

Титульный лист материалов по дисциплине
(заполняется по каждому виду учебного материала)

ДИСЦИПЛИНА	Проектирование и обучение нейронных сетей <small>(полное наименование дисциплины без сокращений)</small>
ИНСТИТУТ	Информационные технологии
КАФЕДРА	Вычислительная техника <small>полное наименование кафедры</small>
ВИД УЧЕБНОГО МАТЕРИАЛА	Лекция <small>(в соответствии с пп.1-11)</small>
ПРЕПОДАВАТЕЛЬ	Сорокин Алексей Борисович <small>(фамилия, имя, отчество)</small>
СЕМЕСТР	7 семестр, 2023/2024 <small>(указать семестр обучения, учебный год)</small>

11. ЛЕКЦИЯ. Рекуррентные сети

Сеть Хемминга

Сеть Хемминга – это трехслойная рекуррентная структура, которую можно считать развитием сети Хопфилда, была предложена Р. Липпманом. Она позиционируется как специализированное гетероассоциативное запоминающее устройство. Основная идея функционирования сети состоит в минимизации расстояния Хемминга между тестовым вектором, подаваемым на вход сети, и векторами обучающих выборок, закодированными в структуре сети. Обобщенная структура сети Хемминга представлена на рисунке 4.

Первый ее слой имеет однонаправленное распространение сигналов от входа к выходу и фиксированные значения весов. Второй слой, MAXNET, состоит из нейронов, связанных обратными связями по принципу «каждый с каждым», при этом в отличие от структуры Хопфилда существует ненулевая связь входа нейрона со своим собственным выходом. Веса нейронов в слое MAXNET постоянны. Разные нейроны связаны отрицательной обратной связью с весом $-\varepsilon$, при этом обычно величина ε обратно пропорциональна числу образов. С собственным выходом нейрон связан положительной обратной связью с весом $+1$. Веса пороговых элементов равны нулю. Нейроны этого слоя функционируют в режиме WTA (Winner Takes All — победитель получает все), при котором в каждой фиксированной ситуации активизируется только один нейрон. Выходной однонаправленный слой формирует выходной вектор, в котором только один нейрон имеет выходное значение, равное 1, а все остальные – равные 0.

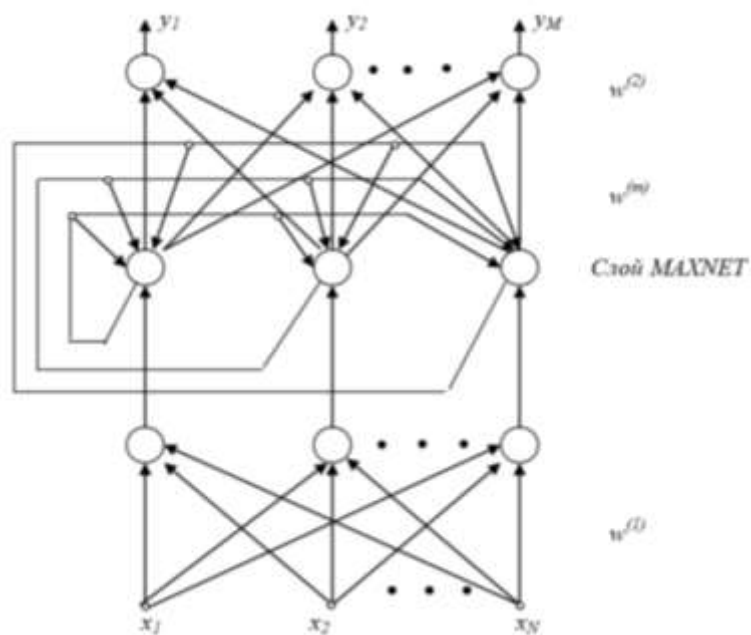


Рис. 4. Обобщенная структура сети Хемминга

Веса первого слоя соответствуют входным векторам-образцам $x^{(t)}$, поэтому $w_{ij}^{(1)} = x_j^{(t)}$ для $p, t = 1, 2, \dots, p, i = t, j = 1, 2, \dots, N$, то есть веса первого нейрона запоминают компоненты первого входного вектора. Веса второго нейрона – компоненты второго вектора и т. д.

Аналогично веса выходного слоя соответствуют очередным векторам образов $y^{(t)}$, $w_{li}^{(2)} = y_l^{(t)}$, $i, t, l = 1, 2, \dots, p$.

В случае нейронов слоя MAXNET, функционирующих в режиме WTA, веса сети должны усиливать собственный сигнал нейрона и ослаблять остальные сигналы. Для достижения этого принимается $w_{ij}^{(m)} = 1$, а также

$$-\frac{1}{p-1} < w_{is}^{(m)} < 0 \quad (16)$$

для $i \neq s$. Для обеспечения абсолютной сходимости алгоритма веса $w_{ij}^{(m)}$ должны отличаться друг от друга. Р. Липпман в своей работе принял

$$w_{is}^{(m)} = -\frac{1}{p-1} + \xi, \quad (17)$$

где ξ - случайная величина с достаточно малой амплитудой. В процессе функционирования сети в режиме распознавания можно выделить три фазы. В первой из них на вход подается N -элементный вектор x . После предъявления этого вектора на выходах нейронов первого слоя генерируются сигналы, задающие начальные состояния нейронов второго слоя. Нейроны первого слоя рассчитывают расстояния Хемминга между поданными на вход сети вектором x и векторами весов $w^{(t)} = x^{(t)}$ отдельных нейронов этого слоя. Значения выходных сигналов этих нейронов определяются по формуле:

$$\hat{y}_i = 1 - \frac{d_H(x^{(t)}, x)}{N}, \quad (18)$$

где $d_H(x^{(t)}, x)$, обозначает расстояние Хемминга между входными векторами $x^{(t)}$ и x , то есть число битов, на которое различаются эти два вектора. Значение $\hat{y}_i = 1$, если $x = x^{(t)}$, и $\hat{y}_i = 0$, если $x = -x^{(t)}$. В остальных случаях значения \hat{y}_i лежат в интервале $[0, 1]$.

Сигналы \hat{y}_i нейронов первого слоя становятся начальными состояниями нейронов слоя MAXNET на второй фазе функционирования сети.

