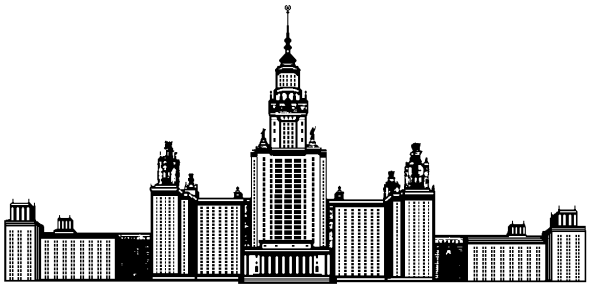


Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

Косарев Евгений Александрович

Векторные представления текстов

ЭССЭ

Москва, 2021

Содержание

§1. Аннотация

Работа с текстовыми данными представляет большой интерес для исследователей и разработчиков машинного обучения. Ежегодно выпускаются алгоритмы, изобретаются подходы, позволяющите эффективно работать с текстом. Задача осложняется тем, что текст разбивается на абзацы, предложение и слова, которые семантически связаны друг с другом.

Целью этой работы является попытка собрать наиболее популярные подходы к построению векторынх представлений текстов и провести обзор каждого из них.

§2. Введение

В настоящее время векторные представления слов и текстов используются в моделях обработки и понимания естественного языка (NLP и NLU). Успешное и информационно емкое кодирование позволили существенно улучшить результаты в таких разделах машинного обучения, как классификация текста, текстовое ранжирование, перевод, машинное понимание [8].

Изначально кодирование текстов решалось накоплением различных статистик, как встречаемость слов в документах, отсюда рассчитывалась и их важность. Одно из самых простых и популярных текстовый представлений было TF-IDF или мешок слов (BOW) [5]. Однако, такие представления имеют слишком большую размерность и не учитывают контекст, в котором встречается, например, предложение к тексте.

От алгоритмов, кодирующих текст или предложение, требовался учет контекста. Реализовали эту идею такие модели, как Paragraph Vectors, skip-thoughts [7, 13]. В то же время, исследователям хотелось получить более общий способ построения векторного представления текстов. Попыткой воплотить эти желания стал автокодировщик [2].

Однако, все еще не получалось сравнивать объекты разной природы. Алгоритм StarSpace [14] позволил кодировать объекты неким общим набором признаков, тем самым вкладывая их в одно общее пространство.

Активно развивались глубокие нейросетевые модели [9, 3, 15], позволяющие более гибко конструировать текстовые эмбединги. Однако на их обучение требовалось значительное время. Появилась идея создать случайный кодировщик [16].

Начало эры современных текстовых моделей положил алгоритм BERT, взятый за основу текущей работы [11]. Механизм внимания позволил выделять из контекста семантически значимые слова, что существенно повысило качество работы алгоритмов. BERT породил плеяду алгоритмов на его основе [12, 6]. С помощью удачной архитектуры удалось повысить качество ранжирования текстов, появилась возможность делать более качественное векторное представление одинаковых по смыслу предложений на разных языках.

Последним результатом, который разбирается в этом обзоре - алгоритм SimCSE [4]. Это одна из самых свежих попыток повысить качество работы BERT-подобных нейронных сетей.

Целью обзора является рассмотрение областей применения и изучение мехнаизма представленных алгоритмов.

§3. Представление текстов

Эмбеддинги для текстов являются хорошим инструментом для сравнения, нахождения похожих текстов. В обзоре рассматриваются различные походы к решению этой задачи.

3.1. Наивный подход

Представление текстов можно свести к более простой задаче - задаче представления слов. Так как любой текст можно разбить на слова, то, получив векторные представления всех этих слов, их можно сагрегировать в один вектор. Самый простой подход - усреднить вектора слов. Пусть text $= \{w_1, w_2, \ldots, w_n\}$, $emb(w_i)$ - веткорное представление i-го слова, тогда векторное представлене текста можно записать через формулу:

$$emb(text) = \frac{\sum_{i=1}^{n} c_i * emb(w_i)}{n}$$

Здесь c_i - веса важности слов, в наивном подходе их можно взять равными единице. Однако такой подход уступает нейросетевым способам получения текстовых представлений (см. рис. 1).

Model	Error rate
Vector Averaging	10.25%
Bag-of-words	8.10 %
Bag-of-bigrams	7.28 %
Weighted Bag-of-bigrams	5.67%
Paragraph Vector	**3.82%**

Рис. 1: Результаты экспериментов для задачи поиска информации [7]

3.2. Paragraph Vectors

В этом разделе вводится понятие распределенного векторного представления слов. Хорошо известная структура для изучения векторов слов показана на рис. 2. Задача состоит в том, чтобы предсказать слово, учитывая другие слова в контексте.

Составляется матрциа W, в которой по столбцам записано векторное представление слов, и представления идут в том же порядке, в котром слова встречаются в тексте. Агрегируя эти векторные представления суумированием или конкатенацией, можно предсказать следующее слово в предложении.

Таким образом, имея последовательность слов $w_1, w_2, \ldots, w_T$, модель максимизирует следующее выражение:

$$\frac{1}{T}\sum_{t=k}^{T-k} \log p(w_t | w_{t-k}, \ldots, w_{t+k})$$

Обычно для предсказания следюущего слова испольвуется многоклассовая классификация, например, softmax:

$$p(w_t | w_{t-k}, \ldots, w_{t+k}) = \frac{\exp y_{wt}}{\sum_i \exp y_i}$$

При этом, все y_i вычисляются как ненормализованные логарифмированные вероятности для каждого получаемого слова i по формуле:

$$y = b + Uh(w_{t-k}, \ldots, w_{t+k}; W)$$

Здесь U и b - параметры softmax, h - один из вариантов агрегации слов - суммирование эмбеддингов или их конкатенация.

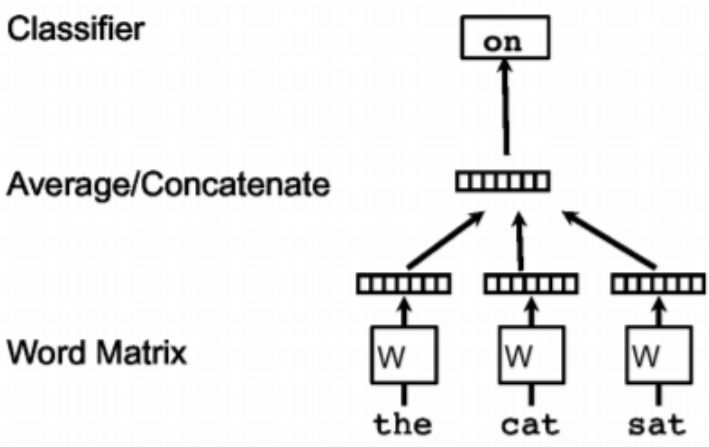


Рис. 2: Фреймворк для получения векторов слов. Контекст из трех слов (“the“cat” и “sat”) используется для предсказания четвертого слова (“on"). [7]

3.2.1 A distributed memory model

Основываясь на предыдущих выкладках, проведем модификацию предложенного алгоритма. В модели Paragraph-Vector каждый абзац кодируется униклальным вектором-столбцом из матрицы D. Матрица W составляется по старым правилам. Агрегацию веторов слов и абзаца можно производить двумя способами, как и раньше. Но в экспериментах используется конкатенация. Формула для вычисления y_i отсается аналогичной, однако в вычислении h(...) используются обе матрицы W и D (рис. 3).

Маркер абзаца можно рассматривать как другое слово. Он действует как память, в которой хранится то, чего не хватает в текущем контексте или теме абзаца. По этой причине алгоритм называется моделью с распределенной памятью (PVDM).

Контекст определяется сколбзяющим окном заданной длины. Эмбеддинг абзаца составляется по словам, лежащим в этом обзаце. Вектора слов являются общими и одинаковыми для всех абзацев.

