Neural Nets (NN), с элементами Deep Learning

Белкова Анна, Редкокош Кирилл, Лобанова Полина, гр. 21.М03-мм

11 декабря 2022 г.

Содержание

1 Постановка задачи. Аппроксимация особым классом функций

Пусть X — множество объектов, Y — множество ответов. Пусть есть обучающая выборка $X^n = (x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^p$. Обозначим $(x^1, \ldots, x^p) \in \mathbb{R}^p$ — вектор признаков объекта $x \in X$. Рассмотрим следующую задачу построения предсказывающей модели:

$$Q(a, X^n) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(a, x_i, y_i) \to \min_w,$$

где предсказывающую функцию a зададим следующим образом (рассмотрим особый класс функций):

$$a(x, w) = \sigma(\langle w, x \rangle) = \sigma\left(\sum_{j=1}^{p} w_j x^j - w_0 \right),$$

где $w_k \in \mathbb{R}$, $k = 0, \ldots, p$ — параметры; $\sigma : \mathbb{R} \to \mathbb{R}$ — функция активации; $\mathcal{L}(a, x_i, y_i)$ — функция потерь, $i = 1, \ldots, n$.

Задача классификации. Пусть $Y = \{+1, -1\}$. Если $\sigma(z) = \text{sign}(z)$, то $a(x, w)$ — линейный классификатор, и задача выглядит следующим образом:

$$Q(w, X^n) = \sum_{j=1}^{n} \mathcal{L}\left(a(x_j, w), y_j\right) = \sum_{j=1}^{n} [y_j \langle w, x_j \rangle < 0] \to \min_w .$$

Задача регрессии. Пусть $Y = \mathbb{R}$. Если взять $\sigma(z) = z$, то получим многомерную линейную регрессию:

$$Q(w, X^n) = \sum_{j=1}^{n} \mathcal{L}\left(a(x_j, w), y_j\right) = \sum_{j=1}^{n} \left(\langle w, x_j \rangle - y_j\right)^2 \to \min_w .$$

Рассмотренный класс функций удобно представить схематически (рисунок 1). Такой класс функций является простейшей математической моделью нервной клетки — нейрона.

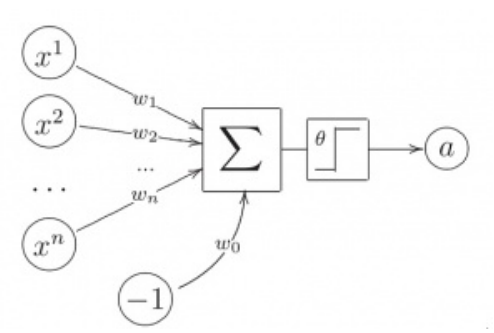


Рис. 1: Схема особого класса функций.

Если $\sigma = [z \geq 0]$, то $\sigma\left(\sum_{j=1}^{p} w_j x^j - w_0\right)$– нейрон МакКаллока-Питтса.

Попадая в нейрон, импульсы складываются с весами $w_1, \ldots, w_n$. Если вес положительный, то соответствующий синапс возбуждающий, если отрицательный, то тормозящий. Если суммарный импульс превышает заданный порог активации w_0, то нейрон возбуждается и выдаёт на выходе 1, иначе выдаётся 0. Таким образом, нейрон вычисляет n-арную булеву функцию.

В теории нейронных сетей функцию σ, преобразующую значение суммарного импульса в выходное значение нейрона, принято называть функцией активации. Таким образом, модель МакКаллока-Питтса эквивалентна пороговому линейному классификатору.

$x^j \in \mathbb{R}^n$ называются числовыми входами, $w_j \in \mathbb{R}$ — весовые коэффициенты (синаптические веса), $\sigma(z)$ — функция активации, w_0 — порог активации. Смысл термина "порог активации" становится понятен, если взять конкретную функцию активации $\sigma(z)$, к примеру, выпрямитель $ReLU(p) = \max(0, p)$.

В качестве функций активации чаще всего используются следующие функции:

- Сигмоидная функция: $\sigma(z) = \frac{1}{1+e^{-a\,z}}$, $a \in \mathbb{R}$;

- Softmax: $SM_i(z) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$;
- Гиперболический тангенс: $\sigma(z) = \frac{e^{a\,z} - e^{-a\,z}}{e^{a\,z} + e^{-a\,z}}$, $a \in \mathbb{R}$;
- Выпрямитель: $ReLU(p) = \max(0, p)$;

2 Аппроксимация функций особым классом

Любая ли функция может быть аппроксимирована введенным нами особым классом функций.

2.1 Булевы функции в виде нейронов

Рассмотрим простейшие булевы функции — 'НЕ', 'И', 'ИЛИ'. Каждая из этих функций может быть представлена в виде одного нейрона.

1. Логическая операция 'НЕ' может быть представлена в виде $\neg x^1 = [-x^1 + \frac{1}{2} > 0]$. На рисунке 2 представлен нейрон, реализующий эту операцию.

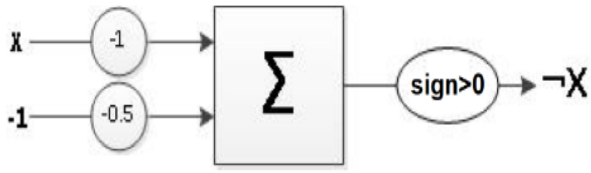


Рис. 2: Представление логической операции 'НЕ' в виде нейрона.

2. Логические операции 'ИЛИ', 'И' могут быть представлены в таком виде: $x^1 \vee x^2 = [x^1 + x^2 - \frac{1}{2} > 0]$ и $x^1 \wedge x^2 = [x^1 + x^2 - \frac{3}{2} > 0]$. На рисунке 3 представлены нейроны, реализующие эти булевы функции.

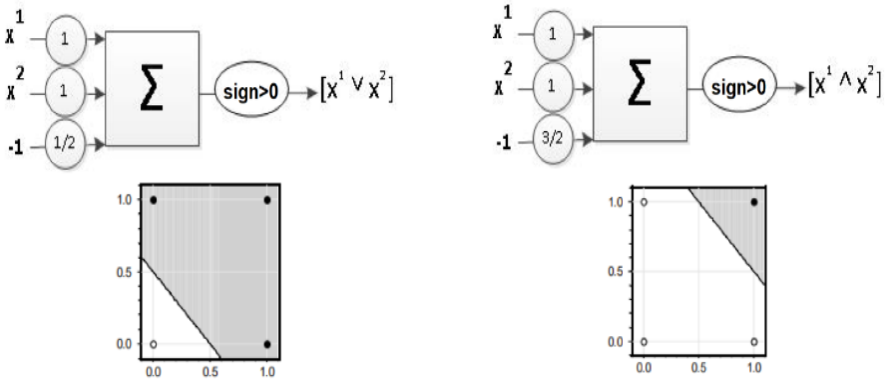


Рис. 3: Представление логических операций 'ИЛИ' и 'И' в виде нейронов.

3. Задача 'исключающего ИЛИ'. Такая операция не может быть реализована одним нейроном с двумя входами x^1 и x^2. Возможны два варианта решения такой задачи.

