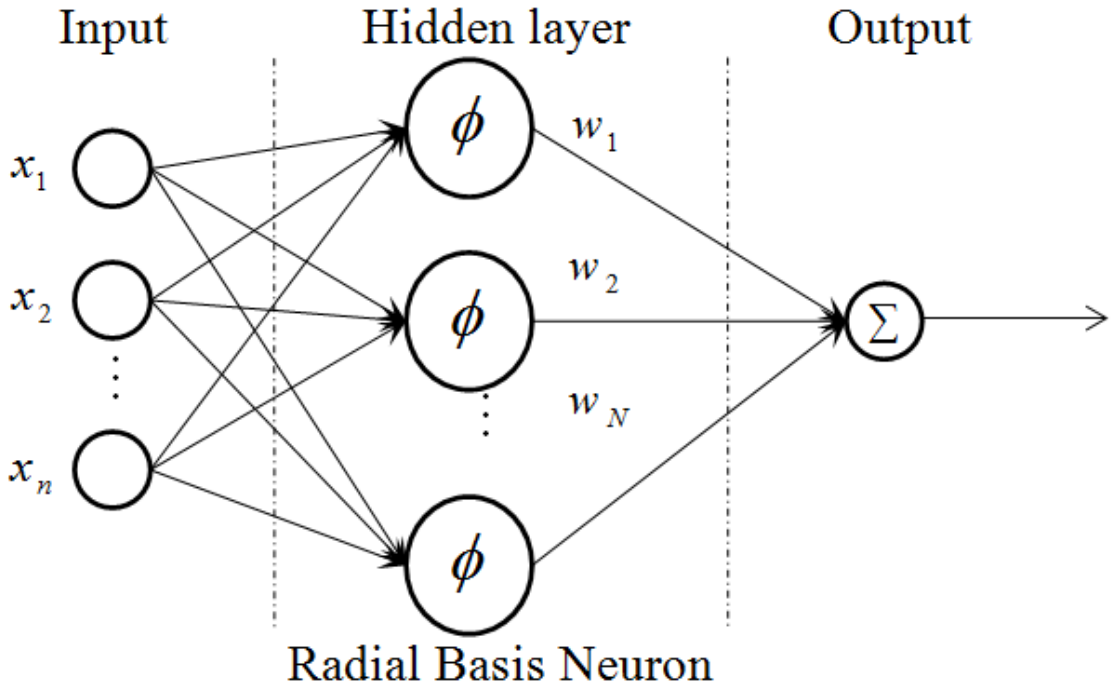


Рисунок 3. Радиально - базисная нейронная сеть (схема ResearchGate.com)

Пример кода радиально – базисной сети с сайта Keras

```
def prior(kernel_size, bias_size, dtype=None):
    n = kernel_size + bias_size
    prior_model = keras.Sequential(
        [
            tfp.layers.DistributionLambda(
                lambda t: tfp.distributions.MultivariateNormalDiag(
                    loc=tf.zeros(n), scale_diag=tf.ones(n)
                )
            )
        ]
    )
```

```
    return prior_model
def posterior(kernel_size, bias_size, dtype=None):
    n = kernel_size + bias_size
    posterior_model = keras.Sequential(
        [
            tfp.layers.VariableLayer(
                tfp.layers.MultivariateNormalTriL.params_size(n), dtype=dtype
            ),
            tfp.layers.MultivariateNormalTriL(n),
        ]
    )
    return posterior_model
        // создание модели

def create_bnn_model(train_size):
    inputs = create_model_inputs()
    features = keras.layers.concatenate(list(inputs.values()))
    features = layers.BatchNormalization()(features)

    # Create hidden layers with weight uncertainty using the DenseVariational layer.
    for units in hidden_units:
        features = tfp.layers.DenseVariational(
            units=units,
            make_prior_fn=prior,
            make_posterior_fn=posterior,
            kl_weight=1 / train_size,
            activation="sigmoid",
        )(features)

    # The output is deterministic: a single point estimate.
    outputs = layers.Dense(units=1)(features)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model
        // обучение байесовской нейронной сети

num_epochs = 500
train_sample_size = int(train_size * 0.3)
small_train_dataset =
train_dataset.unbatch().take(train_sample_size).batch(batch_size)

bnn_model_small = create_bnn_model(train_sample_size)
run_experiment(bnn_model_small, mse_loss, small_train_dataset, test_dataset)
```

Алгоритм обучения радиально – базисной нейронной сети

Радиально – базисная нейронная сеть обучается в три этапа:

1 этап: назначение центров в РНБ – слое. Эти центры оптимизируются с помощью обучения без учителя. Для этого можно использовать, например, алгоритм Кохонена. Здесь желательно разместить центры, отражая кластеризацию входных данных [1]. При этом возникает две подзадачи – определение центров и расчет парамеров ширины функции. При этом должно обеспечиваться – равномерность распределения и полнота покрытия области определения[4].

2 этап: определение отклонений алгоритмом «ближайшего соседа».

3 этап: линейная оптимизация по дельта-правилу, в направлении обратном распространению ошибки.

Отметим, что в ряде источников выделяют только два этапа обучения радиально – базисной сети.

Также выделяют гибридный алгоритм обучения радиально – базисной сети, который функционирует за счет чередования двух этапов - отыскания коэффициентов линейного слоя, т.е. выходного и адаптацией параметров радиальных функций во входном слое. Каждый такой итерационный шаг обучения заканчивается уточнением параметров радиальных функций.

Выделяют также обучение радиально – базисной сети с помощью алгоритма кластеризации.

Существуют также варианты обучения радиально – базисной сети с помощью генетических алгоритмов (гибридные интеллектуальные системы).

Недостатки радиально – базисных сетей:

- предварительно должно быть известно число эталонов, а также эвристики для построения активационных функций нейронов скрытого слоя;

Достоинства радиально – базисных сетей:

- в них нет этапа обучения в обычном понимании (т.е. повышенная скорость).

Сети RBF представляют собой, в отличие от многослойного персептрона, локальные аппроксиматоры нелинейного отображения ввода-вывода. Их основные преимущества - короткая фаза обучения и пониженная чувствительность к порядку представления обучающих данных. Однако во многих случаях мы обнаруживаем, что гладкое отображение достигается только в том случае, если количество радиальных базисных функций, необходимых для охвата входного пространства, становится очень большим.

[4] http://www.igce.comcor.ru/AI_mag/NN/RadNets/RadNets.html

Это затрудняет возможность применения данных сетей для многих практических приложений.

Отметим существование сети General regression neural network (GRNN)

Нейронная сеть регрессии

(Regression neural network)

Данные сети также часто называют Байесовскими вероятностными сетями регрессии.

Нейронная сеть регрессии довольна похожа на радиально – базисную нейронную сеть и применяется практически для такого же круга задач. Первый слой сети регрессии идентичен первому слою радиально – базисной сети, а второй слой строится на принципе обобщенной сети регрессии. Второй слой высчитывает поэлементное произведение строки W и вектора выхода первого слоя.

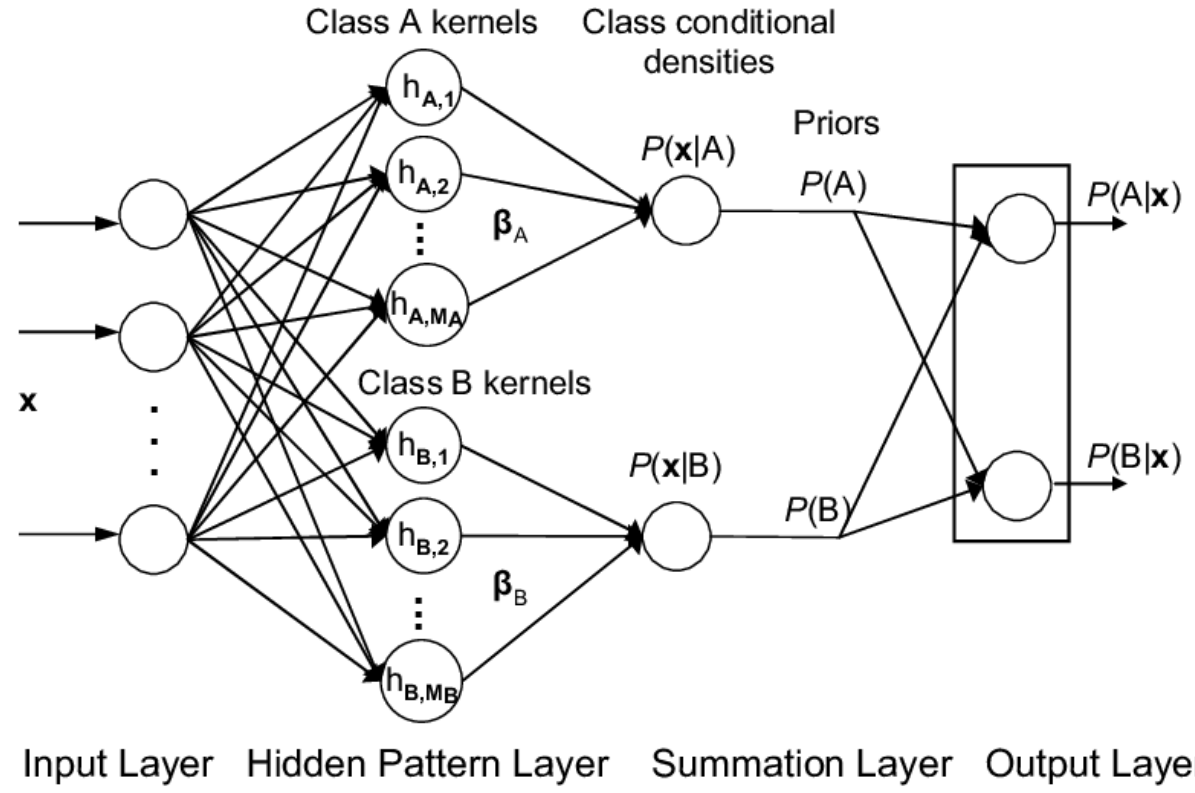


Рисунок 4. Байесовская вероятностная сеть регрессии (схема – http://ResearhGate.net)

На рисунке 4 показана нейронная сеть регрессии с четырьмя слоями: входной, выходной, слой радиальных центров (центры – кластеры обучающих данных), слой элементов регрессии. Слои могут содержать квадратную матрицу потерь.

Байесовские вероятностные сети используются только для задач классификации.

Есть еще обширный класс (правда, на практике не получил широкого распранения) Вероятностных нейронных сетей (ВНС)

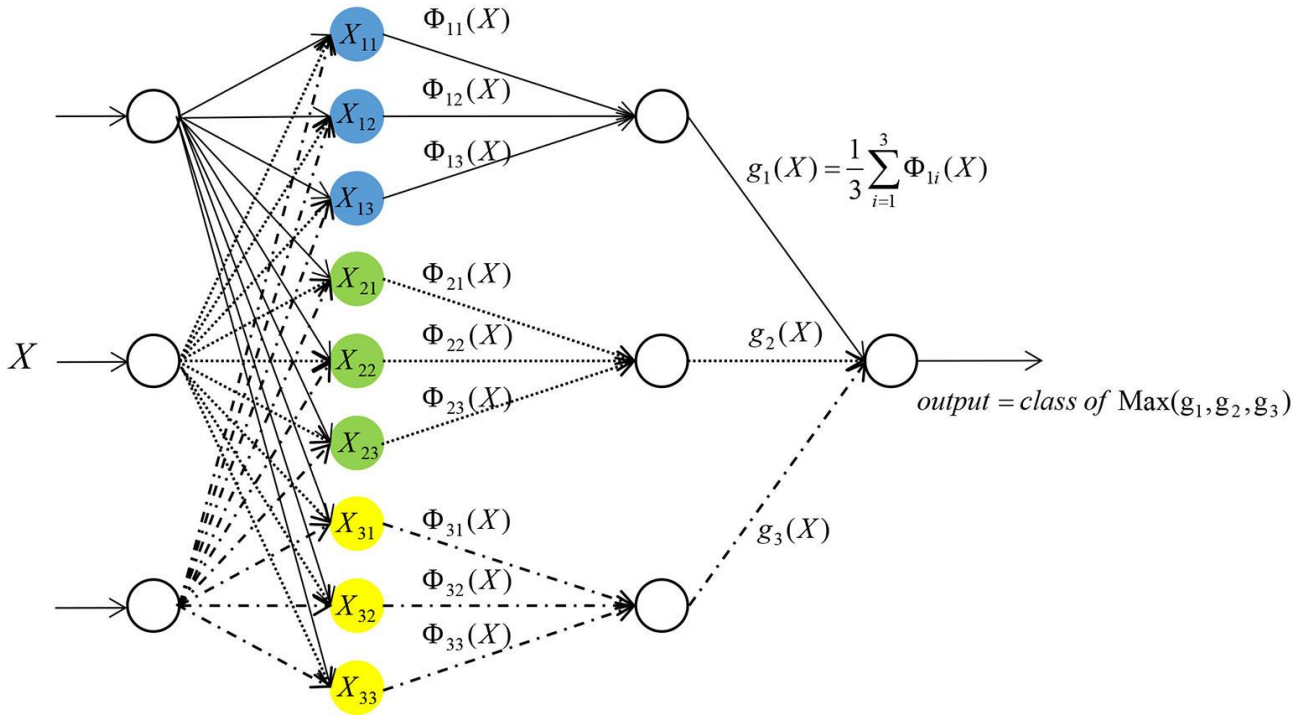


Рисунок 5. Пример нейронной сети с сайта https://www.frontiersin.org

Пример реализации сети PBNN на Keras

Создание наборов данных для обучения и оценки[5]

```
def get_train_and_test_splits(train_size, batch_size=1):
    # We prefetch with a buffer the same size as the dataset because th dataset
    # is very small and fits into memory.
    dataset = (
        tfds.load(name="wine_quality", as_supervised=True, split="train")
        .map(lambda x, y: (x, tf.cast(y, tf.float32)))
        .prefetch(buffer_size=dataset_size)
        .cache()
    )
    # We shuffle with a buffer the same size as the dataset.
    train_dataset = (
        dataset.take(train_size).shuffle(buffer_size=train_size).batch(batch_size)
    )
    test_dataset = dataset.skip(train_size).batch(batch_size)
```

[5] Пример с официального сайта Keros https://keras.io/

```
    return train_dataset, test_dataset
```

Далее обучение сети

```
hidden_units = [8, 8]
learning_rate = 0.001


def run_experiment(model, loss, train_dataset, test_dataset):

    model.compile(
        optimizer=keras.optimizers.RMSprop(learning_rate=learning_rate),
        loss=loss,
        metrics=[keras.metrics.RootMeanSquaredError()],
    )

    print("Start training the model...")
    model.fit(train_dataset, epochs=num_epochs, validation_data=test_dataset)
    print("Model training finished.")
    _, rmse = model.evaluate(train_dataset, verbose=0)
    print(f"Train RMSE: {round(rmse, 3)}")

    print("Evaluating model performance...")
    _, rmse = model.evaluate(test_dataset, verbose=0)
    print(f"Test RMSE: {round(rmse, 3)}")
```

Создание выходов модели:

```
FEATURE_NAMES = [
    "fixed acidity",
    "volatile acidity",
    "citric acid",
    "residual sugar",
    "chlorides",
    "free sulfur dioxide",
    "total sulfur dioxide",
    "density",
    "pH",
    "sulphates",
    "alcohol",
]
```

```
def create_model_inputs():
    inputs = {}
    for feature_name in FEATURE_NAMES:
        inputs[feature_name] = layers.Input(
            name=feature_name, shape=(1,), dtype=tf.float32
        )
    return inputs
```

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ

1. Подготовить массивы входных векторов и целей для радиальной базисной нейронной сети.

2. Реализовать сеть программно, определить количество нейронов сети и параметры сети для поставленной задачи.

3. Исследовать работу сети для разных значений среднеквадратичной ошибки (количество нейронов).

4. Исследовать работу сети при различных радиально – базисных функциях, определить лучшую из них для решаемой задачи.

5. Построить необходимые графики.

6. Написать отчет о проделанной работе и сдать его преподавателю.

Возможные варианты применения радиальной базисной сети в лабораторном практикуме:

- классификация и распознание изображений;

- классификация многомерных объектов.

- прогнозирование временных рядов.

Примечание: одним из лучших существующих программных средств для реализации радиально – базисных сетей является Matlab Neural Toolbox [1].

Контрольные вопросы к практическому занятию

1. Какую функцию называют радиально – базисной?
2. Какое отличие топологии радиально-базисной сети от многослойного персептрона?
3. Сколько выходов может быть у радиально – базисной сети?

4. Каким образом происходит обучение радиально – базисной сети?
5. Почему радиально – базисные сети реже используется, чем многослойный персептрон?
6. Для каких задач можно использовать радиально – базисные сети? Приведите примеры.
7. В чем преимущества и недостатки радиально – базисных сетей?
8. В чем отличие радиально – базисной сети от нейронной сети регрессии?
9. Каким образом функционирует байесовская вероятностная сеть регрессии?

Литература:

1. Ярушкина Н.Г. Основы теории нечетких и гибридных систем. М.: Финансы и статистика. 2004. 320 с.
2. Хайкин С. Нейронные сети: полный курс. 2-е изд.: Пер. с. англ. – М.: Издательский дом «Вильямс», 2006 – 1104 с.
3. Суровцев И. С., Клюкин В. И., Пивоварова Р. П., Нейронные сети. – Воронеж: ВГУ, 1994. – 224с.
4. Заенцев И. В. Нейронные сети: основные модели. Учебное пособие . Воронеж: ВГУ. 1998.- 76с.
5. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер. с польск. И.Д. Рудинского. М.: Горячая линия – Телеком. 2006. с. 452.
6. Specht, D. F. (2002-08-06). "A general regression neural network". IEEE Transactions on Neural Networks. 2 (6): 568–576.
7. Fallah, Nader; Gu, Hong; Mohammad, Kazem; Seyyedsalehi, Seyyed Ali; Nourijelyani, Keramat; Eshraghian, Mohammad Reza (2009). "Nonlinear Poisson regression using neural networks: A simulation study". Neural Computing and Applications. 18 (8): 939–943.
8. J. Moody and C. J. Darken, "Fast learning in networks of locally tuned processing units," Neural Computation, 1, 281-294 (1989)
9. Martin D. Buhmann (2003). Radial Basis Functions: Theory and Implementations. Cambridge University.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №3. Реализация рекуррентной нейронной сети для задачи глубокого обучения

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ:

21. Внимательно ознакомиться с заданием на практическую работу;

22. Изучить теоретический материал по а) лекции, б) непосредственно к данной работе.
23. Разработать план реализации задания
24. Подготовить входную выборку (если необходимо)
25. Разработать математическую модель алгоритма решения поставленной задачи
26. Программно на любом доступном языке программирования реализовать поставленную задачу
27. Провести эксперименты (если необходимо)
28. Написать отчет о проделанной работе
29. Защитить практическую работу преподавателю
30. Сдать отчет по работе преподавателю (в установленном формате)

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Выполненная практическая работа должна сопровождаться отчетом. Отчет должен содержать не менее 8 страниц текста в формате MS Word (*.doc или *.docx) или LibreOficce (*.odt). Форматирование страницы – все поля по 2 см, межстрочный интервал - полуторный, шрифт Times New Roman 14, нумерация внизу страницы справа. В отчете должны присутствовать: архитектура нейронной сети, результаты экспериментов (таблицы, графики), если разработан уникальный алгоритм построения модели, то необходима блок – схема работы алгоритма. Таблицы и рисунки обозначаются в стандартном варианте.

Весь исходный код разработанного программного обеспечения приводить не надо.

Отчет сдается только в электронном виде (распечатывать не надо).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Краткая классификация рекуррентных сетей:

- сети, имеющие обратные связи локального типа;
- сети, имеющие связи глобального типа.

По большому, существует немного вариантов, как замкнуть обратную связь от нейрона (наверно, все они уже испробованы в том или ином виде.)

Рекуррентные нейронные сети

Рекуррентные нейронные сети (*англ. Recurrent neural network; RNN*) – семейство топологий искусственных нейронных сетей, в которых связи между элементами образуют направленную последовательность, благодаря которой появляется возможность обрабатывать серию событий (последовательную). Еще одна важная особенность – имеется использовать свою внутреннюю память для обработки последовательностей произвольной длины. Рекуррентные сети часто используют для решения задач, где целое можно

разбить на части, например, распознавание машинописного и рукописного текста, распознавание речи, работа с видеопотоком и др.

Рекуррентные нейронные сети отличаются от нейросетей прямого распространения сигнала наличием хотя бы одной обратной связи. Таким образом, в этой сети может существовать один слой с обратными связями, а также может быть нейроны с обратной связью, где выход нейрона возвращается в себя как вход. Наличие обратной связи оказывает глубокое влияние на способность к обучению сеть и дает ей некоторые особые свойства (есть аналогия с обратной связью в ТАУ). Кроме того, в этих контурах обратной связи могут используются дополнительные ветви, состоящие из элементов единичной задержки, которые приводят к нелинейному динамическому поведению в силу нелинейного характера нейронов (точнее – их функций активации). Нелинейная динамика имеет ключевое значение роль в рекуррентных сетях.

В отличие от сетей с прямым распространением сигнала, рекуррентные сети могут быть чувствительны и адаптированы к прошлым данным (т.е. они хранят последовательность сигналов за несколько временных итераций). Среди нескольких Архитектуры рекуррентных нейронных сетей более подходящие для моделирования и управления нелинейными системами.

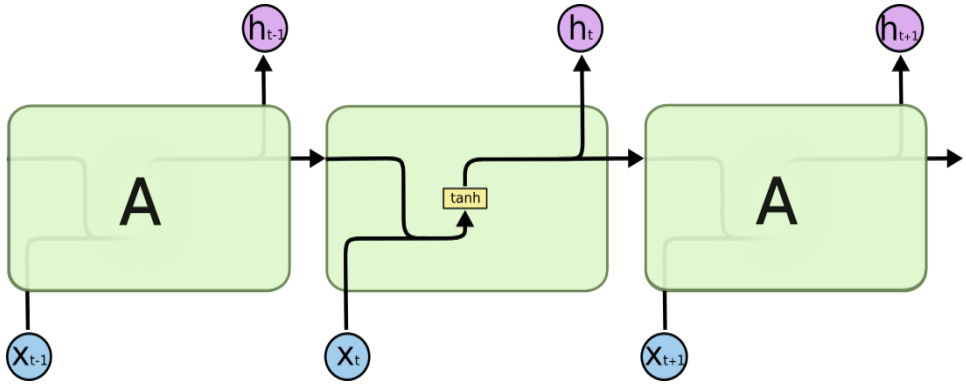


Рисунок 6. Схема рекуррентной сети

Количество типов рекуррентных нейронных сетей достаточно велико, но можно выделить несколько базовых вариантов:

- Сеть с долговременной и кратковременной памятью (LSTM);

- Управляемый рекуррентный блок (GRU).

Обычно топология рекуррентной нейронной сети имеет довольно простую структур, которая состоит из чередующихся нейронов, функция активации которых в большинстве случаев – гиперболический тангенс.

Но! Самой первой рекуррентной нейронной сетью по сути является сеть Хопфилда, которая была предложена в 1982 году. Другой известной с 80-х годов рекуррентной сеть является сеть Хемминга.

Сеть LSTM

Сеть с долговременной и кратковременной памятью LSTM (другое название – *долгая краткосрочная память*). Данная сеть была предложена в 1997 году З. Хохрайтером и Ю. Шмидхубертом (Германия). Сеть LSTM хорошо приспособлена для решения задач классификации и прогнозирования, когда временные ряды разделены интервалами различной длительности. В частности, считается (и подтверждено экспериментально), что LSTM хорошо работает с распознаванием рукописного текста. Сеть LSTM состоит из цепочек повторяющихся блоков.

Рекуррентные сети LSTM и GRU имеют топологию «цепочка» из повторяющихся блоков (см. рисунок ниже). При этом базовая версия LSTM состоит из четырех взаимодействующих между собой слоев нейронов.

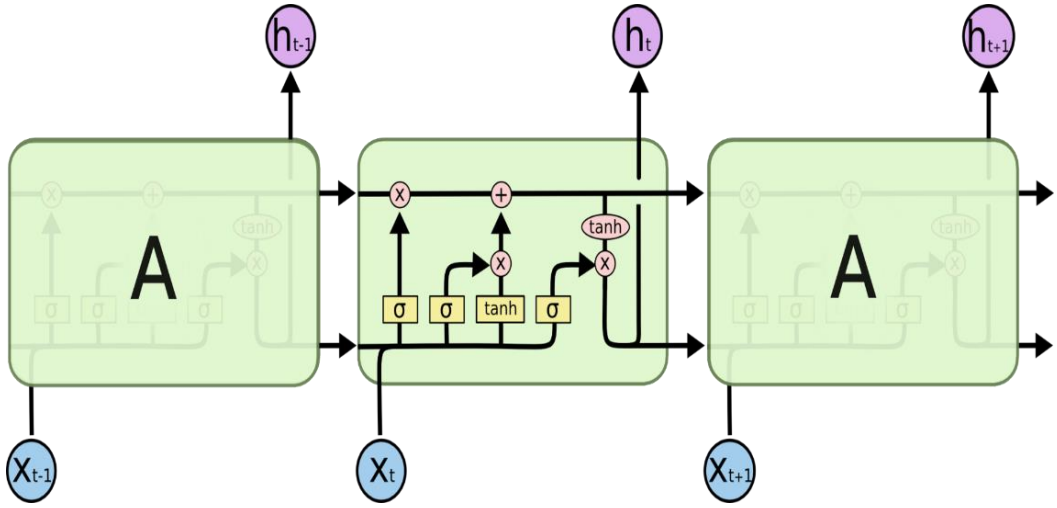


Рисунок 7. Схема LSTM сети (Википедия)

Алгоритм работы сети LSTM:

1. Определение ненужной (или избыточной) информации из входного сигнала. Это происходит в первом (сигмоидальном) слое. Сигмоидальный слой возвращает «0» (игнорировать) или «1» (сохранить) для каждого числа из состояния ячейки. Формула вычисления:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

2. Работа слоя входного фильтра
   2.1. Первый слой входного фильтра (функция активации - сигмоида). В данном слое определяется, какие значения необходимо обновить.

