Искусственные нейронные сети

К. В. Воронцов
vokov@forecsys.ru

Этот курс доступен на странице вики-ресурса
http://www.MachineLearning.ru/wiki
«Машинное обучение (курс лекций, К.В.Воронцов)»

9 апреля 2018

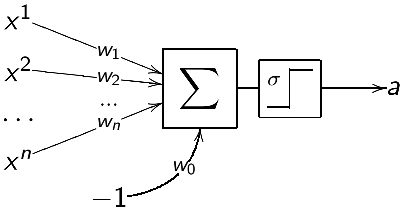
Содержание

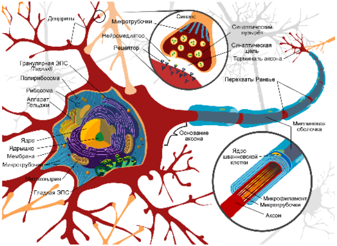
Напоминание: линейная модель нейрона МакКаллока-Питтса

$f_j \colon X \to \mathbb{R}$, $j = 1, \dots, n$ — числовые признаки;

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

где $w_0, w_1, \dots, w_n \in \mathbb{R}$ — веса признаков;
$\sigma(z)$ — функция активации, например, $\operatorname{sign}(z)$, $\frac{1}{1+e^{-z}}$, $(z)_+$





Линейные методы классификации и регрессии

Задача классификации: $Y = \{\pm 1\}$, $a(x, w) = \operatorname{sign}\langle w, x_i \rangle$;
$\mathscr{L}(M)$ — невозрастающая функция отступа, например,
$\mathscr{L}(M) = \ln(1 + e^{-M})$, $(1 - M)_+$, e^{-M}, $\frac{1}{1+e^M}$, и др.

$$Q(w; X^\ell) = \sum_{i=1}^{\ell} \mathscr{L}\bigl(\underbrace{\langle w, x_i \rangle y_i}_{M_i(w)}\bigr) \to \min_w;$$

Задача регрессии: $Y = \mathbb{R}$, $a(x, w) = \sigma(\langle w, x_i \rangle)$;

$$Q(w; X^\ell) = \sum_{i=1}^{\ell} \bigl(\sigma(\langle w, x_i \rangle) - y_i\bigr)^2 \to \min_w;$$

**Насколько богатый класс функций реализуется нейроном?
А сетью (суперпозицией) нейронов?**

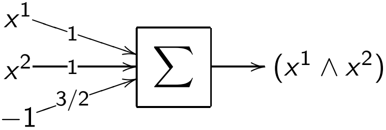
Нейронная реализация логических функций

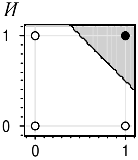
Функции И, ИЛИ, НЕ от бинарных переменных x^1 и x^2:

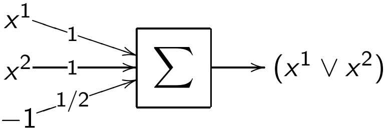
$$x^1 \wedge x^2 = \left[x^1 + x^2 - \tfrac{3}{2} > 0\right];$$

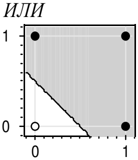
$$x^1 \vee x^2 = \left[x^1 + x^2 - \tfrac{1}{2} > 0\right];$$

$$\neg x^1 = \left[-x^1 + \tfrac{1}{2} > 0\right];$$





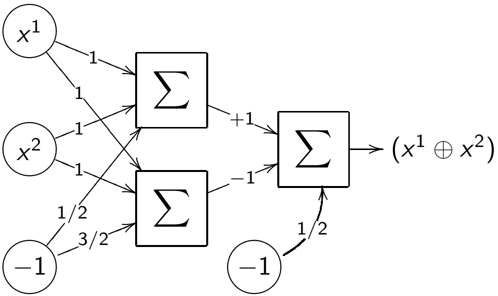


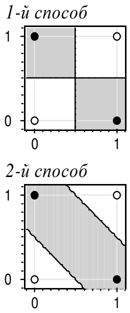


Логическая функция XOR (исключающее ИЛИ)

Функция $x^1 \oplus x^2 = [x^1 \neq x^2]$ не реализуема одним нейроном.
Два способа реализации:

- Добавлением нелинейного признака:
  $x^1 \oplus x^2 = \left[x^1 + x^2 - 2x^1x^2 - \frac{1}{2} > 0\right]$;
- Сетью (двухслойной суперпозицией) функций И, ИЛИ, НЕ:
  $x^1 \oplus x^2 = \left[(x^1 \vee x^2) - (x^1 \wedge x^2) - \frac{1}{2} > 0\right]$.





Любую ли функцию можно представить нейросетью?

