applied sciences

MDPI

Article

Sequential Characteristics Based Operators Disassembly Quantization Method for LSTM Layers

Yuejiao Wang, Zhong Ma * and Zunming Yang

Xi'an Microelectronics Technology Institute, Xi'an 710065, China
* Correspondence: mazhong@mail.com; Tel.: +86-187-1089-0519

Abstract: Embedded computing platforms such as neural network accelerators deploying neural network models need to quantize the values into low-bit integers through quantization operations. However, most current embedded computing platforms with a fixed-point architecture do not directly support performing the quantization operation for the LSTM layer. Meanwhile, the influence of sequential input data for LSTM has not been taken into account by quantization algorithms. Aiming at these two technical bottlenecks, a new sequential-characteristics-based operators disassembly quantization method for LSTM layers is proposed. Specifically, the calculation process of the LSTM layer is split into multiple regular layers supported by the neural network accelerator. The quantization-parameter-generation process is designed as a sequential-characteristics-based combination strategy for sequential and diverse image groups. Therefore, LSTM is converted into multiple mature operators for single-layer quantization and deployed on the neural network accelerator. Comparison experiments with the state of the art show that the proposed quantization method has comparable or even better performance than the full-precision baseline in the field of character-/word-level language prediction and image classification applications. The proposed method has strong application potential in the subsequent addition of novel operators for future neural network accelerators.

Keywords: embedded neural network accelerator; operators disassembly quantization method; quantization parameter generation process; sequential characteristics based combination strategy

Citation: Wang, Y.; Ma, Z.; Yang, Z. Sequential Characteristics Based Operators Disassembly Quantization Method for LSTM Layers. *Appl. Sci.* **2022**, *12*, 12744. https://doi.org/10.3390/app122412744

Academic Editor: Luis Javier Garcia Villalba

Received: 1 November 2022
Accepted: 7 December 2022
Published: 12 December 2022

1. Introduction

In order to realize high-speed and low-power operations, embedded neural network accelerators generally only support low-precision numerical operations, such as 8-bit or 4-bit low-bit integer operations. In contrast, the original numerical precision of neural network models is generally 32-bit floating-point numbers. Therefore, quantization algorithms need to be employed to compress the original network by reducing the number of bits of precision required to represent weights or activations [1]. In the calculation process of the neural network accelerator deployed with the quantization algorithm, the number of data transfers can be effectively reduced, the bottleneck of transmission bandwidth can be reduced, and the calculation speed can be improved [2]. Quantization converts the floating-point value of the neural network into a fixed-point integer value with a set of scale factors. These scale factors, sometimes also called quantization parameters, are calculated according to the dynamic range threshold and quantization bit width of the floating-point data during the conversion process. When the upper and lower boundaries of the data dynamic range threshold are symmetrical, it is called a symmetric quantization, and when they are asymmetrical, it is called asymmetric quantization. The goal of a quantization algorithm is to determine the quantization parameters of a neural network model.

This paper is an extended version of a conference paper [3]. Unlike the conference paper, the method proposed here determines a quantization parameter generation method for the LSTM layer for processing sequential features. Multi-bit quantization retraining is added to be supported on CPU/GPU. This method is universal, the corresponding classical

benchmark datasets are selected for evaluating different applications such as character-level language prediction, word-level language prediction, and image classification.

Many of the most widely used AI applications are now based on Long Short-Term Memory (LSTM) recurrent neural networks (RNNs), which learn from experience to solve all kinds of previously unsolvable problems. The LSTM principle has become a foundation of what is now called deep learning, especially for sequential data. LSTM-based systems can learn to translate languages; control robots; analyze images; summarize documents; recognize speech, videos, and handwriting; run chatbots; predict diseases, click rates, and stock markets; compose music; and much more [4].

LSTM is mainly used to perform tasks in sequential data processing. It can effectively solve the problem of gradient explosion in RNN and can effectively capture the dependencies between contexts. In these real-time machine learning applications with sequential features, LSTM is severely limited by latency, energy, and size. In order to improve hardware efficiency, many researchers have proposed the use of the quantization technology of intelligent algorithms to relieve the technical pressure of platform computing power and improve computing efficiency.

However, there are two technical bottlenecks regarding LSTM quantization: one is the embedded computing platform, and the other is the quantization algorithm itself. We will start with a general overview from these two perspectives.

Technical Bottleneck 1: Most of the current embedded computing platforms do not directly support performing the quantization operation for the LSTM layer. There are two difficulties: (1) such hardware interfaces rely too much on neural network models; (2) there is a lack of quantization tools to support LSTM calculation.

Technical Bottleneck 2: The influence of sequential input properties of LSTM has not been taken into account by quantization algorithms. There are two problems: (1) ordinary quantization calibration images, which are used for regular layers, do not have sequential characteristics; (2) quantization calibration images adapted to LSTM do not ensure the diversity of quantization.

Aiming at these two technical bottlenecks in both the embedded computing platform and the quantization algorithm, this paper proposes a sequential-characteristics-based operators-disassembly-quantization method for LSTM layers. There are two innovations in this proposed method to improve the technical bottlenecks described above:

(1) We design an operators disassembly quantization process for the LSTM layer, so that the neural network accelerator can support the LSTM layer on the embedded platform.
(2) The quantization parameter generation process is designed as a sequential-characteristics-based combination strategy for LSTM quantization calibration. The advantages of this design are that sequential characteristics are added to the quantization algorithms. It breaks through the limitations of existing quantization methods that rely on the quantization calibration sets to be still images and can only support model structure with traditional layer types.

2. Related Works

The main acceleration targets of the current neural network accelerators on the embedded Field Programming Gate Arrays (FPGA) platform [5] are typical neural network operators, such as convolution, full connection, pooling, etc. However, with the increasing complexity of the problems solved by deep learning technology, some new types of complex operators have gradually emerged, which are mainly represented by variant form models of RNN [6], such as LSTM [7] and GRU [8]. The RNN neural network model is widely used in the field of natural language processing, but it has problems with vanishing or exploding gradients when processing large amounts of text.

The improved form of RNN is LSTM, which introduces three gated structures, the forget gate, the input gate, and the output gate, in the unit, which can effectively solve this problem. The forget gate is responsible for the selective deletion of the input information

and the hidden state information output by the previous unit, the input gate is responsible for selectively storing the retained new information into the unit state, and the output gate is responsible for selectively outputting the information in the unit state to the hidden layer.

LSTM removes or adds information to the cell state, called gates: input gate (i_t), forget gate (f_t), output gate (o_t), cell input vector (g_t), cell state vector (c_t), and hidden state vector (h_t) can be defined as follows:

$$i_t = sigmoid(W_{hi} * h_{t-1} + W_{xi} * x_t + b_i) \quad (1)$$

$$f_t = sigmoid(W_{hf} * h_{t-1} + W_{xf} * x_t + b_f) \quad (2)$$

$$o_t = sigmoid(W_{ho} * h_{t-1} + W_{xo} * x_t + b_o) \quad (3)$$

$$g_t = \tanh(W_{hg} * h_{t-1} + W_{xg} * x_t + b_g) \quad (4)$$

$$c_t = (f_t . * c_{t-1}) + (i_t . * g_t) \quad (5)$$

$$h_t = o_t . * \tanh(c_t) \quad (6)$$

where W_{xi}, W_{xf}, W_{xo}, W_{xg} is the weight matrix of the input node; W_{hi}, W_{hf}, W_{ho}, W_{hg} is the weight matrix of the hidden state; and b_i, b_f, b_o, b_g is the bias term.

LSTM has been widely used in various sequential data predictions [9]. In the field of text processing, LSTM can perform text classification, text sequence annotation, text translation, image generation text, text-generated speech, speech recognition [10], etc. In the field of computer vision, LSTM is primarily used for video classification, image annotation, video annotation [11], and recently popular visual Q&A. In the field of engineering practice, computation offloading in mobile edge computing can also be predicted by LSTM [12].

The quantization for LSTM compresses the original LSTM network by reducing the numerical bit width of the model and reducing the calculation and inference time, and it has a direct relationship with the hardware. The specific quantization process for most quantization algorithms is as follows. It uses the layers as the basic processing unit. First, the distribution and range of weight and output data of each layer of the neural network model are analyzed in advance. Then, each floating-point value is represented by a low-bit integer based on the uniform-symmetric [13] quantization strategy. Finally, the quantization-calibrated weights and activation thresholds, namely quantization parameters, are calculated to generate a guide file for the neural network accelerator to map floating-point data into integer data.

