НЕЙРОСЕТЕВЫЕ МЕТОДЫ В ЗАДАЧЕ СЕНТИМЕНТ-АНАЛИЗА

Рябыкин Алексей Сергеевич[1], Сухов Егор Аркадьевич[2]
1) Московский Авиационный Институт, студент, бакалавр, Москва, Россия;
2) Московский Авиационный Институт, доцент кафедры 802, Москва, Россия;
ras.unlucky@gmail.com, sukhov.george@gmail.com

Аннотация

Современный мир, содержащий уже слабо измеримый объем данных и отличающийся либерализмом общественного сознания, срастив эти две парадигмы, породил невозможность охвата потока мнений в формате ручного анализа. Востребованность в автоматизации сентимент-анализа заволакивает ныне немалое количество сфер: от информационной безопасности в лице выявления угроз и выбросов агрессии, экономической науки в качестве анализа фондовых рынков путем мониторинга настроений до определения лояльности потребителей в продукт-менеджменте. Очевидным является желание получить качественную модель, способную изменяться вместе с языком, совмещающую рациональность разработки и точность. Модели, основанные на правилах, предельно ясно, будут лучшими относительно последнего пункта, однако разработка и последующие изменения потребуют долгой деятельности и постоянной модерации соответственно. В следствие этого, востребованными являются реализации, способные обучаться с хорошим качеством и легко дообучаться при лингвистических мутациях. В данной работе произведена попытка поиска, реализации и сравнения нейросетевых моделей глубокого обучения, подходящих под поставленные для них требования. Рассмотренные модели были реализованы как по отдельности, так и в конкатенации. Лучшим результатом работы является конечная мета-модель, получившая значение метрики accuracy = 0.965 на тестовом наборе данных.

Ключевые слова

[*сентимент-анализ, векторное представление слов, свертка, рекуррентная сеть, градиентный спуск, обратное распространение ошибки, рекуррентный блок*]

Введение

Решение задачи семантического анализа необходимо содержит предварительную обработку текста, в лучшем случае, включающую:

1) приведение текста к единому регистру (обычно, нижнему);
2) токенизацию текста – разбиение блоков текста на более мелкие атомарные единицы: n-граммы, слова, ныне актуально – предложения или даже абзацы;
3) морфологическую разметку: с целью различать омонимы, несущие отличные семантические роли. Например, «гладь» существительное, значащее ровную поверхность и, одновременно с этим, глагол в повелительном наклонении «гладь». В английском языке, который выбран для анализа методов нейросетевого моделирования в этой статье, подобное встречается реже, однако имеет место быть: “can” в смысле мочь, уметь (глагол) и “can” как консервная банка (существительное);
4) лемматизацию (приведение слова к лемме). Этот процесс выполняется для оптимального сокращения необходимого словаря, редуцируя слова до их корней;
5) векторизацию слова: представление токенов в качестве численных векторов для последующей обработки.

Лишь первая из приведенных выше задач является очевидной и простой. Остальные же требуют творческих путей решений, однако, хоть и стоят упоминания, подобные подзадачи будут рассмотрены в статье лишь косвенно. В ней акцент сфокусирован исключительно на задаче классификации, как бинарной (определение полярности текста), так и многоклассовой (когда речь заходит о ранжировании мнений) методами нейросетевого моделирования. Рассмотрены варианты решения подобных проблем методами глубокого обучения: CNN (convolution neural networks 1 dimension, одномерные сверточные нейронные сети), RNN (recurrent neural networks, рекуррентные нейронные сети), в частности, LSTM (long short-term memory, долгая краткосрочная память), GRU (gated recurrent unit, управляемый рекуррентный блок). Рассмотрены их положительные стороны и слабые места, способы борьбы с последними. Кроме того, применен метод ансамблирования Stacking к обученным моделям для получения более уверенного и лучшего результата с использованием современных способов прототипирования моделей (PyTorch и TensorFlow) и с применением программно-аппаратной архитектуры параллельных вычислений Nvidia CUDA. Используемыми данными послужили рецензии на кино с платформы IMDB (в случае бинарной классификации выборка составлялась на основе оценки: 10 – отзыв положительный, 1 – отзыв отрицательный), отзывы с сайта YELP и комментарии социальной сети Twitter. Язык исследования: английский. Для адаптации результатов под работу с другими языками, необходима другого рода предобработка данных, сами архитектуры моделей сохранят свою работоспособность.

Таблица 1. Результаты исследований

Данные		Модели			
		CNN	LSTM	GRU	META
IMDB (2)	Train	0.9963	0.9982	0.9751	-
	Val	0.8991	0.88	0.9612	-
	Test	0.8996	0.8765	0.9353	-
Twitter (3)	Train	0.93	0.8431	0.9937	-
	Val	0.8542	0.8343	0.9605	-
	Test	0.8771	0.841	0.9574	-
Yelp (2)	Train	0.5538	0.9172	0.9670	-
	Val	0.534	0.9762	0.9562	-
	Test	0.613	0.945	0.9459	-
Merged (2)	Score	-	-	-	0.965

Основная часть (Результаты исследований)

Сверточные нейронные сети

Созданные для задач компьютерного зрения сверточные нейронные сети [1-3] были весьма продуктивно применены и в других сферах глубокого обучения, в том числе и NLP (natural language preprocessing, обработка

естественного языка). Использование их достигло апогея в ответ на ограниченность применения полносвязных сетей для обработки данных с сеточной топологией [5].

