

# Машинное обучение (Machine Learning)

Глубокое обучение: автокодеры (Deep Learning:  
autoencoders)

Уткин Л.В.

Санкт-Петербургский политехнический университет Петра Великого



# Содержание

- 1 Автокодер
- 2 Стек автокодеров
- 3 Модификации автокодеров
- 4 Автокодер и word embedding

# Автокодер

# Автокодер - определение (Autoencoder)

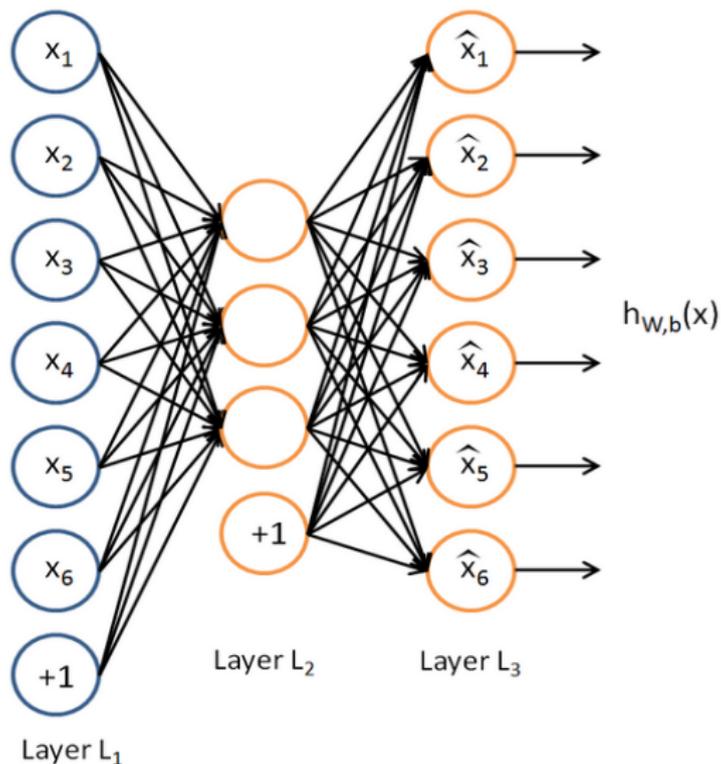
Из Wikipedia:

*An autoencoder is an artificial neural network and its aim is to learn a compressed representation for a set of data. This means it is being used for dimensionality reduction.*

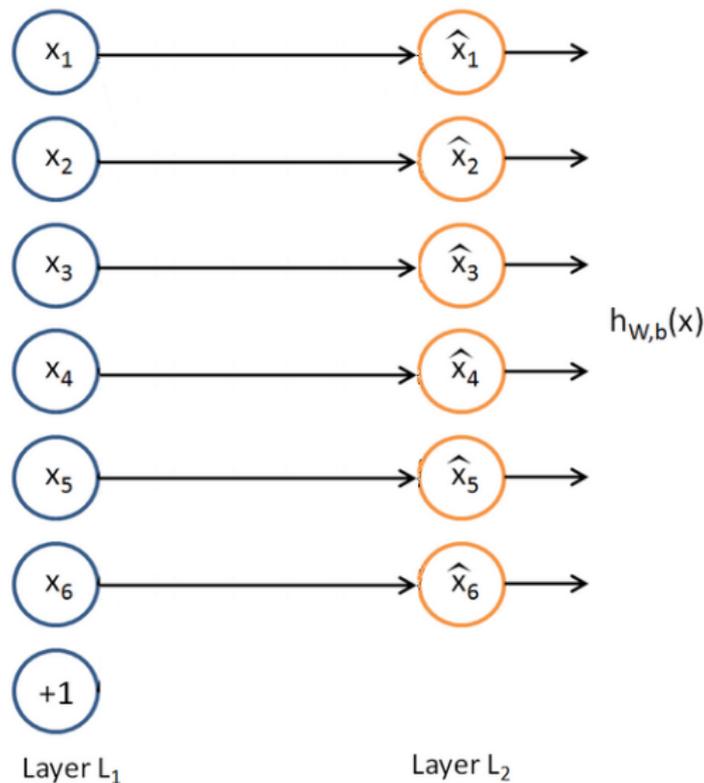
# Автокодер

- Обучение без учителя
- Выборка  $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots)$ , где  $\mathbf{x}^{(k)} = (x_1^{(k)}, \dots, x_m^{(k)})$ ,  $m$  признаков.
- **Автокодер** - это нейронная сеть, которая использует алгоритм обратного распространения так, что в результате обучения получаем выход идентичный входу, т.е.  $\mathbf{y}^{(i)} = \mathbf{x}^{(i)}$ .
- Другими словами: автокодер пытается обучиться аппроксимации тождественной функции.

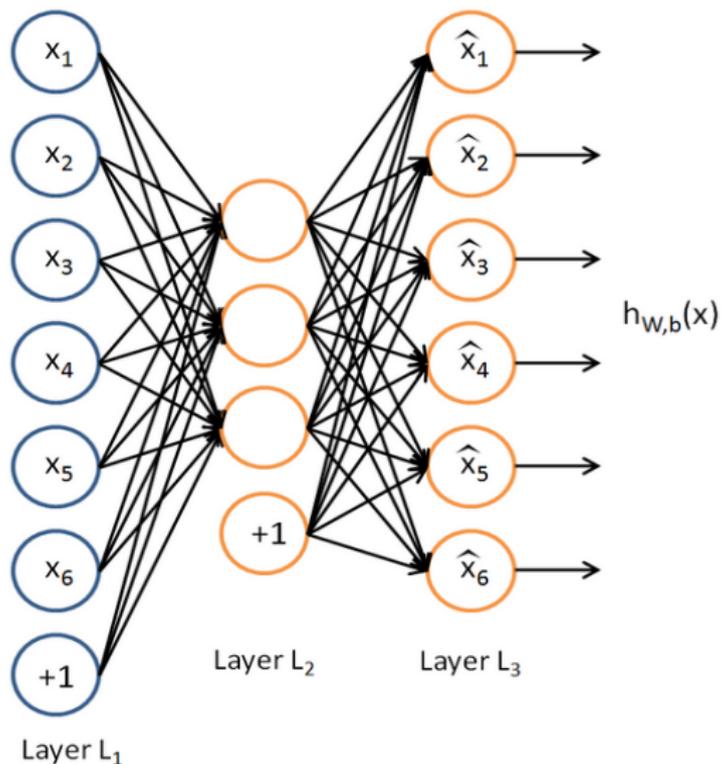
# Автокодер (иллюстрация)



