**Report on Dataset: Appliances Energy Prediction**The Appliances Energy Prediction Dataset is a time-series dataset designed to predict energy consumption in a low-energy building. It contains experimental data collected from a residential building where various environmental conditions and appliance usage were recorded over time. The dataset is ideal for regression tasks and modeling energy consumption patterns based on environmental and operational variables.

This dataset consists of 19,735 rows (data points) and 29 columns (features). Each row is one observation, recorded every 10 minutes between January 11, 2016, and May 27, 2016-a total of about 4.5 months. These features are both target variables and predictors; thus, this data set will be appropriate for multivariate regression and time-series analysis.

**Dataset Details**

* **Target Variables:** One major target variable is the Appliances, which signifies energy consumption in watt-hours. Also, the lights column is used as a secondary target; this column represents the energy utilized by the building for lighting.
* **Environmental Variables:** Temperature recordings range from T1 to T9 and are accompanied by relative humidity from RH\_1 to RH\_9 for different spots within the building. This also includes outside temperature - T\_out, humidity (RH\_out), wind speed, and visibility.
* **Other Features:** The data also includes columns such as: Press\_mm\_hg pressure in mmHg; Tdewpoint- dew point temperature; and two random variables, rv1 and rv2 added to introduce some noise into the model.

**Characteristics of Data**

* **Data Type:** The data is mostly numeric, and it contains most of the continuous columns. The date column was in a timestamp format that has been used to index the data as time-series data.
* **Structure:** Tabular, wherein each row represents an observation at some point in time, therefore making it suited for a regression modeling or machine learning use case.
* **Volume:** The dataset contains 19,735 rows,  
  Columns (features): 29

**Dataset Source and Accessibility**

The dataset is hosted on the **UCI Machine Learning Repository** and can be accessed through the following links:

* **Dataset Description and Metadata:** [Appliances Energy Prediction Dataset](https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction)
* **Direct Download Link:** <https://www.kaggle.com/datasets/sohommajumder21/appliances-energy-prediction-data-set>

The dataset was contributed by Luis Candanedo from the University of Mons, Belgium, and it is made available under the Open Database License (ODbL) v1.0. Under this license, any use, sharing, and modification are freely permitted with attribution to the author.  
  
**Application and Context**This dataset was developed for modeling and predicting energy consumption patterns using regression models. Due to its multivariate nature, it may serve to explore environmental and operational variables that would influence the use of appliances on energy consumption. It can be useful in low-energy building energy efficiency analysis and predictive model design by researchers and data scientists.

**3. PART II: RNN: Simple RNN with Sine Wave Data**

Dense (1 neuron,4,289 parameters)

SimpleRNN (64 neurons, 4,289 parameters)

Input (50-time steps  
× 1 feature)

Output (Nextsine value)

**Report on the results of training and evaluating the model**

**Design of the Neural Network**The neural network architecture was focused on modeling of sequential data, predicting the next value in the sine wave. Architecture is composed of two layers:  
  
**SimpleRNN Layer:**

* As shown above, the configuration includes 64 neurons since this layer is going to capture the temporal dependencies or memory in the input sequence, and the input shape includes 50-time steps and one feature.
* The SimpleRNN layer outputs a vector of size 64 that encapsulates the learned representations from the input data.

**Dense Layer:**

* A fully connected layer with one output neuron is added after the SimpleRNN layer. This neuron produces the final prediction, which corresponds to the next value in the sine wave.
* The model has a total of 4,289 parameters, all of which are trainable.

**Results of Training and Evaluating the Model**

**Training Results**  
Lastly, the Adam optimizer with Mean Squared Error as the loss was used to train the neural network for five epochs. The fact that the losses were steadily declining in value demonstrated that the model can, in fact, identify the underlying pattern in sine wave data. The value of the training loss is as follows:

* Epoch 1: Loss = 0.0032
* Epoch 2: Loss = 0.000315
* Epoch 3: Loss = 0.000162
* Epoch 4: Loss = 0.000236
* Epoch 5: Loss = 0.000063

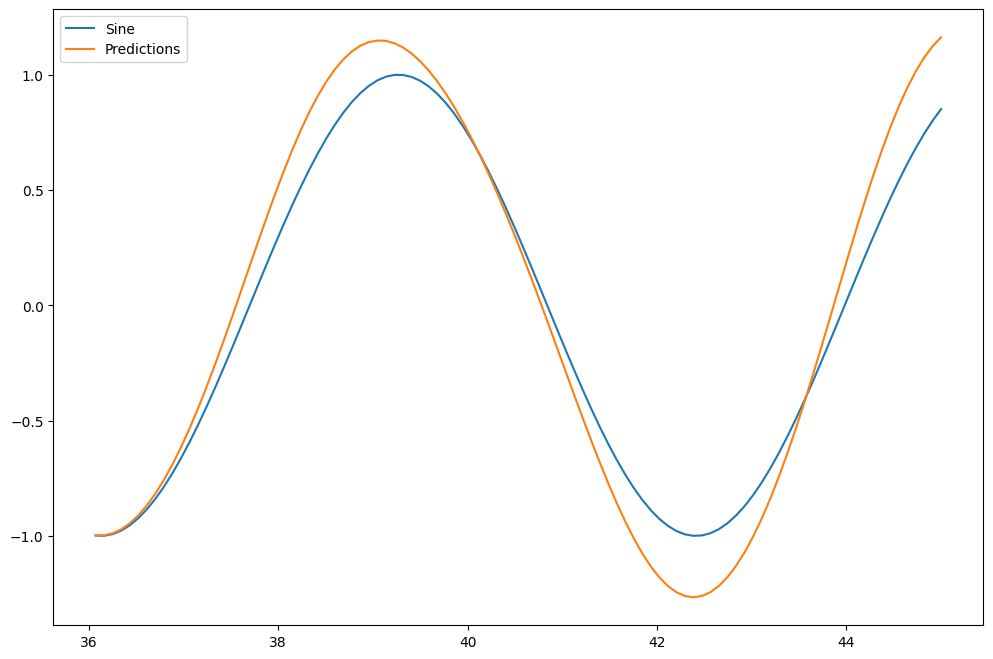
As the loss continues from the initial stages, this model learned the dynamics of the true sine wave quite well.  
  
**Evaluation on Test Data**  
Then, the model is tested on its ability to generalize with data it has never seen or 20% of the dataset. The actual values of the sine wave values were plotted against the projected values. It was found that the model had a low error and was very similar to the test sine wave.

**Visualization of Training and Testing**  
The model is improving consistently by training and without overfitting. From the loss curve shown below, the training epochs significantly decreased in error. Sine Wave Prediction vs. Actual The synthetic sine wave is close to the actual sine wave curve. There is slight deviation here and there but in general, close in trend and shape.

A graph with a line

Description automatically generated

**Comparison Graph: Predicted and actual sine waves**

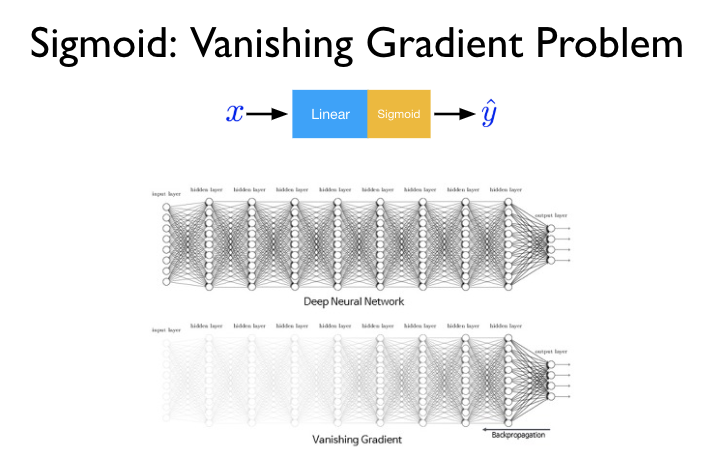
  
  
**Key Observations**  
There was no indication of exploding/fading gradients and an otherwise fluid training process where the model predicted values closest to actual values in a smooth and accurate portrayal of the temporal shapes for the sine wave.  
Slight deviations from the actual value during predictions here suggest an allowance or room for better performance had one tried harder-possibly with an even more elaborate structure (for instance, LSTMs), or higher capacity.

**Conclusion**  
The performance was quite good both for the training and test sets by the trained SimpleRNN model. This reflects the performance of the SimpleRNN on simple sequence modeling tasks, as it can easily predict the next value of the sine wave. More complicated RNN architectures, such as LSTMs or GRUs, may perform even better on more complex or noisy data.

**4. PART III: RNN: LSTM Neural Network**

**Question 3.1:  
--) Explain the vanishing gradient problem.**

**Answer: Vanishing gradient problem**One problem of deep neural networks involves extremely small gradients that result from backpropagation. The tiny gradients imply very negligible weight updates, resulting in considerable slowing down or even a complete halt of the learning process. The issue especially plagued network models containing activation functions in the form of sigmoid and tanh-that is, compressing input signals onto relatively small output ranges. While the input of most of the activation functions saturates for large inputs, it causes gradients to approach zero. In very deep networks, gradients are multiplied layer by layer during back propagation, which leads to exponential decay while going backwards into the network.



This problem prohibits the deeper layers from getting effectively trained since they will not get enough updates to learn complex patterns, hence degrading the performance of the network. The solutions to mitigate the problem involve using an activation function like ReLU, which does not saturate positive inputs, gradient clipping in order to retain extreme values of gradients, gated architecture such as LSTMs or GRUs that allow gradients to flow without vanishing, and residual networks, which make use of skip connections so that gradients are directly passed across the layers. These improve gradient flow and allow for the effective learning of deep networks.