Определение: Пусть $u, v: \mathbb{R}^n \to \mathbb{R}$ – две функции, интегрируемые относительно меры Лебега на пространстве $\mathbb{R}^n$. Тогда функция $u * v: \mathbb{R}^n \to \mathbb{R}$ называется сверткой и определяется соотношением:

$$(u * v)(x) \coloneqq \int_{\mathbb{R}^n} u(y)v(x-y)dy,$$

где у иногда называют «давностью». При этом функция u называется входом (input), функция v – ядром (kernel). Результат свертки именуется картой признаков (feature map). В задаче классификации текстов будем рассматривать одномерную свертку в силу векторного представления текстовой информации. Для $n = \infty$ формула примет вид:

$$(u * v)(x) \coloneqq \int_{-\infty}^{\infty} u(y)v(x-y)dy$$

Однако, так как задача классификации текстов представляет собой конечный, дискретный случай, окончательно формула принимает следующий вид:

$$(u * v)(x) \coloneqq \sum_{y=0}^{K} u(y)v(x-y),$$

где K – размер ядра.

Для иллюстрации операции свертки рассмотрим функции u, v как сигналы Рис. 1. Сдвигая ядро v относительно входного сигнала u и посчитав «сходство» фрагмента сигнала с ядром (в случае векторов, речь идет о скалярном произведении) для каждого смещения, получим результат свертки.

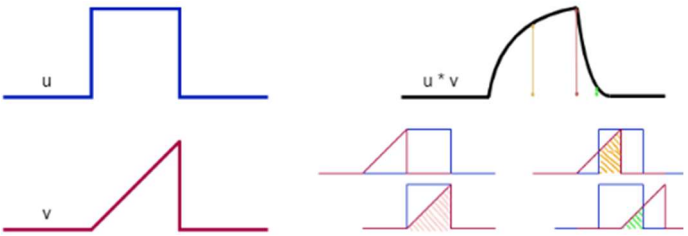

Рис. 1. Пример операции свертки для двух сигналов

Для изображений характерно матричное представление, каждому элементу которого соответствует значение пикселя. На Рис.2. приведен случай с изображением размерности 5×5.

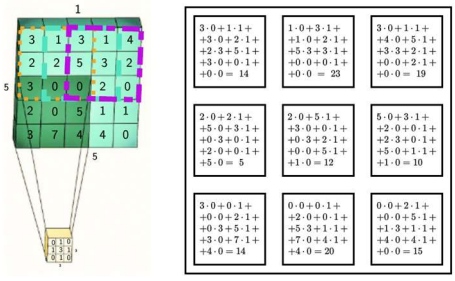

Рис. 2. Пример операции свертки для изображения 5x5x1 с ядром свертки 3x3x1

Ядро свертки (размерности $3\times 3 \times 1$) прикладывается к каждому фрагменту изображения с задаваемым шагом (stride). На Рис.2. приведен пример со значением шага, равным единице. Видно, что каждый нейрон связан лишь с частью пикселей, если речь идет о первом сверточном слое или нейронов, для последующих слоев (это свойство называют разреженной связностью). Необходимым становится обучение лишь значений ядер свертки, что значительно меньше, чем в случае традиционных нейронных сетей (полносвязных).

Как уже было сказано ранее, слова (после векторизации) представляют собой плоские вектора. Поэтому для задач обработки естественного языка применяются одномерные сверточные нейронные сети. Токены, представленные в векторной форме, конкатенируются в матрицу текста. Затем выбирается k строк матрицы, вытягиваются в плоский вектор, к которому «прикладывается» (берется скалярное произведение) одномерное ядро размерности $k \cdot s$, где s – размерность вектора токена. Этот процесс проиллюстрирован на Рис. 3.1. и Рис. 3.2.

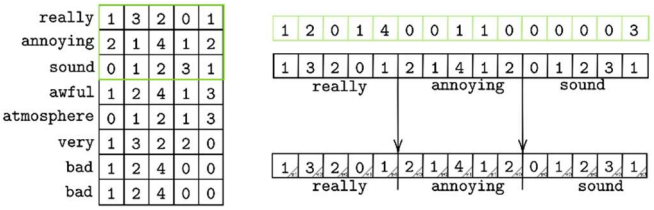

Рис. 3.1. Применение свертки для извлечения признаков из текстов

Двигаясь таким образом по матрице текста, можно получить столбец значений – результатов операции одномерной свертки. Для получения большего количества столбцов, которые, в свою очередь, образуют матрицу признаков, используется несколько ядер.

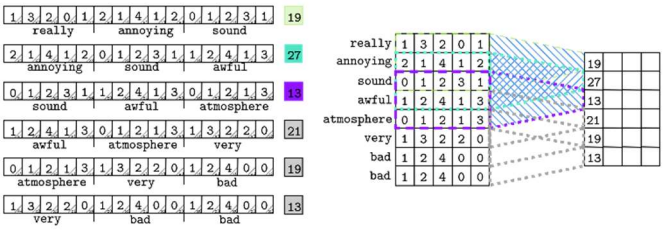

Рис. 3.2. Применение свертки для извлечения признаков из текстов

После применения операции свертки как для изображений, так и для других структур, в том числе и текстов, возможно применение слоя субдискритизации (Pooling, подвыборки). Пулинг работает подобно сверточному слою, однако вместо скалярного произведения применяет другие функции на фрагменты, например, возвращает максимальный или средний выходы в прямоугольной окрестности (max-пулинг [10] и average-пулинг соответственно).

Max-Pooling:

$$f_{MP}(x) = \max_i x_i.$$

Average-Pooling:

$$f_{AP}(x) = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

Логика употребления слоя пулинга заключается в том, что выявленные сверточным слоем закономерности избыточны в своей подробности, значит, возможно уплотнение до менее подробного результата. Однако, если в задаче необходимо сохранять точную пространственную информацию, применение пулинга по всем признакам существенно увеличивает ошибку модели, что исследовано экспериментально [7]. Сверточный слой и слой пулинга, следующий за ним, образуют сверточный блок. Чем больше количество сверточных блоков, тем более абстрактной или высокоуровневой становится карта признаков.