2.2. Второй слой фильтра забвения формирует вектор новых значений (по определенным на первом слое входного фильтра). Формулы работы:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_c * [h_{t-1}, x_t] + b_c)$$

3. Процедура замены старого состояния ячеек памяти на новые значения по формуле:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

4. Определение выходных значений сети LSTM
   4.1. Применение сигмоидального фильтра
   4.2. Применение фильтра с гиперболическим тангенсом – значения ячеек проходят через данный фильтр и умножаются на выходы сигмоидального слоя. Формулы:

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

На последнем шаге необходимо определить, что нужно получить на выходе сети. Выходные данные будут основаны на состоянии ячейки с применением некоторых фильтров. Первым делом применяется сигмоидальный фильтр для определения информации, которая будет подаватьбся на выход. После этого все значения состояния ячейки проходят через слой гиперболического тангенса, для того чтобы на выходе получить значения от -1 до 1 и перемножаются с данными полученными на выходе сигмоидального слоя. Это позволяет выводить лишь необходимую информацию (источник – Википедия).

Нейронная Сеть управляемых рекуррентных блок GRU

По топологии сеть управляемых градиентных блоков GRU (англ. Gated Recurrent Unit) управляемых рекуррентных блоков является упрощенной сетью LSTM. Предложен в 2014 году (авторы: Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun & Bengio, Yoshua). В данной сети нет выходных вентилей, а также вентиль фильтра забвения и входной вентиль объединены в один вентиль, если сравнивать с LSTM, т.е. меньше параметров и вычислений. Сеть GRU обучается быстрее, чем LSTM, но по результатам экспериментов дает хуже результат. Также существуют ограничения на длину входной информации. Можно использовать сеть GRU для макетирования нейросетевого решения поставленной задачи и, в случае успешного результата, решать задачу на сети LSTM.

GRU направлен на решение проблемы исчезающего (уменьшающегося) градиента (*Vahishing gradient problem*), которая появляется при работе со стандартной рекуррентной нейронной сетью - в некоторых случаях градиент будет крайне малым, что фактически не позволит синаптическому весу изменить свое значение. В худшем случае это может полностью остановить

дальнейшее обучение нейронной сети. GRU также можно рассматривать как разновидность LSTM, потому что оба они спроектированы одинаково и в некоторых случаях дают одинаково хорошие результаты. Чтобы решить проблему исчезающего градиента, GRU использует вентили обновления и сброса. По сути, это два вектора, которые определяют, какую информацию нужно передать на выход. Их особенность заключается в том, что их можно обучить хранить информацию за несколько итераций (можно расширять до определенных размеров), не стирая ее во времени и не удаляя информацию, не имеющую отношения к текущей решаемой задаче.

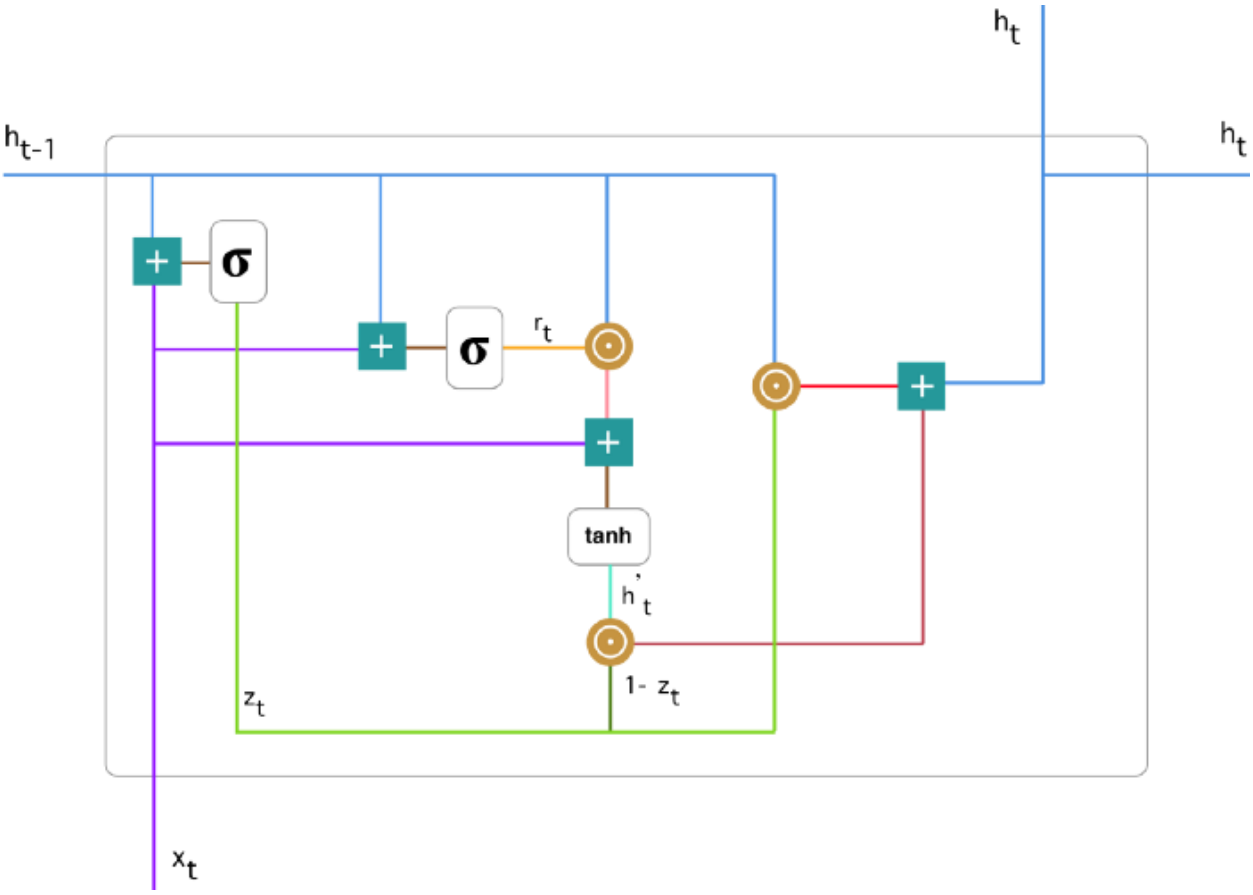


Рисунок 8. Структура сети GRU

Кратко остановимся на вопросе предварительной очистки текстовых данных для работы с ними рекуррентными нейронными сетями. Если использовать русский алфавит, то необходимо заменить буквы «ё» на «е», «й» на «и», а также убрать предлоги.

Введем обозначения:

X_t – входной вектор на шаге t

h_t – вектор скрытого слоя на шаге t

δ- нейросетевой слой с сигмоидальной функцией активации

tanh – нейросетевой слой с тангенсоидой в качестве функции активации.

Также на схеме есть элементы поэлементного умножения и сложения.

Алгоритм работы сети GRU:

1. Инициализация
2. Определение, что забыть, а что оставить по формуле

$$Z_t = \delta(W_z * [h_{t-1}, x_t] + b_z)$$

Далее Z_t, поэлементно вычитается из 1 и умножается на вектор состояния на предыдущем слое

$$h_t \otimes (1 - Z_t)$$

$$\widehat{h_t} = \tanh(W_{xh} * x_t + W_{hh} * (r_t \otimes h_{t-1}) + b_z)$$

Или в развернутом виде то, что получаем на выходе

$$\widehat{h_t} = h_{t-1} \otimes (1 - Z_t) + \widehat{h_t} \otimes Z_t$$

В принципе работы блока GRU есть некоторое сходство с принципов работы фильтра Калмана.

Исследователи отмечают, что управляемый рекуррентный блок дает очень хорошие результаты при работе с аудио форматами, а также в синтезе речи.

ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Задача 1. Необходимо программно реализовать нейронную сеть Хопфилда с варьируемым числом нейронов и решить с помощью разработанного программного обеспечения прикладную задачу (по выбору студента).

Задача 2. Необходимо программно реализовать нейронную сеть Хемминга с варьируемым числом нейронов и решить с помощью разработанного программного обеспечения прикладную задачу (по выбору студента).

Задача 3. Необходимо программно реализовать нейронную сеть LSTM или GRU с варьируемым числом блоков и решить с помощью разработанного программного обеспечения прикладную задачу (по выбору студента).

Контрольные вопросы:

1. Чем сеть LSTM отличается от сети GRU?

2. Какие задачи можно решать с помощью сети GRU?
3. Какие задачи можно решать с помощью сети LSTM?
4. Чем отличается сеть Хемминга от сети Хопфилда?
5. Можно ли прогнозировать с помощью сети GRU?
6. Какие активационные функции применяются в сети GRU и почему?
7. Что такое «вентиль» в сети GRU? И чем он отличается от «вентиля» в сети LSTM?

Литература:

1. Рутковская Д., Пилиньский М.. Рутковский Л. «Нейронные сети, генетические алгоритмы и нечеткие системы». Пер. с польск., И.Д. Рудинского. М.: Горячая линия – Телеком, 2006. 452 с.
2. Хайкин С. Нейронные сети: полный курс. 2-е изд.: Пер. с. англ. – М.: Издательский дом «Вильямс», 2006 – 1104 с.
3. Заенцев И. В. Нейронные сети: основные модели. Учебное пособие . Воронеж: ВГУ. 1998.- 76с.
4. Шумков Е.А. Система поддержки принятия решений предприятия на основе нейросетевых технологий. Дисс. канд. техн. наук. Краснодар: КубГТУ. 2004.

Пример программного кода, реализующий LSTM
(Тараненко М., КубГТУ, 2021)

```
def load_data(ticker):
    df = yf.download(ticker, START, TODAY, interval="5M")
    df.reset_index(inplace=True)
    df['Datetime'] = df['Datetime'].dt.tz_localize(None)
    return df

df = load_data(selected_stock)

df['Datetime'] = pd.to_datetime(df['Datetime'])
df.set_axis(df['Datetime'], inplace=True)
df.drop(columns=['Open', 'High', 'Low', 'Volume'], inplace=True)
print(df)
import plotly
import plotly.graph_objects as go
print("Plotly Version: ",plotly.__version__)

trace = go.Scatter(
    x = df['Datetime'],
```

```
    y = df['Close'],
    mode = 'lines',
    name = 'Data'
)
layout = go.Layout(
    title = "",
    xaxis = {'title' : "Date"},
    yaxis = {'title' : "Close (Dollars)"}
)
fig = go.Figure(data=[trace], layout=layout)
fig.show()

close_data = df['Close'].values
close_data = close_data.reshape((-1,1))

split_percent = 0.80
split = int(split_percent*len(close_data))

close_train = close_data[:split]
close_test = close_data[split:]

date_train = df['Datetime'][:split]
date_test = df['Datetime'][split:]

print(len(close_train))
print(len(close_test))

look_back = 1

train_generator = TimeseriesGenerator(close_train, close_train, length=look_back,
batch_size=5)
test_generator = TimeseriesGenerator(close_test, close_test, length=look_back,
batch_size=1)

model = Sequential()
model.add(
    LSTM(10,
        activation='relu',
        input_shape=(look_back,1))
)
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

num_epochs = 25
```

```
model.fit_generator(train_generator, epochs=num_epochs, verbose=1)

prediction = model.predict_generator(test_generator)

close_train = close_train.reshape((-1))
close_test = close_test.reshape((-1))
prediction = prediction.reshape((-1))

trace1 = go.Scatter(
    x = date_train,
    y = close_train,
    mode = 'lines',
    name = 'Data'
)
trace2 = go.Scatter(
    x = date_test,
    y = prediction,
    mode = 'lines',
    name = 'Prediction'
)
trace3 = go.Scatter(
    x = date_test,
    y = close_test,
    mode='lines',
    name = 'Ground Truth'
)
layout = go.Layout(
    title = "Google Stock",
    xaxis = {'title' : "Date"},
    yaxis = {'title' : "Close"}
)
fig = go.Figure(data=[trace1, trace2, trace3], layout=layout)
fig.show()

close_data = close_data.reshape((-1))

def predict(num_prediction, model):
    prediction_list = close_data[-look_back:]

    for _ in range(num_prediction):
        x = prediction_list[-look_back:]
        x = x.reshape((1, look_back, 1))
        out = model.predict(x)[0][0]
        prediction_list = np.append(prediction_list, out)
```

```
    prediction_list = prediction_list[look_back - 1:]
    return prediction_list
def predict_dates(num_prediction):
    last_date = df['Datetime'].values[-1]
    prediction_dates = pd.date_range(last_date, periods=num_prediction + 1).tolist()
    return prediction_dates

num_prediction = 1
forecast = predict(num_prediction, model)
forecast_dates = predict_dates(num_prediction)

trace1 = go.Scatter(
    x = df['Datetime'].tolist(),
    y = close_data,
    mode = 'lines',
    name = 'Data'
)
trace2 = go.Scatter(
    x = forecast_dates,
    y = forecast,
    mode = 'lines',
    name = 'Prediction'
)
layout = go.Layout(
    title = "Google Stock",
    xaxis = {'title' : "Date"},
    yaxis = {'title' : "Close"}
)

fig = go.Figure(data=[trace1, trace2], layout=layout)
fig.show()
```

```
# Get sizes of each of the datasets
num_cv = int(cv_size*len(df))
num_test = int(test_size*len(df))
num_train = len(df) - num_cv - num_test

# Split into train, cv, and test
train = df[:num_train]
cv = df[num_train:num_train+num_cv]
train_cv = df[:num_train+num_cv]
test = df[num_train+num_cv:]

RMSE = []
```

```
mape = []
for N in range(1, Nmax+1): # N is no. of samples to use to predict the next value
    est_list = get_preds_mov_avg(train_cv, 'adj_close', N, 0, num_train)

    cv.loc[:, 'est' + '_N' + str(N)] = est_list
    RMSE.append(math.sqrt(mean_squared_error(est_list, cv['adj_close'])))
    mape.append(get_mape(cv['adj_close'], est_list))
# Set optimum N
N_opt = 2
est_list = get_preds_mov_avg(df, 'adj_close', N_opt, 0, num_train+num_cv)
test.loc[:, 'est' + '_N' + str(N_opt)] = est_list

global currentGraph
# Clear all graphs drawn in figure
plt.clf()
# Plot adjusted close over time
rcParams['figure.figsize'] = 10, 8 # width 10, height 8

ax = train.plot(x='date', y='adj_close', style='bx-', grid=True)
ax = cv.plot(x='date', y='adj_close', style='yx-', grid=True, ax=ax)
ax = test.plot(x='date', y='adj_close', style='gx-', grid=True, ax=ax)
ax = test.plot(x='date', y='est_N2', style='rx-', grid=True, ax=ax)
ax.legend(['train', 'validation', 'test', 'predictions with N_opt=2'])
ax.set_xlabel("date")
ax.set_ylabel("USD")
ax.set_xlim([date(2020, 10, 30), date(2021, 1, 1)])
ax.set_ylim([180, 200])
ax.set_title('Zoom in to test set')
```

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №4. Нейронная сеть Элмана

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ:

31. Внимательно ознакомиться с заданием на практическую работу;
32. Изучить теоретический материал по а) лекции, б) непосредственно к данной работе.
33. Разработать план реализации задания
34. Подготовить входную выборку (если необходимо)
35. Разработать математическую модель алгоритма решения поставленной задачи
36. Программно на любом доступном языке программирования реализовать поставленную задачу
37. Провести эксперименты (если необходимо)
38. Написать отчет о проделанной работе
39. Защитить практическую работу преподавателю
40. Сдать отчет по работе преподавателю (в установленном формате)

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Выполненная практическая работа должна сопровождаться отчетом. Отчет должен содержать не менее 8 страниц текста в формате MS Word (*.doc или *.docx) или LibreOficce (*.odt). Форматирование страницы – все поля по 2 см, межстрочный интервал - полуторный, шрифт Times New Roman 14, нумерация внизу страницы справа. В отчете должны присутствовать: архитектура нейронной сети, результаты экспериментов (таблицы, графики), если разработан уникальный алгоритм построения модели, то необходима блок – схема работы алгоритма. Таблицы и рисунки обозначаются в стандартном варианте.

Весь исходный код разработанного программного обеспечения приводить не надо.

Отчет сдается только в электронном виде (распечатывать не надо).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Использование рекуррентных нейронных сети (RNN) как сетей для идентификации систем и контроллеров в обратных связях систем нейроуправления дают ряд потенциальных преимуществ в сравнении с классическими многослойными сетями типа многослойный персептрон. Рекуррентные сети предоставляют возможности для кодирования и представление внутреннего или скрытого состояния. Также, рекуррентные сети могут давать прогнозы с большим горизонтом прогнозирования (даже при наличии шумов измерений за счет их фильтрации в своих блоках с задержкой).

Нейронная сеть Элмана (иногда пишут *Элмена*) относится к рекуррентным сетям (иногда их относят к частично – рекуррентным сетям [**Ошибка! Источник ссылки не найден.**]). Была предложена в 1990 году.

Элман предложил частично рекуррентную нейронную сеть, где прямые связи модифицируются, а повторяющиеся соединения фиксируются. Она имеют набор контекстных узлов для хранения внутренних состояний. Таким образом, сеть Элмана имеет уникальные возможности по хранению и использованию предыдущих значений.

Структура сети Элмана следующая – в дополнение к стандартным скрытому и выходному слоям нейронов добавлен специальный слой обратной связи (т.н. *контекстный слой* или *слой состояний*). Данный слой получает сигналы со скрытого слоя и через элементы задержки z^{-1} подает их на входной слой, тем самым сохраняя обрабатываемую информацию в течении одного временного такта (т.н. *эпизодная подача данных* или *limited memory effect*).

Элементы сети Элмана – стандартные нейроны, адаптивные линейные ассоциаторы и элементы задержки z^{-1}.

Решающим (или строительным) блоком сети Элмана являются классические нейроны с типом функции активации – гиперболический тангенс.

Выходной слой сети Элмана может быть нелинейным.

Алгоритм работы сети Элмана можно описать с помощью следующих уравнений:

$$u_j(k+1) = \sum_{j=1}^{n} w_{ji}^{[1]} x_i(k) + \sum_{i=1}^{n1} w_{ji}^{C} o_i(k) + \theta_j^{[1]}$$

$$o_j(k+1) = \phi(u_j(k+1)) = \tanh(u_j(k+1)), \_\_\_\_ j = 1,2,3...n1$$

$$y_j(k+1) = \sum_{i=1}^{n1} w_{ji}^{[2]} o_i(k+1) + \theta_j^{[2]}, \ j = 1,2,3...m$$

Так же можно описать в матричной форме. Стуктурная схема показана на рисунке ниже (взято из [1]):

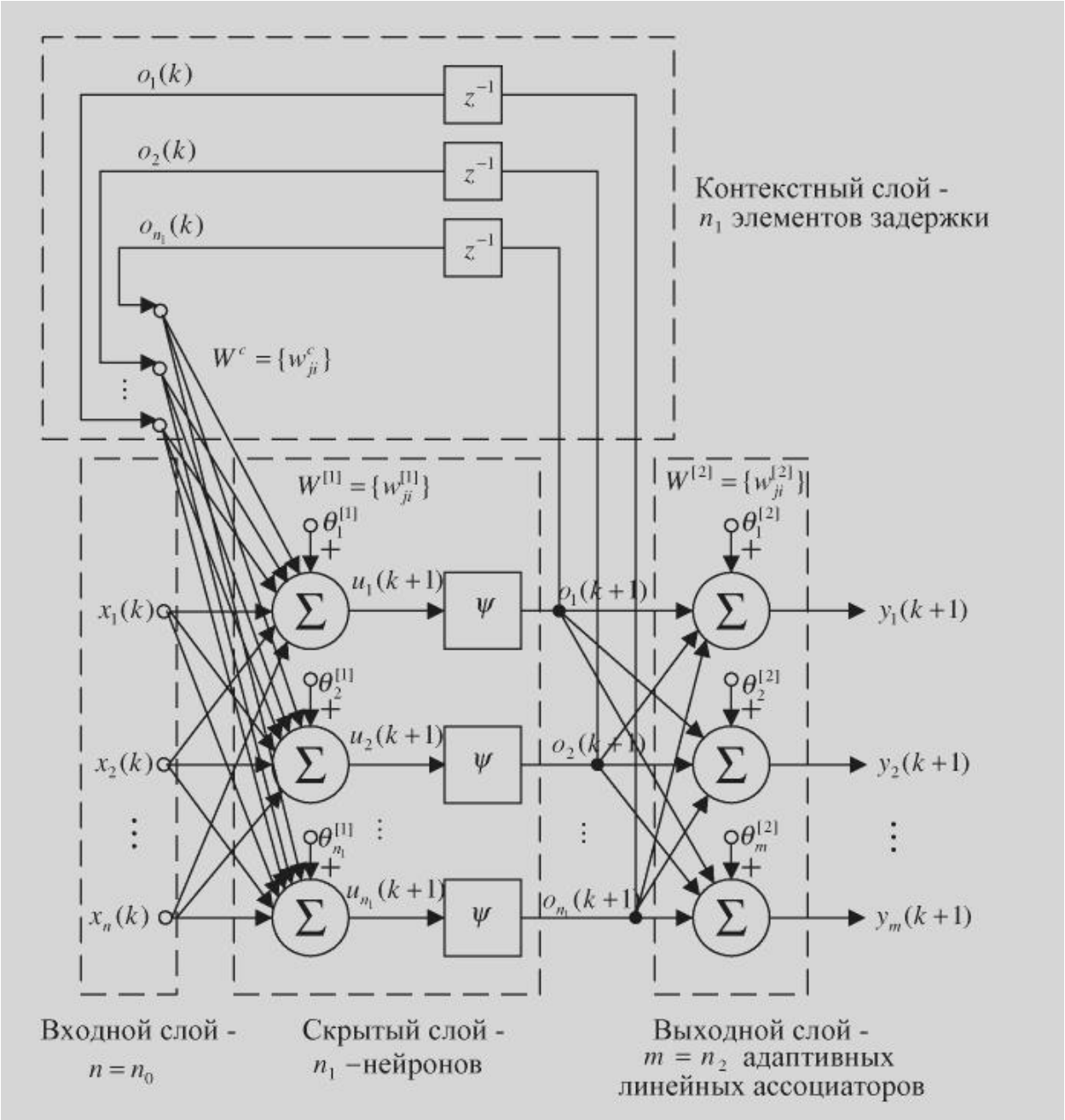


Рисунок 9. Нейронная сеть Элмана (схема - Круглов)

Алгоритм обучения сети Элмана основан на градиентном методе наискорейшего спуска, основанного на работах Р.Вильямса и Д.Зипсера. По сути - это тот же самый алгоритм обратного распространения ошибки.

Существуют следующие возможные варианты работы сети Элмана[6]:

- "много в один" (many-to-one) рис.2а - скрытый слой последовательно изменяет своё состояние, из его конечного состояния вычисляется выход сети, эту схему можно использовать для классификации текстов ;

[6] Источник: http://mechanoid.kiev.ua/neural-net-rnn.html

- "один во много" (one-to-many) рис.2b - скрытый слой инициализируется одним входом, из цепочки его последующих состояний генерируются выходы сети, эту схему можно использовать для аннотирования изображений ;
- "много во много" (many-to-many) рис.2c - на каждый вход сеть выдаёт выход, который зависит от предыдущих входов, эту схему можно использовать для классификации видео ;
- "много во много" (many-to-many) рис.2d - скрытый слой последовательно изменяет своё состояние, его конечное состояние служит инициализацией для выдачи цепочки результатов, эту схему можно использовать для создания систем машинного перевода и чат-ботов.

Первоначально сеть Элмана разрабатывалась для детектирования границ между словами в непрерывном потоке текста.

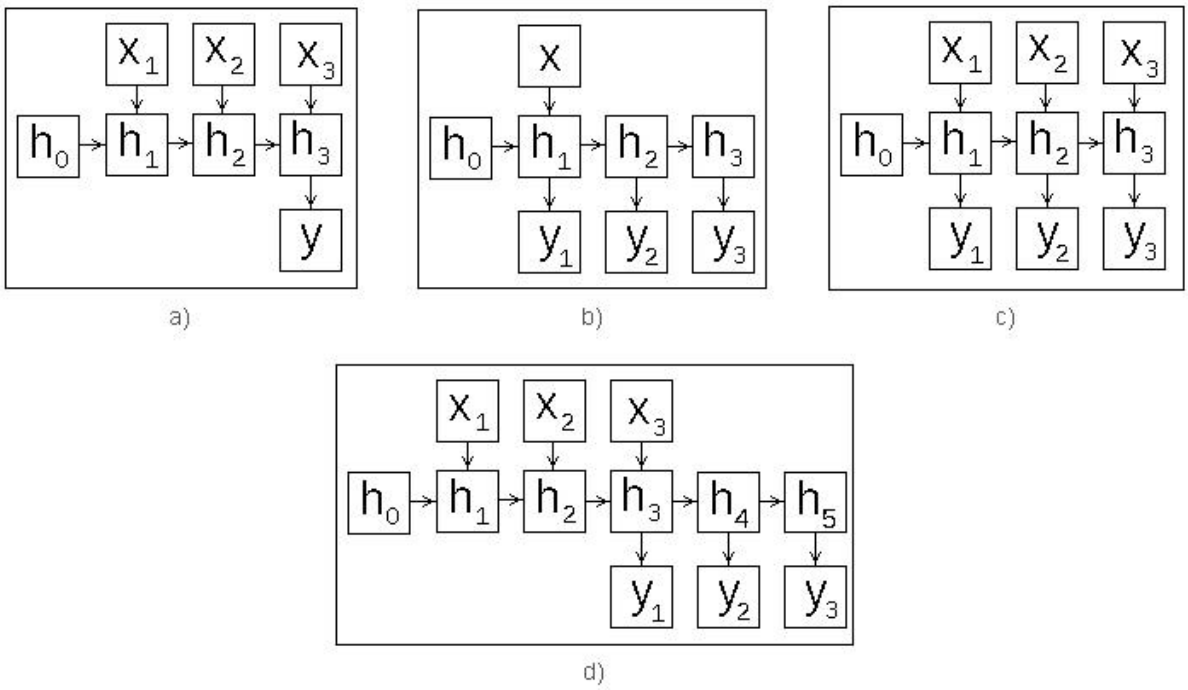


Рисунок 10. Варианты работы сети Элмана

Алгоритм обучения сети Элмана следующий:

Для обучения сети Элмана применяются те же градиентные методы, что и для обычных сетей прямого распространения, но с определёнными модификациями для корректного вычисления градиента функции ошибки. Он вычисляется с помощью модифицированного метода обратного распространения, который носит название Backpropagation through time (метод обратного распространения с разворачиванием сети во времени, сокр. BPTT). Идея метода - развернуть последовательность, превратив рекуррентную сеть в "обычную" (рис.2a). Как и в методе обратного распространения для сетей прямого распространения, процесс вычисления градиента (изменения весов) происходит в три следующих этапа.

