

Interpreting the market whisper: Applying RNN encoder-decoders to multistep forecast the price of pound sterling in Chinese yuan

Student name: Yishan Liu

Student number: 200267117

Supervisor name: Dr Fredrik Dahlqvist

Programme of study: MSc Data Science and Artificial Intelligence

I. Abstract

This project has 3 aims:

1. Investigate the possibility of long term (30 days) price prediction using deep learning methods. This is exceptionally ambitious as it is already challenging to produce an accurate 5-day forecast in the field.
2. Compare the forecasting performance of 5 RNN encoder-decoders and their non encoder-decoder version.
3. Investigate the limitations of evaluation metrics and use visualisation to solve it.

3 results are concluded:

1. The best performing model discovered is the bidirectional GRU encoder-decoder.
2. All models failed to accurately forecast future prices.
3. However, many models produce directionality calls which indicates the possibility of accurate long term forecasting.

II. Introduction

Economists never agree with each other (Gregory Mankiw, 2020). In economics, there has been an “invisible hand” identified by Adam Smith (1776), guiding the economy for centuries. Fama (1970) argued that the hand i.e. the market, is so efficient that it is impossible to predict the future market price. Shiller (1981) disagreed with Fama, and Hansen (1982) disagreed with Shiller. All three were awarded the Nobel economics prize (The Nobel Prize, 2013), because at the time no one knew who was right due to limited techniques for analysing the market.

When economists raise problems and questions, engineers and mathematicians create tools and methods to solve and answer them. Sometimes their tool is so advanced that it can “answer questions that we don’t even know what the questions are yet”, said Bill Nelson (2022) – administrator of the National Aeronautics and Space Administration (NASA) – when he introduced the Webb telescope.

To tackle the disagreement among the three Nobel prize winners, and to understand or even visualise the “invisible hand”, tools similar to the Webb telescope that can answer unknown questions would be useful. One of the dimensions for such tools to slice into this problem could be the price charts, because in the economy, prices allocate resources (Gregory Mankiw, 2020). Since many price charts appear similar to acoustic wave graphs, one might metaphorise the prices as the market whisper. Coincidentally, deep neural networks (DNN) can recognise speech (Dahl et al., 2012; Deng, Hinton and Kingsbury, 2013; Hinton et al., 2012), translate languages (Collobert and Weston, 2008; Liu et al., 2013; Schwenk, 2012), and even answer questions (Abbasiataeb and Momtazi, 2021; Iyyer et al., 2014; Sharma and Gupta, 2018).

In recent years, the recurrent neural network (RNN) encoder-decoder was also used in these tasks: question answering (Allaouzi, Ahmed and Benamrou, 2019; Nie et al., 2017; Yin et al., 2016), machine translation (Bahdanau, Cho and Bengio, 2014; Chen et al., 2017; Kumar and Sarawagi, 2019; Makin, Moses and Chang, 2020; Nakayama and Nishida, 2017), and speech recognition (Bahdanau et al., 2016; Chiu et al., 2018; Lu et al., 2015; Lu, Zhang and Renais, 2016; Toshniwal et al., 2018). The RNN encoder-decoder accomplished state-of-the-art translation results during development when it was invented by Cho et al. (2014b), and Sutskever, Vinyals and Le (2014) independently (Goodfellow, Bengio and Courville, 2016). Thus, the RNN encoder-decoder might be the Webb telescope for visualising the “invisible hand”, interpreting the market whisper, and answering the unknown questions about the market economy.

To put this idea into practice, this paper takes a small step towards interpreting the market whisper by attempting to multistep forecast the yuan priced sterling using RNN encoder-decoders. Compared to the longest forecasting window of 4 days in the literature (Zhang et al., 2022), this paper aims to push the boundaries of long term forecasting by extending the window to 30 days. Longer forecasting windows can help investors to formulate their views on the future course of the market and create better trading strategies. Hopefully, the discoveries from this paper can support further research into the use of deep learning to study the market.

Context: The under-recognition of deep learning in the financial industry

Despite the research efforts major financial institutions e.g. JPMorgan Chase & Co (2019) and academic organisations e.g. University of Oxford (2022) put in place, deep learning as a tool in quantitative finance is under-recognised in many financial organisations. For example, there is almost no discussion about deep learning in quantitative finance in the Financial Times which is one of the most important newspapers for the industry (Financial Times, 2022). Another example is that in Bloomberg terminals deep learning is not recognised as one of the investment strategies when it comes to fund classification e.g. the classification of exchange traded funds (ETF) or hedge funds (Bloomberg L.P., 2013). This makes it hard to measure the returns generated by deep learning in the industry. Therefore, another method is created by this paper to measure the application of deep learning in the industry. However, the result is the same, as not a single fund mention “deep learning” in its description. TABLE. 4 in the appendix demonstrates the detail of this matter (Bloomberg L.P., 2022a).

One reason for this issue is that the financial industry prefers methods that can be explained with human

generated mathematical concepts. Thus, classical methods e.g. stochastic calculus, are more trusted by the industry. In comparison, deep learning methods generate concepts that cannot be quickly interpreted by experts, nor can they be understood by the management or decision makers in the industry. Therefore, it might not replace classical models and be widely implemented in practice for profit generation.

One of the innovations of this paper could mitigate this trust issue from another angle. Since humans can intake information faster by looking at pictures than reading texts (Keim, 2002), visualising the comparison of model prediction and the actual price data could demonstrate the reliability of deep learning methods. This idea is inspired by rocket engineer Wernher von Braun (1939) who advocated space exploration by making a television series after the war (Man in Space, 1955). In addition, results visualisation is rarely discussed in the literature.

III. Literature review

1. Introduction

When Google engineer Blake Lemoine “observed sentience” in an artificial intelligence (AI) application, one naturally remembers the words of the Victorian mathematician Ada Lovelace who worked on the first mechanical computer, the Analytical Engine. Her comments on this steam-powered computer are as follows: “The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform.... Its province is to assist us in making available what we’re already acquainted with” (Chollet, 2018; Helmore, 2022; Hollings, Martin and Rice, 2018). One might wonder: if one day AIs truly became sentient, how would they interpret and react to this remark by the creator of their ancestor (Scott, 2012)?

Hartree (1947) points out that the statement made by Lovelace does not imply that it is impossible to construct machines that have the ability to originate. Another pioneer in AI, Alan Turing, agrees with Hartree, but goes one step further and concludes that general-purpose computers could certainly be capable of learning and originality (Turing, 1950). Thus, machine learning arose. It arose from the idea that machines could go beyond the instructions programmed by humans and learn on their own. This idea revolutionised classical programming: instead of inputting instructions and data for the computers to generate answers, rules can be created by computers from inputting data and the answers. This is illustrated in Fig. 1:

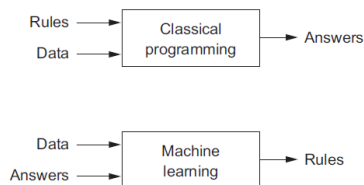


Fig. 1. Classical programming paradigm and machine learning (Chollet, 2018).

2. Deep learning and time series analysis

The challenge for analysing time series data is related with the temporal dependencies contained within the data (Gamboa, 2017). Classical methods for time series analysis can be found in econometrics, a statistical subject for testing economic theories (Wooldridge, 2018). One commonly used

model is the autoregressive integrated moving average (ARIMA) model (Verbeek, 2017). Classical models similar to ARIMA usually rely on human generated features that require expertise in the relevant field. Deep learning on the other hand can create new features on their own and can be applied to forecasting (Gamboa, 2017).

The following sections introduce a few deep learning models, such as deep feedforward networks, recurrent neural networks (RNN), gated recurrent units (GRU), long short-term memory (LSTM), and RNN encoder-decoder.

3. Deep feedforward networks

The ordinary deep learning model is the deep feedforward networks, also known as multilayer perceptrons (MLP). The goal of various MLPs are to approximate arbitrary functions. To achieve this goal, the hidden neurons of the MLPs perform certain computations on the inputs, such as basic logic operations. Given sufficient neurons or nodes, many MLPs can capture sophisticated interactions among the inputs, even with a single hidden layer. Because of this, the MLPs are also known as the universal approximators. The parameters of the neural networks are learnt from data. Subsequently, this learning process generates the best function approximation for the arbitrary functions (Goodfellow, Bengio and Courville, 2016).

Therefore, deep learning is fundamentally a process of optimisation. This is because the search for parameters that best fit the input data is one of the major processes of deep learning (Zhang et al., 2021).

4. Recurrent neural networks

In deep feedforward networks, information flows forward and there are no feedback connections. This is the reason these models are named as “feedforward”. When the output of the model is able to be fed back into the model, the model becomes a recurrent neural network (RNN). This is achieved with recurrent connection as demonstrated in Fig. 2:

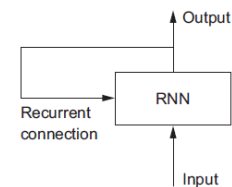


Fig. 2. A recurrent neural network (Chollet, 2018).

This added extension of feedback connection enables explicit handling of order between data points in the input during optimisation. MLPs and convolutional neural networks (CNN) do not have this unique characteristic (Brownlee, 2018). In theory, a RNN is able to connect each output to the complete “history” of preceding inputs. If a MLP is a universal approximator, a RNN is capable of approximating almost all sequence-to-sequence mapping with sufficient hidden units (Graves, 2012).

For many RNNs, the “history” of the previous inputs is stored in “hidden states”. A RNN is a neural network with “hidden states”. The application of “hidden states” is one of the main reasons for RNNs to achieve sequence-based specialisation (Goodfellow, Bengio and Courville, 2016). The “hidden states” are equivalent to the memory of RNNs (Zhang et al., 2021).