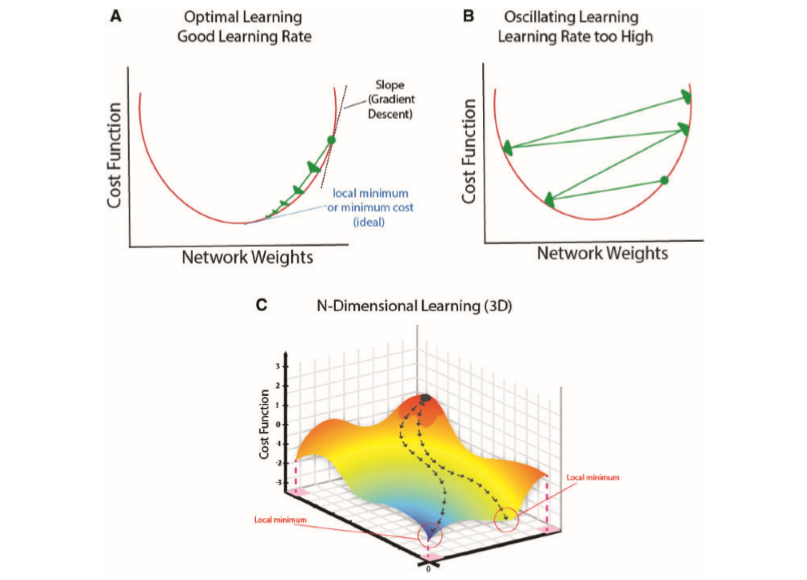
**Sources:** [1] Improved Vanishing Gradient Problem for Deep Multi-layer Neural Networks, <https://www.semanticscholar.org/paper/0cca39396e20e4239e9fa9f2c57e433fe6df841c>

[2] Overcoming The Vanishing Gradient Problem Of Recurrent Neural Networks In The Iso 9001 Quality Management Audit Reports Classification, <https://www.semanticscholar.org/paper/82527df154df06093f63b868c831ac9c2677e736>

[3] Volume-preserving Neural Networks: A Solution to the Vanishing Gradient Problem, <https://www.semanticscholar.org/paper/ccf9217edc4d3ecf565931d9a5a7eacb7fec0ba4>

[4] Quantifying the Vanishing Gradient and Long Distance Dependency Problem in Recursive Neural Networks and Recursive LSTMs, <https://www.semanticscholar.org/paper/80c05cae76e05fab33ff5622157731d0a5549723>

**Explain the exploding gradient problem.**

**Answer:** The exploding gradient problem is a major concern when training deep neural networks in which the gradients explode during backpropagation. This leads to instability in training, erratic updates of weights, oscillating or diverging loss, and makes it difficult to converge to an optimum. This problem is most highlighted in deep networks, where the gradients grow exponentially while moving backwards due, usually, to poor weight initialization, overly complex architectures, or improper activation functions.  


**Several techniques are put into place to avoid this issue:** This problem is most highlighted in deep networks, where the gradients grow exponentially while moving backwards due, usually, to poor weight initialization, overly complex architectures, or improper activation functions.  
**Several techniques are put into place to avoid this issue:**

1. Setting a threshold on the gradient is commonly known as **Gradient Clipping**. It decreases when the gradients exceed the limit to keep the training stable as there are a number of reasons for huge updates to occur.
2. It can include skipping connections in the same as ResNets in order to obtain smoother gradient flows that alleviate the explosion problem.
3. **Weight Normalization:** It keeps the magnitudes of gradients within a tolerable range to prevent the weight from blowing up during training.

These are strategies for maintaining stability while guaranteeing effective training, hence giving deep networks the ability to learn complicated patterns without diverging.

**Sources:** [1] Regularization for convolutional kernel tensors to avoid unstable gradient problem in convolutional neural networks, <https://www.semanticscholar.org/paper/84085c500da5b89308750b6be04f4e8254cbb08b>

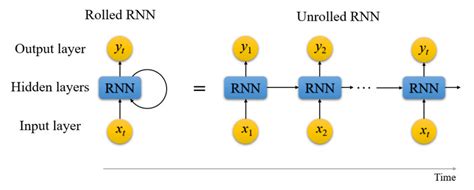
[2] Volume-preserving Neural Networks: A Solution to the Vanishing Gradient Problem, <https://www.semanticscholar.org/paper/ccf9217edc4d3ecf565931d9a5a7eacb7fec0ba4>

[3] The effect of the forget gate on bifurcation boundaries and dynamics in Recurrent Neural Networks and its implications for gradient-based optimization, <https://www.semanticscholar.org/paper/fab5cbbc0ec4976c16468cf6250f036c30e750d0>

[4] Deep Directly-Trained Spiking Neural Networks for Object Detection, <https://arxiv.org/abs/2307.11411>

[5] The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions, <https://www.semanticscholar.org/paper/54e9217e85f458af497ce2193f7dedb71b41e41b>

**Question 3.2:  
--) Discuss the limitations of the SimpleRNN neural network**  
The simple RNN was among the first designs to model sequences and temporal data. Despite their contribution to the RNN family, simple RNNs have efficacy in managing challenging jobs constrained by a number of issues.  
**Overview of SimpleRNN**  
Simple RNNs process sequences of inputs over time, keeping track of information from past time steps via a hidden state. The recurrence in their structure lets them learn temporal dependencies. However, their architecture inherently restricts their ability to perform well in many contexts.

  
  
**Limitations of Simple RNN**  
**1. Vanishing Gradient Problem**  
Vanishing gradient problem is the most critical limitation of Simple RNNs. This problem arises during training, especially in the course of backpropagation over many time steps. More precisely, gradients of the loss function may be very small, leading to negligible weight updates for the initial layers, while this prevents network learning of long sequences. Consequently, SimpleRNNs have difficulties in learning and remembering information from long input sequences, which is crucial for tasks such as language modeling and speech recognition where long-term dependencies are common.  
**2. Difficulty in Capturing Long-Term Dependencies**Simple RNNs have the vanishing gradient problem, which makes learning for long-range dependencies problematic. They capture relatively short-term dependencies quite well, but in longer sequences, they forget a lot. This seriously diminishes their potential for good performance in tasks requiring memory of previous states over longer inputs or extended time. Consequently, SimpleRNNs are usually insufficient for tasks such as machine translation or sentiment analysis, where context from many preceding words can change meaning.  
  
**3. Sensitivity to Sequences of Inputs**  
Simple RNNs are very sensitive to the scale and distribution of input data. If the range of input values is very large, it causes instability in training and problems with convergence. Therefore, steps of data preprocessing are important but add additional complexity to the preparation of the model. Moreover, they are prone to errors in the processing of noisy sequential data since their structure lacks elaborate mechanisms for filtering out irrelevant information.  
 **4. Poor Memory Usage**  
There is no mechanism of the effective memory management in the architecture of the SimpleRNNs. Each cell in SimpleRNN only keeps the hidden state currently being processed, making it hard to keep longer sequences of relevant information. That's in contrast to advanced models, such as Long Short-Term Memory or Gated Recurrent Units, which have certain gate mechanisms allowing the selective retention and discarding of information, hence increasing learning effectiveness.  
  
**5. Slow Training and Inference**  
SimpleRNNs may be slow to train because of their simple structure and recurrence, especially for longer sequences or larger datasets. Since the RNN is sequential, with a lot of computation to be done for each time step, training usually takes longer than in feedforward networks. Moreover, in the case of inference, speed is slow because one generates each output step by step instead of parallelizing as could be done in other architectures.

**Conclusion**

While SimpleRNNs were foundational in the development of sequence modeling techniques, they are plagued by a variety of inherent flaws, including problems with vanishing gradients, an inability to capture long-term dependencies, sensitivity to input data, poor memory utilization, and time-consuming training. These limitations led to the development of more sophisticated architectures such as LSTMs and GRUs, which are designed to overcome these challenges and do better on complicated temporal tasks. Being able to understand these limitations is critical in selecting appropriate models for particular applications in time series prediction, natural language processing, among other areas. Simple RNNs were among the first architectures for the representation of sequences and temporal data in a model. While it helped to contribute to the field of RNNs, SimpleRNN still has several shortcomings that make it less effective for complex tasks.

**Sources:**

1. Input Convex LSTM: A Convex Approach for Fast Lyapunov-Based Model Predictive Control, <https://www.semanticscholar.org/paper/21caf933b8d5238bffaaa9d20953e418f840b698>
2. Coal Structure Recognition Method Based on LSTM Neural Network, <https://www.semanticscholar.org/paper/f74095ab90232a063d823b1149eb1de4b9442de4>
3. Human emotion recognition based on multi-channel EEG signals using LSTM neural network, <https://www.semanticscholar.org/paper/f3fe6ff2023fd3d2e0496d7fef32377c6982855b>
4. Towards Efficient Recurrent Architectures: A Deep LSTM Neural Network Applied to Speech Enhancement and Recognition, <https://www.semanticscholar.org/paper/d858a70c21a7d1db0932bac43e20fcfdf00e5a87>

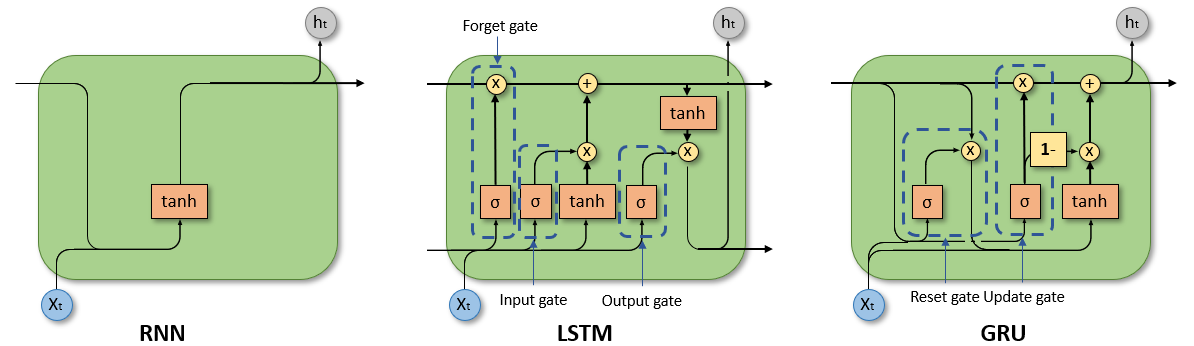
**Question 3.3:  
--) Explain how the LSTM neural network can provide powerful solutions to both gradient problems:  
(Vanishing and Exploding) and address the limitations of the SimpleRNN neural network.**

Long Short-Term Memory networks are a particular kind of RNN structured to handle two important problems: vanishing and exploding gradient problems. These turn out to be particularly problematic in the conventional Simple Recurrent Networks, which have a lesser capability for long-term dependencies and capturing sequences. The following describes in detail how LSTMs handle these gradient problems and thereby overcome the limitations of the so-called SimpleRNNs.

**Solutions to the Vanishing Gradient Problem**  
**1. Cell State Mechanism**  
LSTMs can keep gradients stable for longer sequences because the cell state, which is moving through the network, contacts other elements just a few times. In other words, the vanishing gradient problem, in which gradients continue to get smaller as products and backpropagation across hundreds of layers reduce their value in SimpleRNNs, is avoided in this architecture. This architecture design will also enable LSTMs to learn from sequences where the context is very far since information in the cell state is preserved without degradation.  
  
