**Midterm Assessment ADTA 5560.701 Recurrent Neural Networks for Sequence Data**

**Yog Chaudhary**

11727095

ADTA 5560 Recurrent Neural Networks for Sequence Data

Professor: Dr. Thuan L Nguyen

University of North Texas **Nov 09, 2024**

**3. PART I: Learning Process in Neural Networks (20 Points)**

Discuss the learning process of a neural network like Feedforward Neural Network (FFNN).

FFNN stands for feedforward neural network, which is the basic type of artificial neural network representing the main architecture of neural networks. The process of learning in FFNN could be broadly outlined as follows:

**Initialization:**

* The learning starts with the initialization of weights and bias in a neural network.
* The weights are assigned small random values for symmetry breaking, and one could initialize the bias to zero.

**Forward Propagation:**

* Feed-forward: Input data is provided to the network.
* Multiplying the input values by the weights and adding a bias to them.
* The weighted sum is passed through an activation function to introduce non-linearity.
* This process flows until all hidden layers are covered, till a final output is derived.

**Losses Calculation:**

* The output from the network: There is consideration for a loss or cost function in order to compare the actual target values.
* This difference between the actual and predicted values is quantified through loss.

**Back-propagation:**

* Back-propagation: The major computation for training the network will be the computation of the backpropagation of the gradients w.r.t the weights and biases.
* Gradients are computed using the chain rule of calculus back-propagation, layer by layer, starting from the output.
* These gradients are the direction and step size that each of the parameters should move toward minimizing the loss.

**Gradient Descent:**

* The computed gradients are further utilized in updating the weights and biases in the network.
* Optimizer names are common in the list for this purpose, including Gradient Descent; this would update the parameters in the opposite direction of the gradients.
* Learning rate: The rate or step size taken at each update, which is a hyperparameter.

**Iterative Process:**

* Steps 2 to 5 are iteratively performed over several epochs or iterations through the whole dataset.
* The model increases performance by continuously adjusting the parameters to reduce the loss.

**Testing and Validation:**

* The model is verified on a validation set about the generalization.
* Testing is done on unseen data when satisfactory performance is achieved to check the generalization ability for new examples.

**Hyperparameter Tuning:**

* The fine-tuning of hyperparameters, like learning rate, number of hidden layers, and neurons per layer, plays an important role in the model's performance.

**Regularization:**

* Other regularization techniques that could prevent over-fitting and help generalize better with the model involve using dropout or an algorithm with L2 regularization.

That process of learning would continue until the model converges at a point where the loss is at its minimum; that would imply the FFNN has learned to map input data to desired outputs effectively.

SOURCES: <https://builtin.com/data-science/feedforward-neural-network-intro>

<https://www.turing.com/kb/mathematical-formulation-of-feed-forward-neural-network>

**4. PART II: Sequence Data and Memory (20 Points)**

**2.1: Special relationship between sequence data like language and memory**

Considering sequence data, such as language, there is a particular strong connection with memory, as it naturally would possess temporal structure. That is, in language words are ordered, context and are not independent. Thus, a model that either tries to understand or generate sequences should therefore contain memory able to handle such temporal dependencies.

**Temporal dependency:** These are inherently sequential for any given language, with each word dependent on its predecessor. The meaning of a sentence or paragraph gives the correct setting that individual words should fall under. In this quality lies the peculiarity in handling sequence data.

**Contextual Understanding:** It would amount to memory that in sequence data, what has come earlier in the sequence must be kept in view to understand what is in current view. For example, in a sentence, a correct interpretation of a particular word demands a recall of either a subject or object that was mentioned earlier. This will result in an accurate interpretation.

**Long-Term Dependencies:** In languages, it is often found that most of the dependencies extend over a very long distance. For instance, certain word meanings are driven by their context, sometimes by words coming way ahead in the sequence. Effective language modeling should capture such long-term dependencies. This is obviously related to memory.

**Dynamic Nature:** The sequence data is of a dynamic nature; it should change the model understanding as newer elements are encountered. This dynamic nature further brings to notice the importance of memory that would enable the model to update and refine knowledge given the evolution of context.

SOURCES: <https://towardsdatascience.com/sequence-models-and-recurrent-neural-networks-rnns-62cadeb4f1e1>

**Question 2.2: Recurrent neural network is a good fit for processing sequence data like language**

Recurrent neural networks maintain this special internal memory state, they are relatively more known to handle such sequence data like language. Here is why RNNs will work:

**Sequential Information Processing:** Since RNNs have been designed for sequential data, only one element is processed at any given time, while the internal state keeps track of everything that has been seen so far. Intrinsic excellent adaptability to tasks where the order in which inputs come in makes much difference, say, in language modeling or machine translation.

