DOI: 10.15514/ISPRAS-2024-36(1)-1

Нейросетевые методы сжатия векторов для задачи приближенного поиска ближайших соседей

[1] *И.О. Буянов*, ORCID: 0009-0000-6994-151X, <*buyanov.igor.o@yandex.ru*>
[1] *В.В. Ядринцев*, ORCID: 0000-0002-4981-2515, <*vvyadrincev@gmail.com*>
[1,2,3,4] *И.В. Соченков,* ORCID: 0000-0003-3113-3765, <*sochenkov@isa.ru*>
[1] *Федеральный исследовательский центр Информатика и Управление РАН,*
119333, Россия, г. Москва, ул. Вавилова, д. 44, кор. 2
[2] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.
[3] *Университет Иннополис,*
420500, Россия, р. Татарстан, г. Иннополис, ул.Университетская, д. 1.
[4] *Сеченовский Университет*
119048, Россия, г. Москва, ул. Трубецкая, д. 8, стр. 2

Аннотация. В статье проверяется гипотеза применимости нейросетевых автокодировщиков как метод векторного сжатия для задачи приближенного поиска ближайших соседей. Проверка проводилась на нескольких больших датасетах с различными архитектурами автокодировщиков и индексов. Она показала, что, хотя ни одна из комбинаций автокодировщиков и индексов не может полностью превзойти чистые решения, в некоторых случаях они могут быть полезными. Мы также выявили некоторые эмпирические связи оптимальной размерности скрытого слоя и внутренней размерности наборов данных. Было также показано, что функция потерь является определяющим фактором качества сжатия.

Ключевые слова: приближенный поиск соседей; автокодировщики; крупномасштабный набор данных.

Для цитирования: Буянов И.О., Ядринцев В.В., Соченков И.В. Нейросетевые методы сжатия векторов для задачи приближенного поиска ближайших соседей. Труды ИСП РАН, том 36, вып. 1, 2024 г., стр. 7–22. DOI: 10.15514/ISPRAS–2024–36(1)–1.

Благодарности: Работа выполнена при поддержке Фонда содействия инновациям (Договор № 12ГУКодИИС12-D7/72692 о предоставлении гранта на выполнение проекта открытых библиотек от 27 декабря 2021 г).

Neural Vector Compression In Approximate Nearest Neighbor Search On Large Datasets

[1] *I.O. Buyanov*, ORCID: 0009-0000-6994-151X, <*buyanov.igor.o@yandex.ru* >
[1] *V.V. Yadrinsev*, ORCID: 0000-0002-4981-2515, <*vvyadrincev@gmail.com* >
[1,2,3,4] *I.V. Sochenkov*, ORCID: 0000-0003-3113-3765, <*sochenkov@isa.ru* >
[1] *Federal Research Center "Computer Science and Control" of the Russian,*
44, Vavilov st., Moscow, 119333, Russia
[2] *Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
[3] *Innopolis University,*
1, Universitetskaya st., Innopolis, 420500, Russia.
[4] *Sechenov University,*
8, Trubetskaya st., Moscow, 119048, Russia.

Abstract. The paper examines the hypothesis of the applicability of neural autoencoders as a method of vector compression in the pipeline of approximate nearest neighbor search. The evaluation was conducted on several large datasets using various autoencoder architectures and indexes. It has been demonstrated that, although none of the combinations of autoencoders and indexes can fully outperform pure solutions, in some cases, they can be useful. Additionally, we have identified some empirical relationships between the optimal dimensionality of the hidden layer and the internal dimensionality of the datasets. It has also been shown that the loss function is a determining factor for compression quality.

Keywords: approximate nearest neighbor search; autoencoders; large datasets.

For citation: Buyanov I.O., Yadrinsev V.V., Sochenkov I.V. Neural vector compression in Approximate Nearest Neighbor Search on Large Datasets. *Trudy ISP RAN/Proc. ISP RAS*, vol. 36, issue 1, 2024. pp. 7-22 (in Russian). DOI: 10.15514/ISPRAS-2024-36(1)-1.

Acknowledgements. We thank Foundation for Assistance to Small Innovative Enterprises for funding this work (Agreement №12GUKodIIS12-D7/72692).

1. Введение

В настоящее время объем данных, доступный в Интернете, растет экспоненциально. Этот рост создает потребность в эффективных методах поиска информации, способных справиться с такими объемами. Информационный поиск (Information Retrieval, IR) – это процесс поиска и получения информации, соответствующей потребностям пользователя. Он включает в себя как поиск, так и фильтрацию больших объемов данных, чтобы представить наиболее релевантные результаты. Это особенно важно в таких областях, как научные исследования, где точность и актуальность полученной информации могут существенно повлиять на результативность проекта. К тому же экспоненциальный рост объема данных ведет к усложнению задачи поиска релевантной информации из-за постоянного ее роста.

Одной из наиболее активных областей исследований в области информационного поиска является задача приближенного поиска ближайших соседей (Approximate Nearest Neighbor, ANN). Она возникает в большом числе приложений: от систем рекомендаций до компьютерного зрения, где целью является поиск ближайших соседей для запроса в большой базе векторов, где точные алгоритмы поиска ближайших соседей могут быть вычислительно затратными и непрактичными для больших наборов данных. Это требует от исследователей поиска приближенных алгоритмов, которые могут возвращать приближенное решение с приемлемой точностью. Данные методы обычно включают построение такой структуры данных, как дерево или граф для эффективного сужения пространства поиска и избегания проверки всех возможных кандидатов. Другим возможным направлением является исследование различных методов сжатия векторов.

В частности, проблема с большими наборами данных возникает из-за того, что общая размерность векторов варьируется от сотен до нескольких тысяч. Это ведет к высоким требованиям к размеру хранилища, а также к оперативной памяти. Несмотря на то, что графические процессоры позволяют проводить быстрые матричные вычисления высокой размерности, способы сжатия векторного представления для экономии ресурсов представляют практический интерес.