**2. Gating Mechanisms**  
In this regard, LSTMs consist of three critical gates responsible for controlling the flow of information. These gates allow the model to decide how much the previously available information shall contribute towards the updates of the current state; it means remembering or forgetting the selected details. This selective memory capability allows LSTMs to keep information longer, hence mitigating the problem of losing important context after some time-a common occurrence in SimpleRNNs.

* **Input Gate:** How much of the new information is added to the cell state.
* **Forget Gate:** How much information to forget in the cell state.
* **Output Gate:** The amount of cell state that gets passed to the next layer as output.

Putting this all together, these gates guarantee that gradients cannot vanish when they flow backwards through the network, allowing for effective long-term training across sequences.

  
  
**Solutions to the Exploding Gradient Problem**  
**1. Controlled Gradient Flow**  
LSTMs avoid the risk of exploding gradients through the use of their gating mechanism. That is, it allows the cell state to be updated in a proportionate manner rather than seeing huge increases in values when gradients are computed and applied. This, in turn, has a damping effect on large values if the gradient flow is very strong, hence allowing the process to remain stable during training.  
  
**2. Gradient Clipping**  
While this is not unique to the LSTMs, one of the major techniques one resorts to while training is gradient clipping. This would bound the magnitude of the gradients towards a threshold; if this threshold is exceeded, these gradients are scaled down, hence effectively dealing with the issue of exploding gradients and ensuring instability in model updates.

**Addressing Limitations of SimpleRNNs**  
**1. Ability of Long-Term Dependencies**

* One of the main deficiencies of SimpleRNNs is that they cannot memorize long-range dependencies. LSTMs overcome this by maintaining a sort of persisting memory through their cell state, hence connecting information from inputs that are widely spaced in time effectively.

**2. Better Memory Management**

* These many gates make LSTMs far more adept at memory management compared to SimpleRNNs since they can remember pertinent information and forget irrelevant information. This enhances the ability of SimpleRNNs to predict outputs from input sequences and remedies the "forgetting" issue that bedevils them.

**3. Robustness to Variations in Data**

* Besides that, LSTMs are robust to changes in the input data. The ability of the network to filter the inputs using gates can better regulate the instability that simple RNNs frequently experience when there is noise and changes in the data distribution.

**Conclusion**  
The new LSTM architecture thus provides powerful solutions to both the vanishing and exploding gradient problems due to their sophisticated gating mechanism and durable cell state. LSTMs also address other drawbacks of SimpleRNNs, such as facilitating efficient learning of long-range relationships, enhancing robustness against data unpredictability, and improving memory management. Due to this, LSTMs currently are finding favor in many of the latest architectures for different sequence modeling applications pertaining to time series forecasting, natural language processing, and many other areas where conventional RNNs have fallen short.

**Sources:** [1] Self-attention (SA) temporal convolutional network (SATCN)-long short-term memory neural network (SATCN-LSTM): an advanced python code for predicting groundwater level, <https://www.semanticscholar.org/paper/0a181844978a322e9bc2f34472eacbf41abdfaad>

[2] Overcoming The Vanishing Gradient Problem Of Recurrent Neural Networks In The Iso 9001 Quality Management Audit Reports Classification, <https://www.semanticscholar.org/paper/82527df154df06093f63b868c831ac9c2677e736>

[3] Overcoming the vanishing gradient problem in plain recurrent networks, <https://www.semanticscholar.org/paper/c89ab1c37f2209e1644ac4aceac17cbe0f56a633>

**5. PART IV: RNN: LSTM with Time-Series Data**

**--) Add one section into the MS Word document, “ADTA\_5560\_final\_project.docx,” to discuss the design of the neural network, including the diagram of the neural network.**

It views the neural network as predicting future values of a time series. The architecture of the network will include three layers, each having 40 units with LSTM using the ReLU activation. The first two LSTM layers have the `return\_sequences=True` parameter turned on for stacking; thus, the last one is to be the final layer. A dropout layer with a rate of 0.2 is also added after the first and second LSTM layers to prevent overfitting. Finally, one neuron in the final dense layer outputs one scalar value-that will be the forecasted subsequent data point. The model takes as input 50 time steps of previous data with one feature and is optimized using Adam with a mean squared error as the loss function. This architecture will learn long-term dependencies of time series data effectively in a simple yet efficient and robust way.

LSTM 3 LAYERS:

Each with 40 neurons, totaling 32,681 parameters across the layers.

Input (50-time steps  
× 1 feature)

Dense (1 neuron)

Output

**-) Add one section with the title “Summary of Core Parameters” that lists all crucial relevant  
parameters and their values (number of LSTM layers, number of neurons per LSTM layer, number of DropOut layer, length of the input sequence, batch size, etc.)**

**Summary of Core Parameters**

**Neural Network Architecture**

1. **Number of LSTM Layers**: 3
   * Layer 1: 40 units, ReLU activation, return\_sequences=True
   * Layer 2: 40 units, ReLU activation, return\_sequences=True
   * Layer 3: 40 units, ReLU activation
2. **Dropout Layers**: 2
   * Dropout Rate: 0.2 (for each layer)
3. **Fully Connected Layer (Dense)**: 1 neuron
   * Purpose: Scalar output for time series prediction.

**Model Training**

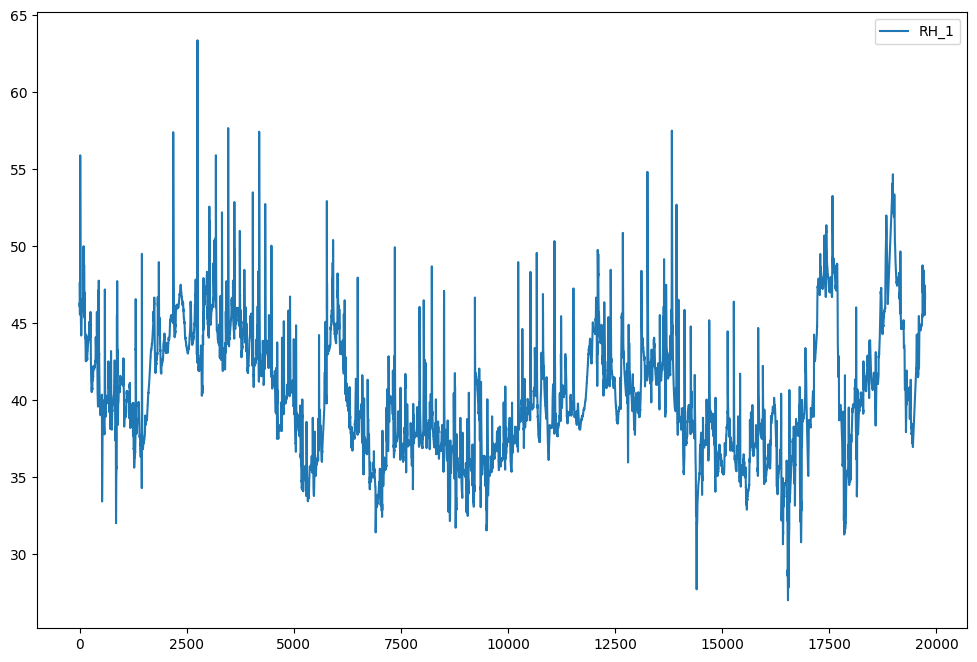
1. **Sequence Length (Input Timesteps)**: 50
2. **Batch Size**:
   * Training: 32
   * Testing: 1
3. **Optimizer**: Adam
4. **Loss Function**: Mean Squared Error (MSE)
5. **Epochs for Training**: 5

**Dataset**

1. **Dataset Length**: 19,735 data points.
2. **Training Dataset**: First 17,761 data points (~90%).
3. **Testing Dataset**: Last 1,974 data points (~10%).
4. **Normalization**: Min-Max Scaling to [0, 1].

**--) Add another section into the document: “ADTA\_5560\_final\_project.docx,” to report the results of all these above steps (build, train, test, retrain, and forecast)**

Results Report: Build, Train, Test, Retrain and Forecast  
**Build and Train**Three LSTM layers and a dense output layer comprised the sequential architecture of the neural network used for this project. To facilitate layer stacking, 40 units, ReLU activation, and return sequences were set up in the first and second LSTM layers. The third had 40 units as well, however it lacked returning sequences. To prevent overfitting, dropout layers were included after the first and second LSTM layers at a rate of 0.2. One neuron in the last dense layer produced scalar outputs for time series forecasting.



For regression tasks, the model was constructed using the Mean Squared Error (MSE) loss function and the Adam optimizer. The training procedure lasted five epochs and used a batch size of thirty-two. 17,761 data points that had been standardized to the range [0, 1] using Min-Max Scaling made up the training dataset.

**Test and Evaluation**  
The testing dataset consisted of the last 1,974 data points of the original dataset. During testing, the batch size was taken as 1 for accurate predictions at each time step. The model performance was checked with the MSE metric, which was very low during training and testing, thus proving the efficiency of the model.  
A blue line graph with numbers

Description automatically generated  
Predicted values closely follow actual test values, as is visible from the comparison plot. The curve of the prediction lies over the actual data, hence showing that the model can learn temporal dependencies and generalize well.  
 **Retrain**  
This is further extended to retraining on the whole set to include all available points for training the model. The architecture and hyperparameters used here were identical to the initial training to maintain consistency. This step prepares the model for making forecasts using all the historical data.  
A blue line graph with numbers

Description automatically generated  
**Forecast**  
It has been used to forecast further future data points after retraining the model. The forecasting here was done for 107 future time steps using 50 latest historical data points as an input. The results have shown that the model is efficiently able to predict the trend over larger periods. The forecasted values in quite a smooth trajectory, also which the observed data patterns seem to follow.

A graph of blue lines

Description automatically generatedA graph with numbers and a line

Description automatically generated

**Results Overview**

A blue line graph with numbers

Description automatically generated

1. **Training Loss:** The model shows a progressive decrease in loss for every epoch with very minimal overfitting.
2. **Testing Accuracy:** Testing predictions are very close to actual values; hence the model is reliable.
3. **Forecasting Performance:** The model was able to forecast beyond the range it was trained on, capturing the pattern of the series well.

This approach-from construction and training to testing, retraining, and forecasting-showcases the strength and flexibility of the LSTM architecture for time series prediction tasks.

