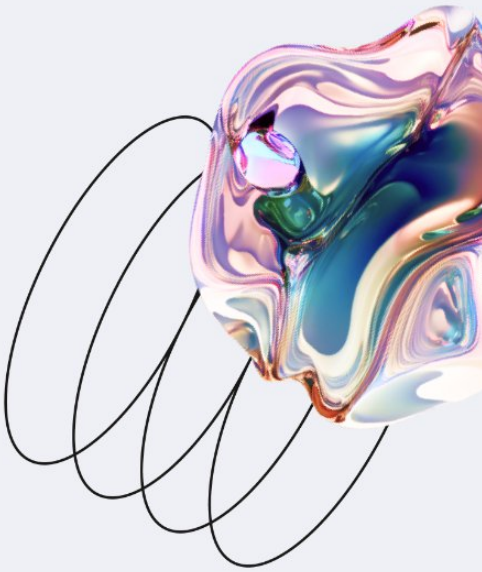
GeekBrains



Рекуррентные нейронные сети

Архитектура нейронных сетей

- Шрифт – IBM Plex Sans
  Размер текста — 12, межстрочный интервал — 1,15, интервал между абзацами — 10, выравнивание по ширине
  Размер заголовков первого уровня — 22, второго — 18, третьего — 16

Оглавление

Введение

Всем привет! Это четвертая лекция на курсе архитектура нейронных сетей. Сегодня мы погрузимся в захватывающий мир **рекуррентных нейронных сетей** или RNN (Recurrent neural network). Машинное обучение продолжает свое развитие и RNN играют ключевую роль в решении задач, связанных с последовательными данными, такими как текст, звук, временные ряды и другими. Эти сети обладают способностью запоминать предыдущие состояния, что делает их мощным инструментом для анализа и прогнозирования последовательных паттернов. Давайте погружаться глубже в мир RNN и изучать, как они работают, а также в каких областях они находят свое применение.

Словарь терминов

Рекуррентная нейронная сеть — это тип искусственной нейронной сети, предназначенной для обработки последовательностей данных. Они особенно хорошо работают для задач, требующих последовательностей, таких как данные временных рядов, голос, естественный язык и другие действия.
LSTM – это тип RNN, предназначенный для решения проблемы исчезновения градиента, которая может возникнуть в стандартных RNN
GRU — это еще один тип RNN, предназначенный для решения проблемы исчезающего градиента
Механизм внимания — это метод, который можно использовать для повышения производительности RNN при выполнении задач, включающих длинные входные последовательности
Градиент — это значение, используемое для корректировки внутренних весов сети, позволяющее сети обучаться

Рекуррентные нейронные сети

Введение в RNN

Когда дело доходит до последовательных данных или данных временных рядов, традиционные сети прямого распространения не могут использоваться для обучения и прогнозирования. Требуется механизм для сохранения прошлой или исторической информации для прогнозирования будущих значений. Рекуррентные нейронные сети представляют собой вариант обычных искусственных нейронных сетей прямого распространения, которые могут работать с последовательными данными и могут быть обучены хранить знания о прошлом.

RNN представляют собой современный алгоритм обработки последовательных данных и используются, например, в голосовом поиске Google. Это первый алгоритм, который запоминает введенные данные благодаря внутренней памяти, что делает его идеально подходящим для задач машинного обучения , включающих последовательные данные.

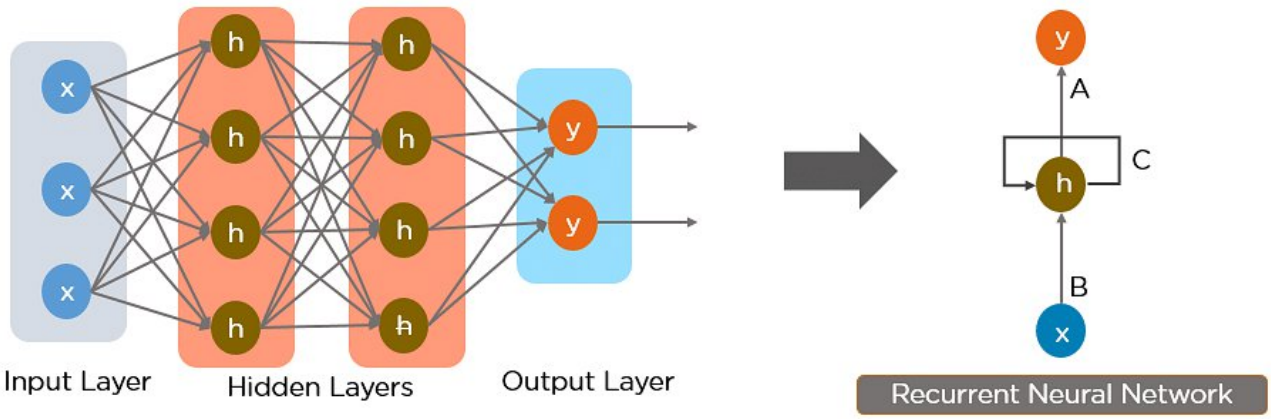
Как и многие другие алгоритмы глубокого обучения, рекуррентные нейронные сети относительно стары. Первоначально они были созданы в 1980-х годах, но только в последние годы мы увидели их истинный потенциал. Увеличение вычислительной мощности наряду с огромными объемами данных, с которыми нам теперь приходится работать, а также изобретение долговременной кратковременной памяти (LSTM) в 1990-х годах выдвинуло RNN на передний план.

Благодаря своей внутренней памяти RNN могут запоминать важные сведения о полученных входных данных. Это позволяет им очень точно предсказывать, что будет дальше. Вот почему они являются предпочтительным алгоритмом для последовательных данных, таких как временные ряды , речь, текст, финансовые данные, аудио, видео, погода и многое другое. Рекуррентные нейронные сети могут обеспечить гораздо более глубокое понимание последовательности и ее контекста по сравнению с другими алгоритмами.

Архитектура сети

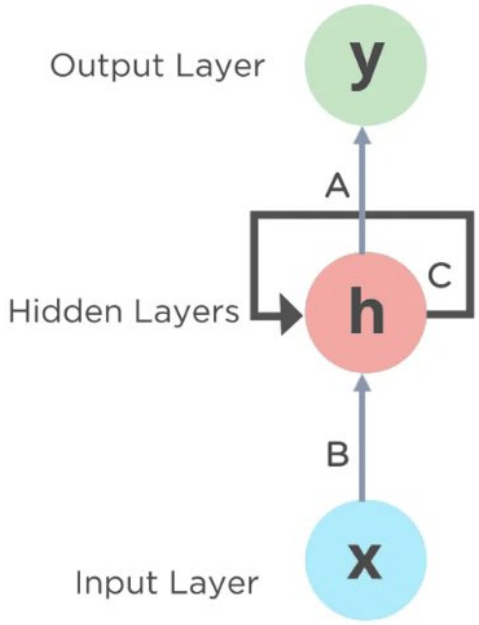
RNN работает по принципу сохранения выходных данных определенного слоя и подачи их обратно на вход, чтобы спрогнозировать выходные данные слоя.

