Neural Nets (NN), с элементами Deep Learning

Белкова Анна, Редкокош Кирилл, Лобанова Полина

гр. 21.М03-мм
Санкт-Петербургский государственный университет
Кафедра статистического моделирования

11 декабря 2022 г.

Постановка задачи

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 $(x^1, \dots, x^p) \in \mathbb{R}^p$ — вектор признаков объекта $x \in X$. Рассмотрим следующую задачу построения предсказывающей модели:

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

Постановка задачи

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:

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

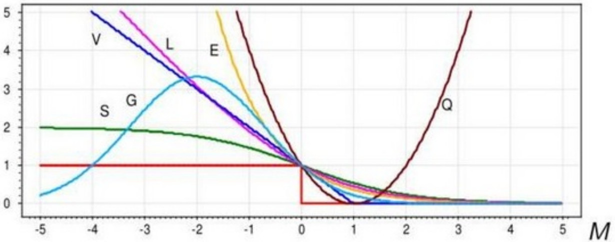
где предсказывающую функцию a зададим следующим образом (рассмотрим особый класс функций):

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

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, $i = 1, \ldots, n$.

Функции потерь в задачах классификации

Функции потерь $\mathscr{L}(M)$ в задачах классификации на два класса



$E(M) = e^{-M}$ — экспоненциальная (AdaBoost);
$L(M) = \log_2(1 + e^{-M})$ — логарифмическая (LogitBoost);
$G(M) = \exp(-cM(M + s))$ — гауссовская (BrownBoost);
$Q(M) = (1 - M)^2$ — квадратичная;
$S(M) = 2(1 + e^{M})^{-1}$ — сигмоидная;
$V(M) = (1 - M)_+$ — кусочно-линейная (SVM);

Основные задачи

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

Задача классификации. Пусть $Y = \{+1, -1\}$. Если $\sigma(z) = \operatorname{sign}(z)$, 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:

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

Нейрон

Рассмотренный класс функций удобно представить схематически. Такой класс функций является простейшей математической моделью нервной клетки — нейрона. Если $\sigma = [z \geq 0]$, то $\sigma\left(\sum_{j=1}^{p} w_j x^j - w_0\right)$– нейрон МакКаллока-Питтса.

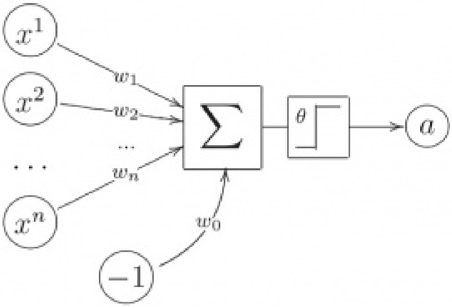


Рис.: Схема особого класса функций.

Функции активации

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: $ReLU(p) = \max(0, p)$;

Операция 'НЕ'

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

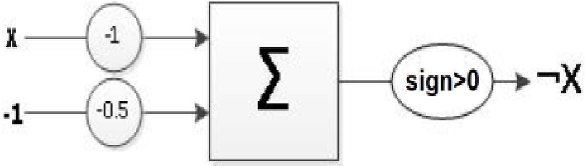


Рис.: Представление логической операции 'НЕ' в виде нейрона.

Операции 'ИЛИ', 'И'

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

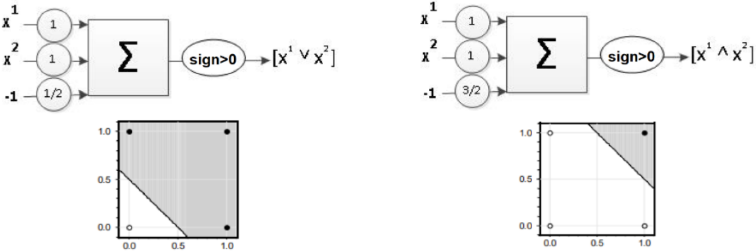


Рис.: Представление логических операций 'ИЛИ' и 'И' в виде нейронов.

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.

