Appendix for iRNN: Integer-only Recurrent Neural Network

Eyyüb Sari, Vanessa Courville, Vahid Partovi Nia

Huawei Noah's Ark Lab
{eyyub.sari, vanessa.courville, vahid.partovinia}@huawei.com

Appendix

Specific details on LSTM-based models

For BiLSTM cells, nothing stated in section Integer-only LSTM network is changed except that we enforce the forward LSTM hidden state $\overrightarrow{\mathbf{h}}_t$ and the backward LSTM hidden state $\overleftarrow{\mathbf{h}}_t$ to share the same quantization parameters so that they can be concatenated as a vector. If the model has embedding layers, they are quantized to 8-bit as we found they were not sensitive to quantization. If the model has residual connections (e.g. between LSTM cells), they are quantized to 8-bit integers. In encoder-decoder models the attention layers would be quantized following section Integer-only attention. The model's last fully-connected layer's weights are 8-bit quantized to allow for 8-bit matrix multiplication. However, we do not quantize the outputs and let them remain 32-bit integers as often this is where it is considered that the model has done its job and that some postprocessing is performed (e.g. beam search).

Refresher on quantization

This section is intended for readers who are novices in model quantization and Jacob et al. (2017)'s work. For the examples, we focus on 8-bit quantization (i.e. $b = 8$) unless stated otherwise.

Quantization-Aware Training We perform Quantization-Aware Training (QAT), in which the quantization effects are simulated during training to let the network adapt to the quantization error. Simulating quantization is done via Fake Quantization where a floating-point value is first quantized, then dequantized to the corresponding floating-point value in the quantization set. The fake quantization function is a non-differentiable operation; therefore we set its derivative to be the straight-through estimator (STE) (Hinton 2012)

$$\text{fakequant}(x) = \text{r}(x), \;\; \text{fakequant}'(x) = 1 \tag{1}$$

For a weight matrix, the quantization range $x_{\min}$ and $x_{\max}$ are respectively the minimum and maximum values within the matrix. We use symmetric quantization as weight distributions are typically symmetric around zero and would enable better speedups at inference time. Activations, on the other hand, are dynamic; thus, we keep track of a moving average of their minimum and maximum values as described in Jacob et al. (2017).

Rules about integer-only operations We describe how to correctly map a model to integer-only operators for all element-wise multiplication and additions. Refer to Jacob et al. (2017) describes how to do this for matrix-multiplications and other operators. At inference, the inputs to any operation are already quantized. For the following explanations, we define two quantized inputs q_a with scaling factor S_a and zero-point Z_a, q_b with scaling factor S_b and zero-point Z_b. We also define q_c with scaling factor S_c and zero-point Z_c that will be the quantized output of the operations. Note that all the scaling factors and zero-point come from the set of quantization parameters determined during QAT. The multiplication operation does not require q_a and q_b to share quantization parameters, therefore it is defined as,

$$q_c = \Big\lfloor \frac{S_a S_b}{S_c}\Big(q_a q_b - q_a Z_b - q_b Z_a + Z_a Z_b\Big)\Big\rceil + Z_c, \tag{2}$$

where the multiplication $q_a q_b$ is a b-bit multiplication (example given in Appendix). Addition can take two forms based on the quantization parameters of q_a and q_b. If they share the same quantization parameters (i.e. $S_b = S_a$ and $Z_b = Z_a$), it can be computed as follows,

$$q_c = \Big\lfloor \frac{S_a}{S_c}(q_a + q_b - 2Z_a)\Big\rceil + Z_c \tag{3}$$

where $q_a + q_b$ is a b-bit addition. However, if they do not share the same quantization parameters,

$$q_c = \Big\lfloor \frac{S_a}{S_c}(q_a - Z_a) + \frac{S_b}{S_c}(q_b - Z_b)\Big\rceil + Z_c \tag{4}$$

rescaling q_a and q_b is needed, which means the addition has to be computed with more bitwidth, e.g. 32-bit (examples given in Appendix). All constants involving floating-points values (e.g. scaling factors) can be expressed as integer fixed-point multipliers, therefore no floating-points operations are required at inference (Appendix). Refer to Appendix for more information.