1) Прямой проход – вычисляются состояния слоев;

2) Обратный проход – вычисляются ошибки слоев

3) Вычисление изменения весов, на основании данных полученных на первом и втором этапах.

Сеть Элмана получила распространение в системах управления движущимися объектами (т.к. здесь важна именно динамика изменения), при построении систем технического зрения, задач прогнозирования и т.д.

На базе сети Элмана строится нейронная сеть RAAM (рекурсивная авто – ассоциативная память), по сути «двойная сеть Элмана», которая используется для сжатия и шифрования информации.

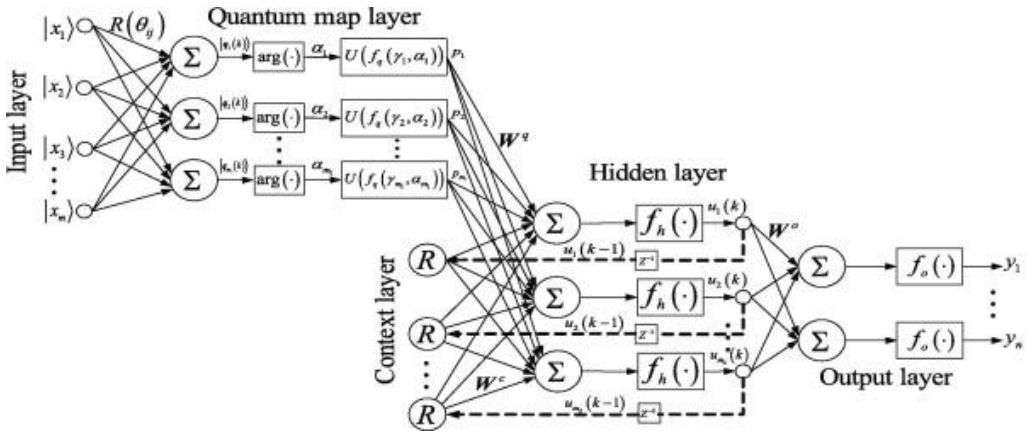


Рисунок 11. Вариант представления нейронной сети Элмана (https://www.sciencedirect.com)

Существует также PCA – Elman алгоритм обучения данной сети, где процесс обучения осуществляется вместе с (некоторой) оптимизацией структуры. Выделяют также PLS алгоритм для сети Элмана предварительного анализа входных данных и уменьшения размерности.

На странице https://www.hindawi.com/journals/cin/2014/724317/fig2/ показан пример каскада сетей Элмана.

Сеть Джордана

Аналогом сети Элмана является сеть Джордана (служит также для распознавания временных последовательных образов), также рекуррентная. Также как сеть Элмана она имеет скрытый, выходной, входной и контекстный слой, со следующим принципиальным отличием – сигнал на контекстный слой поступает с выходов сети, при этом в элемент задержки внесен фактор (или коэффициент) забывания $0 \leq \alpha \leq 1$. И соответственно, сигнал на выходе контекстных нейронов определяется не только текущими выходами сети, но и их экспоненциально взвешенными предыдущими значениями. Считается, что введение фактора забывания обеспечивает данной сети более «глубокую»

память, чем у сети Элмана, а также обладает повышенными свойствами аппроксимации и прогнозирования.

Сеть Джоржана работает по рекуррентным соотношениям [1]:

$$\begin{cases} u_j(k+1) = \sum_{i=1}^{n} w_{ji}^{[1]} x_i(k) + \sum_{i=1}^{m} w_{ji}^{c} (y_i(k) + \alpha y_i(k-1)) + \theta_j^{[1]}, \\ o_j(k+1) = \psi^{[1]}(u_j(k+1)), \quad j = 1,2,\ldots,n_1, \end{cases}$$

$$y_j(k+1) = \psi^{[2]}\left(\sum_{i=1}^{n_1} w_{ji}^{[2]} o_i(k+1) + \theta_j^{[2]} \right), \quad j = 1,2,\ldots,m$$

Которые также можно записать и в матричной форме.

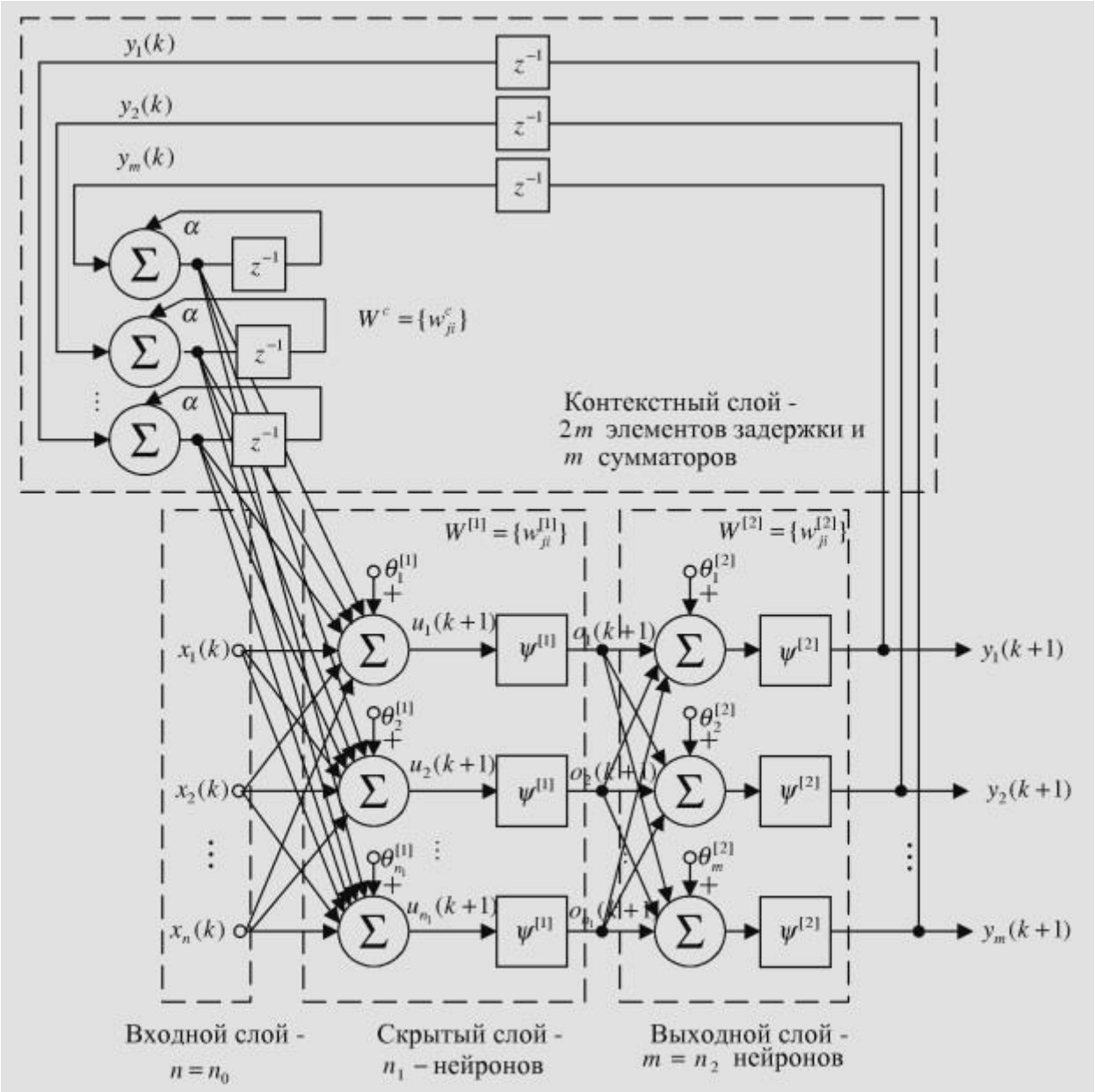


Рисунок 12. Схема нейронной сети Джордана (Бодянский Е.В., Руденко О.Г.)

Сеть Джордана решает тот же круг задач, что и сеть Элмана, но есть некоторые преимущества за счет некоторой модернизации в архитектуре сети Джордана.

Отдельно отметим довольно широкий класс рекуррентных нейронных сетей временными задержками, которые идут не через обратные связи, а находятся в линиях прямой передачи данных. Это т.н. сети TDNN – Time Delay Neural Networks/

Пример практического применения нейронных сетей

Задача прогнозирования складских запасов [2].

Одним из немногих способов, способных учесть большое количество факторов и историю продаж за прошлые годы, является использование искусственной нейронной сети. Задача прогнозирования объемов продаж и складских запасов с помощью нейронных сетей обычно описывается без единого, обобщенного подхода. Ниже будут даны базовые, обобщенные рекомендации по прогнозированию объема продаж с помощью нейронной сети.

На рисунке ниже показана нейронная сеть с входами по истории продаж и дополнительными параметрами.

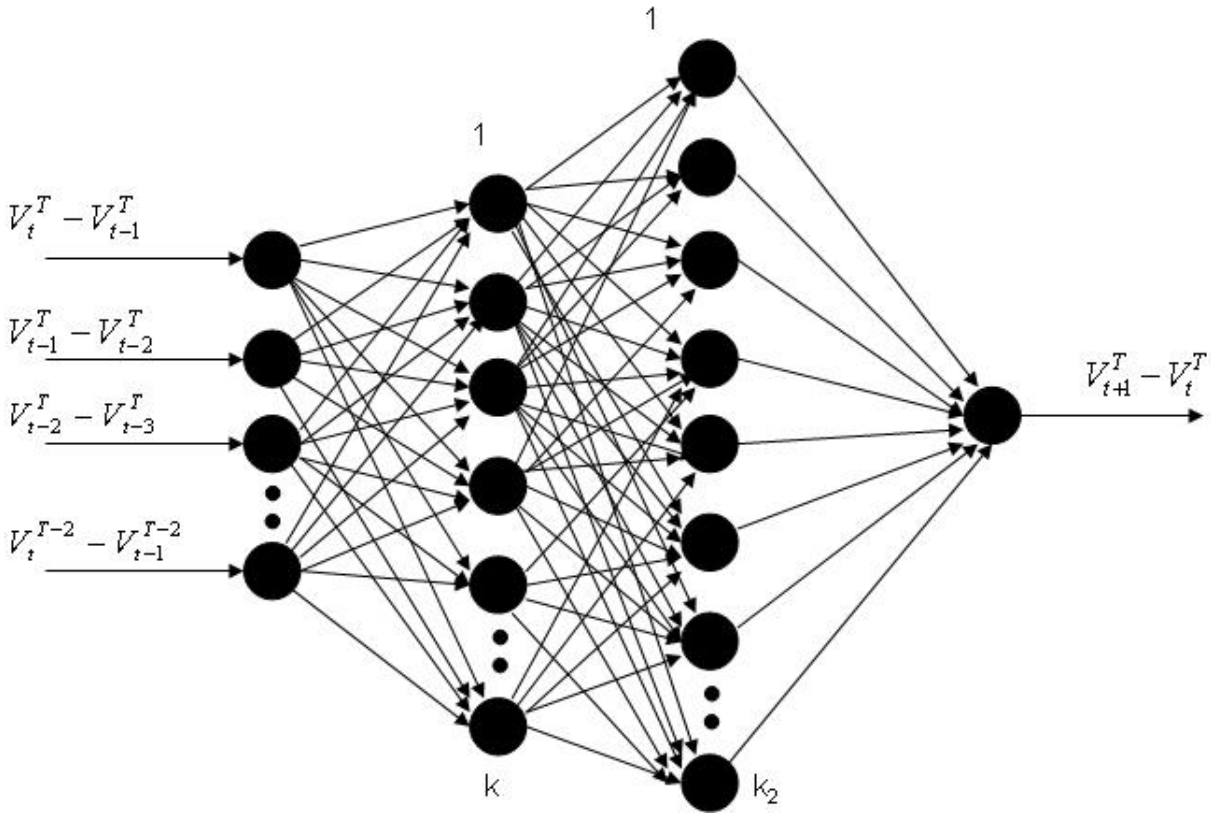


Рисунок 13. Прогнозирующая нейронная сеть

В качестве входных данных логично подавать «скользящее окно» за последний месяц – данные по четырем неделям (если горизонт прогнозирования - неделя), а также данные по продажам по данной неделе прошлых годов (обычно за 1 – 3 года). «Скользящее окно» см. Таблица 1. Отметим, что подается не абсолютное значение продаж, а относительное. Также прогнозируется количество товара[7], а не сумма продаж. Выходом является значение $V_{t+1}^T - V_t^T$, где t - временная итерация (день или неделя), а T - год. В ряде случаев достаточно, чтобы *хорошо* прогнозировался знак изменения объема продаж. В некоторых случаях важен не столько точный прогноз конкретного товара, сколько прогнозирование и планирование ассортиментных поставок продукции

[7] Часто удобнее брать укрупненные единицы измерения количества, например, поддон, коробка, цистерна и т.д.

Кроме вышеперечисленных этапов прогнозирования продаж, в случае нейросетевого подхода, необходимо также выполнить дополнительные шаги: определение размера и структуры выборки, нормирование входных и выходных переменных, подбор архитектуры нейронной сети, ее структуры и выбор метода обучения согласно архитектуре. И собственно обучить нейронную сеть, проверить ее на валидационной выборке и в итоге использовать в реальных условиях.

В качестве архитектуры нейронной сети целесообразно применять многослойный персептрон, как самый универсальный, а в качестве метода обучения – алгоритм обратного распространения ошибки или алгоритм RProp, если необходимо прогнозировать только знак изменения продаж.

Главный недостаток нейронной сети – нужен специалист, разбирающийся в них. В тоже время обученная сеть может использоваться достаточно долго без перенастройки. В случае, если переобучение необходимо, можно задействовать принцип обучения с подкреплением, чтобы минимизировать участие эксперта в настройке нейронной сети. Особенно адаптивная система прогнозирования необходима, если компания оперируют большим количеством номенклатуры продаваемых товаров.

В задаче определения складских запасов необходимо определять значение запасов по большому количеству номенклатуры. Отметим следующий широко распространенный подход – большая часть номенклатуры имеет небольшой оборот и, как отмечено выше, наибольший оборот идет по 20-25% от всего ассортимента товаров. Поэтому целесообразно прогнозировать только эти 20-25%, а остальные, если руководствоваться терминами ABC – анализа, категорию C – либо не прогнозировать, либо использовать метод по последней скорости продаж, а категорию B – прогнозировать по группам товаров. Очень часто используется XYZ – анализ, как фильтр прогнозировать объемы продаж конкретного товар или нет.

Также необходимо отдельно остановиться на горизонте прогнозирования. Как известно – тактически прогнозируется с коротким или среднесрочным горизонтом, неделя – месяц, но стратегически менеджмент компании интересует долгосрочный прогноз – на квартал и более. Но, чем дальше горизонт прогнозирования, тем выше ошибка прогноза. Здесь возможны несколько вариантов применения нейронных сетей, но они требуют дополнительной проверки. Вариант А): использование для прогнозирования нейронных сетей с несколькими выходами, при этом каждый выход отвечает за номер временной итерации, например, 1-выход за $t+1$ неделю, 2-й за $t+2$ неделю и т.д. Вариант Б): используется нейронная сеть с одним выходом и прогнозное значение на $t+1$ итерацию подается на входы нейронной сети и получается значение на $t+2$ итерацию и т.д. Отметим, что для долгосрочного прогноза априори требуется обучающая выборка большего размера и ширина скользящего окна также увеличивается.

Здесь отметим, что данное прогнозирование складских запасов можно производить с помощью сети Элмана (или Джордана).

Рассмотрим другой пример – прогнозирование потока постояльцев в гостинице, согласно [3]. Главным параметром в работе гостиничного бизнеса без сомнения является количество постояльцев гостиницы. Чтобы быть готовым к увеличению количества постояльцев, необходимо заранее сделать запасы, например, продуктов питания в кафе / ресторан, нанять дополнительный персонал и т.д. В то же время при возможном будущем снижении количества постояльцев необходимо принять превентивные меры, так как от количества постояльцев напрямую зависит доход. Таким образом, необходим прогноз количества постояльцев гостиницы. Существует большое количество методов прогнозирования для различных классов задач. В данном случае, на наш взгляд, наиболее применимыми являются искусственные нейронные сети. Количество постояльцев зависит от большого числа, как измеримых, так и трудно измеримых факторов. К измеримым факторам, которые после масштабирования можно подавать на входы нейронной сети, можно отнести: количество посетителей за предыдущие временные интервалы, количество забронированных мест на данный интервал год назад и т. д.. Но допустим, как учесть действия конкурентов? Как спрогнозировать отдачу при проведении в городе крупного спортивного соревнования или крупной конференции? Как учесть отдачу от веб – сайта гостиницы при условии постоянного колебания в рейтинге Яндекса и Google? При этом данные вопросы зачастую имеют «подводные камни», допустим не секрет, что устроители крупных мероприятий заселяют своих участников в «свои» гостиницы, в среде Интернет возможен «черный пиар» конкурентов и т.д. Для данных факторов необходимо формировать шкалы измерений, по которым эксперты будут проставлять баллы тому или иному трудноизмеримому фактору.

Для решения задачи прогнозирования количества постояльцев гостиницы, по результатам экспериментов авторов вполне достаточно многослойного персептрона (1 скрытый слой, количество нейронов в котором зависит от количества входов и размера исторической выборки) и обучения по методу обратного распространения ошибки. Наилучшие результаты прогнозирования достигались, если брался операционный интервал равный одной неделе. Основными входными данными являются количество постояльцев гостиницы за предыдущие периоды, при этом данный временной ряд необходимо подавать по принципу «скользящего окна». Выход многослойного персептрона для данной задачи один – количество постояльцев на следующий временной интервал. Отметим, что при подготовке выборки необходимо также учитывать сезонный фактор и праздничные дни.

ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

1. Необходимо спроектировать и программно реализовать нейронную сеть Элмана или Джордана (нечетная последняя цифра в номере зачетки студента – сеть Элмана, четная – сеть Джордана).

2. Решить с помощью разработанной нейросети Элмана (или Джордана) одну из следующих задач: прогнозирование временных рядов (финансовые, медицинские, сейсмологические и др.); прогнозирование поведения динамических объектов (например: ВУЗ, предприятие, показатели муниципалитета и др.)

3. Сравнить результат работы сети Элмана (или Джоржана) с результатом работы стандартного многослойного персептрона для выбранной задачи (Лабораторная работа №1).

4. Подготовить отчет о выполнении практического занятия и защитить его преподавателю.

Контрольные вопросы:

1. Чем отличается принцип работы сети Элмана от сети Джордана?
2. Как решаются прикладные задачи с помощью нейронных сетей?
3. В чем отличие алгоритма обучения сети Элмана от алгоритма обратного распространения ошибки?
4. Какие функции активации используются в сети Элмана?
5. Зачем нужны элементы задержки в сетях Элмана и Джордана?
6. Как осуществить прогноз финансовых временных рядов с помощью сети Элмана?

Литература:

1. Бодянский Е.В. Руденко О.Г. Искусственные нейронные сети: архитектуры, обучение, применения. Харьков: ТЕЛЕТЕХ. 2004. 369 с.
2. Шумков Е.А. Задача прогнозирования складских запасов // Научные труды КубГТУ. 2016, №7.
3. Ключко В.И., Шумков Е.А., Карнизьян Р.О. Прогнозирование потока постояльцев гостиницы с помощью ИНС. //Материалы II Межвузовской научно-практической конференции "Автоматизированные информационные и электроэнергетические системы", Краснодар, КубГТУ, 2012. с. 66-67

Пример исходного кода сети Элмана на ЯП Python

```
 # Copyright (C) 2011 Nicolas P. Rougier[8]
def sigmoid(x):
    ''' Sigmoid like function using tanh '''
    return np.tanh(x)


def dsigmoid(x):
    ''' производная сигмоиды '''
    return 1.0-x**2


class Elman:
    ''' Elman network '''


    def __init__(self, *args):
        ''' Инициализация персептрона заданного размера.  '''

        self.shape = args
        n = len(args)

        # Строим слои
        self.layers = []

        # Input layer (+1 unit for bias
        #             +size of first hidden layer)
        self.layers.append(np.ones(self.shape[0]+1+self.shape[1]))

        # Hidden layer(s) + output layer
        for i in range(1,n):
            self.layers.append(np.ones(self.shape[i]))

        # Build weights matrix
        self.weights = []
        for i in range(n-1):
            self.weights.append(np.zeros((self.layers[i].size,
                                          self.layers[i+1].size)))

        # dw задержка
        self.dw = [0,]*len(self.weights)

        # Сбрасываем веса
        self.reset()
```

[8] https://github.com/rougier/neural-networks/blob/master/elman.py

```
def reset(self):
    ''' Reset weights '''

    for i in range(len(self.weights)):
        Z = np.random.random((self.layers[i].size,self.layers[i+1].size))
        self.weights[i][...] = (2*Z-1)*0.25


def propagate_forward(self, data):
    ''' Propagate data from input layer to output layer. '''

    # Set input layer with data
    self.layers[0][:self.shape[0]] = data
    # and first hidden layer
    self.layers[0][self.shape[0]:-1] = self.layers[1]

    # Propagate from layer 0 to layer n-1 using sigmoid as activation function
    for i in range(1,len(self.shape)):
        # Propagate activity
        self.layers[i][...] = sigmoid(np.dot(self.layers[i-1],self.weights[i-1]))

    # Return output
    return self.layers[-1]


def backpropogate(self, target, lrate=0.1, momentum=0.1):
    ''' Back propagate error related to target using lrate. '''

    deltas = []

    # Compute error on output layer
    error = target - self.layers[-1]
    delta = error*dsigmoid(self.layers[-1])
    deltas.append(delta)

    # Compute error on hidden layers
    for i in range(len(self.shape)-2,0,-1):
        delta = np.dot(deltas[0],self.weights[i].T)*dsigmoid(self.layers[i])
        deltas.insert(0,delta)

    # Update weights
    for i in range(len(self.weights)):
```

```
            layer = np.atleast_2d(self.layers[i])
            delta = np.atleast_2d(deltas[i])
            dw = np.dot(layer.T,delta)
            self.weights[i] += lrate*dw + momentum*self.dw[i]
            self.dw[i] = dw

        # Return error
        return (error**2).sum()


# ----------------------------------------------------------------------------
if __name__ == '__main__':
    import matplotlib
    import matplotlib.pyplot as plt

    # Example 1: learning a simple time serie
    # ------------------------------------------------------------------------
    network = Elman(4,8,4)
    samples = np.zeros(6, dtype=[('input',  float, 4), ('output', float, 4)])
    samples[0]  = (1,0,0,0), (0,1,0,0)
    samples[1]  = (0,1,0,0), (0,0,1,0)
    samples[2]  = (0,0,1,0), (0,0,0,1)
    samples[3]  = (0,0,0,1), (0,0,1,0)
    samples[4]  = (0,0,1,0), (0,1,0,0)
    samples[5]  = (0,1,0,0), (1,0,0,0)
    for i in range(5000):
        n = i%samples.size
        network.propagate_forward(samples['input'][n])
        network.backpropogate(samples['output'][n])
    for i in range(samples.size):
        o = network.propagate_forward( samples['input'][i] )
        print 'Sample %d: %s -> %s' % (i, samples['input'][i], samples['output'][i])
        print '               Выход сети: %s' % (o == o.max()).astype(float)
        print
```

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №5. Разработка сверточной нейронной сети

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ:

41. Внимательно ознакомиться с заданием на практическую работу;
42. Изучить теоретический материал по а) лекции, б) непосредственно к данной работе.
43. Разработать план реализации задания
44. Подготовить входную выборку (если необходимо)
45. Разработать математическую модель алгоритма решения поставленной задачи
46. Программно на любом доступном языке программирования реализовать поставленную задачу
47. Провести эксперименты (если необходимо)
48. Написать отчет о проделанной работе
49. Защитить практическую работу преподавателю
50. Сдать отчет по работе преподавателю (в установленном формате)

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Выполненная практическая работа должна сопровождаться отчетом. Отчет должен содержать не менее 8 страниц текста в формате MS Word (*.doc или *.docx) или LibreOficce (*.odt). Форматирование страницы – все поля по 2 см, межстрочный интервал - полуторный, шрифт Times New Roman 14, нумерация внизу страницы справа. В отчете должны присутствовать: архитектура нейронной сети, результаты экспериментов (таблицы, графики), если разработан уникальный алгоритм построения модели, то необходима блок – схема работы алгоритма. Таблицы и рисунки обозначаются в стандартном варианте.

Весь исходный код разработанного программного обеспечения приводить не надо.

Отчет сдается только в электронном виде (распечатывать не надо).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Сверточные нейронные сети (англ. - *convolutional neural network, CNN*) – это специальная архитектура нейронных сетей, которая была предложена Яном Лекуном в уже далеком 1988 году. Можно дать следующее определение: «*Сверточная нейронная сеть – тип глубокой нейронной сети, показывающий наилучшие результаты в области распознавания изображений*».

Немного из истории работы Лекуна. Суть заключалась в том, что необходимо было распознать в автоматическом режиме ZIP – коды на конвертах (по всей видимости это была сортировочная станция на одном из

Парижских почтампов). Лекуну это удалось сделать с помощью многослойного персептрона (?) с добавлением нескольких предварительных слоев (о них позже). Хотя эксперимент был удачным, заказчиков, да и научную общественность насторожил тот факт, что сеть обучалась более 3-х суток! И это в 1988 году! (Сейчас сверточные сети обучаются неделями, правда на гигантских массивах информации, но никого это не останавливает). Поэтому работа была положена, что называется «в ящик» до … примерно начала 2000-х годов. Следует отметить, что есть параллели работы Лекуна с топологией неокогнитрона, предложенного еще в конце 70-х годов.

Также кратко представим биографию Я. Лекуна (*Yann LeCun*):

- родился в 1960 году в Париже (француз по национальности);

- докторская степень по информатике в Университете Пьера и Мари Кюри (Париж, Франция);

- работал в «AT&T Bell», сейчас один из руководителей в Facebook (Meta);

- преподавал в Нью-Йоркском университете;

- лауреат Премии Тьюринга в 2018 году совместно с Бенжио и Хинтоном за развитие глубокого обучения;

- кроме всего прочего – один из создателей технологии сжатия DjVu

Преимущества и недостатки сверточных нейронных сетей

1. Один из лучших алгоритмов по распознаванию и классификации изображений.
2. По сравнению с послойно - полносвязной нейронной сетью (типа персептрона) — гораздо меньшее количество настраиваемых весов. Нейросеть обобщает информацию, а не запоминает пиксели каждой показанной картинки в мириадах весовых коэффициентов, как это делает персептрон *(это стандартная формулировка из сети Интернет – на самом деле в СНС и используется многослойный персептрон! Т.е. существует некоторое недопонимание топологии)*.
3. Удобное распараллеливание вычислений, а, следовательно, возможность реализации алгоритмов работы и обучения сети на графических процессорах.
4. Относительная устойчивость к повороту и сдвигу распознаваемого изображения.
5. Обучение при помощи классического метода обратного распространения ошибки.