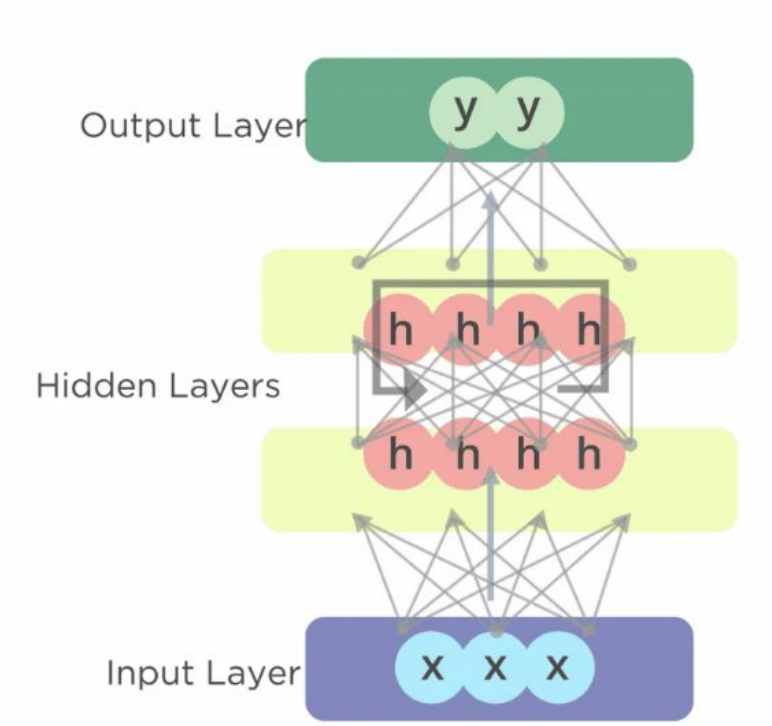
Ниже показано, как можно преобразовать нейронную сеть прямого распространения в рекуррентную нейронную сеть:



Простая рекуррентная нейронная сеть.

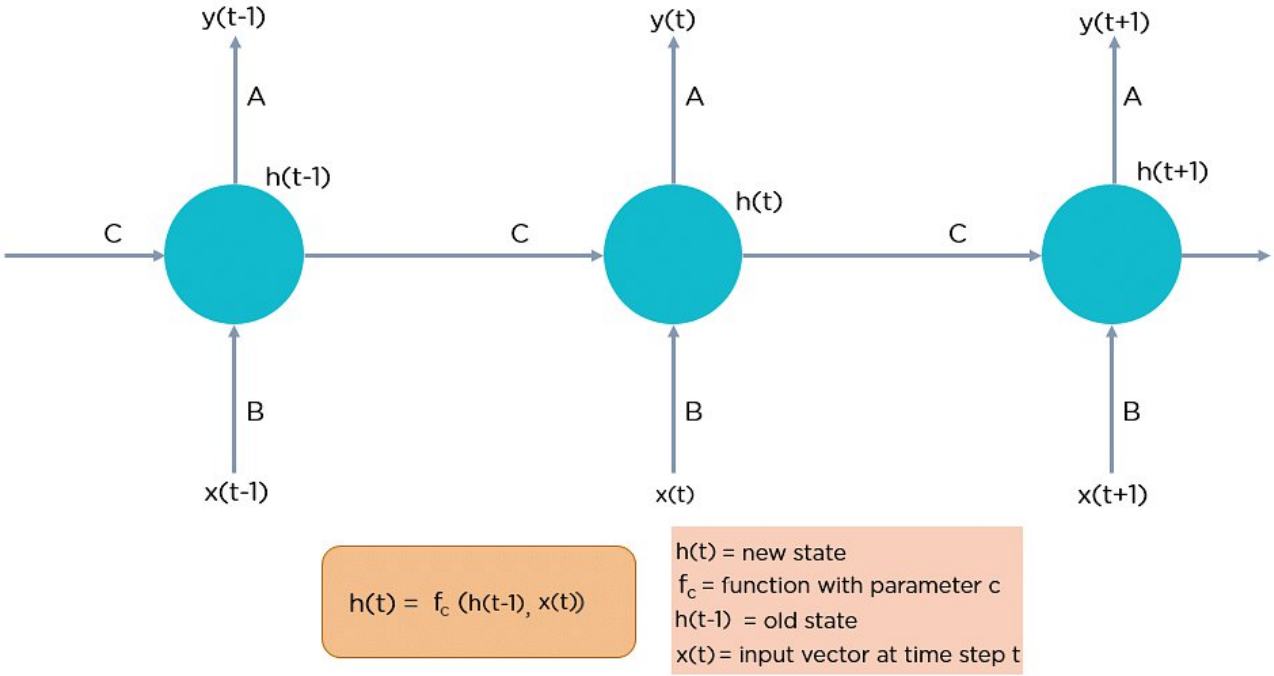
Узлы в разных слоях нейронной сети сжимаются, образуя один слой рекуррентных нейронных сетей. A, B и C — параметры сети.





Полностью подключенная рекуррентная нейронная сеть.

Здесь «x» — входной слой, «h» — скрытый слой, а «y» — выходной слой. A, B и C — параметры сети, используемые для улучшения результатов модели. В любой момент времени t текущий вход представляет собой комбинацию входов в точках x(t) и x(t-1). Выходные данные в любой момент времени возвращаются в сеть для улучшения выходных данных.



Полностью подключенная рекуррентная нейронная сеть.

Теперь, когда вы понимаете, что такое рекуррентная нейронная сеть, давайте рассмотрим различные типы рекуррентных нейронных сетей.

Почему рекуррентные нейронной сети

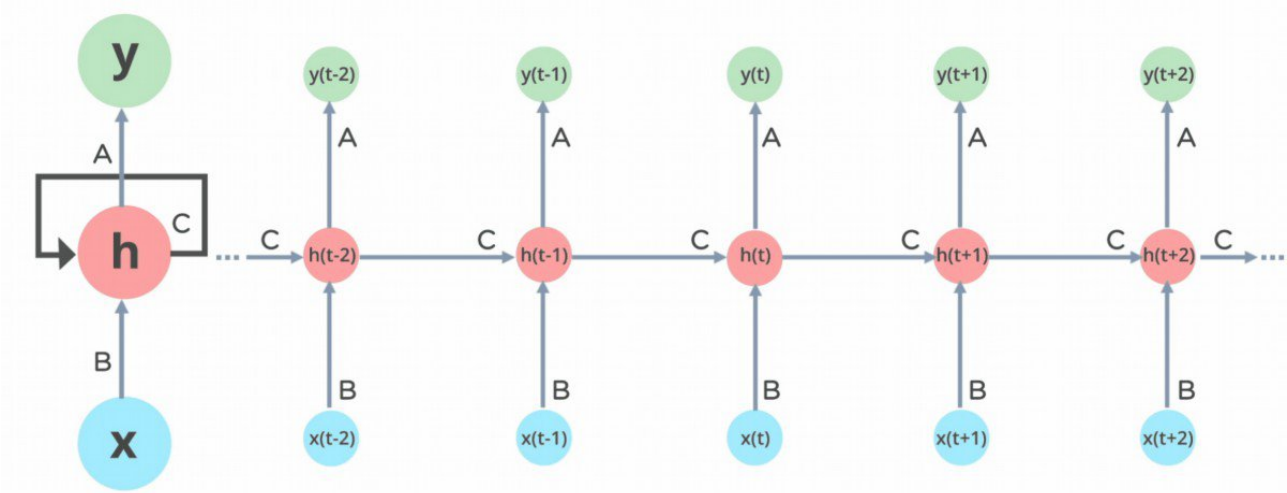
RNN были созданы потому, что в нейронной сети прямого распространения было несколько проблем:

- Невозможно обрабатывать последовательные данные
- Учитывает только текущий ввод
- Невозможно запомнить предыдущие вводы

Решением этих проблем является RNN. RNN может обрабатывать последовательные данные, принимая текущие входные данные и ранее полученные входные данные. RNN могут запоминать предыдущие входные данные благодаря своей внутренней памяти.

Как работают рекуррентные нейронные сети

В рекуррентных нейронных сетях информация циклически перемещается к среднему скрытому слою.



Работа рекуррентной нейронной сети.

Входной слой «x» принимает входные данные в нейронную сеть, обрабатывает их и передает на средний уровень.

