Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

Образовательный курс
«Введение в глубокое обучение с использованием Intel® neon™ Framework»

Обучение без учителя: автокодировщики, ограниченные машины Больцмана, развертночные сети

При поддержке компании Intel

Кустикова Валентина,
к.т.н., ст.преп. каф. МОСТ ИИТММ,
ННГУ им. Н.И. Лобачевского

Содержание

- Постановка задачи
- Автокодировщики
- Ограниченная машина Больцмана
- Глубокая машина Больцмана
- Глубокая доверительная сеть
- Развертывающие нейронные сети

ПОСТАНОВКА ЗАДАЧИ

Проблема

- Набор данных ImageNET: 14 197 122
- Набор данных PASCAL Visual Object Challenge 2012 (semantic segmentation): ~10 000
- Набор данных IMDB-WIKI: 460 723 + 62 328
- …

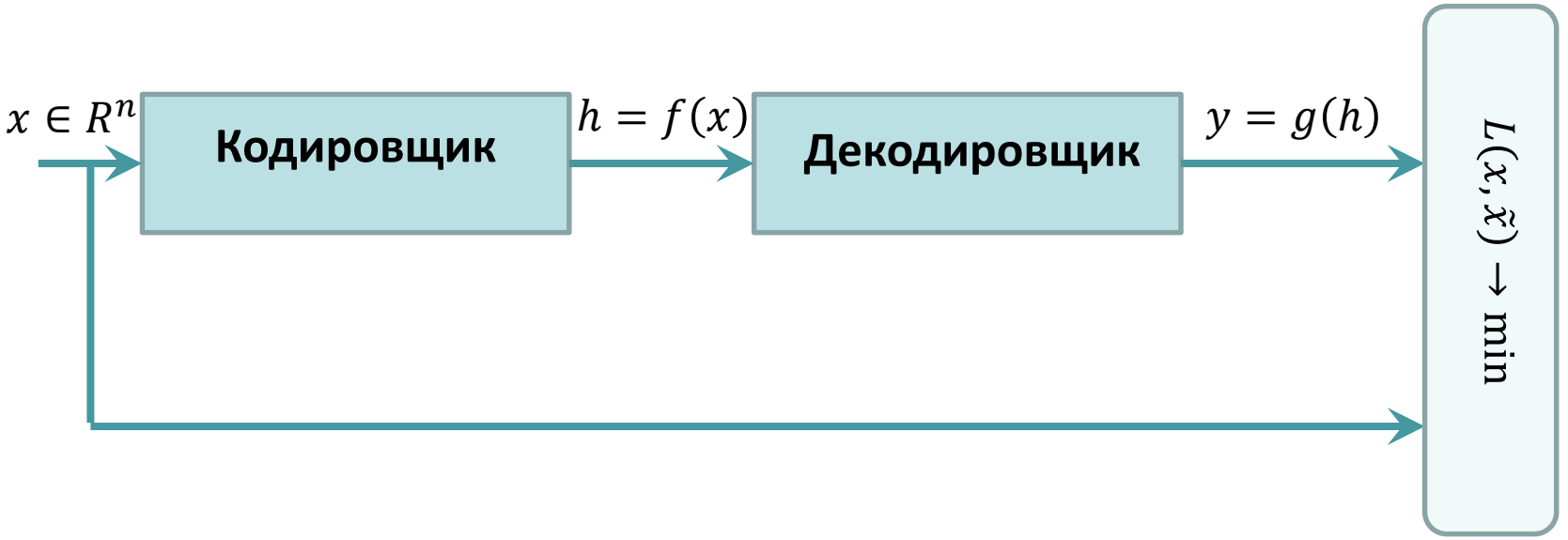
- ***Как снизить объем размеченных данных, которые необходимы для построения эффективно работающих глубоких нейросетевых моделей?***

Задача

- Предварительно настроить веса сети
  - Построить «хорошее» начальное приближение для значений весов с целью последующего обучения сети на основании размеченного тренировочного множества примеров
  - В процессе предварительной настройки не использовать размеченные данные

АВТОКОДИРОВЩИКИ

Автокодировщик (1)



Автокодировщик (2)

- ***Автокодировщик*** (Autoencoder) – нейронная сеть, которая пытается максимально приблизить значения выходного сигнала к входному, т.е. наилучшим образом аппроксимировать тождественное преобразование
- Сеть разделяется на две принципиальные части:
  - ***Кодировщик*** $h = f(x)$, обеспечивающий кодирование входных данных
  - ***Декодировщик*** $y = g(h)$, восстанавливающий по коду вход

- Автокодировщики можно рассматривать как сеть прямого распространения, поэтому для обучения допустимо использовать метод обратного распространения ошибки, основанный на применении градиентных методов

Простейший автокодировщик

- Простейший автокодировщик состоит из двух полносвязных слоев
- Слои описываются преобразованиями следующего вида:

$$h = f(x) = s_f(W_h \cdot x + b_h), \qquad y = g(h) = s_g\left(W_y \cdot h + b_y\right),$$

где функция $s_f(\cdot)$ – функция активации (сигмоидальная функция, гиперболический тангенс, ReLU)

- Обучение сводится к решению задачи минимизации функционала $J(\theta)$ по набору параметров $\theta = \{W_h, W_y, b_h, b_y\}$:

$$J(\theta) = \sum_{x \in D_n} L\left(x, g\left(f(x)\right)\right) \to \min_{\theta}$$

- В качестве функции ошибки $L\left(x, g\left(f(x)\right)\right)$ выбирается квадратичная функция, либо кросс-энтропия

Связь автокодировщиков с анализом главных компонент (1)

- Пусть $L\left(x, g\left(f(x)\right)\right)$ является квадратичной функцией ошибки, тогда минимизируемый функционал имеет вид:

$$J(\theta) = \sum_{x \in D_n} \left\| x - g\left(f(x)\right) \right\|^2$$

- Пусть функции f и g являются линейными, тогда функционал можно записать следующим образом:

$$J(\theta) = \sum_{x \in D_n} \|x - VWx\|^2,$$

где W – матрица прямого преобразования (кодирующий блок), V – матрица обратного преобразования (декодирующий блок)

- Оптимальное решение задачи минимизации соответствует решению задачи снижения размерности пространства (проекция входных данных на подпространство)

Связь автокодировщиков с анализом главных компонент (2)

- Если функции f и g являются нелинейными, то решение задачи минимизации функционала представляет собой решение нелинейной задачи анализа главных компонент

Регуляризованный автокодировщик

- В минимизируемый функционал вносятся параметры регуляризации – функция штрафа
- Квадратичный штраф по параметрами скрытого слоя

$$J(\theta) = \sum_{x \in D_n} L\left(x, g\left(f(x)\right)\right) + \lambda \sum_{i,j} w_{ij}^2 \to \min_{\theta}, \;\; W = \left(w_{ij}\right)$$

Разреженные автокодировщики (1)

- ***Разреженные автокодировщики*** – разновидность регуляризованных автокодировщиков
- В простейшем случае штраф представляет собой L_1-норму сигнала на скрытом слое, и минимизируемый функционал имеет вид:

$$J(\theta) = \sum_{x \in D_n} \left(L\left(x, g\left(f(x)\right)\right) + \lambda \sum_i |h_i| \right) \to \min_\theta$$

- В общем случае функционал выглядит следующим образом:

$$J(\theta) = \sum_{x \in D_n} \left(L\left(x, g\left(f(x)\right)\right) + \Omega\left(f(x)\right) \right) \to \min_\theta$$

Разреженные автокодировщики (2)

- Рассмотрим разреженный автокодировщик как аппроксимацию генеративной модели с латентными переменными, в которой для поиска параметров можно применить метод максимального правдоподобия
- Предположим, что имеется модель, в которой x – видимые переменные (входной сигнал), h – скрытые переменные
- Функция ошибки автокодировщика:

$$L\left(x, g\left(f(x)\right)\right) = -\log p(x|h)$$

- Тогда функция правдоподобия для генеративной модели представляется следующим образом:

$$\log p(x) = \log \sum_{h} p(h, x)$$

Разреженные автокодировщики (3)

- Автокодировщик приближает эту сумму посредством точечной оценки $p(h, x)$ только для одного наиболее вероятного h
- Поэтому для каждого выбранного h максимизируется логарифм совместного распределения:

$$\log p(h, x) = \log p(h) + \log p(x|h)$$

- Если каждая переменная скрытого слоя подчиняется распределению Лапласа $p(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|}$, то получаем функцию штрафа, пропорциональную абсолютному значению:

$$-\log p(h) = \sum_i \left(\lambda|h_i| - \log \frac{\lambda}{2} \right) = \Omega(h) + const,$$

- Штраф представляет не что иное, как L_1-норму
- Выбор другого распределения приводит к другому виду функции штрафа

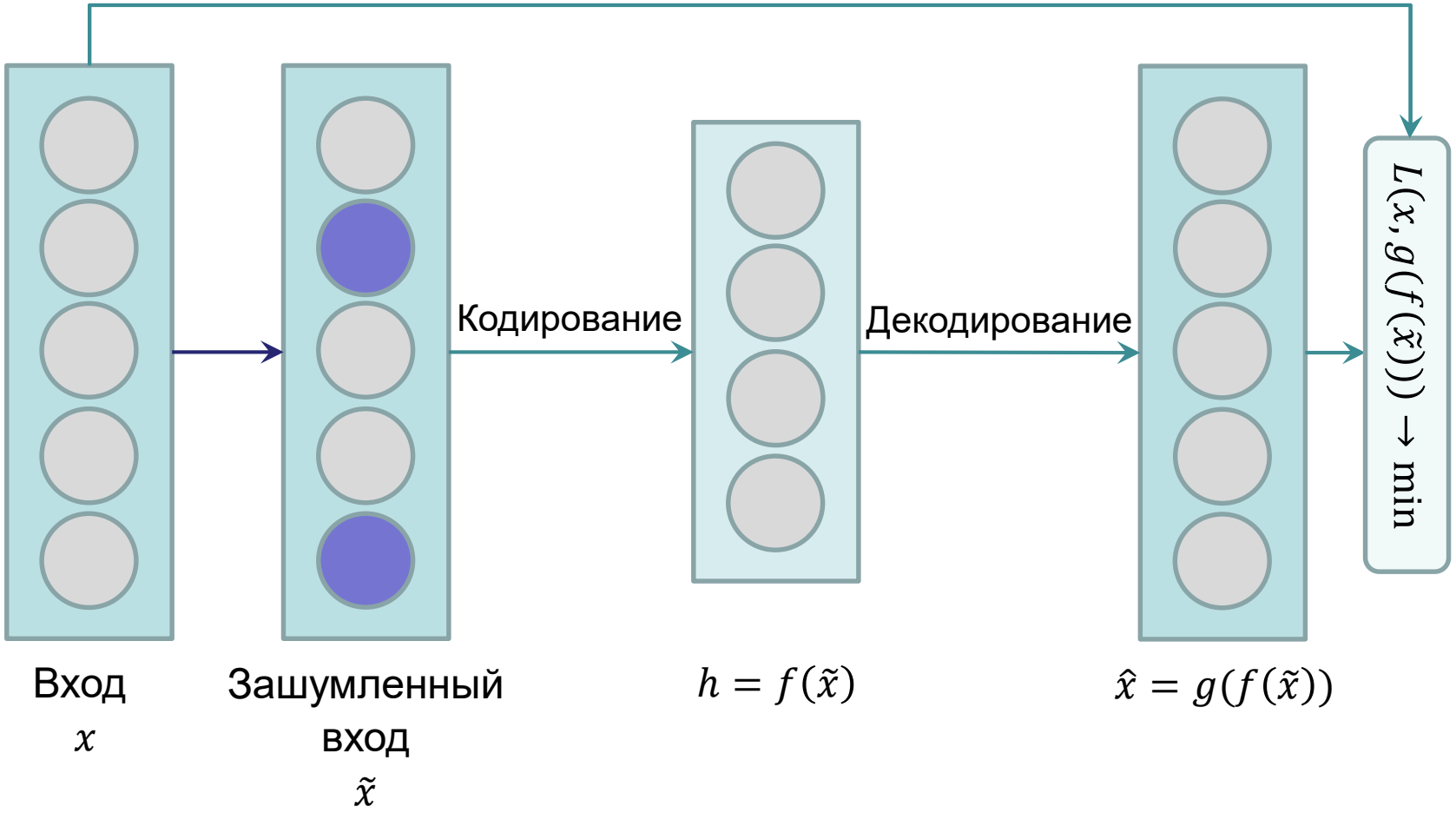
Разреженные автокодировщики (4)

