EVALUATION OF PREDICTIVE ALGORITHMS

CS 566 - Analysis of Algorithms

BU MET College

Fall, 2024

Kangjin Wang, Chen-Wei Hsu, and Zhiyuan Zhang

ABSTRACT

The purpose of this paper is to explore the underlying theory of neural networks, focusing on the core concepts of Artificial Neural Networks (ANN) and their working mechanisms. After an in-depth analysis of the operating principles of ANNs, this paper will focus on several neural network structures that excel in time series data analysis, advanced prediction tasks, and regression problems: Recurrent Neural Network (RNN), Long Short-Term Memory Network (RNN), Long Short-Term Memory Network (LSTM) and Gated Recurrent Unit (GRU). Each of these models has its own characteristics and has significant advantages in dealing with complex time-dependent relationships and nonlinear data. The main goal of this paper is to experimentally compare the performance of these three algorithms and analyze their applicability, advantages and disadvantages in different tasks.

CONTENTS

I. Artificial neural networks

An artificial neural network (ANNs) is a computational model inspired by the biological neural networks in the human brain. ANNs are used to solve complex tasks by learning from data. The network is made up of layers of nodes, or "neurons," which are connected by weighted edges. These layers include an input layer, one or more hidden layers, and an output layer. The weights on the connections are adjusted during training to optimize the network's performance.

The operation of each neuron consists of two main steps. First, all input data is weighted and summed, with a bias term added to the result. Next, the combined result is processed through an activation function to produce the final output value. Mathematically, this process is expressed as

$$y = f\left(\sum_{i=1}^{n} \omega_i x_i + b\right),$$

where x_i and w_i are the inputs and weights, b is the bias, and f() is the activation function. The activation function provides the network with non-linear modeling capability, which is essential for capturing complex patterns in data. The choice of activation function plays a crucial role in determining the performance of the network.

Common activation functions include the sigmoid, tanh functions, and ReLU (Rectified Linear Unit), each offering unique advantages depending on the problem at hand. The choice of activation function has a significant impact on the network's ability to learn and generalize from data. Furthermore, by stacking multiple layers of neurons, ANNs can create deep

architectures capable of learning hierarchical patterns, making them powerful tools for tasks such as image recognition, speech processing, and natural language understanding.

The sigmoid function represented as a mathematical equation is as follows:

$$f(x)=\frac{1}{1+e^{-x}},$$

It has excellent mathematical properties, with its derivative given by f(x)(1−f(x)). However, when the input approaches negative infinity, the function value approaches 0. This characteristic makes it prone to causing vanishing gradient issues in neural networks. The tanh function can be seen as an extension of the sigmoid function, achieved by appropriately scaling and shifting the sigmoid function. The mathematical equation is:

$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=2Sigmoid(2x)-1,$$

On the other hand, the ReLU function, which stands for Rectified Linear Unit, is a simple yet effective method for providing non-linear transformations. It is commonly used in hidden layers as it accelerates the training process of neural networks and helps mitigate the vanishing gradient problem. This function is designed to solve issues related to gradient disappearance, with the main idea being to introduce non-linearity into the activation function while keeping the gradient constant in certain regions. As a result, it ensures faster learning, particularly when training deep networks. The mathematical equation is:

$$f(x)=\max(0,x)=\begin{cases}0, x\le 0\\ x, x>0\end{cases},$$

Compared to the tanh function, the ReLU function converges much faster, making it more suitable for deep neural networks. When x>0, the derivative of ReLU is constantly 1, which can effectively avoid the problem of vanishing gradients. However, when x<0, the derivative of ReLU is 0, causing some neurons to stop training.

II. Recurrent neural network

RNNs are a type of artificial neural network where connections between nodes can create cycles, allowing output from some nodes to affect subsequent input to the same nodes. This makes RNNs well-suited for tasks involving sequential data, such as time series analysis, natural language processing, and speech recognition. Additionally, unlike traditional feed-forward neural networks, RNNs have an internal memory which captures information about what has been processed so far, making them powerful for predictive tasks.

