Глубокие 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)»

31 октября 2017 • ШАД Яндекс

Содержание

Задача обучения нейронной сети (напоминание)

Дано: выборка $x_i = (x_i^1, \ldots, x_i^n)$, y_i, $i = 1, \ldots, \ell$
Найти: вектор весов w модели $a(x, w)$
Критерий: минимум суммарных потерь

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

где $\mathscr{L}_i(w) \equiv \mathscr{L}\big(a(x_i, w), y_i\big)$ — функция потерь.

Метод стохастического градиента (Stochastic Gradient):

$$w_{jh}^{k+1} := w_{jh}^k - \eta \frac{\partial \mathscr{L}_i(w^k)}{\partial w_{jh}} \quad \text{— покоординатная запись}$$

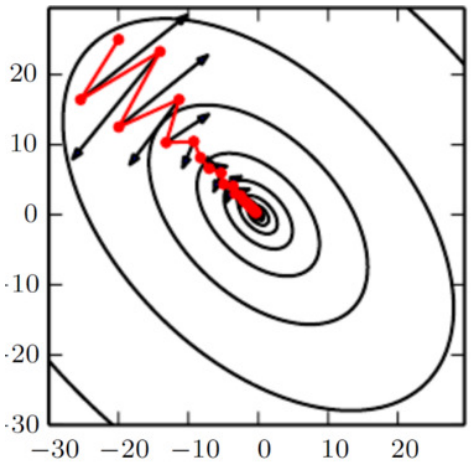
$$w := w - \eta \mathscr{L}_i'(w) \quad \text{— векторная запись без } k$$

где η — градиентный шаг (learning rate), k — номер итерации

Метод накопления импульса (momentum)

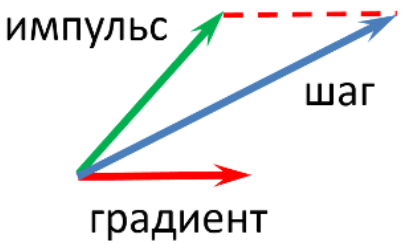
Momentum — экспоненциальное скользящее среднее градиента по $\approx \frac{1}{1-\gamma}$ последним итерациям [Б.Т.Поляк, 1964]:

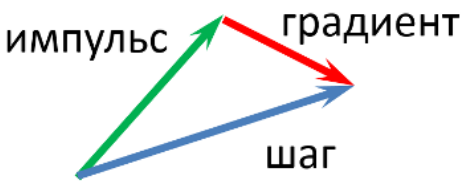
$$v := \gamma v + \eta \mathscr{L}_i'(w)$$
$$w := w - v$$



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

AdaGrad (adaptive gradient) — адаптация скоростей изменения весов, нормировка на корень суммы квадратов производных:

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

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

Недостаток — слишком быстрое увеличение суммы G.

RMSProp (running mean square) — экспоненциальное скользящее среднее по $\approx \frac{1}{1-\alpha}$ последним итерациям вместо суммы:

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

AdaDelta — ещё одно обобщение AdaGrad

Идея: возьмём за основу RMSProp:

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

и нормируем приращения весов не только на средние производные, но и на средние приращения весов.

AdaDelta (adaptive learning rate):

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

Здесь можно брать $\eta = 1$.

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: $\gamma = 0.9$, $\alpha = 0.999$, $\varepsilon = 10^{-8}$

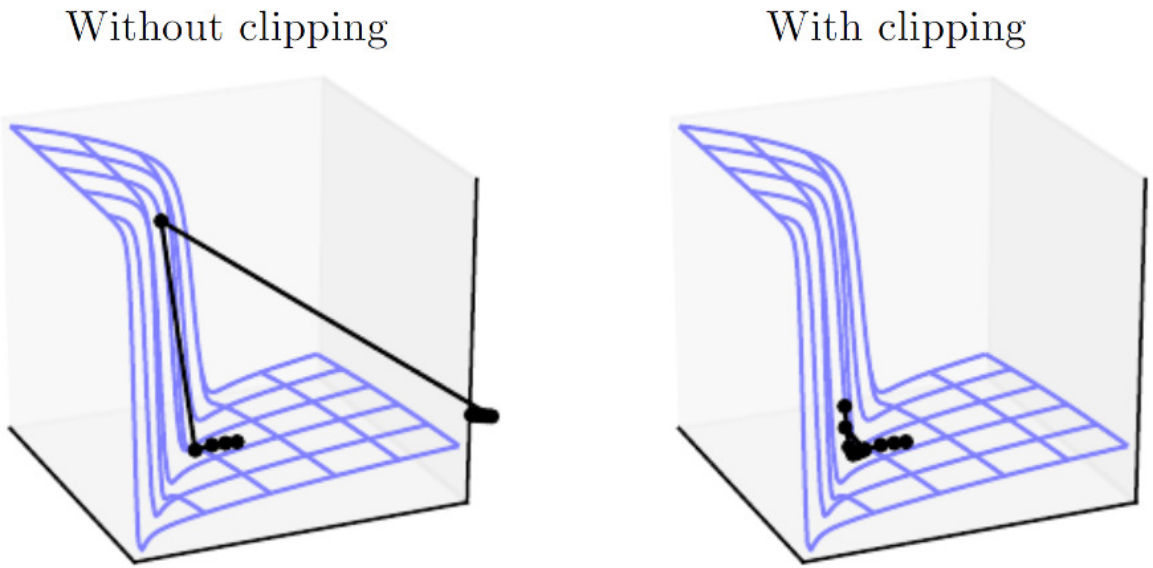
Nadam (Nesterov-accelerated adaptive momentum):
те же формулы для v, $\hat{v}$, G, $\hat{G}$,

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

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

Проблема взрыва градиента и эвристика gradient clipping

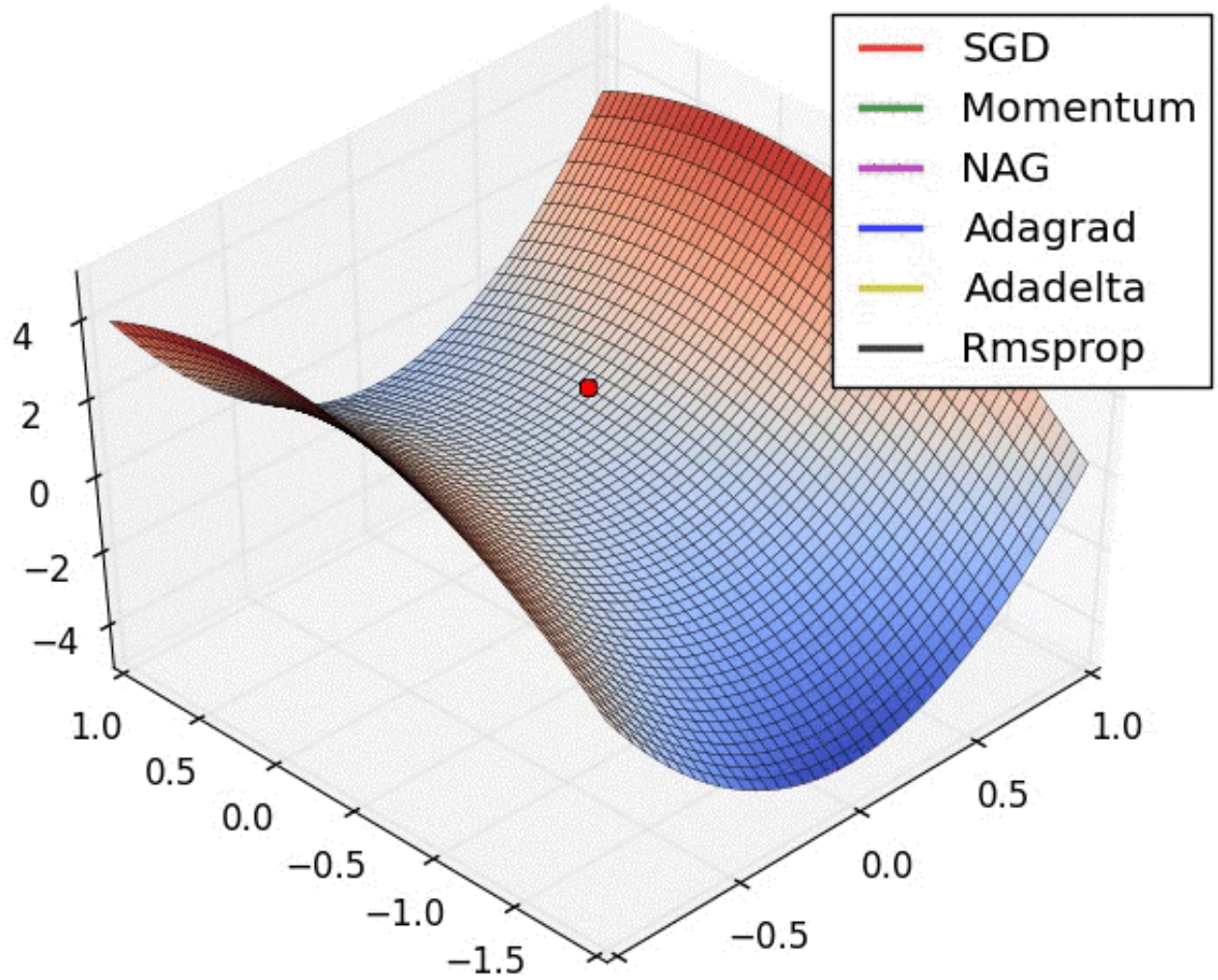
Проблема взрыва градиента (gradient exploding)



