ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 17 за 2022 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Р.С. Ехлаков, В.А. Судаков

Прогнозирование стоимости котировок при помощи LSTM и GRU сетей

Статья доступна по лицензии
Creative Commons Attribution 4.0 International

Рекомендуемая форма библиографической ссылки: Ехлаков Р.С., Судаков В.А. Прогнозирование стоимости котировок при помощи LSTM и GRU сетей // Препринты ИПМ им. М.В.Келдыша. 2022. № 17. 13 с. **https://doi.org/10.20948/prepr-2022-17**
https://library.keldysh.ru/preprint.asp?id=2022-17

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

Р.С. Ехлаков, В.А. Судаков

Прогнозирование стоимости котировок при помощи LSTM и GRU сетей

Москва – 2022

Ехлаков Р.С., Судаков В.А.

Прогнозирование стоимости котировок при помощи LSTM и GRU сетей

В работе рассматриваются современные рекуррентные нейронные сети (RNN). Наибольшее внимание уделяется популярным и мощным архитектурам – длинная цепь элементов краткосрочной памяти (LSTM) и управляемые рекуррентные блоки (GRU). Написан программный комплекс для прогнозирования стоимости котировок, и проведено сравнение двух методов.

Ключевые слова: рекуррентная нейронная сеть, длинная цепь элементов краткосрочной памяти, управляемые рекуррентные блоки, прогнозирование стоимости котировок.

Roman Sergeevich Ekhlakov, Vladimir Anatolievich Sudakov

Forecasting the cost of quotes using LSTM & GRU networks

The paper considers modern recurrent neural networks (RNN). Most attention is paid to popular and powerful architectures – long chain of elements of short-term memory (LSTM) and controlled recurrent units (GRU). A software package for forecasting the cost of quotations has been written and a comparison of two methods has been made.

Key words: RNN, LSTM, GRU, forecasting the cost of quotes.

Оглавление

Введение

Рекуррентные нейронные сети (Recurrent Neural Networks, RNN) — вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки. Рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины, поэтому они часто используются в таких задачах, где нечто целостное разбито на части, например, в задачах обработки естественного языка, распознавания рукописного текста или распознавания речи [1].

Одна из главных проблем стандартных нейронных сетей (Neural Networks, NN) и сверточных сетей (Convolutional Neural Network, CNN) заключается в том, что они слишком ограничены: они принимают вектор фиксированного размера в качестве входных данных (например, изображение) и производят вектор фиксированного размера в качестве вывода (например, вероятности различных классов). Помимо этого, модели выполняют сопоставление, используя фиксированное количество вычислительных шагов (например, количество слоев в модели). RNN позволяют работать над последовательностями векторов на входе, выходе или в самом общем случае и тем и другим. Вот несколько примеров того, как могут выглядеть RNN:

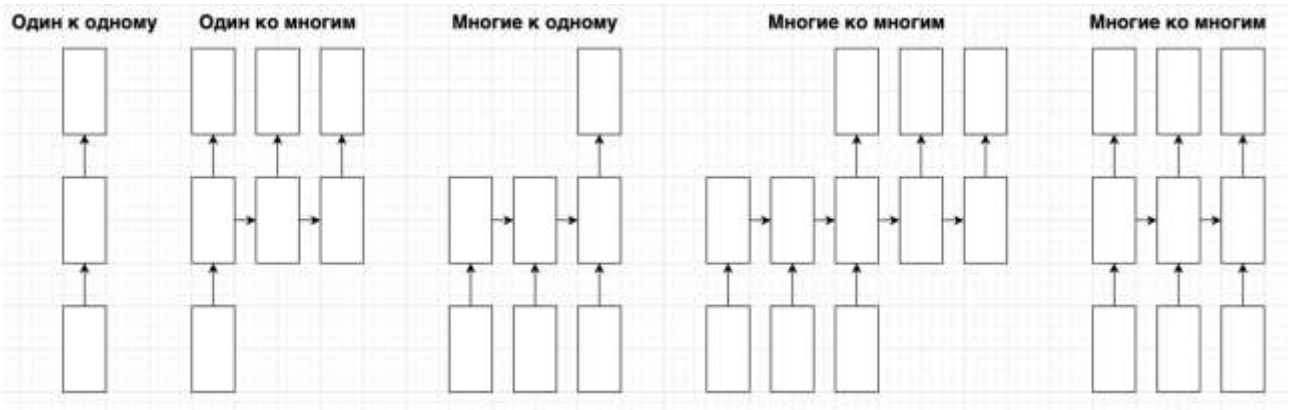

Рис. 1. Структура сетей RNN

Каждый прямоугольник является вектором, а стрелки представляют функции, например, умножение матрицы. Слева направо: (1) нейронная сеть – ввод и вывод фиксированного размера (например, классификация изображений); (2) выход последовательности (например, субтитры к изображению принимают изображение и выводят предложение слов); (3) последовательность ввода (например, анализ настроений, когда предложение классифицируется как выражение положительных или отрицательных настроений); (4) входные данные и последовательности (например, машинный перевод: RNN читает предложение на английском языке, а затем выводит

предложение на французском языке); (5) синхронизированные входные и выходные данные последовательности (например, классификация видео, где можно пометить каждый кадр видео) [2].

Как и следовало ожидать, рекуррентная нейронная сеть более гибкая по сравнению со стандартной нейронной сетью, которая с самого начала имеет фиксированное количество вычислительных шагов. Более того, RNN объединяет входной вектор со своим вектором состояния с фиксированной (но изученной) функцией для получения нового вектора состояния. Это можно интерпретировать в терминах программирования как запуск фиксированной программы с определенными параметрами на вход и внутренними переменными. Рассматриваемые таким образом, RNN по существу описывают программы, являются тьюринг-полными и могут моделировать произвольные программы с правильными параметрами.

Алгоритм RNN достаточно прост: на вход подается простой вектор X, а на выходе – вектор Y. На содержимое выходного вектора влияют не только входные параметры, но и все исторические данные входных параметров, которые использовались в прошлом. Описанный как класс алгоритм RNN состоит из одношаговой функции:

```
rnn = RNN()
y = rnn.step(x) # x - входной вектор, y - выходной вектор RNN
```

Класс RNN имеет некоторое внутреннее состояние, которое он обновляет каждый раз при вызове функции $step()$. В простейшем случае функция состояния состоит из скрытого вектора h. Реализация функции $step()$ в стандартном RNN алгоритме:

```
class RNN:
  # ...
  def step(self, x):
    # обновляем скрытое состояние
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # вычисляем выходной вектор
    y = np.dot(self.W_hy, self.h)
    return y
```

Параметрами RNN являются три матрицы: W_hh , W_xh, W_hy. Скрытое состояние $self.h$ инициализируется нулевым вектором. Функция $np.tanh$ реализует нелинейность, которая сжимает возможные значения до диапазона $[-1,1]$. Внутри функции $tanh$ существует два параметра: первый основан на предыдущем скрытом состоянии, второй – на текущем входном параметре.

Матрица RNN инициализируется случайными числами, и основная часть работы во время обучения уходит на поиск матриц, которые приводят к желаемому поведению, измеренному с помощью некоторой функции потерь, которая выражает предпочтения выходных типов данных y на входные последовательности x.

Такая способность обрабатывать последовательности делает RNN очень полезными в задачах машинного перевода (например, Google Translate): текстовая последовательность подается в качестве входных параметров и затем преобразуется в переведенный текст. Также RNN активно применяется для анализа настроения (например, определения положительного/отрицательного отзыва): текст подается в качестве входного параметра, а затем производится выходная классификация [3-5].

Виды RNN и ее параметры

Рассмотрим RNN со связью «многие ко многим» («many to many») со входными параметрами $x_0, x_1, \dots, x_n$, в результате которых получится $y_0, y_1, \dots, y_n$. Такие x_i и y_i являются векторами и могут иметь произвольные размеры. Алгоритм RNN итеративно обновляет скрытое состояние h и может иметь произвольную размерность. На любом шаге t:

1. Скрытое состояние h_t рассчитывается с использованием предыдущего скрытого состояния h_{t-1} и входного параметра x_t.
2. Выходной параметр y_t рассчитывается при помощи h_t.