'Исключающее ИЛИ'

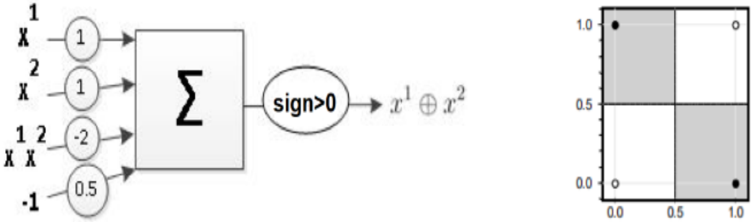
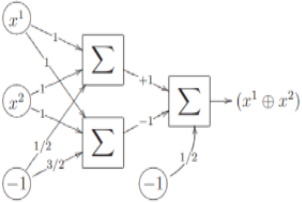


Рис.: Представление логической операции 'исключающее ИЛИ' в виде одного нейрона.

'Исключающее ИЛИ'

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.



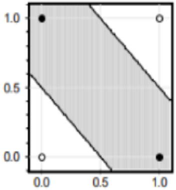


Рис.: Представление логической операции 'исключающее ИЛИ' в виде суперпозиции нейронов.

Насколько богатый класс функций может быть реализован нейроном

Theorem (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

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

Следствия

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).

Алгоритм обратного распространения ошибки BackProp

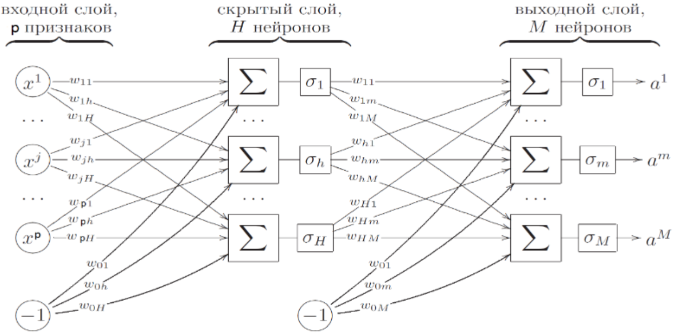


Рис.: Двухслойная нейронная сеть

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

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

Проблема оптимизации

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.

Ошибки на выходном и промежуточном слоях

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.

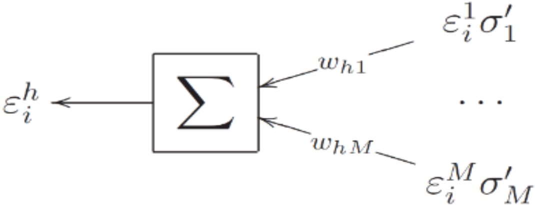
$$\frac{\partial \mathcal{L}_i(w)}{\partial a^m} = a^m(x_i) - y_i^m = \varepsilon_i^m \quad \text{— ошибка на выходном слое,}$$

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

По аналогии назовем это также ошибкой на нейроне промежуточного слоя.

Вычисление ε

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



Формулы для компонент вектора градиента

Итак, имеем формулы для компонент вектора градиента:

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

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

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.

Обучение двухслойной нейронной сети методом обратного распространения ошибки (back-propagation)

Input: $\mathbb{X}^n = \{x_i, y_i\}_{i=1}^n$ — 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

Output: w_{jh}, w_{hm} — веса

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

repeat

Выбираем x_i из $\mathbb{X}^n$;

Прямой ход:

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

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

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

Обратный ход: $\varepsilon_i^h := \sum_{m=1}^{M} \varepsilon_i^m \sigma'_m w_{hm}, \quad h = 1, \dots, H;$

Градиентный шаг:

$$w_{hm} := w_{hm} - \eta \varepsilon_i^m \sigma'_m u^h(x_i), \qquad h = 0, \dots, H,\ m = 1, \dots, M;$$

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

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

until Q *не сойдется*;

Достоинства и недостатки

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.

Инициализация весов

- 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.
- Существует вариант инициализации весов в зависимости от корреляции признака и столбца ответов:

$$w_{jh} = \frac{\langle x^j, y \rangle}{\langle x^j, x^j \rangle} + \varepsilon_{jh}.$$

- Начальное приближение также можно сформировать по-другому. Идея заключается в том, чтобы сначала настроить нейроны первого слоя отдельно, как H однослойных нейронных сетей.