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`

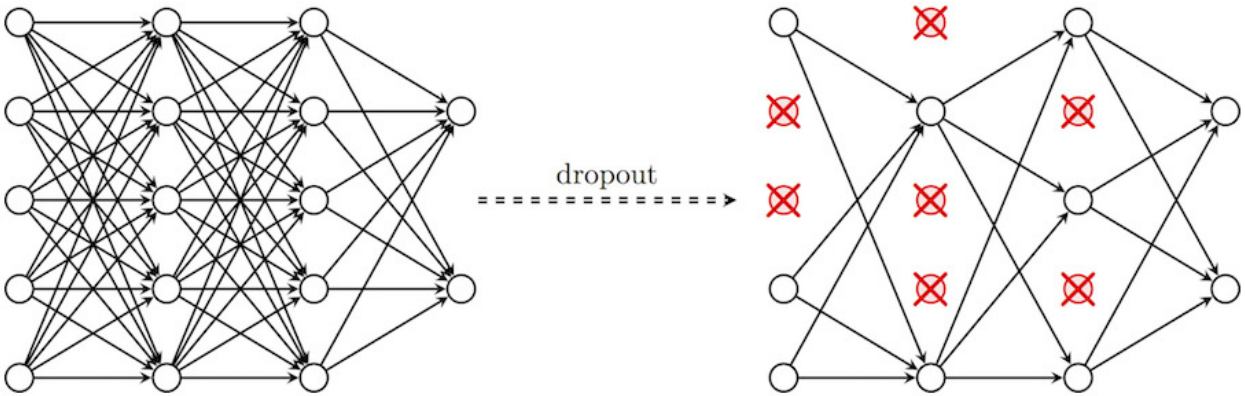
Метод случайных отключений нейронов (Dropout)

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

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

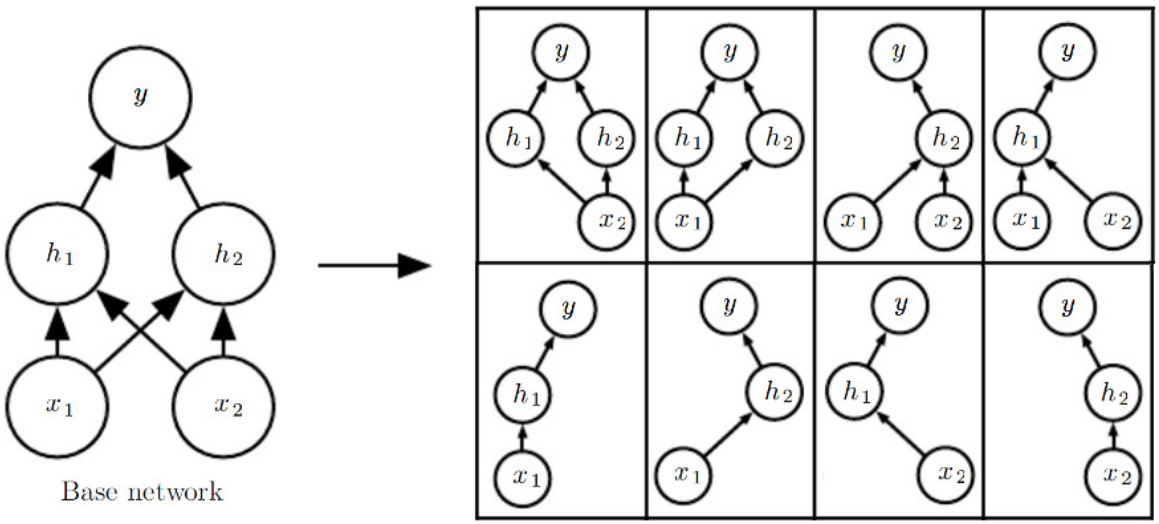
Этап применения: включаем все нейроны, но с поправкой:

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



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 части сети решать одну и ту же исходную задачу вместо того, чтобы подстраиваться под компенсацию ошибок друг друга



Base network

Обратный Dropout и L_2-регуляризация

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

Этап обучения:

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

Этап применения не требует ни модификаций, ни знания p_ℓ:

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

L_2-регуляризация предотвращает рост параметров на обучении:

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

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

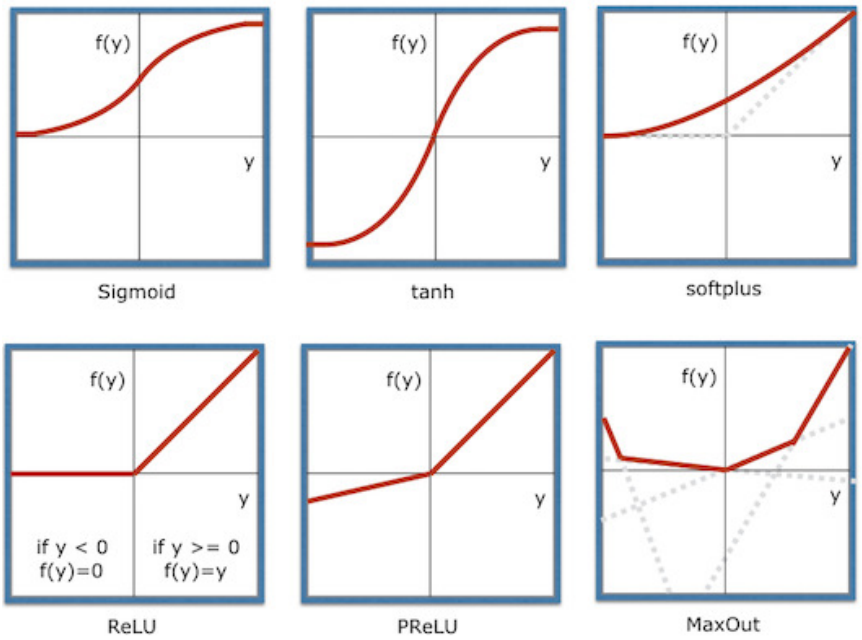
Функции активации ReLU и PReLU (LeakyReLU)