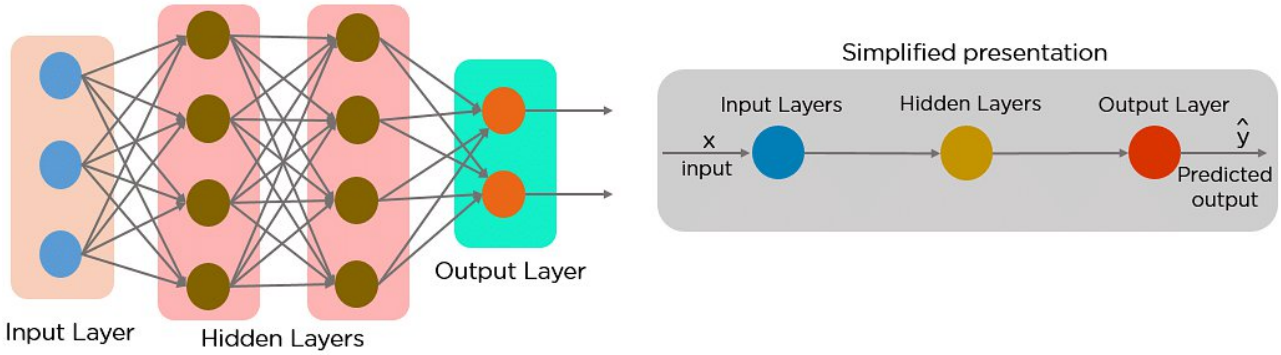
Средний уровень «h» может состоять из нескольких скрытых слоев, каждый из которых имеет свои собственные функции активации, веса и смещения. Если у вас есть нейронная сеть, в которой предыдущий слой не влияет на различные параметры разных скрытых слоев, т.е. нейронная сеть не имеет памяти, то вы можете использовать рекуррентную нейронную сеть.

Рекуррентная нейронная сеть стандартизирует различные функции активации, веса и смещения, чтобы каждый скрытый слой имел одинаковые параметры. Затем, вместо создания нескольких скрытых слоев, она создаст один и прокрутит его столько раз, сколько потребуется.

Нейронные сети прямого распространения против рекуррентных нейронных сетей

Нейронная сеть с прямой связью позволяет информации течь только в прямом направлении, от входных узлов через скрытые слои к выходным узлам. В сети нет циклов и петель.

Ниже показано, как выглядит упрощенное представление нейронной сети прямого распространения:



Нейронная сеть с прямой связью

В нейронной сети с прямой связью решения принимаются на основе текущих входных данных. Она не запоминает прошлые данные, и у нее нет возможности предсказывать будущее. Нейронные сети прямого распространения используются в общих задачах регрессии и классификации.

Преимущества рекуррентной нейронной сети

Рекуррентные нейронные сети имеют ряд преимуществ перед другими типами нейронных сетей, в том числе:

1. Возможность обработки последовательностей переменной длины

RNN предназначены для обработки входных последовательностей переменной длины, что делает их хорошо подходящими для таких задач, как распознавание речи, обработка естественного языка и анализ временных рядов.

2. Память прошлых входов

RNN имеют память о прошлых входных данных, что позволяет им собирать информацию о контексте входной последовательности. Это делает их полезными для таких задач, как моделирование языка, где значение слова зависит от контекста, в котором оно появляется.

3. Совместное использование параметров

RNN используют один и тот же набор параметров на всех временных шагах, что уменьшает количество параметров, которые необходимо изучить, и может привести к лучшему обобщению.

4. Нелинейное отображение

RNN используют нелинейные функции активации, что позволяет им изучать сложные нелинейные сопоставления между входами и выходами.

5. Последовательная обработка

RNN обрабатывают входные последовательности друг за другом, что делает их вычислительно эффективными и легко распараллеливаемыми.

6. Гибкость

RNN можно адаптировать к широкому спектру задач и типов ввода, включая текстовые, речевые и графические последовательности.

7. Улучшенная точность

Было выяснено, что RNN достигают высокой производительности при решении различных задач моделирования последовательностей, включая моделирование языка, распознавание речи и машинный перевод.

Эти преимущества делают RNN мощным инструментом для моделирования и анализа последовательностей и привели к их широкому использованию в различных приложениях, включая обработку естественного языка, распознавание речи и анализ временных рядов.

Недостатки рекуррентных нейронных сетей

Хотя рекуррентные нейронные сети (RNN) имеют ряд преимуществ, у них есть и некоторые недостатки. Вот некоторые из основных недостатков RNN:

1. Исчезающие и взрывающиеся градиенты

RNN могут страдать от проблемы исчезновения или взрыва градиентов, что может затруднить эффективное обучение сети. Это происходит, когда градиенты функции потерь по отношению к параметрам становятся очень маленькими или очень большими по мере распространения во времени.

2. Вычислительная сложность

Обучение RNN может оказаться дорогостоящим в вычислительном отношении, особенно при работе с длинными последовательностями. Это связано с тем, что сеть должна обрабатывать каждый ввод последовательно, что может занимать много времени.

3. Сложность фиксации долгосрочных зависимостей

Хотя RNN предназначены для сбора информации о прошлых входных данных, им может быть сложно уловить долгосрочные зависимости во входной последовательности. Это связано с тем, что градиенты могут стать очень маленькими по мере распространения во времени, что может привести к тому, что сеть забудет важную информацию.

4. Отсутствие параллелизма

RNN по своей сути являются последовательными, что затрудняет распараллеливание вычислений. Это может ограничить скорость и масштабируемость сети.

5. Сложность выбора правильной архитектуры

Существует множество различных вариантов RNN, каждый из которых имеет свои преимущества и недостатки. Выбор правильной архитектуры под конкретную ситуацию может оказаться сложной задачей и потребовать обширных экспериментов и настройки.

6. Сложность интерпретации результатов

Вывод RNN может быть трудным для интерпретации, особенно при работе со сложными входными данными, такими как естественный язык или аудио. Это может затруднить понимание того, как сеть делает свои прогнозы.

Эти недостатки важны при принятии решения о том, использовать ли RNN для конкретной задачи. Однако многие из этих проблем можно решить путем тщательного проектирования и обучения сети, а также с помощью таких методов, как механизмы регуляризации и внимания.

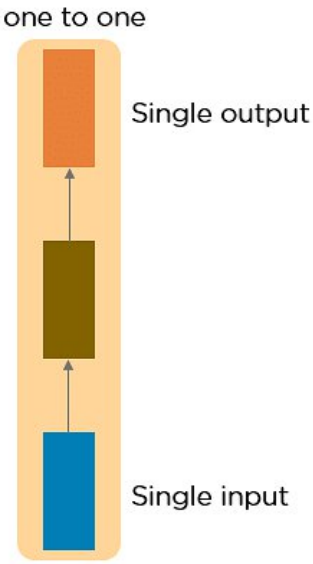
Типы RNN

Существует четыре типа рекуррентных нейронных сетей:

1. Один к одному
2. Один ко многим
3. Многие к одному
4. Многие ко многим

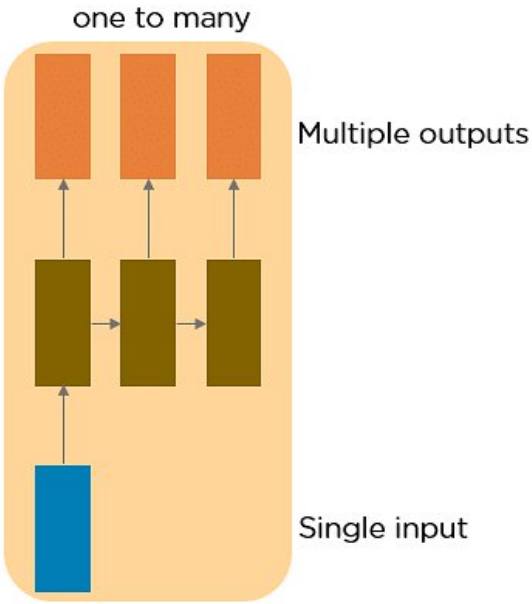
Один к одному RNN

Этот тип нейронной сети известен как ванильная нейронная сеть. Он используется для общих задач машинного обучения и имеет один вход и один выход.



