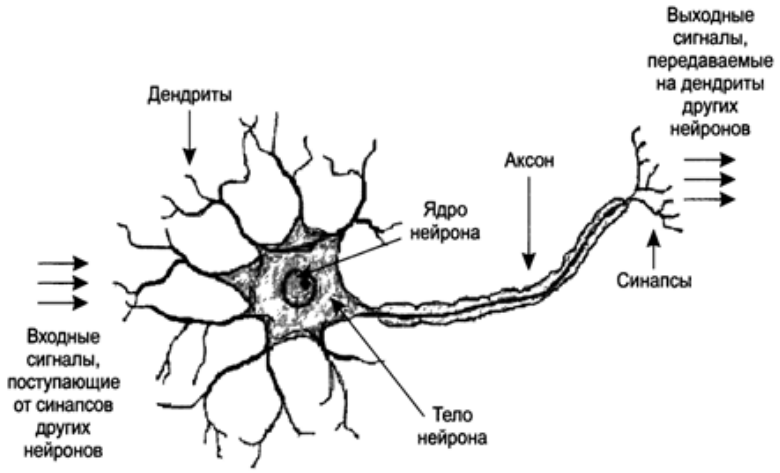
Neural Nets (NN), с элементами Deep Learning

Е. Ларин, Ф. Ежов, И. Кононыхин

1 Математическая модель нейрона

Биологический нейрон состоит из тела диаметром от 3 до 100 мкм, содержащего ядро, другие органеллы и отростки. Выделяют два вида отростков. Аксон — обычно длинный отросток, приспособленный для проведения возбуждения от тела нейрона. Дендриты — как правило, короткие и сильно разветвлённые отростки, служащие главным местом образования влияющих на нейрон возбуждающих и тормозных синапсов (разные нейроны имеют различное соотношение длины аксона и дендритов). Нейрон может иметь несколько дендритов и обычно только один аксон. Один нейрон может иметь связи с 20 тысячами других нейронов. Кора головного мозга человека содержит около 80 миллиардов нейронов.



1.1 Перцептрон (Перцептрон Розенблатта)

Математическая модель искусственного нейрона была предложена У. Маккалоком и У. Питтсом вместе с моделью сети, состоящей из этих нейронов. Авторы показали, что сеть на таких элементах может выполнять числовые и логические операции. Практически сеть была реализована Фрэнком Розенблаттом в 1958 году как компьютерная программа, а впоследствии как электронное устройство — перцептрон. Первоначально нейрон мог оперировать только с сигналами логического нуля и логической единицы, поскольку был построен на основе биологического прототипа, который может пребывать только в двух состояниях — возбужденном или невозбужденном. Развитие нейронных сетей показало, что для расширения области их применения необходимо, чтобы нейрон мог работать не только с бинарными, но и с непрерывными (аналоговыми) сигналами. Такое обобщение модели нейрона было сделано Уидроу и Хоффом, которые предложили в качестве функции срабатывания нейрона использовать логистическую кривую.

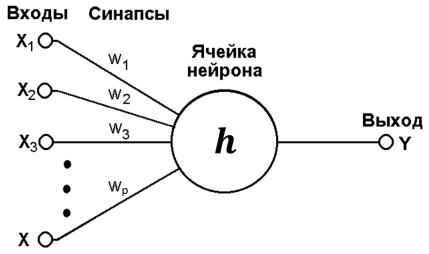
Связи, по которым выходные сигналы одних нейронов поступают на входы других, часто называют синапсами по аналогии со связями между биологическими нейронами. Каждая связь характеризуется своим весом. Связи с положительным весом называются возбуждающими, а с отрицательным — тормозящими. Нейрон имеет один выход, часто называемый аксоном по аналогии с биологическим прототипом. С единственного выхода нейрона сигнал может поступать на произвольное число входов других нейронов.

Математически нейрон представляет собой взвешенный сумматор, единственный выход которого определяется через его входы и матрицу весов.

Введем обозначения:

- p — количество признаков
- $\{x_1, \cdots, x_p\}$ — индивид (сигналы на входах нейрона).
- $\{w_1, \cdots, w_p\}$ — веса нейрона.
- Функция активации
$$f(h) = \begin{cases} 0 & h < 0 \\ 1 & h \geqslant 0 \end{cases}$$
- y — выход сети.

