

УДК 004.85

DOI 10.52167/1609-1817-2025-138-3-311-321

Д.С. Харченко, В.А. Мадин, О.С. Салыкова, И.В. Иванова

Baitursynuly University, Костанай, Казахстан

E-mail: vmadin@mail.ru

## АНАЛИЗ МОДЕЛЕЙ RNN И TRANSFORMER В ЗАДАЧАХ ОБРАБОТКИ ЕСТЕСТВЕННОГО ЯЗЫКА

**Аннотация.** Задачи обработки текста и генерации текста являются ключевыми задачами обработки естественного языка. Существует множество различных методов, подходов и моделей, направленных на выполнение данных задач. В статье проведен сравнительный анализ актуальных моделей искусственного интеллекта, таких как RNN и Transformer, которые применяются в области обработки естественного языка и выполняют функции обработки и генерации текстов. Впоследствии анализа которых были выявлены сильные и слабые стороны данных моделей в эффективности обработки естественного языка.

**Ключевые слова:** искусственный интеллект, нейронные сети, рекуррентные нейронные сети, RNN, трансформер, Transformer.

### Введение.

Искусственный интеллект (ИИ) проник во многие сферы нашей жизни. На основе технологий ИИ работают современные поисковые системы. ИИ помогает службам безопасности ловить преступников, а медикам проводить анализ заболеваний. В маркетинге используются рекомендательные алгоритмы, основанные на ИИ. В медиа индустрии ИИ уже применяется для генерации красочных изображений, иллюстраций и человекоподобной речи. Несмотря на то, что с каждым годом ИИ становится всё продуктивнее, в сфере обработки естественного языка (NLP) он имеет ряд существенных недостатков. Эти проблемы заключается в многозначности языка, сложности работы с контекстом и эмоциональном анализе текста. Решением этих проблем занимаются исследователи и инженеры по всему миру. Опираясь на имеющиеся теоретические публикации ученых, в том числе на отдельные работы Дафна Коллер «Support Vector Machine Active Learning With Applications To Text Classification» [1], Юрген Шмидхубер «Long Short-Term Memory, in Neural Computation» [2], Эндрю Блэн и Майкл Джордан «Latent Dirichlet Allocation» [3], Ашиш Васвани и Ники Пармар «Attention Is All You Need» [4], эти и многие другие ученые вносят свой вклад в работу по улучшению методов, подходов и моделей ИИ.

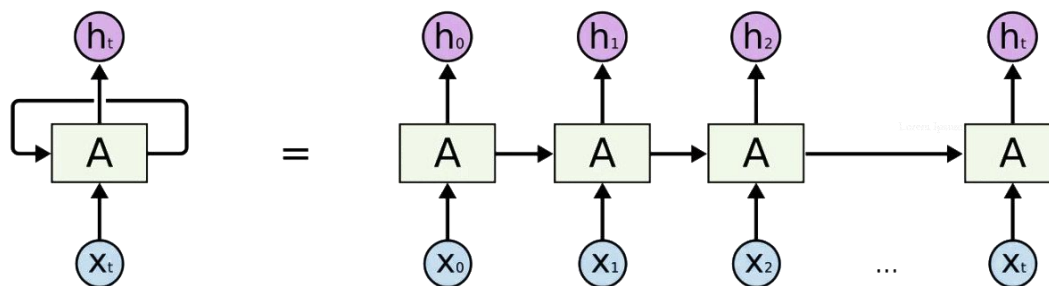


Рисунок 1 - Архитектура RNN

На данный момент в области обработки естественного языка большую популярность набирает модель трансформер, которая была разработана на основе нейронных сетей и которая составляет конкуренцию предшествующей популярной модели рекуррентных нейронных сетей.

