Implementation of a batch normalized deep LSTM recurrent network on a smartphone for human activity recognition

DOI: 10.1109/BHI.2019.8834480

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):

Published in:
IEEE-EMBS BHI 2019

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester’s Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.
Implementation of a batch normalized deep LSTM recurrent network on a smartphone for human activity recognition

Tahmina Zebin  
School of Computer Science and Engineering  
University of Westminster  
London, UK  
e-mail: t.zebin@westminster.ac.uk

Ertan Balaban, Krikor B. Ozanyan and Alexander J. Casson  
School of Electrical and Electronic Engineering  
University of Manchester  
Manchester, UK  
e-mail: {ertan.balaban, k.ozanyan, alex.casson}@manchester.ac.uk

Niels Peek  
Health e-Research Centre  
University of Manchester  
Manchester, UK  
e-mail: niels.peek@manchester.ac.uk

Abstract—In this paper we present a Long-Short Term Memory (LSTM) deep recurrent neural network (RNN) model for the classification of human daily life activities by using the accelerometer and gyroscope data of a smartphone. The proposed model was trained by using the open-source TensorFlow library, optimised and deployed on an Android smartphone. Hardware resource requirements for the implementation are empirically investigated and the effect of data quantization on the accuracy of the implementation is discussed. In addition, we profile the power budget for running the proposed model on smartphone. Results of this work will be of use for deep learning implemented on edge computing devices, which leverages the user privacy as the raw data never leaves the person.

I. INTRODUCTION

Technological advances and competition in the electronics market have pushed the performance of the low and mid-range microprocessor, and made edge computation a viable option over the conventional centralized computing systems. Edge computing greatly reduces the amount of data to be transmitted, and hence the power budget, as even in conservative assumptions it approximately requires 50nJ/bit for widely used wireless transmitters [1]. Since data are processed at the edge devices, virtually there is no communication latency, which is desirable for real time operations. Possibly the best outcome of the edge computing is the improved privacy, because personal data can stay inside the devices memory and do not need to be carried over vulnerable communication links.

Deep learning techniques have been applied to a variety of fields and proved their usefulness in many applications such as speech recognition, language modelling and video processing. Models such as Convolutional Neural Networks (CNN) and RNN employ a data driven approach to learning discriminative features from raw sensor data to infer complex, sequential, and contextual information in a hierarchical manner [11]. They are highly suited for exploiting temporal correlations in data sets that makes them suitable for applications such as human activity recognition (HAR) classification, where potentially a large amount of data is available and human movements are encoded in a sequence of successive samples in time and the current activity is not defined by one small window of data alone.

In particular the fusion of pre-trained deep learning models and mobile computing is of special interest and has enabled numerous applications such as smart activity tracking, intelligent personal assistant, real time language translation on smartphones and smartwatches. Several authors have proposed deep learning models, and implemented them on smartphones for facial expressions detection using camera data [2] or human activity detection using accelerometers built-in to smartphones [3], [4], [5]. Commercial human activity monitoring devices relying on inertial sensors such as Fitbit have gained popularity for daily activity tracking. However, the accuracy of these devices is not satisfactory for applications such as gait analysis or step counting for slow walking [6]. Type of the activity might affect the accuracy as well: [10] reported that although step counting for over ground walking with errors less than 1% is achievable, in the case of stair walking detection such devices might overestimate distance by as high as 45%.

Hybrid deep learning models combining CNN and LSTM RNNs have achieved a high level of classification accuracy [8]. [7] used a DeepConvLSTM network and achieved 91.7% accuracy. [8] reported an F1 score of 90.8% with a factorized LSTM network. Stacked LSTM architecture has the capability of providing better generalization and robust temporal pattern learning [12], and to the best of our knowledge no previous study has used it for HAR classification. Although state of the art inertial sensors incorporates accelerometers and gyroscopes, most studies have focused solely on accelerometer data. LSTMs architecture presented in this work uses both accelerometer and gyroscope data for the feature-free classifi-
cation of human activities. To make the network fast and robust we have employed dropout regularization and the recently introduced batch normalization method [13].

The remainder of this paper is organized as follows: Section II introduces the model architecture for the proposed LSTM and CNN models. Deployment, compilation and training steps are described, and also a detailed description of performance and optimization strategy is given. Section III gives the details of resource requirements and power budget for the implementation.

II. METHODS

A. Deep neural network architecture: LSTM and CNN

Fig. 1. Proposed deep LSTM network architecture for HAR classification.