Формальный нейрон



$$h = \sum_i x_i * w_i \qquad y = f(h)$$

Тогда формально математическую модель перцептрона можно определить как:

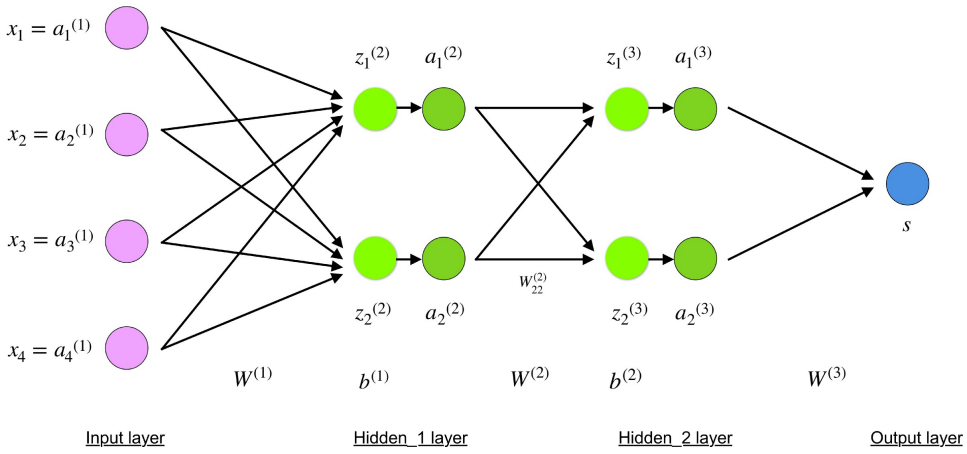
$$y = f(h) = f(\sum_{i=1}^{p} x_i w_i + x_0 w_0)$$

. Возможные значения сигналов на входах нейрона считают заданными в интервале $[0, 1]$. Дополнительный вход x_0 и соответствующий ему вес w_0 используются для инициализации нейрона. Под инициализацией подразумевается смещение активационной функции нейрона по горизонтальной оси, то есть формирование порога чувствительности нейрона. Кроме того, иногда к выходу нейрона специально добавляют некую случайную величину, называемую сдвигом. Сдвиг можно рассматривать как сигнал на дополнительном, всегда нагруженном, синапсе.

К сожалению, перцептрон может решать только линейно разделимые задачи. Он не способен решить некоторые тривиальные задачи, например задачу классификации на основе исключающего "ИЛИ"(Exclusive OR - XOR).

1.2 Многослойный перцептрон (Многослойный перцептрон Румельхарта)

В многослойном перцептроне каждый слой кроме выходного включает нейрон смещения и полностью связан со следующим слоем.



Обозначим функцию $\sigma(z)$ — сигмоидой, если $\lim\limits_{z\to-\infty}\sigma(z) = 0$ и $\lim\limits_{z\to+\infty}\sigma(z) = 1$.

Теорема Цыбенко:
Если функция активации $\sigma(z)$ — непрерывная сигмоида, то для любой непрерывной на $[0,1]^p$ функции $f(x)$ существуют такие значение параметров $w_h \in \mathbb{R}^p$, $w_0 \in \mathbb{R}$, $\alpha_h \in \mathbb{R}$, что двухслойная сеть

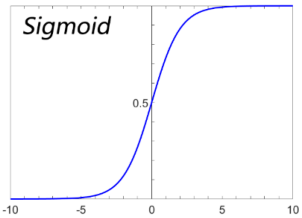
$$a(x) = \sum_{h=1}^{H} \alpha_h \sigma(\langle x, w_h\rangle + w_0)$$

равномерно приближает $f(x)$ с любой точность ε:

$$|a(x) - f(x)| < \varepsilon, \text{ для всех } x \in [0,1]^p$$

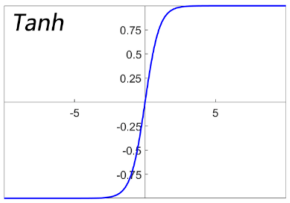
Данная теорема говорит о том, что с помощью линейных операций и одной нелинейной функции активации можно приблизить любую непрерывную функцию с любой желаемой точностью.