Заключим, что сверточная нейронная сеть достаточно успешно решает задачу извлечения признаков из данных. Экстраполируем этот вывод на одномерную плоскость для работы с текстами: сверточные нейронные сети, извлекая закономерности в

текстовых данных, способны различать контекст. Причем ширину контекста можно приблизительно идеализировать шириной рецептивного поля (или пятна восприятия). На Рис. 4. ширина рецептивного поля на первом слое равна трем, а ширина рецептивного поля на втором слое равна четырем, что позволяет расширить контекст. Однако всё же речь идет о, с точки зрения языка, маленьких паттернах, словосочетаниях и коротких фразах. Для масштабного и более полного моделирования языка необходимо делать очень глубокие сети, что неминуемо приводит к увеличению числа обучаемых параметров.

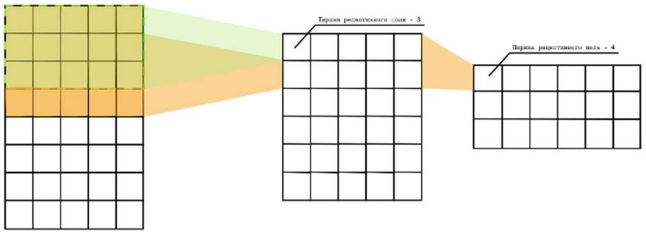

Рис. 4. Ширина рецептивных полей

Есть некоторые способы увеличения контекста, которые являются спорными и требуют аккуратного использования: ранее уже описанный пулинг и прореживание (dilation) [8]. Последнее заключается в применении свертки к фрагменту, из которого удалена часть элементов. Например, возьмем k строк матрицы текста не подряд, а через одну, но значит это то, что мы пропускаем каждое второе слово, что не может однозначно хорошо сказаться на поставленной задаче.

Для решения задачи классификации после череды сверточных блоков (сверточный слой или сверточный слой + пулинг) конечную карту признаков любой размерности, в зависимости от предыдущих слоев, необходимо растянуть в вектор и подать в полносвязный слой с функцией активацией, например, SoftMax (для классификации n-классов, $n \in \mathbb{N}$) или Sigmoid (для бинарной классификации), чтобы получить вероятности классов.

Формула размерности карты признаков, учитывая всевозможные преобразования, после применения сверточного слоя или пулинга:

$$L_{\text{вых}} = \frac{L_{\text{вх}} + 2 \cdot \text{padding} - \text{dilation} \cdot (\text{kernel size} - 1) - 1}{\text{stride}} + 1,$$

где padding (замощение) – параметр, используемый в случае, когда есть необходимости в сохранении размерности при выполнении операции свертки или пулинга: недостающие элементы для этого заполняются нулями.

Итого примерный вид архитектуры сверточной нейронной сети для задачи классификации текстов:

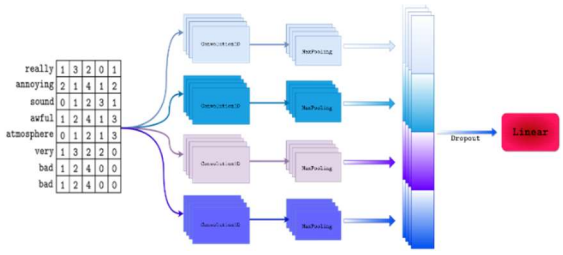

Рис. 5. Архитектура сверточной нейронной сети (4 сверточных слоя, 4 слоя субдискритизации, линейный слой с дропаут)

Таблица 2. Метрика accuracy на разных этапах обучения.

Данные	Модели								
	CNN (1 слой)			CNN (3 + 3 пулинг)			CNN (5+5)		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
IMDB (2)	**0.9996**	0.8757	0.889	0.997	**0.8994**	0.8765	0.9963	0.8991	**0.8996**
Twitter (3)	0.913	0.8338	0.8447	**0.945**	0.8491	0.8312	0.93	**0.8542**	**0.8771**
Yelp (2)	0.5312	0.5671	0.5431	0.5538	0.534	**0.613**	**0.5652**	**0.5843**	0.548
Epoch	5			5			5		

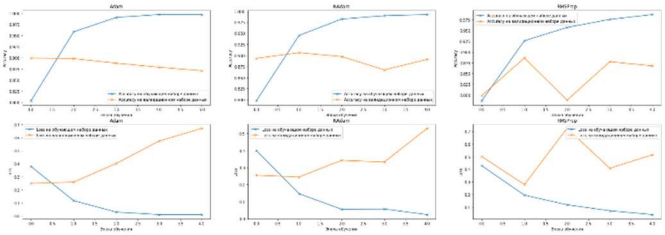

Рис. 6.1. Поведения accuracy и loss в зависимости от оптимизатора

Таблица 3. Затраченное время на разных фреймворках.

