PART I : Learning Process in Neural Networks

A Feedforward Neural Network (FNN) processes data by propagating input through layers of nodes until it reaches the output layer. The learning process typically involves three main steps:

1. **Forward Propagation**: The input data passes through each layer, transforming weight multiplication and activation functions to produce an output. This output is then compared with the actual target (label) to calculate the error using a loss function.
2. **Backward Propagation**: The network calculates the gradients of the loss function concerning each weight using the chain rule of calculus (backpropagation). This step allows the network to understand how each weight contributed to the error.
3. **Weight Update**: Using an optimization algorithm such as gradient descent, the network adjusts its weights to minimize the error. This process iterates over multiple epochs (passes over the training data) to improve the network's predictions iteratively (Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, p. 289-290).

PART II: Sequence Data and Memory

**Question 2.1: Discuss the Special Relationship Between Sequence Data Like Language and Memory**

Sequence data, such as language, depends on context and order, meaning past data points can influence the interpretation of future ones. RNNs are designed to capture these dependencies by maintaining a state that carries information from previous time steps, effectively making them memory enabled. This allows RNNs to capture patterns and contextual information, particularly suitable for language processing and time-series analysis (Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, p. 500).

**Question 2.2: Discuss Why the Recurrent Neural Network (RNN) is a Good Fit for Processing Sequence Data Like Language**

RNNs have an architecture that is designed for sequence processing, in order to keep track of previous inputs, hidden state is maintained. This is what makes them very efficient in the task where current input depends on previous inputs, for example in language modeling and speech recognition. What set RNNs apart from the traditional feedforward networks was their ability to retain context over time, something that is absent in case of normal feedforward network as they have no memory of temporally registered signal. More sophisticated alternatives such as LSTM and GRU cells improve the ability of RNNs to capture long-term dependencies (Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, p. 501).

PART III: Simple RNN Cell and McCulloch-Pitts Model

**Question 3.1: Discuss the Simple RNN Cell Showing That It Is a Version of the McCulloch-Pitts Model Implemented in a Real Artificial Neural Network**

The Simple RNN cell extends the foundational McCulloch-Pitts model, which operates as a binary neuron computing weighted sums of inputs and applying a threshold function. Unlike the static nature of the McCulloch-Pitts model, the Simple RNN cell includes feedback loops and state maintenance, allowing it to model temporal sequences by updating its state based on inputs over time steps (Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, p. 502).

**Question 3.2: Discuss and Prove That the Simple RNN Cell Has Computation Power**

The Simple RNN cell can process sequences by integrating current inputs with previous hidden states. This allows it to capture and model time-dependent patterns, showing its computational power in sequential tasks. However, basic RNNs may face challenges like vanishing gradients over long sequences, which limits their computational depth. More sophisticated architectures such as LSTMs and GRUs address these limitations by introducing mechanisms for long-term memory retention (Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, p. 503).

PART IV: Simple RNN with Sine Wave Data using Keras

* **Network Design Discussion**

A diagram of a diagram

Description automatically generated Implementing our sine wave prediction model utilizes a Sequential architecture in Keras, following best practices in deep learning for time series prediction. The network architecture consists of carefully designed layers to capture the periodic nature of sine waves while maintaining computational efficiency.

The implemented neural network consists of:

1. **Input Layer**
   * Shape: (10, 1) for sequence length of 10
   * Handles normalized sine wave data
2. **SimpleRNN Layer**
   * 64 neurons
   * tanh activation function
   * Maintains temporal information
3. **Dense Output Layer**
   * Single neuron for prediction
   * Linear activation for regression

**Training and Evaluation Results**

The model was trained on sine wave data with the following parameters:

* Training split: 80%
* Validation split: 20%
* Batch size: 32
* Epochs: 50
* Optimizer: Adam
* Loss function: MSE

Results:

64-Unit Model Performance:

==================================================

Test Loss (MSE): 0.000031

Test MAE: 0.004658

A graph of two people

Description automatically generated with medium confidence

A graph with a line

Description automatically generated

PART V: Redesigning the Simple RNN

**Network Redesign Discussion**

1. **Architectural Changes**
   * Increased SimpleRNN neurons to 128
   * Maintained other layer configurations
   * Enhanced memory capacity
2. **Design Rationale**
   * Larger capacity for complex patterns
   * Better feature extraction capability
   * Improved temporal dependency handling
3. **Expected Improvements**
   * Better handling of long-term dependencies
   * More sophisticated pattern recognition
   * Reduced prediction error

**Performance Comparison**

**Model Performance Comparison:**

**==================================================**

64-Unit Model:

Mean Absolute Error: 0.803255

Standard Deviation of Error: 0.580921

128-Unit Model:

Mean Absolute Error: 0.806446

Standard Deviation of Error: 0.583205

Improvement Metrics:

MAE Improvement: -0.40%

Error Stability Improvement: -0.39%

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

**A graph with a line

Description automatically generated**

**3. Performance Insights**

a) MSE and Initial MAE:

* + The 128-unit model shows significant improvement in basic test metrics
  + MSE reduced by 41.9% (from 0.000031 to 0.000018)
  + Initial MAE improved by 23.1% (from 0.004658 to 0.003581)

b) Extended Analysis:

* + Slightly increased mean absolute error (+0.40%)
  + Marginally higher error standard deviation (+0.39%)
  + These differences suggest minimal practical impact of increased capacity

**Conclusion**

The comparative analysis reveals an interesting pattern where the 128-unit model shows better performance on immediate test metrics but slightly worse results in extended analysis. This suggests that for sine wave prediction:

1. The 64-unit architecture provides sufficient capacity
2. Increasing model size offers marginal or no practical benefits
3. The simpler architecture may be more efficient and stable

These findings indicate that architectural efficiency might be more important than raw capacity for this particular task, and future improvements might better focus on optimization techniques rather than increasing model size.

**Reference:**

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media.