Computing scaling factor and zero-point Given a range $[x_{\min}, x_{\max}]$, e.g. $[-1, 1]$ and a bitwidth b, e.g. 8; the scaling

factor S_x and zero-point Z_x are computed as follow,

$$S_x = \frac{x_{\max} - x_{\min}}{2^b - 1} = \frac{2}{255} \approx 0.0078 \quad (5)$$

$$Z_x = \left\lfloor \frac{-x_{\min}}{S_x} \right\rceil = \left\lfloor \frac{1}{S_x} \right\rceil = 128. \quad (6)$$

To not clutter the writing, we will assume S_x exactly equals 0.0078. If $b = 16$, we are able to represent more numbers, i.e. the scaling factor will be smaller,

$$S_x = \frac{x_{\max} - x_{\min}}{2^b - 1} = \frac{2}{65535} \approx 3.05 \times 10^{-5} \quad (7)$$

$$Z_x = \left\lfloor \frac{-x_{\min}}{S_x} \right\rceil = \left\lfloor \frac{1}{S_x} \right\rceil = 32768. \quad (8)$$

Quantizing a floating-point number Given a number within the previously defined range , i.e. $x \in [-1, 1]$, and the scaling factor S_x (5) and zero-point Z_x (6), we can quantize it following $q_x = \mathrm{q}(x) = \left\lfloor \frac{x}{S_x} \right\rceil + Z_x$. For instance, for $x = 0.2$,

$$\mathrm{q}(x) = \left\lfloor \frac{x}{S_x} \right\rceil + Z_x = \left\lfloor \frac{0.2}{0.0078} \right\rceil + 128 = 154, \quad (9)$$

hence $q_x = 154$. The corresponding floating-point representation of q_x, r_x, is

$$\mathrm{r}(x) = S_x(q_x - Z_x) \quad (10)$$

$$= 0.0078(154 - 128) = 0.2028. \quad (11)$$

Hence, after quantizing the floating-point value 0.2, its dequantized value is 0.2028, so the error introduced is $\epsilon = |x - r_x| = 0.0028$. The purpose of QAT with fake quantization (1) is to try to force the network to be insensitive to noise introduced by these ϵ errors. The error introduced by quantization is bounded by is $\epsilon \leq \frac{S_x}{2}$. The smaller the scaling factor, the better the precision. The solutions to make the scaling factors small are either to have a smaller range $[x_{\min}, x_{\max}]$ or to use a bigger bitwidth b. In practice, for a given bitwidth, we would like to have a large range including all possible values the network can produce, but also to have small errors. These two principles are contradictory, it is the range-precision trade-off.

Quantized multiplication Define $u \in [u_{\min}, u_{\max}] = [-1, 1]$ and $w \in [w_{\min}, w_{\max}] = [0, 5]$, the multiplication between two numbers from those ranges will fall into $[z_{\min}, z_{\max}] = [-5, 5]$. For 8-bit quantization, we have $S_u \approx 0.0078$, $Z_u = 128$, $S_w \approx 0.0196, Z_w = 0, S_z \approx 0.0392, Z_z = 128$. Given $u = -0.8$ and $w = 2.3$, we have $q_u = 25$ and $q_w = 117$. Following (2),

$$q_z = \left\lfloor \frac{S_u S_w}{S_z} \Big(q_u q_w - q_u Z_w \right. \quad (12)$$

$$\left. - q_w Z_u + Z_u Z_w \Big) \right\rceil + Z_z \quad (13)$$

$$= \left\lfloor \frac{0.0078 * 0.0196}{0.0392} \Big(25 * 117 \right. \quad (14)$$

$$\left. - 25 * 0 - 117 * 128 + 128 * 0 \Big) \right\rceil + 128 \quad (15)$$

$$= 81. \quad (16)$$

The floating-point representation of q_z is $r_z = -1.8424$. r_z is close to $uv = -1.8399$. Note that we lost precision at two levels, the first time when quantizing u and v, then the second time when quantizing z, the output of the multiplication.