К недостаткам можно отнести:

1. При работе с реальными данными в практических задачах учитывается большое количество гиперпараметров, т.е. на самом деле существует проблема сложности настройки.
2. При работе с большими данными сеть требует значительных аппаратных ресурсов.

Архитектура сверточной нейронной сети

Сверточная нейронная сеть состоит из следующего типа слоев:

- слой свертки (*convolutional layer*) – главный блок сверточной нейронной сети (также называют *сверточный слой*).

- слой подвыборки (subsampling layer)

- слой активации () – получает на вход скалярный результат каждой свертки. При этом можно считать, что функция (или слой) активации встроена в слой свертки.

- несколько слоев обычного многослойного персептрона.

Сверточные слои представляют собой слои на которых происходит операция свертки (отсюда и название данных слоев) – ядро свертки, представляющее из себя матрицу весов (обычно это матрица 3x3) «скользит» над двумерным изображением, поэлементно выполняя операцию умножения с той частью входных данных, над которой оно сейчас находится, и затем суммирует все полученные значения в один выходной пиксель

Ядро свертки

Ядро свертки есть матрица, обычно 3x3, «пробегает» над двумерным изображением (но лучше сказать – матрицей, ведь не только с изображениями работает сверточная нейронная сеть) и преобразует в двумерную карту признаков. При этом признаки являются взвешенными суммами входных признаков (*расположенных примерно в том же месте, что и выходной пиксель на входном слое*).

То, что было рассмотрено выше называется операцией 2-D свертки и в таком случае изображение имеет один входной канал. Однако на практике большинство изображений имеют минимум 3 выходных канала (по цветам RGB – Red, Green, Blue). И в таком случае ядро свертки будет разным для каждого входного канала, а сумма ядер будет называться фильтром. Таким образом, операция 3-D свертки будет проходить немного иначе[9].

[9] https://neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set/

Для того чтобы создать выходной канал – карту признаков, каждое ядро скользит по своему входному каналу, создавая свою версию этого канала. Затем все эти версии суммируются и создают новый общий канал, к которому добавляется базовое смещение, образуя при этом новую карту признаков (активации).

Каждый слой нейросети находит похожие на находящиеся в нем фильтры участи изображения, что похоже на операцию свертки двух функций. Математически эта операция показана формулой:

$$f[x,y]\cdot g[x,y]=\sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty} f[n_1,n_2]\cdot g[x-n_1,y-n_2]$$

Проведя аналогию со сверткой изображения можно сказать, что эта операция является поэлементным умножением и суммой матрицы изображения $f[x,y]$ и фильтра $g[x,y]$.

Размерность выхода сверточного слоя, где вход имел размерность $w\times h$, а фильтр $k_x\times k_y$, считается по формуле:

$$n\times\left(w-k_x+1\right)\times\left(h-k_y+1\right)$$

где n – количество сверток слоя.

После одного или нескольких слоев свертки обычно идет слой подвыборки, где происходит уменьшение размерности изображения путем сжатия блоков признаков пикселей (обычно размером 2x2) до одного признака с использованием функции максимума. Можно так же использовать и другие функции, например, среднего значения, однако на практике преимущество пуллинга с функцией максимума неоспоримо.

После чередования операций сверток и подвыборки данные преобразуются из двумерного формата в одномерный и передаются в полносвязный слой обычной нейронной сети. Последний слой обеспечивает классификацию изображения.

Сам классификатор представлен слоями обычных нейронов, связанных друг с другом. Выход каждого нейрона рассчитывается стандартно по формуле:

$$OUT=f(w_0+\sum_{n=1} w_n x_n)$$

где w_n – вес синаптической связи нейрона;

x_n – входные сигналы;

w_0 – параметр смещения;

$f()$ – функция активации

Проблема сглаживания краев матрицы изображения

В процессе сверки края матрицы, по сути, обрезаются, т.к. крайние пиксели (элементы матрицы) никогда не оказываются в центре ядра (потому что ядру тогда не над чем будет скользить). Для того чтобы такого не происходило следует добавить к краям поддельные пиксели (обычно нулевого значения) – это операция называется **padding**. Таким образом, ядро при проскальзывании позволяет неподдельным пикселям оказываться в своем центре, а затем распространяется на поддельные пиксели за пределами края, создавая выходную матрицу того же размера, что и входная.

Функции активации

Изначально в сверточных нейронных сетях использовались тангенсоида и сигмоида (см. Лекция 1), но в начале 2000-х годов было предложено использовать функцию ReLU (*rectified linear unit*) и ее модификации Noisy ReLU, Leaky ReLU. Использование данной функции позволяет обойти проблему затухания или увеличения градиентов если используются сигмоида или тангесоида (которые нелийнены).

График функции ReLU показан на Рисунке .

Формула

$$f(s) = \max(0, s)$$

Некоторые преимущества использования ReLU[10]:

а) производная ReLU равна либо нулю, либо единице и поэтому не может произойти разрастание или затухание градиентов;

б) использование данной функции приводит к прореживанию весов;

в) меньше вычислительных затрат в сравнении с вычислением сигмоиды и гиперболического тангенса;

г) отсекает ненужные детали в канале при отрицательном выходе.

[10] https://habr.com/ru/post/348000/

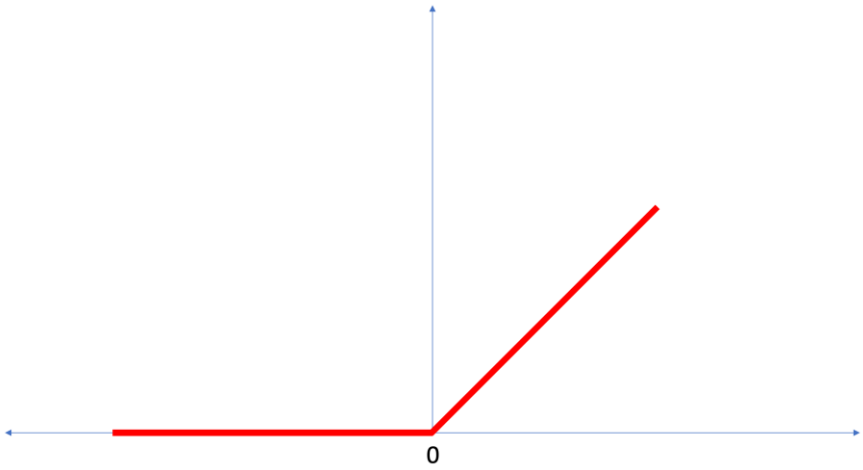


Рисунок 14. Вид функции ReLU

К недостаткам можно отнести следующее – при большом значении сигнала, проходящего через ReLU, нейрон может быть «загнан» в ноль и далее любой сигнал, идущий через данный нейрон всегда равен нулю. Данная проблема обходится за счет выбора надлежащей скорости обучения. Другой вариант - подобрать другие весовые коэффициенты, но основное решение заключается в том, чтобы изменить горизонтальную линию графика на небольшой наклон

$$f(x) = \max(0.01x, x)$$

Эта функция называется Leaky ReLU. Она идентична обычной ReLU, однако, при отрицательных данных градиент не будет равным нулю, при этом она по-прежнему легко вычислима.

Также часто используют функцию Softmax.

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$$

«Softmax – это обобщение логистической функции для многомерного случая. Функция преобразует вектор $\vec{x}$ размерности J в вектор f той же размерности, где каждая координата f_i полученного вектора представлена вещественным числом в интервале от нуля до единицы, и сумма координат равна единице» [**Ошибка! Источник ссылки не найден.**].

Принцип работы сверточной нейронной сети

Суть работы сверточной нейронной сети заключается в переходе от конкретных особенностей изображения к более абстрактным деталям за несколько итераций к выделению понятий более высокого уровня. Считается, что сеть самонастраивается и сама формирует необходимую иерархию абстрактных признаков, отбрасывая при этом незначительные детали и выделяя существенные. Функция Softmax применяется в машинном обучении для задач

классификации, когда количество возможных классов больше двух (для двух классов используется логистическая функция). Координаты f_i полученного вектора при этом трактуются как вероятности того, что объект принадлежит к классу i.

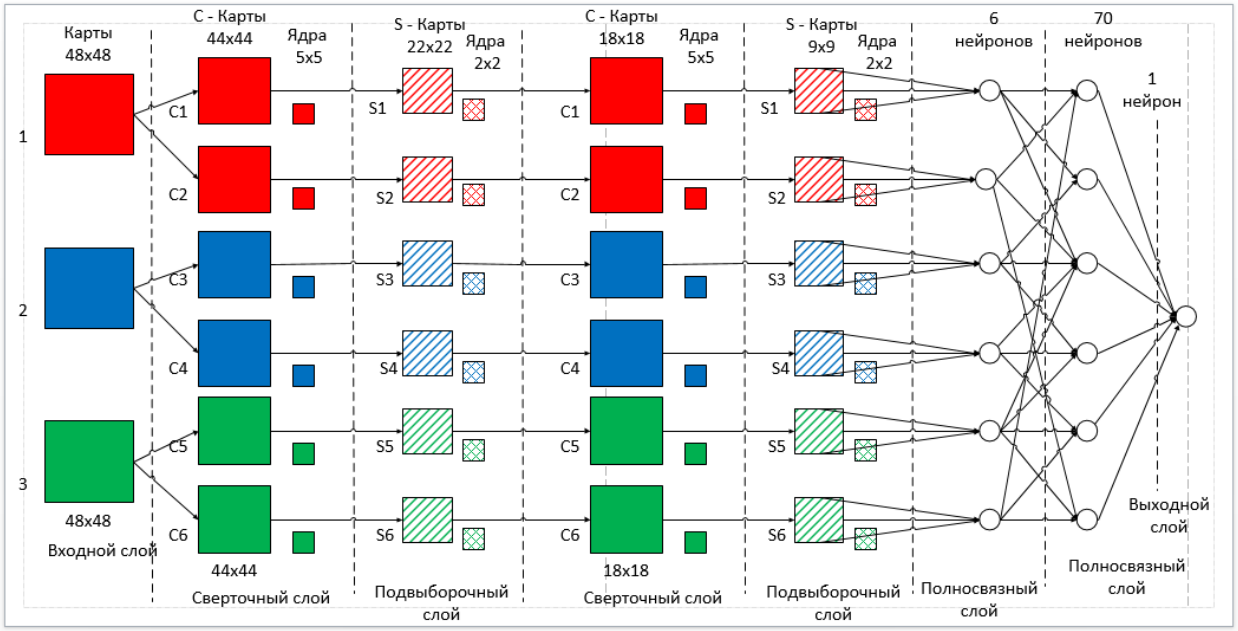


Рисунок 15. Вариант топологии сверточной нейронной сети (habr.com)

Функция потерь

Функция потерь вычисляет эффективность нейронной сети по отношению обучающей выборки к необходимым результатам. Эта функция одномерна. Варианты расчета функции потерь для СНС – можно выделить следующие:

- перекрестная энтропия;
- с.к.о.;
- экспоненциальная;
- расстояние Кульбака – Лейблера и некоторые другие.

Формулы можно посмотреть на слайдах презентации к лекции. Некоторые комментарии:

- основной момент – необходимо рассчитать ошибку между априорными и полученными данными;

- по ряду мнений – перекрестная энтропия является более динамичной и предпочтительна для использования в качестве оценки функции потерь.

- у функции потерь не должно быть каких либо зависимостей от активационных значений кроме выходных[11].

Приведем пример функции потерь по перекрестной энтропии для двух несвязанных переменных p и q:

$$H(p,q) = -\sum_{x} p(x) \log q(x)$$

где q – заданное распределение вероятностей, p – истинное распределение.

Другая трактовка перекрестной энтропии для распознавания изображения:

$$-\sum_{c=1}^{M} (y_c \cdot \log \hat{y}_c)$$

где M - количество классов (т.е. 1000 в ImageNet), а y_C - прогноз модели для этого класса (т.е. результат softmax для класса C). Из-за того, что метки закодированы, а y представляет собой вектор единиц и нулей (1000 × 1), y_C равно либо 1, либо 0.

Еще один пример функции потерь:

$$L = \frac{1}{N} \sum_i L_i\left(f\left(x_i, W\right), y_i\right)$$

где L_i – это функция потерь, x_i – входные данные, W – весовая матрица, y_i – прогноз для класса, L – общие потери всех обучающих данных.

Метрики обучения сверточной нейронной сети

Часто вводятся метрики precision – точность, recall – полнота. Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. Для этого существует параметр f1-score, являющийся F-мерой – средним гармоническим между параметрами precision и recall, имеющий представление, описанное формулой

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

[11] https://neurohive.io/ru/osnovy-data-science/osnovy-nejronnyh-setej-algoritmy-obuchenie-funkcii-aktivacii-i-poteri/

где β – вес точности в метрике. И здесь, конечно, необходимо учитывать количество экземпляров изображений данного класса.

Библиотеки для реализации сверточных нейронных сетей:

Keras

Keras – это библиотека для Python с открытым исходным кодом, которая позволяет легко создавать нейронные сети. Она совместима с Microsoft Cognitive Toolkit, Theano и MXNet. Tensorflow является одной из самых популярных и распространённых платформ на Python для разработки алгоритмов глубокого обучения.

TensorFlow

«TensorFlow – открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации жестов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google.»

Модуль Scikit Learn

Является одним из самых популярных библиотек машинного обучения. Эта библиотека включает в себя различные алгоритмы классификации, регрессии и кластеризации и позволяет взаимодействовать с другими библиотеками численного моделирования.

Также при построении нейронных сетей и построения графики будет полезна библиотека Matplotlib, Pandas (библиотека для работы с массивами), pyplot (библиотека визуализации) и конечно NumPy (Python библиотеки).

Anaconda

Для установки всех необходимых библиотек, которые нам понадобятся при создании сверточных нейронных сетей, необходим менеджер пакетов. Одним из лучших является Anaconda. Anaconda – дистрибутив для языка программирования Python, включающий набор популярных свободных библиотек, объединенных проблематиками науки о данных и машинного обучения. Основная цель – поставка единым согласованным комплектом наиболее востребованных соответствующим кругом пользователей тематических модулей (таких как NumPy, SciPy, Astropy и других – по состоянию на 2020 год содержит более 1,5 тысяч) с разрешением возникающих

зависимостей и конфликтов, которые неизбежны при одиночной установке. А также системой управления изолированными виртуальными средами.

Основная особенность дистрибутива – оригинальный менеджер разрешения зависимостей Conda с графическим интерфейсом Anaconda Navigator, что позволяет отказаться от стандартных менеджеров пакетов (таких, как pip для Python). Дистрибутив скачивается единожды, и вся последующая конфигурация, в том числе установка дополнительных модулей, может проводиться в offline-режиме.

Данный дистрибутив имеет поддержку платформ Linux, Windows и macOS. Распространяется по лицензии BSD, существует также коммерческая версия (Anaconda Enterprise).

Использование данной платформы очень удобно при разработке искусственных нейронных сетей. Проект можно разбивать на блоки, запуская каждый блок с кодом по отдельности. При этом связь между блоками не теряется. Так же в ней при установке уже имеются в комплектации необходимые библиотеки для разработки машинного обучения.

PyCharm

PyCharm – это интеллектуальная среда разработки, которая включает в большой набор средств для эффективной разработки на языке Python. Компания JetBrains выпускает три версии данного программного продукта – бесплатную версию PyCharm Community Edition, образовательную версию PyCharm Educational Editon, а также PyCharm Professional Edition, поддерживающую большой набор возможностей. Для разработки веб-приложения моделирования СУЗ планируется использовать версию PyCharm Community. IDE предоставляет удобную навигацию по коду, а также различные виды рефакторинга. PyCharm выполняет инспекцию и автодополнение во время написания кода. Она основывается на информации, которую получает во время исполнения кода. Особенности PyCharm Community:

- полнофункциональная IDE для разработки на Python, в том числе для многоязычных веб-приложений с фреймворками;
- поддержка фреймворков Django, Flask, Google App Engine, Pyramid, web2py;
- поддержка различных языков программирования, таких как JavaScript, CoffeeScript, TypeScript, Cython и другие;
- поддержка языков разметки HTML и CSS;
- обнаружение кода, который может повторяться;
- удалённая разработка;
- поддержка работы с базами данных.

Реализация сверточной нейронной сети с помощью TensorFlow

Построение нейронной сети требует настройки слоёв, а затем сборки модели с функциями оптимизации и потерь. Базовым элементом при построении нейронной сети является слой. Слой извлекает представление из данных, которые поступили ему на вход.

Пусть сеть состоит из шести слоёв (согласно исходного кода ниже):

1. Входного tf.keras.layers.Flatten – этот слой преобразует изображения размером 224x224 пикселей в 1D-массив. На этом слое у нас нет никаких параметров для обучения, так как этот слой занимается только преобразованием входных данных.

2. Скрытый слой «fc1» – полносвязный слой из 128 нейронов. Каждый нейрон принимает на вход все значения с предыдущего слоя, изменяет входные значения согласно внутренним весам и смещениям во время тренировки и возвращает единственное значение на следующий слой.

3. Под этим пунктом описано обобщение скрытых слоёв «fc2», «fc3», «fc4». Все эти слои полносвязные и выполняют те же функции, что слой «fc1». Они реализованы для того, чтобы иметь большую возможность для обучения. Единственное отличие слоя «fc4» от других слоёв – он содержит 64 нейрона.

3. Между слоями указан параметр Dropout – метод регуляризации искусственных нейронных сетей, предназначен для уменьшения переобучения сети за счет предотвращения сложных коадаптаций отдельных нейронов на тренировочных данных во время обучения.

4. Выходной слой ts.keras.layers.Dense – softmax-слой состоит из пяти нейронов, каждый из которых представляет определенный класс элемента одежды. Как и в предыдущем слое, каждый нейрон принимает на вход значения всех 64 нейронов предыдущего слоя. Веса и смещения каждого нейрона на этом слое изменяются при обучении таким образом, чтобы результирующее значение было в интервале от нуля до единицы, и представляло собой вероятность того, что изображение относится к этому классу. Сумма всех выходных значений пяти нейронов равна одному. Пример инициализации слоев приведён в листинге ниже.

Листинг – Инициализация слоёв

```
base_model = model1  # Топлесс
# Добавление верхнего слоя
x = base_model.output
x = Flatten()(x)
x = Dense(128, activation='relu', name='fc1')(x)
x = Dense(128, activation='relu', name='fc2')(x)
x = Dense(128, activation='relu', name='fc3')(x)
x = Dropout(0.5)(x)
x = Dense(64, activation='relu', name='fc4')(x)
predictions = Dense(5, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
// выполнение настроек
model.compile(optimizer=tf.optimizers.Adam(),
```

```
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
```

После реализации необходимо обучить модель. За выполнение этой операции отвечает функция model.fit. Код данной функции представлен в листинге ниже.

Листинг – Функция запуска обучения нейросети

```
model.fit(X_train_rgb, y_train_rgb, epochs=200, batch_size=16, validation_data=(X_test_rgb, y_test_rgb), verbose=1, callbacks =[early_stopping, model_checkpoint])
```

Для начала нужно выделить последовательность действий при обучении на тренировочном наборе данных:

1. Набор входных данных необходимо повторять бесконечное количество раз, используя метод dataset.repeat(). Параметр epochs определяет количество всех обучающих итераций для выполнения;

2. Метод dataset.shuffle() перемешивает все изображения для того, чтобы на обучение модели не влиял порядок подачи входных данных;

3. Метод dataset.batch() сообщает методу тренировки model.fit использовать блоки изображений и метки при обновлении внутренних переменных модели.

Обучение происходит посредством вызова метода model.fit:

train_dataset отправляется на вход модели;

модель учится сопоставлять входное изображение с меткой;

параметр epochs=200 ограничивает количество тренировок до полных обучающих итераций по набору данных, что в итоге дает тренировку на нескольких примерах.

Во многих высокоуровневых модулях для работы с НС (нейронные сети) существуют функции чтения/записи обученных моделей или только их весов из/в файлы формата HDF5.

ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Общее задание – реализовать программно сверточную нейронную сеть и решить с ее помощью задачу классификации (использовать многослойный персептрон из первого практического занятия).

1. Реализовать операцию свертка

2. Реализовать сверточный и субдискредитирующий слои

3. Состыковать сверточные и субдискретизирующие слои с многослойным персептроном.

4. Обучить разработанную СНС классификации объектов выбранной предметной области.

5. Реализовать многослойный персептрон с использованием библиотеки TensorFlow и провести эксперименты.

В процессе экспериментов, после программной разработки СНС, необходимо оценить следующие параметры для решаемой задачи:

А. скорость ответа (на 3-4 различных ЭВМ, с указанием типа процесса, его тактовой частоты и оперативной памяти: форм-фактор, тактовая частота, тактовая частота шины, пропускная способность, объем);

Б. точность решения задачи (например, точность распознавания изображения), в зависимости от размера входных карт.

Контрольные вопросы к практическому занятию:

1. Принцип работы слоя пуллинга.
2. Чем сверточная нейронная сеть отличается от стандартного многослойного персопетрона?
3. Какие задачи можно решать с помощью сверточной нейронной сети?
4. Можно ли прогнозировать с помощью сверточной нейронной сети? Если да, то каким образом?
5. Как происходит свертка входного изображения для сверточной нейронной сети?
6. Сколько слоев необходимо для свертки изображения?
7. Какие подходы Вы знаете для ускорения процесса обучения сверточной нейронной сети?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №6. Применение сверточной нейронной сети для задачи распознавания изображений

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ:

51. Внимательно ознакомиться с заданием на практическую работу;
52. Изучить теоретический материал по а) лекции, б) непосредственно к данной работе.
53. Разработать план реализации задания
54. Подготовить входную выборку (если необходимо)

55. Разработать математическую модель алгоритма решения поставленной задачи

56. Программно на любом доступном языке программирования реализовать поставленную задачу

57. Провести эксперименты (если необходимо)

58. Написать отчет о проделанной работе

59. Защитить практическую работу преподавателю

60. Сдать отчет по работе преподавателю (в установленном формате)

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Выполненная практическая работа должна сопровождаться отчетом. Отчет должен содержать не менее 8 страниц текста в формате MS Word (*.doc или *.docx) или LibreOficce (*.odt). Форматирование страницы – все поля по 2 см, межстрочный интервал - полуторный, шрифт Times New Roman 14, нумерация внизу страницы справа. В отчете должны присутствовать: архитектура нейронной сети, результаты экспериментов (таблицы, графики), если разработан уникальный алгоритм построения модели, то необходима блок – схема работы алгоритма. Таблицы и рисунки обозначаются в стандартном варианте.

Весь исходный код разработанного программного обеспечения приводить не надо.

Отчет сдается только в электронном виде (распечатывать не надо).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Общие сведения о графических файлах (изображениях)

Определение – «цифровое изображение – это последовательность кадров (или один кадр) в виде дискретного массива точек (т.н. ‘пикселей’), расположенных в памяти либо устройства ввода, либо непосредственно системы технического зрения (СТЗ)» [3].

По большому счету формат изображений можно разделить на монохромные и цветные. При этом монохромные изображения (как и черно – белые фотографии) не теряют актуальность и часто используются при решении прикладных задач, т.к. 1) они меньше занимают памяти, 2) быстрые обрабатываются, 3) … для большого круга задач можно обойтись монохромными изображениями.

Но, если углубиться, то выделяют 4 типа изображений: монохромные, полутоновые, в естественных цветах и палитровые. Кратко разберем эти 4 типа изображений:

1) Монохромные (другое название - двухградационные) до сих пор часто используются при решении простых прикладных задач, в частности,

если требуется определить только наличие какого – либо объекта в поле зрения. Это самое компактное изображение, один пиксель в нем кодируется одним битом! Лучше сказать – кодируется яркость пикселя. Но нередко используют байт.

2) Полутоновые изображения. Здесь яркость пикселя кодируется одним байтом, т.е. можно закодировать 256 оттенков (2^8 = 256 или от 0 до 255). В ряде источников [3] отмечается, что это самое распространенный тип изображений в промышленных и бытовых задачах. Как пример – система видеонаблюдения, работающая ночью.
3) Цветное изображение в естественных тонах. Здесь цвет каждого пикселя сохраняется в так называем формате RGB – тройки, т.е. требуется 3 байта, с помощью которых можно закодировать примерно 16,8 миллионов различных цветов! Это называется True Color. Этот формат, по сути, наиболее близок к тому, что воспринимают глаза человека – он естественен. Данный тип часто применяется в таких областях, как металлургия, медицина, безопасность и т.д. Но для данного формата уже требуются значительные массивы хранения.
4) Палитровое представление изображения. Можно рассматривать, как упрощенный True Color – он более компактен. Здесь палитра передается с помощью 16 или 256 RGB – троек и цвета здесь определяются косвенно. Цвет пикселя кодируется либо 4, либо 8 битами, при этом используется ссылка на цветовую палитру, а не прямое представление цвета. Это позволяет уменьшить размер изображения, но возникает проблема с подменой цветов – часто встречаются цвета, которых не было в исходном изображении. Как с этим бороться? Поступают просто – преобразовывают в полутоновый формат.

Векторное изображение.

Векторное изображение – это совокупность независимых математических объектов (контуров), при этом каждый из них можно перемещать и масштабировать, что очень удобно (Corel Draw!). В программах, обрабатывающих векторные изображения есть соглашения, как обрабатывать эти математические объекты (+атрибуты: цвет и толщина линий). Данный формат крайне компактен, но не подходит для работы с реальными изображениями (нет, когда мы выделяем, допустим, контур лица и с ним работаем, то это как раз допустимо, а вот, когда обрабатываем пейзаж – там многое теряется в данном представлении, можно сказать, что изображение в векторном формате - нереалистично). Наиболее распространенные векторные форматы в таблице ниже.

Формат	**Разработчик**	**Параметры**	

DXF	AutoDesk		
SGO	Silicon Graphics		
POV	Фирма POV-Team		

Растровое изображение.

Растровое изображение есть набор отдельных пикселей, записанных в единицу объема памяти в виде матрицы (т.н. «битовая карта»). При этом физический размер пикселя напрямую связан с разрешением по полю устройства получения изображения [3], т.е. число пикселей на дюйм (dot per inch – dpi).