   - Первый вариант — пополнить пространство признаков, добавить нелинейное преобразование исходных признаков. Например, если добавить произведение исходных признаков, тогда нейрон будет строить уже не линейную, а полиномиальную разделяющую поверхность. Таким образом, мы перейдем к спрямляющему пространству признаков. Функцию 'исключающего ИЛИ' можно представить в виде $x^1 \oplus x^2 = [x^1 + x^2 - 2x^1x^2 - \frac{1}{2} > 0]$, схема нейрона представлена на рисунке 4.

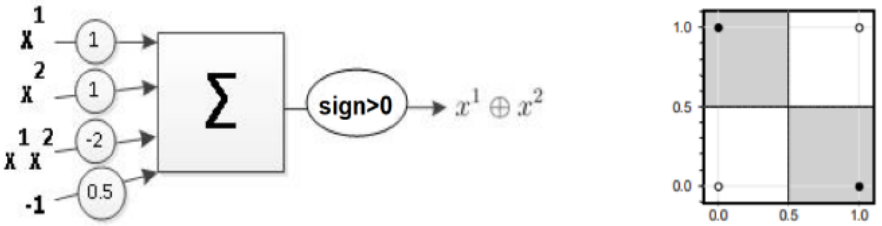


Рис. 4: Представление логической операции 'исключающее ИЛИ' в виде одного нейрона.

- Второй вариант — построить композицию из нескольких нейронов. Например, 'исключающее ИЛИ' можно представить в таком виде: $x^1 \oplus x^2 = [\neg((x^1 \vee x^2) - (x^1 \wedge x^2)) > 0]$. Получаем суперпозицию нейронов — нейронную сеть (рисунок 5).

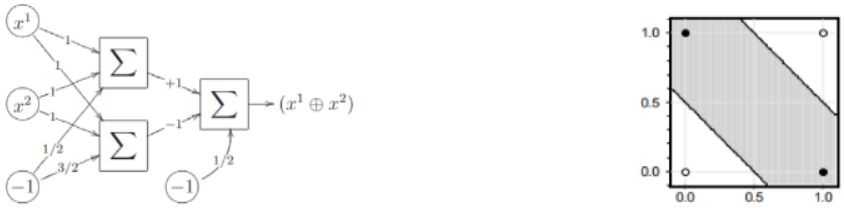


Рис. 5: Представление логической операции 'исключающее ИЛИ' в виде суперпозиции нейронов.

2.2 Аппроксимация функций суперпозицией нейронов

Возникает вопрос — насколько богатый класс функций может быть реализован нейроном?

Теорема 1 (Цыбенко, 1989). *Пусть $\sigma(x)$ — ограниченная и монотонно возрастающая непрерывная функция; $C(I_{p_0})$ — множество непрерывных функций на $[0,1]^{p_0}$.*

Тогда $\forall f \in C(I_{p_0})$ и $\forall \varepsilon > 0$ $\exists$ $p_1 \in \mathbb{Z}$ и α_i, b_i, $w_{ij} \in \mathbb{R}$, $i = 1, \ldots, p_1$, $j = 1, \ldots, p_0$, такие что для любого $x = (x^1, \ldots, x^{p_0}) \in I_{p_0}$ выполняется

$$|F(x^1, \ldots, x^{p_0}) - f(x^1, \ldots, x^{p_0})| < \varepsilon,$$

где

$$F(x^1, \ldots, x^{p_0}) = \sum_{i=1}^{p_1} \alpha_i \, \sigma \left(\sum_{j=1}^{p_0} w_{ij} x^j - w_{0i} \right).$$

В случае классификации приближаемой функцией является функция, задающая границу между классами.

Из этой теоремы можно сделать вывод, что любую непрерывную функцию можно приблизить нейронной сетью с любой желаемой точностью. А также, что для этой сети требуется один скрытый слой и одна нелинейная функция активации.

Замечание 1. *Верно следующее:*

1. *Двухслойная сеть в $\{0,1\}^n$ позволяет реализовать любую булеву функцию (ДНФ).*
2. *Двухслойная сеть в $\mathbb{R}^n$ позволяет реализовать любой выпуклый многогранник.*
3. *Трехслойная сеть в $\mathbb{R}^n$ позволяет отделить любую многогранную область, не обязательно выпуклую и не обязательно связную.*
4. *С помощью линейных операций и одной нелинейной функции активации можно приблизить любую непрерывную функцию с любой точностью (теорема Горбаня, 1998).*

3 Алгоритм обратного распространения ошибки BackProp

Рассмотрим для удобства двухслойную нейронную сеть. Пусть $Y = \mathbb{R}^M$. Все приведенные ниже рассуждения можно будет обобщить на произвольное количество слоев. Пусть выходной слой состоит из M нейронов с функциями активации σ_m и выходами $a^m, m = 1, \ldots, M$. Перед ним находится скрытый слой из H нейронов с функциями активации σ_h и выходами $u^h, h = 1, \ldots, H$. Веса синаптических связей между h-м нейроном скрытого слоя и m-м нейроном выходного слоя будем обозначать через w_{hm}. Схема описанной нейронной сети представлена на рисунке 6.

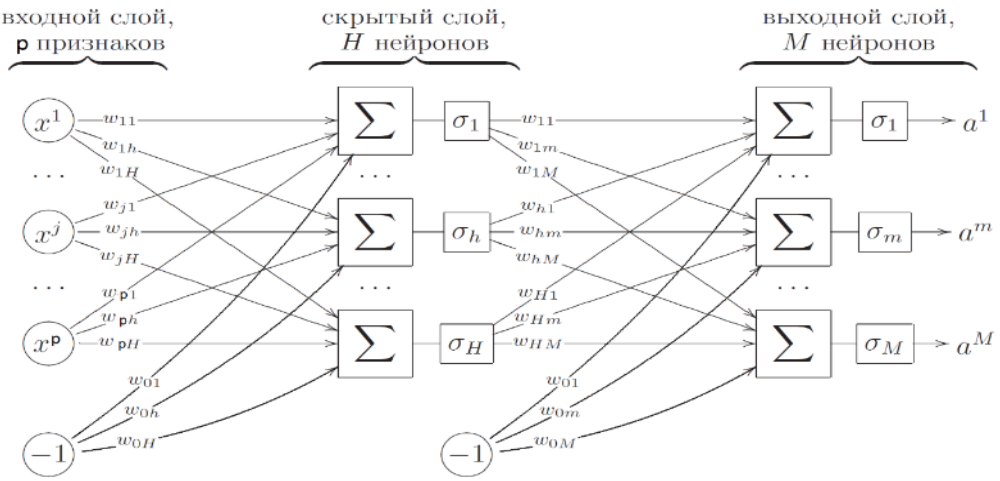


Рис. 6: Двухслойная нейронная сеть

Проблема: Посчитаем число параметров в такой модели. В случае двухслойной нейронной сети получим $(p+1)H+(H+1)M$ весовых коэффициентов. Для решения поставленной задачи используются градиентные методы, в частности, стохастический градиентный спуск, однако при таком большом количестве параметров посчитать градиент довольно трудоемко. Для решения этой проблемы возник метод обратного распространения ошибки, который с некоторыми затратами памяти эффективно вычисляет градиент.

