

# Reduced/Mixed precision in Machine Learning

Shao Jie Hu Chen, Jaesung Kim

Technical University of Munich

School of Computation, Information and Technology

Munich, 24.01.2024



*TUM Uhrenturm*

# Outline

1. Introduction
2. Quantization scales
3. Low/mixed precision in Gaussian Processes
4. Low/mixed precision in Kernel approximation
5. Low/mixed precision in Neural Networks
6. Conclusion

# Outline

## 1. Introduction

1.1 Concept of low/mixed precision

1.2 Quantization in other fields

1.3 Challenges in low and mixed precision in Machine Learning

## 2. Quantization scales

## 3. Low/mixed precision in Gaussian Processes

## 4. Low/mixed precision in Kernel approximation

## 5. Low/mixed precision in Neural Networks

## 6. Conclusion

## Concept of low/mixed precision



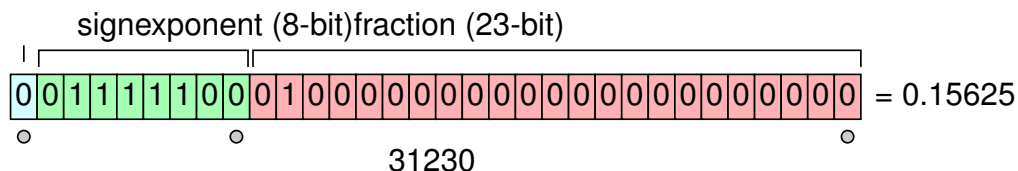
**Problem.** Which are the numbers that may be represented using  $b$  bits?

## Concept of low/mixed precision

**Problem.** Which are the numbers that may be represented using  $b$  bits?

**Solution.** Different number representation systems may be used. For example:

- **Natural number representation.** We assign each number representable by the bits such as the lowest number is 0 and each number higher than that is increased by 1.
- **Integer representation.** The same philosophy apply, but in order to represent negative number too, we have to reserve first bit  $s$  to represent sign.
- **Floating point representation (IEEE-754).** We represent real values by using exponentiation (the first bit is reserved for sign, some of them are reserved for exponent representation and the remaining bits are used to represent fractional part of value). See figure for  $b = 32$  bits:



**Figure:** IEEE 754 representation using 32 bits. Source: Wikimedia Commons.

## Concept of low/mixed precision

- Each representation system has its own representation range and precision. See the following chart for a comparison:

Datatype	Bits	Representation range	Precision
float32	32	$[-3.4 \times 10^{38}, 3.4 \times 10^{38}]$	$1.19 \times 10^{-7}$
float16	16	$[-6.6 \times 10^4, 6.6 \times 10^4]$	$4.88 \times 10^{-4}$
int32	32	$[-2.15 \times 10^6, 2.15 \times 10^6]$	0.5
int8	8	$[-128, 127]$	0.5
int4	4	$[-8, 7]$	0.5
bool	1	true, false	-

**Table:** Comparison between different datatypes representations. Precision is established according to widespread definition in different programming languages.

- The computation resources to operate with each datatype also varies. For example, for an Intel Core i3 2370M, sum and multiplication are 38% faster in int32 than in float32.
- Usually, the values used in Machine Learning require the precision of floating-point representation, at the expense of the computation resources required.

## Concept of low/mixed precision

- Each representation system has its own representation range and precision. See the following chart for a comparison:

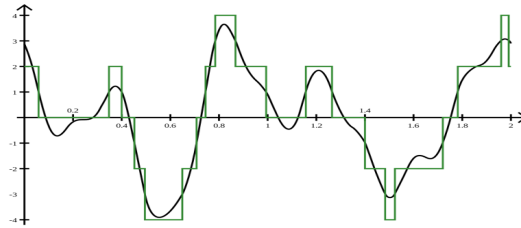
Datatype	Bits	Representation range	Precision
float32	32	$[-3.4 \times 10^{38}, 3.4 \times 10^{38}]$	$1.19 \times 10^{-7}$
float16	16	$[-6.6 \times 10^4, 6.6 \times 10^4]$	$4.88 \times 10^{-4}$
int32	32	$[-2.15 \times 10^6, 2.15 \times 10^6]$	0.5
int8	8	$[-128, 127]$	0.5
int4	4	$[-8, 7]$	0.5
bool	1	true, false	-

**Table:** Comparison between different datatypes representations. Precision is established according to widespread definition in different programming languages.

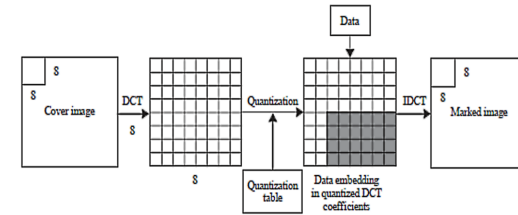
- The computation resources to operate with each datatype also varies. For example, for an Intel Core i3 2370M, sum and multiplication are 38% faster in int32 than in float32.
- Usually, the values used in Machine Learning require the precision of floating-point representation, at the expense of the computation resources required.
- Idea.** How about exploring solutions where other datatypes with **lower precision** than floating-point representation (full precision), or even using **mixed precision** by combining floating-point and other datatypes in the same model? This motivates the usage of quantized data.

# Quantization in other fields

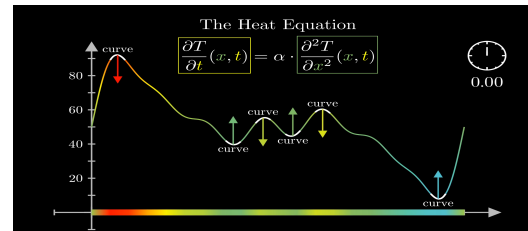
In several other fields, reducing the precision from initial values (typically when continuous values/functions need to be represented in a computer) has been useful.



(a) Signal Processing



(b) Image Processing



(c) Numerical Analysis



# Challenges in low and mixed precision in Machine Learning

The main benefits that may be achieved for using low and mixed precision in Machine Learning are: (i) reduction of inference/training time, (ii) energy consumption and (iii) reduction of memory overhead.

However, different challenges remain:

# Challenges in low and mixed precision in Machine Learning

The main benefits that may be achieved for using low and mixed precision in Machine Learning are: (i) reduction of inference/training time, (ii) energy consumption and (iii) reduction of memory overhead.

However, different challenges remain:

1. **Choose a correct representation scale using lower precision datatypes.** Trade-off between accuracy and overhead has to be considered when choosing a datatype.

# Challenges in low and mixed precision in Machine Learning

The main benefits that may be achieved for using low and mixed precision in Machine Learning are: (i) reduction of inference/training time, (ii) energy consumption and (iii) reduction of memory overhead.

However, different challenges remain:

1. **Choose a correct representation scale using lower precision datatypes.** Trade-off between accuracy and overhead has to be considered when choosing a datatype.
2. **Choose the quantization method used for each use case.** Dividing the range of values in equal chunks will suffice? Maybe for some specific use cases, it is more interesting to concentrate values in intervals where there is a concentration of values.

# Challenges in low and mixed precision in Machine Learning

The main benefits that may be achieved for using low and mixed precision in Machine Learning are: (i) reduction of inference/training time, (ii) energy consumption and (iii) reduction of memory overhead.

However, different challenges remain:

1. **Choose a correct representation scale using lower precision datatypes.** Trade-off between accuracy and overhead has to be considered when choosing a datatype.
2. **Choose the quantization method used for each use case.** Dividing the range of values in equal chunks will suffice? Maybe for some specific use cases, it is more interesting to concentrate values in intervals where there is a concentration of values.
3. **Study the stability of the results/predictions obtained in a low/mixed precision Machine Learning model with respect to full precision version.** Will minor changes in precision in individual operations affect to the final prediction?

# Challenges in low and mixed precision in Machine Learning

The main benefits that may be achieved for using low and mixed precision in Machine Learning are: (i) reduction of inference/training time, (ii) energy consumption and (iii) reduction of memory overhead.

However, different challenges remain:

1. **Choose a correct representation scale using lower precision datatypes.** Trade-off between accuracy and overhead has to be considered when choosing a datatype.
2. **Choose the quantization method used for each use case.** Dividing the range of values in equal chunks will suffice? Maybe for some specific use cases, it is more interesting to concentrate values in intervals where there is a concentration of values.
3. **Study the stability of the results/predictions obtained in a low/mixed precision Machine Learning model with respect to full precision version.** Will minor changes in precision in individual operations affect to the final prediction?
4. **Study the variation of precision of low/mixed precision Machine Learning model with respect to full precision version.** Under which circumstances can we bound the prediction error induced by quantization?

