Speed-up of neural networks with Quantization

Marek Petrovič, 506463

Why quantization

- Lower bit widths for computations
- Integer vs. Float
- Network compression
- PyTorch, TensorFlow support

Current hardware

- Significant memory saving
  - 8bit vs 32bit
- Significant energy saving
  - 18-30x less energy
- ARM processors
  - INT8 hardware support
  - FP32 software support
- OpenVINO INT8 vs FP32
  - 3,9 times faster*
- NVIDIA Turing Tensor Cores
  - INT8 16x
  - INT4 32x

	NVIDIA A100	**NVIDIA Turing**
Tensor Core	FP64, TF32, bfloat16, FP16, INT8, INT4, INT1	FP16, INT8, INT4, INT1
Cuda Core	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, INT8

NVIDIA. NVIDIA Tensor Cores: Versatility for HPC & AI. *NVIDIA Tensor Cores*. Available from: https://www.nvidia.com/en-us/data-center/tensor-cores/

*** OPENVINO TOOLKIT**. INT8 vs FP32 Comparison on Select Networks and Platforms - OpenVINO™ Toolkit. *OpenVINO*. Available from: https://docs.openvinotoolkit.org/latest/_docs_performance_int8_vs_fp32.html

Quantization

$$x \in [\alpha, \beta]$$

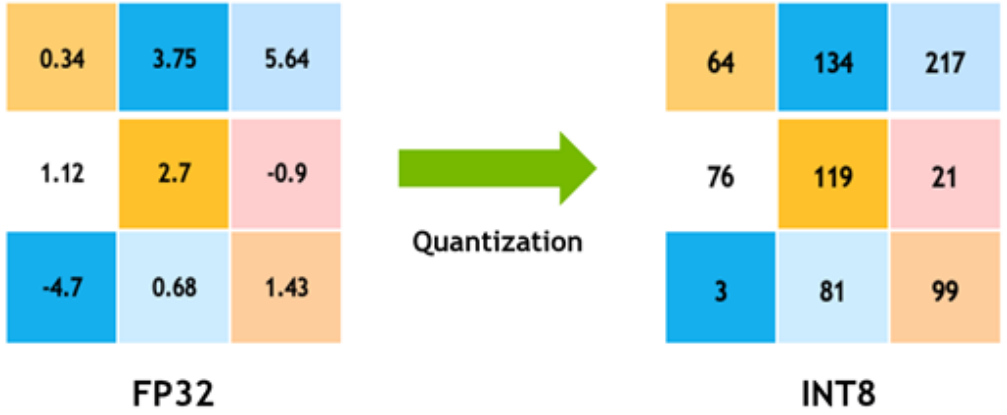
$$x_q \in [\alpha_q, \beta_q]$$

$$s = \frac{\beta - \alpha}{\beta_q - \alpha_q}$$

$$z = round(\frac{\beta\alpha_q - \alpha\beta_q}{\beta - \alpha})$$

$$x_q(x, s, z) = round\left(\frac{x}{s} + z\right)$$

$$x = s(x_q - z)$$



x – float value
x_q – quantized result
s – scale, positive floating point
z – zero point, integer value
INT8 range is 256

Dynamic quantization

- weights ahead, activations on inference
  - Modules instead of functional
    - nn.ReLU instead of nn.functional.relu
  - What to quantize

```
# original model
# all tensors and computations are in floating point
previous_layer_fp32 -- linear_fp32 -- activation_fp32 -- next_layer_fp32
                    /
linear_weight_fp32

# dynamically quantized model
# linear and LSTM weights are in int8
previous_layer_fp32 -- linear_int8_w_fp32_inp -- activation_fp32 -- next_layer_fp32
                     /
   linear_weight_int8
```