Quantized addition As mentioned in section Quantization, addition with quantized numbers can take two forms. The first form is when the two numbers to be added share the same scaling factor and zero-point. For instance, given $x_1 = -0.3, x_2 = 0.7$ from $[-1, 1]$, and $S_x = 0.0078, Z_x = 128$, we have $q_{x_1} = 90$ and $q_{x_2} = 218$. The result value y will fall into the range $[-2, 2]$, therefore $S_y \approx 0.0157$ and $Z_y = 128$. Then, because they share the same quantization parameters, following (3),

$$q_y = \left\lfloor \frac{1}{S_y} \Big(S_x(q_{x_1} - Z_x) + S_x(q_{x_2} - Z_x) \Big) \right\rceil + Z_y \quad (17)$$

$$= \left\lfloor \frac{S_x}{S_y} \Big(q_{x_1} + q_{x_2} - 2Z_x \Big) \right\rceil + Z_y \quad (18)$$

$$= \left\lfloor \frac{0.0078}{0.0157} \Big(90 + 218 - 256 \Big) \right\rceil + 128 \quad (19)$$

$$= 154. \quad (20)$$

We have $r_y = 0.4082$, while $x_1 + x_2 = 0.3999$. The second form is when the two numbers do not share the same scaling factor and zero-point. Define $a \in [a_{\min}, a_{\max}] = [-1, 1]$ and $b \in [b_{\min}, b_{\max}] = [0, 5]$, the addition between two numbers from those ranges will fall into $[c_{\min}, c_{\max}] = [-1, 6]$. We get $S_a \approx 0.0078$, $Z_a = 128$, $S_b \approx 0.0196$, $Z_b = 0$, $S_c \approx 0.0274$, $Z_c = 36$. For $a = -0.9$, $b = 3.9$, we have $q_a = 13$ and $q_b = 199$. The quantized addition result q_c, following (4), is,

$$q_c = \left\lfloor \frac{S_a}{S_c}(q_a - Z_a) + \frac{S_b}{S_c}(q_b - Z_b) \right\rceil + Z_c \quad (21)$$

$$= \left\lfloor \frac{0.0078}{0.0274}(13 - 128) + \frac{0.0196}{0.0274} 199 \right\rceil + 36 \quad (22)$$

$$= 146 \quad (23)$$

and $r_y = 3.0140$ while $a + b = 3.0$.

Inference based on integer-only arithmetic For the inference to be integer-only arithmetic, all values have to be expressed as integers. However, in the expressions described throughout the previous sections, there often are some terms involving scaling factors that are not integers (e.g. $\frac{S_u S_w}{S_z}$ in (13)). Therefore the question one might ask is how to go around this fact? Such constants are expressed as fixed-point integers, which are computed off-line (after training but before inference). The Q-format is a way to describe the properties of a fixed-point integer number. We will base the following explanations on several assumptions to simplify the concepts, as fixed-point integer arithmetic is a vast concept. A fixed-point integer having format Qi.f is said to have one sign bit, i integral bits and f fractional bits, bitwidth $1 + i + f$, a resolution of 2^{-f} and span the range $[-2^{-f}, 2^i - 2^{-f}]$. For instance, Q3.4 represents numbers in $[-8, 7.9375]$ with resolution 0.0625 and Q0.30 represents numbers in $[-1, 0.99999]$ with $\approx 9 \times 10^{-10}$. Assuming we do not have to worry about

overflows, quantizing a floating-point number M to a fixed-point integer M_{fx} is given by,

$$M_{fx} = \lfloor 2^f M \rceil. \tag{24}$$

Going from a fixed-point integer M_{fx} to floating-point number M is given by,

$$M = 2^{-f} M_{fx} \tag{25}$$

For instance, with $M = \frac{S_u S_w}{S_z} = 0.0039$ and the fixed-point representation of M in Q-format Q0.30 is $M_{fx} = 4187593$ and mapped back to floating-point as $M = 0.00389$. An equivalent expression to (25) is

$$M = M_{fx} \gg f \tag{26}$$

where $\gg$ is the right shift operator. The last piece needed to perform integer-only inference is to incorporate the fixed-point numbers in the expressions involving floating-point constants, often of the form $q_y = \lfloor Mq \rceil + Z_y$, which can be expressed as

$$q_y = \left\lfloor 2^{-f} M_{fx} q \right\rceil + Z_y \tag{27}$$

$$= M_{fx} q \gg f + \tag{28}$$

$$(M_{fx} q \gg (f-1)) \;\&\; 0\text{x}1 + Z_y. \tag{29}$$

For instance, computing (13) using the fixed-point representation of $M = \frac{S_u S_w}{S_z} = 0.0039$ in Q0.30 is given by,