# Outline

## 1. Introduction

## 2. Quantization scales

### 2.1 Uniform quantization

### 2.2 Non-uniform quantization

### 2.3 Calibration

## 3. Low/mixed precision in Gaussian Processes

## 4. Low/mixed precision in Kernel approximation

## 5. Low/mixed precision in Neural Networks

## 6. Conclusion

# Uniform quantization

**Motivation.** How do we represent an interval  $I = [a, b]$  scaled into another interval  $\tilde{I} = [\tilde{a}, \tilde{b}]$ ?

# Uniform quantization

**Motivation.** How do we represent an interval  $I = [a, b]$  scaled into another interval  $\tilde{I} = [\tilde{a}, \tilde{b}]$ ?

- We define an affine transformation  $f : I \rightarrow \tilde{I}$  which maps the interval  $I$  into the new range, that is,  $f(x) = mx + n$ , with  $f(a) = \tilde{a}$  and  $f(b) = \tilde{b}$ . The terms  $m$  and  $n$  characterize the affine transformation (intervals  $I$  and  $\tilde{I}$  may be omitted).
- For quantized values:  $Q(x) = \lfloor x/S \rfloor - Z$  ( $m = \frac{1}{S}$ ,  $n = -Z$  and conversion into integers).



# Uniform quantization

**Motivation.** How do we represent an interval  $I = [a, b]$  scaled into another interval  $\tilde{I} = [\tilde{a}, \tilde{b}]$ ?

- We define an affine transformation  $f : I \rightarrow \tilde{I}$  which maps the interval  $I$  into the new range, that is,  $f(x) = mx + n$ , with  $f(a) = \tilde{a}$  and  $f(b) = \tilde{b}$ . The terms  $m$  and  $n$  characterize the affine transformation (intervals  $I$  and  $\tilde{I}$  may be omitted).
- For quantized values:  $Q(x) = \lfloor x/S \rfloor - Z$  ( $m = \frac{1}{S}$ ,  $n = -Z$  and conversion into integers).

**Definition (Uniform quantization function).** Let  $S \in \mathbb{R}$  and  $Z \in \mathbb{N}$ . A **quantization function**  $Q$  of scale  $S$  and zero-valued in  $Z$  is the mapping  $Q : \mathbb{R} \rightarrow \mathbb{N}$ , where:

$$Q(x) = \lfloor x/S \rfloor - Z, \forall x \in \mathbb{R}$$

The parameter  $S$  is called the **scaling factor** and the parameter  $Z$  is the **zero value factor**.

**Definition (Uniform dequantization function).** Let  $S \in \mathbb{R}$  and  $Z \in \mathbb{N}$  be, respectively, the scaling factor and the zero value factor. A **dequantization function**  $\overline{Q}$  of scale  $S$  and zero-valued in  $Z$  is the mapping  $\overline{Q} : \mathbb{N} \rightarrow \mathbb{R}$ , where:

$$\overline{Q}(n) = S(n + Z), \forall n \in \mathbb{N}$$

In general,  $\overline{Q}(Q(x)) \neq x$ , for  $x \in \mathbb{R}$ . **Information is lost** because of quantization.

# Non-uniform quantization

Sometimes, instead of distributing equally the range of values to be quantized, we can opt to concentrate values to intervals where we know that provides more useful insight than other ranges. This motivates the creation of non-uniform quantization functions.

**Definition (Non-uniform quantization function).** Let  $A = \{a_0, \dots, a_m\}$  be the different quantized values and consider a partition of  $\mathbb{R}$   $P = \{-\infty \leq \Delta_0 < \Delta_1 < \dots < \Delta_{m+1} \leq \infty\}$ . A **non-uniform quantization function** is the mapping  $Q : \mathbb{R} \rightarrow A$ , where:

$$Q(x) = a_i, \forall x \in [\Delta_i, \Delta_{i+1}), i \in \mathbb{N}$$

**Definition (Non-uniform dequantization function).** Let  $A = \{a_0, \dots, a_m\}$  be the different quantized values associated to the partition of  $\mathbb{R}$   $P = \{-\infty \leq \Delta_0 < \Delta_1 < \dots < \Delta_{m+1} \leq \infty\}$ . A **non-uniform dequantization function** is the mapping  $\overline{Q} : A \rightarrow \mathbb{R}$  given by:

$$\overline{Q}(a_i) = \Delta_i, \forall i \in \{1, \dots, m\}$$

In general  $\overline{Q}(Q(x)) \neq x$ , for  $x \in \mathbb{R}$  (**information loss**). Some examples of non-uniform quantization are:

- Logarithmic scale.
- Float32 to Float16 conversion (half precision).

# Calibration

The choice of the scaling factor  $S$  is essential for quantization.

- If  $S$  is too small compared to the values, several numbers won't be represented adequately in the quantized system (there would be overflow).
- If  $S$  is too big, different values would have the same quantized value, leading to poor precision.

A **calibration technique** must be applied.

# Calibration

The choice of the scaling factor  $S$  is essential for quantization.

- If  $S$  is too small compared to the values, several numbers won't be represented adequately in the quantized system (there would be overflow).
- If  $S$  is too big, different values would have the same quantized value, leading to poor precision.

A **calibration technique** must be applied. Several techniques are used in the state-of-the-art:

# Calibration

The choice of the scaling factor  $S$  is essential for quantization.

- If  $S$  is too small compared to the values, several numbers won't be represented adequately in the quantized system (there would be overflow).
- If  $S$  is too big, different values would have the same quantized value, leading to poor precision.

A **calibration technique** must be applied. Several techniques are used in the state-of-the-art:

1. **Min-max technique.** It consists of setting the lowest and highest values representable by the quantization schedule to map to the lowest  $r_{\min}$  and highest value  $r_{\max}$  in the real values to be considered, respectively.

# Calibration

The choice of the scaling factor  $S$  is essential for quantization.

- If  $S$  is too small compared to the values, several numbers won't be represented adequately in the quantized system (there would be overflow).
- If  $S$  is too big, different values would have the same quantized value, leading to poor precision.

A **calibration technique** must be applied. Several techniques are used in the state-of-the-art:

1. **Min-max technique.** It consists of setting the lowest and highest values representable by the quantization schedule to map to the lowest  $r_{\min}$  and highest value  $r_{\max}$  in the real values to be considered, respectively.
2. **Symmetric quantization technique.** If  $r_{\min}$  is the lowest value and  $r_{\max}$  is the highest value of the values to be represented, the parameter  $S$  is chosen such that the lowest value and the highest value are  $\max(|r_{\min}|, |r_{\max}|)$ .

# Calibration

The choice of the scaling factor  $S$  is essential for quantization.

- If  $S$  is too small compared to the values, several numbers won't be represented adequately in the quantized system (there would be overflow).
- If  $S$  is too big, different values would have the same quantized value, leading to poor precision.

A **calibration technique** must be applied. Several techniques are used in the state-of-the-art:

1. **Min-max technique.** It consists of setting the lowest and highest values representable by the quantization schedule to map to the lowest  $r_{\min}$  and highest value  $r_{\max}$  in the real values to be considered, respectively.
2. **Symmetric quantization technique.** If  $r_{\min}$  is the lowest value and  $r_{\max}$  is the highest value of the values to be represented, the parameter  $S$  is chosen such that the lowest value and the highest value are  $\max(|r_{\min}|, |r_{\max}|)$ .
3. **KL Divergence entropy.** The term  $S$  is chosen so that the original distribution and the quantized distribution by  $S$  minimize the Kullback–Leibler divergence. Used in TensorRT by NVIDIA.

# Calibration

The choice of the scaling factor  $S$  is essential for quantization.

- If  $S$  is too small compared to the values, several numbers won't be represented adequately in the quantized system (there would be overflow).
- If  $S$  is too big, different values would have the same quantized value, leading to poor precision.

A **calibration technique** must be applied. Several techniques are used in the state-of-the-art:

1. **Min-max technique.** It consists of setting the lowest and highest values representable by the quantization schedule to map to the lowest  $r_{\min}$  and highest value  $r_{\max}$  in the real values to be considered, respectively.
2. **Symmetric quantization technique.** If  $r_{\min}$  is the lowest value and  $r_{\max}$  is the highest value of the values to be represented, the parameter  $S$  is chosen such that the lowest value and the highest value are  $\max(|r_{\min}|, |r_{\max}|)$ .
3. **KL Divergence entropy.** The term  $S$  is chosen so that the original distribution and the quantized distribution by  $S$  minimize the Kullback–Leibler divergence. Used in TensorRT by NVIDIA.
4. **Percentile.** This consists of choosing  $S$  such as at least  $x\%$  of values of the original distribution are correctly represented in the distribution (typically, it's the 99%).