The quantization strategy of mapping floating-point data to integer data is to select a reasonable threshold based on the dynamic range of the data to ensure that the loss of information is minimized. Therefore, the original information is truncated before mapping through the threshold, and it is mapped to a low bit-precision range. This avoids the problem of wasted dynamic range resources. There are many quantization strategies for selecting thresholds, such as the minimum–maximum (MIN–MAX) quantization strategy used by Tengine, which makes the absolute maximum value the threshold value. Taking 8-bit quantization as an example, we can see that it is proportionally mapped to the range of plus or minus 127. This simple MIN–MAX clipping only works for uniform distributions. When the data are non-uniformly distributed, the loss of dynamic range can cause a considerable loss of accuracy. Most users, such as NVIDIA, have proposed using the KL-divergence (KLD)-based method [14] to measure the degree of difference in the distributions before and after the quantization to find the optimal threshold value. KL divergence has some disadvantages, such as being unbounded asymmetric, and the results obtained are meaningless if the two distributions do not overlap. There are also some other quantization strategies that use the average–maximum (AVG–MAX), Easy Quant [15], etc., to determine the optimal threshold value. AVG–MAX takes the average value according to the change in value, taking into account the principles of simplicity and balance. EasyQuant achieves close to Int8 and Float32 performance even at Int7 or lower bits. However, its accuracy and performance advantages over KLD are mainly reflected in the int7 type, and

the accuracy of int8 seems similar to KLD. Google's TensorFlow [16] applies exponential moving averages (EMA) for the activation quantization, which calculates the bounds of activations on the fly. It modifies the bounds after each iteration, which is too frequent to be suitable for quantization parameter learning. The comparison table of key characteristics and limitations of different quantization strategies is shown in Table 1.

Table 1. Comparison table of key characteristics and limitations of quantization strategies.

Quantization Strategy	Presenter/ Researcher	Characteristics	Limitations
MIN-MAX	Tengine	The principle is simple, the operation is fast and efficient, and it is suitable for uniform distribution.	Non-uniform distribution of data results in a significant loss of accuracy.
KLD	NVIDIA	Measure the degree of difference in the distributions before and after the quantization to find the optimal threshold value.	Unbounded, asymmetric, and the results obtained are meaningless if the two distributions do not overlap.
Easy Quant	/	Achieves close to Int8 and Float32 performance even at Int7 or lower bits.	The accuracy of int8 seems to be similar to KLD.
AVG-MAX	/	Taking into account the principle of simplicity and balance.	Same as MIN-MAX.
EMA	TensorFlow	Calculates the bounds of activations on the fly.	Frequently modifying boundaries is not suitable for quantization parameter learning.
Ours	/	It is a general quantization method that can use many of the quantization strategies described earlier.	Offline quantization training is complex and time-consuming.

We want to achieve LSTM quantization that performs as well as quantization on the regular layers. For this purpose, we analyze previous quantization methods and observe that, at present, the quantization of the LSTM layer still has the following two problems:

(1) LSTM has not only high-density computation such as matrix-vector multiplication [17] but also has complex control flow, which will significantly affect the execution speed of LSTM [18]. It is implemented on a specific FPGA platform [19]. However, such hardware interfaces rely too much on neural network models and hardware platforms and lack generality [20].

(2) Most of the current general-purpose neural network accelerators cannot directly support the quantization calculation of LSTM. Firstly, the LSTM layer cannot be directly executed by the neural network accelerator on the embedded FPGA platform. Secondly, there is a lack of quantization tools to support LSTM calculation [21]. Even if the LSTM layer is converted into an existing regular layer, the traditional quantization calibration methods are still applied; i.e., sequential characteristics are not taken into account. That is, all quantization calibration sets are repeatedly fed into each time step of the LSTM as input. The activation quantization threshold is selected according to the distribution of layer-by-layer output data, regardless of the size of the time step [22]. It is very easy to cause the quantization dataset of the LSTM layer not to have diversity and sequential consistency, which in turn leads to a considerable loss of quantization accuracy.

Therefore, aiming at the neural network model fused with LSTM layers based on sequential data, it is necessary to design a quantization method for neural network models that integrates the LSTM layer.

That is, on the one hand, the LSTM layer is split from the software level into multiple basic operators supported by the neural network accelerator, so that the neural network accelerator can support the LSTM layer on the embedded platform, and it can be combined with the traditional model quantization algorithm to broaden the scope of application of the quantization algorithm.

On the other hand, the combination strategy using quantization calibration sets takes into account the sequential characteristics of the LSTM layer. The combined images are designed for the quantization calibration of regular layers. When encountering the LSTM layer, according to the time step of LSTM, the diverse image groups are gradually sent to each time step, where each diverse image group has sequential characteristics. It solves the hardware deployment problem of the sequential forecasting model and improves the accuracy of the quantization algorithm. This not only ensures the diversity and scientificity of quantization but also improves the quantization accuracy of models with LSTM layers.

3. Proposed Methodology

3.1. Problem Definition

The quantization method for LSTM layers deployed on neural network accelerators discussed in this paper is as follows.

(1) The embedded computing platforms such as the neural network accelerator deploying the neural network models need to quantize the values into low-bit integers through quantization operations. However, most of the current embedded computing platforms with a fixed-point architecture do not directly support performing the quantization operation for the LSTM layer. The original structure of LSTM consists of 6 types of mapping as shown in Equations (1)–(6), named $\{i_t, f_t, o_t, g_t, c_t, h_t\}$, respectively. However, embedded computing platform (such as NPU) can only handle mapping cases with linear or nonlinear relationships such as $Y = A + B$, $Y = A \cdot B$ or $Y = f(A)$. Therefore, we divide the LSTM layer into a set of regular layers, and its computation process is a combination of the fully connected layer (FC), the Eltwise layer (Eltwise Add, Eltwise Prod), and the nonlinear layer (Sigmoid, Tanh). That is, as shown in Equation (7), we strive to find a correspondence φ between the LSTM and the NPU in this paper, and split and map the LSTM into a certain type of Y supported by NPU.

$$\varphi(LSTM) = \begin{pmatrix} \varphi(i_t) \\ \varphi(f_t) \\ \varphi(o_t) \\ \varphi(g_t) \\ \varphi(c_t) \\ \varphi(h_t) \end{pmatrix} = \begin{Bmatrix} A + B \\ A \cdot B \\ f(A) \end{Bmatrix} = Y \tag{7}$$

(2) LSTM is mainly used to perform tasks in sequential data processing. However, the influence of sequential input data of LSTM has not been taken into account by quantization algorithms. Aiming at this bottleneck, the goal of the proposed quantization algorithm is to determine the quantization parameters of neural network models with the LSTM layer and deploy them on the NPU. The original numerical precision of neural network models is generally 32-bit floating-point numbers. The core computation of a 32-bit floating-point model can be expressed as $Y = W \cdot X$. Since NPU only supports integer matrix multiplication, quantization converts the floating-point weight W and activation X of the neural network into a fixed-point integer value W_q and X_q by $\phi(x)$, as shown in (8), where a and b are called quantization parameters.

$$\phi(x) = \phi(round(ax+b)) = \begin{pmatrix} \phi(W) \\ \phi(X) \end{pmatrix} = \begin{Bmatrix} W_q \\ X_q \end{Bmatrix} \tag{8}$$

Therefore, the calculation of $Y_q = W_q \cdot X_q$ can be performed on the NPU. However, it will cause $Y_q \neq Y$. Therefore, Y_q needs to be inversely quantized to Y by $Y = \phi^{-1}(Y_q)$. Then, finding the reasonable values of the quantization parameters a and b becomes the problem that needs to be solved in this paper.

3.2. Overall Method Design

The proposed quantization method combines the quantization operation with the neural network accelerator hardware. It is described in detail below. First of all, the operators' disassembly is designed so that the calculation process of LSTM is split into layer-by-layer calculation sequences. Then, the quantization calibration process is designed as a sequential-characteristics-based combination strategy for sequential and diverse image groups. The input and output vectors of the separated LSTM are reasonably connected to the existing traditional type layer. Finally, the quantization parameters are generated and deployed on the embedded computing platform (such as NPU) for accuracy calculation testing.

In this section, we begin specific hardware and software deployment design processes. In order to design a quantization deployment scheme for LSTM layers and verify the effectiveness of the scheme, it is necessary to deploy models on different platforms for specific data sets, including PC and NPU. Quantization is carried out for the hardware platform. Finally, the model inference outputs on different platforms are classified and predicted, and the accuracy evaluation is given. The overall design process is shown in Figure 1.