В данной работе мы исследуем нейронные сети с архитектурой автокодировщика в качестве компрессора векторов для поиска ближайших соседей. Вероятно, они могут расцениваться неподходящими для этой задачи в связи с высокими вычислительными затратами. Тем не менее мы решили исследовать, какие возможные преимущества могут принести автокодировщики в роли векторного компрессора.

В статье мы проверяем гипотезу об использовании автокодировщиков в качестве метода сжатия векторов путем проведения обширных тестов различных методов приближенного поиска ближайших соседей с использованием нескольких автокодировщиков на нескольких наборах данных. Мы также покажем, что существует граница сжатия, пересечение которой на некоторых алгоритмах поиска вызывает быстрое падение качества независимо от типа автокодировщиков, показывающих определенный уровень качества до этой границы. Мы также показываем эмпирически, как это может быть связано с внутренней размерностью данных.

2. Смежные работы

Первые алгоритмы поиска были основаны на древовидных методах, таких как KD-tree [1] и Ball tree [2], которые организовывают данные в древовидную структуру для результативного поиска. Эти методы широко применяются для приближенного поиска ближайших соседей в пространствах низкой и средней размерности, где данные могут быть эффективно организованы в такую структуру. Предполагается несколько вариантов древовидных методов для улучшения их производительности, включая использование случайных KD-trees и иерархической кластеризации. Эти методы эффективны и действенны для данных с более низкой размерностью.

Как было упомянуто ранее, существует множество исследований по методам сжатия. Один из наиболее популярных методов сжатия – квантизация произведений (product quantization, PQ) [3]. Метод подразумевает разделение исходного пространства данных на подпространства и последующую их квантизацию независимо друг от друга. Метод показал высокую эффективность для высокоразмерных данных и применяется в таких областях, как поиск изображений и распознавание речи.

Еще одной значимой группой алгоритмов являются графовые методы. Например, NSW (Navigable Small World) [4] использует структуру сети "малого мира" для эффективной «навигации» по данным. Суть этой структуры заключается в поддержании баланса между локальными и глобальными связями. Этот метод доказал свою эффективность для высокоразмерных данных и используется в таких областях, как поиск изображений и поиск текста. HNSW [5] (Hierarchical Navigable Small World) является расширением NSW и использует иерархическую структуру, что улучшает эффективность исходного подхода. Графы HNSW разделяют данные на несколько уровней, каждый из которых содержит графы разного масштаба. Этот подход можно сравнить с почтой: сначала письмо пересылается между странами (граф стран), затем между городами (граф городов) и затем уже между городскими отделениями (граф отделений). Таким образом метод позволяет с высокой точностью осуществлять поиск по данным на разных масштабах и повышает общую эффективность приближенного поиска ближайших соседей. HNSW показал свою эффективность для высокоразмерных данных и применяются в таких областях, как поиск изображений и системы рекомендаций.

Еще одно направление исследований, которое также смежно с нашим, – это создание векторных представлений, потенциально пригодных для приближенного поиска ближайших соседей. Например, в работе [6] авторы предлагают обучаемый слой индексирования векторных представлений. Вместе со специальной регуляризации функции потерь, этот слой может быть встроен в любую глубокую нейросеть для задачи информационного поиска, позволяя совместно обучать векторные представления и поисковый индекс для них.

3. Проблема поиска ближайших соседей

Поиск ближайших соседей является основной проблемой во многих приложениях машинного обучения и анализа данных, где целью является поиск k ближайших соседей заданной точки запроса в большом наборе данных. Пусть $X = \{x_1, x_2, \ldots, x_n\}$ – набор из n векторов в d-мерном пространстве, а q – вектор запроса в d-мерном пространстве. K ближайших соседей q в X определяются как k векторов x_i в X, минимизирующих функцию расстояния $dist(q, x_i)$:

$$min_{\{i=1,2,..,k\}} dist(q, x_i)$$

где $dist(q, x_i)$ – функция расстояния, которая измеряет сходство или различие между вектором запроса q и вектором базы данных x_i. В данном случае используется евклидово расстояние.

Прямой подход, вычисляющий расстояния между точкой запроса и каждой точкой в наборе данных, имеет временную сложность $O(nd)$, что является непрактичным для больших наборов данных. Алгоритмы ANN стремятся найти приближенный набор из k ближайших соседей, которые находятся в непосредственной близости к истинным ближайшим соседям, тем самым существенно снижая вычислительную сложность.

Точность алгоритмов ANN обычно измеряется в терминах полноты (recall), которая представляет собой долю найденных истинных ближайших соседей, относительно всех истинных ближайших соседей. Пусть $NN(q)$ обозначает множество истинных k ближайших соседей вектора q в X. Полнота $R@k$ алгоритма ANN, который возвращает множество k векторов $S(q)$ для данной точки запроса q, определяется следующим образом:

$$R@k = \frac{|S(q) \cap NN(q)|}{|NN(q)|}$$

где $|S(q) \cap NN(q)|$ – количество элементов в пересечении множеств $S(q)$ и $NN(q)$, а $|NN(q)|$ – количество элементов в множестве $NN(q)$.

Эффективность алгоритмов ANN измеряется по времени запроса, то есть времени, необходимого для поиска k ближайших соседей для данной точки запроса. Целью алгоритмов ANN является достижение разумного соотношения между точностью и эффективностью за счет баланса между качеством приближенного решения и вычислительной сложностью его нахождения.

4. Использование автокодировщика для сжатия набора данных

Нейронные автокодировщики [7] представляют собой тип нейронных сетей, используемых для обучения без учителя. Они состоят из кодирующей (энкодера) и декодирующей (декодера) частей. Цель заключается в восстановлении входных данных на выходе. Энкодер отображает входные данные в скрытое пространство меньшей размерности, в то время как декодер отображает векторы из скрытого пространства обратно в исходное. Мы полагаем, что возможно использовать энкодер для снижения размерности векторизованных наборов данных.