# Outline

1. Introduction
2. Quantization scales
3. Low/mixed precision in Gaussian Processes
4. Low/mixed precision in Kernel approximation
5. Low/mixed precision in Neural Networks
6. Conclusion

## Gaussian Process model

Consider a given dataset  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ , where  $x_i \in \mathbb{R}^D$  and  $y_i \in \mathbb{R}$  ( $D$  is the number of features to be considered). We approximate the values using:

$$y_i = f(x_i) + \varepsilon_i, \text{ where } f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)), \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

$m(\cdot)$ ,  $k(\cdot, \cdot)$  are the mean function and the covariance kernel, respectively.

The negative log likelihood and the corresponding gradient with hyperparameters  $\theta$  are given by:

$$\mathcal{L}(\theta) = -\log p(y|X, \theta) = \frac{1}{2} \log(|\tilde{K}|) + \frac{1}{2} y^T \tilde{K}^{-1} y + \frac{N}{2} \log(2\pi), \text{ where } \tilde{K} = (K + \sigma^2 I)_{i,j}$$

$$\nabla_{\theta} \mathcal{L} = \frac{1}{2} \text{Tr}(\tilde{K}^{-1} \nabla_{\theta} \tilde{K}) - \frac{1}{2} y^T \tilde{K}^{-1} (\nabla_{\theta} \tilde{K}) \tilde{K}^{-1} y \approx \frac{1}{M} \sum_{i=1}^M z_i^T \tilde{K}^{-1} \nabla_{\theta} \tilde{K} z_i - \frac{1}{2} y^T \tilde{K}^{-1} (\nabla_{\theta} \tilde{K}) \tilde{K}^{-1} y$$

Computation of  $\tilde{K}^{-1} y$  is required.

# Gaussian Process model

Consider a given dataset  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ , where  $x_i \in \mathbb{R}^D$  and  $y_i \in \mathbb{R}$  ( $D$  is the number of features to be considered). We approximate the values using:

$$y_i = f(x_i) + \varepsilon_i, \text{ where } f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)), \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

$m(\cdot)$ ,  $k(\cdot, \cdot)$  are the mean function and the covariance kernel, respectively.

The negative log likelihood and the corresponding gradient with hyperparameters  $\theta$  are given by:

$$\mathcal{L}(\theta) = -\log p(y|X, \theta) = \frac{1}{2} \log(|\tilde{K}|) + \frac{1}{2} y^T \tilde{K}^{-1} y + \frac{N}{2} \log(2\pi), \text{ where } \tilde{K} = (K + \sigma^2 I)_{i,j}$$

$$\nabla_{\theta} \mathcal{L} = \frac{1}{2} \text{Tr}(\tilde{K}^{-1} \nabla_{\theta} \tilde{K}) - \frac{1}{2} y^T \tilde{K}^{-1} (\nabla_{\theta} \tilde{K}) \tilde{K}^{-1} y \approx \frac{1}{M} \sum_{i=1}^M z_i^T \tilde{K}^{-1} \nabla_{\theta} \tilde{K} z_i - \frac{1}{2} y^T \tilde{K}^{-1} (\nabla_{\theta} \tilde{K}) \tilde{K}^{-1} y$$

Computation of  $\tilde{K}^{-1} y$  is required.

1. **Cholesky decomposition.**  $\rightsquigarrow$  Inefficient ( $\mathcal{O}(N^3)$ ) and numerically instable.
2. **Conjugate gradient method.**  $\rightsquigarrow$  Generally more efficient ( $\mathcal{O}(JN^2)$ , where  $J$  is the number of iterations of the method) and numerically stable.

# Conjugate gradient method

The conjugate gradient method proposes to approximate  $\tilde{K}^{-1}y$  by resolving the equation system  $\tilde{K}x = y$ . The following algorithm is followed:

---

**Algorithm** Conjugate Gradient algorithm.

---

**Require:** Linear system  $\tilde{K}x = y$ . Initial estimation  $x_0$ , tolerance  $\varepsilon$ , preconditioner function  $P(\cdot)$

**Ensure:** An estimated solution  $x_J$ ,  $J \in \mathbb{N}$ , such that  $x_J \approx \tilde{K}^{-1}y$ .

- 1: Initialize values:  $k = 0$ ,  $r_0 = \tilde{K}(x_0) - y$ ,  $d_0 = -r_0$ ,  $z_0 = P(r_0)$  and  $\gamma_0 = r_0^T z_0$ .
  - 2: **while**  $\|r_k\|_2 > \varepsilon$  **do**
  - 3:     Calculate next iteration:  $x_{k+1} = x_k + \alpha_k d_k$ , where  $\alpha_k = \frac{\gamma_k}{d_k^T \tilde{K} d_k}$ .
  - 4:     Prepare next iteration (1):  $r_{k+1} = r_k - \alpha_k \tilde{K} d_k$ ,  $z_{k+1} = P(r_{k+1})$ ,  $\gamma_{k+1} = r_{k+1}^T z_{k+1}$ .
  - 5:     Prepare next iteration (2):  $\beta_{k+1} = \frac{\gamma_{k+1}}{\gamma_k}$  and  $d_{k+1} = -r_{k+1} + \beta_{k+1} d_k$ .
  - 6:     Increase  $k$  by 1.
  - 7: **end while**
-

# Conjugate gradient method

The conjugate gradient method proposes to approximate  $\tilde{K}^{-1}y$  by resolving the equation system  $\tilde{K}x = y$ . The following algorithm is followed:

---

**Algorithm** Conjugate Gradient algorithm.

---

**Require:** Linear system  $\tilde{K}x = y$ . Initial estimation  $x_0$ , tolerance  $\varepsilon$ , preconditioner function  $P(\cdot)$

**Ensure:** An estimated solution  $x_J$ ,  $J \in \mathbb{N}$ , such that  $x_J \approx \tilde{K}^{-1}y$ .

- 1: Initialize values:  $k = 0$ ,  $r_0 = \tilde{K}(x_0) - y$ ,  $d_0 = -r_0$ ,  $z_0 = P(r_0)$  and  $\gamma_0 = r_0^T z_0$ .
  - 2: **while**  $\|r_k\|_2 > \varepsilon$  **do**
  - 3:     Calculate next iteration:  $x_{k+1} = x_k + \alpha_k d_k$ , where  $\alpha_k = \frac{\gamma_k}{d_k^T \tilde{K} d_k}$ .
  - 4:     Prepare next iteration (1):  $r_{k+1} = r_k - \alpha_k \tilde{K} d_k$ ,  $z_{k+1} = P(r_{k+1})$ ,  $\gamma_{k+1} = r_{k+1}^T z_{k+1}$ .
  - 5:     Prepare next iteration (2):  $\beta_{k+1} = \frac{\gamma_{k+1}}{\gamma_k}$  and  $d_{k+1} = -r_{k+1} + \beta_{k+1} d_k$ .
  - 6:     Increase  $k$  by 1.
  - 7: **end while**
- 

- (Maddox et al., 2022) propose to apply half precision in  $\gamma_n$ .
- An initial vanilla approach by just halving precision did not lead to good results.
- Further tuning is required to the original algorithm.  $\rightsquigarrow$  Enhanced Stability Conjugate Gradient method.

# Enhanced Stability Conjugate Gradient method (Maddox et al., 2022)

The main problem encountered by the authors was the instability of the method in half precision induced by quantization error.  
The following errors were detected:

## Enhanced Stability Conjugate Gradient method (Maddox et al., 2022)

The main problem encountered by the authors was the instability of the method in half precision induced by quantization error. The following errors were detected:

1. Overflow error of  $\gamma_n$ .  $\rightsquigarrow$  Store  $\gamma_n$  using a logarithmic scale. (**Conjugate gradient rescaling**)
2.  $r_k$  should be orthogonal between them (but are not due to half precision).  $\rightsquigarrow$  **Re-orthogonalization** at each step.

# Enhanced Stability Conjugate Gradient method (Maddox et al., 2022)

The main problem encountered by the authors was the instability of the method in half precision induced by quantization error. The following errors were detected:

1. Overflow error of  $\gamma_n$ .  $\rightsquigarrow$  Store  $\gamma_n$  using a logarithmic scale. (**Conjugate gradient rescaling**)
2.  $r_k$  should be orthogonal between them (but are not due to half precision).  $\rightsquigarrow$  **Re-orthogonalization** at each step.

