1. Аннотация

Расчётно-пояснительная записка 20 с., 10 рис..

В курсовом работе поставлена задача: оптимальное обучение рекуррентных нейронных сетей с целью построения итерации каскада. Для решения данной задачи было проведено исследование технологий нейронных сетей.

2. Оглавление

1. Аннотация 1
2. Оглавление 2
3. Виды рекуррентных нейронных сетей 3
4. Выбор РНС в соответствии с поставленной задачей 7
5. Построение РНС 8
6. Принцип максимального правдоподобия 10
7. Softmax 12
8. Построение и обучение РНС 16
9. Алгоритм обратного распространения ошибки 18
10. Обучение РНС методом обратного распространения ошибки 19

3. 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)

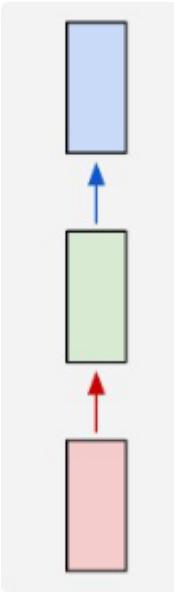

Рис. 1. РНН с типом связи один к одному

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

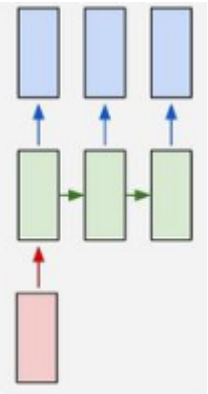

Рис. 2. РНН с типом связи один ко многим

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

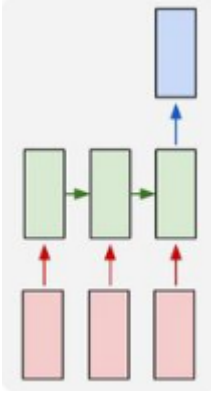

Рис. 3. РНН с типом связи многие к одному

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

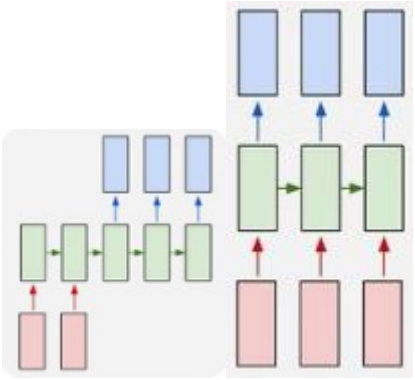

Рис. 4. РНН с типом связи многие ко многим

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.

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.

4. Выбор РНС в соответствии с поставленной задачей

Представим поставленную задачу в виде развернутого графа:

