1. Аннотация

Расчётно-пояснительная записка 52 с., 13 рис., 28 источников.

В курсовом проекте поставлена задача: построение каскадов с помощью рекуррентных нейронных сетей. Для решения данной задачи было проведено исследование технологий нейронных сетей, затем была разработана собственная нейронная сеть.

В ходе выполнения курсовой работы была создана нейронная сеть, набор данных для обучения и тестирования нейронной сети. Написано 284 строк кода на языке Python 3.7.0

2. Оглавление

3. Введение

В современном мире существует огромное количество разновидностей динамических систем с дискретным временем, также называемых каскадом. Биологические популяции, физические эксперименты, измеряемые в разные промежутки времени, видеоряд — и это лишь малая часть примеров каскадов. Современные методы в области машинного обучения позволяют решить многие задачи, связанные с динамическими системами.

С помощью такой модели для машинного обучения, как рекуррентные нейронные сети, возможно создание математической модели каскадов, а также построение необходимого количества итераций каскада.

Отсутствие понимания того, как искусственный интеллект достигает результатов, является одной из причин низкого уровня доверия к современным технологиям искусственного интеллекта. Именно поэтому объяснение работы нейросетей, в частности рекуррентных, является одним из важнейших направлений в российской национальной стратегии развития искусственного интеллекта.[1]

4. Обзор литературы

4.1. Динамические системы

Общее представление динамической системы подробно описано в статье [2]. В статье динамическая система рассматривается как модель для описания и прогнозирования взаимодействия во времени между несколькими компонентами явления, которые рассматриваются как система. В статье указываются следующие компоненты динамической системы:

- Динамический компонент указывает, что время является неотъемлемым элементом модели. В динамических моделях время имеет основополагающее значение как для базовой структуры данных, так и для понимания того, как разворачивается процесс.
- Системный компонент предполагает, что исследовательские вопросы позиционируются с участием нескольких взаимодействующих компонентов большего целого. В контексте динамической системы это означает, что взаимодействующие компоненты ведут себя упорядоченно, следуя правилам, которые могут быть идентифицированы и определены.
- Модельный компонент указывает, что динамические связи между компонентами системы представлены в виде формальных математических уравнений.

В некоторых моделях динамических систем данные организованы по времени как последовательность повторных наблюдений данной переменной во времени, называемая данными временных рядов. В статье [3] рассматриваются модели временных рядов для представления развития системы. Статьи [4], [5] описывают динамические модели, где распределения времени отклика учитываются при формулировании и прогнозировании моделей (например, одновременное моделирование вероятности выбора и времени отклика выбора при прогнозировании выбора).

Описание основных элементов модели динамической системы подробно представлено в статье [6]:

- Состояние системы, которое представляет всю системную информацию в определенный момент времени.
- Пространство состояний системы, которое представляет все возможные состояния системы, которые могут возникнуть
- Функция перехода состояния, которая описывает, как состояние системы изменяется со временем.

Значение и применение дискретных динамических систем описано в [7]. В данном источнике каскад – это динамическая система с дискретным временем - где временная переменная моделируется как дискретная, а временная задержка встроена в систему.

В работе [8] рассматриваются неавтономные динамические системы с дискретным временем. В центре внимания данной работы две формулировки дискретных по времени неавтономных динамических систем:

- Двухпараметрические полугруппы.
- Системы с косыми произведениями.

4.2. Нейронные сети

Основные возможности применения и перспективы развития нейронных сетей описаны в статье [9]. Из основных направлений можно выделить:

- Поиск информации
- Распознавание изображений
- Перевод
- Воспроизведение речи

В статье [10] дана классификация нейронных сетей по структуре, количеству слоев, типу связей, структуре нейрона и т.д. В пункте 6.1 данной курсовой работы дана подробная классификация.

Проблема тренировки нейронной сети и алгоритм обратного распространения ошибки описаны в работе [11]. Основной проблемой тренировки является переобучение. Это проблема возникает, если слишком долго обучать сеть на одних и тех же данных.

В книге [12] раскрыты основные математические принципы, лежащие в основе нейронных сетей и пример нейросети, распознающей написанные от руки цифры.

Источник [13] иллюстрирует графические модели для описания распределения вероятностей, использующих Байесовский метод для распознавания образов и алгоритмы приближенного вывода ситуаций, в которых точные ответы получить невозможно.

Процесс тренировки генеративной сверточной нейронной сети для генерации изображений объектов по типу и цвету, с интерполяцией рядов изображений и заполнением «пустых мест» недостающими элементами разобран в работе [14].

4.3. Рекуррентные нейронные сети (РНС)

В статье [15] представлены результаты аналитического исследования РНС и их обобщающая классификация, выполненная с позиций динамических систем. В работе выделены основные динамические режимы работы РНС, а также определены наиболее перспективные направления в развитии методов обучения РНС с учетом выявленных достоинств и недостатков существующих подходов.

В источнике [16] описаны следующие виды РНС: Long Short-Term Memory – долгая краткосрочная память и Gated Recurrent Unit. Также

выделены преимущества каждой нейросети в соответствии с поставленной задачей.

В статье [17] разобраны способы решения проблемы в облачном центре обработки данных с помощью прогнозирования рабочей нагрузки. Модель прогнозирования рабочей нагрузки разработана с использованием сетей с кратковременной памятью (LSTM). Предложенная модель протестирована на трех эталонных наборах журналов веб-сервера.

Применение рекурсивных рекуррентных нейронных сетей для моделирования процесса декодирования и синтаксического разбора в статистическом машинном переводе предложено в работе [18].

4.4. Обучение нейронных сетей

Общие принципы методов обучения нейронных сетей приведены в статье [19].

В работе [20] предлагается алгоритм для нахождения оптимальной структуры многослойного персептрона, основанный на расчете и минимизации критериев Бартлетта или Мураты-Амари, оценивающих ошибку обобщения для пробных шагов модификации структуры нейросети элементарными структуровоздействующими операциями, лучшая из которых выбирается для применения на текущем шаге или до момента смены поведения критерия.

В статье [21] для определения оптимального размера искусственной нейронной сети в случае отсутствия независимой тестовой выборки имеются несколько индикаторов (NIC-критерий Мураты и Амари, критерии Бартлетта, Баррона), теоретически связывающих прогнозируемый уровень ошибки обобщения с внутренними свойствами обученной нейросети. Подобные

индикаторы позволяют пользователю целенаправленно вести изменения структуры и размера сети (вместо проб методом "тыка"), вводить штрафные функции вторичной оптимизации (наподобие регуляризующих штрафов) для явной минимизации этих критериев. Данная работа развивает это направление, описывает результаты экспериментов для 6 задач классификации с учителем, показывает возможность идентификации момента наступления переобучения при превышении оптимального размера нейросети, показывает возможность определять структурные уровни сложности задач, например, моменты перехода от компетенции малопараметрических моделей (традиционные линейные регрессии, линейные дискриминанты) к компетенции многопараметрических нейромоделей.

В статье [22] исследуется гипотеза о том, что для повышения обобщающих способностей нейросети и ее отказоустойчивости (предотвращения снижения качества решения при повреждении элементов нейросети) необходимо не допускать роста чувствительности решения к изменениям весов синапсов. Оценивать и снижать чувствительность можно разными способами, в работе исследована эффективность повышения отказоустойчивости в процессе обучения нейронной сети путем временного запрещения коррекции наиболее чувствительных синапсов.

Работа [23] дает понятия о том, как оцениваются изменения выходных сигналов нейронной сети при вариациях весов синапсов или значений входных/промежуточных сигналов сети. Критерий качества решения задачи (т.е. сравнение выхода нейросети с требуемым ответом) не используется. Экспериментально показана возможность использования таких оценок для исключения избыточных (малозначимых элементов) из нейросети - даже без дообучения сети удается удалять несколько десятков процентов от имеющегося в сети общего числа синапсов при сохранении точности решения задач классификации.

В статье [24] исследуется действенность одного индикатора эффективности схемы нормализации данных. Показано, что оптимизация предобработки может приводить к увеличению скорости обучения нейросети на порядок и более и к изменению внутренних свойств нейросетей.

В источнике [25] предложена целевая функция для задач нейросетевой нелинейной регрессии, позволяющая задавать допустимую невязку по точности решения каждого примера обучающей выборки и устойчивая к выбросам в данных.

В работе [26] рассматривается, как обратное распространение ошибки можно "распространить" на входные сигналы нейросети для коррекции их значений. Этот прием используется для решения обратных задач с помощью нейросетей, обученных решению прямой задачи. Иначе говоря, если обученная нейронная сеть выдает прогноз на основе значений некоторых показателей (входных сигналов), и этот прогноз отличается от желаемого, то можно потребовать от нейросети так откорректировать входные сигналы, чтобы ответ совпал с требуемым, а затем попытаться привести свойства реальности в соответствие с предлагаемыми сетью значениями входных сигналов.