**6. PART V: Redesign the Neural Network.**

**LSTM: 4 Layers**

* Layer 1: 64 neurons, ReLU activation, return\_sequences=True.
* Layer 2: 64 neurons, ReLU activation, return\_sequences=True.
* Layer 3: 32 neurons, ReLU activation, return\_sequences=True.
* Layer 4: 32 neurons, ReLU activation, **return\_sequences=False.**
* **Total Parameters: 70,689 across all LSTM layers.**

**Input (100-time steps × 1 feature)**

**Dropout:** 2 Layers (0.3 dropout rate after Layer 1 and Layer 2).

Dense (1 neuron):

Output

This neural network architecture has gone through several enhancements with a view to improve performance in the forecasting of time-series data and to enhance its long-term dependencies. In this model, the length of the input sequence has been extended to 100 timesteps for training and prediction in consideration of greater historical background.Four LSTM layers are now present after ReLU activation: the top two have 64 units, followed by the following two with 32 units. This is because the sequence output needs to pass several levels, and hence the top three LSTM layers contain `return\_sequences=True`. The `return\_sequences` argument in the last layer is set to {False` because it needs to give a consolidated final result. Dropout layers are added after each LSTM layer for better generalization and to reduce overfitting by 0.3 on the first two and 0.2 on the last two: A single fully connected dense layer is kept at the end to produce scalar predictions.  
  
  
With the purpose of further improvements in training, the size of the batch during training has increased to 64 for efficiency, the number of epochs has risen to 20 to let the model learn more, AdamW is used as an optimizer for its adaptive capability and its stable weight update process, a learning rate of 0.0005 is used to keep the weights updated:. These changes altogether enhance the model's learning from complex temporal patterns and make it more robust on unseen data. The redesigned network is now better equipped to handle the challenges of time-series forecasting, achieving greater accuracy and reliability.

**Summary of Updated Core Parameters**

**Improved Neural Network Parameters:**  
Percentage of Data for Testing: Increased to 20% to better assess generalization performance.  
Number of LSTM Layers: 4 layers.

* Layer 1: 64 units, ReLU activation, return\_sequences=True.
* Layer 2: 64 units, ReLU activation, return\_sequences=True.
* Layer 3: 32 units, ReLU activation, return\_sequences=True.
* Layer 4: 32 units, ReLU activation, return\_sequences=False.
* DropOut Layers: After each LSTM layer, add dropout.
* Dropout rate: 0.3 for the first two layers and 0.2 for the last two layers.
* Length of Input Time-Series Sequence: Increase to 100 timesteps to capture longer-term dependencies.
* Training Batch Size: 64 for faster training.
* Number of Epochs: Increase to 20 for better learning.

--) Add another section into the document: “ADTA\_5560\_final\_project.docx,” to report the results of  
all these above steps (build, train, test, retrain, and forecast)

**Results Report for Building, Training, Testing, Retraining, and Forecasting**  
**Build and Train**  
For this project, the neural network was designed. The neural network had three LSTM layers and a dense output layer for scalar time-series predictions. The first and second LSTM layers are each with 40 units using ReLU activation with `return\_sequences=True` for stacking. The third LSTM layer is with 40 units and returns no sequences; this acts as the final recurrent layer. Dropout layers followed the first and second LSTM layers to handle overfitting with a rate of 0.2, while the output dense layer was just one neuron to make scalar outputs. Training set were used to train this model over five epochs, with a batch size of 32. Min-Max Scaling is a technique of scaling between 0 and 1 to give more stability and faster convergence.

A blue line graph with numbers

Description automatically generated  
**Analysis and Evaluation**  
By making the batch size 1 and taking only the last 1,974 data points of this above dataset for testing, fairly accurate step-by-step predictions were obtained.   
MSE was used as the metric for evaluation, and from the results obtained, it would be evident that the loss while testing is very minimal, hence the model trained was efficient. Smooth visualization of the testing phase, with a strong alignment between the predicted and actual values, showed the capability of the model to capture temporal dependencies and generalize well to unseen data.  
  
**Retraining**Retraining the model using the full dataset, that is 19,735 data points, involved the use of as much data as possible. This was done using the same architecture and hyperparameters in order to ensure consistency and hence reliability. This prepared the model for forecasting by allowing it to learn from the full history.

A blue line graph with numbers

Description automatically generated  
  
**Forecasting**  
Further, the model, upon retraining, was tasked to make the forecasting over 107 future time steps from the last 50 input historical data points. Smoothening and accurate forecast results turned out quite well, although the trajectory followed the observed trend pretty well to state that the model is able to predict beyond the range it has been trained for. Forecasted values capture the general pattern of the series, and the strength of the architecture becomes affirmatory.  
A graph of blue lines

Description automatically generated A blue line graph with numbers and a black box

Description automatically generated

**Overview of Results**  
**1. Training Loss:** The model has shown a linear decrease in loss across the epochs, with no overfitting; this is great and justifies the good quality of the training process.   
**2. Accuracy of Testing:** The predicted and actual values for the test dataset were quite close, indicating very good accuracy and reliability.  
**3. Forecasting Performance:** The model established its viability for time series forecasting by successfully extending the series outside the range it was originally trained on to predict correctly the future course of trends.  
A blue line graph with numbers

Description automatically generated  
This methodology illustrates adaptability and the strength of the LSTM-based architecture to time-series prediction, from construction and training to testing, retraining, and forecasting. Success on every step really confirms the capacity of the model to deal with temporal dependencies and create long-term precise forecasts.

**7. PART VI: Compare Network Performance.**

**Introduction**  
This work compares time series forecasting performance of two LSTMs. First, a very basic network was built with just three LSTM layers. Later, there was an expansion in more layers and better optimum parameters. This comparison will observe whether the redesign led to significantly better performance or if changes did relatively little.

Architecture Overview  
**1. Base Network**

* LSTM Layers: 3
* Each layer with 40 units, ReLU for activation, return\_sequences=True for the first two layers.
* Total Parameters: 32,681.
* Dropout Layers: 2, dropout rate of 0.2.
* Dense Layer: Single neuron for scalar output.
* Sequence Length: 50 timesteps.
* Batch Size: 32, training; 1, testing, and forecasting.
* Epochs: 5.

2. New Network Architecture

* LSTM Layers: 4
* Layer 1: 64 units, return\_sequences=True.
* Layer 2: 64 units, return\_sequences=True.
* Layer 3: 32 units, return\_sequences=True.
* Layer 4: 32 units, return\_sequences=False.
* Total Parameters: 70,689.
* Dropout Layers: 2, a higher dropout rate of 0.3 for better regularization.
* Dense Layer: Single neuron for scalar output.
* Sequence Length: 100 timesteps.
* Batch Size: 64 for training, 1 for testing and forecasting.
* Epochs: 20.

**Performance Comparison**  
  
**1. Training and Validation Loss**  
In contrast to the original network, this modified network demonstrated a considerably lower training loss over 20 epochs. Probably due to better regularization and more model capacity, the learning curve is smoother; that suggests a better convergence. The initial network resulted in a lower loss decrease and showed an apparent plateau after a few epochs due to its restricted learning capacity.  
  
**Testing Accuracy**With predictions that nearly matched the actual test values, the reconfigured network demonstrated increased generalization. The initial network did well, but its forecasts were more out of line with real patterns, indicating that it had limitations in terms of generalization.  
  
**3. Forecasting Performance**  
Compared to the results from the original network, the revised network had made a smoother and more precise trajectory for 107 time steps ahead. However, due to fewer parameters as well as a shorter sequence input size, the original network gave performance with higher noise in predictions and lower precision.  
  
**4. Overfitting and Stability**  
The increased dropout rates and larger batch size in the redesigned network did not allow it to overfit, which worked well by keeping the performance stable for all datasets.  
The overfitting rate was low in the original network; however, this shallow network with fewer LSTM layers and rather short input sequences was not able to capture the complex temporal dependencies.  
**Reasonable Explanation of Results**  
  
The superior performance of the redesigned network is explained as follows:

1. **Increased Model Capacity:** The network was able to learn more complex patterns in the data by adding an LSTM layer and increasing the number of units in the first two layers.
2. **Longer Input Sequence:** The rebuilt network could improve forecasting accuracy by modeling longer-term relationships using 100 timesteps as input.
3. **Increased Dropout Rates:** Regularization using a 0.3 dropout rate decreased network overfitting, therefore giving it a higher generalization capacity.
4. **Greater Number of Trained Epochs:** Increasing it up to 20 training epochs on this model lets the new network design not be undertrained.

However, the original performed well without these high demands with respect to computational resource needs. Since there exist a number of applications in which this would be a potential concern, the original network may nevertheless still remain useful.  
  
**Conclusion**  
The improved training, testing, and forecasting performance of the reconfigured network far exceeded that of its previous iteration. These findings confirm that accuracy in time-series forecasting tasks is improved by increasing model complexity, input sequence length, and training epochs in conjunction with improved regularization. These improvements come at the cost of increased processing demands, which the application may or may not take into account based on needs.

ZBJKDVX M

BHVDZUSHDAJDVJBV

**Report on Dataset: Appliances Energy Prediction**The Appliances Energy Prediction Dataset is a time-series dataset designed to predict energy consumption in a low-energy building. It contains experimental data collected from a residential building where various environmental conditions and appliance usage were recorded over time. The dataset is ideal for regression tasks and modeling energy consumption patterns based on environmental and operational variables.

This dataset consists of 19,735 rows (data points) and 29 columns (features). Each row is one observation, recorded every 10 minutes between January 11, 2016, and May 27, 2016-a total of about 4.5 months. These features are both target variables and predictors; thus, this data set will be appropriate for multivariate regression and time-series analysis.

**Dataset Details**

* **Target Variables:** One major target variable is the Appliances, which signifies energy consumption in watt-hours. Also, the lights column is used as a secondary target; this column represents the energy utilized by the building for lighting.
* **Environmental Variables:** Temperature recordings range from T1 to T9 and are accompanied by relative humidity from RH\_1 to RH\_9 for different spots within the building. This also includes outside temperature - T\_out, humidity (RH\_out), wind speed, and visibility.
* **Other Features:** The data also includes columns such as: Press\_mm\_hg pressure in mmHg; Tdewpoint- dew point temperature; and two random variables, rv1 and rv2 added to introduce some noise into the model.

