**Methodology**

The internal state of the Recurrent Neuron Network (RNN) can exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process input sequences of arbitrary timing, which makes them easier to handle. Traditional neural networks (DNNs) cannot model time series, and the output of neurons in the previous layer can only be passed to the neurons in the next layer. In the recurrent neural network (RNN), the output of the neuron can be passed to itself at the next moment, and it also outputs a hidden layer state for the current layer to use when processing the next sample. Fully connected neural network with self-loop feedback. The timing information of many tasks is very important, that is, the information input before and after a sample is related. The time sequence information of sample appearance is very important for speech recognition, natural language processing, video recognition and other problems, so for such problems, RNN modeling can be used. The classic RNN model structure is shown in the figure:

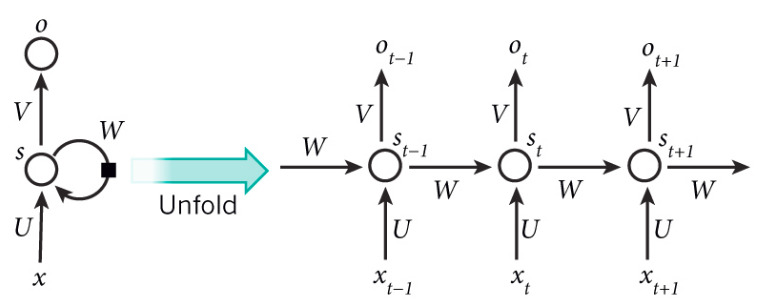


Fig. 1. a recurrent neural network and the unfolding in time of the computation involved in its forward computation

We choose LSTM as the neuron in the neural network because LSTM can solve the problem that RNN cannot handle long-distance dependencies. In response to the problems of RNN, LSTM has two main improvements: a special variable is set to store the cell state, so that the network has a long-term memory. The "gate operation" is introduced to change the accumulation in the gradient into accumulation to solve the problem of gradient disappearance. In the RNN model, the input is the hidden layer state at the previous moment, the input value at the current moment, the output is the hidden layer state at the current moment, and the intermediate operation only contains an activation function.

图示

描述已自动生成

Fig. 2. the internal structure of LSTM neuron

**Data Preparation**

To feed the data to RNN model for training, the data should format as paired sequences, hence the dataset should be windowed like bottom figure. In this experiment, each data chunk with 48 window size will construct as a pair of sequence, the first 24 cells will be the input and the last 24 cells will be the expected prediction. Each time the window move will shifting forward 24 cells.

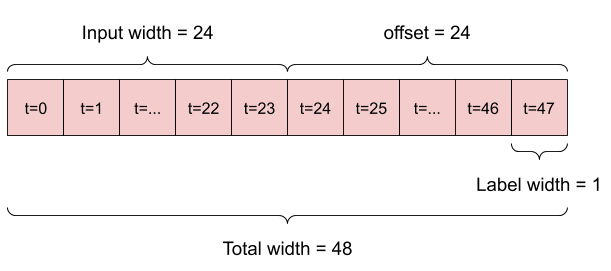


Fig. 3. data windowing method for creating feedings

**Modelling and Results**

The model we designed contains an input layer with 64 LSTM cells and a Dense layer to output the prediction based on the input. Here, the model will accumulate 24 hours of internal state and then make a single prediction for the next 24 hours. In this one-shot format, the LSTM only needs to generate output at the last time step, so we set return\_sequences=False in tf.keras.layers. Some paired sequences are sampled to shows the performance of our model. The dotted line is the input with 24 months, the green circles show the real trend in the next 24 month, and the cross icons denote the prediction result. As shown in the figure below, the predictions present smoother compared to the real trend and the model is hard to fit the trend with high volatility. The rule also applies to the underemployment rate dataset as shown in the next figure below.