# Почему бы не сделать проще?



# Все дело в скрытом слое L2



# Все дело в скрытом слое L2

- Получаем сжатое представление примеров обучающей выборки
- это возможно, если
  - имеет место *корреляция* части признаков
  - заставить нейроны скрытого слоя быть “разреженными” при условии  $s_2 > t$  (число нейронов скрытого слоя больше, чем входного слоя)
- понижение размерности аналогично методу главных компонент

# Сжатие автокодером

$$\mathbf{z}^{(k)} = f(W_1 \mathbf{x}^{(k)} + b_1)$$

$$\hat{\mathbf{x}}^{(k)} = f(W_2 \mathbf{z}^{(k)} + b_2)$$

Целевой функционал (ошибка реконструирования):

$$\begin{aligned} R(W_1, b_1, W_2, b_2) &= \sum_{k=1}^n (\hat{\mathbf{x}}^{(k)} - \mathbf{x}^{(k)})^2 \\ &= \sum_{k=1}^n (f(W_2 \cdot f(W_1 \mathbf{x}^{(k)} + b_1) + b_2) - \mathbf{x}^{(k)})^2 \rightarrow \min \end{aligned}$$

# Разреженность нейронов скрытого слоя

*Большинство нейронов должны быть почти неактивные, т.е. их выход близок 0.*

- Как это сделать?
- Каким-то образом наложить ограничения на их уровень активации  $a_j^{(2)}$  в процессе обучения.
- $a_j^{(2)}(\mathbf{x})$  - уровень активации (выход)  $j$ -го нейрона скрытого слоя  $L_2$  в зависимости от обучающего вектора  $\mathbf{x}$ .

# Разреженность нейронов скрытого слоя

- Средний уровень активации  $j$ -го нейрона по всей обучающей выборке:

$$\hat{\rho}_j = \frac{1}{n} \sum_{i=1}^n a_j^{(2)}(\mathbf{x}^{(i)})$$

- Параметр разреженности:  $\rho$  (малое число близкое к 0, например 0.05)
- Цель:  $\hat{\rho}_j = \rho$  (хотя бы примерно)

# Разреженность нейронов скрытого слоя

- Штрафное слагаемое - расстояние (дивергенция) Кульбака — Лейблера (KL) - мера удаленности друг от друга двух вероятностных распределений

$$\sum_{j=1}^{s_2} KL(\rho, \hat{\rho}_j) = \sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}$$

- Штрафное слагаемое равно 0, если  $\hat{\rho}_j = \rho$ .
- Общий функционал риска

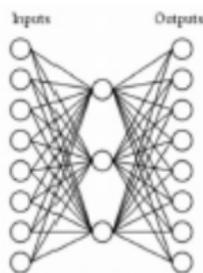
$$R_{\text{разреж}}(W, b) = R(W, b) + \beta \sum_{j=1}^{s_2} KL(\rho, \hat{\rho}_j)$$

# Почти глубокое обучение

- Исходные данные: обучающая выборка  
 $\mathbf{X} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)})$
- Результат обучения автокодера (без учителя):
  - веса всех соединений между первым и вторым слоем  
 $W^{(1)} = (w_{11}^{(1)}, w_{12}^{(1)}, \dots, w_{m,s_2}^{(1)}, w_{01}^{(1)}, \dots, w_{0,s_2}^{(1)})$
  - значения активации нейронов скрытого слоя  
 $a_1^{(2)}(\mathbf{x}^{(i)}), \dots, a_{s_2}^{(2)}(\mathbf{x}^{(i)}), i = 1, \dots, n$
- Теперь можно заменить исходную выборку  
 $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)})$  новой выборкой  $a_1^{(2,i)}, \dots, a_{s_2}^{(2,i)}$
- А если точнее, то

$$((\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})) \rightarrow ((a^{(2,1)}, y^{(1)}), \dots, (a^{(2,n)}, y^{(n)}))$$

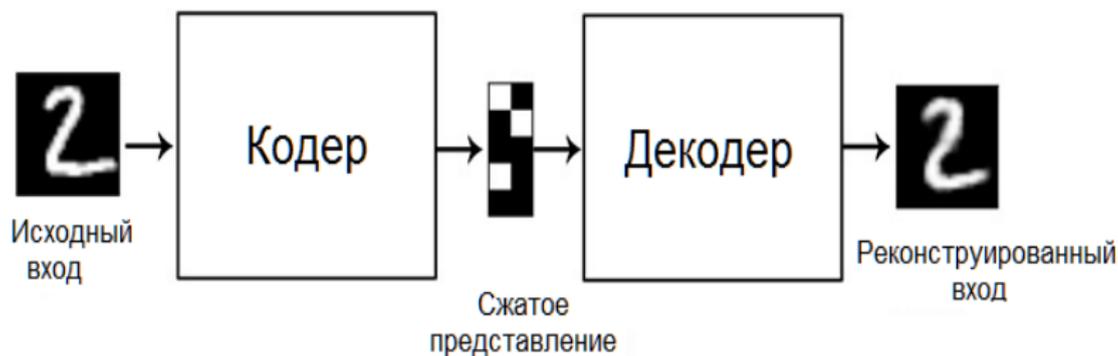
# Представление данных в скрытом слое



Input	Hidden Values	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .01 .11 .88	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .22 .99 .99	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001

© Eric Xing @ CMU, 2015

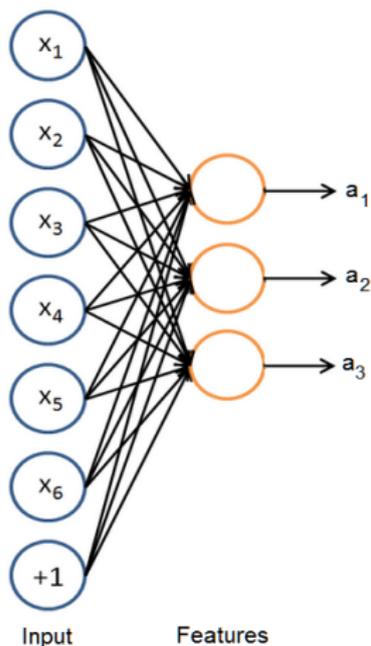
# Еще одна иллюстрация автокодера



<https://blog.keras.io/building-autoencoders-in-keras.html>

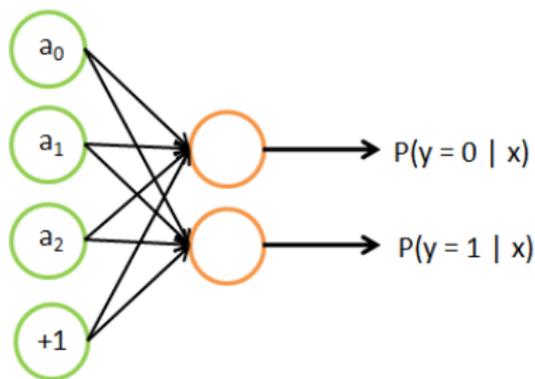
# Как посмотреть новые признаки?

