electronics

Article

Light Recurrent Unit: Towards an Interpretable Recurrent Neural Network for Modeling Long-Range Dependency

Hong Ye [1], Yibing Zhang [1], Huizhou Liu [2,*], Xuannong Li [3], Jiaming Chang [1] and Hui Zheng [1,*]

[1] School of Internet, Anhui University, Hefei 230039, China; 20070@ahu.edu.cn (H.Y.); y22301037@stu.ahu.edu.cn (Y.Z.); y02014454@stu.ahu.edu.cn (J.C.)
[2] State Grid Anhui Electric Power Co., Ltd., Hefei 230041, China
[3] State Grid Hefei County Electric Power Supply Company, Hefei 231200, China; 19810692337@163.com
[*] Correspondence: 18756027866@163.com (H.L.); huizheng@ahu.edu.cn (H.Z.)

Abstract: Recurrent neural networks (RNNs) play a pivotal role in natural language processing and computer vision. Long short-term memory (LSTM), as one of the most representative RNNs, is built upon relatively complex architecture with an excessive number of parameters, which results in large storage, high training cost, and lousy interpretability. In this paper, we propose a lightweight network called Light Recurrent Unit (LRU). On the one hand, we designed an accessible gate structure, which has high interpretability and addresses the issue of gradient disappearance. On the other hand, we introduce the Stack Recurrent Cell (SRC) structure to modify the activation function, which not only expedites convergence rates but also enhances the interpretability of the network. Experimental results show that our proposed LRU has the advantages of fewer parameters, strong interpretability, and effective modeling ability for variable length sequences on several datasets. Consequently, LRU could be a promising alternative to traditional RNN models in real-time applications with space or time constraints, potentially reducing storage and training costs while maintaining high performance.

Keywords: interpretability; lightweight; long-range dependency; recurrent neural network

check for updates

Citation: Ye, H.; Zhang, Y.; Liu, H.; Li, X.; Chang, J.; Zheng, H. Light Recurrent Unit: Towards an Interpretable Recurrent Neural Network for Modeling Long-Range Dependency. *Electronics* **2024**, *13*, 3204. https://doi.org/10.3390/electronics13163204

Academic Editors: Ping-Feng Pai and Yingke Chen

Received: 3 July 2024
Revised: 23 July 2024
Accepted: 9 August 2024
Published: 13 August 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the development of deep learning, recurrent neural retworks (RNNs), as one of the most representative deep neural networks (DNNs), have been widely used in various fields, including mechanical fault diagnosis, emotion analysis, and stock forest [1–3].

Although RNNs theoretically can capture information from variable-length sequences, their performance in practical applications is often suboptimal due to the problems of gradient vanishing and explosion [4,5]. Specifically, during the training process, the gradients of weights tend to decay or grow rapidly in the back-propagation steps, resulting in instability in updating network weights and hindering the ability of the RNN to model long-term dependencies [6,7]. While the problem of exploding gradients can be tackled with a simple clipping strategy [8,9], there is no easy way to adequately address vanishing gradients in a vanilla RNN.

One of the most popular alternatives to the vanilla RNN is long short-term memory (LSTM) [10,11], which addresses the gradient vanishing problem in the training process. The core advantage of LSTM is that the hidden state is updated by superposition of multiple component "gates" instead of using transfer operators such as matrix multiplication. Although LSTM has improved the limitations of RNN to some extent, there are still some problems. Concretely, LSTM significantly increases the model's complexity, making the model not only inefficient in training but also difficult to interpret [12]. These problems also exist in another RNN variant, namely the Gated Recurrent Unit (GRU) model [13].

As such, to address the limitations inherent in LSTM and RNNs, in this work, we propose a Light Recurrent Unit (LRU) model, which offers a compact structure coupled with enhanced interpretability. The design philosophy of the Light Recurrent Unit (LRU) aims to balance model performance and computational efficiency. It employs a compact structure, reducing the number of gating units to lower the network parameters and computational complexity while maintaining accuracy in processing long sequences. This simplification not only makes it easier to track hidden state changes across successive time steps but also enhances model interpretability. In addition, the activation function is modified to accelerate the convergence of the training process. In the meantime, this modification enhances the interpretability of the function and improves the memory capacity of RNNs for capturing long-term dependency information. Through these innovations, the LRU can retain long-term memory effectively, thereby better handling various tasks involving long sequences.

Moreover, the design of the LRU theoretically suits environments with limited resources. Due to its simplified structure and reduced computational requirements, the LRU has the potential for deployment on platforms with constrained computational resources, such as mobile devices and embedded systems. This design takes into account the typically limited computational power and battery life of these devices, necessitating efficient algorithms to handle complex tasks. By reducing the number of gating units, the LRU decreases computational load and memory demands, theoretically exhibiting superior performance in such environments.

For example, the study by Zhang et al. [14] presents a low-cost, low-power, and privacy-preserving facial expression recognition system based on edge computing, evaluating four deep learning algorithms, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and deep neural networks (DNNs). This demonstrates the practical application potential of lightweight RNN models in resource-constrained environments. Similarly, the study by Al-Nader et al. [15] proposes a novel scheduling algorithm for improving the performance of multi-objective, safety-critical wireless sensor networks using long short-term memory (LSTM), further supporting the application potential of our model in these specific settings. Future research can further validate the performance of the LRU in these specific settings.

The main contributions of this paper can be summarized as follows.

- The proposed LRU introduces a latent hidden state that is highly interpretable. It has a minimal number of gates in all possible gated recurrent structures: only one gate to control whether the past memories should be kept or not, so that the requirement for training data, model tuning, and training time can be reduced, while at the same time, the model accuracy is maintained.
- The proposed LRU leverages the Stack Recurrent Cells (SRCs) to modify the activation function, consequently improving the gradient flow in deep networks. This modification leads to accelerated convergence rates of the network and enhances the interpretability of the model learning parameter.
- Experimental results on various tasks demonstrate that LRU can keep long-term memory to better process long sequences. Despite reduced model complexity, LRU has overall better accuracy as well as faster convergence speed compared to LSTM.

The remaining sections are organized as follows. Section 2 discusses related work on representative RNN variants. Section 3 demonstrates the design of the proposed LRU model. Section 4 shows that LRU obtains competitive or even better performance compared to previous RNN models on various tasks that require long-term dependencies. Section 5 draws conclusions.

2. Background for RNN

In recent times, self-attention-based models, including the Transformer architecture [16] and its derivatives, have excelled in numerous tasks. For example, Huang et al. [17] examined the relationship between peer feedback classified by deep learning and online learning burnout, demonstrating the potential of deep learning techniques in educational settings. Zheng et al. [18] proposed a modified Transformer architecture based on relative position encoding, which showed outstanding performance across various tasks [18]. However, RNNs have significantly fewer parameters and computational demands compared to Transformer-based models, making them still very common in many applications.

By assigning additional weights to the network graph, RNNs create a loop mechanism within the network that allows the network to explicitly learn and utilize the context information of the input sequence, and they are therefore well-suited for processing tasks involving sequence input. RNN architectures have made an enormous improvement in a wide range of machine learning problems with sequential input involved [19–21].

In the widespread application of RNNs, particularly in tasks like text classification and sentiment analysis that require learning long-term dependencies, optimizing RNN parameters remains challenging. The primary reason is the "vanishing gradient" problem, which hampers learning long-term dependencies. RNNs rely on temporal unfolding to fit and predict time series data, updating parameters based on multiple time steps. However, the vanishing gradient problem limits this unfolding, causing updates to be influenced only by recent time steps and not capturing distant historical information. As the temporal distance of dependencies increases, the difficulty of training RNNs also rises. In response to these challenges, researchers have explored various approaches. The following sections highlight some of the prominent directions in current RNN research aimed at addressing these issues.

