Методы машинного обучения

Шорохов С.Г.

кафедра математического моделирования и искусственного интеллекта

Лекция 7. Автокодировщики

Понижение размерности

Широко распространенная форма машинного обучения без учителя – это понижение размерности, когда требуется обучить отображение из видимого пространства высокой размерности ($\mathbf{x} \in \mathbb{R}^d$) в латентное (скрытое) пространстве меньшей размерности ($\mathbf{z} \in \mathbb{R}^l$).

Суть задачи понижения размерности состоит в том, чтобы имея данные с большим количеством признаков (столбцов), преобразовать их в новую таблицу с меньшим количеством столбцов. Количество строк (объектов) при этом останется неизменным.

Уменьшить размерность можно двумя способами. Первый способ называется отбором признаков. В этом случае алгоритм оставляет в наборе данных только те признаки, которые будут наиболее важны для решения задачи. Еще алгоритм уберет признаки, у которых много пропущенных значений. Также алгоритм найдет и удалит коррелирующие признаки – то есть те, которые сильно связаны друг с другом.

Кроме отбора есть второй способ – агрегация признаков. В этом случае алгоритм соберёт несколько составных признаков, или даже на их основе создаст новые. Этот способ часто используется для обработки изображений и для работы с большими массивами данных.

Модели агрегации признаков

Модель агрегации признаков может быть либо параметрической моделью вида $\mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta})$, применимой к любым входным данным $\mathbf{x}$, либо непараметрической моделью, когда вычисляется погружение или эмбеддинг (embedding) $\mathbf{z}_i$ для всех входных данных $\mathbf{x}_i$, принадлежащих некоторому набору данных $\mathbf{D}$, и только для них. Второй подход чаще всего используется для визуализации данных, а первый еще и как шаг предобработки для других видов алгоритмов обучения. Например, мы могли бы понизить размерность, обучив погружение $\mathbf{x}$ в $\mathbf{z}$, а затем обучить на этом погружении простой линейный классификатор, который отображает $\mathbf{z}$ в $\mathbf{y}$.

Задача понижения размерности — это скорее вспомогательная задача для решения других задач машинного обучения, например классификации, регрессии или кластеризации, однако и самостоятельные применения у задачи понижения размерности тоже есть

Цели понижения размерности

- Сжатие данных. Большая таблица данных может занимать много места на жестком диске, и уменьшив количество признаков в N раз, мы уменьшим размер файла с данными в те же самые N раз.
- Ускорение предсказаний. Если алгоритму предсказания (например, алгоритму предсказания ухода клиента или алгоритму кластеризации) нужно обработать тысячи признаков, это будет занимать гораздо больше времени, чем если он будет обрабатывать десятки признаков. При этом во многих задачах, особенно связанных с онлайн-сервисами, существуют ограничения на время выполнения предсказаний, например поисковый запрос должен выполняться за доли секунды.
- Визуализация данных. Если алгоритм понижения размерности составит новую таблицу с двумя столбцами, такие данные будет легко визуализировать, отложив по осям два новых признака.
- Более компактное и корректное описание объектов. Новые признаки могут описывать объекты более емко, чем исходные, что упростит работу другим алгоритмам. Более того, среди исходных признаков могут быть такие, которые ухудшают предсказания, и при понижении размерности эти признаки будут удалены.

Метод главных компонент (PCA)

Метод главных компонент (Principal Component Analysis, PCA) – это метод, позволяющий найти пространственный базис, который наилучшим образом отражает дисперсию (разброс) в наборе данных.

Допустим, что данные уже центрированы путем вычитания средних значений $\mathbf{x} \leftarrow \mathbf{x} - \mathbb{E}[\mathbf{x}]$.

Первая главная компонента (направление с максимальной дисперсией) находится путем максимизации дисперсии данных, умноженных на вектор $\mathbf{w}_1$ единичной длины:

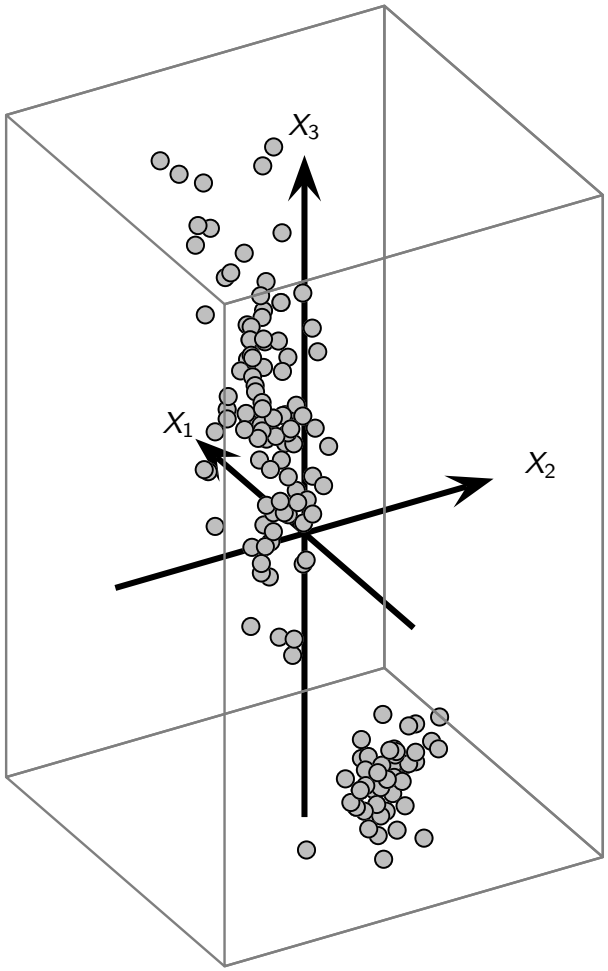
$$y_1 = \mathbf{x}^T\mathbf{w}_1,\ \|\mathbf{w}_1\| = 1 \Rightarrow \mathbb{E}\left[y_1^2\right] = \mathbb{E}\left[\mathbf{w}_1^T\mathbf{x}\mathbf{x}^T\mathbf{w}_1\right] = \mathbf{w}_1^T\mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right]\mathbf{w}_1 = \mathbf{w}_1^T\boldsymbol{\Sigma}_\mathbf{x}\mathbf{w}_1,$$

где $\boldsymbol{\Sigma}_\mathbf{x}$ – ковариационная матрица. Отсюда получаем, что решение (направление) задается собственным вектором ковариационной матрицы $\boldsymbol{\Sigma}_\mathbf{x}$, соответствующим наибольшему собственному значению матрицы $\boldsymbol{\Sigma}_\mathbf{x}$:

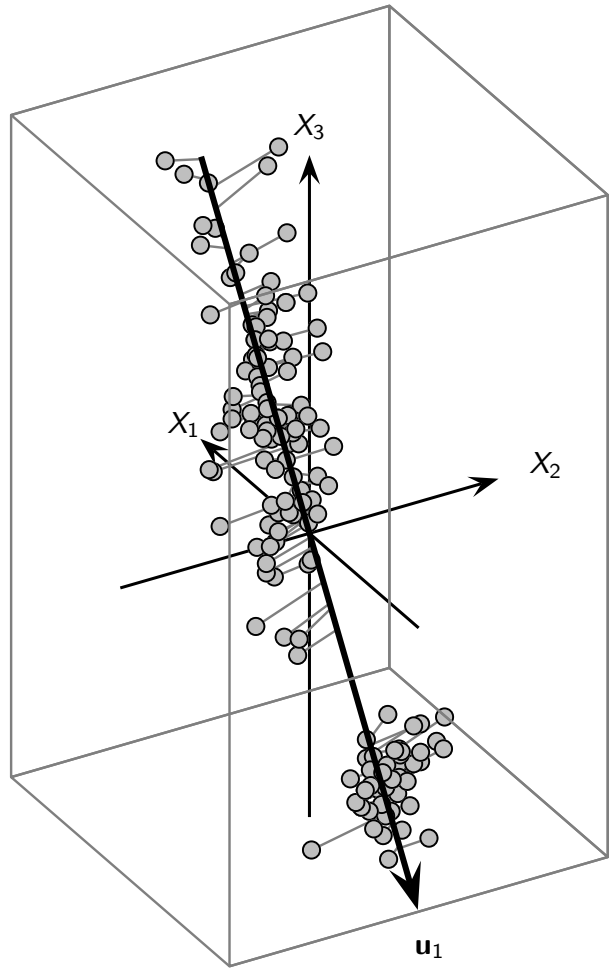
$$\mathbf{w}_1^* = \underset{\mathbf{w}_1:\,\|\mathbf{w}_1\|=1}{\arg\max}\ \mathbf{w}_1^T\boldsymbol{\Sigma}_\mathbf{x}\mathbf{w}_1$$

Вторая главная компонента находится как направление с максимальной дисперсией при условии ортогональности направлению первой главной компоненты (и т.д.).

