SpringNet: Transformer and Spring DTW for Time Series Forecasting

Yang Lin[1(✉)], Irena Koprinska[1(✉)], and Mashud Rana[2]

[1] School of Computer Science, University of Sydney, Sydney, NSW, Australia
ylin4015@uni.sydney.edu.au, irena.koprinska@sydney.edu.au
[2] Data61, CSIRO, Sydney, Australia
mdmashud.rana@data61.csiro.au

Abstract. In this paper, we present SpringNet, a novel deep learning approach for time series forecasting, and demonstrate its performance in a case study for solar power forecasting. SpringNet is based on the Transformer architecture but uses a Spring DTW attention layer to consider the local context of the time series data. Firstly, it captures the local shape of the time series with Spring DTW attention layers, dealing with data fluctuations. Secondly, it uses a batch version of the Spring DTW algorithm for efficient computation on GPU, to facilitate applications to big time series data. We comprehensively evaluate the performance of SpringNet on two large solar power data sets, showing that SpringNet is an effective method, outperforming the state-of-the-art DeepAR and LogSparse Transformer methods.

Keywords: Time series forecasting · Solar power forecasting · Transformer · Dynamic Time Warping · Deep learning · Dynamic programming

1 Introduction

Time series forecasting is an important task in many domains, e.g. forecasting stock prices, sales and spending, traffic flow, electricity consumption and generated solar power. The traditional autoregressive and state-space models fit each of the related time series independently and require expertise in manually selecting trend and seasonality which limits their applicability [1].

Recently, deep learning methods have been investigated as an alternative. Salinas et al. [2] proposed DeepAR, a probabilistic forecasting model based on sequence-to-sequence Long Short Term Memory (LSTM) neural networks. However, the vanishing and exploding gradient problem of LSTM makes training difficult, especially when processing long sequences. The Transformer architecture [3] has been recently proposed to model sequential data with attention mechanism only, without any recurrent or convolutional layers. Its main advantage is the ability to access any part of the historical sequence regardless of distance. Li et al. [1] proposed the LogSparse Transformer, a modification of the Transformer

H. Yang et al. (Eds.): ICONIP 2020, LNCS 12534, pp. 616–628, 2020.
https://doi.org/10.1007/978-3-030-63836-8_51

for time series forecasting, aiming to overcome the problem of locality-agnostics and memory bottleneck by employing convolutional attention layers and sparse attention mechanism. However, the LogSparse Transformer may have limited ability to capture the time series shape information because the shape could be distorted after being projected into latent space with lower dimension after convolutions. For example, two series with similar shapes that are shifted or scaled over the time axis may have completely different results after convolutions.

On the other hand, Dynamic Time Warping (DTW) [4] is a classic trajectory similarity measure that can handle temporal distortions, such as shifting and scaling in the time axis. It has also been used in sequential modelling tasks, including time series analysis [5–7]. The main drawback of DTW is its high complexity, due to the non-parallelizable characteristics of dynamic programming, which limits its applicability. A variation of DTW is the Spring DTW algorithm [8], which identifies subsequences in a data stream, that are similar to a given template. We propose to use the String DTW algorithm as an attention mechanism for Transformer architectures.

In this paper, we present a new deep learning approach, SpringNet, for time series forecasting. SpringNet is based on the Transformer architecture but utilizes Spring DWT attention layers that measure the similarities of query-key pairs of sequences. We assume that attending to the shape of time series patterns directly would be beneficial to achieve accurate prediction. SpringNet is the first Transformer that attends to the shape of time series patterns directly with Spring attention layers. We also propose a batch version of the Spring DTW algorithm for GPU acceleration, by identifying a batch of matched subsequences concurrently.

The effectiveness of the proposed SpringNet approach is comprehensively evaluated for solar power forecasting using two big data sets. The results show that SpringNet outperformed the state-of-the-art deep learning models DeepAR and LogSparse Transformer and the persistence baseline, especially under random fluctuations of data. The batch version of the Spring DWT algorithm in SpringNet was also found to be significantly faster than the original Spring DWT.

2 Case Study: Solar Power Forecasting

Solar photovoltaic (PV) power is a cost-effective and sustainable electricity source. However, the power output is highly variable as it depends on the weather conditions. PV power forecasting is needed to quantify the uncertainty associated with power generation and ensure the successful integration of PV systems into the electricity grid. Previous work on solar power forecasting includes statistical methods such as autoregressive integrated moving average [9] and machine learning methods such as neural networks [10] and support vector regression [11].