- Разреженные автокодировщики обычно используются для того, чтобы обучить признаки для решения какой-либо задачи (например, задачи классификации)
- Такого рода автокодировщики отражают статистические свойства тренировочного набора данных, а не просто действуют как функция идентификации
- В результате можно получить модель, которая изучила полезные свойства набора данных

Шумоподавляющие автокодировщики (1)

- Цель обучения ***шумоподавляющих автокодировщиков*** – обеспечить шумоподавление у искусственно искаженных входных данных, т.е. восстановить чистый вход из искаженной версии
- Шумоподавляющие автокодировщики минимизируют функцию ошибки $L\left(x, g\left(f\left(\tilde{x}\right)\right)\right)$, где $\tilde{x}$ – копия входного сигнала x, в который добавлен некоторый шум

Шумоподавляющие автокодировщики (2)



Шумоподавляющие автокодировщики (3)

- Минимизируемый функционал имеет вид:

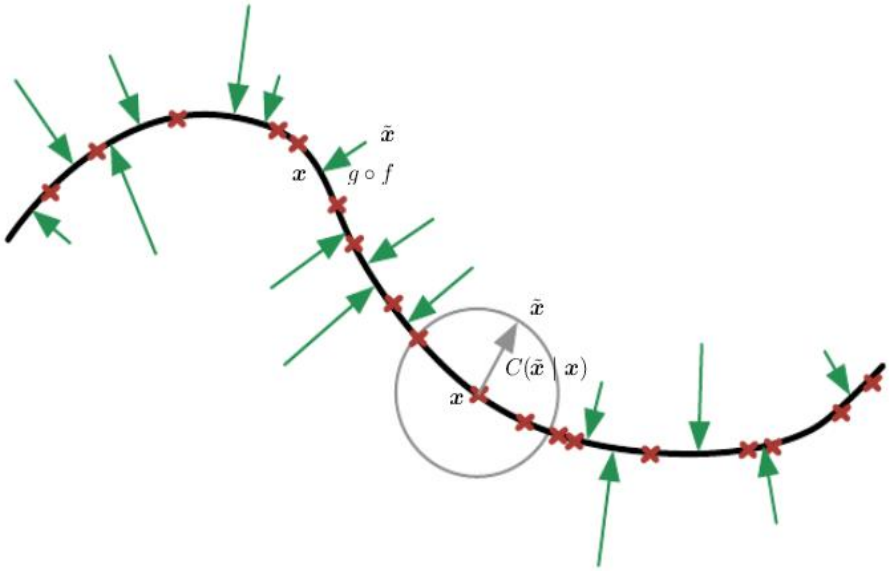
$$J(\theta) = \sum_{x \in D_n} \mathbb{E}_{\tilde{x} \sim C(\tilde{x}|x)} \left(L\left(x, g\left(f(\tilde{x})\right)\right)\right) \to \min_{\theta},$$

где $\mathbb{E}_{\tilde{x} \sim C(\tilde{x}|x)}(.)$ – математическое ожидание по всем зашумленным сигналам $\tilde{x}$, полученным из входного сигнала x в соответствии с некоторым случайным процессом $C(\tilde{x}|x)$

- Возможные искажения:
  - Аддитивный изотропный Гауссовский шум
  - Шум вида «соль и перец» для изображений из градаций серого (случайно возникающие черные и белые пиксели)
  - Маскирующий шум, т.е. установка случайно выбранных входов в ноль (независимо для каждого примера обучения)

Шумоподавляющие автокодировщики (4)

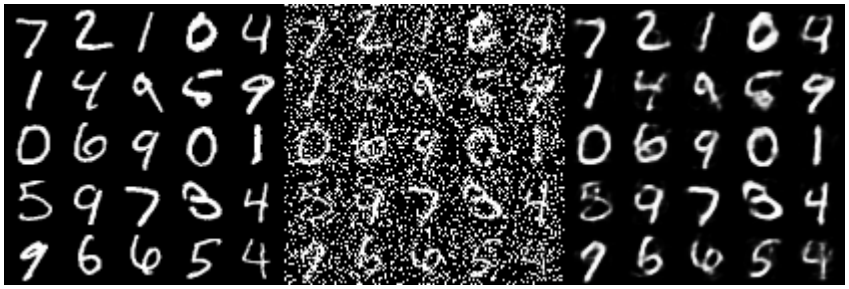
- Красные крестики – тренировочная выборка
- Серая окружность – набор искаженных значений входа x
- Черная линия – многообразие, которое аппроксимирует тренировочную выборку
- Если автокодировщик обучается с целью минимизации квадратичной ошибки, то $g(f(\tilde{x}))$ позволяет оценить $\mathbb{E}_{\tilde{x}\sim C(\tilde{x}|x)}\left(L\left(x, g(f(\tilde{x}))\right)\right)$
- При этом каждый вектор $g(f(\tilde{x})) - \tilde{x}$ указывает в направлении ближайшей точки на многообразии



* Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org].

Пример применения шумоподавляющих авкодировщиков для удаления шумов

- Цель – построить сеть, которая по зашумленному изображению восстанавливает оригинальное изображение без шума (закон формирования шума известен)
- Тренировочный набор данных – множество изображений, не содержащих шум. Тогда x – изображение из тренировочного набора, а $\tilde{x}$ – зашумленное изображение
- Обучение – определение параметров сети, которая позволяет наилучшим образом восстановить изображение из зашумленной копии
- Тестирование – восстановление оригинального изображения на основании входного изображения



Исходные изображения | Зашумленные изображения | Восстановленные изображения

* OpenDeep. Tutorial: Your First Model (DAE) [http://www.opendeep.org/v0.0.5/docs/tutorial-your-first-model].

Сжимающие автокодировщики

- ***Сжимающие автокодировщики*** (Contractive Autoencoder) – разновидность регуляризованных автокодировщиков, для которых минимизируемый функционал имеет вид:

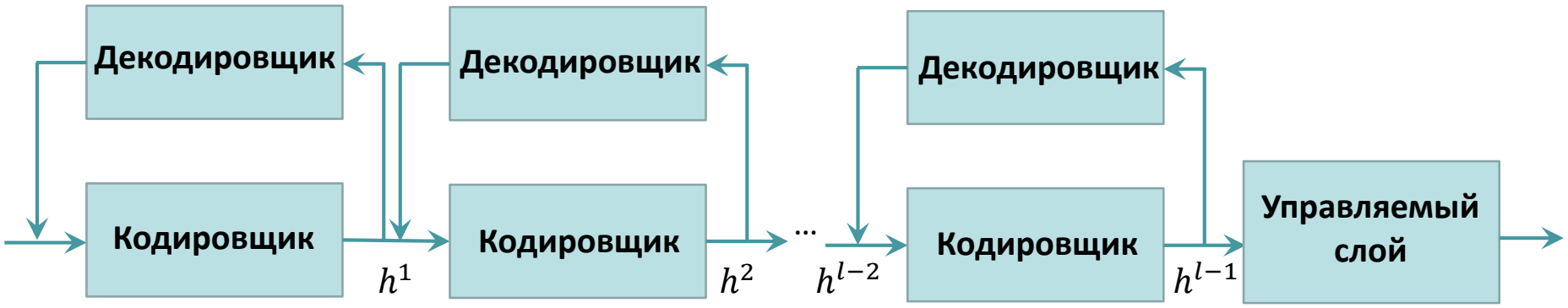
$$J(\theta) = \sum_{x \in D_n} \left(L\left(x, g\left(f(x)\right)\right) + \lambda \left\| J_f(x) \right\|_F^2 \right) \to \min_\theta,$$

где $J_f(x)$ – Якобиан векторной функции $f(x)$, $\left\| J_f(x) \right\|_F^2$ – Фробениусова норма Якобиана $J_f(x)$:

$$J_f(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \cdots & \frac{\partial f_n}{\partial x_n}(x) \end{bmatrix}, \qquad \left\| J_f(x) \right\|_F^2 = \sum_{ij} \left(\frac{\partial f_j}{\partial x_i}(x) \right)^2$$

Стек автокодировщиков

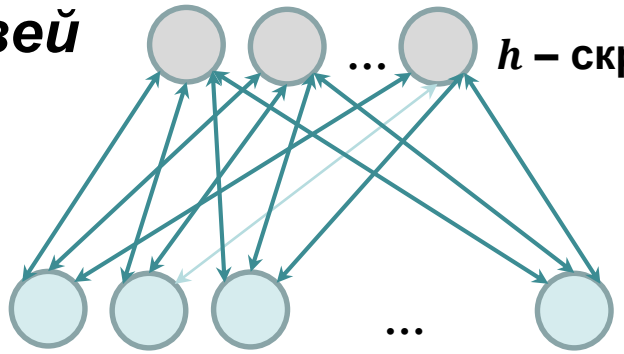
- В случае работы с многослойными сетями можно построить стек автокодировщиков
- Обучается каждый автокодировщик последовательно, что позволяет постепенно снижать размерность пространства признаков и настраивать параметры кодирующих слоев