5. Gated recurrent units and long short-term memory

Vanishing gradients always happens when training many RNNs (Graves, 2012). The calculations of gradients in the hidden layers of many RNNs involves long products of matrices with values between 0 and 1. As a result, the weights of RNN hidden layers can become unstable as the elements of the input sequences iterate through the recurrent connections (Chollet, 2018; Graves, 2012). Fig. 3 illustrates: the darker the shades of the nodes, the larger the values of the weights in the RNN layers. Vanishing gradient happens when new inputs overwrite the weights in the hidden layers causing the RNN to “forget” the inputs from earlier steps.

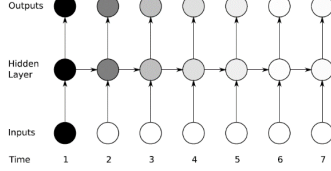


Fig. 3. The vanishing gradient problem for RNNs (Graves, 2012).

Gated Recurrent Unit (GRU) (Cho et al., 2014a) and Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) are two of the numerous attempts to address vanishing gradients (Graves, 2012; Zhang et al., 2021). LSTM is one of the earliest proposed methods. The LSTM architecture is more complicated than GRU, therefore, harder to compute. Due to the simpler nature of GRU, it will be discussed before LSTM.

1) GRU

The added gates for the hidden states in many GRUs differentiates them from ordinary RNNs. These gates are essential for creating the gated hidden states.

a. Gated hidden states

The gated hidden states controls when to update or reset hidden states. The dedicated mechanism for creating the gated hidden states can be learned. Therefore, it fulfils the following requirements for better forecasting performance listed below (Zhang et al., 2021):

- Store or “memorise” highly significant early observations for the forecast.
- Skip or “forget” irrelevant observations.
- Reset hidden states to reflect logical break downs of the sequence.

b. Reset gate and update gate

The reset gate and the update gate as in Fig. 4 are two fundamental mechanisms for computing the gated hidden states.

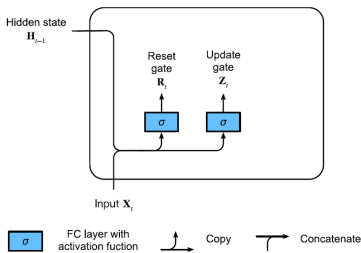


Fig. 4. Computation of the reset gate and the update gate in a GRU (Zhang et al., 2021).

Both gates are defined as functions of input and the hidden state from the previous timestep:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r)$$

$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z)$$

where for a given timestep t , $\mathbf{R}_t, \mathbf{Z}_t \in \mathbb{R}^{n \times d}$ (n is the number of input samples and d is the number of input features) are the reset gate and update gate; $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ and $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$ (h is the number of hidden units) are the input and the hidden states from the previous timestep; $\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h \times h}$ are weight parameters; $\mathbf{b}_r, \mathbf{b}_z \in \mathbb{R}^{1 \times h}$ are the biases; finally σ is a non-linear activation function named sigmoid (Maass, 1997; Zhang et al., 2021).

The sigmoid function squashes input values into an interval of (0, 1). Thus, when the entries of the reset gate \mathbf{R}_t is close to 0, the pre-existing hidden states are “forgotten” or reset to defaults; conversely, when the entries of the reset gate \mathbf{R}_t is close to 1, all previous hidden states are “memorised” and a conventional RNN is recovered.

This leads to the computation for the candidate hidden state as in (1):

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h) \quad (1)$$

where $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ are the weight parameters; $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ is the bias; \odot is the Hadamard (elementwise) multiplication operator. Fig. 5 depicts this:

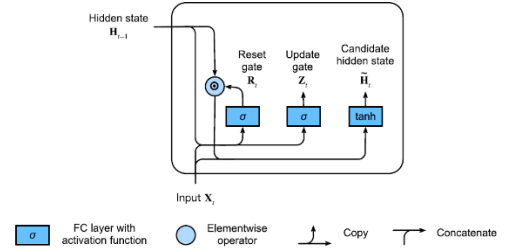


Fig. 5. Computation of candidate hidden state in a GRU (Zhang et al., 2021).

Thus, whenever the reset gate \mathbf{R}_t is close to 0, the candidate hidden state $\tilde{\mathbf{H}}_t$ becomes an output of a MLP with input \mathbf{X}_t .

The reset gate controls how much the candidate hidden states remembers the previous hidden states. The update gate works similarly to the reset gate. However, the update gate manipulates how similar the new hidden state is to the hidden state from the last timestep. (2) and Fig. 6 show the final update equation and the complete computational flow inside the hidden block of a GRU when both reset and update gates are in action:

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t \quad (2)$$

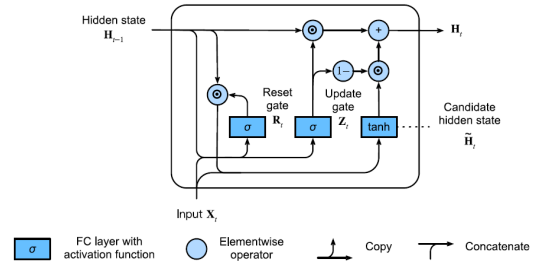


Fig. 6. Computational flow inside the hidden block of a GRU (Zhang et al., 2021).

Similar to the reset gate \mathbf{R}_t , when the update gate \mathbf{Z}_t is close to 0, the information from the input \mathbf{X}_t is sufficiently utilised to update the new hidden state \mathbf{H}_t . As a result, the new hidden state \mathbf{H}_t can fully differentiate itself from the hidden state in the last timestep \mathbf{H}_{t-1} ; in contrast, when the update gate \mathbf{Z}_t is close to 1, the information from \mathbf{X}_t is neglected,

making the new hidden state \mathbf{H}_t approach to a copy of the previous hidden state \mathbf{H}_{t-1} (Zhang et al., 2021).

2) LSTM

It is intriguing that LSTM has a more sophisticated architecture but was developed decades earlier than GRU. LSTM was created by Hochreiter and Schmidhuber in the 1990s. It was the outcome of their research on vanishing gradients. LSTMs differentiate them from GRUs with the use of two gates instead of one to control the similarity of the new hidden state to the previous hidden state. Therefore, one of the main differences between the two models is the number of gates used in their mechanism governing input and forgetting (Zhang et al., 2021).

a. Candidate memory cell

To appreciate this difference, one needs to first understand the computation of candidates memory cell in LSTM. The candidate memory cell in LSTM is similar to the candidate hidden state in GRU. This becomes clear when the equations for computing the two concepts are presented together in (3) and (1):

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c) \quad (3)$$

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h) \quad (1)$$

where $\tilde{\mathbf{C}}_t \in \mathbb{R}^{n \times h}$ is the candidate memory cell in LSTM, $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ is the candidate hidden state in GRU, and $\mathbf{R}_t \in \mathbb{R}^{n \times x}$ is the reset gate in GRU.

The only difference in the computation of $\tilde{\mathbf{C}}_t$ and $\tilde{\mathbf{H}}_t$ is that $\tilde{\mathbf{H}}_t$ uses the reset gate \mathbf{R}_t to alter the previous hidden state \mathbf{H}_{t-1} before linear transformation. Therefore, the calculation of the candidate hidden state $\tilde{\mathbf{H}}_t$ in GRU is more complicated than that of the candidate memory cell $\tilde{\mathbf{C}}_t$ in LSTM.

b. Memory cell

The computing of memory cell in LSTM and that of the hidden state in GRU is also similar. They both are the sums of two elementwise multiplications. Thus, enables them to regulate the updating and forgetting (Goodfellow, Bengio and Courville, 2016; Zhang et al., 2021). However, there is one key aspect that distinguishes them: instead of using a single gate to generate memory cell in GRU, LSTM uses two dedicated gates for this. (4) and (2) compares the two computations closely:

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t \quad (4)$$

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t \quad (2)$$

where $\mathbf{C}_t, \mathbf{H}_t \in \mathbb{R}^{n \times h}$ are the memory cell in LSTM and the hidden state in GRU, $\mathbf{F}_t, \mathbf{I}_t \in \mathbb{R}^{n \times h}$ are the forget gate and the input gate in LSTM, $\mathbf{Z}_t \in \mathbb{R}^{n \times h}$ is the update gate in GRU.

c. Hidden state

Finally, the hidden state of LSTM is calculated as:

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t)$$

where $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ is the hidden state of LSTM, $\mathbf{O}_t \in \mathbb{R}^{n \times h}$ is the output gate which is unique to LSTM compared to GRU. The output gate \mathbf{O}_t ensures the values in the hidden state \mathbf{H}_t are in the interval (0, 1). This is because the two gates used to transform memory cell \mathbf{C}_t do not always add up to one elementwise. Therefore, one might conclude that the GRU is better designed to save computational expensiveness. However, there is always a trade off between computational cost and representational power.

The output gate \mathbf{O}_t , the forget gate \mathbf{F}_t , and the input gate \mathbf{I}_t of the LSTM are all computed as follows:

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i)$$

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f)$$

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o)$$

Fig. 7 illustrates the data flow of calculations in an LSTM graphically:

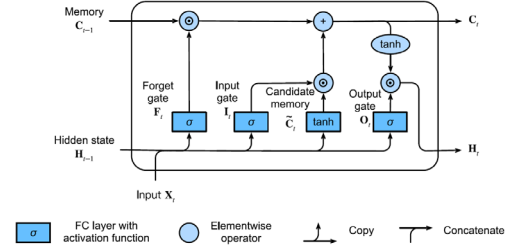


Fig. 7. Computing the hidden state in an LSTM model (Zhang et al., 2021).