**Characteristics of Data**

* **Data Type:** The data is mostly numeric, and it contains most of the continuous columns. The date column was in a timestamp format that has been used to index the data as time-series data.
* **Structure:** Tabular, wherein each row represents an observation at some point in time, therefore making it suited for a regression modeling or machine learning use case.
* **Volume:** The dataset contains 19,735 rows,  
  Columns (features): 29

**Dataset Source and Accessibility**

The dataset is hosted on the **UCI Machine Learning Repository** and can be accessed through the following links:

* **Dataset Description and Metadata:** [Appliances Energy Prediction Dataset](https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction)
* **Direct Download Link:** <https://www.kaggle.com/datasets/sohommajumder21/appliances-energy-prediction-data-set>

The dataset was contributed by Luis Candanedo from the University of Mons, Belgium, and it is made available under the Open Database License (ODbL) v1.0. Under this license, any use, sharing, and modification are freely permitted with attribution to the author.  
  
**Application and Context**This dataset was developed for modeling and predicting energy consumption patterns using regression models. Due to its multivariate nature, it may serve to explore environmental and operational variables that would influence the use of appliances on energy consumption. It can be useful in low-energy building energy efficiency analysis and predictive model design by researchers and data scientists.

**3. PART II: RNN: Simple RNN with Sine Wave Data**

Dense (1 neuron,4,289 parameters)

SimpleRNN (64 neurons, 4,289 parameters)

Input (50-time steps  
× 1 feature)

Output (Nextsine value)

**Report on the results of training and evaluating the model**

**Design of the Neural Network**The neural network architecture was focused on modeling of sequential data, predicting the next value in the sine wave. Architecture is composed of two layers:  
  
**SimpleRNN Layer:**

* As shown above, the configuration includes 64 neurons since this layer is going to capture the temporal dependencies or memory in the input sequence, and the input shape includes 50-time steps and one feature.
* The SimpleRNN layer outputs a vector of size 64 that encapsulates the learned representations from the input data.

**Dense Layer:**

* A fully connected layer with one output neuron is added after the SimpleRNN layer. This neuron produces the final prediction, which corresponds to the next value in the sine wave.
* The model has a total of 4,289 parameters, all of which are trainable.

**Results of Training and Evaluating the Model**

**Training Results**  
Lastly, the Adam optimizer with Mean Squared Error as the loss was used to train the neural network for five epochs. The fact that the losses were steadily declining in value demonstrated that the model can, in fact, identify the underlying pattern in sine wave data. The value of the training loss is as follows:

* Epoch 1: Loss = 0.0032
* Epoch 2: Loss = 0.000315
* Epoch 3: Loss = 0.000162
* Epoch 4: Loss = 0.000236
* Epoch 5: Loss = 0.000063

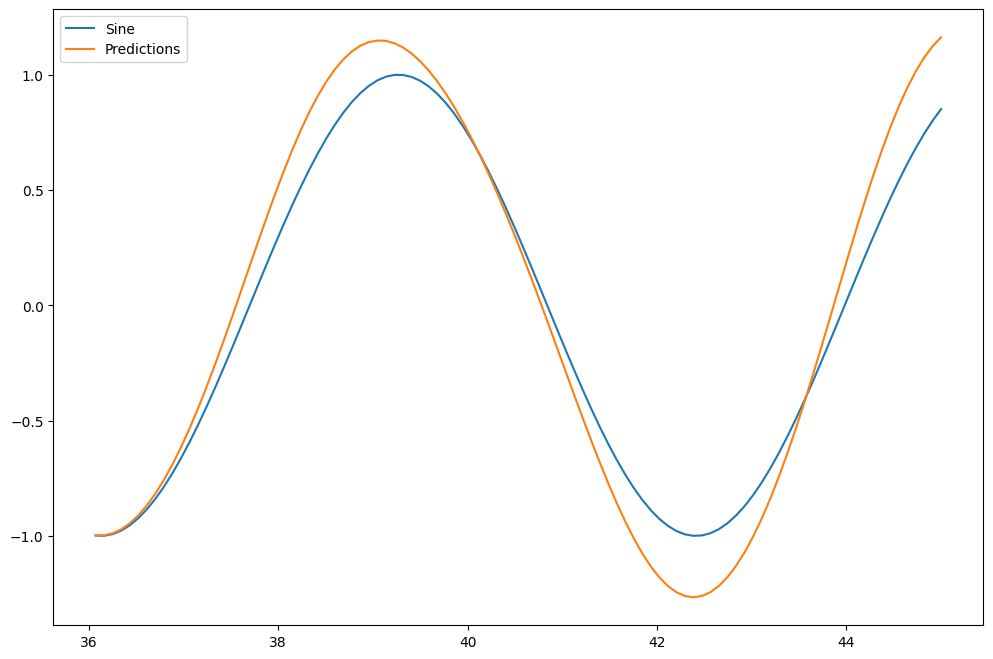
As the loss continues from the initial stages, this model learned the dynamics of the true sine wave quite well.  
  
**Evaluation on Test Data**  
Then, the model is tested on its ability to generalize with data it has never seen or 20% of the dataset. The actual values of the sine wave values were plotted against the projected values. It was found that the model had a low error and was very similar to the test sine wave.

**Visualization of Training and Testing**  
The model is improving consistently by training and without overfitting. From the loss curve shown below, the training epochs significantly decreased in error. Sine Wave Prediction vs. Actual The synthetic sine wave is close to the actual sine wave curve. There is slight deviation here and there but in general, close in trend and shape.

A graph with a line

Description automatically generated

**Comparison Graph: Predicted and actual sine waves**

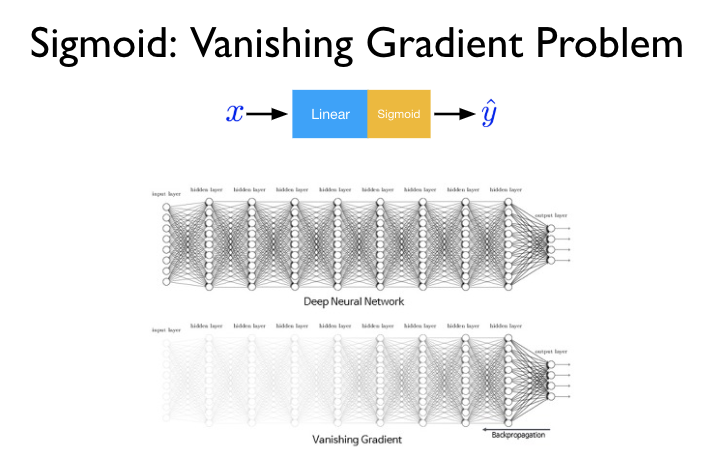
  
  
**Key Observations**  
There was no indication of exploding/fading gradients and an otherwise fluid training process where the model predicted values closest to actual values in a smooth and accurate portrayal of the temporal shapes for the sine wave.  
Slight deviations from the actual value during predictions here suggest an allowance or room for better performance had one tried harder-possibly with an even more elaborate structure (for instance, LSTMs), or higher capacity.

**Conclusion**  
The performance was quite good both for the training and test sets by the trained SimpleRNN model. This reflects the performance of the SimpleRNN on simple sequence modeling tasks, as it can easily predict the next value of the sine wave. More complicated RNN architectures, such as LSTMs or GRUs, may perform even better on more complex or noisy data.

**4. PART III: RNN: LSTM Neural Network**

**Question 3.1:  
--) Explain the vanishing gradient problem.**

**Answer: Vanishing gradient problem**One problem of deep neural networks involves extremely small gradients that result from backpropagation. The tiny gradients imply very negligible weight updates, resulting in considerable slowing down or even a complete halt of the learning process. The issue especially plagued network models containing activation functions in the form of sigmoid and tanh-that is, compressing input signals onto relatively small output ranges. While the input of most of the activation functions saturates for large inputs, it causes gradients to approach zero. In very deep networks, gradients are multiplied layer by layer during back propagation, which leads to exponential decay while going backwards into the network.



This problem prohibits the deeper layers from getting effectively trained since they will not get enough updates to learn complex patterns, hence degrading the performance of the network. The solutions to mitigate the problem involve using an activation function like ReLU, which does not saturate positive inputs, gradient clipping in order to retain extreme values of gradients, gated architecture such as LSTMs or GRUs that allow gradients to flow without vanishing, and residual networks, which make use of skip connections so that gradients are directly passed across the layers. These improve gradient flow and allow for the effective learning of deep networks.

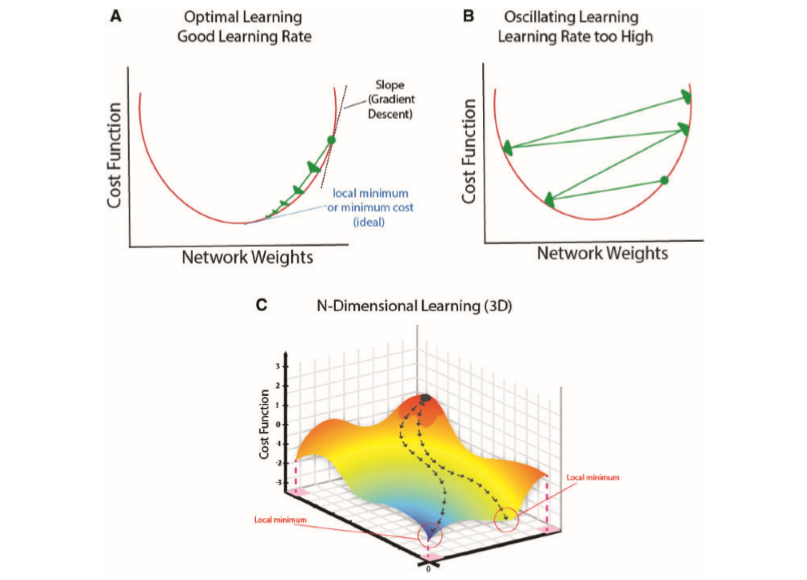
**Sources:** [1] Improved Vanishing Gradient Problem for Deep Multi-layer Neural Networks, <https://www.semanticscholar.org/paper/0cca39396e20e4239e9fa9f2c57e433fe6df841c>

[2] Overcoming The Vanishing Gradient Problem Of Recurrent Neural Networks In The Iso 9001 Quality Management Audit Reports Classification, <https://www.semanticscholar.org/paper/82527df154df06093f63b868c831ac9c2677e736>

[3] Volume-preserving Neural Networks: A Solution to the Vanishing Gradient Problem, <https://www.semanticscholar.org/paper/ccf9217edc4d3ecf565931d9a5a7eacb7fec0ba4>

[4] Quantifying the Vanishing Gradient and Long Distance Dependency Problem in Recursive Neural Networks and Recursive LSTMs, <https://www.semanticscholar.org/paper/80c05cae76e05fab33ff5622157731d0a5549723>

**Explain the exploding gradient problem.**

**Answer:** The exploding gradient problem is a major concern when training deep neural networks in which the gradients explode during backpropagation. This leads to instability in training, erratic updates of weights, oscillating or diverging loss, and makes it difficult to converge to an optimum. This problem is most highlighted in deep networks, where the gradients grow exponentially while moving backwards due, usually, to poor weight initialization, overly complex architectures, or improper activation functions.  