The overall method design consists of four parts; dataset configuration, model deployment for different platforms, LSTM sequential quantization, and classification prediction. The specific breakdown is as follows:

- Part I: Firstly, configure a reasonable dataset set for different platforms. Secondly, set up a certain number of quantization calibration sets for the hardware platform. Thirdly, design the quantization calibration process as a sequential-characteristics-based combination strategy for sequential and diverse image groups. See Section 3.4 for details.
- Part II: On the PC side, the model is pushed forward based on the Caffe deep learning software framework. On the NPU side, the quantization deployment of the hardware platform is carried out. For a neural network model with both regular and LSTM layers, the existing uniform-symmetric quantization is performed on the regular and LSTM layers.
- Part III: The detailed process of LSTM layer quantization on the hardware platform is as follows: the LSTM layer is divided into a set of regular layers, and its computation process is a combination of the fully connected layer (FC), the Eltwise layer (Eltwise Add, Eltwise Prod), and the nonlinear layer (Sigmoid, Tanh). It is reasonably connected with other regular layers of the model to generate quantization parameters and deploy them on the neural network accelerator. See Section 3.3 for details.
- Part IV: Finally, the quantized output of the NPU side is inverse-quantized to a floating-point representation type. The model outputs on the PC and NPU side are separately classified and predicted, and the accuracy evaluation results are given.

The proposed method determines a quantization-parameter-generation method for the LSTM layer for processing sequential features. Multi-bit quantization retraining is supported on CPU/GPU. This method is implemented entirely on a hardware with a fixed-point architecture. Once complex quantization training is completed offline, the quantized LSTM model can be directly deployed on the embedded neural network accelerator. It not only quickly implements the support of the embedded computing platform for the LSTM but also effectively improves the quantization accuracy of the model.

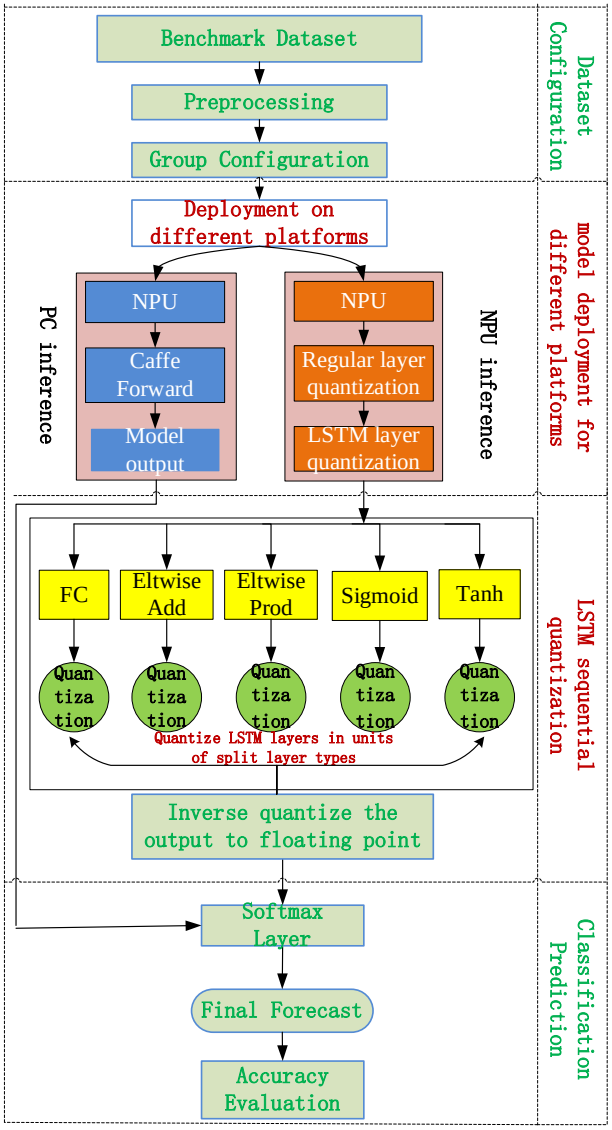


Figure 1. Overall method design process.

3.3. LSTM Operators Disassembly Design

The LSTM is a specific network layer in the neural network model, but the computational logic of this layer is more complex than the typical layer. The input of the LSTM layer is x and $Cont$, and the output is h. x is a tensor of (T, I), where T is the time step and I is the input feature dimension. $Cont$ is a continuity identifier, consisting of 0, 1, where 0 means start and 1 means continuous, and the dimension is T. h is a tensor of (T, O), where O is the output feature dimension.

LSTM first divides the calculation into T small computing units and then divides x into T (1, I) tensors equally according to the size of the time step T, named $\{x_0, x_1 \cdots x_t, \cdots x_{T-1}\}$, respectively. These T divided inputs will be used as the input x_t of each computational unit in the LSTM. It is worth noting here that h_0 and c_0 are all-zero tensor sequences (1,O), and h_t and c_t are the result of the previous h_{t-1} and c_{t-1}. Finally, the sequence of T tensors (1, O) from h_0 to h_{T-1} are combined into one output h (T, O) as the output of the LSTM layer.

In the specific Caffe implementation process, the input of LSTM can be expressed as $W_h * h + W_x * x + b$, so we use two fully connected layers to replace the matrix multiplication operations in Equations (1)–(4). The first is $W_x \times x + b$, and the second is $W_h \times h$, where $W_x = [W_{xi}, W_{xf}, W_{xo}, W_{xg}]$, $W_h = [W_{hi}, W_{hf}, W_{ho}, W_{hg}]$, $b = [b_{xi}, b_{xf}, b_{xo}, b_{xg}]$.

In terms of layer type, Equations (1)–(6) are divided into the LSTM layer operation processes calculated in the following 9-layer order:

Layer 1—FC layer:

$$data\_fc1 = W_x * x_t + b \tag{9}$$

Layer 2—FC layer:

$$data\_fc2 = W_h * h_t \tag{10}$$

Layer 3—Eltwise Add layer:

$$data\_sum = data\_fc1 + data\_fc2 \tag{11}$$

Layer 4—Non-linear layer:

$$\begin{array}{c} data\_it = sigmoid(data\_sum[0:O]) \\ data\_ft = sigmoid(data\_sum[O:2*O]) \\ data\_ot = sigmoid(data\_sum[2*O:3*O]) \\ data\_gt = tanh(data\_sum[3*O:4*O]) \end{array} \tag{12}$$

Layer 5—Eltwise Prod layer:

$$eltwise1 = data\_it * data\_gt \tag{13}$$

Layer 6—Eltwise Prod layer:

$$eltwise2 = data\_ft * data\_ct \tag{14}$$

Layer 7—Eltwise Add layer:

$$data\_ct = eltwise1 + eltwise2 \tag{15}$$

Layer 8—Non-linear layer:

$$data\_ct\_tanh = tanh(data\_ct) \tag{16}$$

Layer 9—Eltwise Prod layer:

$$data\_ht = data\_ot * data\_ct\_tanh \tag{17}$$

Because hardware accelerators can only process integer data, when we perform non-linear function operations, there are two steps in total. The first step is to build a nonlinear lookup table in the quantization software. If there are two nonlinear functions, two lookup tables are created. The specific operation is to first dequantize all integer values in the bit width range to a floating point according to the inverse of Equation (8). The dequantized floating-point values are then mapped according to the nonlinear functions sigmoid and tanh, respectively. Finally, the mapped value is quantized to an integer again according to Equation (8). The sequence of input integers and output integers form a lookup table. The second step is to feed the lookup table into the neural network accelerator. The corresponding rules between the source register and the destination register are used to map one-by-one to complete the calculation of the nonlinear layer.

3.4. LSTM Sequential Quantization Design

The LSTM layer is disassembled and becomes a collection of many regular layers, and the quantization calibration method of the regular layers can also be applied. That is, all quantization calibration sets are first fed into the regular layers in front of the LSTM, and the activation quantization threshold is selected according to the distribution of layer-by-layer output data. When encountering an LSTM layer, regardless of the size of the time step, the input quantization calibration set is repeatedly fed into each time step of the LSTM

as input to each computational unit in the LSTM. Finally, each output tensor sequence is combined into a single output as the output of the LSTM layer. However, since the LSTM layer processes datasets that have sequential characteristics, once processed according to the basic methods as shown in Figure 2, it is very easy to cause a large loss of quantization accuracy.

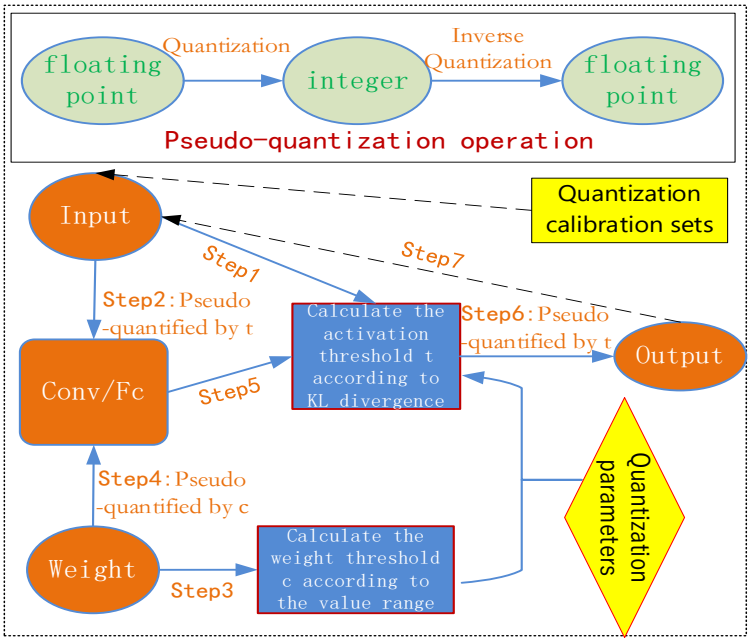


Figure 2. Overall flow chart of the quantization algorithm.