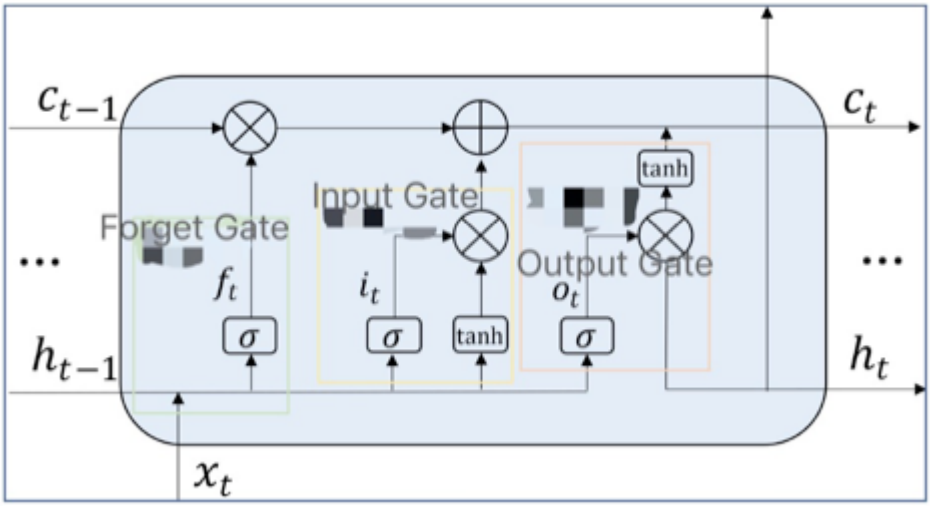
The fundamental concept behind RNNs is their looped structure, which enables the network to retain and utilize information from prior steps in the sequence. This characteristic is crucial for accurately capturing the dependencies within sequential data. At each time step t, the RNN receives an input and the hidden state from the previous time step, producing the current hidden state and an output. This can be represented by the following equations:

$$O_t = g(V \cdot S_t)$$
$$S_t = f(U \cdot X_t + W \cdot S_{t-1})$$

where U, V, and W are weight matrices that the network needs to learn. Functions g and f represent the activation functions for the output layer and hidden layer, respectively. The structure of an RNN enables it to process data elements in sequence, where information from previous steps can impact the outcomes of future steps. Nonetheless, RNNs have certain drawbacks, such as the vanishing and exploding gradient problems, which hinder their ability to learn long-term dependencies effectively. To overcome these challenges, advanced versions like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) incorporate gating mechanisms. These mechanisms control the flow of information, allowing these models to better capture long-term dependencies in sequential data.

III. Long Short-Term Memory Network（LSTM）

The Long Short-Term Memory Network is a variant of RNN, it solves problems appearing in traditional RNN such as vanishing and exploding gradients. LSTM performs well at capturing long-term dependencies, which makes it very suitable for dealing with long-sequence data or jobs needing complex network structures. Indeed, LSTMs are widely used in scenarios such as predictions of short-term passenger flow in urban rail systems
The core part of LSTM is its cell. Three gates are included in the cell: Forget gate, input gate and output gate, those gates control the flow of information and allow the model to decide what information to keep, what to forget and what to output in each step. This structure makes LSTMs dealing with long-term dependency problems in sequential data efficiently. The input of those three gates include the output of last stage ht-1, the current information xt and the previous cell state ct-1. Forget Gate determines to discard which part of the past information. Input Gate determines the new information of the current cell state. Output Gate determines which part of the current cell state is used to output.



LSTM updates its cell state and output based on the following equations:

$$i_t = \sigma(\mathbf{W}_{xi} x_t + \mathbf{W}_{hi} h_{t-1} + \mathbf{W}_{ci} \circ c_{t-1} + b_i),$$

$$c_t = f_t \circ c_{t-1} + i_t \circ tanh(\mathbf{W}_{xc} x_t + \mathbf{W}_{hc} h_{t-1} + b_c),$$

$$f_t = \sigma(\mathbf{W}_{fx} x_t + \mathbf{W}_{fh} h_{t-1} + \mathbf{W}_{cf} \circ c_{t-1} + b_f),$$

$$o_t = \sigma(\mathbf{W}_{xo} x_t + \mathbf{W}_{ho} h_{t-1} + \mathbf{W}_{co} \circ c_t + b_o),$$

