LSTM-based PSS Design for Modern Power Systems

Khaled Aleikish \textsuperscript{a}, Thomas Øyvang \textsuperscript{b},

\textsuperscript{a,b} Department of Electrical engineering, Information Technology and Cybernetics, University of South-Eastern Norway, Norway,
Khaled.Aleikish@usn.no, Thomas.Oyvang@usn.no

Abstract

With the ever-increasing incorporation of wind and solar power in the electric power system, enhanced performance of classical bulk hydropower plants for robust operation of the power system is required. This current energy transition may cause a rapid increase in undesirable low-frequency oscillations (LFOs) in modern power system operations. A power system stabilizer (PSS) located at hydropower plants is one solution to damp such oscillations. This paper presents a new method based on Long Short-Term Memory (LSTM) neural networks for sine-wave phase shifting to possibly enhance PSS damping. The proposed controller considers the PSS input and the rotor speed deviation as a damped sinusoidal signal, simplifying PSS control and real-time optimization of PSSs parameters. Results show that the proposed LSTM architecture is able to learn multiple damped sine waves with different frequencies and decay rates. Therefore, the proposed controller can operate on the entire range of LFOs, unlike simple feedforward neural network (FNN) controllers, which can only learn and function on a single LFO frequency.

1. Introduction

The shift towards a more sustainable energy system demands increased stability properties from the hydropower fleet. Power system stabilizers (PSS), formerly known as supplementary excitation control systems, were developed to enhance the damping of low-frequency oscillations (LFOs) and increase power transfer limits [4, 5, 9–11]. In Norway, it is obligatory to install PSS on synchronous generators with a capacity of 30 MW and above (type D) [16]. The first PSSs were installed in the Nordic power system around 1970 to reduce power oscillations and increase power exchange on the interconnections between Norway and Sweden. PSS design and tuning were re-visited through the 1990s when the system loading and demand for power exchange increased, becoming one of the most cost-effective solutions for enhancing power system stability [12]. Conventionally, PSSs are only tuned and validated during the commissioning of the machine. These start-up settings of the PSS have the intention to dampen a wide range of low-frequency oscillations in the grid system during operation [18]. However, as the power system develops and expands with more intermittent energy sources such as wind and solar, new challenges are introduced to the operation of the power system. High-impedance weak grid systems and reductions in short-circuit power capabilities will transform the grid characteristics and may adversely affect the damping performance of the online PSS operation.

Over the past couple of decades, advancements in machine learning algorithms and computing power have enabled researchers to explore automatic calibration of PSS parameters to changing grid conditions [2, 3, 15]. Moreover, [6] proposed two methods to enhance the small-signal stability of a single-machine infinite bus (SMIB) system. Firstly, a particle swarm optimization (PSO) algorithm was used to determine optimal parameters for a conventional power system stabilizer (CPSS) [8]. CPSS is a simplified version of the PSS1A type PSS [1]. The PSO algorithm optimizes the CPSS parameters for a specific value of the external (Thévenin) impedance connecting the synchronous machine to the infinite bus. However, PSO algorithms are computationally expensive and potentially slow at finding solutions. Hence, a simple feedforward neural network (FNN) was used to map a range of external impedance values to the corresponding set of optimized parameters by the PSO algorithm. The end design is an auto-tuning system that automatically adjusts CPSS parameters in response to changes in the external network impedance.

Secondly, a model-free approach to PSS design was proposed in [6]. A simple FNN-based PSS, called the sine shifting neural network (SSNN) controller, was developed without relying on complex electrical machine theory. Unlike the first method, which augmented the CPSS with an auto-tuning system based on artificial neural networks, the SSNN controller completely replaces the CPSS. This paper proposes replacing the simple FNN architecture of the SSNN controller with a more complex neural network architecture to improve the stability performance of the PSS when subjected to a wide range of LFO in the electricity grid. Specifically, the proposed approach, which is called the Sine Shifting LSTM (SSLSTM) controller, uses a Long Short-Term Memory (LSTM) neural network, which is a type of recurrent neural network (RNN) architecture, to expand the operational range of the SSNN controller. While the SSNN controller can only function correctly over a single LFO frequency, the proposed SSLSTM controller can operate effectively over a wide range of LFO frequencies (0.1 to 2.5 Hz) with minimal performance impact. Moreover, a detailed discussion on best-practice for
picking training sets and training options is included. The paper is organized as follows. Section 2 briefly describes the excitation control system of synchronous generators and CPSS’s role in the control loop. Section 3 describes the SSNN controller, while section 4 describes LSTM networks. Section 5 describes the proposed controller. Simulation and results are presented in Section 6, and results are discussed in Section 7. Finally, conclusions and future work are given in Section 8.