---

**Algorithm** Enhanced Stability Conjugate Gradient algorithm.

---

**Require:** Linear system  $\tilde{K}x = y$ . Initial estimation  $x_0$ , tolerance  $\varepsilon$ , preconditioner function  $P(\cdot)$

**Ensure:** An estimated solution  $x_J$ ,  $J \in \mathbb{N}$ , such that  $x_J \approx \tilde{K}^{-1}y$ .

- 1: Initialize values:  $k = 0$ ,  $r_0 = \tilde{K}(x_0) - y$ ,  $d_0 = -r_0$ ,  $z_0 = P(r_0)$  and  $\log \gamma_0 = L\Sigma E(r_0^T z_0)$ .
  - 2: **while**  $\|r_k\|_2 > \varepsilon$  **do**
  - 3:     Calculate next iteration:  $x_{k+1} = x_k + \alpha_k d_k$ , where  $\alpha_k = \exp(\log \gamma_k - L(d_k^T K d_k))$ .
  - 4:     Prepare next iteration (1):  $r_{k+1} = r_k - \alpha_k \tilde{K} d_k$ ,  $z_{k+1} = P(r_{k+1})$ ,  $\log \gamma_{k+1} = L\Sigma E(r_{k+1}^T z_{k+1})$ .
  - 5:     Re-orthogonalization of  $r_{k+1}$ :  $r_{k+1} = r_{k+1} - \sum_{j=0}^k u_j^T r_{k+1} u_j$ .
  - 6:     Prepare next iteration (2):  $\beta_{k+1} = \exp(\log \gamma_{k+1} - \log \gamma_k)$  and  $d_{k+1} = -r_{k+1} + \beta_{k+1} d_k$ .
  - 7:     Increase  $k$  by 1.
  - 8: **end while**
-



## Enhanced vs original version

---

### Algorithm Conjugate Gradient algorithm.

---

**Require:** Linear system  $\tilde{K}x = y$ . Initial estimation  $x_0$ , tolerance  $\varepsilon$ , preconditioner function  $P(\cdot)$

**Ensure:** An estimated solution  $x_J$ ,  $J \in \mathbb{N}$ , such that  $x_J \approx \tilde{K}^{-1}y$ .

- 1: Initialize values:  $k = 0$ ,  $r_0 = \tilde{K}(x_0) - y$ ,  $d_0 = -r_0$ ,  $z_0 = P(r_0)$  and  $\gamma_0 = r_0^T z_0$ .
  - 2: **while**  $\|r_k\|_2 > \varepsilon$  **do**
  - 3:     Calculate next iteration:  $x_{k+1} = x_k + \alpha_k d_k$ , where  $\alpha_k = \frac{\gamma_k}{d_k^T \tilde{K} d_k}$ .
  - 4:     Prepare next iteration (1):  $r_{k+1} = r_k - \alpha_k \tilde{K} d_k$ ,  $z_{k+1} = P(r_{k+1})$ ,  $\gamma_{k+1} = r_{k+1}^T z_{k+1}$ .
  - 5:     Prepare next iteration (2):  $\beta_{k+1} = \frac{\gamma_{k+1}}{\gamma_k}$  and  $d_{k+1} = -r_{k+1} + \beta_{k+1} d_k$ .
  - 6:     Increase  $k$  by 1.
  - 7: **end while**
- 

---

### Algorithm Enhanced Stability Conjugate Gradient algorithm.

---

**Require:** Linear system  $\tilde{K}x = y$ . Initial estimation  $x_0$ , tolerance  $\varepsilon$ , preconditioner function  $P(\cdot)$

**Ensure:** An estimated solution  $x_J$ ,  $J \in \mathbb{N}$ , such that  $x_J \approx \tilde{K}^{-1}y$ .

- 1: Initialize values:  $k = 0$ ,  $r_0 = \tilde{K}(x_0) - y$ ,  $d_0 = -r_0$ ,  $z_0 = P(r_0)$  and  $\log \gamma_0 = L \Sigma E(r_0^T z_0)$ .
  - 2: **while**  $\|r_k\|_2 > \varepsilon$  **do**
  - 3:     Calculate next iteration:  $x_{k+1} = x_k + \alpha_k d_k$ , where  $\alpha_k = \exp(\log \gamma_k - L(d_k^T \tilde{K} d_k))$ .
  - 4:     Prepare next iteration (1):  $r_{k+1} = r_k - \alpha_k \tilde{K} d_k$ ,  $z_{k+1} = P(r_{k+1})$ ,  $\log \gamma_{k+1} = L \Sigma E(r_{k+1}^T z_{k+1})$ .
  - 5:     Re-orthogonalization of  $r_{k+1}$ :  $r_{k+1} = r_{k+1} - \sum_{j=0}^k u_j^T r_{k+1} u_j$ .
  - 6:     Prepare next iteration (2):  $\beta_{k+1} = \exp(\log \gamma_{k+1} - \log \gamma_k)$  and  $d_{k+1} = -r_{k+1} + \beta_{k+1} d_k$ .
  - 7:     Increase  $k$  by 1.
  - 8: **end while**
-

# Convergence discussion of Enhanced Stability Conjugate Gradient algorithm

- An important metric is the **effective dimension of a kernel**  $K$ ,  $N_{\text{eff}}(K, \sigma^2) = \sum_{i=1}^N \frac{\lambda_i}{\lambda_i + \sigma^2}$ .  $\lambda_1, \dots, \lambda_N$  represent the eigenvalues of  $K$  and  $\sigma^2$  is a hyperparameter called noise.
- The lower the value of  $N_{\text{eff}}$ , the better the generalization of the model. (T. Zhang, 2005)

# Convergence discussion of Enhanced Stability Conjugate Gradient algorithm

- An important metric is the **effective dimension of a kernel**  $K$ ,  $N_{\text{eff}}(K, \sigma^2) = \sum_{i=1}^N \frac{\lambda_i}{\lambda_i + \sigma^2}$ .  $\lambda_1, \dots, \lambda_N$  represent the eigenvalues of  $K$  and  $\sigma^2$  is a hyperparameter called noise.
- The lower the value of  $N_{\text{eff}}$ , the better the generalization of the model. (T. Zhang, 2005)

The following result was proven:

**Theorem. (Bounded generalization of Enhanced Stability Conjugate Gradient, adapted from (Maddox et al., 2022)).** Let  $K$  be the covariance kernel of a Gaussian Process model,  $(\lambda_i)_{i \in I}$  be the eigenvalues of  $K$ ,  $\sigma^2$  the noise parameter and  $Q$  be a quantization function. Then, the following inequality holds:

$$\mathbb{E} \left( \sum_{i=1}^N \frac{Q(\lambda_i)}{Q(\lambda_i) + \sigma^2} \right) \geq N_{\text{eff}}(K, \sigma^2)$$

Therefore, the half precision model is expected to perform worse in terms of generalization than the full precision version (as expected).

# Convergence discussion of Enhanced Stability Conjugate Gradient algorithm

- An important metric is the **effective dimension of a kernel**  $K$ ,  $N_{\text{eff}}(K, \sigma^2) = \sum_{i=1}^N \frac{\lambda_i}{\lambda_i + \sigma^2}$ .  $\lambda_1, \dots, \lambda_N$  represent the eigenvalues of  $K$  and  $\sigma^2$  is a hyperparameter called noise.
- The lower the value of  $N_{\text{eff}}$ , the better the generalization of the model. (T. Zhang, 2005)

The following result was proven:

**Theorem. (Bounded generalization of Enhanced Stability Conjugate Gradient, adapted from (Maddox et al., 2022)).** Let  $K$  be the covariance kernel of a Gaussian Process model,  $(\lambda_i)_{i \in I}$  be the eigenvalues of  $K$ ,  $\sigma^2$  the noise parameter and  $Q$  be a quantization function. Then, the following inequality holds:

$$\mathbb{E} \left( \sum_{i=1}^N \frac{Q(\lambda_i)}{Q(\lambda_i) + \sigma^2} \right) \geq N_{\text{eff}}(K, \sigma^2)$$

Therefore, the half precision model is expected to perform worse in terms of generalization than the full precision version (as expected). Experimentally:

- Speedup of  $\times 2$ .
- Stable half precision conjugate gradient matched closely (but worse) to convergence behavior of single precision ones.

# Outline

1. Introduction
2. Quantization scales
3. Low/mixed precision in Gaussian Processes
4. Low/mixed precision in Kernel approximation
  - 4.1 Parameters quantization
  - 4.2 Random Fourier Features
5. Low/mixed precision in Neural Networks
6. Conclusion

## Kernel approximation

- Kernel methods map an initial input data space into a higher dimensional feature space by using a kernel matrix  $K$ .
- Number of parameters increases.  $\rightsquigarrow$  Higher computational cost.
- Quantization seems to be interesting for Kernel methods.

Two approaches using quantization may be highlighted:

1. Quantization is applied to kernel  $K$  directly.  $\rightsquigarrow$  **Parameters quantization (Qin et al., 2014)**
2. Quantization is applied to parameters in a technique used to approximate the values of kernel  $K$ .  $\rightsquigarrow$  **Low Precision Random Fourier Features (Rahimi and Recht, 2007)**, for example

## Parameters quantization (Qin et al., 2014)

The main idea is to quantize the different values of the kernel matrix  $K$ .