2.1 Data Sets

We use two solar power data sets: Sanyo[1] and Hanergy.[2] They contain solar power generation data from two PV plants in Alice Springs, Northern Territory, Australia. The Sanyo dataset contains solar power data from 01/01/2011 to 31/12/2017, and the Hanergy dataset - from 01/01/2011 to 31/12/2016.

We also collected weather data from nearby weather stations. The weather data includes *temperature, humidity, global and diffuse radiation*. In addition, we also prepared weather forecast data based on historical weather data. Specifically, weather forecasts are formed by adding 20% Gaussian noise to the observed weather data since we did not have access to weather forecast data. Both solar power and weather data are aggregated to 30-min intervals by taking the average , and only the data between 7 am and 5 pm is considered [10]. The missing values in the raw data (1.25% in Sanyo and 3.24% in Hanergy) are filled using the multivariate imputation by chained equations algorithm [12]. Both solar power and weather data are normalized to have zero mean and unit variance.

In addition to solar power and weather data, we also consider the calendar information as inputs to the prediction models. The calendar information (time features) we consider include *month, hour-of-the-day, minute-of-the-hour* [1,2].

2.2 Problem Statement

We use the solar power for day d with associated covariate information for days d (weather and calendar features) and day $d+1$ (weather forecast and calendar features) to forecast the solar power for the next day $d+1$. Specifically, given is a set of N: 1) solar power time series $\{\mathbf{PV}_{i,1:T_l}\}_{i=1}^{N}$, where $\mathbf{PV}_{i,1:T_l} \triangleq [\mathrm{PV}_{i,1}, \mathrm{PV}_{i,2}, ..., \mathrm{PV}_{i,T_l}]$, T_l is the input sequence length, $T_l = 20$ (1 day), and $\mathrm{PV}_{i,t} \in \Re$ is the ith PV power generated at time t; 2) associated time-based multi-dimensional covariate vectors $\{\mathbf{X}_{i,1:T_l+T_h}\}_{i=1}^{N}$, where $T_h = 20$ (1 day) denotes the length of forecasting horizon. The covariates for our case study include: weather $\{\mathbf{W1}_{i,1:T_l}\}_{i=1}^{N}$, weather forecasts $\{\mathbf{WF}_{i,T_l+1:T_l+T_h}\}_{i=1}^{N}$ and calendar features $\{\mathbf{Z}_{i,1:T_l+T_h}\}_{i=1}^{N}$. Our goal is to predict the PV power for the the next T_h time steps after T_l, i.e. $\{\widehat{\mathbf{PV}}_{i,T_l+1:T_l+T_h}\}_{i=1}^{N}$.

The overall structure of SpringNet is illustrated in Fig. 1. The model's input and output arrangement is the same as that of DeepAR [2] and LogSparse Transformer [1]. At each time step, the inputs of the model are the **PV** values at its previous time step and the covariates **X** (weather **W1** and calendar **Z** features) at the current time step. At the first time step of the encoder, $\mathrm{PV}_{i,0}$ value is initialized as zero. The decoder uses weather forecasts **WF** features instead of weather features **W1**. The decoder is autoregressive, which takes the observation at the previous time step $\widehat{\mathrm{PV}}$ as an input at the current time step.

[1] http://dkasolarcentre.com.au/source/alice-springs/dka-m4-b-phase.
[2] http://dkasolarcentre.com.au/source/alice-springs/dka-m16-b-phase.