Набор данных «Ирисы»: оптимальный базис 1d

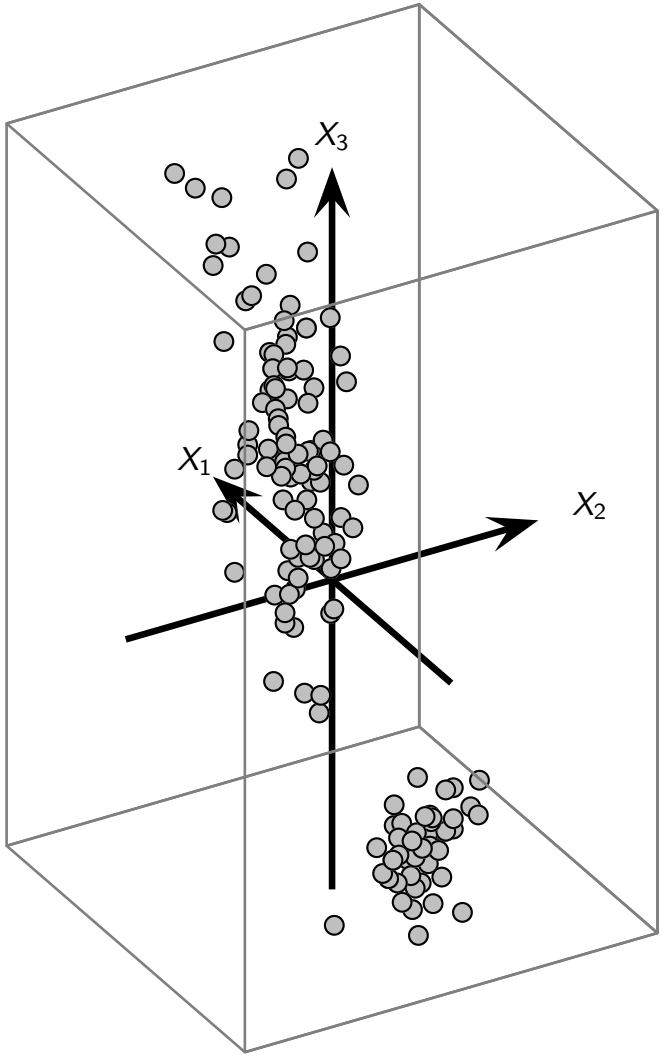


Набор данных "Ирисы": 3D

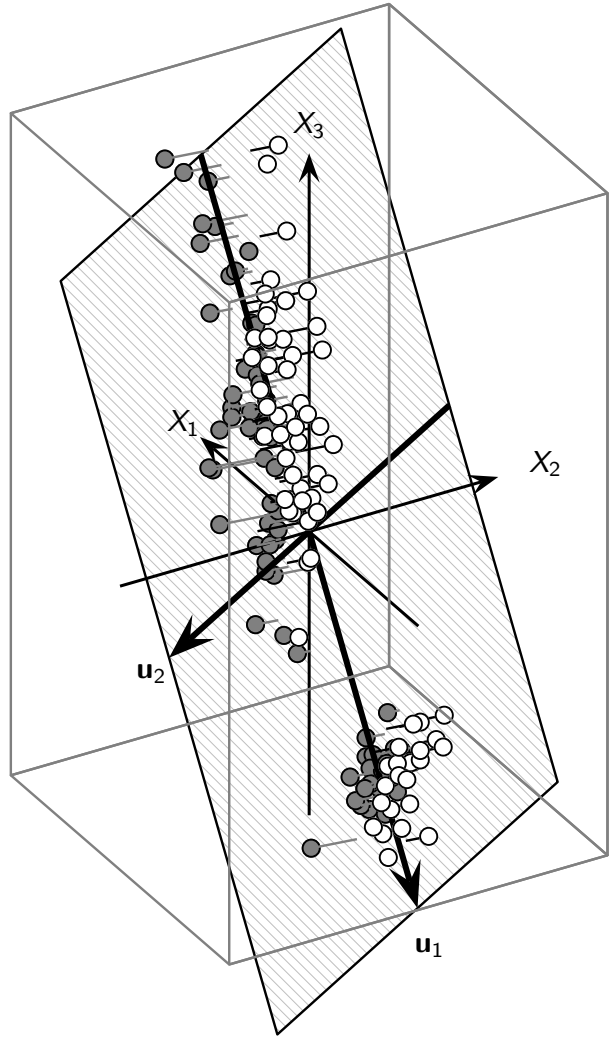


Оптимальный базис: 1D

Набор данных «Ирисы»: оптимальный базис 2d



Набор данных "Ирисы": 3D



Оптимальный базис: 2D

Алгоритм PCA (метода главных компонент)

Data: $\mathbf{D}$ – исходный набор данных, α – доля общей дисперсии
Result: $\mathbf{A}$ – набор данных уменьшенной размерности

1 $\overline{\mu} = \frac{1}{n}\sum_{i=1}^{n}\overline{x}_i$ // вычислить средние значения
2 $Z = \mathbf{D} - \overline{1}\cdot\overline{\mu}^T$ // центрировать данные
3 $\Sigma = \frac{1}{n}\left(Z^T Z\right)$ // вычислить матрицу ковариации
4 $\overline{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_d)$ // вычислить собственные значения
5 $U = \begin{pmatrix} \overline{u}_1 & \overline{u}_2 & ... & \overline{u}_d \end{pmatrix}$ // вычислить собственные векторы
6 $f(r) = \frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i},\ r = 1, 2, ..., d$ // вычислить доли общей дисперсии
7 Выбрать наименьшее r, такое, что $f(r) \geqslant \alpha$ // выбрать размерность
8 $U_r = \begin{pmatrix} \overline{u}_1 & \overline{u}_2 & ... & \overline{u}_r \end{pmatrix}$ // сокращенный базис
9 $\mathbf{A} = \left\{\overline{a}_i \mid \overline{a}_i = U_r^T\overline{x}_i,\ i = \overline{1,n}\right\}$ // уменьшенная размерность данных

Algorithm 1: Алгоритм PCA

Главные компоненты набора данных «Ирисы»

Матрица ковариации (для первых 3 признаков):

$$\Sigma = \begin{pmatrix} 0.681 & -0.039 & 1.265 \\ -0.039 & 0.187 & -0.320 \\ 1.265 & -0.320 & 3.092 \end{pmatrix}$$

Собственные значения и собственные вектора Σ

$$\lambda_1 = 3.662,\ \lambda_2 = 0.239,\ \lambda_3 = 0.059$$

$$\overline{u}_1 = \begin{pmatrix} -0.390 \\ 0.089 \\ -0.916 \end{pmatrix},\ \overline{u}_2 = \begin{pmatrix} -0.639 \\ -0.742 \\ 0.200 \end{pmatrix},\ \overline{u}_3 = \begin{pmatrix} -0.663 \\ 0.664 \\ 0.346 \end{pmatrix}$$

Общая дисперсия равна $\lambda_1 + \lambda_2 + \lambda_3 = 3.662 + 0.239 + 0.059 = 3.96$. Доля общей дисперсии для различных значений r равна

r	1	2	3
$f(r)$	0.925	0.985	1.0

Достаточно двух главных компонент ($r = 2$), чтобы охватить долю дисперсии $\alpha = 0.95$

Применение PCA к графическим данным (1)

Рассмотрим в качестве примера применение метода PCA к тестовому изображению Lenna (TIFF 512x512 24bit RGB color).

TIFF (Tagged Image File Format) – это формат хранения растровых графических изображений.

При использовании формата TIFF без сжатия появляются графические файлы большого размера.

Применение PCA к графическим данным (2)

Применяя к набору данных тестового изображения метод PCA и оставляя только первую главную компоненту, а именно, используя представление:

$$\begin{bmatrix} R_{i,j} \\ G_{i,j} \\ B_{i,j} \end{bmatrix} = \begin{bmatrix} 0.767785 \\ 0.45439 \\ 0.4517034 \end{bmatrix} Y_{i,j},$$

где $Y_{i,j}$ – значения первой главной компоненты, соответствующие пикселю (i, j), получим следующее изображение

PCA как сжатие с минимальной ошибкой MSE

Пусть построено m-мерное подпространство, порожденное ортонормальным базисом из m главных компонент $\mathbf{w}_1, ..., \mathbf{w}_m$:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \dots & \mathbf{w}_m \end{bmatrix}, \mathbf{W}^T\mathbf{W} = \mathbf{I}_m$$

Можно спроектировать d-мерные векторы данных $\mathbf{x}$ на это подпространство: $\mathbf{z} = \mathbf{W}^T\mathbf{x}$

Реконструкция $\mathbf{x}$, которая остается внутри m-мерного подпространства, порожденного $\mathbf{W}$, равна

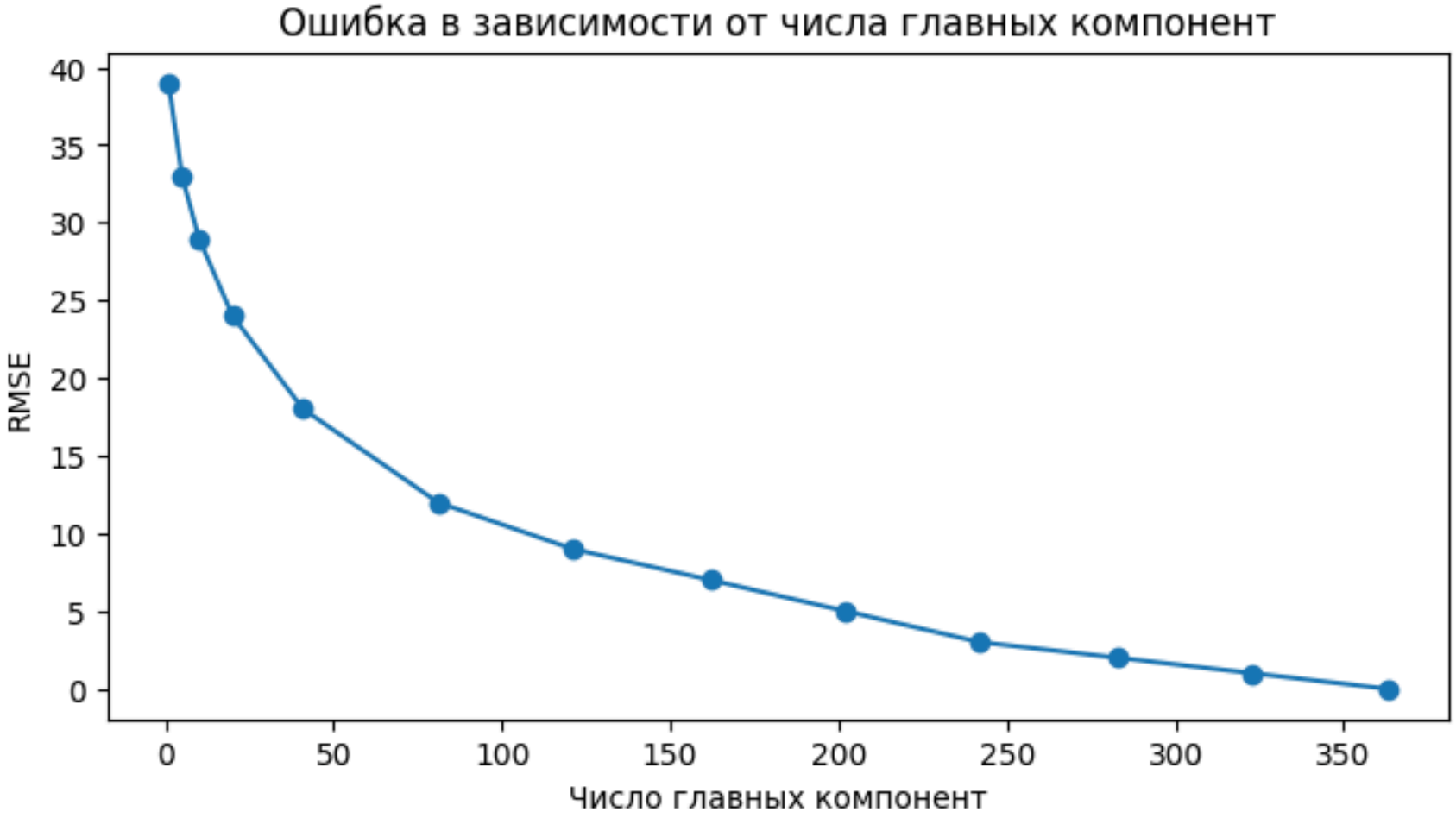
$$\hat{\mathbf{x}} = \mathbf{W}\mathbf{z} = \mathbf{W}\mathbf{W}^T\mathbf{x}$$

Поэтому мы находим такое $\mathbf{W}$, что среднеквадратическая ошибка между исходными данными $\mathbf{x}$ и реконструированным вектором $\hat{\mathbf{x}}$ сведена к минимуму:

$$\mathbf{W}_{PCA} = \underset{\mathbf{W}:\,\mathbf{W}^T\mathbf{W}=\mathbf{I}_m}{\arg\min} \mathbb{E}\left[\left\| x - \underbrace{\mathbf{W}\mathbf{W}^T\mathbf{x}}_{\hat{\mathbf{x}}} \right\|^2\right]$$

Выбор числа латентных измерений в PCA

Как выбирать число латентных измерений (количество главных компонент) m в методе PCA? При увеличении количества главных компонент ошибка обычно быстро убывает, что означает, что большую часть зависимостей можно отразить при помощи небольшого числа главных компонент.