Для начала поставим промежуточную задачу эффективного вычисления частных производных $\frac{\partial \mathcal{L}_i(w)}{\partial a^m}$ и $\frac{\partial \mathcal{L}_i(w)}{\partial u^h}$. Идея состоит в том, что при первом вычислении сети мы будем сохранять некоторые величины, которые впоследствии помогут быстро посчитать градиент.

В случае двухслойной сети: $a^m(x_i)$, $m = 1, \ldots, M$ на объекте x_i:

$$a^m(x_i) = \sigma_m\left(\sum_{h=0}^{H} w_{hm} u^h(x_i)\right), \qquad u^h(x_i) = \sigma_h\left(\sum_{j=0}^{p} w_{jh} x_i^j\right).$$

Пусть для конкретности

$$\mathcal{L}_i(w) = \frac{1}{2}\sum_{m=1}^{M}(a^m(x_i) - y_i^m)^2,$$

для других функций потерь рассуждения можно провести по аналогии.

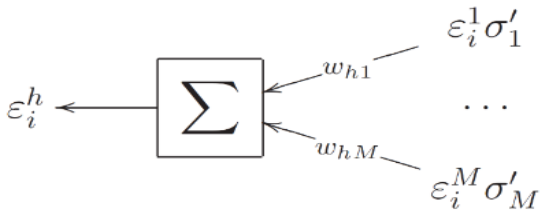
Выпишем выражения для частных производных. Для краткости записи будем обозначать $\sigma'_m = \sigma'_m(\sum\limits_{h=0}^{H} w_{hm}u^h(x_i))$ и аналогично для других производных в соответствующих точках.

$$\frac{\partial \mathcal{L}_i(w)}{\partial a^m} = a^m(x_i) - y_i^m = \varepsilon_i^m \quad \text{— ошибка на выходном слое,}$$

$$\frac{\partial \mathcal{L}_i(w)}{\partial u^h} = \sum_{m=1}^{M}(a^m(x_i) - y_i^m)\sigma'_m w_{hm} = \sum_{m=1}^{M}\varepsilon_i^m \sigma'_m w_{hm} = \varepsilon_i^h.$$

По аналогии назовем это также ошибкой на нейроне промежуточного слоя.

Теперь заметим, что ε_i^h вычисляется по ε_i^m, если запустить сеть в обратном порядке, справа налево:



Итак, имеем формулы для компонент вектора градиента:

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{hm}} = \frac{\partial \mathcal{L}_i(w)}{\partial a^m}\frac{\partial a^m}{\partial w_{hm}} = \varepsilon_i^m \sigma_m' u^h(x_i), \qquad h = 0,\ldots,H, \quad m = 1,\ldots,M,$$

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{jh}} = \frac{\partial \mathcal{L}_i(w)}{\partial u^h}\frac{\partial u^h}{\partial w_{jh}} = \varepsilon_i^h \sigma_h' f_j(x_i), \qquad j = 0,\ldots,p, \quad h = 1,\ldots,H.$$

Будем сохранять промежуточные величины ε_i^m и ε_i^h, чтобы эффективно вычислить компоненты вектора градиента. Полученный алгоритм — это метод стохастического градиентного спуска с быстрым вычислением градиента.

Алгоритм 1: обучение двухслойной нейронной сети методом обратного распространения ошибки (back-propagation)

Входные данные: $\mathbb{X}^n = \{x_i, y_i\}_{i=1}^n$ — обучающая выборка, $x_i \in \mathbb{R}^p$, $y_i \in \mathbb{R}^M$, H — число нейронов на скрытом слое, η — темп обучения, параметр λ

Выходные данные: w_{jh}, w_{hm} — веса

Инициализация весов $w_{jh},\ w_{hm}$;

повторять

  Выбираем x_i из $\mathbb{X}^n$;

  Прямой ход:

  $u^h := \sigma_h(\sum\limits_{j=0}^{n} w_{jh}x_i^j), \quad h = 1,\ldots,H;$

  $a^m := \sigma_m(\sum\limits_{h=0}^{H} w_{hm}u_i^h), \quad \varepsilon_i^m := a_i^m - y_i^m, \quad m = 1,\ldots,M;$

  $\mathcal{L}_i := \sum\limits_{m=1}^{M} (\varepsilon_i^m)^2$;

  Обратный ход: $\varepsilon_i^h := \sum\limits_{m=1}^{M} \varepsilon_i^m \sigma_m' w_{hm}, \quad h = 1,\ldots,H;$

  Градиентный шаг: $w_{hm} := w_{hm} - \eta\varepsilon_i^m\sigma_m' u^h(x_i), \qquad h = 0,\ldots,H,\ m = 1,\ldots,M;$

  $w_{jh} := w_{jh} - \eta\varepsilon_i^h\sigma_h' f^j(x_i), \qquad j = 0,\ldots,p,\ h = 1,\ldots,H$;

  $Q := (1-\lambda)Q + \lambda\mathcal{L}_i;$

до тех пор, пока Q *не сойдется*;

Достоинства метода обратного распространения ошибок:

- эффективность: быстрое вычисление градиента. В случае двухслойной сети прямой ход, обратный ход и вычисление градиента требуют порядка $O(Hp + HM)$ операций;
- метод легко обобщается на любые функции потерь, функции активации, произвольное количество слоев и произвольную размерность входов и выходов;
- возможно динамическое (потоковое) обучение;
- на сверхбольших выборках не обязательно брать все x_i.

Недостатки (есть все те же, что и у метода стохастического градиента):

- метод не всегда сходится;
- возможна медленная сходимость;
- застревание в локальных минимумах;
- проблема переобучения.

4 Некоторые способы улучшения сходимости

Для улучшения сходимости применимы некоторые эвристики:

1. инициализация весов;
2. порядок предъявления объектов;
3. оптимизация величины градиентного шага;
4. регуляризация (сокращение весов).

Инициализация весов. Есть несколько вариантов выбора начальных весов. Рассмотрим некоторые из них.

1. Часто веса инициализируются случайно, небольшими по модулю значениями. Например, в качестве начального приближения берутся случайные значения из отрезка $[-\frac{1}{2k}, \frac{1}{2k}]$, где k — число нейронов в том слое, из которого выходит связь.

2. Существует вариант инициализации весов в зависимости от корреляции признака и столбца ответов:
$$w_{jh} = \frac{\langle x^j, y\rangle}{\langle x^j, x^j\rangle} + \varepsilon_{jh}.$$
В таком случае чем больше похожи x^j и y, тем больше вес. Чтобы веса не инициализировались одинаково, к ним добавляется некоторая случайность.

