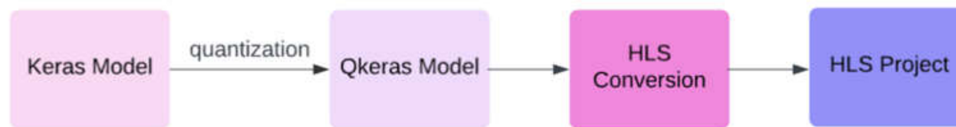
Quantization Aware Training for RNNs in HLS4ML

Yihui Chen, Elham E Khoda, Scott Hauck
University of Washington
Department of Electrical and Computer Engineering
{yihuic, ekhoda, hauck}@uw.edu

1. Introduction

In the previous work *ULTRA-LOW LATENCY RECURRENT NEURAL NETWORK INFERENCE ON FPGAS FOR PHYSICS APPLICATIONS WITH HLS4ML*, we convert three RNN models in hls4ml and estimate its performance and resource usage on FPGA. During the conversion, we introduce a process called quantization. "The weights and biases in trained models are typically stored with 32-bit floating-point precision. However, 32-bit 188 floating-point calculations are often not required for optimal network inference and are costly to implement on FPGAs. Other quantization techniques can offer more efficient ways of compressing neural networks by reducing the number of bits used to represent the weights and biases, ideally with no or minimal loss in performance."

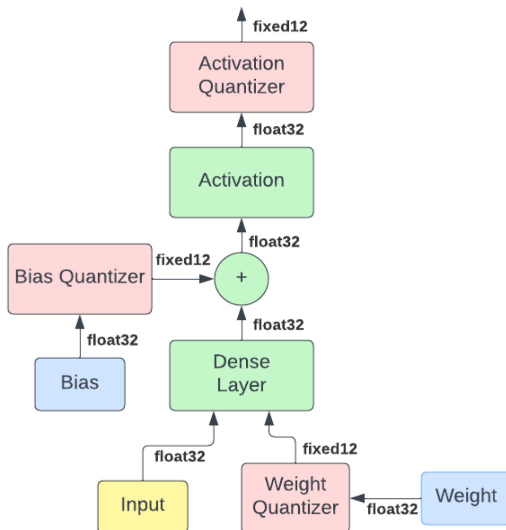


The quantization we did in previous work is called posted training quantization, which is the process of quantizing the whole keras model after it is trained. Post training quantization process is embedded in hls4ml process but it is less accurate in same number of bits compared to quantization aware training. In this paper, we focus on doing quantization aware training for the keras RNN model and fit the quantized model into our hls4ml workflow. Therefore we implement a workflow from floating point machine learning model to quantized machine learning model to hls4ml.

2. Qkeras (can highlight autoqkeras)

Qkeras is a quantization extension to Keras that provides a drop-in replacement for some of the Keras layers. In this project, we use Qkeras to replace all of the layers in our RNN model with quantized layer and train the Qkeras model(which is how we do quantization aware training). The way Qkeras does quantization is to add a quantized layer after each original layer. Therefore for qkeras model, it

is not fully running at fixed point numbers. Most of the calculation for each layer is still running at floating point numbers but after the calculation, the layer will quantize the output of each layer into fixed point numbers.



When it comes to bits selection, we are now trying all the possible bits combinations and find the sweet point where using the least number of bits to achieve comparable accuracy (above 95% of the Keras model accuracy). We did this because we want to make the bit selection through our whole model consistent.
However, Qkeras also offers a feature called autoqkeras. By using autoqkeras, it will help us to find the most suitable bits for quantization but this will cause each layer of the model using different bits for quantization. Also using autoqkeras is more time-consuming and requires more calculation resources.

3. Implemented Details

Right now the hls4ml still not support qkeras model directly. However we can stilluse the quantized model in hls4ml by loading the weight of qkeras model back into the original keras model and convert the keras model into hls4ml.

4. Performance

4.1 Models

The 3 models we selected as demonstrations are the Top Quark Tagging model, Jet Flavor Tagging model, and Quick Draw model.

Top quark tagging models utilize deep learning algorithms to identify and classify top quark events from complex collision data. By analyzing the kinematic

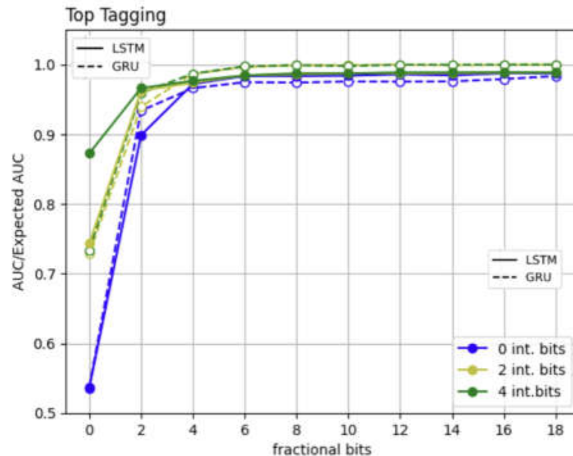
and geometric properties of particles produced in collisions, these models can accurately distinguish top quark events from background noise.