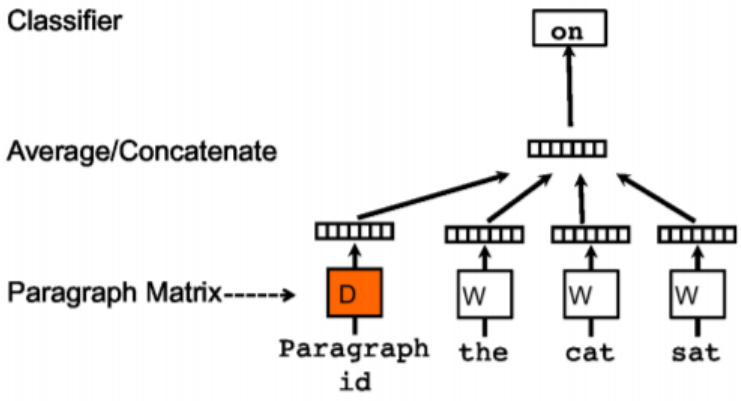


Рис. 3: Модель обучения представления абзацев, очень похожа на предыдщуий алгоритм. Вектор абзаца фактически является новым словом. [7]

Обучние модели производится путем семплирования абзаца и контекста из него.

Для получения эмбеддинга нового абзаца требуется выполнить шаг градиентного спуска для элментов матрицы D в представленной модели. Остальные параметры принимаются фиксированными.

Важной особенностью подхода является то, что бизким по смыслу словам строятся близкие эмбеддинги, а так же ведется учет порядка слов для кодирования абзацев.

3.2.2 Distributed bag of words

Откажемся теперь от порядка слов. Для обучения будем семплировать контекст и случайно выбирать из него одно слово. По вектору слова и абзаца поставим задачу классификации (рис. 4). Такой подход называется PV-DBOW, он отличается упрощенной схемой обучения, а так же требует меньше данных, так как не нужно хранить векторные представления слов.

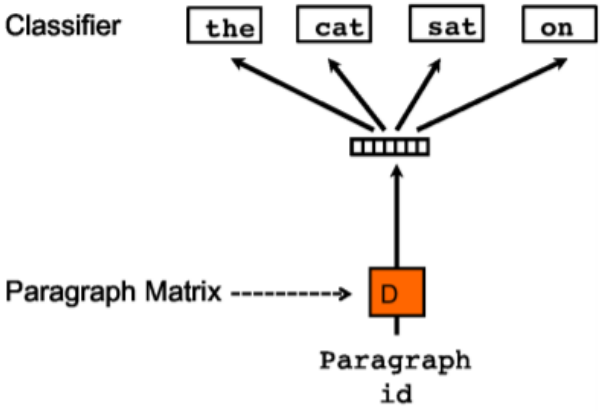


Рис. 4: Схема работы модели PV-DBOW, где по вектору абзаца модель учится предсказывать некий контекст [7]

3.2.3 Замечения

- Вместо слов можно использовать модель со скипграммами.
- В экспериментах стоит комбинировать вектора абзацев, полученные по результатм обоих алгоритмов. Авторы статьи утверждают, что этот подход дает лучший результат.

3.3. The skip-thoughts model

Идея Paragraph Vector интересна тем, что позволяет получать векторные предстовления "без учителя"с помщью использования в модели индикатора распределнного предложения. Однако время, требуемое на предсказания результирующего вектора весьма велико. Авторы предлагают использовать альтернативную функцию потерь, которая поможет абстрагировать модель на скипграммах до всего текста. Таким образом, модель не предсказывает контекст по отдельно выбранному слову, а кодирует предложение так, чтобы по результирующему вектору можно было восстановить контекст (рис. 5). Поэтому операцию объединения векторных представлений слов для получения представления предложения можно заменить на кодировщик (encoder) этого предложения. Поэтому меняется лишь целевая функция.

Модель можно рассматривать в качестве кодировщика-декодировщика. Происходит кодирование исходного предложения, а соседние восстанавливаются по созданному векторному представлению. Кодировщик можно улучшить, используя механизм attention для выделения наиболее значимых слов в исходном предложении. Запишем оптимизируемый фунционал (нужно просуммировать его по всем тройкам соседних предложений):

$$\sum_t \log P(w_{i+1}^t | w_{i+1}^{<t}, h_i) + \sum_t \log P(w_{i-1}^t | w_{i-1}^{<t}, h_i)$$

Смысл оптимизации состоит в том, чтобы сгенерировать такое векторное представление предложения s_i, чтобы с совокупностью информации о предложении s_{i-1} хорошо предсказать s_{i+1} предложение. Алгоритм демонстрирует высокое качество работы, однако является довольно затратным по скорости работы.

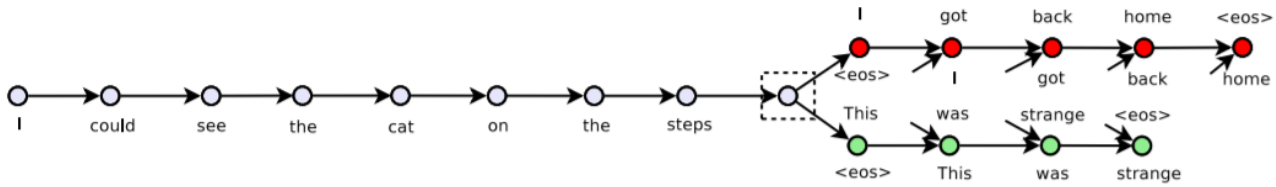
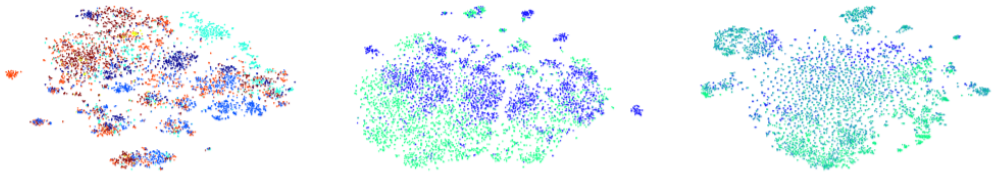


Рис. 5: Схема работы модели skip-thoughts. Предложения нумеруются и выбирается 3 последовательных предложения s_{i-1}, s_i, s_{i+1}. Кодируется i-е предложение и производится попытка восстановить следующее и предыдущее предложения. По цветам разнесены компоненты алгоритма, имеющие одинаковые параметры. Тэг <eos> означает концовку предложения. [13]

Качество полученных векторных представлений демонстрируются на рис. 6. Разными цветами обозначены предложения разных классов, и по разбросу можно судить и хорошем качестве представлений, полученных skip-thoughts моделью.



(a) TREC (b) SUBJ (c) SICK

Рис. 6: Эмбеддинги модели на трех датасетах. Точки окрашиваются в зависимости от их меток (тип вопроса для TREC, субъективность/объективность для SUBJ). В наборе данных SICK каждая точка представляет собой пару предложений, и точки окрашиваются в соответствии с градиентом на основе их связности. [13]

3.4. Autoencoder pretraining

Идея метода состоит в том, чтобы модель по предложению конструировала вектор-представление и по нему могла восстановить исходное предложение. Как показывают эксперименты, для таких задач лучше использовать предобученные модели, чем инициализировать их случайными весами, тогда качество кодирования будет существенно выше. Важным преимуществом метода по сравнению остальными - отсутствие требования к разметке данных, поэтому обучение может происходить на произвольных данных. Если в качестве обучающих данных использовать связанные между собой тексты, то повысится обощающая сопособность алгоритмов, построенных на полученных моделях. Это свойство особенно полезно в задачах с ограниченной разметкой [2].

В предлагаемом подходе кодировщик и декодировщик - это две нейронные сети с общим набором весов. Иллюстрация работы автокодировщика представлена на рис.7.

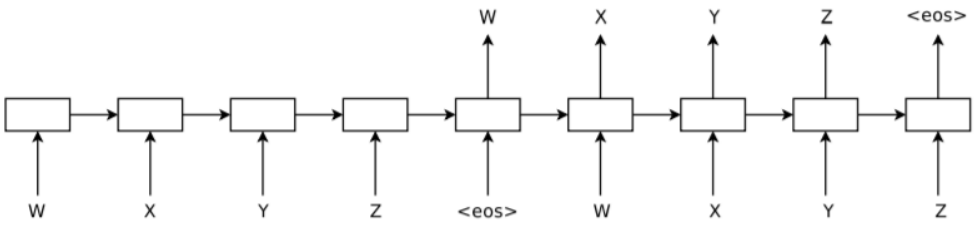


Рис. 7: Показана работа алгоритма при кодировке и декодировке последовательности "WXYZ". Кодировщик использует рекуррентную нейронную сеть, чтобы задать входную последовательность скрытым состоянием, а потом из него восстановить входные данные [2].