Dynamic quantization

- Quantization, De-quantization
- Steps:
  1. Weights quantized prior to inference
  2. Activations in FP32
  3. Compute scale, zero point for activations tensor (during runtime)
  4. Quantize activation tensor

```
# original model
# all tensors and computations are in floating point
previous_layer_fp32 -- linear_fp32 -- activation_fp32 -- next_layer_fp32
                    /
linear_weight_fp32

# dynamically quantized model
# linear and LSTM weights are in int8
previous_layer_fp32 -- linear_int8_w_fp32_inp -- activation_fp32 -- next_layer_fp32
                      /
   linear_weight_int8
```

Static Quantization

- Static – weights and activations ahead, calibration set
  - Same preparations as dynamic quantization
  - QuantStub/ DeQuantStub
  - nn.quantized.FloatFunctional
  - Fuse Modules

```
# original model
# all tensors and computations are in floating point
previous_layer_fp32 -- linear_fp32 -- activation_fp32 -- next_layer_fp32
                    /
    linear_weight_fp32

# statically quantized model
# weights and activations are in int8
previous_layer_int8 -- linear_with_activation_int8 -- next_layer_int8
                    /
  linear_weight_int8
```

Static Quantization

- Steps:
  1. Representative calibration set
  2. Collect distribution statistics of activation layers
  3. Compute scales, zero points
  4. Quantize network
  5. Inference

```
# original model
# all tensors and computations are in floating point
previous_layer_fp32 -- linear_fp32 -- activation_fp32 -- next_layer_fp32
                     /
    linear_weight_fp32

# statically quantized model
# weights and activations are in int8
previous_layer_int8 -- linear_with_activation_int8 -- next_layer_int8
                     /
  linear_weight_int8
```

Quantization-aware training

- Training/Fine-tuning
- Effects of quantization in training

```
# original model
# all tensors and computations are in floating point
previous_layer_fp32 -- linear_fp32 -- activation_fp32 -- next_layer_fp32
                       /
    linear_weight_fp32

# model with fake_quants for modeling quantization numerics during training
previous_layer_fp32 -- fq -- linear_fp32 -- activation_fp32 -- fq -- next_layer_fp32
                            /
   linear_weight_fp32 -- fq

# quantized model
# weights and activations are in int8
previous_layer_int8 -- linear_with_activation_int8 -- next_layer_int8
                      /
   linear_weight_int8
```

https://pytorch.org/docs/stable/quantization.html#model-preparation-for-quantization

Quantization-aware training

- Fake Quantization layers – non-differentiable
- STE- Straight Through Estimators
- Steps:
  1. Train as usual
  2. After some epochs freeze batch norm mean and variance estimates
  3. After some epochs freeze quantizer parameters (scale and zero points)
     1. Fine tuning the weight only

$$x = s\left(clip\left(round\left(\frac{x}{s_x} + z_x\right), \alpha_q, \beta_q\right) - z_x\right)$$

$$\frac{\partial \hat{x}}{\partial x} = \begin{cases} 1 & \alpha \le x \le \beta \\ 0 & \end{cases}$$

https://pytorch.org/docs/stable/quantization.html#model-preparation-for-quantization

PyTorch

- INT8 support
- Dynamic, static, quant.-aware

	Static Quantization	Dynamic Quantization
nn.Linear	Y	Y
nn.Conv1d/2d/3d	Y	N
nn.LSTM	N	Y
nn.GRU	N	Y
nn.RNNCell	N	Y
nn.GRUCell	N	Y
nn.LSTMCell	N	Y
nn.EmbeddingBag	Y (activations are in fp32)	Y
nn.Embedding	Y	N
nn.MultiheadAttention	Not Supported	Not supported
Activations	Broadly supported	Un-changed, computations stay in fp32