Рекуррентные нейронные сети (RNN), впервые представленные в работе Дэвида Румельхарта, Джеймса Хинтона и Рональда Уильямса [5], представляют собой класс нейронных сетей, спроектированных для работы с последовательными данными и учёта зависимостей в различных временных шагах [6], [7]. В RNN скрытое состояние с предыдущего временного шага передается обратно в сеть (рисунок 1), позволяя ей сохранять информацию о предыдущих состояниях, что делает их особенно эффективными в задачах обработки естественного языка, временных рядов, и других приложениях, где важно учитывать контекст.

Наиболее часто используемыми разновидностями RNN являются LSTM и GRU [2], [6]. LSTM – сети с кратковременной памятью – представляют собой разновидность архитектуры, предназначенной для борьбы с проблемой затухания и взрыва градиента, и обладают специальной архитектурой для эффективной работы с памятью [2], [8]. По своей структуре LSTM представляют собой цепочку, состоящую из LSTM-модулей, включающих в себя несколько структур (вентилей), пропускающих через себя информацию определенным образом (рисунок 2).

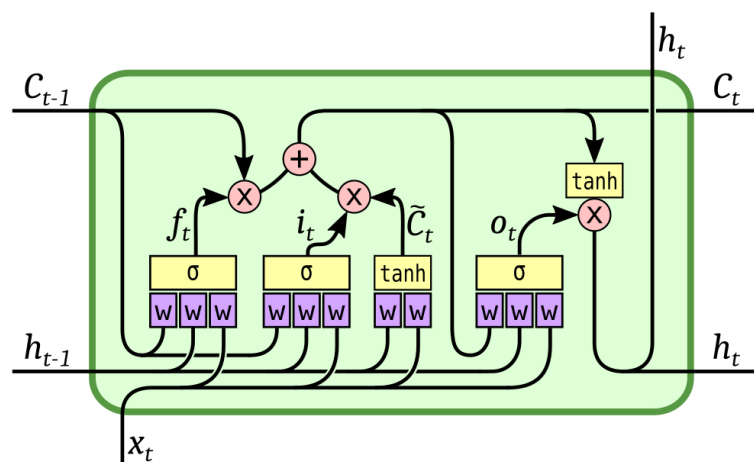


Рисунок 2 - Архитектура LSTM

Архитектура GRU – управляемые рекуррентные блоки – похожа на LSTM версия, но более простая и с меньшим количеством параметров. Модель GRU во многом схожа с LSTM, однако имеет меньше вентилей (рис. 3) и за счет этого ее обучение происходит быстрее [9], [10].

RNN и в особенности их разновидности GRU и LSTM хорошо подходят для задач с постоянно изменяющимися данными [6], [11]. Из-за обратных связей рекуррентные нейронные сети обладают своего рода памятью и это позволяет создавать различные языковые модели на их основе.

Однако архитектура RNN имеет ряд проблем. Во-первых, RNN обрабатывают данные последовательно и из-за этого, при обработке длинных и сложных предложений сеть «забывает» то, что было в начале. Даже использование архитектур LSTM или GRU не могут полностью решить эту проблему. Во-вторых, архитектура не поддерживает параллельную обработку данных из-за чего невозможно ускорить процесс обучения. В-третьих, RNN модели всё ещё сталкиваются с проблемой исчезающего градиента, что создаёт сложности в их обучении на большом количестве данных.

