Рекуррентные нейронные сети (RNN)

Рекуррентные нейронные сети (RNN)

RNN – это тип нейронных сетей, предназначенный для обработки последовательных данных.

- В отличие от обычных нейросетей, RNN имеют **память** о предыдущих состояниях.
- Используются в обработке текста, речи, временных рядов и прогнозировании.

Почему нужны RNN?

🔹 Обычные нейронные сети (MLP, CNN) не работают с последовательностями.

🔹 RNN учитывают предшествующие данные, что важно для анализа последовательностей.

🔹 Используются там, где важно **контекстное понимание**:

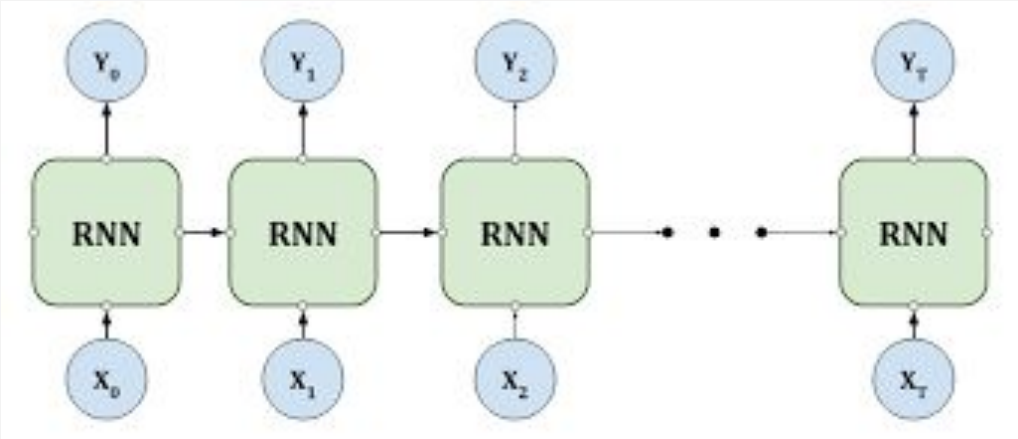
- **Обработка текста** 📖 (перевод, генерация, анализ настроений)
- **Распознавание речи** 🎙️ (автоматические субтитры, голосовые помощники)
- **Прогнозирование временных рядов** 📈 (финансы, погода, медицина)

Как работает RNN?

- RNN имеют **петлевую связь**: выход одного слоя передается обратно на вход следующего.
- В отличие от обычных нейронных сетей, у RNN есть **внутреннее состояние** (hidden state), которое обновляется на каждом шаге.
- Выход зависит не только от текущего входа, но и от **предыдущих состояний**.
- Основная идея – **обучение через время** (Backpropagation Through Time, BPTT).

Основные типы связей RNN

- **Многие ко многим** (перевод текста)
- **Одно ко многим** (генерация текста, музыки)
- **Многие к одному** (анализ текста, классификация эмоций)



Проблемы стандартных RNN

- **Ванишинг градиент (Vanishing Gradient)** – сложность обучения на длинных последовательностях.
- **Эксплодирующий градиент (Exploding Gradient)** – неустойчивость обучения.
- **Проблемы долгосрочной зависимости** – RNN плохо запоминают информацию на длинных последовательностях.
- **Ограниченная способность к обучению** – сложность обработки сложных структур данных.
- **Зависимость от порядка данных** – ошибки на ранних этапах обучения могут повлиять на весь процесс.

LSTM – решение проблем стандартных RNN

LSTM (Long Short-Term Memory) – рекуррентная нейронная сеть, решающая проблему vanishing gradient.

Основные компоненты:

- Входной, выходной и забывающий **гейты (gates)**.
- **Ячейка памяти (cell state)** – хранит долгосрочную информацию.

Преимущества: Удержание долгосрочных зависимостей. Контролируемое "забывание" ненужной информации. Более стабильное обучение.