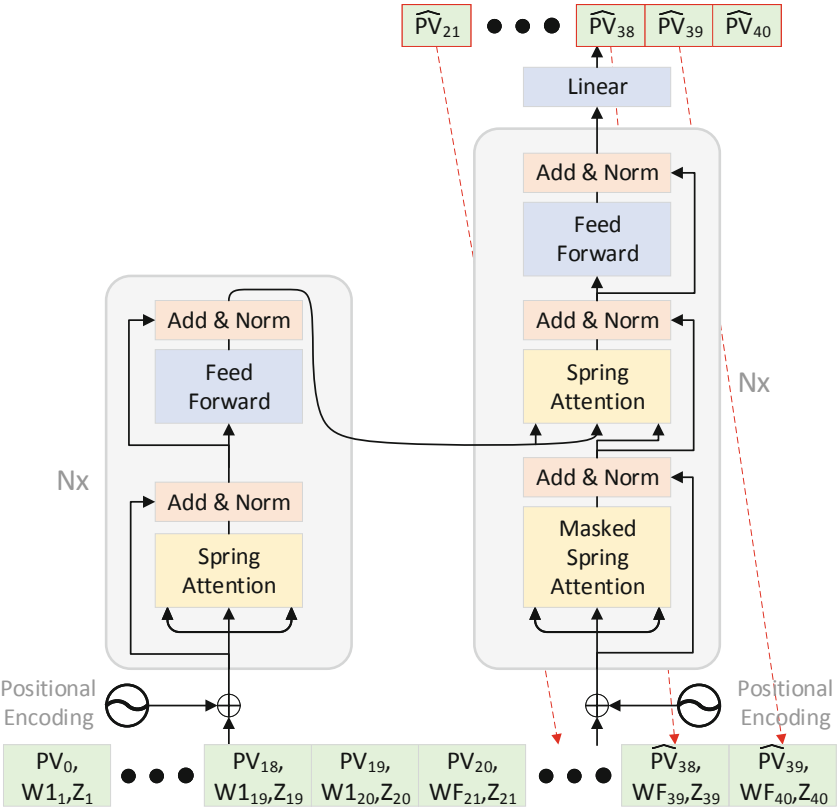

Fig. 1. Summary of SpringNet

3 Background

3.1 Transformer

The Transformer [3] is a new architecture which uses only attention mechanism for processing sequential data. Compared to the widely used sequence models, it does not use any recurrent or convolutional layers, but keeps the encoder-decoder design and uses stacked multi-head self-attention and fully connected layers, which could run in parallel.

Each layer of the encoder contains a multi-head self-attention layer followed by a feed-forward layer, while that of decoder contains an additional encoder-decoder attention layer between the self-attention layer and the feed-forward layer. The multi-head attention uses scaled dot product with the queries Q, keys K and values V. The queries, keys and values are obtained from previous layer output for self-attention and encoder output for encoder-decoder attention. Given input $\mathbf{X}$ of the attention layer, the hth query, key and value matrix can be computed through linear projections with the trainable weights:

$W_h^Q, W_h^K \in \Re^{d_x \times d_k}$ and $W_h^V \in \Re^{d_x \times d_v}$ as shown in (1), where d_k, d_v and d_x are the dimensionality of K, V and $\mathbf{X}$. The encoder-decoder attention layer takes the encoder output to compute keys and values and uses previous decoder output to compute queries.

$$Q_h = \mathbf{X}W_h^Q; K_h = \mathbf{X}W_h^K; V_h = \mathbf{X}W_h^V \tag{1}$$

3.2 LogSparse Transformer

Li et al. [1] proposed the LogSparse Transformer, an improved version of the Transformer for time series forecasting. In particular, they addressed two weaknesses: 1) locality-agnostics (lack of sensitivity to local context which makes the model prone to anomalies) and 2) memory bottleneck - quadratic space complexity as the sequence length increases.

The LogSparse Transformer introduces casual convolutions to transform inputs linearly into queries and keys in the attention layer. The convolution design allows the model to capture local context with a series of queries and keys to further improve the accuracy. Another improvement is the LogSparse attention mechanism, which lets the model attend to part of the past history. The use of LogSparse attention reduces the memory complexity to $O(L(log_2 L)^2)$, where L is the sequence length, which is important for overcoming the memory bottleneck that occurs frequently for long sequences.

3.3 Spring Algorithm

Sakurai et al. [8] proposed the Spring algorithm for finding non-overlapping subsequences in data streams that are similar to a query sequence, using the DTW distance measure. Compared to the naive DTW subsequence searching method, which has $O(n^3 m)$ time complexity (where n is the length of the sequence and m is the length of the query sequence), the Spring algorithm with star padding and Subsequence Time Warping Matrix (STWM) is significantly faster and requires $O(m)$ space and $O(m)$ time per time-tick.

The Spring algorithm has attracted significant interest due to its effectiveness and efficiency. Cai et al. [7] proposed DTWNet which uses the Spring algorithm as a feature extractor for time series classification.