Jet flavor tagging models, which goal is similar to the quark tagging model, employ machine learning techniques to identify and categorize the flavors of jets produced in high-energy collisions.

The Quick Draw model is a remarkable application of machine learning that enables users to sketch objects, which are then recognized and classified by an artificial intelligence algorithm. By leveraging deep learning techniques, the Quick Draw model can learn to interpret a wide variety of hand-drawn sketches and identify the corresponding objects with impressive accuracy.

4.2 Quantization Performance

For top-tag model, after quantization, the accuracy of model can achieve nearly identical to keras model at 2 int bits and 6 fractional bits, which is in toal 9 bits. This is smaller than the 32 bits we use in keras model. Also the model with LSTM layer performances better than GRU layer with 0 or 2 integer bits but not with 4 integer bits. After convert it into HLS model, hls4ml perdicts its estimated utlization and performance.(model with GRU is on the left and model with LSTM is on the right)



Layer (type)	Output Shape	Param #
layer1 (QGRU)	(None, 20)	1680
layer2 (QDense)	(None, 64)	1344
relu_0 (QActivation)	(None, 64)	0
layer4 (QDense)	(None, 1)	65
output_sigmoid (Activation)	(None, 1)	0

Total params: 3,089
Trainable params: 3,089
Non-trainable params: 0

Layer (type)	Output Shape	Param #
layer1 (QLSTM)	(None, 20)	2160
layer3 (QDense)	(None, 64)	1344
relu_0 (QActivation)	(None, 64)	0
layer5 (QDense)	(None, 1)	65
output_sigmoid (Activation)	(None, 1)	0

Total params: 3,569
Trainable params: 3,569
Non-trainable params: 0

== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	-	-	-	-	-
Instance	33	887	12501	93664	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	683	-
Register	-	-	4448	-	-
Total	33	887	16949	94353	0
Available SLR	1344	3072	864000	432000	320
Utilization SLR (%)	2	28	1	21	0
Available	5376	12288	3456000	1728000	1280
Utilization (%)	~0	7	~0	5	0

== Performance Estimates

+ Timing:
  * Summary:

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.639 ns	0.62 ns

+ Latency:
  * Summary:

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
146	146	0.730 us	0.730 us	140	140	function

== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	-	-	-	-	-
Instance	43	1101	14693	105179	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	763	-
Register	-	-	4436	-	-
Total	43	1101	19129	105948	0
Available SLR	1344	3072	864000	432000	320
Utilization SLR (%)	3	35	2	24	0
Available	5376	12288	3456000	1728000	1280
Utilization (%)	~0	8	~0	6	0

== Performance Estimates

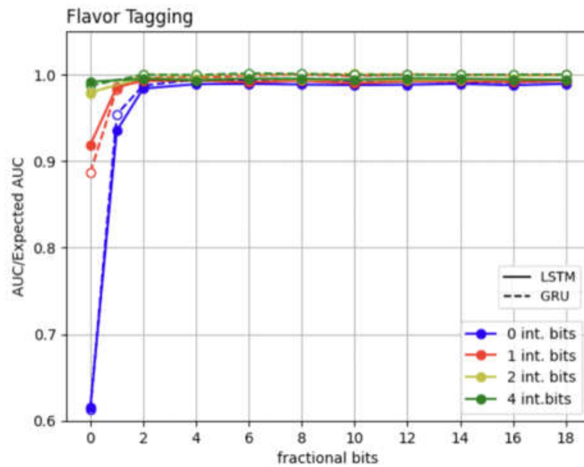
+ Timing:
  * Summary:

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.639 ns	0.62 ns

+ Latency:
  * Summary:

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
166	166	0.830 us	0.830 us	160	160	function

For b-tagging model, after quantization, the accuracy of model can achieve nearly identical to keras model at 2 int bits and 2 fractional bits, which is in toal 5 bits. This is smaller than the 32 bits we use in keras model. Also the model with LSTM layer performances mostly identical compares to GRU layer when fractional bits are larger than 2 bits.