3.5. StarSpace

Название метода происходит из принципа его работы. Любые входные объекты, которые можно представить в виде наборов некоторых признаков, (так называемые star) алгоритм переводит в общее для всех объектов любого типа пространство (так называемое space). Благодаря такой гибкости, метод позволяет получать эмбеддинги для любых входных объектов, в том числе и текстов, позволяют их ранжировать между собой и производить поиск в коллекциях объектов, причем сущность запроса может отличаться от сущности всей коллекции. Например, поиск по картинкам можно осуществлять текстовым запросом.

В StarSpace можно помещать объекты разных типов, например, пользователей можно кодировать с помощью наборов просмотренных документов, фильмов, тексты - с помощью мешка слов, tf-idf, n-грамм. Оптимизируя определенный критерий качества, можно настроить близость определенных объектов друг с другом.

Введем матрицу F размера $D \times d$, где D - число признаков, d - размерность пространтсва, в которое кладываются объекты, состоящие из признаков. Тогда любой объект a можно представить в виде: $a = \sum_{i \in a} F_i$, где F_i - i-я строка матрицы F. То есть, любой объект можно представить в виде суммы признаков.

Введем следующую функцию потерь:

$$\sum_{(a,b) \in E^+, b^- \in E^-} L^{batch}(sim(a,b), sim(a,b_1^-), \ldots, sim(a,b_k^-))$$

Здесь вводится множество E^+ - множество похожих друг на друга пар объектов. Для выбранной пары (a,b) множество E^- генерирует такой набор объектов b^-, что все объекты из этого набора не похожи на объект a. В [14] предлагается использовать стратегию k-отрицатльной выборки, то есть выбрать k случайных объектов из того же множества, откуда был выбран объект b.

Функция близости sim(...) задается либо косинусным расстоянием, либо скалярным произведением. Первая метрика работает намного лучше, когда требуется сравнивать объекты большой размерности. L^{batch} - функция ошибки, она минимальна, когда значение $sim(a,b)$ велико, а значения $sim(a,b_i^-), \forall i$ малы. То есть, функционал минимизирует близость непохожих объектов и максимизирует похожесть тех объектов, которые по постановке задаче являются близкими друг к другу.

На рис. 8 демонстрируется эффективность предложеннго метода по сравнению с другими.

3.5.1 Замечания

- Функцию близости sim(a,b) можно выбрать более сложной. Например, это может быть $max_{b^*}[sim(a,b^*)]$, где b^* - набор из нескольких кандидатов.

Task	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	SICK-R	SICK-E	STS14
Unigram-TFIDF*	73.7	79.2	90.3	82.4	-	85.0	73.6 / 81.7	-	-	0.58 / 0.57
ParagraphVec (DBOW)*	60.2	66.9	76.3	70.7	-	59.4	72.9 / 81.1	-	-	0.42 / 0.43
SDAE*	74.6	78.0	90.8	86.9	-	78.4	73.7 / 80.7	-	-	0.37 / 0.38
SIF(GloVe+WR)*	-	-	-	82.2	-	-	-	-	84.6	0.69 / -
word2vec*	77.7	79.8	90.9	88.3	79.7	83.6	72.5 / 81.4	0.80	78.7	0.65 / 0.64
GloVe*	78.7	78.5	91.6	87.6	79.8	83.6	72.1 / 80.9	0.80	78.6	0.54 / 0.56
fastText (public Wikipedia model)*	76.5	78.9	91.6	87.4	78.8	81.8	72.4 / 81.2	0.80	77.9	0.63 / 0.62
StarSpace [word]	73.8	77.5	91.53	86.6	77.2	82.2	73.1 / 81.8	0.79	78.8	0.65 / 0.62
StarSpace [sentence]	69.1	75.1	85.4	80.5	72.0	63.0	69.2 / 79.7	0.76	76.2	0.70 / 0.67
StarSpace [word + sentence]	72.1	77.1	89.6	84.1	77.5	79.0	70.2 80.3	0.79	77.8	0.69/0.66
StarSpace [ensemble w+s]	76.6	80.3	91.8	88.0	79.9	85.2	71.8 / 80.6	0.78	82.1	0.69 / 0.65

Рис. 8: На различных данных приводтся сравнения методов по метрикам точности, полноты, корреляций Пирсона/Спирмена между косинусными предсказанными расстояними и близостью, указанной асессорами. Чем выше значение, тем более высокий результат демонстрирует алгоритм [14].

- По предыдущему пункту можно модифицировать множество E^+ и E^-, чтобы они выдавали сразу несколько объектов и, в случае E^-, отбирали их по другой логике.

3.6. Deep Average Network (DAN)

Главным свойством любой нейронной сети является повышение абстракции представления входных данных с каждым новым слоем. Рассмотрим варианты предлагаемых архитектур:

3.6.1 Neural Bag-of-Words Models (NBOW)

Рассмотрим задачу классификации, сопоставив входной последовательности слов(тексту X) k меток. Тогда, зададим для каждого слова $\forall w$ из текста X его векторное представление v_w. Модель NBOW состоит из агрегации векторов слов и логистической регрессии [3]. Запишем это более формально:

1. Строится вектор агрегации z такой, что

$$z = g(w \in X) = \frac{1}{|X|} \sum_{w \in X} v_w$$

2. Он пропускается через линейный слой с softmax:

$$\hat{y} = softmax(W_s * z + b),$$

Здесь W_s - матрица размером k x d (k - число меток, d - размерность признаковго пространства эмбеддингов) и b - вектор смещения.

$$softmax(q) = \frac{\exp(q)}{\sum_{j=1}^{k} \exp(q_j)}$$

Обучение производится путем минимизации следующего функционала:

$$l(\hat{y}) = \sum_{p=1}^{k} y_p \log(\hat{y_p})$$

Слабой стороной этого метода является низкая восприимчивость к изменениям во входных данных.

3.6.2 DAN

Мотивацию создания архитектуры удобно проиллюстрировать через сравнение с RecNN. Архитектура RecNN на рис. 9 требует размещения линейного и softmax преобразования при обработке каждого следующего слова во входной последовательности. Такой прием важен для того, что избежать затухания градиента в структуре рекуррентных сетей, однако он вычислительно трудоемкий.

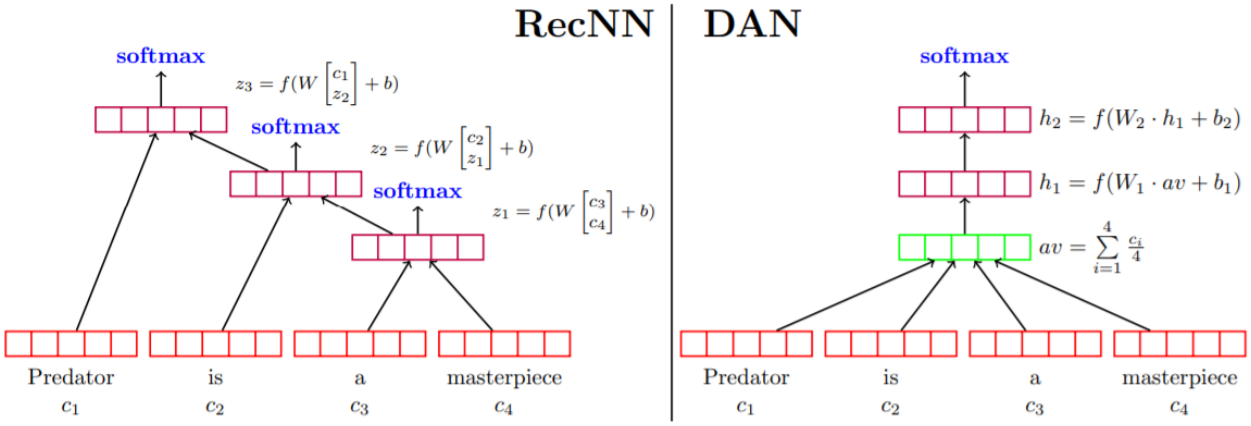


Рис. 9: Сравнение архитектр RecNN и DAN [3].