Правдоподобие профиля

Хотя график ошибки не имеет U-образной формы, часто кривая содержит участок, где ошибка резко переходит от сравнительно больших значений к сравнительно малым. Идея в том, что для $m < m^*$, где m^* – истинная латентная размерность, скорость убывания функции ошибки будет высокой, а для $m > m^*$ выигрыш будет меньше, потому что модель и так уже достаточно сложна. Один из способов автоматизировать обнаружение этого изменения градиента кривой – вычислить правдоподобие профиля (profile likelihood).

Пусть λ_m – некоторая мера ошибки, вносимой моделью размера m, так что $\lambda_1 \geqslant \lambda_2 \geqslant \ldots$. Рассмотрим разбиение этих значений на две группы – для $k < m^*$ и $k > m^*$, где m^* – некоторая пороговая величина, которую нужно найти.

Логарифмическое правдоподобие профиля

Разобьем данные на части и вычислим правдоподобие с применением объединенной оценки дисперсии:

$$\mu_1(m) = \frac{1}{m}\sum_{k\leqslant m}\lambda_k,\ \mu_2(m) = \frac{1}{d-m}\sum_{k>m}\lambda_k,$$

$$\sigma^2(m) = \frac{1}{d}\left(\sum_{k\leqslant m}(\lambda_k - \mu_1(m))^2 + \sum_{k>m}(\lambda_k - \mu_2(m))^2\right).$$

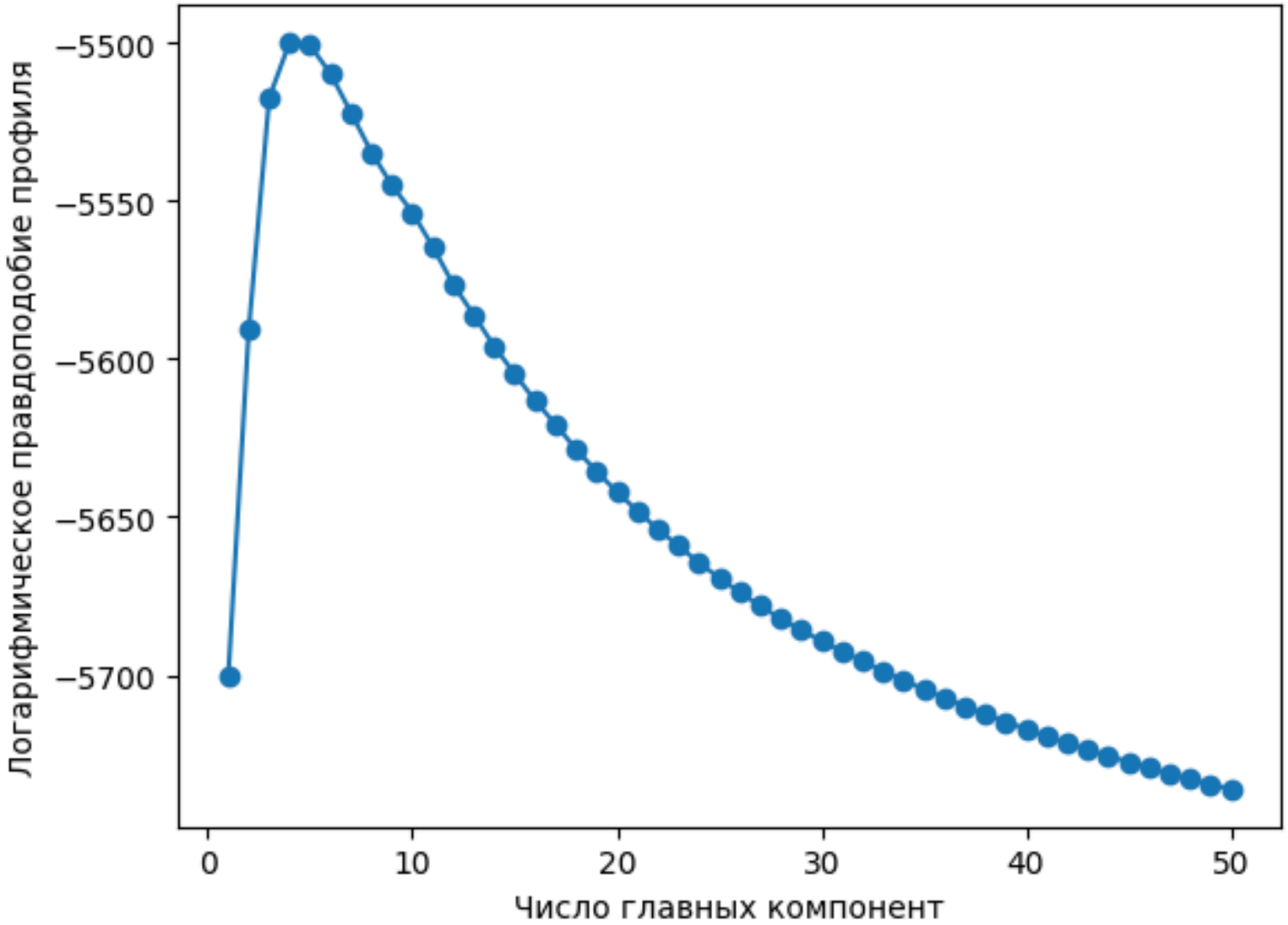
Вычисляем логарифмическое правдоподобие профиля по формуле:

$$l(m) = \sum_{k=1}^{m}\log\mathcal{N}\left(\lambda_k \mid \mu_1(m), \sigma^2(m)\right) + \sum_{k=m+1}^{d}\log\mathcal{N}\left(\lambda_k \mid \mu_2(m), \sigma^2(m)\right),$$

где $\mathcal{N}$ – плотность вероятности нормального распределения.

График правдоподобия профиля

Из графика ниже вытекает, что величина $m^* = \arg\max_m l(m)$ корректно определена.



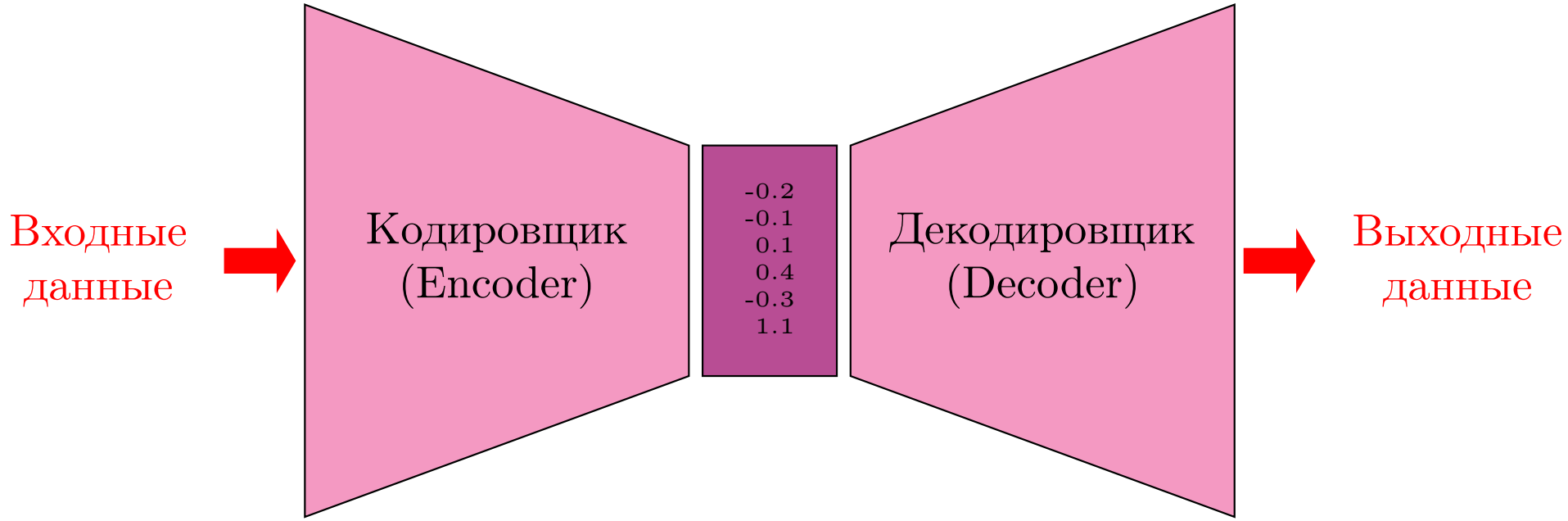
Автокодировщики

Нейронные сети обычно используются при обучении с учителем (supervised learning). Это означает, что для каждой точки обучающего набора данных $\mathbf{x}_i$ имеется отклик y_i. Во время обучения модель нейронной сети будет изучать взаимосвязь между входными данными и откликами. Теперь предположим, что имеются только немаркированные данные, то есть у нас есть только обучающий набор данных $\mathbf{D}$, состоящий из n точек $\mathbf{x}_i \in \mathbb{R}^d$, где $i = 1, ..., n$.

Автокодировщики были введены Румелхартом, Хинтоном и Уильямсом в 1986 году с целью «научиться восстанавливать входные наблюдения $\mathbf{x}_i$ с минимально возможной ошибкой».