ОГРАНИЧЕННАЯ МАШИНА БОЛЬЦМАНА

Ограниченная машина Больцмана

- ***Ограниченная машина Больцмана*** (Restricted Boltzmann Machine) – вероятностный аналог автокодировщика
- Машина Больцмана – стохастическая модель, компонентами которой являются стохастические нейроны
- ***Стохастический нейрон*** может находиться в двух вероятностных состояниях, которым формально можно присвоить значения $+1$ (включенное состояние) и -1 (выключенное состояние), либо $+1$ и 0 соответственно
- Для машины Больцмана характерно наличие ***симметричных синаптических связей***



h – скрытые нейроны

x – видимые нейроны

Обозначения

- $x = (x_i)_{0 \le i \le N}$ – видимые нейроны, $h = \left(h_j\right)_{0 \le j \le K}$ – скрытые нейроны
- w_{ij} – синаптические связи между нейронами i и j
  - $w_{ij} = w_{ji}$ – условие симметрии матрицы
  - $w_{ii} = 0$ – отсутствие связи нейрона с самим собой
- Использование сдвига достигается посредством добавления фиктивных элементов между узлом с постоянным сигналом $+1$ и нейроном

Цель обучения (1)

- По аналогии с термодинамикой энергия ограниченной машины Больцмана описывается одним их двух уравнений:
  - Если состояниям нейронов соответствуют $+1$ и -1, то уравнение имеет вид

$$E(x,h) = -\frac{1}{2}\sum_{i=0}^{N}\sum_{\substack{j=0 \\ j\neq i}}^{K} w_{ji}h_j x_i$$

  - Если состояниям соответствуют $+1$ и 0 , то уравнение имеет вид

$$E(x,h) = -\sum_{i=0}^{N}\sum_{\substack{j=0 \\ j\neq i}}^{K} w_{ji}h_j x_i$$

Цель обучения (2)

- Нейронная сеть моделирует функцию совместной плотности вероятности следующего вида:

$$p(x,h) = \frac{e^{-E(x,h)}}{Z}, \qquad Z = \sum_{r=0}^{L-1}\sum_{t=0}^{S-1} e^{-E\left(x^{(r)},h^{(t)}\right)},$$

где Z – нормализующий коэффициент,

L – количество образов вектора видимых нейронов x,

S – количество образов вектора скрытых нейронов,

$p(x,h)$ представляет собой распределение Гиббса

- Задача обучения сети сводится к максимизации функции:

$$p(x) = \sum_{t=0}^{S-1} p\left(x,h^{(t)}\right) = \frac{1}{Z}\sum_{t=0}^{S-1} e^{-E\left(x,h^{(t)}\right)} \to \max$$

Условные вероятности в случае бинарных состояний входного вектора (1)

- Вероятность того, что при данном входном векторе x активируется одно из скрытых состояний $h_k = 1$:

$$E_1 = E(x, h_k = 1) = -\sum_{i=0}^{N}\sum_{\substack{j=0 \\ j\neq i \\ j\neq k}}^{K} w_{ji} h_j x_i - \sum_{\substack{i=0 \\ i\neq k}}^{N} w_{ki} x_i ,$$

$$E_0 = E(x, h_k = 0) = -\sum_{i=0}^{N}\sum_{\substack{j=0 \\ j\neq i \\ j\neq k}}^{K} w_{ji} h_j x_i , \qquad E_1 - E_0 = -\sum_{\substack{i=0 \\ i\neq k}}^{N} w_{ki} x_i$$

$$p(h_k = 1|x) = \frac{e^{-E_1}}{e^{-E_1} + e^{-E_0}} = \frac{1}{1 + e^{-\sum_{\substack{i=0 \\ i\neq k}}^{N} w_{ki} x_i}} = sigm\left(\sum_{\substack{i=0 \\ i\neq k}}^{N} w_{ki} x_i\right),$$

Условные вероятности в случае бинарных состояний входного вектора (2)

- Поскольку все нейроны скрытого слоя независимы, то условную вероятность можно выразить как

$$p(h|x) = \prod_{j=0}^{K} p(h_j|x)$$

- По аналогии можно вывести формулу вычисления $p(x|h)$:

$$p(x_k = 1|h) = sigm\left(\sum_{\substack{j=0 \\ j \neq k}}^{K} w_{jk} h_j\right),$$

$$p(x|h) = \prod_{i=0}^{N} p(x_i|h)$$

Условные вероятности в случае вещественных состояний входного вектора

- Функция энергии имеет модифицированный вид, а $p(x_k|h)$ моделируется с использованием нормального распределения:

$$E(x,h) = -\frac{1}{\sigma}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{K} w_{ji}h_j x_i - \sum_{j=1}^{K} b_j h_j + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i - c_i)^2,$$

$$p(x_k|h) = N\left(\sigma\sum_{j=1}^{K} w_{ji}h_j + c_k, \sigma^2\right),$$

где b_j, c_i – вектора сдвига, σ – стандартное среднеквадратичное отклонение

Вычисление градиентов функции распределения вероятности

- ***Цель обучения:***

$$p(x) = \sum_{t=0}^{S-1} p\left(x, h^{(t)}\right) = \frac{1}{Z} \sum_{t=0}^{S-1} e^{-E\left(x, h^{(t)}\right)} \to \max$$

- Градиенты функции энергии по параметрам:

$$\frac{\partial E(x,h)}{\partial w_{ji}} = -h_j x_i$$

- Производная от экспоненты в отрицательной степени $e^{-E(x,h)}$ и нормализующего коэффициента Z:

$$\frac{\partial e^{-E(x,h)}}{\partial w_{ji}} = e^{-E(x,h)} \frac{\partial\left(-E(x,h)\right)}{\partial w_{ji}} = h_j x_i e^{-E(x,h)}$$

$$\frac{\partial Z}{\partial w_{ji}} = \sum_{r=0}^{L-1} \sum_{t=0}^{S-1} \frac{\partial e^{-E\left(x^{(r)}, h^{(t)}\right)}}{\partial w_{ji}} = \sum_{r=0}^{L-1} \sum_{t=0}^{S-1} h_j^{(r)} x_i^{(t)} e^{-E\left(x^{(r)}, h^{(t)}\right)}$$

Производная функции плотности вероятности

- Производная от функции вероятности $p(x)$ выражается следующим образом:

$$\frac{\partial p(x)}{\partial w_{ji}} = -\frac{1}{Z^2}\frac{\partial Z}{\partial w_{ji}}\sum_{t=0}^{S-1} e^{-E\left(x,h^{(t)}\right)} + \frac{1}{Z}\sum_{t=0}^{S-1}\frac{\partial e^{-E\left(x,h^{(t)}\right)}}{\partial w_{ji}}$$

$$\frac{d\left(f(x)\cdot g(x)\right)}{dx} = \frac{d\left(f(x)\right)}{dx}\cdot g(x) + f(x)\cdot\frac{d\left(g(x)\right)}{dx}$$

$$= -\frac{1}{Z^2}\left(\sum_{r=0}^{L-1}\sum_{t=0}^{S-1} h_j^{(r)} x_i^{(t)} e^{-E\left(x^{(r)},h^{(t)}\right)}\right)\left(\sum_{t=0}^{S-1} e^{-E\left(x,h^{(t)}\right)}\right)$$

$$+\frac{1}{Z}\sum_{t=0}^{S-1} h_j^{(t)} x_i e^{-E\left(x,h^{(t)}\right)}$$

$$= -\frac{1}{Z}p(x)\sum_{r=0}^{L-1}\sum_{t=0}^{S-1} h_j^{(r)} x_i^{(t)} e^{-E\left(x^{(r)},h^{(t)}\right)} + \sum_{t=0}^{S-1} h_j^{(t)} x_i\, p\left(x,h^{(t)}\right)$$

Переход к максимизации логарифма функции распределения вероятности

- Максимизация вероятности равносильна максимизации логарифма вероятности:

$$p(x) \to \max \quad \Longrightarrow \quad \ln p(x) \to \max$$

Производная логарифма по параметрам системы

- Производная от логарифма функции вероятности по параметрам системы:

$$\frac{\partial \ln p(x)}{\partial w_{ji}} = \frac{1}{p(x)} \frac{\partial p(x)}{\partial w_{ji}}$$

$$= -\frac{1}{Z} \sum_{r=0}^{L-1} \sum_{t=0}^{S-1} h_j^{(r)} x_i^{(t)} e^{-E\left(x^{(r)}, h^{(t)}\right)} + \sum_{t=0}^{S-1} h_j^{(t)} x_i \frac{p\left(x, h^{(t)}\right)}{p(x)}$$

$$= -\sum_{r=0}^{L-1} \sum_{t=0}^{S-1} h_j^{(r)} x_i^{(t)} p\left(x^{(r)}, h^{(t)}\right) + \sum_{t=0}^{S-1} h_j^{(t)} x_i \, p\left(h^{(t)} | x\right)$$

Правило Больцмана

- Формула для изменения весов сети в ходе обучения:

$$\Delta w_{ji} = \eta \frac{\partial \ln p\left(x^{(k)}\right)}{\partial w_{ji}} = \eta \left(M\left[x_i^{(k)} h_j\right] - M\left[x_i h_j\right] \right),$$

где $M[.]$ – математическое ожидание дискретной случайной величины, а η – скорость обучения (параметр метода)

- Данное правило называется ***правилом обучения Больцмана*** (Boltzmann Learning Rule)