2.1. RNN with Special Initialization

Some researchers have attempted to capture long-term dependencies in simple, non-gated RNN through better weight initialization. Le et al. [22] proposed IRNN, which uses an identity matrix to initialize the recurrent weight matrix. The critical innovation in IRNN is to produce near-identity projection at hidden states. However, this model is reported to be fragile to hyperparameter settings and fails easily in training [23–25]. Talathi et al. [26] proposed np-RNN based on IRNN. Their np-RNN adds a stronger constraint on initial recurrent weights by forcing the recurrent weights to be a normalized-positive definite matrix, with all except the highest eigenvalue less than one. While these models help to ease the gradient vanishing problem at the beginning of training, they cannot completely avoid the issue throughout the entire training process.

2.2. RNN with Structure Constraints

Another direction for addressing the gradient vanishing problem in RNN is to add certain constraints to the model structure [27]. Mikolov et al. [28] proposed the Structure Constraint Recurrent Network (SCRN), which forces a diagonal block of the recurrent matrix to be equal to a reweighted identity throughout the training. They declare that the reweighted identity block in the recurrent matrix changes their state slowly, which helps the entire network capture a longer history. Hu et al. [29] analyzed the gating mechanisms in LSTMs and proposed a structure called the Recurrent Identity Network (RIN), which adds an extra identity map projection to the hidden layer. These models, however, cannot efficiently improve the model performance, especially when compared with gated RNNs.

3. Proposed Model

In this section, we first introduce the traditional RNN and LSTM models and point out existing issues. Secondly, we introduce the proposed LRU model. Compared with traditional models, the contextual information stored in the hidden layer of LRU can be transmitted over a longer distance in the time domain, and we further modify the activation function to achieve faster model convergence and improved interpretability.

3.1. RNN and LSTM

As shown in Figure 1a, the state update in a basic RNN can be described as follows:

$$\mathbf{h}_t = \delta(U\mathbf{h}_{t-1} + W\mathbf{x}_t + b), \tag{1}$$

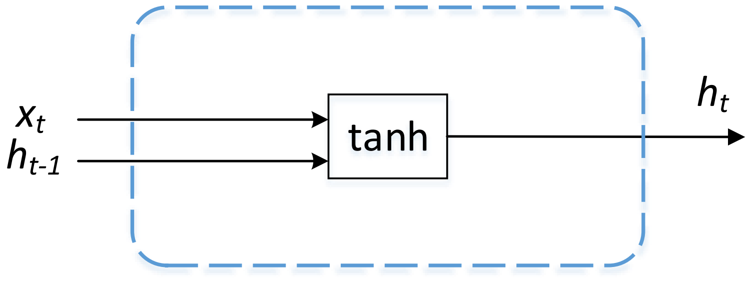
where index t indicates the current position in the input sequence, $\mathbf{x}_t$ and $\mathbf{h}_t$ are the input and hidden state at time t, U and W denote the parameter matrices related to $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$, respectively, b is a bias term, and δ is an element-wise activation function that applies to the hidden states. The term *recurrent* in RNN outlines that the last hidden state vector is recurrently fed back to the input to compute the next state.

Gated RNNs, especially LSTM and its variants, address the gradient vanishing problem in RNNs mainly by introducing component-wise gates to control the information flow within the network. The proposed LSTM model in [10] differs from the original RNN in three aspects.

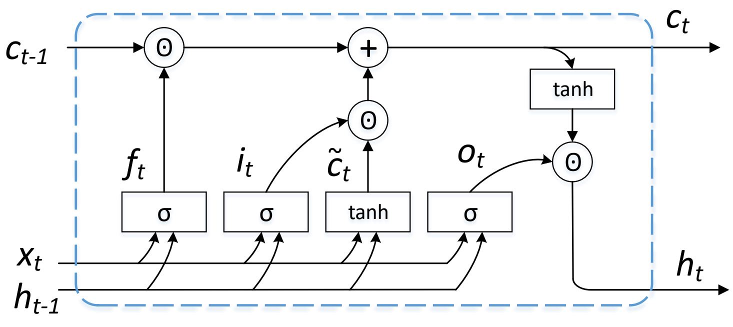
(1) There is one more state vector $\mathbf{c}_t$ in the hidden layer in addition to $\mathbf{h}_t$. Also, $\mathbf{c}_t$ is designed as a "concealed" state that maintains long-term memories, while $\mathbf{h}_t$ maintains short-term memories. Further, $\mathbf{c}_t$ is also called the Constant Error Carousel (CEC), since it is updated additively instead of by using a matrix operation. This design ensures that the gradient flow retains stable in updating $\mathbf{c}_t$.

(2) An input gate $\mathbf{i}_t$ is applied to the input to determine which part of new information can be added to $\mathbf{c}_t$ at time t. The inputs are transformed as the update $\tilde{\mathbf{c}}_t$. Further, $\tilde{\mathbf{c}}_t$ and $\mathbf{c}_{t-1}$ together form the new cell state $\mathbf{c}_t$.

(3) An output gate $\mathbf{o}_t$ is added to control which part of $\mathbf{c}_t$ should be output as the hidden state $\mathbf{h}_t$.



(**a**)



(**b**)

Figure 1. Data flow and operations in various RNN models, including basic recurrent neural network (**a**) and long short-term memory RNN (**b**).

In [30], a forget gate $\mathbf{f}_t$ was first applied to $\mathbf{c}_t$ to determine which part of the old memory should be kept, before adding new information to $\mathbf{c}_t$. This new LSTM structure with a forget gate has been widely applied thereafter. The overall structure of this three-gated LSTM is illustrated in Figure 1b, and the update rules are as follows:

$$\begin{aligned}
\mathbf{i}_t &= \delta(U_i\mathbf{h}_{t-1} + W_i\mathbf{x}_t + b_i),\\
\mathbf{f}_t &= \delta(U_f\mathbf{h}_{t-1} + W_f\mathbf{x}_t + b_f),\\
\mathbf{o}_t &= \delta(U_o\mathbf{h}_{t-1} + W_o\mathbf{x}_t + b_o),\\
\tilde{\mathbf{c}_t} &= tanh(U_c\mathbf{h}_{t-1} + W_c\mathbf{x}_t + b_c),\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}_t},\\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t).
\end{aligned} \tag{2}$$

In Equation (2), W_j and U_j $(j = i, f, o, c)$ are parameter matrices. The activation δ for the gates usually uses logistic sigmoid to constrain the value of gates within $[0, 1]$. For each dimension of information, a gate of value 0 means that the gate is "closed", and 1 means that the gate is "fully open". For example, i_t of value 0 forces the model to discard any input information in time t, and $\mathbf{f}_t$ of value 1 allows the model to keep all previous memories. The value of each gate is determined during training by the current input and hidden information. A tanh nonlinearity is applied when computing $\tilde{\mathbf{c}_t}$ and $\mathbf{h}_t$, while $\odot$ indicates the element-wise product.

When LSTM was first proposed, researchers attempted to increase the model's complexity for better performance. For example, recent studies have explored adding "peephole" connections between $\mathbf{c}_t$and the three gates ($\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{o}_t$), allowing the cell state to influence the gates more directly. While these modifications have shown performance improvements in some applications, they can also introduce additional complexity and may not always be effective [9,31]. On the other hand, some gates have been empirically found to be less effective than others [32,33].