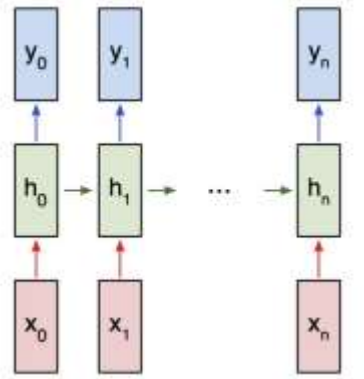

Рис. 2. РНС со связью «многие ко многим»

Алгоритм RNN использует 3 набора параметров для выполнения расчетов:

- W_{xh} используется для всех x_t и h_t;
- W_{hh} используется для всех h_{t-1} и h_t;
- W_{hy} используется для всех h_t и y_t.

Также используются два смещения:

- b_h добавляется при расчете h_t;
- b_y добавляется при расчете y_t.

Три набора параметров и два смещения представляют собой весь алгоритм RNN:

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h),$$
$$y_t = W_{hy}h_t + b_y. \quad (1)$$

Наборы параметров используются при расчете матрицы, а смещения добавляются к полученным результатам. Затем используется гиперболическая функция tanh как функция активации для первого уравнения, но также могут быть использованы другие функции (например, сигмоида) [6].

Длинная цепь элементов краткосрочной памяти (LSTM)

Наиболее мощной и популярной архитектурой RNN является длинная цепь элементов краткосрочной памяти (Long short-term memory; LSTM), предложенная в 1997 году З. Хохрайтером и Ю. Шмидхубером. Как и большинство рекуррентных нейронных сетей, LSTM-сеть является универсальной в том смысле, что при достаточном числе элементов сети она может выполнить любое вычисление, на которое способен обычный компьютер, для чего необходима соответствующая матрица весов, которая может рассматриваться как программа. В отличие от традиционных рекуррентных нейронных сетей, LSTM-сеть хорошо приспособлена к обучению на задачах классификации, обработки и прогнозирования временных рядов в случаях, когда важные события разделены временными лагами с неопределённой продолжительностью и границами. Относительная невосприимчивость к длительности временных разрывов даёт LSTM преимущество по отношению к альтернативным рекуррентным нейронным сетям, скрытым марковским моделям и другим методам обучения для последовательностей в различных сферах применения. Из множества достижений LSTM-сетей можно выделить наилучшие результаты в распознавании несегментированного слитного рукописного текста и победу в 2009 году на соревнованиях по распознаванию рукописного текста (ICDAR). LSTM-сети также используются в задачах распознавания речи, например LSTM-сеть была основным компонентом сети, которая в 2013 году достигла рекордного порога ошибки в 17,7% в задаче распознавания фонем на классическом корпусе естественной речи TIMIT [7]. Рисунок 3 представляет собой графическое представление устройства LSTM-сети.

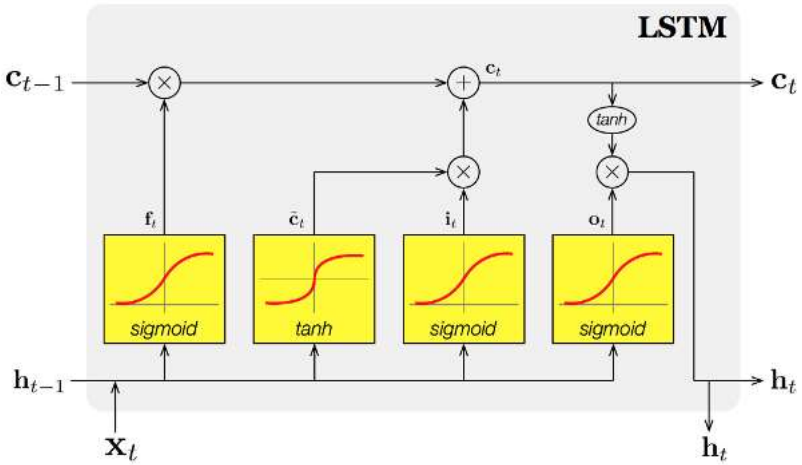

Рис. 3. Схематичное представление LSTM-сети

Традиционная LSTM с вентилями забывания $c_0 = 0$ и $h_0 = 0$ ($\circ$ обозначает произведение Адамара):

$$\begin{aligned} f_t &= \sigma_g\big(W_f x_t + U_f h_{t-1} + b_f\big), \\ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i), \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o), \\ c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c), \\ h_t &= o_t \circ \sigma_h(c_t), \end{aligned} \qquad (2)$$

где x_t – входной вектор; h_t – выходной вектор; c_t – вектор состояний; W, U и b – матрицы параметров и вектор; f_t, – вектор вентиля забывания, вес запоминания старой информации; i_t – вектор входного вентиля, вес получения новой информации; o_t – вектор выходного вентиля, кандидат на выход; σ_g – функция активации на основе сигмоиды; σ_c – функция активации на основе гиперболического тангенса; σ_h – функция активации на основе гиперболического тангенса.