Формально пусть $\boldsymbol{x}$ будет входным вектором, а $\boldsymbol{h}$ – вектор в скрытом пространстве. Энкодер отображает $\boldsymbol{x}$ в $\boldsymbol{h}$ с помощью функции $f(\boldsymbol{x}) = \boldsymbol{h}$. Аналогично, пусть $\mathbf{y}$ будет выходным вектором, а $\mathbf{g}$ – функция, отображающая $\mathbf{h}$ обратно в $\mathbf{y}$, $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{h})$. Цель автокодировщика

состоит в минимизации ошибки восстановления (the reconstruction loss), которая является разницей между входными данными **x** и выходными данными **y**. Одним из распространенных способов измерения ошибки восстановления является среднеквадратическая ошибка (MSE):

$$L(x, y) = \frac{1}{n}\sum_{\{i=1\}}^{\{n\}}(\boldsymbol{x}_i - \boldsymbol{y}_i)^2$$

где n – количество векторов на входе. Основное свойство, которым должен обладать автокодировщик для задачи поиска ближайших соседей, – сохранять отношения между векторами при преобразовании векторного пространства. В экспериментах мы используем несколько модификаций описанного автокодировщика, который мы будем называть Vanilla. Полагаем, что модифицированные варианты сохраняют отношения между точками лучше, чем Vanilla.

Для начала рассмотрим автокодировщик "The Neighborhood Reconstructing Autoencoders" [8], идея которого заключается в использовании аппроксимации функции декодера с помощью локальных графов. Эти графы отражают локальную геометрию распределения данных, что позволяет сделать автокодировщик более устойчивым к переобучению и проблемам связности. Для этого метода строится полностью связанный граф соседства, который используется в функции потерь:

$$L = \sum_i \sum_{\{x \in N(x_i)\}} ||x - f_\theta(g_{\phi(x)}; g_{\phi(x_i)})||$$

где $N(x_i)$ – множество соседних точек для x_i, а $\tilde{f}(.; g_\phi(x_i))$ – аппроксимация декодера $f_\phi(x)$ относительно закодированной точки $g_\phi(x)$.

Другая модификация – DCEC [9] (Deep Clustering with Convolutional Autoencoder), где в архитектуру автокодировщика Vanilla был добавлен слой кластеризации после энкодера. Этот слой является обучаемыми центроидами для кластеров в данных, что, как мы считаем, позволит проводить более детальное разделение векторного пространства и, как следствие, улучшит качество алгоритмов поиска, не основанных на графе. Математически слой кластеризации, как было сказано выше, представляет собой центроиды кластеров в виде обучаемых весов и отображает каждый скрытый вектор z_i в нестрогую метку q_i, которая рассчитывается следующим образом:

$$q_{\{ij\}} = \frac{\left(1 + \left|\left|z_i - \mu_i\right|\right|^2\right)^{-1}}{\sum_j \left(1 + \left|\left|z_j - \mu_j\right|\right|^2\right)^{-1}}$$

Функция потерь кластеризации определяется как: $L_c = KL(P||Q)$ где P – целевое распределение. Авторы предлагают эвристическое целевое распределение (см. уравнение 8). Тем не менее распределение может быть любым, которое (1) способно создавать нестрогие метки, (2) улучшать чистоту (purity) кластеров, (3) обращать внимание на точки с высокой уверенностью и (4) нормализовывать влияние центроидов, чтобы предотвратить искажение векторного пространства. Наконец, функция потерь кластеризации добавляется к функции потерь восстановления с помощью гиперпараметра Y, который контролирует ее влияние:

$$L = L_{rec} + \gamma L_c$$

Наконец, мы рассмотрим гиперболический автокодировщик (Hyperbolic Autoencoder [10]), работающий в гиперболическом пространстве. Авторы этого подхода используют модель сферы Пуанкаре, которая определяется как $B^n = \{x \in R^n: ||x||\}$ с метрическим тензором Римана. Особенностью этого автокодировщика является метрика расстояния между двумя точками (см. уравнение 2), которая делает расстояния близкимит в евклидовом пространстве экспоненциально большими в гиперболическом пространстве. Это позволяет эффективно моделировать сложные сети и структуры похожие на деревья.

5. Описание наборов данных

Чтобы сделать наш тест достаточно обширным, мы используем шесть наборов данных, основные статистические показатели которых приведены в табл. 1. Описание используемых датасетов:

- SIFT-Small Dataset [3]: компактная версия набора данных SIFT-1M, который состоит из 10 000 дескрипторов SIFT, извлеченных из набора изображений.
- SIFT-1M Dataset: один миллион дескрипторов SIFT, извлеченных из набора изображений. Обычно используется в качестве стандарта для оценки производительности алгоритмов поиска ближайших соседей.
- GIST Dataset [3]: набор данных из 1 миллиона дескрипторов GIST. Другой часто используемый эталон для алгоритмов ANN.
- GIST Dataset 1B: такой же, как датасет GIST, но с количеством в один миллиард дескрипторов.
- Wiki-LASER Dataset: набор данных, который мы создали на основе статей из Википедии. Мы разбили каждый текст на предложения и закодировали их с использованием метода LASER (Language Agnostic Sentence Representations) [11].
- Wiki-LASER Dataset Small: уменьшенная версия датасета Wiki-LASER Dataset.
- Open Images Dataset (OID) [12]: набор данных, который содержит более 9 миллионов изображений с более чем 30 миллионами ограничительных рамок. Набор разработан задач обнаружения объектов, классификации и обнаружения визуальных отношений. Мы используем сеть ResNet18 для получения векторных представлений изображений, которые затем были нормализованы и преобразованы в единый вид.
- Open Images Dataset small (OID small): уменьшенная версия датасета OID.