.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 15, 6)]           0

gru (QGRU)                   (None, 120)               46080

dense_0 (QDense)             (None, 50)                6050

relu_0 (QActivation)         (None, 50)                0

dense_1 (QDense)             (None, 10)                510

relu_1 (QActivation)         (None, 10)                0

dense_2 (QDense)             (None, 3)                 33

output_softmax (Activation)  (None, 3)                 0

=================================================================
Total params: 52,673
Trainable params: 52,673
Non-trainable params: 0
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 15, 6)]           0

lstm1 (QLSTM)                (None, 120)               60960

dense_0 (QDense)             (None, 50)                6050

relu_0 (QActivation)         (None, 50)                0

dense_1 (QDense)             (None, 10)                510

relu_1 (QActivation)         (None, 10)                0

dense_2 (QDense)             (None, 3)                 33

output_softmax (Activation)  (None, 3)                 0

=================================================================
Total params: 67,553
Trainable params: 67,553
Non-trainable params: 0
_________________________________________________________________
```

5. Discussion

5.1 Super high accuracy in qkeras quantization aware training ?

For people who try to do the quantization aware training using qkeras, they might find it surprising that the model can achieve nearly identical accuracy with very tiny bits (such as only 2 integer bits, 4 fractional bits and 1 sign bits, total of 7 bits). The reason for such high accuracy in qkeras is that qkeras does not fully quantized our model. Instead, for most of the quantized layers, qkeras just combine the original layer with a quantizer after it to quantize the output of the layer. Therefore for most of the quantized layers (especially for activation layers), the calculation is still running in floating point numbers but qkeras just make the output of them to be fixed point numbers.

5.2 How to convert qkeras model into hls4ml ?

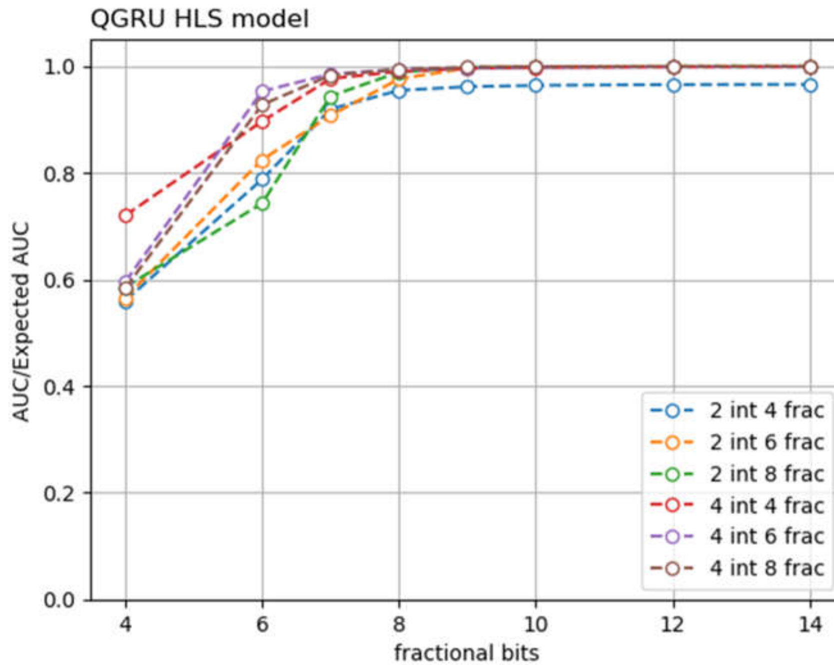
Right now, hls4ml is still not offically supported qkeras. When we trying to do so the error shows hls4ml couldn't recongize layers in qkeras. Instead, the trick I did is to load the weights of qkeras model back into the keras model and use the keras model with quantized weight and covert it into hls4ml.

```
## Load qkeras weight back into keras model
qgru = load_model('./toptag_model/qgru_2int/model_qgru_4frac.h5', custom_objects={'QGRU':QGRU,'QDense':QDense,'qu
model_save_quantized_weights(qgru, "gru22test.h5")
gru.load_weights('gru22test.h5')
```

5.3 Accuracy of qerkas model drops a lot in hls4ml ?

When we convert the model into hls4ml, the accuracy will drop a lot compares to what we got in qkeras. This happens for all qkeras model but the accuracy drops more when there are RNN layers in qkeras model. As we discussed in 5.1, the way qkeras quantize the model is not fully quantized but normally just quantizes the output of each layer. However, in hls4ml, we are

doing fully quantized since floating point calculation is not supported on FPGA. The plot below shows the changes in accuracy when we give different numbers of bits for qkeras model.



Taking quantized activation functions as an example, when we see the code in qkeras activation function, we will find that there is no difference inside

the calculation of the activation function.

How the Qkeras quantized layer works (example of quantized relu layer)

Part of quantized_relu code

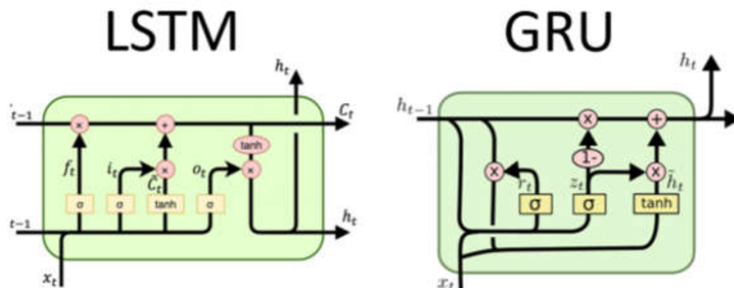
```
def __call__(self, x):
  if not self.built:
    self.build(var_name=self.var_name, use_variables=self.use_variables)