Во второй фазе инициировавшие MAXNET сигналы удаляются, и из сформированного ими начального состояния запускается итерационный процесс. Итерационный процесс завершается в момент, когда все нейроны,

кроме нейрона-победителя с выходным сигналом не равным 0, перейдут в нулевое состояние. Задача нейронов этого слоя состоит в определении победителя, то есть нейрона, у которого выходной сигнал отличен от 0. Процесс определения победителя выполняется согласно формуле:

$$y_i(t) = f\left(\sum_s w_{is}^{(m)} y_s(t-1)\right) = f\left(y_i(t-1) + \sum_{s \neq i} w_{is}^{(m)} y_s(t-1)\right) \quad (19)$$

при начальном значении $y_s = \hat{y}_i$. Функция активации $f(y)$ нейронов слоя MAXNET задается выражением:

$$f(y) = \begin{cases} y & \text{для } y \geq 0 \\ 0 & \text{для } y < 0 \end{cases} \quad (20)$$

Итерационный процесс (19) завершается в момент, когда состояние нейронов стабилизируется, и активность продолжает проявлять только один нейрон, тогда как остальные пребывают в нулевом состоянии. Активный нейрон становится победителем и через веса $w_{is}^{(2)}$ ($s = 1, 2, \dots, N$) линейных нейронов выходного слоя представляет вектор $y^{(t)}$, который соответствует номеру вектора $x^{(t)}$, признанному слоем MAXNET в качестве ближайшего к входному вектору x .

В третьей фазе этот нейрон посредством весов, связывающих его с нейронами выходного слоя, формирует на выходе сигнал, равный 1, его номер является номер входного образца, к которому принадлежит входной вектор.

Входные узлы сети принимают значения, задаваемые аналогичными компонентами вектора x . Нейроны первого слоя рассчитывают расстояние Хемминга между входным вектором x и каждым из p закодированных векторов-образцов $x^{(t)}$, образующих веса нейронов первого слоя. Нейроны в слое MAXNET выбирают вектор с наименьшим расстоянием Хемминга, определяя, таким образом, класс, к которому принадлежит предъявленный входной вектор x . Веса нейронов выходного слоя формируют вектор, соответствующий классу входного вектора. При p нейронах первого слоя, емкость запоминающего устройства Хемминга также равна p , так как каждый нейрон представляет единственный класс. Важным достоинством сети Хемминга считается небольшое, по сравнению с сетью Хопфилда, число взвешенных связей между нейронами. Так, например, 100-входная сеть Хопфилда, кодирующая 10 различных векторных классов, должна содержать 10000 взвешенных связей, тогда как аналогичная сеть Хемминга содержит 1100 связей, из которых 1000 весов находятся в первом слое, а 100 – в слое MAXNET.

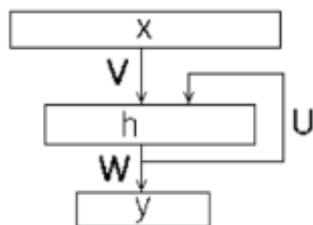
Простые рекуррентные нейронные сети. Сеть Элмана

Рекуррентную нейронную сеть Элмана так же называют Simple Recurrent Neural Network (Simple RNN, SRN). Существуют актуальные задачи обработки данных, при решении которых требуется анализировать последовательности объектов, при этом порядок следования объектов играет существенную роль в задаче. Например, это задача распознавания речи, где рассматриваются последовательности звуков или задачи обработки текстов на естественном языке, где анализируются последовательности слов.

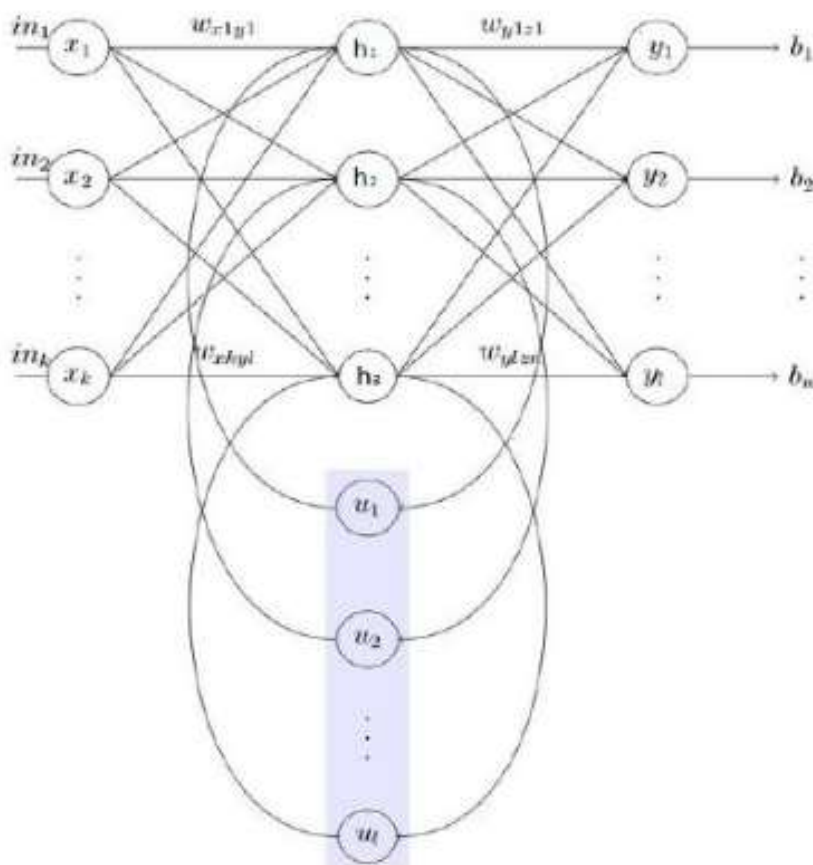
Для решения такого рода задач можно применять рекуррентные нейронные сети, которые в процессе работы могут сохранять информацию о своих предыдущих состояниях.

Искусственная нейронная сеть Элмана состоит из трех слоёв - входного (распределительного) слоя, скрытого и выходного (обрабатывающего). При этом скрытый слой имеет обратную связь сам на себя.