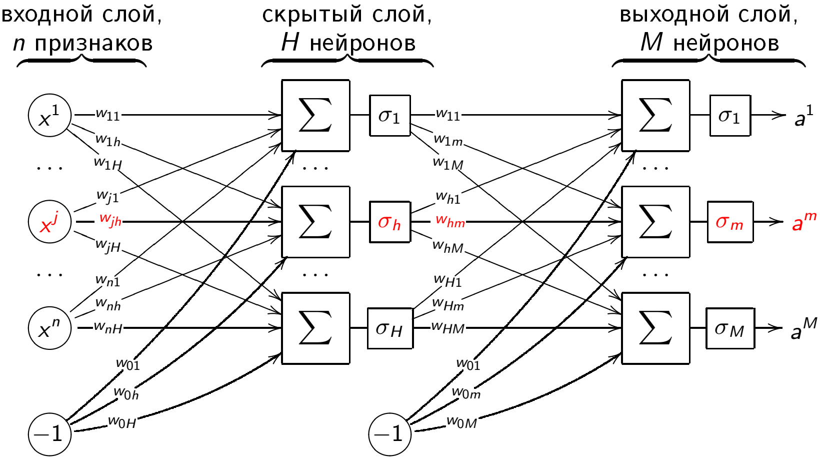
- Двухслойная сеть в $\{0,1\}^n$ позволяет реализовать произвольную булеву функцию (ДНФ).
- Двухслойная сеть в $\mathbb{R}^n$ позволяет отделить произвольный выпуклый многогранник.
- Трёхслойная сеть $\mathbb{R}^n$ позволяет отделить произвольную многогранную область, не обязательно выпуклую, и даже не обязательно связную.
- С помощью линейных операций и одной нелинейной *функции активации* σ можно приблизить любую непрерывную функцию с любой желаемой точностью.

Практические рекомендации:

- Двух-трёх слоёв теоретически достаточно.
- Глубокие сети — это встроенное обучение признаков.

Двухслойная нейронная сеть

Пусть для общности $Y = \mathbb{R}^M$, для простоты слоёв только два.



Вектор параметров модели $w \equiv \bigl(w_{jh}, w_{hm}\bigr) \in \mathbb{R}^{Hn+H+MH+M}$.

Напоминание: Алгоритм SG (Stochastic Gradient)

Минимизация средних потерь на обучающей выборке:

$$Q(w) := \frac{1}{\ell}\sum_{i=1}^{\ell}\mathscr{L}_i(w) \to \min_w .$$

Вход: выборка X^ℓ; темп обучения η; параметр λ;

Выход: вектор весов $w \equiv \bigl(w_{jh}, w_{hm}\bigr)$;

1: инициализировать веса w и текущую оценку $Q(w)$;
2: **повторять**
3: выбрать объект x_i из X^ℓ (например, случайно);
4: вычислить потерю $\mathscr{L}_i := \mathscr{L}_i(w)$;
5: градиентный шаг: $w := w - \eta\mathscr{L}_i'(w)$;
6: оценить значение функционала: $Q := (1-\lambda)Q + \lambda\mathscr{L}_i$;
7: **пока** значение Q и/или веса w не стабилизируются;

Задача дифференцирования суперпозиции функций

Выходные значения сети $a^m(x_i)$, $m = 1..M$ на объекте x_i:

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

Пусть для конкретности $\mathscr{L}_i(w)$ — средний квадрат ошибки:

$$\mathscr{L}_i(w) = \frac{1}{2}\sum_{m=1}^{M}\left(a^m(x_i) - y_i^m\right)^2.$$

Промежуточная задача: найти частные производные

$$\frac{\partial \mathscr{L}_i(w)}{\partial a^m}; \qquad \frac{\partial \mathscr{L}_i(w)}{\partial u^h}.$$

Быстрое вычисление градиента

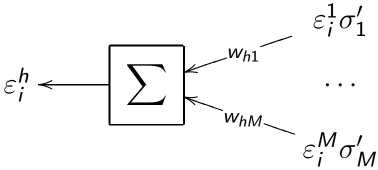
Промежуточная задача: частные производные

$$\frac{\partial \mathscr{L}_i(w)}{\partial a^m} = a^m(x_i) - y_i^m = \varepsilon_i^m$$

— это ошибка на выходном слое;

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

— назовём это *ошибкой на скрытом слое*. Похоже, что ε_i^h вычисляется по ε_i^m, если запустить сеть «задом наперёд»:



Быстрое вычисление градиента

Теперь, имея частные производные $\mathscr{L}_i(w)$ по a^m и u^h, легко выписать градиент $\mathscr{L}_i(w)$ по весам w:

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

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

Алгоритм обратного распространения ошибки BackProp:

Вход: $X^\ell = (x_i, y_i)_{i=1}^\ell \subset \mathbb{R}^n \times \mathbb{R}^M$; параметры H, λ, η;
Выход: синаптические веса w_{jh}, w_{hm};

1: ...

Алгоритм BackProp

1: инициализировать веса w_{jh}, w_{hm};

2: **повторять**

3: выбрать объект x_i из X^ℓ (например, случайно);

4: прямой ход:

$u_i^h := \sigma_h\bigl(\sum_{j=0}^J w_{jh} x_i^j\bigr), \quad h = 1..H;$

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

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

5: обратный ход:

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

6: градиентный шаг:

$w_{hm} := w_{hm} - \eta \varepsilon_i^m \sigma'_m u_i^h, \quad h = 0..H, \quad m = 1..M;$

$w_{jh} := w_{jh} - \eta \varepsilon_i^h \sigma'_h x_i^j, \quad j = 0..n, \quad h = 1..H;$

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

8: **пока** Q не стабилизируется;

Алгоритм BackProp: преимущества и недостатки

Преимущества:

- быстрое вычисление градиента;
- обобщение на любые σ, $\mathscr{L}$ и любое число слоёв;
- возможно динамическое (потоковое) обучение;
- на сверхбольших выборках не обязательно брать все x_i;
- возможно распараллеливание;

Недостатки — все те же, свойственные SG:

- возможна медленная сходимость;
- застревание в локальных минимумах;
- проблема «паралича сети» (горизонтальные асимптоты σ);
- проблема переобучения;
- подбор комплекса эвристик является искусством;

Стандартные эвристики для метода SG

Применимы те же эвристики, что и в обычном SG:

- инициализация весов (+ послойное обучение сети)
- порядок предъявления объектов
- адаптивные градиентные шаги
- диагональный метод Левенберга-Марквардта
- регуляризация L_2 или L_1
- выбивание из локальных минимумов (jogging of weights)
- chunking: разбиение суммы $\sum_i \mathscr{L}_i(w)$ на группы слагаемых

Новые проблемы связаны с выбором архитектуры сети:

- выбор числа слоёв и числа нейронов;
- выбор значимых связей;
- выбор функций активации в каждом нейроне.

Динамическое наращивание сети

1. обучение при заведомо недостаточном числе нейронов H;
2. после стабилизации $Q(w)$ — добавление нового нейрона и его инициализация путём обучения
   - либо по случайной подвыборке $X' \subseteq X^\ell$;
   - либо по объектам с наибольшими значениями потерь;
   - либо по случайному подмножеству входов;
   - либо из различных случайных начальных приближений;
3. снова итерации BackProp;

Эмпирический опыт: 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.

Прореживание сети (OBD — Optimal Brain Damage)

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

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

где $Q''(w) = \Bigl(\frac{\partial^2 Q(w)}{\partial w_{jh}\partial w_{j'h'}}\Bigr)$ — гессиан, размера $\bigl(H(n{+}M{+}1){+}M\bigr)^2$.

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

$$\delta^{\mathsf{T}} Q''(w)\delta = \sum_{j=0}^{n}\sum_{h=1}^{H}\delta_{jh}^2\frac{\partial^2 Q(w)}{\partial w_{jh}^2} + \sum_{h=0}^{H}\sum_{m=0}^{M}\delta_{hm}^2\frac{\partial^2 Q(w)}{\partial w_{hm}^2}.$$

Хотим обнулить вес: $w_{jh} + \delta_{jh} = 0$. Как изменится $Q(w)$?

Определение. *Значимость* (salience) веса w_{jh} — это изменение функционала $Q(w)$ при его обнулении: $S_{jh} = w_{jh}^2\frac{\partial^2 Q(w)}{\partial w_{jh}^2}$.

Прореживание сети (OBD — Optimal Brain Damage)

1. В BackProp вычислять вторые производные $\frac{\partial^2 Q}{\partial w_{jh}^2}$, $\frac{\partial^2 Q}{\partial w_{hm}^2}$.
2. Если процесс минимизации $Q(w)$ пришёл в минимум,то
   - упорядочить все веса по убыванию S_{jh};
   - удалить N связей с наименьшей значимостью;
   - снова запустить BackProp.
3. Если $Q(w, X^\ell)$ или $Q(w, X^k)$ существенно ухудшился, то вернуть последние удалённые связи и выйти.

Отбор признаков с помощью OBD — аналогично.

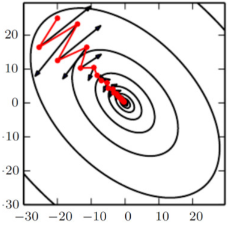
Суммарная значимость признака: $S_j = \sum\limits_{h=1}^{H} S_{jh}$.

Эмпирический опыт: результат постепенного прореживания обычно лучше, чем BackProp изначально прореженной сети.

Метод накопления импульса (momentum)

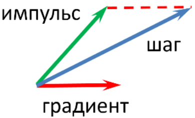
Momentum — экспоненциальное скользящее среднее градиента по $\approx \frac{1}{1-\gamma}$ последним итерациям [Б.Т.Поляк, 1964]:

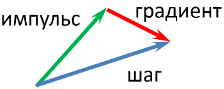
$$v := \gamma v + \eta \mathscr{L}_i'(w)$$
$$w := w - v$$



NAG (Nesterov's accelerated gradient) — стохастический градиент с импульсом Нестерова [1983]:

$$v := \gamma v + \eta \mathscr{L}_i'(w - \gamma v)$$
$$w := w - v$$





Адаптивные градиенты

RMSProp (running mean square) — адаптация скорости изменения весов, скользящим средним по $\approx \frac{1}{1-\alpha}$ итерациям:

$$G := \alpha G + (1-\alpha)\,\mathscr{L}_i'(w) \odot \mathscr{L}_i'(w)$$
$$w := w - \eta \mathscr{L}_i'(w) \oslash \left(\sqrt{G} + \varepsilon\right)$$

где $\odot$ и $\oslash$ — покординатное умножение и деление векторов.