Recently, researchers have started to investigate the redundancy within the LSTM structure [32,34–36]. Among these models, the most representative one is the Gated Recurrent Unit (GRU) [33], which lowers the number of gates in LSTM by removing the output gate. Although GRU has fewer trainable weights than LSTM (about 3/4), it still achieves similar performance on various tasks [32,37]. This phenomenon rests with the fact that, among the three gates in LSTM, the forget gate is essential [6,33,38], and the effect of the input and output gates is less obvious [32]. We argue that it is possible to design a simplified gated RNN model that further lowers complexity while maintaining accuracy.

3.2. Proposed LRU

This section describes in detail the design of the proposed Light Recurrent Unit (LRU). Our motivation is to present an RNN model with both an accessible structure and high interpretability. The overall structure and data flow of LRU are illustrated in Figure 2. At any time step t, LRU takes as input x_t and updates its hidden state h_t as follows:

$$\begin{aligned}
\tilde{\mathbf{h}_t} &= tanh(W_h\mathbf{x}_t),\\
\mathbf{f}_t &= \delta(U_f\mathbf{h}_{t-1} + W_f\mathbf{x}_t + b_f),\\
\mathbf{h}_t &= (1 - \mathbf{f}_t) \odot \mathbf{h}_{t-1} + \mathbf{f}_t \odot \tilde{\mathbf{h}_t}.
\end{aligned} \tag{3}$$

Here, $\tilde{\mathbf{h}_t}$ is a transformed input that adds information to $\mathbf{h}_t$, and $\mathbf{f}_t$ is a vector that controls both:

- The portion that is remembered in the last state h_{t-1}, for each component;
- The portion that is added to h_t from $\tilde{\mathbf{h}_t}$ for each component.

Therefore, $\mathbf{f}_t$ couples the input and forget gates in LSTM by specifying that

$$\mathbf{i}_t = 1 - \mathbf{f}_t, \forall t. \tag{4}$$

Compared with the update vector $\tilde{\mathbf{c}}_t$ that has similar functionality in LSTM, $\tilde{\mathbf{h}}_t$ omits the influence from the last hidden state. This change makes it easier to track the changes in hidden states in consecutive time steps, as will be further discussed in Section 3.4.

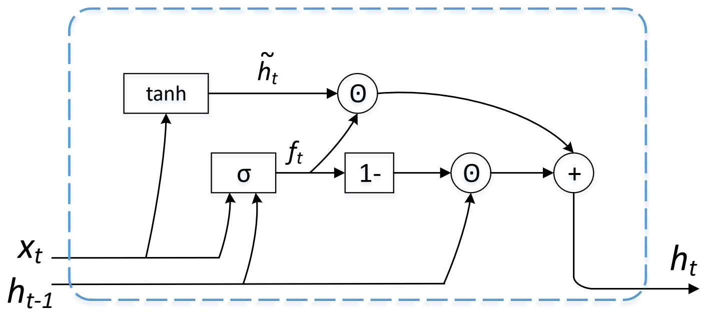


Figure 2. Data flow and operations in the proposed LRU (Light Recurrent Unit).

3.3. Stack Recurrent Cells

Similar to other RNN variants, LRU can be stacked with multiple recurrent layers to improve its memorization capacity. This can be achieved by feeding the output vector of the previous layer as the input to the next layer. Formally, considering an L-layered LRU, the computation at time step t is specified by the following equations:

$$\mathbf{h}_t^{(0)} = \mathbf{x}_t, \tag{5}$$

For $l = 1, 2, \ldots, L$

$$\begin{aligned}\tilde{\mathbf{h}}_t^{(l)} &= tanh(W_h^{(l)}\mathbf{h}_t^{(l-1)}),\\ \mathbf{f}_t^{(l)} &= \delta(U_f^{(l)}\mathbf{h}_{t-1}^{(l)} + W_f^{(l)}\mathbf{x}_t^{(l)} + b_f^{(l)}),\\ \mathbf{h}_t^{(l)} &= (1 - \mathbf{f}_t^{(l)}) \odot \mathbf{h}_{t-1}^{(l)} + \mathbf{f}_t^{(l)} \odot \tilde{\mathbf{h}}_t^{(l)}.\end{aligned} \tag{6}$$

The highway network [39] has been proven to improve the gradient flow in deep networks such as CNN. It uses a skip connection that directly links the hidden state to the input, allowing the gradient to propagate to the previous layer directly. This component can be applied in a stacked LRU by specifying that, for $l \leq 2$, $W_h^{(l)} = \mathbf{I}$, and *tanh* activation is replaced with identity mapping, such that:

$$\tilde{\mathbf{h}}_t^{(l)} = \mathbf{h}_t^{(l-1)}, for\ l \geq 2. \tag{7}$$

The following pseudocode in Algorithm 1 outlines this process, demonstrating the step-by-step procedure involved in computing the relationship as described in Equation (6).

Description of the main variables:

- x_t: Input sequence.
- $h_t^{(l)}$: Hidden state of layer l at time step t.
- $\tilde{h}_t^{(l)}$: Candidate hidden state of layer l at time step t.
- $f_t^{(l)}$: Forget gate of layer l at time step t.
- $W_h^{(l)}$: Weight matrix for candidate hidden state of layer l.
- $U_f^{(l)}$: Weight matrix for forget gate of layer l.
- $W_f^{(l)}$: Weight matrix for forget gate input x_t of layer l.
- $b_f^{(l)}$: Bias for forget gate of layer l.
- δ: Activation function (sigmoid).
- $tanh$: Activation function (hyperbolic tangent).
- $\odot$: Element-wise multiplication.

Algorithm 1 Computation of an L-layered Light Recurrent Unit (LRU)

```
1:  Input: Input sequence $x_t$
2:  Output: Hidden states $h_t^{(l)}$ for all layers
3:  Step 1: Initialization
4:  Initialize the hidden state for the first layer: $h_t^{(0)} = x_t$
5:  Step 2: Computation for each layer
6:  for $l = 1, 2, \ldots, L$ do
7:      Compute the candidate hidden state:
            $\tilde{h}_t^{(l)} = tanh(W_h^{(l)} h_t^{(l-1)})$
8:      Compute the forget gate:
            $f_t^{(l)} = \delta(U_f^{(l)} h_{t-1}^{(l)} + W_f^{(l)} x_t + b_f^{(l)})$
9:      Update the hidden state:
            $h_t^{(l)} = (1 - f_t^{(l)}) \odot h_{t-1}^{(l)} + f_t^{(l)} \odot \tilde{h}_t^{(l)}$
10: end for
11: Step 3: Output
12: Return the hidden states $h_t^{(l)}$ for all layers
```

3.4. Analysis

In this section, the properties and behavior of the proposed LRU are discussed.

First, LRU can be regarded as a gated RNN derived from LSTM, but with many fewer parameters. This property makes the learning process faster and less vulnerable to overfitting. Assuming the input vector dimension is m and the hidden state dimension is n, LSTM has four sets of parameters that determine $\mathbf{f}$, $\mathbf{i}$, $\mathbf{o}$, and $\tilde{\mathbf{c}}$, resulting in a parameter number of $4 \times m \times n + 4 \times n \times n$ (bias term omitted for simplicity). Meanwhile, LRU only has two sets of parameters, one for calculating the forget gate $\mathbf{f}$, the other for $\tilde{\mathbf{h}}$. The total parameter number in LRU is only $2 \times m \times n + n \times n$. In the case where $m = n$, the parameter number of LRU is 37% of LSTM; in the case of $m \ll n$, LRU has only 25% the parameter size of that of LSTM. In short, given the fact that LRU has only one gate and no recurrent connections, LRU can be regarded as a minimal design of any gated RNN units. Despite the simplicity, the utilization of the forget gate allows LRU to process sequence learning without suffering from the gradient vanishing problem, as demonstrated later in the experimental section on various tasks.

