|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AMRITA SCHOOL OF ARTIFICIAL ENGINEERING**  AMRITA VISHWA VIDYAPEETHAM  COIMBATORE - 641 112  **April - 2025**  **B.TECH ARTIFICIAL INTELLIGENCE IN DATA SCIENCE**  **AND MEDICAL ENGINEERING**  Modeling and Predicting Bacterial Growth  Using an Analog Circuit and RNN   |  |  | | --- | --- | | 24AIM113 | Introduction to NN, CNN and GNN | | 24AIM114 | Analog System Design | |

|  |
| --- |
| **AMRITA VISHWA VIDYAPEETHAM**  COIMBATORE - 641 112    BONAFIDE CERTIFICATE  This is to certify that the report entitled **“Modeling and Predicting Bacterial Growth Using an Analog Circuit and RNN”** submitted by:  RAGUL U - CB.AI.U4AIM24036RAMKUMAR R. - CB.AI.U4AIM24033  SHWETHA P. - CB.AI.U4AIM24042  PRAGALYA M - CB.AI.U4AIM24032    for the final project of **2nd semester** in **B.TECH ARTIFICIAL INTELLIGENCE IN DATA SCIENCE AND MEDICAL ENGINEERING** is a Bonafide record of the work carried out at Amrita School of Artificial intelligence, Coimbatore.  *Submittedfor the final evaluation on ... ... ... ... ... ...*  **FACULTY FACULTY** |
| **INTRODUCTION**    Bacterial growth cultures are crucial for diagnosing infections, studying bacterial properties, and developing treatments. They allow scientists to isolate and grow bacteria in a controlled environment, enabling the study of their metabolism, pathogenicity, and antibiotic susceptibility. However, traditional bacterial culturing methods has several drawbacks. These include the risk of contamination, the inability to culture all bacteria (non-culturable organisms), and limitations in mimicking natural growth conditions.Their microscopic size makes it hard to precisely monitor the growth of bacterial cultures. In this project we try to simulate the bacterial growth using components like operational amplifiers (op-amps), resistors, and capacitors. The resulting output voltage from the circuit represents the bacterial population at different time points. By adjusting parameters such as resistance or capacitance, we can simulate different growth rates and environmental conditions. Then we collect the output data from the analog circuit and use it to train a Deep Learning model—specifically, a Long Short-Term Memory (LSTM) neural network. This model is capable of learning temporal patterns in the data and predicting future bacterial growth based on past trends. By combining the real-time simulation and deep learning, our project creates a fast, safe, and intelligent system for modeling and forecasting bacterial behavior under varying conditions.  **LIMITATIONS AND CHALLENGES**  **Choosing the Equation**  Initially, we considered using the Gompertz equation (non-linear ordinary differential equation), a well-known model for cancer growth. However, we ultimately chose the logistic growth model, as it more accurately represents growth that follows a logarithmic pattern and is widely used for modeling population dynamics, including bacterial growth.  **Getting the pre-existing data set for the bacterial growth data.**  For training and evaluating the RNN model, we needed pre-existing data. Since the circuit simulation and hardware components were not yet built, we searched for relevant data. Eventually, we found a research paper that detailed experiments conducted on a specific bacterium under different nutrient levels. The experimental data from this study was published on the “Figshare” platform. After evaluating the dataset, we determined that it was ideal for testing and training our RNN model.    **Finding the right values of Resistors and Capacitors.**  Initially, we were randomly adjusting the values of the capacitor and resistor. However, we later decided to perform theoretical calculations to determine more precise values. This approach led us to a specific set of resistor and capacitor values. To further validate our choices, we used LTSpice to simulate the circuit with different component values before finalizing the design based on our calculated results.  **Getting the analog output from the Arduino UNO (Trying to use it as an ADC)**  Initially, we planned to use the Arduino UNO to capture the analog signal and convert it into digital form. However, this approach introduced significant errors in the output. To address this issue, we switched to the ADALM2000, which features a 12-bit ADC compared to theArduino UNO's 10-bit ADC. The higher resolution of the ADALM2000 provides four times more precision, making it better suited for accurately measuring small voltage changes. Additionally, its built-in tools further enhance data acquisition and analysis, ensuring more reliable results  **Choosing TL081 Op-amp over LM741**  LM741 was already present in the Kit. We chose the TL081 operational amplifier for this project due to its high input impedance, low input bias current, and low offset voltage, which are crucial for accurately modeling bacterial growth in an analog system. The TL081 provides a wide bandwidth and high slew rate, ensuring precise signal amplification without significant distortion. Additionally, its low noise characteristics make it suitable for sensitive measurements, minimizing interference in the analog computations. Compared to general-purpose op-amps like the LM741, the TL081 offers better performance in precision applications, making it ideal for simulating bacterial growth dynamics before feeding the data into a deep learning model.  **The vanishing gradient and the exploding gradient problem**  One of the key challenges in training deep learning models like RNNs and LSTMs is the vanishing and exploding gradient problem. When gradients become too small, the model fails to learn long-term patterns (vanishing gradients), and when they grow too large, training becomes unstable (exploding gradients). To address this in our project, we used LSTM networks with built-in gating mechanisms to manage information flow. Additionally, we applied **gradient clipping** to prevent exploding gradients, used **proper weight initialization**, and tuned hyperparameters like **batch size, learning rate, and dropout** to ensure stable and efficient training.  **Comparision between the models**   | **Aspect** | **LSTM** | **1D CNN** | **GRU** | | --- | --- | --- | --- | | **Model Type** | Long Short-Term Memory (Recurrent Network) | 1D Convolutional Neural Network | Gated Recurrent Unit (Recurrent Network) | | **Primary Strength** | Captures long-term temporal dependencies | Captures local patterns (like slope/shape of population curve) | Efficiently captures temporal dependencies with fewer parameters | | **Sequence Input Handling** | Uses time-step sequences (e.g. 5 values per input) | Same sequence format; treats as local temporal structure | Same input format as LSTM | | **Model Layers** | 1 or more LSTM layers + Dense + Regression | Conv1D (16, 32 filters), MaxPooling, Dense, Regression | 1 GRU layer + Dense + Regression | | **Trainable Parameters** | Higher (due to memory gates in LSTM) | Moderate (no recurrence, just convolution) | Lower than LSTM, fewer gates | | **Noise Handling** | Noisy data for training only | Same (adds small noise to training data only) | Same as LSTM and CNN setup | | **Regularization** | Often dropout or none | Uses dropoutLayer(0.2) after Conv and FC layers | Can use dropout; your code may or may not include it | | **Performance (Test R², RMSE, MAE)** | Good at modeling smooth long-term trends | Strong if growth curve has regular, repetitive structure | Competitive to LSTM, often more stable and faster to train | | **Training Time** | Moderate to slow (due to recurrent nature and backprop through time) | Fast (no recurrence, fully parallelizable convolutions) | Faster than LSTM, slower than CNN | | **Model Complexity** | High (more hyperparameters and internal states to tune) | Medium (simple stacking of filters) | Medium (simpler than LSTM, easier to tune) | | **Best Use Case** | Long-term prediction, time dependencies matter | Local patterns like sudden shifts, bursts, or short-term dynamics | Sequence prediction with speed and similar accuracy as LSTM | | **Evaluation Metrics in Code** | RMSE, MAE, R² for Train/Test | Same as LSTM | Same as LSTM and CNN | | **Prediction Curve Fit (Plot)** | Often smooth, may under/overfit complex bumps | Captures bumps/dips well if local patterns exist | Produces balanced and smooth predictions | | **Ease of Implementation** | Requires careful tuning of hidden size, sequence length, dropout | Simpler to build and tune | Easier than LSTM, fewer hyperparameters |   **Model Performance Metrics (MAE, RMSE, R²)**   1. **Long Short-Term Memory (LSTM) neural network.