AdaDelta (adaptive learning rate) — двойная нормировка приращений весов, после которой можно брать $\eta = 1$:

$$G := \alpha G + (1-\alpha)\,\mathscr{L}_i'(w) \odot \mathscr{L}_i'(w)$$
$$\delta := \mathscr{L}_i'(w) \odot \frac{\sqrt{\Delta}+\varepsilon}{\sqrt{G}+\varepsilon}$$
$$\Delta := \alpha\Delta + (1-\alpha)\delta^2$$
$$w := w - \eta\delta$$

Комбинированные градиентные методы

Adam (adaptive momentum) = импульс + RMSProp:

$$v := \gamma v + (1-\gamma)\mathscr{L}_i'(w) \qquad \hat{v} := v(1-\gamma^k)^{-1}$$

$$G := \alpha G + (1-\alpha)\mathscr{L}_i'(w) \odot \mathscr{L}_i'(w) \qquad \hat{G} := G(1-\alpha^k)^{-1}$$

$$w := w - \eta \hat{v} \oslash \left(\sqrt{\hat{G}} + \varepsilon\right)$$

Калибровка $\hat{v}$, $\hat{G}$ увеличивает v, G на первых итерациях.
Рекомендация: $\gamma = 0.9$, $\alpha = 0.999$, $\varepsilon = 10^{-8}$

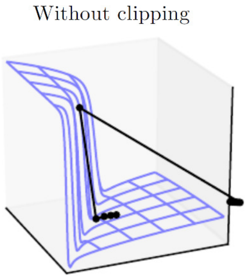
Nadam (Nesterov-accelerated adaptive momentum):
те же формулы для v, $\hat{v}$, G, $\hat{G}$,

$$w := w - \eta\left(\gamma\hat{v} + \frac{1-\gamma}{1-\gamma^k}\mathscr{L}_i'(w)\right) \oslash \left(\sqrt{\hat{G}} + \varepsilon\right)$$

Timothy Dozat. Incorporating Nesterov Momentum into Adam. ICLR-2016.

Проблема взрыва градиента и эвристика gradient clipping

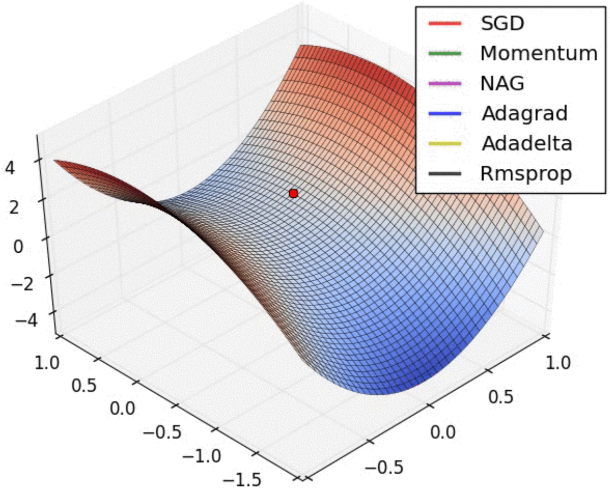
Проблема взрыва градиента (gradient exploding)



Эвристика Gradient Clipping:
если $\|g\| > \theta$ **то** $g := g\theta/\|g\|$

При грамотном подборе γ проблема взрыва градиента не возникает, и эвристика Gradient Clipping не нужна.

Сравнение сходимости методов



Alec Radford's animation: `http://cs231n.github.io/assets/nn3/opt1.gif`

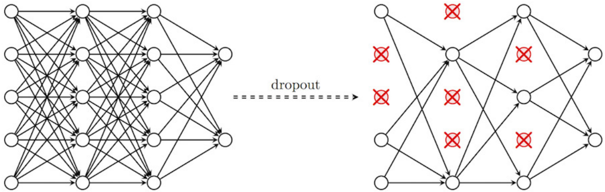
Метод случайных отключений нейронов (Dropout)

Этап обучения: делая градиентный шаг $\mathscr{L}_i(w) \to \min\limits_w$, отключаем h-ый нейрон ℓ-го слоя с вероятностью p_ℓ:

$$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h\Bigl(\sum_j w_{jh} x_{ij}^\ell\Bigr), \qquad \mathsf{P}(\xi_h^\ell = 0) = p_\ell$$

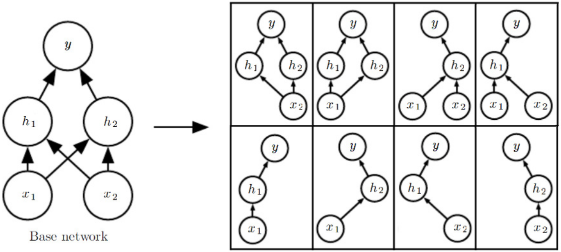
Этап применения: включаем все нейроны, но с поправкой:

$$x_{ih}^{\ell+1} = (1 - p_\ell)\sigma_h\Bigl(\sum_j w_{jh} x_{ij}^\ell\Bigr)$$



Интерпретации Dropout

1. аппроксимируем простое голосование по 2^N сетям с общим набором из N весов, но с различной архитектурой связей
2. регуляризация: из всех сетей выбираем более устойчивую к утрате pN нейронов, моделируя надёжность мозга
3. сокращаем переобучение, заставляя разные части сети решать одну и ту же исходную задачу вместо того, чтобы подстраивать их под компенсацию ошибок друг друга



Обратный Dropout и L_2-регуляризация

На практике чаще используют не Dropout, а *Inverted Dropout*.