Some might question if all the gates in LSTM are necessary. It turns out that the most crucial part in an LSTM is the forget gate (Greff et al. 2015). Moreover, the key to boost the performance of LSTM is to simply add a bias of 1 to the forget gate (Jozefowicz et al. 2015). This finding was proposed by Gers et al. (2000) a few years after LSTM was published. Additionally, the study by Marchi et al., (2015) finds that LSTMs perform better at non-linear prediction.

Finally, extensions of GRU or LSTM include CNN-LSTM (Donahue et al., 2014) or CNN-GRU (Nguyen, Cai and Chu, 2019), in which the RNN interpret outputs from the CNN, while a further extension convLSTM (Shi et al., 2015) uses convolution (LeCun, 1989) to read inputs.

6. RNN encoder-decoder

When it comes to mapping an input sequence and an output sequence of different length, a new architecture is required: RNN encoder-decoder.

As suggested by its name, the model has two major components. The encoder transforms the input variable length sequence to a fixed shape state. The decoder then maps the fixed shape state to an output sequence with variable length. This is illustrated in Fig. 8:

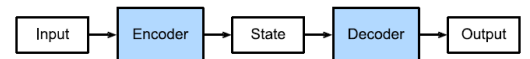


Fig. 8. The encoder-decoder architecture (Zhang et al., 2021).

When the encoder and decoder are designed with RNNs, the RNN encoder encodes the information from the input sequence in its hidden state. To construct the output sequence, the RNN decoder computes the value of the next timestep based on the data have been seen or generated, along with the encoded representation of the input sequence.

1) The RNN encoder

The RNN encoder encodes information from the input sequence in the context variable \mathbf{c} . Assume the input sequence is $\mathbf{x}_1, \dots, \mathbf{x}_t$, where \mathbf{x}_t is the feature vector at timestep t in the input sequence. At timestep t , the RNN transforms \mathbf{x}_t and the previous hidden state \mathbf{h}_{t-1} into the current hidden state \mathbf{h}_t . A function f can be used to express the transformation inside the recurrent layer of the RNN: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$. Usually, the encoder computes the context variable \mathbf{c} by transforming the hidden states at all timesteps through a function q : $\mathbf{c} = q(\mathbf{h}_1, \dots, \mathbf{h}_t)$.

Bidirectional RNNs (Schuster and Paliwal, 1997) can be used to construct the encoder. In this case, the hidden state will not only depend on the input subsequence at, before, but also after the current timestep. Thus, the hidden state will encode the information of the entire sequence (Zhang et al., 2021). Bidirectional RNNs generally perform better than unidirectional RNNs (Graves and Schmidhuber, 2005). In addition, CNN and convLSTM can also be used as encoders.

2) The RNN decoder

Suppose the label sequence from the training data is y_1, \dots, y_T . Timestep t' can be differ from timestep t of the input sequence, because the encoder-decoders allow the input and output to have variable length. The decoder models the conditional probability of the decoder output $y_{t'}$ on the context variable c and the previous output subsequence $y_1, \dots, y_{t'-1}$. This conditional probability on sequences can be represented as $P(y_{t'} | y_1, \dots, y_{t'-1}, c)$.

To model such conditional probability, at current timestep t' , the RNN decoder transforms c , $y_{t'-1}$, and the previous hidden state $s_{t'-1}$ into current hidden state $s_{t'}$. Subsequently, a function g can represent the transformation in the hidden layer of the RNN decoder: $s_{t'} = g(c, y_{t'-1}, s_{t'-1})$. In the end, an output layer is needed to generate the final output.

A limitation of the encoder-decoder was observed by Bahdanau, Cho and Bengio (2014) in their machine translation study. The context variable c fails to summary a long sequence perfectly. Thus they suggest to have a variable length c . They also introduced the attention mechanism. It can learn to link elements in the output sequence to elements in c .

7. Conclusion

According to past studies, it is hard to distinguish a better model between GRU and LSTM. For instance, some observed that the GRU outperforms the LSTM (Gallicchio, Micheli and Pedrelli, 2019; Yamak, Yujian and Gadosey, 2019), some find the opposite (Althelaya, El-Alfy and Mohammed, 2018; Rajagukguk, Ramadhan and Lee, 2020), some claim that both models perform equally worse than ARIMA or models that are similar to ARIMA (Yamak, Yujian and Gadosey, 2019; Liu, Lin and Feng, 2021). An interesting experiment conduct by Flores, Tito and Centty (2020) suggest a hybrid GRU-LSTM outperforms the stacked non-hybrid versions.

There are more contradicting results in the past analyses when considering all models mentioned above. For example, Mehtab and Sen (2022) find the LSTM outperforms the LSTM encoder-decoder. Mohtasham Khani, Vahidnia and Abbasi (2021) observe that the LSTM not only outperforms the LSTM encoder-decoder but also the bidirectional LSTM, and the CNN-LSTM. In contrast, Iqbal et al. (2021) notice that convLSTM encoder-decoder outperforms LSTM and convLSTM; moreover, Zhang et al. (2021) discover that LSTM encoder-decoder outperforms GRU encoder-decoder, GRU, and LSTM.

Therefore, model performances may vary depending on the specific forecasting tasks it undertakes. It is necessary to conduct multiple experiments for selecting the best model. Thus, this paper attempts to compare 10 different models.

IV. Methodology

Before explaining the methodology, some hardware and software specifications will be introduced. The project is implemented entirely in Python 3.7.13 (van Rossum and Drake, 2009) that runs on Google Colab (Bisong, 2019). The data preparation relies on libraries such as pandas (Reback et al., 2021) and numpy (Harris et al., 2020). The deep learning models are developed using Keras 2.8.0 (Chollet et al., 2019), and TensorFlow 2.8.2 (Abadi et al., 2015). The models are trained on NVIDIA GPUs. The evaluation result is tested utilising scikit-learn (Pedregosa et al., 2011). The result visualisation is conducted using matplotlib (Hunter, 2007).

1. Problem statement

The most challenging task in this project is to investigate the possibility of multistep price forecasting 30 days into the future. To take on this challenge, five RNN encoder-decoders are developed. Their performances are compared to their non encoder-decoder version. Thus, the models this project uses are: GRU, bidirectional GRU, CNN-GRU, convLSTM, and bidirectional convLSTM; GRU encoder-decoder, bidirectional GRU encoder-decoder, CNN-GRU encoder-decoder, convLSTM encoder-decoder, and bidirectional convLSTM encoder-decoder.

The comparison intends to discover whether the encoder-decoder version of deep learning models can improve the multistep forecast performance of their original version. This project also attempts to use result visualisation as part of the evaluation process, rather than only relying on evaluation metrics widely used such as MSE or RMSE. This new visualised evaluation method enables quick quality assessment of the long term prediction results.

The deep learning pipelines developed in this paper includes five steps: 1. Data preparation, 2. Metric function and callback definition, 3. Model definition, 4. Training and testing, 5. Model evaluation, 6. Result visualisation. They are introduced in the following sections.

2. Data preparation

The data used in this paper are the five-year daily price data of the Great British pound (GBP) in Chinese yuan (CNY). The data starts from 16th January 2017, and ends on 14th January 2022. Compared to many markets that only trade on working days, the foreign exchange (FX) market trades 24/7, creating better data continuity.

To begin with, the data are downloaded from the Bloomberg terminal (Bloomberg L.P., 2022b). They are then uploaded to github (2022) for fast data loading on Google Colab. Once the price data are imported into Google Colab, they are transferred in to an array with a length of 1305.

Next, the data array is split into training and testing data with lengths 1044 and 261. Then, both training and testing data are split further into input and labels. The input has 100 timesteps, while the labels have 30 timesteps. This enables the model to use the price data from the previous 100 days as input to forecast the price in the next 30 days. After this step, the input training data and their label have shapes (915, 100) and (915, 30); as for the testing data, their shapes are (132, 100) and (132, 30).

Finally, the input data is reshaped from [samples, timesteps] to [samples, timesteps, features] for feeding into the model. Since this paper implements a univariate time

series prediction, the number of features is 1. The labels do not need to be reshaped. Therefore, after reshaping, the input training and testing data have shapes (915, 100, 1) and (132, 100, 1).

CNN-GRU, convLSTM, and convLSTM encoder-decoder require different input shapes: [samples, timesteps, rows, features]. It is not ideal for them to have a single sequence of input i.e. a single read. Thus, the input with 100 timesteps are split into 5 subsequences, each with a length of 20 timesteps. The CNN and the convLSTM can then read across the 5 timesteps and perform the convolutional process on the 20 days of data within each timestep. Thus, the shapes for their input training and testing data were (915, 5, 20, 1) and (132, 5, 20, 1). However, CNN-GRU encoder-decoder still has the input shape [samples, timesteps, features] which is not ideal. This is due to the advanced technicality the ideal method requires.

3. Metric function and callbacks

Before introducing the model designs, a metric function and four callbacks are defined. The root mean square error (RMSE) metric function is applied here. Since RMSE metric function is not defined in Keras, it is defined manually. The RMSE is the square root of mean square error (MSE). Their calculations are:

$$MSE = (1/n) * \sum (y' - y)^2 \quad (5)$$

$$RMSE = MSE^{1/2} \quad (6)$$

where n is the number of predictions, y is the label value, and y' is the prediction value.