В статье [27] описан результат применения эмпирической схемы: при высокой чувствительности выходных сигналов обученной нейронной сети к значениям отдельных независимых переменных и невозможности обучения нейросети решению задачи именно на этом малом числе переменных визуализация данных в подпространстве высокочувствительных признаков может помочь увидеть нетипичные примеры-выбросы, которые нейросеть при обучении запомнила. Несколько итераций ручного удаления выбросов, нового обучения нейросети, расчета чувствительностей и визуализации могут повысить обобщающие способности итоговой нейромодели, обученной по очищенной выборке, и приближению

чувствительности решения к изменениям значений признаков к реальной информативности этих признаков.

В источнике [28] показано, что обучение нейросети на основе базового метода обратного распространения ошибки (обучение с постоянным шагом, коррекция синапсов после просмотра каждого очередного примера выборки) при оптимальном выборе длины шага в среднем не уступает по эффективности применению "быстрых" методов градиентной оптимизации наподобие метода сопряженных градиентов, т.е. использующих суммарный по выборке градиент и оптимизацию шага вдоль направления спуска. Проигрыш максимум в 20 раз (для худшей из задач) опровергает утверждение о стабильном проигрыше в несколько порядков на любых задачах. Результаты подтверждают недавно полученные зарубежными авторами выводы о преимуществе обучения с попримерной коррекцией над методами обучения по суммарному градиенту.

5. Динамическая система

5.1. Общее представление

Динамические системы - это раздел математики, посвященный изучению систем, управляемых согласованным набором законов во времени, таких как разностные и дифференциальные уравнения. Акцент динамических систем заключается в понимании геометрических свойств траекторий и долговременного поведения. За последние 40 лет, с открытием хаоса и странных аттракторов, теория динамических систем приобрела значительный интерес, и было обнаружено, что она имеет связи со многими различными областями математики (такими как теория чисел и топология) и науки. Динамические системы могут моделировать невероятный диапазон поведения, такого как движение планет в солнечных системах, распространение болезней среди населения, форма и рост растений, взаимодействие оптических импульсов или процессы, которые регулируют электронные цепи и сердцебиение.

Чтобы создать динамическую систему, нужно решить что будет развиваться со временем и какое правило определяет, как будет проходить развитие со временем. Таким образом, динамическая система - это модель, описывающая временную эволюцию системы.

Первым шагом в создании динамической системы является определение того, что является основой, которое будет развиваться со временем. Чтобы сделать это, нужно создать набор переменных, которые дают полное описание системы в любой конкретный момент времени.

Под «полным описанием» не обязательно подразумевается, что переменные будут полностью описывать реальную моделируемую систему. Но переменные должны полностью описывать состояние математической системы. В динамической системе, если значения этих

переменных в определенное время известны, то известно все о состоянии системы в это время. Чтобы смоделировать какую-либо реальную систему, разработчик должен выбрать, какие переменные будут формировать полное описание математической модели.

Переменные, которые полностью описывают состояние динамической системы, называются переменными состояния . Множество всех возможных значений переменных состояния является пространством состояний.

Пространство состояний может быть дискретным и состоять из изолированных точек, например, если переменные состояния могут принимать только целые значения. Он может быть непрерывным, состоящим из гладкого набора точек, например, если переменные состояния могут принимать любое действительное значение. В случае, когда пространство состояний является непрерывным и конечномерным, его часто называют фазовым пространством, а число переменных состояния является размерностью динамической системы. Пространство состояний также может быть бесконечномерным.

Вторым шагом в создании динамической системы является определение правила эволюции динамической системы во времени. Это правило должно быть определено, чтобы переменные состояния были полным описанием состояния системы в следующем смысле: значение переменных состояния в конкретный момент времени должно полностью определять эволюцию всех будущих состояний. Если эволюция во времени зависит от переменной, не включенной в пространство состояний, то правило, объединенное с пространством состояний, не определяет динамическую систему. Нужно либо изменить правило, либо дополнить пространство состояний необходимыми переменными, чтобы сформировать динамическую систему.

5.2. Дискретные динамические системы

Динамические системы, в которых состояние системы развивается с дискретными временными шагами, называются дискретными динамическими системами или каскадами.

При моделировании системы как каскада, делается состояние системы в определенной последовательности. Состояния могут появляться один раз в год, раз в миллисекунду или даже нерегулярно.

Идея заключается в том, что записывается любая переменная, определяющая состояние системы: выбранные переменные состояния, которые эволюционируют через пространство состояний. Чтобы завершить описание динамической системы, нужно указать правило, которое определяет, с учетом начального состояния, какой должна быть результирующая последовательность будущих состояний.

5.3. Формальное представление каскадов

В соответствии с вышесказанным классическая форма каскада:

$$S^{(t)} = f\left(S^{(t-1)}; \theta\right) \text{ (1)},$$

где $S^{(t)}$ – состояние системы в момент времени t, θ – параметр динамической системы.

Выражение (1) рекуррентное, потому что состояние S в момент времени t ссылается на состояние в момент времени $t-1$

Если каскад имеет конечное число временных шагов τ, то его можно представить в виде графа, применив это определение $\tau - 1$ раз. Так если развернуть выражение (1) для τ = 3 шагов:

$$s^{(3)} = f\left(s^{(2)}; \theta\right) = f(f\left(s^{(1)}; \theta\right); \theta) \text{ (2)}.$$

После такой развертки путем повторного применения определения получается выражение, не содержащее рекурсии. Такое выражение можно представить традиционным ациклическим ориентированным графом вычислений. Развернутый граф вычислений выражения (2) показан на рис. 1.

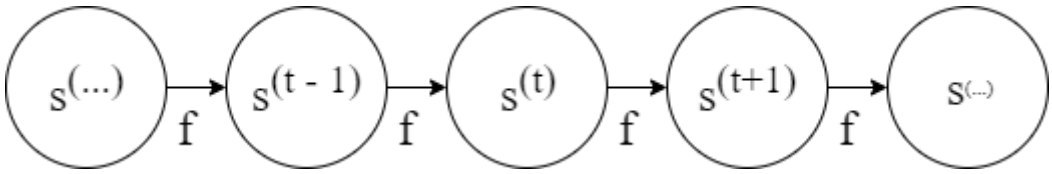

Рис. 1. Классическая динамическая система, описываемая выражением (1), в виде развернутого графа вычислений. Каждая вершина представляет состояние в некоторый момент t, а функция f отображает состояние в момент t на состояние в момент $t + 1$. Одни и те же параметры (значение θ, параметризующее f) используется на всех временных шагах.

6. Нейронные сети

6.1. Общее представление

Нейронная сеть — это последовательность нейронов, соединенных между собой синапсами. Структура нейронной сети пришла в мир программирования прямиком из биологии. Благодаря такой структуре, машина обретает способность анализировать и даже запоминать различную информацию. Нейронные сети также способны не только анализировать входящую информацию, но и воспроизводить ее из своей памяти. Другими словами, нейросеть это машинная интерпретация мозга человека, в котором находятся миллионы нейронов передающих информацию в виде электрических импульсов.

Нейронные сети классифицируются по:

1. Типу входной информации:
    - Аналоговые нейронные сети (используют информацию в форме действительных чисел)
    - Двоичные нейронные сети (оперируют с информацией, представленной в двоичном виде)
    - Образные нейронные сети (оперируют с информацией, представленной в виде образов: знаков, иероглифов, символов)
2. Характеру обучений:
    - С учителем (выходное пространство решений нейронной сети известно)
    - Без учителя (нейронная сеть формирует выходное пространство решений только на основе входных воздействий. Такие сети называют самоорганизующемися
    - С подкреплением (система назначения штрафов и поощрений среды
3. Характеру настройки синапсов:

- Сети с фиксированными связями (весовые коэффициенты нейронной сети выбираются сразу, исходя из условий задачи, при этом $\frac{dW}{dt} = 0$, где W – весовые коэффициенты сети
- Сети с динамическими связями (для них в процессе обучения происходит настройка синаптических связей, то есть $\frac{dW}{dt} \neq 0$, где W – весовые коэффициенты сети

4. Времени передачи сигнала. В ряде нейронных сетей активирующая функция может зависеть не только от весовых коэффициентов связей, но и от времени передачи импульса (сигнала) по каналам связи
5. Характеру связей:
    - Сети прямого распространения (все связи направлены строго от входных нейронов к выходным)
    - Рекуррентные нейронные сети (сигнал с выходных нейронов скрытого слоя частично передается обратно на входы нейронов входного слоя (обратная свзя).
    - Радиально-базисные функции (сети, использующие в качестве активационных функций радиально-базисные
    - Самоорганизующиеся карты (соревновательные сети с обучением без учителя, выполняющие задачу визуальзации и кластеризации)

Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной, скрытый и выходной. В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Соответственно, есть входной слой, который получает информацию, n скрытых слоев (обычно их не больше 3), которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные данные (input) и выходные данные (output). В случае входного нейрона: input=output. В остальных, в поле input попадает суммарная

информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации и попадает в поле output.