Схема нейронной сети Элмана имеет вид:



Общий вид сети Элмана:



В отличие от обычной сети прямого распространения, входной образ рекуррентной сети это не один вектор, а последовательность векторов $\{x(1), \dots, x(n)\}$. Векторы входного образа в заданном порядке подаются на вход, при этом новое состояние скрытого слоя зависит от его предыдущих состояний. Сеть Элмана можно описать следующими соотношениями:

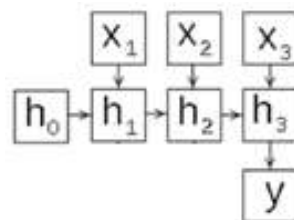
$$h(t) = f(V \times x(t) + U \times h(t-1) + b_h)$$

$$y(t) = g(V \times h(t) + b_y)$$

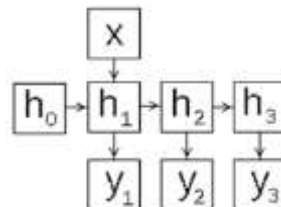
$x(t)$ - входной вектор, $h(t)$ - состояние скрытого слоя для входа $x(t)$ ($h(0) = 0$), $y(t)$ - выход сети для входа $x(t)$, U - весовая матрица распределительного слоя, W - весовая (квадратная) матрица обратных связей скрытого слоя, b_h - вектор сдвигов скрытого слоя, V - весовая матрица выходного слоя, b_y - вектор сдвигов выходного слоя, f - функция активации скрытого слоя, g - функция активации выходного слоя.

В зависимости от того как сформировать вход и выход рекуррентной сети, можно разными способами задать схему её работы. Развернём схему рекуррентной сети во времени:

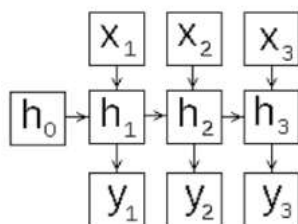
а) **«много в один»** (many-to-one) - скрытый слой последовательно изменяет своё состояние, из его конечного состояния вычисляется выход сети, используется для классификации текстов;



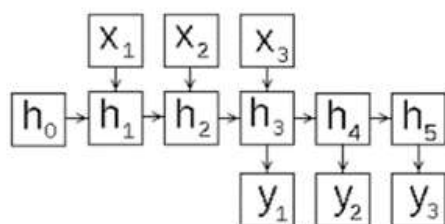
б) **«один во много»** (one-to-many) - скрытый слой инициализируется одним входом, из цепочки его последующих состояний генерируются выходы сети, используется для аннотирования изображений;



в) **«много во много»** (many-to-many) - на каждый вход сеть выдаёт выход, который зависит от предыдущих входов, используется для классификации видео;



d) «**много во много**» (many-to-many) - скрытый слой последовательно изменяет своё состояние, его конечное состояние служит инициализацией для выдачи цепочки результатов, используется для создания систем машинного перевода и чат-ботов.

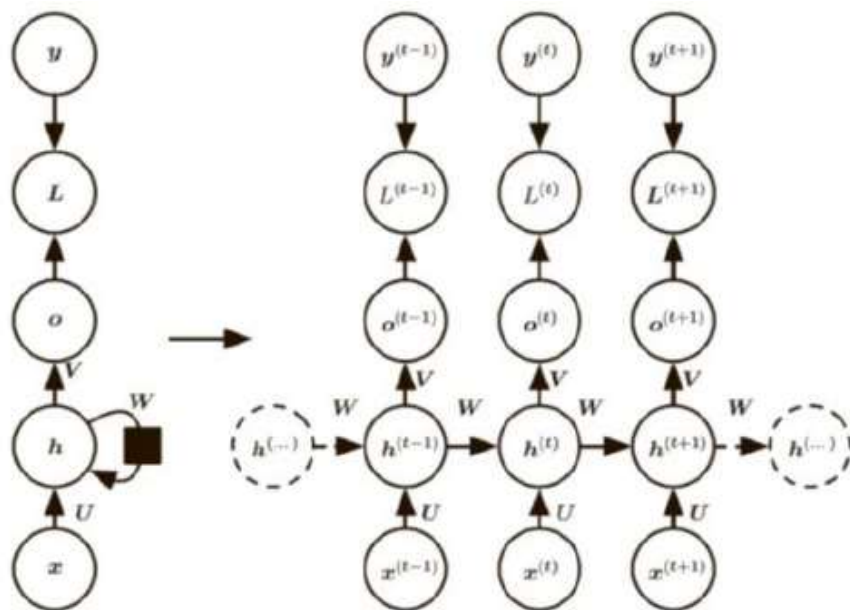


Рассмотрим метод обучения рекуррентной нейронной сети Элмана по схеме «много в один» (many-to-one) для реализации классификатора объектов, заданных последовательностями векторов. Для обучения сети Элмана применяются те же градиентные методы, что и для обычных сетей прямого распространения, но с определёнными модификациями для корректного вычисления градиента функции ошибки. Он вычисляется с помощью модифицированного метода обратного распространения, который носит название Backpropagation through time (метод обратного распространения с разворачиванием сети во времени, BPTT).

Идея метода - развернуть последовательность, превратив рекуррентную сеть в «обычную».

Метод обучения рекуррентной нейронной сети (BPTT)

Метод обучения рекуррентной нейронной сети (BPTT) Рассмотрим рекуррентную нейронную сеть и ее функцию потерь, представленную в виде рекуррентных связей (слева). И то же в виде развернутого во времени графа вычислений, в котором каждая вершина ассоциирована с одним моментом времени (справа).

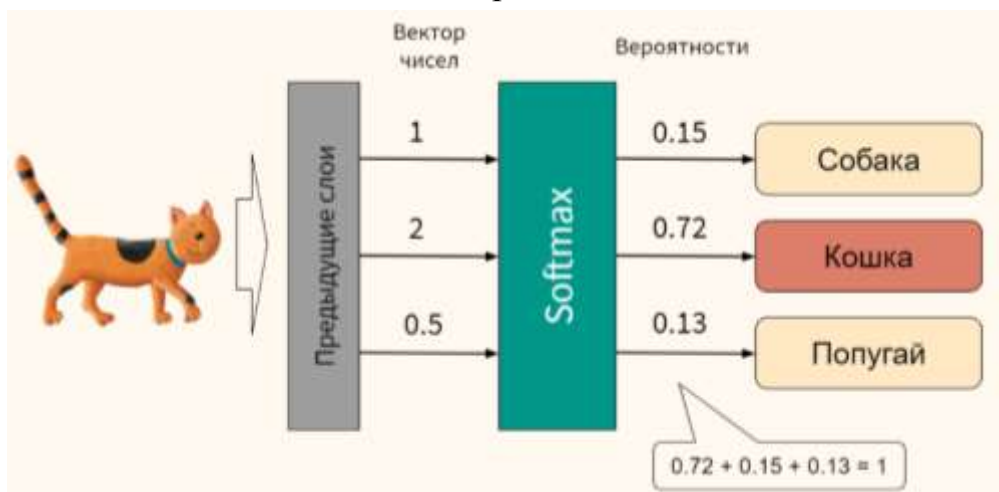


Представленный граф – это граф вычислений потерь при обучении рекуррентной сети, которая отображает входную последовательность значений x в соответствующую выходную последовательность значений o . Функция потерь L измеряет, насколько далеко каждый элемент o отстоит от соответствующей метки y . В случае применения к выходам функции **softmax** можно предполагать, что o – ненормированные логарифмические вероятности. Внутри функция L вычисляет **softmax**(o) и сравнивает эту величину с меткой y .