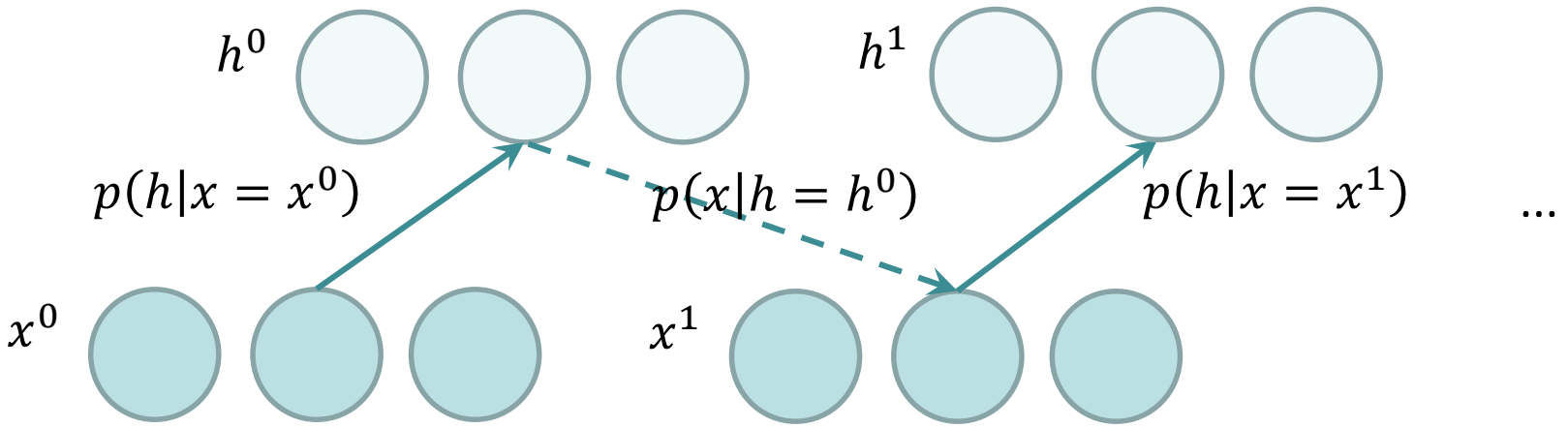
Алгоритм обучения (1)

- Алгоритм обучения машины Больцмана предложен Дж. Хинтоном в 2002 году, основывается на процедуре ***сэмплирования Гиббса*** (Gibbs sampling)
- Алгоритм обучения имеет название ***алгоритма контрастной дивергенции*** (Contrastive Divergence, CD-k)
- Алгоритм описывает схему вычислений в ходе обучения весов машины Больцмана

Алгоритм обучения (2)

1. Состояние видимых нейронов x приравнивается к входному образу сети $x^0 = x^{(0)}$
2. Выводятся вероятности состояний скрытого слоя h^0 из распределения $p(h|x = x^0)$
3. Цикл по индексу t, состоящий из k итераций. При этом осуществляется сбор статистики
    1. Выводятся вероятности состояний видимого слоя x^t из распределения $p(x|h^{t-1})$
    2. Выводятся вероятности состояний скрытого слоя h^t из распределения $p(h|x = x^t)$
4. Коррекция весов сети и переход к следующему входному образу

Алгоритм обучения (3)

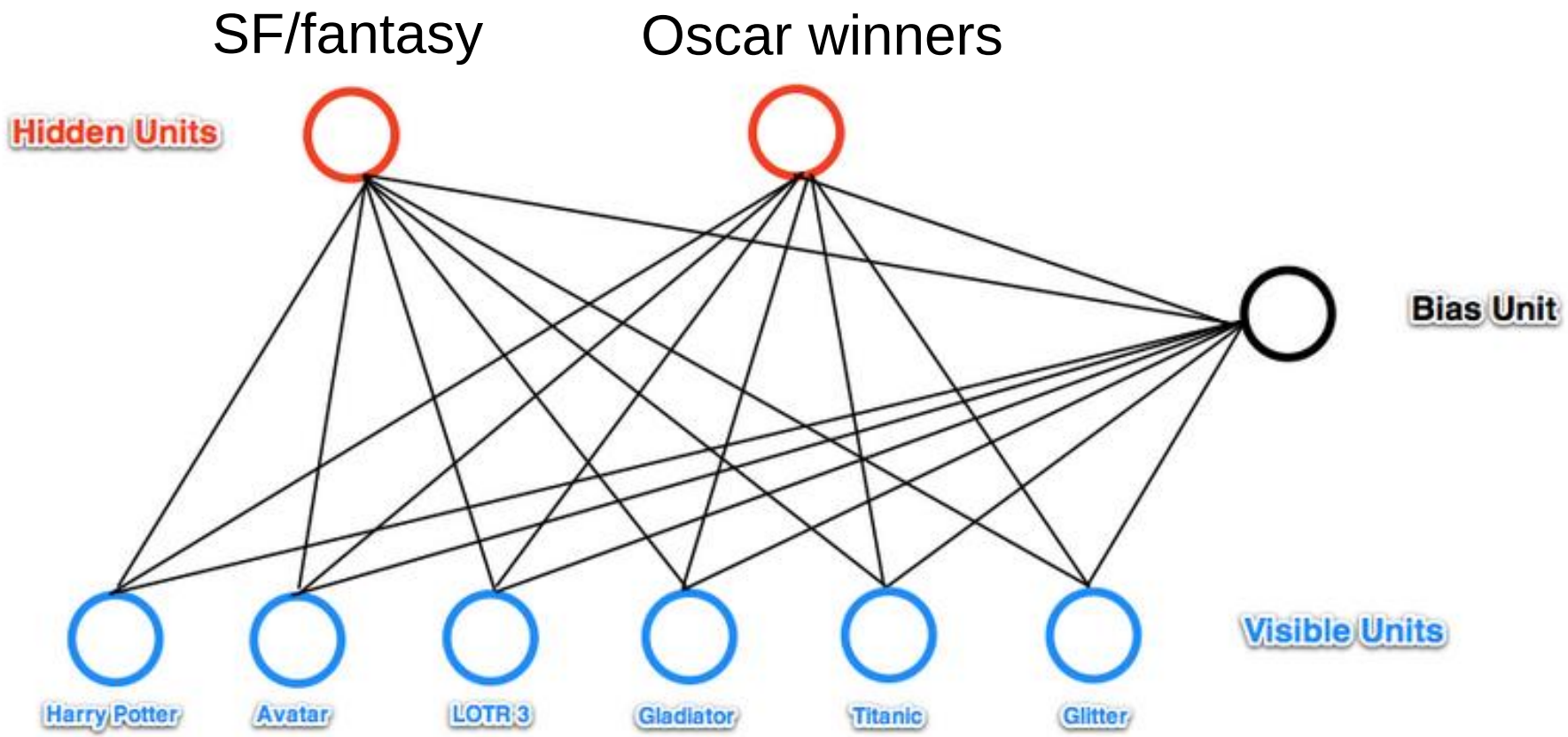


Интерпретация работы ограниченной машины Больцмана на примере (1)

- Видимые нейроны = фильмы
- Скрытые нейроны = группы фильмов (фантастика; фильмы, получившие Оскар)

- Человек выбирает из 6 фильмов такие, которые ему нравятся => активирует входные элементы (бинарные значения)
- По каждому фильму принимается решение о принадлежности определенной группе фильмов => активируется тот или иной скрытый элемент (бинарные состояния)
- 6 видимых элементов отправляют сообщения скрытым переменным
- Энергия активации скрытого элемента – взвешенная сумма сообщений, приходящих от входных элементов

Интерпретация работы ограниченной машины Больцмана на примере (2)



Интерпретация работы ограниченной машины Больцмана на примере (3)

- Проход снизу-вверх:
  - RBM пытается объяснить предпочтения человека с точки зрения скрытых переменных, т.е. фильмы какой группы предпочитает человек
  - При этом если человек выбрал фильмы Harry Potter, Avatar, и LOTR 3, то это не означает, что будет активирован скрытый элемент, который соответствует группе SF/fantasy
  - Это означает, что будет увеличена вероятность активации этого скрытого элемента вследствие высокой энергии активации

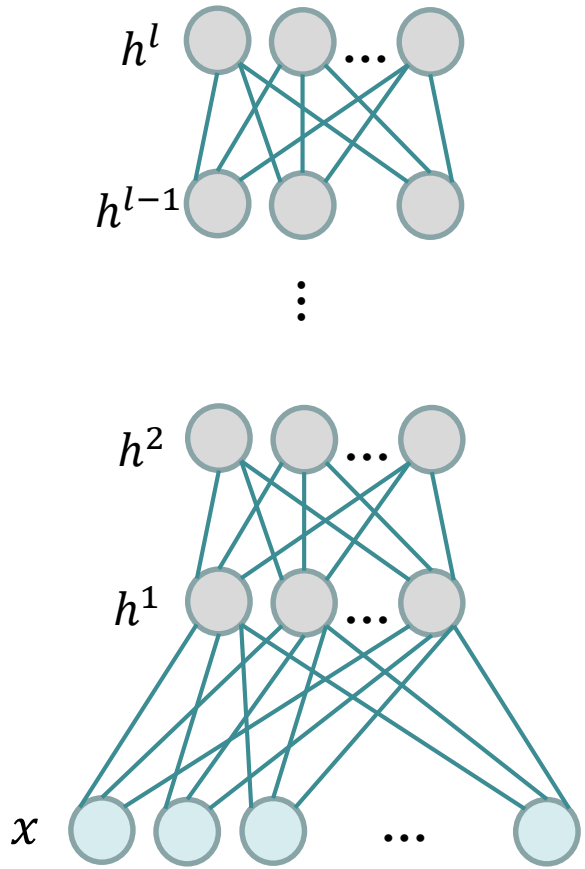
Интерпретация работы ограниченной машины Больцмана на примере (4)

- Проход сверху-вниз:
  - Если известно, что некоторый человек предпочитает SF/fantasy, то можно спросить RBM, какие фильмы относятся к указанной группе и какие фильмы могут понравится данному человеку
  - В этом случае скрытые переменные отправляют сообщения видимым => обновляются состояния видимых переменных
  - При этом нельзя гарантировать, что человеку всегда будут порекомендованы фильмы Harry Potter, Avatar, и LOTR 3. Может быть рекомендовано только некоторое подмножество

ГЛУБОКАЯ МАШИНА БОЛЬЦМАНА И ГЛУБОКАЯ СЕТЬ ДОВЕРИЯ

Глубокая машина Больцмана

- По аналогии со стеком автокодировщиков можно сформировать стек ограниченных машин Больцмана
- Если состояния нейронов каждого скрытого слоя зависят от предыдущего и следующего слоя (симметричные связи), то такая модель называется ***глубокой машиной Больцмана*** (Deep Boltzmann Machine, DBM)



Обучение глубокой машины Больцмана (1)

- Параметры данной модели можно обучать по аналогии с однослойной машиной Больцмана
- Для этого необходимо определить функцию совместного распределения через энергию системы $E\left(x, h^1, h^2, \dots, h^l;\ \theta\right)$, выразить вероятность вектора видимых нейронов $p(x;\theta)$:

$$E\left(x, h^1, h^2, \dots, h^l;\ \theta\right) = -\sum_{i=1}^{l} \left(h^{i-1}\right)^T W^i h^i,$$

$$p(x;\theta) = \frac{1}{Z} \sum_{h^1, h^2, \dots h^l} e^{-E\left(x, h^1, h^2, \dots, h^l;\ \theta\right)},$$