**Definition. (Quantized kernel)** Let  $K \in \mathbb{R}^{N \times N}$  be a kernel. A **quantized kernel** is a mapping  $k_q : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$  such that there exists a function  $q : \mathbb{R}^D \rightarrow \mathbb{N}^D$ , with  $q(x) = (q_1(x_1), \dots, q_D(x_D))$ , with  $q_i : \mathbb{R} \rightarrow \mathbb{N}$  a quantization function for all  $i \in \{1, \dots, D\}$ , such that:

$$k_q(x, y) = K(q(x), q(y)), \forall x, y \in \mathbb{R}^D$$

This quantization is specially interesting in use cases where high dimensionality is found.

## Parameters quantization (Qin et al., 2014)

The main idea is to quantize the different values of the kernel matrix  $K$ .

**Definition. (Quantized kernel)** Let  $K \in \mathbb{R}^{N \times N}$  be a kernel. A **quantized kernel** is a mapping  $k_q : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$  such that there exists a function  $q : \mathbb{R}^D \rightarrow \mathbb{N}^D$ , with  $q(x) = (q_1(x_1), \dots, q_D(x_D))$ , with  $q_i : \mathbb{R} \rightarrow \mathbb{N}$  a quantization function for all  $i \in \{1, \dots, D\}$ , such that:

$$k_q(x, y) = K(q(x), q(y)), \forall x, y \in \mathbb{R}^D$$

This quantization is specially interesting in use cases where high dimensionality is found. Two techniques are used to build a quantized kernel:

1. **Additive quantized kernel.** Quantization is applied term by term and the value of each kernel is also calculated term by term (supposing data dimension independence).

$$k_q(x, y) = \sum_{d=1}^D k_{q_d}(x_d, y_d) = \sum_{d=1}^D \phi_{q_d}(x_d)^\top K_d \phi_{q_d}(y_d)$$

The kernel  $K$  is "diagonal".

2. **Block quantized kernels.** This idea generalizes the previous one. Instead of considering each dimension separately, a quantization by arranging a set of dimensions is performed.

The kernel  $K$  is "a block matrix".



## Convergence discussion of parameters quantization technique



- Currently, no theoretical guarantees of convergence has been studied for these techniques.  $\rightsquigarrow$  An open research line.

# Convergence discussion of parameters quantization technique

- Currently, no theoretical guarantees of convergence has been studied for these techniques.  $\rightsquigarrow$  An open research line.
- Experimentally, on visual feature matching benchmark:
  - Quantized kernel outperformed with just a few bits to represent each feature dimension compared to ordinary kernel (Euclidean distance or polynomial kernel).

Descriptor	Kernel	Dimensionality	Train on Yosemite		Train on Notredame		Mean
			Notredame	Liberty	Yosemite	Liberty	
SIFT[15]	Euclidean	128	24.02	31.34	27.96	31.34	28.66
SIFT[15]	$\chi^2$	128	17.65	22.84	23.50	22.84	21.71
SIFT[15]	AQK(8)	128	10.72	16.90	10.72	16.85	13.80
SIFT[15]	AQK(8)	256	9.26	14.48	10.16	14.43	12.08
SIFT[15]	BQK(8)	256	<b>8.05</b>	<b>13.31</b>	<b>9.88</b>	<b>13.16</b>	<b>11.10</b>
SQ-4-DAISY [4]	Euclidean	1360	10.08	16.90	10.47	16.90	13.58
SQ-4-DAISY [4]	$\chi^2$	1360	10.61	16.25	12.19	16.25	13.82
SQ-4-DAISY [4]	SQ [4]	1360	8.42	15.58	9.25	15.58	12.21
SQ-4-DAISY [4]	AQK(8)	$\leq 1813$	<b>4.96</b>	<b>9.41</b>	<b>5.60</b>	<b>9.77</b>	<b>7.43</b>
PR-proj [21]	Euclidean[21]	$< 64$	7.11	14.82	10.54	12.88	11.34
PR-proj [21]	AQK(16)	$\leq 102$	<b>5.41</b>	<b>10.90</b>	<b>7.65</b>	<b>10.54</b>	<b>8.63</b>

**Figure:** Performance of kernels on different datasets with different descriptors. It reports False positive rate at 95% recall.

- Even more compact encoding ( $\leq 64$ bits), after projecting to 32 dimension with PCA, it brought better performance due to potential benefits of decorrelating features and joint compression.
- Unlike in Gaussian Processes, no further fine tuning was needed.

# Random Fourier Features

- Random Fourier Features is a techniques used to approximate each element of the kernel  $K$ .
- To do so, a randomized mapping  $z : \mathbb{R}^D \rightarrow \mathbb{R}^R$ , where  $D$  is the dimension of each feature of the dataset considered (ideally  $R \ll N$ ) is built such that  $k(x, y) \approx z(x)^T z(y)$ .
- The construction of  $z$  is built by following this algorithm:

---

**Algorithm** Random Fourier Features (RFF) computation. (Rahimi and Recht, 2007)

---

**Require:** A kernel function such that  $k(x, y) = k(x - y)$ .

**Ensure:** Randomized feature map  $z : \mathbb{R}^D \rightarrow \mathbb{R}^N$  which approximates the kernel function  $k(x, y) \approx z(x)^T z(y)$ .

- 1: Compute Fourier transform  $p$  of kernel:  $p(\omega) = \frac{1}{2\pi} \int e^{-i\omega^T \delta} k(\delta) d\delta$
  - 2: Draw  $\omega_1, \dots, \omega_N \in \mathbb{R}^D$  samples iid from Fourier Transform  $p$ . Draw  $b_1, \dots, b_N \in \mathbb{R}$  iid from uniform distribution on  $[0, 2\pi]$ .
  - 3: Define  $z(x) = \sqrt{\frac{2}{D}} [\cos(\omega_1^T x + b_1) \dots \cos(\omega_N^T x + b_N)]^T$ .
-

# Random Fourier Features

- Random Fourier Features is a techniques used to approximate each element of the kernel  $K$ .
- To do so, a randomized mapping  $z : \mathbb{R}^D \rightarrow \mathbb{R}^R$ , where  $D$  is the dimension of each feature of the dataset considered (ideally  $R \ll N$ ) is built such that  $k(x, y) \approx z(x)^T z(y)$ .
- The construction of  $z$  is built by following this algorithm:

---

**Algorithm** Random Fourier Features (RFF) computation. (Rahimi and Recht, 2007)

---

**Require:** A kernel function such that  $k(x, y) = k(x - y)$ .

**Ensure:** Randomized feature map  $z : \mathbb{R}^D \rightarrow \mathbb{R}^N$  which approximates the kernel function  $k(x, y) \approx z(x)^T z(y)$ .

- 1: Compute Fourier transform  $p$  of kernel:  $p(\omega) = \frac{1}{2\pi} \int e^{-i\omega^T \delta} k(\delta) d\delta$
  - 2: Draw  $\omega_1, \dots, \omega_N \in \mathbb{R}^D$  samples iid from Fourier Transform  $p$ . Draw  $b_1, \dots, b_N \in \mathbb{R}$  iid from uniform distribution on  $[0, 2\pi]$ .
  - 3: Define  $z(x) = \sqrt{\frac{2}{D}} [\cos(\omega_1^T x + b_1) \dots \cos(\omega_N^T x + b_N)]^T$ .
- 

- (J. Zhang et al., 2019) propose to quantize the values obtained using the Random Fourier Features.  $\rightsquigarrow$  Low Precision Random Fourier Features.

## Low Precision Random Fourier Features (J. Zhang et al., 2019)

- **Idea.** Given  $b$  bits, we quantize each of each feature  $z_i(x), i \in D$  of a point data  $x$ .
- In memory, only the integer  $i$  (using  $b$  bits) representing the quantized value of each point data  $x$  is stored.
- $b$  is intended to be smaller than the usual amount of bits used in full-precision (usually, 32 bits).

# Low Precision Random Fourier Features (J. Zhang et al., 2019)

- **Idea.** Given  $b$  bits, we quantize each of each feature  $z_i(x), i \in D$  of a point data  $x$ .
- In memory, only the integer  $i$  (using  $b$  bits) representing the quantized value of each point data  $x$  is stored.
- $b$  is intended to be smaller than the usual amount of bits used in full-precision (usually, 32 bits).

The (adapted) algorithm proposed by the authors is:

---

**Algorithm** Low-Precision Random Fourier Features (LP-RFF) computation.

---

**Require:** Feature map  $z : \mathbb{R}^D \rightarrow \mathbb{R}^N$  obtained from previous algorithm. A value  $x \in \mathbb{R}^D$ .