Синапс это связь между двумя нейронами. У синапсов есть 1 параметр — вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. У того нейрона, у которого вес будет больше, та информация и будет доминирующей в следующем нейроне (пример — смешение цветов). На самом деле, совокупность весов нейронной сети или матрица весов — это своеобразный мозг всей системы. Именно благодаря этим весам, входная информация обрабатывается и превращается в результат.

Функция активации — это способ нормализации входных данных. Функций активации достаточно много: линейная функция, сигмоид, гиперболический тангенс и т.д.

Общий вид нейронной сети указан на рис.2.

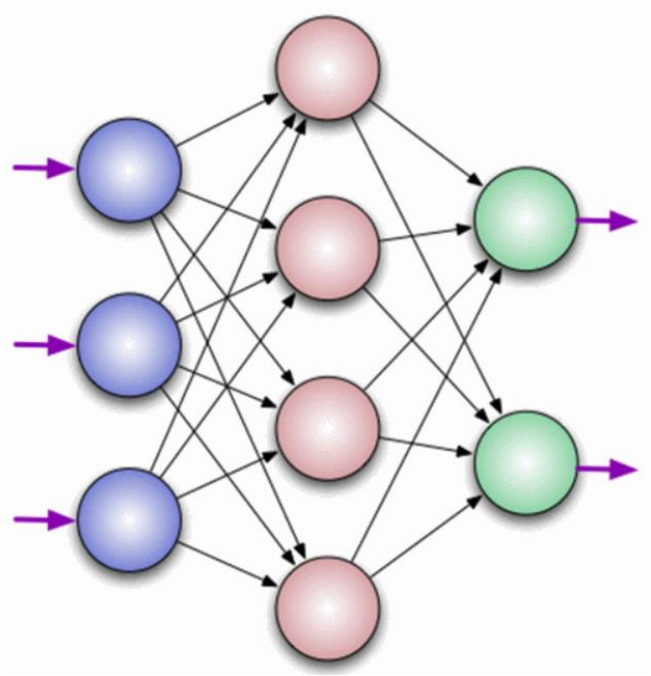

Рис.2. Модель нейронной сети, где синем цветом указаны входные нейроны, скрытые – красным, выходные – зеленым. Черные стрелки – синапсы, фиолетовые – входные и выходные данные.

6.2. Выбор нейронной сети для нахождения итераций каскада

Рассмотрим динамическую систему, управляемую внешним сигналом $x^{(t)}$:

$$S^{(t)} = f\left(S^{(t-1)}, x^{(t)}; \theta\right). \quad (3)$$

Теперь состояние содержит информацию обо всей прошлой последовательности.

Проанализировав виды нейронных сетей и выражение (3), можно сделать вывод о необходимости использования рекуррентной нейронной сети.

7. Рекуррентные нейронные сети (РНС)

7.1. Общее представление

Рекуррентные нейронные сети можно строить разными способами. Как почти любую функцию можно рассматривать как нейронную сеть прямого распространения, так и практически любую рекуррентную функцию можно рассматривать как РНС. Чтобы подчеркнуть, что состояние динамической системы – это на самом деле скрытые блоки сеть, перепишем уравнение (3), используя для представления состояния переменную h:

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}; \theta\right). \quad (4)$$

РНС удовлетворяющая выражению (4) показана на рис. 3.

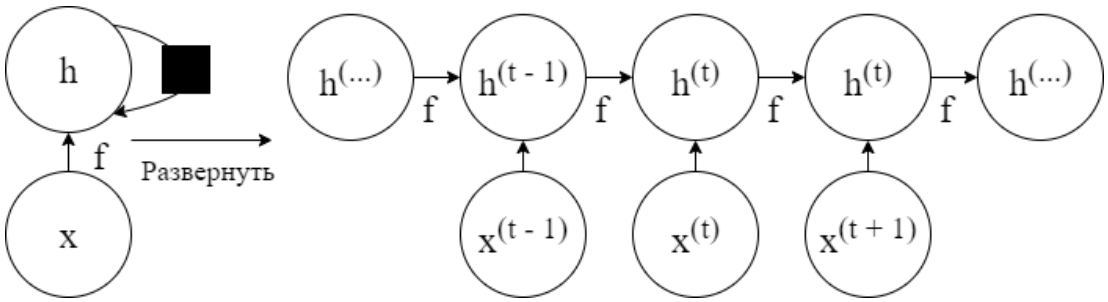

Рис. 3. Рекуррентная сеть без выходов. Эта сеть обрабатывает информацию из входа x, включая ее в состояние h, которое передается дальше во времени. (Слева) Принципиальная схема. Черный квадрат обозначает задержку на один временно шаг. (Справа) Та же сеть в виде развернутого графа вычислений, в котором каждая вершина ассоциирована с одним моментом времени

Когда РНС обучают решать задачу, в которой требуется предсказывать будущее по прошлому, сеть обычно обучается использовать $h^{(t)}$ как сводку относящихся к задаче аспектов последовательности входных данных, предшествующей моменту времени t. В общем случае в сводке по

необходимости утрачивается часть информации, потому что она отображает последовательность произвольной длины $\left(x^{(t)}, x^{(t-1)}, \ldots, x^{(1)}\right)$ на вектора фиксированной длины $h^{(t)}$. В зависимости от критерия обучения некоторые аспекты прошлой последовательности могут запоминаться в сводке с большой точностью, чем остальные. Например, если РРНС используется для статистического моделирования языка, как правило, для предсказания следующего слова по известным предыдущим, то достаточно сохранить только информацию, необходимую для предсказания остатка предложения.

Уравнение (4) можно изобразить двумя способами. Первый способ – нарисовать диаграмму, содержащую по одному узлу для каждой компоненты, которая могла бы существовать в физической реализации модели, например в биологической нейронной сети. В этом случае сеть определяет схему, которая содержит физические детали и работает в режиме реального времени, а ее текущее состояние может оказывать влияние на будущее; этот вариант изображен в левой части рис. 3.

Черный квадрат на принципиальной схеме сети означает, что взаимодействие имеет место с задержкой на один временной шаг, т. е. происходит переход из состояния в момент t в состояние в момент $t + 1$. Другой способ изобразить РНС – нарисовать развернутый граф вычислений, в котором каждая компонента представлена многими переменными состояния, по одной на временной шаг. Каждая переменная на каждом временном шаге изображается в виде отдельной вершины графа вычислений, как в правой части рис. 3. Развертка - это операция, которая отображает принципиальную схему в граф вычислений с повторяющимися частями. Размер развернутого графа зависит от длины последовательности.

Можно представить развернутое рекуррентное выражение после t шагов функции $h^{(t)} = g^{(t)}\left(x^{(t)}, x^{(t-1)}, \dots, x^{(1)}\right) = f\left(h^{(t-1)}, x^{(t)}; \theta\right)$

Функция $g^{(t)}$принимает на входе всю прошлую последовательность $\left(x^{(t)}, x^{(t-1)}, \dots, x^{(1)}\right)$ и порождает текущее состояние, но развернутая рекуррентная структура позволяет представить $g^{(t)}$ в виде многократного применения функции f. Таким образом процесс развертки дает два важных преимущества:

1. Независимо от длины последовательности размер входа обученной модели всегда один и тот же поскольку, он описывается в терминах перехода из одного состояния в другое, а не в терминах истории состояний переменной длины.
2. Одну и ту же функцию перехода f с одними и теми же параметрами можно использовать на каждом шаге

Эти два фактора позволяют обучить одну модель f, которая действует на всех временных шагах и для последовательностей любой длины, не прибегая к обучению отдельных моделей $g^{(t)}$ для каждого временного шага. Обучение единственной разделяемой модели открывает возможность обобщения на такие длины последовательности, которые не встречались в обучающем наборе, и позволяет оценивать модель при наличии гораздо меньшего числа обучающих примеров, чем понадобилось бы без разделения параметров.