Second, the removal of recurrent connections in LRU also allows us to describe the learned model in a quite straightforward perspective. Since the hidden state in LRU is updated in an additive, non-recurrent fashion, at each step, the hidden state can be regarded as a weighted average of all previous inputs in the same layer. Formally,

$$\begin{aligned} \mathbf{h}_t^{(l)} &= (1 - \mathbf{f}_t^{(l)}) \odot \mathbf{h}_{t-1}^{(l)} + \mathbf{f}_t^{(l)} \odot \tilde{\mathbf{h}_t}^{(l)} \\ &= \sum_{i=1}^{t} \mathbf{w}_{i,t}{}^{(l)} \odot \tilde{\mathbf{h}_i}^{(l)}, \end{aligned} \tag{8}$$

$$\mathbf{w}_{i,t}{}^{(l)} = \prod_{k=i+1}^{t} \mathbf{f}_k^{(l)} \odot (1 - \mathbf{f}_i^{(l)}). \tag{9}$$

Therefore, the hidden state at time t can be tracked back to all previous inputs in the same layer, with assigned weights indicating the inputs' relative importance. It is easy to prove that all $\mathbf{w}_{i,t}{}^{(l)}$ sum up to 1 for each l. Therefore, $\mathbf{h}_t^{(l)}$ can be viewed as a weighted average of previous inputs. It should be noted that these weights are also component-wise vectors, allowing us to analyze the behavior of each neuron of the hidden state. This property can also be regarded as a soft attention mechanism, as will be shown in Section 4.2.

Despite the simplicity, the utilization of the forget gate allows LRU to process sequence learning without suffering from the gradient vanishing problem. In traditional RNNs, the gradient vanishing phenomenon during training primarily arises from two sources. Firstly, the repeated multiplication of the hidden state weight matrix causes the gradient to be suppressed in most positions, a problem that becomes particularly pronounced during long sequence training, ultimately leading to gradient vanishing. Secondly, commonly used activation functions such as sigmoid and *tanh* lead to an overall scale reduction in gradient back-propagation, further exacerbating the gradient vanishing problem. The LRU effectively mitigates the gradient vanishing issues associated with traditional RNNs by introducing a simplified gate structure and a direct candidate hidden state update mechanism. As demonstrated later in the experimental section, across various tasks, the LRU excels in sequence learning tasks.

4. Experiments

In this section, the performance of the proposed LRU is tested on various tasks, including the adding problem in Section 4.1, pixel-wise MNIST handwritten digit classification in Section 4.2, and word-level language modeling in Section 4.3. These tasks cover a broad range of application domains and sequence lengths. The codes for training and evaluating the models are written in Pytorch. Experiments are performed on a server equipped with 8 NVIDIA Tesla K40m GPUs (NVIDIA Corporation, Santa Clara, CA, USA), dual Intel Xeon E5-2620 (Intel Corporation, Santa Clara, CA, USA) 2.00 GHz processor, CUDA 8.0, and cuDNN 6021.

4.1. The Adding Problem

4.1.1. Task Description

The adding problem [10] is a widely used benchmark for testing the memorizing ability of RNNs. It requires RNNs to solve a complex long-time lag problem involving distributed, high-precision, continuous-valued representations. Each element of an input sequence is a pair of components. The first component is a real value uniformly sampled from the range $[-1, 1]$; the second is a marker, which is either 1.0 or -1.0. The objective is to calculate the sum of the first components of the first two pairs that are marked by the second component equal to 1.0. The RNNs are trained to minimize the mean square error (MSE) on the training set. It should be noted that a baseline MSE for this task is 0.167, in which case the model always predicts 1.0 regardless of the input [10].

4.1.2. Setup

RNN models included in this test are basic RNN, LSTM, IRNN, and the proposed LRU. Four different sequence lengths—$T = 100, 400, 750, 1500$—are tested. For all models, one hidden layer of 100 neurons is used. The optimizer is SGD with an initial learning rate of 0.1 and a momentum of 0.9. No learning rate decay is performed. Gradients are clipped when they exceeded 10 to avoid gradient exploding. A mini-batch of 32 is used to accelerate training. Processing a mini-batch is referred to as "a training step" in this test for simplicity. The model performance is evaluated on a validation set of 100 mini-batches every 1000 training steps. Training stops after processing 100,000 mini-batches. Both the training and validation data are randomly generated after each model evaluation.

4.1.3. Results

The test results are shown in Figure 3, where $T = N$ indicates a sequence length of N. The problem becomes much more difficult as the sequence length grows, since the dependency between output and the two input numbers becomes more distant. It is observed from Figure 3 that simple RNN and IRNN fail to converge when T grows to 400 and 750, respectively. Therefore, the results of these two models on longer sequences are not presented. Both LSTM and LRU can reach convergence even when the sequence length is 1500, exhibiting their abilities to reserve a very long memory. However, LRU converges faster than LSTM when the sequence length is long (≥750). When $T = 750$, LRU takes fewer than 30,000 steps before obtaining a validation error lower than 0.01, while LSTM takes more than 45,000 steps. When $T = 1500$, LSTM spends more than 75,000 steps before the validation error becomes lower than 0.1, and the validation error fluctuates even after training for all 100,000 steps. On the other hand, LRU performs significantly better than LSTM under this setting, being able to reach stable convergence (validation error ≤ 0.01) within 50,000 steps. Results in this task demonstrate that LRU can effectively address the gradient vanishing problem by labeling long sequences. The performance of LRU is at least comparable to LSTM under different settings and is even better when the sequence length is very long and the parameter number is much lower.

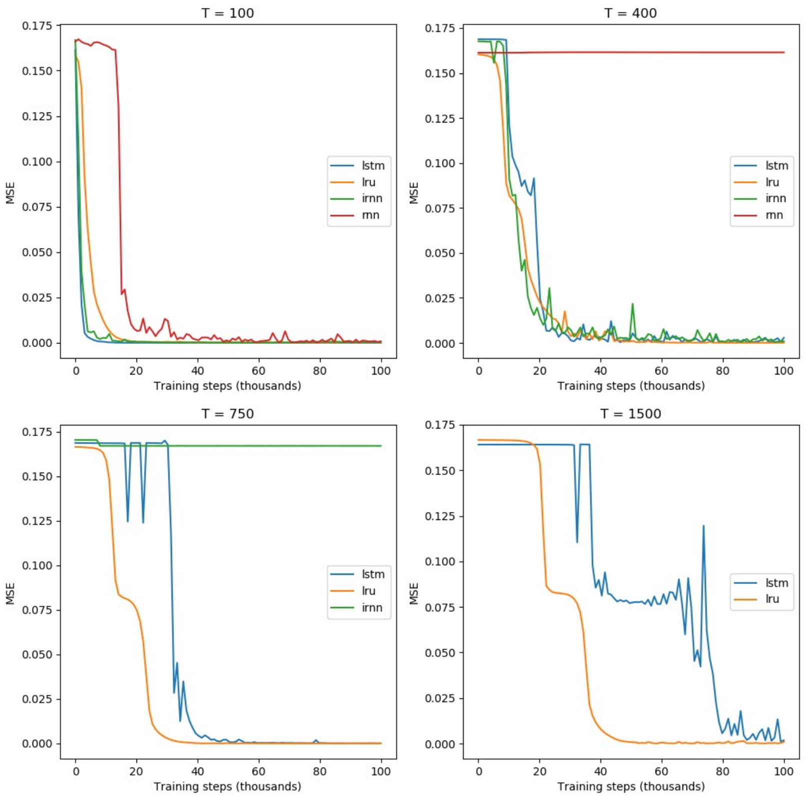


Figure 3. Test errors for the adding problem of different lengths. Basic RNN and IRNN fail to converge when T is 400 and 750, respectively.