DAN - решение, позволяющее сократить объем вычислений, сохранив способность улавливать даже небольшие изменения во входных данных. Отличие от NBOW заключается в увеличении числа слоев внутренних слоев до двух. В реальности не удается найти существенных отличий между временем обучения DAN и NBOW.

3.6.3 Замечания

- Для повышения устойчивости работы алгоритма предлагается использовать dropout случайных слов. Этот прием позволяет увеличить число входных последовательностей до $2^{|X|}$. Запишем этот подход более формально:

  $r_w \sim Bernoulli(p)$, где p - вероятность dropout-а. $\hat{X} = \{w | w \in X and r_w > 0\}$. Тогда:

$$z = g(w \in X) = \frac{1}{|\hat{X}|} \sum_{w \in |\hat{X}|} v_w$$

3.7. USE

Трансформер - глубокая модель с блоками внимания [1]. Устройство трансформера представлено на рис. 10. Подаются две входыне последовательности в алгоритм, attention блоки преобразоывают признаковые представление предложений во все более информативные и в конце происходит классификация с использованием softmax. Блок attention способен выделить ниболее важные составляющие входных в него данных. Механизм работы блока self-attention можно представить в виде полного графа, где каждый объект сравнивается с каждым.

Первая модель кодирования предложений на основе трансформера строит вложения предложений с использованием подграфа в механизме self-attention. С помощью подграфа вычисляются контекстно-зависимые представления слов в предложении, которые учитывают как порядок, так и похожесть других слов.

Вторая модель использует DAN из предыдущего пункта обзора.

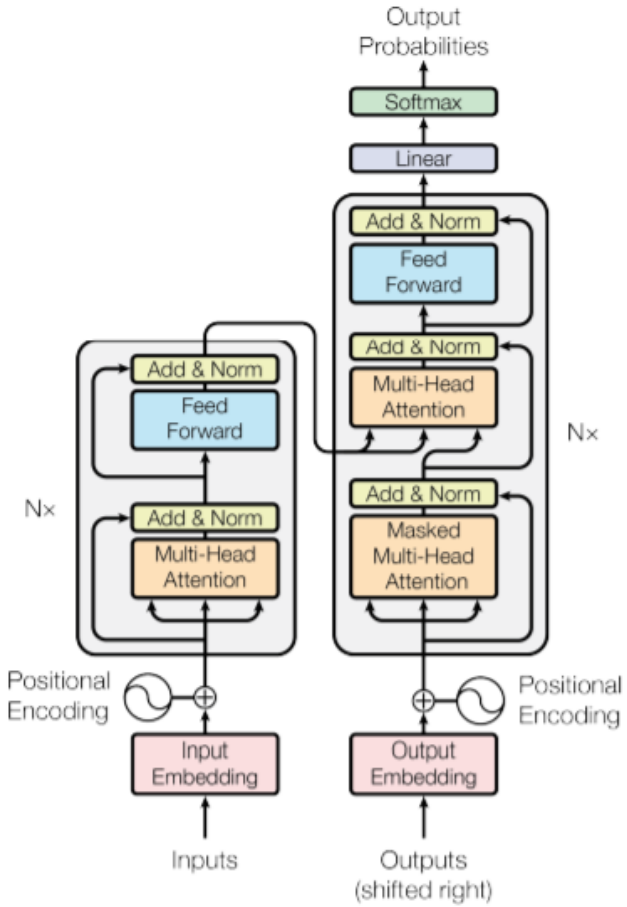


Рис. 10: Схема работы трансформера [1].

Была введена функция угловой близости двух эмбеддингов:

$$sim(u, v) = 1 - \arccos\left(\frac{u \cdot v}{\|u\|\|v\|}\right)/\pi$$

Модели можно комбинировать вместе с различными представлениями слов, было проведено сравнение получившихся подходов, результаты которого приведены на рис. 11.

В таблице USE_T - энкодер на основе трансформера, USE_D - энкодер на основе DAN. w2v w.e. означает, что для кодирования слов использовался предобученный кодировщик word2vec, построенный на скрипграммах, lrn w.e. - использует случайно инициализированные представления слов. В качестве метрик качества выступают точность и корреляция Пирсона. Применение USE повышает значения метрик, тем самым демонстрируя лучший результат.

3.8. Deep Structured Semantic Model (DSSM)

Типичной областью применения DSSM моделей является задача текстового ранжирования, построения эмбеддингов текстов и запросов. Модель позволяет эффективно решать поставленные задачи, а так же отличать тексты по степени релевантности запросу.

Более формально, алгоритм выглядит так:

- На первом шаге вычисляем величину $l_1 = W_1 * x$
- Далее, $l_i = f(W_i * l_{i-1} + b_i)$, где функция активации задается гиперболическим тангенсом.

$$tanh(x) = \frac{1 - \exp -2x}{1 + \exp -2x}$$

Model	MR	CR	SUBJ	MPQA	TREC	SST	STS Bench (dev / test)
Sentence & Word Embedding Transfer Learning							
USE_D+DAN (w2v w.e.)	77.11	81.71	93.12	87.01	94.72	82.14	–
USE_D+CNN (w2v w.e.)	78.20	82.04	93.24	85.87	97.67	85.29	–
USE_T+DAN (w2v w.e.)	81.32	86.66	93.90	88.14	95.51	86.62	–
USE_T+CNN (w2v w.e.)	81.18	87.45	93.58	87.32	98.07	86.69	–
Sentence Embedding Transfer Learning							
USE_D	74.45	80.97	92.65	85.38	91.19	77.62	0.763 / 0.719 (r)
USE_T	81.44	87.43	93.87	86.98	92.51	85.38	0.814 / 0.782 (r)
USE_D+DAN (lrn w.e.)	77.57	81.93	92.91	85.97	95.86	83.41	–
USE_D+CNN (lrn w.e.)	78.49	81.49	92.99	85.53	97.71	85.27	–
USE_T+DAN (lrn w.e.)	81.36	86.08	93.66	87.14	96.60	86.24	–
USE_T+CNN (lrn w.e.)	81.59	86.45	93.36	86.85	97.44	87.21	–
Word Embedding Transfer Learning							
DAN (w2v w.e.)	74.75	75.24	90.80	81.25	85.69	80.24	–
CNN (w2v w.e.)	75.10	80.18	90.84	81.38	97.32	83.74	–
Baselines with No Transfer Learning							
DAN (lrn w.e.)	75.97	76.91	89.49	80.93	93.88	81.52	–
CNN (lrn w.e.)	76.39	79.39	91.18	82.20	95.82	84.90	–

Рис. 11: Сравнение архитектр RecNN и DAN [15]