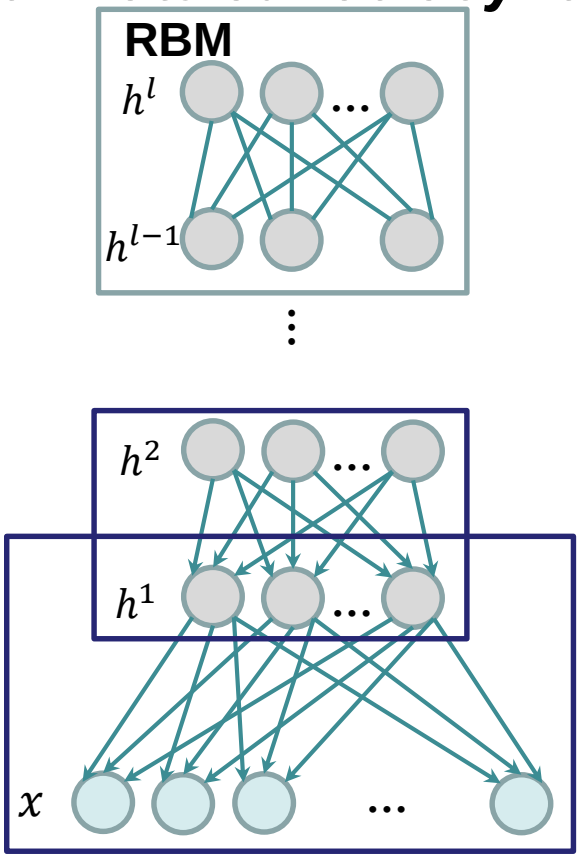
где $x = h^0$ – вектор видимых нейронов, h^i – вектор скрытых нейронов, $\theta = \{W^1, \dots, W^l\}$ – набор параметров системы

Обучение глубокой машины Больцмана (2)

- Далее необходимо получить формулы для вычисления условных вероятностей $p\left(x\middle|h^{(1)}\right)$, $p\left(h^k\middle|h^{k-1}\right)$, $p\left(h^k\middle|h^{k-1},h^{k+1}\right)$ и производных функции распределения вероятности по каждому параметру системы
- Процедура обучения весов будет проходить тем медленнее, чем дальше скрытые нейроны расположены от слоя видимых нейронов

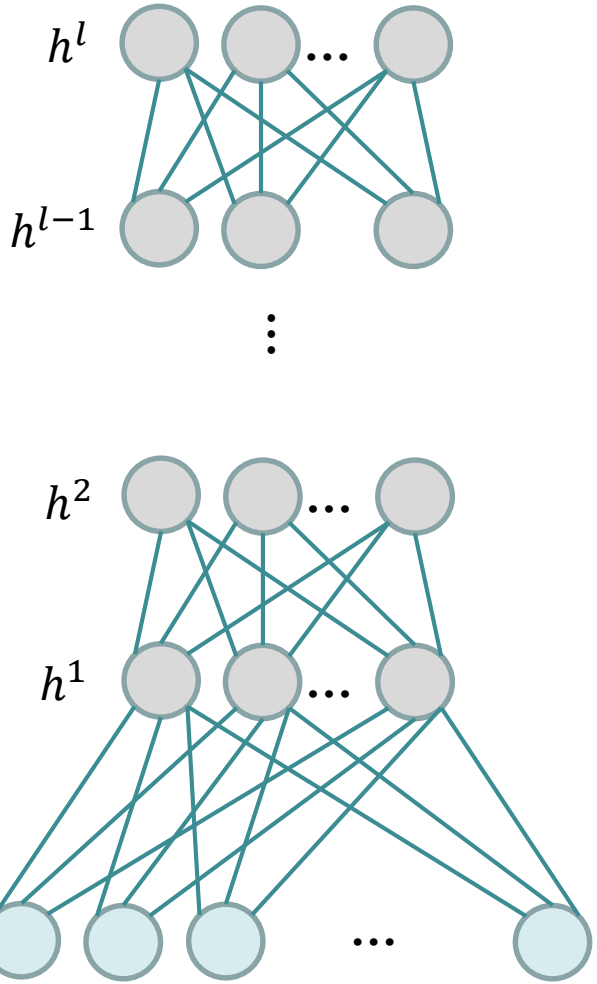
Глубокая сеть доверия

- В 2006 году Дж. Хинтон предложил подход, позволяющий быстро проинициализировать параметры модели
- Данный подход предполагает ***«жадное» послойное обучение сети***
- Обучение выполняется в предположении, что каждый слой нейронов не зависит от следующего
- После того, как стек ограниченных машин Больцмана обучен, систему можно рассматривать как единую вероятностную модель, называемую ***глубокой сетью доверия*** (Deep Belief Network, DBN)

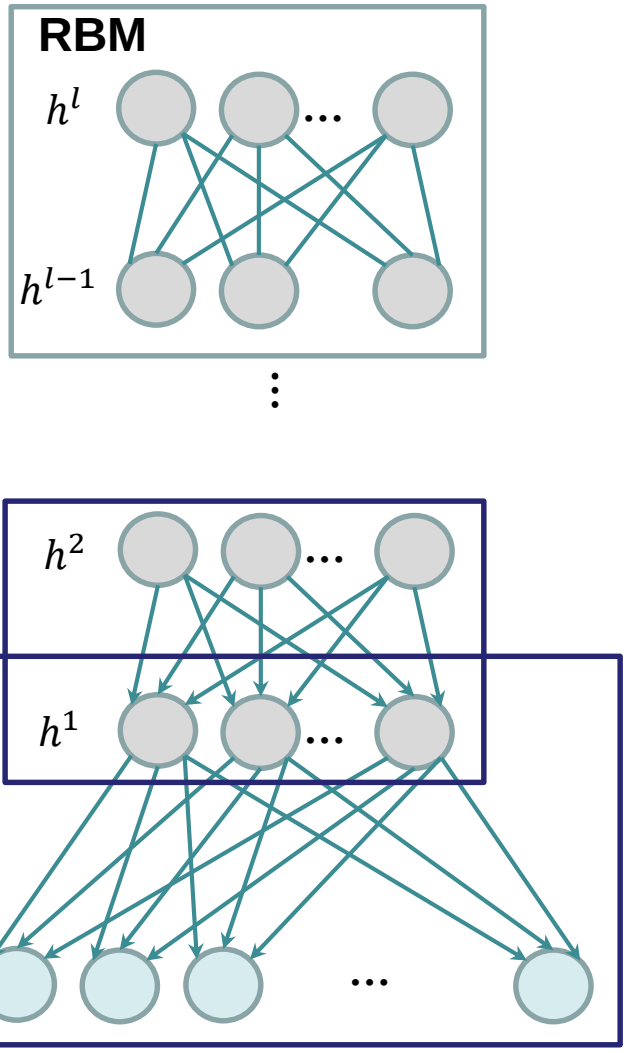


Отличие глубокой машины Больцмана и глубокой сети доверия

Глубокая машина Больцмана



Глубокая сеть доверия



Сигмоидальная сеть доверия

- ***Сигмоидальная сеть доверия***, содержащая l слоев, моделирует функцию совместного распределения $p\left(x, h^1, h^2, \dots, h^l; \theta\right)$
- Функция совместного распределения вероятности может быть выражена в виде произведения условных вероятностей:

$$p\left(x, h^1, \dots, h^l; \theta\right) = p(x|h^1; \theta)\left(\prod_{k=1}^{l-2} p\left(h^k \middle| h^{k+1}; \theta\right)\right) p\left(h^{l-1}, h^l; \theta\right)$$

Функции распределения условной вероятности для сигмоидальной сети доверия

- Поскольку нейроны на слое независимы, и функция активации на каждом слое сигмоидальная, то функции распределения условной вероятности имеют вид:

$$p(h^k) = \prod_i p(h_i^k),$$

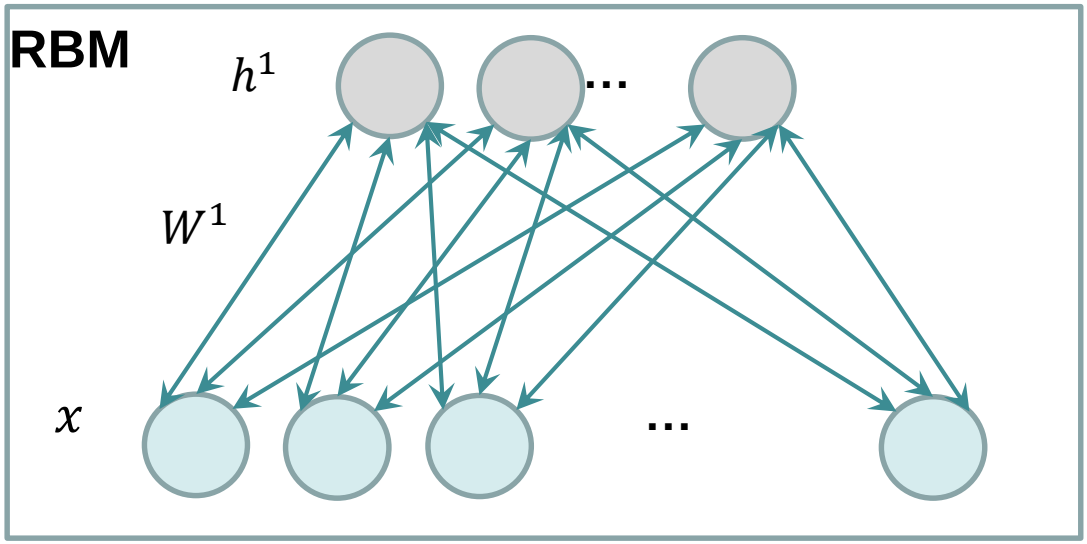
$$q(h^k|h^{k-1};\theta) = p(h^k|h^{k-1};\theta) = \prod_j p(h_j^k|h^{k-1};\theta)$$

$$= \prod_j sigm(W_{j.}^k h^{k-1}),$$

$$p(h^{k-1}|h^k;\theta) = \prod_j p(h_j^{k-1}|h^k;\theta) = \prod_j sigm\left(W_{.j}^{k^T} h^k\right)$$