State of the art

- Static quantization – Distiller Fr.
  - Good results with CNNs
  - INT8 (Exp1.) almost same as FP32
  - INT4 (Exp.5) unusable
  - LAPQ – mixed INT8, INT4

model	Top1-acc	Top5-acc	Diff Top1	Diff Top5
ResNet18	78.29%	93.94%	0.00%	0.00%
Exp. 1	78.21%	93.89%	-0.08%	-0.05%
Exp. 5	27.16%	50.83%	-51.13%	-43.11%
LAPQ 2	61.74%	84.77%	-16.55%	-9.17%
ResNet50	87.04%	97.56%	0.00%	0.00%
Exp. 1	86.83%	97.50%	-0.21%	-0.06%
Exp. 5	19.66%	37%	-67.38%	-60.20%

model	mAP	Diff
SSD ResNet50	12.71%	0.00%
8bit	12.27%	0.44%
7bit	12.05%	0.66%

State of the art

- Q8BERT: Quantized 8Bit BERT
- Quantization-aware training during fine-tuning phase
- Symetric linear quantization, weights and activations INT8

Dataset	Metric	BERT baseline accuracy(STD)	QAT BERT 8bit (STD)	DQ BERT 8bit(STD)
CoLA	Matthew's corr	**58.48 (-1.54)**	**58.48 (-1.32)**	56.74 (-0.61)
MRPC	F1	**90 (-0.23)**	89.56 (-0.18)	87.88 (-2.03)
MRPC-Large	F1	90.86 (-0.55)	**90.9 (-0.29)**	88.18 (-2.19)
QNLI	Accuracy	90.3 (-0.44)	**90.62 (-0.29)**	89.34 (-0.61)
QNLI-Large	Accuracy	91.66 (-0.15)	**91.74 (-0.36)**	88.38 (-2.22)
QQP	F1	87.84 (-0.19)	**87.96 (-0.35)**	84.98 (-0.97)
RTE	Accuracy	**69.7 (-1.5)**	68.78 (-3.52)	63.32 (-4.58)
SST-2	Accuracy	**92.36 (-0.59)**	92.24 (-0.27)	91.04 (-0.43)
STS-B	Pearson corr	**89.62 (-0.31)**	89.04 (-0.17)	87.66 (-0.41)
STS-B-Large	Pearson corr	**90.34 (-0.21)**	90.12 (-0.13)	83.04 (-5.71)
SQuADv1.1	F1	88.46 (-0.15)	**87.74 (-0.15)**	80.02 (-2.38)

* https://arxiv.org/abs/1910.06188

State of the art

- I-BERT: Integer-only BERT Quantization
- Static Quantization, Quantization-Aware FineTuning
- All operations INT8/32 (no cast to floating point)

			RoBERTa- Base									
	Precision	**Int-only**	**MNLI-m**	**MNLI-m**	**QQP**	**QNLI**	**SST-2**	**CoLA**	**STS-B**	**MRPC**	**RTE**	**Avg.**
Baseline	**FP32**	✗	87.8	87.4	90.4	92.8	94.6	61.2	91.1	90.9	78	86
I-BERT	**INT8**	✓	87.5	87.4	90.2	92.8	95.2	62.5	90.8	91.1	79.4	86.3
Diff			-0.3	0	-0.2	0	0.6	1.3	-0.3	0.2	1.4	0.3

* https://arxiv.org/abs/2101.01321

State of the art

- I-BERT: Integer-only BERT Quantization
- Static Quantization, Quantization-Aware FineTuning
- All operations INT8/32 (no cast to floating point)

			RoBERTa- Large									
	Precision	**Int-only**	**MNLI-m**	**MNLI-m**	**QQP**	**QNLI**	**SST-2**	**CoLA**	**STS-B**	**MRPC**	**RTE**	**Avg.**
Baseline	**FP32**	✗	90	89.9	92.8	94.1	96.3	68	92.2	91.8	86.3	89
I-BERT	**INT8**	✓	90.4	90.3	93	94.5	96.4	69	92.2	93	87	89.5
Diff			0.4	0.4	0.2	0.4	0.1	1	0	1.2	0.7	0.5

* https://arxiv.org/abs/2101.01321

State of the art

- I-BERT: Integer-only BERT Quantization
- Static Quantization, Quantization-Aware FineTuning
- All operations INT8/32 (no cast to floating point)