8. Оптимальное обучение рекуррентных нейронных сетей с целью построения итераций каскада

8.1. Виды рекуррентных нейронных сетей

РНС можно классифицировать по типу связи между входными и выходными данными:

1. Один к одному (по сути является обычной нейронной сетью)

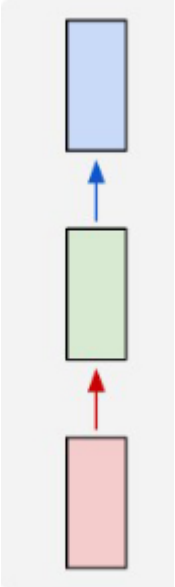

Рис. 4. РНН с типом связи один к одному

2. Один ко многим (один вход ко многим выходам)

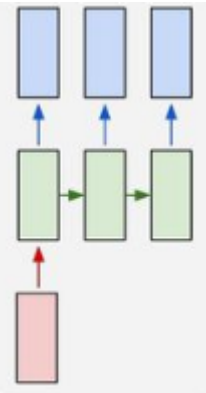

Рис. 5. РНН с типом связи один ко многим

3. Многие к одному (много входов к одному выходу)

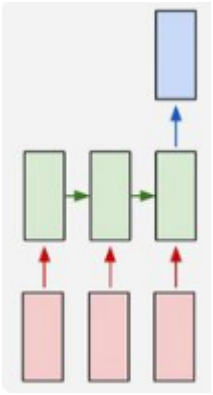

Рис. 6. РНН с типом связи многие к одному

4. Многие ко многим (много входов ко многим выходам)

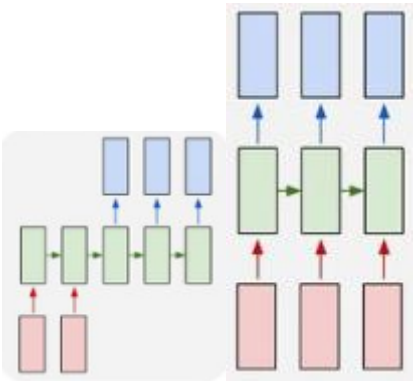

Рис. 7. РНН с типом связи многие ко многим

РНС также можно классифицировать по внутреннему устройтсву взаимодействия:

1. Обычные РНС это сети типа FFNN(Feed Forward Neural networks) , но с особенностью: нейроны получают информацию не только от предыдущего слоя, но и от самих себя предыдущего прохода. Это означает, что порядок, в котором вы подаёте данные и обучаете сеть, становится важным. Большой сложностью сетей RNN является проблема исчезающего (или взрывного) градиента, которая заключается в быстрой потере информации с течением времени. Взрывные градиенты – это проблема, при которой накапливаются большие градиенты ошибок, что приводит к очень большим обновлениям весов модели нейронной сети во время обучения. Это приводит к тому, что модель нестабильна и неспособна учиться на основе тренировочных данных.
2. Сети с долгой краткосрочной памятью (long short term memory, LSTM) стараются решить вышеупомянутую проблему потери информации, используя фильтры и явно заданную клетку памяти. У каждого нейрона есть клетка памяти и три фильтра: входной, выходной и забывающий. Целью этих фильтров является защита информации. Входной фильтр определяет, сколько информации из предыдущего слоя будет храниться в клетке. Выходной фильтр определяет, сколько информации получат следующие слои.
3. Управляемые рекуррентные нейроны (gated recurrent units, GRU) — это небольшая вариация предыдущей сети. У них на один фильтр меньше, и связи реализованы иначе. Фильтр обновления определяет, сколько информации останется от прошлого состояния и сколько будет взято из предыдущего слоя.

4. Нейронные машины Тьюринга (neural Turing machines, NTM) можно рассматривать как абстрактную модель LSTM и попытку показать, что на самом деле происходит внутри нейронной сети. Ячейка памяти не помещена в нейрон, а размещена отдельно с целью объединить эффективность обычного хранилища данных и мощь нейронной сети. Собственно, поэтому такие сети и называются машинами Тьюринга — в силу способности читать и записывать данные и менять состояние в зависимости от прочитанного они являются тьюринг-полными.
5. Двунаправленные RNN, LSTM и GRU (bidirectional recurrent neural networks, bidirectional long / short term memory networks и bidirectional gated recurrent units, BiRNN, BiLSTM и BiGRU) не показаны в таблице, поскольку они ничем не отличаются от своих однонаправленных вариантов. Разница заключается в том, что эти сети используют не только данные из “прошлого”, но и из “будущего”. Например, обычную сеть типа LSTM обучают угадывать слово “рыба”, подавая буквы по одной, а двунаправленную — подавая ещё и следующую букву из последовательности. Такие сети способны, например, не только расширять изображение по краям, но и заполнять дыры внутри.

8.2. Виды рекуррентных нейронных сетей

Из разнообразия РНС, можно сделать вывод, что разные РНС решают разные задачи. Чтобы выбрать вид РНС необходимо установить условие задачи.

Пусть в качестве каскада будет видеоряд, тогда задача будет звучать так: по n кадрам построить видео из n кадров, где выходной кадр момента

времени t должен совпадать со входным кадром момента времени $t + 1$, а выходной кадр для последнего временного состояния, должен не нарушать общую тенденцию каскада.

8.3. Выбор РНС в соответствии с поставленной задачей

Представим поставленную задачу в виде развернутого графа:

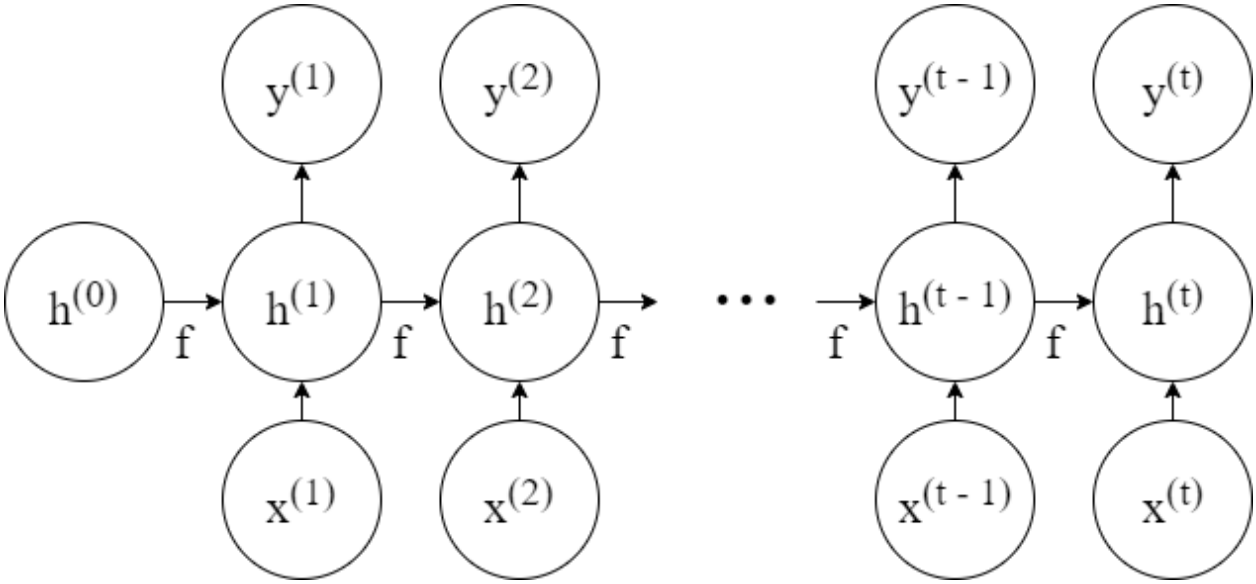

Рис. 8. Развернутый граф в соответствии с поставленной задачей, где $x^{(i)}$ - i-ый входной кадр, а $y^{(i)}$ - i-ый выходной кадр, $i \in [1, \dots, t]$

Данный граф имеет много входов и много выходов, что соответствует РНС с типом связи многие ко многим. В качестве типа РНС выберем обычную РНС, т.к. в реалиях данной задачи проблемы исчезающего градиента не возникнет.