$$q_z = \left\lfloor 2^{-f} M_{fx} q \right\rceil + Z_z \tag{30}$$

$$= \left\lfloor 2^{-30}(4187593 \times -12051) \right\rceil + 128 \tag{31}$$

$$= -\Big((50464683243 \gg 30) \tag{32}$$

$$+ ((50464683243 \gg 29) \;\&\; 0\text{x}1) \Big) + 128 \tag{33}$$

$$= -(46 + 1) + 128 = 81 \tag{34}$$

which is the same result obtaing previously (16). We extracted the negative sign out of the binary shifts, as shifting on negative number has different behavior depending on the software and hardware implementation. Note that in practice, when we know a constant can only be positive, it will be expressed as an unsigned fixed-point integer and therefore the sign bit can be allocated to either the integral or fractional part.

Experimental details

We provide a detailed explanation of our experimental setups. The models' number of parameters and training time are reported in Table 1.

LayerNorm LSTM on PTB We provide detailed information about how the language modeling on PTB experiments are performed. The vocabulary size is 10k, and we follow dataset preprocessing as done in Mikolov et al. (2012). We report the best *perplexity* per word on the validation set and test set for a language model of embedding size 200 with one LayerNormLSTM cell of state size 200. The lower the perplexity, the better the model performs. In these experiments, we are focusing on the relative increase of perplexity between the full-precision models and their 8-bit quantized counterpart. We did not aim to reproduce state-of-the-art performance on PTB and went with a naive set of hyper-parameters. The full-precision network is trained on for 100 epochs with batch size 20 and BPTT (Werbos 1990) window size of 35. We used the SGD optimizer with weight decay of 10^{-5} and learning rate 20, which is divided by 4 when the loss plateaus for more than 2 epochs without a relative decrease of 10^{-4} in perplexity. We use gradient clipping of 0.25. We initialize the quantized models from the best full-precision checkpoint and train from another 100 epochs. For the first 5 epochs we do not enable quantization to gather range statistics to compute the quantization parameters.

Mogrifier LSTM on WikiText2 We describe the experimental setup for Mogrifier LSTM on WikiText2. Note that we follow the setup of Melis, Kočiský, and Blunsom (2020) where they do not use dynamic evaluation (Krause et al. 2018) nor Monte Carlo dropout (Gal and Ghahramani 2016). The vocabulary size is 33279. We use a 2 layer Mogrifier LSTM with embedding dimension 272, state dimension 1366, and capped input gates. We use 6 modulation rounds per Mogrifier layer with low-rank dimension 48. We use 2 Mixture-of-Softmax layers (Yang et al. 2018). The input and output embedding are tied. We use a batch size of 64 and a BPTT window size of 70. We train the full-precision Mogrifier LSTM for 340 epochs, after which we enable Stochastic Weight Averaging (SWA) (Izmailov et al. 2018) for 70 epochs. For the optimizer we used Adam (Kingma and Ba 2014) with a learning rate of $\approx 3 \times 10^{-3}$, $\beta_1 = 0$, $\beta_2 = 0.999$ and weight decay $\approx 1.8 \times 10^{-4}$. We clip gradients' norm to 10. We use the same hyper-parameters for the quantized models from which we initialize with a pre-trained full-precision and continue to train for 200 epochs. During the first 2 epochs, we do not perform QAT, but we gather min and max statistics in the network to have a correct starting estimate of the quantization parameters. After that, we enable 8-bit QAT on every component of the Mogrifier LSTM: weights, matrix multiplications, element-wise operations, activations. Then we replace activation functions in the model with quantization-aware PWLs and continue training for 100 epochs.

We perform thorough ablation on our method to study the effect of each component. Quantizing the weights or the weights and matrix multiplications covers about 0.1 of the perplexity increase. There is a clear performance drop after adding quantization of element-wise operations with an increase in perplexity of about 0.3. This is both due to the presence of element-wise operations in the cell and hidden states computations affecting the flow of information across timesteps and to the residual connections across layers. On top of that, adding quantization of the activation does not impact the performance of the network.