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

Функция положительной срезки (rectified linear unit)

$$\text{ReLU}(y) = \max\{0, y\}$$

$$\text{PReLU}(y) = \max\{0, y\} + \alpha \min\{0, y\}$$

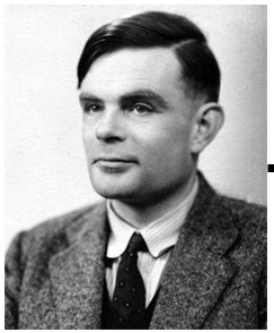


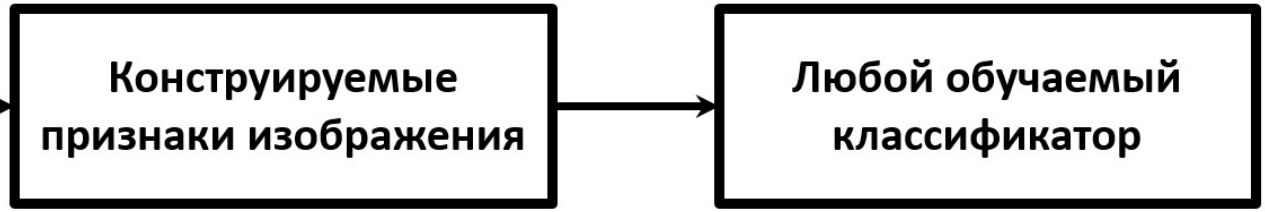
Расширение выборки (dataset augmentation)

Любые преобразования, не меняющие класс объекта

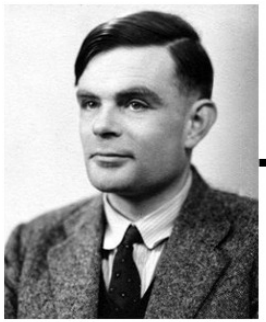
Классификация изображений

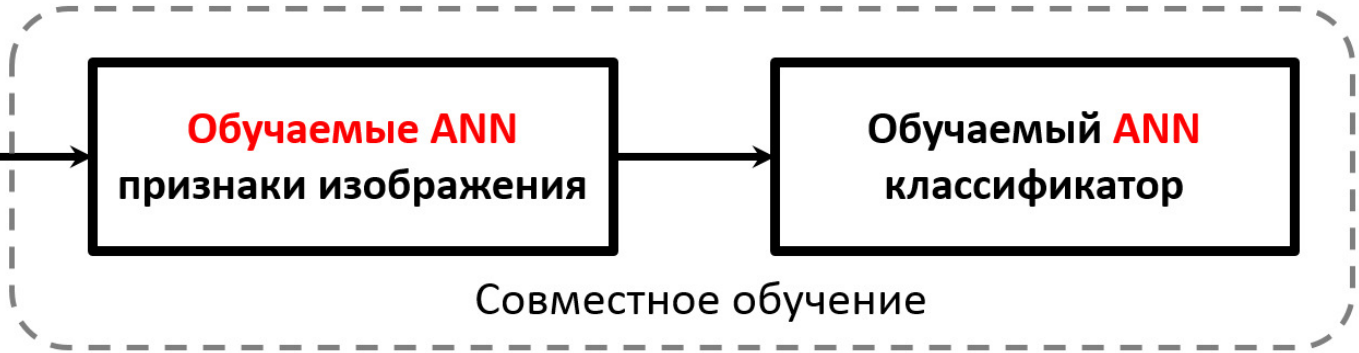
Классический подход к распознаванию изображений:





Современный подход — end-to-end deep learning:



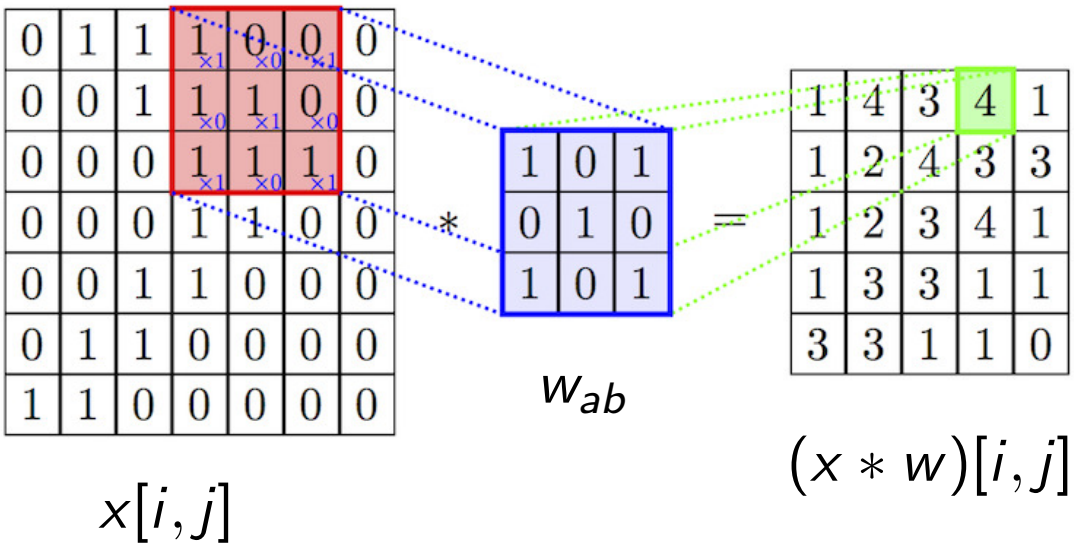


Свёрточный слой нейронов (convolution layer)

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

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

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

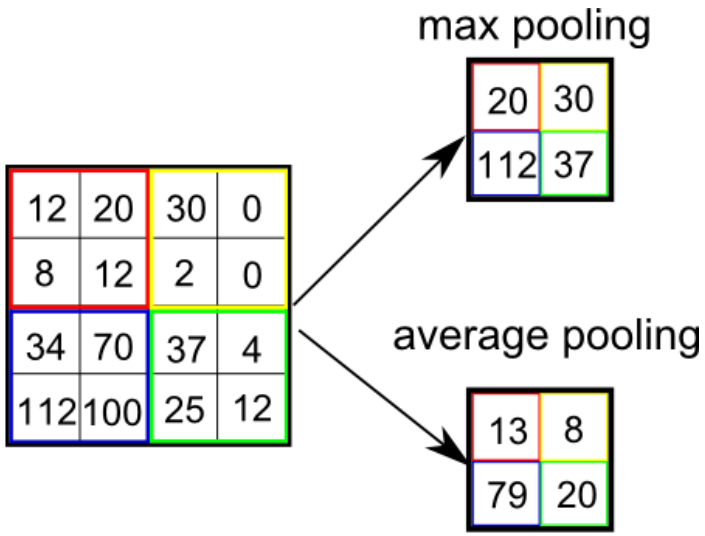


Объединяющий слой нейронов (pooling layer)