Поговорим немного о функции **softmax**

Если сигмоида подходит для бинарной классификации, то **softmax** применяется для задач многоклассовой классификации. Т.е. **softmax** — это логистическая функция для многомерного случая.

Она может гарантировать, что сумма всех выходных нейронов равна 1, а значение интервала $[0,1]$, соответствующего каждому выходу, является вероятностью выхода, а наибольшая вероятность используется в приложении. Результатом является окончательный прогноз.



Форма функции Softmax следующая:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}$$

K - указывает на количество классов.

Для того, чтобы было понятно, как работает эта формула, рассмотрим простой пример, в котором нужно найти принадлежность объекта к одному из пяти классов. Допустим, на последнем слое мы посчитали z по формуле:

$$z_i = w_i a_i + b_i$$

и получили вектор значений:

$$z_i = \begin{bmatrix} 1 \\ -3 \\ 2 \\ 3 \\ -1 \end{bmatrix}$$

Сначала посчитаем, знаменатель формулы, то есть, просуммируем все элементы вектора:

$$\begin{bmatrix} e^1 \\ e^{-3} \\ e^2 \\ e^3 \\ e^{-1} \end{bmatrix} = \begin{bmatrix} 2,72 \\ 0,05 \\ 7,39 \\ 20,09 \\ 0,37 \end{bmatrix}$$

значение при этом окажется приблизительно равным 30,62:

$$\sum_{k=1}^5 e^{z_k} = 30,62$$

Остаётся каждое из значений вектора поделить на получившуюся сумму, что на выходе даст вот такой вектор:

$$\begin{bmatrix} 0,09 \\ 0,001 \\ 0,24 \\ 0,66 \\ 0,01 \end{bmatrix}$$

В итоге, с вероятностью 66% объект из этого примера можно отнести к третьему классу (если счёт начинать с нуля).

Продолжим

В РНС имеются связи между входным и скрытым слоями, параметризованные матрицей весов U , рекуррентные связи между скрытыми блоками, параметризованные матрицей весов W , и связи между скрытым и выходным слоями, параметризованные матрицей весов V .

Рассмотренная сеть универсальна в том смысле, что любая функция, может быть вычислена и такой рекуррентной сетью конечного размера. Результат можно прочесть из РНС после выполнения числа шагов, асимптотически линейно зависящего от длины входной последовательности.

Когда РНС используется как машина Тьюринга, она принимает на входе двоичную последовательность, а ее выходы можно дискретизировать (представление непрерывной функции дискретной совокупностью её значений при разных наборах аргументов) для получения двоичного результата. «Входом» машины Тьюринга является спецификация вычисляемой функции, поэтому той же сети, которая моделирует машину Тьюринга, достаточно для решения всех задач. Теоретическая РНС, применяемая в доказательстве, умеет моделировать неограниченный стек, представляя его активации и веса рациональными числами неограниченной точности.

Теперь выведем уравнения прямого распространения для РНС. Будем предполагать, что в качестве функции активации используется гиперболический тангенс. Кроме того, будем предполагать, что выход дискретный, как если бы РНС применялась для предсказания слов или символов. Естественный способ представления дискретных величин – рассматривать выход как ненормированные логарифмические вероятности каждого возможного значения дискретной величины. Тогда на этапе постобработки можно применить операцию *softmax* и получить вектор \hat{y} нормированных вероятностей. Прямое распространение начинается с задания начального состояния $h^{(0)}$.

Затем для каждого временного шага от $t = 1$ до $t = \tau$ применяем следующие уравнения обновления:

$$\begin{aligned} a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)} & o^{(t)} &= c + Vh^{(t)} \\ h^{(t)} &= \tanh(a^{(t)}) & \hat{y}^{(t)} &= \text{softmax}(o^{(t)}) \end{aligned} \quad (1)$$

где параметрами являются векторы смещения b и c , а также матрицы весов U , V и W соответственно для связей между входным и скрытым слоями, между скрытым и выходным слоями и между скрытыми блоками. Это пример рекуррентной сети, которая отображает входную последовательность на выходную той же длины. Полная потеря для данной входной последовательности x в совокупности с последовательностью меток y равна сумме потерь по всем временным шагам.

Например, если $L(t)$ – отрицательное логарифмическое правдоподобие $y(t)$ при условии $x(1), \dots, x(t)$, то

$$L(\{x^{(1)}, \dots, x^{(t)}\}, \{y^{(1)}, \dots, y^{(t)}\}) = \sum_t L^{(t)} =$$

$$= -\sum \log p_{model}(y^{(t)}|\{x^{(1)}, \dots, x^{(t)}\}) \quad (2)$$

где $p_{model}(y^{(t)}|\{x^{(1)}, \dots, x^{(t)}\})$ берется из элемента $y^{(t)}$ выходного вектора модели $\hat{y}^{(t)}$. Вычисление градиента этой функции потерь относительно параметров – дорогая операция. Для вычисления градиента необходимо выполнить прямое распространение, двигаясь слева направо по развернутому графу, а затем обратное распространение, двигаясь справа налево по тому же графу. Время работы имеет порядок $O(\tau)$, и его нельзя уменьшить за счет распараллеливания, потому что граф прямого распространения принципиально последовательный.

Состояния, вычисленные во время прямого прохода, необходимо хранить до повторного использования на обратном проходе, поэтому объем потребляемой памяти также имеет порядок $O(\tau)$.

Применим обобщенный алгоритм обратного распространения к развернутому графу вычислений. Градиенты, полученные в результате обратного распространения, можно затем использовать в сочетании с любым универсальным градиентным методом для обучения РНС.