2. Excitation systems

A typical excitation control system consists of an exciter automatic voltage regulator (AVR), and a PSS. The AVR regulates the generator terminal voltage by controlling the amount of current supplied to the generator field winding by the exciter, while the PSS is a feed-forward supplementary control device. The primary function of the PSS is to damp generator rotor oscillations (LFO’s) and enhance both steady-state stability and transient stability. A well-tuned excitation system provides benefits such as improved oscillation damping, relay coordination and enhanced first-swing transient stability. Fig. 1 shows a block diagram of a grid-connected synchronous generator’s excitation control system. In the figure, the PSS output \( V_{PSS} \) is an auxiliary control signal passed to the AVR. The AVR input can be expressed as:

\[
\Delta V_{\Sigma} = \Delta V + V_{PSS}.
\]  

Here, the voltage error \( \Delta V \) is expressed as:

\[
\Delta V = V_{ref} - V_f.
\]  

Fig. 2 shows a block diagram of a CPSS. In this figure, the CPSS is made up of four parts: (a) amplifier gain, (b) a signal washout high-pass filter, (c) lead elements for phase compensation, and (d) a limiter. In addition, some CPSS also include a signal sensor and a low-pass filter stage, which is not shown in the figure. A common signal used as input in the CPSS is the speed deviation \( \omega \). In most studies the amplifier gain \( K_{PSS} \) and the time constants of the phase compensation stage are typically tuned to damp the LFOs. The other parts of the CPSS ensure that it functions as intended and does not disrupt the AVR action.

Classical tuning and performance evaluation of the PSS are typically done through phases compensation, root locus, and time domain analysis [9–11]. In phase compensation, which is the most widely used approach, the stabilizer is tuned to compensate for the phase lags through the generator, the excitation system, and the power system in such a way that torque changes are in phase with speed changes.

3. Neural Network PSS design

In [6] a Sine Shifting Neural Network (SSNN) Controller was developed. This approach has been evaluated in this paper. The SSNN approach is based on the assumption that the rotor speed deviation \( \Delta \omega \) can be considered a damped sinusoidal signal. However, the neural network used to build the controller was trained on a sinusoidal signal without taking damping into account. That is, the controller input, the speed deviation \( \Delta \omega \) was expressed as:

\[
\Delta \omega \approx A_\omega \sin(\omega_t t).
\]  

Here, \( A_\omega \) and \( \omega_t \) are the amplitude and the frequency of the sinusoidal signal, respectively. Also, the controller output was expressed as an identical, phase-shifted sine wave

\[
\Delta \omega \approx A_\omega \sin(\omega_t t + \beta).
\]  

where \( \beta \) is a control variable that represents the desired phase shift. Ideally, the controller would require only two inputs: the speed deviation \( \Delta \omega \) and the desired phase shift \( \beta \). However, since the SSNN controller was designed using FNN, which is a simple neural network architecture with no internal memory, additional inputs were required. The additional inputs are the speed deviation values at the two previous time steps: \( \Delta \omega_{t-1} \) and \( \Delta \omega_{t-2} \). Algorithm 1 shows the pseudo-code for generating the SSNN training data. In essence, the

**Algorithm 1** Pseudo-code to generate the training data set for the SSNN controller [6]. An FNN-based SSNN requires four inputs.

1: for Every amplitude \( A_j \) in \( A \)’s range do
2: for Every phase shift \( \beta_j \) in \( \beta \)’s range do
3: for each time step \( t_k \) in one period do
4: SSNN input \( 1 = \beta_j \)
5: SSNN input \( 2 = A_j \sin(\omega t_k) \)
6: SSNN input \( 3 = A_j \sin(\omega t_{k-1}) \)
7: SSNN input \( 4 = A_j \sin(\omega t_{k-2}) \)
8: SSNN output \( = A_j \sin(\omega t_{k} + \beta_j) \)
9: end for
10: end for
11: end for