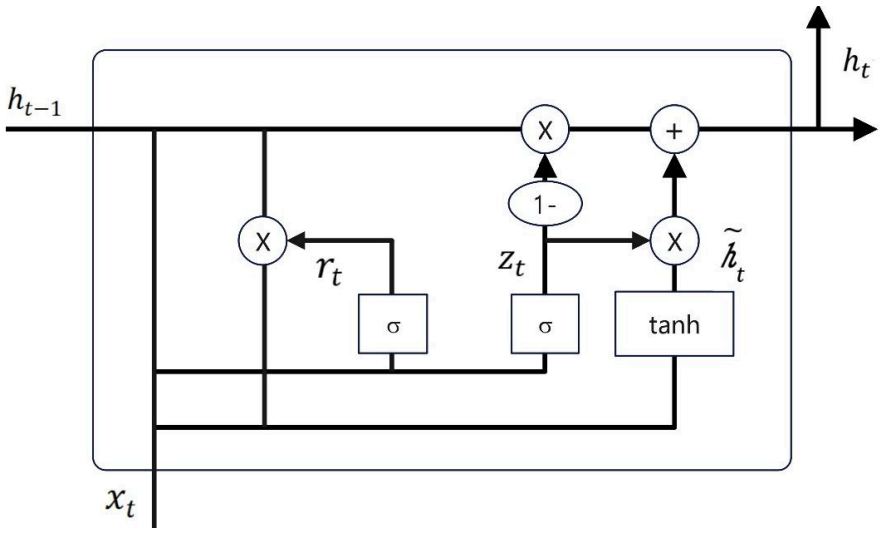
$$h_t = o_t \circ tanh(c_t),$$

W represents weight matrices, b is biases, ⊙ is Hadamard product, σ and tanh are activation functions. These components work together to allow LSTMs to effectively process long-term dependencies in time-series data, especially in jobs such as classification or prediction.

IV. Gated Recurrent Unit (GRU)

As a variant of Long Short-Term Memory Network, the main advantage of Gate Recurrent Unit is the capability to solve problems such as inability to long-term memory and backpropagate gradient. GRU is similar to LSTM on the structure and function of the model, but GRU is more concise compared to LSTM. GRU reduces complexity by using fewer gates and parameters, making it more efficient while still maintaining good performance. GRU is also designed to solve gradient-related issues and can handle long-term dependencies effectively. GRU has two main gates: Update Gate and Reset Gate. Update Gate helps in deciding how much previous information should be kept, while Reset Gate decides how to combine the new input with the previous memory information.

Because of the simplification of structure, GRU has similar even better performance than LSTM in many tasks especially in the environment of limited resources. For instance, on a small dataset, because GRU has less parameters and less training time, the time of convergence is faster. Besides, GRU has better memory ability under some circumstances on the front end of sequence information. This ability is especially crucial for the applications that are sensitive to information about the beginning of a sequence, such as voice detected and tasks in natural language processing. Here's the basic structure of GRU:



In this structure, the input xt and the hidden state ht-1 of the last state is the key input information. The Reset gate rt calculates through application sigmoid function σ to the linear combination of xt and ht-1, the function of the gate is to make sure how much information to keep from the last hidden state. Here's the equations:

$$r_t=\sigma(W_r \cdot [h_{t-1},\ x_t]+b_r)$$

br is the weight of the Reset gate, br is the bias term. Update gate zt calculate in the similar way, it decides keeping how much information from the past and input how much new information from the current hidden state. Here's the equations:

$$z_t=\sigma(W_z \cdot [h_{t-1},\ x_t]+b_z)$$

Wz here is the weight of Update gate, bz is the bias term. Then calculate the new candidate hidden state ht, this involves another linear combination of ht-1 and xt, and apply tanh function. The new candidate hidden state consider the function of Reset gate, the equation is:

$$\tilde{h}_t =| tanh\ (W \cdot [r_t * h_{t-1}, x_t] + b)$$

W and b are according to weight and bias. The last hidden state ht is calculated by the combination of the last hidden state information to the new candidate hidden state. It allows the model to decide how much past information to keep and how much to input efficiently. This mechanism makes GRU able to capture information and prevent vanishing gradients. The equation is:

$$h_t = (1 - z_t) * \tilde{h}_t + z_t * h_{t-1}$$

V. Comparison of RNN, LSTM and GRU algorithm

This paper compares the LSTM model with several other prediction algorithms, including Gated Recurrent Unit (GRU) and Recurrent Neural Network (RNN), and analyzes their performances in the prediction of time-series data.LSTM is a mainstream choice for many current prediction tasks, due to its unique design of the memory unit, which demonstrates significant advantages in capturing long time LSTM, with its unique design of memory units, has shown significant advantages in capturing long-time dependencies and processing complex time series data, and is currently the mainstream choice for many prediction tasks. In the experiments, each client uses the global model to make independent predictions, and the experimental results show that the LSTM model performs better than GRUs and RNNs in handling long data series, multi-step prediction, and recognizing dynamically changing patterns, especially in complex and long time-series data, and its prediction accuracy and robustness are better than those of GRUs and RNNs. In contrast, GRU, due to its relatively simplified structure, can achieve prediction results close to those of LSTM in some specific contexts, but is slightly less effective in dealing with longer time-dependent or nonlinear data. Traditional RNNs, on the other hand, are more suitable for short time series or simple pattern prediction tasks due to the gradient vanishing problem, which limits their performance in long time dependency modeling. The experimental results fully demonstrate the wide applicability and excellent performance of LSTM in time series data analysis, while GRU and RNN have some utility in specific contexts, but it is difficult to surpass the performance of LSTM in complex prediction tasks.

VI. Implementation of Recurrent neural network

```
def split_sequence(sequence, n_steps):
   X, Y = list(), list()
   for i in range(len(sequence)):
      end_ix = i + n_steps
      if end_ix > len(sequence)-1:
```

```
        break
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    Y.append(seq_y)
  return np.array(X), np.array(Y)
```

VII. Implementation of Long Short-Term Memory Network

```
def build_model():
  model = Sequential()
  model.add(LSTM(128, return_sequences=True,
activation='relu',input_shape=(3, feature_dim)))
  model.add(LSTM(units=64, activation='relu', return_sequences=True))
  model.add(LSTM(units=16, activation='relu', return_sequences=False))
  model.add(Dense(1))
  model.compile(optimizer='adam', loss='mse')
  return model
```

VIII. Implementation of Gated Recurrent Unit

```
def split_sequence(sequence, n_steps):
  X, Y = list(), list()
  for i in range(len(sequence)):
    # find the end of this pattern
    end_ix = i + n_steps
    # check if we are beyond the sequence
    if end_ix > len(sequence)-1:
      break
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    Y.append(seq_y)
  return np.array(X), np.array(Y)
```

```
def normalize_list(input_list):
    max_val = max(input_list)
    min_val = min(input_list)
    normalized_list = [(x - min_val) / (max_val - min_val) for x in
input_list]
    return normalized_list
```

Reference:

Abumohsen, M., Owda, A. Y., & Owda, M. (2023, February 27). *Electrical load forecasting using LSTM, GRU, and RNN algorithms.* MDPI. https://doi.org/10.3390/en16052283

Agatonovic-Kustrin, S., & Beresford, R. (2000). Basic Concepts of Artificial Neural Network (ANN) modeling and its application in Pharmaceutical Research. *Journal of Pharmaceutical and Biomedical Analysis*, *22*(5), 717–727. https://doi.org/10.1016/s0731-7085(99)00272-1

Al-Selwi, S. M., Hassan, M. F., Abdulkadir, S. J., Muneer, A., Sumiea, E. H., Alqushaibi, A., & Ragab, M. G. (2024). RNN-LSTM: From applications to modeling techniques and beyond—systematic review. *Journal of King Saud University - Computer and Information Sciences*, *36*(5), 102068. https://doi.org/10.1016/j.jksuci.2024.102068

Han, S.-H., Kim, K. W., Kim, S., & Youn, Y. C. (2018). Artificial Neural Network: Understanding the basic concepts without mathematics. *Dementia and Neurocognitive Disorders*, *17*(3), 83. https://doi.org/10.12779/dnd.2018.17.3.83

Yadav, D. C., & Pal, S. (2022). Measure the superior functionality of machine intelligence in Brain tumor disease prediction. *Artificial Intelligence-Based Brain-Computer Interface*, 353–368. https://doi.org/10.1016/b978-0-323-91197-9.00005-9

Yi, D., Bu, S., & Kim, I. (2019). An enhanced algorithm of RNN using trend in time-series. *Symmetry*, 11(7), 912. https://doi.org/10.3390/sym11070912

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv preprint arXiv:1412.3555. https://arxiv.org/abs/1412.3555

Pudikov, A., & Brovko, A. (2020). Comparison of LSTM and GRU Recurrent Neural Network Architectures. In Recent Research in Control Engineering and Decision Making (pp. 114–124). Springer. https://doi.org/10.1007/978-3-030-65283-8_10

Ghojogh, B., & Ghodsi, A. (2023). Recurrent Neural Networks and Long Short-Term Memory Networks: Tutorial and Survey. arXiv preprint arXiv:2304.11461. https://arxiv.org/abs/2304.11461