Рассмотрим пример вычисления градиентов для уравнений (1) и (2) РНС. В рассматриваемом графе вычислений имеются параметры U, V, W, b и c , а также последовательность вершин, индексированных временем t . Для каждой вершины N можно рекурсивно вычислить градиент, зная градиенты, вычисленные в вершинах, следующих за ней в графе. Рекурсия начинается с вершин, непосредственно предшествующих окончательной потере: $\partial L / \partial L^{(t)} = 1$. Предполагается, что выходы $o^{(t)}$ используются в качестве аргумента функции *softmax* для получения вектора вероятностей выходов \hat{y} . Предполагается также, что функция потерь – это отрицательное логарифмическое правдоподобие истинной метки $y^{(t)}$ при известных к этому моменту входах. Градиент $\nabla o(t)L$ по выходам в момент t для всех i, t имеет вид:

$$(\nabla_{o(t)} L) = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - 1_{i,y^{(t)}}$$

Мы движемся в направлении от конца последовательности к началу. В последний момент времени τ у $h^{(\tau)}$ есть только один потомок, поэтому вычислить градиент можно: $\nabla_{h(\tau)} L = V^T \nabla_{o(\tau)} L$.

Затем можно совершать итерации назад во времени для обратного распространения градиентов от $t = \tau - 1$ до $t = 1$. Заметим, что потомками $h^{(t)}$ (для $t < \tau$) являются $o^{(t)}$ и $h^{(t-1)}$.

Следовательно, градиент равен

$$\begin{aligned}\nabla_{h^{(t)}} L &= \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}} \right)^T (\nabla_{h^{(t+1)}} L) \left(\frac{\partial o^{(t)}}{\partial h^{(t)}} \right)^T (\nabla_{o^{(t)}} L) \\ &= W^T (\nabla_{h^{(t+1)}} L) \text{diag} \left(1 - (h^{(t+1)})^2 \right) + V^T (\nabla_{o^{(t)}} L)\end{aligned}$$

где $\text{diag} \left(1 - (h^{(t+1)})^2 \right)$ – диагональная матрица с элементами $1 - (h^{(t+1)})^2$. Это якобиан функции гиперболического тангенса, ассоциированный со скрытым блоком i в момент $t + 1$. Получив градиенты по внутренним вершинам графа вычислений, мы можем затем получить градиенты по вершинам параметров. Поскольку параметры разделяются между временными шагами, следует аккуратно подходить к обозначениям аналитических операций с участием этих переменных. Необходимо вычислять вклад в градиент одного ребра графа вычислений.

Но оператор $\nabla_W f$, применяемый в математическом анализе, принимает во внимание вклад W в значение f , вносимый всеми ребрами графа вычислений. Для разрешения этой неоднозначности введем фиктивные переменные $W(t)$, определенные как копии W , только каждая $W(t)$ используется лишь на временном шаге t . Тогда $\nabla_{W(t)} f$ можно использовать для обозначения вклада весов на шаге t в градиент. В этой нотации градиенты по остальным параметрам имеют вид

$$\begin{aligned}\nabla_c L &= \sum_t \left(\frac{\partial o^{(t)}}{\partial c} \right)^T \nabla_{o^{(t)}} L = \sum_t \nabla_{o^{(t)}} L \\ \nabla_b L &= \sum_t \left(\frac{\partial h^{(t)}}{\partial b^{(t)}} \right)^T \nabla_{h^{(t)}} L = \sum_t \text{diag} \left(1 - (h^{(t+1)})^2 \right) \nabla_{h^{(t)}} L \\ \nabla_V L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{V o_i^{(t)}} = \sum_t (\nabla_{o^{(t)}} L) h^{(t)} \\ \nabla_W L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{W^t h_i^{(t)}} = \sum_t \text{diag} \left(1 - (h^{(t)})^2 \right) (\nabla_{h^{(t)}} L) h^{(t-1)} \\ \nabla_U L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{U^t h_i^{(t)}} = \sum_t \text{diag} \left(1 - (h^{(t)})^2 \right) (\nabla_{h^{(t)}} L) x^{(t)}\end{aligned}$$

Нам не нужно вычислять градиент по $x(t)$ для обучения, потому что среди его предков в графе вычислений, определяющем потерю, нет параметров.

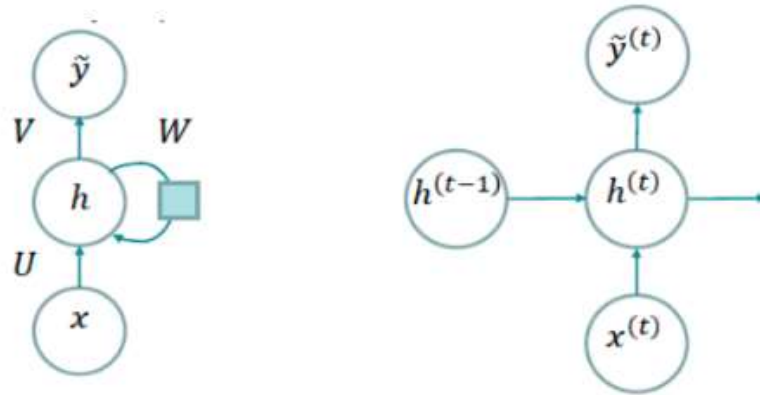
Глубокие рекуррентные нейронные сети

Вычисления в большинстве типовых рекуррентных сетей можно разложить на три блока параметров и соответствующих им преобразований: входа в скрытое

состояние; преобразование от предыдущего скрытого состояния в следующее скрытое состояние; преобразование скрытого состояния.

Обычная рекуррентная сеть описывается следующими соотношениями:

$$h^{(t)} = f(Ux^{(t)} + Wh^{(t-1)} + b), \quad \tilde{y}^{(t)} = g(Vh^{(t)} + c)$$



В зависимости от того, насколько сложными (глубокими) являются преобразования, выделяется несколько типов рекуррентных сетей

1 Рекуррентная сеть с глубоким преобразованием входного сигнала в скрытый (Deep Transition RNN, DT-RNN)

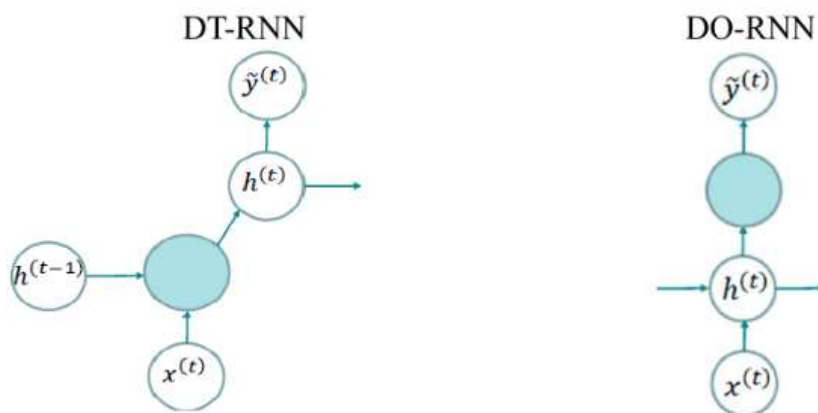
$$h^{(t)} = f(Ux^{(t)} + Wh^{(t-1)} + b) = \varphi_L \left(U_L^T \varphi_{L-1} \left(U_{L-1}^T \varphi_{L-2} \left(\dots \varphi_1 (Ux^{(t)} + Wh^{(t-1)} + b) \right) \right) \right)$$

где L – количество слоев сети между входным и скрытым слоями.