Данные	Модели						Объем
	CNN (1 слой)		CNN (3+ 3 пулинг)		CNN (5 + 5)		
	TF	PyTorch	TF	PyTorch	TF	PyTorch	
IMDB (2)	222	81	307	100	383	132	50000
Twitter (3)	385	313	497	433	754	532	75000
Yelp (2)	100421	120212	220053	189429	315371	435341	560000

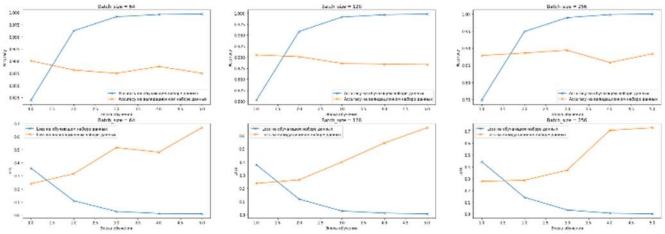

Рис. 6.2. Поведения accuracy и loss в зависимости от batch size

Таблица 4. Метрики качества для датасета IMDB

	precision	recall	F1-score	support
negative	0.87	0.93	0.9	7490
positive	0.93	0.86	0.89	7510
accuracy			0.9	15000
macro avg	0.9	0.9	0.9	15000
weighted avg	0.9	0.9	0.9	15000

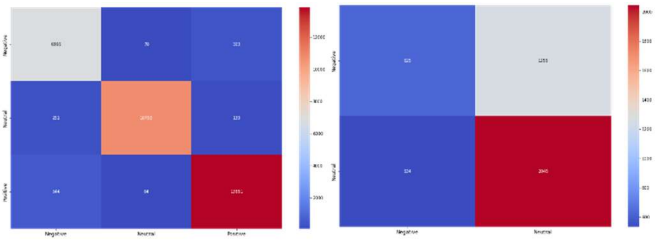

Рис. 6.3. Confusion Matrices для многоклассовой и бинарной классификаций соответственно

Рекуррентные нейронные сети

Ширина рецептивных полей сверточных нейронных сетей позволяет определять частичный контекст текстовой информации: прореживание и пулинг дают возможность увеличить ширину рецептивных полей, однако, использование их может привести к росту ошибки в отдельных задачах. Это не мешает отлично справляться с задачей классификации текстов одномерным сверточным сетям, что доказывают, в первую очередь, результаты, несильно уступающие моделям, написанным на основе правил. Но всё становится значительно хуже при постановке задачи моделирования языка. К тому же, сверточные нейронные сети, как и традиционные, требуют входные данные фиксированной длины, что накладывает ограничение на плоское пространство текстов. В среднем, такие ограничения не оказывают критического влияния, но объемные тексты могут нести важные для классификации детали в той своей части, которая не может быть проанализирована из-за ограничения.

В случае сверточных нейронных сетей, текст позиционировался как пространственная структура. В случае рекуррентных нейронных сетей [4,11] следует рассматривать его как последовательность, информация об элементе которой находится в зависимости от предшествующих ему элементов, другими словами, текст имеет направленное повествование. Подобная структура присуща не только текстам, но также и аудиосигналам или временным рядам. Основная идея рекуррентных нейронных сетей зиждется на введении скрытых состояний, и определении новых состояний через предыдущие. Таким образом, происходит латентное запоминание информации. То есть, в отличие от других нейронных сетей (сверточных, полносвязных), рекуррентная определяет своё состояние не только за счет входных данных, но и за счет предыдущих состояний.

Рис. 7. Развернутая рекуррентная сеть без выходов.

На каждом шаге рекуррентной сети:

1. Прочитать очередной элемент последовательности x^t, применить к нему линейное преобразование:
$$z^t = W_{input} \cdot x_t;$$
2. Вычислить новое значение состояния, исходя из старого:
$$h^t = act(W_{hi} \cdot h^{t-1} + z_t);$$
3. Выйти из рекуррентного шага:
$$y^t = W_{output} \cdot h^t.$$

Обучаемыми параметрами являются веса W_{input}, W_{hidde} , W_{output}, участвующие в линейных преобразованиях. Нулевое состояние сети может задаваться разными способами: исходя из экспериментальных показателей, лучше всего использовать малодисперсный шум [].

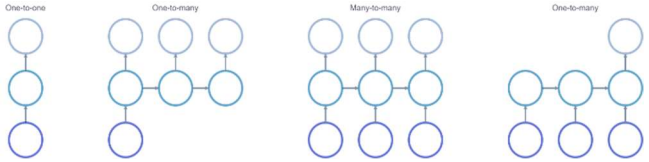

Рис. 8. Архитектуры рекуррентных нейронных сетей.

Рекуррентные нейронные сети не могут подвергнуться параллельным вычислениям, поскольку происходит последовательная обработка данных, как следствие, обучение проходит ощутимо дольше, чем для других типов нейронных сетей.

Но основной проблемой является то, что во время обучения возникают проблемы сходимости: затухание или взрыв градиента. [9]

Затухание и взрыв градиента

Рассмотрим прямой проход (forward pass) по классической рекуррентной нейронной сети (Vanilla RNN) на Рис.9.

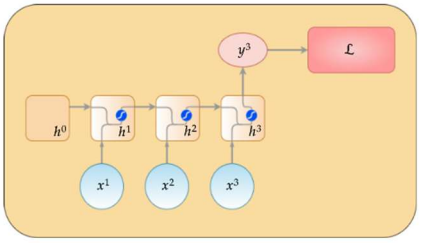

Рис. 9. Vanilla RNN.

Введем обозначения:

Входная последовательность $x^t \in \mathbb{R}$, скрытое состояние $h^t = f(w \cdot h^{t-1} + x^t)$, где f – нелинейная, дифференцируемая функция активации, $w \in \mathbb{R}$ параметр функции перехода; выход $y^t = g(h^t)$, функция потерь (функция ошибки, функционал качества) $\mathcal{L}(y^t)$.