3. Начальное приближение также можно сформировать по-другому. Идея заключается в том, чтобы сначала настроить нейроны первого слоя отдельно, как H однослойных нейронных сетей. Затем по-отдельности настраиваются нейроны второго слоя, которым на вход подаётся вектор выходных значений первого слоя. Чтобы сеть не получилась вырожденной, нейроны первого слоя должны быть существенно различными. Будет хорошо, если они будут хоть как-то приближать целевую зависимость, тогда второму слою останется только усреднить результаты первого слоя, сгладив ошибки некоторых нейронов. Для этого необходимо обучать нейроны первого слоя на различных случайных подвыборках, либо подавать им на вход различные случайные подмножества признаков. Так как при формировании начального приближения не требуется особая точность, поэтому отдельные нейроны можно обучать простейшими градиентными методами.

Выбор градиентного метода. Из-за того, что градиентные методы первого порядка сходятся довольно медленно, они редко применяются на практике. Ньютоновские методы второго порядка также непрактичны, потому что они требуют вычисления матрицы вторых производных функционала $Q(w)$, имеющей слишком большой размер. Поэтому можно использовать следующие варианты улучшения сходимости.

1. **Метод стохастического градиента с адаптивный шагом.** Идея заключается в том, что на каждом шаге подбирается параметр η_*:
$$Q(w - \eta\frac{\delta Q}{\delta w}) \to \min_{\eta}.$$
Для решения этой задачи не обязательно находить точный минимум, поэтому можно использовать простейшие методы оптимизации.

2. **Диагональный метод Левенберга-Марквардта.** В методе Ньютона-Рафсона второго порядка:

$$w := w - \eta(Q^{''}(w))^{-1}Q^{'}(w),$$

где $Q^{''}(w) = \left(\frac{\partial^2 Q(w)}{\partial w_{jh}\partial w_{j'h'}}\right)$ — гессиан размера $(H(p+M+1)+M)^2$.

Эвристика состоит в том, что мы считаем, что гессиан диагонален:

$$w_{jh} := w_{jh} - \eta\left(\frac{\partial^2 Q(w)}{\partial w_{jh}^2} + \mu\right)^{-1}\frac{\partial Q(w)}{\partial w_{jh}},$$

μ — параметр, предотвращающий обнуление знаменателя,
$\frac{\eta}{\mu}$ — темп обучения на ровных участках функционала $Q(w)$, где вторая производная равна нулю.

Данный метод является неким усреднением методов первого порядка и второго порядка. Методы 1-ого порядка плохо сходятся вблизи окрестности минимума, там функция плохо приближается линейной, но хорошо приближается квадратичной. Метод Левенберга-Марквардта быстрее сходится в окрестности минимума из-за того, что учитываются вторые производные. Таким образом, метод Левенберга-Марквардта вблизи точки минимума похож на метод 2-ого порядка, а вдали — на метод 1-ого порядка.

5 Выбор структуры нейронной сети

Выбор структуры сети заключается в выборе числа слоёв, числа нейронов и числа связей для каждого нейрона. Существуют различные стратегии поиска оптимальной структуры сети, например, постепенное наращивание или построение заведомо слишком сложной сети с последующим упрощением. Мы рассмотрим эти варианты далее.

Неправильный выбор структуры сети приводит к проблемам недообучения и переобучения. Понятно, что слишком простые сети не способны адекватно моделировать целевые зависимости в реальных задачах. Слишком сложные сети имеют избыточное число свободных параметров, которые в процессе обучения настраиваются не только на восстановление целевой зависимости, но и на воспроизведение шума.

Выбор числа слоёв. Если знаем, что классы линейно разделимы, то нам достаточно ограничиться одним слоем. Если граница между классами нелинейная и извилистая, то в большинстве случаев достаточно взять двухслойную сеть. Трёхслойными сетями имеет смысл пользоваться для представления сложных многосвязных областей. Теоретически, можно взять нейронную сеть с большим количеством слоев, однако тогда хуже сходятся градиентные методы, и тем труднее нам будет её обучить.

Выбор числа нейронов в скрытом слое (выбор H). Имеется несколько способов выбора числа нейронов в скрытых слоях.

1. Визуальный способ. Если граница классов (или кривая регрессии) слишком сглажена — количество нейронов в слое нужно увеличить, а если есть резкие колебания, то, наоборот, уменьшить. Этот способ подходит для задач с небольшим числом признаков.

2. По внешнему критерию. Можно смотреть на среднюю ошибку на тестовой выборке или использовать cross-validation. Недостаток этого способа — высокая трудоёмкость. Приходится много раз заново строить сеть при различных значениях параметра H, а в случае скользящего контроля — ещё и при различных разбиениях выборки на обучающую и контрольную части.

Динамическое наращивание сети. Состоит в следующих шагах:

1. Обучение сети при заведомо недостаточном числе нейронов $H \ll n$, пока ошибка не перестаёт убывать;

2. Добавление нового нейрона и его инициализация путем обучения

   - либо по случайной подвыборке $X^{'} \subseteq X^n$;
   - либо по объектам с наибольшими значениями потерь;
   - либо по случайному подмножеству входов;
   - либо из различных случайных начальных приближений.

3. Снова итерации BackProp;

Эмпирический опыт заключается в том, что после добавления новых нейронов ошибка, обычно, сначала резко возрастает, затем быстро сходится к меньшему значению. Общее время обучения обычно лишь в 1.5–2 раза больше, чем если бы в сети сразу было нужное количество нейронов. Полезная информация, накопленная сетью, не теряется при добавлении новых нейронов. Также полезно наблюдать за внешним критерием: прохождение $Q(X^k)$ через минимум является надежным критерием останова.

Удаление избыточных связей (OBD — Optimal Brain Damage). Идея заключается в удалении тех связей, к изменению которых функционал $Q(w)$ наименее чувствителен. Уменьшение числа весов снижает склонность сети к переобучению.

Пусть w — локальный минимум $Q(w)$, тогда $Q(w)$ можно аппроксимировать квадратичной формой по формуле Тейлора:

$$Q(w+\delta) = Q(w) + \frac{1}{2}\delta^{\mathrm{T}} Q''(w)\delta + o(\|\delta\|^2),$$

где $Q''(w) = \frac{\partial^2 Q(w)}{\partial w_{jh}\partial w_{j'h'}}$ — гессиан размера $(H(p+M+1)+M)^2$.

Пусть гессиан $Q''(w)$ диагонален, тогда

$$\delta^{\mathrm{T}} Q''(w)\delta = \sum_{j=0}^{p}\sum_{h=0}^{H}\delta_{jh}^2\frac{\partial^2 Q(w)}{\partial w_{jh}^2} + \sum_{h=0}^{H}\sum_{m=0}^{M}\delta_{hm}^2\frac{\partial^2 Q(w)}{\partial w_{hm}^2}.$$

Хотим обнулить вес, это эквивалентно условию $w_{jh} + \delta_{jh} = 0$. Определим *значимость (salience)* веса w_{jh} как изменение функционала $Q(w)$ при его обнулении: $S_{jh} = w_{jh}^2\frac{\partial^2 Q(w)}{\partial w_{jh}^2}$. Делаем следующие шаги:

1. В BackProp вычисляем вторые производные $\frac{\partial^2 Q(w)}{\partial w_{jh}^2}$, $\frac{\partial^2 Q(w)}{\partial w_{hm}^2}$.
2. Если процесс минимизации $Q(w)$ пришел в минимум, то
   - упорядочиваем веса по убыванию S_{jh};
   - удаляем d связей с наименьшей значимостью;
   - снова запускаем BackProp.
3. Если $Q(w, X^n)$ или $Q(w, X^k)$ существенно ухудшился, то необходимо вернуть последние удаленные связи и выйти.