Our basic quantization algorithm is described in detail as follows. When the quantization algorithm generates quantization parameters of regular layers and LSTM layers, a pseudo-quantization operation is inserted into the weight and output of each layer. That is, the floating-point value is first quantized to a low-bit integer and then inversely quantized back to a floating-point value. Each time, the pseudo-quantized values are used to perform operator calculations, such as convolution, pooling, upsampling, etc. After the calculation is completed, pseudo-quantization is performed again, and the quantization parameters are counted and sent to the next layer. The output of the current layer is used as the input of the next layer and so on. Note that the first layer requires additional pseudo-quantization operations on the input. The overall flow chart of the quantization algorithm for neural network models that integrates the LSTM layer is shown in Figure 2.

In this section, the quantization calibration process is designed as a sequential-characteristics-based combination strategy for LSTM quantization calibration. The neural network model integrating LSTM layers has both regular layers and LSTM layers. The quantization calibration set of the network model built only for regular layers is relatively simple. Many diverse image groups with different categories, backgrounds, angles, and lightings are selected. The data processed by LSTM are generally sequential; that is, the data collected at different times for the described phenomenon over time. Therefore, this paper designs the sequential-characteristics-based combination strategy of sequential and diverse image groups.

Assuming that the time step of the LSTM layer is T, the model input size is $C \times H \times W$. Thus, N diverse image datasets groups of different categories, different backgrounds, different angles, and different lightings are selected. Each group contains T groups of sequential datasets with training consistency; that is, the dimension of the quantized calibration set is $N \times T$. On the one hand, the combined $N \times T$ images are used for the quantization calibration of regular layers. On the other hand, when encountering the LSTM layer, according to the time step of LSTM, N diverse image groups are gradually sent to

each time step, where each diverse image group has sequential characteristics. The specific quantization calibration process is shown in Figure 3.

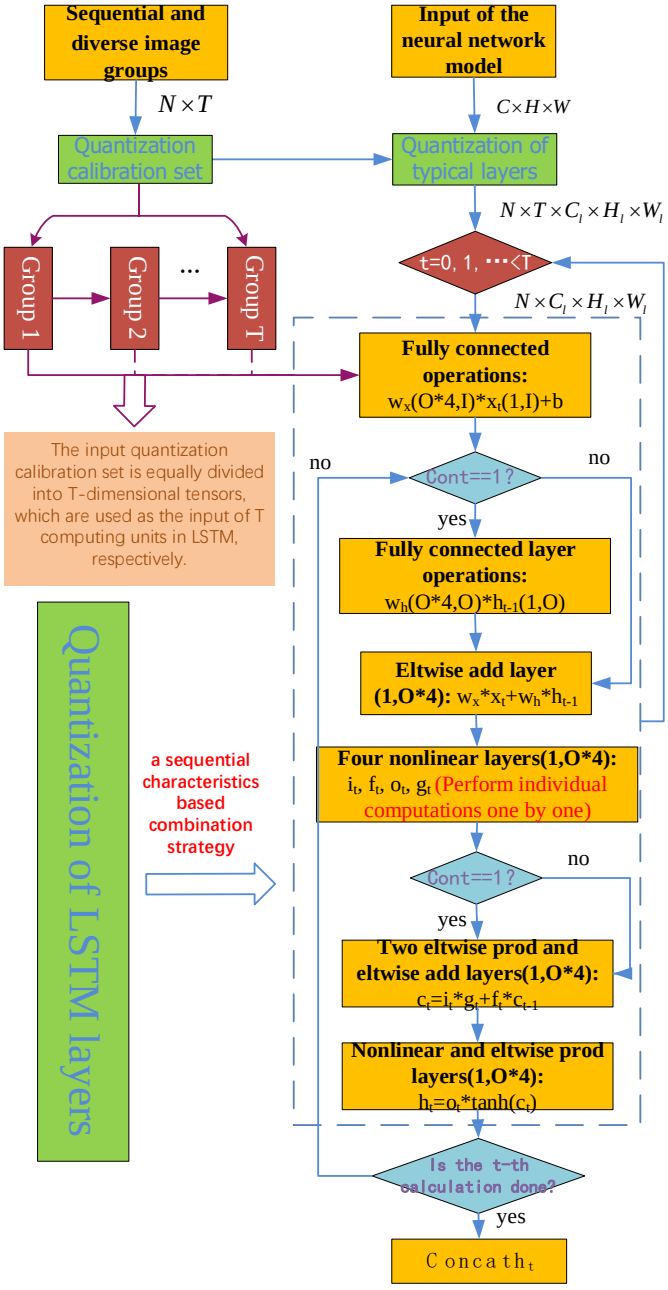


Figure 3. Quantization calibration process of LSTM layers.

More precisely, when generating the quantization parameters, firstly all the $N \times T$ groups data are sent to the regular layer, the activation quantization threshold is selected according to the output data distribution of this layer, and the data dimension of each layer l is $N \times T \times C_l \times H_l \times W_l$. When the LSTM layer is encountered, according to the size of the time step T, the input quantization calibration set is equally divided into T tensors of dimension, which are, respectively, used as the input of the T computing units in the LSTM. Weights and activation quantization thresholds are selected layer by layer according to the weights and activation output distribution ranges for N sets of input data. Finally, the T output tensor sequences are combined into one output as the output of the LSTM layer. At this time, if there is still a regular layer after the LSTM layer, we continue to calibrate the $N \times T$ sets of output data to the subsequent layers.

4. Experiments

In this section, quantization experiments are carried out to verify the proposed quantization method's effectiveness.

First, we describe the implementation details of the proposed quantization method in Section 4.1. Then, most importantly, we compare the experimental results in the relevant literature with the methods proposed in this article from character-level language prediction, word-level language prediction, and image classification in Section 4.2. Then, in Section 4.3, we perform the ablation experiment of the algorithm itself from the perspectives of sequential quantization calibration and activation quantization clipping. In addition, considering the acceleration requirements of the actual hardware architecture implementation, performance analysis is conducted in Section 4.4 for the quantization deployment of CPU, GPU, and NPU, respectively.

4.1. Implementation Details

This section first gives the experimental settings for the following three types of experiments. Then, the principles of quantization implementation and the basic software that the experiment relies on are described in detail.

4.1.1. Experiment Settings

The experimental settings and evaluation metrics are shown in Table 2. In comparison experiments with the state of the art, character-level language prediction application is performed on two datasets: (i) Leo Tolstoy's War and Peace and (ii) Penn Treebank Corpus [23]. Performance Evaluation metrics are bits per character (BPC), variation in BPC, and size of LSTM parameters. In a word-level language-prediction application, Penn Treebank dataset was selected and evaluated by the perplexity per word (PPW), variation in PPW, and relative MSE. The UCF101 dataset is used in image classification applications. Both ablation experiments were performed on the UCF101 dataset and evaluated by the Acc Loss (Top1). Performance experiments are conducted by running speed for the quantization deployment platforms of CPU, GPU, and NPU, respectively.

Table 2. Experimental settings and evaluation metric.

Experiment Type	Application/Platform	Benchmark Datasets	Evaluation Metric
Comparison experiments	Character-level language prediction	Penn Treebank/War and Peace	BPC/Variation/Size
	Word-level language prediction	Penn Treebank	PPW/Variation/ Relative MSE
	Image classification	UCF101	Acc Loss (Top1)
Ablation experiments	W/ and W/O sequential quantization calibration	UCF101	Acc Loss (Top1)
	Comparison between clipping methods	UCF101	Acc Loss (Top1)
Performance experiments	PC(CPU/GPU)	/	Running speed
	NPU	/	Running speed

4.1.2. Experimental Details

The methodology described in Section 3 was implemented as two software tools based on the Caffe. These tools are quantization and simulation software, which are designed to complete the generation of quantization parameters and the NPU calculation simulation [24]. The quantization software pre-analyzes the distribution and range of data at each layer of the network model, calculates the quantization parameters, and generates a guide file for the NPU to map floating-point data to low-precision data. Then,

the simulation software simulates the runtime and NPU calculation process to verify the correctness of the hardware calculation results.