Управляемые рекуррентные блоки (GRU)

Механизм вентилей для рекуррентных нейронных сетей был представлен в 2014 году. Было установлено, что его эффективность при решении задач моделирования музыкальных и речевых сигналов сопоставима с использованием долгой краткосрочной памяти (LSTM). По сравнению с LSTM у данного механизма меньше параметров, т.к. отсутствует выходной вентиль (рис. 4). Другими словами, блоку GRU не нужно использовать блок памяти для управления потоком информации, как блоку LSTM. Он может напрямую использовать все скрытые состояния без какого-либо контроля. GRU имеют меньше параметров и, следовательно, могут обучаться немного быстрее или нуждаться в меньшем количестве данных для обобщения. Но с большими

данными LSTM с более высокой выразительностью могут привести к лучшим результатам [8-9].

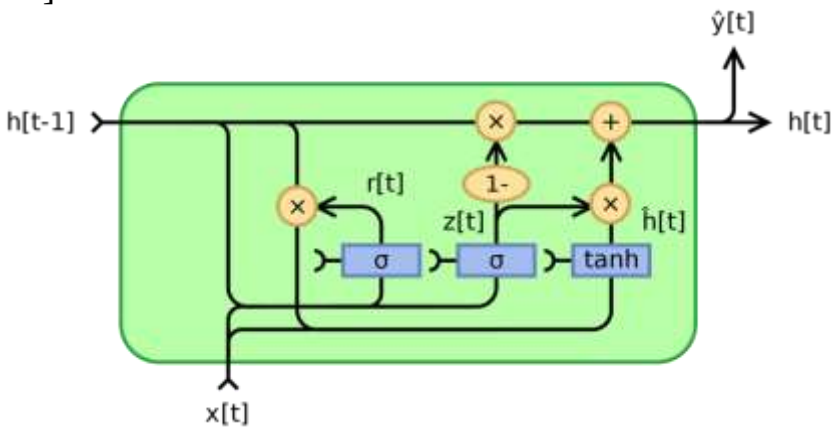

Рис. 4. Схематичное представление GRU-сети

Архитектура (∘ обозначает произведение Адамара), $h_0 = 0$:

$$\begin{aligned} z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z), \\ r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r), \\ h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ (W_h x_t + U_h(r_t \circ h_{t-1}) + b_h), \end{aligned} \tag{3}$$

где x_t – входной вектор; h_t – выходной вектор; z_t – вектор вентиля обновления; r_t – вектор вентиля сброса; W, U и b – матрицы параметров и вектор, σ_g – функция активации на основе сигмоиды; σ_h – функция активации на основе гиперболического тангенса.

Прогнозирование стоимости котировок при помощи LSTM и GRU сетей

Ячейка LSTM берет предыдущее состояние памяти c_{t-1} и выполняет поэлементное умножение с вентилем забывания f_t, чтобы определить, присутствует ли состояние памяти c_t. Если значение вентиля забывания равно 0, то предыдущее состояние памяти полностью забывается, иначе значение вентиля забывания равно 1, тогда предыдущее состояние памяти полностью передается в ячейку:

$$\begin{aligned} c_t &= c_{t-1} \cdot f_t \\ c_t &= c_t + (i_t \cdot c'_t) \\ H_t &= \tanh c_t \end{aligned} \tag{4}$$

Листинг LSTM кода:

```
# Архитектура LSTM
regressor = Sequential()
```

```
regressor.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape[1],1)))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
regressor.add(Dense(units=1))

# Компилирование RNN
regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
regressor.fit(X_train,y_train,epochs=50,batch_size=32)

# Подготовка тестового набора аналогично обучающему – для 60 целых
наборов есть 60 предыдущих значений, которые невозможно получить без
остальных, параметр – High
dataset_total = pd.concat((dataset["High"][:'2021'],dataset["High"]['2022':]),axis=0)
inputs = dataset_total[len(dataset_total)-len(test_set) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)

# Подготовка и прогнозирование
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)

# Визуализация результатов
plot_predictions(test_set,predicted_stock_price)

# Оценка модели
return_rmse(test_set,predicted_stock_price)
```

Полученные результаты при помощи LSTM метода отображены на рисунке 5.