Аналогично, OBD можно использовать для отбора информативных признаков для нейрона скрытого слоя. Суммарная значимость признака: $S_j = \sum_{h=1}^{H} S_{jh}$. Из сети удаляем один или несколько признаков с наименьшей S_j.

6 Глубокое обучение в парадигме методов машинного обучения

Глубокое обучение — это совокупность широкого семейства методов машинного обучения, основанных на имитации работы человеческого мозга, а именно, на взаимодействии нейронов. Термин «глубокое обучения» появился ещё в 1980-х, но до середины 2000-х для реализации методов глубокого обучения не хватало вычислительных мощностей существующих компьютеров. Стоит отметить, что часто под данным термином подразумеваются многослойные нейронные сети.

Существенное отличие глубокого обучения от машинного обучения как такового заключается в том, что часть, касаемая feature engineering (feature extraction) в классических методах машинного обучения решается вручную, а в глубоком обучении она решается (может решаться) автоматически.

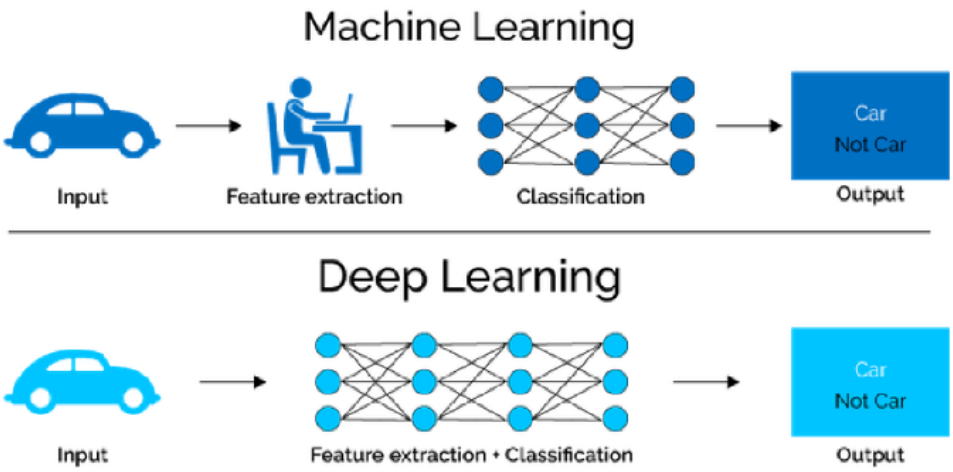


Рис. 7: Отличие между машинным обучением и глубоким обучением

Основными представителями алгоритмов глубокого обучения являются:

- сверточные нейронные сети (CNN): анализ изображений, текстов, речи. Обобщаются на случаи данных с некоторой локальной структурой;
- рекуррентные нейронные сети (RNN): обработка последовательностей векторов;
- Генеративно-состязательные сети (GAN): преобразование изображений в изображения и прогресса возраста.(модификация изображений)

Свёрточные нейронные сети используются, как правило, для анализа изображений, текстов и речи, а рекуррентные — для обработки объектов, характеризующихся наличием последовательной структуры. Например, рекуррентные нейронные сети часто используют для обработки текстов на естественном языке, обработки аудио и видео или временных рядов.

6.1 Архитектура рекуррентной нейронной сети RNN

Рассмотрим устройство рекуррентных нейронных сетей. Пусть $\boldsymbol{x}_t$ — входной вектор в момент времени t,
$\boldsymbol{h}_t$ — вектор скрытого состояния в момент времени t,
$\boldsymbol{y}_t$ — выходной вектор в момент времени t. Стоит заметить, что в некоторых приложениях $\boldsymbol{y}_t \equiv \boldsymbol{h}_t$.
Таким образом, схема рекуррентной нейронной сети может быть представлена следующим образом:

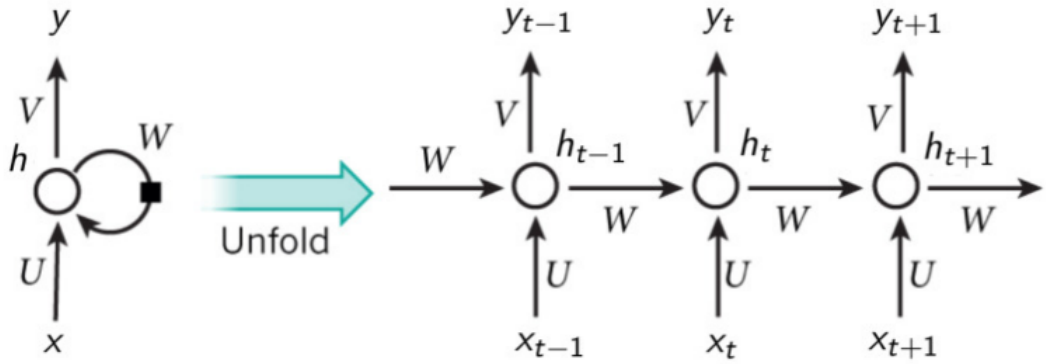


Рис. 8: Схема рекуррентной нейронной сети с развёрткой обратной связи

На рисунке слева обратная связь схематично обозначена стрелкой, выходящей из фрагмента (модуля) сети со скрытым состоянием h и входящей в него же. Наличие такой обратной связи позволяет

передавать информацию от одного шага сети к другому, справа можно увидеть развёрнутый вид этой рекуррентной сети. W, U и V – это матрицы параметров для которых справедливы следующие соотношения:
$\boldsymbol{h}_t = \sigma_h(Ux_t + Wh_{t-1})$
$\boldsymbol{y}_t = \sigma_y(Vh_t)$, где σ_h и σ_y – это функции активации.

Поясним разворачивание (unfolding) обратной связи на примере работы рекуррентной нейронной сети с предложением, положив для определённости $t = 1$. В таком случае x будет являться самим предложением, а X_0 будет первым словом в данном предложении. Это слово мы подаём на вход нейрону. Обработав его, нейрон выдаст значение y_0. Переходя к обработке следующего слова x_1, нейрон получит на вход не только само это слово, но и информацию, полученную от обработки первого слова этого предложения x_0. Тем самым, сеть сможет уловить некоторую взаимосвязь между первым и вторым словом в предложении. В качестве примера можно также рассмотреть временной ряд. В этом случае x — рассматриваемый временной ряд целиком, а h — его неизвестный сигнал. В то время как x_t — вектора вложения данного ряда (размера 1).