4.2. *MNIST Handwritten Digit Classification*

4.2.1. Dataset

The MNIST database [40] is a large database of handwritten digits that is commonly used for training various image processing systems. The database has a training set of 60,000 examples and a testing set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image (28 × 28). Some sample test images are shown in Figure 4. In this experiment, the object is to classify the digits by presenting the 784 pixels in each sample image sequentially to the RNN, one pixel at a time. This task is called “sequential MNIST” in the following sections. The sequential MNIST problem has a very long time dependency since the network needs to keep a 784 time step length of memory. To make the task even harder, another experiment is carried out where all pixels within each sample image are randomly permuted before feeding it to the network, which will be referred to as “permuted MNIST”.

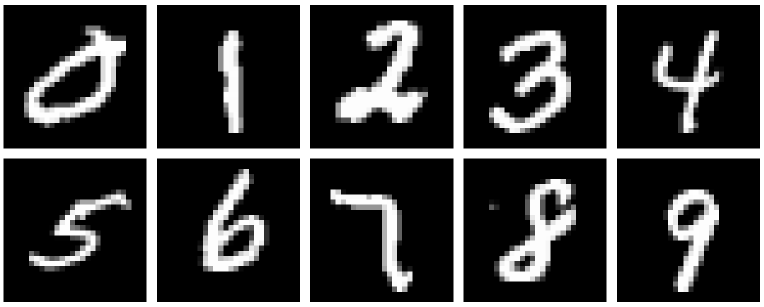


Figure 4. Sample images in the MNIST database labeled from '0' to '9'.

4.2.2. Setup

We tested our implemented LSTM, IRNN, and LRU on this dataset, and results of other models from previous publications are also presented. The training script is derived from the "MNIST example" in the Pytorch official repository (https://github.com/pytorch/examples/tree/master/mnist), accessed on 8 August 2024, with necessary modifications to add support for sequential classification using RNN. The examples in the training set are randomly shuffled at the beginning of every epoch. Both training and testing sets are pre-normalized using the statistics in the training set. For all models, the input layer size is 1, and a softmax layer of 10 units form the output, where each unit corresponds to one digit ranging from '0' to '9'. The 28×28 image is flattened to a 784-dimensional vector as the input to the RNN. One hidden layer of 100 units is used in all the models. Adam [41] is chosen as the optimizer to adjust the learning rate for each parameter automatically. For all models, the initial learning rate is set to 1×10^{-4}. The learning rate is halved every 20 epochs. The gradient clip threshold is set to 10. No further regularization techniques are utilized. A batch size of 128 is used to accelerate training. All models are trained for 100 epochs, and the best performance on the testing set is reported.

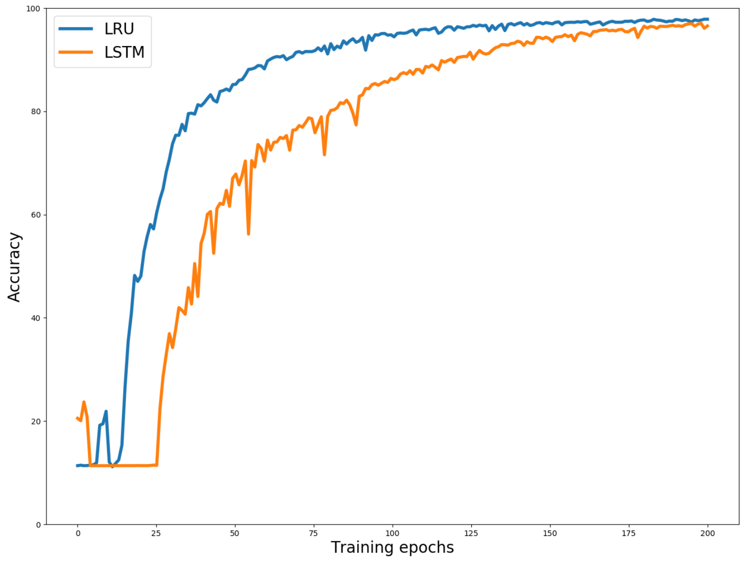
4.2.3. Results

The test results of different RNN models are reported in Table 1. LRU achieves better overall test accuracies than does LSTM, while LRU has only 30,000 parameters, which is 38% of LSTM's 79,000 parameters. On sequential MNIST, our implemented LSTM model achieves a 97.0% accuracy, while LRU is slightly better at 98.5%. On permuted MNIST, LRU achieves a test accuracy of 91.5%, which outperforms LSTM by a relative 5.3% margin. The training curves of LSTM and LRU on the two tasks are displayed in Figure 5. It is observed that LRU converges much faster than LSTM, especially in the first 25 epochs, which is consistent with the results in the adding task.

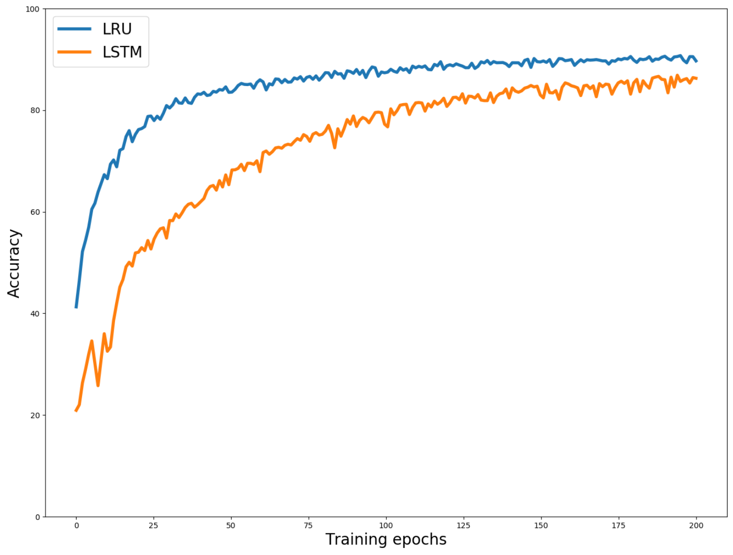
Table 1. Test accuracy of different models on the MNIST and permuted MNIST (pMNIST) digit classification task.

Model	MNIST (%)	pMNIST (%)
SRU [38]	89.0	-
IRNN [22]	95.0	82.0
IRNN (our implementation)	96.1	88.9
uRNN [25]	95.1	91.4
RIN [29]	95.4	86.2
np-RNN [24]	96.8	-
LSTM (our implementation)	97.0	86.9
LRU (our implementation)	**98.5**	**91.5**

As discussed in Section 3.4, the hidden states of an LRU can be considered as the weighted average of all previous inputs' transformations. In the MNIST task, the input is a flattened vector of a 2D image, allowing us to visualize the learned weight vector by reshaping it into a 2D image. We discover that, among all hidden states, about 10% represent very similar distributions as the whole input image, while some other states also show interesting features that might be interpreted as a specific position of the input image. Figure 6a,b show some selected states that are observed to have certain semantic functionalities. It is observed that these states learn to focus on pixels that are discriminative by assigning high weights to these positions (white pixels). Positions that are less discriminative (black pixels) are vastly skipped.



(**a**)



(**b**)

Figure 5. Test set classification accuracy comparison (LRU vs. LSTM) on the MNIST dataset. (**a**) Accuracy plot for sequential MNIST. (**b**) Accuracy plot for permuted MNIST.