The callback functions enable the deep learning pipelines to save checkpoints and write the TensorBoard log during training. Four callback functions are used here: the first callback writes checkpoints during training; the second one stops the optimisation when RMSE increases on the testing data i.e. early stopping; the third callback is responsible for logging the progress to TensorBoard during training; the final callback is a learning rate (lr) scheduler which reduces the lr when the RMSE on the testing data does not decrease after the previous epoch.

4. Model definition

The design of a GRU encoder-decoder is revealed in TABLE. 1, it is the focus of this project and will be explained in detail.

TABLE. 1. GRU encoder-decoder model summary

Layer (type)	Output Shape
=====	=====
GRU encoder	(None, 512)
Repeat encoding	(None, 30, 512)
GRU decoder	(None, 30, 512)
Output layer (linear, time distributed)	(None, 30, 256)
Output layer (linear, time distributed)	(None, 30, 1)

As shown in TABLE. 1, the GRU encoder reads and interprets the input data. It outputs a fixed length vector representing its interpretation of the input sequence. The activation function for the GRU is rectified linear unit (ReLU) (Agarap, 2018). For the decoder to read this representation produced by the encoder, the fixed length encoding vector needs to be repeated n times, where n is the number of timesteps in the label i.e. 30.

After repeat encoding, the repeated sequence is fed into the GRU decoder. The decoder generates the output sequence conditioned on the fixed length encoding vector. ReLU activation is also used for the GRU decoder. The output of the decoder has the same shape as the repeated fixed length encoding vector. Finally, two output layers will interpret the output of the decoder and produce the final 30-day prediction. The TimeDistributed wrapper is utilised to apply the linear layers to every timestep in the output of the decoder. This enables the output to take on any arbitrary values. To prevent "NaN" values generated during training, the weights in the output layers are initialised with values close to zero: between -0.03 and 0.03.

Other encoder-decoders work similarly to the GRU encoder-decoder and are not specifically explained. All the model summaries can be found in TABLE. 7 and TABLE. 8 in the appendix.

5. Training and testing

In this paper, the models are trained with the root mean squared propagation (RMSprop) (Tieleman and Hinton, 2012) and optimised using the MSE loss function. RMSprop is similar to the adaptive moment estimation (Adam) (Kingma and Ba, 2014) version of stochastic gradient descent (SGD) (Ruder, 2016). However, there is one important difference between RMSprop and Adam on how they update the parameters. RMSprop uses the momentum on the rescaled gradient, while Adam utilises the running average of first and second moment of the gradient. Since RMSprop was not widely explored in the literature, this paper attempts to fill the gap and experiment with it.

The model was tested with the testing data in the end of every epoch. The training and testing MSE and RMSE are visualised when the training is finished for quick identification of the epoch at which the model starts to overfit.

6. Model evaluation

After training and testing, the models are evaluated. This is conducted using the built-in API from Keras: model.evaluate(). It returns the loss (MSE) and the metric (RMSE) for the models in test mode.

To test the evaluation results, the MSE and RMSE are calculated again manually. To implement this, another built-in API from Keras: model.predict() can be used to generate predictions. Thus, the MSE and RMSE can be calculated again manually using Eq.(5). and Eq.(6). The MSEs and RMSEs calculated manually are compared with these computed using model.evaluate(). Taking into account the rounding errors, the two methods produce similar MSEs and RMSEs in all the results obtained.

7. Visualisation

Eventually, the predictions are visualised. This step is also an extension of model evaluation, since humans can quickly evaluate visualised results. To achieve this, the model predictions are plotted along with the actual data.

The visualisation step requires its own data preparation substep. The training and testing data created before are not used to produce the predictions for plotting, because they sacrificed or lost some data for creating the labels. The data used for predictions are generated without considering labels. Thus, they are larger in size with shapes (944, 100) and (161,

100), comparing to the training and testing data with shape (915, 100) and (132, 100).

There are no predictions for the first 100 timesteps, since the first 30-day predictions are created with the past 100-day data. After the first 30-day forecast is plotted, the next 30-day forecast is plotted from timestep 130. Even though a 30-day forecast from timestep 101 is available, it overlaps 29 timesteps with the first 30-day predictions. Thus, it is not shown. The same rule applies to all overlapping predictions to ensure a clean presentation.

To help readers identify each 30-day forecasting window, ticks are added to the x-axis. They are located at the beginning of every 30-day forecasting window except for the last one. The reason for this is because the forecasting window before the last 30-day forecasting window is not always 30 timesteps long as the training and testing data lengths: 1044 and 261, minus the input sequence length: 100, is not always divisible by the forecasting window length: 30. Thus, this paper merges the last 30-day forecasting window with the one before it.

Ultimately, the actual training and testing data, and the predictions are plotted.

V. Results and conclusions

The detailed results from all models are presented in TABLE. 5 and TABLE. 6 in the appendix.

First of all, the most noticeable finding is that the bidirectional GRU and its encoder-decoder version outperform all other models. This discovery is supported by

both the evaluation metrics and the results visualisation. The testing RMSEs for the two models are the smallest in their groups: the non encoder-decoder group and the encoder-decoder group. The testing RMSE of the bidirectional GRU encoder-decoder is 0.122 compared to 0.127 obtained from the bidirectional GRU. This performance gain is visualised in TABLE 2. where the predictions from the bidirectional GRU encoder-decoder are closer to the actual data. From TABLE. 2, it is clear that the encoder-decoder reacts better to large price swings. Therefore, the best model discovered in this project is the bidirectional GRU encoder-decoder. This result confirms the discovery by Graves and Schmidhuber (2005), and Sutskever, Vinyals and Le (2014). The later group found that reversing the input sequence improved the prediction performance for longer sequences.

The second finding is that a model can achieve better MSE and RMSE results even if all its predictions are on a single straight lines as demonstrated in TABLE. 3. This result supports the necessity for a results visualisation method proposed by this paper, rather than only relying on evaluation metrics. It is interesting however, the straight line prediction in this case is better in practice, since it avoids making the wrong directionality call. The smaller RMSE suggests that the straight-line-predicting model generates better profits.

Finally, despite all models fail to accurately forecast future prices, many models produce directionality calls which indicates the possibility of accurate long term forecasting.

TABLE. 2. Results visualisation comparison: bidirectional GRU versus its encoder-decoder version