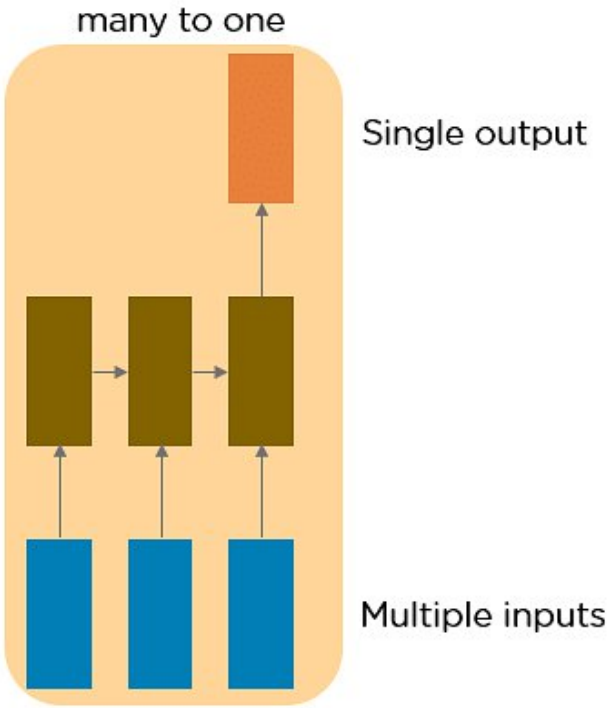
Один ко многим RNN

Нейронная сеть этого типа имеет один вход и несколько выходов. Примером этого является подпись к изображению. В этом случае происходит классификация и поиск объекта на изображении.



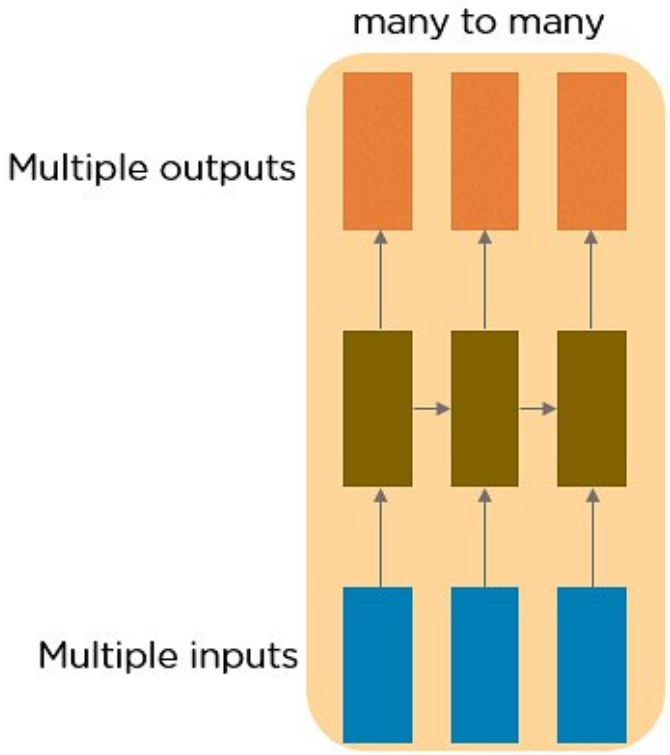
Многие к одному RNN

Этот RNN принимает последовательность входных данных и генерирует один выходной сигнал. Анализ настроений — хороший пример такого рода сети, где предложение можно классифицировать как выражающее положительные или отрицательные настроения.



Многие ко многим RNN

Эта RNN принимает последовательность входных данных и генерирует последовательность выходных данных. Машинный перевод является одним из примеров.

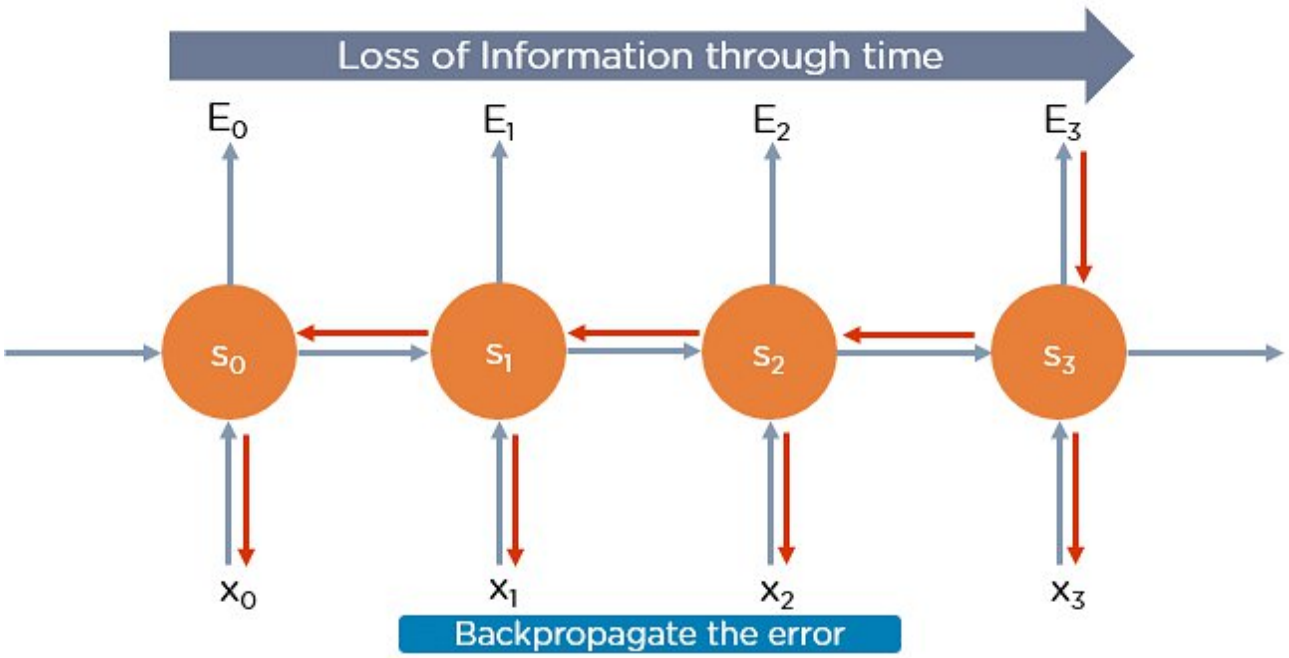


Две проблемы стандартных RNN

Проблема исчезающего градиента

Рекуррентные нейронные сети позволяют моделировать зависящие от времени и последовательные задачи с данными, такие как прогнозирование фондового рынка, машинный перевод и генерация текста. Однако вы обнаружите, что RNN сложно обучать из-за проблемы градиента.

RNN страдают от проблемы исчезновения градиентов. Градиенты несут информацию, используемую в RNN, и когда градиент становится слишком маленьким, обновления параметров становятся незначительными. Это затрудняет изучение длинных последовательностей данных.