Выбор градиентного метода

- **Метод стохастического градиента с адаптивный шагом.** Идея заключается в том, что на каждом шаге подбирается параметр η_*:

$$Q(w - \eta \frac{\delta Q}{\delta w}) \to \min_{\eta}.$$

- **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. Эвристика состоит в том, что мы считаем, что гессиан диагонален:

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

Выбор числа слоёв в случае классификации

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)

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

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;

Удаление избыточных связей (OBD — Optimal Brain Damage)

Пусть гессиан $Q''(w)$ диагонален, тогда. Определим *значимость (salience)* веса w_{jh} как изменение функционала $Q(w)$ при его обнулении: $S_{jh} = w_{jh}^2 \frac{\partial^2 Q(w)}{\partial w_{jh}^2}$.

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.

Глубокое обучение в парадигме методов машинного обучения

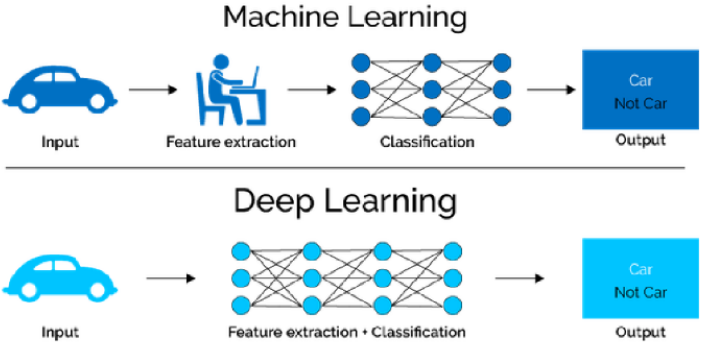


Рис.: Отличие между машинным обучением и глубоким обучением

Глубокое обучение в парадигме методов машинного обучения

Основные представители:

- 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.

Рекуррентные нейронные 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 $\boldsymbol{y}_t \equiv \boldsymbol{h}_t$.

Разворачивание рекуррентной нейронной сети:

$\boldsymbol{h}_t = \sigma_h(Ux_t + Wh_{t-1})$

$\boldsymbol{y}_t = \sigma_y(Vh_t)$

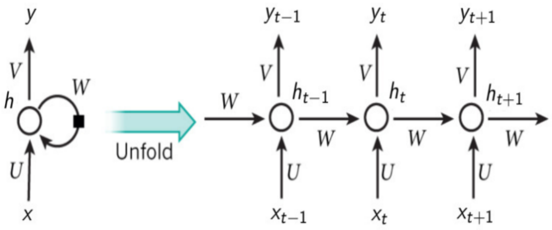


Рис.: Рекуррентная нейронная сеть

Обучение рекуррентной сети RNN

Возникает 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 $\hat{y}_t$.

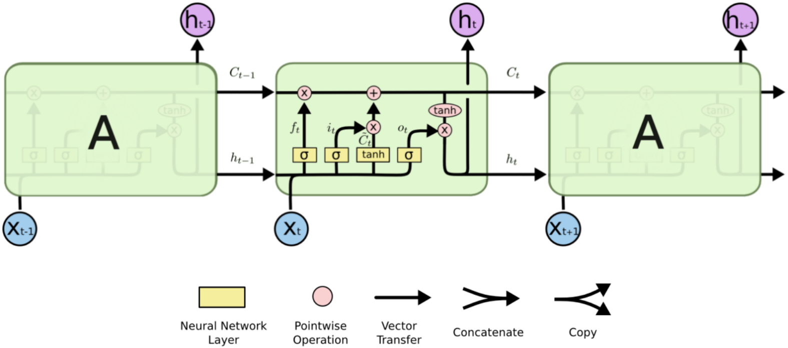
Используется вариант обратного распространения ошибок — Backpropagation Through Time (BPTT):

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

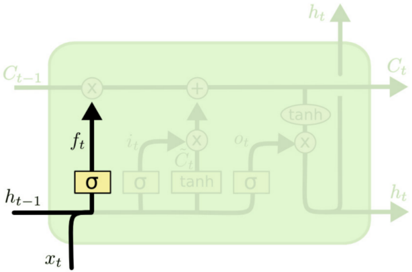
Проблема: Затухание/взрыв градиентов, если $\frac{\partial h_i}{\partial h_{i-1}} \nrightarrow 1$, нужно ограничить частную производную(ввести архитектуру, чтобы эта величина стремилась к 1)

LSTM (long short-term memory)

Мотивация LSTM: сеть должна долго помнить контекст, какой именно — сеть должна выучить сама. Поэтому вводится C_t — вектор состояния сети в момент t.