Функции активации:



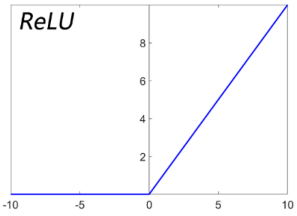
$$f(x) = \frac{1}{1+e^{-x}}$$

(a)



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

(b)



$$f(x) = \max(0, x)$$

(c)

2 Обучение MLP

Задачей оптимизации является минимизация средних потерь на обучающей выборке:

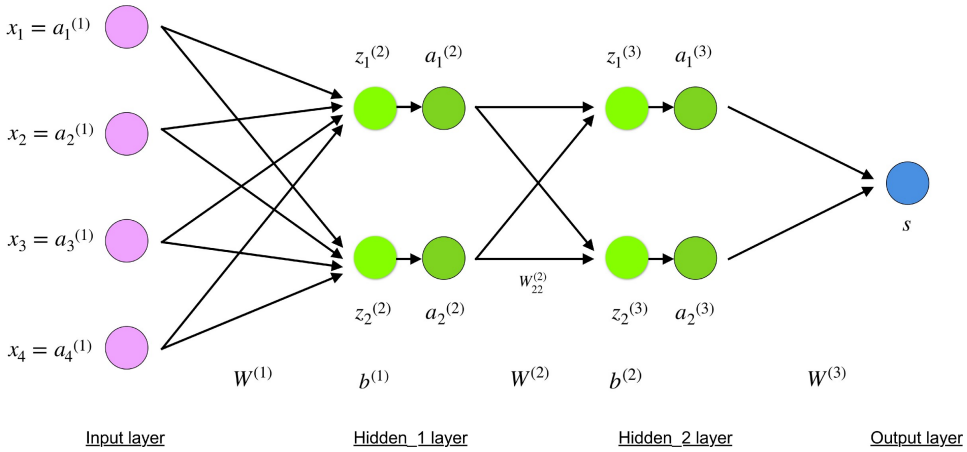
$$Q(w) := \frac{1}{n}\sum_{i=1}^{n} \mathcal{C}_i(w) \to \min_w$$

$\mathcal{C}$ — любая дифференцируемая функция потерь.

Обучение методом стохастического градиента.
Вход: выборка X^n; «скорость» обучение ϵ; параметр λ;
Выход: оптимизированные значение весов w.

1. инициализация весов w и текущую оценку $Q(w)$
2. повторять
3. выбрать объект x_i из X^n;
4. вычислить потерю $\mathcal{C}_i := \mathcal{C}_i(w)$
5. градиентный шаг: $w := w - \epsilon\mathcal{C}_i'(w)$
6. оценить значение функционала: $Q := (1-\lambda)Q + \lambda\mathcal{C}_i$
7. пока значение Q и/или весов w не стабилизируются

Обучение MLP разделяют на 2 этапа - forward и backward propogation. Forward propogation относится к подсчету функционала Q на текущей итерации, в то время как backward prop - к обновлению весов сети.



Forward prop.

$$x = a^{(1)} \quad Input\ layer$$
$$z^{(2)} = W^{(1)}x + b^{(1)} \quad neuron\ value\ at\ Hidden_1\ layer$$
$$a^{(2)} = f(z^{(2)}) \quad activation\ value\ at\ Hidden_1\ layer$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \quad neuron\ value\ at\ Hidden_2\ layer$$
$$a^{(3)} = f(z^{(3)}) \quad activation\ value\ at\ Hidden_2\ layer$$
$$s = W^{(3)}a^{(3)} \quad Output\ layer$$

$$C = cost(s, y)$$

s — полученная оценка с помощью нейронной сети. y — правильный ответ. $\mathcal{C}$ — значение ошибки между s и y по какой-нибудь метрики (например MSE).

Backward prop.

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \qquad chain\ rule$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ definition$$

$$m\ -\ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}a_k^{l-1} \qquad final\ value$$

Отсюда, изменение весов сети выглядит следующим образом:

$$while\ (termination\ condition\ not\ met)$$

$$w := w - \epsilon\frac{\partial C}{\partial w}$$

$$b := b - \epsilon\frac{\partial C}{\partial b}$$

$$end$$

Преимущества:

- быстрое вычисление градиента
- обобщение на любые функции активации, функции потерь и количество слоев
- возможность потокового обучения
- возможность распараллеливания

Недостатки:

- медленная сходимость
- застревания в локальных экстремумах

- проблема переобучения

На сегодняшний день алгоритм стохастического градиента считается устаревшим и редко используется для обучения нейронных сетей. Более быстрыми модификациями являются:

- Stochastic Gradient (SG)
- Метод накопления импульса (Momentum)
- NAG (Nesterov's accelerated gradient)
- RMSProp (running mean square)
- AdaDelta (adaptive learning rate)
- Adam (adaptive momentum)
- Nadam (Nesterov-accelerated adaptive momentume)

3 Recurrent neural network(RNN)

Рекуррентные нейронные сети — вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки. В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому сети RNN применимы в таких задачах, где нечто целостное разбито на части, например:

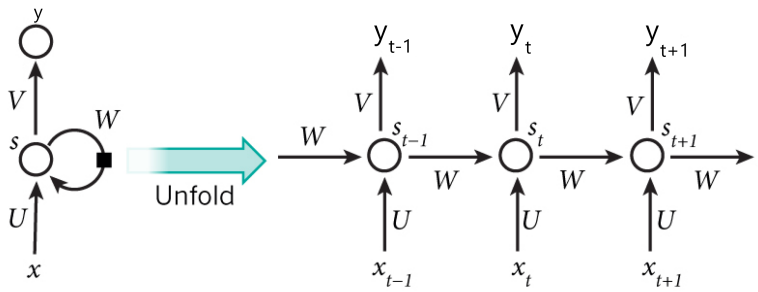
- Прогнозирование временных рядов
- Управление технологическими процессами
- Классификация текстов или их фрагментов
- Анализ тональности документа / предложений / слов
- Машинный перевод
- Распознавание речи
- Синтез речи
- Синтез ответов на вопросы, разговорный интеллект
- Генерация подписей к изображениям
- Генерация рукописного текста
- Интерпретация генома и другие задачи биоинформатики

Введем обозначения:

x_t — входной вектор в момент времени t

s_t — вектор скрытого слоя в момент времени t

y_t — выходной вектор (в некоторых случаях $y_t \equiv s_t$)



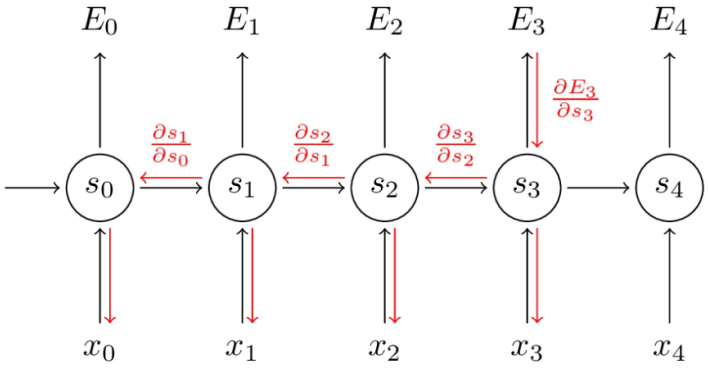
Задача:

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

$\mathcal{C}_t(U, V, W) = \mathcal{C}(y_t(U, V, W))$ — потеря от предсказания y_t.

Обучаются рекуррентные сети методом Back-propagation Through Time (BPTT).

$$\frac{\partial \mathcal{C}_t}{\partial W} = \frac{\partial \mathcal{C}_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial s_i}{\partial s_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