Есть уже обученная сеть



# Задача классификации

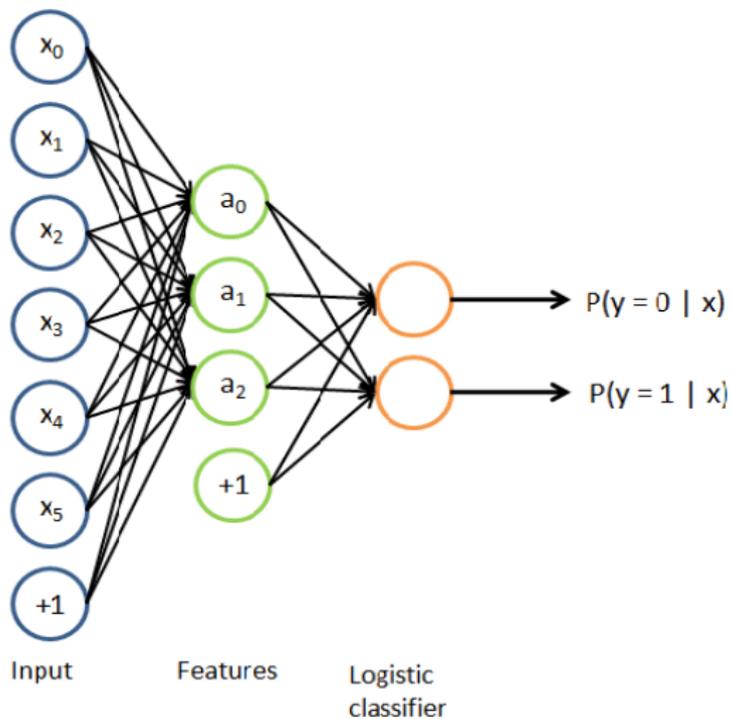
Если необходимо решить задачу классификации, то используем сеть



Input  
(features)

Logistic  
classifier

# Общий случай



# Обучение (почти глубокое) в два этапа

- 1 Первый слой весов  $W^{(1)}$ , отображающий входные данные  $\mathbf{x}$  в значения активации нейронов скрытого слоя  $a_1^{(2,i)}, \dots, a_{s_2}^{(2,i)}$ , обучается как часть обучения автокодера.
- 2 Второй слой весов  $W^{(2)}$ , отображающий значения активации нейронов скрытого слоя  $a_1^{(2,i)}, \dots, a_{s_2}^{(2,i)}$  в выход  $y$ , обучается, используя логистическую регрессию, SVM, SoftMax регрессию и т.д.

# Обучение (третий этап)

Можно использовать всю нейронную сеть для дальнейшей модификации всех параметров модели, чтобы попытаться уменьшить ошибку обучения. В частности “тонкая настройка” (fine-tune) параметров при помощи градиентного спуска может быть использована.

# Стек автокодеров

# Глубокие нейронные сети

- Была рассмотрена нейронная сеть с тремя слоями: входной, скрытый и выходной
- **Глубокая сеть**: несколько скрытых слоев для преобразования более сложных признаков на входе
- Проблема: **как обучить такую сеть?**
- Вариант решения: **жадный** алгоритм послойного обучения

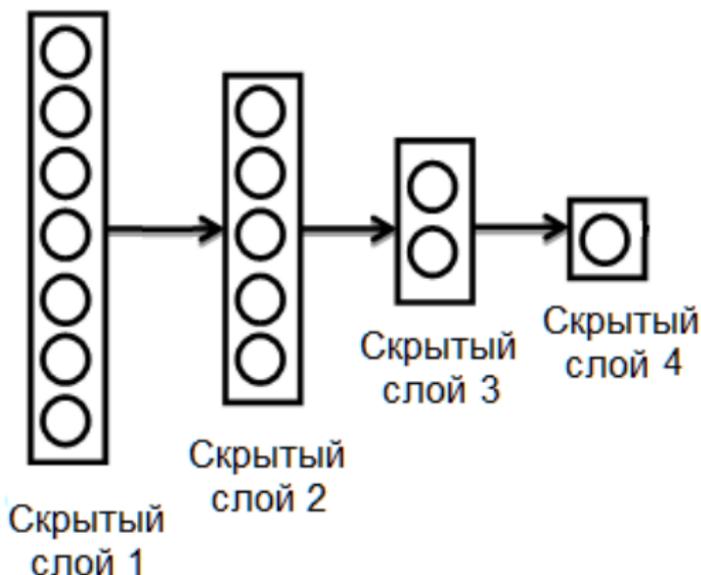
# Жадный алгоритм послойного обучения (коротко)

Основная идея: обучать последовательно слои так, что

- 1 первый скрытый слой обучается первым;
- 2 после этого обучаем второй слой и т.д.;
- 3 на каждом этапе используется “старая” сеть с  $k - 1$  скрытыми слоями и к ним добавляется  $k$ -ый слой, вход которого - выход уже обученных  $k - 1$  слоев.

# Стек автокодеров

**Стек автокодеров** - нейронная сеть, состоящая из нескольких слоев автокодеров, в которой выход каждого слоя связан со входами следующего за ним слоя



# Стек автокодеров

- Пусть  $W^{(k,1)}, W^{(k,2)}, b^{(k,1)}, b^{(k,2)}$  - параметры  $W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}$  для  $k$ -го автокодера
- Кодирование:

$$a^{(l)} = f(z^{(l)}), \quad z^{(l+1)} = W^{(l,1)}a^{(l)} + b^{(l,1)}$$

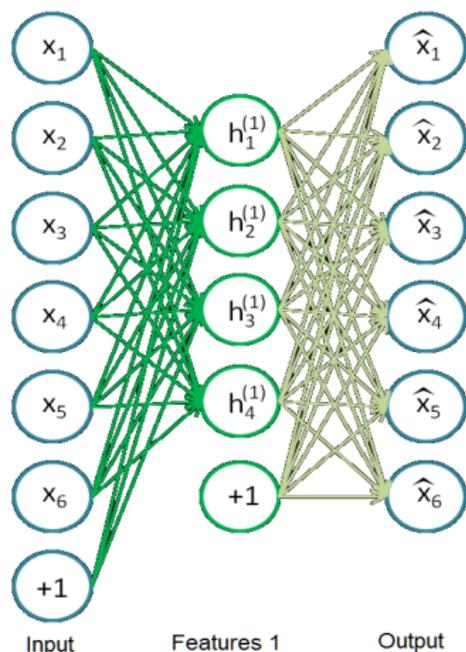
- Декодирование:

$$a^{(t+l)} = f(z^{(t+l)}), \quad z^{(t+l+1)} = W^{(t-l,2)}a^{(t+l)} + b^{(t-l,2)}$$