2 Рекуррентная сеть с глубоким преобразованием скрытого сигнала в выходной (Deep Output RNN, DO-RNN)

$$\tilde{y}^{(t)} = g(Vh^{(t)} + c) = \psi_L \left(V_L^T \psi_{L-1} \left(V_{L-1}^T \psi_{L-2} \left(\dots \psi_1 (Vh^{(t)} + c) \right) \right) \right)$$

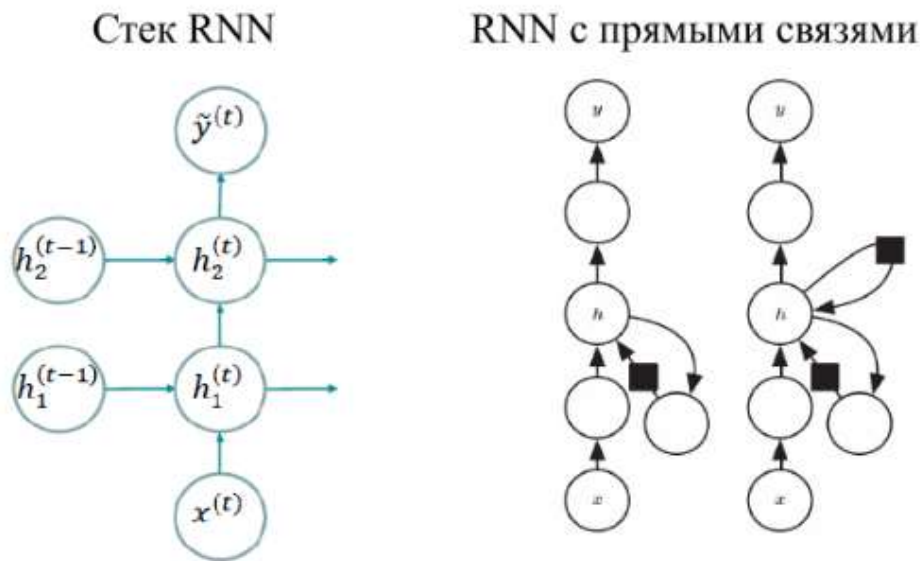
где L – количество слоев сети между скрытым и выходным слоями.



3 Стек из обычных рекуррентных сетей

$$h_l^{(t)} = f_l \left(W_l^T h_l^{(t-1)} + U_l^T h_{l-1}^{(t)} \right)$$

где $h_l^{(t)}$ – скрытое состояние системы на слое с номером l в момент времени t .
Эффект удлинения пути можно сгладить путем добавления прямых связей.



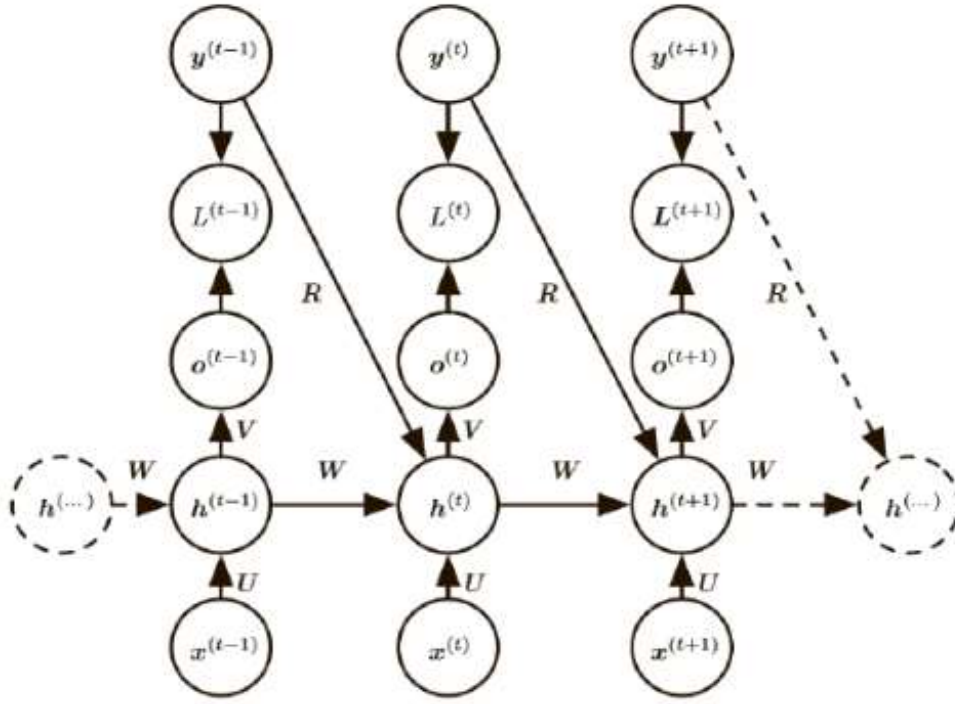
Двунаправленные РНС

Все рассмотренные до сих пор рекуррентные сети имели «каузальную» структуру, т. е. на состояние в момент t влияет только информация в прошлые моменты времени $x(1), \dots, x(t-1)$ и текущий вход $x(t)$. Некоторые модели допускают также влияние прошлых значений y на текущее состояние, если значения y доступны.

Но во многих приложениях необходимо получать предсказание $y(t)$, которое может зависеть от всей входной последовательности. Например, в задаче распознавания речи правильная интерпретация текущего звука как фонемы может зависеть от нескольких следующих фонем из-за коартикуляции (наложение артикуляции, характерной для последующего звука, на весь предшествующий звук) и даже от нескольких следующих слов из-за лингвистических зависимостей между соседними словами: если акустически допустимы две интерпретации текущего слова, то, чтобы различить их, возможно, понадобится заглянуть далеко в будущее (или в прошлое).

Двунаправленные рекуррентные нейронные сети были придуманы для удовлетворения именно этой потребности. Они оказались чрезвычайно успешными в распознавании рукописных текстов, речи и биоинформатике.