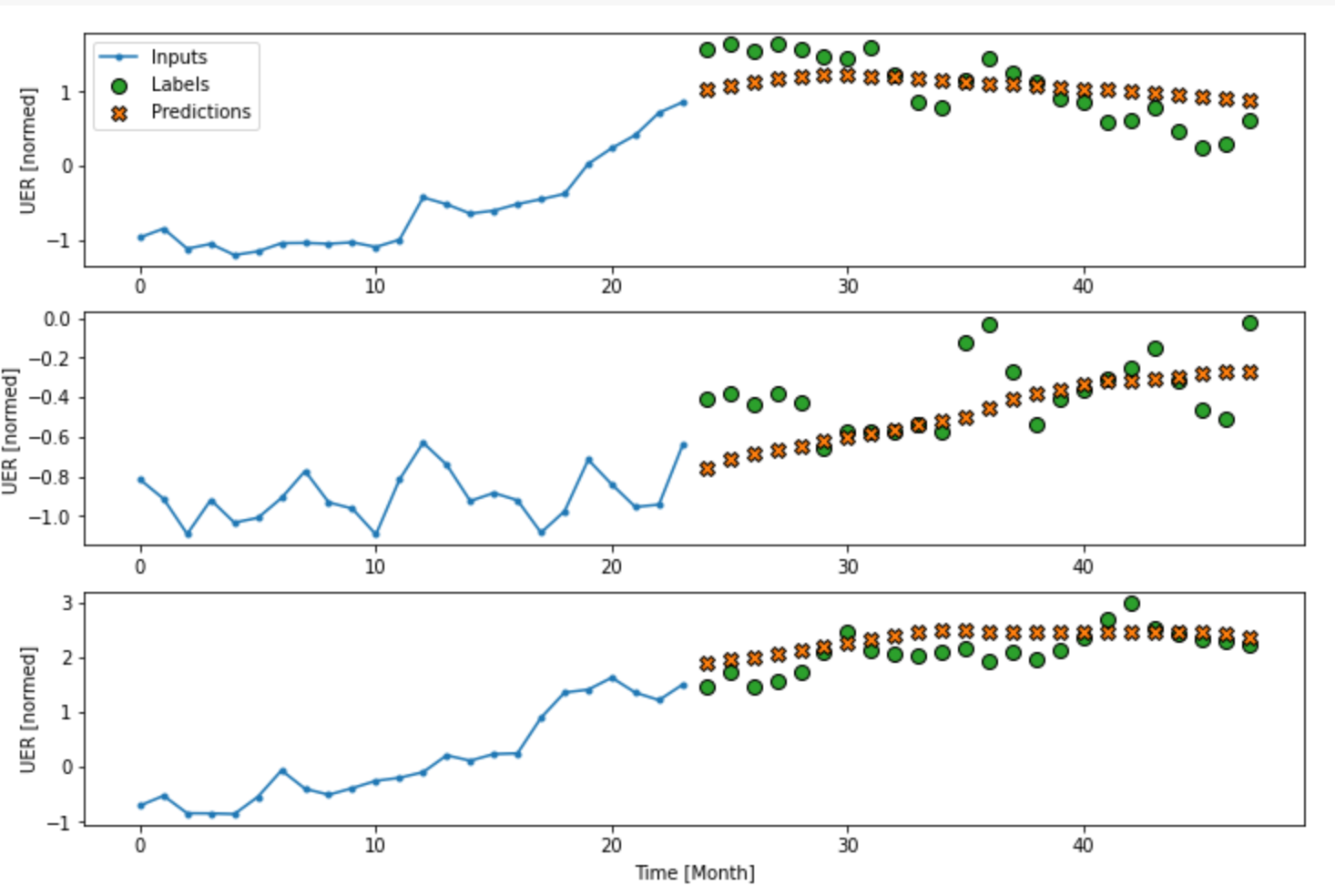
****

Fig. 4. prediction results in random sample sequences

图形用户界面, 图表, 折线图

描述已自动生成

Fig. 5. the prediction result for training data

**Conclusion:**

As shown in the table below, the Holt-Winters additive got the best performance that benefited from it is better to sensing seasonal fluctuations. The trend variation had small change during the 30 years as presented in the figure above. We know that the additive decomposition is the most appropriate if the magnitude of the seasonal fluctuations, or the variation around the trend-cycle, does not vary with the level of the time series. On the contrary, the multiplicative decomposition is more appropriate when the variation in the seasonal pattern, or the variation around the trend-cycle, appears to be proportional to the level of the time series. However, the ARIMA model cannot recognize the pattern of the feeding is unstable cause it got a bad performance in this dataset. SARIMA performs better than ARIMA in this dataset due to the consideration of seasonal factors. Unlike the models mentioned before, RNNs can process arbitrary input sequences using their internal memory, but it is hard to fit the regular fluctuations.

Tab. 1. Performance comparison

|  |  |
| --- | --- |
| Model | RMSE |
| Decomposition\_additive | 2.795327 |
| Holtwinter\_additive | 0.161466 |
| Holtwinter\_multiplication | 0.279546 |
| SARIMA(5,1,1)(0,1,1,12) | 0.181048 |
| Neural Network | 0.237509 |

**References:**

1. <https://medium.com/analytics-vidhya/undestanding-recurrent-neural-network-rnn-and-long-short-term-memory-lstm-30bc1221e80d>
2. <https://www.tensorflow.org/tutorials/structured_data/time_series>