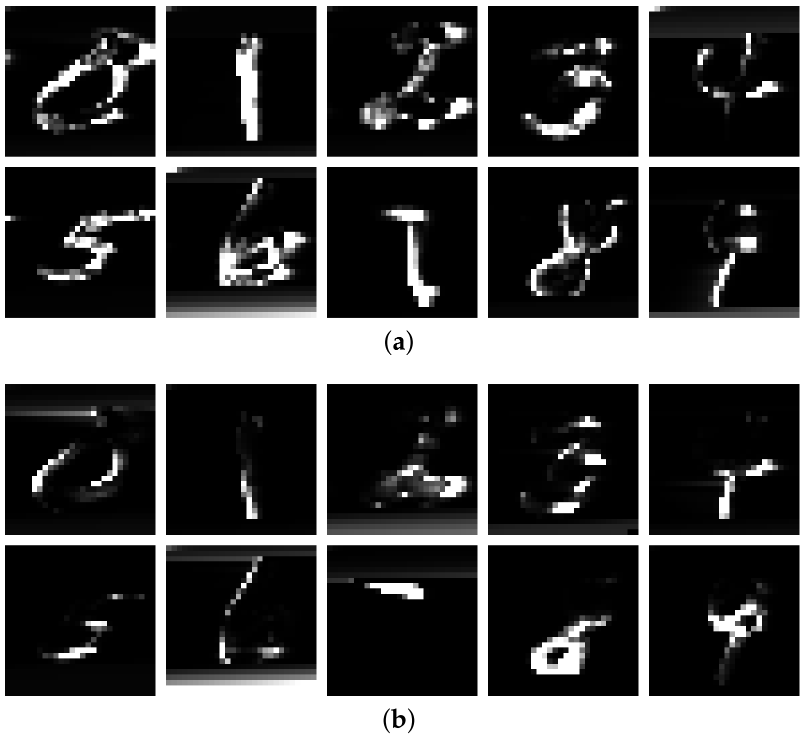


Figure 6. Learned weight visualization in the MNIST experiment. (**a**) Learned weights in LRU that display similar distributions as the whole input image. (**b**) Learned weights in LRU that display similar distributions as specific positions of the input image.

4.2.4. Weight Visualization

This phenomenon also suggests that the hidden state in LRU can be seen as a feature pyramid of input sequence on different levels. This is just a glimpse of the inner functionalities of RNNs that make the model efficient for learning the latent feature of input sequences. Perhaps in the future, more detailed analysis can provide future insight into what RNNs are really learning.

4.3. Language Modeling

4.3.1. Dataset

In this section, the performance of LRU is evaluated on two widely used benchmarks for word-level language modeling: Penn Treebank (PTB) and WikiText2 (WT-2).

The PTB dataset (http://www.cis.upenn.edu/treebank/), accessed on 8 August 2024, is a plain-text corpus dedicated to natural language processing (NLP) tasks. It is one of the most commonly used datasets in the language model domain. The dataset includes 2499 English articles from the 1989 Wall Street Journal, totaling approximately 1 million words. The dataset comes with a dictionary of 10,000 words.

The WT-2 dataset (https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-2-v1.zip), accessed on 8 August 2024, is another widely used dataset in NLP research. It consists of 720 articles sourced from Wikipedia, retaining capitalization, punctuation, and numbers. It is about twice the size of the PTB dataset, featuring a vocabulary of over 33,000 words.

4.3.2. Setup

For the two language modeling benchmarks, we mainly follow the configuration of previous work [42]. The test model uses three layers of LRU with an embedding size of 400 and a hidden size of 1150. We use a batch size of 20 and truncated back-propagation with 70 steps. The dropout probability is 0.4 for the input embedding and output softmax layers. The optimizer is SGD with an initial learning rate of 10.0, and gradients with a magnitude larger than 0.25 are clipped during training. The model is trained up to 500 epochs. A weight decay of 1.2×10^{-6} is applied to control the magnitude of the weight matrices.

4.3.3. Results

The test results of LRU on PTB and WT-2 datasets are given in Table 2. The proposed LRU achieves comparable or even better performance with existing models. On the PTB dataset, LRU outperforms existing models by achieving a validation PPL (perplexity) of 60.5 and a test PPL of 58.2, which improves 0.7% and 0.2%, respectively, over a four-layer LSTM that has 3× more parameters. On the WT-2 dataset, the proposed LRU achieves a validation PPL of 69.4 and a test PPL of 66.1, which is comparable with the four-layer skip-connection LSTM. The fact that LRU achieves good accuracy with a relatively small number of parameters makes the model very efficient in training and storage. This result suggests that LRU could be a candidate alternative RNN model for real-time applications that have space or time requirements.

Table 2. Performance of RNN models on PTB and WT-2 word-level language modeling tasks, measured by perplexity. A lower perplexity value represents better performance.

		PTB			WT-2	
Model	**#Params**	**Val**	**Test**	**#Params**	**Val**	**Test**
LSTM [32]	20 M	83.3	79.8	-	-	-
LSTM+regularization [43]	20 M	86.2	82.7	-	-	-
LSTM+regularization [43]	66 M	82.2	78.4	-	-	-
Variational LSTM [44]	20 M	81.8	79.7	-	-	-
Variational LSTM [44]	66 M	77.3	75.0	-	-	-
Variational LSTM+augmented loss [45]	24 M	75.7	73.2	28 M	91.5	87.0
Variational LSTM+augmented loss [45]	51 M	71.7	68.5	-	-	-
Variational Recurrent Highway Network [46]	23 M	67.9	65.4	-	-	-
4-layer skip-connection LSTM [47]	24 M	60.9	58.3	24 M	69.1	65.9
AWD-LSTM [42]	24 M	60.7	58.8	33 M	69.1	66.0
LRU	8 M	60.5	58.2	8 M	69.4	66.1

4.4. *Ball Bearing Health Monitoring*

4.4.1. Dataset

In this section, by utilizing the ball bearing dataset from Case Western Reserve University (CWRU), we aim to assess the efficacy of the Light Recurrent Unit (LRU).

The bearing dataset from Case Western Reserve University (CWRU), provided by the Department of Electrical Engineering and Computer Science, is widely used for fault diagnosis and prediction research. This dataset records vibration signal data from bearings used in rotating machinery systems. The experimental setup, as shown in Figure 7, includes a 1.5 KW (2 horsepower) motor, a torque sensor/encoder, and a power meter.

The dataset contains vibration signals from both normal operating conditions and various fault types (inner race fault, outer race fault, ball fault) at a sampling frequency of 12 kHz. Each fault type includes multiple fault depths (0.007, 0.014, 0.021, and 0.028 inches). Additionally, the dataset provides experimental data under different working conditions, with multiple samples for each fault mode, making it suitable for diverse experimental needs. Table 3 provides an overview of the dataset specifically utilized in this research.

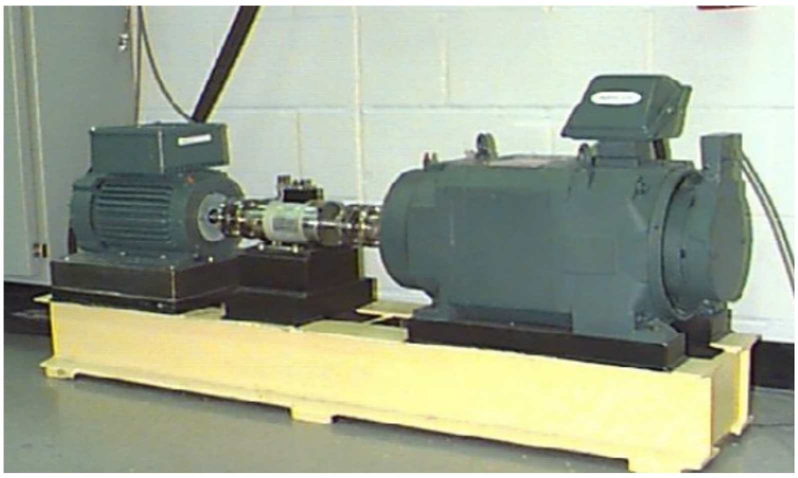


Figure 7. Experimental setup utilized by CWRU.