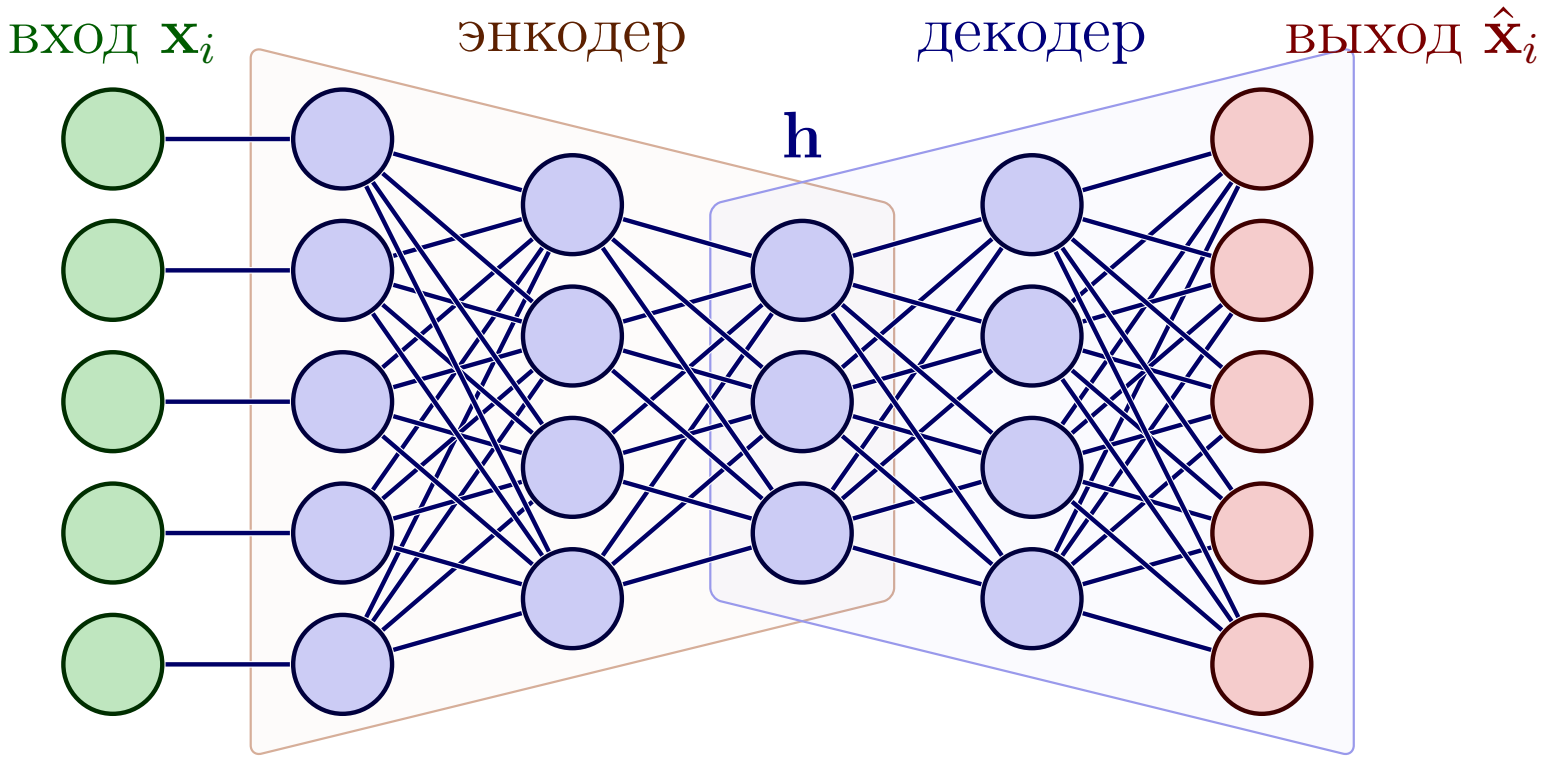
Автокодировщик — это тип алгоритма, основной целью которого является обучение «информативному» представлению данных, которое можно использовать для различных приложений, путем обучения достаточно хорошей реконструкции набора входных данных.

Типичная архитектура автокодировщика



Основными компонентами автокодировщика являются кодировщик, представление скрытых (латентных) признаков и декодер. Кодировщик и декодер — это просто функции, представленные нейронными сетями, тогда как представление скрытых признаков обычно является числовым тензором. Вообще говоря, требуется, чтобы автокодировщик достаточно хорошо реконструировал входные данные. В то же время он должен создавать скрытое представление (выход кодировщика), которое является полезным, например, имеет меньшую размерность.

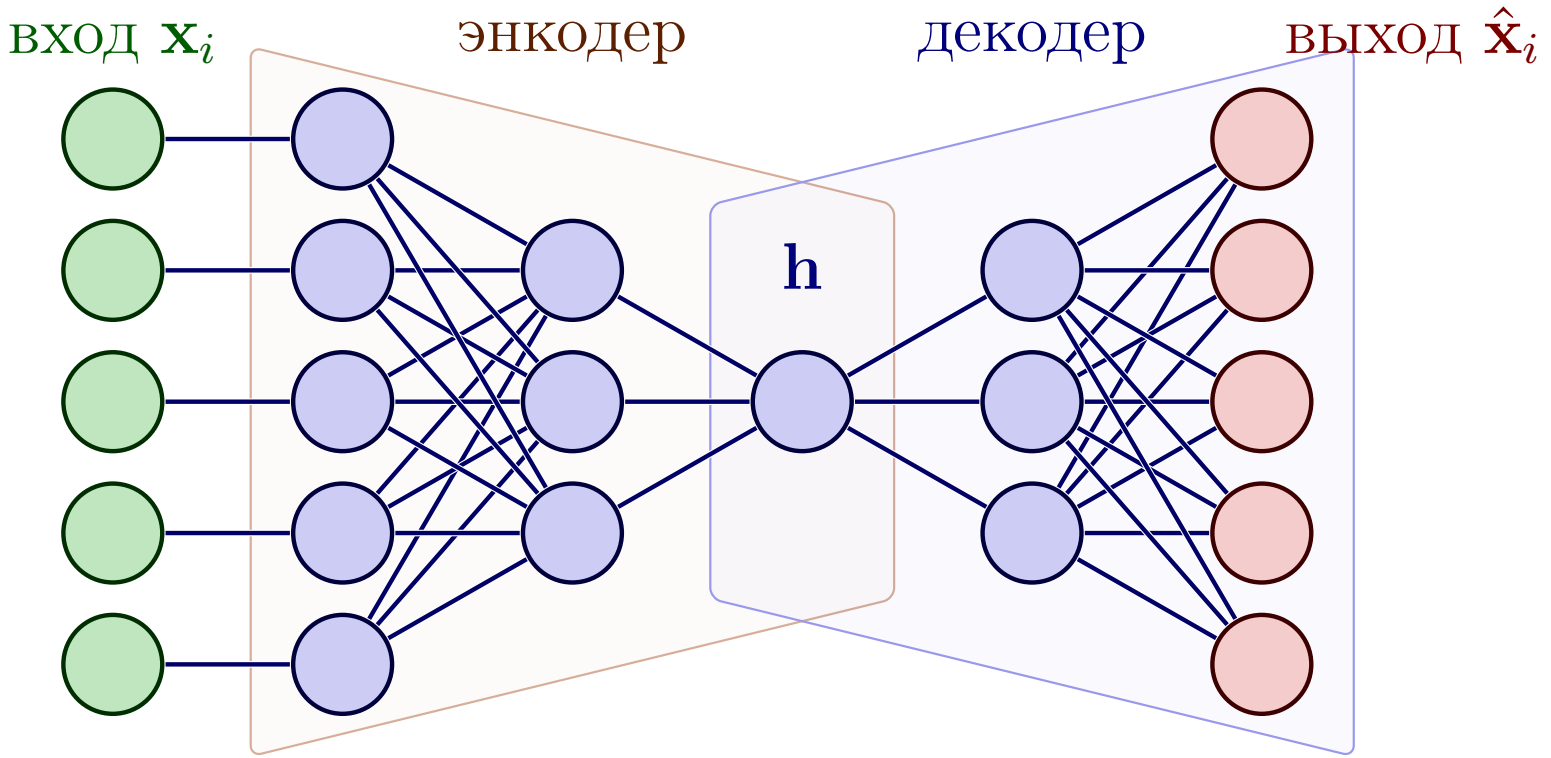
Особенности автокодировщика



Автокодировщик – нейронная сеть прямого распространения, которая

- кодирует входное значение $\mathbf{x}_i$ в скрытое представление $\mathbf{h}$
- декодирует скрытое представление $\mathbf{h}$ в выходное значение $\hat{\mathbf{x}}_i$
- модель обучается на функции потерь, которая гарантирует, что выходное значение $\hat{\mathbf{x}}_i$ будет близко к входному значению $\mathbf{x}_i$

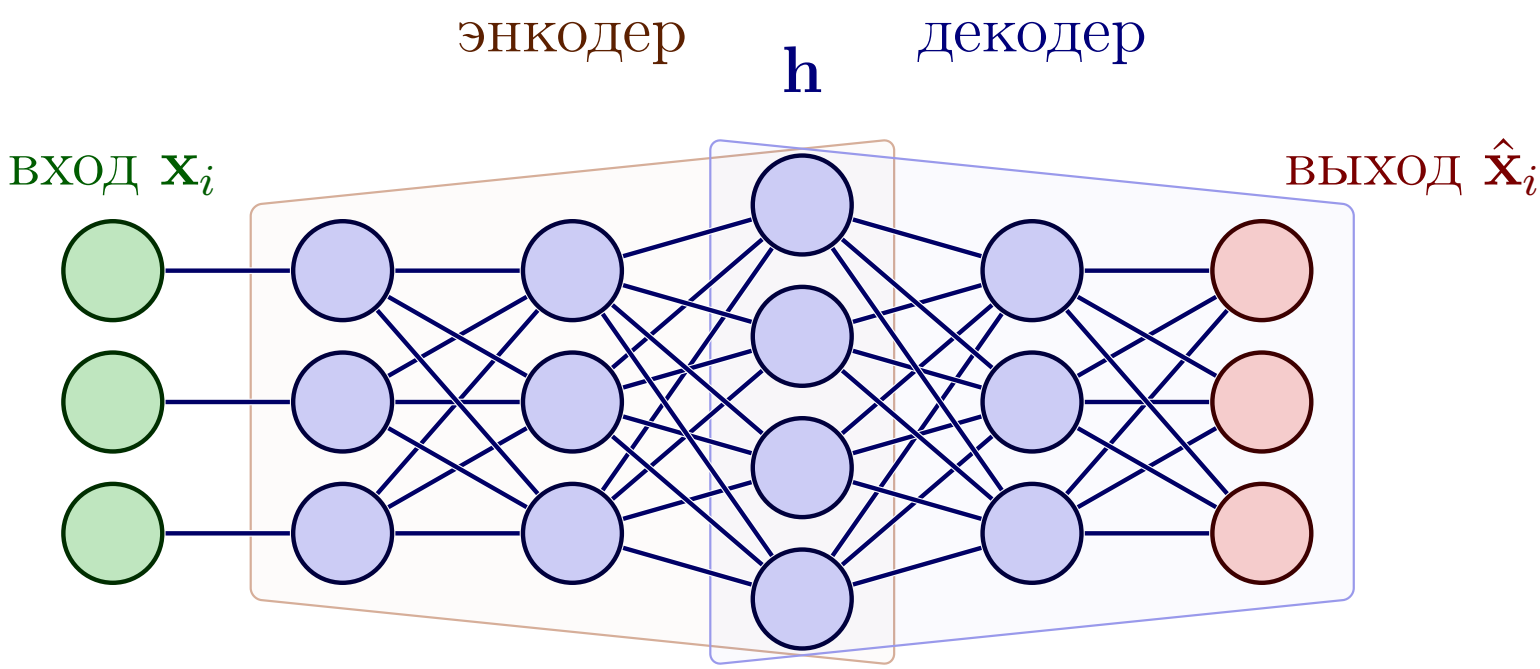
Автокодировщик с сужением



Автокодировщик, для которого $\dim(\mathbf{h}) < \dim(\mathbf{x}_i)$, называется автокодировщиком с сужением (under complete autoencoder).