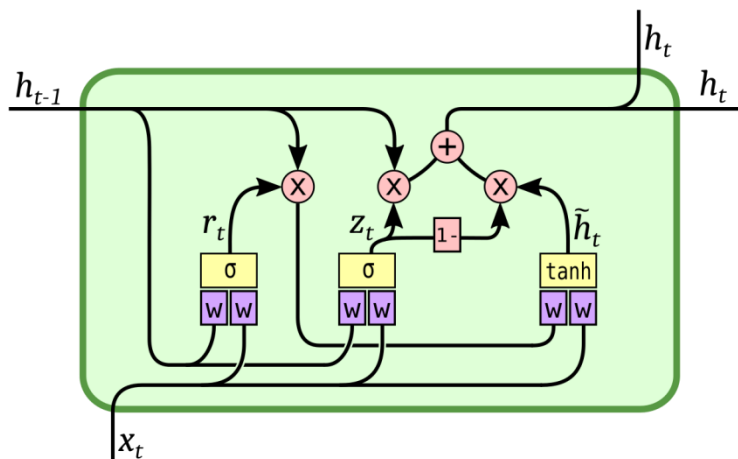


Рисунок 3 - Архитектура GRU

Всё изменилось с внедрением в область NLP Трансферного обучения (Transfer Learning) - механизма машинного обучения, идея которого заключается в «переносе» накопленных знаний из одной задачи в другую [12]. Проблема непонимания семантической нагрузки слов и предложений привела разработчиков к попытке применить алгоритмы трансферного обучения к моделям NLP. Таким образом в 2017 году разработчиками из компании Google была разработана новая архитектура Transformer, представленная в статье «Attention Is All You Need» [4]. А уже в 2018 году были созданы первые модели NLP с архитектурой Transformer: Generative Pre-Training (GPT, генеративная модель предварительного обучения) и Bidirectional Encoder Representation Transformers (BERT, двунаправленная нейронная сеть-кодировщик) [13], [14], [15].

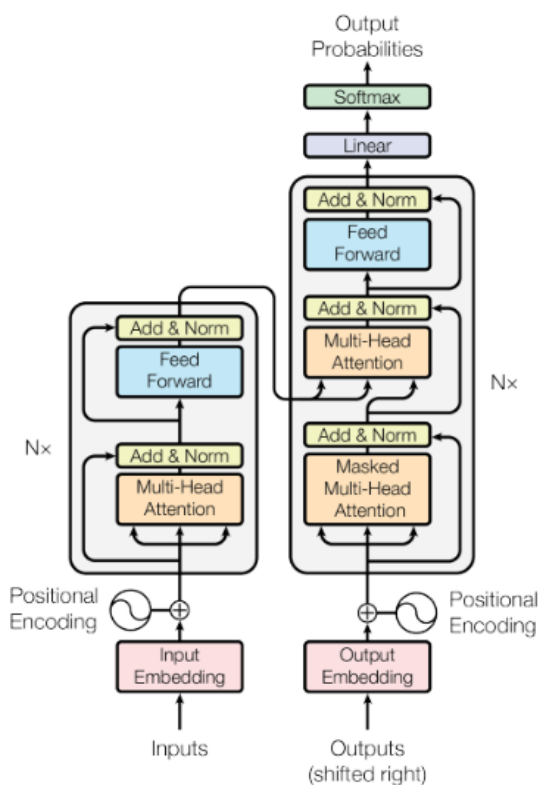


Рисунок 4 - Архитектура Transformer

Трансформеры представляют собой более свежую архитектуру нейронной сети, представленную для улучшения ограничений RNN [4], [13]. Вместо того, чтобы полагаться на последовательную обработку входных данных, как RNN, трансформеры используют механизмы внимания для взвешивания важности различных элементов в последовательности ввода (рисунок 4). Эти механизмы внимания позволяют обрабатывать входные данные более эффективно и точно, чем RNN, что приводит к повышению производительности во многих задачах обработки естественного языка. Более того, трансформеры можно легко распараллелить во время обучения, что способствует более быстрому времени вычислений по сравнению с RNN.

В 2019 году для обработки текстов на естественном языке было предложено использовать новую нейронную архитектуру Transformer-XL. Классические Трансформеры обладают потенциалом обучения долгосрочной последовательности, но ограничены контекстом с фиксированной длиной. Архитектуру Transformer-XL в свою очередь позволяет изучать зависимость за пределами фиксированной длины без нарушения временной когерентности, а также решает проблему фрагментации контекста. Она состоит из механизма повторения на уровне сегментов и новой схемы позиционного кодирования. Наиболее известной из языковых моделей на основе архитектуры Transformer-XL является модель XLNet. XLNet – это модифицированная цель обучения языковых моделей, базирующаяся на условных распределениях перестановок результатов токенизации в заданной последовательности. Иначе говоря, метод XLNet не рассматривает каждое отношение отдельно, а производит выборку из всех возможных перестановок позволяя более качественно решать различные задачи NLP [13], [16].