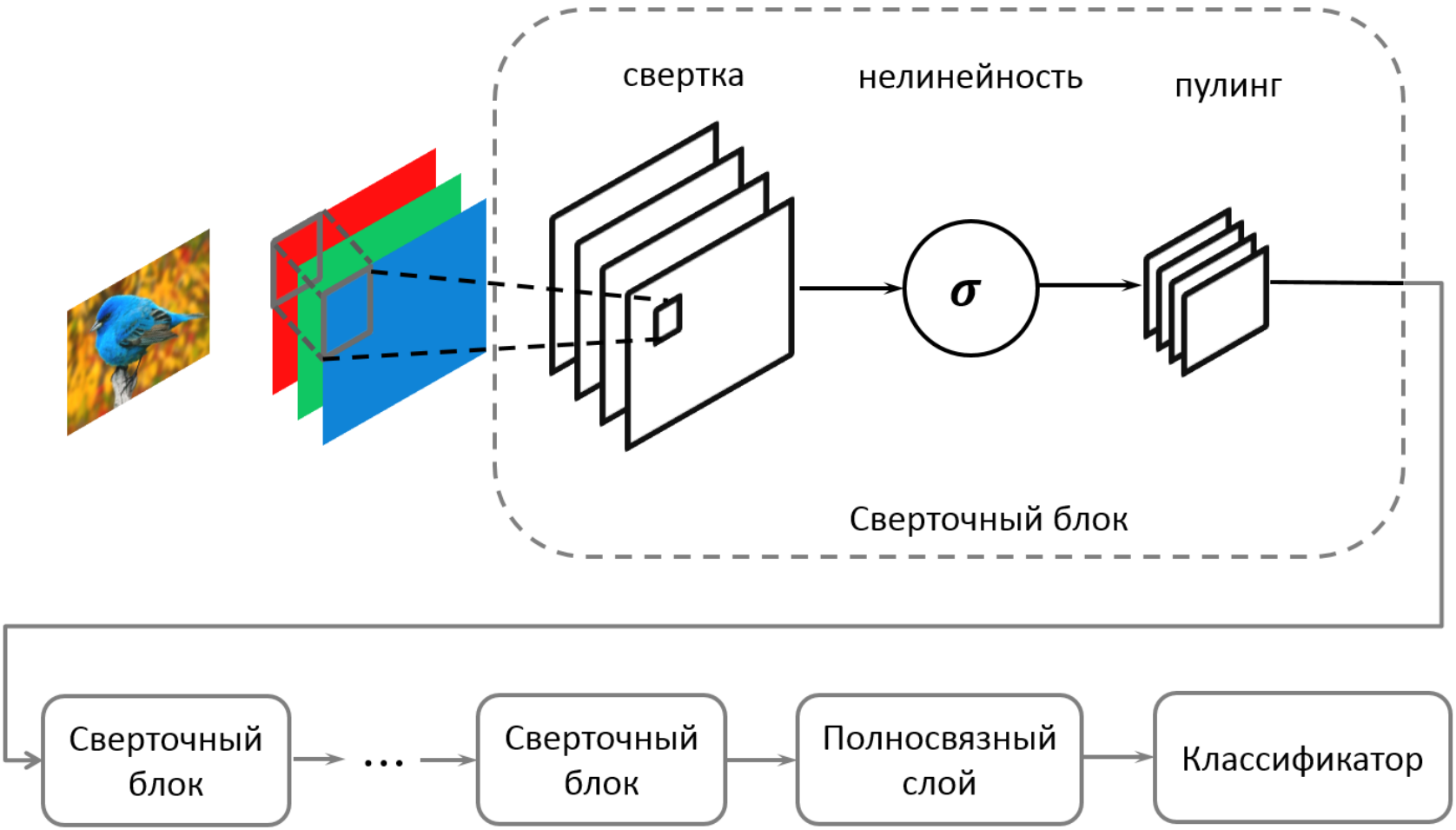
Объединяющий нейрон — это необучаемая свёртка с шагом $h > 1$, агрегирующая данные прямоугольной области $h \times h$:

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

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

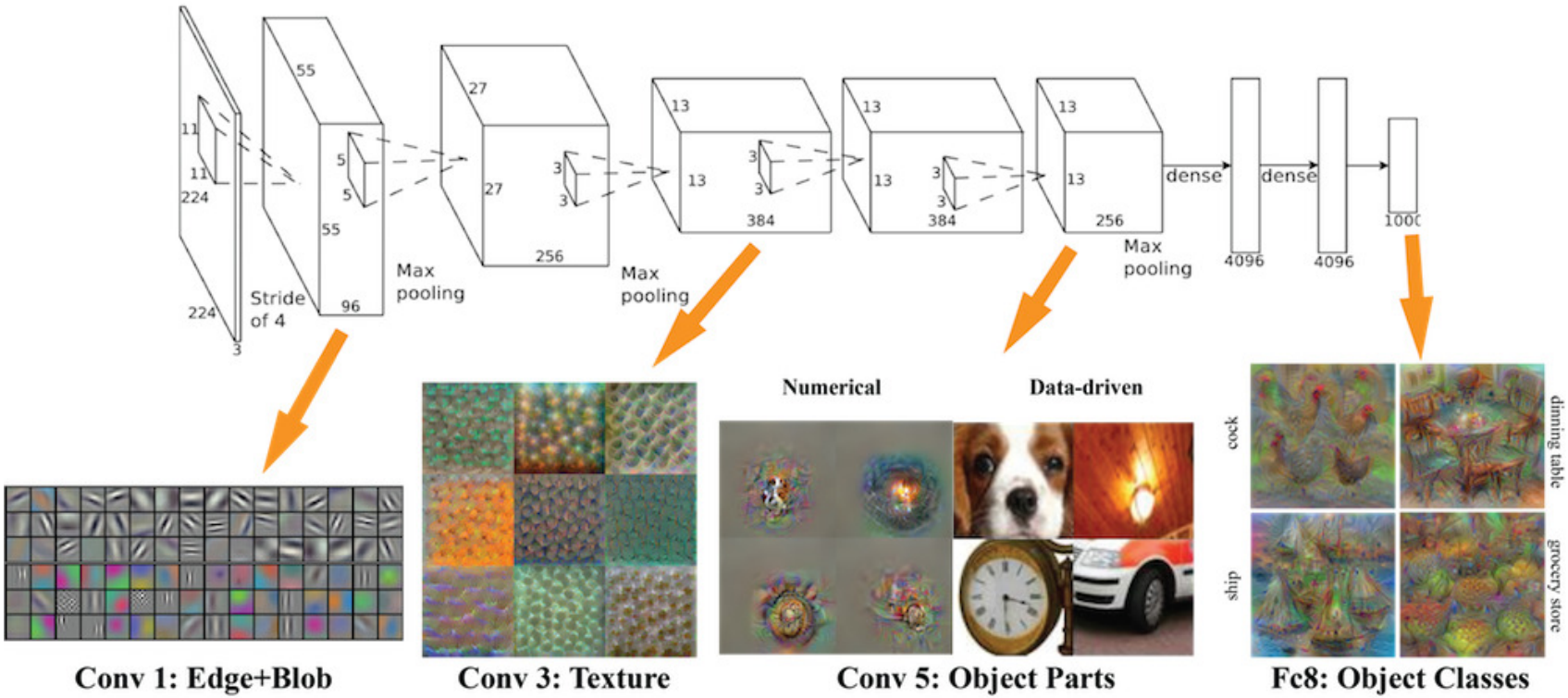


Стандартная схема сверточной сети (Convolutional NN)