Табл. 1. Сводная статистика наборов данных. BPC означает "байт на компонент"
Table 1. Statistic of the datasets. BPC means "byte per component"

Название	Размерность	Vec cnt	BPC	Query vec cnt	Learn vec cnt	Orig size
siftsmall	128	10,000	4	100	25,000	5,120,000
sift1m	128	1,000,000	4	10000	100,000	512,000,000
gist	960	1,000,000	4	1000	500,000	3,840,000,000
sift1b	128	1,000,000,000	1	10000	100,000,000	128,000,000,000
OID small	512	1,012,239	4	1012	101,223	2,073,065,472
OID	512	8,390,600	4	83906	843,480	17,183,948,800
Wiki-LASER Dataset small	1024	576,940	4	576	57,694	2,363,146,240
Wiki-LASER Dataset	1024	87,817,400	4	87783	1,317,261	359,700,070,400

6. Подготовка к тестированию

Для тестирования мы выделили разные флаги и компоненты и составляем из них разные комбинации. Результаты перспективных комбинаций мы свели в таблицы для каждого конкретного набора данных (см. Табл. 3-10 в конце статьи). Таблицы разделены на две части. Первая часть (табл. 3, 5, 7, 9) состоит из результатов, вторая (табл. 4, 6, 8, 10) описывает

некоторые параметры комбинации. Строки могут быть сопоставлены по столбцу ID. Мы используем реализации различных индексов из библиотеки Faiss [13]. Тестовые случаи делятся на две группы: первая группа – это случаи, когда используется автокодировщик в сочетании с индексом и/или квантования из Faiss, а вторая группа – это случаи, когда используются только компоненты Faiss. В качестве индексов мы используем Flat, IVFPQ, HNSW и NSG (столбец index). В качестве автокодировщиков (столбец enc) мы используем все описанные варианты: Vanilla, DCEC, HyperAE, NRAE.

Кроме того, экспериментируем с нормализацией входных и выходных векторов (столбцы norm inp vect, norm embs и norm out vect), скрытой размерностью автокодировщиков (столбец hidden dim) и установкой размерности скрытого слоя, равной размерности входа автокодировщика (столбец set l hidden). Эти параметры представлены в виде столбцов во второй таблице. Если столбец не указан, это означает, что он имеет одно и то же значение в каждой ячейке. По умолчанию отсутствие значения означает то, что действие не было выполнено. Также указываем константы, такие как размер пакета (batch size), равный 8, количество эпох обучения, равное 5, параметр nprobe, равный 20.

Мы вычисляем несколько метрик, таких как различные варианты полноты (recall), которые мы кодируем как $n - R@k$, где n – это количество истинных ближайших соседей, а k – это топ релевантных векторов, возвращенных индексом. Мы также вычисляем коэффициент сжатия (cr), экономию пространства (ss), размер индекса на диске (index size), время поиска одного вектора (o-v-s), общее время поиска (s) и также время обучения и индексации (t_i). Все эксперименты были проведены в два этапа. Сначала на небольшом подмножестве датасета мы ищем лучшие комбинации параметров, затем запускаем отобранные варианты на всем наборе данных.

7. Выбор размерности автокодировщика

Важным моментом является выбор размерности скрытого слоя, который эквивалентен коэффициенту сжатия. Нет четкого способа определить оптимальную размерность скрытого слоя, которая бы давала максимальное сжатие при минимальной потере качества.

В ходе предварительного исследования было обнаружено, что существует граница сжатия, пересечение которой вызывает стремительное снижение качества, независимо от типа автокодировщика. Это характерно для алгоритмов поиска, показывающих определенный уровень качества до достижения этой границы, например, HNSW на рис. 1. Изучение этого явления привело к выводу, что эта граница является внутренней размерностью, которая определяется как размерность многообразия, к которому принадлежит вложенное пространство [14]. По альтернативному определению, это можно рассматривать как минимальное число измерений, необходимое для представления данных.

Мы провели эксперименты на синтетически сгенерированных данных с известной внутренней размерностью. Для генерации мы использовали пакет scikit-dimension [15], в котором воспользовались функцией генерации данных в режиме «Nonlinear manifold». Затем обучили автокодировщик Vanilla на сгенерированном наборе и вычислили функцию потерь на тестовом наборе, отобранном из сгенерированных данных, при этом размер скрытого слоя варьируется от исходной размерности до 10. В качестве внутренних размерностей тестируем значения 8, 32 и 64. На рис. 2 видно, что значение функции потерь начинает расти после того, как размер скрытого слоя становится меньше внутренней размерности. Мы повторили этот эксперимент с реальными данными, в частности с наборами данных SIFT1M и Open Images Dataset. Мы обнаружили, что потери автокодировщика начинают увеличиваться приблизительно в том же диапазоне, в котором ухудшается качество поиска.

Поскольку экспериментально показано существование оптимальной размерность для сжатия, мы исследуем все алгоритмы оценки внутренней размерности, реализованные в пакете scikit-dimension. Суть эксперимента очень проста – подать на вход каждого алгоритма из

программного пакета датасет, для которого мы имеем гипотезу о внутренней размерности, а затем сравнить результат выдачи алгоритма с нашей гипотезой.

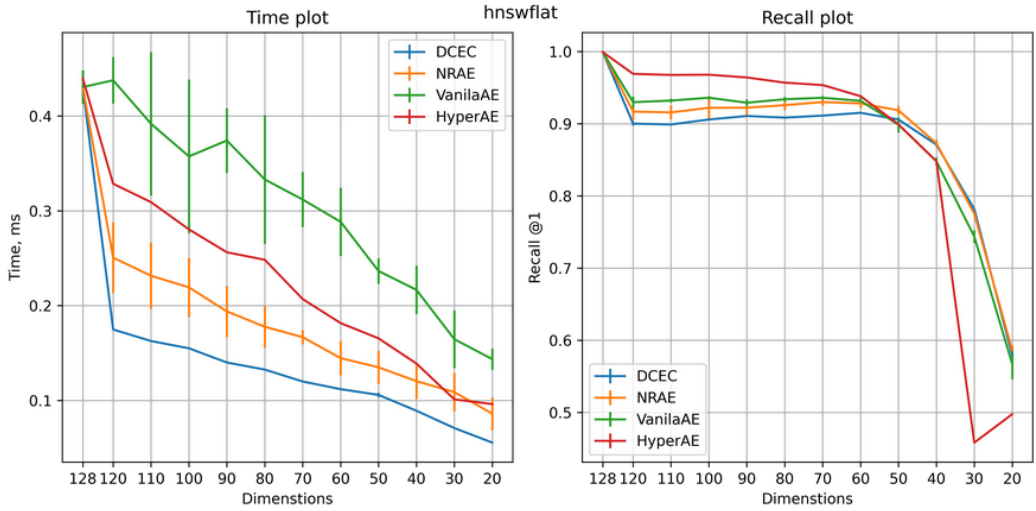

Рис. 1. Тест HNSW с автокодировщиками.
Левый график отображает время вывода. Правый график отображает показатель полноты (recall)
Fig. 1. Test of HNSW with autoencoder.
Left plot shows the inference time, right plot shows performance in terms of recall