4 Proposed Approach: SpringNet

4.1 Motivation and Novelty

The daily pattern in solar power data could vary significantly over time because it is highly sensitive to weather conditions. For examples, although the overall solar power pattern repeats everyday (an inverse U shape with a peak in the middle of the day), fluctuations caused by weather conditions could occur several times during the day, significantly changing this pattern. The time of occurrence and the magnitude of these fluctuations vary substantially.

Algorithm 1: SpringNet attention algorithm

input : $Q, K \in \Re^{n_{batch} \times n_{head} \times L \times d_k}$, $V \in \Re^{n_{batch} \times n_{head} \times L \times d_v}$, $L_{sub} \in \mathbb{Z}^+$, $\mathcal{F} : (\Re^{N \times L_{sub} \times d_k}, \Re^{N \times L \times d_k}, \mathbb{Z}^+) \rightarrow \Re^{N \times n_{top} \times 2}$, $n_{top} \in \mathbb{Z}^+$

output: $O \in \Re^{n_{batch} \times n_{head} \times L \times d_v}$

init: $Q_{sub} \triangleq \Re^{L \times n_{batch} \times n_{head} \times L_{sub} \times d_k}$; $K_{temp} \triangleq \Re^{L \times n_{batch} \times n_{head} \times L \times d_k}$; $V_{sub} \triangleq \Re^{n_{batch} \times n_{head} \times n_{top} \times d_v}$; $D \triangleq \Re^{n_{batch} \times n_{head} \times n_{top} \times 1}$; $N = L \times n_{batch} \times n_{head}$;

`// Preprocessing`

for $l \leftarrow 1$ **to** L **do**

  $Q_{sub}[l, :, :, :, :] = Q[:, :, l : l + L_{sub}, :]$;

  $K_{temp}[l] = K$;

end

reshape Q_{sub} to $\Re^{N \times L_{sub} \times d_k}$;

reshape K_{temp} to $\Re^{N \times L \times d_k}$;

`// Batch Spring DTW function`

$matrix \leftarrow \mathcal{F}(Q_{sub}, K, n_{top})$;

$V_{sub}, D \leftarrow V, matrix$;

$O = \text{softmax}(D) \times V_{sub}$;

The Transformer cannot capture the local context of such time series data with its canonical self-attention layers [1]. While the LogSparse Transformer is able to capture local context, it could miss the time series shape information during convolutions.

On the other hand, DTW was designed to measure time series similarity with temporal distortions. Motivated by the success of recent works that combine DTW with deep learning [5–7], we leverage DTW to compute attention scores.

The Spring DTW algorithm for subsequences matching is effective and efficient (compared to the naive DTW) but is not suitable to be implemented on GPU because it processes one pair of sequences at a time. To the best of our knowledge, recent applications still use the original version of the Spring algorithm and proceed on a sample-by-sample basis.

Below we present our proposed approach, SpringNet, which assumes that it is beneficial for forecasting models to capture series shape information, especially when the repeatable fluctuations occur frequently. SpringNet is a Transformer architecture that attends to the shape of time series patterns directly by using the SpringNet attention algorithm. The SpringNet attention algorithm is based on the Spring subsequence matching algorithm but allows to process a batch of query-key sequence pairs concurrently and is thus suitable for GPU computation.

4.2 Model Architecture

We adopt the general Transformer architecture but replace the multi-head attention layer of Transformer or the convolutional attention of LogSparse Transformer with our Spring attention layer, as shown in Fig. 1. The SpringNet attention algorithm is illustrated in Algorithm 1, where L_{sub} is the query subsequence