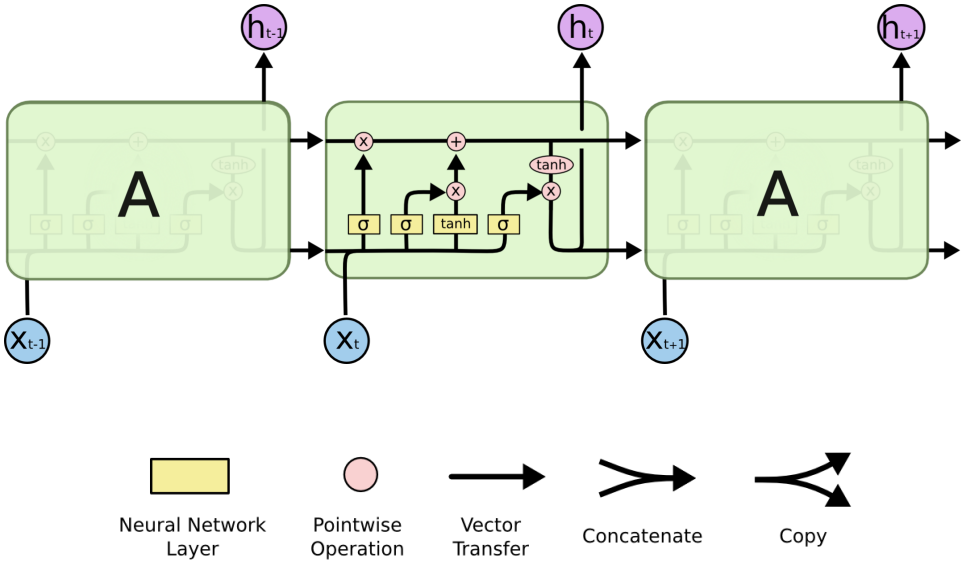


4 LSTM

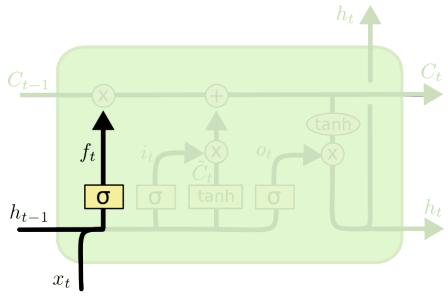
В отличие от традиционных рекуррентных нейронных сетей, LSTM-сеть хорошо приспособлена к обучению на задачах классификации, обработки и прогнозирования временных рядов в случаях, когда важные события разделены временными лагами с неопределённой продолжительностью и границами. Относительная невосприимчивость к длительности временных разрывов даёт LSTM преимущество по отношению к альтернативным рекуррентным нейронным сетям, скрытым марковским моделям и другим методам обучения для последовательностей в различных сферах применения.

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

Архитектура:

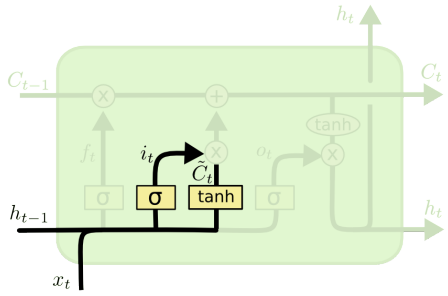


Шаг 1:



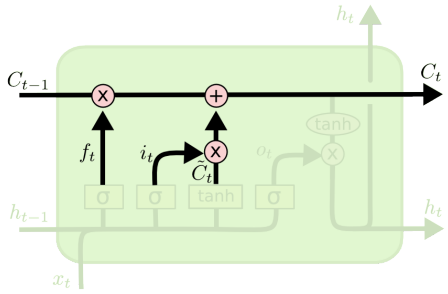
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Шаг 2:



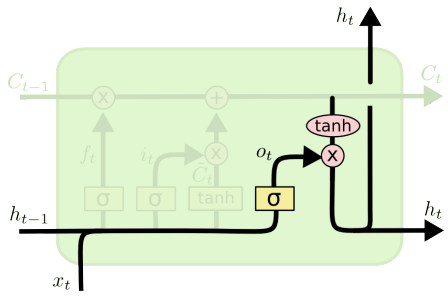
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Шаг 3:



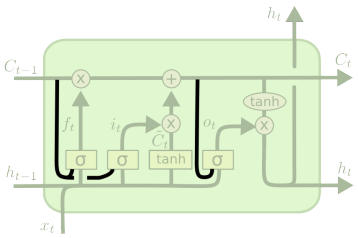
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Шаг 4:



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

LSTM с «замочными скважинами» (peepholes)



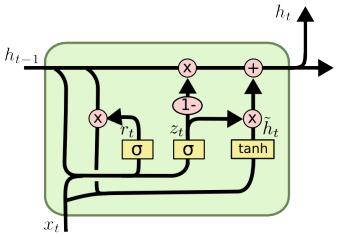
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t]\ +\ b_o\right)$$

Все фильтры «подглядывают» вектор состояния C_{t-1} или C_t.

Увеличивается число параметров модели.

Замочную скважину можно использовать не для всех фильтров.