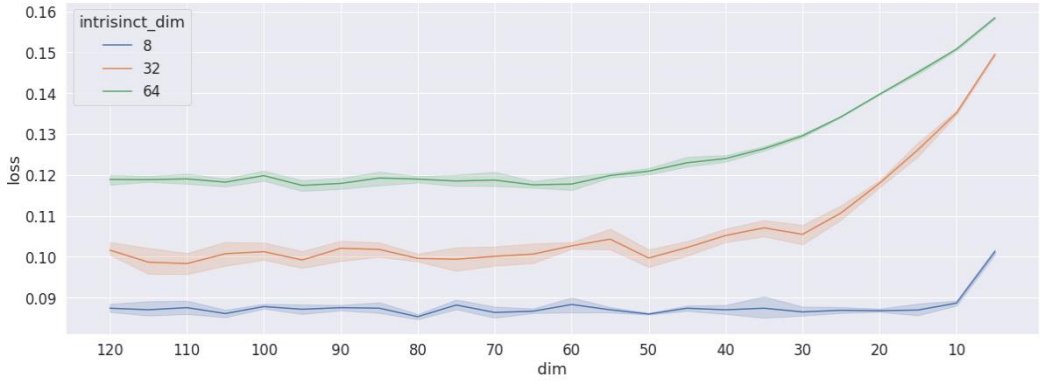

Рис. 2. Тестирование функции потерь автокодировщика Vanilla при различных внутренних размерностях синтетических наборов данных
Fig. 2. Results of the Vanilla autoencoder testing with synthetic datasets with various intrinsic dimensions

Исходя из эмпирического исследования, мы считаем, что внутренняя размерность SIFT1M составляет около 60. Тем не менее все алгоритмы показали значения существенно далекие от 60, как показано в табл. 2 в столбце «Размерность». Кроме того, некоторые алгоритмы требуют значительное время на выполнение, как видно в столбце «Время выполнения».

Кроме того, экспериментально не удалось подтвердить, что полученное нами число является внутренней размерностью. Возможно, это число им не является, а возможно алгоритмы не справились с задачей, учитывая какой большой разброс оценок мы получили. Стоит отметить, что мы не калибровали параметры алгоритмов и запускали их со стандартными значениями. Мы оставляем вопросы автоматического выбора оптимальной размерности, а также природу этой оптимальности для будущих исследований.

Табл. 2. Результаты теста алгоритма оценки внутренней размерности на наборе данных SIFT1M
Table 2. Results of the intrinsic dimension estimation algorithms on SIFT1M dataset

Имя	Размерность	Время выполнения, с
ESS	25.4422	9324.32
DANCo	nan	265.773
CorrInt	9.99273	420.92
FisherS	3.11696	304.551
KNN	3	695.634
lPCA	10	0.88
MADA	21.9273	1956.41
MiND_ML	1	255.691
MLE	0	256.54
MOM	19.5008	253.178
TLE	19.1451	288.2
TwoNN	10.4751	831.291

8. Влияние функции потерь на качество методов поиска

Известно, что функция потерь для обучения во многом определяет свойства и поведение нейронной сети. Для достижения наилучшего качества поиска необходимо, чтобы векторы в сжатом пространстве находились в таких же отношениях, как и в исходном. Можно попытаться сформулировать это требование в виде функции потерь. В работе [16] предлагается функция потерь, которая заставляет автокодировщик напрямую воссоздавать отношения расстояний, сравнивая матрицы расстояний в исходном и сжатом пространствах.

$$L_{distance}(\phi;\, X, Z) \;=\; \sum_{(i,j)\in\, G_X} \left| d_X(x_i, x_j) - \gamma\; d_Z(z_i, z_j) \right|^2$$

Мы провели тестирование этой функции потерь на двух датасетах SIFT1M и GIST. Эмпирически получив для них оптимальную размерность, мы обучили автокодировщики со среднеквадратичной ошибкой (MSE) и вышепредставленной функцией потерь. В качестве методов поиска мы использовали HNSW и IVFPQ. Результаты представлены на рис. 3. Из него видно, что новая функция потерь обеспечивает существенный прирост по качеству для обоих датасетов. Результаты показывают, что выбор функции потерь может существенно влиять на конечный результат. Это тестирование было проведено после получения главных результатов, поэтому дальнейшее изучение дизайна функции потерь мы оставим на будущее.

9. Основные результаты

Как видно из результатов, методы поиска на основе графов NSG и HNSW показывают практически идеальные результаты по всем вариантам полноты для всех наборов данных (табл. 4, 6, 8). Недостатками этих методов является то, что они используют векторы без изменений и добавляют избыточные затраты памяти, что можно видеть по метрикам cs и ss. Они также требуют меньше всего времени при непосредственном использовании и имеют среднюю скорость индексирования.

Ни одна из комбинаций, в которых присутствуют автокодировщики, не показывает результатов, которые полностью превосходят чистую систему Faiss. Можно сделать вывод, что сжатие автокодировщика сильно искажает векторное пространство. Это выражается в нарушении локальных отношений: векторы, являющиеся соседями в исходном пространстве,

могут оказаться далеко друг от друга в сжатом пространстве. При сравнении только автокодировщиков заметно, что один автокодировщик работает лучше других в зависимости от набора данных. Соответственно, можно выделить, что HyperAE работает лучше в большинстве случаев. В отличие от него, NRAE не показал приемлемых результатов ни в одном варианте.