#### **Материалы и методы.**

В нашей реализации мы использовали рекуррентные нейронные сети и трансформер. Проверка эффективности данных архитектур проходила в областях классификации слов и классификации текстов. В ходе работы модели решали пару поставленных перед ними задач:

- 1) Классификация имен по признаку языка.
- 2) Классификация отзывов по негативной/позитивной оценке.

Оценка эффективности архитектур проходила методом сравнения по трём параметрам: точность (насколько точно модель определяет правильный ответ), потери (насколько модель при неправильном ответе была далека от правильного) и время обучения.

Для программной реализации всех моделей с целью проверить их эффективность в решении задач NLP мы написали код на языке Python, используя возможности библиотеки PyTorch. PyTorch – это фреймворк машинного обучения на языке Python включающий в себя различные архитектуры и готовые модели нейронных сетей. Также для обучения и тестирования моделей использовались датасеты из специальной библиотеки машинного обучения – scikit-learn. Для работы с датасетами формата CSV использовались возможности библиотеки pandas. Для построения графиков использовалась библиотека matplotlib.

#### **Результаты.**

Ход решения первой задачи, классификации имен, строился следующим образом. Для начала был взят dataset с именами на 18 языках. Каждое имя было преобразовано и преобразована в Тензор - векторный набор данных понятный для нейросетевых моделей:

```
def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor
```

Рисунок 5 - Преобразование строки в тензор

Далее были реализованы экземпляры моделей RNN и Transformer только с кодирующим слоем, так как использование декодирующего слоя для решения данной задачи не требуется (рисунки 6, 7).

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.h2o = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.h2o(hidden)
        return self.softmax(output), hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 64
rnn = RNN(n_letters, n_hidden, n_categories)
```

Рисунок 6 - Модель RNN

Далее шла тренировка и тестирование моделей. И на выходе мы получили следующие результаты работ моделей, показанные на рисунках 8 и 9. Исходя из результатов тестирования видно, что модель RNN решает эту задачу быстрее, чем Transformer почти в 5 раз и выдает ответ на 10% точнее. В решении данной задачи Transformer показал себя крайне медлительной и неэффективной моделью, что связано с его усложненной архитектурой, которая учитывает большое количество параметров и которой сложно работать с таким маленьким набором параметров как имена.

# Модель Transformer

```
class TransformerModel(nn.Module):
    def init (self, input_size, output_size, d_model=64, nhead=16, num_encoder_layers=1):
        super(TransformerModel, self). init ()
        self.embedding = nn.Embedding(input_size, d_model)
        self.transformer_encoder = nn. TransformerEncoder(
            nn. TransformerEncoderLayer(d_model, nhead),
            num_layers=num_encoder_layers
        )
        self.fc = nn.Linear(d_model, output_size)