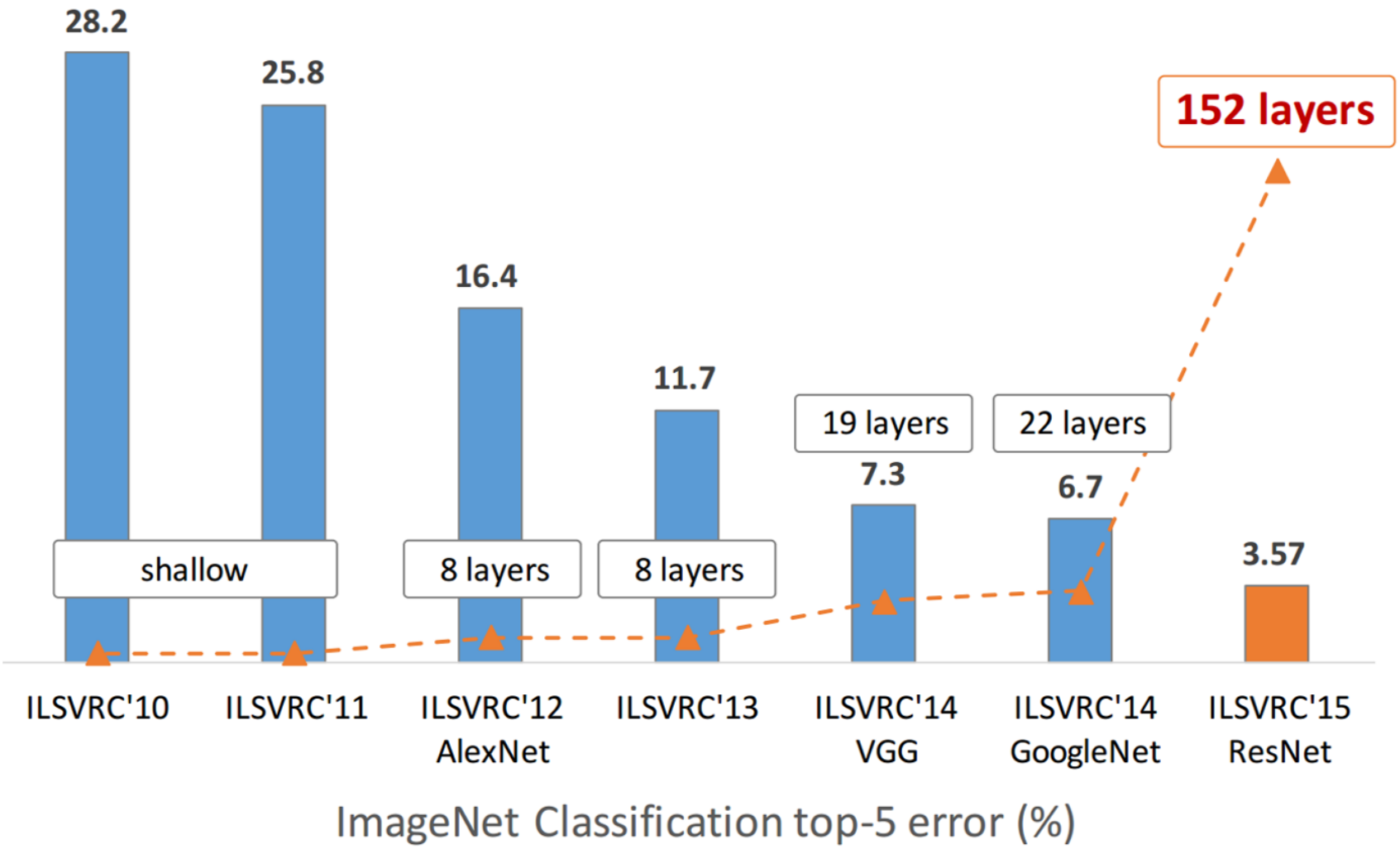
Обучение иерархии признаков

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

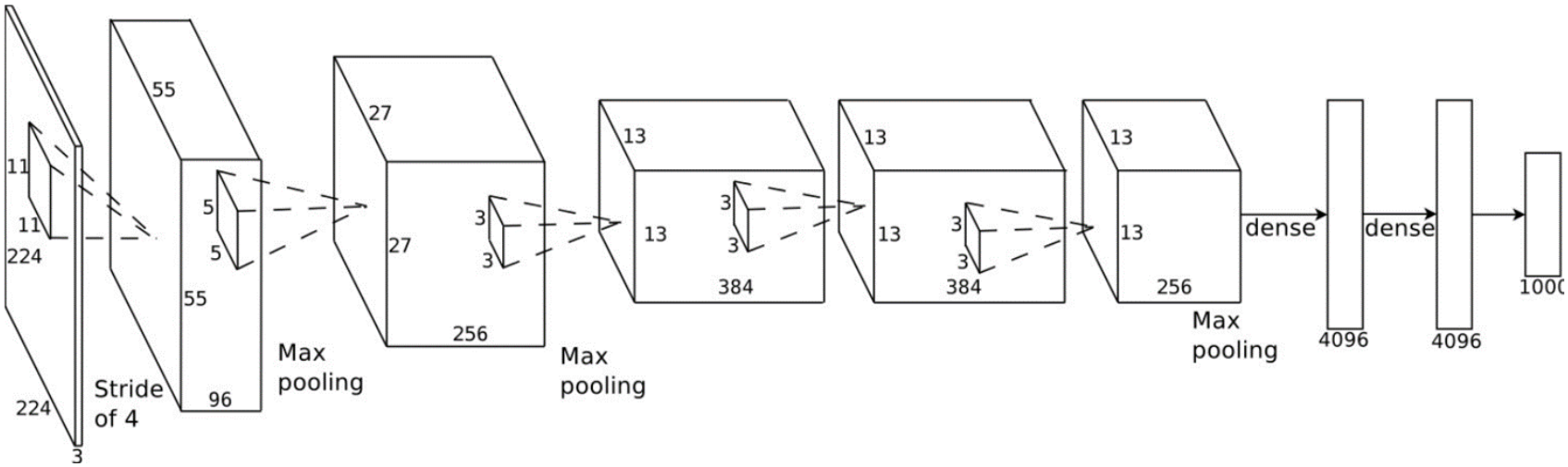


Выборка изображений ImageNet

Развитие свёрточных сетей (краткая история ImageNet)



AlexNet: первый глубокий прорыв на ImageNet

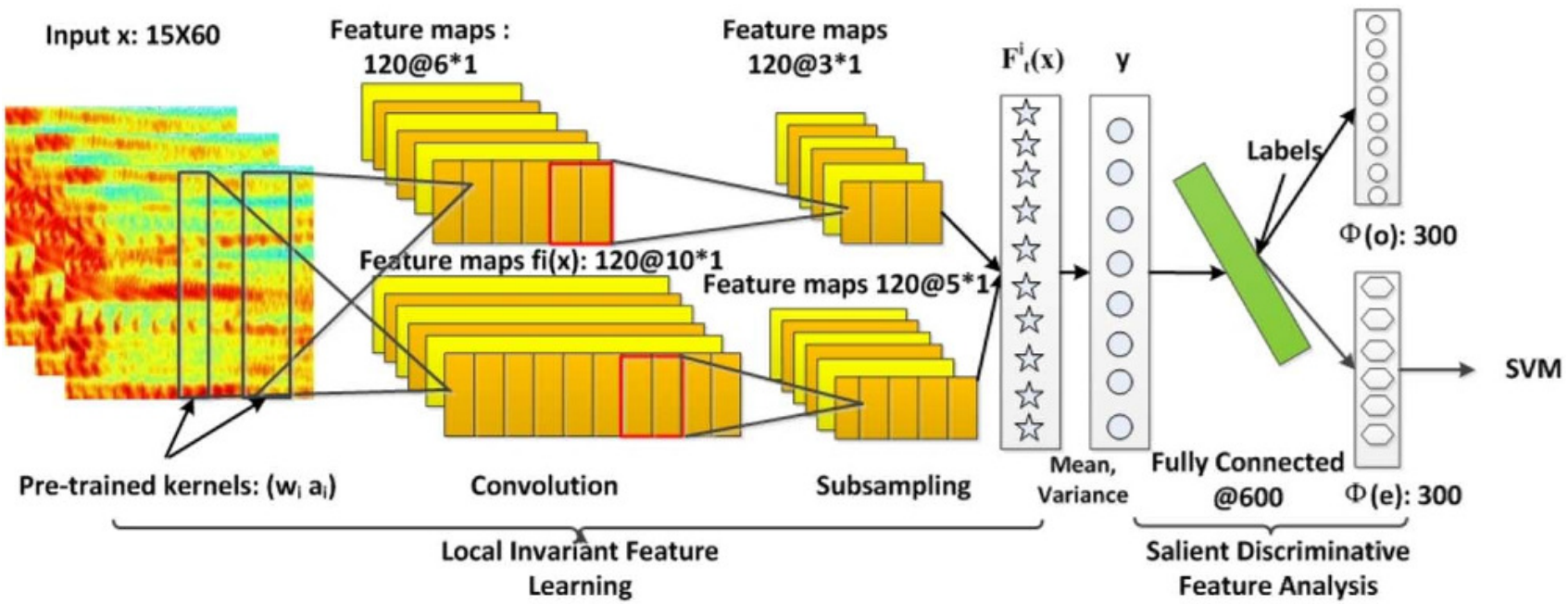


- ReLU + Dropout + расширение выборки
- 60 млн параметров (в основном в полносвязных слоях)
- Подбор размеров фильтров и пулинга
- GPU