ESPRESSO LSTM on LibriSpeech The encoder is composed of 4 CNN-BatchNorm-ReLU blocks followed by 4 BiLSTM layers with 1024 units. The decoder consists of 3 LSTM layers of units 1024 with Bahdanau attention on the encoder's hidden states and residual connections between

Experiment	Params	Full-precision	iRNN	+PWL	Total GPU hours
LM PTB	4M	1h	13h	10h	24h
LM WikiText2	44M	37h	87h	50h	174h
ASR LibriSpeech	174M	144h	36h	12h	192h

Table 1: Details about the models parameters and training time for each different experiments setup: language modeling on PTB (LM PTB), language modeling on WikiText2 (LM WikiText2), automatic speech recognition on LibriSpeech (ASR LibriSpeech). Parameters are given in the order of millions. We report the total number of V100 GPU hours as well as the GPU hours used for each different training phase, i) pre-trained model (Full-precision), ii) quantized model without PWL (iRNN), iii) quantized model with PWL (+PWL).

each layer. The dataset preprocessing is exactly the same as in Wang et al. (2019). We train the model for 30 epochs on one V100 GPU, which takes approximately 6 days to complete. We use a batch size of 24 while limiting the maximum number of tokens in a mini-batch to 26000. Adam is used with a starting learning rate of 0.001, which is divided by 2 when the validation set metric plateaus without a relative decrease of 10^{-4} in performance. Cross-entropy with uniform label smoothing $\alpha = 0.1$ (Szegedy et al. 2016) is used as a loss function. At evaluation time, the model predictions are weighted using a pre-trained full-precision 4-layer LSTM language model (shallow fusion). Note that we consider this language model an external component to the ESPRESSO LSTM; we do not quantize it due to the lack of resources. However, we already show in our language modeling experiments that quantized language models retain their performance. We refer the reader to Wang et al. (2019) and training script[1] for a complete description of the experimental setup. We initialize the quantized model from the pre-trained full-precision ESPRESSO LSTM. We train the quantized model for only 4 epochs due to the lack of resources. The quantized model is trained on 6 V100 GPUs where each epoch takes 2 days, so a total of 48 GPU days. The batch size is set to 8 mini-batch per GPU with maximum 8600 tokens. We made these changes because otherwise, the GPU would run out of VRAM due to the added fake quantization operations. For the first half of the first epoch, we gather statistics for quantization parameters then we enable QAT. The activation functions are swapped with quantization-aware PWL in the last epoch. The number of pieces for the quantization-aware PWLs is 96, except for the exponential function in the attention, which is 160 as we found out it was necessary to have more pieces because of its curvature. The number of pieces used is higher than in the language modeling experiments we did. However, the difference is that the inputs to the activation functions are 16-bit rather than 8-bit although the outputs are still quantized to 8-bit. It means we need more pieces to capture the inputs resolution better. Note that it would not be feasible to use a 16-bit Look-Up Table to compute the activation functions due to the size and cache misses, whereas using 96 pieces allows for a 170x reduction in memory consumption compared to LUT.

[1]https://github.com/freewym/espresso/blob/master/examples/asr_librispeech/run.sh

References

Gal, Y.; and Ghahramani, Z. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In Lee, D.; Sugiyama, M.; Luxburg, U.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Hendrycks, D.; and Gimpel, K. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Hinton, G. 2012. Neural networks for machine learning. *Coursera*, video lectures.

Izmailov, P.; Podoprikhin, D.; Garipov, T.; Vetrov, D.; and Wilson, A. G. 2018. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A. G.; Adam, H.; and Kalenichenko, D. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2704–2713.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980.

Krause, B.; Kahembwe, E.; Murray, I.; and Renals, S. 2018. Dynamic Evaluation of Neural Sequence Models. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 2766–2775. PMLR.

Melis, G.; Kočiský, T.; and Blunsom, P. 2020. Mogrifier LSTM. In *International Conference on Learning Representations*.

Mikolov, T.; Sutskever, I.; Deoras, A.; Le, H.-S.; Kombrink, S.; and Cernocky, J. 2012. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 8: 67.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

Wang, Y.; Chen, T.; Xu, H.; Ding, S.; Lv, H.; Shao, Y.; Peng, N.; Xie, L.; Watanabe, S.; and Khudanpur, S. 2019. Espresso: A Fast End-to-End Neural Speech Recognition Toolkit. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 136–143.

