LEAN & GREEN INFERENCE WITH PYTORCH QUANTIZATION

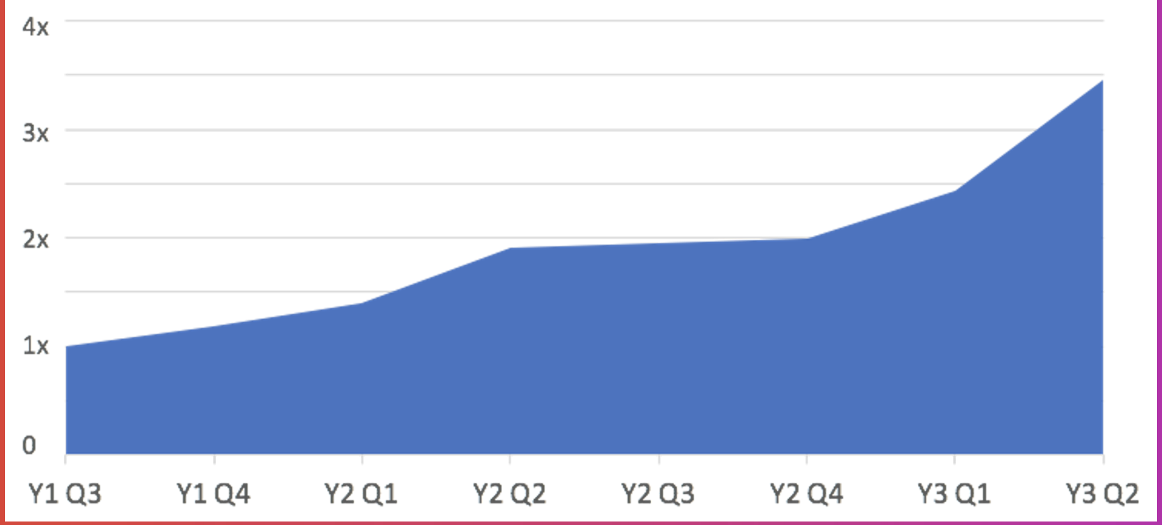
SURAJ SUBRAMANIAN
PYTORCH

AGENDA

EFFICIENT AI: NEED OF THE HOUR

NEED FOR EFFICIENT AI

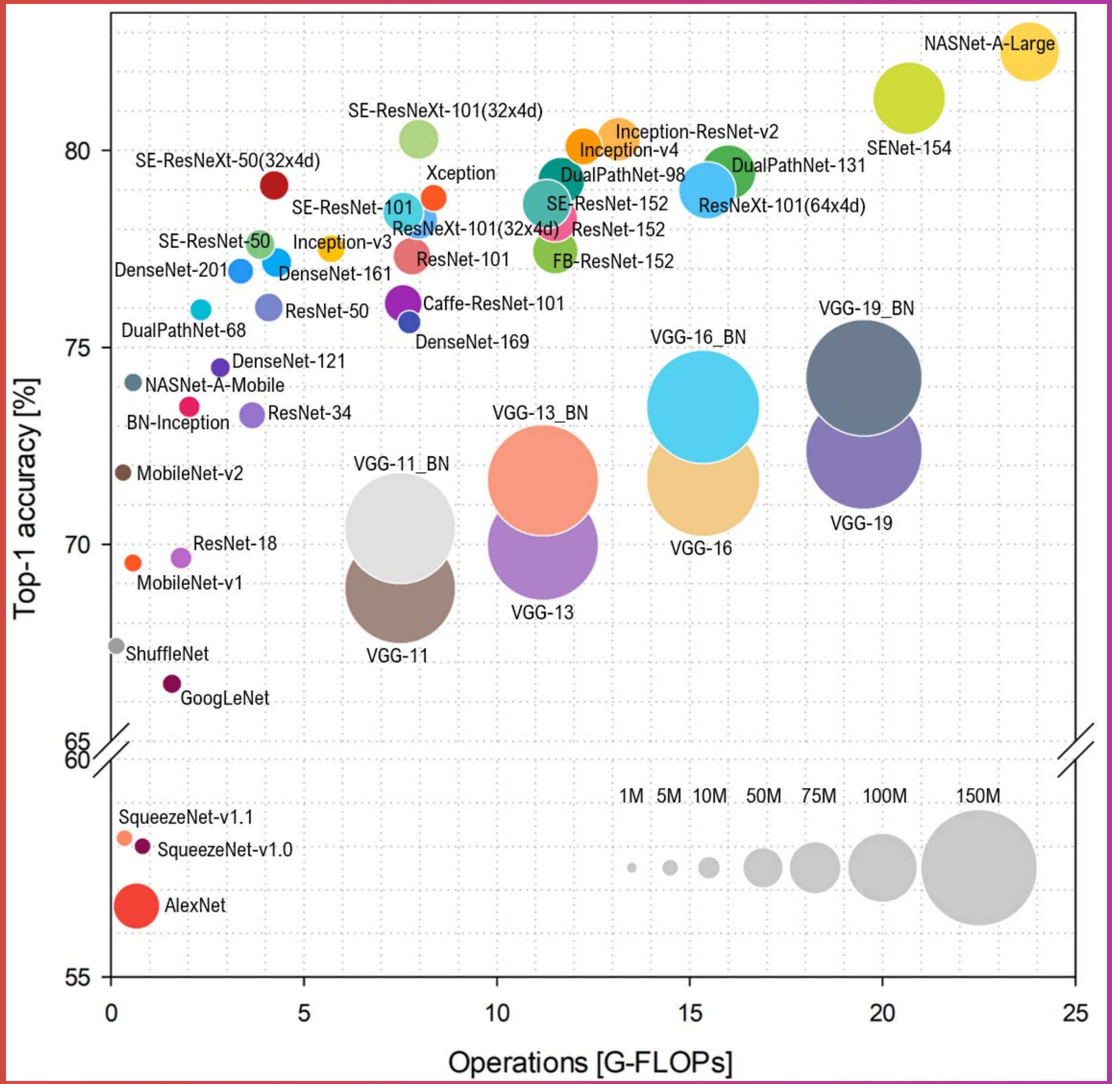
DL INFERENCE POWER CONSUMPTION IS DOUBLING EVERY YEAR



Source: Deep Learning Inference in Meta Data Centers

NEED FOR EFFICIENT AI

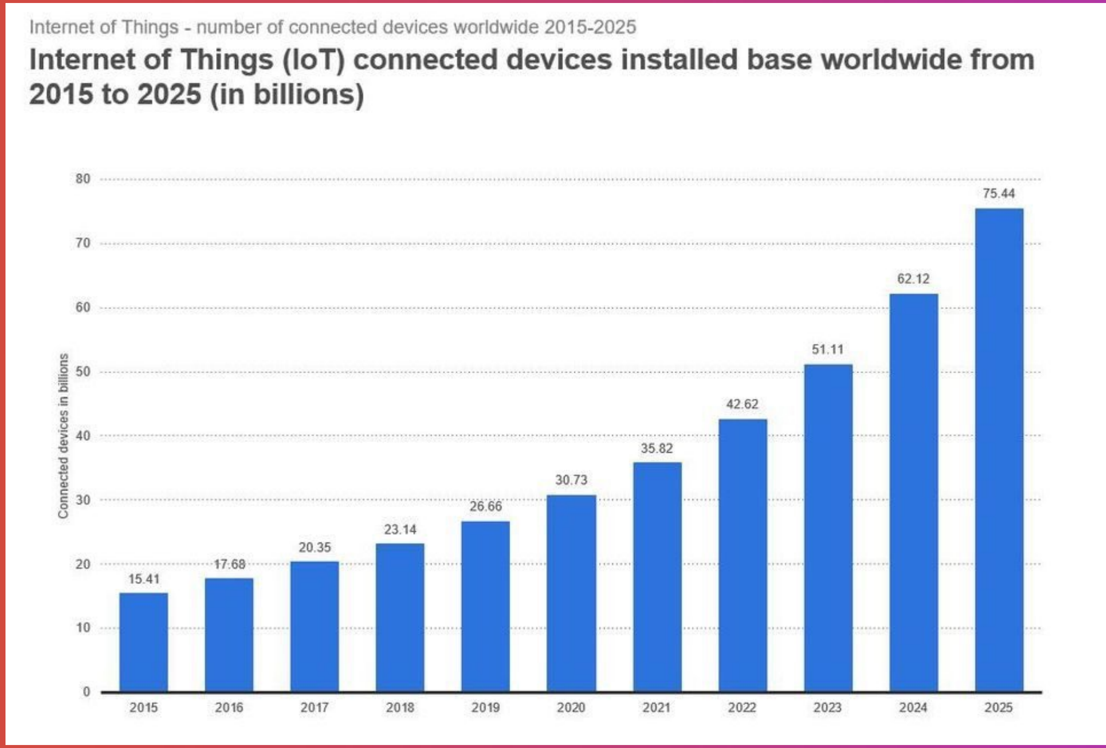
BIGGER MODELS = BETTER ACCURACY



Bianco, Simone, et al. "Benchmark analysis of representative deep neural network architectures.

NEED FOR EFFICIENT AI