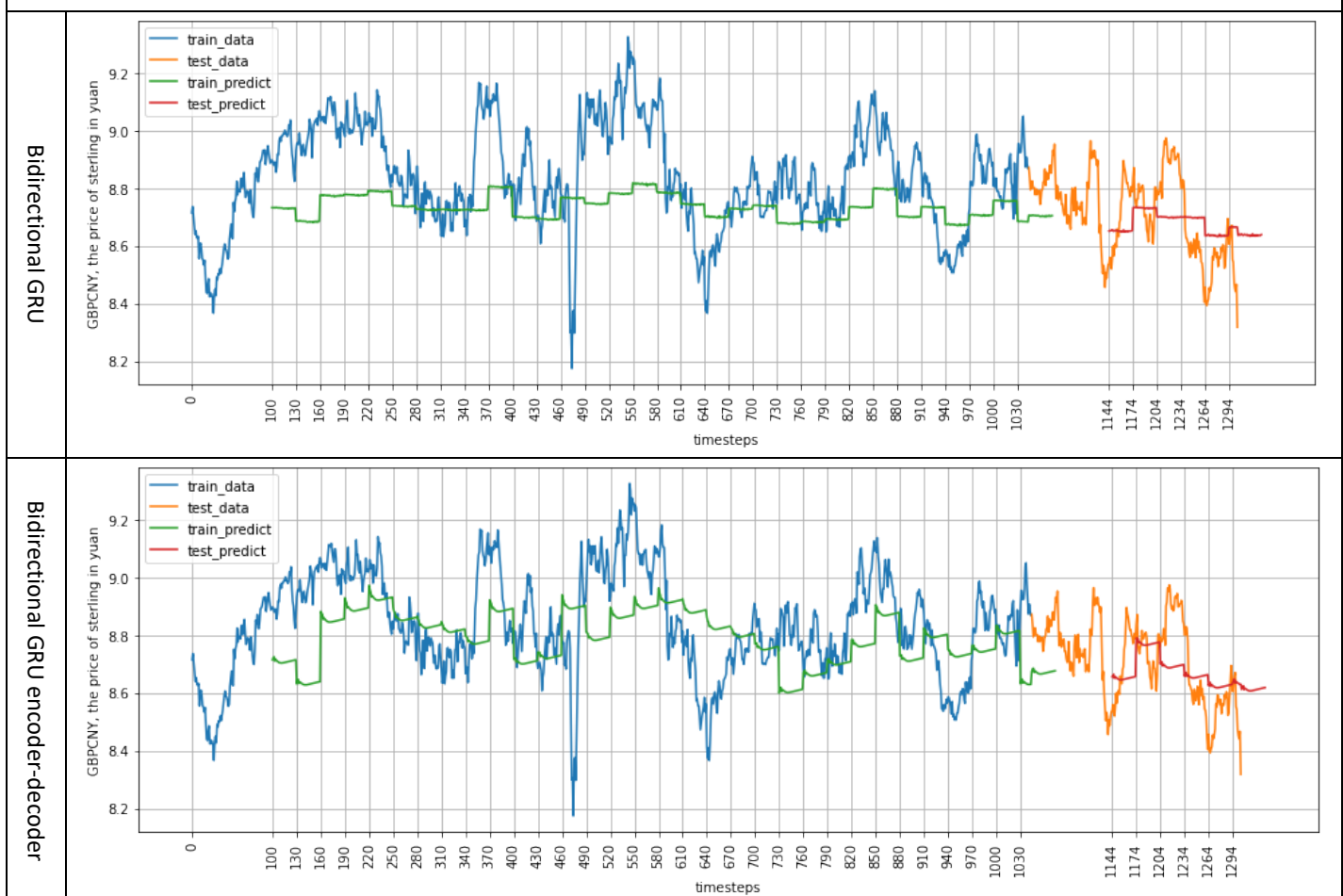
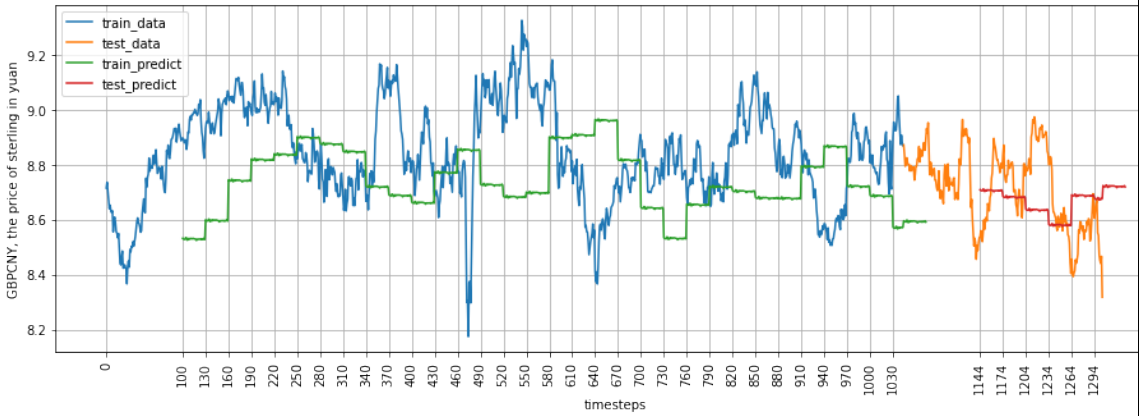
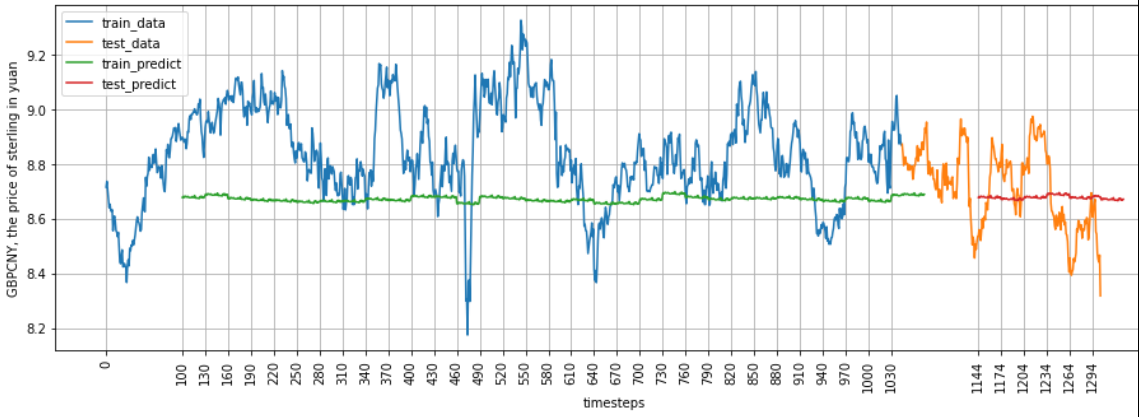


TABLE. 3. Comparing the evaluation metrics and prediction results visualisation of convLSTM and bidirectional convLSTM²

CONVLSTM		Train MSE: 0.064 Test MSE: 0.025 Train RMSE: 0.23 Test RMSE: 0.15
Bidirectional convLSTM		Train MSE: 0.063 Test MSE: 0.023 Train RMSE: 0.22 Test RMSE: 0.14

VI. Future work

1. Utilise the variable input and output sequence length

This paper fails to apply variable sequence length with the encoder-decoders. The input and the label are all fixed length. Future analyses can feed variable length sequence to the encoder-decoder. The choice of the sequence length can be based on the shape of segments of the price chart. For example, the input can always contain a V shaped segment, and the labels can always contain an Λ shaped segment. Thus, the encoder-decoder could predict when will an Λ shaped price movement happen, given a V shaped price movement happens previously.

2. Expand to multivariate forecasting

Due to lack of data, the bigger and more sophisticated models in this paper underperform. Other than collecting more of the existing univariate data, extending into multivariate forecasting is another way to increase the data size. The variables to consider include the exchange rates for other currencies such as the US dollar, Euro, and yen; central bank interest rates, future and option prices, stock indices, or even commodity prices. L1 regularisation (Ng, 2004) can be used for selecting the suitable variables or features.

3. Apply the transformer models

The results from this paper shows that using the encoder-decoder version of the GRU and the bidirectional GRU can enhance the forecasting performance. In comparison, the transformer model which uses the attention mechanism is different to all models used in this paper (Vaswani et al., 2017). When the RNN encoder-decoders achieved superior results on machine translation tasks in 2014, the transformer models showed even better results three years later. Therefore, transformer models could have the potential to reach better accuracy for long term forecasting.

Furthermore, a transformer encoder-decoder-attention model was implemented in speech recognition. The hybrid model outperformed the transformer model (Zeyer et al., 2019). However, that method is not applied in finance and therefore requires further study.

A more advanced future study may attempt to feed both text data such as news, and numerical data such as prices to the transformer model. Since the market is sensitive to news, models that process both news and price data may achieve better long term forecasting performance.

Reference list

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M. and Levenberg, J. (2015). *TensorFlow*:

² Best evaluation metric results are highlighted

Large-Scale Machine Learning on Heterogeneous Distributed Systems. [online] Available at: <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf>.

Abbasiantaeb, Z. and Momtazi, S. (2021). Text-based question answering from information retrieval and deep neural network perspectives: A survey. *WIREs Data Mining and Knowledge Discovery*. doi:10.1002/widm.1412.

Agarap, A.F. (2018). *Deep Learning using Rectified Linear Units (ReLU)*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1803.08375>.

Allaouzi, I., Ahmed, M. and Benamrou, B. (2019). *An Encoder-Decoder model for visual question answering in the medical domain*.

Althelaya, K.A., El-Alfy, E.-S.M. and Mohammed, S. (2018). *Stock Market Forecast Using Multivariate Analysis with Bidirectional and Stacked (LSTM, GRU)*. [online] IEEE Xplore. doi:10.1109/NCG.2018.8593076.

Bahdanau, D., Cho, K. and Bengio, Y. (2014). *Neural Machine Translation by Jointly Learning to Align and Translate*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1409.0473>.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P. and Bengio, Y. (2016). *End-to-end attention-based large vocabulary speech recognition*. [online] IEEE Xplore. doi:10.1109/ICASSP.2016.7472618.

Bisong, E. (2019). Google Colaboratory. *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp.59–64. doi:10.1007/978-1-4842-4470-8_7.

Bloomberg L.P. (2013). *BLOOMBERG FUND CLASSIFICATIONS Guide to our new Funds Classification System*. Available through Bloomberg terminal using command LPHP HF:0:1 3756368 GO.

Bloomberg L.P. (2022a). *Fund Screening*. Available through Bloomberg terminal using command FSRC .

Bloomberg L.P. (2022b). *FX Session Chart GBPCNY 2017/1/16 - 2022/1/14*. Available through Bloomberg terminal using command GBPCNY GO.

Brownlee, J. (2018). *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery.

Chen, H., Huang, S., Chiang, D. and Chen, J. (2017). Improved Neural Machine Translation with a Syntax-Aware Encoder and Decoder. *arXiv:1707.05436 [cs]*. [online] Available at: <https://arxiv.org/abs/1707.05436> [Accessed 14 Jul. 2022].

Chiu, C.-C., Sainath, T.N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R.J., Rao, K., Gonina, E., Jaitly, N., Li, B., Chorowski, J. and Bacchiani, M. (2018). State-of-the-Art Speech Recognition with Sequence-to-Sequence Models. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. doi:10.1109/icassp.2018.8462105.

Cho, K., van Merriënboer, B., Bahdanau, D. and Bengio, Y. (2014a). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. [online] Available at: <https://arxiv.org/abs/1409.1259>.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. (2014b). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. [online] Available at: <https://arxiv.org/abs/1406.1078?context=cs> [Accessed 12 Jul. 2022].

Chollet, F. (2018). *Deep Learning with Python*. Shelter Island (New York, Estados Unidos): Manning, Cop.

Chollet, F. & others (2019). *Keras*. [online] Available at: <https://github.com/keras-team/keras>.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing. *Proceedings of the 25th international conference on Machine learning - ICML '08*. doi:10.1145/1390156.1390177.

Dahl, G.E., Yu, D., Deng, L. and Acero, A. (2012). Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1), pp.30–42. doi:10.1109/tasl.2011.2134090.

Deng, L., Hinton, G. and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: an overview. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. doi:10.1109/icassp.2013.6639344.

Donahue, J., Hendricks, L.A., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K. and Darrell, T. (2014). *Long-term Recurrent Convolutional Networks for Visual Recognition and Description*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1411.4389>.