The inputs of the quantization software shown in Figure 4 include the structure file (prototxt) and the parameter file (caffemodel) of the model, and the quantization calibration images. The output is the structure file with the quantization parameters added. The quantization parameters are used to guide the NPU to map floating-point data into integer data. Therefore, the structure file with the quantization parameters added, the parameter file, and the test image dataset make up the inputs to the simulation software. Its output is the accuracy calculation results.

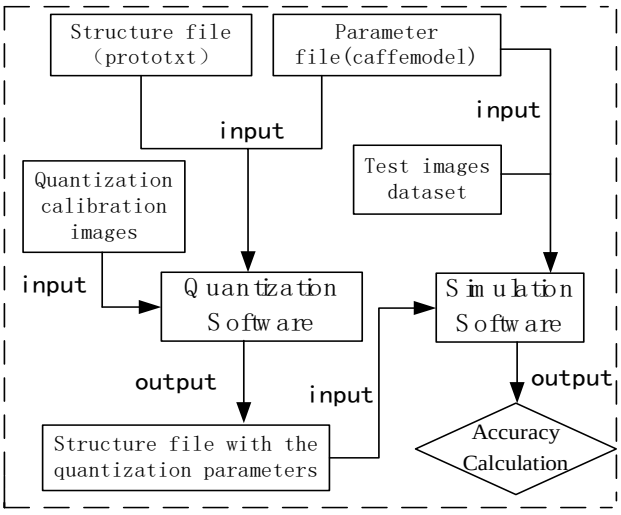


Figure 4. Inputs and outputs of quantization and simulation software.

In detail, this is accomplished by taking two steps for each layer: one for the weights and one for the activations. For the weight, count the weight threshold c of the current layer according to the value range, truncate the weight of the current layer to the [−c, c] interval, and then use the uniform-symmetric quantization method to quantize the weight, as shown in the following equation:

$$quantize(w, n, c) = round(clamp(w, c)/s) \times s \tag{18}$$

where the scaling factor s is obtained by the weight threshold c and the quantization bit width n:

$$s = c/(2^{n-1} - 1) \tag{19}$$

For activation values, the quantization method is similar, except for one point, where the activation threshold t is chosen by finding the optimal value x that minimizes the KL divergence in the original activation value distribution and the quantized activation value distribution:

$$t = \underset{x}{\operatorname{argmin}} D_{KL}(a \Big|\Big| quantize(a, n, x)) \tag{20}$$

Therefore, the quantization algorithm is deployed on the embedded neural network accelerator according to the weights and activation thresholds generated above, namely quantization parameters. According to the quantization parameter threshold and bit width of the current layer, the quantization coefficient is calculated according to Equation (19). Then, the signed integer weight is calculated according to Equation (18) and the activation thresholds are generated according to Equation (20). Finally, the output of the last layer of NPU calculation is inverse-quantized into floating-point values.

4.2. Comparison with the State of the Art

In order to verify the effectiveness of the improved LSTM quantization method, comparison experiments with related works are carried out.

Experiments are performed on character-/word-level language modeling. We compare the full-precision LSTM and popular state-of-the-art quantized LSTMs, including (i) 1-bit LSTMs, binarized using BinaryConnect (BCN) [25], binary weight network (BWN) [26], and loss-aware binarization (LAB) [27]; (ii) 2-bit LSTMs, ternarized using ternary weight networks (TWN) [28] and loss-aware ternarization with approximate solutions (LAT) [29]. Analogous to BinaryConnect, we also include a baseline called TerConnect1, which ternarizes weights to $\{-1, 0, +1\}$ using the same threshold as TWN but does not scale the ternary weights. (iii) Multi-bit LSTMs: For simplicity of notation, we denote all the compared multi-bit quantization methods, including uniform [30], balanced [31], greedy [32], refined greedy [32], and alternating LSTM [33] quantization as Uniform, Balanced, Greedy, Refined, and Alternating, respectively. Above all, we apply these methods to quantize the full precision pre-trained weight (activation is not quantified) of the single LSTM layer.

4.2.1. Character-Level Language Prediction

The LSTM takes as input a character sequence and predicts the next character at each time step. The training objective is the cross-entropy loss over target sequences, and performance is evaluated by bits per character (BPC). Experiments were performed on two benchmark datasets: (i) Leo Tolstoy's War and Peace and (ii) Penn Treebank Corpus. On War and Peace and Penn Treebank, we used a one-layer LSTM with 512 hidden units as in [34]. Adam is used as the optimizer.

Table 3 shows the testing BPC values and size of LSTM parameters. The method with the lowest BPC in each group is highlighted.

Table 3. Test BPC and size (in KB) of LSTM on character-level language modeling.

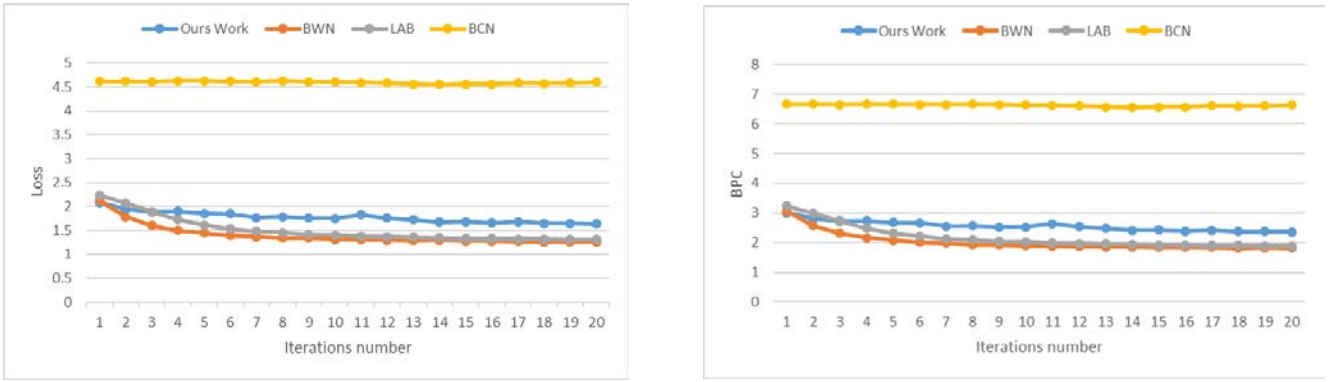
Precision		Quantization Method	Penn Treebank			War and Peace		
			Units = 512			Units = 512		
			BPC	Variation	Size	BPC	Variation	Size
32-bit	[35]	-	1.45	/	4504	1.72	/	4800
1-bit	[26]	BWN	**1.600**	**0.15**	149	**1.822**	**0.102**	158
	[27]	LAB	1.601	0.151	149	1.887	0.167	158
	[25]	BCN	5.840	4.39	149	6.633	4.913	158
	/	Ours	2.133	0.683	149	2.357	0.637	158
2-bit	[28]	TWN	1.517	0.067	289	1.754	0.034	308
	[29]	LAT	1.536	0.086	289	1.828	0.108	308
	/	TCN	5.840	4.39	289	3.493	1.773	308
	/	Ours	**1.446**	**0.004**	289	**1.723**	**0.063**	308
4-bit	/	Ours	**1.572**	**0.122**	548	**1.778**	**0.058**	556
8-bit	/	Ours	**1.471**	**0.021**	1036	**1.783**	**0.063**	1073

Comparison with Full-Precision LSTM: On both the Penn Treebank and the War and Peace datasets, the 2-bit, 4-bit, and 8-bit quantized LSTM performs similarly to the full-precision baseline. Moreover, compared with the full-precision LSTM, the 1-bit quantized LSTM has competitive performance but requires much less storage.

Comparison of Different Quantization Methods: For the quantized 1-bit LSTM, BWN and LAB perform significantly better than BCN. They have an additional scaling parameter, which is empirically smaller than 1, and can thus alleviate the exploding-gradient problem. Regardless of the dataset, compared to other comparison methods, our quantization method achieves the lowest BPC values by 1.446 and 1.723 in the 2-bit

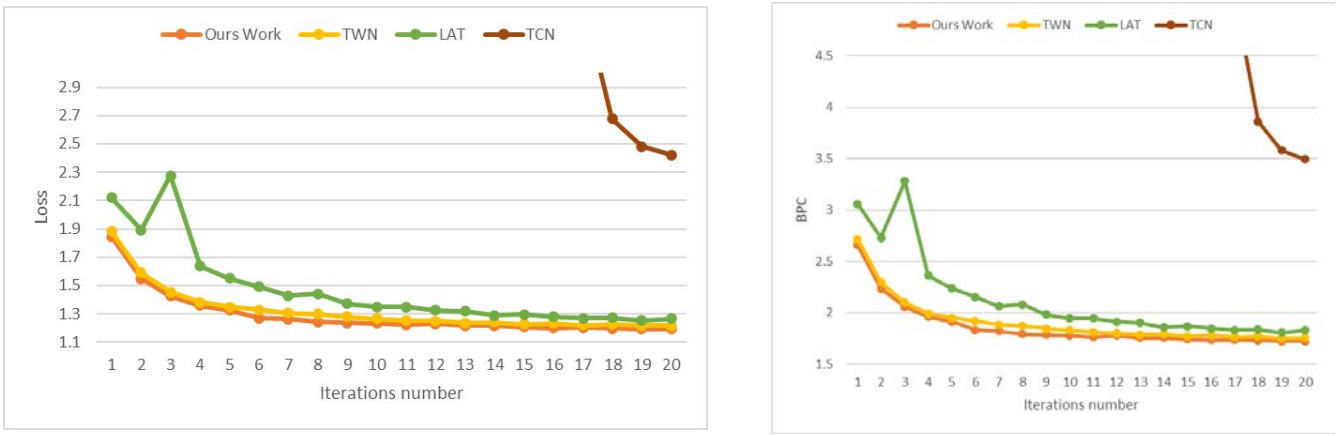
precision, respectively. The 4-bit and 8-bit quantized LSTMs also achieved relatively small quantization errors and have certain competitive performance advantages.