Если автокодировщик способен реконструировать $\hat{\mathbf{x}}_i$ идеальным образом из $\mathbf{h}$, то это означает, что $\mathbf{h}$ представляет собой кодировку $\mathbf{x}_i$ без потерь, которая содержит в себе все важные характеристики $\mathbf{x}_i$

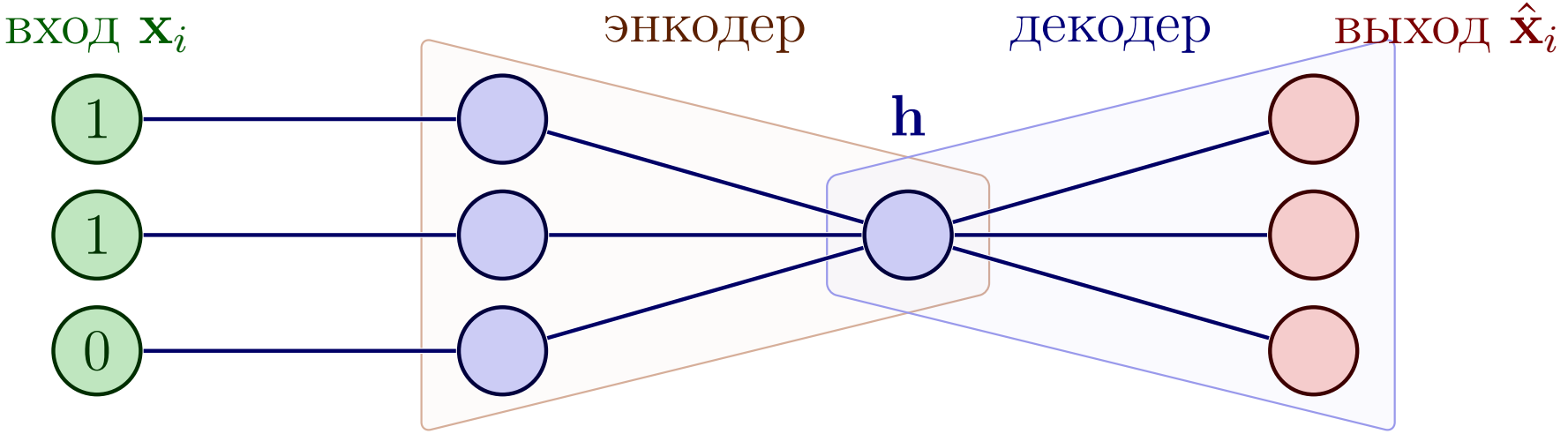
Автокодировщик с расширением



Автокодировщик, для которого $\dim(\mathbf{h}) \geqslant \dim(\mathbf{x}_i)$, называется автокодировщиком с расширением (over complete autoencoder).

В этом случае автокодировщик может изучить тривиальное кодирование путем копирования $\mathbf{x}_i$ в скрытое представление $\mathbf{h}$ и затем декодирование путем копирования $\mathbf{h}$ в $\hat{\mathbf{x}}_i$. Такое тождественное кодирование бесполезно, так как оно не извлекает из данных какой-либо полезной информации.

Функции активации в автокодировщике

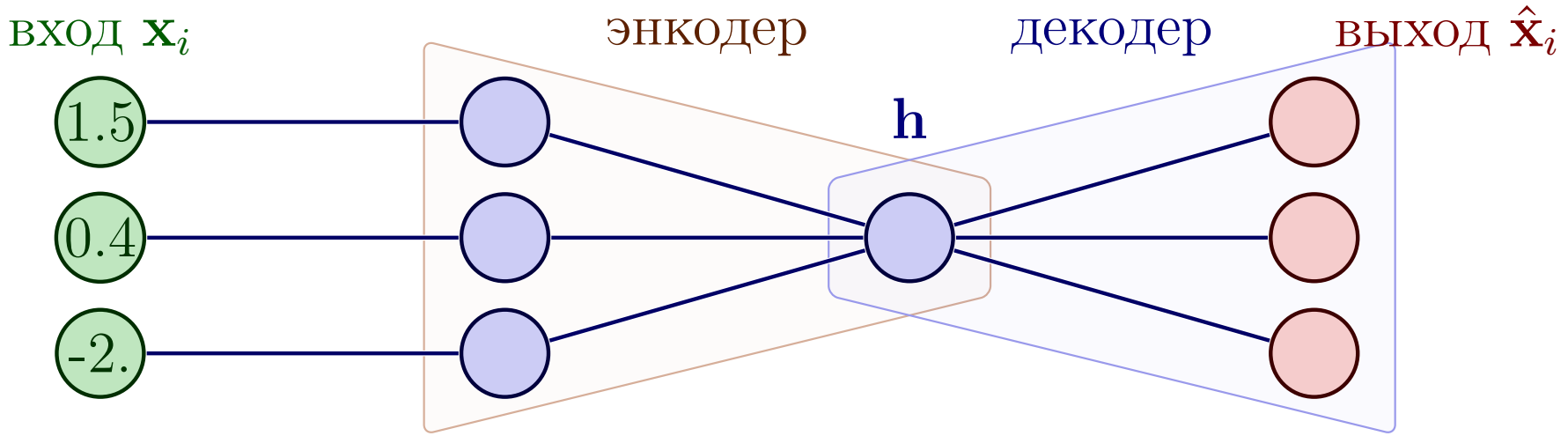


Пусть все входные значения являются бинарными ($x_{ij} \in \{0, 1\}$). Какая из функций активации будет наиболее адекватна в выходном слое?

- $\hat{\mathbf{x}}_i = \tanh(\mathbf{W}^*\mathbf{h} + \mathbf{b}_o)$
- $\hat{\mathbf{x}}_i = \mathbf{W}^*\mathbf{h} + \mathbf{b}_o$
- $\hat{\mathbf{x}}_i = \text{sigmoid}(\mathbf{W}^*\mathbf{h} + \mathbf{b}_o)$

Сигмоида ограничивает выходные значения между 0 и 1, поэтому выбираем в качестве функции активации сигмоиду.

Функции активации в автокодировщике

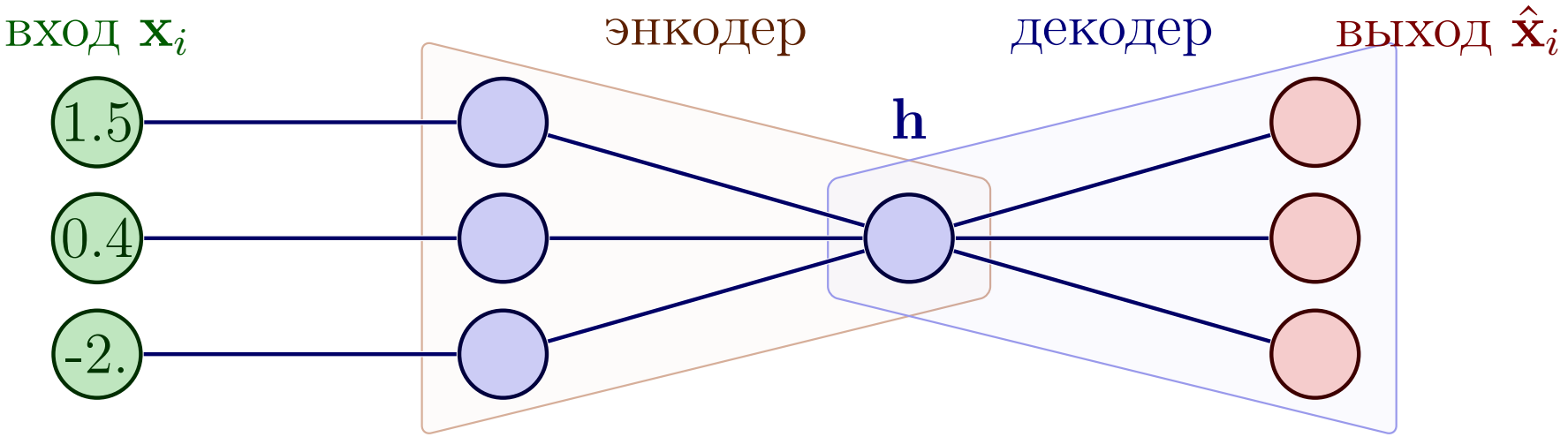


Пусть все входные значения являются произвольными ($x_{ij} \in \mathbb{R}$). Какая из функций активации будет наиболее адекватна в выходном слое?

- $\hat{\mathbf{x}}_i = \tanh\left(\mathbf{W}^*\mathbf{h} + \mathbf{b}_o\right)$
- $\hat{\mathbf{x}}_i = \mathbf{W}^*\mathbf{h} + \mathbf{b}_o$
- $\hat{\mathbf{x}}_i = \text{sigmoid}\left(\mathbf{W}^*\mathbf{h} + \mathbf{b}_o\right)$

Сигмоида и tanh ограничивает выходные значения между $[0, 1]$ и $[-1, 1]$, поэтому выбираем в качестве функции активации тождественную функцию.