6.2 Обучение рекуррентной нейронной сети RNN

В процессе обучения RNN стоит задача минимизации следующего функционала:

$$\sum_{t=0}^{T} \mathcal{L}_t(\mathbf{U}, \mathbf{V}, \mathbf{W}) \to \min_{\mathbf{U},\mathbf{V},\mathbf{W}},$$

где $\mathcal{L}_t(\mathbf{U}, \mathbf{V}, \mathbf{W}) = \mathcal{L}(y_t(\mathbf{U}, \mathbf{V}, \mathbf{W}))$ — функция потерь от предсказания $\hat{y}_t$.

Используется вариант обратного распространения ошибок — Backpropagation Through Time (BPTT):

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}.$$

Поскольку рекуррентная сеть получается очень глубокой, то становится острой проблема затухающего градиента. Это происходит засчёт того, что подсчитывая градиент в методе обратного распространения ошибки (backpropagation) при обновлении весов, переходя от одного скрытого слоя к другому, у нас появляется множитель в виде некоторой матрицы в очень большой степени. Элементы этой матрицы могут быть очень близки к нулю, тогда возводя эту матрицу в очень большую степень мы получим вырождение градиента с экспоненциальной скоростью в ноль (затухание). Существует и противоположная крайность такой проблемы, называемая, взрывом градиента. Одним из решений данной проблемы является ограничение частной производной: $\frac{\partial h_i}{\partial h_{i-1}} \nrightarrow 1$, нужно ограничить частную производную(ввести архитектуру, чтобы эта величина стремилась к 1). Для решения проблемы взрыва градиента можно рассматривать модернизированную версию RNN, а именно сети долгой краткосрочной памяти (Long short-termmemory, LSTM).

6.3 Нейронные сети долгой краткосрочной памяти (LSTM)

Мотивацией к созданию LSTM сетей послужило следующее соображение. Рекуррентные нейронные сети хорошо справились бы, например, со следующей задачей: мы хотели бы предсказать последнее слово в предложении «облака плывут по небу» на основании предыдущих. В таком случае для предсказания нам не нужен более широкий контекст, довольно очевидно из предыдущего контекста, что последним будет слово «небо». То есть в таком примере, где дистанция между актуальной информацией и местом, где она понадобилась, невелика, рекуррентные нейронные сети могут обучиться, используя информацию из прошлого. Однако, можно столкнуться со случаем, когда необходимо больше контекста для предсказания последнего слова, как, например, в предложении «Я много лет жил во Франции со своими родителями и старшими братьями, поэтому я бегло говорю по-французски». Ближайшие к последнему слову контекст предполагает, что последним словом будет название языка, но, чтобы установить, какого именно языка, нам необходимо помнить о слове «Франции» из более отдалённого контекста.

Таким образом, известно, что по мере роста расстояния между актуальной информацией и точкой её применения, рекуррентные нейронные сети теряют способность находить смысловые связи. Эта проблема именуется проблемой долговременных зависимостей. Сети LSTM разработаны специально, чтобы избежать проблемы долговременной зависимости.

6.4 Устройство LSTM сети

Детально рассмотрим устройство и принцип работы сетей LSTM. Структура LSTM, как и для любой RNN сети, напоминает цепочку, состоящую из повторяющихся модулей. Однако, в отличие от рекурентных сетей, где модуль состоит из одного слоя нейронов, LSTM сеть имеет уже четыре взаимодействующих между собой слоя. Ключевым состоянием LSTM сети является вектор состояния ячейки C_t в момент времени t.

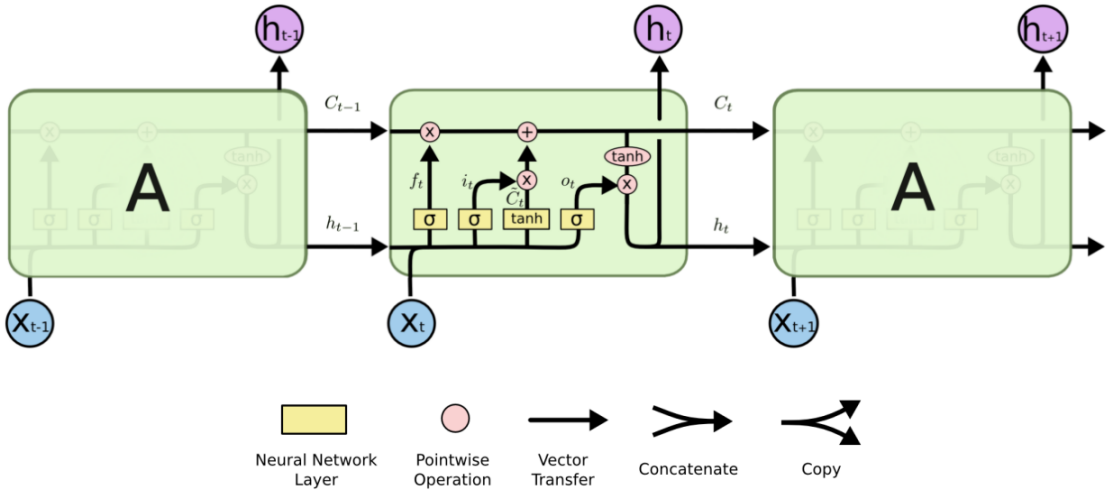


Рис. 9: Структура LSTM сети и условные обозначения

Перейдём к пошаговому разбору алгоритма работы LSTM сети.

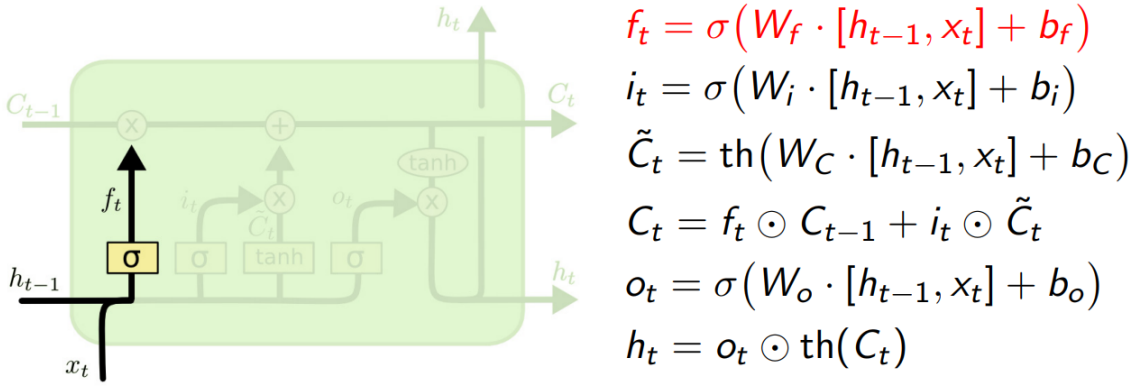
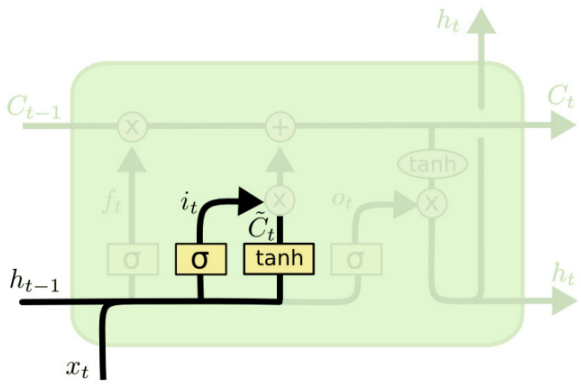


Рис. 10: Слой фильтра забывания

На первом шаге первому слою LSTM ячейки в момент времени t поступают на вход два значения: h_{t-1} – вектор скрытого состояния с предыдущего момента времени и x_t — входной вектор в момент времени t. Данный слой называется слоем Фильтра забывания (forget gate) с параметром W_f, b_f решает, какие координаты вектора C_{t-1} надо запомнить.Преобразование входных данных осуществляется следующим образом:

$$f_t = \sigma(W_f[h_t - 1, x_t] + b_f,$$

где $[h_{t-1}, x_t])$ конкатенация векторов, σ сигмойдная функция.