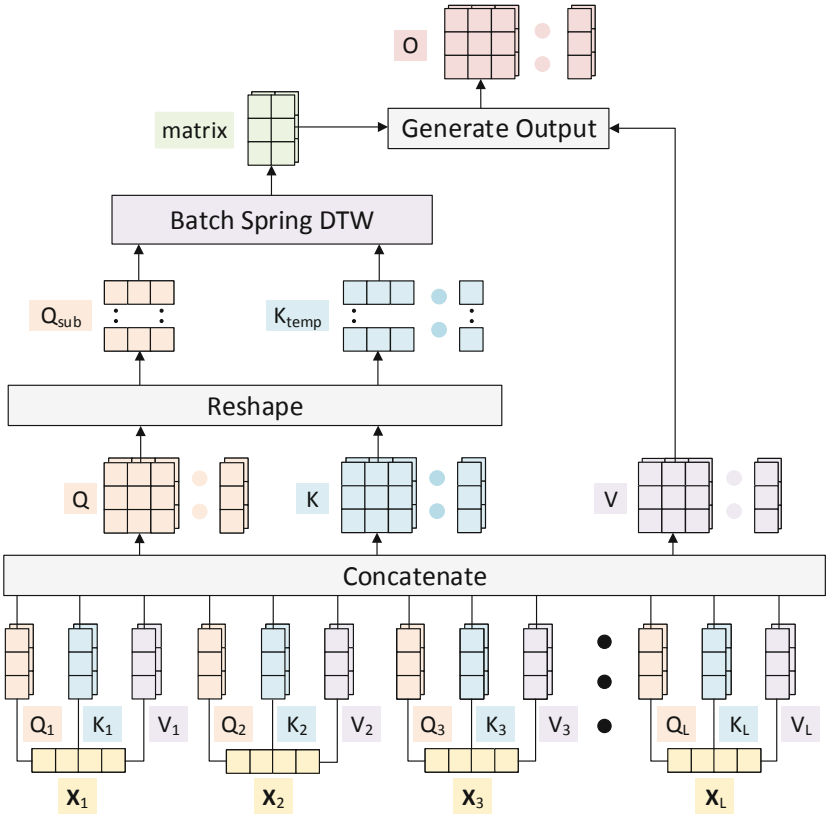

Fig. 2. SpringNet attention mechanism

length, n_{top} is the number of best-matched subsequences from the keys and $\mathcal{F}$ denotes the batch Spring DTW function as shown in Algorithm 2. Instead of mapping a piece of subsequence into a query and key like the convolutional attention, Spring attention transforms single time point into individual query and key and observes the shape of query and key series pattern. Spring attention identifies the subsequences of keys that match query series.

In lines 2 to 7 of Algorithm 1, we preprocess the queries and keys by extracting subsequences from queries, repeating keys and reshaping the tensors as the input of batch Spring DTW function $\mathcal{F}$. $Q[:,:,l:l+L_{sub},:]$ in line 3 indicates the extraction of a tensor from Q with all elements in the $1st$, $2nd$ and $4th$ axis and the elements from the lth to the $l+L_{sub}$ position in the $3rd$ axis of Q. $\mathcal{F}$ produces the *matrix* that stores the DTW distance D of n_{top} best-matched subsequences from the key series and their ending indexes. Then, values which indexes are stored in *matrix* are extracted as V_{sub}. Finally, we use DTW distance D and matched values V_{sub} to compute the Spring attention output O.

Whenever n_{top} is less than the number of keys, the Spring attention is sparse. The n_{top} controls the number of time steps that are attended to. Thus, the attention could be sparser and memory usage could be lower with smaller n_{top}. L_{sub} controls the Spring attention locality and SpringNet with short L_{sub} tends to capture short-term pattern.

Figure 2 shows the feedforward dataflow of the SpringNet attention mechanism. Similarly to the Transformer, SpringNet extracts queries, keys and values from individual inputs and concatenates them as tensors. Then, these queries and keys are reshaped to Q_{sub} and K_{temp} (see lines 2 to 7 of Algorithm 1) and passed to the Batch Spring DTW algorithm (function $\mathcal{F}$). Finally, the values and $matrix$ computed by the Batch Spring DTW algorithm are used to generate the attention output O (see lines 9 to 10 of Algorithm 1).

4.3 Batch Spring Attention

We follow the design of the Spring algorithm for subsequence mining but propose a batch version in Algorithm 2. The batch Spring attention algorithm achieves the same functionality as the Spring algorithm (see Sect. 3.3). The advantage of our Spring attention algorithm is the ability to process multiple query-key pairs concurrently on GPU, in order to speed up the Spring attention layer's feedforward speed.

We create multiple matrices and arrays to store temporary variables: 1) D_{prev} and D_{now} store the previous and current DTW distance, S_{prev} and S_{now} store the previous and current starting position of all samples; the four matrices come from the STWM; 2) Dis stores the DTW distance of matched subsequences, J_e stores the ending position of matched subsequences; both arrays are used to update the $matrix$ via function updateMatrix, which is the output of batch Spring DTW function. The function updateMatrix ensures $matrix$ only keeps the Dis and J_e of n_{top} best-matched subsequences.