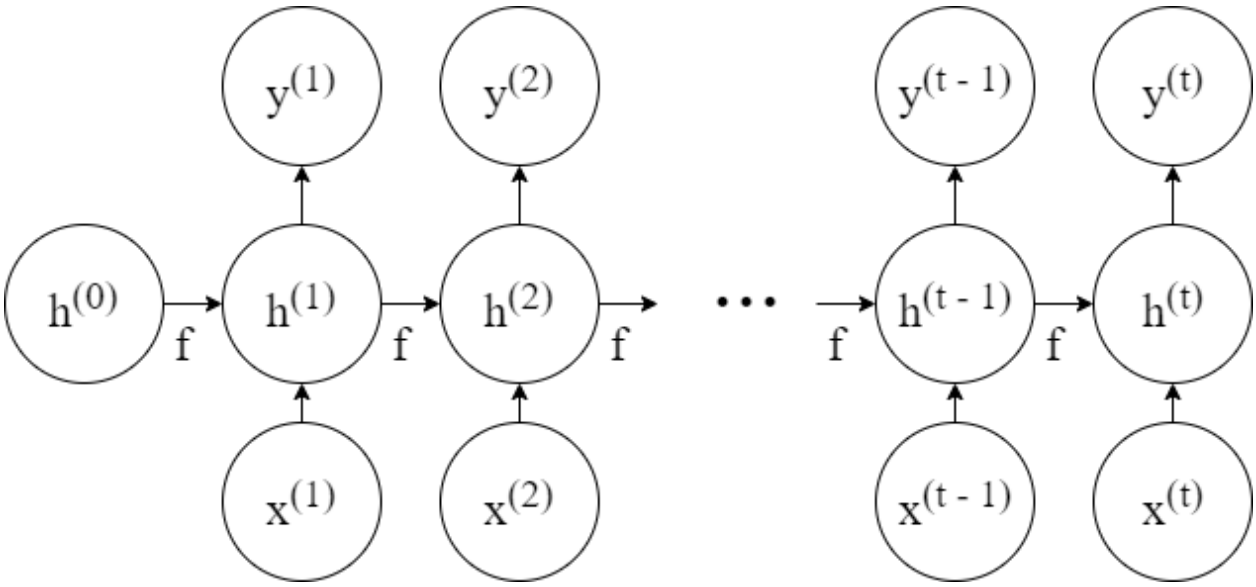

Рис. 5. Развернутый граф в соответствии с поставленной задачей, где $x^{(i)}$ - i-ый входной кадр, а $y^{(i)}$ - i-ый выходной кадр, $i \in [1, \dots, t]$

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.

5. Построение РНС

РНС с типом связи многие ко многим:

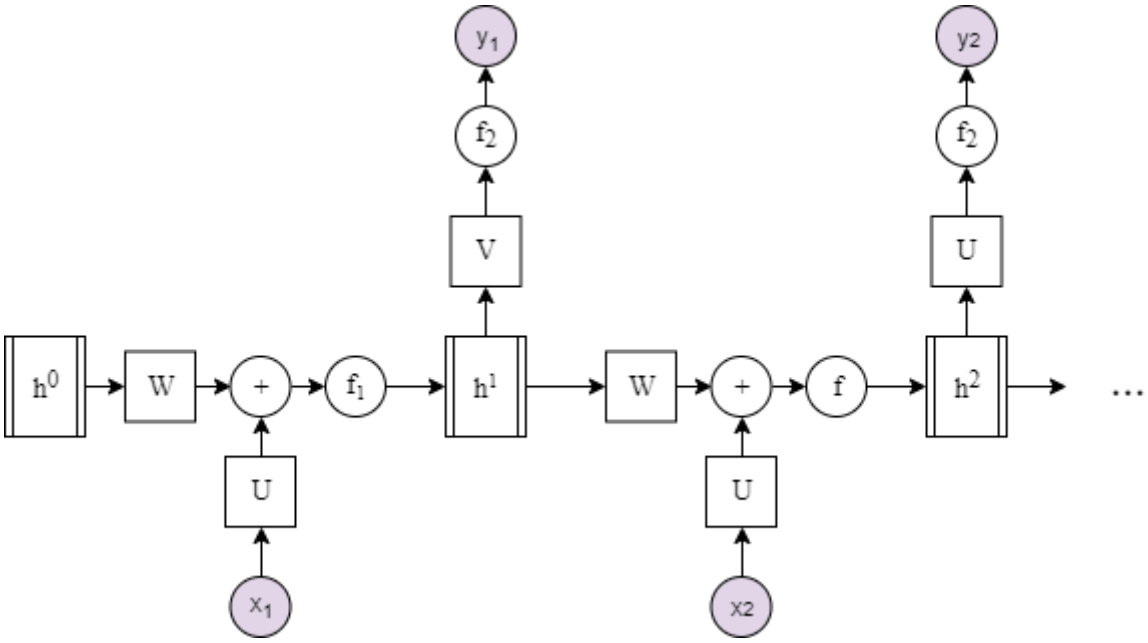

Рис. 6. РНС с типом связи многие ко многим, где h^i - состояние системы, x_i – входные параметры, f_1 – функция активации для скрытых блоков, f_2 – функция выхода, W, U, V – матрицы весов,

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

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

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

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

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

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.