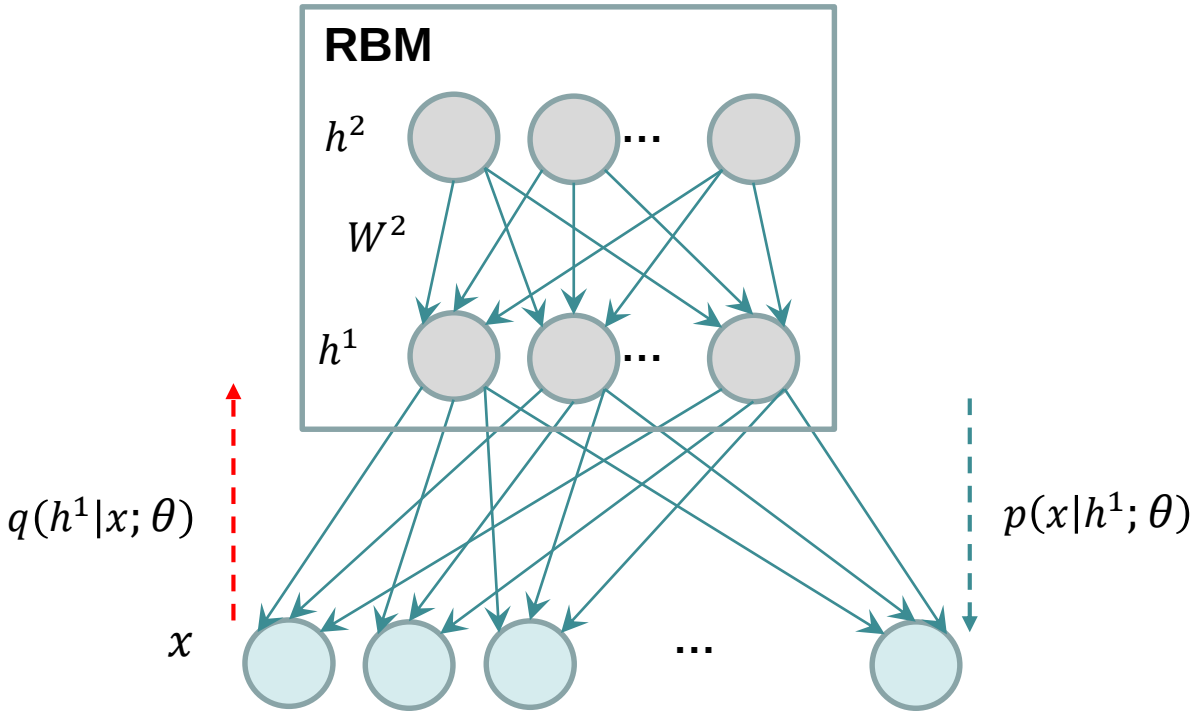
«Жадное» послойное обучение сигмоидальной сети доверия (1)

1. Построение ограниченной машины Больцмана из слоя видимых нейронов x и первого слоя скрытых нейронов h^1 и обучение весов первого слоя W^1



«Жадное» послойное обучение сигмоидальной сети доверия (2)

2. Построение ограниченной машины Больцмана из скрытых слоев h^1 и h^2 и обучение весов второго слоя W^2. При этом весовая матрица W^1 фиксируется, а значения вектора h^1 сэмплируются из распределения $q(h^1|x; W^1)$, полученного на предыдущем шаге



«Жадное» послойное обучение сигмоидальной сети доверия (3)

3. Формирование ограниченных машин Больцмана из следующих последовательно идущих пар слоев h^k и h^{k+1} и обучение весов соответствующих весов W^{k+1}. При этом весовая матрица W^k фиксируется, а h^k сэмплируется из распределения $q\left(h^k \middle| h^{k-1}; \theta\right)$

Примечание: по завершении «жадного» послойного обучения глубокой сети доверия можно дополнительно выполнить тонкую настройку параметров, например, с помощью алгоритма «wake-sleep»*

* Hinton G.E., Dayan P., Frey B. J., Neal R.M. The wake-sleep algorithm for unsupervised neural networks // Science. – 1995. – Vol. 268, pp. 1558–1161.

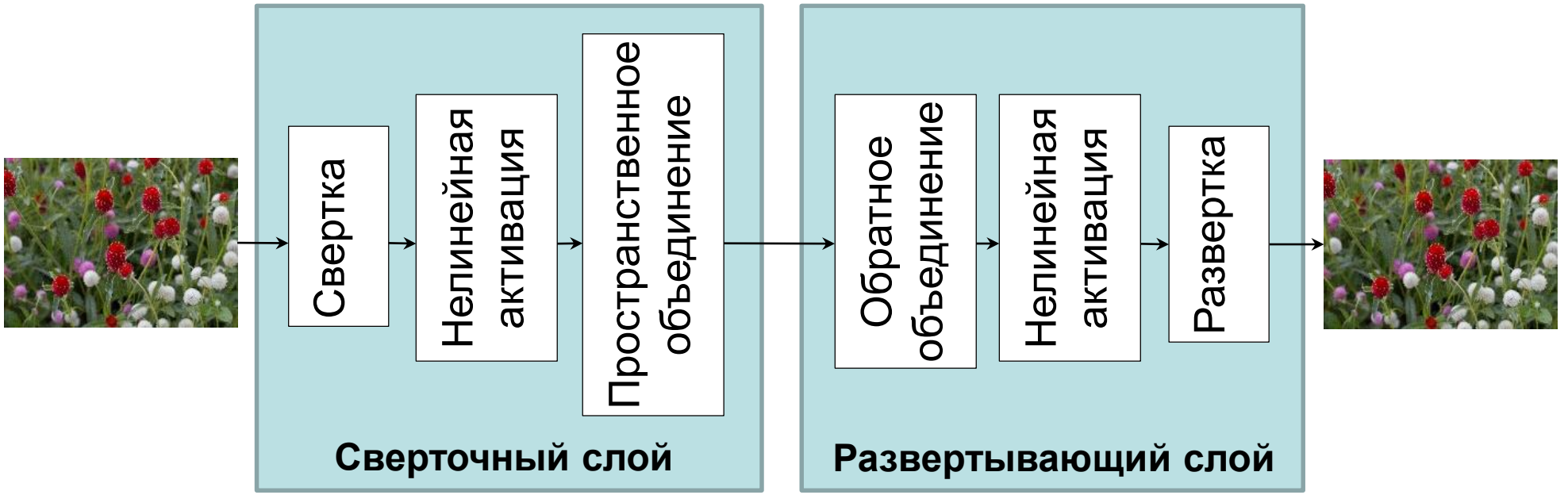
РАЗВЕРТЫВАЮЩИЕ НЕЙРОННЫЕ СЕТИ

Развертывающие нейронные сети (1)

- ***Развертывающие нейронные сети*** (Deconvolutional Neural Networks) изначально предложены как сверточная разновидность разреженных автокодировщиков
- Используются для визуализации карт признаков сверточных сетей
- Позднее идея развертывающих сетей получила широкое применение при решении задачи семантической сегментации
- В простейшем случае такого типа сети состоят из двух блоков:
  - Сверточный слой – последовательное применение преобразований свертки, функции активации и пространственного объединения
  - Развертывающий слой – выполнение обратных преобразований сверточного слоя обратного объединения (unpooling), активации и развертывания (deconvolution)

56

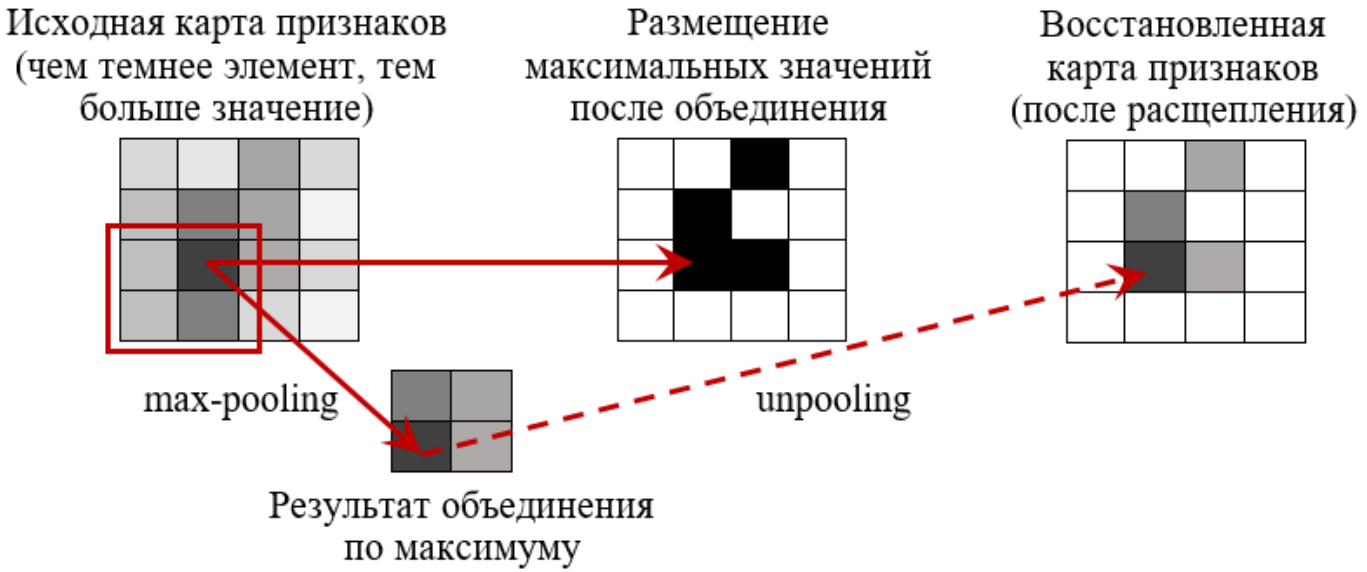
Развертывающие нейронные сети (2)



- В качестве функции активации использована функция отсечения положительной части ReLU
- В качестве операции пространственного объединения – функция объединения по максимальному значению

Реализация обратных преобразований (1)

- ***Расщепление*** (unpooling) – обратное к объединению
  - Обращение реализуется посредством сохранения индексов элемента карты признаков, в котором было достигнуто максимальное значение
  - При развертывании значение максимума размещается в соответствующей ячейке, а окрестные значения обнуляются



Реализация обратных преобразований (2)

- ***Обратная нелинейная активация***
  - Обратной функцией к ReLU является она же
  - Авторы метода аргументируют выбор такого обратного преобразования тем, что свертка применяется к положительной части при прямом проходе сети, значит при обратном проходе свертка на этапе развертывания также должна применяться к положительной части

Реализация обратных преобразований (3)

- ***Развертка***
  - Развертка предполагает применение тех же фильтров, что и при вычислении свертки
  - Отличие состоит лишь в том, что ядра фильтров транспонируются
  - На выходе формируется изображение, близкое к исходному

Обучение развертывающей сети (1)

- Задача обучения развертывающей сети состоит в том, чтобы перед выполнением операции развертки получить карту признаков, применение набора фильтров к которой позволяет получить выход, максимально близкий к входному изображению

- На входе сети имеется изображение $y = (y_1, y_2, \dots, y_s)$
- На выходе сформировано изображение $\tilde{y} = (\widetilde{y_1}, \widetilde{y_2}, \dots, \widetilde{y_s})$, где s – количество каналов изображения
- ***Цель обучения*** – восстановление карты признаков $z = (z_1, z_2, \dots, z_K)$, которая обеспечивает наилучшее приближение к входному изображению. При этом выходной сигнал сети вычисляется посредством применения сверток к карте признаков z