**Ensure:** Quantized value  $\tilde{z}(x)$ .

- 1: Initialize  $\tilde{z}(x)$ .
  - 2: **for**  $i \in \{1, \dots, N\}$  **do**
  - 3:     Calculate  $z = z_i(x)$ .
  - 4:     Round  $z$  to  $\bar{z}$  with probability  $\frac{\bar{z}-z}{\bar{z}-\underline{z}}$  and to  $\underline{z}$  with probability  $\frac{z-\underline{z}}{\bar{z}-\underline{z}}$
  - 5:     Store in  $\tilde{z}_i(x)$  the integer  $j \in \mathbb{Z}$  corresponding to the rounded value of  $z$ .
  - 6: **end for**
-

# Convergence discussion of Low Precision Random Fourier Features

**Definition. (( $\Delta_1, \Delta_2$ )-spectral approximation, (Li and Li, 2021)).** Let  $A, B$  be symmetric matrices with the same order. For  $\Delta_1, \Delta_2 \in \mathbb{R}^+$ ,  $A$  is a  $(\Delta_1, \Delta_2)$ -spectral approximation of  $B$  if  $(1 - \Delta_1)B \preceq A \preceq (1 + \Delta_2)B$  (where  $X \preceq Y \iff Y - X$  is positive semidefinite matrix).

**Theorem. (Convergence of Random Fourier Features)** Let  $K$  be a kernel and  $\tilde{K}$  be an approximation of  $K$  using Random Fourier Features as indicated in Algorithm 5. Suppose that  $\tilde{K} + \lambda I$  is a  $(\Delta_1, \Delta_2)$ -spectral approximation of  $K + \lambda I$ , with  $\Delta_1 \in [0, 1)$ ,  $\Delta_2 \geq 0$  and  $\lambda \geq 0$  a regularization constant. The following inequality holds:

$$\mathcal{R}(f_{\tilde{K}}) \leq \frac{1}{1 - \Delta_1} \hat{\mathcal{R}}(f_K) + \frac{\Delta_2}{1 + \Delta_2} k_m$$

$\mathcal{R}(f_{\tilde{K}})$  denotes the expected error for the approximated kernel  $\tilde{K}$ ,  $\hat{\mathcal{R}}(f_K)$  is an upper bound for the estimated error for the kernel  $K$  and  $k_m$  is a constant that depends on  $m = \text{rank}(\tilde{K})$ .

As  $\Delta_1$  and  $\Delta_2$  become smaller, the better the approximation performed by  $\tilde{K}$ . Therefore, the approximation  $\tilde{K}$  of a kernel  $K$  using Random Fourier Features is good as long as  $\tilde{K} + \lambda I$  is a good  $(\Delta_1, \Delta_2)$ -spectral approximation of  $K + \lambda I$ .

# Convergence discussion of Low Precision Random Fourier Feature

Similarly, a convergence result was also proven for Low Precision Random Fourier Features.

**Theorem. (Convergence of Low Precision Random Fourier Features, J. Zhang et al., 2019)** Let  $K$  be a kernel,  $\tilde{K}$  be an approximation of  $K$  using Low Precision Random Fourier Feature using  $b$  bits as indicated in Algorithm 6 and  $\lambda \geq 0$  a regularization constant. Suppose that  $\|K\| \geq \lambda \geq \delta_b^2 = 2/(2^b - 1)^2$ . If  $\Delta_1 \geq 0$  and  $\Delta_2 \geq \delta_b^2/\lambda$ , then the following inequality holds:

$$P[(1 - \Delta_1)(K + \lambda I_n) \preceq \tilde{K} + \lambda I_n \preceq (1 + \Delta_2)(K + \lambda I_n)] \geq 1 - a(\exp(b_m) + \exp(c_m))$$

where  $a = 8 \text{tr}((K + \lambda I_n)^{-1}(K + \delta_b^2 I_n))$ ,  $b_m = \frac{-m\Delta_1^2}{\frac{4n}{\lambda}(1+\frac{2}{3}\Delta_1)}$  and  $c_m = \frac{-m(\Delta_2 - \frac{\delta_b^2}{\lambda})^2}{\frac{4n}{\lambda}(1+\frac{2}{3}(\Delta_2 - \frac{\delta_b^2}{\lambda}))}$ .

As a consequence, by denoting  $1 - \rho = 1 - a(\exp(b_m) + \exp(c_m))$ :

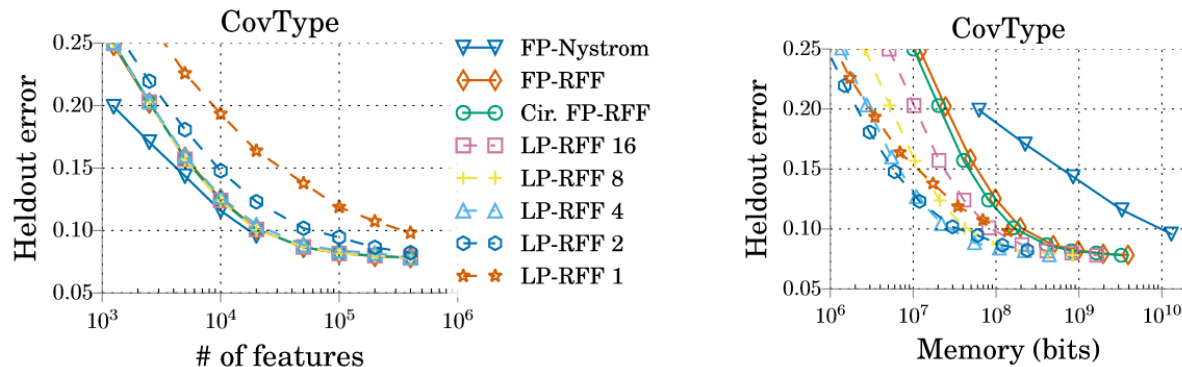
- If  $\Delta_1 \leq 3/2$  and  $m \geq \frac{8n/\lambda}{\Delta_1^2} \log\left(\frac{a}{\rho}\right)$ , then  $(1 - \Delta_1)(K + \lambda I_n) \preceq \tilde{K} + \lambda I_n$  with probability at least  $1 - \rho$ .
- If  $\Delta_2 \in [\delta_b^2/\lambda, 3/2]$  and  $m \geq \frac{8n/\lambda}{(\Delta_2 - \delta_b^2/\lambda)^2} \log\left(\frac{a}{\rho}\right)$ , then  $\tilde{K} + \lambda I_n \preceq (1 + \Delta_2)(K + \lambda I_n)$  with probability at least  $1 - \rho$ .

Therefore, as long as we have a good  $(\Delta_1, \Delta_2)$ -spectral approximation of kernel  $K$  using Low Precision Random Fourier Feature, a good estimation may be achieved even using  $b$  bits of precision.



# Convergence discussion of Low Precision Random Fourier Feature

(J. Zhang et al., 2019) experimentally compared the trade-off between accuracy and memory obtained between Low Precision Random Fourier Features and (full precision) Random Fourier Features. For example:



**Figure:** Comparison of error obtained in CovType dataset (Blackard, 1998) between Low Precision Random Fourier Features and (full precision) Random Fourier Features with respect to the number of features and the memory used, respectively. Obtained from (J. Zhang et al., 2019).

- Low precision version of Random Fourier Features achieves similar results in accuracy with respect to full precision.
- Given a restriction of available memory, low precision version tends to have a better result than full precision version.
- Model compression up to  $\times 4.7$  is achieved.

# Outline

1. Introduction
2. Quantization scales
3. Low/mixed precision in Gaussian Processes
4. Low/mixed precision in Kernel approximation
5. Low/mixed precision in Neural Networks
  - 5.1 Quantization granularity and training
  - 5.2 Mixed precision
  - 5.3 Training 4-bit quantized model
  - 5.4 Ultra quantization
6. Conclusion

## Quantization granularity and training

- An Artificial Neural Network may be characterized by the weights  $w$ , biases  $b$  and activations  $\phi$  defined for each neuron.
- In some cases, the number of parameters is huge. For example, VGG-16 network (Simonyan and Zisserman, 2014) contains about 140 million 32-bit floating parameters.  $\rightsquigarrow$  **High computational costs.**
- Quantization techniques have been applied for its potential benefits in terms of memory-saving and inference speed-up.

## Quantization granularity and training

- An Artificial Neural Network may be characterized by the weights  $w$ , biases  $b$  and activations  $\phi$  defined for each neuron.
- In some cases, the number of parameters is huge. For example, VGG-16 network (Simonyan and Zisserman, 2014) contains about 140 million 32-bit floating parameters.  $\rightsquigarrow$  **High computational costs**.
- Quantization techniques have been applied for its potential benefits in terms of memory-saving and inference speed-up.