additional inputs serve as external memory cells to the SSNN. However, information stored in these memory cells is lost as new data is measured, which limits the functionality of the SSNN. Furthermore, the performance assessment in [6] showed that under the FNN architecture, the SSNN performs well only at the frequency at which it was trained. In addition, a slight attenuation or gain was observed in the amplitude when \( \beta \not= 0 \). The results indicate that a controller based on FNN can only output the correct amplitude for undamped sine waves. This is evident from the amplitude of the oscillations.
in the results; until the amplitude decays, the controller outputs the correct amplitude. For damped sine waves, the controller outputs the correct amplitude only when $\beta = 0$. Moreover, according to [6], it is practically impossible to train an FNN to differentiate between sine waves of different frequencies; training SSNN for more than one frequency under the FNN architecture results in a network that performs as if it was trained on the average of all the frequencies. Consequently, the results in [6] show the controller’s performance deteriorates rapidly at all frequencies except for the one for which it was trained. To correct the amplitude and the phase drift, [6] describes several approaches. One approach assumes that the frequency of oscillation is known at the time of the disturbance and proposes training several SSNNs at different frequencies and enabling the one that was trained for the current frequency of the oscillation. However, this approach of multiple SSNNs is valid only if the oscillation frequency can be determined in real-time and relatively fast to avoid degrading the PSS’s initial performance.

4. Long Short-Term Memory network

For a better understanding on the underlying architecture in the proposed method this section describes the Long Short-Term Memory (LSTM) network. LSTM is an advanced type of RNN that is capable of learning long-term dependencies between time steps of time-series data or any other type of sequential data [7]. Fig. 3 shows a block diagram of an LSTM cell (left) and LSTM layer (right). A single LSTM layer can contain $N$ LSTM cells, where $N$ depends on the length of the longest sequence of interest. In addition, it is also possible to stack several LSTM layers in a single neural network architecture to create deeper LSTM networks. Furthermore, each LSTM cell consists of three gates: (a) a Forget gate, which controls what information should be discarded from the old cell state $c_{t-1}$, (b) an Update gate, which controls the flow of new information into the new cell state $c_t$, and (c) an Output gate, which controls the value of the next hidden state $h_t$ (also called the output state). At time step $t$, the cell state $c_t$ and the hidden state $h_t$ are expressed as:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \sigma_{\text{tanh}}(c_t).$$

Here, the symbol $\odot$ denotes the Hadamard product (element-wise multiplication). Moreover, the functions $f_t$, $i_t$ and $o_t$ are given by:

$$f_t = \sigma_{\text{sigmoid}}(W_fx_t + R_fh_{t-1} + b_f)$$

$$g_t = \sigma_{\text{tanh}}(W_pg_t + R_p h_{t-1} + b_p)$$

$$i_t = \sigma_{\text{sigmoid}}(W_ix_t + R_i h_{t-1} + b_i)$$

$$o_t = \sigma_{\text{sigmoid}}(W_ox_t + R_o h_{t-1} + b_o).$$

Here, the matrices $W_f, R_f$ and $b$ represents the learnable parameters (weights) of the LSTM cell. To view and analyze the learnable parameters in MATLAB, the neural network can be imported to the Deep Network Designer app, which can analyze the network parameters. For example, Tab. 1 shows the analysis results of an LSTM network with 128 hidden units using MATLAB’s Deep Network Designer application. The total learnable parameters in the network is $67.2 \times 10^3$.

```
<table>
<thead>
<tr>
<th>Type</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Sequence input</td>
<td></td>
</tr>
<tr>
<td>2. LSTM</td>
<td>Weights: $1 \times 128$</td>
</tr>
<tr>
<td>3. Fully Connected</td>
<td>RecurrentWeights: $512 \times 2$</td>
</tr>
<tr>
<td>4. Regression Output</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 3: An LSTM cell (left) and LSTM layer (right). In MATLAB, the default activation function for $f_t$, $i_t$ and $o_t$ is the sigmoid function (represented by green lines), while the hyperbolic tangent function (tanh) is used for $g_t$ [13, 17].
5. Sine Shifting LSTM Controller

In this work, a Sine Shifting controller based on LSTM networks is developed to phase shift the speed deviation $\Delta \omega$ to some optimal value. In the CPSS structure shown in Fig. 2 the required phase shift is obtained using two lead/lag stages. The time constants of these stages are tuned to produce a control signal that induces positive damping in the synchronous machine. Moreover, the proposed SSLSTM controller like the SSNN controller uses a single control parameter, $\beta$, to obtain the required phase shift. However, in contrast to the SSNN controller which can only function properly on a single LFO frequency, SSLSTM can function on the entire range of LFO frequencies.

The ability of LSTMs to predict discrete sine functions was studied by [14]. However, in this work it is focused on phase-shifting rather than forecasting future values. The objective is to develop a simple controller with only two control parameters, $K_{\text{PSS}}$ to control the gain, and $\beta$ to control the phase. It should be pointed out that the optimal values of $K_{\text{PSS}}$ and $\beta$ are outside the scope of this work and left for future research.

5.1. Generating the training data sets

Training, validation, and testing data sets are required to develop the machine learning model. In this study, different techniques were evaluated to generate the data sets: (a) an Expanding Window, (b) a Sliding Window, (c) a Sliding Data, and (d) An Expanding-Sliding Window. Depending on the technique, training time and model performance may be adversely affected. Moreover, in these techniques each predictor has a dimension of $2 \times W$, where $W$ is the window width (A sequence contains at least two points, therefore the minimum value of $W$ is 2). However, the window length is fixed to 2 (the number of features). Unlike the SSNN (Eq. 3 and Eq. 4), the SSLSTM features $e^{-\lambda t} A \sin(\omega t)$ and $\beta$ include the Decay constant $\lambda$. Also, the targets, $e^{-\lambda t} A \sin(\omega t + \beta)$, have a fixed dimension of $1 \times 1$.

Fig. 4 shows the Expanding Window method. In this method, the data sets consist of sequences of varying width. This method guarantees that targets are generated for all time steps $\{t_1, t_2, \ldots, t_{\text{end}}\}$. However, this method will add unnecessary information to all sequences beyond $S_{\text{PSS}}$. This implies that any target generated after the first period will have a predictor pair that contains unnecessary data points since one sine wave period contains all the necessary information to learn $\lambda$ and $\omega$. Hence, a better solution is to discard these redundant data points to save memory space, reduce training time, and enhance training performance.

In Fig. 5 the Sliding Window method is presented. In this method, the data sets consist of sequences of fixed width. This method guarantees that enough information is contained in all predictors when the window width is greater or equal to $N$. However, if the window width is greater or equal to $N$, then no targets are generated for time steps that comes before $N$. Moreover, Fig. 6 shows a comparison between the data sets generated by the Expanding Window method and the Sliding Window method. In the figure, no targets were created for $t_1$ and $t_2$ in the Sliding Window method. To create targets for $t_3$ and $t_4$, either the window width can be reduced from 4 to 2, or the time steps prior to $t_0$ can be filled with zeros (pre-padding the data array with zeros).

Fig. 7 shows the Sliding Data method, which is a slight modification to the Sliding Window method. In this method, as in the Sliding Window method, the data sets consist of sequences of fixed width. However, the data points slide into the window. This slight modification guarantees that targets are generated for all time steps, as in the Expanding Window method. However, this method pre-pads the window/predictors with zeros, which results in an inefficient memory allocation for all targets with predictors length less than the window width. This is illustrated in Fig. 7 for the three sequences $\{S_1, S_2, S_3\}$.