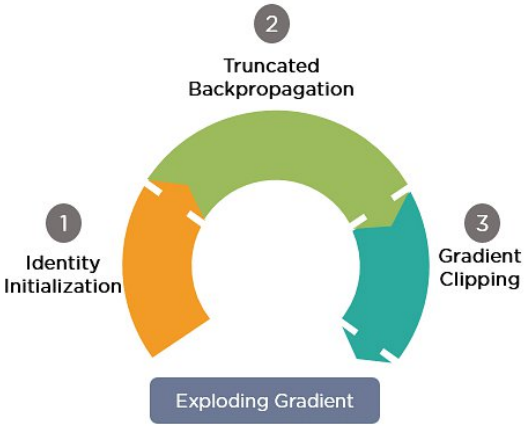


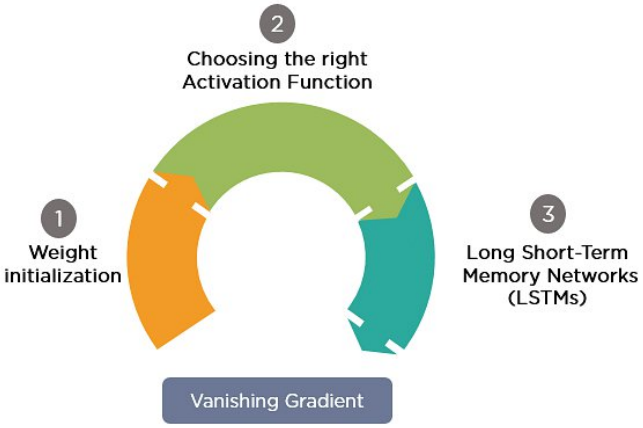
Проблема взрывного градиента

Если при обучении нейронной сети градиент имеет тенденцию экспоненциально расти, а не затухать, это называется взрывным градиентом. Эта проблема возникает, когда накапливаются большие градиенты ошибок, что приводит к очень большим обновлениям весов модели нейронной сети во время процесса обучения.

Длительное время обучения, низкая производительность и плохая точность являются основными проблемами градиентных задач.

Решения градиентных проблем





Теперь давайте обсудим самый популярный и эффективный способ решения проблем градиента, то есть сеть долгой краткосрочной памяти (LSTM - long short-term memory).

Во-первых, давайте поймем долгосрочные зависимости.

Предположим, вы хотите предсказать последнее слово в тексте: «Облака находятся в _____».

Самый очевидный ответ на этот вопрос — «небо». Нам не нужен дополнительный контекст, чтобы предсказать последнее слово в приведенном выше предложении.

Рассмотрим следующее предложение: «Я живу в Испании последние 10 лет… Я свободно говорю на _____».

Предсказанное вами слово будет зависеть от нескольких предыдущих слов в контексте. Здесь вам нужен контекст Испании, чтобы предсказать последнее слово в тексте, и наиболее подходящий ответ на это предложение — «Испанский». Разрыв между соответствующей информацией и моментом, когда она необходима, мог стать очень большим. LSTM помогут вам решить эту проблему. О LSTM мы будем с вами говорить на следующей лекции.

Общие функции активации

Рекуррентные нейронные сети, как и другие, используют функции активации, чтобы внести нелинейность в свои модели. Вот некоторые распространенные функции активации, используемые в RNN:

1. Сигмоидная функция:

Сигмоидная функция обычно используется в RNN. Она имеет диапазон от 0 до 1, что делает ее полезной для задач двоичной классификации. Формула сигмоидной функции:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

2. Функция гиперболического тангенса (Tanh):

Функция tanh также часто используется в RNN. Она имеет диапазон от -1 до 1, что делает ее полезной для задач нелинейной классификации. Формула функции tanh:

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

3. Функция выпрямленной линейной единицы (Relu):

Функция ReLU — это нелинейная функция активации, которая широко используется в глубоких нейронных сетях. Она имеет диапазон от 0 до бесконечности, что делает ее полезной для моделей, требующих положительных выходных данных. Формула функции ReLU:

ReLU(x) = max(0, x)

4. Функция Leaky Relu:

Функция Leaky ReLU аналогична функции ReLU, но она вносит небольшой наклон к отрицательным значениям, что помогает предотвратить «мертвые нейроны» в модели. Формула функции Leaky ReLU:

Leaky ReLU(x) = max(ax, x), где a (alpha) - отрицательный уклон, который является маленьким положительным числом, например, 0,01.

5. Функция Softmax:

Функция softmax часто используется на выходном слое RNN для задач классификации нескольких классов. Она преобразует выходные данные сети в распределение вероятностей по возможным классам. Формула функции softmax:

$$softmax(x) = \frac{e^{x}}{\sum e^{x}}$$

Это всего лишь несколько примеров функций активации, используемых в RNN. Выбор функции активации зависит от конкретной задачи и архитектуры модели.

Обратное распространение ошибки во времени

Обратное распространение ошибки во времени — это когда мы применяем алгоритм обратного распространения ошибки к рекуррентной нейронной сети, которая имеет на входе данные временных рядов.

В типичной RNN в сеть одновременно подается один вход и получается один выход. Но при обратном распространении ошибки вы используете как текущие, так и предыдущие входные данные. Это называется временным шагом, и один временной шаг будет состоять из множества точек данных временных рядов, одновременно входящих в RNN.

После того как нейронная сеть обучилась на заданном временном интервале и предоставила вам выходные данные, они используются для расчета и накопления ошибок. После этого сеть разворачивается, а веса пересчитываются и обновляются с учетом ошибок.

Варианты архитектуры RNN

Существует несколько вариантов архитектур RNN, которые были разработаны на для устранения ограничений стандартной архитектуры RNN. Вот несколько примеров:

Сети с долгой краткосрочной памятью (LSTM)
LSTM — это тип RNN, предназначенный для решения проблемы исчезновения градиента, которая может возникнуть в стандартных RNN. Это достигается за счет введения трех вентильных механизмов, которые контролируют поток информации через сеть: входной вентиль, вентиль забывания и выходной вентиль. Эти вентили позволяют сети LSTM выборочно запоминать или забывать информацию из входной последовательности, что делает ее более эффективной для долгосрочных зависимостей.

Сети GRU (GRU)
GRU — это еще один тип RNN, предназначенный для решения проблемы исчезающего градиента. Он имеет два шлюза: шлюз сброса и шлюз обновления. Вентиль сброса определяет какую часть предыдущего состояния следует забыть, а вентиль обновления – какую часть нового состояния следует

запомнить. Это позволяет сети GRU выборочно обновлять свое внутреннее состояние на основе входной последовательности.

Двунаправленные RNN
Двунаправленные RNN предназначены для обработки входных последовательностей как в прямом, так и в обратном направлении. Это позволяет сети захватывать как прошлый, так и будущий контекст. Это полезно для задач распознавания речи и обработки естественного языка.

Кодер-декодер RNN
Кодера-декодер RNN состоят из двух RNN:

- сети кодера, которая обрабатывает входную последовательность и создает векторное представление входных данных фиксированной длины;
- сети декодера, которая генерирует выходную последовательность на основе представления кодера.

Эта архитектура обычно используется для задач последовательного преобразования, таких как машинный перевод.

Механизмы внимания

Механизмы внимания — это метод, который можно использовать для повышения производительности RNN при выполнении задач, включающих длинные входные последовательности. Они работают, позволяя сети избирательно обрабатывать различные части входной последовательности, а не обрабатывать все части одинаково. Это может помочь сети сосредоточиться на наиболее важных частях входной последовательности и игнорировать ненужную информацию.

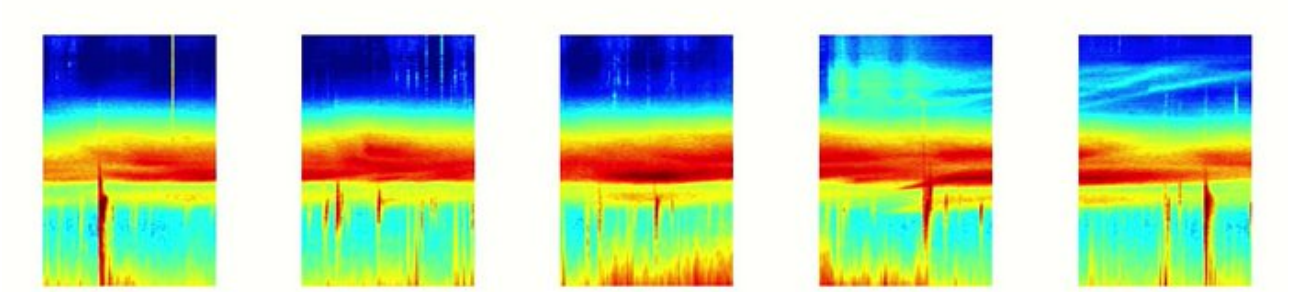
Это всего лишь несколько примеров множества вариантов архитектур RNN, которые разрабатывались на протяжении многих лет. Выбор архитектуры зависит от конкретной задачи и особенностей входных и выходных последовательностей.