$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big)$$
$$\tilde{C}_t = \mathrm{th}\big(W_C \cdot [h_{t-1}, x_t] + b_C\big)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\big(W_o \cdot [h_{t-1}, x_t] + b_o\big)$$
$$h_t = o_t \odot \mathrm{th}(C_t)$$

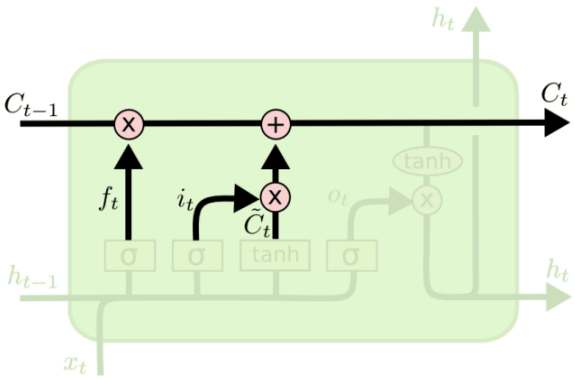
Рис. 11: Слой входного фильтра и tanh-слой

На следующем шаге необходимо решить, какая новая информация будет храниться в состоянии ячейки. Данный шаг будет осуществляться в два этапа. Сначала слой фильтра входных данных (input gate) с параметрами W_i, b_i определяет, какие координаты вектора состояния нужно обновить:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Затем tanh-слой с параметрами W_C, b_C строит вектор новых значений-кандидатов $\widetilde{C}_t$, которые можно добавить в состояние ячейки:

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C).$$



$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big)$$
$$\tilde{C}_t = \mathrm{th}\big(W_C \cdot [h_{t-1}, x_t] + b_C\big)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\big(W_o \cdot [h_{t-1}, x_t] + b_o\big)$$
$$h_t = o_t \odot \mathrm{th}(C_t)$$

Рис. 12: Обновление состояния ячейки

На третьем шаге новое состояние C_t формируется как смесь старого состояния C_{t-1} с фильтром f_t и вектора значений-кандидатов $\widetilde{C}_t$ с фильтром i_t (используем по-координатное умножение):

$$C_t = f_t \cdot C_{t-1} + i_t \widetilde{C}_t.$$

На этом шаге. очевидно, настраиваемых параметров нет, так как мы используем уже полученные ранее результаты.

Теперь осталось только решить, какую информацию мы хотим получить на выходе. Выходные данные будут основаны на нашем состоянии ячейки, к которому будут применены некоторые фильтры.

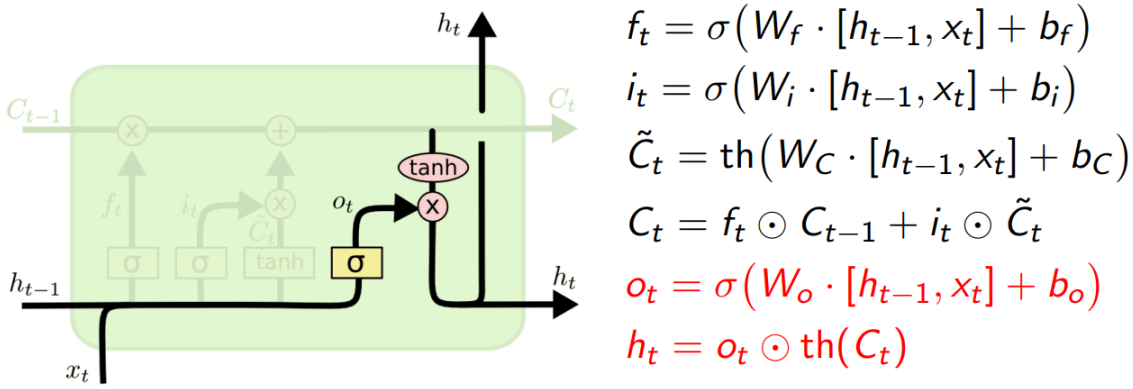


Рис. 13: Обновление состояния ячейки

Сначала мы применяем фильтр выходных данных (output gate) с параметрами W_o,b_o который решает, какую информацию из состояния ячейки мы будем выводить, то есть какие координаты вектора состояния C_t нужно выдать:

$$o_t = \sigma(W_o[h_t, x_t] + b_o).$$

Затем значения состояния ячейки проходят через tanh-слой, чтобы получить на выходе значения из диапазона от -1 до 1 в качестве выходного сигнала h_t, и перемножаются с выходными значениями o_t , что позволяет выводить только требуемую информацию:

$$h_t = o_t \cdot (C_t).$$

6.5 Вариации LSTM

Существует множество различных вариаций LSTM сетей. Отличие между ними незначительны, однако, о некоторых из них стоит упомянуть. Одна из популярных вариаций LSTM, предложенная Герсом и Шмидхубером, характеризуется добавлением так называемых «смотровых глазков» или «замочных скважин» (peephole connections). С их помощью слои фильтров могут видеть состояние ячейки.