    def forward(self, src):
        src = self.embedding(src)
        src = src.permute(1, 0, 2)
        output = self.transformer_encoder(src)
        output = self.fc(output[-1, :])
        return output
```

Рисунок 7 - Модель Transformer

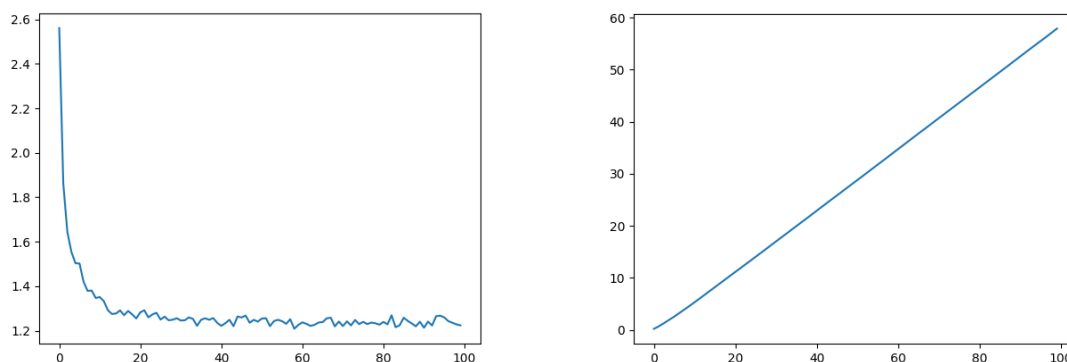


Рисунок 8 - Результат модели RNN. Слева график потерь, справа график точности

В свою очередь модель RNN имеет более простую архитектуру и способна эффективно работать с малым набором параметров. Из результата следует вывод, что для решения задач с небольшим набором параметров, таких как классификация имён, эффективнее всего использовать модели на основе RNN.

Решение второй задачи, классификация отзывов по негативной/позитивной оценке, проходило следующим образом. В качестве материала для обучения и тестирования модели была использована база данных IMDB, представляющая собой файл формата CSV, которая содержит в себе большой объём отзывов и их оценки, позитивной или негативной.

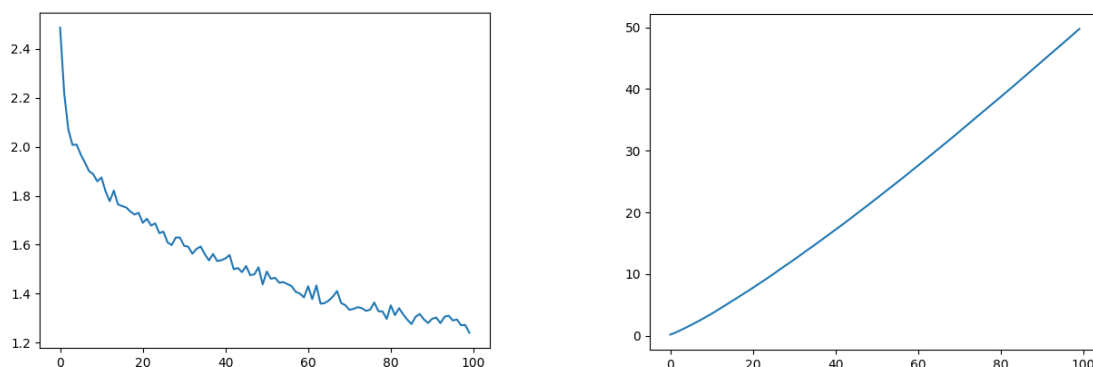


Рисунок 9 - Результат модели Transformer. Слева график потерь, справа график точности

С помощью библиотеки pandas эти данные были считаны и также преобразованы в тензор. Для решения этой задачи использовалась модифицированная модель RNN LSTM и классическая не урезанная модель Transformer:

```
# Преобразование в тензор
class TextDataset(Dataset):
    def init (self, texts, labels):
        self.texts = texts
        self.labels = labels

    def len (self):
        return len(self.texts)

    def getitem (self, idx):
        return torch.FloatTensor(self.texts[idx]),
        torch.LongTensorUseful.labels[idx]])

text_data_train = vectorizer.fit_transform(texts_train).toarray()
text_data_test = vectorizer.fit_transform(texts_test).toarray()

#Создание DataLoader
b_size = 1
train_dataset = TextDataset(text_data_train, labels_train)
test_dataset = TextDataset(text_data_test, labels_test)
train_loader = DataLoader(train_dataset, batch_size=b_size, shuffle=--_e
test_loader = DataLoader(test_dataset, batch_size=b_size, shuffle==alsE)

# Модель RNN
class RNN(nn.Module):
    def init (self, in_layer, emb_layer, hid_layer, out_layer, num_layers,
dropout):
        super(RNN, self). init ()
        self.embedding = nn.Embedding(in_layer, emb_layer)
        self.lstm = nn.LSTM(emb_layer, hid_layer, num_layers,
dropout=dropout, batch_first=True)
```