Структура ячейки LSTM

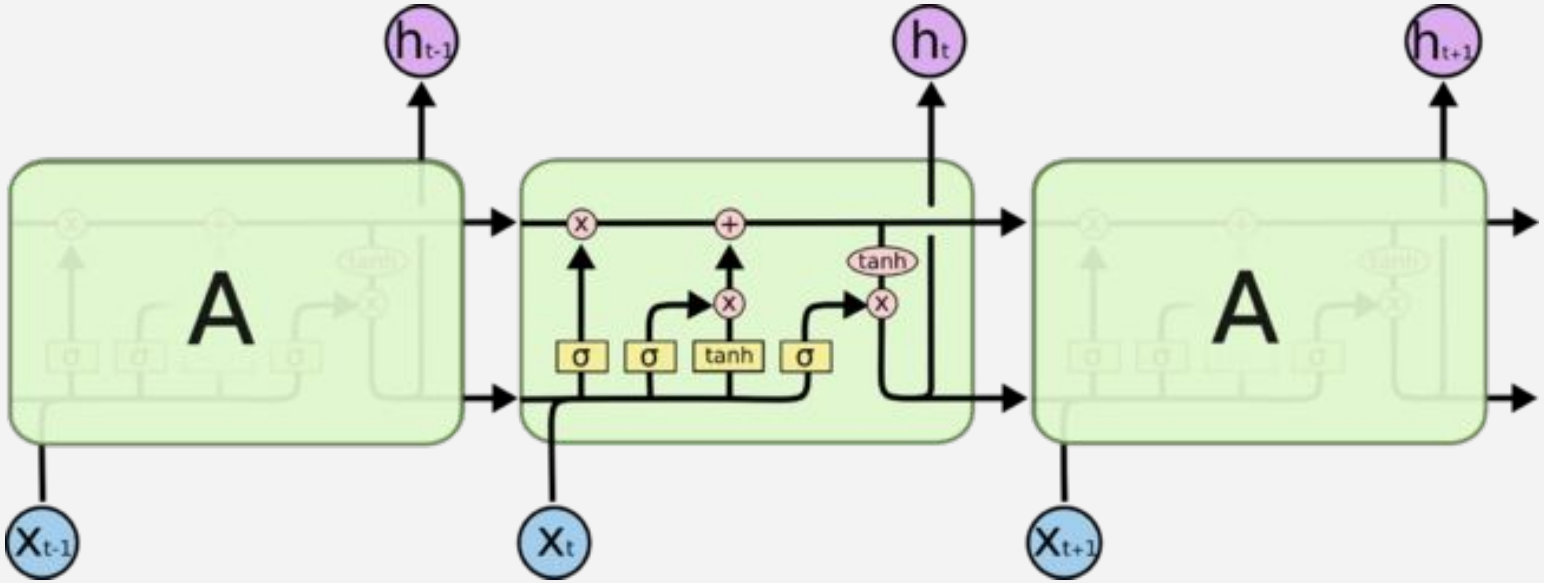
LSTM имеет три ключевых механизма, называемых **гейтами (gates)**:

1. **Забывающий гейт (Forget Gate)** – решает, какую часть информации удалить.
2. **Входной гейт (Input Gate)** – определяет, какую новую информацию добавить.
3. **Обновление состояния ячейки (Cell State)**
4. **Выходной гейт (Output Gate)** – управляет тем, что передается в следующий временной шаг.

Формулы гейтов

- $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ – забывающий гейт
- $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ – входной гейт
- $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$ – кандидат на обновление
- $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$ – обновление состояния
- $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$ – выходной гейт
- $h_t = o_t * \tanh(C_t)$ – новое скрытое состояние

Графическая схема работы LSTM



Пример работы LSTM на временных рядах

Временные ряды – это последовательность данных, зависящая от времени

Пример: прогнозирование цен на акции, температуры, спроса на продукцию.

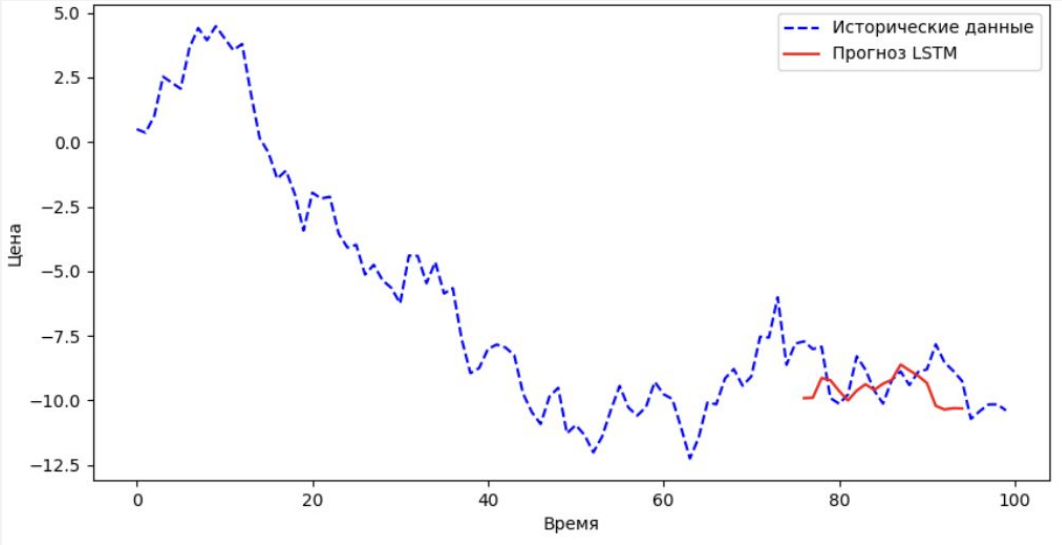
LSTM помогает учитывать долгосрочные зависимости в данных.