А теперь простыми словами…

Итак, RNN — это нейронные сети, которые хорошо моделируют данные последовательности. Чтобы понять, что это значит, давайте проведем мысленный эксперимент. Предположим, вы сделали снимок мяча, движущегося во времени.

Предположим также, что вы хотите предсказать направление движения мяча. Итак, как бы вы это сделали, имея только ту информацию, которую вы видите на экране? Ну, вы можете предположить, но любой ответ, который вы получите, будет случайным предположением. Без знания того, где находился мяч, у вас не будет достаточно данных, чтобы предсказать, куда он направляется.
Если вы сделаете множество снимков положения мяча подряд, у вас будет достаточно информации, чтобы сделать более точный прогноз.

Итак, эта последовательность – особый порядок, в котором одно следует за другим. Имея эту информацию, вы теперь можете видеть, что мяч движется вправо. Такие последовательности существуют во многих формах. Звук — это естественная последовательность. Вы можете разбить аудиоспектрограмму на куски и передать их в RNN.



Аудиоспектрограмма, разрезанная на куски
Текст — это еще одна форма последовательностей. Вы можете разбить текст на последовательность символов или последовательность слов.

Последовательная память
Итак, RNN хорошо обрабатывают данные последовательности для прогнозов. Но как?
Они делают это, используя концепцию, которую можно назвать последовательной памятью. Чтобы получить хорошее представление о том, что означает последовательная память попробуйте произнести алфавит в уме. Это довольно легко, правда. Если вас учили этой конкретной последовательности, она должна быстро прийти к вам.

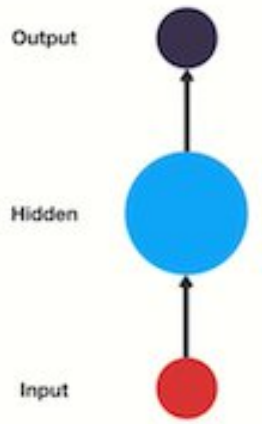
А теперь попробуйте произнести алфавит задом наперед. Это намного сложнее. Если вы раньше не практиковали эту конкретную последовательность, вам, скорее всего, придется нелегко.

Или начните с буквы E. Сначала вам будет сложно назвать первые несколько букв, но затем, когда ваш мозг уловит закономерность, остальное придет само собой.

Итак, есть вполне логичная причина, почему это может быть сложно. Вы изучаете алфавит как последовательность. Последовательная память — это механизм, который облегчает вашему мозгу распознавание последовательностей.

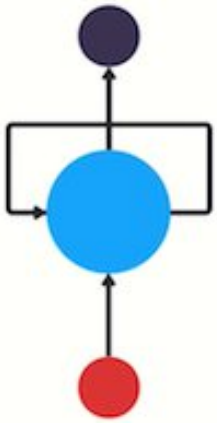
Рекуррентные нейронные сети

Давайте посмотрим на традиционную нейронную сеть, также известную как нейронная сеть с прямой связью. У нее есть входной слой, скрытый слой и выходной слой.



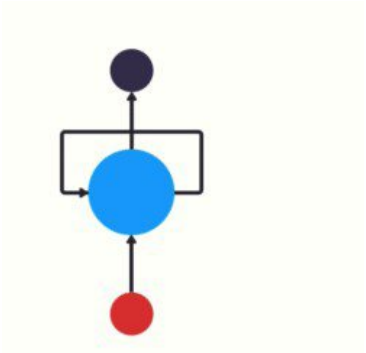
Нейронная сеть с прямой связью

Как мы можем заставить нейронную сеть прямого распространения использовать предыдущую информацию для воздействия на последующие? Что, если мы добавим в нейронную сеть цикл, который сможет передавать предыдущую информацию вперед?



Рекуррентная нейронная сеть

И это, по сути, то, что делает рекуррентная нейронная сеть. RNN имеет циклический механизм, который действует как магистраль, позволяющая информации перетекать от одного шага к другому.



Передача скрытого состояния на следующий временной шаг

Эта информация представляет собой скрытое состояние, которое представляет собой предыдущие входные данные. Давайте рассмотрим вариант использования RNN, чтобы лучше понять, как это работает.
Допустим, мы хотим создать чат-бота. Чат-бот может классифицировать намерения по введенному пользователем тексту.

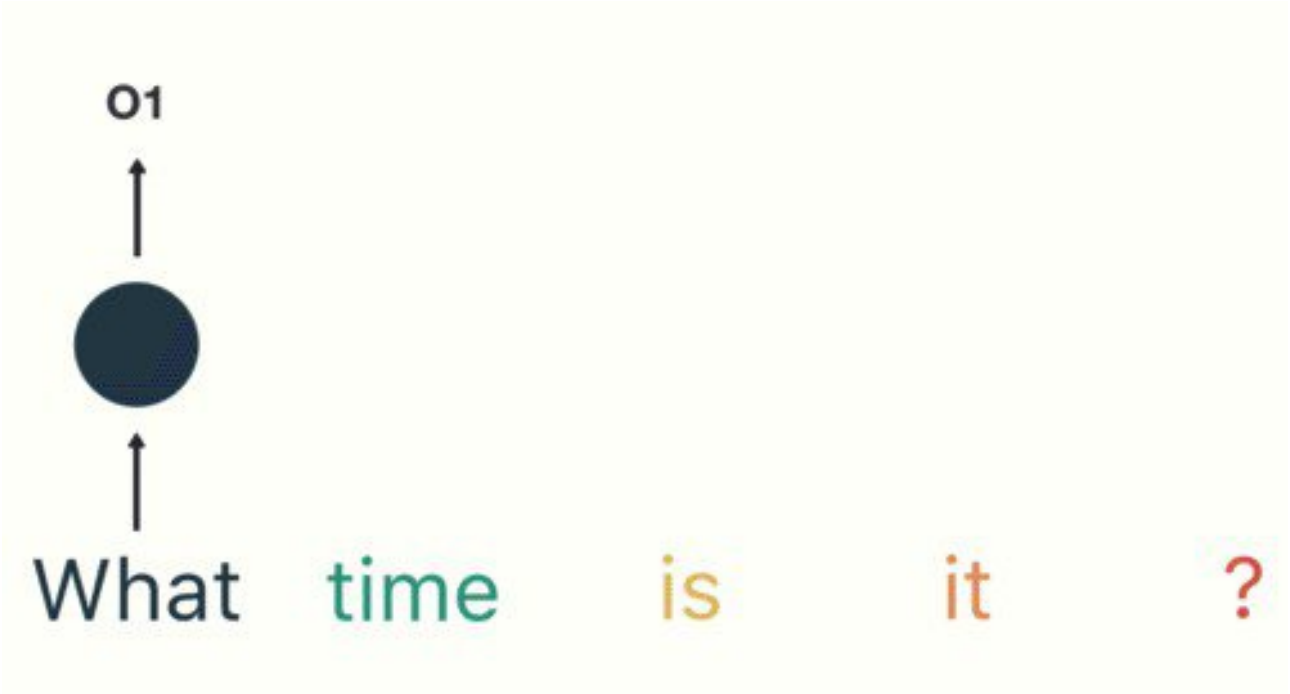
Классификация намерений на основе вводимых пользователем данных

Чтобы бот справился с задачей сначала нужно закодировать последовательность текста с помощью RNN. Затем передать выходные данные RNN в нейронную сеть прямого распространения, которая будет классифицировать намерения.
Итак, пользователь вводит… *сколько сейчас времени?* .Разобьем предложение на отдельные слова. RNN работает последовательно, поэтому мы передаем его по одному слову за раз.