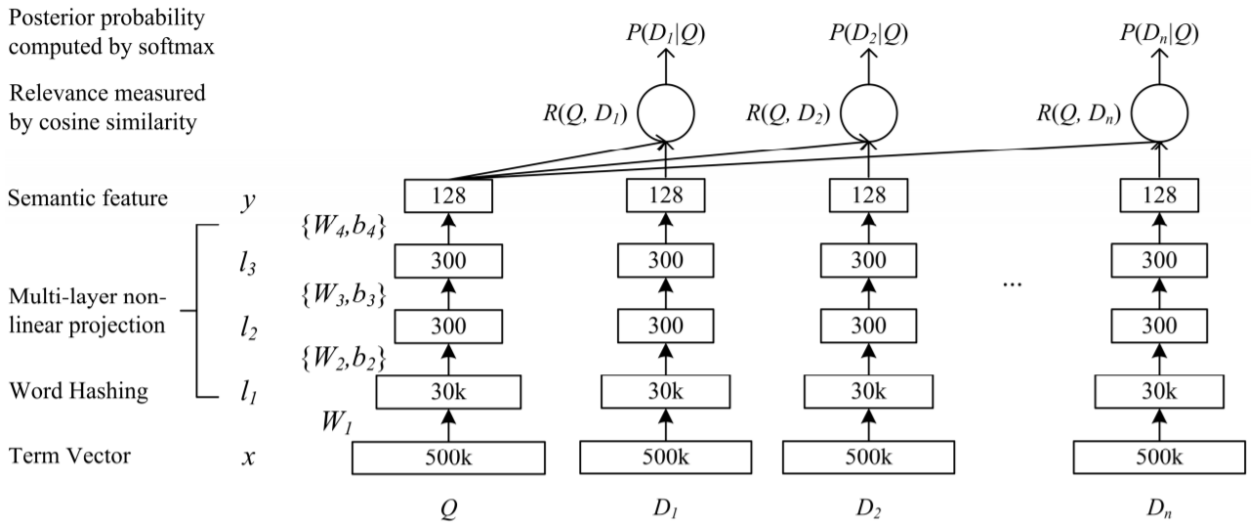


Рис. 12: Иллюстрация работы DSSM. С помощью глубоких нейронных сетей разреженные векторы высокой размерности кодируются в семантическом пространстве [9].

- Близость запроса и документа вычисляется через косинусное расстояния их эмбеддингов: $R(D,Q) = cosine(y_Q, y_D) = \frac{y_Q^T y_D}{\|y_Q\|\|y_D\|}$

Так как тексты кодируются с помощью модели мешка слов (Bag of words), то первый этап алгоритма может оказаться вычислительно сложным и дорогостоящим по памяти из-за необходимости хранить большую матрицу W_1.

Чтобы упростить алгоритм, предлагается разбить входной текст на n-граммы и закодировать его суммой one-hot векторов. Например, в английском алфавите существует всего 10306 триграмм. И ессли триграмма встретилась в текстовом разбиении, то на соотвесвующей ей позици ставится 1. Такое хэширование весьма эффективно и практически не допускает коллизий (рис. 13). Кроме того, хеширование слов способно сопоставлять морфологические вариации одного и того же слова с точками, близкими друг к другу в пространстве букв n-грамм. Преобразование исходных представлений слов в n-граммы производится с помощью фиксированного линейного преобразование в 30 тысячемерное пространство.

Модель обучается таким образом, чтобы для релевантных пар запрос-документ значе-

	Letter-Bigram		Letter-Trigram	
Word Size	**Token Size**	**Collision**	**Token Size**	**Collision**
40k	1107	18	10306	2
500k	1607	1192	30621	22

Рис. 13: Зависимость числа коллизий от количества слов и вида n-грамм [9].

ние близости было большим, для нерелевантных - маленьким. Минимизируется следующая функция:

$L = -\log \prod_{(Q,D)} P(D|Q)$, где $P(Q|D) = \frac{exp(\gamma R(Q,D)}{\sum_{d \in D} exp \gamma R(Q,d)}$. γ - гиперпараметр softmax. В D^+ добавляются 4 случайно выбранных нерелевантных документа.

3.8.1 Замечания

Для повышения устойчивости модели, можно изменить механизм подмешивания нерелевантных документов в набор D^+.

- Увеличить число нерелевантных документов.
- Добавлять лишь те нерелевантные документы, у которых высокая вероятность по алгоритму оказаться релевантными.

3.9. Random encoder

Векторные представления текстов зачастую являются композицией эмбеддингов слов. Пусть $e_1, \ldots, e_n$ - вложения слов, тогда вложение текста ищется в виде специальной функции $h = f_\theta(e_1, \ldots, e_n)$, где θ - определнный набор обучаемых параметров. Все методы, рассмотренные в обзоре, можно вкратце описать этой схемой функционирования. Однако, на обучение моделей может быть потрачено весьма продолжительное время. Предлагается отказаться от обучения модели, задав начальные веса случайными величинами [16].

3.9.1 Bag of random embedding projections (BOREP)

Случайным образом инициализируем матрицу $W \in \mathbb{R}^{D \times d}$, где D - размер вложения, d - размер входного текстового представления слов, в нашем случае bag of words. Матрица инициализируется из равномерного распределения:

$$W \sim U[-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}]$$

И вектора-вложения вычисляются по формуле $h = f_{pool}(We_i)$, где f_{pool} - задаваемая функция. В качестве нее можно выбрать:

- $f_{pool}(x) = \sum(x)$ - суммируются вектора
- $f_{pool}(x) = \max(x)$ - аналогия с max pooling
- $f_{pool}(x) = |x|^{-1} \sum(x)$ -усреднение векторов

Предлагается добавить нелинейность в виде $max(0, h)$.

3.9.2 Random LSTMs

Случайном образом инициализируем веса двунаправленной LSTM из аналогичного предыдущему пункту распределения. Здесь d - размер ячейки внутреннего состояния LSTM. Таким образом, вложение вычисляется по формуле:

$$h = f_{pool}(BiLSTM(e_1, \ldots, e_n))$$

3.9.3 Echo state Networks

Изначально ESN использовались для предсказания последовательностей, то есть для каждого момента времени вычислялось предсказание очередной части последовательности и минимизировалась суммарная ошибка предсказаний. Более формально алгоритм с модификацией выглядит так:

$$\hat{h_i} = f_{pool}(W^i e_i + W^h h_{i-1} + b^i)$$

$$h_i = (1 - \alpha) * h_{i-1} + \alpha \hat{h_i}$$

Здесь все параметры инициализируются случайными числами и не обучаются. $\alpha \in (0, 1]$ отвечает за долгосрочную память внутреннго состояния. В модели обучаются лишь парамтры на последнем нулевом слое:

$$\hat{y_i} = W^0[e_i; h_i] + b^0$$

[a,b] - обозначает конкатенацию векторов a и b. Агрегация результатов происходит по формуле $h = \max(ESN(e_1, \ldots, e_n))$.

3.9.4 Результаты

Приводятся результаты сравнения подходов с другими способами кодирования предложения. Стоит заметить, что существуют такие задачи, в которых алгоритмы со случайным кодированием показывают лучший результат (рис 14).

Model	Dim	MR	CR	MPQA	SUBJ	SST2	TREC	SICK-R	SICK-E	MRPC	STSB
BOE	300	77.3(.2)	78.6(.3)	87.6(.1)	91.3(.1)	80.0(.5)	81.5(.8)	80.2(.1)	78.7(.1)	72.9(.3)	70.5(.1)
BOREP	4096	77.4(.4)	79.5(.2)	88.3(.2)	91.9(.2)	81.8(.4)	**88.8(.3)**	85.5(.1)	82.7(.7)	73.9(.4)	68.5(.6)
RandLSTM	4096	77.2(.3)	78.7(.5)	87.9(.1)	91.9(.2)	81.5(.3)	86.5(1.1)	85.5(.1)	81.8(.5)	**74.1(.5)**	72.4(.5)
ESN	4096	**78.1(.3)**	**80.0(.6)**	**88.5(.2)**	**92.6(.1)**	**83.0(.5)**	87.9(1.0)	**86.1(.1)**	**83.1(.4)**	73.4(.4)	**74.4(.3)**
InferSent-1 = paper, G	4096	81.1	86.3	90.2	92.4	84.6	88.2	88.3	86.3	76.2	75.6
InferSent-2 = fixed pad, F	4096	79.7	84.2	89.4	92.7	84.3	90.8	88.8	86.3	76.0	78.4
InferSent-3 = fixed pad, G	4096	79.7	83.4	88.9	92.6	83.5	90.8	88.5	84.1	76.4	77.3
Δ InferSent-3, BestRand	-	*1.6*	*3.4*	*0.4*	*0.0*	*0.5*	*2.0*	*2.4*	*1.0*	*2.3*	*2.9*
ST-LN	4800	79.4	83.1	89.3	93.7	82.9	88.4	85.8	79.5	73.2	68.9
Δ ST-LN, BestRand	-	*1.3*	*3.1*	*0.8*	*1.1*	*-0.1*	*0.5*	*-0.3*	*-3.6*	*-0.9*	*-5.5*

Рис. 14: Сравнение способов кодирования текстов на различных датасетах [16].