Fig. 8 shows the Expanding-Sliding Window method. This method combines the Expanding Window and Sliding Window methods to ensure memory-efficient generation of targets for all time steps. In this study, this approach proved to be the most effective method for generating the data sets and was selected to develop the SSLSTM PSS. Also, it is possible to add redundant data for the first quarter of the first period using the Expanding Window method to speed up and improve model training for early predictions. To illustrate this, Fig. 9 shows a histogram of a training data set generated by the Expanding-Sliding Window method for 2.2, 3.2, and 4.2 Hz sine waves. In addition, the figure illustrates how the Expanding Window method can be used to add redundant data to the training data set. However, it is advised not to add too much redundant data or the model can overfit.
Figure 6: A comparison between data sets generated by the Expanding Window and the Sliding Window methods. Each sample consists of two features and W data points. Where W is the window width.

Figure 7: (c) The Sliding Window method. In this method, the data points slide into a window with fixed width. Although all sequences have the same number of data points, some might contain zeros (pre-padding the sequences with zeros before training).

5.2. Neural network architecture of SSLSTM
The proposed neural network architecture of SSLSTM is shown in Fig. 10 using Deep Network Designer in MATLAB. The network layers were described in Tab. 1. In addition, zscore normalization option was used in the sequenceInputLayer layer. Also, the Output mode was set to 'last' in the LSTM layer to perform sequence-to-one regression.

5.3. Training options
This section describes MATLAB’s training options used in this work. Moreover, Tab. 3 lists the training options used in this study. Apart from the training options shown in the table, no other changes were made to the default training options. Throughout this study, the Adam optimizer was used for training the network. In MATLAB, except for solverName and Plots, all other options in the table are considered optional arguments. These arguments can be categorized based on their function into eight groups:

(a) Plots and Display options, (b) Mini-Batch options, (c) Validation options, (d) Solver options, (e) Gradient Clipping options, (f) Sequence options, (g) Hardware options, and (h) Checkpoints options.

Table 3: Training Options.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 solverName</td>
<td>adam</td>
</tr>
<tr>
<td>2 Verbose</td>
<td>0</td>
</tr>
<tr>
<td>3 Plots</td>
<td>training-progress</td>
</tr>
<tr>
<td>4 MaxEpochs</td>
<td>100</td>
</tr>
<tr>
<td>5 MiniBatchSize</td>
<td>2048</td>
</tr>
<tr>
<td>6 Shuffle</td>
<td>every-epoch</td>
</tr>
<tr>
<td>7 InitialLearnRate</td>
<td>0.01</td>
</tr>
<tr>
<td>8 LearnRateSchedule</td>
<td>piecewise</td>
</tr>
<tr>
<td>9 LearnRateDropPeriod</td>
<td>1</td>
</tr>
<tr>
<td>10 LearnRateDropFactor</td>
<td>0.95</td>
</tr>
<tr>
<td>11 GradientThreshold</td>
<td>1</td>
</tr>
<tr>
<td>12 SequenceLength</td>
<td>longest</td>
</tr>
<tr>
<td>13 SequencePaddingDirection</td>
<td>left</td>
</tr>
</tbody>
</table>

First, from the Plots and Display options, the option Plots was set to "training-progress" to visualize the training progress since it is easier to monitor the training progress by the accuracy and loss plots of the validation and training sets. The downside to this is that after training the network for a long time, it can become difficult to tell if the metrics are improving or not since the plots do not automatically scale to the last few training iterations. In this case, it is best to use Verbose to monitor the training progress, which displays the training progress metrics in the command window.

Secondly, under the Mini-Batch options, the number of Epochs was fixed to a relatively large value when comparing models with different hyperparameters. In this work, the value of MaxEpochs typically range from 50 to 500, ensuring that the model does not get stuck in a local minimum. Moreover, the value of MiniBatchSize is a trade-off between training speed and accuracy. In this work, the max value of MiniBatchSize was limited by the GPU memory, while the minimum value was limited by the available training time. Setting MiniBatchSize
Figure 9: A histogram of a training data set generated by the Expanding-Sliding Window method for 2.2, 3.2, and 4.2 Hz. The redundant data generated by the Expanding window method does not contain any new information, but it helps improve model training speed on early targets by exposing the model to the data more than once during one training epoch.

Figure 10: The proposed architecture of SSLSTM [13]. The layers are described in Tab. 1.

to a small value increases the training time and the regularization effect of mini-batches. Thus, the model generalizes better, resulting in a lower validation’s Root Mean Square Error (RMSE). In addition, it was observed that shuffling the training data after each epoch helps to reduce the validation RMSE. Thus, the option Shuffle was set to "every-epoch".

Thirdly, for the Solver options the InitialLearnRate, LearnRateDropPeriod, and LearnRateDropFactor were set with the options MaxEpochs and taken Shuffle into account to give the best training performance. The strategy adopted was to decrease the learning rate for every few epochs as training data is shuffled. Also, for Gradient Clipping options, GradientThreshold is set to 1 instead of “inf” (default) to improve the training stability at high learning rate. This helps prevent gradient explosions and speeds up the training process.