The loss function and BPC iteration curves for quantization retraining at different quantization precisions are shown in Figures 5 and 6. As can be seen from Figure 5, the loss and BPC of our method drops faster than the BCN in the 1-bit quantization precision, which is the same as other methods. As shown in Figure 6, our loss and BPC values converge the fastest of all the comparison methods in the 2-bit quantization precision.



(**a**) (**b**)

Figure 5. Comparison of the loss and BPC in the 1-bit quantization precision. (**a**) The 1-bit iteration curves of the loss. (**b**) The 1-bit iteration curves of the BPC.



(**a**) (**b**)

Figure 6. Comparison of the loss and BPC in the 2-bit quantization precision. (**a**) The 2-bit iteration curves of the loss. (**b**) The 2-bit iteration curves of the BPC.

Comparison with Different Bits: To provide a comprehensive view of the BPC value of our quantization method in different quantization precisions, we conducted experiments by directly quantizing the trained full precision weight. The results on War and Peace and Penn Treebank datasets are shown in Figure 7. Among all the compared multi-bit quantization processes, we can see that our proposed method is still a competitive presence across all varying bits. As the number of bits increases, our method gradually makes more accurate predictions on Character-level language modeling.

Figure 7. The BPC of our quantization method on different datasets across all varying bits.

4.2.2. Word-Level Language Prediction

In this section, we perform experiments to predict the next word on the Penn Treebank dataset. A one-layer LSTM is verified with d = 300 as in [33], and d = 650 as in [34]. We use the same data preparation and training procedures as in [34]. The optimizer is SGD. The quantization methods are retrained considering the perplexity per word (PPW) metric, which is an index of how "confused" the language model is when predicting the next word. For the 2-bit alternating LSTM, only units = 300 are reported in [33].

We proposed a benchmark between our results and other works from the literature. To make the benchmark as fair as possible, we consider that other manuscripts working with LSTM-based models are all based on quantization-aware training (QAT). The focus of our comparison is not on the original floating-point accuracy, but rather on the variation in the metric when applying quantization.

Table 4 shows the testing PPW results and size of LSTM parameters with different units. For the 1-bit quantized LSTM, our method and BCN do not achieve comparable performance to the full-precision counterpart. However, in the case of 2-bit quantized LSTM, our method achieves minimum quantization variations of 3.31 and 4.64 compared to the full-precision baseline at the two hidden element counts, respectively, compared to all other methods. Again, in Table 4, we notice that our method leads to the lowest PPW values, 94.81 and 92.24 lower than all comparison methods shown.

Table 4. Test PPW and variation of LSTM for word-level language modeling on Penn Treebank.

Precision		Quantization Method	Penn Treebank			
			Units = 300		Units = 650	
			PPW	Variation	PPW	Variation
32-bit	[34]	-	91.5	/	87.6	/
1-bit	[26]	BWN	97.17	5.67	85.07	2.53
	[25]	BCN	4330.24	4238.74	2626.91	2539.31
	/	Ours	**157.43**	**65.93**	**135.64**	**48.04**
2-bit	[28]	TWN	95.9	4.4	92.47	4.87
	/	TCN	121.79	30.29	117.35	29.75
	[33]	Alternating	103.1	11.6	-	-
	/	Ours	**94.81**	**3.31**	**92.24**	**4.64**

Notice that the comparison is made in terms of bit widths rather than fixed-point arithmetic because other works do not actually consider the hardware application of the obtained quantized models. Our method, instead, is described considering the subsequent

hardware implementation of our models on architectures completely based on uniform-symmetric quantization strategy.

To further assess the effects of the other different quantization methods on the single LSTM Layer, we also make a comparison between the results obtained with the proposed quantization method and the results from other works in the literature by the PPW and relative mean squared error (MSE) metric. In LSTM architectures, we have considered hidden units 300.

Table 5 records the PPW and relative MSE of quantized weight matrices with the full-precision. The lowest PPW and relative MSE values are shown in bold. As shown in Table 5, we can see that our proposed method can achieve the lowest metric across all quantization methods.

Table 5. Test PPW and Relative MSE of larger bits on Penn Treebank.

Precision		Quantization Method	Penn Treebank		
			Units = 300		
			PPW	Variation	Relative MSE
32-bit	[34]	-	89.8	/	/
3-bit	[30]	Uniform	227.3	137.5	0.404
	[31]	Balanced	9106.4	9016.6	0.745
	[32]	Greedy	99.4	9.6	0.071
	[32]	Refined	95.4	5.6	0.060
	[33]	Alternating	93.8	4	0.043
	/	Ours	**91.67**	**1.87**	**0.025**
4-bit	[30]	Uniform	216.3	126.5	0.302
	[31]	Balanced	8539.8	8450	0.702
	[32]	Greedy	95.0	5.2	0.042
	[32]	Refined	93.1	3.3	0.030
	[33]	Alternating	91.4	1.6	0.019
	/	Ours	**89.52**	**−0.28**	**0.005**

Since quantized models usually have comparable or even better performance than the full-precision baseline, it is worth mentioning that our method is lower than the PPW value of the full-precision model at 4-bit precision. This is by no means uncommon in the contrasting method. In other words, the quantization method proposed by the paper not only surpasses the existing classical quantization algorithm under different bit widths on different datasets but also has comparable or even better performance than the full precision baseline.

4.2.3. Image Classification

Deep neural networks (NN) can be categorized into convolution-based NN (ConvNet), fully connected NN (FCNet), and recurrent NN (RNN) [36]. ConvNet is composed of convolutional and pooling layers and is good at visual feature extraction. FCNet is composed of multiple fully connected layers and is usually used for classification. RNN is composed of fully connected layers with feedback paths and gating operations and performs well in sequential data processing. ConvNet uses Alexnet or Vgg16, etc., to extract spatial features from video frames, and RNN uses LSTM or GRU, etc., to extract sequential features.

In order to verify the advantages of the improved algorithm, this experiment adopted four test models combined with ConvNet + FCNet + RNN for the accuracy test of the classification task. CLDNN [37], LRCN [11], AlexNet-LSTM [38], and Vgg16-LSTM [11] were selected for comparison. Operators covered by the Vgg16-LSTM classification model architecture are very comprehensive. They include both regular operators such as convolution and emerging complex operators such as LSTM. The sequential images are successively extracted from the image and sequence features, and finally, the classification prediction result of Golf Swing is obtained. All the above contrast test models are trained in the

Caffe framework. In this paper, sequential feature-based LSTM quantization algorithm improvements are made to the latter two models. The results are shown in Table 6, where "✔" indicates that an improved sequential quantization calibration method has been added. It is evaluates the accuracy loss (Acc loss) after quantization.

Table 6. Quantization accuracy loss on LSTM-based models.

	Model	Sequential Quantization	Bit-Width	Origin	ConvNet Proportion	FCNet Proportion	RNN Proportion	Acc Loss
[37]	CLDNN	none	All 8	All 16	20%	60%	20%	<2.0%
[11]	LRCN	none	9/8/8/8/9/8/8/8/8/8	All 16	50%	30%	20%	<1.8%
[38]	AlexNet-LSTM	none	AlexNet:9/8/8/6/6/8/8/8, others:12	All 16	50%	30%	20%	<1.8%
/		✔	All 8	All 32	50%	30%	20%	**<0.8%**
[11]	Vgg16-LSTM	none	Vgg16:8,others:12	All 16	76.19%	14.29%	9.52%	<2.0%
/		✔	All 8	All 32	76.19%	14.29%	9.52%	**<1.7%**

In general, since the model applied by our method is based on the accuracy of a floating-point model of 32-bit, our technique improves quantization accuracy on average over the fixed 16-bit mode at baseline. In Vgg16-LSTM, RNN takes up a small proportion (9.52%) of total computations, which means acceleration on RNN is not very noticeable compared to ConvNet and FCNet.