- Основная информация - в  $a^{(t)}$  - значения активации на самом глубоком слое

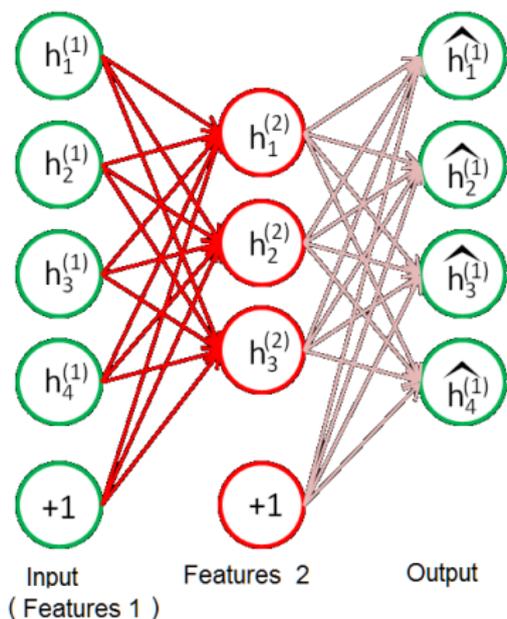
# Пример стека автокодеров (1)

Сначала обучим автокодер на входных данных  $\mathbf{x}^{(k)}$  для первичных признаков  $h^{(1)(k)}$ :



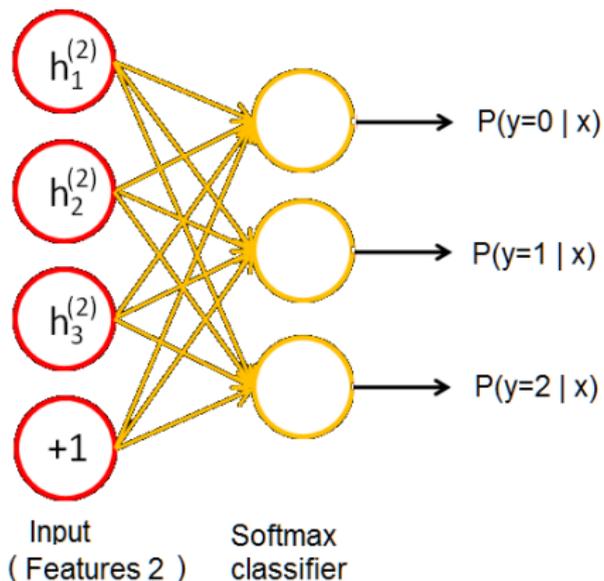
# Пример стека автокодеров (2)

Затем используем значения активации  $h^{(1)(k)}$  для каждого  $x^{(k)}$  для обучения другого автокодера и получения вторичных признаков  $h^{(2)(k)}$ :



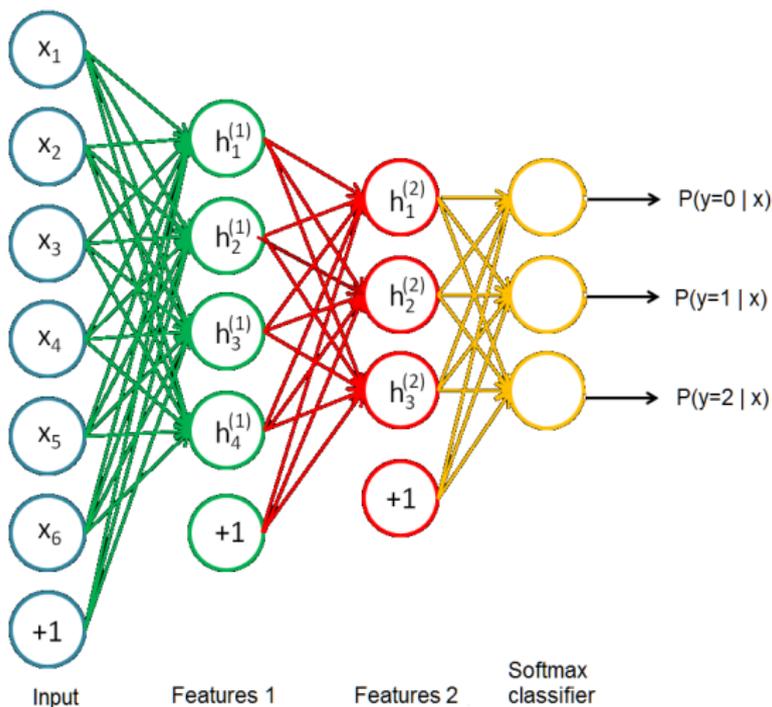
# Пример стека автокодеров (3)

Вторичные признаки совместно с  $y^{(k)}$  теперь могут использоваться для классификации:



# Пример стека автокодеров (4)

В итоге комбинируем все три слоя вместе:



# Стек автокодеров (еще раз жадный алгоритм обучения)

- 1 Обучаем первый автокодер ( $l = 1$ ) со всеми  $\mathbf{x}^{(k)}$ , используя обратное распространение ошибки.
- 2 Обучаем второй автокодер ( $l = 2$ ). Так как входной слой для  $l = 2$  - первый скрытый слой, то выходной слой для  $l = 1$  удаляется из сети. Обучение начинается с фиксации входной выборки слоя  $l = 1$ , которая является выходом слоя  $l = 2$ . Веса  $l = 2$  модифицируются, используя обратное распространение.
- 3 Процедура повторяется для всех слоев.

# Проблема использования автокодера

- Нет четкой связи между входом и выходом всей сети, например, при распознавании MNIST, нет возможности отобразить нейроны последнего скрытого слоя автокодера в цифры изображений.
- В этом случае, решение - добавить один или два слоя к последнему (глубокому) слою. Тогда вся нейронная сеть может рассматриваться как обычный персептрон и обучаться при помощи обратного распространения (fine-tuning).