3.10. Sentence BERT (SBERT)

BERT - довольно популярный алгоритм, построенный на базе трансформера, позволяющий с высокой степенью качества решать, например, такие задачи, как оценка близости текстов по содержанию. Sentence-BERT является модификацией предобученной нейросетевой модели BERT, использующей сиамские и триплетные сетевые структуры для получения семантически значимых эмбеддингов предложений, смысловую близость которых можно сравнить с помощью косинусного расстояния.

3.10.1 BERT

BERT - предобученная модель, основанная на архитектуре трансформера. На вход модели подается два предложения, применяются multi-head attention и полученные данные подаются в специальную функцию для оценки уровня похожести предложений. Однако, из BERT-а в явном виде не удается получить эмбеддинг предложения. Чтобы его найти, нужно подать на вход модели предложение и усреднить все выходы модели. Так же качество получаемых эмбеддингов зависит от степени и качества обучения, которое в свою очередь занимает весьма продолжительное время.

3.10.2 SBERT

Отличие предлагаемой архитектуры от канонической в том, что SBERT добавляет операцию пулинга (pooling) в последний слой для получения вложения нужного размера. Обычно используется функция усреднения.

Обучение происходит по схеме на рис. 15.

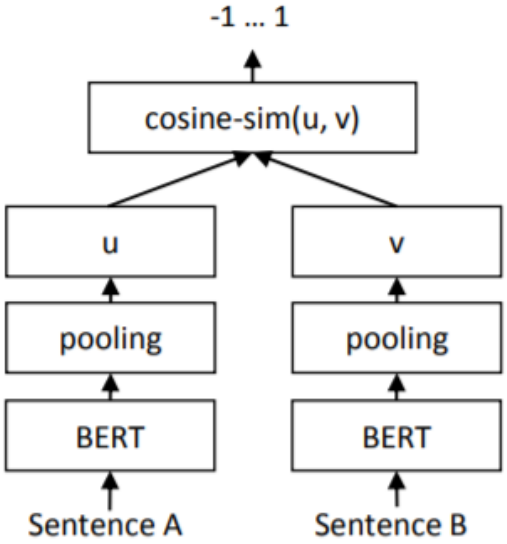


Рис. 15: Схема оценивания качества модели [11].

Алгоритм можно модифицировать на случай использования триплетов из BERT-ов. Тогда на вход алгоритму будут подаваться: a - некотрое предложение, p - предложение, относящеемся к положительному классу, n - к отрицательному. Положительность и отрицательность классов задается спецификой решаемой задачи. Например, можно оценивать похожесть предложений друг на друга. Минимизируем функционал, который поощряет близость к объектам из положительного класса и большое расстояние до объектов из отрицательного класса:

$$max(\|s_a - s_p\| - \|s_a - s_n\| + \epsilon, 0)$$

Здесь s_i - эмбеддинг i-го предложения, $\|...\|$ - расстояние между предложениями, отступ ϵ вносит поправку, на которую положительное предложение должно быть ближе, по сравнению с отрицательном, к предложению a.

Эксперименты показывают, что выбранный способ построения эмбеддингов демонстрирует лучшие результаты, чем другие алгоритмы в задаче оценки близости текстов косинусным расстоянием (рис 16).

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

Рис. 16: Сравнение способов кодирования текстов на различных датасетах с помощью ранговой корреляции Спиремена [11].

3.11. Language-agnostic BERT Sentence Embedding

Подходы с использованием предобучения для моделирования маскированного языка (mask language modelling, MLM) с последующем дообучением под конкретные типы задач оказались весьма эффективным инструментом для их решения. Однако предварительно обученные MLM модели не могут качественно кодировать предложения. Классический BERT генерирует эмбединги для представлений текстов на исходном языке, многоязычный BERT (multilingual BERT) использует двойные кодировкщики, но ни один из них не использует MLM подход. Для многоязычных моделей обучение обычно строится по всем парам предложений на различных языках, что практически квадратично увеличивает исходную текстовую выборку.

Предалагается использовать модель с MLM и TLM (моделирования переводов) предобученными кодировщиками. Модель будет работать со 112 яыками. С помощью кодировщиков будет составляться векторное представление исходного и целевого текста (на другом языке). Мера их скодства выбирается косинусной. На рис 17 иллюстрируется метод работы алгоритма:

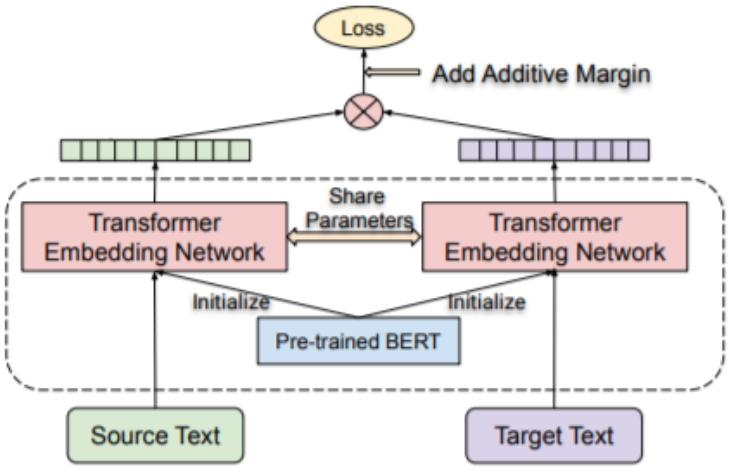


Рис. 17: Юллюстрация работы двойного кодировщика [6].

Обучение двунаправленного двойного кодировщика осуществляется посредством оптимизациеи следующей функции:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\frac{\exp(\phi(x_i, y_i) - m)}{\exp(\phi(x_i, y_i) - m) + \sum_{n=1, n\neq i}^{N}\exp(\phi(x_i, y_n))}$$

Здесь $\phi(a, b)$ - мера близости двух векторных представлений. Функция потерь пытается обучить представления по N - 1 альтернтиве в батче с учетом дисконтирования близости век-

торов числом m. Итого, итоговая ошибка считается по сумме ошибок в двух направлениях:

$$L_{final} = L + \hat{L}$$

Обученная модель демонстрирует хорошие результаты, по сравнению с рассматриваемыми аналогами на датасете BUCC рис. 18.

	Models	fr-en			de-en			ru-en			zh-en		
		P	R	F	P	R	F	P	R	F	P	R	F
Forward	Artetxe and Schwenk (2019a)	82.1	74.2	78.0	78.9	75.1	77.0	-	-	-	-	-	-
	Yang et al. (2019a)	**86.7**	85.6	86.1	90.3	88.0	89.2	84.6	91.1	87.7	86.7	**90.9**	88.8
	LaBSE	86.6	**90.9**	**88.7**	**92.3**	**92.7**	**92.5**	**86.1**	**91.9**	**88.9**	**88.2**	89.7	**88.9**
Backward	Artetxe and Schwenk (2019a)	77.2	72.7	74.7	79.0	73.1	75.9	-	-	-	-	-	-
	Yang et al. (2019a)	83.8	85.5	84.6	89.3	87.7	88.5	83.6	90.5	86.9	**88.7**	87.5	88.1
	LaBSE	**87.1**	**88.4**	**87.8**	**91.3**	**92.7**	**92.0**	**86.3**	**90.7**	**88.4**	87.8	**90.3**	**89.0**

Рис. 18: precision, recall, f1 считаются по индикатору близости по косинусной метрике. В качестве таргета берется предложения на английском языке, источником - остальные языки. В обратном проходе наоборот. [6].

3.12. REPBERT

В поиске документов важной задачей является их ранжирование в порядке убывания релевантности запросу. Поэтому появилась потребность в адаптации BERT-а под поисковые запросы и способность более качественного ранжирования пар запрос-доумент. Архитектура BERT остается не тронутой, как и способ получения эмбедингов. В качестве оценки меры близости запроса и документа (Rel(q, d)) берется скалярное произведение их эмбедингов.