LSTM (long short-term memory)



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

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

$$\tilde{C}_t = \mathrm{th}\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$

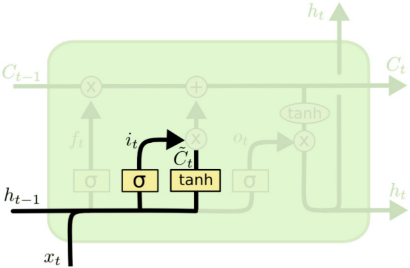
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$

$$h_t = o_t \odot \mathrm{th}(C_t)$$

Фильтр забывания (forget gate) с параметром W_f, b_f решает, какие координаты вектора C_{t-1} надо запомнить.
$\odot$ – операция покомпонентного перемножения векторов,
$[h_{t-1}, x_t]$ конкатенация векторов,
σ сигмоидная функция.

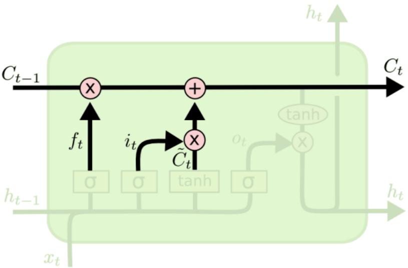
LSTM (long short-term memory)



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

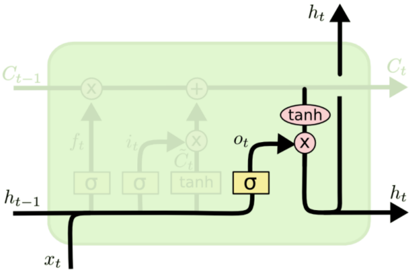
Фильтр входных данных (input gate) с параметром W_i, b_i решает, какие координаты вектора состояния надо обновить. Модель нового состояния с параметрами w_C, b_C формирует вектор $\widetilde{C}_t$ значений-координатов нового состояния.

LSTM (long short-term memory)



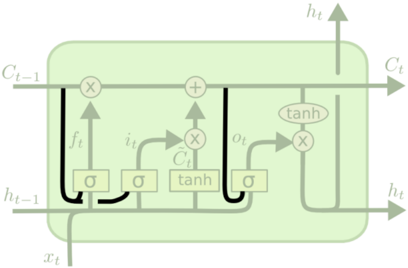
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \text{th}\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$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

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.
Настраиваемых параметров нет.



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

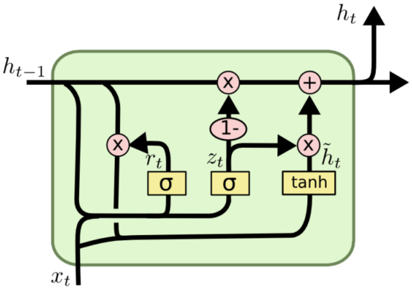
Фильтр входных данных с параметрами W_o, b_o решает какие координаты вектора состояния C_t надо выбрать.
Выходной сигнал h_t формируется из вектора состояния C_t с помощью нелинейного преобразования th и фильтра o_t.



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

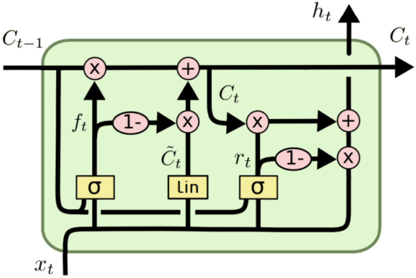
Все фильтры «подтягивают» вектор состояния C_{t-1} или C_t.
Увеличивается число параметров модели.
Замочную скважину можно использовать не для всех фильтров.

LSTM: Gated Recurrent Unit (GRU)



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

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.



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

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.

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.