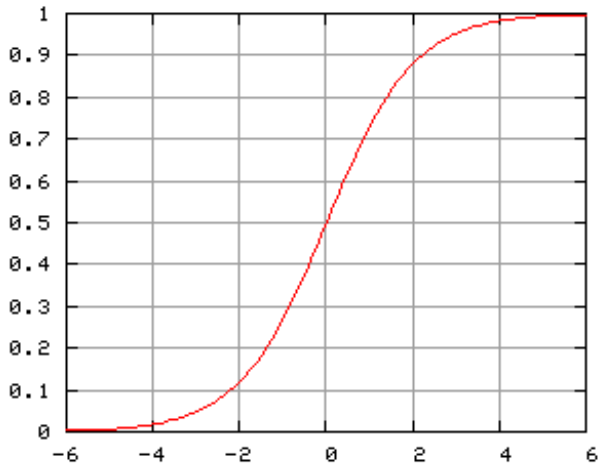

Рис. 7. График функции $\frac{1}{1+e^{-x}}$ – сигмоид

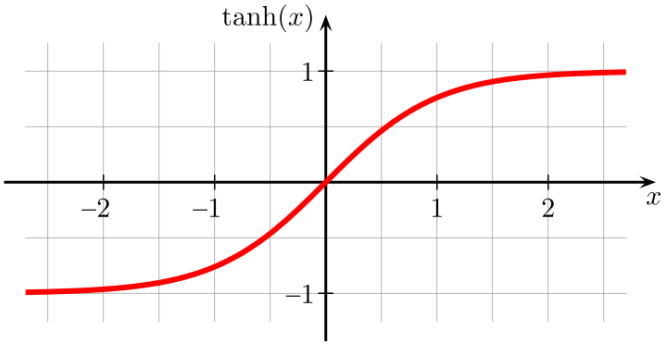

Рис. 8. График функции $\frac{2}{1+e^{-2x}} - tanh(x)$

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.

6. 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*.

7. 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. 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:

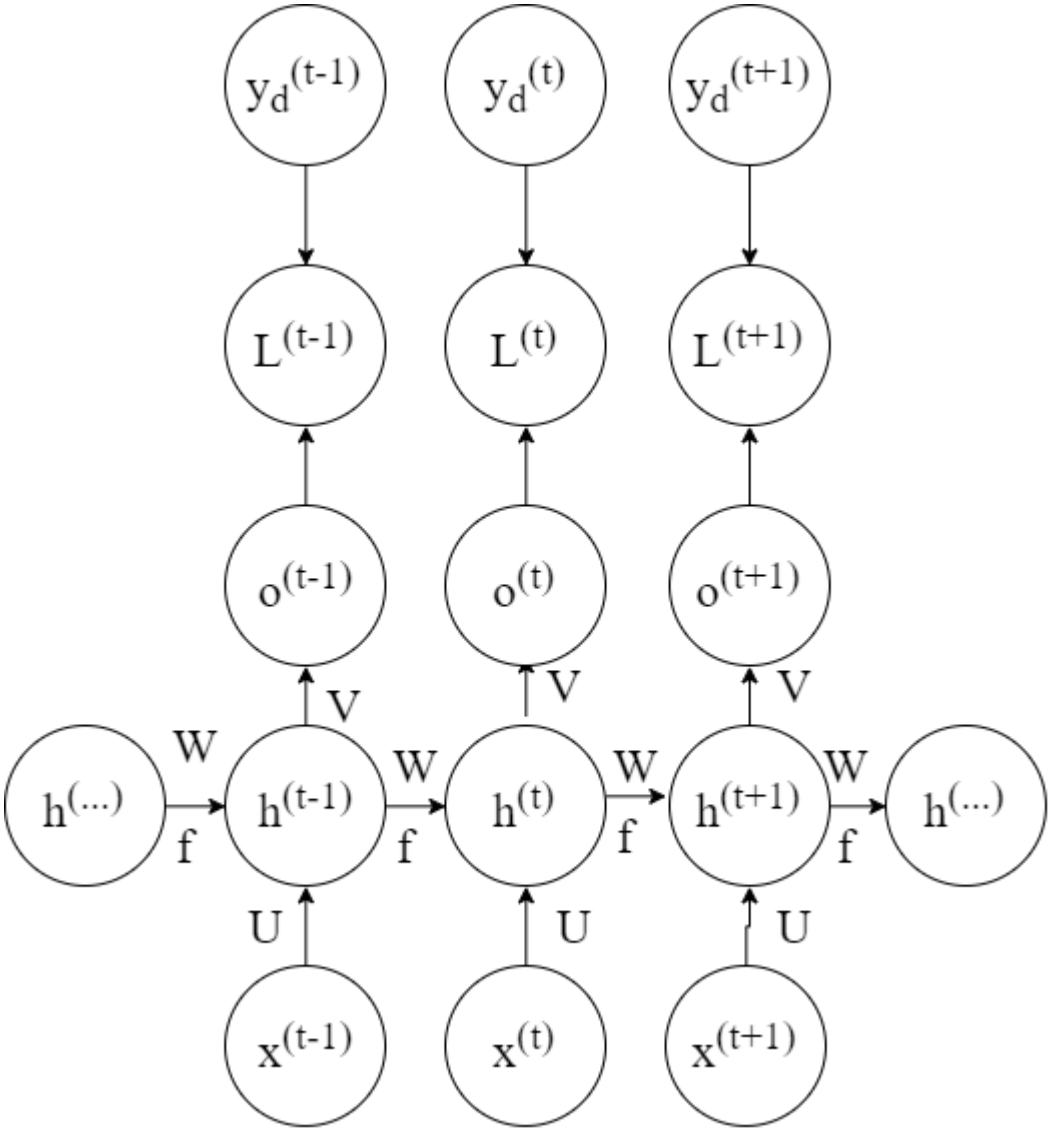

Рис. 9. Граф вычислений потерь при обучении рекуррентной сети,

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 y_d. 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 y_d.

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.

9. 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).

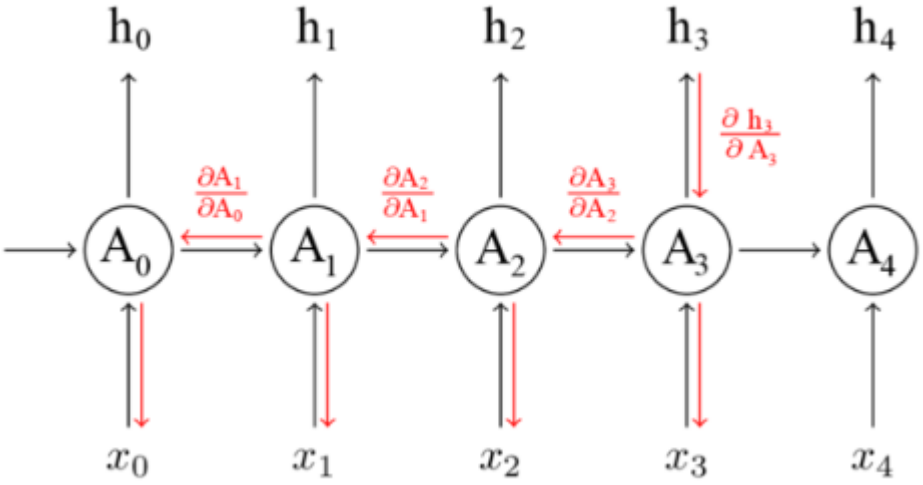

Рис. 10. Общий вид алгоритма обратного распространения ошибки для РНС.

10. 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:

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

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

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

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

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

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