**Memory:** RNNs have a memory that captures what has happened in the upstream sequence. That same memory enables the network to learn dependencies and context and works perfectly for tasks which require understanding and generating sequences such as sentences.

**Variable-Length Sequences:** RNNs support variable-length sequences, whereby they automatically adjust with respect to the length of input data. Essentially, this feature is very important in language processing tasks, as sentences or documents are of different lengths.

**Contextual Learning:** RNNs learn quite efficiently in contexts from sequences of inputs. The internal state of an RNN can be considered, to some extent, a sort of memory that keeps the information binding the processing of subsequent elements within a sequence.

**Long-term Dependencies:** RNNs learn long-term dependencies of sequences. They are able to consider the relationship between words or events that occur very far apart in a sequence. This is an important aspect of understanding tasks such as language and sentiment.

In particular, the recurrence and memory of RNNs make them very suitable for handling and understanding sequence data such as language**.**

**SOURCES**: <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>

<https://www.ibm.com/topics/recurrent-neural-networks>

**5. PART III: Simple RNN Cell and McCulloch-Pitts Model (20 Points)**

**Question 3.1: Simple RNN Cell showing that it is a version of the McCulloch-Pitts model that is implemented in a real artificial neural network.**

The Simple RNN cell can be looked upon as a modern cousin of the McCulloch-Pitts model, which was among the first models ever proposed for an artificial neuron. Here's how the Simple RNN cell is related to the McCulloch-Pitts :

**Binary Threshold Neuron-McCulloch-Pitts Model:** The M-P model came forward with a binary threshold neuron, wherein the binary input values would be 0 or 1 and the neuron-a binary output-depending on whether the weighted summation of inputs crossed a threshold value.

**Activation Function inside a Simple RNN Cell:** While the simple RNN cell does take the basic idea of the threshold neuron, it does contribute an activation function that is continuous. The simple RNN cell outputs in continuous value through an activation function-largely tanh or sigmoid-instead of a binary output.

**Memory Element:** The biggest novelty in the Simple RNN cell is the presence of a memory element. It maintains some sort of an internal state that could carry information from previous inputs of the sequence. This memory will help the cell bring in the temporal dependencies that were missing in the M-P model.

**Feedback Loop:** The Simple RNN cell has a feedback loop where, after every step, the output could flow into the present computation. It is this feedback that actually makes dynamic behavior across time in Simple RNN and, hence, will make it suitable for processing sequences.

Basically, McCulloch-Pitts gave a starting point for artificial neurons, and the Simple RNN cell added to that, adding a continuous activation function, a memory element, and feedback. Thus, it is a powerful model, really suited for sequential data.

**Question 3.2: Discuss and prove that the Simple RNN cell has computation power.**

We could argue that there is computation capability in the Simple RNN cell as follows:

**Universal Approximation Theorem:** A theorem that says if a network is big enough, comprised of simple nonlinear units, it can approximate any continuous function to any degree of accuracy.

**Simple RNN as a network:** Since a simple RNN cell already includes an activation function and can learn weights, the cell can be considered a nonlinear unit by itself.

**Chaining of RNN Cells:** A number of simple RNN cells can be chained together in order to generate a more complex network. They may process their current inputs but also the information that has been processed previously by other cells in the chain.

**Mathematical Representation of RNN Cell**  
Mathematically, here's what is computed within a simple RNN cell: A black text on a white background

Description automatically generated

The hidden ht carries information through time, providing the model with a memory mechanism that allows it to learn temporal dependencies in the data.

**Recursive Nature:** The RNN cell introduces recurrence by taking its previous output as input. The recurrence thus introduced will be helpful for complicated computation, especially in the case of data in sequences where the present depends on the past.

Putting these together, since a Simple RNN cell is a nonlinear unit and can be interconnected arbitrarily with other such units, it would follow that it can in principle be part of any network which approximates any continuous function. Its recursive nature also lets it be capable of a complex computation over sequences of input. This demonstrates that a Simple RNN cell is computationally powerful.

While the Simple RNN cell is already a powerful tool in the processing of sequential data, a number of natural limitations are arising, including the problem of learning long-term dependencies. Advanced RNN architectures were developed to alleviate some of these limitations while retaining the essential qualities of both memory and sequential processing. Advanced RNNs include Long Short-Term Memory networks.