# Еще одна проблема использования автокодера

- Выходной слой:

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}, \quad a^{(3)} = f(z^{(3)})$$

- $a^{(3)}$  - приближение  $x = a^{(1)}$ .
- Если  $f(z^{(3)})$  - функция активации - сигмоид, то необходимо нормализовать  $x$  к  $[0; 1]$ , так как выход сигмоида - число  $[0; 1]$ .

# Линейный декодер

- Функция активации нейронов скрытого слоя - сигмоид или  $\tanh$   $a^{(2)} = \sigma(W^{(1)}\mathbf{x} + b^{(1)})$ , где  $\sigma(\cdot)$  - сигмоид.
- Но функция активации выхода - линейная функция  $\hat{\mathbf{x}} = a^{(3)} = W^{(2)}a + b^{(2)}$  - **линейный декодер**.
- Можно теперь использовать любые  $\mathbf{x}$  без нормализации.

# Модификации автокодеров

# Denoising Autoencoder (Зашумленный автокодер)

- Вход автокодера зашумляется (различные стратегии):  
 $\mathbf{x}^{(k)} \rightarrow \tilde{\mathbf{x}}^{(k)}$ , например,  $\tilde{\mathbf{x}} = \mathbf{x} + \varepsilon$ ,  $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$
- Скрытый слой:  $\mathbf{z}^{(k)} = W_1 \tilde{\mathbf{x}}^{(k)} + b_1$
- Выходной слой без изменений:

$$\mathbf{x}^{(k)} \approx \hat{\mathbf{x}}^{(k)} = W_2 \mathbf{z}^{(k)} + b_2$$

- Ошибка реконструирования

$$R = \sum_{k=1}^n (\hat{\mathbf{x}}^{(k)} - \mathbf{x}^{(k)})^2, \text{ не } \sum_{k=1}^n (\hat{\mathbf{x}}^{(k)} - \tilde{\mathbf{x}}^{(k)})^2!$$

- Робастность, пропущенные данные

# k-Sparse Autoencoder (k-разреженный автокодер)

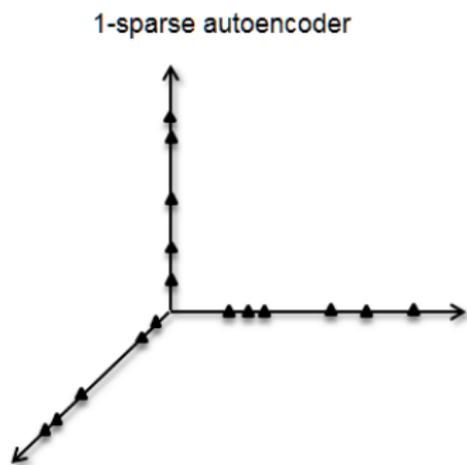
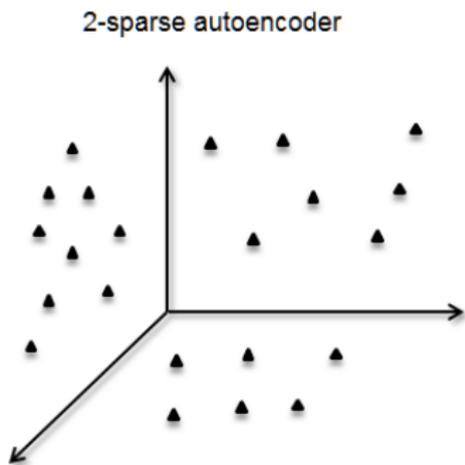
- Makhzani A, Frey B. k-Sparse Autoencoders. arXiv preprint arXiv:1312.5663. 2013 Dec 19
- Функция активации нейронов скрытого слоя является линейной, т.е.

$$\mathbf{z}^{(k)} = f(W_1 \mathbf{x}^{(k)} + b_1) = W_1 \mathbf{x}^{(k)} + b_1$$

- **Но** отбираются  $k$  наибольших скрытых нейронов, а остальные обнуляются
- Это вносит нелинейность

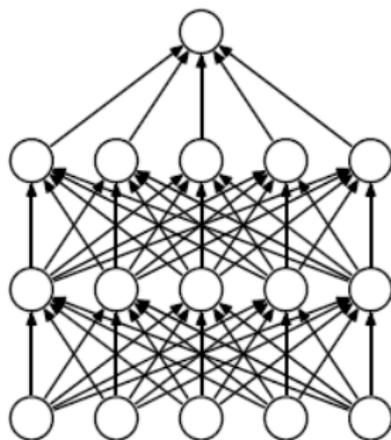
# k-Sparse Autoencoder - интересное свойство

3 нейрона в скрытом слое

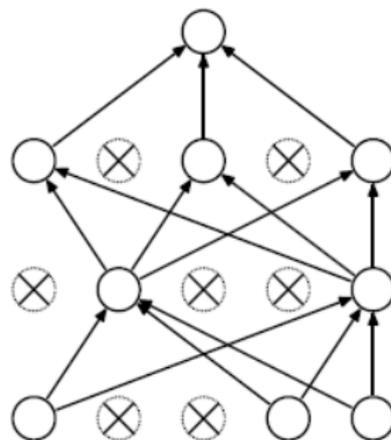


# Dropout Neural Networks

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15 (2014) 1929-1958.



Standard Neural Net



After applying dropout

# Dropout Neural Networks - прямое распространение

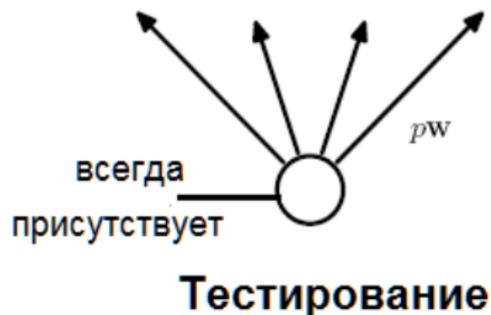
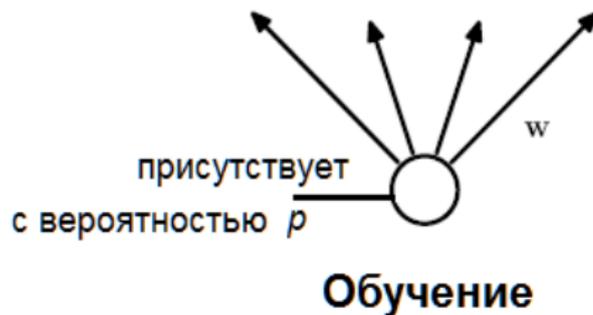
Стандартная сеть	Dropout
	$\mathbf{r}^{(k)} \sim \text{Bernulli}(p)$
	$\tilde{\mathbf{x}}^{(k)} = \mathbf{r}^{(k)} \cdot \mathbf{x}^{(k)}$
$\mathbf{z}^{(k)} = f(W_1 \mathbf{x}^{(k)} + b_1)$	$\mathbf{z}^{(k)} = f(W_1 \tilde{\mathbf{x}}^{(k)} + b_1)$
$\mathbf{y}^{(k)} = f(W_2 \mathbf{z}^{(k)} + b_2)$	$\mathbf{y}^{(k)} = f(W_2 \mathbf{z}^{(k)} + b_2)$