Этап обучения:

$$x_{ih}^{\ell+1} = \frac{1}{1-p_\ell}\xi_h^\ell \sigma_h\Bigl(\sum_j w_{jh} x_{ij}^\ell\Bigr), \qquad \mathsf{P}(\xi_h^\ell = 0) = p_\ell$$

Этап применения не требует ни модификаций, ни знания p_ℓ:

$$x_{ih}^{\ell+1} = \sigma_h\Bigl(\sum_j w_{jh} x_{ij}^\ell\Bigr)$$

L_2-регуляризация предотвращает рост параметров на обучении:

$$\mathscr{L}_i(w) + \tfrac{\lambda}{2}\|w\|^2 \to \min_w$$

Градиентный шаг с Dropout и L_2-регуляризацией:

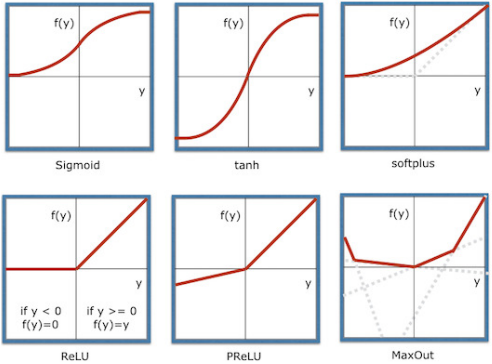
$$w := w(1-\eta\lambda) - \eta\,\frac{1}{1-p_\ell}\xi_h^\ell\,\mathscr{L}_i'(w)$$

Функции активации ReLU и PReLU (LeakyReLU)

Функции $\sigma(y) = \frac{1}{1+e^{-y}}$ и $\mathrm{th}(y) = \frac{e^{y}-e^{-y}}{e^{y}+e^{-y}}$ могут приводить к затуханию градиентов или «параличу сети»

Функция положительной срезки (rectified linear unit)

$$\mathrm{ReLU}(y) = \max\{0, y\}; \qquad \mathrm{PReLU}(y) = \max\{0, y\} + \alpha \min\{0, y\}$$



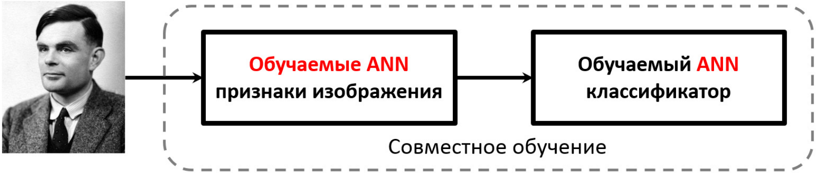
Расширение выборки (dataset augmentation)

Любые преобразования, не меняющие класс объекта

Классификация изображений

Классический подход к распознаванию изображений:

Современный подход — end-to-end deep learning:

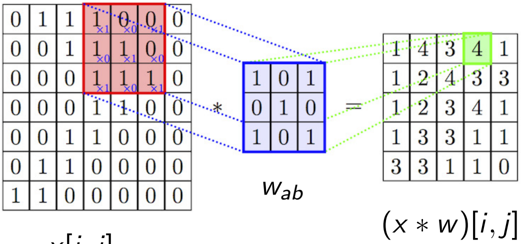


Свёрточный слой нейронов (convolution layer)

$x[i,j]$ — исходные признаки, пиксели $n \times m$-изображения
w_{ab} — ядро свёртки, $a = -A, \dots, +A$, $b = -B, \dots, +B$

Неполносвязный свёрточный нейрон с $(2A+1)(2B+1)$ весами:

$$(x * w)[i,j] = \sum_{a=-A}^{A} \sum_{b=-B}^{B} w_{ab}\, x[i+a, j+b]$$



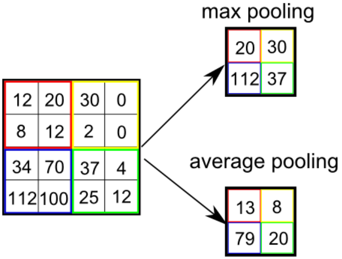
$x[i,j]$

Объединяющий слой нейронов (pooling layer)

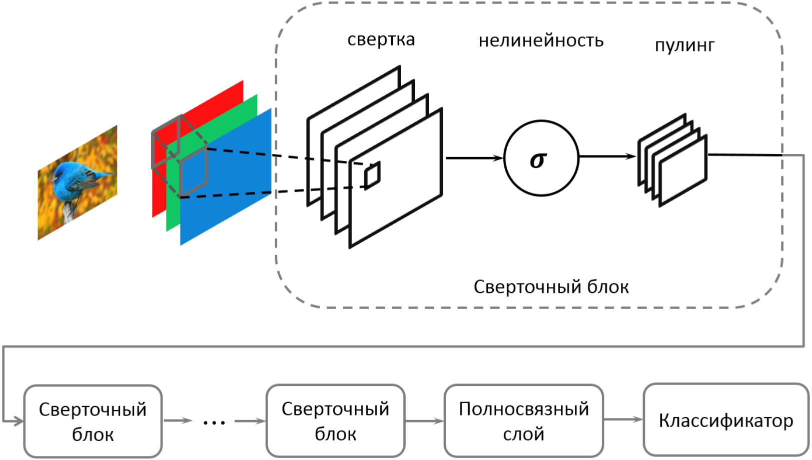
Объединяющий нейрон — это необучаемая свёртка с шагом $h > 1$, агрегирующая данные прямоугольной области $h \times h$:

$$y[i,j] = F\bigl(x[hi, hj], \ldots, x[hi+h-1, hj+h-1]\bigr),$$

где F — агрегирующая функция: max, average и т.п.
max-pooling позволяет обнаружить элемент в любой из ячеек