1. $h^0 \in \mathbb{R}$;
2. $h^1 = f(w \cdot h^0 + x^1)$;
3. $h^2 = f(w \cdot h^1 + x^2) = f(w \cdot f(w \cdot h^0 + x^1) + x^2)$
4. $h^3 = f(w \cdot h^2 + x^3) = f(w \cdot f(w \cdot h^1 + x^2) + + x^3) = f(w \cdot f(w \cdot f(w \cdot h^0 + x^1) + x^2) + x^3)$

Выполним обратный проход (back propagation through time [12]):

$$\mathcal{L}(y^3) = \mathcal{L}(g(h^3)) = \mathcal{L}\left(g\left(f(w \cdot h^2 + x^3)\right)\right) = = \mathcal{L}\left(g\left(f(w \cdot f(w \cdot h^1 + x^2) + x^3)\right)\right)$$

Применяя градиентный спуск, нам понадобится производная по параметрам:

$$\frac{\partial \mathcal{L}(y^3)}{\partial w} = \frac{\partial \mathcal{L}(y^3)}{\partial g} \cdot \frac{\partial g}{\partial w} = \frac{\partial \mathcal{L}(y^3)}{\partial g} \cdot \frac{\partial g}{\partial f} \cdot \frac{\partial f(w \cdot h^2 + x^3)}{\partial w}$$

$$\frac{\partial f(w \cdot h^2 + x^3)}{\partial w} = f'(w \cdot h^2 + x^3) \cdot (w \cdot h^2 + x^3)' = = f_3' \cdot (w \cdot h^2)' = f_3' \cdot \left(w' \cdot h^2 + w \cdot h^{2'}\right) = f_3' \cdot (h^2 + w \cdot f(w \cdot h^1 + x^2)') =$$

$$= f_3' \cdot \left(h^2 + w \cdot f_2' \cdot (h^1 + w \cdot f(w \cdot h^0 + x^1)')\right) = = \sum_{i=1}^{3} \left(h^{i-1} \cdot w^{3-i} \prod_{j=i}^{3} f_j' \right)$$

Именно это умножение является причиной затухания или взрыва градиента. Одной из часто употребляемых функций активации является гиперболический тангенс, производная которого лежит в диапазоне $0 < tanh' < 1$.

Умножение большого числа таких значений приведет к затуханию градиента (vanishing gradient) [12]. Борьба с этим – ограничение количества скрытых состояний, что, по сути, ограничивает последовательности. Основная идея этих сетей теряет смысл.

Эту проблему можно интерпретировать как заполненность последовательности маловажными в рамках задачи токенами, которые требуют маленькие веса, участвующие в этом произведении и, как следствие, ухудшают оптимизацию (происходит затухание градиента).

С другой стороны, если модуль произведения больше единицы, то произойдет взрыв градиента (переполнение и падение точности). Борьбой с этим является ввод фиксированной границы градиента, до которой снижается градиент, ее превысивший.

Затухающий градиент так же связан с долгосрочными зависимостями [13]. Любые флуктуации в краткосрочных зависимостях будут подавлять долгосрочные. В случае, если модель способна представить долгосрочные зависимости, градиент долгосрочного взаимодействия будет экспоненциально ниже краткосрочного. [5]

LSTM

Борьба между мощностью и обучаемостью рекуррентных

нейронных сетей неразрывно связана с поведением градиента при их обучении. В 1990-е годы было предложено немало способов решения этой проблемы, среди которых временные задержки внутри состояний [14], временные постоянные. Но наиболее часто ныне используемым решением стала архитектура LSTM [4, 17].

Сеть LSTM использует концепцию вентильных нейронных сетей (gated Recurrent Neural Networks), основанную на идее контролированного выбора путей между состояниями. Путей, на которых не обнуляются и не устремляются в бесконечность производные. Веса связей скрытых состояний в такой структуре могут изменяться на каждом временном (или пространственном, зависит от структуры анализируемых данных) шаге.

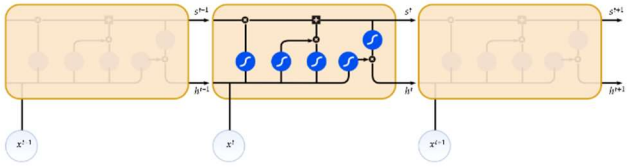

Рис. 10.1. Срез развернутого LSTM слоя.

Идея вентилей состоит в том, чтобы контролировать вектор значений за счет его умножения на вектор шлюза (вентиля), который управляет потоком ошибки. Целью является сохранение постоянного объема потока ошибки.

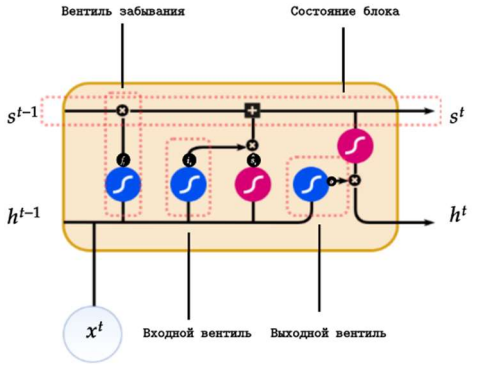

Рис. 10.2. LSTM блок.

Запишем формальный прямой ход LSTM блока в момент времени t:

- Конкатенируем h^{t-1} и x^t. Применяем линейное преобразование и нелинейную функцию активации:

$$f_t = \sigma\left(W_f \cdot [h^{t-1}, x^t] + b_f\right);$$
$$i_t = \sigma(W_i \cdot [h^{t-1}, x^t] + b_i);$$
$$\tilde{s}^t = \text{act}(W_{\tilde{s}} \cdot [h^{t-1}, x^t + b_{\tilde{s}}]);$$
$$o_t = \sigma(W_o \cdot [h^{t-1}, x^t] + b_0);$$

  act – функция активации (например, $tanh$);
- Получим сигнал как линейную комбинацию:

$$s^t = f_t \cdot s^{t-1} + i_t \cdot \tilde{s}^t;$$

- Выходим из рекуррентного шага:

$$h^t = o_t \cdot tanh(s_t).$$

f_t – вентиль забывания (forgetting gate), обучаемый за счет весов W_f и сдвига b_f, определяющий насколько нужно забыть состояние с прошлого блока.
i_t – входной вентиль (input gate), обучаемый за счет весов W_i и сдвига b_i, ответственный за чувствительность ко входу: определяет, насколько важно для запоминания полученное новое состояние $\tilde{s}^t$. Состояние $\tilde{s}^t$ в рекуррентных сетях без вентильной парадигмы было бы выходом блока.
o_t – выходной вентиль (output gate), применяемый для увеличения точности аппроксимации нейронной сети за счет нелинейности функции активации.