Table 3. Fault depth, fault type, and fault abbreviations.

Fault Depth	**Fault Type**	**Fault Abbreviations**
	Healthy bearing	N
0.007 inch	Inner race	IRF_007
	Ball	BF_007
	Outer race (Centered)	ORF1_007
	Outer race (Orthogonal)	ORF2_007
	Outer race (Opposite)	ORF3_007
0.014 inch	Inner race	IRF_0014
	Ball	BF_0014
	Outer race (Centered)	ORF_0014
0.021 inch	Inner race	IRF_0021
	Ball	BF_0021
	Outer race (Centered)	ORF1_0021
	Outer race (Orthogonal)	ORF2_0021
	Outer race (Opposite)	ORF3_0021
0.028 inch	Inner race	IRF_028

4.4.2. Setup

In this experiment, we utilized the CWRU dataset to evaluate the performance of a custom Light Recurrent Unit (LRU) model. The model consists of a single LRU layer with a hidden size of 16. We set the batch size to 128 and applied dropout to both the input embedding and output softmax layers, with a dropout probability of 0.2. The cross-entropy loss function was employed for training, and the Adam optimizer was chosen with a learning rate of 0.05. Alongside this, we used an exponential learning rate scheduler with a decay rate (gamma) of 0.99. Training was conducted for 500 epochs. All experiments were performed on a GPU-equipped computing platform. These configurations were selected to ensure efficient training and robust model performance, aiming to achieve high classification accuracy in the task of bearing fault diagnosis.

4.4.3. Results

As illustrated in Figure 8, the confusion matrix demonstrates the model's classification accuracy across various fault types. The experimental results, as shown in Table 4, demonstrate that our Light Recurrent Unit (LRU) model achieved a classification accuracy of 97.14% in the task of bearing fault diagnosis. Compared to other benchmark models reported in the literature [48], our LRU model demonstrated superior accuracy: CNN at 90.46%, MAML at 92.51%, Reptile at 92.63%, and Reptile with Gradient Consistency at 93.48%. Although our model's accuracy is slightly lower than the more complex and specifically designed EML model (98.78%), the LRU model features a simpler design and lower computational resource requirements, significantly reducing complexity and resource demands while maintaining high classification accuracy. Additionally, our model considers all available fault depths (0.007 inches, 0.014 inches, 0.021 inches, and 0.028 inches), offering broader fault coverage compared to the models in the literature.

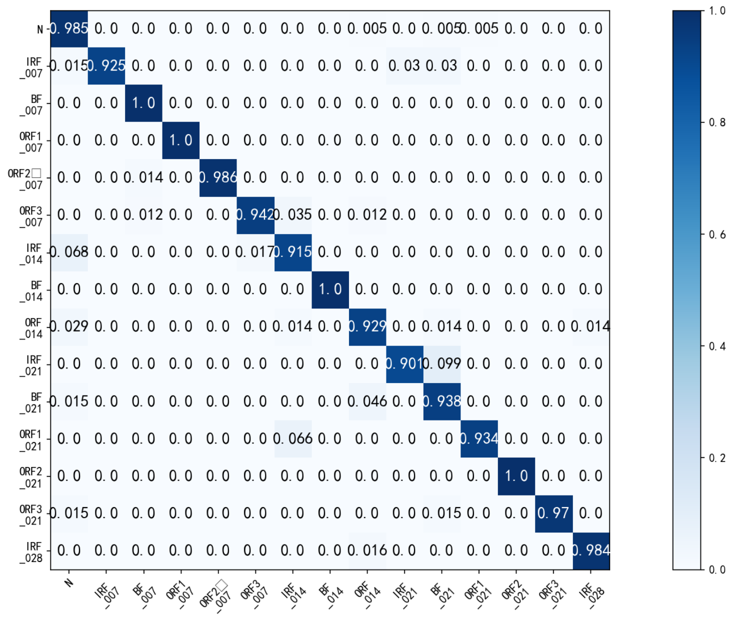


Figure 8. Confusion matrix for LRU model performance on the CWRU dataset.

Table 4. Comparison with existing literature for the CWRU dataset.

References	Models	Accuracy
[48]	CNN	90.46%
	MAML	92.51%
	Reptile	92.63%
	Reptile with GC	93.48%
	EML	98.78%
Ours	**LRU**	**97.14%**

Additionally, we conducted a hyperparameter search to evaluate the impact of different settings on the performance of the LRU model. Specifically, (a) in Table 5 presents the classification accuracy of the model with different learning rates while keeping the batch size fixed at 128, and (b) in Table 5 shows the model performance with different batch sizes while the learning rate is fixed at 0.05.

Table 5. Comparison of different hyperparameters on LRU model performance. (**a**) Different learning rates (batch size = 128). (**b**) Different batch sizes (learning rate = 0.05).

(a)		**(b)**	
Learning Rate	**Accuracy (%)**	**Batch Size**	**Accuracy (%)**
0.05	97.14	64	97.06
0.01	96.03	128	97.14
0.005	80.1	256	95.90

Despite the EML model being specifically designed for bearing fault diagnosis, our LRU model still performs excellently in this task. This indicates that the LRU model possesses strong generalization and adaptability, achieving efficient performance across various tasks. Its simplified structure and efficient computation make it particularly advantageous in resource-constrained environments.

5. Conclusions

This paper introduces the Light Recurrent Unit (LRU), a novel RNN model designed to capture long-term dependencies in sequences with enhanced interpretability of learned states. Experimental results on extensive datasets demonstrate that the proposed LRU model converges quickly on long sequence tasks and often surpasses state-of-the-art RNN models in performance. Based on the evaluation results, LRU emerges as a promising alternative RNN for real-time applications, offering reduced memory usage and training times. More importantly, the high interpretability of LRU enhances our understanding and facilitates progress in learning RNNs. Beyond analysis and understanding, the effectiveness of LRU will be evaluated in more diverse and complex tasks, and advanced regularization techniques will be used on LRU to further improve its accuracy. Our future research will extend the modifications of the LSTM architecture, aiming to develop even more efficient models. These future studies will address emerging challenges in the field of machine learning.

Author Contributions: Conceptualization, H.Y.; Formal analysis, H.Y.; Investigation, H.Y. and X.L.; Software, H.Y., Y.Z., and J.C.; Validation, H.Y., Y.Z., and J.C.; Writing—original draft preparation, H.Z.; Writing—review and editing, Y.Z., J.C., and H.Z.; Data curation, Y.Z. and J.C.; Methodology, H.L.; Project administration, H.Z.; Supervision, H.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data supporting reported results are available from the corresponding author upon reasonable request.