4.3. Ablation Experiments

In this section, ablation experiments with the algorithm are conducted on the UCF101 dataset with a sequential quantization calibration design and the activation quantization clipping methods, respectively. UCF101 dataset described by Soomro et al. in [39] is a video-action recognition classification dataset with 101 categories. A total of 2083 video samples in the first 55 categories were selected as the test dataset on PC and FPGA.

4.3.1. W/ and W/O Sequential Quantization Calibration

To further analyze the effectiveness of the improvements made in sequential quantization calibration design for LSTM, an ablation experiment was performed on the UCF101 dataset.

Different quantization calibration methods are used for AlexNet-LSTM and Vgg-LSTM networks to obtain better accuracy calculation results. The experimental results are shown in Table 7, where "✔" indicates that an improved quantization calibration method was added, and "✔" without identification indicates that the regular layer's quantization calibration method (which is mentioned in subsection C of Section 3) was used. Experimental results show that the accuracy loss of the combined strategy with sequential characteristics adopted for the quantization calibration set is reduced from 1.544% (or 2.707%) to 0.803% (or 1.68%). It confirms that this strategy can improve the accuracy of LSTM quantization deployment.

4.3.2. Comparision between Clipping Methods

The goal of a quantization algorithm is to generate quantization parameters. The activation threshold for each layer is a quantization parameter that is selected according to the dynamic range of the activation value. After investigation, LSTM layer activation quantization clipping methods include KL-divergence, MIN–MAX, AVG–MAX, Easy Quant, ADMM, etc. Here, the first three were selected for the ablation experiments on the UCF101 dataset.

Table 7. Experiment on sequential quantization calibration.

Model	Sequential Quantization	Inference Acc on PC	Quantization Acc on NPU	Acc Loss
AlexNet-LSTM	none	52.808%	51.264%	−1.544%
	✔		52.005%	**−0.803%**
Vgg16-LSTM	none	56.889%	54.182%	−2.707%
	✔		55.209%	**−1.68%**

Table 8 shows the accuracy calculation results of different test models under different activation clipping methods, where "✔" indicates that the corresponding activation clipping method was added. It can be seen from the accuracy loss results that no matter which test model it is based on, and no matter which activation quantization clipping method is used, the quantization loss using the proposed method only differs by 0.803% in the best accuracy and 2.812% in the worst.

Table 8. Experiment on activation quantization clipping.

Model	Sequential Quantization	KL-Divergence	MIN-MAX	AVG-MAX	Acc Loss
AlexNet-LSTM	✔	✔			−0.803%
			✔		−1.104%
				✔	−0.892%
Vgg16-LSTM	✔	✔			−1.68%
			✔		−2.075%
				✔	−2.812%

In order to show the specific values of the inference accuracy on PC and the calculation accuracy on NPU in more detail, we have plotted the following comparative experiment based on different test models. The quantization accuracy experiments in Figure 8 show that, compared with the inference accuracy on PC, the model deployed on the NPU using the proposed method reflects high accuracy loss and the practical value of the proposed quantization method.

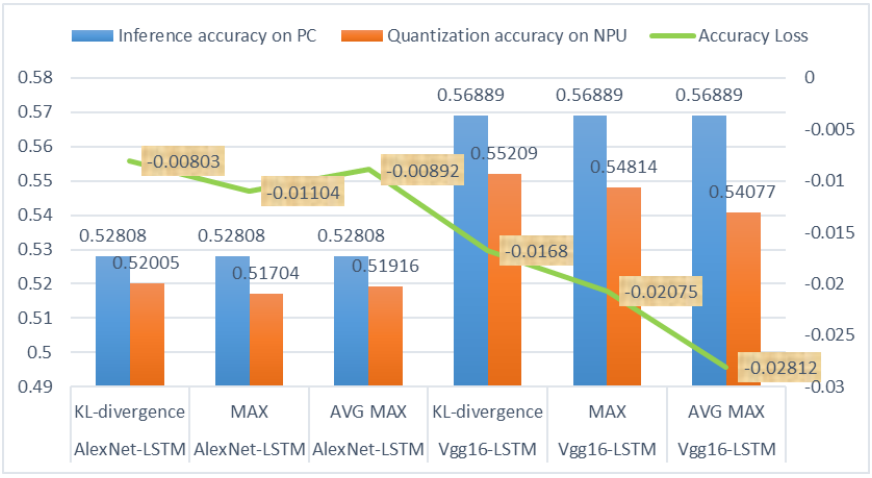


Figure 8. Detailed comparative experiment on different activation quantization clipping methods.

Therefore, these two ablation experiments show that the operator disassembly operation of the LSTM layer can solve the problem that the existing NPU cannot directly compute the LSTM layer.

4.4. Performance Analysis

It is very important to measure the deployment performance of the model on different platforms, and it has a strong reference for improving the quantization scheme and FPGA hardware design. Since the speed of simulating NPU calculations on the PC side is slower, we deployed the proposed method to a real NPU, embedding the core calculations of the model on the neural network accelerator. Quantization and classification prediction are carried out on the neural network accelerator.

The experiments on the PC were carried out on Windows 10. The CPU is Intel(R) Core(TM) i7-8700 K, 3.70 GHz, the GPU is NVIDIA GeForce GTX1070, and the deep learning framework is Torch 1.4.1 and Caffe. The experiment on NPU uses the TIANJI NPU3.0 neural network accelerator proposed by Xi'an Microelectronics Technology Institute [40]. This accelerator is implemented based on Xilinx ZCU102 FPGA, with self-controllable IP and the application development tool chain. It supports multi-model online switching and pipeline parallel acceleration [41]. It supports real-time detection and positioning applications of satellites, spacecrafts, and other targets and can meet the needs of visual ranging and intelligent obstacle avoidance applications.

We test the running speed of LRCN models of different backbone networks on the CPU, GPU, and NPU, respectively. At the same time, the running speed of the backbone network itself on the NPU was also tested. Moreover, the specific speed of the LSTM layer was obtained by the difference between the execution speed of the AlexNet-LSTM or Vgg16-LSTM networks and the backbone network (AlexNet or Vgg16) on the NPU. Note that the execution speed here is the average model deployment time over a single time step. The details are shown in Table 9.

Table 9. Comparison experiment of running speed.

Model	PC(CPU)	PC(GPU)	NPU
AlexNet Alone	/	/	25.7 ms
AlexNet-LSTM	42.6 ms (1.6 times)	34.2 ms (1.3 times)	26.4 ms
LSTM Alone	/	/	**0.7 ms**
Vgg16 Alone	/	/	102.4 ms
Vgg16-LSTM	194.6 ms (1.87 times)	39.7 ms (0.38 times)	103.8 ms
LSTM Alone	/	/	**1.4 ms**

Experiments show that the execution speed of the AlexNet-LSTM model on the neural network accelerator is 1.6 times faster than on the CPU and 1.3 times faster than on the GPU. The Vgg16-LSTM model is executed fastest on the GPU due to the small filter size and large model depth. The execution time is 39.7 ms. The execution time on the neural network accelerator is still 1.87 times faster than on the CPU. Moreover, from the data point of view, a single time step of the LSTM with the hidden state output feature dimension of 256 on the NPU is about 1 ms at 20 W power consumption.

5. Conclusions

Aiming at the problem that the existing neural network accelerator cannot directly execute the LSTM layers, the quantization method integrating LSTM layers is proposed. One of the innovations is that the LSTM layer is split into multiple regular layers supported by the neural network accelerator. The other is that the quantization calibration set takes into account the sequential characteristics of the LSTM layer. The proposed method for LSTM layers differs by only 0.803% in terms of accuracy between the deployment model

on the NPU and the benchmark model on the PC. Execution speed on neural network accelerators is up to 1.87 times faster than on CPUs and 1.3 times faster than on GPUs.

The method not only meets the accuracy and speed requirements of complex sequential forecasting tasks but also quickly and efficiently implements the support of the neural network accelerator for the LSTM layer on the embedded platform. The quantized LSTM model can be widely used in various sequential data-prediction tasks. The quantization parameter generation methods can be migrated to other RNN models. The operators' disassembly operation of LSTM has strong application potential in terms of the quantization aspects of novel operators for future neural network accelerators. It can be widely deployed on various embedded computing platforms with low power consumption and limited resources. What needs to be accomplished in the future is improving the quantization accuracy of LSTM models with mixed-bit-width configurations and deploying them on hardware platforms that support mixed-bit-width architectures.