За счет того, что сигмоида принимает значения в диапазоне $0 < \sigma < 1$, все значения вентилей находятся в этом же диапазоне, что позволяет управлять амплитудой значений состояний, не меняя знак и контролируя объем потока ошибки.

Внутренние пируэты с новыми и старыми состояниями позволяют классифицировать их важность, как следствие настроить запоминание и забывание того, что нужно для решения задачи и бесполезно, соответственно. LSTM достаточно успешно представляет долгосрочные зависимости, моделирует язык, позволяет решать широкий спектр задач, но не лишена проблем. За счет появления новых внутренних обучаемых весов и сдвигов, увеличивается количество обучаемых параметров в 4 раза по сравнению с обычной рекуррентной нейронной сетью.
LSTM часто переобучается и может показывать плохие результаты на новых данных.

Таблица 5. Метрика accuracy на разных этапах обучения

Данные	Модели								
	LSTM (1 блок)			LSTM (2 блока)			CNN + LSTM (1 слой + 1 пулинг + 1 блок)		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
IMDB (2)	0.9968	0.8809	0.8573	0.9533	**0.8834**	0.8673	**0.9982**	0.88	**0.8765**
Twitter (3)	**0.855**	0.84	0.778	0.8431	0.8343	**0.841**	0.814	**0.8542**	0.8175
Yelp (2)	**0.9172**	**0.9472**	**0.945**	0.91	0.864	0.8621	-	-	-
Epoch	13			10			10		

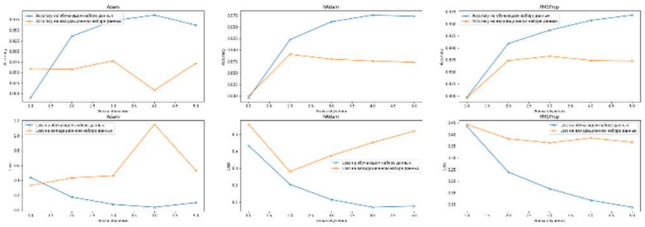

Рис. 11.1. Поведения accuracy и loss в зависимости от оптимизатора.

Таблица 6. Затраченное время на разных фреймворках.

Данные	Модели						Объем
	LSTM (1 блок)		LSTM (2 блока)		LSTM + CNN		
	TF	PyTorch	TF	PyTorch	TF	PyTorch	
IMDB (2)	225	201	8400	8513	1800	1680	50000
Twitter (3)	541	650	28327	31652	3366	3254	75000
Yelp (2)	432002	542331	479032	492151	-	-	560000

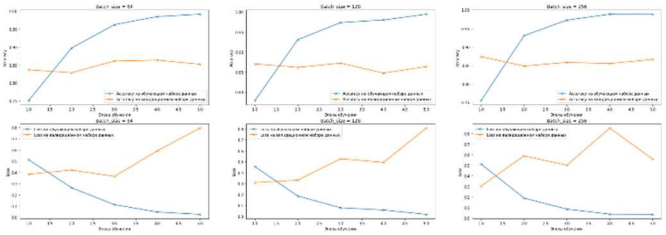

Рис. 11.2. Поведения accuracy и loss в зависимости от batch size.

Таблица 7. Метрики качества для датасета IMDB

	precision	recall	F1-score	support
negative	0.96	0.91	0.93	7490
positive	0.93	0.86	0.89	7510
accuracy			0.93	15000
macro avg	0.94	0.93	0.93	15000
weighted avg	0.94	0.93	0.93	15000

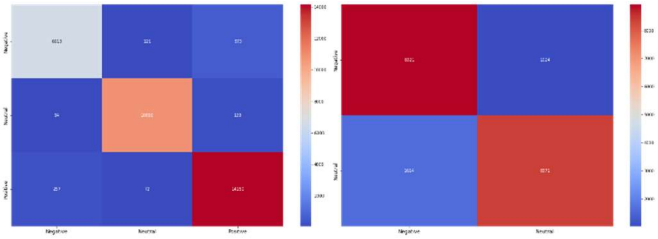

Рис. 11.3. Confusion Matrices для многоклассовой и бинарной классификаций соответственно

GRU

Уменьшить количество обучаемых параметров без сильной потери качества удалось сети GRU (gated recurrent unit) [15]. Основная идея остается той же – контроль постоянства объема потока ошибки.
В сети GRU один вентиль управляет и коэффициентом забывания, и решением об обновлении блока состояния:

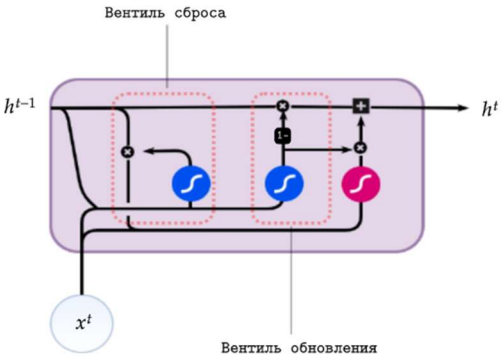

Рис. 12.2. GRU блок.