Finally, from the Sequence options, SequencePaddingDirection was set to "left". This is because any padding done in the final time steps of the sequence can negatively impact the training process in a Sequence-to-one regression. For SequenceLength, the value depends on the method used for generating the data. When the Expanding Window method was used, the sequence length was set to “longest”. In this case, the sequences in each mini-batch are padded to the length of the mini-batch’s longest sequence. However, when the Sliding Window method is used, the sequence length is set to the length of the longest possible sequence; it was set to the length of one period of the lowest frequency of interest. For example, if the lowest frequency considered is 0.1 Hz, and the step size is $10^{-3}$, then SequenceLength is set to $10^3$.

6. Results and validation
In order to demonstrate SSLSTM’s ability to learn multiple frequencies and decay rates, four simulation tests were conducted for $\beta \in \{0^\circ, 45^\circ, 90^\circ, 180^\circ \}$. In these tests, the SSLSTM was trained on the data range in Tab. 4 with a step size of $0.5 \times 10^{-2}$, Fig. 11, 12, 13, and 14 show the results of these tests. In the figures, SSLSTM’s predictions match the ground truth (the ideal expected output) over a wide range of $\beta$, while SSNN’s predictions only match when $\beta = 0^\circ$.

Table 4: Data resolution and range. $\omega$: frequency, $\lambda$: decay rate, $A$: sine amplitude, $\beta$: phase-shift.

<table>
<thead>
<tr>
<th>Lower bound</th>
<th>Upper bound</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega$</td>
<td>2.2 Hz</td>
<td>4.2 Hz</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$A$</td>
<td>$1 \times 10^{-5}$</td>
<td>0.1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$0^\circ$</td>
<td>$180^\circ$</td>
</tr>
</tbody>
</table>

Figure 11: SSLSTM performance with $\beta = 0^\circ$.

The equations describing the dynamics of a synchronous generator during transient operation are quite stiff e.g., fast changing differential equations. Hence, often small step-sizes of Ode solvers are desired. In [6] the solver (ode23tb variable-step) step size was set to $3 \times 10^{-3}$. In this work, it was not possible to generate training data with a step size of $3 \times 10^{-3}$ due to limited hardware resources. To validate and compare the performance of SSLSTM and SSNN, both models were trained with a step size of $10^{-3}$. Fig. 15 shows the simulation results for SSLSTM and SSNN when a disturbance occurs after 4s of simulation time.
7. Discussion
In this work, the neural network architecture and the training options were tuned to improve the model performance. These are summarized in Tab. 1, Tab. 2, and Tab. 3. Nevertheless, training LSTM models is computationally expensive compared to FNN models since the predictors of LSTM models are sequences. In this work, to train and tune the models in a reasonable amount of time, the training data was generated with the smallest possible step size and limited to the range of interest. Although it is possible to reduce the amount of data generated by increasing the step size to $10^{-2}$, and instead train the model on more frequencies and decay rates, it would decrease the controller performance on LFOs in the upper range. For example, a control action each 0.01s maybe sufficient to damp 0.1 Hz LFOs, but it might not be sufficient to damp 3 Hz LFOs effectively. A step size of $10^{-3}$ achieves the best possible model accuracy and generalization capability in the range of interest while keeping model training and tuning time to a minimum. Also, besides the data resolution and step size, it is also possible to generate more observations by generating more periods per sine wave. When generating targets for the current period, it makes sense to discard data points from the previous period to save memory space. Thus, the Sliding Window method is preferred to generate predictor sequences for targets after the second period. It is also possible to increase the sliding window width to include more information in each predictor. While this would increase the model accuracy at the cost of increased memory usage, it will not improve the model generalization capability. Thus, this approach was not preferred in this work. It is also possible to improve the model accuracy at the cost of training time by increasing the number of epochs and reducing the mini-batch size. In Fig. 11, 12, 13, and 14, SSLSTM’s ability to learn multiple frequencies and decay rate was demonstrated. Moreover, In Fig. 15 SSLSTM and SSNN
results show promising results and the main findings are:

In this paper, LSTM networks were used to develop a new sine-wave phase shifter for stability enhancements of electric power systems through the PSS. Simulation results show promising results and the main findings are:

- LSTM networks are capable of learning and tracking sine waves with multiple frequencies and decay rates.
- SSLSTM outperforms the SSNN controller at all frequencies except for the one SSNN was trained to phase shift.
- Training LSTM networks to learn periodic signals with a wide range of frequencies entails selecting the smallest step size to sample the highest frequency of interest while avoiding increasing the computational load significantly during model training. Training LSTM to learn long sequences, such as a 0.1 Hz LFO with a high sample rate, requires significant computing power.
- Carefully selecting the best method to generate the training data set can significantly improve the model performance.

Suggestions for future work:

- Develop an online adaptive PSS by combining SSLSTM and auto-tuning algorithms to adjust $\beta$ and $K_{PSS}$ during operation.
- Combine LSTM with other neural networks to improve prediction accuracy and reduce the amount of data required to train the LSTM network.

References