Обучение развертывающей сети (2)

- Выходной сигнал сети вычисляется посредством применения сверток к карте признаков z:

$$\widetilde{y_k} = \sum_i \langle z_i, f_{i,k} \rangle$$

- Функция ошибки представляет собой евклидову норму

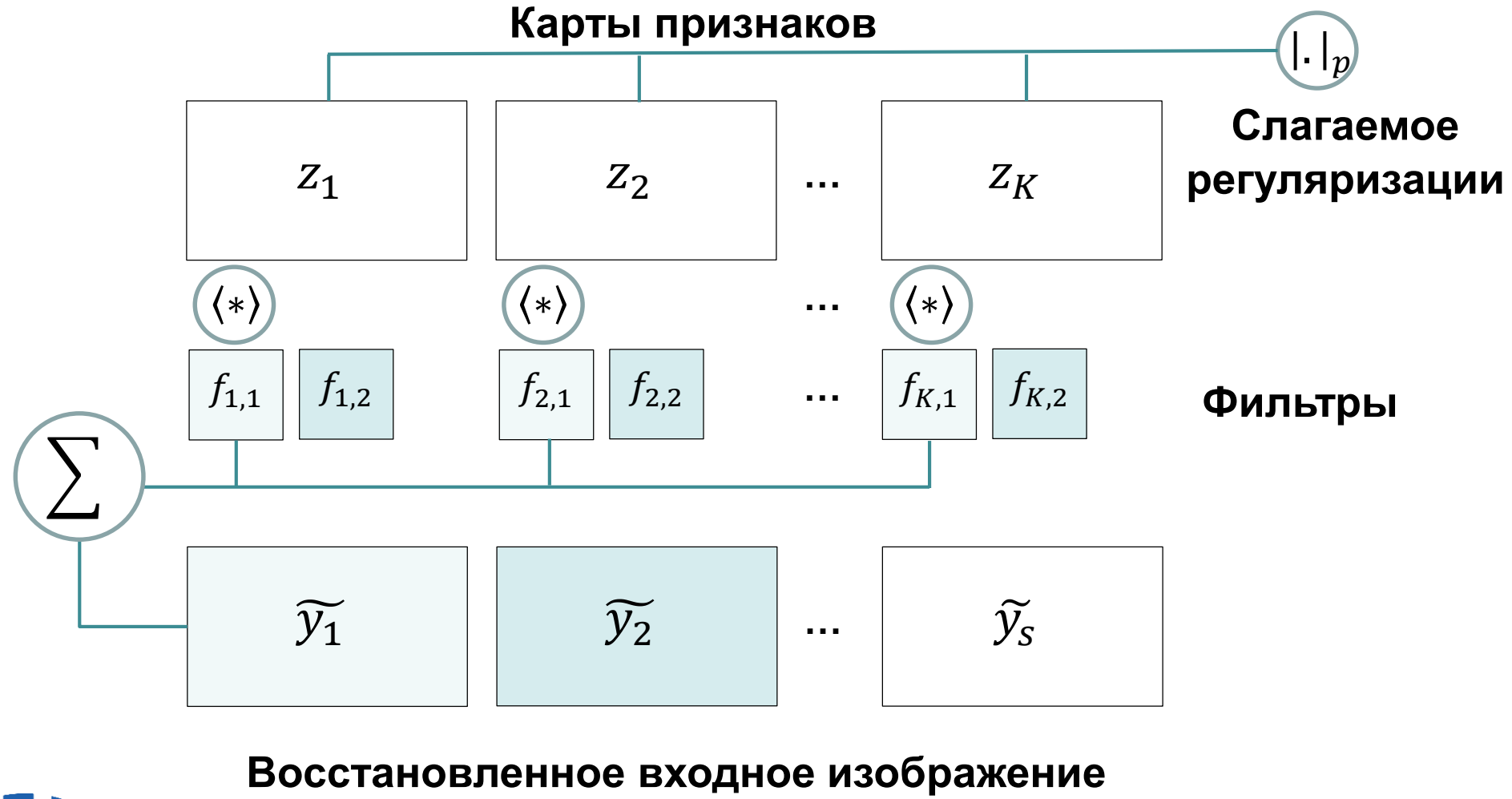
$$L(y, \tilde{y}) = \sum_k \left\| \sum_i \langle z_i, f_{i,k} \rangle - y_k \right\|_2^2 ,$$

- Минимизируемый функционал имеет следующий вид:

$$J(\theta) = \frac{\lambda}{2} \sum_{y \in I} \sum_k \left\| \sum_i \langle z_i, f_{i,k} \rangle - y_k \right\|_2^2 + \sum_{y \in I} \sum_i |z_i|^p \to \min_{z,\theta} ,$$

где $\theta = \{f_{i,k}\}$ – параметры системы, I – входные изображения

Обучение развертывающей сети (3)



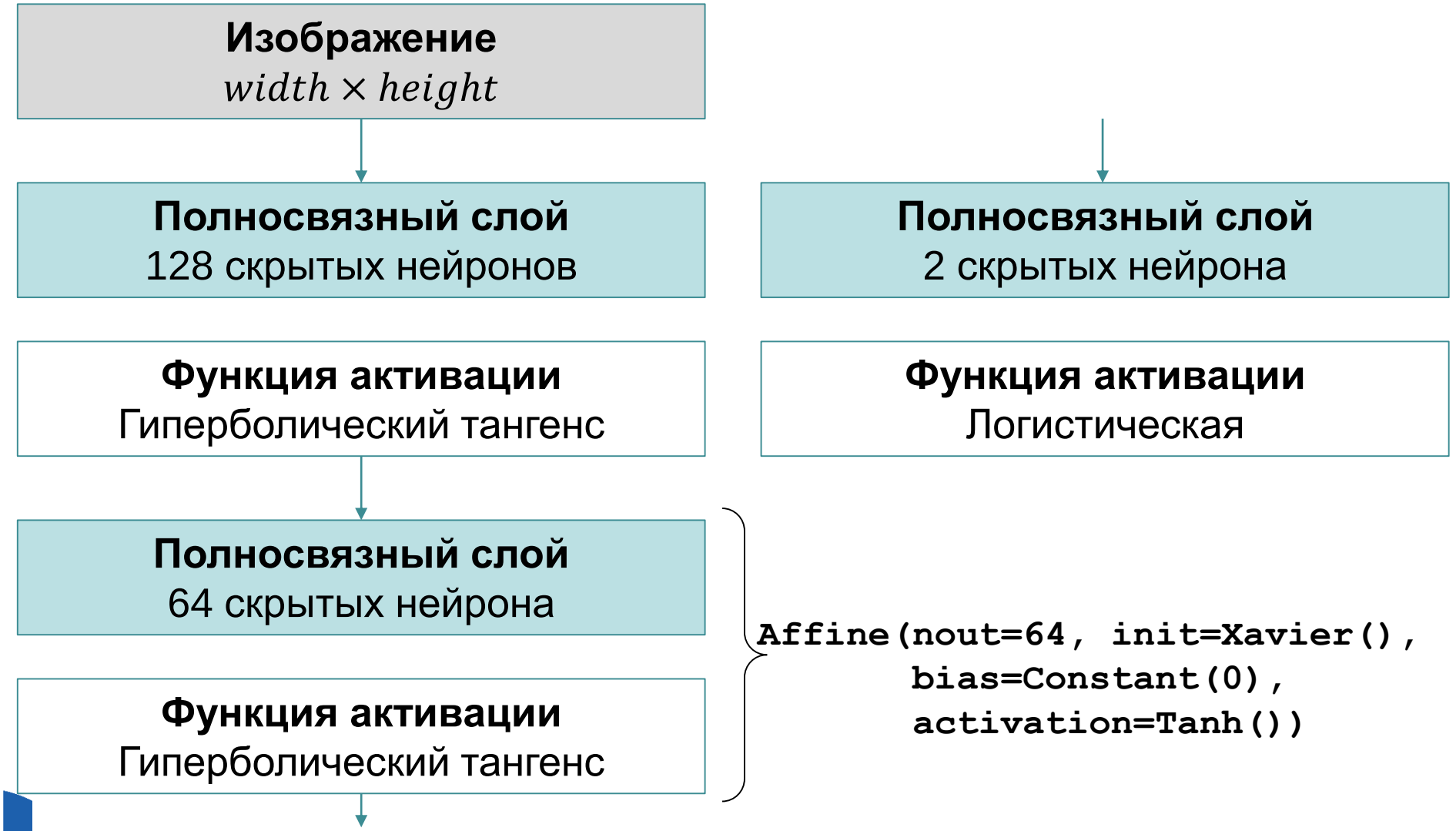
Обучение развертывающей сети (4)

- Приведенная задача оптимизации решается с использованием итерационного алгоритма ISTA (Iterative Shrinkage-Thresholding Algorithm)*

* Beck A., Teboulle M. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems [https://people.rennes.inria.fr/Cedric.Herzet/Cedric.Herzet/Sparse_Seminar/Entrees/2012/11/12_A_Fast_Iterative_Shrinkage-Thresholding_Algorithmfor_Linear_Inverse_Problems_(A._Beck,_M._Teboulle)_files/Breck_2009.pdf].