```

        self.fc = nn.Linear(hid_layer, out_layer)
    def forward(self, x, lengths):
        x = x.type(torch.long)
        embedded = self.embedding(x)
        packed = pack_padded_sequence(embedded, lengths.cpu(),
enforce_sorted=False)
        output, _ = self.lstm(packed)
        output, _ = pad_packed_sequence(output, batch_first=True)
        output = self.fc(output[:, -1, :])
        return output

model = RNN(in_layer emb_layer, hid_layer, out_layer, num_layers,
dropout).to(device) criterion = nn.BCEwithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.000035)

# Модель Transformer
class TransformerModel(nn.Module):
    def __init__(self, in_dim, emb_dim, hid_dim, out_dim, n_layers,
n_heads, dropout):
        super(TransformerModel, self).__init__()
        self.embedding = nn.Embedding(in_dim, emb_dim)
        self.transformer = nn.Transformer(
            emb_dim,
            nhead=n_heads,
            num_encoder_layers=n_layers,
            num_decoder_layers=n_layers,
            dim_feedforward=hid_dim,
            dropout=dropout
        )
        self.fc = nn.Linear(emb_dim, out_dim)

    def forward(self, src, src_mask):
        embedded = self.embedding(src)
        embedded = embedded.permute(1, 0, 2)
        output = self.transformer(embedded, embedded,
src_key_padding_mask=src_mask)
        output = output.permute(1, 0, 2)
        output = self.fc(output)
        return output