$\mathbf{r}^{(k)}$  - вектор, состоящий из 0 и 1, сгенерированных с распределением Бернулли так, что 1 имеет вероятность  $p$

# Dropout Neural Networks - обучение

- Обучение такое же, как и для стандартной сети
- **Отличия:**
  - прямое и обратное распространение для каждого обучающего примера осуществляется с “прореженной” сетью
  - градиенты для каждого параметра усредняются по всем обучающим примерам
  - если для текущего примера параметр “выкинут”, то градиент равен 0

# Dropout Neural Networks - тестирование



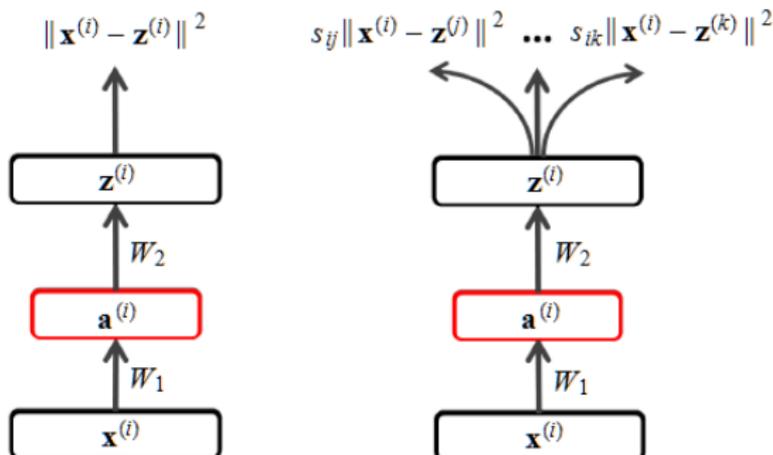
# Generalized autoencoder - Обобщенный автокодер (1)

- Выборка  $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots)$ , где  $\mathbf{x}^{(k)} = (x_1^{(k)}, \dots, x_m^{(k)})$ ,  $m$  признаков.
- $\mathbf{x}^{(i)}$  реконструируется  $k$  ближайших элементов обучающего множества  $\Omega_i = (\mathbf{x}^{(j)}, \dots, \mathbf{x}^{(k)})$
- Вклад каждого элемента  $s_{ij} \|\mathbf{x}^{(i)} - \mathbf{z}^{(j)}\|^2$  - весовое расстояние между  $\mathbf{x}^{(i)}$  и  $\mathbf{z}^{(j)}$
- $s_{ij}$  - показатель близости элементов  $\mathbf{x}^{(i)}$  и  $\mathbf{x}^{(j)}$
- Ошибка реконструкции:

$$R(W_1, b_1, W_2, b_2) = \sum_{i=1}^n \sum_{j \in \Omega_i} s_{ij} \|\mathbf{x}^{(i)} - \mathbf{z}^{(j)}\|^2 \rightarrow \min_{W_1, b_1, W_2, b_2}$$

# Обобщенный автокодер (2)

Wang W., Huang Y., Wang Y., Wang L. Generalized Autoencoder: A Neural Network Framework for Dimensionality Reduction // Proceedings of the CVPR 2014 Workshop, Columbus, Ohio, 2014, 490-497.



# Обобщенный автокодер - алгоритм обучения

- 1 Вычислить веса  $s_{ij}$  для всех  $i$  и  $j$
- 2 Определить множество индексов ближайших соседей  $\Omega_i$
- 3 Минимизировать  $R(W_1, b_1, W_2, b_2)$ , используя градиентный метод и модифицировать  $W_1, b_1, W_2, b_2$
- 4 Вычислить скрытое представление  $\mathbf{a}^{(i)}$  и модифицировать веса  $s_{ij}$  и множество  $\Omega_i$
- 5 Повторить шаги 3 и 4 до сходимости

# Обобщенный автокодер - частные случаи

- Стандартный автокодер  $\Omega_i = \{i\}$ :  $s_{ii} = 1$ ,  $s_{ij} = 0$ ,  $j \neq i$
- Реконструкция данных из того же класса  $\Omega_i$  - множество индексов класса, которому принадлежит  $\mathbf{x}^{(i)}$ :  $s_{ij} = 1/n_{c_i}$  (число элементов класса  $c_i$ )
- Реконструкция в  $k$  ближайших соседей:  
 $s_{ij} = \exp(-\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2 / t)$ ,  $t$  - настраиваемый параметр
- Реконструкция в  $k_1$  ближ. соседей класса  $c_i$  и  $k_2$  ближ. соседей класса  $c_j \neq c_i$ :  $s_{ij} = 1$ , если  $j \in \Omega_i^{(k_1)}$  и  $s_{ij} = -1$ , если  $j \in \Omega_j^{(k_2)}$

# Contractive autoencoder - Сжимающий автокодер

- Rifai S., Vincent P., Muller X., Glorot X., Bengio Y. Contractive Auto-Encoders: Explicit Invariance During Feature Extraction // Proceedings of the 28th International Conference on Machine Learning, 2011, pp. 833-840.
- Сжимающий автокодер получается добавлением специального штрафного слагаемого (регуляризация) к функции ошибки, которое “заставляет” промежуточное представление (в скрытом слое) становиться робастным к малым изменениям входных обучающих данных.