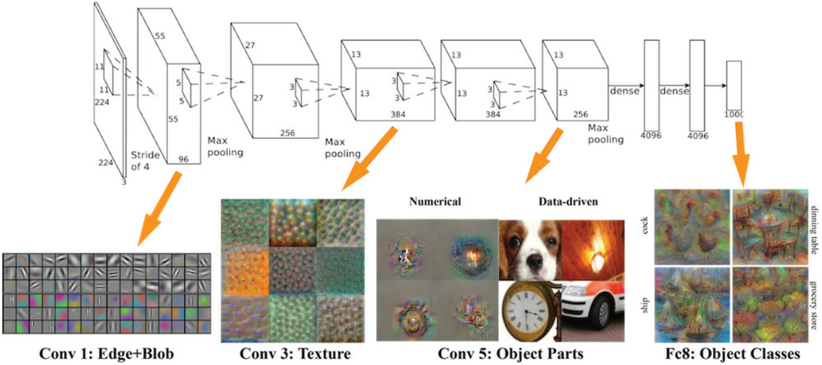


Стандартная схема сверточной сети (Convolutional NN)



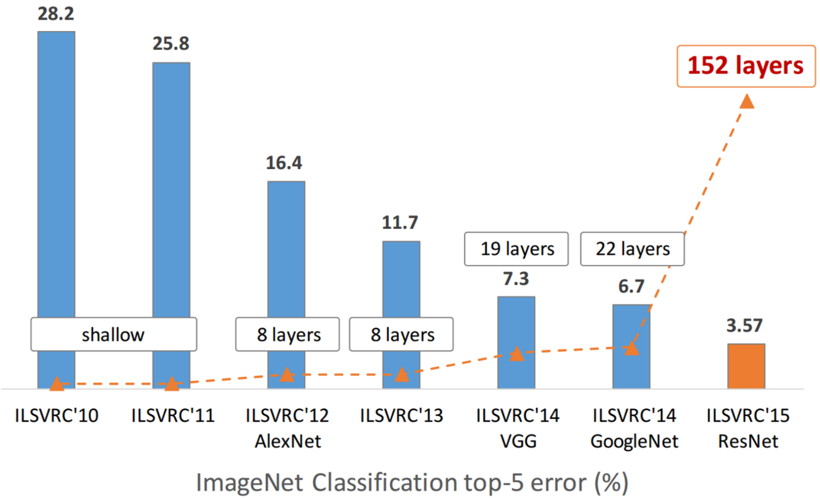
Обучение иерархии признаков

Чем выше слой, тем более крупные и сложные элементы изображений он способен распознавать

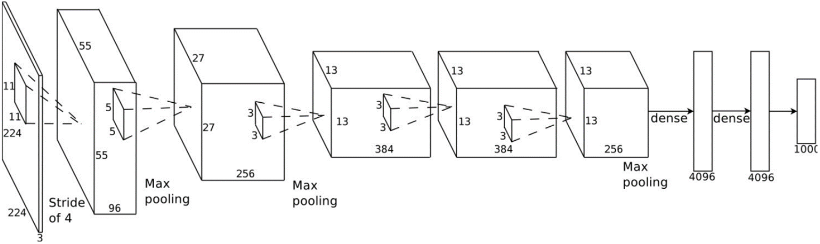


Выборка изображений ImageNet

Развитие свёрточных сетей (краткая история ImageNet)



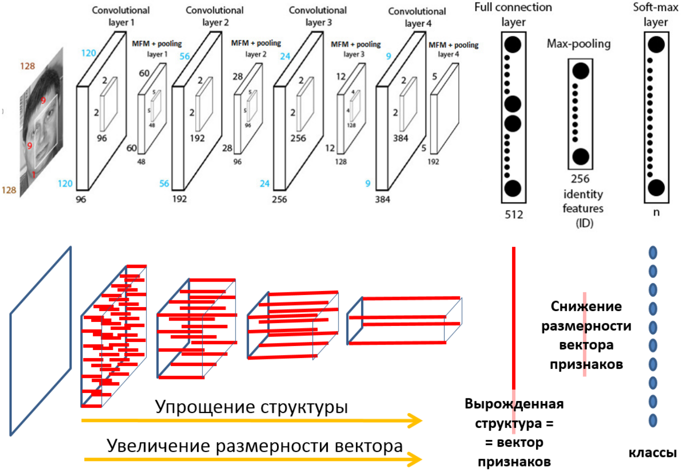
AlexNet: первый глубокий прорыв на ImageNet



- ReLU + Dropout + расширение выборки
- 60 млн параметров (в основном в полносвязных слоях)
- Подбор размеров фильтров и пулинга
- GPU

Krizhevsky A., Sutskever I., Hinton G. ImageNet Classification with Deep Convolutional Neural Networks. 2012.