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

Приложение: распознавание речевых сигналов

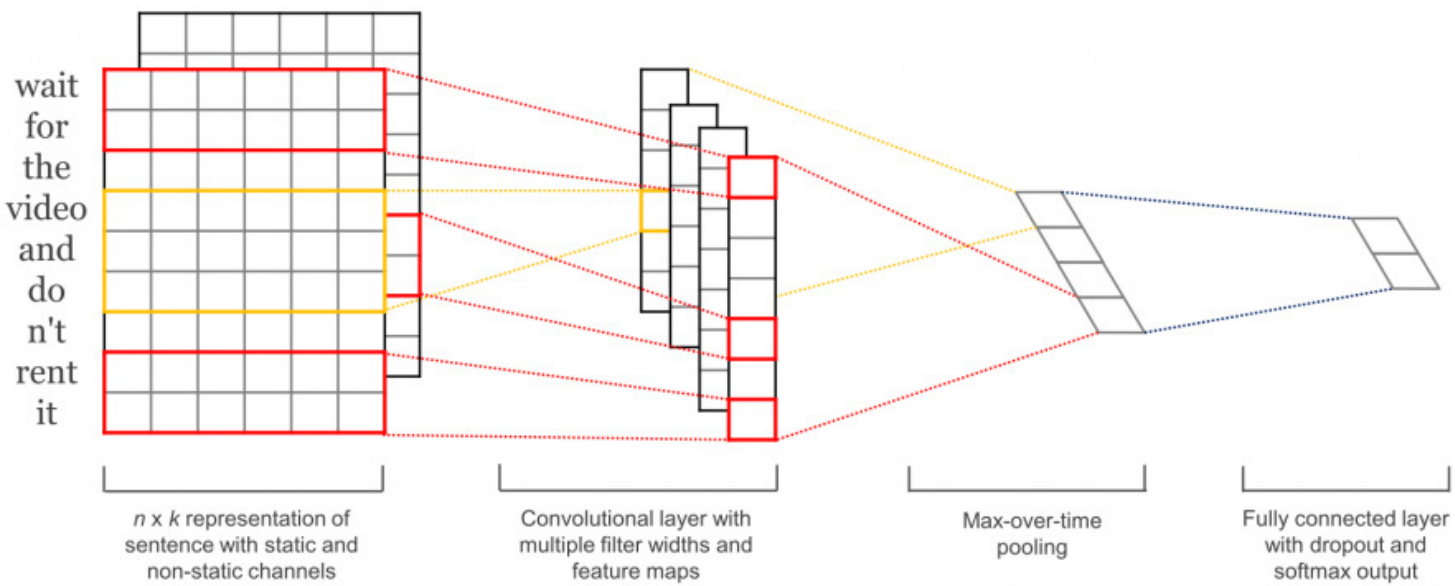
Последовательные фрагменты сигнала представляются векторами спектрального разложения



Qirong Mao, Ming Dong, Zhengwei Huang, Yongzhao Zhan. Learning salient features for speech emotion recognition using convolutional neural networks. 2014.

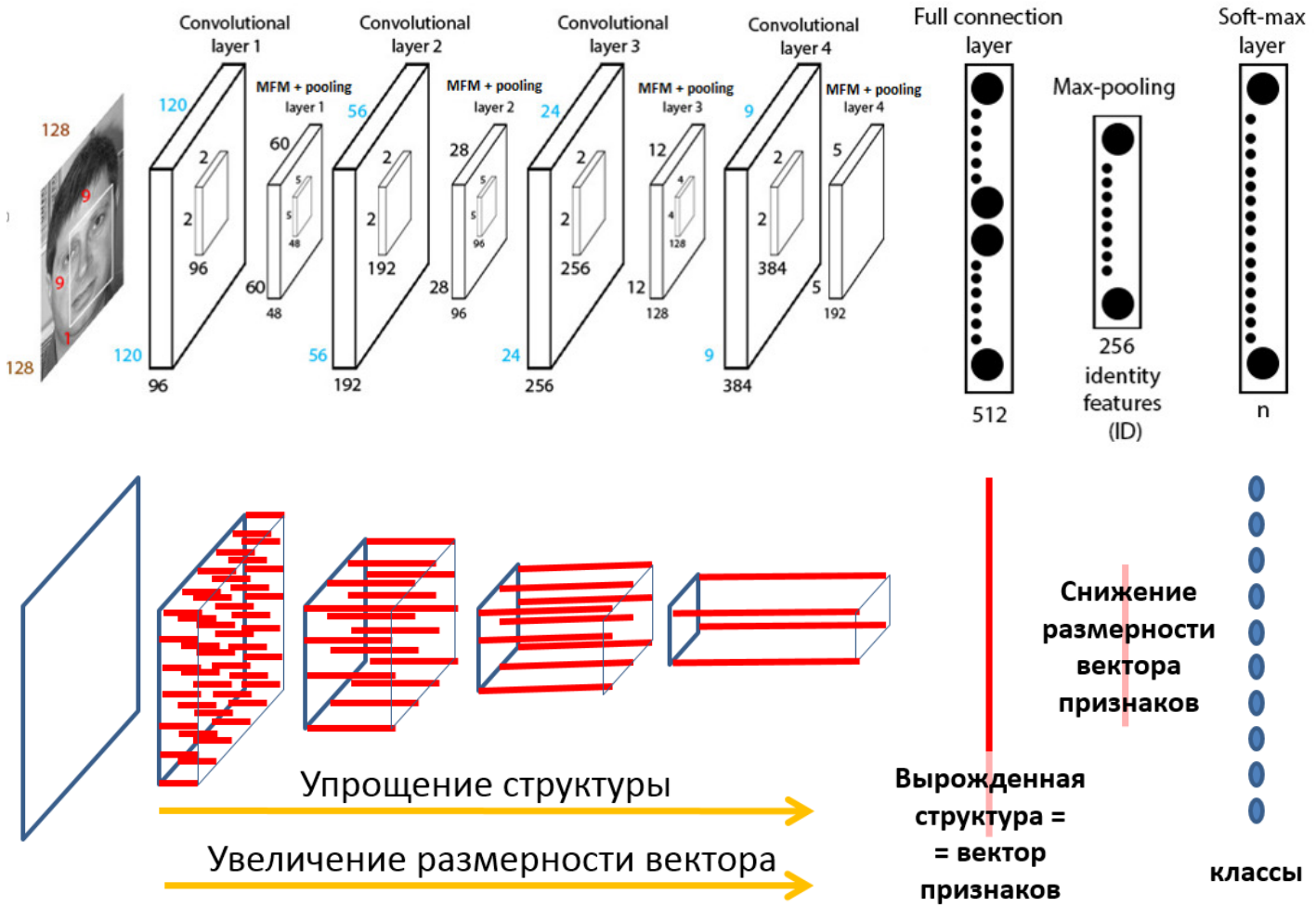
Приложение: классификация предложений в тексте

Последовательные слова представляются векторами
с помощью word2vec или тематической модели



Yoon Kim. Convolutional neural networks for sentence classification. 2014

Идея обобщения CNN на любые структурированные данные



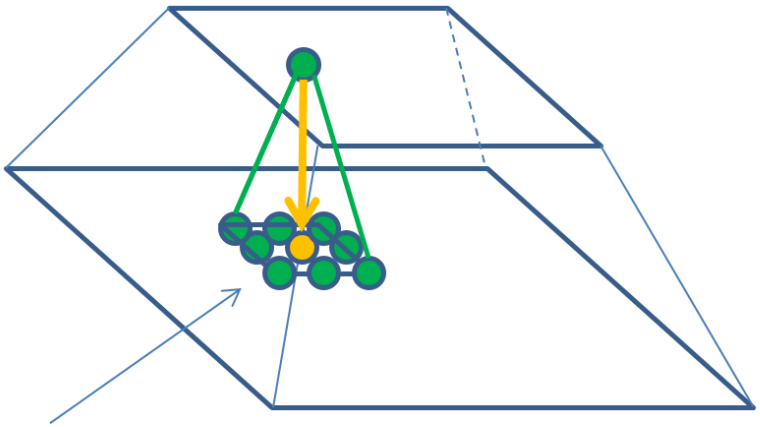
Визильтер Ю.В., Горбацевич В.С. Структурно-функциональный анализ и синтез глубоких конволюционных нейронных сетей. ММРО-2017.