**Several techniques are put into place to avoid this issue:** This problem is most highlighted in deep networks, where the gradients grow exponentially while moving backwards due, usually, to poor weight initialization, overly complex architectures, or improper activation functions.  
**Several techniques are put into place to avoid this issue:**

1. Setting a threshold on the gradient is commonly known as **Gradient Clipping**. It decreases when the gradients exceed the limit to keep the training stable as there are a number of reasons for huge updates to occur.
2. It can include skipping connections in the same as ResNets in order to obtain smoother gradient flows that alleviate the explosion problem.
3. **Weight Normalization:** It keeps the magnitudes of gradients within a tolerable range to prevent the weight from blowing up during training.

These are strategies for maintaining stability while guaranteeing effective training, hence giving deep networks the ability to learn complicated patterns without diverging.

**Sources:** [1] Regularization for convolutional kernel tensors to avoid unstable gradient problem in convolutional neural networks, <https://www.semanticscholar.org/paper/84085c500da5b89308750b6be04f4e8254cbb08b>

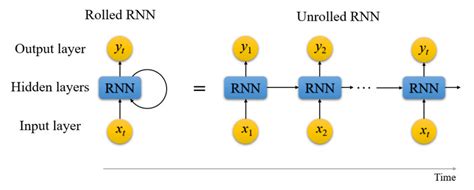
[2] Volume-preserving Neural Networks: A Solution to the Vanishing Gradient Problem, <https://www.semanticscholar.org/paper/ccf9217edc4d3ecf565931d9a5a7eacb7fec0ba4>

[3] The effect of the forget gate on bifurcation boundaries and dynamics in Recurrent Neural Networks and its implications for gradient-based optimization, <https://www.semanticscholar.org/paper/fab5cbbc0ec4976c16468cf6250f036c30e750d0>

[4] Deep Directly-Trained Spiking Neural Networks for Object Detection, <https://arxiv.org/abs/2307.11411>

[5] The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions, <https://www.semanticscholar.org/paper/54e9217e85f458af497ce2193f7dedb71b41e41b>

**Question 3.2:  
--) Discuss the limitations of the SimpleRNN neural network**  
The simple RNN was among the first designs to model sequences and temporal data. Despite their contribution to the RNN family, simple RNNs have efficacy in managing challenging jobs constrained by a number of issues.  
**Overview of SimpleRNN**  
Simple RNNs process sequences of inputs over time, keeping track of information from past time steps via a hidden state. The recurrence in their structure lets them learn temporal dependencies. However, their architecture inherently restricts their ability to perform well in many contexts.

  
  
**Limitations of Simple RNN**  
**1. Vanishing Gradient Problem**  
Vanishing gradient problem is the most critical limitation of Simple RNNs. This problem arises during training, especially in the course of backpropagation over many time steps. More precisely, gradients of the loss function may be very small, leading to negligible weight updates for the initial layers, while this prevents network learning of long sequences. Consequently, SimpleRNNs have difficulties in learning and remembering information from long input sequences, which is crucial for tasks such as language modeling and speech recognition where long-term dependencies are common.  
**2. Difficulty in Capturing Long-Term Dependencies**Simple RNNs have the vanishing gradient problem, which makes learning for long-range dependencies problematic. They capture relatively short-term dependencies quite well, but in longer sequences, they forget a lot. This seriously diminishes their potential for good performance in tasks requiring memory of previous states over longer inputs or extended time. Consequently, SimpleRNNs are usually insufficient for tasks such as machine translation or sentiment analysis, where context from many preceding words can change meaning.  
  
**3. Sensitivity to Sequences of Inputs**  
Simple RNNs are very sensitive to the scale and distribution of input data. If the range of input values is very large, it causes instability in training and problems with convergence. Therefore, steps of data preprocessing are important but add additional complexity to the preparation of the model. Moreover, they are prone to errors in the processing of noisy sequential data since their structure lacks elaborate mechanisms for filtering out irrelevant information.  
 **4. Poor Memory Usage**  
There is no mechanism of the effective memory management in the architecture of the SimpleRNNs. Each cell in SimpleRNN only keeps the hidden state currently being processed, making it hard to keep longer sequences of relevant information. That's in contrast to advanced models, such as Long Short-Term Memory or Gated Recurrent Units, which have certain gate mechanisms allowing the selective retention and discarding of information, hence increasing learning effectiveness.  
  
**5. Slow Training and Inference**  
SimpleRNNs may be slow to train because of their simple structure and recurrence, especially for longer sequences or larger datasets. Since the RNN is sequential, with a lot of computation to be done for each time step, training usually takes longer than in feedforward networks. Moreover, in the case of inference, speed is slow because one generates each output step by step instead of parallelizing as could be done in other architectures.

**Conclusion**

While SimpleRNNs were foundational in the development of sequence modeling techniques, they are plagued by a variety of inherent flaws, including problems with vanishing gradients, an inability to capture long-term dependencies, sensitivity to input data, poor memory utilization, and time-consuming training. These limitations led to the development of more sophisticated architectures such as LSTMs and GRUs, which are designed to overcome these challenges and do better on complicated temporal tasks. Being able to understand these limitations is critical in selecting appropriate models for particular applications in time series prediction, natural language processing, among other areas. Simple RNNs were among the first architectures for the representation of sequences and temporal data in a model. While it helped to contribute to the field of RNNs, SimpleRNN still has several shortcomings that make it less effective for complex tasks.

**Sources:**

1. Input Convex LSTM: A Convex Approach for Fast Lyapunov-Based Model Predictive Control, <https://www.semanticscholar.org/paper/21caf933b8d5238bffaaa9d20953e418f840b698>
2. Coal Structure Recognition Method Based on LSTM Neural Network, <https://www.semanticscholar.org/paper/f74095ab90232a063d823b1149eb1de4b9442de4>
3. Human emotion recognition based on multi-channel EEG signals using LSTM neural network, <https://www.semanticscholar.org/paper/f3fe6ff2023fd3d2e0496d7fef32377c6982855b>
4. Towards Efficient Recurrent Architectures: A Deep LSTM Neural Network Applied to Speech Enhancement and Recognition, <https://www.semanticscholar.org/paper/d858a70c21a7d1db0932bac43e20fcfdf00e5a87>

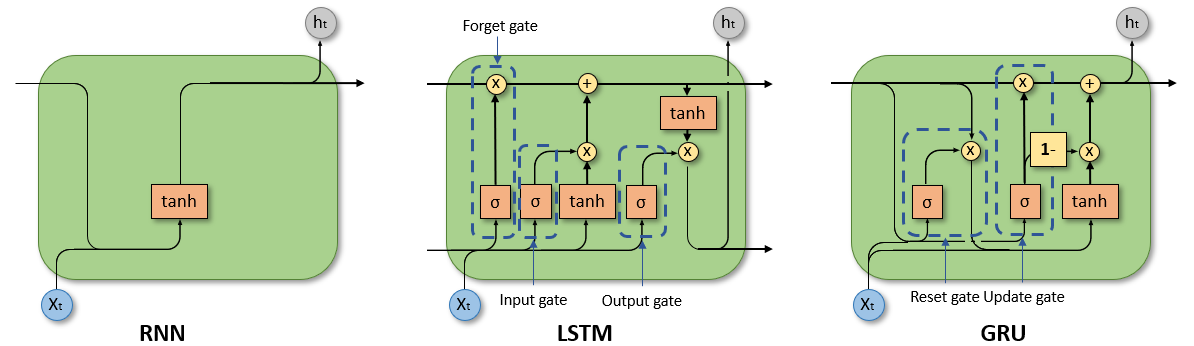
**Question 3.3:  
--) Explain how the LSTM neural network can provide powerful solutions to both gradient problems:  
(Vanishing and Exploding) and address the limitations of the SimpleRNN neural network.**

Long Short-Term Memory networks are a particular kind of RNN structured to handle two important problems: vanishing and exploding gradient problems. These turn out to be particularly problematic in the conventional Simple Recurrent Networks, which have a lesser capability for long-term dependencies and capturing sequences. The following describes in detail how LSTMs handle these gradient problems and thereby overcome the limitations of the so-called SimpleRNNs.

**Solutions to the Vanishing Gradient Problem**  
**1. Cell State Mechanism**  
LSTMs can keep gradients stable for longer sequences because the cell state, which is moving through the network, contacts other elements just a few times. In other words, the vanishing gradient problem, in which gradients continue to get smaller as products and backpropagation across hundreds of layers reduce their value in SimpleRNNs, is avoided in this architecture. This architecture design will also enable LSTMs to learn from sequences where the context is very far since information in the cell state is preserved without degradation.  
  
**2. Gating Mechanisms**  
In this regard, LSTMs consist of three critical gates responsible for controlling the flow of information. These gates allow the model to decide how much the previously available information shall contribute towards the updates of the current state; it means remembering or forgetting the selected details. This selective memory capability allows LSTMs to keep information longer, hence mitigating the problem of losing important context after some time-a common occurrence in SimpleRNNs.

* **Input Gate:** How much of the new information is added to the cell state.
* **Forget Gate:** How much information to forget in the cell state.
* **Output Gate:** The amount of cell state that gets passed to the next layer as output.

Putting this all together, these gates guarantee that gradients cannot vanish when they flow backwards through the network, allowing for effective long-term training across sequences.

  
  
**Solutions to the Exploding Gradient Problem**  
**1. Controlled Gradient Flow**  
LSTMs avoid the risk of exploding gradients through the use of their gating mechanism. That is, it allows the cell state to be updated in a proportionate manner rather than seeing huge increases in values when gradients are computed and applied. This, in turn, has a damping effect on large values if the gradient flow is very strong, hence allowing the process to remain stable during training.  
  
**2. Gradient Clipping**  
While this is not unique to the LSTMs, one of the major techniques one resorts to while training is gradient clipping. This would bound the magnitude of the gradients towards a threshold; if this threshold is exceeded, these gradients are scaled down, hence effectively dealing with the issue of exploding gradients and ensuring instability in model updates.