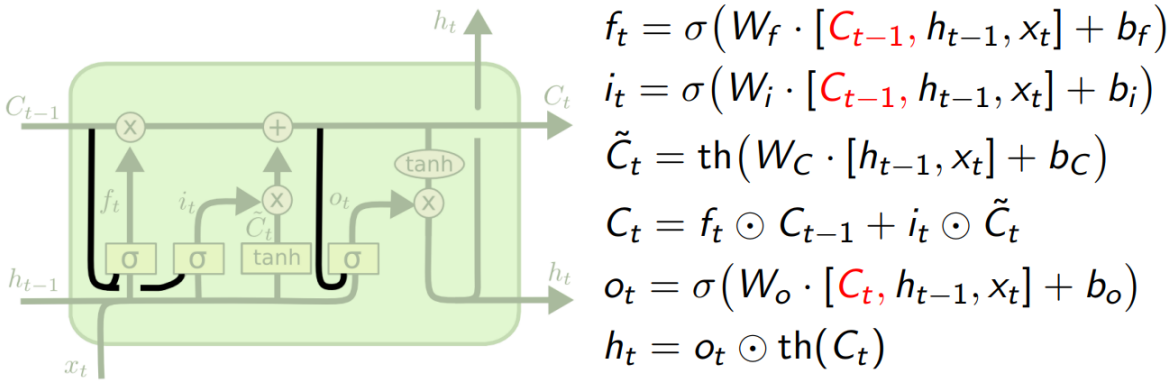
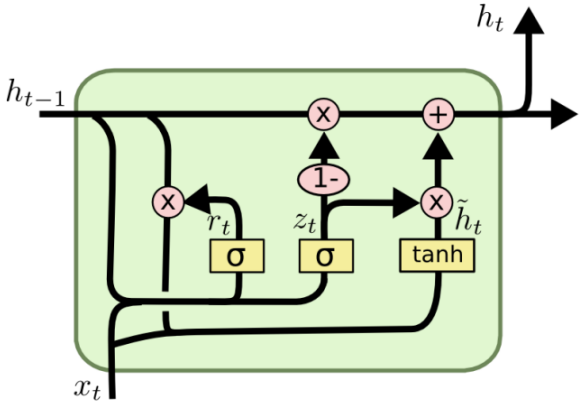


Рис. 14: LSTM с “замочными скважинами”

На схеме выше «замочные скважины’» есть у каждого слоя, но в некоторых случаях они добавляются лишь к некоторым слоям. В такой вариации LSTM число параметров модели увеличивается, а, кроме того, формулы 1,2,5 видоизменяются.

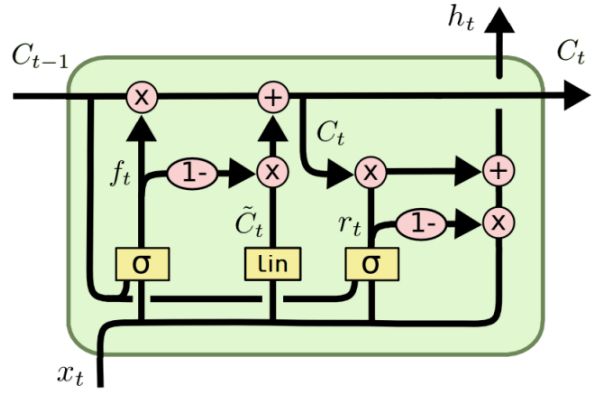
Немного больше от стандартных LSTM сетей отличается следующая модификация — управляемые рекуррентные нейроны (блоки) (Gated Recurrent Unit, GRU). Впервые эта вариация LSTM была описана в работе [2]. Важной особенностью данной нейронной сети является объединение фильтра забывания (forget gate) и входа (input gate) в один фильтр обновления (update gate). Кроме того, состояние ячейки C_t объединяется со скрытым состоянием h_t. Построенная в результате модель проще, чем стандартная LSTM (она имеет меньше параметров), и популярность ее неуклонно возрастает



$$z_t = \sigma\big(W_z \cdot [h_{t-1}, x_t]\big)$$
$$r_t = \sigma\big(W_r \cdot [h_{t-1}, x_t]\big)$$
$$\tilde{h}_t = \mathrm{th}\big(W_h \cdot [r_t \odot h_{t-1}, x_t]\big)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Рис. 15: LSTM: Gated Recurrent Unit (GRU)

Используется только состояние h_t, вектор C_t не вводиться.
Фильтр обновления (update gate) вместо входного и забывающего.
Фильтр перезагрузки (reset gate) решает, какую часть памяти нужно перенести дальше с прошлого шага.



$$f_t = \sigma\big(W_f x_t + v_f \odot C_{t-1} + b_f\big)$$
$$\tilde{C}_t = W_C x_t$$
$$C_t = f_t \odot C_{t-1} + (1 - f_t) \odot \tilde{C}_t$$
$$r_t = \sigma\big(W_r x_t + v_r \odot C_{t-1} + b_r\big)$$
$$h_t = r_t \odot C_t + (1 - r_t) \odot x_t$$

LSTM: Simple Recurrent Unit (SRU)

С предыдущего шага передаётся только C_{t-1}.
Два фильтра забывания (forget gate) и перезагрузки (reset gate).
Сквозные связи (skip connections): x_t передаётся на все слои.
Облегчённая рекурентность $v_f \odot C_{t-1}$ вместо $W_f Ct - 1$, позволяет вычислить координаты векторов параллельно.

Существует множество других модификаций LSTM сетей, как, например, глубокие управляемые рекуррентные нейронные сети (Depth Gated RNNs). Были проведены работы по сравнению самых популярных вариаций LSTM, авторы которой приходят к выводу о том, что все модификации приблизительно одинаковы.

6.6 Применение к временным рядам

Прогноз при помощи сетей прямого распространения:

- проходим окном длины L вдоль ряда, каждое окно — L-мерный элемент обучающей выборки;
- в качестве входного слоя используем L последних наблюдений ряда, в качестве выходного — значение прогноза на 1 точку вперед.

Рекуррентные сети:

- RNN — архитектура many-to-one;
- предыдущий выход переиспользуется для предсказания на несколько точек.

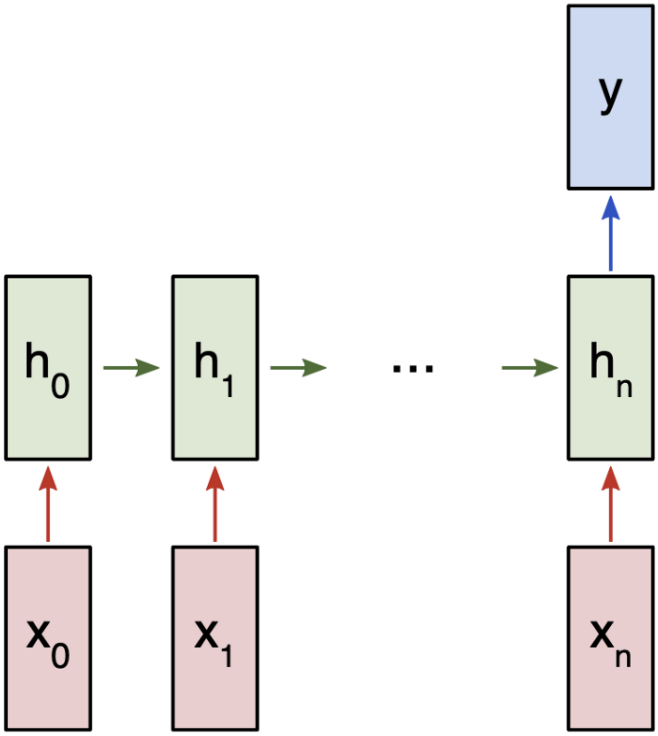


Рис. 16: архитектура many-to-one

Здесь точки $x_0, \ldots, x_n$— последняя точка временного ряда, h_i – скрытое состояние сети, полученное на предыдущем шаге (в прошлый момент времени), и мы строим прогноз на $n+1$ точку вперёд. Если бы мы хотели сделать прогноз только на несколько точек вперёд, то, как частный случай, мы бы использовали структуру many to many.