Функции потерь для автокодировщика

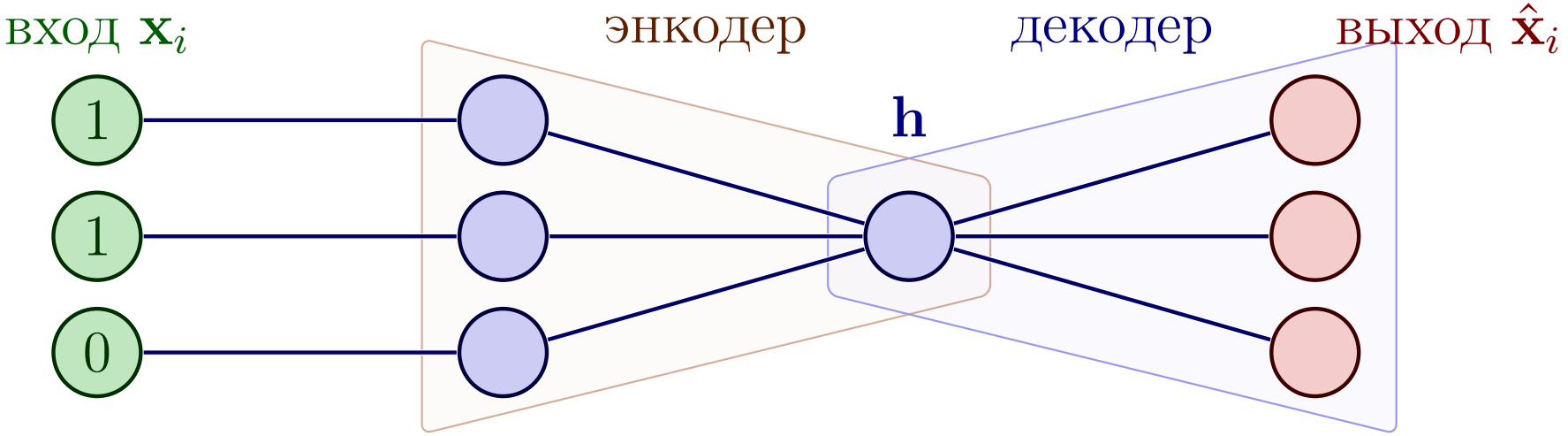


Пусть все входные значения являются произвольными ($x_{ij} \in \mathbb{R}$). Цель автокодировщика – реконструировать выходные значения $\hat{\mathbf{x}}_i$ как можно более близкими к входным значениям $\mathbf{x}_i$. Это можно формализовать при помощи следующей суммы квадратов ошибки

$$\sum_i \left(\hat{\mathbf{x}}_i - \mathbf{x}_i\right)^T \left(\hat{\mathbf{x}}_i - \mathbf{x}_i\right)$$

Тогда можно обучать автокодировщик по аналогии с нейронной сетью прямого распространения при помощи обратного распространения ошибки.

Функции потерь для автокодировщика



Пусть все входные значения являются бинарными и используется декодер с сигмоидой, которая согдает выходные значения от 0 до 1, которые могут быть интерпретированы как вероятности.

Для одного n-мерного вектора входных значений $\mathbf{x}_i$ потери будут оцениваться при помощи бинарной кросс-энтропии

$$-\sum_j \left(x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log \left(1 - \hat{x}_{ij}\right)\right)$$

PCA как автокодировщик со сжатием

Метод PCA может рассматриваться как автокодировщик. Будем обучать отображение из $\mathbf{x}$ в $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = g\left(f\left(\mathbf{x}\right)\right),$$

где:

- $f\left(\mathbf{x}\right) = \mathbf{W}_f\,\mathbf{x} + \mathbf{b}_f$ – кодировщик (линейный)
- $g\left(\mathbf{z}\right) = \mathbf{W}_g\,\mathbf{z} + \mathbf{b}_g$ – декодер (линейный)
- $\mathcal{L}\left(\mathbf{x}\right) = \mathbb{E}\left[\left\|\mathbf{x} - g\left(f\left(\mathbf{x}\right)\right)\right\|^2\right]$ – функция потерь

Если мы не ограничиваем функции f и g, то мы можем изучить тривиальное тождественное отображение:

$$\hat{\mathbf{x}} = g\left(f\left(\mathbf{x}\right)\right) = \mathbf{W}_g\left(\mathbf{W}_f\,\mathbf{x} + \mathbf{b}_f\right) + \mathbf{b}_g = \left(\mathbf{W}_g\,\mathbf{W}_f\right)\mathbf{x} + \left(\mathbf{W}_g\,\mathbf{b}_f + \mathbf{b}_g\right) = \mathbf{x}$$

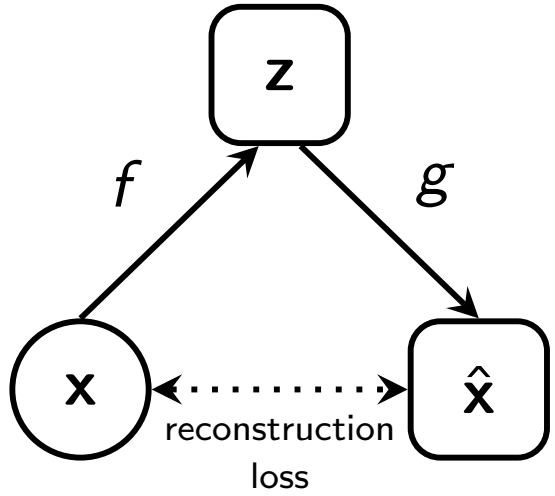
$$\Longrightarrow \mathbf{W}_g = \mathbf{W}_f^{-1},\ \mathbf{b}_g = -\mathbf{W}_g\,\mathbf{b}_f$$

Если размерность $\mathbf{z}$ меньше размерности $\mathbf{x}$, то автокодирование полезно, так как мы сжимаем данные. Таким образом, PCA может быть реализован с помощью автоэнкодера со сжатием.

Как можно улучшить сжатие?

Пусть имеем линейный автокодировщик

$$\hat{\mathbf{x}} = g\left(f\left(\mathbf{x}\right)\right),$$
$$f\left(\mathbf{x}\right) = \mathbf{W}_f\,\mathbf{x} + \mathbf{b}_f,$$
$$g\left(\mathbf{z}\right) = \mathbf{W}_g\,\mathbf{z} + \mathbf{b}_g$$



Как можно улучшить сжатие, чтобы получить меньшую ошибку реконструкции

$$\mathcal{L}\left(\mathbf{x}\right) = \mathbb{E}\left[\left\|\mathbf{x} - g\left(f\left(\mathbf{x}\right)\right)\right\|^2\right]$$

со слоем сужения того же размера?

Мы можем использовать нелинейные кодировщик f и декодер g.

Глубокие автокодировщики с сужением

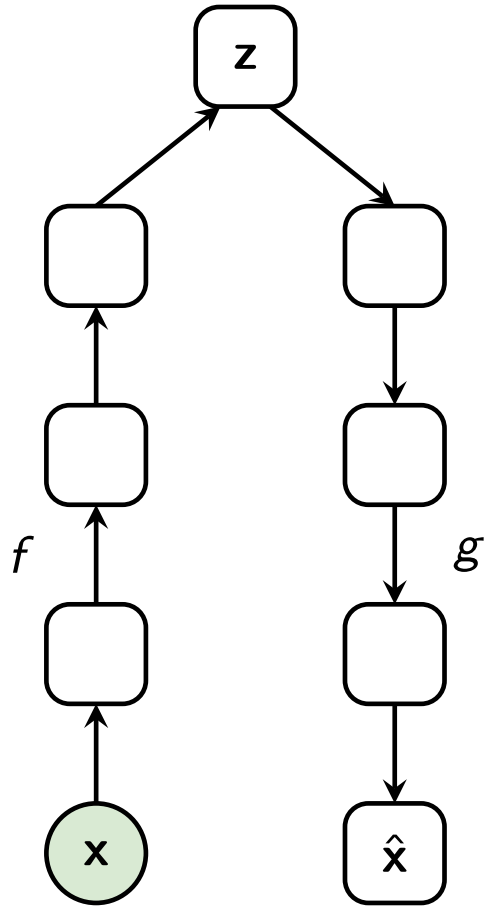
Глубокий автокодировщик:

- и кодировщик, и декодер являются глубокими нейронными сетями
- критерием оптимизации является среднеквадратическая ошибка реконструкции:

$$\boldsymbol{\theta}_f, \boldsymbol{\theta}_g = \arg\min_{\boldsymbol{\theta}_f, \boldsymbol{\theta}_g} \mathbb{E}\left[\|\mathbf{x} - g\left(f\left(\mathbf{x}; \boldsymbol{\theta}_f\right); \boldsymbol{\theta}_g\right)\|^2\right]$$

Автокодировщик с сужением:

- чтобы предотвратить изучение тривиальной (тождественной) функции f, мы используем латентное пространство $\mathbf{z}$ с меньшим количеством измерений (слой сужения).



Шумоподавляющие автокодировщики

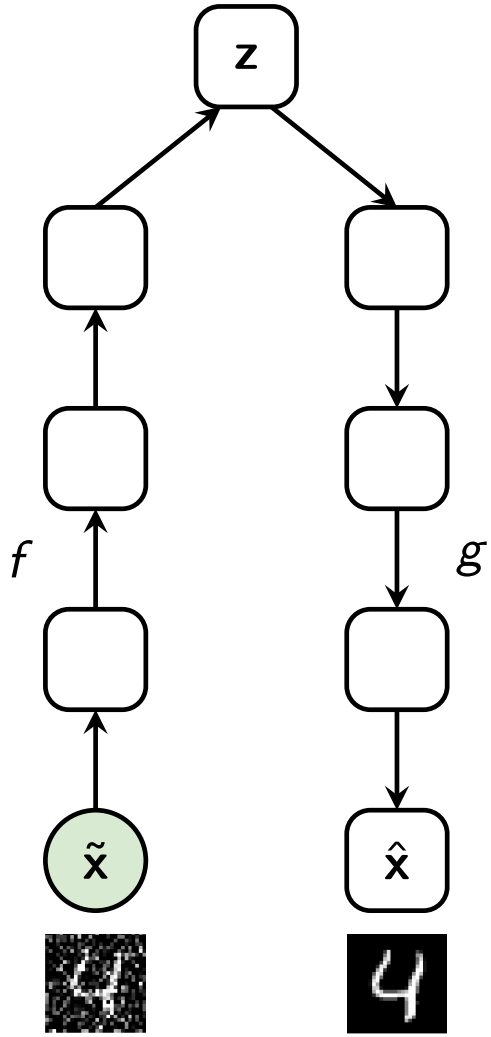
Полезный способ управления емкостью автокодировщика – прибавить шум к его входу, а затем обучить модель реконструировать чистую (неискаженную) версию оригинальных данных.

$$\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\varepsilon},\ \boldsymbol{\varepsilon} = (\varepsilon_1, ..., \varepsilon_d)\,,\ \varepsilon_i \sim \mathcal{N}\left(0,\ \sigma^2\right)$$

$$\boldsymbol{\theta}_f,\ \boldsymbol{\theta}_g = \arg\min_{\boldsymbol{\theta}_f,\,\boldsymbol{\theta}_g} \mathbb{E}\left[\left\|\mathbf{x} - \underbrace{g\left(f\left(\tilde{\mathbf{x}};\boldsymbol{\theta}_f\right);\boldsymbol{\theta}_g\right)}_{\hat{\mathbf{x}}}\right\|^2\right]$$

Это называется шумоподавляющим автокодировщиком (DAE).

Можно рассматривать добавление шума к входным данным как способ регуляризации автокодировщика (регуляризация путем введения шума), но с шумоподавляющими автокодировщиками связан другой важный результат.