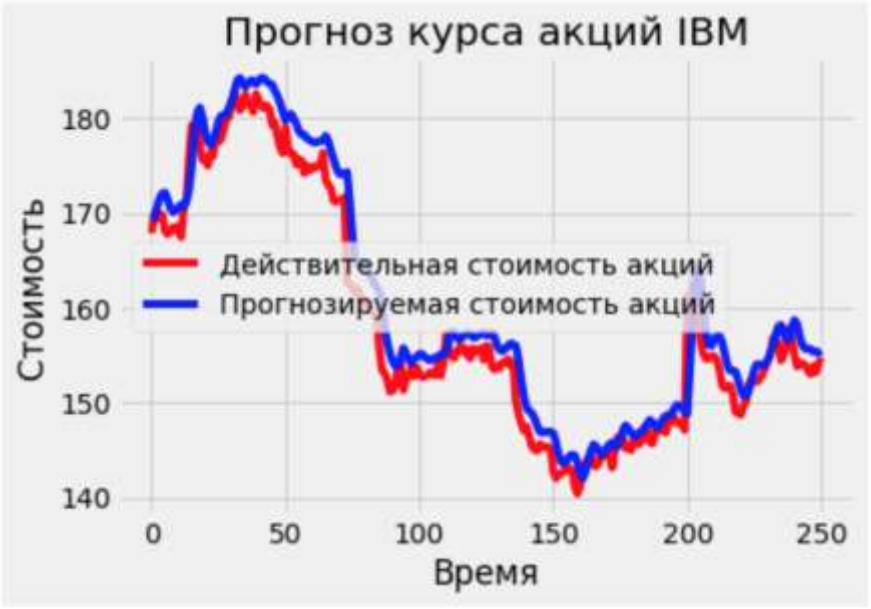

Рис. 5. Результаты расчета LSTM-сети

GRU почти аналогичен LSTM, за исключением того, что у GRU есть два шлюза: шлюз сброса и шлюз обновления. Шлюз сброса определяет, как объединить новый ввод с предыдущей памятью, а шлюз обновления определяет, какую часть предыдущего состояния следует сохранить.

Листинг GRU кода:

```
# Архитектура GRU
regressorGRU = Sequential()

# Первый слой GRU
regressorGRU.add(GRU(units=50, return_sequences=True,
input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))

# Второй слой GRU
regressorGRU.add(GRU(units=50, return_sequences=True,
input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))

# Третий слой GRU
regressorGRU.add(GRU(units=50, return_sequences=True,
input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))

# Четвертый слой GRU
```

```
regressorGRU.add(GRU(units=50, activation='tanh'))
regressorGRU.add(Dropout(0.2))

# Выходной слой
regressorGRU.add(Dense(units=1))

# Создание RNN сети
regressorGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum=0.9,
nesterov=False),loss='mean_squared_error')

# Подготовка X_test и прогнозирование цен
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
GRU_predicted_stock_price = regressorGRU.predict(X_test)
GRU_predicted_stock_price = sc.inverse_transform(GRU_predicted_stock_price)

# Визуализация результатов
plot_predictions(test_set,GRU_predicted_stock_price)

# Оценка модели
return_rmse(test_set,GRU_predicted_stock_price)
```

Полученные результаты при помощи GRU метода отображены на рисунке 6.