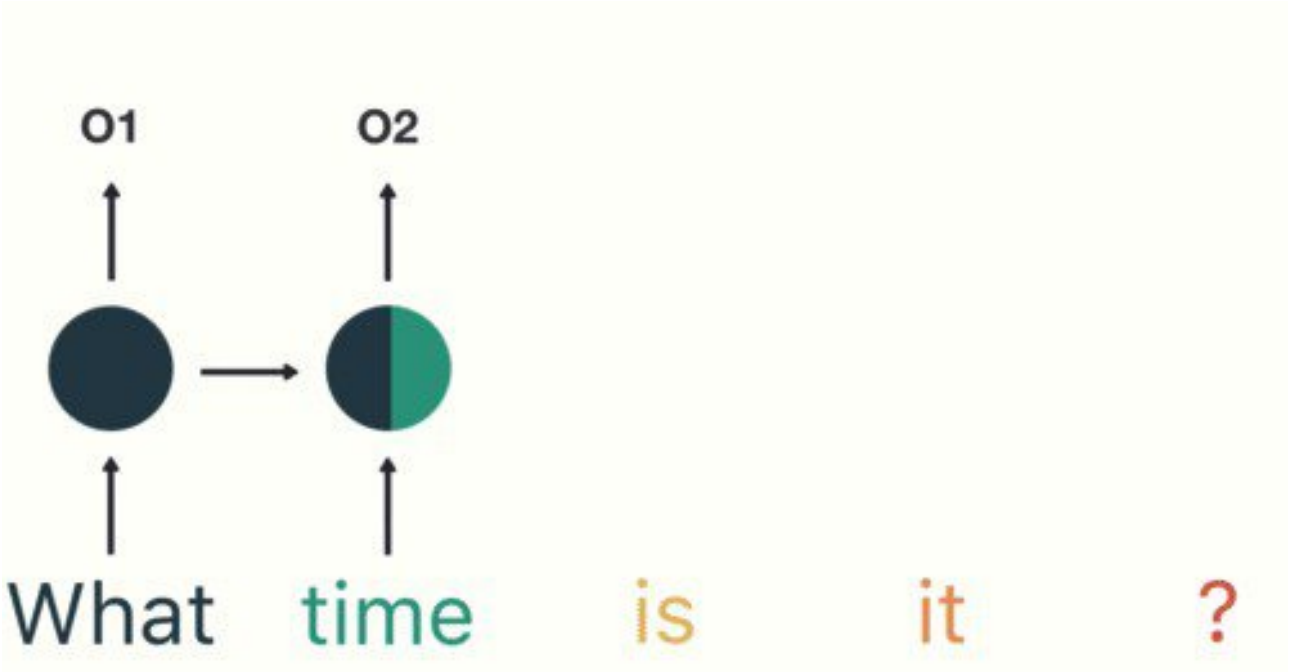
Разбиение предложения на последовательности слов

Первый шаг — ввести «What» в RNN. RNN кодирует «What» и выдает результат.

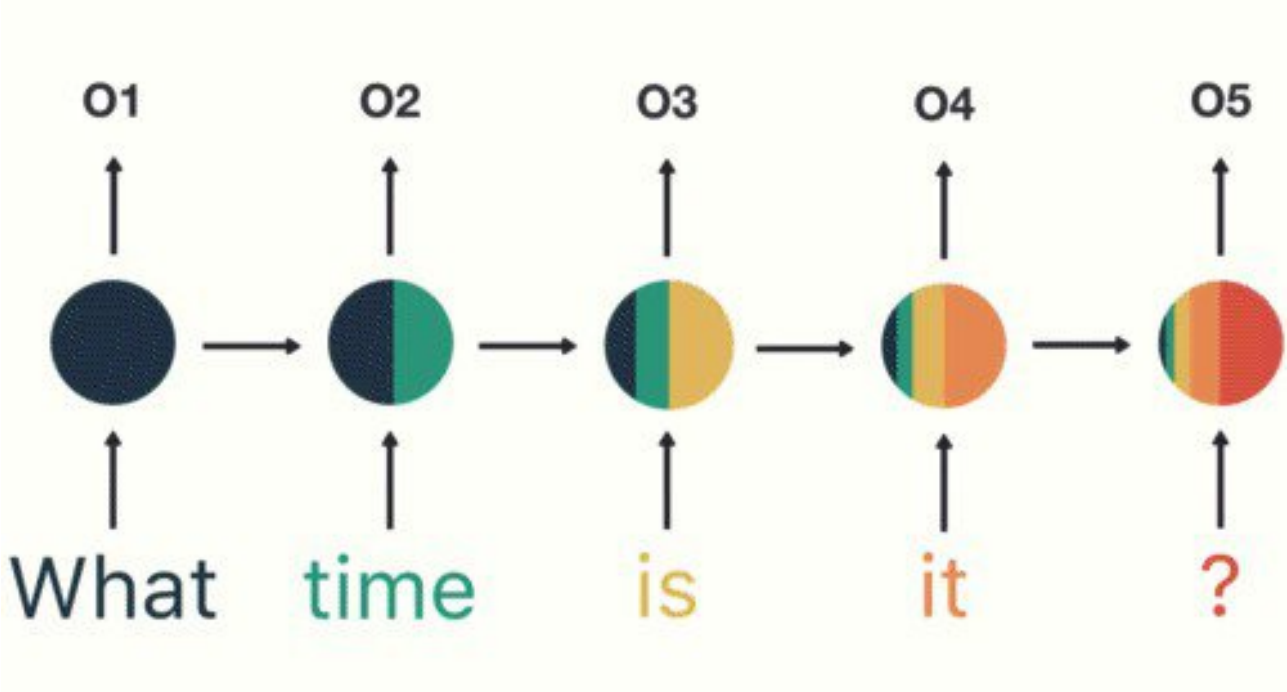
На следующем шаге мы вводим слово «время» и скрытое состояние из предыдущего шага. Теперь RNN располагает информацией как по слову «Что», так и по слову «время».



Повторяем этот процесс до последнего шага. На последнем этапе вы можете видеть, что RNN закодировала информацию из всех слов на предыдущих шагах.



Поскольку конечный результат был создан из остальной части последовательности, мы должны иметь возможность взять окончательный результат и передать его на уровень прямой связи для классификации намерения.



На python это можно представить так

```
rnn = RNN()
ff = FeedForwardNN()
hidden_state =[0.0, 0.0, 0.0, 0.0]

for word in input:
    output, hidden_state = rnn(word, hidden_state)

prediction = ff(output)
```

Псевдокод для потока управления RNN

Сначала вы инициализируете свои сетевые уровни и исходное скрытое состояние. Форма и размер скрытого состояния будут зависеть от формы и размера вашей рекуррентной нейронной сети. Затем вы перебираете свои входные данные, передаете слово и скрытое состояние в RNN.

RNN возвращает выходные данные и измененное скрытое состояние. Вы продолжаете цикл, пока у вас не закончатся слова. Последним вы передаете выходные данные на уровень прямой связи, и он возвращает прогноз. Вот и все! Поток управления прямым проходом рекуррентной нейронной сети представляет собой цикл for.