Устройство представления разрешения	**Разрешение (обычное)**
Монитор компьютера	100 dpi
Принтер	600 dpi
Фотонаборный аппарат	3500 dpi

Главное достоинство растрового изображения – реалистичность. Вот допустим, туман с помощью векторной графики представить невозможно (… можно, но размер файла гигантский получиться), а в векторном – без труда. Но для реальных изображений (в растровом формате), конечно, требуется значительный объем информации. Поэтому, во всех уважающих себя растровых форматах предусмотрено сжатие изображения. Но, здесь, конечно, момент отметим – сжатие изображения в естественных тонах малоэффективно. Растровые изображения невозможно подвергать реальной трансформации. Наиболее распространенные, на взгляд автора, форматы растровых изображений в таблице ниже.

Формат	**Разработчик**	**Параметры**	
PCX	ZSoft Inc.		
BMP	Microsoft		
TIFF	Aldus Corporation		

GIF	CompuServe Inc.		
TGA	Treuvision Inc.		

Отметим существование комбинированного формата изображения, в котором закодирована и векторная и растровая информация – это так называемые метафайлы, среди которых наибольшее распространение получили форматы CGM (фирма ISO) и WMF (фирма Microsoft).

Структура графического файла

Графический файл – это заголовок и данные.

Структура заголовка:

- Магическое число – стоит вначале файла и указывает формат, допустим, GIF.

- содержание заголовка: формат, тип (монохромный, полутоновое, в естественных цветах);

- использовано ли сжатие;

- позиция видеоданных (если есть).

Далее идет таблица цветов, которое при обработке трансформируется в RGB- тройки. Также в файлах присутствуют метки и теги, которые позволяют графическому редактору или программе, использующей изображение, получить быстрый доступ к тому или иному месту на изображении. Здесь еще есть различие в форматах – в некоторых есть теги и фиксированные поля, а в некоторых есть фиксированные поля и потоки. Подробнее см. [3].

Сжатие изображений обычно происходит с помощью «заслуженных» алгоритмов RLE (патент на изобретение от 1996 года, но корни уходят в 60-е годы прошлого столетия) и LZW (Лемпеля – Зива – Велча, датирован далеким 1984 годом).

Библиотеки для работы с изображениями

PIL

Библиотека обработки изображений PIL или Python Imaging Library. Данная библиотека перестала обновляться с 2009 года, но на его основе была разработана ещё более усовершенствованная библиотека Pillow. Она также предназначена для работы с растровыми изображениями и имеет достаточно большое количество необходимого инструментария.

Pillow

OpenCV

Для анализа, классификации и обработки изображений, а также работы с веб-камерой используется библиотека с открытым исходным кодом OpenCV. В данной библиотеке собрано большое количество алгоритмов компьютерного зрения и численных алгоритмов общего назначения. Изначально разрабатывалась в Центре разработки ПО Intel (нередко отмечают, что российскими программистами). Основные модули библиотеки:

- **cxcore** —ядро содержит базовые структуры данных и алгоритмы;

- **CV** — модуль обработки изображений и компьютерного зрения;

- **Highgui** — модуль для ввода/вывода изображений и видео;

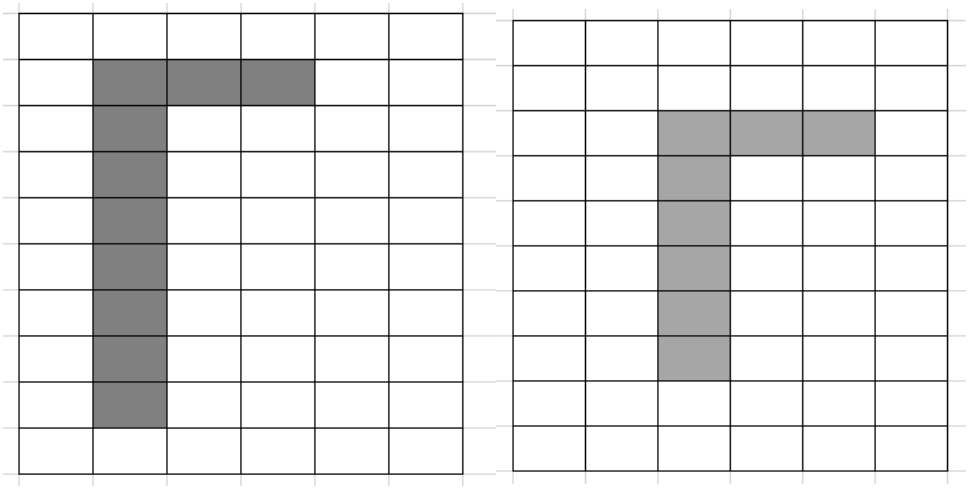
Распознавание изображений с помощью искусственных нейронных сетей.

Рассмотрим классический пример, как использовать нейронную сеть типа многослойный персептрон для распознавания изображений. По большому счету – это подход Ф. Розенблатта конца 50-х годов прошлого столетия.

Пусть у нас имеется фотоматрица, на которой фиксируются буквы русского алфавита (или цифры – не важно) в монохромном формате (см. лекция 4-1). В данном случае матрица 6 х 10. На изображениях ниже представлена буква «Г» - она может быть смещена, увеличена, уменьшена, могут быть некоторые искажения или шум. Человек без труда в большинстве случаев распознает букву (ну если она не совсем затерта, или не «почерк врача»). Да, здесь речь идет о машинописном тексте, об рукописном речь будет идти дальше. А вот распознать букву для технических устройств, точнее сказать, для алгоритмов распознавания изображений – задача посложнее. Почему? Ответ такой – человек (ребенком) тоже не сразу начинает понимать, где какая буква написана! Он долго учится! Очень долго! Если сравнивать с методами автоматического распознавания текста. В принципе, взрослому человеку, допустим россиянину, дать выучить арабский алфавит за пару дней и потом спрашивать, где какая буква, не говоря уже о словах – думаю, это вызовет крайнее затруднение. При этом, арабу дать русский алфавит и потом спрашивать – результат, думаю, также не будет отличаться. Так вот вернемся к фотоматрице. Как ее обработать с помощью нейронной сети? Понятно, что входами на нейронную сеть будут именно эти ячейки фотоматрицы и их (входов) будет:

$$6*10 = 60$$

Т.е. у нейронной сети типа многослойный персептрон будет 60 входов! Много это или мало? Для старых компьютеров, как у Ф. Розенблатта – это конечно много. Даже для «Intel Pentium – II», по опыту и памяти автора, тоже много. А вот для современных «Intel Core Coffee Lake i7» с 6-ю ядрами, тактовой частотой в 4 ГГц и трехуровневым кэшем - это уже очень немного. Далее – вопрос «Сколько выходов необходимо сделать в нейронной сети при распознавании букв русского алфавита?». Ответ очевиден – 33 буквы, значит 33 выхода. Далее, немного поразмыслив – буквы «й», «ъ», «ь», «ё», «ы» можно удалить, т.к. они в реальных, практических задачах практически не используются (например, их нет в автомобильных номерах). При этом – буквы «ё» и «й», если необходимо, можно заменить на «е» и «и», без особых потерь (ну в ряде случаев, может, конечно, потеряться суть слова или предложения, но это редко).



Далее, возникают некоторые трудности, в частности, как закодировать выходы нейросети, чтобы они отвечали каждый за свою букву? Можно, конечно, пойти «напрямую» - первый выход за букву «А», второй за «Б» и т.д., но по опыту автора – это не самый лучший вариант. Т.е. делаем разметку входных образцов:

Входы «образец буквы» - «номер выхода по порядку равен 1, остальные 0», «0,0,0,1,1,0,0,0,0,0,1,0,0,1,…» => »1,0,0,0,0,0….» и тренируем нейронную сеть, чтобы она правильно «зажигала» выходы на подаваемую букву. При этом в выходном слое, чтобы на выходах четко были 0 и 1 необходимо использовать в нейронах выходного слоя линейные функции активации, а это требует некоторой, возможно не совсем корректной, модернизации стандартного алгоритма обратного распространения ошибки!

Можно пойти несколько иначе – использовать в выходах нелинейные функции активации и получать не только 0 и 1, а и промежуточные значения, далее ставить блок интерпретатора (но программно – это просто цикл с перебором), который бы выбирал нейрон с наибольшим значением выхода. Т.е. здесь трактовка в вероятностном смысле.

Можно обойти многие проблемы, если использовать обучение без учителя (допустим по правилу Хебба), т.е. сеть сама будет распределять выходы по входам.

Можно использовать сеть Кохонена, опять же обучающуюся без учителя, но она потребует больше время на обучение и возможно будет работать с низкой точностью.

Если с буквами и цифрами задача, в принципе, не такая сложная, то с распознаванием более трудных изображений, например, лица человека, задача становится более трудной.

Некоторые подзадачи распознавания изображений

Реставрация изображения.

Часто бывает, что на полученных изображениях или видеопотоке заметно какое-либо искажение, из-за которого может теряться качество картинки. Искажения может вызывать зашумление. Искажение зависит от многих параметров и иметь достаточно сложный характер. Зашумления условно можно разделить на естественные и искусственные. К естественным можно отнести такие критерии как:

освещённость;
затуманенность;
природные осадки, дождь, снег;
блики, тени.

К искусственным обычно относят такие критерии как:

характеристики оптических систем (расфокусировка, замутнённость зеркал и линз, дисторсия);
характеристики устройств оцифровки;
характеристики электронной регистрирующей аппаратуры;
характеристики каналов передачи данных.

Пример изображения, искажённого зашумлением показан на рисунке ниже.

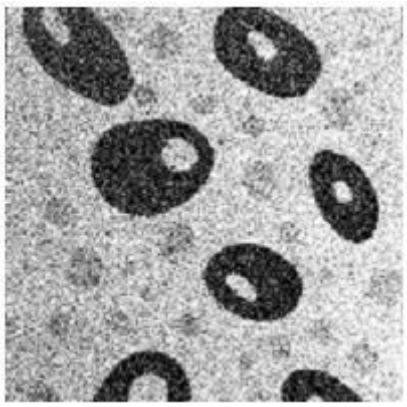


Рисунок 16. Изображение, искажённое зашумлением

При всём этом процесс формирования изображения зависит от многих достаточно сложных нелинейных уравнений, различных математических моделей. К примеру, только параметры яркости имеют зависимость от множества геометрических формул и преобразований. Исходя из этого следует, что к таким изображениям необходимо применять фильтрацию для более качественного распознавания.

Фильтрация изображения – это процедура обработки некоторого растрового изображения для улучшения его качества. Иначе задачу можно назвать фильтрацией помех. Подразумевается, что изначально существовало оригинальное изображение, к которому было применён какой-либо вид искажения, иными словами зашумления.

Существует достаточно большое множество различных фильтров. В зависимости от типа шума применяется какой-либо тип фильтра. Можно выделить самые распространённые из них:

сглаживающие фильтры;
фильтры Винера;
медианные фильтры;
ранжирующие фильтры.

Также существуют линейные и нелинейные фильтры. Так как одним из самых распространённых шумов является белый шум Гаусса, соответственно для шумов такого вида часто применяются линейные фильтры. Среди линейных фильтров можно выделить следующие:

линейная оконная фильтрация;
скользящее среднее в окне;
фильтрация Гаусса.

«Линейные операции состоят из умножения каждого пикселя окрестности на соответствующий коэффициент и суммирование этих произведений для получения результирующего отклика процесса в каждой точке *(x, y)*. Если окрестность имеет размерность $m \times n$, то потребуется $m * n$ коэффициентов. Эти

коэффициенты сгруппированы в виде матрицы, которая называется фильтром, маской, фильтрующей маской, ядром, шаблоном или окном.» [1]

Под процессом линейной пространственной фильтрации понимается смещение центра фильтрующей маски от точки к точке изображения. В каждой точке *(x, y)* откликом фильтра является сумма произведений коэффициентов фильтра и соответствующих пикселей окрестности, которые накрываются фильтрующей маской. Оптимальным размером маски считается размер 3×3.

В ходе исследований было принято решение использовать именно фильтрацию Гаусса, так как она эффективно борется с различного рода белыми шумами, в том числе и зашумлением Гаусса.

Фильтрация Гаусса

Для примера рассмотрим фильтрацию Гаусса. Фильтрация Гаусса – вид электронного фильтра, применяемого для шумоподавления в изображениях или видеопотоках. Данный фильтр, как и многие другие электронные фильтры, имеет импульсную переходную функцию. В данном случае этой функцией является функция Гаусса, описываемая формулой:

$$\mathrm{g}(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

где a, b, c – произвольные вещественные числа. Данная функция введена Гауссом в 1809 году, как функция плотности нормального распределения. В этом случае параметры выражаются через среднеквадратичное отклонение σ и математическое ожидание μ. Выражения вещественных чисел представлены в формуле:

$$a = \frac{1}{\sigma\sqrt{2\pi}}, b = \mu, c = \sigma$$

Ниже представлен график функции Гаусса.

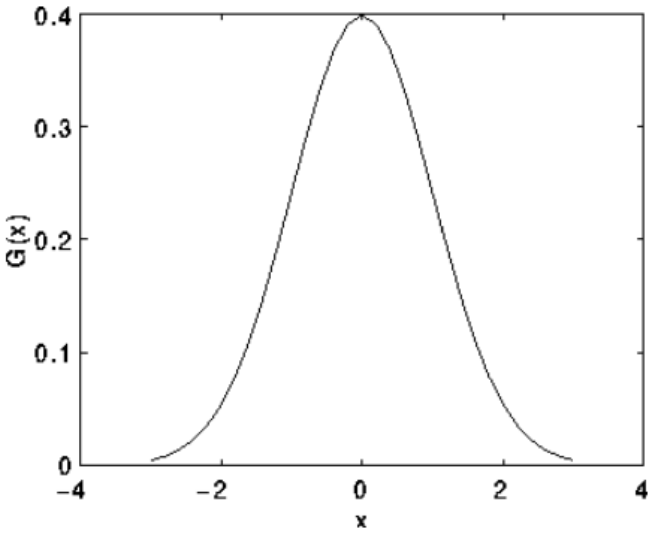


Рисунок 17. График функции Гаусса

Данный фильтр использует Гауссово распределение для вычисления преобразований и применяет его к изображению попиксельно. Как правило, шум распределяется равномерно по всему изображению, при этом не зависит от пикселей. Главное условие – математическое ожидание значения шума должно быть равным нулю. При этом важно учесть, что слишком большое окно фильтрации будет уменьшать интенсивность шума, но побочным эффектом этого фильтра будет являться размытие некоторых деталей на картинке.

Бинаризация изображения

Нередко в прикладных задачах по обработке изображений необходимо провести процесс бинаризации.

«Бинаризация – это один из простых методов препарирования изображения. В общем понимании, после необходимой обработки изображение бесцветным чёрно-белым, у которого убраны лишние детали. Самый важный параметр данного преобразования – это порог. Порогом является критерий проверки интенсивности точки изображения» [2]. Общую схему бинаризации можно посмотреть на слайдах к презентации.

Самыми распространёнными методами бинаризации изображения являются:

глобальный фиксированный порог;
локальный адаптивный порог;
метод Оцу.

«В отличии от методов глобального и адаптивного порогов, значение порога которых задаётся вручную, в методе Оцу значение порога вычисляется автоматически». Благодаря этой особенности и был выбран данный метод.

Метод Оцу использует гистограмму распределения значений яркости пикселей растрового изображения. Математическое описание гистограммы представлено в нижеприведенных формулах.

$$\omega_0(k) = \sum_{i=1}^{k} p_i \,,$$

$$\omega_1(k) = \sum_{i=k+1}^{L} p_i = 1 - \omega_0(k) \,,$$

$$\mu_0(k) = \sum_{i=1}^{k} \frac{ip_i}{\omega_0} \,,$$

$$\mu_1(k) = \sum_{i=k+1}^{L} \frac{ip_i}{\omega_1}$$

Далее строится гистограмма по значениям $p_i = n_i/N$, где N – это общее количество пикселей на изображении, n_i – количество пикселей с уровнем яркости i. Диапазон яркостей делится на два класса с помощью порогового значения уровня яркости k, где k – целое значение от нуля до L. Каждому классу соответствуют относительные частоты $\omega_0\omega_1$. Среднее арифметическое классов выражается через μ_0 и μ_1.

После этого происходит вычисление максимального значения оценки качества разделения изображения на две части. Вычисление в математическом виде представлено в формуле:

$$\eta(k) = \max_{k=1} \left(\frac{\sigma^2_{\text{кл}}(k)}{\sigma^2_{\text{общ}}} \right)$$

где $\sigma^2_{\text{кл}} = \omega_0\omega_1(\mu_1 - \mu_0)^2$ – межклассовая дисперсия, а $\sigma^2_{\text{общ}}$ – это общая дисперсия для всего изображения целиком.

Рисунок 18. Процесс бинаризации изображения (Пустовалов В.)

Пример построения модели нейронной сети с помощью Keras

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
```

После построения компилируем модель методом compile, где параметр loss – это функция потерь (алгоритм измерения того, насколько далеко находится желаемое значение от спрогнозированного), optimizer – функция оптимизации (алгоритм "подгонки" внутренних параметров (весов и смещений) модели для минимизации функции потерь), а metrics – метрики (используются для мониторинга процесса тренировки и тестирования), см листинг ниже.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy
']
```

Вывод структуру модели с помощью метода model.summary().

Обучаем нейронную сеть с помощью метода model.fit (листинг ниже), в котором указываем количество эпох в параметре epochs равное 20, параметром batch_size равным 32 указываем использовать при обучении блоки по 32 изображения. Переменные X_train и X_val отвечают за наборы обучающих данных в размере 31368 изображений и валидационных данных в размере 7841

изображений, а переменные y_train и y_val за метки изображений наших данных соответственно.

```
epochs = 20
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
validation_data=(X_val, y_val))
```

Вывод графиков с помощью класса plt:

```
import matplotlib.pyplot as plt
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('График точности')
plt.xlabel('Эпохи')
plt.ylabel('Точность')
plt.legend()
plt.figure(1)
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('График потерь')
plt.xlabel('Эпохи')
plt.ylabel('Потери')
plt.legend()
```

Пример графика точности обучения:

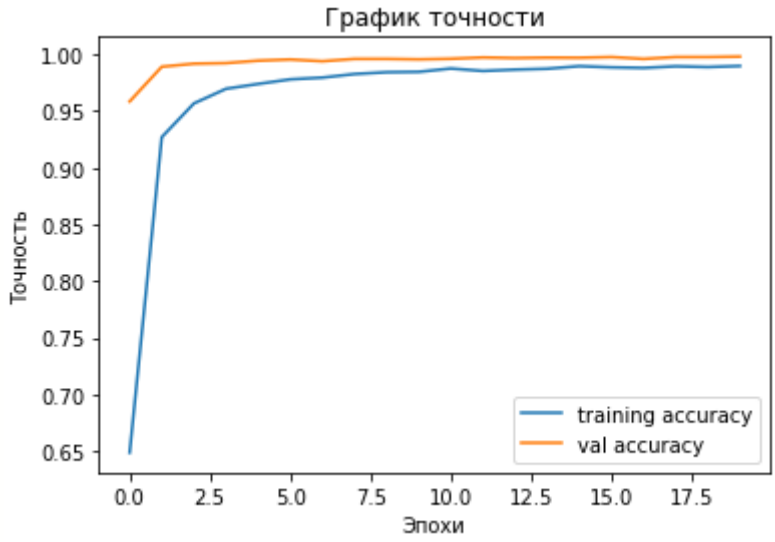


Рисунок 19. График точности обучения

Тестирование работы обученной нейронной сети. Сначала преобразование тестовых данных:

```
y_test=pd.read_csv("/content/Test.csv")
labels=y_test['Path'].values
y_test=y_test['ClassId'].values

data=[]

for f in labels:
    image=cv2.imread('/content/test/'+f.replace('Test/', ''))
    image_from_array = Image.fromarray(image, 'RGB')
    size_image = image_from_array.resize((height, width))
    data.append(np.array(size_image))

X_test=np.array(data)
X_test = X_test.astype('float32')/255
```

После чего создаем предсказания с помощью метода model.predict_classes для каждого тестового изображения X_test и считаем их точность в соответствии с метками классов тестовых данных y_test.

pred = model.predict_classes(X_test)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, pred)

Пример загрузки изображения:

image=cv2.imread('/content/Meta/33.png')

image = cv2.resize(image,(150,150))

```
cv2_imshow(image)
image = cv2.resize(image,(30,30))
cv2_imshow(image)
image = np.expand_dims(image, axis=0)
image.shape
print(image.shape)
predImage = model.predict_classes(image)
print(predImage)
```

ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

1. Собрать обучающую выборку (датасет) с изображениями по определенной тематике (предметной области). Если обучающая выборка не размечена, то разметить обучающую выборку.
2. С помощью сверточной нейронной сети реализовать задачу классификации изображений. Оценить точность классификации.
3. Сформировать отчет о результатах практического занятия и защитить его преподавателю.

Контрольные вопросы к практическому занятию

1. Каким образом сверточная нейронная сеть распознает изображения?
2. Нужно ли уменьшать изображения в обучающей выборке? Если да, то в каких случаях?
3. Сколько слоев в многослойном персептроне сверточной нейронной сети необходимо для распознавания изображений? Можно ли составить зависимость «размер датасета – размер изображания – мощность СНС»?
4. Есть ли различие при обработке изображений разных форматов с помощью СНС?
5. Как повысить скорость обучения СНС?

Литература:

1. Куртова А.А. Пространственная фильтрация // Научно-практическая конференция студентов «Наука и практика: проектная деятельность – от идеи до внедрения» (Томск, 26.11.2012 – 17.12.2012) – Томск., 2012. – 4 с.
2. Исрафилов Х.С. Исследование методов бинаризации изображений // Вестник науки и образования. – 2017. – №6. – С.44-51.

3. Воротников С.А. Информационные устройства робототехнических систем
4. Прохоренок Н. А. OpenCV и Java. Обработка изображений и компьютерное зрение. — СПб.: БХВ-Петербург, 2018. — 320 с.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №7. Обработка текста с использованием сверточной нейронной сети

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ:

61. Внимательно ознакомиться с заданием на практическую работу;
62. Изучить теоретический материал по а) лекции, б) непосредственно к данной работе.
63. Разработать план реализации задания
64. Подготовить входную выборку (если необходимо)
65. Разработать математическую модель алгоритма решения поставленной задачи
66. Программно на любом доступном языке программирования реализовать поставленную задачу
67. Провести эксперименты (если необходимо)
68. Написать отчет о проделанной работе
69. Защитить практическую работу преподавателю
70. Сдать отчет по работе преподавателю (в установленном формате)

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Выполненная практическая работа должна сопровождаться отчетом. Отчет должен содержать не менее 8 страниц текста в формате MS Word (*.doc или *.docx) или LibreOficce (*.odt). Форматирование страницы – все поля по 2 см, межстрочный интервал - полуторный, шрифт Times New Roman 14, нумерация внизу страницы справа. В отчете должны присутствовать: архитектура нейронной сети, результаты экспериментов (таблицы, графики), если разработан уникальный алгоритм построения модели, то необходима блок – схема работы алгоритма. Таблицы и рисунки обозначаются в стандартном варианте.

Весь исходный код разработанного программного обеспечения приводить не надо.

Отчет сдается только в электронном виде (распечатывать не надо).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Задача обработки естественных языков

Проблемой извлечения важных данных из неструктурированного текста занимается особое направление искусственного интеллекта: обработка естественного языка, или NLP (*англ. Natural Language Processing*), которая лежит на стыке нескольких направлений науки – лингвистика, математика и искусственный интеллект. Компьютерная лингвистика тесно связана с искусственным интеллектом, который, на взгляд автора, должен в итоге выполнять следующие функции на уровне человека – зрение и распознавание образов, понимание и генерация речи. И вторая функция, как раз и реализуется с помощью компьютерной лингвистики (если брать в целом).

Выделим основные области лингвистики:

1) Фонология;
2) Морфология;
3) Синтаксис
4) Семантика и прагматика
5) Лексикография

Понимание естественного языка иногда считают AI-полной задачей, потому как распознавание живого языка требует огромных знаний системы об окружающем мире и возможности с ним взаимодействовать (*Википедия*).

NLP – подраздел информатики и искусственного интеллекта, посвященный тому, как компьютеры анализируют естественные языки. NLP позволяет применять алгоритмы машинного обучения для текста и речи.

NLP изучает проблемы компьютерного анализа и синтеза естественных языков. Применительно к искусственному интеллекту анализ означает понимание языка, а синтез — генерацию грамотного текста.

Распространенные задачи обработки естественного языка:

– распознавание речи, перевод текста в текстовом документе;
– синтез речи;
– информационный поиск: извлечение информации из текстов, путем выявления сущности некоторых типов и установления отношения между ними;
– классификация текстовых данных;
– кластеризация текстовых данных;
– анализ социальных сетей: извлечение мнений пользователей социальных сетей о заданных объектах;
– создание вопросно-ответных систем, диалоговых систем и задач машинного перевода и др..

Решение этих задач будет означать создание более удобной формы взаимодействия компьютера и человека.

Есть также раздел в искусственном интеллекте – **компьютерная лингвистика**, в рамках которого также занимаются обработкой текстовых документов. При этом, обработку естественного языка, нередко включают, как раздел компьютерной лингвистики. Кстати, компьютерная лингвистика, практически одновременно зародилась и в СССР и в США. В СССР это

датировано 1958 годов – вышел сборник «Машинный перевод и прикладная лингвистика» (интересовал перевод между языками народов СССР). В СССР прикладной лингвистикой занимались такие знаменитые математики, как Соболев С.Л. и Канторович Л.В. В США основоположником считается лингвист Н. Хомский, который работал над формализацией структур естественных языков.

Но официально днем рождения компьютерной лингвистики считается Джоржтаунский эксперимент, датированный январем 1954 году, в котором было машинно переведено 60 простых текстов с русского на английский (русский язык тогда очень ценился!).