Our Algorithm 2 has two nested loops starting at line 4 and 5 to identify all subsequences in parallel, while the original Spring DTW algorithm would have a third outer loop to iterate through all subsequence templates. In each Spring attention layer, there are $n_{batch} \times n_{head} \times L$ subsequence templates (Q_{sub}). In lines 5 to 15 of Algorithm 2, we compute the DTW distance and subsequence starting position of the subsequence point at the jth time step. The candidate subsequences are identified and saved in lines 16 to 21. Finally, we update the Dis, J_e and STWM to proceed to the next time step in lines 22 to 29.

5 Experimental Setup

All prediction models were implemented using PyTorch 1.5 and CUDA 10.1. For both data sets, we use the last year as test set, the second last year as validation set for hyperparameter tuning, and the remaining data (5 years for Sanyo and 4 years for Hanergy) as training set.

Algorithm 2: Batch Spring DTW algorithm

input : $Q_{sub} \in \Re^{N \times L_{sub} \times d_k}$, $K_{temp} \in \Re^{N \times L \times d_k}$, $n_{top} \in \mathbb{Z}^+$
output: $matrix \in \Re^{N \times n_{top} \times 2}$

init: $D_{prev}, D_{now}, S_{prev}, S_{now} \triangleq \Re^{N \times L_{sub}}$; $J_e, Dis, check \triangleq \Re^N$; $k \in \mathbb{Z}^+$; $matrix \triangleq \Re^{N \times n_{top} \times 2}$;
$D_{prev}[:,:], J_e[:], Dis[:], matrix[:,:,:] = \infty$;
$D_{now}[:,:], S_{prev}[:,:], S_{now}[:,:] = 0$;
for $j \leftarrow 1$ **to** L **do**
    `// Update subsequences DTW distance and starting position`
    for $i \leftarrow 1$ **to** L_{sub} **do**
        if $i == 1$ **then**
            $D_{now}[:,i] = ||K_{temp}[:,j,:] - Q_{sub}[:,i,:]||$;
            $S_{now}[:,i] = j$;
        else
            $D_{now}[:,i] = ||K_{temp}[:,j,:] - Q_{sub}[:,i,:]|| + \min(D_{now}[:,i-1], D_{prev}[:,i], D_{prev}[:,i-1], \text{axis} = 2)$;
            $Distance \leftarrow$ concatenate$(D_{now}[:,i-1], D_{prev}[:,i], D_{prev}[:,i-1])$ along the $3rd$ axis;
            $Start \leftarrow$ concatenate$(S_{now}[:,i-1], S_{prev}[:,i], s_{prev}[:,i-1])$ along the $3rd$ axis;
            $S_{now}[:,i] = Start[:,:,\text{argmin}(Distance, \text{axis} = 3)]$;
        end
    end
    `// Identify new matched subsequences`
    $check[:] = 0$;
    for $i \leftarrow 1$ **to** L_{sub} **do**
        $check[D_{now}[:,i] >= Dis[:] \cap S_{now}[:,i] > J_e[:]] += 1$;
    end
    $index = (check == L_{sub})$;
    `// Store the information of new matched subsequences`
    $matrix[index] = \text{updateMatrix}(matrix[index], Dis[index], J_e[index])$;
    `// Reset for the incomig time point`
    $Dis[index] = \infty$;
    $index \leftarrow$ duplicate $index$ along the $2nd$ dimension for L_{sub} times;
    $D_{now}[S_{now} <= J_e \cup index] = \infty$;
    $index = (D_{now}[:, L_{sub}] <= Dis[:])$;
    $Dis[index] = D_{now}[index, L_{sub}]$;
    $J_e[index] = j$;
    $D_{prev} = D_{now}$;
    $S_{prev} = S_{now}$;
end

We use three models for comparison: two state-of-the-art autoregressive deep learning models (DeepAR and LogSparse Transformer) and a persistence model. DeepAR [2] is a widely used sequence-to-sequence forecasting model, while the LogSparse Transformer [1] is a recently proposed variation of the Transformer

architecture for time series forecasting; we denote it as "Transformer" in Tables 1 and 2. The persistence baseline is a typical baseline in forecasting and considers the PV power output of the previous day as the prediction for the next day.