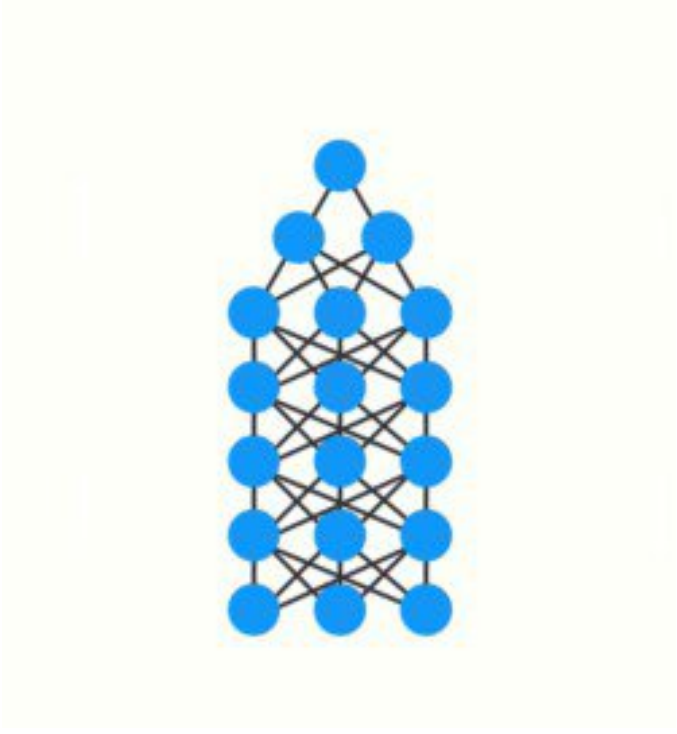
Исчезающий градиент

Возможно, вы заметили странное распределение цветов в скрытых состояниях. Это иллюстрирует проблему с RNN, известную как кратковременная память.

Окончательное скрытое состояние RNN

Кратковременная память вызвана печально известной проблемой исчезающего градиента, которая также распространена в других архитектурах нейронных сетей. Поскольку RNN обрабатывает больше шагов, у нее возникают проблемы с сохранением информации из предыдущих шагов. Как видите, информация от слов «что» и «время» на последнем временном шаге практически отсутствует. Кратковременная память и исчезающий градиент обусловлены природой обратного распространения ошибки – алгоритмом, который используется для обучения и оптимизации нейронных сетей. Чтобы понять, почему это так, давайте посмотрим на влияние обратного распространения на нейронную сеть с глубокой прямой связью.

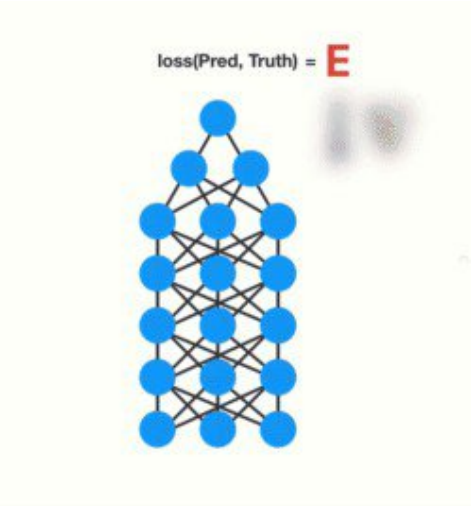
Обучение нейронной сети состоит из трех основных этапов. Сначала она делает проход вперед и делает прогноз.Затем она сравнивает прогноз с истиной, используя функцию потерь. Функция потерь выводит значение ошибки, которое является оценкой того, насколько плохо работает сеть. Наконец, она использует это значение ошибки для обратного распространения, которое вычисляет градиенты для каждого узла в сети.



Градиент — это значение, используемое для корректировки внутренних весов сети, позволяющее сети обучаться. Чем больше градиент, тем больше корректировка и наоборот. Вот в чем проблема. При обратном распространении каждый узел в слое

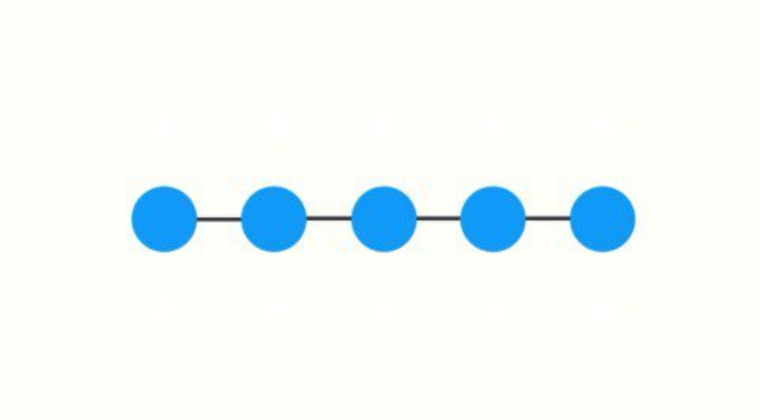
вычисляет свой градиент с учетом эффектов градиентов в слое перед ним. Таким образом, если корректировки предыдущих слоев невелики, то корректировки текущего слоя будут еще меньше.

Это приводит к экспоненциальному уменьшению градиентов при обратном распространении вниз. Более ранние слои не могут обучаться, поскольку внутренние веса практически не корректируются из-за чрезвычайно малых градиентов. И это проблема исчезающего градиента.



Градиенты уменьшаются по мере обратного распространения вниз.

Давайте посмотрим, как это применимо к рекуррентным нейронным сетям. Вы можете рассматривать каждый временной шаг в рекуррентной нейронной сети как слой. Чтобы обучить рекуррентную нейронную сеть, вы используете приложение обратного распространения ошибки, называемое обратным распространением ошибки во времени. Значения градиента будут экспоненциально уменьшаться по мере прохождения каждого временного шага.



Градиенты уменьшаются по мере обратного распространения во времени.

Опять же, градиент используется для корректировки весов нейронных сетей, что позволяет им обучаться. Небольшие градиенты означают небольшие корректировки. Это приводит к тому, что ранние слои не обучаются.
Из-за исчезновения градиентов RNN не изучает долгосрочные зависимости на разных временных шагах. Это означает, что существует вероятность того, что слова «что» и «время» не учитываются при попытке предсказать намерение пользователя. Затем сеть должна сделать наилучшее предположение с помощью ответа «это?». Это довольно двусмысленно и было бы сложно даже для

человека. Таким образом, неспособность обучаться на более ранних временных шагах приводит к тому, что сеть имеет кратковременную память.

LSTM и GRU

Итак, RNN страдают от кратковременной памяти, так как же нам с этим бороться? Чтобы смягчить кратковременную память, были созданы две специализированные рекуррентные нейронные сети. Один из них назывался «Долгая краткосрочная память» или сокращенно LSTM. Другой — закрытые рекуррентные сети или GRU. LSTM и GRU по существу функционируют так же, как RNN, но они способны изучать долгосрочные зависимости, используя механизмы, называемые «воротами». Эти ворота представляют собой различные тензорные операции, которые могут узнать, какую информацию добавить или удалить в скрытое состояние. Благодаря этой способности кратковременная память не представляет для них особой проблемы.

Подводя итог, можно сказать, что RNN хороши для обработки данных последовательностей для прогнозирования, но страдают от кратковременной памяти. Проблема с кратковременной памятью ванильных RNN не означает их полного пропуска и использования более развитых версий, таких как LSTM или GRU. Преимущество RNN заключается в более быстром обучении и использовании меньше вычислительных ресурсов. Это потому, что требуется вычислить меньше тензорных операций. Вам следует использовать LSTM или GRU, если вы планируете моделировать более длинные последовательности с долгосрочными зависимостями.

Заключение

Мы коснулись лишь верхушки айсберга в изучении RNN. Рекуррентные сети предоставляют множество возможностей для решения разнообразных задач, и их применение огромно в таких областях как:

- обработка естественного языка,
- распознавание речи,
- временные ряды
- искусственного интеллекта.