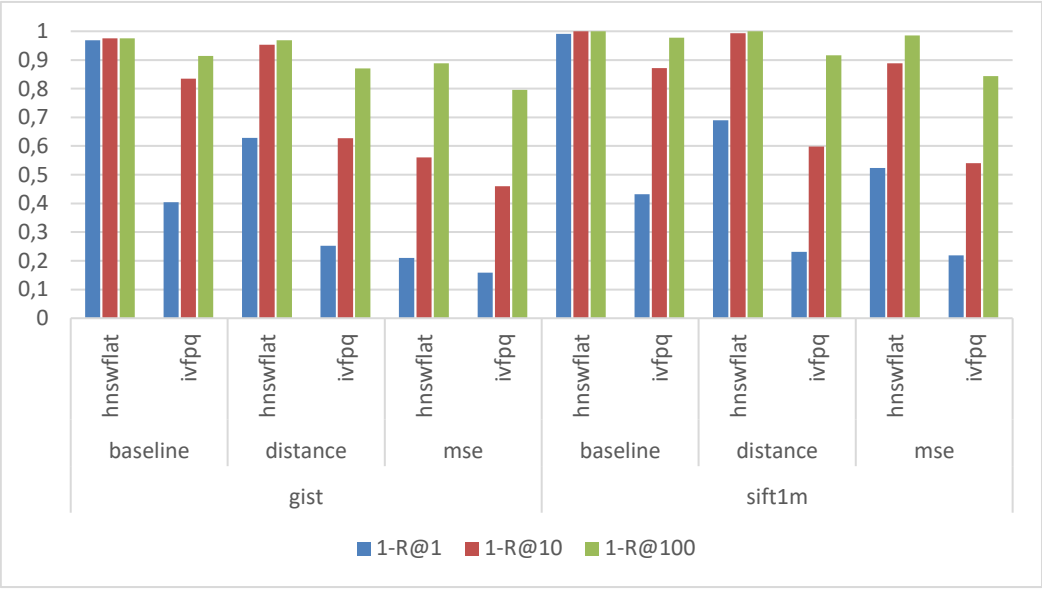

Рис. 3. Сравнение качества методов ANN в зависимости от разных функций потерь (ФП). Результаты без сжатия указаны как baseline

Fig. 3. Results of the influence testing of various losses on ANN performance. Baseline is compressless result

Для наборов данных Wiki и Open Images комбинации показывают идентичные результаты по метрике $1 - R@k$, поэтому возможно сократить сравнение до $10 - R@k$. Можно отметить, что комбинация Vanilla AE с индексом Flat сравнима с HNSW, но при этом обеспечивает двукратное уменьшение размера. Однако скорость работы комбинации с использованием Vanilla AE гораздо медленнее.

10. Заключение

В данной статье мы исследовали применимость нейронных автокодировщиков в качестве компонента сжатия векторов в конвейере поиска соседей. Рассмотрели автокодировщик Vanilla и его варианты, которые лучше сохраняют геометрию пространства при сжатии, с различными методами индексирования.

Мы обнаружили, что автокодировщики не смогли сжать исходное векторное пространство таким образом, чтобы полученное пространство точно сохраняло локальные отношения между векторами, хотя некоторые из них делают это лучше других. В ходе работы также отметили связь между нижней границей сжатия автокодировщщиков и оценкой внутренней размерности. Также показали, что возрастание функции потерь автокодировщика может служить хорошим сигналом о достижении внутренней размерности, в то время как ни один из протестированных алгоритмов ее оценки не может обработать такую высокую размерность.

Эксперименты показали, что использование кодировщиков в некоторых случаях позволяет сохранить вектор меньшей размерности с уверенностью, что выбранная метрика качества

сохраняет тот же уровень. Таким образом, это может дать преимущество, например, в задаче хранения сжатых векторов.

В дальнейшей работе мы будем продолжать искать автокодировщик, который минимально искажает пространство при сжатии данных настолько, насколько это возможно. Как видно из экспериментов, прежде всего необходимо уделить внимание функции потери. У нас есть гипотеза, согласно которой точки из плотных областей интерферируют между собой, что приводит к ошибкам. Если это так, то стоит рассмотреть добавление регуляризатора к функции потерь. Мы также попытаемся разработать способы автоматического выбора размерности автокодировщика.

Табл. 3. Основные результаты для SIFT1M
Table 3. Main results for SIFT1M

id	1-R@100	1-R@10	1-R@1	10-R@100	10-R@10	100-R@100	index_size	cr	ss	o-v s	s	t_i
0	0.56	0.33	0.16	0.38	0.15	0.18	2.56E+08	2	0.5	0.0282	58.2	80.57
1	0.56	0.33	0.16	0.38	0.15	0.17	5.28E+08	0.97	-0.03	0.0058	0.67	98.56
2	1	0.93	0.57	0.98	0.62	0.65	2.56E+08	2	0.5	0.0259	57.75	289.5
3	1	0.94	0.58	0.98	0.63	0.67	2.56E+08	2	0.5	0.0257	57.88	299.66
4	1	0.93	0.56	0.97	0.61	0.65	2.56E+08	2	0.5	0.0283	57.87	108.23
5	0.98	0.82	0.41	0.92	0.49	0.56	24082612	21.26	0.95	0.0192	12.76	131.78
6	1	0.92	0.56	0.97	0.61	0.65	72133300	7.1	0.86	0.0257	17.92	190.7
7	1	0.93	0.56	0.97	0.61	0.65	2.56E+08	2	0.5	0.0283	57.87	108.23
8	1	0.93	0.56	0.97	0.61	0.65	2.56E+08	2	0.5	0.0283	57.87	108.23
9	0.99	0.87	0.44	0.96	0.54	0.62	24164532	21.19	0.95	0.0097	4.65	28.87
10	1	1	0.81	1	0.87	0.89	72264372	7.09	0.86	0.0099	4.75	99.95
11	1	1	0.81	1	0.85	0.85	72663732	7.05	0.86	0.0069	1.79	108.84
12	1	1	0.9	1	0.86	0.66	7.84E+08	0.65	-0.53	0.007	0.74	31.54
13	1	1	0.99	1	0.99	0.94	5.96E+08	0.86	-0.16	0.007	1.04	114.66