Пример: прогнозирование цен на акции

Входные данные: исторические цены на акции

LSTM анализирует изменения цен и выявляет тренды

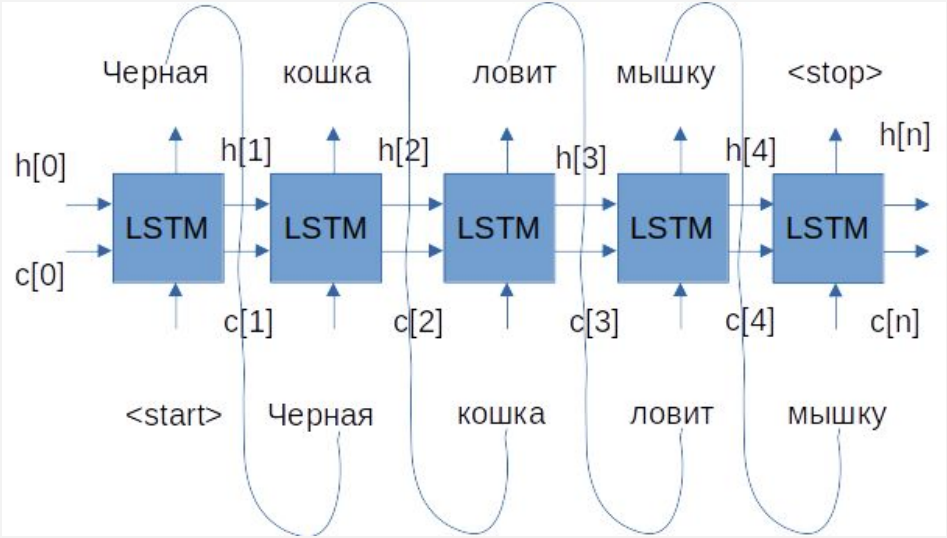
Выход: предсказание цены на следующий день



Применение LSTM в обработке текста (NLP)

LSTM широко используется в задачах обработки естественного языка (NLP), таких как:

- Машинный перевод
- Генерация текста
- Анализ тональности
- Распознавание речи



Как работает LSTM в NLP?

1. *Входные данные* – последовательность слов, представленных в виде векторов.
2. *LSTM* анализирует контекст слов, учитывая длинные зависимости.
3. *На выходе* – предсказанное слово, эмоция или переведенный текст.

Пример: Машинный перевод

Вход: "How are you?"

LSTM обрабатывает последовательность слов.

Выход: "Как дела?"