Conflicts of Interest: Author Huizhou Liu was employed by the company State Grid Anhui Electric Power Co., Ltd. Author Xuannong Li was employed by the State Grid Hefei Country Electric Power Supply Company. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Andrianandrianina Johanesa, T.V.; Equeter, L.; Mahmoudi, S.A. Survey on AI Applications for Product Quality Control and Predictive Maintenance in Industry 4.0. *Electronics* **2024**, *13*, 976. [CrossRef]
2. Xie, Z.; Du, S.; Lv, J.; Deng, Y.; Jia, S. A hybrid prognostics deep learning model for remaining useful life prediction. *Electronics* **2020**, *10*, 39. [CrossRef]
3. Song, H.; Choi, H. Forecasting stock market indices using the recurrent neural network based hybrid models: Cnn-lstm, gru-cnn, and ensemble models. *Appl. Sci.* **2023**, *13*, 4644. [CrossRef]
4. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [CrossRef] [PubMed]
5. Hochreiter, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* **1998**, *6*, 107–116. [CrossRef]
6. Hochreiter, S.; Bengio, Y.; Frasconi, P.; Schmidhuber, J. *Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies*; Wiley-IEEE Press: New York, NY, USA, 2001.
7. Zhao, J.; Huang, F.; Lv, J.; Duan, Y.; Qin, Z.; Li, G.; Tian, G. Do RNN and LSTM have long memory? In Proceedings of the International Conference on Machine Learning, Virtual, 13–18 July 2020; pp. 11365–11375.
8. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1310–1318.
9. Landi, F.; Baraldi, L.; Cornia, M.; Cucchiara, R. Working memory connections for LSTM. *Neural Netw.* **2021**, *144*, 334–341. [CrossRef] [PubMed]
10. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
11. Sherstinsky, A. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Phys. D Nonlinear Phenom.* **2020**, *404*, 132306. [CrossRef]

12. Yadav, H.; Thakkar, A. NOA-LSTM: An efficient LSTM cell architecture for time series forecasting. *Expert Syst. Appl.* **2024**, *238*, 122333. [CrossRef]
13. Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1724–1734.
14. Zhang, J.; Xie, X.; Peng, G.; Liu, L.; Yang, H.; Guo, R.; Cao, J.; Yang, J. A Real-Time and Privacy-Preserving Facial Expression Recognition System Using an AI-Powered Microcontroller. *Electronics* **2024**, *13*, 2791. [CrossRef]
15. Al-Nader, I.; Lasebae, A.; Raheem, R.; Khoshkholghi, A. A Novel Scheduling Algorithm for Improved Performance of Multi-Objective Safety-Critical Wireless Sensor Networks Using Long Short-Term Memory. *Electronics* **2023**, *12*, 4766. [CrossRef]
16. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 5998–6008.
17. Huang, C.; Tu, Y.; Han, Z.; Jiang, F.; Wu, F.; Jiang, Y. Examining the relationship between peer feedback classified by deep learning and online learning burnout. *Comput. Educ.* **2023**, *207*, 104910. [CrossRef]
18. Zheng, W.; Gong, G.; Tian, J.; Lu, S.; Wang, R.; Yin, Z.; Yin, L. Design of a Modified Transformer Architecture Based on Relative Position Coding. *Int. J. Comput. Intell. Syst.* **2023**, *16*, 168. [CrossRef]
19. Pirani, M.; Thakkar, P.; Jivrani, P.; Bohara, M.H.; Garg, D. A comparative analysis of ARIMA, GRU, LSTM and BiLSTM on financial time series forecasting. In Proceedings of the 2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), Ballari, India, 23–24 April 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–6.
20. Lindemann, B.; Maschler, B.; Sahlab, N.; Weyrich, M. A survey on anomaly detection for technical systems using LSTM networks. *Comput. Ind.* **2021**, *131*, 103498. [CrossRef]
21. Al Hamoud, A.; Hoenig, A.; Roy, K. Sentence subjectivity analysis of a political and ideological debate dataset using LSTM and BiLSTM with attention and GRU models. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 7974–7987. [CrossRef]
22. Le, Q.V.; Jaitly, N.; Hinton, G.E. A simple way to initialize recurrent networks of rectified linear units. *arXiv* **2015**, arXiv:1504.00941.
23. Wang, J.; Li, X.; Li, J.; Sun, Q.; Wang, H. NGCU: A new RNN model for time-series data prediction. *Big Data Res.* **2022**, *27*, 100296. [CrossRef]
24. Neyshabur, B.; Wu, Y.; Salakhutdinov, R.R.; Srebro, N. Path-normalized optimization of recurrent neural networks with relu activations. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 3477–3485.
25. Arjovsky, M.; Shah, A.; Bengio, Y. Unitary evolution recurrent neural networks. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1120–1128.
26. Talathi, S.S.; Vartak, A. Improving performance of recurrent neural network with relu nonlinearity. *arXiv* **2015**, arXiv:1511.03771.
27. Dhruv, P.; Naskar, S. Image classification using convolutional neural network (CNN) and recurrent neural network (RNN): A review. In *Machine Learning and Information Processing: Proceedings of ICMLIP 2019*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 367–381.
28. Mikolov, T.; Joulin, A.; Chopra, S.; Mathieu, M.; Ranzato, M. Learning longer memory in recurrent neural networks. *arXiv* **2014**, arXiv:1412.7753.
29. Hu, Y.; Huber, A.; Anumula, J.; Liu, S.C. Overcoming the vanishing gradient problem in plain recurrent networks. *arXiv* **2018**, arXiv:1801.06105.
30. Gers, F.A.; Schmidhuber, J.; Cummins, F. Learning to forget: Continual prediction with LSTM. In Proceedings of the 1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470), Edinburgh, UK, 7–10 September 1999.
31. Ali, M.H.E.; Abdellah, A.R.; Atallah, H.A.; Ahmed, G.S.; Muthanna, A.; Koucheryavy, A. Deep Learning Peephole LSTM Neural Network-Based Channel State Estimators for OFDM 5G and Beyond Networks. *Mathematics* **2023**, *11*, 3386. [CrossRef]
32. Jozefowicz, R.; Zaremba, W.; Sutskever, I. An empirical exploration of recurrent network architectures. In Proceedings of the International Conference on Machine Learning, Lille, France, 7–9 July 2015; pp. 2342–2350.
33. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. In Proceedings of the NIPS 2014 Workshop on Deep Learning, Montreal, QC, Canada, 13 December 2014.
34. Zhou, G.B.; Wu, J.; Zhang, C.L.; Zhou, Z.H. Minimal gated unit for recurrent neural networks. *Int. J. Autom. Comput.* **2016**, *13*, 226–234. [CrossRef]
35. Ravanelli, M.; Brakel, P.; Omologo, M.; Bengio, Y. Light gated recurrent units for speech recognition. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 92–102. [CrossRef]
36. Khan, M.; Wang, H.; Riaz, A.; Elfatyany, A.; Karim, S. Bidirectional LSTM-RNN-based hybrid deep learning frameworks for univariate time series classification. *J. Supercomput.* **2021**, *77*, 7021–7045. [CrossRef]
37. Greff, K.; Srivastava, R.K.; Koutník, J.; Steunebrink, B.R.; Schmidhuber, J. LSTM: A search space odyssey. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *28*, 2222–2232. [CrossRef] [PubMed]
38. Oliva, J.B.; Póczos, B.; Schneider, J. The statistical recurrent unit. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, Sydney, Australia, 6–11 August 2017; pp. 2671–2680.
39. Srivastava, R.K.; Greff, K.; Schmidhuber, J. Training very deep networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 2377–2385.

40. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
41. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
42. Merity, S.; Keskar, N.S.; Socher, R. Regularizing and optimizing LSTM language models. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
43. Zaremba, W.; Sutskever, I.; Vinyals, O. Recurrent neural network regularization. *arXiv* **2014**, arXiv:1409.2329.
44. Gal, Y.; Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 1019–1027.
45. Inan, H.; Khosravi, K.; Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv* **2016**, arXiv:1611.01462.
46. Zilly, J.G.; Srivastava, R.K.; Koutník, J.; Schmidhuber, J. Recurrent highway networks. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, Sydney, Australia, 6–11 August 2017; pp. 4189–4198.
47. Melis, G.; Dyer, C.; Blunsom, P. On the state of the art of evaluation in neural language models. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018
48. Che, C.; Wang, H.; Xiong, M.; Ni, X. Few-shot fault diagnosis of rolling bearing under variable working conditions based on ensemble meta-learning. *Digit. Signal Process.* **2022**, *131*, 103777. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.