Werbos, P. J. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10): 1550–1560.

iRNN Mogrifier LSTM	val	test
w/o PWL	60.40 ± 0.05	57.90 ± 0.01
w/o Quantized Activations	60.40 ± 0.03	57.95 ± 0.003
w/o Quantized Element-wise ops	60.08 ± 0.10	57.61 ± 0.23
w/o Quantized Matmul	60.10 ± 0.05	57.64 ± 0.10
w/o Quantized Weights (Full-precision)	60.27 ± 0.34	58.02 ± 0.34

Table 2: Ablation study on quantized Mogrifier LSTM training on WikiText2. iRNN w/o PWL is the quantized model using LUT instead of PWL to compute the activation function. Best results are averaged across 3 runs, and standard deviations are reported.

Full-precision model	val	test
LayerNorm LSTM	98.58 ± 0.35	94.84 ± 0.21
MadNorm LSTM	$\mathbf{97.20} \pm 0.47$	$\mathbf{93.63} \pm 0.74$

Table 3: Word-level perplexities on PTB for a full-precision LSTM with LayerNorm and a full-precision model with MadNorm. Best results are averaged across 3 runs, and standard deviations are reported.

Algorithm 1: The algorithm recursively reduced the number of pieces until the wanted number of pieces is achieved. The algorithm needs to be provided the function to approximate f, the input scaling factor S_x and zero-point Z_x, the quantization bitwidth b and the number of linear pieces wanted. One iteration of select_knots can be viewed in Figure 2.

def `select_knots` (*knots, intercepts, pwl_nb*) **:**
  dknots $\leftarrow$ knots$[1:] -$ knots$[:-1]$
  dintercepts $\leftarrow$ intercepts$[1:] -$ intercepts$[:-1]$
  slopes $\leftarrow$ dintercepts/dknots
  if `len` *(slopes)* $==$ *pwl_nb***:**
    return *knots, slopes, intercepts*
  else:
    diff_adj_slopes $\leftarrow \left|\text{slopes}[:-1] - \text{slopes}[1:]\right|$
    knot_index_to_remove $\leftarrow$ $\mathbf{argmin}$ diff_adj_slopes
    remaining_knots $\leftarrow$ knots.remove(knot_index_to_remove)
    remaining_intercepts $\leftarrow$ intercepts.remove(knot_index_to_remove)
    return `select_knots` (*remaining_knots, remaining_intercepts, pwl _nb*)

def `create_quantization_aware_pwl` (*f, input_scale, input_zero_point, b, pwl_nb*) **:**
  quantized_knots $\leftarrow [0, ..., 2^b - 1]$ `// Generate every` q_x
  knots $\leftarrow$ input_scale $\odot$ (quantized_knots $-$ input_zero_point) `// Generate every` r_x
  intercepts $\leftarrow f$(knots)
  {knots, slopes, intercepts} $\leftarrow$ `select_knots` (*knots, intercepts, pwl _nb*)
  return *knots, slopes, intercepts*