Идея обобщения CNN на любые структурированные данные



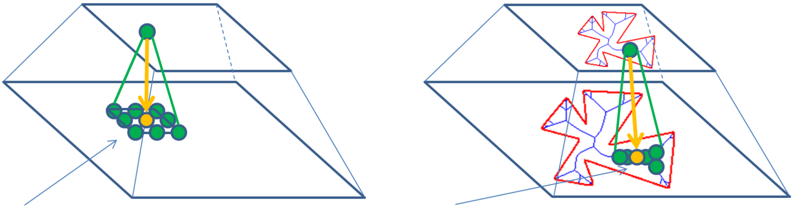
Визильтер Ю.В., Горбацевич В.С. Структурно-функциональный анализ и синтез глубоких конволюционных нейронных сетей. ММРО-2017.

Идея обобщения CNN на любые структурированные данные

Допустим, каждый объект имеет структуру, заданную графом

Свёртка определяется по локальной окрестности вершины
Пулинг агрегирует векторы вершин локальной окрестности

Такая сеть обучится находить и классифицировать подграфы



Прямоугольное окно заданного размера с центром в заданной точке + + операция свёртки по окну

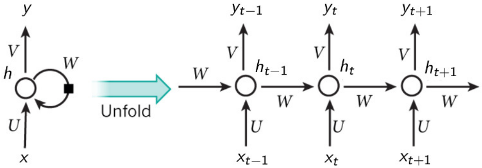
Локальная окрестность, определяемая для любой вершины графа + + операция свёртки по окрестности

Задачи обработки последовательностей

x_t — входной вектор в момент t
h_t — вектор скрытого состояния в момент t
y_t — выходной вектор (в некоторых приложениях $y_t \equiv h_t$)

Разворачивание (unfolding) рекуррентной сети

$$h_t = \sigma_h(Ux_t + Wh_{t-1})$$
$$y_t = \sigma_y(Vh_t)$$



Обучение рекуррентной сети:

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

$\mathscr{L}_t(U, V, W) = \mathscr{L}\bigl(y_t(U, V, W)\bigr)$ — потеря от предсказания y_t

Возможности рекуррентных нейронных сетей

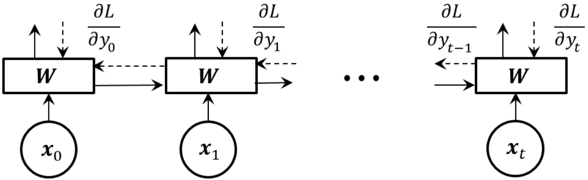
- Классификация текстов или их фрагментов
- Анализ тональности документа / предложений / слов
- Машинный перевод
- Распознавание речи
- Синтез речи
- Синтез ответов на вопросы, разговорный интеллект
- Генерация подписей к изображениям
- Генерация рукописного текста
- Интерпретация генома и другие задачи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

Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. 2015.

Обучение рекуррентных сетей

Специальный вариант обратного распространения ошибок, Backpropagation Through Time (BPTT)

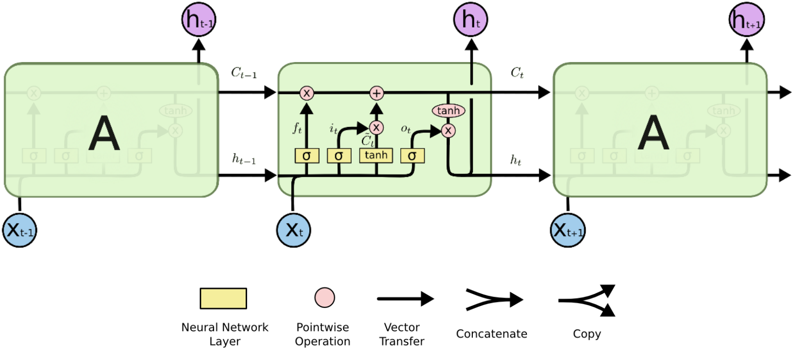
$$\frac{\partial \mathscr{L}_t}{\partial W} = \frac{\partial \mathscr{L}_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\sum_{k=0}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_k}{\partial W}$$



Для предотвращения затухания и взрыва градиентов: $\frac{\partial h_i}{\partial h_{i-1}} \to 1$

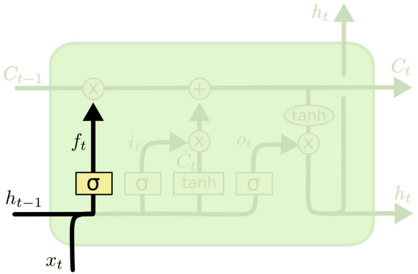
Сети долгой кратковременной памяти (long short-term memory)

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



Hochreiter S., Schmidhuber J. Neural Computation, 9(8), 1997
Greff K., Schmidhuber J. http://arxiv.org/pdf/1503.04069.pdf, 2015

Сети долгой кратковременной памяти (long short-term memory)



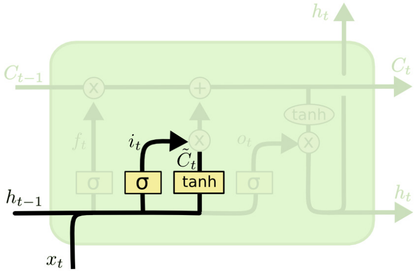
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \mathrm{th}\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$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

Фильтр забывания (forget gate) с параметрами W_f, b_f решает, какие координаты вектора состояния C_{t-1} надо запомнить.