	MRR@10 Dev	MRR@10 Test	R@1000 Dev	Latency (ms/query)
BM25(Anserini) [2]	0.184	0.186	0.853	50
doc2query [2]	0.215	0.218	0.893	90
DeepCT [1]	0.243	0.239	0.913	55
docTTTTTquery [3]	0.277	0.272	**0.947**	64
Ours (RepBERT)	**0.304**	**0.294**	0.943	80
Best non-ensemble, non-BERT [19]	0.298	0.291	0.814	-
BM25 + BERT Large [20]	0.365	0.358	0.814	3,400

Рис. 19: Сравннеие моделей ранжирования [12].

Лосс функция задается как:

$$L(q, d_1^+, \ldots, d_m^+, d_{m+1}^-, \ldots, d_n^-) = \frac{1}{n} \sum_{1 \le i \le m, m < j \le n} max(0, 1 - (Rel(q, d_i^+) - Rel(q, d_j^-)))$$

, где в батче размера n семлируется m релевантных документов и n-m нерелевантрных. Она поощряет увеличивать разность расстояний между запросом-положительным документом и запросом-отрицательным документов. Термины положительности и отрицательности показыают отношение объекта к одному из классов.

Результаты ранжирования (рис. 19) указывают на то, что удалось получить векторные представления текстов такие, что они учитывают специфику поставленной задачи.

3.13. SimCSE

В статье предлагается несколько подходов к решению задачи построения векторных представлений текстов: с и без учителя. Подхрод без учителя предсказывает входную последовательность с использованием drop-out слов в качестве шумовых данных. Обучение с учителям происходит с помощью контрастного обучения, решения задачи бинарной классификации пар предложений на похожие и непохожие. Схемы работы представлены на рис. 20.

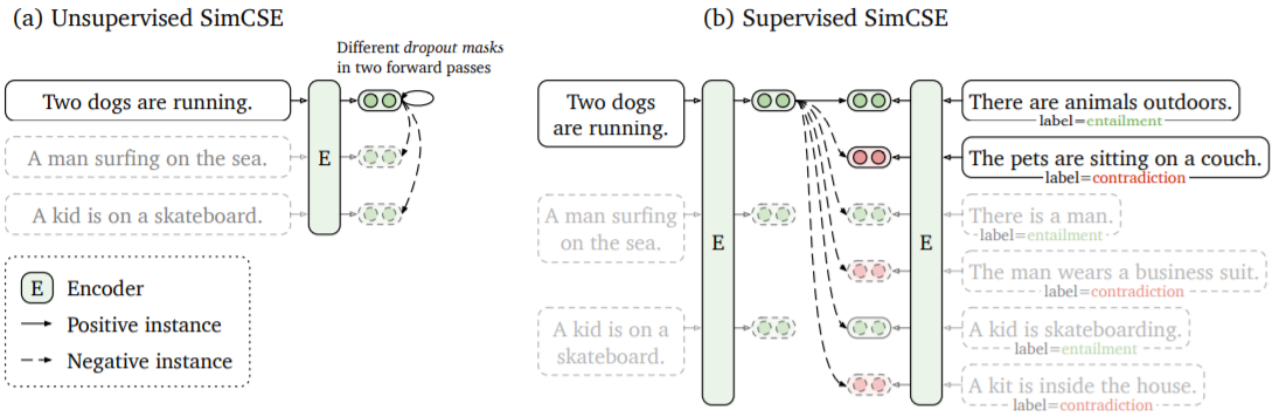


Рис. 20: Unsupervised и supervised SimCSE [4].

Посчитаем кросс-энтропию для батча объектов:

$$l_i = -\log \frac{\exp sim(h_i, h_i^+)/\tau}{\sum_{j=1}^{N} \exp sim(h_i, h_j^+)/\tau}$$

Здесь sim - косинусное рассотояние, τ - гиперпараметр, отвечающий за амортизацию значений функции близости, h_i - векторное представление i-го предложения, $h_i = f_\theta(x_i)$ - результат работы кодировщшика с обучаемыми параметрми θ. В качестве кодировщика используем предобученную модель BERT. Оптимизируя функционал, донастроим веса предобученной модели под особенности решаемой задачи.

Пары похожих предложений будем получать с помощью текстовой аугментации (назовем это noise dropout), а именно удаления слов, вставки и их переупорядочивания, либо из специализированных датасетов.

Введем так же понятия выравнивания (alignment) и равномерности (uniformity).

По распределению положительных пар alignment рассчитывается ожидаемое расстояние между похожими объектами.

$$l_{align} = \mathbb{E}_{(x,x^+)\sim p_{pos}} \|f(x) - f(x^+)\|^2$$

Равномерность изучает меру того, насколько эмбеддинги равномерно распределны по своему пространству (построение ведется генерацией объектов из распределения всех имеющихся данных)

$$l_{uniform} = \log \mathbb{E}_{(x,y)\sim p_{data}} e^{-2\|f(x)-f(x^+)\|^2}$$

3.13.1 Unsupervised SimCSE

Будем считать, что $x_i^+ = x_i$. Пусть $h_i^z = f_\theta(x_i, z)$, z здесь - маска dropout, применяется к полносвязным слоям. Тогда формула l_i меняется соотвествующим образом и оптимизируется.

Представлено так же объяснение важности noise dropout (аугментации данных) в модели (рис. 21). Точками на графике показаны шаги обучения, каждой цвет соотвествует особой стратегии обучения.

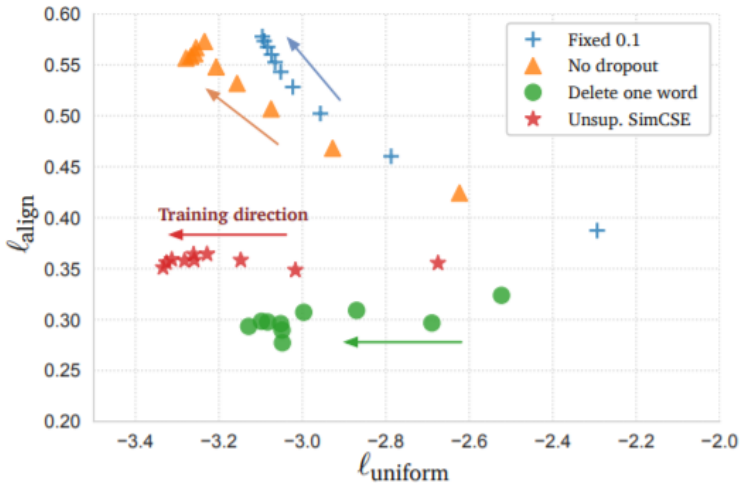


Рис. 21: Чем меньше метрики равномерности и выравнивания, тем лучше работает модель. Видно, что все подходы уменьшают степень равномерности, но использование фиксированного dropout применительно к слоям и не использование его вообще ведет к сущственному росту одной из метрик с ходом обучения. Применение только лишь noise dropout (SimCSE) дает лучший результат. [4].

3.13.2 Supervised SimCSE

Теперь для каждого прделожения имеем похожие и непохожие на него объекты, тогда меняется оптимизируемая функция:

$$l_i = -\log \frac{\exp sim(h_i, h_i^+)/\tau}{\sum_{j=1}^{N}(\exp sim(h_i, h_j^+)/\tau + \exp sim(h_i, h_j^-)/\tau)}$$

3.13.3 Результаты

Обучая исходные кодировщики по имеющимся метрикам, можно получить весьма неплохие результаы. Сравним оба подхода (с учителем и без) с другими моделями в задаче измерения косинусных расстояний между полученным эмбедингами. В таблице (рис. 22) указаны значения корреляции Спирмена предсказаний и правильного ответа.

Предлагаемый подход демонстрирует значительное повышени качества работы кодировщиков после дообучения их с помощью подхода SimCSE. Благодаря этому, векторные вложения текстов получаются более емкими.