Пример в Python (LSTM)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(100, 1)),
    LSTM(50),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')
print(model.summary())
```

Gated Recurrent Unit (GRU) – Альтернатива LSTM

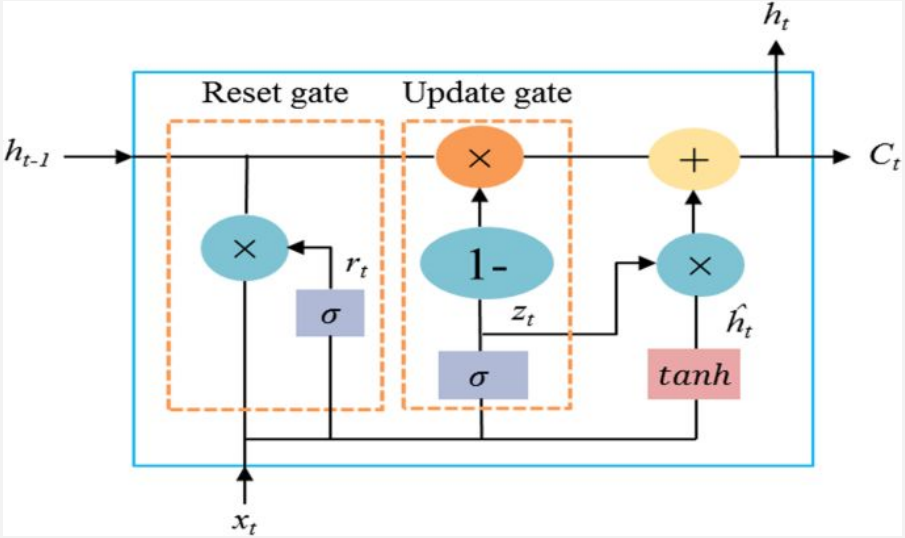
GRU – упрощенная версия LSTM

- Использует меньше параметров → быстрее обучается.
- Схожа по эффективности с LSTM, но требует меньше вычислений.
- Хорошо подходит для небольших датасетов.

Как работает?

В отличие от LSTM, в GRU всего **два** ворот:

- **Обновления** (Update Gate) – решает, какую информацию оставить.
- **Сброса** (Reset Gate) – определяет, что забыть.



Ключевые различия между GRU и LSTM

Характеристика	LSTM	GRU
Количество ворот	3 (входные, забывания, выходные)	2 (обновления, сброса)
Сложность	Высокая	Ниже, чем у LSTM
Скорость обучения	Медленнее	Быстрее
Эффективность	Хорошая на длинных последовательностях	Хорошая, но менее мощная на длинных последовательностях

Где используется GRU?

- Машинный перевод
- Голосовые помощники (Siri, Google Assistant)
- Генерация текста

Где GRU работает лучше?

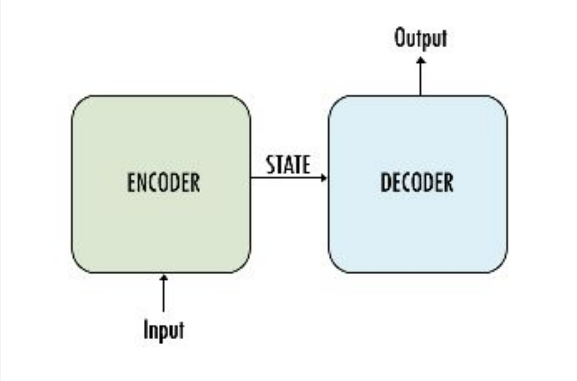
- В задачах, где важна **скорость обработки**. Например, голосовые ассистенты – чем быстрее сеть реагирует, тем лучше.
- В системах **перевода**, когда модели должны работать в реальном времени.

Характеристика	***RNN***	***LSTM***	***GRU***
Способность запоминать долгосрочные зависимости	Плохо	Хорошо	Хорошо
Количество ворот	1 (скрытое состояние)	3 (входные, забывания, выходные)	2 (обновления, сброса)
Вычислительная сложность	Низкая	Высокая	Средняя
Скорость обучения	Высокая	Медленная	Быстрая
Подверженность исчезающему градиенту	✅ Да	❌ Нет	❌ Нет
Применение	Простые временные ряды	Долгосрочные зависимости, сложные задачи	Быстрая обработка, NLP, голосовые ассистенты

Применение RNN в NLP

1. Перевод текста:

- **Пример:** Перевод английского текста на французский.
- Модель: Seq2Seq на основе RNN или LSTM.



Код для Seq2Seq модели

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense

# Входы
encoder_inputs = Input(shape=(None, 100))
encoder = LSTM(128, return_state=True)
encoder_outputs, state_h, state_c =
encoder(encoder_inputs)
encoder_states = [state_h, state_c]
```

Код для Seq2Seq модели

```
# Декодер
decoder_inputs = Input(shape=(None, 100))
decoder_lstm = LSTM(128, return_sequences=True,
return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
initial_state=encoder_states)
decoder_dense = Dense(100, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='categorical_crossentropy')
print(model.summary())
```

Применение RNN в NLP

2. Генерация текста:

- **Пример:** Генерация текстов на основе обученного корпуса данных.
- Применение: Автокомплит в поисковиках.

```
Step    0
  input: 24 ('T')
  expected output: 9 ('E')
Step    1
  input: 9 ('E')
  expected output: 22 ('R')
Step    2
  input: 22 ('R')
  expected output: 1 (' ')
Step    3
  input: 1 (' ')
  expected output: 26 ('V')
Step    4
  input: 26 ('V')
  expected output: 13 ('I')
```

Применение RNN в NLP

3. Анализ тональности текста:

- **Пример:** Определение, является ли отзыв положительным или отрицательным.
- Применение: Управление репутацией брендов.

Код для анализа тональности

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# Данные
texts = ["I love this product", "I hate this service"]
labels = [1, 0]  # 1: положительный, 0: отрицательный

# Преобразование текста
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
```

Код для анализа тональности

```
# Модель
model = LogisticRegression()
model.fit(X, labels)
print(model.predict(vectorizer.transform(["This
product is amazing"])))
```

Ограничения RNN

1. Ограничения в работе с длинными последовательностями: Даже с LSTM и GRU модель может испытывать сложности при обработке очень длинных данных.

2. Высокие вычислительные затраты: RNN требуют больше ресурсов для обучения по сравнению с традиционными алгоритмами.

3. Чувствительность к качеству данных: Ошибки в данных могут значительно ухудшить точность модели.