model = TransformerModel(in_layer, emb_layer, hid_layer, out_layer, n_layers,
n_heads, dropout).to(device)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.06101)

```

В результате при решении данной задачи модель Transformer показала себя с лучшей стороны справившись с задачей в 5 раза быстрее чем RNN, и при этом в точности определения правильных ответов обошла модель RNN на 5%. Все данные показаны на рисунках 10 и 11.

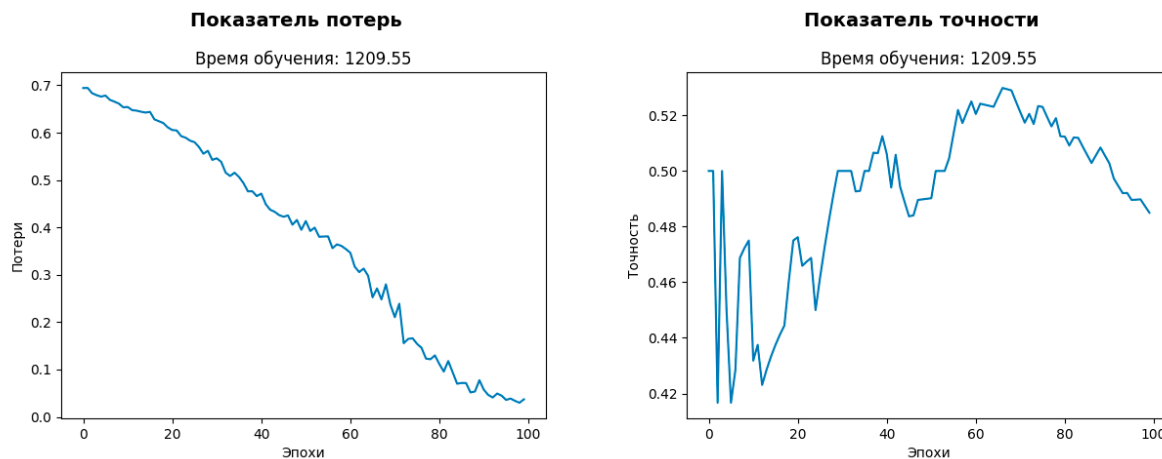


Рисунок 10 - Результат модели RNN

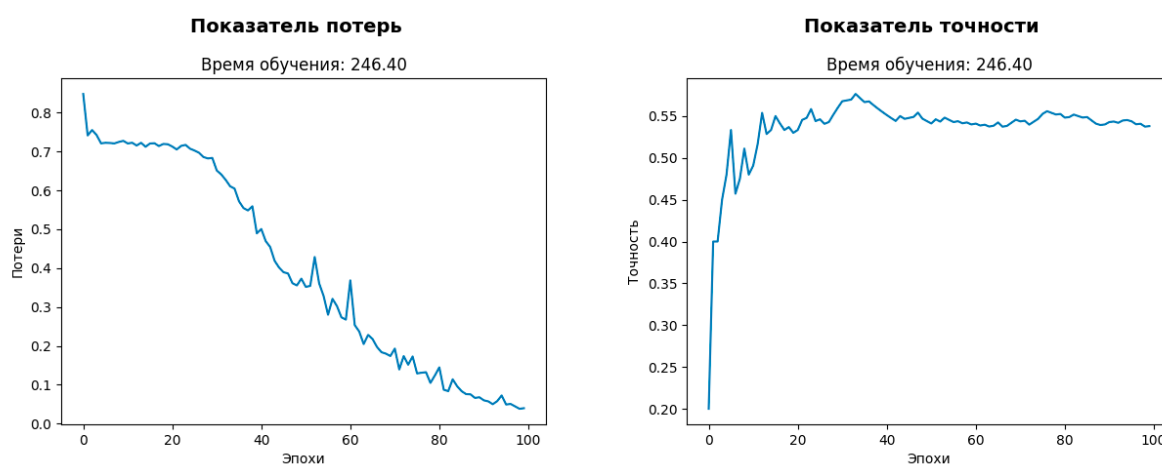


Рисунок 11 - Результат модели Transformer

По результатам решения задачи можно сделать вывод, что модель Transformer благодаря преимуществам своей архитектуры лучше справляется с обработкой текстов, чем модель RNN. Этот пример показывает, что в решении задач с обработкой текстов эффективнее будет применять модель Transformer.

### **Заключение.**

Исследование моделей RNN и Transformer, проведённое в рамках данной работы, выявило их сильные и слабые стороны в контексте задач классификации имён и определения тональности отзывов. Обе модели демонстрируют высокую эффективность в решении задач обработки естественного языка, однако выбор конкретной модели должен основываться на специфике задачи и объёме доступных данных.

Анализ показал, что для задач, характеризующихся малым объёмом входных данных, предпочтение следует отдавать модели RNN ввиду её способности эффективно обрабатывать такие наборы данных. В то же время, когда речь идёт о работе с большими объёмами текстовой информации, модель Transformer выявила себя как наиболее подходящий вариант, особенно в контексте создания больших языковых моделей. Это обусловлено её способностью к параллельной обработке данных и высокой эффективностью в извлечении контекстуальных зависимостей на больших дистанциях в тексте.

Таким образом, результаты нашего исследования подчёркивают важность выбора подходящей модели искусственного интеллекта для конкретной задачи NLP, учитывая характеристики обрабатываемых данных.

## ЛИТЕРАТУРА

- [1] Tong, Simon & Koller, Daphne. (2001). Support Vector Machine Active Learning with Applications to Text Classification. *The Journal of Machine Learning Research*. 2. 45-66. 10.1162/153244302760185243.
- [2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
- [3] Blei, David M., Andrew Y. Ng and Michael I. Jordan. (2001). Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3, 993-1022.
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *NIPS*.
- [5] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. Learning representations by back-propagating errors. *Nature*, 323, 533-536. 1986.
- [6] Sherstinsky A. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network // *Physica D: Nonlinear Phenomena*. 2020. Vol. 404, No. 132306.
- [7] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270– 280, jun 1989.
- [8] Felix Gers. Long Short-Term Memory in Recurrent Neural Networks. PhD thesis, *École Polytechnique Fédérale de Lausanne*, 2001
- [9] Bai S, Kolter J.Z, Koltun V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling // *arXiv:1803.01271v2*. 2018.
- [10] Junyoung Chung, Çağlar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.
- [11] Chastikova V.A., Sotnikov V.V. Method of analyzing computer traffic based on recurrent neural networks // *Journal of Physics: Conference Series. International Conference "High-Tech and Innovations in Research and Manufacturing" [HIRM]*. 2019. P. 012133.