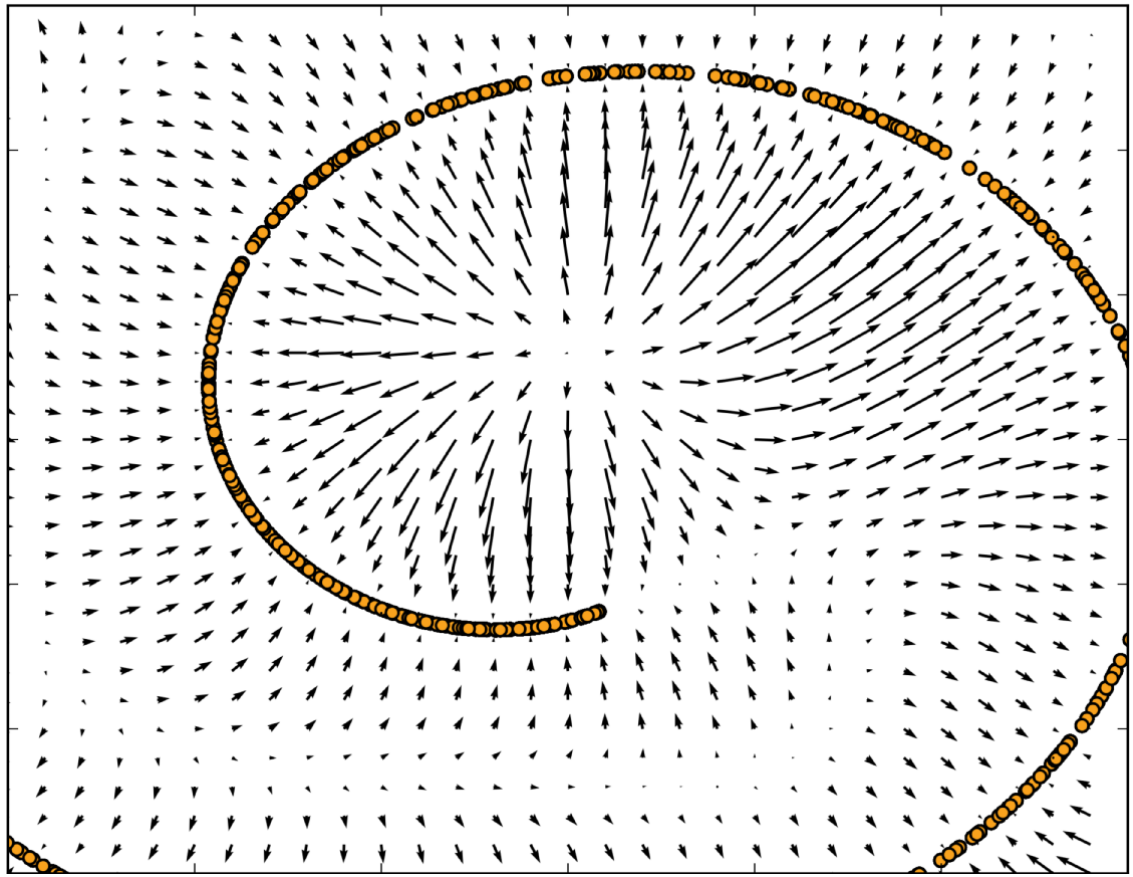
Что изучают шумоподавляющие автокодировщики?

Для Гауссового искажения $\varepsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$ оптимальное шумоподавление равно

$$d\left(\tilde{\mathbf{x}}\right) = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p\left(\tilde{\mathbf{x}}\right)$$

Функция $d\left(\cdot\right)$ учится указывать на более высокую плотность вероятности.

Таким образом, изучая оптимальную функцию шумоподавления $d\left(\mathbf{x}\right)$, мы неявно моделируем распределение данных $p\left(\mathbf{x}\right)$.



Применение автоэнкодеров для классификации

Кодировщик автоэнкодера может быть использован для извлечения признаков при решении задачи классификации. Ключевое предположение здесь состоит в том, что экземпляры с одной и той же меткой должны соответствовать некоторому скрытому представлению, которое может быть аппроксимировано кодировщиком. Для этого автоэнкодер обучается на неразмеченных данных, затем декодер убирается, а кодировщик используется как первая часть модели классификации.

Также автоэнкодеры могут быть использованы в качестве метода регуляризации сети классификации. Например, к кодировщику могут быть подключены две нейронные сети: сеть классификации (обученная на размеченных данных) и сеть декодера (обученная реконструировать данные). Сеть декодера служит для регуляризации сети классификации, а именно, используются потери

$$\tilde{L} = L\left(\mathbf{y}, \hat{\mathbf{y}}\right) + \lambda R\left(\mathbf{x}, \hat{\mathbf{x}}\right),$$

где $R\left(\mathbf{x}, \hat{\mathbf{x}}\right)$ – ошибка реконструкции, $L\left(\mathbf{y}, \hat{\mathbf{y}}\right)$ – ошибка классификации.

Применение автоэнкодеров для кластеризации

Кластеризация — это задача обучения без учителя (без откликов), целью которой является разделение данных на группы таким образом, чтобы выборки в каждой группе были похожи друг на друга и отличались от выборок в других группах. Большинство алгоритмов кластеризации чувствительны к размерам данных и страдают от «проклятия размерности». Предполагая, что данные имеют какое-то низкоразмерное скрытое представление, можно использовать автокодировщики для расчета таких представлений данных, которые состоят из гораздо меньшего количества признаков. Сначала автоэнкодер обучается, затем декодер убирается, как и при классификации. Скрытое представление (выходные данные кодировщика) каждой точки данных затем сохраняется и служит входными данными для любого алгоритма кластеризации (например, алгоритма K-средних). Основным недостатком использования обычных автоэнкодеров для кластеризации является то, что вложение в латентное пространство обучается исключительно для реконструкции, а не для кластеризации.

Автоэнкодеры для обнаружения аномалий

Обнаружение аномалий — это задача обучения без учителя, цель которой состоит в том, чтобы изучить нормальный профиль на основе только экземпляров нормальных данных, а затем идентифицировать экземпляры, не соответствующие нормальному профилю, как аномалии. Этот подход можно применять в различных приложениях, таких как обнаружение мошенничества, мониторинг системы и т. д. Использование автокодировщиков для этих задач основано на предположении, что обученный автокодировщик сможет изучить скрытое подпространство обычных образцов. После обучения это приведет к низкой ошибке реконструкции для нормальных экземпляров и высокой ошибке реконструкции для аномалий.

Автоэнкодеры для рекомендательных систем

Рекомендательная система — это модель, которая стремится предсказать предпочтения или склонность пользователей к товарам. Рекомендательные системы широко используются на веб-сайтах электронной коммерции, в магазинах приложений, у поставщиков онлайн-контента и во многих других коммерческих приложениях. Классическим подходом в моделях рекомендательных систем является коллаборативная фильтрация (CF). В коллаборативной фильтрации предпочтения пользователя выводятся на основе информации о предпочтениях других пользователей. В CF предположение заключается в том, что человеческие предпочтения сильно коррелируют, то есть люди, которые демонстрировали схожие предпочтения в прошлом, будут демонстрировать аналогичные предпочтения и в будущем.

Базовым примером использования автоэнкодеров для рекомендательных систем является модель AutoRec, имеющая два варианта: AutoRec на основе пользователя (U-AutoRec) и AutoRec на основе товаров (I-AutoRec). В U-AutoRec автоэнкодер изучает низкоразмерное представление предпочтений элементов для конкретных пользователей, тогда как в I-AutoRec автоэнкодер изучает низкоразмерное представление предпочтений пользователя для конкретных товаров.

Например, предположим, что в наборе данных M пользователей и N товаров. Пусть $\mathbf{r}_m \in \mathbb{R}^N$ — это (частично наблюдаемый) вектор предпочтений пользователя m, состоящий из его оценок предпочтений каждого из N товаров. Кодировщик U-AutoReco — это отображение $\mathbf{z} = f(\mathbf{r}_m)$, отображающее $\mathbf{r}_m$ в вектор представления $\mathbf{z} \in \mathbb{R}^d$, где $d << N$. Реконструкция при помощи декодера $g(\mathbf{z})$ – это $h(\mathbf{r}_m; \theta) = g(f(\mathbf{r}_m))$, где θ – параметры модели. Целевая функция U-AutoRec определяется как

$$\sum_{m=1}^{M} \|\mathbf{r}_m - h(\mathbf{r}_m; \theta)\|_O^2 + \lambda R(\theta)$$

Здесь норма $\|\cdot\|_O$ означает, что потери определяются только на основе наблюдаемых предпочтений пользователя. Во время прогнозирования мы можем исследовать вектор реконструкции и найти товары, которые, скорее всего, предпочтет пользователь.

Модель I-AutoRec определяется симметричным образом. Пусть $\mathbf{r}_n \in \mathbb{R}^M$ — это (частично наблюдаемый) вектор предпочтений товара n различными пользователями. Целевая функция U-AutoRec определяется как

$$\sum_{n=1}^{N} \|\mathbf{r}_n - h(\mathbf{r}_n; \theta)\|_O^2 + \lambda R(\theta)$$

Во время прогнозирования мы восстанавливаем вектор предпочтений для каждого товара и ищем потенциальных пользователей с высокими прогнозируемыми предпочтениями.

Базовая модель AutoRec может быть расширена за счет включения методов шумоподавления и включения дополнительной информации о пользователях и товарах, таких как демографические данные пользователя или описание товара. Дополнительная информация повышает точность и ускоряет процесс обучения. Удаление шума служит служит в качестве регуляризации, которая не позволяет автокодировщику переобучаться на редких векторах предпочтений, которые не совпадают с общими предпочтениями пользователей.