A schematic diagram of our multi-layer stacked architecture LSTM network for multi-class HAR classification is presented in Fig. 1. The model architecture is novel in its use of longer temporal sequences in the LSTM and its use of batch normalization for HAR with the RNN architecture when compared to ones reported in the literature [9]. The data input of the model is multi-dimensional, containing three channels from a 3-axis accelerometer, and three from a 3-axis gyroscope. These six parallel 1-D time series data are put into a 3-D structure required by an LSTM with the specific number of neurons in one dimension, the number of memory steps to process per time step in another dimension, and different sensor channels on the third dimension. For the architecture shown in Fig. 1 we created a matrix of the form: No. of data rows × 128 samples/sequence × 6 channels. In every data row 127 previous samples were arranged to work as a memory for the new data. To implement and compare both floating-point (float32) and quantized (uint8) precision arithmetic we used TensorFlow lite (TFlite) as the base deep learning library. Table I shows the functional components of the model. We trained the model for the classification of basic locomotor activities (walking, walking up and down the stairs), sedentary (sit or stand) and sleep (lying down) activities. We processed the data from a 20 subjects dataset [14], collected from a waist mounted inertial sensor. At the very first layer, the six channel input time series (segmented as 1 × 128 windows) from 3-axis accelerometer and 3-axis gyroscope is processed by 12 different convolution filters. These filters create a non-linear, distributed representation of the input and can be of variable kernel sizes. These can then be applied over the entire input time series with a specific stride length. Next a max-pooling layer is used to down-sample the temporal features (such as slopes/changes in the time series signal) that the convolution layer has just extracted. We stacked three convolution and pooling layers in our model to obtain more detailed features obtained by a previous layer. We doubled the number of filters after each convolution and pooling layer. The feature map extracted by the filters of the final CNN layer is then flattened out to be used as an automatically extracted feature set by any other classifier. To make a final decision on which activity is present at each point in time the network is converted into a classifier using a fully connected hidden layer with 50 neurons followed by an output dense layer of five neurons with a softmax classifier to provide probabilistic assignments of labels from the raw data. We have utilized the sequential model and with dense, conv1D, maxpooling1D, dropout, and batch normalization layers.

B. Model deployment, compilation and training

Fig. 2 shows the main computational steps required to have the model deployed on an embedded hardware. The first block of the work-flow corresponds to training stage of the deep learning model on TensorFlow, the rest of the stages are required for optimization before the model can be deployed on an edge device.

### Table I
**Description of the convolutional neural network model**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Six channel accelerometer and gyroscope data</td>
</tr>
<tr>
<td>Three convolution layers</td>
<td>Each layer has Conv1D, Maxpooling1D, ReLU activation components</td>
</tr>
<tr>
<td>Output</td>
<td>Features from the previous CNN layers</td>
</tr>
<tr>
<td>Reshape</td>
<td>Flatten to 1-D</td>
</tr>
<tr>
<td>Hidden layer</td>
<td>50 hidden units each, Activation: Rectified Linear Unit</td>
</tr>
<tr>
<td>Batch normalization</td>
<td>Batch size=64</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.2</td>
</tr>
<tr>
<td>Final layer</td>
<td>Six outputs; Activation: 6-way softmax</td>
</tr>
<tr>
<td>Leasing rate</td>
<td>0.002</td>
</tr>
<tr>
<td>Loss function</td>
<td>Categorical cross-entropy</td>
</tr>
</tbody>
</table>
TABLE II
MEMORY CONSUMPTION AND EXECUTION TIME SUMMARY BY NODE TYPE

<table>
<thead>
<tr>
<th>Node type</th>
<th>Count</th>
<th>Memory (KB)</th>
<th>Avg. execution time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv1D weights</td>
<td>72</td>
<td>5869.953</td>
<td>244.813</td>
</tr>
<tr>
<td>BiasAdd</td>
<td>73</td>
<td>5869.953</td>
<td>244.813</td>
</tr>
<tr>
<td>ReLU</td>
<td>72</td>
<td>5869.953</td>
<td>5.700</td>
</tr>
<tr>
<td>MaxPool</td>
<td>18</td>
<td>358.848</td>
<td>5.700</td>
</tr>
<tr>
<td>Concat</td>
<td>1</td>
<td>892.096</td>
<td>1.081</td>
</tr>
<tr>
<td>MatMul</td>
<td>1</td>
<td>4.032</td>
<td>0.660</td>
</tr>
<tr>
<td>Softmax</td>
<td>1</td>
<td>4.032</td>
<td>0.660</td>
</tr>
<tr>
<td>Reshape</td>
<td>1</td>
<td>0</td>
<td>1.081</td>
</tr>
</tbody>
</table>

C. Performance evaluation and optimization

To evaluate the performance of the model, we fed data from 14 volunteers, with approximately 7500 labelled activities as training data to the model and the data from 6 volunteers, 3000 labelled activities, as a test dataset. The test set was separated entirely from the training dataset. To avoid over-fitting the model with training data, 20% of the training dataset was spared as a validation set. We obtained almost 96.4% accuracy on the test set for the deep model described in Table I.