Как следует из названия, двунаправленные РНС являются комбинацией РНС, движущейся вперед во времени (от начала последовательности к ее концу), и РНС, движущейся в обратном направлении. На рисунке показана типичная двунаправленная РНС;



$h(t)$ обозначает состояние той РНС, что движется вперед, а $g(t)$ – той, что движется назад. Это позволяет выходным блокам $o(t)$ вычислять представление, которое зависит как от прошлого, так и от будущего, но наиболее чувствительно к входным значениям вблизи момента t ; при этом задавать окно фиксированного размера вокруг t необязательно (в отличие от сетей прямого распространения, сверточных сетей и обычных РНС с буфером предвыборки фиксированного размера).

Формально на момент времени t сеть можно задать следующим образом

$$\begin{aligned} s_t &= \sigma(b + W_{s_{t-1}} + U_{x_t}) & s'_t &= \sigma(b' + W'_{s'_{t+1}} + U'_{x_t}) \\ o_t &= c + V_{s_t} + V'_{s'_t} & y_t &= h(o_t) \end{aligned}$$

Рассмотренные идеи естественно обобщаются на двумерные входные данные, например изображения, для чего нужны четыре РНС, по одной в каждом направлении: вверх, вниз, влево, вправо. Тогда в каждой точке (i, j) двумерной сети выходной блок O_{ij} мог бы вычислять представление, которое улавливает, в основном, локальную информацию, но может зависеть и от удаленных входов.

По сравнению со сверточной сетью, применение РНС к изображениям обходится дороже, зато может учитывать дальние боковые взаимодействия между элементами одной карты признаков.

РНС. Блоки с утечками

Один из способов включения долгосрочных зависимостей – спроектировать модель, работающую в нескольких временных масштабах, так что одни части модели работают в мелком масштабе и могут обрабатывать мелкие детали, а

другие, работающие в крупном масштабе, эффективно передают информацию из отдаленного прошлого в настоящее.

Существуют различные стратегии построения мелких и крупных масштабов: добавление прямых связей сквозь время; «блоки с утечками», которые интегрируют сигналы с разными временными постоянными; удаление некоторых связей, используемых для моделирования мелких масштабов.

Добавление прямых связей сквозь время

Для получения грубого временного масштаба можно добавить прямые связи между переменными в отдаленном прошлом и переменными в настоящем. Идея таких прямых связей восходит к работе Лина (1996) и вытекает из идеи включения задержек в нейронные сети прямого распространения. В обыкновенной рекуррентной сети рекуррентная связь идет из блока в момент t к блоку в момент $t + 1$. Возможно построить рекуррентные сети с большими задержками.

В рекуррентных сетях, как правило, в результате применения одной и той же операции на каждом шаге длинной временной последовательности создается очень глубокий граф вычислений. При этом градиенты могут исчезать или экспоненциально расти при увеличении числа шагов. Для смягчения этой проблемы введены рекуррентные соединения с временной задержкой d . Теперь градиент экспоненциально убывает как функция τ/d , а не как функция τ .

Поскольку имеются как связи с задержкой, так и одношаговые связи, градиенты все же могут экспоненциально расти относительно τ . Это позволяет алгоритму обучения улавливать долгосрочные зависимости, хотя и не все такие зависимости хорошо представляются подобным образом.

Блоки с утечкой и спектр разных временных масштабов

Еще один способ получить пути, на которых произведение производных близко к 1, — включать блоки с линейными соединениями с самими собой (самосоединениями), имеющими веса, близкие к единице. В формуле вычисления скользящего среднего $\mu^{(t)}$ некоторой величины $v^{(t)}$: $\mu^{(t)} \leftarrow \alpha \mu^{(t+1)} + (1 - \alpha) v^{(t)}$ параметр α является примером линейного самосоединения $\mu^{(t+1)}$ с $\mu^{(t)}$. Если α близко к 1, то скользящее среднее помнит информацию о прошлом на протяжении долгого времени, а если α близко к 0, то информация о прошлом быстро забывается.

Напомню: *Скользящая средняя, скользящее среднее* — общее название для семейства функций, значения которых в каждой точке определения равны некоторому среднему значению исходной функции за предыдущий период.

Скрытые блоки с линейными самосоединениями могут вести себя аналогично скользящему среднему и называются блоками с утечкой, или протекающими.

Прямые связи с пропуском d временных шагов гарантируют, что блок сможет обучиться влиянию значений, находящихся от него на d шагов в прошлом. Использование линейных самосоединений с весом, близким к 1, – другой способ обеспечить блоку доступ к прошлым значениям. И этот подход допускает более плавную и гибкую настройку путем изменения вещественного параметра α , а не целочисленной длины пропуска.

Существуют две основные стратегии задания временных констант для настройки блоков с утечкой. Одна – вручную зафиксировать их значения, например выбрав их из некоторого распределения один раз на этапе инициализации. Вторая – сделать константы свободными параметрами и обучить их.

Удаление связей

Еще один подход к обработке долгосрочных зависимостей – организация состояния РНС в нескольких временных масштабах с целью упростить протекание информации на большое расстояние в более медленном масштабе.

Эта идея отличается от прямых связей сквозь время, поскольку подразумевается активное удаление связей длины 1 и замена их более длинными. Модифицированные таким способом блоки вынуждены работать в более протяженном временном масштабе. Прямые связи добавляют ребра. Блоки, получившие новые связи, могут обучиться работе в протяженном временном масштабе, но могут поступить и наоборот, предпочтя другие, более короткие связи.