8.4. Построение РНС

РНС с типом связи многие ко многим:

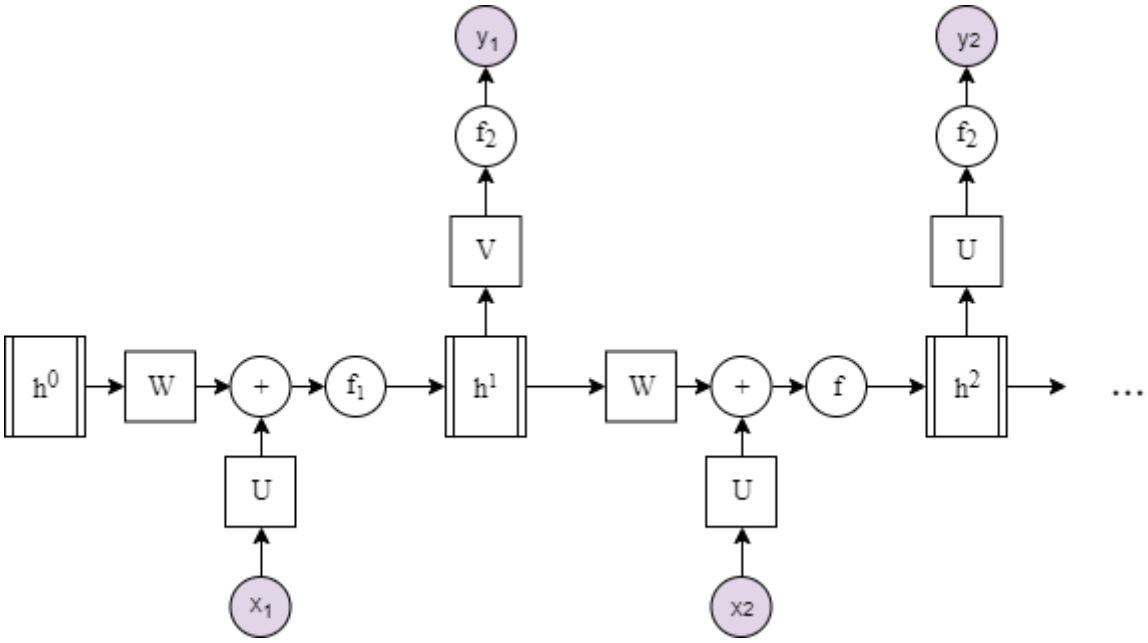

Рис. 8. РНС с типом связи многие ко многим, где h^i - состояние системы, x_i – входные параметры, f_1 – функция активации для скрытых блоков, f_2 – функция выхода, W, U, V – матрицы весов,

Уравнения прямого распространения для РНС, изображенной на рис. 8.:

$$a^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b \quad (5)$$

$$h^{(t)} = f_1(a^{(t)}) \quad (6)$$

$$o^{(t)} = Vh^{(t)} + c \quad (7)$$

$$y^{(t)} = f_2(o^{(t)}) \quad (8),$$

где b и c – вектора сдвига.

Вектора сдвига необходимы, чтобы была возможность сдвигать функцию активации на какое-то пороговое значение.

Чтобы выбрать f_1 и f_2 необходимо определить класс задачи, которую необходимо решить.

Пусть заранее известны все возможные состояния динамической системы, тогда при переходе на следующий временной шаг, необходимо выбрать одно состояние из множества состояний. Такая задача называется: задачей классификации. По своим свойствам сигмоид (см. рис 9) и гиперболический тангенс (см. рис. 10) отлично подходят для данной задачи.

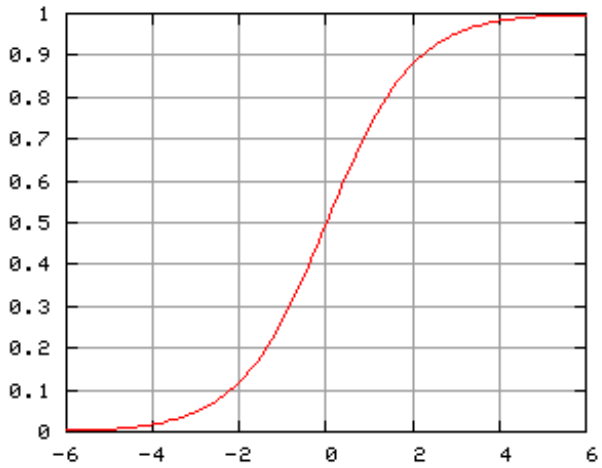

Рис. 9. График функции $\frac{1}{1+e^{-x}}$ – сигмоид

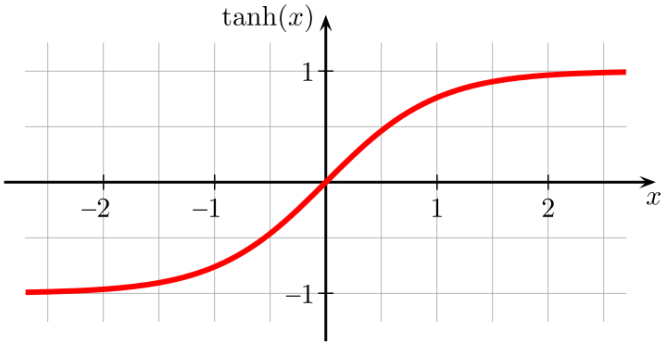

Рис. 10. График функции $\frac{2}{1+e^{-2x}} - tanh(x)$

Т.к. гиперболический тангенс центрирован относительно 0, то с ним легче работать: $f_1 = tanh(x)$

Задача функции f_2 – выдавать индекс класса, который подходит для исходных входных данных.

8.4.1. Принцип максимального правдоподобия

Принцип максимального правдоподобия используется для выявления общего принципа, позволяющего выводить функции, являющиеся хорошими оценками для разных моделей.

Пусть дано множество m примеров $X = \left(x^{(1)}, \dots, x^{(m)}\right)$, независимо выбираемых из неизвестного распределения $P_{data}(X)$. Тогда $P_{model}(X;\ \theta)$ - параметрическое семейство распределений вероятности над одним и тем же пространством, индексированное параметром θ. Иными словами, $P_{model}(X;\ \theta)$ отображает произвольную конфигурацию x на вещественное число, дающее оценку истинной вероятности $P_{data}(x)$.

Тогда оценка максимального правдоподобия для θ определяется формулой:

$$\theta_{ML} = arg\ \max_{\theta} P_{model}(X;\ \theta) = arg\ \max_{\theta} \prod_{i=1}^{m} P_{model}\left(x^{(i)};\ \theta\right). \quad (9)$$

Такое произведение большого числа вероятностей по ряду причин может быть неудобно. Например, оно подвержено потере значимости. Для получения эквивалентной, но более удобной задачи оптимизации заметим, что взятие логарифма правдоподобия не изменяет $arg\ max$, но преобразует произведение в сумму:

$$\theta_{ML} = arg\ \max_{\theta} \sum_{i=1}^{m} log\ P_{model}\left(x^{(i)};\ \theta\right). \quad (10)$$

Поскольку $arg\ max$ не изменяется при умножении функции стоимости на константу, можно разделить правую часть на m и получить выражение в

виде математического ожидания относительно эмпирического распределения $\hat{P}_{data}$, определяемого обучающими данными:

$$\theta_{ML} = \arg\max_{\theta} E_{x\sim\hat{p}_{data}} \log P_{model}(X;\ \theta). \quad (11)$$

Один из способов интерпретации оценки максимального правдоподобия состоит в том, чтобы рассматривать ее как минимизацию расхождения Кульбака–Лейблера между эмпирическим распределением $\hat{p}_{data}$, определяемым обучающим набором, и модельным распределением. Расхождение Кульбака–Лейблера определяется формулой:

$$D_{kL}(\hat{p}_{data}||P_{model}) = E_{x\sim\hat{p}_{data}}[\log\hat{p}_{data}(x) - \log P_{model}(x)]. \quad (12)$$

Первый член разности в квадратных скобках зависит только от порождающего данные процесса, но не от модели. Следовательно, при обучении модели, минимизирующей расхождение КЛ, нужно минимизировать только величину $-E_{x\sim\hat{p}_{data}}[\log P_{model}(x)]$, что тоже самое, что максимизация (11).

Минимизация расхождения КЛ в точности соответствует минимизации перекрестной энтропии между распределениями.

Таким образом, максимальное правдоподобие – это попытка совместить модельное распределение с эмпирическим распределением $\hat{p}_{data}$.

Хотя оптимальное значение θ не зависит от того, максимизируется правдоподобие или минимизируем расхождение КЛ, значения целевых функций различны. При разработке программ часто называется то и другое минимизацией функции стоимости. В таком случае поиск максимального правдоподобия становится задачей минимизации отрицательного логарифмического правдоподобия (ОЛП), или, что эквивалентно, минимизации перекрестной энтропии. Взгляд на максимальное правдоподобие как на минимальное расхождение КЛ в этом случае

становится полезен, потому что известно, что минимум расхождения КЛ равен нулю. А отрицательное логарифмическое правдоподобие может принимать отрицательные значения при вещественных x.

Так как принцип максимального правдоподобия использует вероятности, то сначала их необходимо получить с помощью функции *softmax*.

8.4.2. Softmax

Если требуется представить распределение вероятности дискретной величины, принимающей n значений, то можно воспользоваться функцией *softmax*. Ее можно рассматривать как обобщение сигмоиды.

Функция *softmax* чаще всего используется как выход классификатора для представления распределения вероятности n классов. Реже функция *softmax* используется внутри самой модели, чтобы модель выбрала один из n вариантов какой-то внутренней переменной.

В случае бинарных величин требуется породить одно число:

$$\hat{y} = P(y = 1|x). \quad (13)$$

Поскольку это число должно было находиться между 0 и 1 и поскольку его логарифм должен быть достаточно хорош при градиентной оптимизации логарифмического правдоподобия, можно вместо него порождать число $z = log\ \tilde{P}(y = 1|x)$. Потенцирование и нормировка дают распределение Бернулли, управляемое сигмоидной функцией.