Sent. L.	128				256				
Batch s.	1	2	4	8	1	2	4	8	Avg.
Base	2.42	3.36	3.39	3.31	3.11	2.96	2.94	3.15	3.08
Large	3.2	4	3.98	3.81	3.19	3.51	3.37	3.4	3.56

speed-up INT8 vs FP32

* https://arxiv.org/abs/2101.01321

State of the art

- Quantized Transformer
- Quantization-Aware FineTuning
- BERT model

Model	BLEU
32-bit	27.83
8-bit	26.94
4-bit	23.18
1-bit	2.5
1-bit(weight only)	23.88

WMT-14 EN-DE

Model	Accuracy
32-bit	84.6
8-bit	86.3
4-bit	68.4

MRPC dev

Model	EM	F1
32-bit	81.18	88.52
8-bit	81.05	88.37

SQuAD dev set

* https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15742249.pdf

State of the art

- OpenVINO Toolkit
- Conversion from FP32 to INT8

OpenVINO benchmark model name	Dataset	Intel® Core™ i7-8700T	Intel® Core™ i7-1185G7	Intel® Xeon® W-1290P	Intel® Xeon® Platinum 8270
		Throughput speed-up FP16-INT8 vs FP32			
bert-large-uncased-whole-word-masking-squad-0001	SQuAD	1.6	3.1	1.5	2.5
brain-tumor-segmentation-0001-MXNET	BraTS	1.6	2	1.8	1.8
deeplabv3-TF	VOC 2012 Segmentation	1.9	3	2.8	3.1
densenet-121-TF	ImageNet	1.8	3.5	1.9	3.8
facenet-20180408-102900-TF	LFW	2.1	3.6	2.2	3.7
faster_rcnn_ resnet50_coco-TF	MS COCO	1.9	3.7	2	3.4
inception-v3-TF	ImageNet	1.9	3.8	2	4.1
mobilenet-ssd-CF	VOC2012	1.6	3.1	1.9	3.6
mobilenet-v2-1.0-224-TF	ImageNet	1.5	2.4	1.8	3.9
mobilenet-v2-pytorch	ImageNet	1.7	2.4	1.9	4
resnet-18-pytorch	ImageNet	1.9	3.7	2.1	4.2

* https://arxiv.org/abs/1910.06188

State of the art

- OpenVINO Toolkit
- Conversion from FP32

OpenVINO Benchmark Model Name	Dataset	Metric Name	Intel® Core™ i9-10920X CPU @ 3.50GHZ (VNNI)	Intel® Core™ i9-9820X CPU @ 3.30GHz (AVX512)	Intel® Core™ i7-6700K CPU @ 4.0GHz (AVX2)	Intel® Core™ i7-1185G7 CPU @ 4.0GHz (TGL VNNI)
			Absolute Accuracy Drop, %			
bert-large-uncased-whole-word-masking-squad-0001	SQuAD	F1	0.62	0.71	0.62	0.62
brain-tumor-segmentation-0001-MXNET	BraTS	Dice-index@ Mean@ Overall Tumor	0.08	0.1	0.1	0.08
deeplabv3-TF	VOC 2012 Segmentation	mean_iou	0.09	0.41	0.41	0.09
densenet-121-TF	ImageNet	acc@top-1	0.49	0.56	0.56	0.49
facenet-20180408-102900-TF	LFW	pairwise_ accuracy _subsets	0.05	0.12	0.12	0.05
faster_rcnn_ resnet50_coco-TF	MS COCO	coco_ precision	0.09	0.09	0.09	0.09
inception-v3-TF	ImageNet	acc@top-1	0.02	0.01	0.01	0.02
mobilenet-ssd-CF	VOC2012	mAP	0.06	0.04	0.04	0.06
mobilenet-v2-1.0-224-TF	ImageNet	acc@top-1	0.4	0.76	0.76	0.4
mobilenet-v2-PYTORCH	ImageNet	acc@top-1	0.36	0.52	0.52	0.36
resnet-18-pytorch	ImageNet	acc@top-1	0.25	0.25	0.25	0.25
resnet-50-PYTORCH	ImageNet	acc@top-1	0.19	0.21	0.21	0.19