Model	STS12	STS13	STS14	STS15	STS16	STS-B	SICK-R	Avg.
Unsupervised models								
GloVe embeddings (avg.)♣	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
$BERT_{base}$ (first-last avg.)	39.70	59.38	49.67	66.03	66.19	53.87	62.06	56.70
$BERT_{base}$-flow	58.40	67.10	60.85	75.16	71.22	68.66	64.47	66.55
$BERT_{base}$-whitening	57.83	66.90	60.90	75.08	71.31	68.24	63.73	66.28
IS-$BERT_{base}$♡	56.77	69.24	61.21	75.23	70.16	69.21	64.25	66.58
∗ SimCSE-$BERT_{base}$	**66.68**	**81.43**	**71.38**	**78.43**	**78.47**	**75.49**	**69.92**	**74.54**
$RoBERTa_{base}$ (first-last avg.)	40.88	58.74	49.07	65.63	61.48	58.55	61.63	56.57
$RoBERTa_{base}$-whitening	46.99	63.24	57.23	71.36	68.99	61.36	62.91	61.73
∗ SimCSE-$RoBERTa_{base}$	**68.68**	**82.62**	**73.56**	**81.49**	**80.82**	**80.48**	**67.87**	**76.50**
∗ SimCSE-$RoBERTa_{large}$	**69.87**	**82.97**	**74.25**	**83.01**	**79.52**	**81.23**	**71.47**	**77.47**
Supervised models								
InferSent-GloVe♣	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder♣	64.49	67.80	64.61	76.83	73.18	74.92	76.69	71.22
$SBERT_{base}$♣	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
$SBERT_{base}$-flow	69.78	77.27	74.35	82.01	77.46	79.12	76.21	76.60
$SBERT_{base}$-whitening	69.65	77.57	74.66	82.27	78.39	79.52	76.91	77.00
∗ SimCSE-$BERT_{base}$	**75.30**	**84.67**	**80.19**	**85.40**	**80.82**	**84.25**	**80.39**	**81.57**
$SRoBERTa_{base}$♣	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
$SRoBERTa_{base}$-whitening	70.46	77.07	74.46	81.64	76.43	79.49	76.65	76.60
∗ SimCSE-$RoBERTa_{base}$	**76.53**	**85.21**	**80.95**	**86.03**	**82.57**	**85.83**	**80.50**	**82.52**
∗ SimCSE-$RoBERTa_{large}$	**77.46**	**87.27**	**82.36**	**86.66**	**83.93**	**86.70**	**81.95**	**83.76**

Рис. 22: Сравнение sipervised и unsupervised подходов других моделей с алгоритмом SimCSE [4].

§4. Выводы

В обзоре были рассмотрены различные алгоритмы векторного представления текстов. В более ранних работах авторы старались представить наиболее универсальные методы кодирования текстов. С течением времени алгоритмы становились все более сложными с точки зрения обучения и временных затрат на этот этап. Вместе со времнем работы алгоритма зачастую повышалось и качество составляемых эмбеддингов. Появление механизма attention и сети BERT стало поворотным событием, так как все современные работы по сути являются модификациями исходного алгоритма. Стоит отметить, что в последних работах авторы стараются решить не только общую задачу получения эмбедингов слов. Например, через решение смежных задач (текстового ранжирования, поиска ближайших текстов) получают необходимые результаты.

Стоит отметить, что более сложные методы не всегда дают лучший результат. На рис. 23 видно, что иногда случается так, что более сложные методы дают результаты сравнимые со случайным кодировщиком. Похоже, что на данный момент не существует универсального способа решения задачи кодирования текстов.

Approach	BShift	CoordInv	ObjNum	SentLen	SOMO	SubjNum	Tense	TopConst	TreeDepth	WC
Baseline										
Random Embedding	50.16	51.38	50.82	17.07	50.44	50.79	50.02	4.71	17.57	0.12
Experiments										
ELMo (BoW, all layers, 5.5B)	**85.23**	69.92	**89.06**	**89.28**	**59.20**	**91.16**	89.73	84.50	**48.62**	88.86
ELMo (BoW, all layers, original)	84.29	69.44	88.65	89.03	58.20	90.18	90.33	**84.96**	48.32	89.90
ELMo (BoW, top layer, original)	81.18	68.47	87.61	78.20	58.64	90.16	88.78	81.54	44.97	72.78
Word2Vec (BoW, google news)	40.89	53.24	80.03	53.03	54.29	81.34	86.20	63.14	28.74	90.20
p-mean (monolingual)	50.09	50.45	83.27	86.42	53.27	81.73	88.18	61.66	38.20	**98.85**
FastText (BoW, common crawl)	50.28	53.87	80.08	66.97	55.21	80.66	87.41	67.10	36.72	91.09
GloVe (BoW, common crawl)	49.52	55.28	78.00	73.00	54.21	79.75	85.52	66.20	36.30	88.69
USE (DAN)	60.19	54.28	69.04	57.89	55.01	71.94	80.43	60.21	25.90	60.06
USE (Transformer)	60.52	58.19	74.60	79.84	58.48	77.78	86.15	68.73	30.49	54.19
InferSent (AllNLI)	56.64	68.34	80.54	84.13	55.79	84.45	86.74	78.34	41.02	95.18
SkipThought	70.19	**71.89**	83.55	86.03	54.74	86.06	**90.05**	82.77	41.22	79.64

Рис. 23: Сравнение подходов кодирования текстов с использованием MLP в задачах классификации [10].

§5. Список литературы

[1] Attention is all you need / Ashish Vaswani, Noam Shazeer, Niki Parmar et al. // arXiv preprint arXiv:1706.03762. — 2017.

[2] Dai Andrew M, Le Quoc V. Semi-supervised sequence learning // arXiv preprint arXiv:1511.01432. — 2015.

[3] Deep unordered composition rivals syntactic methods for text classification / Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, Hal Daumé III // Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers). — 2015. — P. 1681–1691.

[4] Gao Tianyu, Yao Xingcheng, Chen Danqi. SimCSE: Simple Contrastive Learning of Sentence Embeddings // arXiv preprint arXiv:2104.08821. — 2021.

[5] Jing Li-Ping, Huang Hou-Kuan, Shi Hong-Bo. Improved feature selection approach TFIDF in text mining // Proceedings. International Conference on Machine Learning and Cybernetics / IEEE. — Vol. 2. — 2002. — P. 944–946.

[6] Language-agnostic bert sentence embedding / Fangxiaoyu Feng, Yinfei Yang, Daniel Cer et al. // arXiv preprint arXiv:2007.01852. — 2020.

[7] Le Q., Mikolov T. Distributed representations of sentences and documents // International conference on machine learning. — 2014. — P. 1188–1196.

[8] Learning deep structured semantic models for web search using clickthrough data / Po-Sen Huang, Xiaodong He, Jianfeng Gao et al. // Proceedings of the 22nd ACM international conference on Information & Knowledge Management. — 2013. — P. 2333–2338.

[9] Learning deep structured semantic models for web search using clickthrough data CIKM / Po-Sen Huang, Xiaodong He, Jianfeng Gao et al. // Google Scholar Google Scholar Digital Library Digital Library. — 2013.

[10] Perone Christian S, Silveira Roberto, Paula Thomas S. Evaluation of sentence embeddings in downstream and linguistic probing tasks // arXiv preprint arXiv:1806.06259. — 2018.

[11] Reimers Nils, Gurevych Iryna. Sentence-bert: Sentence embeddings using siamese bert-networks // arXiv preprint arXiv:1908.10084. — 2019.

[12] RepBERT: Contextualized Text Embeddings for First-Stage Retrieval / Jingtao Zhan, Jiaxin Mao, Yiqun Liu et al. // arXiv preprint arXiv:2006.15498. — 2020.

[13] Skip-thought vectors / Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov et al. // arXiv preprint arXiv:1506.06726. — 2015.

[14] Starspace: Embed all the things! / Ledell Wu, Adam Fisch, Sumit Chopra et al. // Proceedings of the AAAI Conference on Artificial Intelligence. — Vol. 32. — 2018.

[15] Universal sentence encoder / Daniel Cer, Yinfei Yang, Sheng-yi Kong et al. // arXiv preprint arXiv:1803.11175. — 2018.

[16] Wieting John, Kiela Douwe. No training required: Exploring random encoders for sentence classification // arXiv preprint arXiv:1901.10444. — 2019.