Чтобы обобщить это на случай дискретной случайной величины с n значениями, нужно породить вектор $\hat{y}$, для которого $\hat{y}_i = P(y = i|x)$. Требуется не только, чтобы каждый элемент $\hat{y}_i$ находился между 0 и 1, но и чтобы сумма всех элементов была равна 1 и таким образом представляла корректное распределение вероятности. Подход, который работает для

распределения Бернулли, обобщается и на категориальное распределение. Сначала линейный слой предсказывает ненормированные логарифмы вероятностей:

$$z = W^T h + b, \quad (14)$$

где $z_i = log\tilde{P}(y = i|x)$. Функция *softmax* может затем потенцировать и нормировать z для получения желаемого $\hat{y}$. Формально функция *softmax* определяется следующем образом:

$$softmax(z)_i = \frac{exp(z_i)}{\Sigma_j exp(z_j)}. \quad (15)$$

Как и в случае логистической сигмоиды, использование функции exp дает хорошие результаты при обучении softmax с целью порождения выходного значения *y* с применением логарифмического правдоподобия. В этом случае нужно максимизировать $log\ P(y = i, z) = logsoftmax(z)$. Определение *softmax* через exp естественно, потому что логарифм, входящий в логарифмическое правдоподобие, компенсирует потенцирование в *softmax*

$$logsoftmax(z)_i = \ z_i - log\Sigma_j\ exp\big(z_j\big). \quad (16)$$

Первый член в выражении (16) показывает, что вход z_i дает прямой вклад в функцию стоимости. Поскольку этот член не испытывает насыщения, то обучение всегда может продолжиться, даже если вклад z_i во второй член становится очень мал. При максимизации логарифмического правдоподобия первый член поощряет увеличение z_i, а второй – уменьшение всех элементов z. Заметим, что второй член $log\Sigma_j\ exp\big(z_j\big)$ можно грубо аппроксимировать величиной $\max_j z_j$. В основе такой аппроксимации лежит то соображение, что $\exp(z_k)$ несущественно для любого z_k, значительно меньшего, чем $\max_j z_j$. Отсюда следует, что отрицательное логарифмическое правдоподобие в роли функции стоимости всегда сильнее штрафует самое активное неправильное

предсказание. Если правильный ответ уже дает самого большого вклада в *softmax*, то члены $-z_i$ и $log\Sigma_j\, exp(z_j) \approx \max_j z_j = z_i$

приблизительно взаимно уничтожаются. Такой пример, следовательно, даст малый вклад в общую стоимость обучения, в которой будут преобладать другие примеры, пока еще классифицированные неправильно.

Поскольку максимальное правдоподобие – состоятельная оценка, это гарантированно произойдет при условии, что модельное семейство способно представить обучающее распределение. На практике из-за ограниченной емкости и несовершенной оптимизации модель способна только аппроксимировать эти доли.

Многие целевые функции, отличные от логарифмического правдоподобия, не так хорошо сочетаются с функцией *softmax*. Конкретно, целевые функции, в которых не используется логарифм для компенсации функции exp, входящей в *softmax*, не обучаются, когда аргумент exp становится отрицательным и большим по абсолютной величине, что приводит к обнулению градиента. В частности, среднеквадратическая ошибка – плохая функция потерь для блоков *softmax*, она не всегда побуждает модель изменить свой выход, даже если модель весьма уверенно дает неправильные предсказания. Чтобы понять, в чем тут дело, необходимо внимательно рассмотреть саму функцию softmax.

Как и сигмоида, функция активации *softmax* склонна к насыщению. У сигмоиды всего один выход, и она насыщается, когда абсолютная величина аргумента очень велика. У *softmax* выходных значений несколько. Они насыщаются, когда велика абсолютная величина разностей между входными значениями. Когда *softmax* насыщается, многие основанные на ней функции стоимости также насыщаются, если только они не способны обратить насыщающуюся функцию активации.

Чтобы понять, как *softmax* реагирует на разность между входными значениями, необходимо сказать, что выход *softmax* инвариантен относительно прибавления одного и того же скаляра ко всем входам:

$$softmax(z) = softmax(z + c). \quad (17)$$

Пользуясь этим свойством, можно вывести численно устойчивый вариант *softmax*:

$$softmax(z) = softmax(z - max_i z_i). \quad (18)$$

Этот новый вариант позволяет вычислять softmax с малыми численными погрешностями, даже когда z содержит очень большие по абсолютной величине элементы. Изучив численно устойчивый вариант, видно, что на функцию softmax оказывает влияние отклонение ее аргументов от $max_i z_i$.

Значение $softmax(z)_i$ асимптотически приближается к 1, когда соответствующий аргумент максимален ($z_i = max_i z_i$) и z_i много меньше остальных входов. Значение $softmax(z)_i$ может также приближаться к 0, когда z_i не максимально, а максимум намного больше. Это обобщение того способа, которым достигается насыщение сигмоидных блоков, и оно может стать причиной аналогичных трудностей при обучении, если функция потерь не компенсирует насыщения.

Аргумент z функции *softmax* можно породить двумя разными способами. Самый распространенный – просто заставить предыдущий слой нейронной сети выводить каждый элемент z, как описано выше на примере линейного слоя (14). При всей своей простоте этот подход означает, что у распределения избыточное количество параметров. Ограничение, согласно которому сумма n выходов должна быть равна 1, означает, что необходимо только $n - 1$ параметров; вероятность n-го значения можно получить путем

вычитания суммы первых n – 1 вероятностей из 1. Таким образом, один элемент z можно зафиксировать, например потребовать, чтобы z_n= 0. Но это в точности то, что делает сигмоидный блок. Определение $P(y = 1 \mid x) = \sigma(z)$ эквивалентно определению $P(y = 1 \mid x) = softmax(z)_1$ в случае двумерного z и $z_1 = 0$. Оба подхода к *softmax* – с n – 1 и с n аргументами – описывают одно и то же множество распределений вероятности, но обладают разной динамикой обучения. На практике редко возникает существенное различие между перепараметризованным и ограниченным вариантом, а перепараметризованный реализовать проще.

Название «*softmax*» вносит некоторую путаницу. Эта функция ближе к argmax, чем к max. Часть «*soft*» связана с тем, что функция *softmax* непрерывная и дифференцируемая. Функция же argmax, результат которой представлен унитарным вектором, не является ни непрерывной, ни дифференцируемой. Следовательно, *softmax* – это «сглаженный» вариант argmax. Соответствующий сглаженный вариант функции max - $softmax(z)^T z$.

8.5. Построение и обучение РНС

В соответствии с вышеизложенным $f_2 = softmax$. Функция потерь (L), необходимая для обучения – отрицательное логарифмическое правдоподобие. Представим полученную РНС в форму графа:

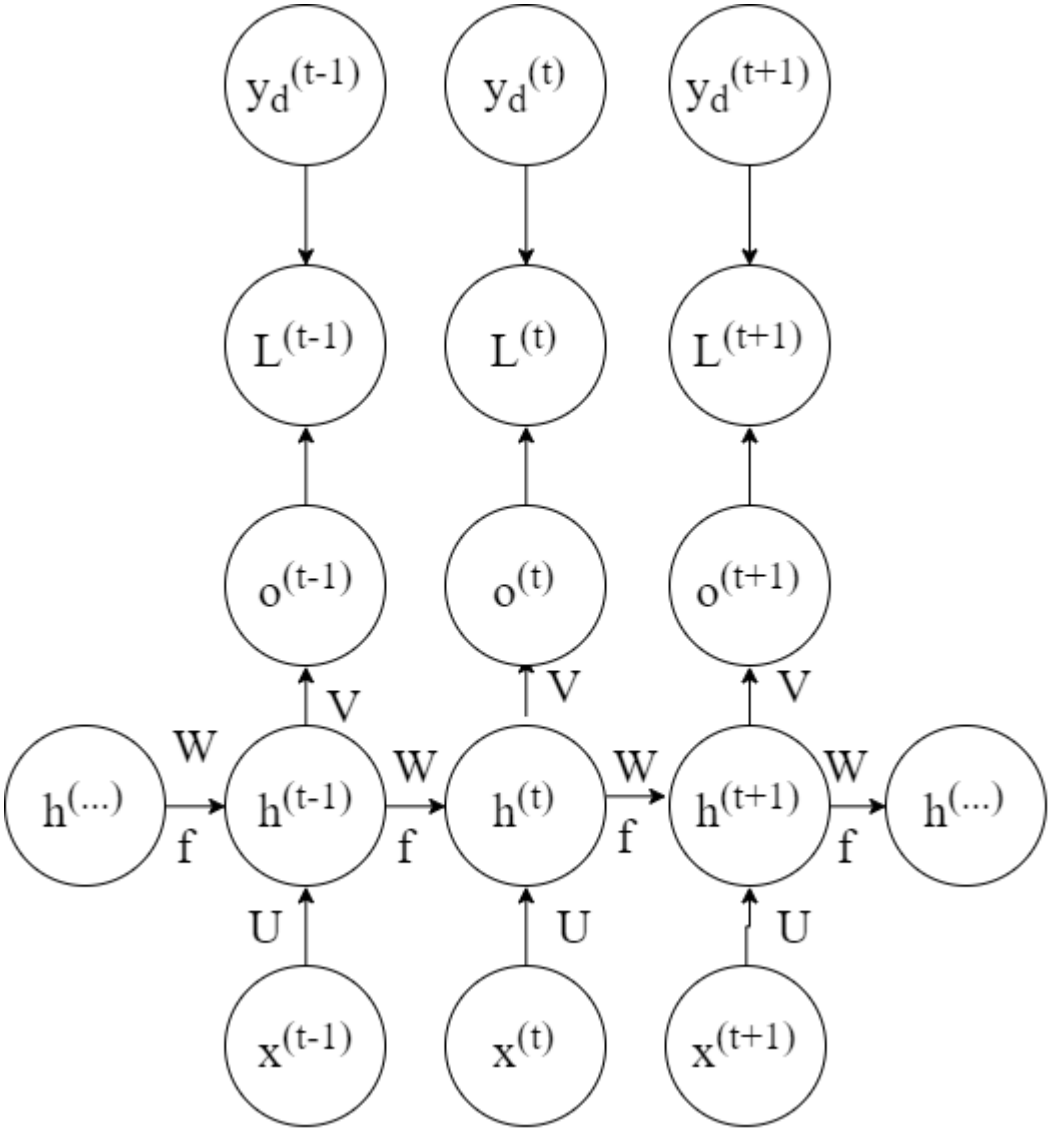

Рис. 10. Граф вычислений потерь при обучении рекуррентной сети,

которая отображает входную последовательность значений x в соответствующую выходную последовательность значений ***o***. Функция потерь L измеряет, насколько далеко каждый элемент ***o*** отстоит от соответствующей метки $\boldsymbol{y_д}$. В случае применения к выходам функции softmax можно предполагать, что ***o*** – ненормированные логарифмические вероятности. Внутри функция L вычисляет $\hat{y} = softmax(\boldsymbol{o})$ и сравнивает эту величину с меткой $\boldsymbol{y_д}$.

В РНС имеются связи между входным и скрытым слоями, параметризованные матрицей весов ***U***, рекуррентные связи между

скрытыми блоками, параметризованные матрицей весов $\boldsymbol{W}$, и связи между скрытым и выходным слоями, параметризованные матрицей весов $\boldsymbol{V}$. Уравнение (5) определяет прямое распространение в этой модели.

8.5.1. Алгоритм обратного распространения ошибки

Обратное распространение ошибки — это способ обучения нейронной сети. Цели обратного распространения: отрегулировать каждый вес пропорционально тому, насколько он способствует общей ошибке. Если итеративно уменьшать ошибку каждого веса, в конце концов будет ряд весов, которые дают хорошие прогнозы.

Алгоритм обратного распространения заключается в вычислении произведения якобиана на градиент для каждой операции в графе:

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z. \quad (19)$$

Обучение РНС аналогично обучению обычной нейронной сети. Также используется алгоритм обратного распространения ошибки, но с небольшим изменением. Поскольку одни и те же параметры используются на всех временных этапах в сети, градиент на каждом выходе зависит не только от расчетов текущего шага, но и от предыдущих временных шагов (см. рис. 11).

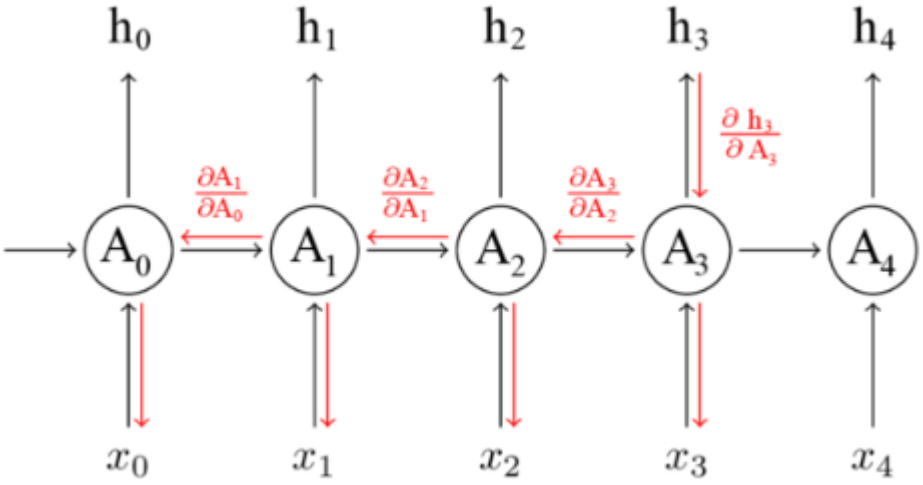

Рис. 11. Общий вид алгоритма обратного распространения ошибки для РНС.

8.6. Обучение РНС методом обратного распространения ошибки

Исходя из рис.10 в нейронной сети имеются параметры $L^{(t)}, V, W. b, c$, а также последовательность вершин, индексированных временем t: $x^{(t)}, h^{(t)}, o^{(t)}$ и $L^{(t)}$. Для каждой вершины N необходимо рекурсивно вычислить градиент $\nabla_N L$, зная градиенты, вычисленные в вершинах, следующих за ней в графе. Рекурсия начинается с вершин, непосредственно предшествующих окончательной потере:

$$\frac{\partial L}{\partial L^{(t)}} = 1 \text{ (20).}$$

Следуя дальше по графу и с учетом, что L – отрицательное логарифмическое правдоподобие и (19):

$$\nabla_{o^{(t)}} L = \frac{\partial H}{\partial H^{(t)}} \frac{\partial H^{(t)}}{\partial o^{(t)}} = \hat{y}^{(t)} - 1 \text{ (21)}$$

Двигаясь в направлении от конца последовательности к началу. В последний момент времени τ у $h^{(\tau)}$ есть только один потомок $o^{(\tau)}$:

$$\nabla_{h^{(\tau)}} L = V^T \nabla_{o^{(\tau)}} L \text{ (22).}$$

В соответствии с обратным распространением градиентов, модно совершать итерации назад во времени: $t = \tau - 1$ до $t = 1$. Т.к. потомками $h^{(t)}$ для $t < \tau$ являются $o^{(t)}$ и $h^{(t+1)}$:

$$\nabla_{h^{(\tau)}} L = \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}}\right)^T \nabla_{h^{(t+1)}} L + \left(\frac{\partial o^{(t)}}{\partial h^{(t)}}\right)^T \nabla_{o^{(t)}} L = W^T \left(\nabla_{h^{(t+1)}} L\right) diag\left(1 - (h^{(t+1)})^{(2)}\right) + V^T \nabla_{o^{(t)}} L, \text{ (23)}$$

где $diag\left(1 - (h^{(t+1)})^{(2)}\right)$ – диагональная матрица с элементами $\left(1 - (h^{(t+1)})^{(2)}\right)$. Это якобиан функции $f_1 = tanh$, скрытого блока i в момент времени $t + 1$.

Получив градиенты по внутренним вершинам графа вычислений, можно получить градиенты по вершинам параметров:

$$\nabla_c L = \sum_t \left(\frac{\partial o^{(t)}}{\partial c}\right)^T \nabla_{o^{(t)}} L = \sum_t \nabla_{o^{(t)}} L, \text{ (24)}$$

$$\nabla_b L = \sum_t \left(\frac{\partial h^{(t)}}{\partial b}\right)^T \nabla_{h^{(t)}} L = \sum_t diag\left(1 - (h^{(t+1)})^{(2)}\right) \nabla_{h^{(t)}} L, \text{ (25)}$$

$$\nabla_V L = \sum_t (\nabla_{o^{(t)}} L)\, h^{(t)^T}, \text{ (26)}$$

$$\nabla_W L = \sum_t diag\left(1 - (h^{(t)})^{(2)}\right) (\nabla_{h^{(t)}} L) h^{(t-1)^T}, \text{ (27)}$$

$$\nabla_V L = \sum_t diag\left(1 - (h^{(t)})^{(2)}\right) (\nabla_{h^{(t)}} L) x^{(t)^T}, \text{ (28)}$$

Уравнения (24) – (28) представляют собой полную модель обучения РНС.