Table 1. Hyperparameters for all models

Model	δ	d_{hid}	n_{layer}	$d_k \& d_v$	n_{head}	L_{sub}	n_{batch}
DeepAR (Sanyo)	0	8	3	–	–	–	256
Transformer (Sanyo)	0.2	12	3	6	3	–	256
SpringNet (Sanyo)	0	24	2	6	3	3	256
DeepAR (Hanergy)	0.2	16	4	–	–	–	512
Transformer (Hanergy)	0.2	12	3	4	2	–	512
SpringNet (Hanergy)	0	12	2	4	2	3	512

All deep learning models are optimized by mini-batch gradient descent with the Adam optimizer, with variable learning rate (initial $\lambda = 0.005$, decay factor $= 0.5$) and maximum number of epochs 100. We used Bayesian optimization for hyperparameter search with a maximum number of iterations 20.

For all models, the dropout rate δ is chosen from $\{0, 0.1, 0.2\}$, the hidden layer dimension size d_{hid} and number of layers n_{layer} are chosen from $\{8, 12, 16, 24, 32\}$ and $\{2, 3, 4, 5\}$. For LogSparse Transformer and SpringNet, the query and value's dimension size $d_k \& d_v$ and number of heads n_{head} are chosen from $\{4, 6, 8, 12\}$ and $\{2, 3, 4, 6, 8\}$. For LogSparse Transformer, we use restart attention range of 20 and local attention range of 3. For SpringNet, the number of best-matched subsequences n_{top} is set to 5 and the subsequence length L_{sub} is chosen from $\{2, 3, 4\}$.

The selected best hyperparameters for all models are listed in Table 1 and used for the evaluation on the test set.

6 Results and Discussion

Table 2 shows the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) of all models for the two data sets. The best result for each metric and data set is highlighted in bold. On both data sets, all deep learning models outperform the baseline model. SpringNet is the most accurate model in terms of MAE on both data sets, followed by LogSparse Transformer and DeepAR. In terms of RMSE, SpringNet is the best performing model for Sanyo and the second best for Hanergy. Overall, SpringNet is the best performing model which shows the effectiveness of the proposed approach which attends to the shape of the time series patterns directly.

In addition, the Batch Spring DTW algorithm significantly speeds up the model training. On the same Tesla P100-16GB GPU, the feedforward process

Table 2. Accuracy of all models

	Sanyo		Hanergy	
Model	MAE	RMSE	MAE	RMSE
Persistence	0.522	0.985	0.703	1.174
DeepAR	0.276	0.381	0.370	0.505
Transformer	0.267	0.384	0.360	**0.478**
SpringNet	**0.258**	**0.380**	**0.358**	0.494

of a single Spring DTW layer (Algorithm 2) takes 2.081 s and 2.082 s per batch on Sanyo and Hanergy set respectively. In comparison, for the original Spring algorithm, the processing time is 12.057 h and 18.426 h per batch on the Sanyo and Hanergy data sets correspondingly. The running time of our Spring DTW layer is relatively stable with respect to the batch size, as long as it does not exceed the GPU memory constraint, while that of the original Spring algorithm without parallelism increases as the number of series pairs increases.