- Fama, E.F. (1970). Efficient Capital Markets: a Review of Theory and Empirical Work. *The Journal of Finance*, [online] 25(2), pp.383–417. doi:10.2307/2325486.
- Financial Times (2022). *FT – About Us*. [online] ABOUT THE FT. Available at: <https://aboutus.ft.com/> [Accessed 30 May 2022].
- Flores, A., Tito, H. and Centty, D. (2020). Comparison of Hybrid Recurrent Neural Networks for Univariate Time Series Forecasting. *Advances in Intelligent Systems and Computing*, pp.375–387. doi:10.1007/978-3-030-55180-3_28.
- Gallicchio, C., Micheli, A. and Pedrelli, L. (2019). Comparison between DeepESNs and gated RNNs on multivariate time-series prediction. *arXiv:1812.11527 [cs, stat]*.
- Gamboa, J.C.B. (2017). *Deep Learning for Time-Series Analysis*. [online] arXiv. Available at: <https://arxiv.org/abs/1701.01887> [Accessed 17 Jun. 2022].
- github (2022). [online] GitHub. Available at: <https://github.com/>.
- Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep learning*. Cambridge, Massachusetts: The Mit Press.
- Graves, A. (2012). *Supervised sequence labelling with recurrent neural networks*. Berlin: Springer.
- Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6), pp.602–610. doi:10.1016/j.neunet.2005.06.042.
- Greff, K., Srivastava, R.K., Koutnik, J., Steunebrink, B.R. and Schmidhuber, J. (2015). LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10), pp.2222–2232. doi:10.1109/tnnls.2016.2582924.
- Gregory Mankiw, N. (2020). *Principles Of Economics*. 9th ed. South-Western.
- Hansen, L.P. (1982). Large Sample Properties of Generalized Method of Moments Estimators. *Econometrica*, [online] 50(4), p.1029. doi:10.2307/1912775.
- Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P. and Gérard-Marchant, P. (2020). Array programming with NumPy. *Nature*, 585(7825), pp.357–362. doi:10.1038/s41586-020-2649-2.
- Hartree, D.R. (1947). Recent Developments in Calculating Machines. *Journal of Scientific Instruments*, 24(7), pp.172–176. doi:10.1088/0950-7671/24/7/302.
- Helmore, E. (2022). Google engineer says AI bot wants to ‘serve humanity’ but experts dismissive. *The Guardian*. [online] 13 Jun. Available at: <https://www.theguardian.com/technology/2022/jun/13/google-ai-bot-sentence-experts-dismissive-blake-lemoine>.
- Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. and Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6), pp.82–97. doi:10.1109/msp.2012.2205597.
- Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735–1780. doi:10.1162/neco.1997.9.8.1735.
- Hollings, C., Martin, U. and Rice, A. (2018). Ada Lovelace and the Analytical Engine. Available at: <https://blogs.bodleian.ox.ac.uk/adalovelace/2018/07/26/ada-lovelace-and-the-analytical-engine/> [Accessed 16 Jun. 2022].
- Hunter, J.D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), pp.90–95. doi:10.1109/mcse.2007.55.
- Iqbal, K., Hassan, A., Hassan, S.S.M.U., Iqbal, S., Aslam, F. and Mughal, K.S. (2021). Forecasting Stock Market Using Machine Learning Approach Encoder-Decoder ConvLSTM. *IEEE Xplore*, pp.43–48. doi:10.1109/FIT53504.2021.00018.
- Iyyer, M., Boyd-Graber, J., Claudino, L., Socher, R. and Ili, H. (2014). *A Neural Network for Factoid Question Answering over Paragraphs*. Association for Computational Linguistics, pp.633–644.
- Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical evaluation of recurrent network architectures. In *ICML’2015*.
- JPMorgan Chase & Co (2019). *J.P. Morgan AI Research*. [online] Technology at Our Firm. Available at: <https://www.jpmorgan.com/technology/artificial-intelligence> [Accessed 30 May 2022].

- Keim, D.A. (2002). Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), pp.1–8. doi:10.1109/2945.981847.
- Kingma, D.P. and Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1412.6980>.
- Kumar, A. and Sarawagi, S. (2019). Calibration of Encoder Decoder Models for Neural Machine Translation. *arXiv:1903.00802 [cs, stat]*. [online] Available at: <https://arxiv.org/abs/1903.00802> [Accessed 14 Jul. 2022].
- LeCun, Y. (1989). Generalization and network design strategies. *Technical Report*.
- Liu, L., Watanabe, T., Sumita, E. and Zhao, T. (2013). *Additive Neural Networks for Statistical Machine Translation*. Association for Computational Linguistics, pp.791–801.
- Liu, X., Lin, Z. and Feng, Z. (2021). Short-term offshore wind speed forecast by seasonal ARIMA - A comparison against GRU and LSTM. *Energy*, 227, p.120492. doi:10.1016/j.energy.2021.120492.
- Lu, L., Zhang, X. and Renais, S. (2016). On training the recurrent neural network encoder-decoder for large vocabulary end-to-end speech recognition. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. doi:10.1109/icassp.2016.7472641.
- Lu, L., Zhang, X., Cho, K. and Renals, S. (2015). *A Study of the Recurrent Neural Network Encoder-Decoder for Large Vocabulary Speech Recognition*.
- Maass, W. (1997). Fast Sigmoidal Networks via Spiking Neurons. *Neural Computation*, 9(2), pp.279–304. doi:10.1162/neco.1997.9.2.279.
- Makin, J.G., Moses, D.A. and Chang, E.F. (2020). Machine translation of cortical activity to text with an encoder–decoder framework. *Nature Neuroscience*, [online] 23(4), pp.575–582. doi:10.1038/s41593-020-0608-8.
- Man in Space*, (1955). IMDb. The Walt Disney Company. 9 Mar.
- Marchi, E., Vesperini, F., Weninger, F., Eyben, F., Squartini, S. and Schuller, B. (2015). *Non-linear prediction with LSTM recurrent neural networks for acoustic novelty detection*. [online] IEEE Xplore. doi:10.1109/IJCNN.2015.7280757.
- Mehtab, S. and Sen, J. (2022). Analysis and Forecasting of Financial Time Series Using CNN and LSTM-Based Deep Learning Models. *Lecture Notes in Networks and Systems*, pp.405–423. doi:10.1007/978-981-16-4807-6_39.
- Mohtasham Khani, M., Vahidnia, S. and Abbasi, A. (2021). A Deep Learning-Based Method for Forecasting Gold Price with Respect to Pandemics. *SN Computer Science*, 2(4). doi:10.1007/s42979-021-00724-3.
- Nakayama, H. and Nishida, N. (2017). Zero-resource machine translation by multimodal encoder–decoder network with multimedia pivot. *Machine Translation*, 31(1-2), pp.49–64. doi:10.1007/s10590-017-9197-z.
- Nelson, B. (2022). *NASA Administrator Bill Nelson Explains the First Image Delivered from the James Webb Telescope to President Biden and Vice President Kamala*. [online] Available at: <https://www.whitehouse.gov/briefing-room/statements-releases/2022/07/11/remarks-by-president-biden-and-vice-president-harris-in-a-briefing-to-preview-the-first-images-from-the-james-webb-space-telescope/> [Accessed 12 Jul. 2022].
- Ng, A. (2004). Feature selection, L1 vs. L2 regularization, and rotational invariance. *Twenty-first international conference on Machine learning - ICML '04*. doi:10.1145/1015330.1015435.
- Nguyen, V., Cai, J. and Chu, J. (2019). Hybrid CNN-GRU model for high efficient handwritten digit recognition. *Proceedings of the 2nd International Conference on Artificial Intelligence and Pattern Recognition - AIPR '19*. doi:10.1145/3357254.3357276.
- Nie, Y., Han, Y., Huang, J., Jiao, B. and Li, A. (2017). Attention-based encoder-decoder model for answer selection in question answering. *Frontiers of Information Technology & Electronic Engineering*, 18(4), pp.535–544. doi:10.1631/fitee.1601232.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, [online] 12(85), pp.2825–2830. Available at: <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>.
- Rajagukguk, R.A., Ramadhan, R.A.A. and Lee, H.-J. (2020). A Review on Deep Learning Models for Forecasting Time Series Data of Solar Irradiance and Photovoltaic Power. *Energies*, 13(24), p.6623. doi:10.3390/en13246623.
- Reback, J., jbrockmendel, McKinney, W., Bossche, J.V. den, Augspurger, T., Cloud, P., Hawkins, S., Roeschke, M., gflyoung, Sinhrks, Klein, A., Petersen, T., Hoefler, P., Tratner, J., She, C., Ayd, W., Naveh, S., Garcia, M., Darbyshire, J.H.M. and Schendel,