Due to limited resources of edge devices we investigated the feasibility of the frozen graph of the model in terms of memory and floating-point operations requirement. Table II lists the memory allocation analysis based on the number of node types we had on the implemented model. The CNN model consumed about 16 MB of memory when transformed and saved in TensorFlows protocol buffer (.pb) format. The memory consumption analysis presented in Table II is an indicator of how many operations (ops) are there and the time consumed to complete these ops. These values give us the opportunity to optimize the architecture or decide for alternative models to reduce these numbers according to the resource availability on an embedded device. The trained models were then optimized and made graph ready to convert to TFlite. We have used the TensorFlows graph transforms and summarize graph to freeze the model and conduct the analysis on the models profile and types of operation necessary for inference. Additionally, we have utilized TensorBoards visualization to evaluate the graph after each step and to identify any unsupported layer that needs custom implementation.

To make the model suitable for resource limited embedded devices we applied the quantization scheme shown in Fig. 4(a) to the trained model. Here the 32-bit floating-point precision weights are reduced to 8-bits unsigned integers, i.e. quantized. Since only the weights are quantized this can be done without requiring any validation data. Another approach would be to quantize the activation values along with the weights of the saved model. A floating-point model to 8-bit precision conversion is applied by calculating the quantizer parameters for all the values to be quantized. Since activations need to be quantized, it is necessary to calculate the dynamic ranges of activations and have that data for calibration. Since quantization should be modelled in both forward and backward passes, we conducted experiments by inserting quantization ops at different layers as shown in Fig. 4(a) to simulate the error introduced by quantization. Since the model graph is frozen and transformed in place, the expected behaviour of the nodes and tensors in different layers were changed.

III. RESULTS

The results for the weight and activation quantization experiments are shown in Fig. 4(b) and Table III. Moving from 32-bits to 8-bits leads to 4× reduction in memory. Fig. 4(b) shows the actual memory footprint of various learning models such as the dense neural network (DNN), CNN and an LSTM model built using the same dataset. As can be seen from the bar diagram and from the memory footprint column in Table III, the CNN model was 7.62 times smaller in size when post training quantization is applied on the weights and activation of the model. This version would also be more suitable for low-end microprocessors that do not support floating-point arithmetic. Our experiments suggested that we can variably quantize (i.e. discretize) the range to only record some values with float32 that have more significant contribution on the model accuracy and round off the rest to unit8 values to still take advantage of the optimized model. In Table III, we also observed a slight increase in prediction accuracy for the test dataset for DNN and LSTM models because of the salient and less noisy weight values available in the saved model for activity prediction. The table also suggests that the minimization of model parameters may not necessarily lead to the optimal network in terms of performance. This
<table>
<thead>
<tr>
<th>Network Type</th>
<th>Memory footprint (MB)</th>
<th>Training (%)</th>
<th>Validation (%)</th>
<th>Test (%)</th>
<th>Memory footprint (MB)</th>
<th>Size ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense Neural Network (DNN)</td>
<td>8.2</td>
<td>97.99</td>
<td>89.04</td>
<td>86.55</td>
<td>87.6</td>
<td>5.98</td>
</tr>
<tr>
<td>Long-Short Term Memory Network (LSTM)</td>
<td>17.4</td>
<td>98.38</td>
<td>96.69</td>
<td>92.2</td>
<td>93.51</td>
<td>4.24</td>
</tr>
<tr>
<td>Convolutional Neural Network (CNN)</td>
<td>16</td>
<td>99.23</td>
<td>95.92</td>
<td>96.4</td>
<td>93.6</td>
<td>7.62</td>
</tr>
</tbody>
</table>

Fig. 4. (a) Weight and activation quantization scheme (b) Memory footprint of various deep learning models in terms of weight and activation, the float32 version is the size of originally trained model and the unit8 is the version of the model when both weight and activation quantization is applied.

means a selective quantization approach (e.g. using k-means clustering) can make the process more efficient than straight linear quantization.

Fig. 3(a) shows the screenshot of the implemented Android app and its output during upstairs walking. Fig. 3(b) shows the absolute power consumption of the implementation. We used Trepn Profiler for these measurements [15]. As it can be seen the average absolute power consumption is roughly 40 mW. In comparison the same value measured for YouTube is roughly 116 mW, which suggest that our HAR application is relatively low power and computationally effective.

IV. CONCLUSION

We have presented deep neural network models for the classification of 6 daily-life activities using raw accelerometer and gyroscope data of a smartphone as the input. The proposed model has been empirically analysed for the memory requirement and execution time for its operations with the objective of deploying the model in wearable or edge devices. We observed that almost all the size and execution time reduction in the optimized model are mostly due to weight quantization, potentially allowing them to be quantized differently to the activations and allowing further optimizations. Proposed model has been validated and successfully implemented on a smartphone. Results of this study would be useful for making smart wearables and devices which stand alone from the cloud, potentially improving the user privacy.

REFERENCES