**   **Screenshot 2025-04-13 at 7.05.16 PM**   1. **LSTM with 1D CNN Layers**   **Screenshot 2025-04-13 at 7.06.05 PM**   1. **GRU (Gated Recurrent Unit)**   **Screenshot 2025-04-13 at 7.10.26 PM**  **Long Short-Term Memory (LSTM)**  LSTM is a specialized type of Recurrent Neural Network (RNN) that is well-suited for modeling long-term dependencies in sequential data. Unlike traditional RNNs, LSTMs address the vanishing gradient problem by using memory cells and three essential gates (input, forget, and output) to control the flow of information.  **Input Gate:** Determines what new information from the current input should be added to the memory cell.  **Forget Gate:** Decides which portion of the previous memory should be discarded, allowing the model to retain only relevant patterns.  **Output Gate:** Determines what information from the cell state should contribute to the final output at each time step.  In our project, our LSTM model predict future **bacterial population growth** based on time-series data generated from an analog circuit. The output data is then normalised (as 0 and 1) and then gaussian noise is added to the training data to make it look like real time data so that the model handels the real time data well.  The architectural structure of the LSTM model is as follows:  **Input Layer:** Accepts a sequence of 5 time steps, each representing a normalized bacterial population value. The input to this layer is a 1-dimensional sequence vector at each time step.  **LSTM Layer:** Consists of 150 memory units with the OutputMode set to 'last', so only the final hidden state (which summarizes the sequence) is passed forward. This layer captures the time dependencies in bacterial growth behavior.  **Fully Connected Layers:** Three dense layers with 100, 50, and 1 units respectively. The intermediate layers are followed by ReLU activation functions to introduce non-linearity, allowing the model to learn complex temporal trends.  **Dropout Layers**: Dropout rates of 20% and 10% are applied after the LSTM and fully connected layers to prevent overfitting. Dropout randomly deactivates a fraction of neurons during training to improve generalization.  **Regression Output Layer:** A final output layer with a single neuron is used to predict the continuous-valued bacterial population at the next time step.  The model is trained using the Adam optimizer with:  Initial learning rate: 0.001  Number of epochs: 150  Mini-batch size: 32  Learning rate drop after 100 epochs  Validation data for early stopping  **Performance and Results:**  The model showed very accurate predictions, with a high R² score (meaning the predictions were very close to the actual population values) and a low RMSE (indicating only small errors between predicted and real values).  This shows that the LSTM model can effectively understand and follow the bacterial growth pattern, even when the data includes random noise or changes caused by varying nutrient levels  **Gated Recurrent Unit (GRU).**  GRU is a type of Recurrent Neural Network (RNN) designed for efficient modeling of sequential data with long-range dependencies. It is an alternate to LSTM with fewer gates while performing making it fastr to train. GRUs merge the input and forget gates into a single update gate, and also use a reset gate, which makes them computationally lighter while still effective at capturing temporal patterns.  Update Gate: Controls how much of the past information needs to be passed along to the future.  Reset Gate: Determines how much of the past information to forget.  In our project, the GRU model is used to predict bacterial population growth over time based on time-series data generated from an analog circuit. The circuit simulates nutrient-limited bacterial proliferation. The raw population data is normalized to a [0,1] range, and slight Gaussian noise is introduced only to the training portion to reflect realistic environmental variability.  The architecture of the GRU-based prediction model is structured as follows:  **1.Input Layer:** Receives a sequence of 5 normalized bacterial population values. Each sequence represents a sliding window of the bacterial growth history.  **2.GRU Layer:** Comprises 120 hidden units with the OutputMode set to 'last', which returns the final hidden state summarizing the entire input sequence. This helps capture the temporal dependencies and growth dynamics in the input series.  **3.Dropout and Fully Connected Layer**s:  A dropout layer (20%) follows the GRU to prevent overfitting.  Two fully connected layers with 60 and 30 units, respectively, each followed by dropout layers (20% and 10%). These layers extract hierarchical features and patterns from the GRU output.  **4.Output Layer:** A fully connected layer with a single unit followed by a regression layer is used to predict the next population value in the sequence.  The model is trained using the Adam optimizer with:  Initial learning rate: 0.001  Number of epochs: 150  Mini-batch size: 32  Learning rate decay (factor of 0.1 after 100 epochs)  Data shuffling at each epoch for better generalization  **Performance and Results:**  The model showed **very accurate predictions**, with a **high R² score** (meaning the predictions were very close to the actual population values) and a **low RMSE** (indicating only small errors between predicted and real values).  This shows that the **LSTM model** can effectively **understand and follow the bacterial growth pattern**, even when the data includes **random noise or changes caused by varying nutrient levels**  **One-Dimensional Convolutional Neural Network (1D CNN)**  A 1D CNN is a type of deep learning model that is highly effective for analyzing sequential data, such as time series. Unlike RNNs (like GRUs or LSTMs), which process sequences step-by-step, CNNs apply filters across the input sequence to detect local patterns. This makes them computationally efficient and particularly good at capturing short-term temporal dependencies.  In this project, a 1D CNN model is used to predict bacterial population growth over time based on time-series data generated from an analog circuit. The circuit simulates bacterial growth under nutrient-limited conditions. To enhance realism, slight Gaussian noise is added only to the training data to mimic environmental fluctuations.  **1D CNN Architecture Overview:**  1. **Input Layer:** Receives a sequence of 5 normalized population values.Each sequence acts as a sliding window over time.  **2.Convolution and Pooling Layers:**  **1st Convolution Layer:** Applies 16 filters of size 3 with padding set to ‘same’. Extracts short-term features from the input sequence.  **ReLU Activation:** Introduces non-linearity.  **Max Pooling Layer:** Reduces the dimensionality by half, helping the model focus on dominant features.  **2nd Convolution Layer:** Applies 32 filters of size 3, again with padding set to ‘same’, enabling deeper pattern extraction.  **ReLU Activation:** Further enhances non-linear feature representation.  **3.Global Pooling and Dropout:**  **Global Max Pooling Layer:** Collapses the temporal dimension, creating a fixed-size feature vector regardless of input length.  **Dropout Layer (20%):** Prevents overfitting by randomly deactivating neurons during training.  Fully Connected Layers:  **First FC Layer:** 50 units with ReLU activation to extract deeper features.  Dropout (20%) to improve generalization.  **Second FC Layer:** Single neuron for final prediction.  **Output Layer:**  **Regression Layer:** Computes the error between predicted and actual values using Mean Squared Error (MSE).  **4.Fully Connected Layers:**  First FC Layer: 50 units with ReLU activation to extract deeper features.  Dropout (20%) to improve generalization.  Second FC Layer: Single neuron for final prediction.  5.**Output Layer:**  Regression Layer: Computes the error between predicted and actual values using Mean Squared Error (MSE).  **Training Details:**  Optimizer: Adam (Adaptive Moment Estimation)  Initial Learning Rate: 0.001  Learning Rate Drop: 0.1 (after 100 epochs)  Epochs: 100  Mini-batch Size: 32  Validation Strategy:  30% of the dataset reserved for testing.  Of the remaining 70%, 80% used for training and 20% for validation.  Early Stopping: Based on validation loss (patience = 25 validation checks)  **Performance and Results:**  After training, the model is tested on a separate held-out portion of the dataset. The predicted population values are de-normalized and plotted against the actual values over time.  The model achieves:  Low RMSE and MAE: Indicating small prediction errors.  High R² value: Suggesting the model accurately explains most of the variation in the actual population data. |