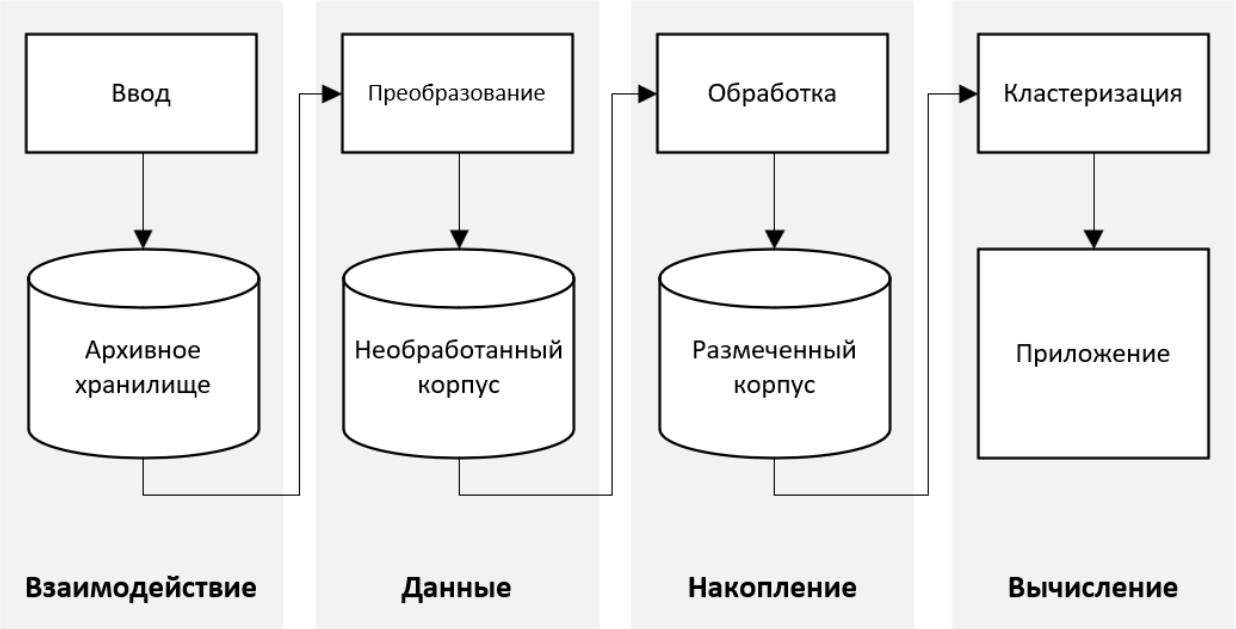


Рисунок 20. Общий принцип работы лингвистической системы

Выделим некоторые проблемы NLP:

- структурная или синтаксическая неоднозначность;

- смысловая неоднозначность (стандартный пример – «за’мок» и «замо’к»);

- падежная неоднозначность;

- референциальная неоднозначность;

- литерация.

С каждой из этих проблем можно бороться различными способами.

Некоторые термины:

Конвейер (pipeline) — это метод объединения последовательности преобразований. Смысл этого подхода в том, чтобы разбить проблему на части и решать их отдельно.

Корпус (corpus/corpora) — это коллекция родственных документов или высказываний на естественном языке. Выделяют даже направление компьютерной лингвистики – **корпусная лингвистика**.

Мешок слов (или Bag of Words) — это модель текстов на натуральном языке, в которой каждый документ или текст выглядит как неупорядоченный набор слов без сведений о связях между ними. Его можно представить в виде матрицы, каждая строка в которой соответствует отдельному документу или тексту, а каждый столбец — определенному слову. Ячейка на пересечении строки и столбца содержит количество вхождений слова в соответствующий документ.

Стемминг (stemming) — это процесс нахождения основы слова (неизменяемой части слова, которая выражает его лексическое значение) для заданного исходного слова.

Стеммер — программная реализация алгоритм *стемминга.*

Стоп-слова (stopwords) – слова в тексте, которые не несут смысловой нагрузки.

Токенизация (tokenization) – это разбиение текста на более мелкие части (токены).

Частотность (frequency) — термин, предназначенный для определения наиболее употребительных слов. Это отношение числа вхождений слова к общему числу слов документа.

Центроид — точка, которая является центром кластера. Каждый центроид является вектором, элементы которого представляют собой средние значения соответствующих признаков, вычисленные по всем записям кластера.

Классически модуль лингвистической модуль обработки текста действует по цепочке:

- определение языка (если это многоязычная лингвистическая система);

- отфильтровывание нечитаемых символов;

- нормализация текста (разделение введенного текста на слова и остальные последовательности символов);

- исправление орфографических и синтаксических ошибок (т.н. спелчекеры);

- удаление предлогов;

- лексический анализ (анализ слов);

- синтаксический анализ (анализ порядка слов в предложении с учётом правил грамматики);

- семантический анализ (анализ значения предложения самого по себе и в его связи с другими предложениями).

Небольшой пример семантического анализа для русского языка (на базе продукционных правил):

Правило 1: ЕСЛИ определение стоит на первом месте и за ним идет существительное, ТО существительное является подлежащим.

Правило 2: ЕСЛИ за подлежащим идет глагол, ТО этот глагол является сказуемым и поясняет, что делает подлежащее.

Правило 3: ЕСЛИ за подлежащим идет сказуемое, а за ним следует существительное, ТО это существительное является дополнением.

Правило 4: ЕСЛИ предложение имеет следующий порядок слов: подлежащее, глагол, дополнение, ТО вся фраза говорит о том, что подлежащее делает (действие, выраженное сказуемым) по отношению к дополнению.

Современный подход к решению задачи с помощью NLP в общем случае можно разделить на следующие этапы:

- нахождение источников данных;

- сбор данных;

- «очистка» данных;

- выбор представления данных (например, «мешок слов»);

- классификация;

- инспектирование и интерпретация модели;

- применение семантики.

Проблема очистки данных обычно содержит следующие шаги:

- удаление нерелевантных символов (но до этого необходимо составить словарь нерелевантных символов);

- токенизация текста;

- удаление нерелевантных слов;

- приведение всех символов в нижний регистр;

- совмещение слов, написанных с ошибками или имеющих альтернативное описание (например, «Апшеронск / Апширонск» или «хорошо / круто»);

- проведение лемматизации, если это необходимо.

Современные примеры применения NLP:

- идентификация различных категорий пользователей или клиентов для прогнозирования их оттока, прибыли от клиентов, продуктовых предпочтений и т.д.;

- детектирование отзывов пользователей о том или ином товаре, услуги или событии (при этом могут быть «вложенные задачи» - например, упоминание в отзывах размера одежды или ее цвета);

- классификация текста в соответствии с его смыслом по таким разделам, как запрос элементарной помощи или подсказки, решение определенной проблемы.

Далее рассмотрим сугубо прикладной пример работы с текстом – определение значимости того или иного события политического или экономического плана (более подробно см. [2, 3]).

События, происходящие в мире, сразу же опубликовываются в ресурсах сети Интернет и задача состоит в том, что бы за заданное время обнаружить новое сообщение, автоматически рейтингировать его и оценить возможные последствия. С другой стороны необходимо на исторических данных исследовать корреляцию событий и их влияние на экономико – политическое поле. При этом данная задача очень сложна, так как данные в ней неструктурированны и их огромное количество. Но нас в данном случае интересует только работа с распознаванием к какой предметной области принадлежит вышедшее событие.

Для решения данных задач необходимо создать информационный ресурс, который бы искал события и сохранял их. Также, для классификации необходима мера, по которой определялась бы «сила» события. В качестве меры можно использовать финансовые временные ряды, как валютного, так и фондового рынках, а также ряды макроэкономических показателей.

В качестве поставщика событий для таких «настольных» систем анализа целесообразно использовать RSS – каналы, а не разрабатывать обработчик поисковых запросов или робота поиска информации в сети. Информационный ресурс в минимальном составе состоит из БД, скрипта загрузки данных и скриптов (макросов, в зависимости от выбранного типа СУБД) обработки данных.

Табличный состав БД следующий (перечислены главные таблицы и основные поля таблиц):

- Страны (Имя, Регион, Население, Площадь, ВВП, первые лица государства и т. д.);
- Корпорации (Название, Страна, выручка, доход, численность рабочих и т.д.);
- События (Страна1, Страна2, Корпорация, Персона, текст, численные показатели,);
- Контракты;
- Макроэкономические показатели и индексы;
- Вспомогательные таблицы.

Использование RSS – каналов выгодно тем, что при использовании адекватного канала можно получить «полный» срез новостей в режиме реального времени по определенной тематике. Также RSS – канал можно подключить к Outlook Express и воспользоваться простым языком VBA для предварительной обработки данных и закачивании их в БД. Но здесь проблема в нахождении адекватных RSS – каналов и в дублировании одного события несколькими каналами. В частности, можно посоветовать использовать канал с сайта http://www.bfm.ru по российским политическим и экономическим событиям. Приведем простой вариант анализа события (новости) при определении к чему оно относится. Пословно читается заголовок RSS – сообщения (слова длинной меньше 3 символов не учитываются, также не учитываются запрещенные слова длинной больше 2 (предлоги и др.)); затем последовательно делается запрос к разным таблицам в БД, к полям с ключевыми словами записей и в случае совпадения новость идентифицируется найденной записью в таблице. При этом, при поиске отсекается последний символ искомого слова – т. е. окончание. Параллельно ведется поиск численной информации. Также ключевые слова заданны, как на русском языке, так и на английском. Данный способ не исключает последующего ручного способа обработки информации, но сводит ручной труд к минимуму.

Практический пример системы распознавания текста

В качестве «полезного» примера применения нейронных сетей и автоматической обработки текста вообще, можно привести вариант использования байесовой нейронной сети для антиспамового фильтра [4]. Полезность такого фильтра в современных реалиях не вызывает сомнения и данный фильтр хорошо себя зарекомендовал, он используется еще с начала 2000-х годов.

Кратко принцип работы. Данный вариант работает на уровне приложения, а не на уровне сети (т.е. это не система обнаружения атак – письмо же ведь может быть «вредоносное») – на входе нейронной сети параметры электронного письма (сигнатуры): адрес отправителя, заголовок и текст, а на выходе нейронной сети – собственно признак «0/1 – спам / не спам».

Как вариант – на вход нейросети можно подавать статистические признаки сообщения, например, процент числа слов и фраз с подозрением на спам (т.е. работает предварительный фильтр на базе специально подобранного, периодически пополняемого словаря), а также не статистические признаки электронного письма: семантические признаки, направленность текста; морфологические признаки и, возможно, орфографические признаки. Данный подход неплохо работает и без нейронной сети, но применение нейронной сети предпочтительнее, т.к. т.н. «спамность» динамически меняется во времени и необходимо перенастраивать, как словарь, так и модель распознавания, а нейронная сеть умеет обобщать накопленный опыт. В работе [4] предлагается

для реализации антиспамового фильтра использовать либо сеть Кохонена, либо гибридную нейронную сеть.

Как замечание – данный подход работает только с текстом, т.е. если в электронное письмо вставлена картинка с рекламным содержимым, то данный подход не будет работать.

Другими прикладными моментами компьютерной лингвистики являются:

А) задача автоматического перевода текста с одного языка на другой;

Б) задача понимания речи и ведение диалога (т.н. чат - боты)

Кратко рассмотрим задачу автоматического перевода текста, а о «чат - ботах» поговорим в одной из следующих лекций.

Задача автоматического перевода текста

Машинный перевод текста с одного языка на другой – это одно из первых направлений искусственного интеллекта. Первые труды (именно по автоматическому переводу) датируются практически в то же время, когда компьютеры стали выходить на промышленный уровень, т.е. 50-е года прошлого века. В настоящее время данное направление продвинулось далеко вперед, достаточно посмотреть уровень перевода таких систем, как «Google Translate» и «Яндекс.Переводчик», но не со всеми специализированными текстами справляются данные системы и далеко не все языки мира они охватили. При этом, точно известно, что в “Google.Translate” (GNMT – поддерживает перевод 30 языковых пар) полномасштабно используются нейронные сети (но основой все – таки является статистические методы). Также выделим системы машинного перевода: «Microsoft Translator», «Amazon Translate», «DeepL».

Обычно выделяют следующие направления машинного перевода:

- Rule-based machine translation (RBMT);

- статистическая трансляция (Statistical machine translation, SMT) – базируется на статистику переводных пар слов и словосочетаний;

- Нейронный машинный перевод (Neural machine translation, NMT) – считается подразделом глубокого обучения.

Нейронный машинный перевод кардинально отличается от предыдущих подходов к машинному переводу. С одной стороны, NMT использует непрерывные представления вместо дискретных символьных представлений в SMT. С другой стороны, NMT использует одну большую нейронную сеть для моделирования всего процесса перевода. Обучение NMT является сквозным, в отличие от отдельно настраиваемых компонентов в статистическом переводе (SMT).

Отметим систему нейросетевого машинного перевода с открытым кодом – OpenNMT от Гарвардского университета.

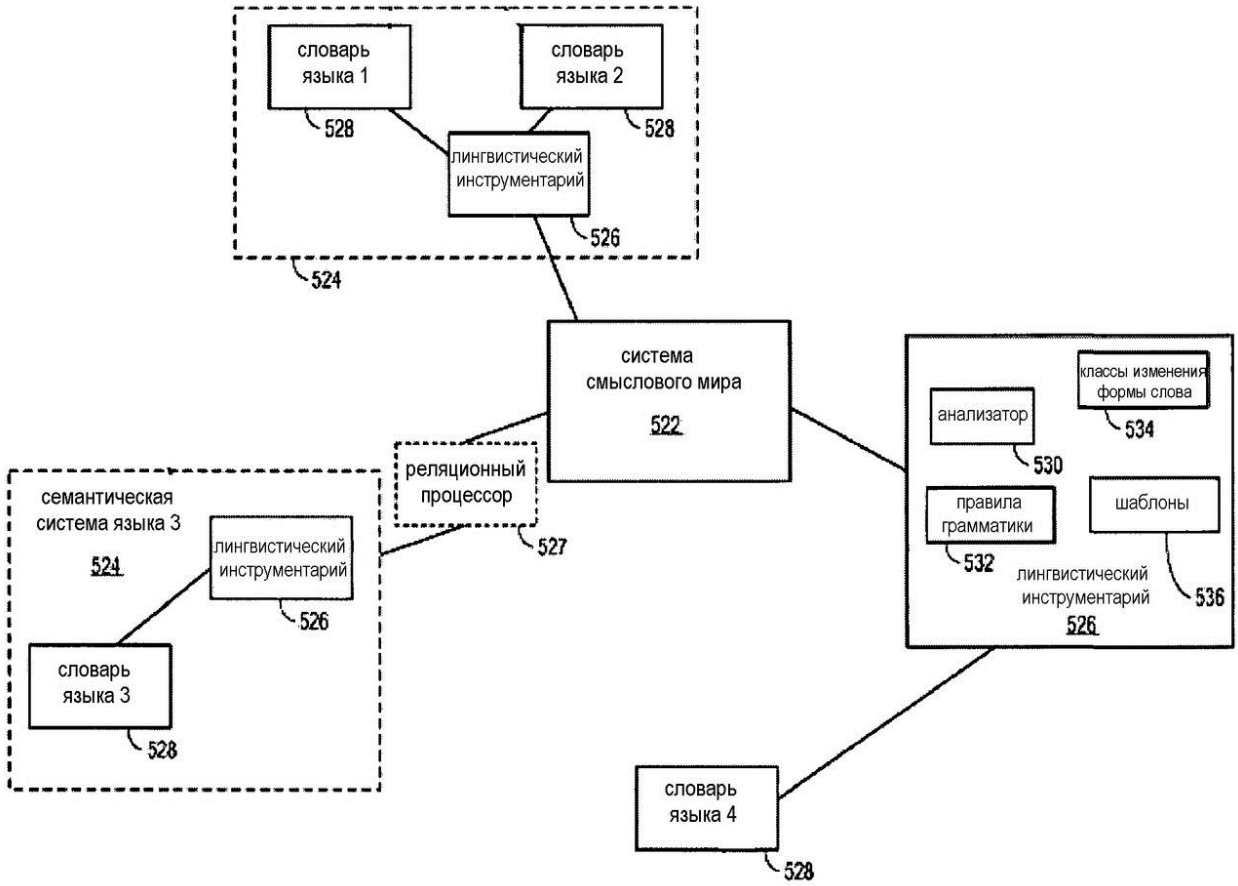


Рисунок 21. Схема обработки в NLTK

Еще приложения компьютерной лингвистики:

- информационный поиск (Informational Retrieval), с которым связаны задачи реферирования, классификации и категоризации, а также рубрицирования документов и текстов. В настоящее время этим занимается т.н. “Text Mining”, который в свою очередь относится к “Data – mining”.

Здесь интересным видится направление реферерирования текста – «выжимки» главного из входного текста, его сути. Можно сказать, что краткое содержание документа. Основной подход к решению проблемы – статистический анализ. Здесь есть еще одна схожая проблема – аннотирование текста, т.е. автоматическое составление аннотации к тексту.

- Задача формирования ответов на вопросы (т.н. FAQ);
- Задача выделение мнений (популярная современна задача. Пример – выявление мнений пользователей определенной соцсети на или иное событие или оценка качества товаров);

- Задача анализа тональности текста – оценка общей тональности высказываний по тому или иному вопросу. Примеры тональных оценок: позитивная, негативная, нейтральная.

Направление распознавания (процесс преобразования речевого сигнала в цифровую информацию) и автоматического синтеза речи будет рассмотрено далее.

Диалоговые системы (чат - боты)

Определение: «*ДИАЛОГОВАЯ СИСТЕМА, или интерактивная система, или система вопрос-ответ — автоматизированная человекомашинная система, работающая в режиме диалога, при котором она отвечает на каждую команду пользователя и по мере надобности обращается к нему за информацией*»[12].

Диалоговые системы относятся к одному из семи важнейших направлений искусственного интеллекта и имеют уже солидную историю.

Краткая история диалоговых систем: одной из первых (и удачных) диалоговых систем была программа – психоаналитик ELIZA (разработчик В. Вейценбаум), которая была создана и успешно работа в 60-х годах прошлого века. Другим пионером диалоговых систем была программа PARRY 1972 года , которая также работа в области психоанализа! Автор – психиатр Кеннет Колбай (Стенфордский университет). Есть интересный эксперимент – в 1972 году по сети ARPANET (предшественник современного Интернета), в котором ELIZA и PARRY общались друг с другом! При этом – данный диалог прошел тест Тьюринга! Опытные психиатры не смогли отличить диалог от диалога с настоящим больных шизофренией (это документально подтверждено).

Диалоговые системы по большому счету классифицируются на две категории – задаче-ориентированные (*General — Task-oriented*) и общего назначения (*Open Domain — Closed Domain*). Вышеупомянутая ELIZA была узкоспециализированной, она могла говорить только на тему психоанализа (в то же время у нее не было четкой задачи, о чем говорить), но по классификации – общего назначения закрытого домена. Есть также другая классификация:

- открытого домена;

- закрытого домена.

Первые – системы, способные говорить на любую тематику, вторые – на строго определенные.

Архитектура современных диалоговых систем (а также голосовых помощников) обычно строится по модульной архитектуре, состоящей из [**Ошибка! Источник ссылки не найден.**]:

[12] https://publishing_dictionary.academic.ru

- модуль распознавания человеческой речи;

- модуля понимания естественного языка;

- модуля набора разговорных навыков;

- модуля составления и ведения диалога;

- модуль генерации естественной речи.

В современных диалоговых системах первый модуль обычно состоит из нейросетевой топологии с возможностью понимания текста, разметки, последовательности и моделей извлечения информации. По сути – это компонент, который определяет направления разговора, предварительно определив тематику (здесь работает функция классификации). В то же время, если разговор общего плана, то возможны различные ньансы. Отдельно выделяют цельные диалоговые системы.

В диалоговых системах сейчас активно применяют нейронные сети, но изначально (и это сохраняется) они использовали экспертные системы различных типов: семантические, фреймовые и продукционные.

Общий принцип работы диалоговой системы, следующий:

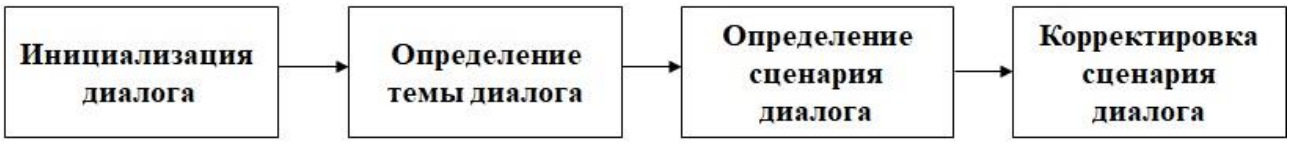


Рисунок 22. Общим принцип функционирования диалоговой системы

Здесь необходимо упомянуть о тесте Тьюринга, который должна пройти каждая уважающая себя диалоговая система: «*Человек взаимодействует с одним компьютером и одним человеком. На основании ответов на вопросы он должен определить, с кем он разговаривает: с человеком или компьютерной программой. Задача компьютерной программы — ввести человека в заблуждение, заставив сделать неверный выбор*»[13].

Примеры современных диалоговых систем:

- «Алиса» от Яндекса;

- Siri от Apple;

- Cortana от Microsoft;

- Amazon Echo / Alexa;

- Google Now / Assistant.

Здесь мы видим «большую пятерку» ИТ – гигантов – все они развивают голосовых помощников (а также системы автоматического перевода) для

[13] https://ru.wikipedia.org/

создания т.н. «эко – ИТ - системы». Все системы (по неподтвержденным данным) используют нейросетевые «движки». У Microsoft есть также система Xiaoice, которая обладает некоторыми «творческими» возможностями – пишет стихи и исполняет песни. Система Siri также использует принципы коллаборативной фильтрации и т.н. персональный подход, когда изучаются в процессе длительного общения с пользователя его предпочтения, т.е. система приспосабливается к пользователю.

Упомянем о международном соревновании разговорного искусственного интеллекта - Conversational Intelligence Challenge (ConvAI) и конкурсе “ Alexa Prize Socialbot Grand Challenge”.

Вопрос определения категории диалога

Рассмотрим один из этапов ведения диалога – определение категории, к которой относится диалог. Нередко для этих целей используется технология т. н. «Мешка слов». Под мешком слов понимается - модель текстов на натуральном языке, в которой каждый документ или текст выглядит как неупорядоченный набор слов без сведений о связях между ними. Его можно представить в виде матрицы, каждая строка в которой соответствует отдельному документу или тексту, а каждый столбец — определенному слову. Ячейка на пересечении строки и столбца содержит количество вхождений слова в соответствующий документ. Одним из основных и сложных моментов данного подхода является определение словаря. При работе с реальными текстовыми массивами получается большой объем словаря, поэтому для сокращения словаря необходимо игнорировать регистр слов и пунктуацию, удалять стоп – слова, предлоги и … Мешок слов естественно формируется на существующей выборке (очевидно, что чем больше выборка, тем качественней сформирован мешок слов).

Далее необходимо ввести кластеры, которые отвечают за категории диалога. Это можно сделать различными способами, в частности: k –means, mini – batch k – means, agglomerative clustering и др.

Далее введем такое понятие, как *сценарий диалога*. Понятно, что диалог может быть совершенно различный, в частности «пустой». Но, будем ориентироваться в сторону профессионального диалога. Также введем понятие соответствия диалога и точность ответов машины. Настройка сценария диалога может проходить, как в режиме реального времена, так и в офф-лайн режиме на наработанной истории. Необходима также т.н. разметка диалога, т.е. режим, в котором эксперт (или пользователь) проставляет коэффициенты точности ответов и соответствия диалога. Одной из перспективных технологий для создания подобных систем с корректировкой являются нейросетевые топологии с подкреплением.

Библиотеки для построения диалоговых систем

Выделим следующие распространенные библиотеки обработки естественного языка:

- NLTK - поддерживает такие задачи, как классификация, стемминг, маркировка, синтаксический анализ и семантическое рассуждение (ЯП – Python, разработчик – Пенсильванский университет, США. Авторы С. Берд и Э. Лопер).

- spaCy – «spaCy разработан, чтобы помочь вам выполнять настоящую работу - создавать настоящие продукты или собирать реальные идеи. Библиотека уважает ваше время и старается не тратить его зря. Его легко установить, а его API прост и продуктивен» - пример машинного перевода translate.google.com . Считается, что spaCy обладает самым быстрым парсером из всех существующих NLP систем, по крайней мере с открытым кодом. Написан на Cython. Поддерживает 7 языков (на 2020 год).

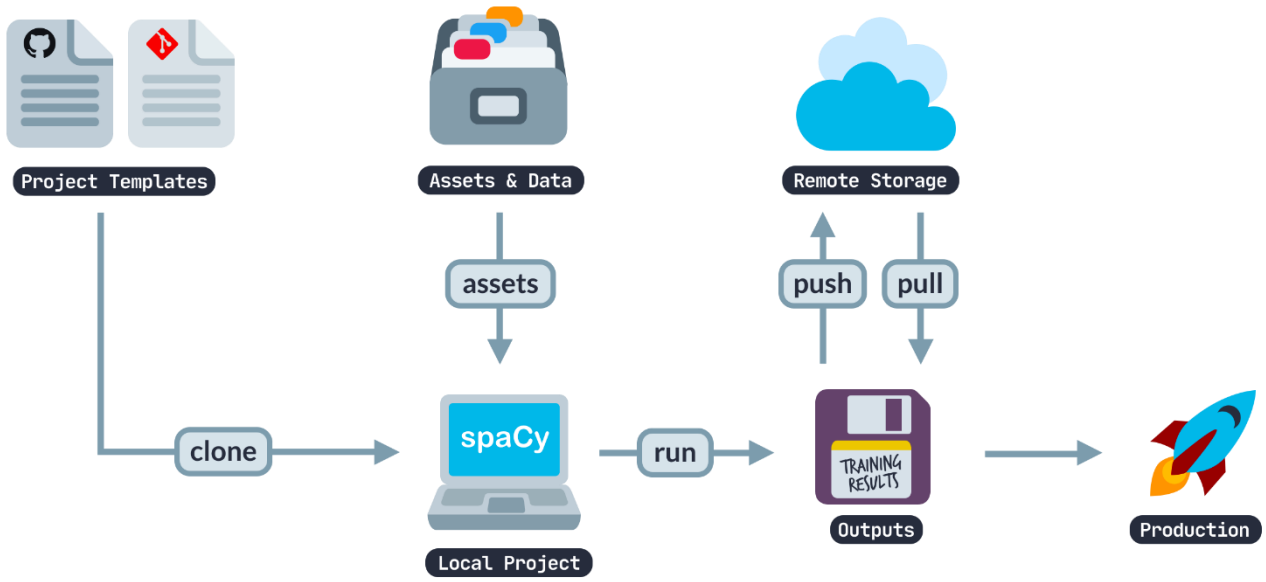


Рисунок 23. Принцип работы spyCy (схема с сайта https://spacy.io/)

«несколько компонентов могут использовать общую модель «токен-вектор», и лемматизатор легко заменить или отключить. Трубопроводы спроектированы так, чтобы быть эффективными с точки зрения скорости и размера и хорошо работать, когда трубопровод запущен на полную мощность» - еще вариант современного машинного перевода.