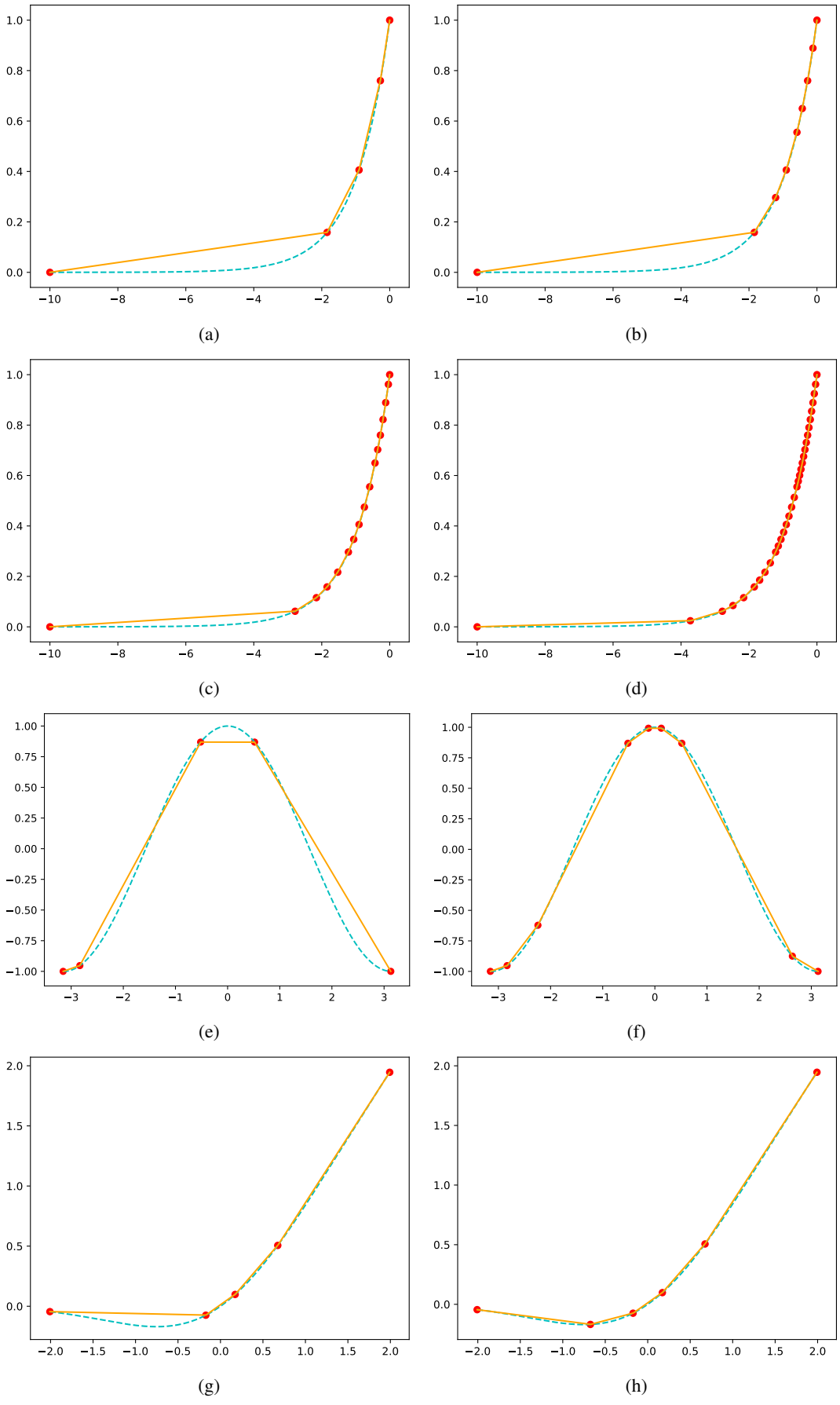
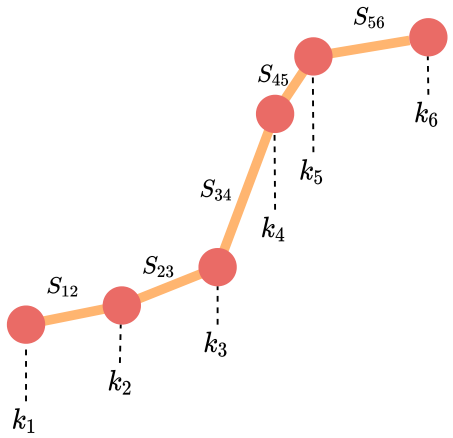


Figure 1: Quantization-aware PWL for several functions: exponential function approximation over $x \in [-10, 0]$ with a) 4, b) 8, c) 16, d) 32 pieces; cosine over $x \in [-\pi, \pi]$ with e) 4, f) 8 pieces; GeLU (Hendrycks and Gimpel 2016) over $x \in [-2, 2]$ with g) 4, h) 5 pieces.



i) Compute all slopes between closest knots k_a and k_b :

$$S_{ab} = \frac{f(k_b) - f(k_a)}{k_b - k_a}$$

$$S_{12} = 0.2, \quad S_{23} = 0.4, \quad S_{34} = 2.7, \quad S_{45} = 1.5, \quad S_{56} = 0.1$$

ii) Compute absolute differences between adjacents slopes :

$$D_{abc} = |S_{bc} - S_{ab}|$$

$$D_{123} = 0.2, \quad D_{234} = 2.3, \quad D_{345} = 1.2, \quad D_{456} = 1.4$$

iii) Remove shared knot k_b from smallest slope difference D_{abc} :