$\odot$ — операция покомпонентного перемножения векторов.

Christopher Olah. http://colah.github.io/posts/2015-08-Understanding-LSTMs

Сети долгой кратковременной памяти (long short-term memory)



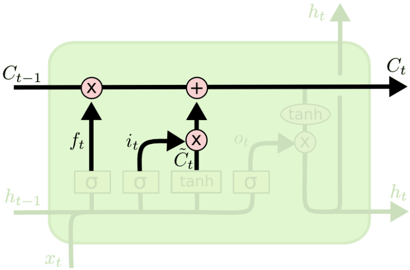
$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big)$$
$$\tilde{C}_t = \mathrm{th}\big(W_C \cdot [h_{t-1}, x_t] + b_C\big)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\big(W_o \cdot [h_{t-1}, x_t] + b_o\big)$$
$$h_t =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

Фильтр входных данных (input gate) с параметрами W_i, b_i решает, какие координаты вектора состояния надо обновить.

Модель нового состояния с параметрами W_C, b_C формирует вектор $\tilde{C}_t$ значений-кандидатов нового состояния.

Christopher Olah. `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

Сети долгой кратковременной памяти (long short-term memory)



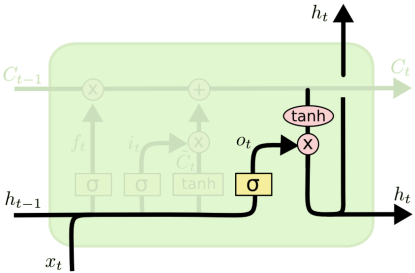
$$f_t = \sigma\bigl(W_f \cdot [h_{t-1}, x_t] + b_f\bigr)$$
$$i_t = \sigma\bigl(W_i \cdot [h_{t-1}, x_t] + b_i\bigr)$$
$$\tilde{C}_t = \text{th}\bigl(W_C \cdot [h_{t-1}, x_t] + b_C\bigr)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\bigl(W_o \cdot [h_{t-1}, x_t] + b_o\bigr)$$
$$h_t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

Новое состояние C_t формируется как смесь старого состояния C_{t-1} с фильтром f_t и вектора значений-кандидатов $\tilde{C}_t$ с фильтром i_t.

Настраиваемых параметров нет.

Christopher Olah. `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

Сети долгой кратковременной памяти (long short-term memory)



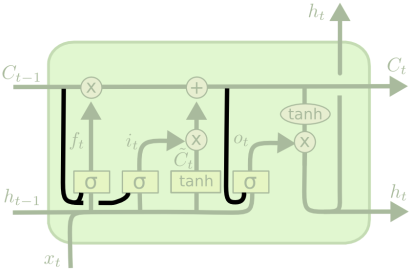
$$f_t = \sigma\bigl(W_f \cdot [h_{t-1}, x_t] + b_f\bigr)$$
$$i_t = \sigma\bigl(W_i \cdot [h_{t-1}, x_t] + b_i\bigr)$$
$$\tilde{C}_t = \text{th}\bigl(W_C \cdot [h_{t-1}, x_t] + b_C\bigr)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\bigl(W_o \cdot [h_{t-1}, x_t] + b_o\bigr)$$
$$h_t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

Фильтр выходных данных (output gate) с параметрами W_o, b_o решает, какие координаты вектора состояния C_t надо выдать.

Выходной сигнал h_t формируется из вектора состояния C_t с помощью нелинейного преобразования th и фильтра o_t.

Christopher Olah. `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

Вариант LSTM с «замочными скважинами» (peepholes)



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

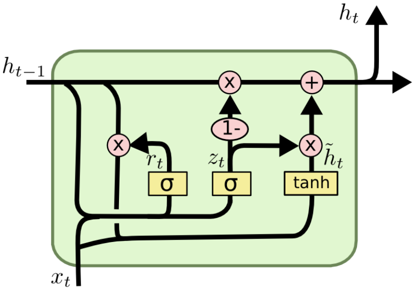
Все фильтры «подглядывают» вектор состояния C_{t-1} или C_t.

Увеличивается число параметров модели.

Замочную скважину можно использовать не для всех фильтров.

Gers F. A., Schmidhuber J. Recurrent Nets that Time and Count. 2000.

Упрощение LSTM: Gated Recurrent Unit (GRU)



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

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.

Cho K. On the properties of neural machine translation: encoder-decoder approaches. 2014.

Резюме

- Нейрон — линейная классификация или регрессия
- Нейронная сеть — суперпозиция нейронов с нелинейными функциями активации
- BackProp — быстрое дифференцирование суперпозиций
- Подбор архитектуры сети и эвристик — это искусство

С чем связана «революция глубокого обучения»

- Большие выборки + распараллеливание на GPU
- Продвинутые градиентные методы ускоряют сходимость
- Регуляризации и dropout предотвращают переобучение
- ReLU предотвращает затухание и взрыв градиентов
- Свёрточные сети: векторизация сложных структур данных
- Рекуррентные сети: механизмы памяти и принятия решений для анализа последовательностей векторов