- fastText - для изучения семантики и классификации текста, созданная Facebook AI Research.

- CoreNLP - ЯП Java, но есть оболочки на Python, первая «промышленная» версия вышла в 2010 году. Создана в Стенфордском университете (США). Работает по принципу «pipeline» (трубопровод). Краткий принцип работы – «текст помещается в объект аннотации, а затем аннотаторы

добавляют информацию в аналитический трубопровод. Результирующая аннотация, содержащая всю аналитическую информацию, может выводиться в виде XML или обычного текста».

CoreNLP проводит морфологический и семантический анализ. Результаты пока оставляют желать лучшего…

- Pattern. Еще одна библиотека NLP на Python. Предоставляет инструменты для частеречной разметки (*part-of-speech tagging*), анализа настроений, векторных пространств, моделирования (SVM), классификации, кластеризации, n-граммы поиска и WordNet.

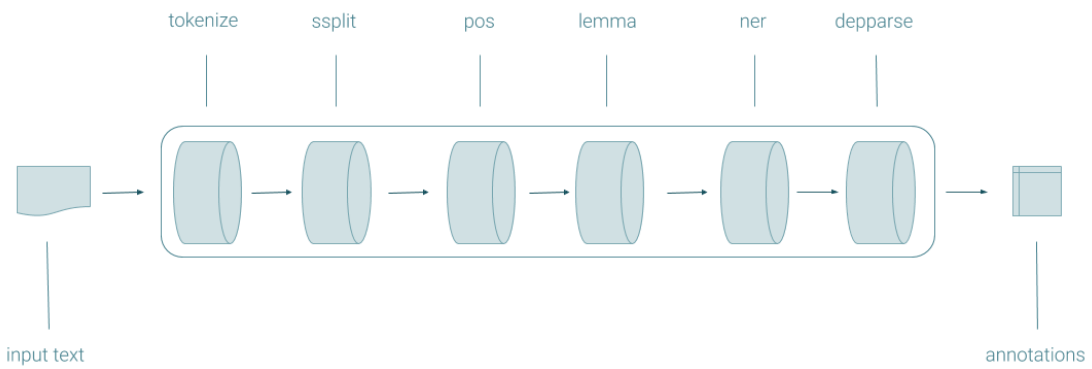


Рисунок 24. Принцип работы CoreNLP

Изображения с https://stanfordnlp.github.io .

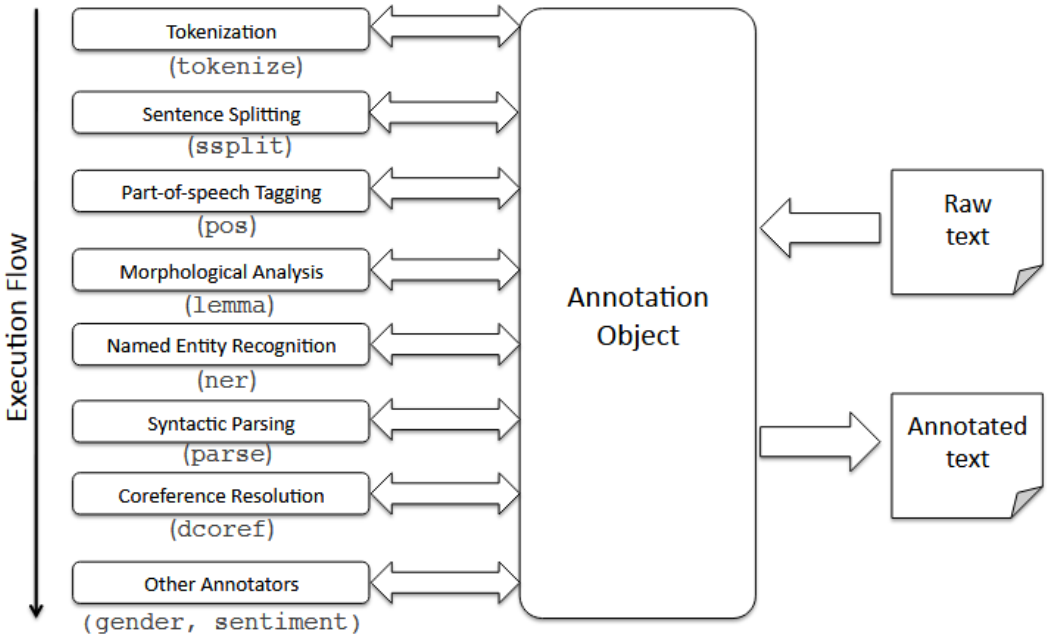


Рисунок 25. Архитектура CoreNLP

- TextBlob – библиотека для обработки текстовых данных на естественном языке. Хорошо работает в задачах анализа настроений (да, есть и такая задача…), POS – маркировка (маркирование семантических предложений в предложениях), извлечение именных фраз. ЯП – Python.

Из нейросетевых инструментов наибольшую популярность получили: TensorFlow, Keras, PyTorch, а также некоторые другие, в основном на языке программирования Python.

Отметим, что среди вышеперечисленных, нет ни одной системы (библиотеки) российского производства (хотя российские программисты наверняка трудились над ними).

ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

В данной работе исследуется применение сверточных нейронных сетей для обработки текста.

Задание 1.

1. Собрать коллекцию текстов на определенную тематику (спорт, кинематограф, нейронные сети и т.д.), можно использовать «датасеты» с сайта http://www.kaggle.com/ . Написать процедуру предварительной обработки текста.
2. Реализовать процедуры разбиения текста на слова и предложения.

3. Адаптировать разработанную ранее сверточную нейронную сеть для работы с текстом.
4. Провести классификацию собранного датасети с помощью сверточной нейронной сети.

Задание 2.

1. Разработать принципиальную схему и алгоритм работы чат – бота для реализации общения на заданную тематику (тематику взять из Задания 1).
2. Реализовать принципиальную схему и алгоритм работы чат – бота на доступном языке программирования (допускается консольный интерфейс).
3. Протестировать разработанный чат – бот и оценить качество ответа на вопросы.

Литература:

1. Бенгфорт Б., Билбро Р., Охеда Т. Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка. — СПб.: Питер, 2019. — 368 с.
2. Шумков Е.А. Расчет силы влияния макроэкономических новостей // Сетевой электронный научный журнал «Труды КубГТУ». №1, 2016
3. Шумков Е.А., Карлов Д.Н. Автоматическая система отслеживания политических и экономических новостей / XXV Международная научно-техническая конференция (летняя сессия) "Математические методы и информационные технологии в экономике, социологии и образовании". Пенза. 2010. сс. 29-31.
4. Ларионова А.В., Хорев П.Б. Метод фильтрации спама на основе искусственной нейронной сети // Интернет-журнал «НАУКОВЕДЕНИЕ» Том 8, No3 (2016) http://naukovedenie.ru/PDF/04TVN316.pdf (доступ свободный).
5. Большакова Е.И., Воронцов К.В. и др. Автоматическая обработка текстов на естественном языке и анализ данных. М.: Изд-во НИУ ВШЭ. 2017. 269 с.
6. Леонтьева Н.Н. Автоматическое понимание текстов: системы, модели, ресурсы. – М.: Академия, 2006.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №8. Разработка нейросетевой топологии с подкреплением для задачи глубокого обучения

ПОРЯДОК ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ:

71. Внимательно ознакомиться с заданием на практическую работу;
72. Изучить теоретический материал по а) лекции, б) непосредственно к данной работе.
73. Разработать план реализации задания
74. Подготовить входную выборку (если необходимо)
75. Разработать математическую модель алгоритма решения поставленной задачи
76. Программно на любом доступном языке программирования реализовать поставленную задачу
77. Провести эксперименты (если необходимо)
78. Написать отчет о проделанной работе
79. Защитить практическую работу преподавателю
80. Сдать отчет по работе преподавателю (в установленном формате)

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Выполненная практическая работа должна сопровождаться отчетом. Отчет должен содержать не менее 8 страниц текста в формате MS Word (*.doc или *.docx) или LibreOficce (*.odt). Форматирование страницы – все поля по 2 см, межстрочный интервал - полуторный, шрифт Times New Roman 14, нумерация внизу страницы справа. В отчете должны присутствовать: архитектура нейронной сети, результаты экспериментов (таблицы, графики), если разработан уникальный алгоритм построения модели, то необходима блок – схема работы алгоритма. Таблицы и рисунки обозначаются в стандартном варианте.

Весь исходный код разработанного программного обеспечения приводить не надо.

Отчет сдается только в электронном виде (распечатывать не надо).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Основные термины и определения

Объект управления (или агент) – то, чем управляем. Например, мобильный робот, механическая торговая система, чат – бот и т.д.

Характеристики объекта управления. Под характеристиками объекта управления будем понимать состояния его параметров, например, запас топлива, заряд аккумуляторной батареи, состояние корпуса, быстродействие и т.д.

Стратегия – это отображение воспринимаемых состояний среды в действия, отвечающие этим состояниям. *Или* – определение характера обучающегося агента в каждый момент времени.

TD – временная разность (от англ. Time – Difference). См. ниже Ошибка временной разности.

ОИС – обобщенная итерация по стратегиям.

Функция оценки – показывает, что есть хорошо в продолжительный период, тогда как функция подкрепления показывает, что есть хорошо в текущий момент. Оценка состояния - это итоговое подкрепление агента, которое предположительно может быть накоплено при последующих стартах из этого состояния. В то время как подкрепление определяет прямую, характерную желательность состояния окружения, оценки показывают долгосрочную желательность состояний после принятия во внимания состояний, которые последуют за текущим, и подкреплений, соответствующих этим состояниям. Например, состояние может повлечь низкое непосредственное подкрепление, но иметь высокую оценку, потому как за ним регулярно следуют другие состояние, которые приносят высокие подкрепления.

Функция подкрепления - определяет цель в процессе обучения с подкреплением. Это соответствие между воспринимаемыми состояниями среды и числом, подкреплением, показывающим присущую желательность состояния. Единственная цель агента состоит в максимизации итогового подкрепления, которое тот получает в процессе длительной работы. Функция отражает и определяет существо проблемы управления для агента. Она может быть использована как базис для изменения правил. Например, если выбранное действие повлекло за собой низкое подкрепление, правила могут быть изменены для того, чтобы в следующий раз выбрать другое действие. В общем случае, функция подкрепления может быть стохастической.

Обычно выделяют два подхода к обучению с подкреплением [6]:

1) Классический подход. В данном подходе процесс обучения происходит за счет поощрения и наказания («кнута и пряника») для достижения *высококлассного поведения*. Данный подход уходит корнями в давние работы по психологии, в частности Павлова – по условному рефлексу и Торндайка– обучение животных. Также к классическому подходу относят использование генетических алгоритмов для задач с подкреплением.

2) *Современный подход*. Данный подход основан на методе динамического программирования, используемом для формирования последовательности действий с учетом будущих состояний без фактического их осуществления. Также используются различные статистические техники. Данный подход будет подробно рассмотрен ниже. Учитывая, что методы динамического программирования требуют полного знания об объекте управления, то

выглядит немного странным использование их в качестве ядра для построения алгоритмов обучения с подкреплением, но данные методы многошаговые и итерационные приводят к ответу постепенно, что и используется в алгоритмах обучения с подкреплением.

Но фундаментально можно выделить три класса методов обучения с подкреплением:

1). с использованием принципов динамического программирования (в т.ч. асинхронного);

2). на базе методов Монте – Карло;

3). на базе метода временных различий.

С другой стороны выделяют две основных принципа построения алгоритмов обучения с подкреплением – с использованием модели (*model - based*) без использования модели (*model - free*). Второй тип оценивает оптимальную политику без использования динамики окружающей среды, первый наоборот ее использует для оценки оптимальной политики.

Для задач, решаемых с помощью обучения с подкреплением характерно следующее:

- разные действия приводят к разным выигрышам;

- агент получает выигрыш с задержкой времени;

- выигрыш зависит от текущего состояния среды.

Агент и среда взаимодействуют на каждом из последовательности дискретных временных шагов, t = 0, 1, 2, 3, … . На каждом временном шаге, t, агент получает некоторое представление о *состоянии* среды, $s_t \in S$, где S это множество всех возможных состояний. На основе состояния агент выбирает *действие*, $a_t \in A(s_t)$, где $A(s_t)$ это множество действий, возможных в состоянии s_t. Во время следующего шага, как часть ответа на действие, агент получает числовое *подкрепление,* $r_{t+1} \in R$, и переводит себя в состояние s_{t+1}.

Одним из ключевых моментов в обучении с подкреплением является собственно само *подкрепление* (англ. – reinforcement: *подъем, укрепление, подкрепление и др.*). *Подкрепление* – это некоторая оценка действий агента или объекта управления (обычно безразмерная). Иногда подкреплением называют *выгодой, вознаграждением или поощрением* [6]. В работах [7, 8] вводится коэффициент эффективности, являющийся модернизированным подкреплением. Задача объекта управления (или *агента*) – максимизировать суммарное вознаграждение по итогам своей работы. Следует отметить, что обычно процесс взаимодействия объекта управления со средой, может быть многократно повторен, т.е. для каждого состояния необходимо рассчитывать

усредненное подкрепление – т.н. задача с *эпизодами*[14]. В тоже время взаимодействие может быть непрерывно и бесконечно, т.е. $T = \infty$ и получается, что возможное вознаграждение по первоначальной формуле может стремится к бесконечности. Для «обхода» данного момента вводится понятие *приведенной величины выгоды* (одна из трактовок *приведения*), которая рассчитывается как:

$$R_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + .. = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1}$$

где γ - *коэффициент приведения* $(0 \le \gamma \le 1)$. Суть применения приведенной величины – объект управления должен стараться выбирать такие действия, которые приводят к максимальному суммарному вознаграждении.

Q – обучение

Основной принцип работы Q- обучения – это ассоциация функции подкрепления с каждой парой ситуация – действие, так называемая «таблица Q-обучения».

Основной принцип работы Q- обучения [3, 4] – это ассоциация функции подкрепления с каждой парой ситуация – действие. Схема Q – обучения, такая же, как в стандартном представлении обучения с подкреплением, единственное отличие – в функции отображения (в данном случае она называется Q – функция), которая зависит от пары ситуация - действие. В Q – обучении присутствует 3 различных функции: память, исследование и корректировка. При ответе на существующую ситуацию, действие выбирается с помощью функции оценки с использованием памяти. После выполнения действия в реальной среде, функция подкрепления выдает значение подкрепления. Это значение $(\in [-1;1])$ используется функцией корректировки для приспосабливания значения оценки Q в соответствии с последними действиями объекта.

Алгоритм Q– обучения строит Q – функцию, которая отображает пары ситуация – действие (i,a) в ожидаемое значение r. $Q(i,a)$ – это оценка системы. Алгоритм использует таблицы для хранения накопленной оценки Q. Непустые ячейки таблиц отражают уже испытанные значения пар.

Общий алгоритм Q – обучения следующий:

[14] Эпизод – это либо разбивка на *подпоследовательности* взаимодействия объекта управления с внешней средой, либо полностью «проигранная» комбинация взаимодействия до терминального состояния. Обычно под эпизодом понимается первое. Примеры эпизода: партия в игре, однократное прохождение лабиринта и другие типы повторяющихся испытаний. Эпизоды также часто в литературе называют *испытаниями*.

- инициализируется память: для всех значений пар ситуация – действие присваивается значение $Q(i,a)=0$, либо небольшое значение, чуть больше 0;
- цикл
  1. пусть i – есть ситуация среды;
  2. функция оценки выбирает действие a для выполнения: $a = Max(Q(i,a'))$, где a' – представляет любое возможное действие[15];
  3. объект отрабатывает действие a в среде на данной итерации, r – подкрепление (r может быть равно нулю) ассоциированное с выполнением действия a в среде;
  4. обновляется память по формуле:

$$Q_{t+1}(i,a) = Q_t(i,a) + \beta(r + g * Max(Q_t(i'a')) - Q_t(i,a))$$

где i' - новая ситуация после выполнения действия a в ситуации i, a' - любое возможное действие.

Существуют следующие варианты Q – обучения без применения искусственных нейронных сетей:

- Q – обучение с использованием расстояния Хемминга;
- Q – обучение со статической кластеризацией.

Также приведем Q – обучение с применением нейронных сетей:

- Q – обучение с использованием многослойного персептрона;
- Q – обучение на базе нейронной сети Кохонена или самоорганизующихся карт Кохонена (SOM);
- Dyna – Q;

Competitive MLP - CMLP (на основе конкуренции между нейронными сетями).

Существует несколько вариантов расчета подкрепления для Q – обучения. Рассмотрим их подробнее. Основной принцип Q – обучения – это накопление значений ценностей действий в различных ситуациях и последующий обоснованный их выбор. Базовым вариантом накопления ценностей действий является *средний выборочный* [6]. Пусть к моменту времени t действие a_j в ситуации s_i было выбрано k_{ij} раз и это привело к получению соответствующих вознаграждений $r_1, r_2, .., r_{k_{ij}}$. Тогда ценность этого действия в ситуации s_i можно определить как простое среднее:

$$Q_t(a_i) = \frac{r_1 + r_2 + .. + r_{k_{ji}}}{k_{ij}}$$

В случае, если $k_{ij} = 0$, т.е. данное действие еще не выбиралось, устанавливается значение по умолчанию, обычно равное нулю. В ходе работы системы с Q – обучением с расчетом по среднему выборочному логично выбирать, то

[15] Процесс выбора а’ может быть стохастическим, это позволяет исследовать новые области

действие в сложившейся ситуации, у которого накопленная ценность выше. Таким образом, всегда выбираются те действия, которые приводят к максимальной текущей ценности. Но, при этом выбираются только, те действия, которые уже были пройдены системой, т.е. нет реального «исследовательского режима». У данного метода есть и плюсы – достаточно простой с минимумом вычислений, и минусы, нет реального поиска в пространстве состояние – действие.

Усложненным вариантом расчета ценности предстоящего действия в данном случае является использование т.н. «жадных»[16] стратегий.

Для понимания Q – обучения важным является матричное и графическое его представление. Обычно Q – матрицу, т.е. матрицу отображающую ценность пар (s_t, a_t) представляют следующим образом (Таблица 1). В таблице n - возможное количество состояний, а m - возможное количество действий (обычно $n \succ m$).

Таблица 1

Представление Q – таблицы (ситуация - действие)

	s_1	s_2	s_3	…	s_n
a_1	Q_{11}	Q_{12}	Q_{13}	…	Q_{1n}
a_2	Q_{21}	Q_{22}	Q_{23}	…	Q_{2n}
a_3	Q_{31}	Q_{32}	Q_{33}	…	Q_{3n}
				…	
a_m	Q_{m1}	Q_{m2}	Q_{m3}	…	Q_{mn}

Но не менее важной является таблица вероятностей перехода из одного состояния в другое (Таблица 2). Отметим, что объект управления может остаться в том же состоянии, что и был.

Таблица 2

Вероятности перехода системы из одного состояния в другое

	s_1	s_2	s_3	…	s_n
s_1	V_{11}	V_{12}	V_{13}	…	V_{1n}
s_2	V_{21}	…	…	…	…

[16] Англ. – greedy.

s_3	V_{31}	…	…	…	…
…	…	…	…	…	…
s_n	V_{n1}	…	…	…	…

Вероятности перехода из одного состояния в другое можно рассчитывать несколькими способами, …

Кроме того, в ряде задач возникает необходимость рассчитывать вероятности перехода из одного состояния в зависимости от выполняемого действия (см. Таблица 3).

Таблица 3

	s_1				s_2				…
	a_1	a_2	…	a_m	a_1	a_2	…	a_m	…
s_1	$V(S_1)_{11}$	$V(S_1)_{11}$		$V(S_1)_{1m}$	$V(S_2)_{11}$	$V(S_2)_{12}$		V_{1m}	…
s_2	$V(S_1)_{21}$								…
s_3	$V(S_1)_{31}$								…
…	…	…	…	…	…	…	…	…	…

Еще один вариант представления Q – таблицы: ведется таблица 4, в которой отражается накопленное подкрепление (значение функции ценности) при переходе из одного состояния в другое, но при этом есть еще одна таблица, в которой прописаны действия, переводящие агента из одного состояния в другое. В некоторых случаях, если $S(t) = S(t+1)$ также может поступать подкрепление.

Таблица 4

Накопление подкрепления при переходе из одного состояния в другое

	S_1	S_2	…	S_n
S_1	**	R_{12}	…	R_{1n}
S_2	R_{21}	**	…	…
…	…	…	**	…

S_n	R_{n1}	R_{n2}	…	**

Таблица 5

Действия, переводящие из одного состояния в другое

$S(t)$	$a(t)$	$S(t+1)$
S_1	a_1	S_2
S_2	a_2	S_4
S_2	a_3	S_1
S_3	a_1	S_4
…	…	…

Понятно, что все таблицы могут иметь высокую размерность.

Достоинства Q – обучения (общие):

- нет необходимости строить внутреннюю модель среды;

- простой механизм выбора действия;

- малая требовательность к вычислительным ресурсам;

Общие недостатки Q – обучения:

- Q – таблицы при решении реальных практических задач могут иметь высокую размерность.

- тенденция к завышению оценок функции награды.

Кратко поясним отличие стандартного Q – Learning от модифицированного – в последнем используется оценка Q_{t+1}, которая связана с выбранным действием, а не максимальная, как в стандартном алгоритма. Это гарантирует, что ошибки временной разности будут рассчитываться правильно и не будут зависеть от того, как выбираются действия с «жадной» политикой и нет необходимости обнулять λ. Также здесь важным является то, что постоянный выбор действия с максимальной текущей оценкой часто приводит к тому, что итоговое значение подкрепления будет не максимальной, т.к. возможны пропуски дальнейших действий с более высокой оценкой.

В алгоритме $Q(\lambda)$ прогноз Q_t не присутствует в выражении корректировки Δw_t в явном виде, за исключением случая, когда действия выбираются в

соответствии с «жадной» политикой[17]. Другая трактовка алгоритма – «данный метод базируется на выполнении обычного одношагового правила обновления для улучшения текущего прогноза Q_t и последующем использовании временной разности между следующими друг за другом `жадными` прогнозами». Т.е. в данном методе действие не выбирается с помощью `жадного` правила, но `жадное` правило присутствует на более высоком уровне.

Сети адаптивной критики

Адаптивные критики являются частью нескольких научных направлений, в частности: обучения с подкреплением и нейроуправления. Часто используется название «*исполнитель - критик*». Также адаптивные критики известны как *приближенное динамическое программирование* (англ. - *Approximated Dynamic Programming, ADP*).

Критик – это блок системы управления, который оценивает качество ее работы. Обучение в адаптивных критиках всегда строится по TD – методу. Название *критик* введено в связи с тем, что данный блок, ответственный за функцию ценности, критикует действия, осуществляемые исполнителем. Критика обычно принимает форму TD – ошибки и этот скалярный сигнал, являющийся единственным сигналом от критика, регулирует все обучение, как критика, так и исполнителя [1]. В основном критик реализует функцию ценности состояния и после каждого выбора действия критик производит оценку нового состояния на предмет улучшения и ухудшения по сравнению с тем, что ожидалось. Оценка считается, как TD – ошибка:

$$\delta_t = r_{t+1} + \gamma \cdot V(s_{t+1}) - V(s_t)$$

где V - текущая функция ценности, реализованная критиком.

Агент задает действия системы управления. Цель агента – максимизировать суммарную награду (подкрепление), которую можно получить в будущем в течение длительного периода времени. Агент оценивает суммарную награду с учетом коэффициента забывания:

$$U(t) = \sum_{k=0}^{\infty} \gamma^k \cdot r(t+k)$$

[17] Данный алгоритм можно интерпретировать, как взвешенную сумму урезанных возвратов [**Ошибка! Источник ссылки не найден.**].

где $U(t)$ - оценка суммарной награды, γ - коэффициента забывания, $0<\gamma<1$. Коэффициент забывания означает следующее – чем дальше в будущее заглядывает агент, тем меньше у него уверенности в оценке награды.

Адаптивных критиков целесообразно использовать в тех случаях, когда не работает итеративная схема формирования матрицы $Q(S(t),a(t))$.

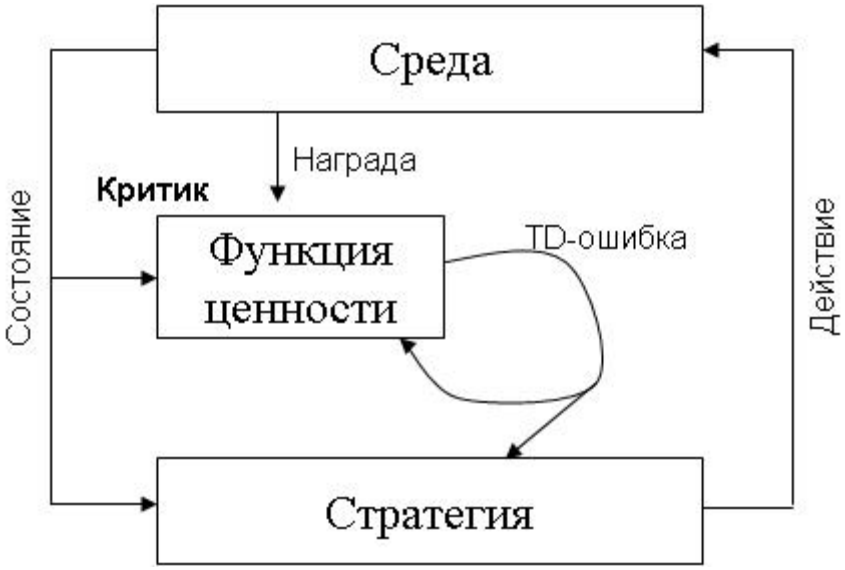


Рисунок 26. Обобщенная схема адаптивного критика

Для подробного рассмотрения современного подхода к обучению с подкреплением необходимо вначале рассмотреть принцип динамического программирования, который является ключевым (отправным моментом) при построении таких систем.

Отметим, что современный подход к обучению с подкреплением также называется *нейродинамическим программированием* [**Ошибка! Источник ссылки не найден.**], в связи с тем, что используется принцип динамического программирования с реализацией на нейронных сетях.