$$\text{argmin } D_{abc} = D_{123} \Rightarrow \text{remove } k_2$$

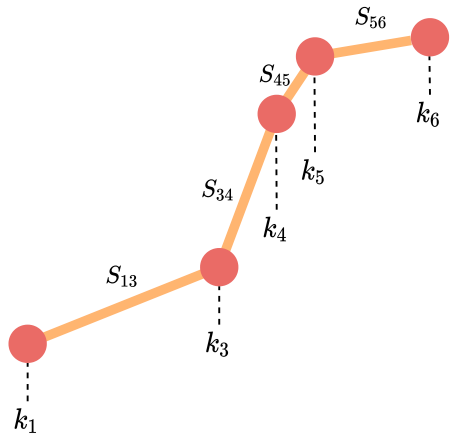


Figure 2: Example of an iteration from our proposed quantization-aware PWL Algorithm 1. The algorithm proceeds to reduce the number of pieces by merging two similar adjacents pieces. In this figure, the slopes S_{12} and S_{23} are the most similar pieces; therefore, the knot k_2 is removed.

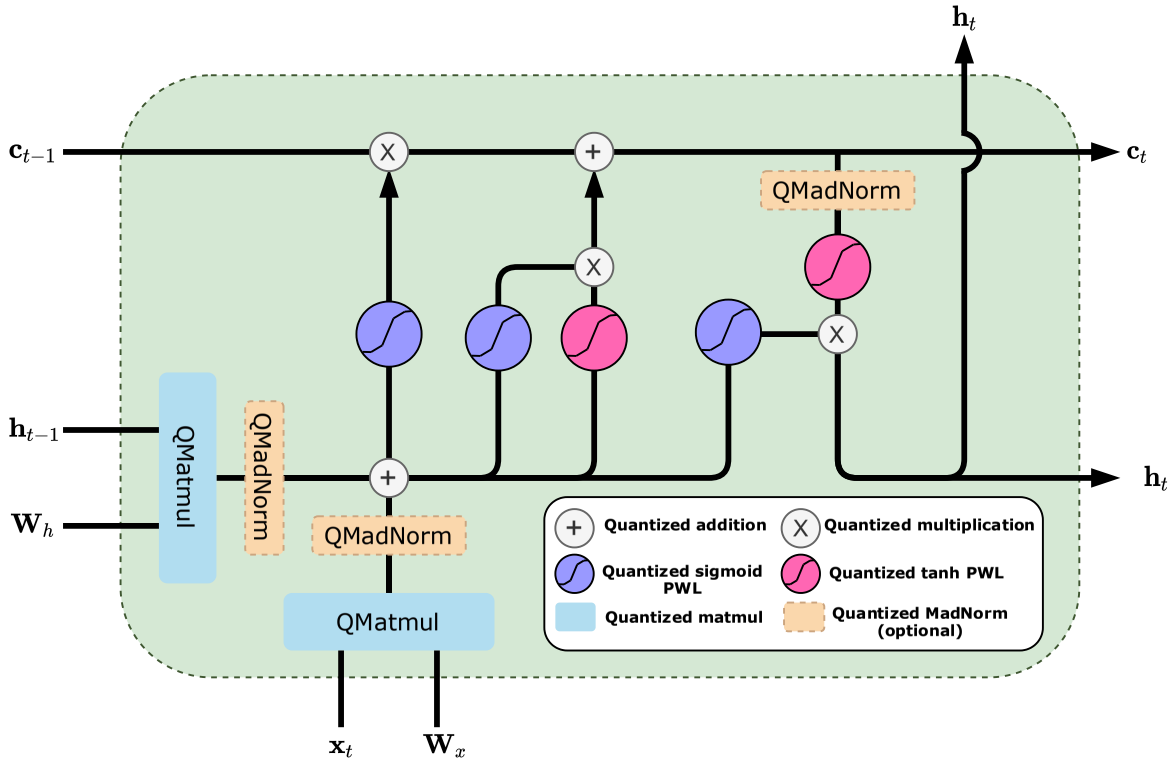


Figure 3: Example of an integer-only LSTM cell (iLSTM). Layer normalization change to quantized integer friendly MadNorm (QMadNorm), full- precision matrix multiplications change to integer matrix multiplication (QMatmul), sigmoid and tanh activations are replaced with their corresponding piecewise linear (PWL) approximations.

Yang, Z.; Dai, Z.; Salakhutdinov, R.; and Cohen, W. W. 2018. Breaking the Softmax Bottleneck: A High-Rank RNN Language Model. arXiv:1711.03953.