ПРИМЕР НАЧАЛЬНОЙ НАСТРОЙКИ ВЕСОВ

Архитектура полносвязной сети



Архитектура стека автокодировщиков для выбранной полносвязной сети

Пример начальной настройки весов с использованием Intel® neon™ Framework (1)

- ***Первый автокодировщик***
  - Входные данные кодировщика – изображения
  - Выходные данные декодировщика – изображения
  - Разметка <изображение, изображение>

Пример начальной настройки весов с использованием Intel® neon™ Framework (2)

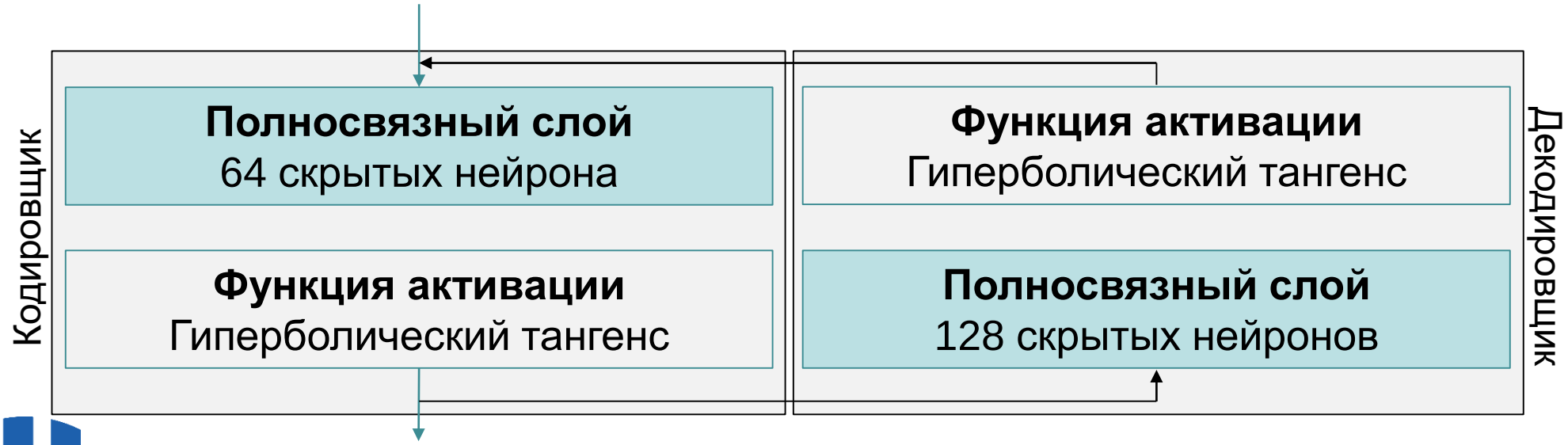
```
# первый автокодировщик
def generate_mlp2b2_128_ae_stacked_step1_model(input_shape):
    output_size=reduce(lambda p, x:p*int(x), input_shape, 1)

    layers = [
        DataTransform(transform=Normalizer(divisor=128.0)),
        Affine(nout=128, init=Gaussian(scale=0.1),
          bias=Constant(0), activation=Tanh(), name='fc_1'),
        Affine(nout=output_size, init=Gaussian(scale=0.1),
          bias=Constant(0), activation=Tanh(), name='fc_-1')
    ]
    model = Model(layers=layers)
    cost = GeneralizedCost(costfunc=SumSquared())
    return (model, cost)
```

Пример начальной настройки весов с использованием Intel® neon™ Framework (3)

- ***Второй автокодировщик***
  - Как построить?

Пример начальной настройки весов с использованием Intel® neon™ Framework (4)

- ***Второй автокодировщик***
  - Входные данные – выход кодирующего блока предыдущего автокодировщика
  - При реализации сохраним кодирующий блок предыдущего кодировщика с предобученными весами
  - Разметка <изображение, выход предыдущего кодировщика>



Пример начальной настройки весов с использованием Intel® neon™ Framework (5)

```
# второй автокодировщик
def generate_mlp2b2_128_ae_stacked_step2_model(input_shape):
    layers = [
        DataTransform(transform=Normalizer(divisor=128.0)),
        Affine(nout=128, init=Xavier(), bias=Constant(0),
            activation=Tanh(), name='fc_1'),
        Affine(nout=64, init=Xavier(), bias=Constant(0),
            activation=Tanh(), name='fc_2'),
        Affine(nout=128, init=Xavier(),
            bias=Constant(0),
            activation=Tanh(), name='fc_-2') ]
    model = Model(layers=layers)
    cost = GeneralizedCost(costfunc=SumSquared())
    return (model, cost)
```

Пример начальной настройки весов с использованием Intel® neon™ Framework (6)

```
# структура сети
def generate_mlp2b2_128_ae_stacked_model():
    layers = [
        DataTransform(transform=Normalizer(divisor=128.0)),
        Affine(nout=128, init=Xavier(), bias=Constant(0),
            activation=Tanh(), name='fc_1'),
        Affine(nout=64, init=Xavier(), bias=Constant(0),
            activation=Tanh(), name='fc_2'),
        Affine(nout=2, init=Xavier(), bias=Constant(0),
            activation=Logistic(shortcut=True), name='cls')
    ]
    model = Model(layers=layers)
    cost = GeneralizedCost(costfunc=CrossEntropyBinary())
    return (model, cost)
```

Пример начальной настройки весов с использованием Intel® neon™ Framework (7)

- Обучение стека автокодировщиков:
  - Подготовка данных обучения для первого автокодировщика (разметка содержит пары <изображение, изображение>)
  - Обучение первого автокодировщика
  - Удаление из модели обученного автокодировщика декодирующего блока
  - Подготовка данных обучения для второго автокодировщика (разметка содержит пары <изображение, выход предыдущего кодировщика>)
  - Обучение второго автокодировщика
  - Удаление из модели обученного автокодировщика декодирующего блока
  - Обучение сети с предварительно обученными весами

Пример начальной настройки весов с использованием Intel® neon™ Framework (8)

- Исходный код функций, которые реализуют приведенную последовательность этапов, разбирается в соответствующей практической работе (**`Practice3_ae/main_train_stacked_autoencoder.py`**)

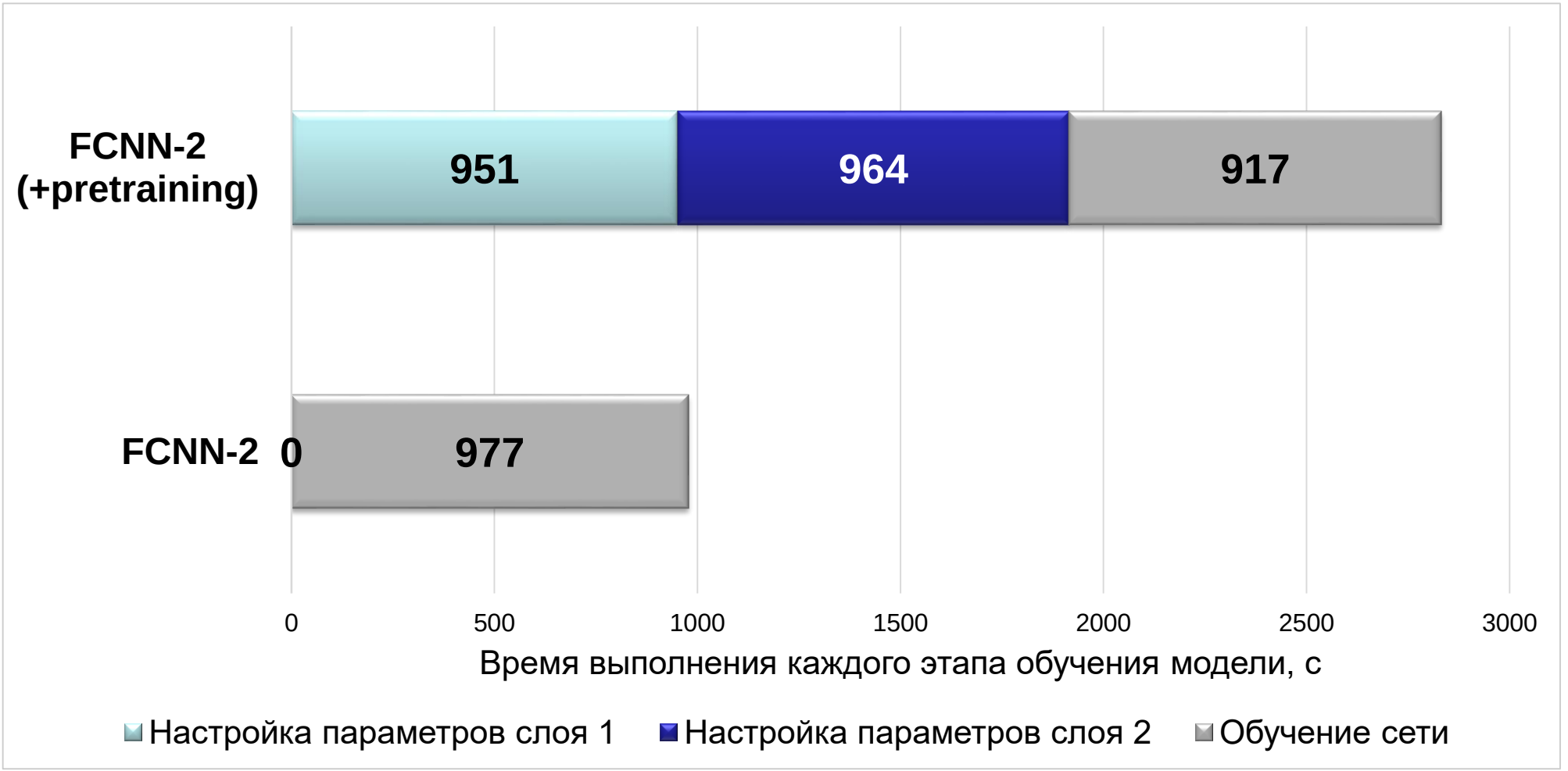
Тестовая инфраструктура

- CPU: Intel® Xeon® CPU E5-2660 0 @ 2.20GHz
- GPU: Tesla K40s 11Gb
- OS: Ubuntu 16.04.4 LTS
- Инструменты:
  - Intel® neon™ Framework 2.6.0
  - CUDA 8.0
  - Python 3.5.2
  - Intel® Math Kernel Library 2017 (Intel® MKL)

Сводные результаты экспериментов

Название	Точность, %	Время обучения, с
FCNN-1	71.2	932
FCNN-2	73.5	977
FCNN-3	**77.7**	1013
CNN-1	79.3	1582
CNN-2	**83.5**	2030
ResNet-18 (90 эпох)	**81.3**	15127
ResNet-50 (30 эпох)	80.9	11849
FCNN-2 (+pretraining, 30 эпох)	**78.9**	2832

Распределение времени между предварительной настройкой параметров и обучением сети



Заключение

- Предварительное обучение параметров сетей требует значительного времени, сопоставимого с временем обучения самой сети
- Разработанные методы случайной инициализации параметров сетей позволяют получить неплохое начальное приближение для оптимизации целевой функции
- В настоящее время методы предварительного обучения параметров достаточно редко используются при решении практических задач, поскольку разработаны эффективные случайные генераторы

Основная литература

- Хайкин С. Нейронные сети. Полный курс. – М.: Издательский дом «Вильямс». – 2006. – 1104 с.
- Осовский С. Нейронные сети для обработки информации. – М.: Финансы и статистика. – 2002. – 344 с.
- Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org].

Авторский коллектив

- **Кустикова Валентина Дмитриевна**
  к.т.н., ст.преп. каф. МОСТ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  valentina.kustikova@itmm.unn.ru
- **Жильцов Максим Сергеевич**
  магистрант каф. МОСТ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  zhiltsov.max35@gmail.com
- **Золотых Николай Юрьевич**
  д.ф.-м.н., проф. каф. АГДМ ИИТММ,
  ННГУ им. Н.И. Лобачевского
  nikolai.zolotykh@itmm.unn.ru