# Сжимающий автокодер

- Штрафное слагаемое - норма Фробениуса якобиана  $J_f(\mathbf{x})$ ,  $\mathbf{x} \in \mathbb{R}^m$ :

$$\|J_f(\mathbf{x})\|_F^2 = \sum_{i,j} \left( \frac{\partial h_j(\mathbf{x})}{\partial x_i} \right)^2$$

- Это частный случай  $p$ -нормы для  $p = 2$ :  
$$\|A\|_F^2 = \sum_{i=1}^n \sum_{j=1}^m a_{ij}^2$$
- Штрафное слагаемое “заставляет” отображение в пространство признаков быть сжатым (contractive) в окрестности обучающих данных

# Функции ошибки автокодеров (напоминание)

- Обычный автокодер:

$$J_{AE}(\theta) = \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))), \theta = \{W_1, b_1, W_2, b_2\},$$

например  $L(\mathbf{x}, g(f(\mathbf{x}))) = \|\mathbf{x} - \mathbf{z}\|^2$

- С регуляризацией:

$$J_{AE+wd}(\theta) = \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \left( \|W_1\|^2 + \|W_2\|^2 \right)$$

- Зашумленный:

$$J_{DAE}(\theta) = \sum_{\mathbf{x} \in S} \mathbb{E}_{\tilde{\mathbf{x}} \sim q(\tilde{\mathbf{x}}|\mathbf{x})} [L(\mathbf{x}, g(f(\tilde{\mathbf{x}})))], \quad \tilde{\mathbf{x}} = \mathbf{x} + \varepsilon$$

# Функции ошибки сжимающего автокодера

- Сжимающий автокодер:

$$J_{AE}(\theta) = \sum_{\mathbf{x} \in S} L(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \left( \|J_f(\mathbf{x})\|_F^2 \right)$$

- Вычисление функции ошибки: если  $f$  - сигмоид, то простое выражение

$$\|J_f(\mathbf{x})\|_F^2 = \sum_{i=1}^d (h_i(1 - h_i))^2 \sum_{j=1}^m \left( W_1^{(i,j)} \right)^2$$

# Отличие сжимающего автокодера от зашумленного

- Зашумленный автокодер делает робастными реконструированные значения, т.е.  $\mathbf{z} = g(f(\mathbf{x}))$
- Сжимающий автокодер делает робастным только скрытый слой  $\mathbf{h} = f(\mathbf{x})$ , т.е. представление признаков
- Это более важно, так как декодер  $g(\cdot)$  не нужен при классификации, а используется только кодер

# Еще отличия сжимающего автокодера от зашумленного

- Зашумленный автокодер более прост в реализации, так как он является простым расширением обычного автокодера и не требует вычисления якобиана скрытого слоя.
- Сжимающий автокодер имеет детерминированный градиент (так как нет случайного шума), что означает, что методы оптимизации второго порядка (сопряженный градиент) могут использоваться, и автокодер может быть более стабильным, чем зашумленный автокодер.

# Векторные представления и автокодер (embedding)

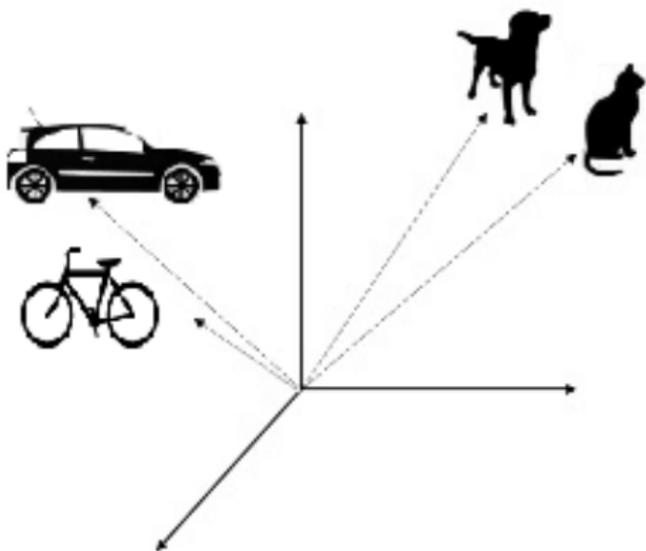
# Проблема представления (embedding)

- Проблема возникает, когда необходимо при сокращении размерности (скрытый слой автокодера) сохранить “семантику” исходного входного множества данных большой размерности.
- Исходные данные большой размерности “вкладываются” в малую размерность с сохранением “геометрии”.
- Многие алгоритмы, например, pointwise mutual information (PMI), обеспечивают локальное сохранение “геометрии”, т.е. относительные расстояния между точками в пространстве большой размерности сохраняются в пространстве малой размерности.

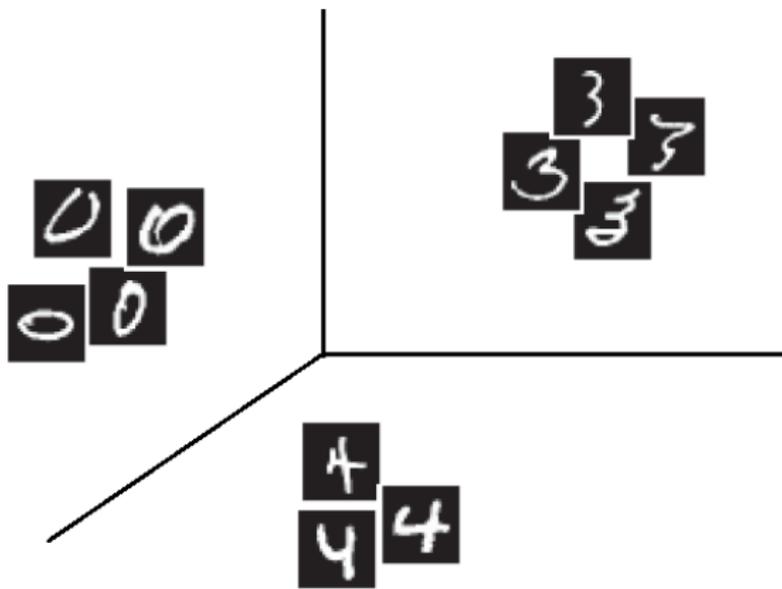
# Примеры векторного представления слов

- “кот”  $\rightarrow (0.1, 1.8, -4.2, 0.36, \dots)$
- “собака”  $\rightarrow (0.13, 1.45, -4.23, 0.41, \dots)$
- “машина”  $\rightarrow (8.31, -7.29, 0.44, -5.28, \dots)$
- “велосипед”  $\rightarrow (7.2, -6.71, 0.43, -2.45, \dots)$
- Семантически “кот” и “собака” близки, “машина” и “велосипед” тоже близки.
- Тогда их представления также должны быть близки.