Запишем формальный прямой ход LSTM блока в момент времени t:

- Конкатенируем h^{t-1} и x^t. Применяем линейное преобразование и нелинейную функцию активации:

$$u_t = \sigma\big(W_z \cdot (x^t, h^{t-1})\big);$$
$$r_t = \sigma\big(W_r \cdot (x^t, h^{t-1})\big);$$
$$\tilde{h}^t = act\big(W \cdot (x^t, r_t \cdot h^{t-1})\big)$$

  act – нелинейная функция активации (например, $tanh$);
- Получим сигнал как линейную комбинацию:

$$h^t = (1 - u_t) \cdot h^{t-1} + z_t \cdot \tilde{u}_t;$$

- Выходим из рекуррентного шага:

u_t – вентиль обновления (update gate), решающий на каждом состоянии, оставить предыдущее значение или обновить.
r_t – вентиль «сброса» (reset gate), позволяющий контролировать поведение обновленного состояния. В результате, количество обучаемых параметров сократилось на треть (до порядка $6d^2$). Сеть стала значительно проще в обучении, не потеряв мощности.

Таблица 8. Метрика accuracy на разных этапах обучения.

Данные	Модели								
	GRU (1 блок)			GRU (2 блока)			CNN + GRU (1 слой +5 пулинг + 1 блок)		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
IMDB (2)	0.9751	**0.9612**	**0.9353**	0.9903	0.8763	0.8931	**0.9969**	0.8851	0.9184
Twitter (3)	0.8788	0.8338	0.8447	**0.9937**	0.9605	0.9574	0.93	**0.8542**	**0.8771**
Yelp (2)	**0.9670**	**0.9562**	**0.9459**	0.954	0.943	0.9126	-	-	-
Epoch		5			5			5	

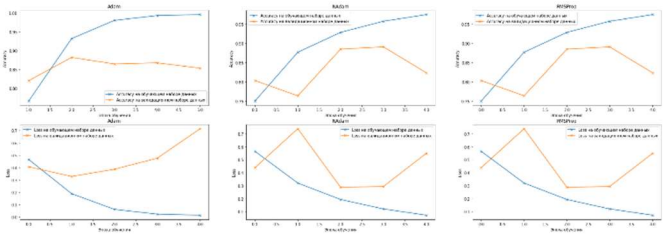

Рис. 13.1. Поведения accuracy и loss в зависимости от оптимизатора.

Таблица 9. Затраченное время на разных фреймворках.

Данные	Модели						Объем
	GRU (1 блок)		GRU (2 блока)		GRU + CNN		
	TF	PyTorch	TF	PyTorch	TF	PyTorch	
IMDB (2)	189	204	569	642	244	183	50000
Twitter (3)	546	498	759	802	650	663	75000
Yelp (2)	433200	4542233	679036	592431	-	-	560000

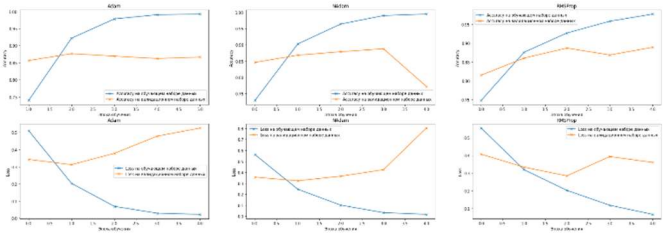

Рис. 13.2. Поведения accuracy и loss в зависимости от batch size.

Таблица 10. Метрики качества для датасета IMDB

	precision	recall	F1-score	support
negative	0.93	0.97	0.95	7490
positive	0.97	0.93	0.95	7510
accuracy			0.95	15000
macro avg	0.95	0.95	0.95	15000
weighted avg	0.95	0.95	0.95	15000

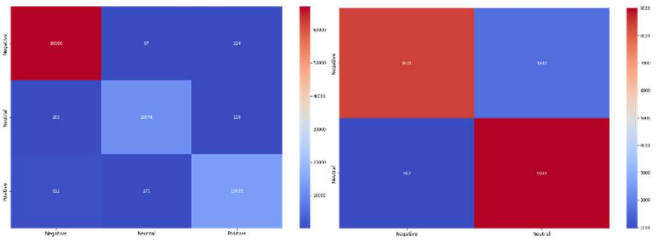

Рис. 13.3. Confusion Matrices для многоклассовой и бинарной классификаций соответственно

Stacking

Полученные обученные модели показывают хорошие результаты, однако их можно дополнительно улучшить с помощью технологий стэкинга. Идея заключается в создании метамодели, которая будет по результатам мета-признаков моделей давать конечное предсказание. Обучение моделей для создания мета-модели проходило по принципу K-Foldation.

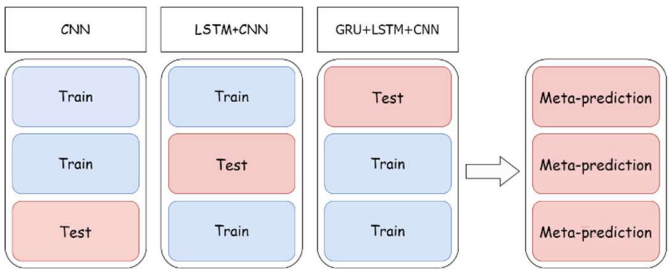

Рис. 14.1. K-foldation.