- J. (2021). *pandas-dev/pandas: Pandas 1.3.5*. [online] Zenodo. Available at: <https://zenodo.org/record/5774815#.YtYIX3bMJJEY> [Accessed 19 Jul. 2022].
- Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1609.04747>.
- Schuster, M. and Paliwal, K.K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), pp.2673–2681. doi:10.1109/78.650093.
- Schwenk, H. (2012). Continuous Space Translation Models for Phrase-Based Statistical Machine Translation. *ACLWeb*, [online] pp.1071–1080. Available at: <https://aclanthology.org/C12-2104> [Accessed 14 Jul. 2022].
- Scott, R. (2012). *Alien: Prometheus*. 20th Century Studios.
- Sharma, Y. and Gupta, S. (2018). Deep Learning Approaches for Question Answering System. *Procedia Computer Science*, 132, pp.785–794. doi:10.1016/j.procs.2018.05.090.
- Shiller, R.J. (1981). Do Stock Prices Move Too Much to be Justified by Subsequent Changes in Dividends? *The American Economic Review*, [online] 71(3), pp.421–436. Available at: <https://www.jstor.org/stable/1802789> [Accessed 12 Jul. 2022].
- Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W. and Woo, W. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *arXiv:1506.04214 [cs]*.
- Smith, A. (1776). *The Wealth of Nations*.
- Sutskever, I., Vinyals, O. and Le, Q.V. (2014). *Sequence to Sequence Learning with Neural Networks*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1409.3215>.
- The Noble Prize (2013). *The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel 2013*. [online] NobelPrize.org. Available at: <https://www.nobelprize.org/prizes/economic-sciences/2013/press-release/> [Accessed 14 Jul. 2022].
- Tieleman, T. and Hinton, G. (2012). *Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude*. Coursera: Neural networks for machine learning.
- Toshniwal, S., Kannan, A., Chiu, C.-C., Wu, Y., Sainath, T.N. and Livescu, K. (2018). *A Comparison of Techniques for Language Model Integration in Encoder-Decoder Speech Recognition*. [online] IEEE Xplore. doi:10.1109/SLT.2018.8639038.
- Turing, A.M. (1950). Computing Machinery and Intelligence. *Mind*, LIX(236), pp.433–460. doi:10.1093/mind/lix.236.433.
- University of Oxford (2022). *Oxford Man Institute of Quantitative Finance*. [online] Oxford Man Institute. Available at: <https://www.oxford-man.ox.ac.uk/> [Accessed 30 May 2022].
- van Rossum, G. and Drake, F.L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Brain, G., Research, G., Jones, L., Gomez, A., Kaiser, Ł. and Polosukhin, I. (2017). *Attention Is All You Need*. [online] Available at: <https://arxiv.org/pdf/1706.03762.pdf>.
- Verbeek, M. (2017). *A Guide to Modern Econometrics*. Wiley.
- Von Braun, W. (1939). Proposal for a Workable Fighter with Rocket Drive.
- Wooldridge, J.M. (2018). *Introductory Econometrics: A Modern Approach*. Cengage Learning.
- Yamak, P.T., Yujian, L. and Gadosey, P.K. (2019). A Comparison between ARIMA, LSTM, and GRU for Time Series Forecasting. *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*. doi:10.1145/3377713.3377722.
- Yin, J., Jiang, X., Lu, Z., Shang, L., Li, H. and Li, X. (2016). Neural Generative Question Answering. *arXiv:1512.01337 [cs]*. [online] Available at: <https://arxiv.org/abs/1512.01337> [Accessed 14 Jul. 2022].
- Zeyer, A., Bahar, P., Irie, K., Schlüter, R. and Ney, H. (2019). A Comparison of Transformer and LSTM Encoder Decoder Models for ASR. *IEEE Xplore*, pp.8–15. doi:10.1109/ASRU46091.2019.9004025.
- Zhang, A., Lipton, Z., Li, M. and Smola, A. (2021a). Dive into Deep Learning. *arXiv*. [online] Available at: <https://arxiv.org/abs/2106.11342> [Accessed 16 Jun. 2022].
- Zhang, B., Zou, G., Qin, D., Lu, Y., Jin, Y. and Wang, H. (2021b). A novel Encoder-Decoder model based on read-first LSTM for air pollutant prediction. *Science of The Total Environment*, 765, p.144507. doi:10.1016/j.scitotenv.2020.144507.

Zhang, K., Cao, H., Thé, J. and Yu, H. (2022). A hybrid model for multi-step coal price forecasting using decomposition technique and deep learning algorithms. *Applied Energy*, 306, p.118011. doi:10.1016/j.apenergy.2021.118011.

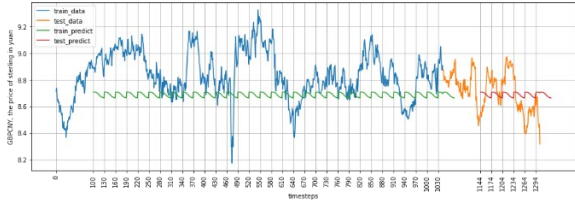
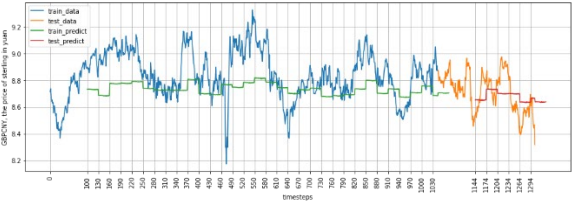
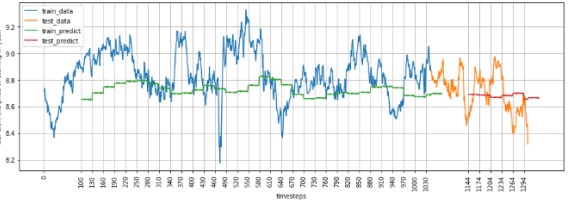
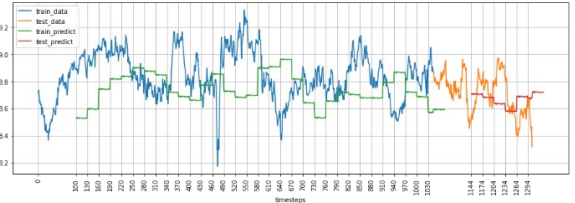
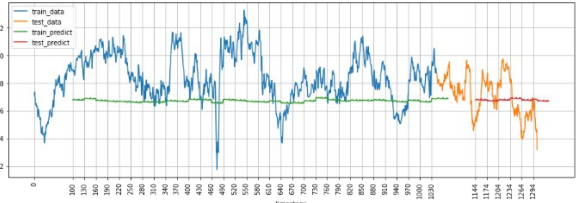
Appendix

1. TABLE. 4. Number of funds mentioned artificial intelligence”, "deep learning" , "machine learning" , and "neural network" in their fund description and number of funds use these technology and the percentage of funds uses the technology

	"artificial intelligence" in fund description	"deep learning" in fund description	"machine learning" in fund description	"neural network" in fund description
Number of funds	87	1	9	0
Number of funds use the technology to generate return	19	0	2	0
Percentage of funds use the technology to generate return	21.84%	0.00%	22.22%	0.00%

Source: Bloomberg L.P. (2022a)

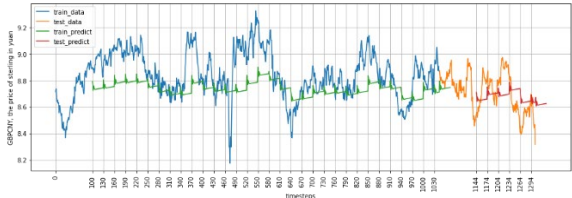
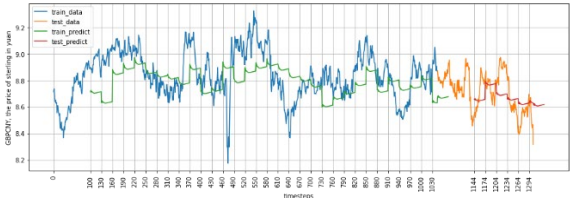
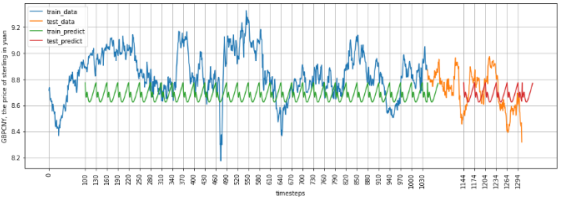
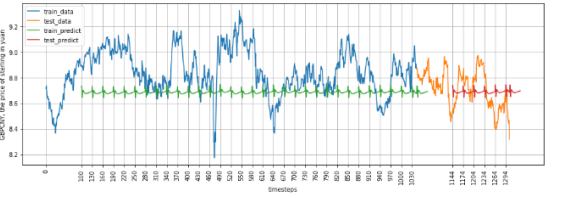
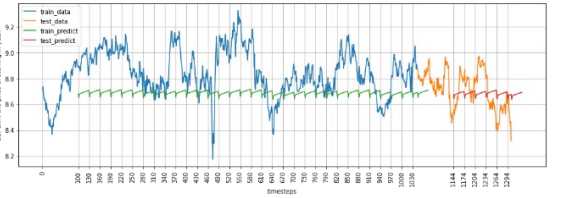
2. TABLE. 5. Compare the results from non encoder-decoders