According to quantization **granularity**, two main approaches may be distinguished:

- Same quantization scheme is applied to each layer of the neural network.  $\rightsquigarrow$  **layerwise quantization**
- Same quantization scheme is applied to subgroups of neurons within a neural network.  $\rightsquigarrow$  **groupwise quantization**

Generally, **groupwise quantization** provides better results, at the expense of consuming more memory.

## Quantization granularity and training

- An Artificial Neural Network may be characterized by the weights  $w$ , biases  $b$  and activations  $\phi$  defined for each neuron.
- In some cases, the number of parameters is huge. For example, VGG-16 network (Simonyan and Zisserman, 2014) contains about 140 million 32-bit floating parameters.  $\rightsquigarrow$  **High computational costs**.
- Quantization techniques have been applied for its potential benefits in terms of memory-saving and inference speed-up.

According to quantization **granularity**, two main approaches may be distinguished:

- Same quantization scheme is applied to each layer of the neural network.  $\rightsquigarrow$  **layerwise quantization**
- Same quantization scheme is applied to subgroups of neurons within a neural network.  $\rightsquigarrow$  **groupwise quantization**

Generally, **groupwise quantization** provides better results, at the expense of consuming more memory.

According to **when** quantization is applied in Neural Networks, two main approaches may be distinguished:

- Quantization is applied to parameters after training is finished.  $\rightsquigarrow$  **Post-Training Quantization**
- Quantization is applied to parameters while training the model (usually the last epochs).  $\rightsquigarrow$  **Quantization-Aware Training**

Generally, **Quantization-Aware Training** provides better results, at the expense of using more computational resources to train quantized parameters.

# Mixed Precision

## Single Precision

- accurate results
- high computational resource, memory usage

## Lower Precision

- reduced resource requirements
- compromise quality or accuracy

## Mixed precision

- performs calculation in lower precision while maintaining values in single precision.
- addresses trade-off between precision and efficiency

## Proposed methods:

(i) Master Copy of Weights in FP32

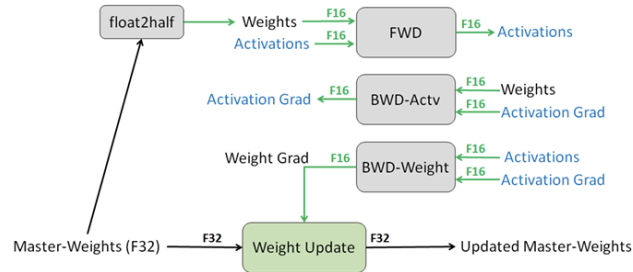
(ii) Loss Scaling

# Mixed Precision

## (i) Master Copy of Weights in FP32

- A storage called FP32 master copy is used
- Before training, weights from FP32 master copy are halved
- During forward and backward propagation, every computation is operated under FP16

→ Avoid loss of gradient information during update



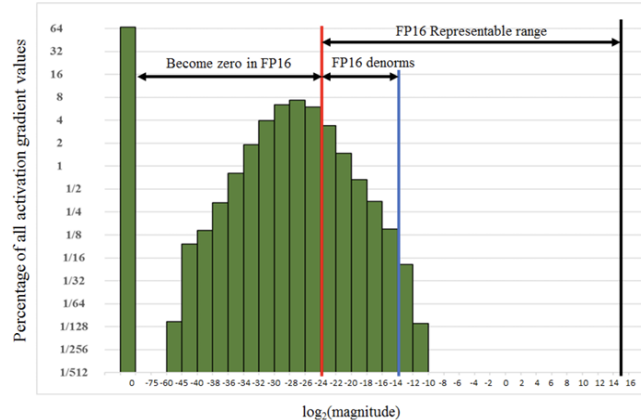
**Figure:** Mixed Precision training iteration

# Mixed Precision

## (ii) Loss Scaling

- Much of FP16 representable range was left unused.
- Weight gradient that exceeds the limits ( $2^{-24}$ ) becomes zero.

→ Scale up the gradients to occupy more representable range of FP16.



**Figure:** Histogram of gradient values during the training of Multibox SSD network



# Differentiable Neural Architecture Search

## Neural Architecture Search (NAS)

- On convolution network, constant precision level for every layer is not favored.
- Brute force search for combination of precision consumes  $\mathcal{O}(\mathcal{M}^N)$ .
- NAS is to find optimal architecture from complicated Conv layers efficiently.
- Search space defines the candidate operation (convolution, fully-connected, pooling etc.)
- Their combination presents a valid network configuration.

NAS problem can be formulated as

$$\min_{a \in A} \min_{w_a} \mathcal{L}(a, w_a) \quad (1)$$

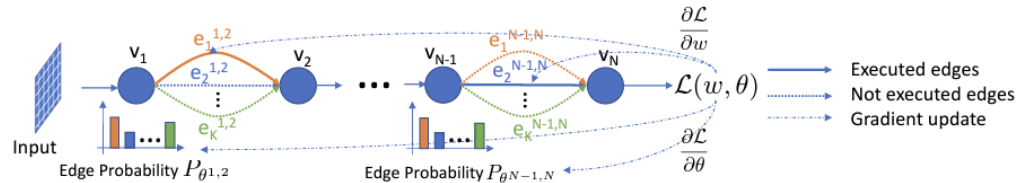
$a$  : neural architecture,  $A$  : architecture space,  $w_a$  : weights of  $a$ ,  $\mathcal{L}(a, w_a)$  : loss function

However, the loss function is differentiable with respect to  $w_a$ , but not to  $a$ .

# Differentiable Neural Architecture Search

## DNAS

- An architecture search space with a stochastic super net where nodes represent intermediate data tensors of the super net (feature maps) and edges represent operators(convolution layers).
- Each layer of the super net contains several parallel edges representing convolution operators with quantized weights and activations with different precisions.



**Figure:**  $\theta$  denotes architecture parameter and  $w$  denotes network weights. Edges are executed stochastically following the distribution  $P_{\theta}$ .

# Training 4-bit quantized model

## Problems

- **Limited dynamic range and quantization error (rounding)**
- Lack gradient representation capacity due to diversity in the gradient values per layer

### (1) Radix-4 FP4 Format

- Quantization into FP4 with common radix-2 showed severe accuracy degradation.
- To cover broad gradient value, FP4 on radix-4 with [sign, exponent, mantissa] = [1,3,0] is used, having dynamic range  $2^{12}$ .

$$\text{round}(x) = \begin{cases} 4^{n-1} & x \leq 4^n/1.6 \\ 4^n & x \geq 4^n/1.6 \end{cases}$$

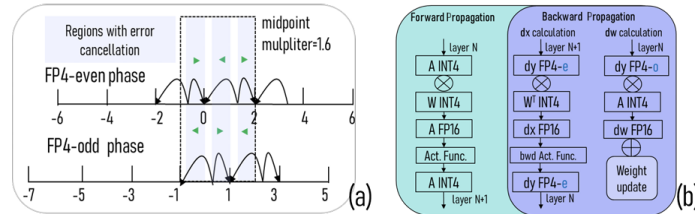
# Training 4-bit quantized model

## Problems

- **Limited dynamic range and quantization error (rounding)**
- Lack gradient representation capacity due to diversity in the gradient values per layer

### (1) Two-Phase Rounding(TPR)

- During backward and update, define two sets of radix-4 FP4 formats (FP4-even:  $2^{even}$ , FP4-odd:  $2^{odd}$ )
- Have two different quantization of  $dL/dy$ . FP4-even for  $dL/dx$  and FP4-odd for  $dL/dW$
- > Enables to look twice at the same gradient with two sets, retaining more information and handle with the quantization error.



**Figure:** (a): TPR cancellation of FP4 errors. (b): The summary of 4-bit training setup.

# Training 4-bit quantized model

## Problems

- Limited dynamic range and quantization error (rounding)
- **Lack gradient representation capacity due to diversity in the gradient values per layer**

## (2) GradScale: Trainable Layer-wise Gradient Scaling

- Commonly loss scale is conducted before backpropagation to align value of gradients to value of representable precision format  
-> however different layers behave different range of gradients
- Rather than fixed scaling factor, scale up the output gradient of each layer respectively.
- Scaling factor per layer is obtained as training parameters.
- GradScale can capture the gradients of all the layers effectively enabling FP4 quantization

# Training 4-bit quantized model

## Problems

- Limited dynamic range and quantization error (rounding)
- **Lack gradient representation capacity due to diversity in the gradient values per layer**

## (2) GradScale: Trainable Layer-wise Gradient Scaling

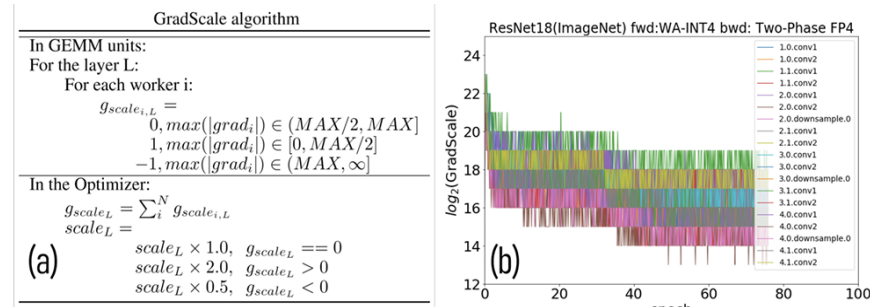


Figure 3: GradScale—a per-layer trainable scaling factor: (a) GradScale Update Algorithm (b) Scaling factor for FP4 gradients for each layer of ResNet18(ImageNet). As can be seen, different layers exhibit widely different ranges—so a layer-wise factor is critical to prevent overflow and underflows.