# Примеры векторного представления слов



# Примеры векторного представления изображений



# Алгоритмы вложения

- Исходная выборка:  $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)})$ ,  $\mathbf{x}^{(i)} \in \mathbb{R}^D$
- Скрытый слой:  $\mathbf{h}^{(k)} = f(W_1 \mathbf{x}^{(k)} + b_1)$ ,  $\mathbf{h}^{(k)} \in \mathbb{R}^d$
- Задача оптимизации:

$$\sum_{1 \leq i \leq j \leq m} L(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}, \varphi_{ij}) \rightarrow \min_{W_1, b_1}$$

- $\varphi_{ij}$  - вес между  $\mathbf{x}^{(i)}$  и  $\mathbf{x}^{(j)}$ :

$$\varphi_{ij} = \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2, \quad \varphi_{ij} = e^{-\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2 / r},$$
$$\varphi_{ij} = \begin{cases} 1, & \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2 \leq \varepsilon \\ 0, & \text{иначе} \end{cases}$$

# Функции потерь

- Laplacian eigenmaps (LE)

$$L(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}, \varphi_{ij}) = \|\mathbf{h}^{(i)} - \mathbf{h}^{(j)}\|^2 \varphi_{ij}$$

- Multidimensional scaling (MDS)

$$L(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}, \varphi_{ij}) = (\|\mathbf{h}^{(i)} - \mathbf{h}^{(j)}\| - \varphi_{ij})^2$$

- Margin-based Embedding ( $l = 1$ )

$$L(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}, \varphi_{ij}) = \begin{cases} \|\mathbf{h}^{(i)} - \mathbf{h}^{(j)}\|^2, & \varphi_{ij} = 1, \\ \max\left(0, l - \|\mathbf{h}^{(i)} - \mathbf{h}^{(j)}\|^2\right), & \varphi_{ij} = 0. \end{cases}$$

# Функционалы потерь

- Стандартный автокодер:

$$J(\theta, \mathbf{x}) = \gamma \sum_{i=1}^n L(\mathbf{x}_i, \mathbf{z}_i) + \beta \sum_{j=1}^d KL(\rho, \hat{\rho}_j) + \frac{\lambda}{2} (\|W_1\|^2 + \|W_2\|^2)$$

- Автокодер с учетом вложения:

$$J_{em}(\theta, \varphi, \mathbf{x}) = \sum_{1 \leq i < j \leq m} L(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}, \varphi_{ij}) + J(\theta, \mathbf{x})$$

- $\gamma, \beta, \lambda$  - параметры, контролирующие баланс между штрафными слагаемыми
- W.Yu, G.Zeng, P.Luo, F.Zhuang, Q.He, Z.Shi. Embedding with Autoencoder Regularization // Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, 2013, pp. 208-223.

# Обучение автокодера

Необходимо найти частные производные:

$$\frac{\partial J_{em}(\theta, \varphi, \mathbf{x})}{\partial W_t}, \quad \frac{\partial J_{em}(\theta, \varphi, \mathbf{x})}{\partial b_t}, \quad t = 1, 2.$$

Определим  $\delta^{(1)}$  и  $\delta^{(2)}$  для скрытого и выходного слоя, соответственно, как

$$\delta^{(1)} = \left( (W_2)^T \delta^{(2)} + \beta \frac{\hat{\rho} - \rho_0}{\hat{\rho}(1 - \rho_0)} \right) \cdot \sigma'(\mathbf{h})$$

$$\delta^{(2)} = \frac{\partial}{\partial \mathbf{z}} \frac{1}{2} \|\mathbf{h} - \mathbf{x}\|^2 = -(\mathbf{h} - \mathbf{x}) \cdot \sigma'(\mathbf{z}),$$

где  $\rho_0 \in \mathbb{R}^d$  - вектор всех  $\rho$ ,

$$\mathbf{h} = W_1 \mathbf{x} + b_1, \quad \mathbf{z} = W_2 \mathbf{h} + b_2, \quad \hat{\rho} = n^{-1} \sum_{i=1}^n \mathbf{h}^{(i)}$$

# Вычисление частных производных

- 1 Случайно инициализируем  $\theta = (W_1, W_2, b_1, b_2)$
- 2 Прямое распространение - вычисляем:  $h^{(i)}, z^{(i)}, \delta^{(1)}, \delta^{(2)}$
- 3 Вычисляем частные производные:

$$\frac{\partial J_{em}(\theta, \varphi, \mathbf{x})}{\partial W_2} = \gamma \delta^{(2)} \mathbf{h}^T + \lambda W_2$$

$$\frac{\partial J_{em}(\theta, \varphi, \mathbf{x})}{\partial b_2} = \gamma \delta^{(2)}$$

$$\frac{\partial J_{em}(\theta, \varphi, \mathbf{x})}{\partial W_1} = \frac{\partial}{\partial W_1} \sum_{i \leq j} L(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}, \varphi_{ij}) + \gamma \delta^{(1)} \mathbf{x}^T + \lambda W_1$$

$$\frac{\partial J_{em}(\theta, \varphi, \mathbf{x})}{\partial b_1} = \frac{\partial}{\partial b_1} \sum_{i \leq j} L(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}, \varphi_{ij}) + \gamma \delta^{(1)}$$

# Результирующий алгоритм обучения

- 1 Вычисляем  $\varphi_{ij}$
- 2 Цикл до критерия остановки
  - $\Delta W_t = 0, \Delta b_t = 0, t = 1, 2.$
  - Вычисляем все частные производные
  - $\Delta W_t = \sum_{i=1}^n \frac{\partial J_{em}(\theta, \varphi, \mathbf{x}^{(i)})}{\partial W_t}, \Delta b_t = \sum_{i=1}^n \frac{\partial J_{em}(\theta, \varphi, \mathbf{x}^{(i)})}{\partial b_t}$
  - Модификация:  $W_t = W_t - \alpha (n^{-1} \Delta W_t),$   
 $b_t = b_t - \alpha (n^{-1} \Delta b_t)$
- 3 Конец цикла
- 4 Вычисляем результаты (представление)  $h$

# Какие автокодеры еще есть?

- Вариационный автокодер (Variational autoencoder)
- Автокодер со значимыми весами (Importance weighted autoencoder)
- Соперничающие автокодеры (Adversarial autoencoders)

**Это все порождающие модели**

# Применение автокодера

- 1 Снижение размерности (компактное представление) данных.
- 2 “Pretraining”: обучение автокодера (без учителя) для получения весов, далее использование этой же конфигурации для глубокой сети и использование весов как исходных (это лучше, чем случайные веса).
- 3 Одноклассовая классификация: автокодер обучается на данных и определяется порог ошибки реконструирования, далее этот порог используется для тестирования аномальных наблюдений.
- 4 и многое другое...

# Вопросы

?