Табл. 4. Вариации параметров для SIFT1M. Скрытый слой равен 64
Table 4. Parameter variations for SIFT1M. Hidden dim is 64

id	index	norm embs	encoder
0	Flat	false	dcec
1	HNSW32	false	dcec
2	Flat	false	hyperae
3	Flat	true	hyperae
4	Flat	false	vanae
5	IVF64,PQ16	false	vanae
6	IVF256,PQ64	false	vanae
7	Flat	false	vanae
8	Flat	false	vanae
9	IVF64,PQ16	false	-
10	IVF256,PQ64	false	-
11	IVF1024,PQ64	false	-
12	HNSW32	false	-
13	NSG32	false	-

Табл. 5. Основные результаты для GIST
Table 5. Main results for GIST

id	1-R@100	1-R@10	1-R@1	10-R@100	10-R@10	100-R@100	index_size	cr	ss	o-v s	s	t_i
0	0.5	0.23	0.09	0.31	0.09	0.12	1.3E+08	29.43	0.97	0.0167	1.58	860.27
1	0	0	0	0	0	0	1.23E+09	3.12	0.68	0.0061	0.58	2,111.7
2	0	0	0	0	0	0	1.23E+09	3.12	0.68	0.0063	0.59	2,100.57
3	0	0	0	0	0	0	1.23E+09	3.12	0.68	0.0061	0.58	2,047.03
4	0	0	0	0	0	0	1.23E+09	3.12	0.68	0.0072	0.57	1,969.21
5	0.07	0.04	0.03	0.03	0.01	0.02	2.19E+09	1.75	0.43	0.006	0.59	2,189.44
6	0.06	0.04	0.02	0.03	0.01	0.02	1.3E+08	29.43	0.97	0.0179	1.68	2,086.03
7	0.06	0.04	0.03	0.03	0.01	0.02	2.5E+08	15.33	0.93	0.029	3.08	2,112.66
8	0.5	0.23	0.09	0.31	0.09	0.12	1.3E+08	29.43	0.97	0.0167	1.58	860.27
9	0.5	0.23	0.09	0.31	0.09	0.12	1.3E+08	29.43	0.97	0.0167	1.58	860.27
10	0.5	0.23	0.09	0.31	0.09	0.12	1.3E+08	29.43	0.97	0.0167	1.58	860.27
11	0.98	0.85	0.42	0.94	0.52	0.59	1.3E+08	29.55	0.97	0.0155	1.47	190
12	1	0.98	0.63	1	0.69	0.69	2.53E+08	15.18	0.93	0.0114	1.15	454.85
13	1	0.99	0.67	1	0.6	0.43	4.11E+09	0.93	-0.07	0.0067	0.68	390.93
14	1	0.98	0.63	1	0.69	0.69	2.53E+08	15.18	0.93	0.0114	1.15	454.85

Табл. 6. Вариации параметров для GIST
Table 6. Parameter variations for GIST

id	index	norm embs	enc	hidden dim	norm inp vect	norm out vect	set l hidden
0	IVF1024,PQ120x8	false	dcec	480	false	false	true
1	HNSW32	false	hyperae	240	false	false	false
2	HNSW32	false	hyperae	240	false	false	true
3	HNSW32	true	hyperae	240	true	true	false
4	HNSW32	true	hyperae	240	true	true	true
5	HNSW32	false	vanae	480	false	false	true
6	IVF1024,PQ120x8	false	vanae	480	false	false	true
7	IVF1024,PQ240x8	false	vanae	480	false	false	true
8	IVF1024,PQ120x8	false	dcec	480	false	false	true
9	IVF1024,PQ120x8	false	dcec	480	false	false	true
10	IVF1024,PQ120x8	false	dcec	480	false	false	true
11	IVF256,PQ120x8	false	-	-	-	-	-
12	IVF1024,PQ240x8	false	-	-	-	-	-
13	HNSW32	false	-	-	-	-	-
14	IVF1024,PQ240x8	false	-	-	-	-	-

Табл. 7. Основные результаты для Open Image Dataset
Table 7. Main results for Open Image Dataset

id	1-R@100	1-R@10	1-R@1	10-R@100	10-R@10	100-R@100	index_size	cr	ss	o-v s	s	t_i
0	0.99	0.99	0.98	0.83	0.54	0.45	1.09E+10	1.57	0.36	0.0147	4.79	4,096.41
1	0.99	0.99	0.98	0.83	0.54	0.45	1.09E+10	1.57	0.36	0.0157	4.83	4,111.58
2	1	1	1	0.8	0.49	0.47	6.1E+08	28.18	0.96	0.0163	75.98	3,462.03
3	1	1	1	0.8	0.49	0.47	6.1E+08	28.18	0.96	0.0163	75.8	3,484.63
4	1	1	1	0.81	0.51	0.48	6.12E+08	28.07	0.96	0.0163	76.52	4,180.76
5	1	1	1	0.81	0.5	0.48	6.12E+08	28.07	0.96	0.0165	76.81	4,178.86
6	0.99	0.99	0.98	0.83	0.54	0.45	1.09E+10	1.57	0.36	0.0147	4.79	4,096.41
7	0.99	0.99	0.98	0.83	0.54	0.45	1.09E+10	1.57	0.36	0.0157	4.83	4,111.58
8	1	0.99	0.92	0.99	0.79	0.58	1.96E+10	0.88	-0.14	0.0171	8.2	1,731.1
9	1	1	1	0.95	0.58	0.57	6.17E+08	27.87	0.96	0.009	20.54	1,487.1
10	1	0.99	0.92	0.99	0.79	0.58	1.96E+10	0.88	-0.14	0.0171	8.2	1,731.1