Итоговый тест по дисциплине

Вопросы итогового теста (2023 года):

1. Теоретический вопрос (4 балла)
2. Задание на регрессию (4 балла)
3. Задание на обратное дифференцирование (4 балла)
4. Задание на сети MLP (4 балла)
5. Задание на сети CNN (4 балла)

ВСЕГО: 20 баллов

Итоговый тест – задание на регрессию (1)

Дан набор данных с двумя числовыми признаками (независимой переменной $\mathbf{X}$ и зависимой переменной $\mathbf{Y}$)

$$\mathbf{D} = \{(1., 2.), (3., 4.), (5., 5.), (7., 5.)\}$$

Найти параметры a и b парной линейной регрессии

$$y(x) = a\,x + b + \varepsilon$$

Решение:

Параметры парной линейной регрессии вычисляются по формулам

$$a = \frac{\sigma_{\mathbf{XY}}}{\sigma^2_{\mathbf{X}}},\ b = \mu_{\mathbf{Y}} - a\,\mu_{\mathbf{X}},$$

где

- $\mu_{\mathbf{X}} = \frac{1}{n}\sum_{i=1}^{n} x_i$, $\mu_{\mathbf{Y}} = \frac{1}{n}\sum_{i=1}^{n} y_i$ – средние значения
- $\sigma^2_{\mathbf{X}} = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu_{\mathbf{X}})^2$ – дисперсия независимой переменной $\mathbf{X}$
- $\sigma_{\mathbf{XY}} = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu_{\mathbf{X}})(y_i - \mu_{\mathbf{Y}})$ – ковариация $\mathbf{X}$и $\mathbf{Y}$

Итоговый тест – задание на регрессию (2)

Поэтому

$$\mu_{\mathbf{X}} = \frac{1}{4}(1+3+5+7) = 4,\ \mu_{\mathbf{Y}} = \frac{1}{4}(2+4+5+5) = 4,$$

$$\sum_{i=1}^{4}(x_i - \mu_{\mathbf{X}})(y_i - \mu_{\mathbf{Y}}) = (1-4)(2-4) + (3-4)(4-4) + (5-4)(5-4) +$$

$$+ (7-4)(5-4) == (-3)(-2) + 0 + 1 + 3 \cdot 1 = 10,$$

$$\sum_{i=1}^{4}(x_i - \mu_{\mathbf{X}})^2 = (1-4)^2 + (3-4)^2 + (5-4)^2 + (7-4)^2 = 9+1+1+9 = 20,$$

$$a = \frac{\sum_{i=1}^{4}(x_i - \mu_{\mathbf{X}})(y_i - \mu_{\mathbf{Y}})}{\sum_{i=1}^{4}(x_i - \mu_{\mathbf{X}})^2} = \frac{10}{20} = 0.5,$$

$$b = \mu_{\mathbf{Y}} - a\,\mu_{\mathbf{X}} = 4 - 0.5 \cdot 4 = 2$$

Итоговый тест – обратное дифференцирование (1)

Вычислить градиент функции $f(x_1, x_2) = e^{x_1 x_2} + x_2^3$ по переменным x_1, x_2 при помощи алгоритма обратного дифференцирования.

Решение:

Функция $f(x_1, x_2)$ может быть вычислена при помощи следующей последовательности элементарных операций и функций:

$$w_1 = x_1,\ w_2 = x_2 \Rightarrow w_3 = w_1 w_2,\ w_4 = w_2^3 \Rightarrow w_5 = e^{w_3} \Rightarrow w_6 = w_4 + w_5$$

или

$$y = f(x_1, x_2) = e^{x_1 x_2} + x_2^3 = e^{w_1 w_2} + w_2^3 = e^{w_3} + w_4 = w_5 + w_4 = w_6$$

Таким образом, имеют место следующие прямые зависимости между переменными w_i:

- w_3 непосредственно зависит от w_1 и w_2
- w_4 непосредственно зависит от w_2
- w_5 непосредственно зависит от w_3
- w_6 непосредственно зависит от w_4 и w_5

Итоговый тест – обратное дифференцирование (2)

Иначе, одни переменные w_i непосредственно влияют на другие переменные w_j следующим образом:

- переменная w_5 непосредственно влияет на w_6
- переменная w_4 непосредственно влияет на w_6
- переменная w_3 непосредственно влияет на w_5
- переменная w_2 непосредственно влияет на w_3 и w_4
- переменная w_1 непосредственно влияет на w_3

При обратном дифференцировании в цепном правиле

$$\frac{\partial y}{\partial w_j} = \sum_i \frac{\partial w_i}{\partial w_j} \frac{\partial y}{\partial w_i} = \frac{\partial w_1}{\partial w_j} \frac{\partial y}{\partial w_1} + \frac{\partial w_2}{\partial w_j} \frac{\partial y}{\partial w_2} + \ldots$$

остаются слагаемые только для тех переменных w_i, которые непосредственно зависят от переменной w_j, т.е. когда $\frac{\partial w_i}{\partial w_j} \neq 0$.

Поэтому получаем

$$\overline{w}_6 = \frac{\partial y}{\partial w_6} = [y \equiv w_6] = \frac{\partial y}{\partial y} = 1$$

$$\overline{w}_5 = \frac{\partial y}{\partial w_5} = \frac{\partial w_6}{\partial w_5}\frac{\partial y}{\partial w_6} = 1 \cdot \frac{\partial y}{\partial w_6} = \overline{w}_6$$

$$\overline{w}_4 = \frac{\partial y}{\partial w_4} = \frac{\partial w_6}{\partial w_4}\frac{\partial y}{\partial w_6} = 1 \cdot \frac{\partial y}{\partial w_6} = \overline{w}_6$$

$$\overline{w}_3 = \frac{\partial y}{\partial w_3} = \frac{\partial w_5}{\partial w_3}\frac{\partial y}{\partial w_5} = e^{w_3} \cdot \frac{\partial y}{\partial w_5} = e^{w_3}\overline{w}_5$$

$$\overline{w}_2 = \frac{\partial y}{\partial w_2} = \frac{\partial w_3}{\partial w_2}\frac{\partial y}{\partial w_3} + \frac{\partial w_4}{\partial w_2}\frac{\partial y}{\partial w_4} = w_1\frac{\partial y}{\partial w_3} + 3w_2^2\frac{\partial y}{\partial w_4} = w_1\overline{w}_3 + 3w_2^2\overline{w}_4$$

$$\overline{w}_1 = \frac{\partial y}{\partial w_1} = \frac{\partial w_3}{\partial w_1}\frac{\partial y}{\partial w_3} = w_2\frac{\partial y}{\partial w_3} = w_2\overline{w}_3$$

Итоговый тест – обратное дифференцирование (4)

Итак,

$$\overline{w}_6 = \frac{\partial y}{\partial w_6} = 1$$

$$\overline{w}_5 = \frac{\partial y}{\partial w_5} = \overline{w}_6$$

$$\overline{w}_4 = \frac{\partial y}{\partial w_4} = \overline{w}_6$$

$$\overline{w}_3 = \frac{\partial y}{\partial w_3} = e^{w_3}\overline{w}_5$$

$$\overline{w}_2 = \frac{\partial y}{\partial w_2} = w_1\overline{w}_3 + 3w_2^2\overline{w}_4$$

$$\overline{w}_1 = \frac{\partial y}{\partial w_1} = w_2\overline{w}_3$$

Поэтому градент функции $f(x_1, x_2)$, вычисленный при помощи обратного дифференцирования, равен

$$\frac{\partial y}{\partial x_1} = \overline{w}_1 = w_2\overline{w}_3 = x_2 e^{x_1 x_2}, \; \frac{\partial y}{\partial x_2} = \overline{w}_2 = x_1 e^{x_1 x_2} + 3x_2^2$$

Итоговый тест – сети MLP (1)

Дана сеть MLP с двумя входными нейронами, скрытым слоем с тремя нейронами с функцией активации ReLU и одним нейроном в выходном слое без активации (с тождественной функцией активации). Матрица весов и вектор смещений для скрытого слоя равны

$$\mathbf{W}_h = \begin{pmatrix} 1 & -2 & -1 \\ 1 & 1 & -2 \end{pmatrix}, \mathbf{b}_h = \begin{pmatrix} 2 \\ -2 \\ -1 \end{pmatrix},$$

вектор весов и смещение для выходного слоя равны

$$\mathbf{W}_o = \begin{pmatrix} 1 \\ 3 \\ -1 \end{pmatrix}, \mathbf{b}_o = -1$$

Вычислить выход нейронной сети MLP для входных данных $\mathbf{x} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$.

Итоговый тест – сети MLP (2)

Решение:

Чистые входы нейронов скрытого слоя равны

$$\mathbf{W}_h^T\mathbf{x} + \mathbf{b}_h = \begin{pmatrix} 1 & 1 \\ -2 & 1 \\ -1 & -2 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \end{pmatrix} + \begin{pmatrix} 2 \\ -2 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ -4 \\ 3 \end{pmatrix} + \begin{pmatrix} 2 \\ -2 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ -6 \\ 2 \end{pmatrix}$$

Выходы нейронов скрытого слоя равны

$$\mathbf{z} = \mathrm{ReLU}\left(\begin{pmatrix} 1 \\ -6 \\ 2 \end{pmatrix} \right) = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$$

Тогда выход нейронной сети равен

$$\mathbf{o} = \mathbf{W}_o^T\mathbf{z} + \mathbf{b}_o = \begin{pmatrix} 1 & 3 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} - 1 = -1 - 1 = -2$$

Итоговый тест – сети CNN (1)

Дан фрагмент (два слоя) сверточной нейронной сети CNN, состоящий из слоя свертки с фильтром $\mathbf{W}$, смещением b, функцией активации ReLU и шагом 1 и слоем макс-пулинга с размером окна 2×2 и шагом 1. Фильтр и смещение для слоя свертки равны

$$\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 0 \end{pmatrix}, b = 1$$

Вычислить выход слоя макс-пулинга для входных данных слоя свертки

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 2 & 1 & 1 \\ 3 & 1 & 4 & 2 & 1 \\ 2 & 1 & 3 & 4 & 3 \\ 1 & 2 & 3 & 1 & 1 \\ 4 & 1 & 3 & 2 & 1 \end{pmatrix}$$

Итоговый тест – сети CNN (2)

Решение:

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 2 & 1 & 1 \\ 3 & 1 & 4 & 2 & 1 \\ 2 & 1 & 3 & 4 & 3 \\ 1 & 2 & 3 & 1 & 1 \\ 4 & 1 & 3 & 2 & 1 \end{pmatrix}, \mathbf{W} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 0 \end{pmatrix}, b = 1$$

Свертка $\mathbf{X}$ и $\mathbf{W}$ с шагом 1 дает матрицу

$$\mathbf{X} * \mathbf{W} = \begin{pmatrix} -3 & 1 & -1 \\ -1 & -3 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

Чистые входы нейронов слоя свертки равны

$$\mathbf{X} * \mathbf{W} + b = \begin{pmatrix} -3 & 1 & -1 \\ -1 & -3 & 4 \\ 0 & 0 & 1 \end{pmatrix} + 1 = \begin{pmatrix} -2 & 2 & 0 \\ 0 & -2 & 5 \\ 1 & 1 & 2 \end{pmatrix}$$

Итоговый тест – сети CNN (3)

Выходы нейронов слоя свертки равны

$$\mathbf{z}_c = \mathrm{ReLU}\left(\mathbf{X} * \mathbf{W} + b\right) = \mathrm{ReLU}\left(\begin{pmatrix} -2 & 2 & 0 \\ 0 & -2 & 5 \\ 1 & 1 & 2 \end{pmatrix}\right) = \begin{pmatrix} 0 & 2 & 0 \\ 0 & 0 & 5 \\ 1 & 1 & 2 \end{pmatrix}$$

Пулы 2×2 с шагом 1 для матрицы выходов нейронов слоя свертки равны

$$\left(\begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 5 \\ 1 & 2 \end{pmatrix}\right),$$

поэтому выход слоя макс-пулинга равен

$$\mathbf{z}_p = \begin{pmatrix} 2 & 5 \\ 1 & 5 \end{pmatrix}$$