  non_sign_bits = self.bits - (self.negative_slope != 0.0)
  x = K.cast(x, dtype="float32")
  m = K.cast(K.pow(2, non_sign_bits), dtype="float32")
  m_i = K.cast(K.pow(2, self.integer), dtype="float32")

  # is_quantized_clip has precedence over relu_upper_bound for backward
  # compatibility.
  m_f = K.cast(
      K.pow(
          tf.constant(2., tf.float32),
          K.cast(self.integer, dtype="float32") - non_sign_bits),
      dtype="float32")
  if self.is_quantized_clip:
    x_u = tf.where(x <= m_i - m_f, K.relu(x, alpha=self.negative_slope),
                   tf.ones_like(x)
  elif self.relu_upper_bound is no
    x_u = tf.where(x <= self.relu_
                   K.relu(x, alpha
                   tf.ones_like(x)
  else:
    x_u = K.relu(x, alpha=self.neg
```

```
Definition    References
Defined on line 24
24   import tensorflow.keras.backend as K
Viewing precise results. Try a search-based lookup. Learn more.
```

Part of relu function in keras

```
def relu(x, alpha=0., max_value=None, threshold=0):
  """Rectified linear unit.

  With default values, it returns element-wise `max(x, 0)`.

  Otherwise, it follows:
  `f(x) = max_value` for `x >= max_value`,
  `f(x) = x` for `threshold <= x < max_value`,
  `f(x) = alpha * (x - threshold)` otherwise.

  Args:
      x: A tensor or variable.
      alpha: A scalar, slope of negative section (default=`0.`).
      max_value: float. Saturation threshold.
      threshold: float. Threshold value for thresholded activation.

  Returns:
      A tensor.
  """
  # While x can be a tensor or variable, we also see cases where
  # numpy arrays, lists, tuples are passed as well.
  # lists, tuples do not have 'dtype' attribute.
  dtype = getattr(x, 'dtype', floatx())
  if alpha != 0.:
    if max_value is None and threshold == 0:
      return nn.leaky_relu(x, alpha=alpha)

    if threshold != 0:
      negative_part = nn.relu(-x + threshold)
    else:
      negative_part = nn.relu(-x)
```

When it comes to RNN layers, the two RNN layers in our models are GRU and LSTM. For GRU layer, there are two different activation layers inside it: tanh and sigmoid. Sigmoid is for calculating the update gate and reset gate and Tanh is for calculating candidate hidden state in GRU. For LSTM layer, there are also using tanh and sigmoid functions. Sigmoid is for calculating the forgetting and input gate and Tanh is for calculating the candidate cell state. Both of them are for calculating output. Since activation functions in qkeras are all calculated in floating point numbers, the accuracy difference between qkeras and hls mode will be huge.



6. Conclusion

This project is mainly for adding quantization aware trianing process before hls4ml process. By doing so we will need to use less bits in calculation and therefore decerease the resouce useage while maintaining similiar performance. Above the three models we disscussed as benckmark for RNN models, toptag is the smallest one and should be the first one to train for people who want to try it. Quickdraw model is the largest one and doing quantization aware training to it takes a lot of time and need a really good GPU. Due to the limited computational resource, I didn't finish the quantization aware training for quickdraw model and the HLS conversion for b-tagging model and toptag model.

7. Code and Data

quantiaztion aware training for above three models: https://github.com/uw-acme/HLS4ML_RNN
Hls4ml conversion:
https://github.com/yihuiccc/hls4ml-RNN-test
Qkeras tutorial for starters:
https://github.com/uw-acme/acme-lab-documentation/blob/main/quantization/Qkeras-Tutorial-AndrewChen.ipynb
toptag dataset:
https://cernbox.cern.ch/s/0CBn5SsUPb5KDnX?redirectUrl=%2Ffiles%2Flink%2Fpublic%2F0CBn5SsUPb5KDnX
btag dataset (pwd:hls-btag):
https://cernbox.cern.ch/s/dYrWPhWQFbAgjh1?redirectUrl=%2Ffiles%2Flink%2Fpublic%2FdYrWPhWQFbAgjh1
quickdraw dataset:
https://console.cloud.google.com/storage/browser/quickdraw_dataset/sketchrnn;tab=objects?pli=1&prefix=&forceOnObjectsSortingFiltering=false