9. Результаты

В качестве входного видео создан набор данных, где змейка размером 3 клетки, двигается по полю размером 5x5. Задача нейронной сети после каждого входного изображения получить другое изображение, непротиворечащее принципам движения змейки, а также являющееся таким же изображением, как и входное на следующем временном шаге. Изображения имеют расширение 25 x 25, на вход нейронной сети изображение поступает в виде вектора [1 x 625]. Количество классов (возможных состояний змейки) = 94, (см.рис. 12).

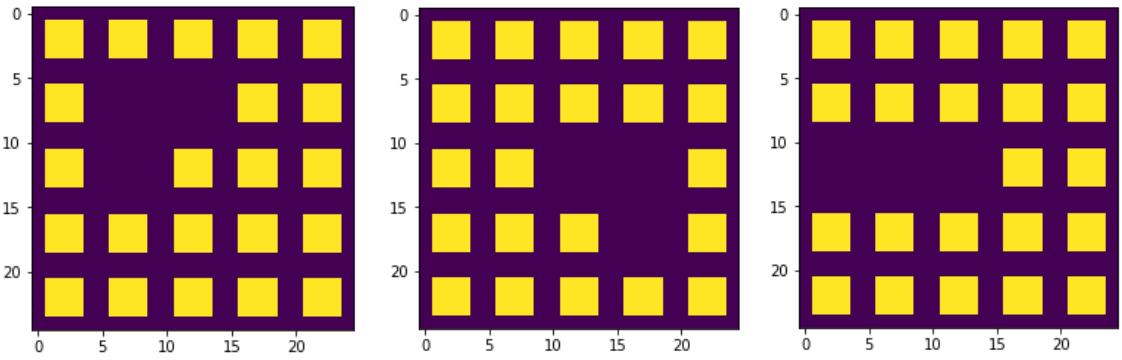

Рис. 12. Несколько из возможных состояний динамической системы

Нейронная сеть написана на языке программирования Python 3.7.0 с использованием библиотеки tensorflow. Код программы находится в приложении.

В результате нейронная сеть имеет точность ~ 95%, при соотношении тестовых данных и тренировочных 1:1, результаты работы нейронной сети:

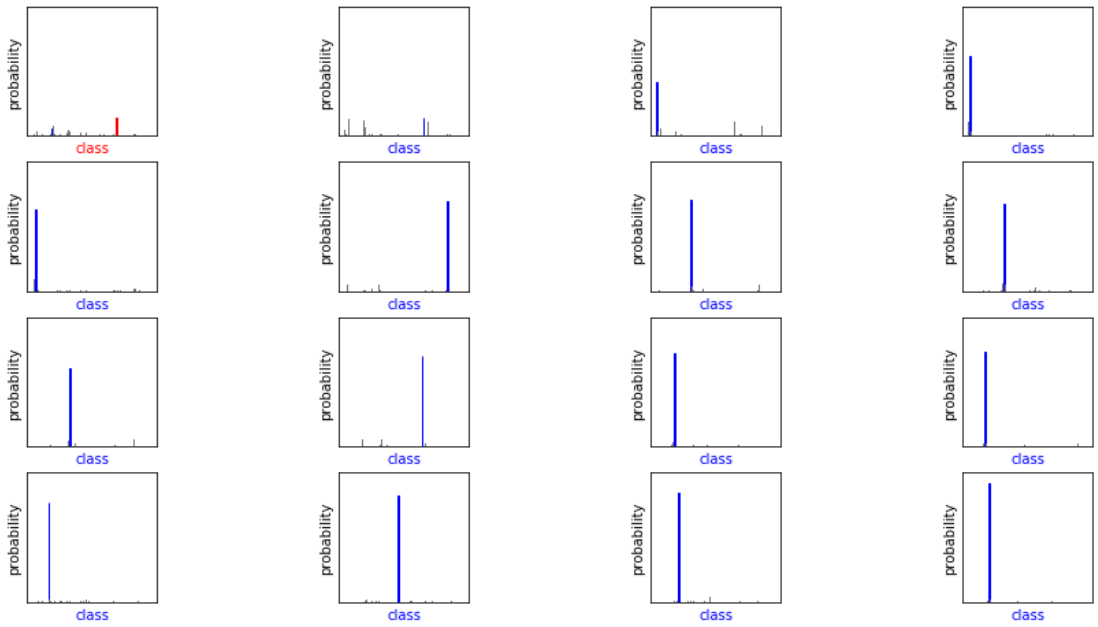

Рис. 13. Предсказания следующей итерации каскада нейронной сетью, где синим правильно предсказанное, красным – неправильное, на изображениях показано классовое распределение

10. Заключение

В связи с отличной точностью при малом количестве обучающих данных на фоне тестовых можно сделать вывод, что рекуррентные нейронные сети могут с высокой точность предсказывать итерации каскада.

11. Список использованных источников

1. Указ президента Российской Федерации о развитии искусственного интеллекта в Российской Федерации №490 от 10 октября 2019 года.
2. Dynamic Systems Modeling MATTHEW IRWIN and ZHENG WANG The Ohio State University, USA 01 August 2017
3. Groshek, J. (2011). Media, instability, and democracy: Examining the Granger-caused relationships of 122 countries from 1946 to 2003. Journal of Communication, 61, 1161–1182.
4. Busemeyer, J. R., & Diederich, A. (2009). Cognitive modeling. Thousand Oaks, CA: SAGE.
5. Wang, Z. (2014). Bridging media processing and selective exposure: A dynamic motivational model of media choices and choice response time. Communication Research, 41, 1064–1087.
6. Busemeyer, J. R. (2005) Dynamic systems: Mathematics. In L. Nadel (Ed.), Encyclopedia of Cognitive Science. Hoboken: NJ: John Wiley & Sons.
7. Mathematical Modeling, Third Edition By Mark M. Meerschaert
8. Dicrete-Time Nonautonomous Dynamical Systems Chapter 2 By P.E. Kloeden, C. Potzsche, M. Rasmussen
9. Фаустова К.И. Нейронные сети: применение сегодня и перспективы развития.
10. Горбачевская Е.Н. Классификация нейронных сетей.
11. Michael Nielsen. Neural Networks and Deep Learning.
12. Tariq Rashid Make Your Own Neural Network.
13. Christopher M. Bishop. Pattern Recognition and Machine Learning.
14. A. Dosovitskiy. Learning to Generate Chairs with Convolutional Neural Networks.
15. Е.Н. Бендерская, К.В. Никитин. Рекуррентная нейронная сеть как динамическая система и подходы к ее обучению.
16. Будыльский Д. В. GRU и LSTM: современные рекуррентные нейронные сети // Молодой ученый. — 2015. — №15. — С. 51-54.

17. J. Kumar, R. Goomer. Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters
18. Андросова Е.Е. Применение рекурсивных рекуррентных нейронных сетей.
19. Васенков Д.В. Методы обучение искусственных нейронных сетей.
20. Царегородцев В.Г. Конструктивный алгоритм синтеза структуры многослойного персептрона // Вычислительные технологии, 2008. Т.13 - Вестник КазНУ им. Аль-Фараби, серия "математика, механика, информатика", 2008. №4 (59). (Совм. выпуск). Часть 3. - с.308-315.
21. Царегородцев В.Г. Определение оптимального размера нейросети обратного распространения через сопоставление средних весов синапсов // Материалы XIV Международной конференции по нейрокибернетике, Ростов-на-Дону, 2005. Т.2. - С.60-64.
22. Царегородцев В.Г. Об исследовании эффективности одного метода построения отказоустойчивых нейросетей // Материалы X Всеросс. семинара "Нейроинформатика и ее приложения", Красноярск, 2002. 185с. - с.157-160.
23. Царегородцев В.Г. Простейший способ вычисления показателей значимости первого порядка для сетей обратного распространения // Материалы X Всеросс. семинара "Нейроинформатика и ее приложения", Красноярск, 2002. 185с. - с.153-156.
24. Царегородцев В.Г. Оптимизация предобработки данных: константа Липшица обучающей выборки и свойства обученных нейронных сетей // Нейрокомпьютеры: разработка, применение. 2003, №7. - С.3-8.
25. Царегородцев В.Г. Робастная целевая функция с допуском на точность решения для нейросети-предиктора // Нейрокомпьютеры: разработка, применение. 2003, №12.
26. Царегородцев В.Г. Уточнение решения обратной задачи для нейросети-классификатора // Нейрокомпьютеры: разработка, применение. 2003, №12.
27. Царегородцев В.Г. Высокая чувствительность отклика нейроклассификатора к колебаниям входов может индицировать наличие выбросов в данных // Материалы XII Всеросс. семинара "Нейроинформатика и ее приложения", Красноярск, 2004. - 196с. - С.158-162.

28. Царегородцев В.Г. Общая неэффективность использования суммарного градиента выборки при обучении нейронной сети // Материалы XII Всеросс. семинара "Нейроинформатика и ее приложения", Красноярск, 2004. - 196с. - С.145-151.