GRU	Bidirectional GRU	CNN-GRU	ConvLSTM	Bidirectional convLSTM
 <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.05752391368150711</p> <p>Test MSE: 0.02265973761677742</p> <p>Train RMSE: 0.21491873264312744</p> <p>Test RMSE: 0.14584466814994812</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.05752391763827146</p> <p>Test MSE: 0.022659731392709336</p> <p>Train RMSE: 0.2395905609090851</p> <p>Test RMSE: 0.15051444743003192</p> <p>Training data mean: 8.841544827586207</p> <p>Train prediction mean: 8.689421</p> <p>Testing data mean: 8.717213409961685</p> <p>Test prediction mean: 8.689277</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.016529776</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.016499793</p>	 <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.03976937010884285</p> <p>Test MSE: 0.0185531172901392</p> <p>Train RMSE: 0.1800970435142517</p> <p>Test RMSE: 0.12780709564685822</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.03976937067438754</p> <p>Test MSE: 0.01855311665546423</p> <p>Train RMSE: 0.19938069237902686</p> <p>Test RMSE: 0.1356421279941473</p> <p>Training data mean: 8.841544827586207</p> <p>Train prediction mean: 8.734475</p> <p>Testing data mean: 8.717213409961685</p> <p>Test prediction mean: 8.683129</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.03647247</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.03962802</p>	 <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.048990052193403244</p> <p>Test MSE: 0.024246906861662865</p> <p>Train RMSE: 0.19983042776584625</p> <p>Test RMSE: 0.14825963973999023</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.048990054022166905</p> <p>Test MSE: 0.024246903744922787</p> <p>Train RMSE: 0.22130616839441483</p> <p>Test RMSE: 0.15570887889836713</p> <p>Training data mean: 8.841544827586207</p> <p>Train prediction mean: 8.727678</p> <p>Testing data mean: 8.717213409961685</p> <p>Test prediction mean: 8.681865</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.04335591</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.0125021525</p>	 <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.06495089828968048</p> <p>Test MSE: 0.025004159659147263</p> <p>Train RMSE: 0.2350589781999588</p> <p>Test RMSE: 0.15016736090183258</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.06495088208357877</p> <p>Test MSE: 0.02500414748230672</p> <p>Train RMSE: 0.2548423741563481</p> <p>Test RMSE: 0.15792475463126315</p> <p>Training data mean: 8.841544827586207</p> <p>Train prediction mean: 8.745656</p> <p>Testing data mean: 8.717213409961685</p> <p>Test prediction mean: 8.667964</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.10889743</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.03748381</p>	 <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.06305424869060516</p> <p>Test MSE: 0.023927541449666023</p> <p>Train RMSE: 0.22582100331783295</p> <p>Test RMSE: 0.1483396589756012</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.06305424737649105</p> <p>Test MSE: 0.02392753740501844</p> <p>Train RMSE: 0.2510999790440946</p> <p>Test RMSE: 0.15467632760866384</p> <p>Training data mean: 8.841544827586207</p> <p>Train prediction mean: 8.67179</p> <p>Testing data mean: 8.717213409961685</p> <p>Test prediction mean: 8.678711</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.009310334</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.00669135</p>

⁴Best evaluation metric results across all models are highlighted

⁴ Best evaluation metric results across all models are highlighted

4. TABLE. 6. Compare the results from encoder-decoders

GRU encoder-decoder	Bidirectional GRU encoder-decoder,	CNN-GRU encoder-decoder,	ConvLSTM encoder-decoder	Bidirectional convLSTM encoder-decoder
<div></div> <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.03755016624927521</p> <p>Test MSE: 0.021501140668988228</p> <p>Train RMSE: 0.175779789686203</p> <p>Test RMSE: 0.13478487730026245</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.03755016534611056</p> <p>Test MSE: 0.02150113514637262</p> <p>Train RMSE: 0.19344108640643307</p> <p>Test RMSE: 0.14624235635938573</p> <p>Training data mean: 8.8415448</p> <p>Train prediction mean: 8.737264</p> <p>Testing data mean: 8.7172134</p> <p>Test prediction mean: 8.68127</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.04829698</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.038502555</p>	<div></div> <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.0361340157687664</p> <p>Test MSE: 0.017722204327583313</p> <p>Train RMSE: 0.17656706273555756</p> <p>Test RMSE: 0.12243817746639252</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.036134016893619864</p> <p>Test MSE: 0.017722201682968338</p> <p>Train RMSE: 0.18994062535148998</p> <p>Test RMSE: 0.13199063083118037</p> <p>Training data mean: 8.841544827586207</p> <p>Train prediction mean: 8.791829</p> <p>Testing data mean: 8.717213409961685</p> <p>Test prediction mean: 8.674898</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.0813758</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.06592484</p>	<div></div> <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.06126480922102928</p> <p>Test MSE: 0.025416381657123566</p> <p>Train RMSE: 0.22355590760707855</p> <p>Test RMSE: 0.15571586787700653</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.06126481624452628</p> <p>Test MSE: 0.025416378394579595</p> <p>Train RMSE: 0.2457190125275923</p> <p>Test RMSE: 0.15917468051336803</p> <p>Training data mean: 8.8415448</p> <p>Train prediction mean: 8.683477</p> <p>Testing data mean: 8.7172134</p> <p>Test prediction mean: 8.683477</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.043824617</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.043824617</p>	<div></div> <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.058769214898347855</p> <p>Test MSE: 0.022925667464733124</p> <p>Train RMSE: 0.2169293910264969</p> <p>Test RMSE: 0.14759452641010284</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.05876921268637836</p> <p>Test MSE: 0.022925663307129467</p> <p>Train RMSE: 0.2422458911728195</p> <p>Test RMSE: 0.15138567003052336</p> <p>Training data mean: 8.8415448</p> <p>Train prediction mean: 8.687228</p> <p>Testing data mean: 8.7172134</p> <p>Test prediction mean: 8.690114</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.014313287</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.014561431</p>	<div></div> <p>Metrics calculated with `model.evaluate()`:</p> <p>Train MSE: 0.056104876101017</p> <p>Test MSE: 0.023265160620212555</p> <p>Train RMSE: 0.2127247005701065</p> <p>Test RMSE: 0.1462825983762741</p> <p>Metrics calculated manually:</p> <p>Train MSE: 0.05610488482060132</p> <p>Test MSE: 0.02326515716196129</p> <p>Train RMSE: 0.23672026318478567</p> <p>Test RMSE: 0.1525120453430656</p> <p>Training data mean: 8.8415448</p> <p>Train prediction mean: 8.690595</p> <p>Testing data mean: 8.717213</p> <p>Test prediction mean: 8.68488</p> <p>Training data standard deviation: 0.17639548961572413</p> <p>Train prediction standard deviation: 0.013160304</p> <p>Testing data standard deviation: 0.14175250034034664</p> <p>Test prediction standard deviation: 0.012573043</p>

⁵ Best evaluation metric results across all models are highlighted

⁵ Best evaluation metric results across all models are highlighted

5. TABLE. 7. Model summary for non encoder-decoders⁶

GRU		Bidirectional GRU		CNN-GRU		ConvLSTM		Bidirectional convLSTM	
Layer (type)	Output Shape	Layer (type)	Output Shape	Layer (type)	Output Shape	Layer (type)	Output Shape	Layer (type)	Output Shape
=====		=====		=====		=====		=====	
gru (GRU)	(None, 512)	gru (Bidirectional)	(None, 512)	conv1d (TimeDistributed)	(None, None, 14, 256)	conv_lstm1d (ConvLSTM1D)	(None, 5, 11, 256)	conv_lstm1d (Bidirectional)	(None, 11, 512)
dense (Dense)	(None, 30)	dense (Dense)	(None, 256)	conv1d (TimeDistributed)	(None, None, 10, 512)	conv_lstm1d (ConvLSTM1D)	(None, 5, 7, 512)	flatten (Flatten)	(None, 5632)
		dense (Dense)	(None, 30)	conv1d (TimeDistributed)	(None, None, 8, 1024)	conv_lstm1d (ConvLSTM1D)	(None, 5, 1024)	dense (Dense)	(None, 2048)
				maxpooling1d (TimeDistributed)	(None, None, 4, 1024)	flatten (Flatten)	(None, 5120)	dense (Dense)	(None, 1024)
				flatten (TimeDistributed)	(None, None, 4096)	dense (Dense)	(None, 2048)	dense (Dense)	(None, 30)
				gru (GRU)	(None, None, 2048)	dense (Dense)	(None, 1024)		
				gru (GRU)	(None, None, 1024)				
				gru (GRU)	(None, 512)				
				dense (Dense)	(None, 256)				
				dense (Dense)	(None, 30)				

⁶ Dense layer is equivalent to linear layer here

6. TABLE. 8. Model summary for encoder-decoders⁷

GRU encoder-decoder		Bidirectional GRU encoder-decoder,		CNN-GRU encoder-decoder,		ConvLSTM encoder-decoder		Bidirectional convLSTM encoder-decoder	
Layer (type)	Output Shape	Layer (type)	Output Shape	Layer (type)	Output Shape	Layer (type)	Output Shape	Layer (type)	Output Shape
=====	=====	=====	=====	=====	=====	=====	=====	=====	=====
gru (GRU)	(None, 512)	gru (Bidirectional)	(None, 512)	conv1d (Conv1D)	(None, 37, 128)	conv_lstm1d (ConvLSTM1D)	(None, 10, 128)	conv_lstm1d (Bidirectional)	(None, 16, 128)
repeat_vector (RepeatVector)	(None, 30, 512)	repeat_vector (RepeatVector)	(None, 30, 512)	conv1d (Conv1D)	(None, 10, 256)	flatten (Flatten)	(None, 1280)	flatten (Flatten)	(None, 2048)
gru (GRU)	(None, 30, 512)	gru (GRU)	(None, 30, 512)	maxpooling1d (MaxPooling1D)	(None, 4, 256)	repeat_vector (RepeatVector)	(None, 30, 1280)	repeat_vector (RepeatVector)	(None, 30, 2048)
dense (TimeDistributed)	(None, 30, 256)	dense (TimeDistributed)	(None, 30, 256)	flatten (TimeDistributed)	(None, None, 1024)	gru (GRU)	(None, 30, 640)	gru (GRU)	(None, 30, 512)
dense (TimeDistributed)	(None, 30, 1)	dense (TimeDistributed)	(None, 30, 1)	repeat_vector (RepeatVector)	(None, 30, 1024)	dense (TimeDistributed)	(None, 30, 320)	dense (TimeDistributed)	(None, 30, 256)
				gru (GRU)	(None, 30, 1024)	dense (TimeDistributed)	(None, 30, 1)	dense (TimeDistributed)	(None, 30, 1)
				gru (GRU)	(None, 30, 512)				
				dense (TimeDistributed)	(None, 30, 256)				
				dense (TimeDistributed)	(None, 30, 1)				

⁷ Dense layer is equivalent to linear layer here