Табл. 8. Вариации параметров для Open Image Dataset
Table 8. Parameter variations for Open Image Dataset

id	index	enc	hidden dim	set l hidden
0	HNSW32	hyperae	256	false
1	HNSW32	hyperae	256	true
2	IVF4096,PQ64	hyperae	128	false
3	IVF4096,PQ64	hyperae	128	true
4	IVF4096,PQ64	hyperae	256	false
5	IVF4096,PQ64	hyperae	256	true
6	HNSW32	hyperae	256	false
7	HNSW32	hyperae	256	true
8	HNSW32	-	-	-
9	IVF4096,PQ64	-	-	-
10	HNSW32	-	-	-

Табл. 9. Основные результаты для Wiki
Table 9. Main results for Wiki

id	1-R@100	1-R@10	1-R@1	10-R@100	10-R@10	100-R@100	index_size	cr	ss	o-v s	s	t_i
0	0.99	0.98	0.97	0.44	0.34	0.24	6.34E+09	56.73	0.98	0.0412	268.86	22,424.1
1	0.99	0.98	0.97	0.33	0.3	0.19	6.34E+09	56.73	0.98	0.0427	263.97	21,576.6
2	0.99	0.98	0.97	0.44	0.34	0.24	6.34E+09	56.73	0.98	0.0412	268.86	22,424.1
3	0.99	0.98	0.97	0.87	0.58	0.52	6.39E+09	56.28	0.98	0.024	103.61	62,943.41

Табл. 10. Вариации параметров для Wiki. Скрытый слой равен 256
Table 10. Parameter variations for Wiki. Hidden dim is 256

id	index	enc	set l hidden
0	IVF16384,PQ64	dcec	true
1	IVF16384,PQ64	dcec	false

2	IVF16384,PQ64	dcec	true
3	IVF16384,PQ64	-	-

Список литературы / References

[1]. J. L. Bentley. Multidimensional binary search trees used for associative searching. Commun. ACM 18, 509–517 (1975).
[2]. S. M. Omohundro. Five balltree construction algorithms (2009).
[3]. H. J´egou, M. Douze, C. Schmid. Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence 33, 117–128 (2011).
[4]. P. Alexander, Y. Malkov, L. Andrey, V. Krylov. Approximate nearest neighbor search small world approach (2011).
[5]. Y. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchicalnavigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence 42, 824–836 (2016).
[6]. H. Zhang, H. Shen, Y. Qiu, Y. Jiang, S. Wang, S. Xu, Y. Xiao, B. Long, W. Yang. Joint learning of deep retrieval model and product quantization based embedding index. Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (2021).
[7]. D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by error propagation (1986).
[8]. Y. Lee, H. Kwon, F. C. Park. Neighborhood reconstructing autoencoders. In Neural Information Processing Systems (2021).
[9]. X. Guo, X. Liu, E. Zhu, J. Yin. Deep clustering with convolutional autoencoders. In International Conference on Neural Information Processing (2017).
[10]. L. Mirvakhabova, E. Frolov, V. Khrulkov, I. Oseledets, A. Tuzhilin. Performance of hyperbolic geometry models on top-n recommendation tasks. Proceedings of the 14th ACM Conference on Recommender Systems (2020).
[11]. H. Gong, H. Schwenk. Multimodal and multilingual embeddings for large-scale speech mining. In Neural Information Processing Systems (2021).
[12]. A. Kuznetsova, H. Rom, N. G. Alldrin, J. R. R. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, V. Ferrari. The open images dataset v4. International Journal of Computer Vision 128, 1956–1981 (2018).
[13]. J. Johnson, M. Douze, H. J´egou. Billion-scale similarity search with gpus. IEEE Transactions on Big Data 7, 535–547 (2017).
[14]. V. Erba, M. Gherardi, P. Rotondo. Intrinsic dimension estimation for locally undersampled data. Scientific Reports 9 (2019).
[15]. J. Bac, E. M. Mirkes, A. N. Gorban, I. Y. Tyukin, A. Y. Zinovyev. Scikit-dimension: A python package for intrinsic dimension estimation. Entropy 23 (2021).
[16]. N. Chen, P. van der Smagt, B. Cseke. Local distance preserving auto-encoders using continuous k-nearest neighbours graphs. ArXiv abs/2206.05909 (2022), https://api.semanticscholar.org/CorpusID:249626370.

Информация об авторах / Information about authors

Игорь Олегович БУЯНОВ – аспирант ФИЦ ИУ РАН, старший разработчик в MTS AI. Сфера научных интересов: обработка естественного языка, анализ пространств эмбеддингов, вычислительная психология.

Igor Olegovich BUYANOV – postgraduate student at FRC CSC RAS, senior developer at MTS AI. Research interests: natural language processing, embedding space analysis, computational psychology.

Василий Владимирович ЯДРИНЦЕВ – младший научный сотрудник в отделе 73 ФИЦ ИУ РАН: интеллектуальных технологий и систем. Сфера научных интересов: индексация векторных данных, обработка естественного языка.

Vasiliy Vladimirovich YADRINSEV – junior researcher at department 73 intellectual technologies and systems of FRC CSC RAS. Research interests: data vector indexing, natural language processing.

Илья Владимирович СОЧЕНКОВ – кандидат физико-математически наук, ведущий научный сотрудник ФИЦ ИУ РАН, ведущий научный сотрудник ИСП РАН, ведущий эксперт-консультант Университета Иннополис. Сфера научных интересов: обработка естественного языка, детекция плагиата.

Ilia Vladimirovich SOCHENKOV – Cand. Sci. (Phys.-Math.), lead researcher at FRC CSC RAS, lead researcher at ISP RAS, lead expert at Innopolis University. Research interests: natural language processing, plagiarism detection.