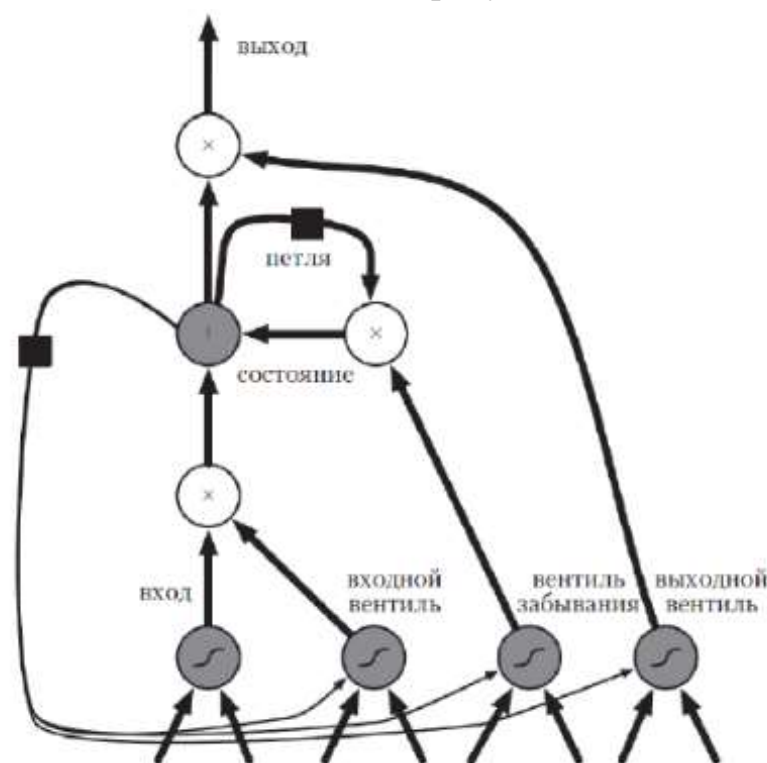
Есть несколько способов принудить группу рекуррентных блоков к работе в разных временных масштабах. Один из них – сделать рекуррентные блоки протекающими, но ассоциировать разные группы блоков с разными фиксированными временными масштабами. Другой способ – производить явные дискретные обновления в разные моменты времени с разной частотой для разных групп блоков. Этот подход показал хорошие результаты на ряде эталонных наборов данных.

Вентильные РНС. Долгая краткосрочная память (LSTM)

Долгая краткосрочная память (Long short-term memory), обычно называемая LSTM-сетями — это особый вид рекуррентных нейронных сетей, способных к запоминанию долговременных зависимостей. Они были введены Сеппом Хохрайтером и Юргеном Шмидхубером в 1997 году и были использованы и развиты многими исследователями в своих работах. Эти сети работают в широком спектре задач и довольно часто используются.

Удачная мысль о введении петель для создания путей, по которым градиент может течь длительное время, – основной вклад в первоначальную модель долгой краткосрочной памяти. Позднее было внесено важнейшее дополнение – вес петли должен быть контекстно-обусловленным, а не фиксированным. Сделав вес петли вентиляльным (управляемым другим скрытым блоком), можно динамически изменять временной масштаб интегрирования. В данном случае имеется в виду, что даже для LSTM с фиксированными параметрами временной масштаб интегрирования может изменяться в зависимости от входной последовательности, поскольку временные константы выводятся самой моделью. Идея LSTM оказалась чрезвычайно успешной во многих приложениях, например: распознавание рукописных текстов, распознавание речи, порождение рукописных текстов, машинный перевод и грамматический разбор.

Принципиальная схема LSTM показана на рисунке.



Согласно принципиальной схеме LSTM, ячейки, рекуррентно связанные между собой, заменяют стандартные скрытые блоки в обыкновенных рекуррентных сетях. Входной признак вычисляется регулярным блоком искусственного нейрона. Его значение можно аккумулировать в состоянии, если сигмоидный входной вентиль это допускает. В блоке состояния имеется линейная петля, вес которой управляется вентилем забывания. Выход ячейки можно перекрыть с помощью выходного вентиля. Во всех вентильных блоках имеется сигмоидная нелинейность, тогда как во входном блоке разрешены произвольные

сплюсывающие нелинейности. Черным квадратом обозначена задержка на одном временном шаге

Есть примеры успешного использования как представленной мелкой, так и более глубоких архитектур.

Рассмотрим схему более подробно. Вместо блока, который просто применяет поэлементную нелинейность к аффинному преобразованию входов и рекуррентным блокам, в рекуррентных LSTM-сетях имеются «LSTM-ячейки», обладающие внутренней рекуррентностью (петлей) в дополнение к внешней рекуррентности РНС.

У каждой ячейки такие же входы и выходы, как у обыкновенной рекуррентной сети, но еще имеются дополнительные параметры и система вентильных блоков, управляющих потоком информации. Самым важным компонентом является блок состояния $s_i^{(t)}$ с линейной петлей, аналогичный описанным выше блокам с утечкой.

Блок внешнего входного вентиля $g_i^{(t)}$ вычисляется аналогично вентилю забывания (с использованием сигмоиды для получения значения от 0 до 1), но со своими параметрами:

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right)$$

Выход $h_i^{(t)}$ LSTM-ячейки можно перекрыть с помощью выходного вентиля $q_i^{(t)}$ в котором также используется сигмоида:

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)}$$

$$q_i^{(t)} = \sigma \left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)} \right)$$

и b_o, U_o, W_o – смещения, веса входов и рекуррентные веса соответственно. Один из вариантов – использовать состояние ячейки $s_i^{(t)}$ как дополнительный вход (со своим весом) всех трех вентилях i -го блока. Тогда потребуются три дополнительных параметра.

РНС с вентильными рекуррентными блоками (GRU)

В недавнее время разработаны вентильные РНС, блоки которых известны также под названием вентильных рекуррентных блоков (gated recurrent unit – GRU). Основное отличие от LSTM заключается в том, что один вентильный блок одновременно управляет и коэффициентом забывания, и решением об обновлении блока состояния. Уравнения обновления имеют вид:

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} x_j^{(t-1)} h_j^{(t-1)} \right)$$

где u обозначает вентиль «обновления», а r – вентиль «сброса». Их значения определяются как обычно:

$$u_i^{(t)} = \sigma \left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)} \right)$$

$$r_i^{(t)} = \sigma \left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)} \right)$$

Вентили обновления и сброса могут «игнорировать» части вектора состояния. Вентили обновления действуют как условные интеграторы с утечкой с линейной функцией по любому измерению, т. е. могут либо скопировать вход (один конец сигмоиды), либо полностью проигнорировать его (противоположный конец), заменив новым «целевым состоянием» (к которому интегратор с утечкой желает сойтись).

Вентили сброса контролируют, какие части состояния использовать для вычисления следующего целевого состояния, и вносят дополнительный нелинейный эффект в соотношение между прошлым и будущим состояниями.

На эту тему возможно еще много вариаций. Например, выход вентиля сброса (или вентиля забывания) можно разделить между несколькими скрытыми блоками. Или использовать произведение глобального вентиля (управляющего целой группой блоков, например всем слоем) и локального вентиля (управляющего одним блоком) для комбинирования глобального и локального управлений. Однако в нескольких исследованиях архитектурных вариантов LSTM и GRU не найдено решения, которое было бы очевидно лучше обоих на широком круге задач.