* https://arxiv.org/abs/1910.06188

Evaluation framework

- Evaluation
  - NMT quality by at least one metric
  - Inference speed
  - Generating summary
- Implement
  - At least one method of speed-up
- Comparison
  - Quality/speed across domains

Main classes

- ModelWrapper
  - wraps Transformers model and tokenizer
  - Allows for Static and Dynamic Quantization
- Dataset
  - Implements various datasets from Datasets (HuggingFace)
- Pipeline
  - For defining and running experiments
  - Scenarios
    - Evaluate, Train and evaluate, Quant. Aware train and eval., Quant. Aware Fine-tune and Eval
- Comparator
  - Compare results of Pipelines

Dynamic Quantization

```python
def _dynamic_quant(model_wrapped: ModelWrapper):
    print(
        f"Using '{get_quant_backend()}' engine for quantization")

    quantized_model = torch.quantization.quantize_dynamic(
        model_wrapped.model, {torch.nn.Linear}, dtype=torch.qint8
    )
    return quantized_model
```

	Static Quantization	Dynamic Quantization
nn.Linear	Y	Y
nn.Conv1d/2d/3d	Y	N
nn.LSTM	N	Y
nn.GRU	N	Y
nn.RNNCell	N	Y
nn.GRUCell	N	Y
nn.LSTMCell	N	Y
nn.EmbeddingBag	Y (activations are in fp32)	Y
nn.Embedding	Y	N
nn.MultiheadAttention	Not Supported	Not supported
Activations	Broadly supported	Un-changed, computations stay in fp32

Example

```python
from enmt.datasets.open_subtitles import OpenSubtitles
from enmt.datasets.opus100 import Opus100
from enmt.datasets.ubuntu import Ubuntu
from enmt.model import ModelWrapper
from enmt.quant_helper import QuantizationMode
from enmt.results import Dataset, Pipeline, Scenario

model = ModelWrapper(pretrained_model_name_or_path="t5-small")

eval = Opus100(lang1='de', lang2='en')
training_args = {'evaluation_strategy': 'epoch', 'learning_rate': 0.00002, 'per_device_train_batch_size': 4, 'per_device_eval_batch_size': 5,
                 'weight_decay': 0.01, 'save_total_limit': 3, 'num_train_epochs': 1, 'predict_with_generate': True, 'no_cuda': True, 'fp16': False, 'push_to_hub': False}



pipe = Pipeline(Scenario.EVAL, model=model, dataset_eval=eval,
                training_args=training_args)
pipe.run()


model.quantize(QuantizationMode.DYNAMIC)
pipe = Pipeline(Scenario.EVAL, model=model, dataset_eval=eval,
                training_args=training_args)
pipe.run()
```

Results so far

- CometML
- INT8 (nn.Linear) almost same BLEU

	eval_bleu	diff	eval_gen_len	eval_runtime	diff	eval_samples_per_second	eval_steps_per_second
t5small commoncrawl							
opus100de-en CPU	11.555		14.673	312.442		6.401	1.28
opus100de-en DynQuant CPU	11.331	1.94%	14.902	174.971	1.786	11.43	2.286
Helsinki-NLP MarianMT opus							
CPU opus100en-sk	37.071		15.366	1128.816		1.772	0.354
Cuda opus100en-sk	37.071		15.366	158.095		12.651	2.53
DynamicQuant2 opus100en-sk	35.853	3.29%	15.504	843.953	1.338	2.37	0.474
DynamicQuant opus100en-sk	35.853	3.29%	15.504	772.028	1.462	2.591	0.518

	size [MB]	diff
t5small commoncrawl		
FP32	242.08377	
INT8	126.49234	47.75%
Helsinki-NLP MarianMT opus		
FP32	301.91386	
INT8	200.59916	33.56%

TODO

- Static quantization
- Quant-aware train, fine-tune
- Comparator class
- Test if is general enough

Questions, suggestions?