LSTM Gated recurrent units (GRU)



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

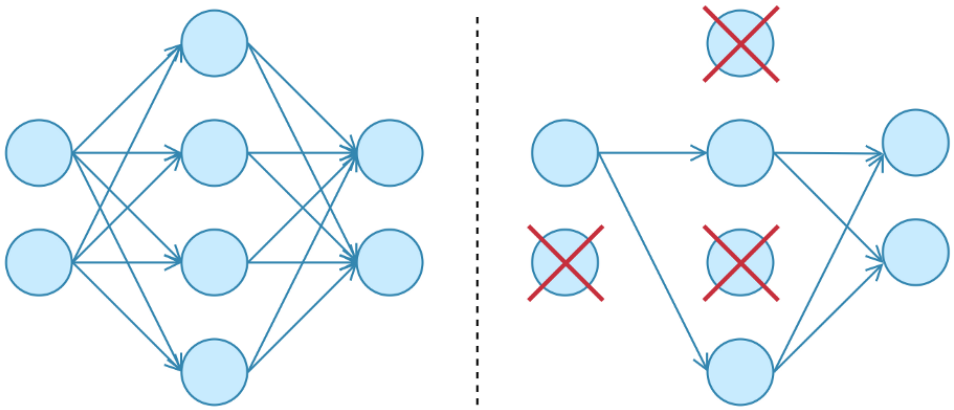
Используется только состояние h_t, вектор C_t не вводится.

Фильтр обновления (update gate) вместо входного и забывающего.

Фильтр перезагрузки (reset gate) решает, какую часть памяти нужно перенести дальше с прошлого шага.

5 Dropout

Главная идея Dropout — вместо обучения одной DNN обучить ансамбль нескольких DNN, а затем усреднить полученные результаты. В стандарт-



ной нейронной сети производная, полученная каждым параметром, сообщает ему, как он должен измениться, чтобы, учитывая деятельность остальных блоков, минимизировать функцию конечных потерь. Поэтому блоки могут меняться, исправляя при этом ошибки других блоков. Это может привести к чрезмерной совместной адаптации (co-adaptation), что, в свою очередь, приводит к переобучению, поскольку эти совместные адаптации невозможно обобщить на данные, не участвовавшие в обучении. Мы выдвигаем гипотезу, что Dropout предотвращает совместную адаптацию для каждого скрытого блока, делая присутствие других скрытых блоков ненадежным. Поэтому скрытый блок не может полагаться на другие блоки в исправлении собственных ошибок.

Обучение: обнуляем выход h-ого нейрона ℓ-го слоя с вероятностью p_ℓ.

$$a_{ih}^{\ell+1} = \xi_h^\ell f_h^\ell(\sum_j w_{jh} a_{ij}^\ell),\ P(\xi_h^\ell = 0) = p_\ell$$

Инференс: вводим поправку

$$a_{ih}^{\ell+1} = (1 - p_\ell) f_h^\ell(\sum_j w_{jh} a_{ij}^\ell)$$

Inverted Dropout:

$$a_{ih}^{\ell+1} = \frac{1}{(1-p_\ell)}\xi_h^\ell f_h^\ell\Big(\sum_j w_{jh}a_{ij}^\ell\Big)$$

- регуляризация: из всех сетей выбираем более устойчивую к утрате pN нейронов.
- сокращаем переобучение, заставляя разные части сети решать одну и ту же исходную задачу.

6 Batch Normalization

Обычно нейронные сети обучают пакетами - подмножество обучающего набора. $B = x_i$ — пакеты данных. Усреднение градиентов $\mathcal{C}(w)$ по пакету ускоряет сходимость алгоритма оптимизации. $B^\ell = \{u_i^\ell\}$ — векторы объектов x_i на выходе ℓ-го слоя.

Batch Normalization:

1. Нормировать каждую j-ю компоненту вектора u_i^ℓ по пакету:

$$\hat{u}_{ij}^\ell = \frac{u_{ij}^\ell - \mu_j}{\sqrt{\sigma_j^2+\varepsilon}}; \quad \nu_j = \frac{1}{|B|}\sum_{x_i\in B} u_{ij}^\ell; \quad \sigma_j^2 = \frac{1}{|B|}\sum_{x_i\in B}(u_{ij}^\ell - \nu_j)^2.$$

2. Добавить линейный слой с настраиваемыми весами:

$$\tilde{u}_{ij}^\ell = \gamma_j^\ell \hat{u}_{ij}^\ell + \beta_j^\ell$$

3. Параметры γ_j^ℓ и β_j^ℓ настраиваются с помощью back-propagation.

Как было отмечено ранее, нормализация по пакету ускоряет сходимость алгоритма оптимизации, а также устраняет так называемую проблему внутреннего ковариантного сдвига, что тоже способствует ускорению обучения. Внутренний ковариантный сдвиг - ситуация, при которой распределение выходов скрытого слоя изменяется на каждой итерации обучения, из-за чего следующему слою придется заново подстраивать свои веса под новое распределение, что существенно тормозит процесс обучения.