# Training 4-bit quantized model

## QLoRA: Quantized Low-Rank Adaptation

: 4-bit quantization finetuning approach that finetunes a 65B parameter model on a single 48GB GPU

- The original pre-trained weights of the model are quantized to 4-bit and kept fixed during finetuning.
- A small number of trainable parameters in the form of low-rank adapters are introduced during finetuning.
- These adapters are trained to adapt the pre-trained model to the specific task it is being finetuned for, in FP32 format.

## Implications

- QLoRA allows the neural network structure to learn it is fine-tuned on a specific task, with model consisting original weights in 4-bit and low-rank adapters in higher precision format.
- Present a potential usage in deployment to mobile phones or other low resource settings.

LLaMA Size Dataset	Mean 5-shot MMLU Accuracy								Mean
	7B		13B		33B		65B		
	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

# Ultra Quantization

**BinaryConnect** : binarize weights into +1 or -1 (1 bit) during forward and backward propagation

→ Not only lower memory but eliminate multiplication.

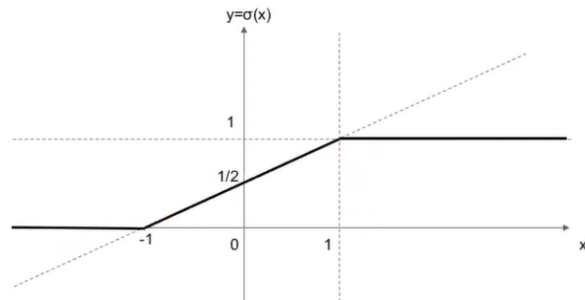
- Deterministic binarization

$$w_b = \begin{cases} +1 & w \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Stochastic binarization

$$w_b = \begin{cases} +1 & \text{with probability } p = \sigma(w) \\ -1 & \text{with probability } 1 - p \end{cases}$$

$$\sigma(x) = \text{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right)$$





# Ultra Quantization

**BinaryConnect** : binarize weights into +1 or -1 (1 bit) during forward and backward propagation

→ Not only lower memory but eliminate multiplication.

---

**Algorithm** SGD training with BinaryConnect.  $C$  is the cost function for minibatch and the functions  $\text{binarize}(w)$  and  $\text{clip}(w)$  specify how to binarize and clip weights.  $L$  is the number of layers.

---

**Require:** a minibatch of (inputs, targets), previous parameters  $w_{t-1}$  (weights) and  $b_{t-1}$  (biases), and learning rate  $\eta$ .

**Ensure:** update parameters  $w_t$  and  $b_t$ .

- 1: **1. Forward Propagation**
  - 2:  $w_b \leftarrow \text{binarize}(w_{t-1})$
  - 3: For  $k = 1$  to  $L$ , compute  $a_k$  knowing  $a_{k-1}$ ,  $w_b$  and  $b_{t-1}$
  - 4: **2. Backward Propagation**
  - 5: Initialize output layer's activations gradient  $\frac{\partial C}{\partial a_L}$
  - 6: For  $k = L$  to 2, compute  $\frac{\partial C}{\partial a_{k-1}}$  knowing  $\frac{\partial C}{\partial a_k}$  and  $w_b$
  - 7: **3. Parameter update**
  - 8: Compute  $\frac{\partial C}{\partial w_b}$  and  $\frac{\partial C}{\partial b_{t-1}}$  knowing  $\frac{\partial C}{\partial a_k}$  and  $a_{k-1}$
  - 9:  $w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$
  - 10:  $b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$
-

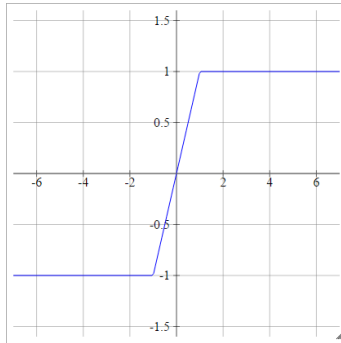
# Ultra Quantization

**Binarized NN** : binarize not only weight but also activation

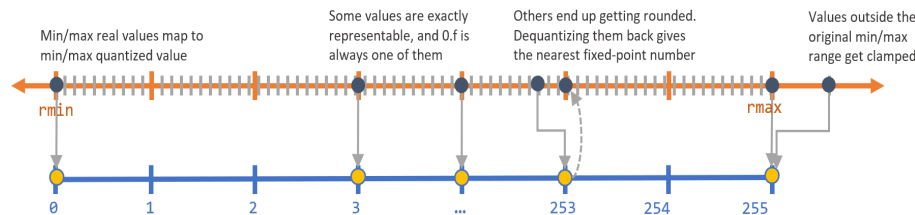
△ Sign function that was used to binarize during forward propagation cannot be differentiated during backward propagation

→ straight-through Estimator (STE) method is used to approximate optimal function under certain threshold and here, *hardtanh* function is used as STE.

$$Htanh(x) = \text{clip}(x, -1, 1) = \max(-1, \min(1, x))$$



**Integer-only Arithmetic Inference** : operates inference using 8-bit integer mainly and a few 32-bit integer



Real value (32-bit real)

$$r = S(q - Z)$$

Scaling parameter (32-bit real)

Quantized value (B-bit integer)

Offset parameter (B-bit integer)

The normal convolution  $a^{(i,k)} = \sum_{j=1}^N x^{(i,j)} w^{(j,k)} + b^{(i,k)}$  is therefore represented as:

$$a_q^{(i,k)} = Z_a + \frac{S_x S_w}{S_a} \sum_{j=1}^N (x_q^{(i,j)} - Z_x)(w_q^{(j,k)} - Z_w) + \frac{S_b}{S_a} (b_q^{(i,k)} - Z_b)$$

- No real values at all during inference.
  - outputs of conv and batchnorm are all integers
  - It can be executed on integer-arithmetic-only hardware, such as edge TPU
- Shows possibility to propel visual recognition technologies into real-time and low-end phone market.

# Outline

1. Introduction
2. Quantization scales
3. Low/mixed precision in Gaussian Processes
4. Low/mixed precision in Kernel approximation
5. Low/mixed precision in Neural Networks
6. Conclusion

# Conclusion

## 1. Introduction to Quantization:

- Overviewed the concept of quantization and its application across diverse fields within machine learning.

## 2. Gaussian Processes:

- Explored reduced precision in Gaussian Processes using techniques such as conjugate gradient re-scaling and kernel quantization for efficient modeling.









## 3. Neural Networks:

- Dive into the extensive study of quantization in Neural Networks, highlighting advancements like mixed precision and various extreme quantization.

## 4. Future Research Directions:

- Outlined future research areas, emphasizing the premature stage of quantization in Gaussian Processes and Kernel approximation.
- Raised critical questions in Neural Networks, such as theoretical guarantees for specific network families and the possibility of establishing a general guideline for quantization in Machine Learning.

# References I

-  Blackard, J. (1998). Coverttype [DOI: <https://doi.org/10.24432/C50K5N>].
-  Li, X., & Li, P. (2021). Quantization algorithms for random fourier features. *International Conference on Machine Learning*, 6369–6380.
-  Maddox, W. J., Potapcynski, A., & Wilson, A. G. (2022). Low-precision arithmetic for fast gaussian processes. *Uncertainty in Artificial Intelligence*, 1306–1316.
-  Qin, D., Chen, X., Guillaumin, M., & Gool, L. V. (2014). Quantized kernel learning for feature matching. *Advances in Neural Information Processing Systems*, 27.
-  Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20.
-  Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
-  Zhang, J., May, A., Dao, T., & Ré, C. (2019). Low-precision random fourier features for memory-constrained kernel approximation. *The 22nd International Conference on Artificial Intelligence and Statistics*, 1264–1274.
-  Zhang, T. (2005). Learning bounds for kernel regression using effective data dimensionality. *Neural computation*, 17(9), 2077–2098.