Тренировочный датасет объемом 1.5 миллионов отзывов (результат присоединения датасетов Amazon и IMDB), был разбит на три части, на двух из которых каждая модель обучалась, затем у нее убирался полносвязный слой с sigmoid функцией активации, чтобы получить признаковое описание начальных данных сложной структуры в виде табличной структуры, а на третьей части каждая модель давала предсказания без пересечения между собой. Мета-предсказания, будучи табличными данными, были адресованы в классификаторы Random Forest, GBM, SVM, обученные по принципу кросс-валидации.

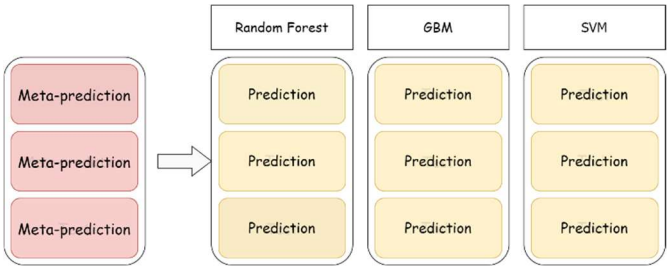

Рис. 14.2. Meta-Models.

Конечный результат получается усреднением мета-предсказаний мета-моделей.

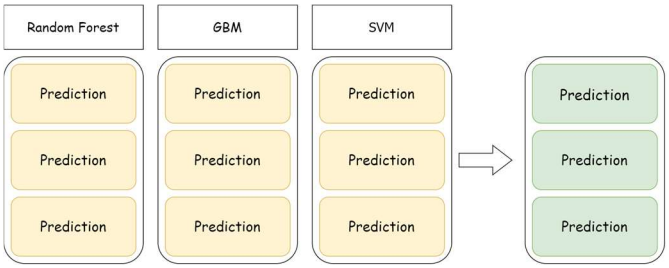

Рис. 14.3. Prediction.

Полученная мета-модель работает несколько лучше других обученных алгоритмов. Итоговая метрика качества accuracy = 0.965.

Заключение

В этой статье для решения задачи классификации текстов были реализованы архитектуры: одномерные сверточные нейронные сети, сети LSTM и GRU. Кроме того, рассмотрен метод ансамблирования моделей, несколько повысивший качество обученных алгоритмов. Обучение моделей происходило с применением технологий регуляризации Batch Normalization [6] и Dropout [8]. Обученные модели показывают отличные результаты, но они не лишены индивидуальных проблем. Одномерные сверточные нейронные сети способны моделировать контекст шириной в несколько слов, словосочетаний, предложений, чтобы увеличить ширину контекста, необходимо делать сети намного более глубокими, что увеличивает количество обучаемых параметров. Это неминуемо приводит к увеличению длительности обучения. Для рекурентных нейронных сетей нет проблемы контекста, поскольку они способны латентно запоминать прошедшую через их состояния информацию, однако проблемы затухающего градиента ухудшают или даже делают невозможным обучение. LSTM решает эту проблему, но имеет в 4 раза больше параметров, чем традиционные рекурентные сети, что замедляет обучение. Как следствие, можно сделать вывод, что самым стабильным по качеству классификатором текстов можно считать GRU, не уступающий по качеству LSTM, однако превосходящий его по скорости обучения. Рассмотренное ансамблирование способно сделать результаты моделей еще более уверенными.

Литература

1. Y. LeCun and Y.Bengio, Convolutional networks for images, speech and time-series. Brain Theory and Neural Networks, 1995;
2. Y. LeCun, K. Kavukcuoglu and C. Farabet. Convolutional networks and applications in vision. In Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, pages 253-256. IEEE, 2010;
3. Y. LeCun, B.Boser, J.S. Denker, D. Henderson, R.E.Howard, W. Hubbard, L.D. Jackel, et al. Handwritten digit recognition with a back-propagation network. In Advances in neural information processing systems, 1990;
4. J. Schmidhuber, “Deep learning in neural networks: an overview”, Neural networks, vol. 61, pp. 29-33, 2015
5. “Deep learning” by Ian Goodfellow, Yoshua Bengio, Aaron Courville, pages 283-308, 2017.
6. Ioffe, S. and Szegedy. C. Batch normalization: Acceleration deep network training by reducing internal covariate shift, 2015;
7. Szegedy, C., Liu, W., Jia, Y., Sermanet, P. Reed, S. Anguelov, D. Erhan, D. Vanhoucke, V. and Rabinovich A. Going deeper with convolutions. Technical report, 2014;
8. Srivastava N., Hinton G., Krizhevsky A, Sutskever I., Salakjutdinov R. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 2014, 15, 1929-1958;
9. B. Hammer, On the Approximation Capability of Recurrent Neural Networks. Neurocomputing, 31(1-4):107-123, 2000;
10. Zhou, Y. and Chellappa R. Computation of optical flow using a neural network. In Neural Networks, 1988.
11. Sepp Honchreiter, Jurden Shmidhuber, Long Short-Term Memory. Neural Comput 1996; 9 (8): 1735-1780;
12. R. J. Williams and D. Zipser. Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity. In Y. Chauvin and D. E. Rumelhart, editors, Back-propogation: Theory, Architectures and Applications, pages 433-486. Lawrence Erlbaum Publishers, 1995
13. S. Hochreiter. Y. Bengio, P. Frasconi and J. Schmidhuber. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-term Dependencies. In S.C. Kremer and J.F. Kolen, editors, A Field Guide to Dynamical Recurrent Networks. IEEE Press, 2001a;
14. K. J. Lang, A. H. Waibel and G. E. Hinton. A Time-delay Neural Network Architecture for Isolated Word Recognition. Neural Networks, 3(1):23-43, 1990;
15. Jozefowicz R., Zaremba W. and Sutskever I. An empirical evaluation of recurrent network architectures. In ICML2015, 2015;