Идея обобщения CNN на любые структурированные данные

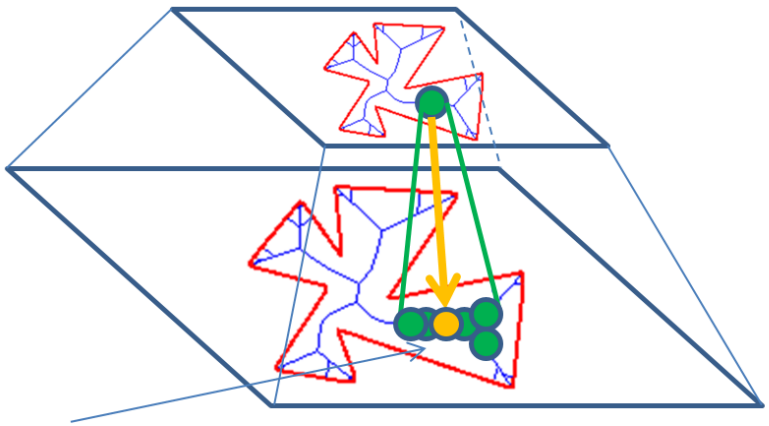
Каждый объект имеет структуру, задаваемую графом

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

Сеть настраивается на обнаружение и классификацию подграфов



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

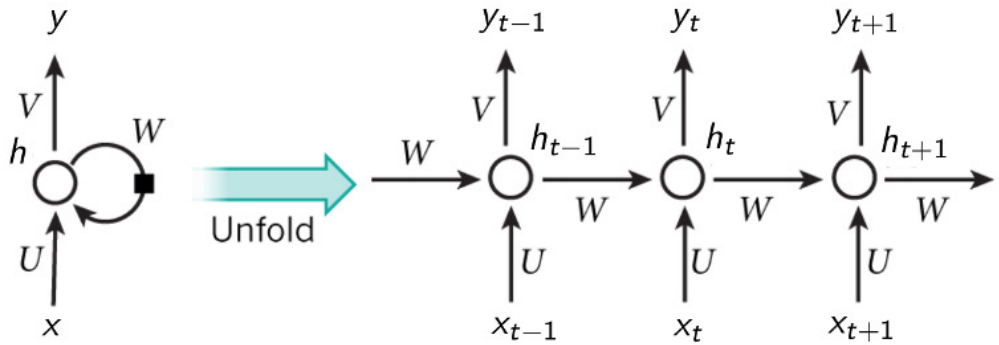
x_t — входной вектор в момент t

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:

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

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

Возможности рекуррентных нейронных сетей

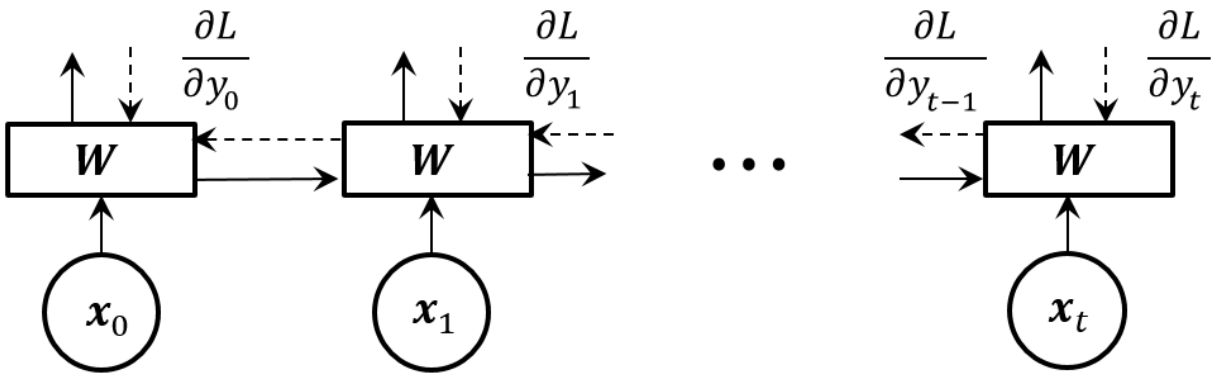
- Классификация текстов или их фрагментов
- Анализ тональности предложений текста
- Машинный перевод
- Speech-to-text / Text-to-speech
- Чат-боты и разговорный интеллект
- Генерация подписей к изображениям
- Генерация рукописного текста
- Интерпретация генома и др. задачи биоинформатики

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

Обучение рекуррентных сетей

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

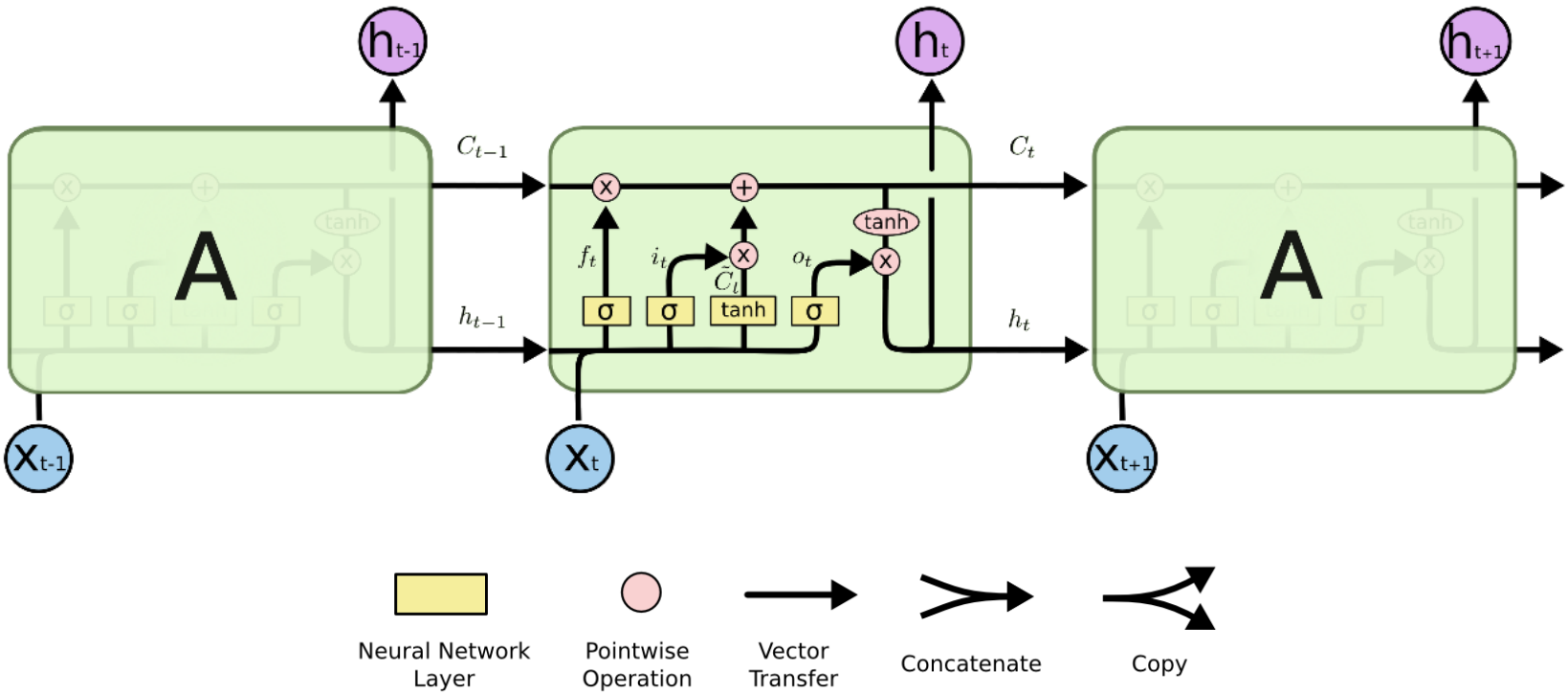
$$\frac{\partial \mathscr{L}_t}{\partial W} = \frac{\partial \mathscr{L}_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\sum_{k=0}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_k}{\partial W}$$