- [12] Transfer Learning [Электронный ресурс] / режим доступа: [https://lena-voita.github.io/nlp\\_course/transfer\\_learning.html](https://lena-voita.github.io/nlp_course/transfer_learning.html)
- [13] Topal M. Onat, Anil Bas, Imke van Heerden. Exploring Transformers in Natural Language Generation: GPT, BERT, and XLNet // *arXiv:2102.08036v1*.
- [14] Gokul Yenduri, Ramalingam M, Chemmalar Selvi G, Supriya Y, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, Deepti Raj G, Rutvij H Jhaveri, Prabadevi B, Weizheng Wang, Athanasios V. Vasilakos, Thippa Reddy Gadekallu. Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. // *arXiv:2305.10435*. 2023.
- [15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // *arXiv:180.04805*. 2019.
- [16] Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context / Z. Dai, Z. Yang, Y. Yang [et al.] // *Proc. ACL*. P. 2978–2988.

**Дмитрий Харченко**, магистрант, Baitursynuly University, Қостанай, Қазақстан, dimhako@gmail.com

**Владимир Мадин**, докторант, Baitursynuly University, Қостанай, Қазақстан, vmadin@mail.ru

**Ольга Салыкова**, т.ғ.к., қауымдастырылған профессор, Baitursynuly University, Қостанай, Қазақстан, solga0603@mail.ru

**Ирина Иванова**, п.ғ.к., қауымдастырылған профессор, Baitursynuly University, Қостанай, Қазақстан, valera\_irina\_69@mail.ru

## ТАБИҒИ ТІЛДІ ӨНДЕУ ТАПСЫРМАЛАРЫНДА RNN ЖӘНЕ TRANSFORMER ТАЛДАУ

**Андатпа.** Мәтінді өңдеу және мәтінді құру тапсырмалары табиғи тілді өндеудегі негізгі міндеттер болып табылады. Бұл міндеттерді орындауға бағытталған көптеген әдістер, тәсілдер және модельдер бар. Мақалада табиғи тілді өңдеу саласында қолданылатын және мәтінді өңдеу және генерациялау функцияларын орындайтын RNN және Transformer сияқты қазіргі жасанды интеллект модельдерінің салыстырмалы талдауы берілген. Кейіннен, талдау табиғи тілді өндеудің тиімділігіндегі осы модельдердің күшті және әлсіз жақтарын анықтады.

**Түйінді сөздер:** жасанды интеллект, нейрондық желілер, қайталанатын нейрондық желілер, RNN, трансформер, Transformer.

**Dmitriy Harchenko**, graduate student, Baitursynuly University, Kazakhstan, Kostanay, dimhako@gmail.com

**Vladimir Madin**, doctoral student, Baitursynuly University, Kazakhstan, Kostanay, vmadin@mail.ru

**Olga Salykova**, candidate of technical sciences, associate professor, Baitursynuly University, Kazakhstan, Kostanay, solga0603@mail.ru

**Irina Ivanova**, candidate of pedagogical sciences, associate professor, Baitursynuly University, Kazakhstan, Kostanay, valera\_irina\_69@mail.ru

## ANALYSIS OF RNN AND TRANSFORMER MODELS IN NATURAL LANGUAGE PROCESSING TASKS

**Abstract.** Text processing and text generation tasks are key tasks in natural language processing. There are many different methods, approaches and models aimed at performing these tasks. The article provides a comparative analysis of current artificial intelligence models, such as RNN and Transformer, which are used in the field of natural language processing and perform text processing and generation functions. Subsequently, the analysis of which revealed the strengths and weaknesses of these models in the efficiency of natural language processing.

**Keywords:** artificial intelligence, neural networks, recurrent neural networks, RNN, transformer, Transformer.

Дата принятия: 13 сентября 2024 года

Дата рецензирования: 31 января 2025 года

Дата утверждения: 09 апреля 2025 года