**Sources:** <https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1> <https://pabloinsente.github.io/the-mcculloch-pitts-artificial-neuron-model>

<https://courses.cs.washington.edu/courses/cse543/22sp/schedule/lecture14.pdf>

**6. PART IV: Simple RNN with Sine Wave Data and Keras (20 Points)**

**Step 1: Architecture Diagram**

**A diagram of a computer

Description automatically generated**

**Step 2: Build and Train the Model (Jupyter Notebook)**

**Step 3: Evaluate the Model**

**Step 4: Report**

**Model Evaluation Report:**

It was developed in Python using the Keras library within the environment of a Jupyter Notebook. The most important steps that have been carried out for building and training the model are **summarized below:**

**Data Preparation**: Further, the sine wave data was split into 70% as training sets and 30% as test sets. The training data underwent normalization through MinMaxScaler, and sequences of 10-time steps each were created to prepare data for RNN input.

Model Architecture: Two major layers that composed the neural network were:

* The SimpleRNN layer consists of 64 units and tahn activation.
* Dense layer with 10 neurons as output.

The model was implemented using the Keras Sequential API. The selected optimizer was Adam, and the loss to be minimized was set as Mean Squared Error-MSE.

**Training:** The model was trained for 50 epochs in total, with the batch size being 16. Here, the dataset has been used for performance checks and also to avoid overfitting. Results of Training: One can see that the training and validation losses decrease as the number of iterations increases for 50 epochs of training; hence, the model learns the sine wave temporal dependencies successfully.

**Evaluate the Model**

Finally, test data was used to evaluate this model. The following are some of the metrics calculated to judge the model's performance:

* MSE: 4.152 - Test Loss
* Test Mean Absolute Error (MAE): 0.0512 (indicative value)

These test loss and MAE values indicate that the model could probably have correctly predicted the values of the sine wave, keeping in mind a low margin of error. A relatively low MAE means the average error in the predictions was quite small.

**Performance Analysis**

Training and validation loss curves converged very well with limited overfitting within bound. The model managed to track the average sinusoidal shape of the test data, reaching both peaks and troughs quite nicely. Some discrepancies in the prediction appeared at every instance of an abrupt change of direction, reflecting a slight deficiency in handling these kinds of abrupt changes; otherwise, the model fared considerably well in capturing the pattern of the sine wave**.**

**Visualization Results**

The plot for Training vs. Validation Loss also showed good convergence: training and validation losses continuously went down, with just a slight increase in the validation loss, hence minimal overfitting. In fact, the plot for True vs. Predicted Values will also show that the model predictions track natural oscillations of the sine wave rather well to capture its periodic nature. Minor amplitude errors have been realized, mostly from sharper bends, as the model showed slight difficulties in catching the abrupt change. The model performance is rather good and would predict most of the pattern of the sine wave while maintaining minor discrepancies over complex curve portions.

**Considerations**

The sine wave data is a periodic function, and success on this may not imply the performance of the model on other, more complicated sequences.

That would be interesting, considering the variability of the model across different datasets in terms of pattern adaptability.

**Conclusion:**

This model is an expert in the narrow task of predicting the next point in some sequence of a sine wave. This success therefore forms the ground for studying other, more complex sequence data, further adapting the model.

**7. PART V: Redesign Simple RNN (20 Points)**

**Step 1: Architecture Diagram**

**A diagram of a computer

Description automatically generated**

Step 2: Redesign the Simple Recurrent Neural Network

Step 3: Evaluate the Redesigned Model on the Test Data

Step 4: Report on Redesigned Simple RNN Model

**Redesign of simple RNN Overview:**

In this redesigned Simple RNN model, the number of neurons in the SimpleRNN layer was changed from 64 to 128. The rest of the structure of the model remained the same, including the Dense output layer, to isolate the effect of adding more neurons. This adjustment increases the model's learning capacity of complex temporal dependencies by enhancing the model's capacity and, consequently, its representational power.

**Potential advantages for reconfigured architectural design:**

Increasing the number of neurons enhances the RNN's ability to capture intricate relationships and dependencies within the time-series data. For periodic data, such as sine waves, a larger SimpleRNN layer can model subtle variations more effectively and maintain longer-term memory over sequences. This helps the model better capture sharp directional changes and high-frequency components.

Therefore, lead to better performance, since the model has more capacity to learn from more complex patterns of the sine wave. Also, the network with more neurons can learn more abstract and fine-grained relationships from the sequential data.

**Results**: The test loss is about 2.832 in the redesigned model, while the test mean absolute error is 0.0431, which has very similar performance compared to the previous model with 64 neurons.

**Conclusion:**

Considering that the testing loss increased comparably to that of the improved model, determination of the number of neurons within the SimpleRNN layer can be regarded as a key decision.