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

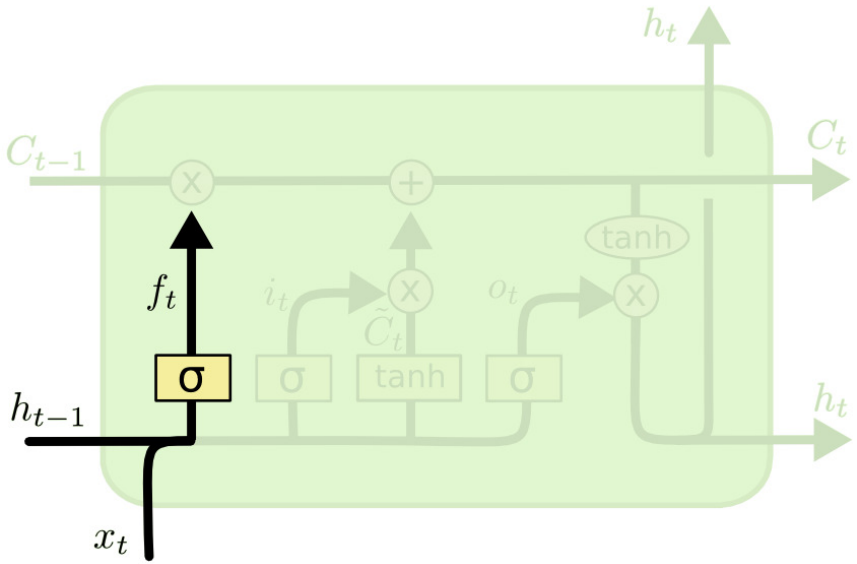
Сети долгой кратковременной памяти (long short-term memory)

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)



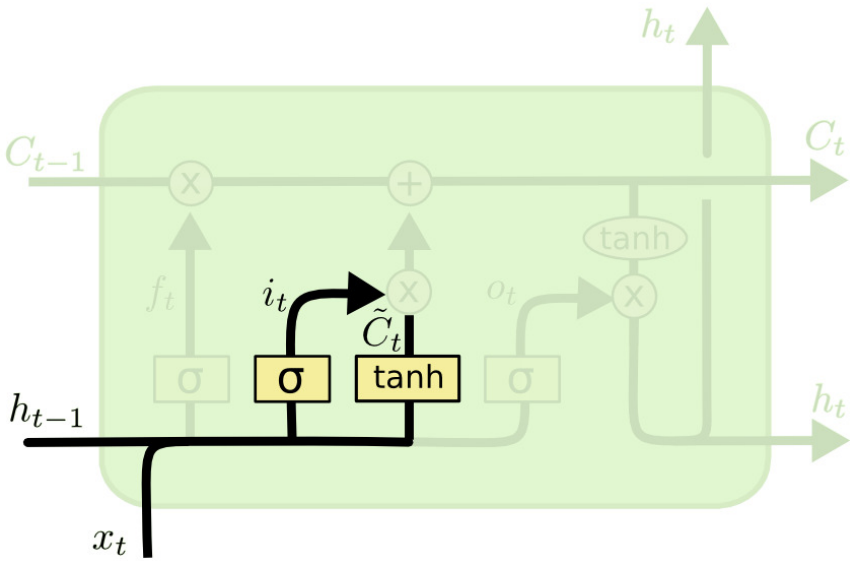
$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big)$$
$$\tilde{C}_t = \mathrm{th}\big(W_C \cdot [h_{t-1}, x_t] + b_C\big)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\big(W_o \cdot [h_{t-1}, x_t] + b_o\big)$$
$$h_t =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

Фильтр 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)



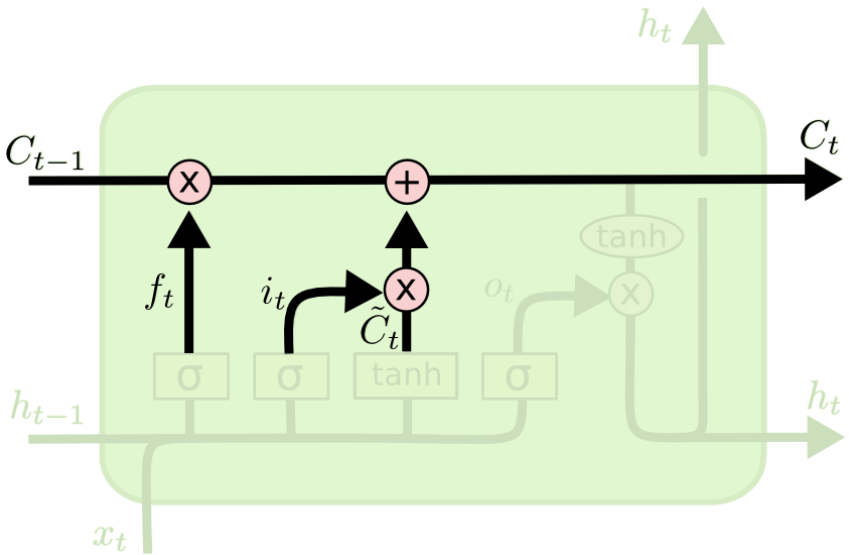
$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big)$$
$$\tilde{C}_t = \text{th}\big(W_C \cdot [h_{t-1}, x_t] + b_C\big)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\big(W_o \cdot [h_{t-1}, x_t] + b_o\big)$$
$$h_t = o_t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

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)



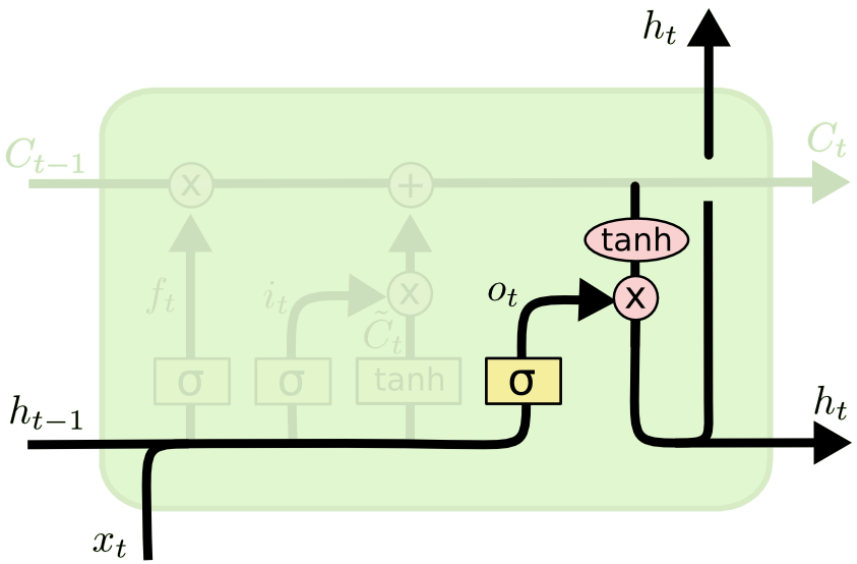
$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big)$$
$$\tilde{C}_t = \mathrm{th}\big(W_C \cdot [h_{t-1}, x_t] + b_C\big)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$o_t = \sigma\big(W_o \cdot [h_{t-1}, x_t] + b_o\big)$$
$$h_t =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

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)



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

Фильтр 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

Резюме

- Продвинутые градиентные методы ускоряют сходимость
- Регуляризации и dropout предотвращают переобучение
- ReLU предотвращает затухание и взрыв градиентов
- Свёрточные сети переводят сложно структурированные данные в вектор фиксированной размерности
- Рекуррентные сети позволяют обрабатывать последовательности векторов
- Подбор архитектуры и параметров сети — это искусство