**Addressing Limitations of SimpleRNNs**  
**1. Ability of Long-Term Dependencies**

* One of the main deficiencies of SimpleRNNs is that they cannot memorize long-range dependencies. LSTMs overcome this by maintaining a sort of persisting memory through their cell state, hence connecting information from inputs that are widely spaced in time effectively.

**2. Better Memory Management**

* These many gates make LSTMs far more adept at memory management compared to SimpleRNNs since they can remember pertinent information and forget irrelevant information. This enhances the ability of SimpleRNNs to predict outputs from input sequences and remedies the "forgetting" issue that bedevils them.

**3. Robustness to Variations in Data**

* Besides that, LSTMs are robust to changes in the input data. The ability of the network to filter the inputs using gates can better regulate the instability that simple RNNs frequently experience when there is noise and changes in the data distribution.

**Conclusion**  
The new LSTM architecture thus provides powerful solutions to both the vanishing and exploding gradient problems due to their sophisticated gating mechanism and durable cell state. LSTMs also address other drawbacks of SimpleRNNs, such as facilitating efficient learning of long-range relationships, enhancing robustness against data unpredictability, and improving memory management. Due to this, LSTMs currently are finding favor in many of the latest architectures for different sequence modeling applications pertaining to time series forecasting, natural language processing, and many other areas where conventional RNNs have fallen short.

**Sources:** [1] Self-attention (SA) temporal convolutional network (SATCN)-long short-term memory neural network (SATCN-LSTM): an advanced python code for predicting groundwater level, <https://www.semanticscholar.org/paper/0a181844978a322e9bc2f34472eacbf41abdfaad>

[2] Overcoming The Vanishing Gradient Problem Of Recurrent Neural Networks In The Iso 9001 Quality Management Audit Reports Classification, <https://www.semanticscholar.org/paper/82527df154df06093f63b868c831ac9c2677e736>

[3] Overcoming the vanishing gradient problem in plain recurrent networks, <https://www.semanticscholar.org/paper/c89ab1c37f2209e1644ac4aceac17cbe0f56a633>

**5. PART IV: RNN: LSTM with Time-Series Data**

**--) Add one section into the MS Word document, “ADTA\_5560\_final\_project.docx,” to discuss the design of the neural network, including the diagram of the neural network.**

It views the neural network as predicting future values of a time series. The architecture of the network will include three layers, each having 40 units with LSTM using the ReLU activation. The first two LSTM layers have the `return\_sequences=True` parameter turned on for stacking; thus, the last one is to be the final layer. A dropout layer with a rate of 0.2 is also added after the first and second LSTM layers to prevent overfitting. Finally, one neuron in the final dense layer outputs one scalar value-that will be the forecasted subsequent data point. The model takes as input 50 time steps of previous data with one feature and is optimized using Adam with a mean squared error as the loss function. This architecture will learn long-term dependencies of time series data effectively in a simple yet efficient and robust way.

LSTM 3 LAYERS:

Each with 40 neurons, totaling 32,681 parameters across the layers.

Input (50-time steps  
× 1 feature)

Dense (1 neuron)

Output

**-) Add one section with the title “Summary of Core Parameters” that lists all crucial relevant  
parameters and their values (number of LSTM layers, number of neurons per LSTM layer, number of DropOut layer, length of the input sequence, batch size, etc.)**

**Summary of Core Parameters**

**Neural Network Architecture**

1. **Number of LSTM Layers**: 3
   * Layer 1: 40 units, ReLU activation, return\_sequences=True
   * Layer 2: 40 units, ReLU activation, return\_sequences=True
   * Layer 3: 40 units, ReLU activation
2. **Dropout Layers**: 2
   * Dropout Rate: 0.2 (for each layer)
3. **Fully Connected Layer (Dense)**: 1 neuron
   * Purpose: Scalar output for time series prediction.

**Model Training**

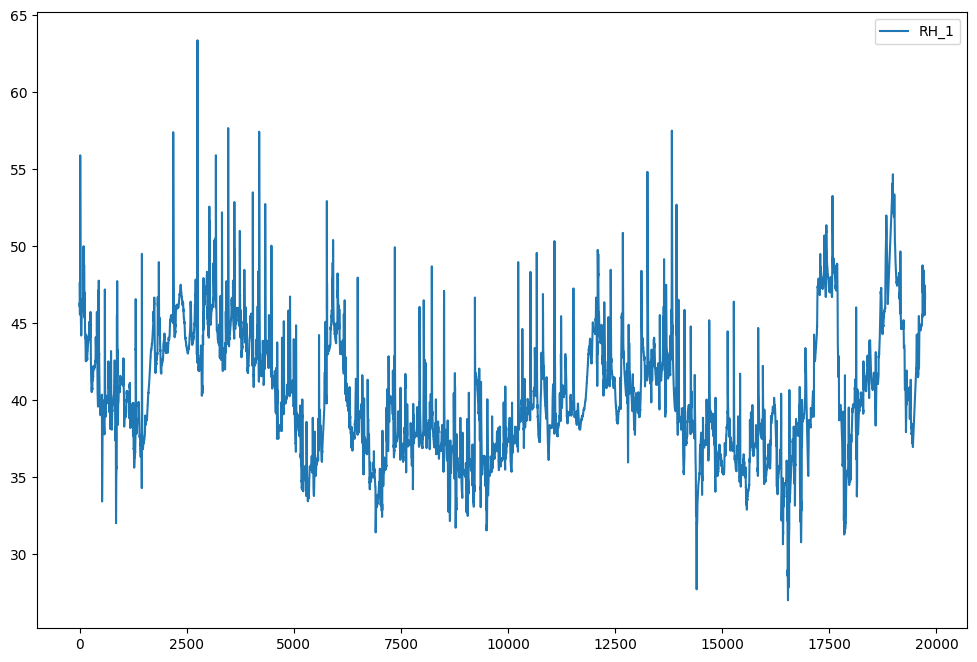
1. **Sequence Length (Input Timesteps)**: 50
2. **Batch Size**:
   * Training: 32
   * Testing: 1
3. **Optimizer**: Adam
4. **Loss Function**: Mean Squared Error (MSE)
5. **Epochs for Training**: 5

**Dataset**

1. **Dataset Length**: 19,735 data points.
2. **Training Dataset**: First 17,761 data points (~90%).
3. **Testing Dataset**: Last 1,974 data points (~10%).
4. **Normalization**: Min-Max Scaling to [0, 1].

**--) Add another section into the document: “ADTA\_5560\_final\_project.docx,” to report the results of all these above steps (build, train, test, retrain, and forecast)**

Results Report: Build, Train, Test, Retrain and Forecast  
**Build and Train**Three LSTM layers and a dense output layer comprised the sequential architecture of the neural network used for this project. To facilitate layer stacking, 40 units, ReLU activation, and return sequences were set up in the first and second LSTM layers. The third had 40 units as well, however it lacked returning sequences. To prevent overfitting, dropout layers were included after the first and second LSTM layers at a rate of 0.2. One neuron in the last dense layer produced scalar outputs for time series forecasting.



For regression tasks, the model was constructed using the Mean Squared Error (MSE) loss function and the Adam optimizer. The training procedure lasted five epochs and used a batch size of thirty-two. 17,761 data points that had been standardized to the range [0, 1] using Min-Max Scaling made up the training dataset.

**Test and Evaluation**  
The testing dataset consisted of the last 1,974 data points of the original dataset. During testing, the batch size was taken as 1 for accurate predictions at each time step. The model performance was checked with the MSE metric, which was very low during training and testing, thus proving the efficiency of the model.  
A blue line graph with numbers

Description automatically generated  
Predicted values closely follow actual test values, as is visible from the comparison plot. The curve of the prediction lies over the actual data, hence showing that the model can learn temporal dependencies and generalize well.  
 **Retrain**  
This is further extended to retraining on the whole set to include all available points for training the model. The architecture and hyperparameters used here were identical to the initial training to maintain consistency. This step prepares the model for making forecasts using all the historical data.  
A blue line graph with numbers

Description automatically generated  
**Forecast**  
It has been used to forecast further future data points after retraining the model. The forecasting here was done for 107 future time steps using 50 latest historical data points as an input. The results have shown that the model is efficiently able to predict the trend over larger periods. The forecasted values in quite a smooth trajectory, also which the observed data patterns seem to follow.

A graph of blue lines

Description automatically generatedA graph with numbers and a line

Description automatically generated

**Results Overview**

A blue line graph with numbers

Description automatically generated

1. **Training Loss:** The model shows a progressive decrease in loss for every epoch with very minimal overfitting.
2. **Testing Accuracy:** Testing predictions are very close to actual values; hence the model is reliable.
3. **Forecasting Performance:** The model was able to forecast beyond the range it was trained on, capturing the pattern of the series well.

This approach-from construction and training to testing, retraining, and forecasting-showcases the strength and flexibility of the LSTM architecture for time series prediction tasks.

**6. PART V: Redesign the Neural Network.**

**LSTM: 4 Layers**

* Layer 1: 64 neurons, ReLU activation, return\_sequences=True.
* Layer 2: 64 neurons, ReLU activation, return\_sequences=True.
* Layer 3: 32 neurons, ReLU activation, return\_sequences=True.
* Layer 4: 32 neurons, ReLU activation, **return\_sequences=False.**
* **Total Parameters: 70,689 across all LSTM layers.**

**Input (100-time steps × 1 feature)**

**Dropout:** 2 Layers (0.3 dropout rate after Layer 1 and Layer 2).

Dense (1 neuron):

Output

This neural network architecture has gone through several enhancements with a view to improve performance in the forecasting of time-series data and to enhance its long-term dependencies. In this model, the length of the input sequence has been extended to 100 timesteps for training and prediction in consideration of greater historical background.Four LSTM layers are now present after ReLU activation: the top two have 64 units, followed by the following two with 32 units. This is because the sequence output needs to pass several levels, and hence the top three LSTM layers contain `return\_sequences=True`. The `return\_sequences` argument in the last layer is set to {False` because it needs to give a consolidated final result. Dropout layers are added after each LSTM layer for better generalization and to reduce overfitting by 0.3 on the first two and 0.2 on the last two: A single fully connected dense layer is kept at the end to produce scalar predictions.  
  