Author Contributions: Conceptualization, Y.W. and Z.M.; Methodology, Y.W.; Software, Y.W. and Z.Y.; Validation, Z.M. and Z.Y.; Formal analysis, Y.W.; Investigation, Y.W. and Z.Y.; Resources, Z.M.; Data curation, Y.W. and Z.Y.; Writing—original draft, Y.W.; Writing—review & editing, Z.M.; Visualization, Y.W.; Project administration, Z.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

Notations	Definition
i_t, f_t, o_t	input gate, forget gate, output gate
g_t, c_t, h_t	cell input vector, cell state vector, hidden state vector
$W_x = [W_{xi}, W_{xf}, W_{xo}, W_{xg}]$	weight matrix of the input state
$W_h = [W_{hi}, W_{hf}, W_{ho}, W_{hg}]$	weight matrix of the hidden node
$b = [b_{xi}, b_{xf}, b_{xo}, b_{xg}]$	bias term
T	time step
Cont	continuity identifier
I	input feature dimension
O	output feature dimension
$x = \{x_0, x_1 \cdots x_t, \cdots x_{T-1}\}$	input of the LSTM layer: a tensor of (T, I)
$h = \{h_0, h_1 \cdots h_t, \cdots h_{T-1}\}$	output of the LSTM layer: a tensor of (T, O)
s	scaling factor
c	weight threshold
t	activation threshold
n	quantization bit width

References

1. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning. *Trained Quantization Huffman Coding Fiber* **2015**, *56*, 3–7.
2. Gong, Y.; Liu, L.; Ming, Y.; Bourdev, L. Compressing deep convolutional networks using vector quantization. *arXiv* **2014**, arXiv:1412.6115.
3. Wang, Y.J.; Ma, Z.; Yang, Z.M. A new quantization deployment method of neural network models integrating LSTM layers. In Proceedings of the 2022 5th International Conference on Pattern Recognition and Artificial Intelligence (PRAI), Chengdu, China, 19–21 August 2022; pp. 1299–1303.
4. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef] [PubMed]
5. Cheng, Y.Q.; He, Z.Z.; Ma, Z.; Bi, R.X.; Mao, Y.H. Intelligent target detection algorithm for embedded FPGA. *Microelectron. Comput.* **2021**, *38*, 87–92.

6. Mikolov, T.; Sutskever, I.; Kai, C.; Corrado, G.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the NIPS'13: Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; pp. 3111–3119.
7. Zaremba, W.; Sutskever, I.; Vinyals, O. Recurrent neural network regularization. *arXiv* **2014**, arXiv:1409.2329.
8. Zhang, L.X.; Hu, W.X. Research on character relationship extraction in Chinese text based on bidirectional GRU neural network and double-layer attention mechanism. *Comput. Appl. Softw.* **2018**, *35*, 130–135.
9. Yang, Z.M.; He, Z.Z.; Ma, Z.; Yang, J. An LSTM acceleration method based on embedded neural network accelerator. In Proceedings of the 4th International Conference on Algorithms, Computing and Artificial Intelligence, Sanya, China, 22–24 December 2021.
10. Amodei, D.; Ananthanarayanan, S.; Anubhai, R.; Bai, J.; Zhu, Z. Deep speech 2: End-to-end speech recognition in English and mandarin. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015.
11. Donahue, J.; Hendricks, L.A.; Rohrbach, M.; Venugopalan, S.; Guadarrama, S. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 677–691. [CrossRef] [PubMed]
12. Zaman, S.K.; Jehangiri, A.I.; Maqsood, T.; Umar, A.I.; Khan, M.A.; Jhanjhi, N.Z. COME-UP: Computation offloading in mobile edge computing with LSTM based user direction prediction. *Appl. Sci.* **2022**, *12*, 3312. [CrossRef]
13. Krishnamoorthi, R. Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper. *arXiv* **2018**, arXiv:1806.08342v1.
14. Szymon, M. 8-bit inference with tensorrt. *GPU Technol. Conf.* **2017**, *2*, 7.
15. Wu, D.; Tang, Q.; Zhao, Y.; Zhang, M.; Zhang, D. EasyQuant: Post-training quantization via scale optimization. *arXiv* **2020**, arXiv:2006.16669.
16. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and training of neural networks for efficient integer arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018.
17. Zhang, P.; Ouyang, W.; Zhang, P.; Xue, J.; Zheng, N. SR-LSTM: State refinement for LSTM towards pedestrian trajectory prediction. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 12077–12086.
18. Zhang, Y.W.; Wang, C.; Gong, L.; Lu, Y.; Zhou, X. A power-efficient accelerator based on FPGAs for LSTM network. In Proceedings of the 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, 5–8 September 2017; pp. 629–630.
19. Zhang, Y.W.; Wang, C.; Gong, L.; Lu, Y.; Zhou, X. Implementation and optimization of the accelerator based on FPGA hardware for LSTM Network. In Proceedings of the 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), Guangzhou, China, 12–15 December 2017; pp. 614–621. [CrossRef]
20. Chang, G.; Neil, D.; Ceolini, E.; Liu, S.C.; Delbruck, T. DeltaRNN: A power-efficient recurrent neural network accelerator. In Proceedings of the 2018 ACM/SIGDA International Symposium, Monterey, CA, USA, 25–27 February 2018.
21. Zeng, X.; Zhi, T.; Zhou, X.; Du, Z.; Guo, Q.; Liu, S.; Wang, B.; Wen, Y. Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. *IEEE Trans. Comput.* **2020**, *69*, 968–985. [CrossRef]
22. Alom, M.Z.; Moody, A.T.; Maruyama, N.; Essen, B.C.V.; Taha, T.M. Effective quantization approaches for recurrent neural networks. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8. [CrossRef]
23. Taylor, A.; Marcus, M.; Santorini, B. The Penn treebank: An overview. In *Treebanks*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 5–22.
24. Wei, L.; Ma, Z.; Wang, Y.J.; Yang, C.J. An Adaptive Quantization Method for Neural Network Accelerators Running on FPGA. Chinese Invention Patent CN202110057445.3, 15 January 2021.
25. Courbariaux, M.; Bengio, Y.; David, J.P. BinaryConnect: Training deep neural networks with binary weights during propagations. In Proceedings of the Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 3105–3113.
26. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. XNOR-Net: ImageNet classification using binary convolutional neural networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016.
27. Hou, L.; Yao, Q.; Kwok., J.T. Loss-aware binarization of deep networks. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
28. Li, F.; Liu, B. Ternary weight networks. *arXiv* **2016**, arXiv:1605.04711.
29. Hou, L.; Yao, Q.; Kwok, J.T. Loss-aware weight quantization of deep networks. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
30. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv* **2016**, arXiv:1609.07061.
31. Zhou, S.C.; Wang, Y.Z.; Wen, H.; He, Q.Y.; Zou, Y.H. Balanced quantization: An effective and efficient approach to quantized neural networks. *J. Comput. Sci. Technol.* **2017**, *32*, 667–682. [CrossRef]
32. Guo, Y.W.; Yao, A.B.; Zhao, H.; Chen, Y. Network sketching: Exploiting binary structure in deep CNNs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017.

33. Chen, X.; Yao, J.; Lin, Z.; Ou, W.; Zha, H. Alternating Multi-Bit Quantization for Recurrent Neural Networks. *arXiv* **2018**, arXiv:1802.00150.
34. Ardakani, A.; Ji, Z.; Smithson, S.C.; Meyer, B.H.; Gross, W.J. Learning recurrent binary/ternary weights. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
35. Hou, L.; Zhu, J.; Kwok, J.T.; Gao, F.; Qin, T.; Liu, T.Y. Normalization helps training of quantized LSTM. In Proceedings of the Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019.
36. Yin, S.Y.; Ouyang, P.; Tang, S.B.; Tu, F.B.; Li, X.D.; Zheng, S.X.; Lu, T.Y.; Gu, J.Y.; Liu, L.B.; Wei, S.J. A high energy efficient reconfigurable hybrid neural network processor for deep learning applications. *IEEE J. Solid-State Circ.* **2018**, *53*, 968–982. [CrossRef]
37. Sainath, T.N.; Vinyals, O.; Senior, A.; Sak, H. Convolutional, long short-term memory, fully connected deep neural networks. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, Australia, 19–24 April 2015; pp. 4580–4584.
38. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the Neural Information Processing Systems Conference, Vancouver, BC, Canada, 8–14 December 2012; pp. 1097–1105.
39. Soomro, K.; Zamir, A.R.; Shah, M. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv* **2012**, arXiv:1212.0402.
40. Jiao, F.; Ma, Y.; Bi, S.Y.; Ma, Z. Design of Instruction Control System for Neural Network Accelerator. *Microelectron. Comput.* **2022**, *39*, 78–85.
41. Ma, Y.; Bi, S.Y.; Jiao, F.; Ma, Z.; Zhou, F.; Nie, Y.C. A CNN Accelerator with High Bandwidth Storage. Chinese Invention Patent CN20210921363.9, 11 August 2021.