Принцип динамического программирования основан на идее Р. Беллмана [5], известной, как *принцип оптимальности Беллмана* и представляет собой поэтапный метод принятия решений[18]. При этом перед принятием следующего решения последствия текущего решения предсказываются на некоторый интервал будущего. Напомним формулировку принципа оптимальности Р. Беллмана:

Оптимальная стратегия имеет следующее свойство - какими бы ни были начальные состояние и решение, остальные решения должны составлять

[18] В то же время, согласно ретроспективе, данной проблеме уделял много внимания русский ученый Марков, а Беллман предложил конкретное уравнение и удачную формулировку. Также, уравнение Беллмана является следствием теоремы Якоби – Гамильтона.

оптимальную стратегию по отношению к состоянию, вытекающему из первого решения.

Напомним рекомендованную последовательность шагов в алгоритме динамического программирования [5]:

1. Выбрать параметры, характеризующие состояние S управляемой системы перед каждой итерацией.
2. Разбить операцию на этапы.
3. Определить набор шаговых управлений x_i для каждого шага и налагаемые на них ограничения.
4. Оценить, какой выигрыш приносит на i-м шаге управление x_i, если система до него была в состоянии S_{i-1}. Таким образом необходимо записать функции выигрыша:

$$w_i = f_i(S_{i-1}, x_i)$$

5. Определить, как изменяется состояние S системы под влиянием управления x_i на i - м шаге, то есть рассчитать:

$$S_{i+1} = \varphi_i(S, x_i)$$

6. Записать основное рекуррентное уравнение динамического программирования, выражающее условный оптимальный выигрыш $W_i(S)$ (с i-го шага и до конца), через уже известную функцию $W_{i+1}(S)$ [19]:

$$W_i(S) = \max\{ f_i(S, x_i) + W_{i+1}(\varphi_i(S, x_i))\}$$

7. Произвести условную оптимизацию последнего m-го шага, задаваясь возможными состояниями S, из которых можно за один шаг перейти в конечное состояние. При этом для каждого возможного состояния S вычисляется условный оптимальный выигрыш по формуле:

$$W_t(S) = \max\{ f_t(S, x_t)\}$$

и определить условное оптимальное управление $x_t(S)$, при котором этот максимум определяется.

8. Произвести условную оптимизацию $(t-1)$ - го шага, $(t-2)$ - го и других шагов по вышеобозначенной формуле, при этом полагая в ней $i = (t-1),(t-2),...$ и для каждого из шагов указать условное оптимальное управление $x_i(S)$, при котором максимум достигается.

[19] Этому выигрышу соответствует условное оптимальное управление на i - м шаге.

9. Произвести безусловную оптимизацию управления на каждом шаге, то есть взять найденное управление на первом шаге[20] x_1^*, затем на втором x_2^* и так далее.

Задача динамического программирования может иметь конечный и бесконечный горизонт. Бесконечный горизонт может быть на самом деле конечным, но с очень большим количеством шагов. Общие ожидаемые затраты в задачах с бесконечным горизонтом определяются по формуле:

$$J^{\pi}(i) = E[\sum_{n=0}^{\infty} \gamma^n \cdot g(X_n, \mu_n(X_n), X_{n+1}) \mid X_0 = i]$$

где $X_0 = i$ - начальное состояние, $\pi = \{\mu_n\}$ - стратегия поведения. Ожидаемое значение затрат вычисляется по цепи Маркова $\{X_1, X_2, X_3, ...\}$. Функция $J^{\pi}(i)$ называется функцией стоимости перехода для стратегии π, начиная с шага i (англ. *cost – to – go - function*). Оптимальное значение функции затрат (в смысле минимума) $J^*(i)$ определяется соотношением:

$$J^* = \min_{\pi} J^{\pi}(i)$$

Решение уравнения оптимальности Беллмана является одним из подходов к поиску оптимальной стратегии, в частности для задачи обучения с подкреплением. Но для задач с подкреплением принцип оптимальности Беллмана применяется не часто по следующим причинам:

- метод динамического программирования требует точной и полной модели окружающей среды, что накладывает определенные ограничения для его использования для большого круга задач, решаемых с помощью обучения с подкреплением
- необходим достаточный вычислительный ресурс;
- сигнал состояния должен обладать марковским свойством (окружающая среда должна описываться как конечный МППР).

Задачи редко удовлетворяют трем вышеперечисленным условиям и поэтому приходится работать с приближенными решениями.

Нейродинамическое программирование позволяет системе обучаться принятию правильных решений с помощью наблюдения за собственным поведением и совершенствовать свое поведение путем использования встроенного механизма усиления.

В задачах нейродинамического (и динамического) программирования существуют две серьезные проблемы:

1) *Проклятие размерности*. В большинстве реальных задач количество возможных состояний и действий в них является настолько большим, что их во

[20] Обычно состояние системы в начальный момент известно и на первом шаге варьировать состояния системы не нужно – необходимо сразу взять оптимальный выигрыш на первом шаге.

– первых трудно все учесть, а во – вторых возникают серьезные требования к вычислительным мощностям. Например, при игре в нарды имеется 10^{20} возможных состояний. Вообще говоря, в задаче динамического программирования содержащей K возможных состояний и M возможных действий в них, каждая итерация алгоритма требует $K^2 \cdot M$ операций для стационарной стратегии.

2) *Неполнота информации*. Алгоритмы итераций по стратегиям и значениям требуют знания вероятностей переходов между состояниями p_{ij} и стоимости этих переходов. Но такие данные зачастую в реальных задачах априори неизвестны.

Поэтому в реальных задачах приходится отказываться от оптимальных стратегий и искать субоптимальные.

Для поиска субоптимальных стратегий обычно применяются нейронные сети, так как они являются одними из лучших аппроксиматоров. При этом, например, в вводят обучение с подкреплением, как подраздел обучения без учителя. В случае нейродинамического программирования нейронная сеть аппроксимирует функцию стоимости перехода $J^*(i)$ для $\forall i \in X$. В частности, для конкретного i функция затрат $J^*(i)$ заменяется подходящей аппроксимацией $\hat{J}(i,w)$, где w - вектор параметров. Функция $\hat{J}(\cdot,\cdot)$ называется приближенной функцией стоимости перехода (см. рисунок).

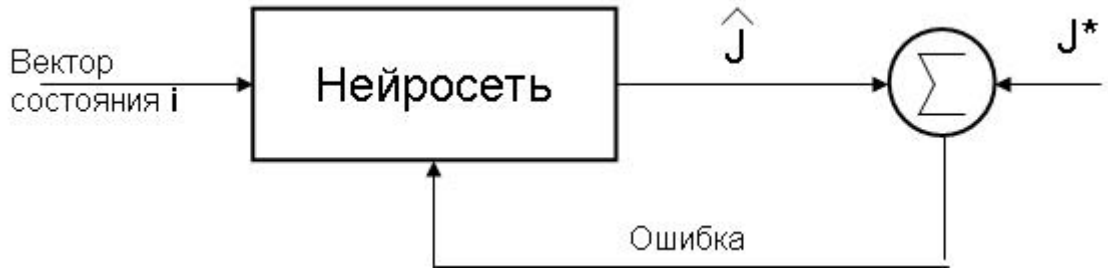


Рисунок 27. Нейросеть в качестве аппроксиматора функции затрат

Заметим, что в ряде работ на вход нейросети подается только номер состояния i, что на взгляд автора в реальных задачах встречается редко, в основном такое описание задач встречается в учебных примерах. Необходимо подавать вектор, описывающий текущее состояние i.

В реальных задачах также зачастую необходимо идентифицировать текущее состояние и для этого, как раз необходим вектор, описывающий текущее состояние объекта управления. Для данной задачи, в случае конечного количества состояний, прекрасно подходит сеть Кохонена.

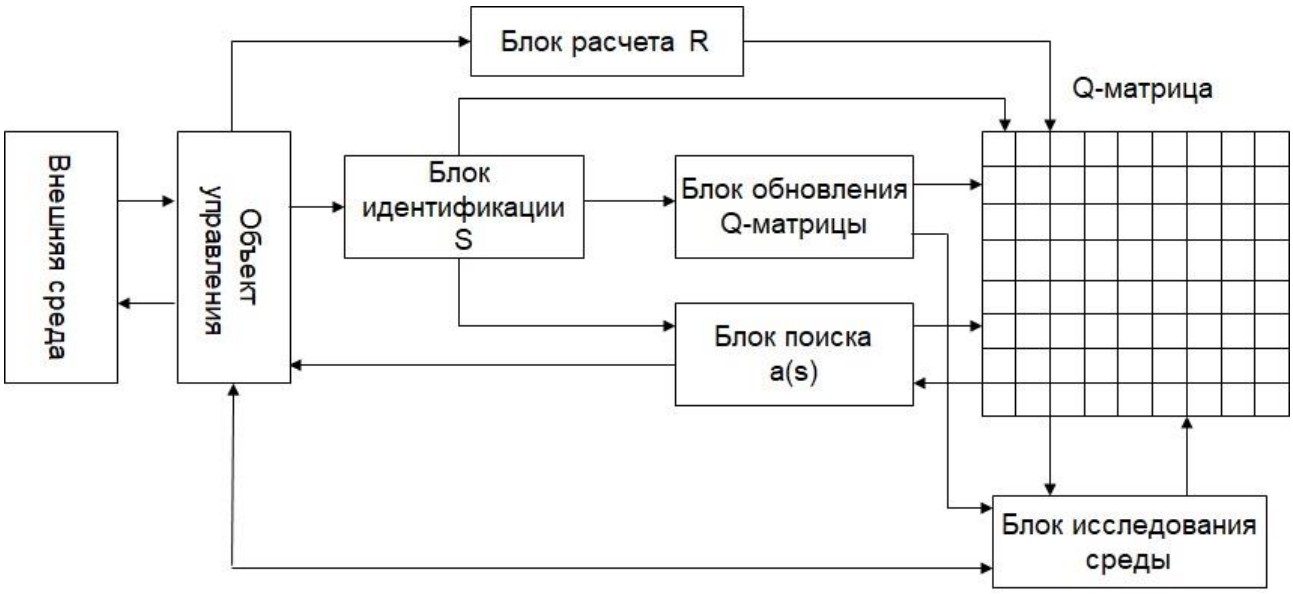


Рисунок 28. Один из вариантов схемы с Q-обучением

Алгоритм SARSA

Рассмотрим простой алгоритм адаптивного критика, и один из первых по публикации – SARSA[21]. Данный алгоритм применяется в том случае если множество возможных ситуаций и действий конечно. В алгоритме SARSA существует следующая цепочка:

$$S(t) \to a(t) \to r(t) \to S(t+1) \to a(t+1)$$

В данном алгоритме пошагово (итеративно) вычисляется оценка величины суммарной награды $Q(S(t), a(t))$, которую получит Агент, если в ситуации $S(t)$ он выполнит действие $a(t)$. Математическое ожидание награды вычисляется по формуле [6]:

$$Q(S(t), a(t)) = E\{r(t) + \gamma \cdot r(t+1) + \gamma^2 \cdot r(t+1) + ...\} \mid S = S(t), a = a(t)$$

Из (3.2) и (3.3) следует:

$$Q(S(t), a(t)) = E\{r(t) + \gamma \cdot Q(S(t+1), a(t+1))\}$$

Ошибка определяется как ОВР:

$$\delta(t) = r(t) + \gamma \div Q(S(t+1), a(t+1)) - Q(S(t), a(t))$$

где δ - есть ошибка временной разности.

На каждой итерации работы системы происходит как выбор действия, так и обучение агента (вернее сказать – переобучение или, в зависимости, от типа

[21] Название SARSA собственно и произошло от сокращения данной цепочки событий (s → a → r → s → a)

нейронной сети - дообучение). Действие в момент времени t выбирается с максимальным значением $Q(S(t), a_j)$ с вероятностью $(1-\xi)$:

$$a(t) = \arg(\max_a \{Q(S(t), a_j)\})$$

где $0 < \xi << 1$. То есть выбор действия происходит с помощью «ξ - жадного правила» .

Переоценка $Q(S, a)$ производится в соответствии с оценкой ошибки $\delta(t)$ - к значению $Q(S(t), a(t))$ добавляется величина, пропорциональная ошибке временной разности:

$$\Delta Q(S(t), a(t)) = \alpha \cdot \delta(t) = \alpha \cdot [r(t) + \gamma \cdot Q(S(t+1), a(t+1)) - Q(S(t), a(t))]$$

где α - параметр скорости обучения.

Приведем полную последовательность действий алгоритма SARSA:

1. Инициализировать произвольно $Q(s, a)$

2. Для каждого эпизода повторять:

2.1. Инициализировать s

2.2. Выбрать a по s, используя стратегию, полученную из Q (можно ε-жадную)

2.3. Повторять для каждого шага эпизода, пока s не станет завершающим состоянием:

2.3.1. Выполнить действие a, найти r, s'.

2.3.2. Найти a' по s', на базе стратегии, полученной из Q (можно ε-жадную)

2.3.3. Рассчитать и обновить значения по формулам:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot Q(s', a' - Q(s, a))]$$

$$s \leftarrow s', \ a \leftarrow a'$$

Так как количество ситуаций и возможных в них действий конечно, то в алгоритме SARSA происходит формирование матрицы $Q(S_j, a_i)$ со всеми возможными ситуациями и действиями. Оценку суммарной награды $Q(S, a)$ можно рассматривать как оценку качества действия $a(t)$ в ситуации $S(t)$

Пример реализации алгоритма SARSA на нейронных сетях показан на слайдах к лекции.

Другими популярными вариантами реализации адаптивных критиков являются:

Q – критик;

V – критик;

В схемах Q – критиков блок критики делает оценку величины суммарной награды $Q(S(t),a(t))$, которую агент ожидает получить в будущем, если он в данной ситуации $S(t)$ выполнит действие определенное действие $a(t)$. Т. е. происходит оценка качества того или иного действия в известной ситуации (аналогично SARSA) [6]. Функционирование Q – критика происходит следующим образом. В момент времени t решатель по вектору текущего состояния $S(t)$ определяет вектор действия $A(t)$, далее выполняется действие $A(t)$, объект управления получает награду $r(t)$. При этом параллельно на Критик подаются векторы $A(t)$ и $S(t)$, по которым Критик производит оценку качества $Q(t)=Q(S(t),A(t))$ действия $A(t)$ в текущей ситуации $S(t)$. После этого объект управления переходит в состояние $S(t+1)$. Момент времени системы инкрементируется $t+1$. Далее все операции повторяются и вычисляется оценка значения $Q(t+1)$, после вычисления которой происходит расчет ошибки временной разности:

$$\delta(t)=r(t)+\gamma\cdot Q(t+1)-Q(t)$$

После накопления определенного количества примеров выполняется обучение нейронных сетей Критика и Решателя, обычно методом обратного распространения ошибки. Корректировку весов синаптических связей Критика и Решателя можно записать следующим образом, согласно:

$$\Delta W_C=\alpha_1\cdot\nabla_{WC}(Q(t))\delta(t)$$

$$\Delta W_A=\alpha_2\sum_K\{\frac{\partial Q(t)}{\partial A_K(t)}\cdot\nabla_{WA}A_K(t)\}$$

В топологиях V – критиков блок Критики делает оценку качества ситуации $V(S(t))$, то есть оценку ожидаемой суммарной награды в данной ситуации. При этом топология дополняется блоком прогноза (см. Рисунок в презентации) и система управления стремится выполнять те действия, которые, согласно прогнозу, приведут к ситуациям $S(t+1)$ с наибольшими оценками $V(S(t+1))$.

Также существуют более сложные топологии адаптивных критиков.

Стоит отметить, что Модель и Критик обычно реализуют на нейронных сетях. В случае нейросетевой реализации параметрами аппроксимирующих функций являются веса синапсов нейронной сети, а оптимизация обычно производится путем подстройки весов, например методом обратного распространения ошибки.

В случае V - критика оценивается ошибка следующей ошибки временной разности:

$$\delta(t)=r(t)+\gamma\cdot V(S^{PR}(t+1))-V(S(t))$$

Критик обучается за счет подстройки весов:

$$\Delta W_C = \alpha \cdot grad_{WC}(V(t)) \cdot \delta(t)$$

В случае реализации Модели на нейронной сети задача прогнозирования решается стандартным образом.

Сети адаптивной критики обладают несомненными достоинствами, такими как:

- старт работы системы, как без знания истории, так и без знания окружающей среды;
- последовательное, пошаговое исследование среды;
- ядро на базе нейронной сети способно аппроксимировать большое пространство состояний – действий.

Но кроме достоинств, сети адаптивной критики обладают и недостатками:

- переобучение нейронных сетей Критика и Агента требуют достаточно продолжительного времени;
- сложность разработки математической модели Критика для реальных задач;

негарантированное поступление подкрепления.

Глубокое обучение с подкреплением

Есть несколько подходов к пониманию «глубокого обучения с подкреплением», в частности: «Глубокое обучение и обучение с подкреплением - это системы, которые обучаются автономно. Разница между ними в том, что глубокое обучение - это обучение на основе обучающего набора с последующим применением этого обучения к новому набору данных, в то время как обучение с подкреплением - это динамическое обучение путем корректировки действий на основе непрерывной обратной связи для максимизации вознаграждения», т. е. «глубокое обучение с подкреплением» есть «что – то» на пересечении этих двух направлений искусственного интеллекта.

Но на самом деле под «глубоким обучением с подкреплением» необходимо понимать использование сверточных нейронных сетей в топологии с подкреплением. Здесь, чтобы соответствовать названию, главное наличие сверточной нейронной сети. Неважно какой блок и какой алгоритм будет с ее помощью реализован: критик, решатель, аппроксиматор Q-таблицы или что-то другое. Но, если серьезно, то для серьезных задач, допустим, для автопилота автомобиля, где требуется а) распознавание дорожной ситуации и б) принятие решения по управлению автомобилей, действительно необходимо использование сложных нейросетевых структур и сверточных («глубоких») нейронных сетей, в частности, потому - что другие методы и алгоритмы либо просто не работают на таких данных, либо дают результат ниже необходимого. В то же время и работа с глубокими нейронными сетями в данных задач находится еще в начале пути.

Можно также дать такую формулировку глубокого обучения с подкреплением: «Глубокое обучение с подкреплением сочетает в себе искусственные нейронные сети со структурой обучения с подкреплением, которая помогает программным агентам узнать, как достичь своих целей. То есть он объединяет аппроксимацию функций и оптимизацию целей, сопоставляя состояния и действия с вознаграждениями, к которым они приводят».

В обучении с подкреплением сверточные сети могут использоваться для распознавания состояния агента, когда ввод является изображением или видеопотоком.

Также в обучении с подкреплением, учитывая изображение, которое представляет состояние, сверточная сеть может ранжировать действия, которые можно выполнить в этом состоянии. Для большей конкретики, Q - функция сопоставляет пары состояние-действие с наибольшим значением (немедленного) вознаграждения со всеми будущими вознаграждениями, которые могут быть получены более поздними действиями по выбранной политике действий объекта управления.

В качестве примера приведем возможные варианты реализации (но лучше сказать - трактовки) глубокого Q-обучения[22].

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Рисунок 29. Пример, расчета Q-функции

Глубокое Q – обучение

Основная идея глубокого Q – обучения (Deep Q-Learning) состоит в том, чтобы адаптировать алгоритм ошибки временной разности таким образом, чтобы обновление для формулы стандартного обновления Q-таблицы будет эквивалентна шагу градиентного спуска при обучении нейронной сети.

На взгляд автора, что такое «deep Q-learning» можно ответить следующей схемой из сети Интернет

[22] Здесь исследования находятся в самом начале пути

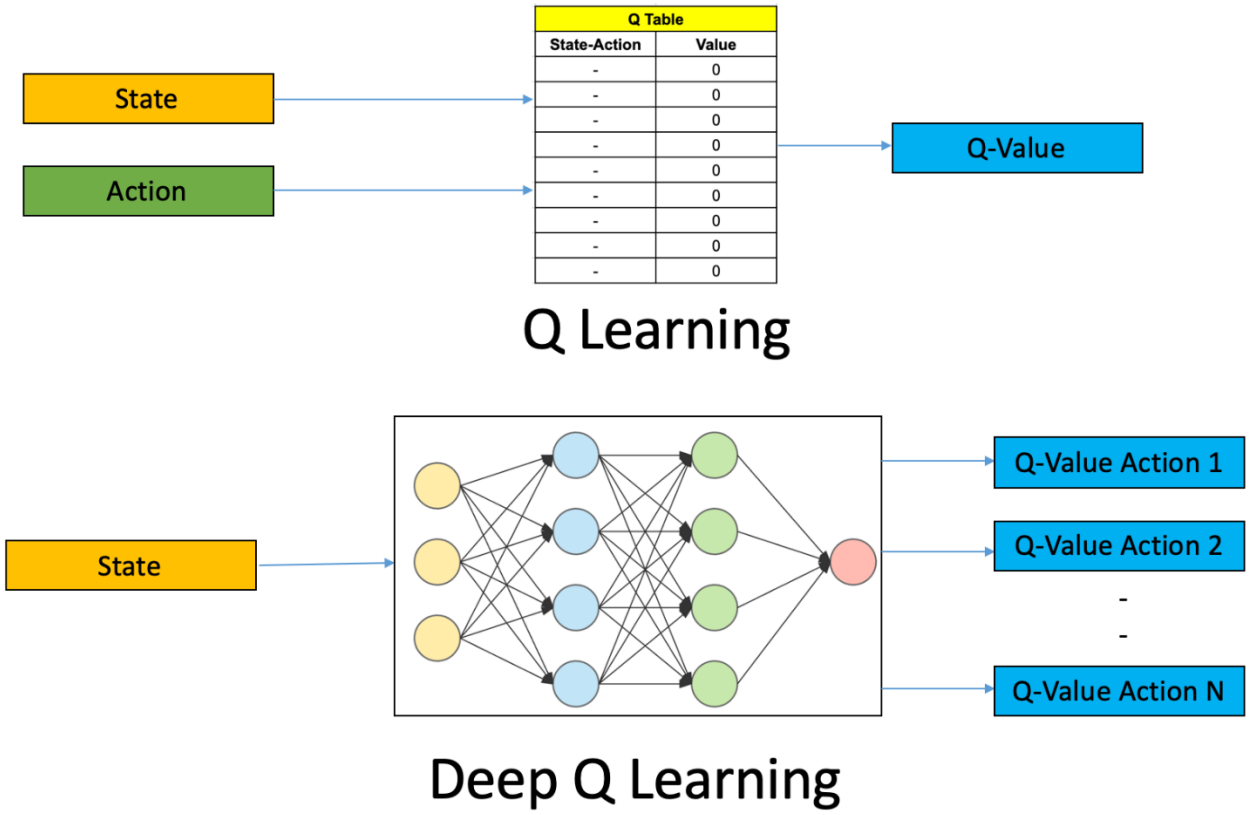


Рисунок 30. Глубокое Q-обучение

Опять же можно сказать, что это «ребрендинг» (но можно сказать, что это не понимание…). Данную схему можно найти, например, в работе В.Кузьмина из Латвии, датированную 2003 годом…

Здесь интересным выглядит момент именно использования сверточных нейронных сетей для «понимания» (обработки) Q – таблиц для равных уровнях детализации задачи. Здесь хорошие результаты показывает алгоритм Double Q-Learning и Noisy DQN (см. слайды к лекции).

ЗАДАНИЕ НА ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

В данном практическом занятии необходимо реализовать совместное применение обучения с подкреплением и сверточных нейронных сетей. Возможные схемы реализации:

1. С помощью сверточной нейронной сети реализовать одну из задач, параметры которой меняются во времени (для примера см. Таблица 5) и включить данную сеть в топологию с подкреплением (варианты см. Таблица 6) и топология с подкреплением должна динамически корректировать работу сверточной нейронной сети.

Таблица 5

№	Задача	Примечание
1	Финансовая торговая система (МТС)	В системе должен быть контуры принятия решений и анализа финансовых временных рядов
2	Медицинская экспертная система	Сверточная нейронная сеть должна оценивать текущее состояние человека, а контур с подкреплением должен корректировать ее работу и рекомендации (возможна реализация в составе экспертной системы)
3	Система анализа продаж	Система должна выдавать сигналы на закупку товаров в зависимости от динамики продаж
4	Система распознавания фигур технического анализа	Система должна выдавать сигналы на совершение торговых операций
5	Система анализа автомобильного трафика	Необходимо оценивать потоки машин в разное время суток и в зависимости от силы потока выставлять длительности сигналов светофоров

Таблица 6

Основные алгоритмы обучения с подкреплением

№	Название
1	Q – learning
2	Fuzzy Q-learning
3	Adaptive Critic Design
4	Bellman reinforcement learning
5	Реализация с помощью метода Монте-Карло

Контрольные вопросы

1. Чем принципиально отличаются Q – критик и V-критик?
2. В чем отличие адаптивных критиков от Q-обучения?
3. Как рассчитывается подкрепление?
4. Как применяются методы Монте-Карло в обучении с подкреплением?
5. Как работает жадное – правило в топологиях обучения с подкреплением?

6. Какие топологии нейронных сетей можно использовать в качестве аппроксиматора Q-таблицы?
7. Сколько выходов может быть у V-критика?

Литература:

1. Prokhorov D., Santiago R., and Wunsch D. "Adaptive Critic Designs: A Case Study For Neurocontrol". Neural Networks, vol. 8, no. 9, pp. 1367-1372, 1995.
2. Sutton R.S., "Learning to Predict by the Methods of Temporal Differences" Machine Learning, vol. 3, pp. 9-44, 1988.
3. Watkins C. Learning from delayed rewards. Ph. D. thesis, Cambridge Univ., Cambridge, England, 1989.
4. Watkins C., Dayan P. Q – learning. // Machine Learning, vol. 8, pp. 279 – 292. 1992.
5. Беллман Р. Динамическое программирование. – М.: Иностранная литература, 1968. – 261 с.
6. Саттон, Барто. Обучение с подкреплением. М.: БИНОМ. Лаборатория знаний. 2012. 400 с.
7. Стасевич В. П., Шумков Е. А., Ключко В. И., Воротников С. А. Адаптивные системы на основе самообучающихся нейросетей // Труды КубГТУ. - 2002. - Вып.2. - С. 192 - 198.
8. Шумков Е.А. "Механические торговые системы" // Сетевой электронный научный журнал КубГТУ. №4, 2017
9. Шумков Е.А., Чистик И.К. Автотрейдер с использованием Q-обучения // "Математические методы и информационные технологии в экономике, социологии и образовании". Сборник статей XXX Международной научно - технической конференции. - Пенза: Приволжский Дом знаний. 2012. С. 126-127.
10. Шумков Е.А. Структуры механических торговых систем // "Прикладная информатика". М: 2012, №3