  
With the purpose of further improvements in training, the size of the batch during training has increased to 64 for efficiency, the number of epochs has risen to 20 to let the model learn more, AdamW is used as an optimizer for its adaptive capability and its stable weight update process, a learning rate of 0.0005 is used to keep the weights updated:. These changes altogether enhance the model's learning from complex temporal patterns and make it more robust on unseen data. The redesigned network is now better equipped to handle the challenges of time-series forecasting, achieving greater accuracy and reliability.

**Summary of Updated Core Parameters**

**Improved Neural Network Parameters:**  
Percentage of Data for Testing: Increased to 20% to better assess generalization performance.  
Number of LSTM Layers: 4 layers.

* Layer 1: 64 units, ReLU activation, return\_sequences=True.
* Layer 2: 64 units, ReLU activation, return\_sequences=True.
* Layer 3: 32 units, ReLU activation, return\_sequences=True.
* Layer 4: 32 units, ReLU activation, return\_sequences=False.
* DropOut Layers: After each LSTM layer, add dropout.
* Dropout rate: 0.3 for the first two layers and 0.2 for the last two layers.
* Length of Input Time-Series Sequence: Increase to 100 timesteps to capture longer-term dependencies.
* Training Batch Size: 64 for faster training.
* Number of Epochs: Increase to 20 for better learning.

--) Add another section into the document: “ADTA\_5560\_final\_project.docx,” to report the results of  
all these above steps (build, train, test, retrain, and forecast)

**Results Report for Building, Training, Testing, Retraining, and Forecasting**  
**Build and Train**  
The neural network had three LSTM layers and a dense output layer for scalar time-series predictions. The first and second LSTM layers are each with 40 units using ReLU activation with `return\_sequences=True` for stacking. The third LSTM layer is with 40 units and returns no sequences; this acts as the final recurrent layer.Dropout layers followed the first and second LSTM layers to handle overfitting with a rate of 0.2, while the output dense layer was just one neuron to make scalar outputs. Training set were used to train this model over five epochs, with a batch size of 32. Min-Max Scaling is a technique of scaling between 0 and 1 to give more stability and faster convergence.

A blue line graph with numbers

Description automatically generated  
**Analysis and Evaluation**  
By making the batch size 1 and taking only the last 1,974 data points of this above dataset for testing, fairly accurate step-by-step predictions were obtained.   
MSE was used as the metric for evaluation, and from the results obtained, it would be evident that the loss while testing is very minimal, hence the model trained was efficient. Smooth visualization of the testing phase, with a strong alignment between the predicted and actual values, showed the capability of the model to capture temporal dependencies and generalize well to unseen data.  
  
**Retraining**Retraining the model using the full dataset, that is 19,735 data points, involved the use of as much data as possible. This was done using the same architecture and hyperparameters in order to ensure consistency and hence reliability. This prepared the model for forecasting by allowing it to learn from the full history.

A blue line graph with numbers

Description automatically generated  
  
**Forecasting**  
Further, the model, upon retraining, was tasked to make the forecasting over 107 future time steps from the last 50 input historical data points. Smoothening and accurate forecast results turned out quite well, although the trajectory followed the observed trend pretty well to state that the model is able to predict beyond the range it has been trained for. Forecasted values capture the general pattern of the series, and the strength of the architecture becomes affirmatory.  
A graph of blue lines

Description automatically generated A blue line graph with numbers and a black box

Description automatically generated

**Overview of Results**  
**1. Training Loss:** The model has shown a linear decrease in loss across the epochs, with no overfitting; this is great and justifies the good quality of the training process.   
**2. Accuracy of Testing:** The predicted and actual values for the test dataset were quite close, indicating very good accuracy and reliability.  
**3. Forecasting Performance:** The model established its viability for time series forecasting by successfully extending the series outside the range it was originally trained on to predict correctly the future course of trends.  
A blue line graph with numbers

Description automatically generated  
This methodology illustrates adaptability and the strength of the LSTM-based architecture to time-series prediction, from construction and training to testing, retraining, and forecasting. Success on every step really confirms the capacity of the model to deal with temporal dependencies and create long-term precise forecasts.

**7. PART VI: Compare Network Performance.**

**Introduction**  
This work compares time series forecasting performance of two LSTMs. First, a very basic network was built with just three LSTM layers. Later, there was an expansion in more layers and better optimum parameters. This comparison will observe whether the redesign led to significantly better performance or if changes did relatively little.

Architecture Overview  
**1. Base Network**

* LSTM Layers: 3
* Each layer with 40 units, ReLU for activation, return\_sequences=True for the first two layers.
* Total Parameters: 32,681.
* Dropout Layers: 2, dropout rate of 0.2.
* Dense Layer: Single neuron for scalar output.
* Sequence Length: 50 timesteps.
* Batch Size: 32, training; 1, testing, and forecasting.
* Epochs: 5.

2. New Network Architecture

* LSTM Layers: 4
* Layer 1: 64 units, return\_sequences=True.
* Layer 2: 64 units, return\_sequences=True.
* Layer 3: 32 units, return\_sequences=True.
* Layer 4: 32 units, return\_sequences=False.
* Total Parameters: 70,689.
* Dropout Layers: 2, a higher dropout rate of 0.3 for better regularization.
* Dense Layer: Single neuron for scalar output.
* Sequence Length: 100 timesteps.
* Batch Size: 64 for training, 1 for testing and forecasting.
* Epochs: 20.

**Performance Comparison**  
  
**1. Training and Validation Loss**  
In contrast to the original network, this modified network demonstrated a considerably lower training loss over 20 epochs. Probably due to better regularization and more model capacity, the learning curve is smoother; that suggests a better convergence. The initial network resulted in a lower loss decrease and showed an apparent plateau after a few epochs due to its restricted learning capacity.

**Testing Accuracy**With predictions that nearly matched the actual test values, the reconfigured network demonstrated increased generalization. The initial network did well, but its forecasts were more out of line with real patterns, indicating that it had limitations in terms of generalization.  
  
**3. Forecasting Performance**  
Compared to the results from the original network, the revised network had made a smoother and more precise trajectory for 107 time steps ahead. However, due to fewer parameters as well as a shorter sequence input size, the original network gave performance with higher noise in predictions and lower precision.  
  
**4. Overfitting and Stability**  
The increased dropout rates and larger batch size in the redesigned network did not allow it to overfit, which worked well by keeping the performance stable for all datasets.  
The overfitting rate was low in the original network; however, this shallow network with fewer LSTM layers and rather short input sequences was not able to capture the complex temporal dependencies.

**Reasonable Explanation of Results**  
  
The superior performance of the redesigned network is explained as follows:

1. Increased Model Capacity: The network was able to learn more complex patterns in the data by adding an LSTM layer and increasing the number of units in the first two layers.
2. Longer Input Sequence: The rebuilt network could improve forecasting accuracy by modeling longer-term relationships using 100 timesteps as input.
3. Increased Dropout Rates: Regularization using a 0.3 dropout rate decreased network overfitting, therefore giving it a higher generalization capacity.
4. Greater Number of Trained Epochs: Increasing it up to 20 training epochs on this model lets the new network design not be undertrained.

However, the original performed well without these high demands with respect to computational resource needs. Since there exist a number of applications in which this would be a potential concern, the original network may nevertheless still remain useful.  
 **Conclusion**  
The improved training, testing, and forecasting performance of the reconfigured network far exceeded that of its previous iteration. These findings confirm that accuracy in time-series forecasting tasks is improved by increasing model complexity, input sequence length, and training epochs in conjunction with improved regularization. These enhancements have a price of being more processing-intensive, which the application may or may not account for, based on the needs.  
  
**8. PART VII: Project Report  
(1) Introduction: Introduce the project by writing an introduction section.**

SimpleRNN and LSTM are the two most popular architectures in the broad field of time series forecasting with RNNs. It covers the architecture's main problem-solving using vanishing and exploding gradients as well as the construction, training, and assessment of recurrent neural networks for time series data prediction.   
The project covers creation, retraining, and fine-tuning of the networks as their strengths and weaknesses become clearer and gives some suggestions on how the networks might be improved in an application that requires good forecast skills.  
  
**(2) Describe what the student has done in PART II, III, IV, and V in the final project.**Tasks Performed in PART II, III, IV, and V  
PART II: RNN with Sine Wave Data

* + A single-layered SimpleRNN network of 64 neurons and a dense output layer were implemented.
  + The model was trained to predict the next point in a series of sine waves.
  + Performances measured by training loss and accuracy reveal that the network could easily handle basic pattern temporal representations.

**LSTM Neural Network - Part III**

* + Because of some of the weaknesses in the SimpleRNN, like the problem of a vanishing gradient and poor memory retention, a three-layer LSTM network was recommended.
  + Time-series data were used for its experimentation, and the network yielded good generalization and stability compared to SimpleRNN.

**PART IV: LSTM with Time-Series Data**

* + Applied the LSTM network to a real-world time-series dataset, predicting energy consumption in a building.
  + This architecture was composed of three LSTM layers with dropout for regularization and a dense output layer.
  + Results indicated that the model learned the temporal dependencies well and generalized well to unseen data.

**PART V: Neural Network Redesign**  
The model is a redesign of the LSTM architecture, having four layers of neurons of different numbers, an increased input sequence length, and changed dropout rates.  
The efficiency was improved by increasing the training batch size, while the number of epochs was raised to improve the learning. Out of all, the new architecture achieved better accuracy and robustness in forecasts.

(3) List what the student has learned from the models (Simple RNN and LSTM) and what  
he/she has experienced while working with them (build, train, test, and forecast)

**Lessons Learned: Knowledge of Models**

* SimpleRNN is performing well for basic temporal pattern analysis and fail while managing long-term dependencies.
* LSTMs, while gaining excellent and complex patterns, have huge developments in vanishing and expanding gradient issues using their improved Gating mechanisms.  
  Pragmatic Problems
* The exact adjustment in the training of RNN will prevent overfitting or underfitting that may result from poor training of a model.Although greater length in input sequences requires more computational resources, this will pay off significantly when moving to more complex models.

**Perspectives from Prediction:**

A well-designed LSTM model will be able to predict correctly and generalize different datasets concerning the future trend. It has to be regularized by including dropout layers to avoid overfitting and ensure stability in the model’s performance.

**(4) Conclusion section should be written to conclude the project report.**

The experiment demonstrated how RNNs and LSTMs can be used to carry out time-series forecasting. The improved LSTM network greatly improved testing and forecasting performance, with enhancements to avoid vanishing gradients and generalization issues. While significantly more computationally expensive than the baseline, this model offers substantial accuracy gains on complex temporal patterns. These results underscore the importance of hyperparameter tuning and model design in time-series forecasting.