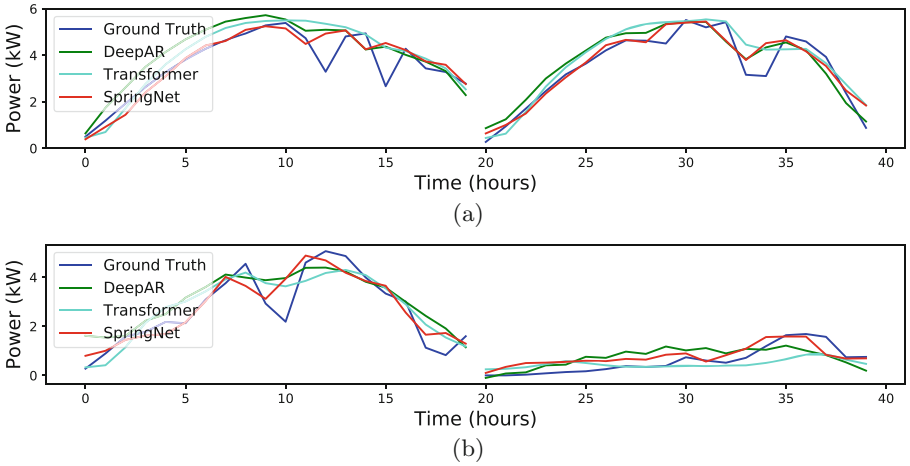

Fig. 3. Actual vs predicted data: (a) Sanyo dataset and (b) Hanergy dataset

Figure 3 plots the actual and predicted solar power data for two days, for both data sets. The actual data shows different levels of fluctuations, higher for Sanyo and lower for Hanergy. We can see that the predictions provided by SpringNet are the closest to the ground truth. SpringNet also forecasts the data fluctuations much better than DeepAR and LogSparse Transformer which tend to produce smooth curves. This shows that SpringNet is able to capture the time series pattern and deal well with repeated fluctuations.

Hence, based on the results, we conclude that SpringNet is a promising method for solar power forecasting - it outperforms all models used for comparison and is more robust to fluctuations. The attention layer in SpringNet helps to capture repeatable fluctuation patterns and provide accurate forecasts especially in the presence of fluctuations.

7 Conclusions

In this paper, we present SpringNet, a new deep learning based approach for time series forecasting, and demonstrate its performance in a case study for solar PV power forecasting. SpringNet is based on the LogSparse Transformer architecture but uses Spring DTW attention layers. We propose the use of Spring attention to overcome the weakness of LogSparse Transformer to effectively capture the time series shape information. In addition, we propose the Batch Spring DTW algorithm to speed up the feedforward operation of the Spring DTW attention algorithm. SpringNet is a generic time series forecasting approach and can be used in different domains. We present a case study for solar power forecasting, using two big data sets from solar plants located in Australia. The results show that SpringNet outperforms the state-of-the-art deep learning forecasting methods DeepAR and LogSparse Transformer, and also a persistent baseline used for comparison. Our experiments suggest that SpringNet can capture shape information from trajectories and make robust predictions. In summary, the results validate our hypothesis that attending to the shape of time series pattern directly is beneficial. We also found that the Batch Spring attention algorithm was significantly faster than the sequential version.

Hence, we conclude thatSpringNet with Batch Spring Attention mechanism is a promising method for time series forecasting.

References

1. Li, S., et al.: Enhancing the locality and breaking the memory bottleneck of Transformer on time series forecasting. In: Conference on Neural Information Processing Systems (NeurIPS) (2019)
2. Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T.: DeepAR: probabilistic forecasting with autoregressive recurrent networks. Int. J. Forecast. **36**, 1181–1191 (2020)
3. Vaswani, A., et al.: Attention is all you need. In: Conference on Neural Information Processing Systems (NeurIPS) (2017)
4. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoust. Speech Signal Process. **26**, 43–49 (1978)
5. Cuturi, M., Blondel, M.: Soft-DTW: a differentiable loss function for time-series. In: International Conference on Machine Learning (ICML) (2017)
6. Guen, V.L., Thome, N.: Shape and time distortion loss for training deep time series forecasting models. In: Conference on Neural Information Processing Systems (NeurIPS) (2019)

7. Cai, X., Xu, T., Yi, J., Huang, J., Rajasekaran, S.: DTWNet: a dynamic time warping network. In: Conference on Neural Information Processing Systems (NeurIPS) (2019)
8. Sakurai, Y., Faloutsos, C., Yamamuro, M.: Stream monitoring under the time warping distance. In: International Conference on Data Engineering (ICDE) (2007)
9. Pedro, H.T., Coimbra, C.F.: Assessment of forecasting techniques for solar power production with no exogenous inputs. Solar Energy **86**, 2017–2028 (2012)
10. Lin, Y., Koprinska, I., Rana, M., Troncoso, A.: Pattern sequence neural network for solar power forecasting. In: International Conference on Neural Information Processing (ICONIP) (2019)
11. Rana, M., Koprinska, I., Agelidis, V.G.: 2D-interval forecasts for solar power production. Solar Energy **122**, 191–203 (2015)
12. Azur, M., Stuart, E., Frangakis, C., Leaf, P.: Multiple imputation by chained equations: what is it and how does it work? Int. J. Methods Psychiatr. Res. **20**, 40–49 (2011)