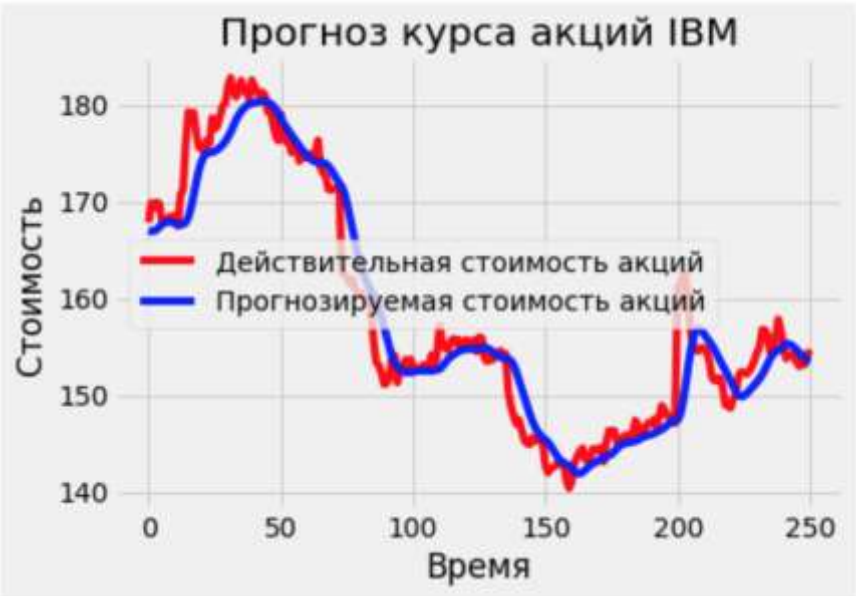

Рис. 6. Результаты расчета GRU-сети

Заключение

В работе была проанализирована разница между RNN модулями: LSTM и GRU. GRU использует меньше параметров обучения и, следовательно, использует меньше памяти и выполняется быстрее, чем LSTM, тогда как модуль LSTM более точен для большего набора данных. LSTM стоит использовать, если мы имеем дело с большими последовательностями и важна точность, GRU используется, когда необходимо получить более быстрые результаты с меньшим потреблением памяти.

По результатам расчетов среднеквадратичная ошибка LSTM составляет не более 4-5%, а среднеквадратичная ошибка GRU – не более 9% по сравнению с абсолютными значениями. Таким образом, можно говорить о высокоточных результатах, а также о возможности применения в расчетах реального прогнозирования.

Библиографический список

1. Qi-Qiao He, Cuiyu Wu, Yain-Whar Si. LSTM with particle Swam optimization for sales forecasting. Electronic Commerce Research and Applications, Volume 51, Jan-Feb 2022. https://doi.org/10.1016/j.elerap.2022.101118.

2. Tasarruf Bashir, Chen Haoyong, Muhammad Faizan Tahir, Zhu Liqiang. Short term electricity load forecasting using hybrid prophet-LSTM model optimized by BPNN. Energy Reports, Volume 8, November 2022, Pages 1678-1686, Energy Reports. https://doi.org/10.1016/j.egyr.2021.12.067.

3. Jiaqi Qin, Yi Zhang, Shixiong Fan, Xiaonan Hu, Yongqiang Huang, Zexin Lu, Yan Liu. Multi-task short-term reactive and active load forecasting method based on attention-LSTM model. International Journal of Electrical Power & Energy Systems, Volume 135, February 2022. https://doi.org/10.1016/j.ijepes.2021.107517.

4. Ezat Ahmadzadeh, Hyunil Kim, Ongee Jeong, Namki Kim, Inkyu Moon. A Deep Bidirectional LSTM-GRU Network Model for Automated Ciphertext Classification. IEEE Access, Volume: 10, P. 3228-3237.

5. Shiva Nosouhian, Fereshteh Nosouhian, Abbas Kazemi Khoshouei. A Review of Recurrent Neural Network Architecture for Sequence Learning: Comparison between LSTM and GRU. Preprints 2021, 2021070252 (doi: 10.20944/preprints202107.0252.v1).

6. Rong Wang, Enmin Zhou. Stock Prediction Based on Optimized LSTM and GRU Models. Hindawi, Scientific Programming, Volume 2021. https://doi.org/10.1155/2021/4055281.

7. Yassine Touzani, Khadija Douzi. An LSTM and GRU based trading strategy adapted to the Moroccan market. Journal of Big Data (2021) 8:126 https://doi.org/10.1186/s40537-021-00512-z.

8. Jongyeop Kim, Seongsoo Kim, Hayden Wimmer, Hong Liu. A Cryptocurrency Prediction Model Using LSTM and GRU Algorithms. IEEE/ACIS 6th International Conference on Big Data, Cloud Computing, and Data Science (BCD). 2021. DOI: 10.1109/BCD51206.2021.9581397.

9. Shwetha Salimath, Triparna Chatterjee, Titty Mathai, Pooja Kamble & Megha Kolhekar. Prediction of Stock Price for Indian Stock Market: A Comparative Study Using LSTM and GRU. International Conference on Advances in Computing and Data Sciences. ICACDS 2021: Advances in Computing and Data Sciences. P. 292-302.