EDGE DEVICES ARE PROLIFERATING



Source: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

EFFICIENT AI: SOLUTIONS?

NEED FOR EFFICIENT AI

Solutions?

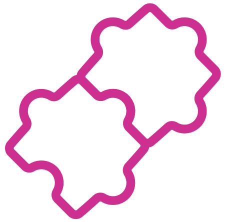
HW/SW CO-DESIGN

- New custom accelerators like TPU
- Optimized kernels to run on AI-specific hardware

NEED FOR EFFICIENT AI

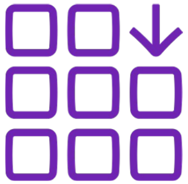
Solutions?

HW/SW CO-DESIGN



- New custom accelerators like TPU
- Optimized kernels to run on AI-specific hardware

EFFICIENT MODEL ARCHITECTURES



- Research-intensive effort
- Insufficient accuracy for critical applications

NEED FOR EFFICIENT AI

Solutions?

HW/SW CO-DESIGN

- New custom accelerators like TPU
- Optimized kernels to run on AI-specific hardware

EFFICIENT MODEL ARCHITECTURES

- Research-intensive effort
- Insufficient accuracy for critical applications

MODEL COMPRESSION

- Simpler approach that doesn't require new hardware or model architectures

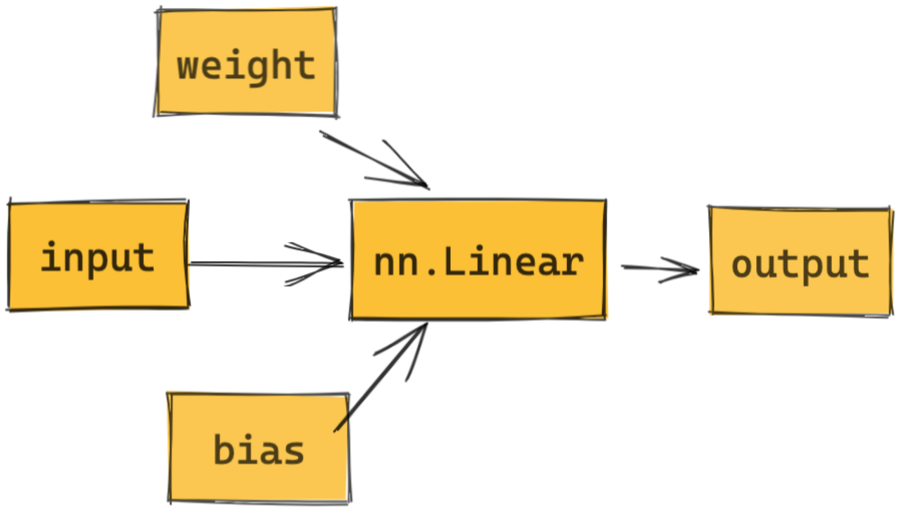
Q U A N T I Z A T I O N 1 0 1

QUANTIZATION 101

Quantization = Reduce the size of data

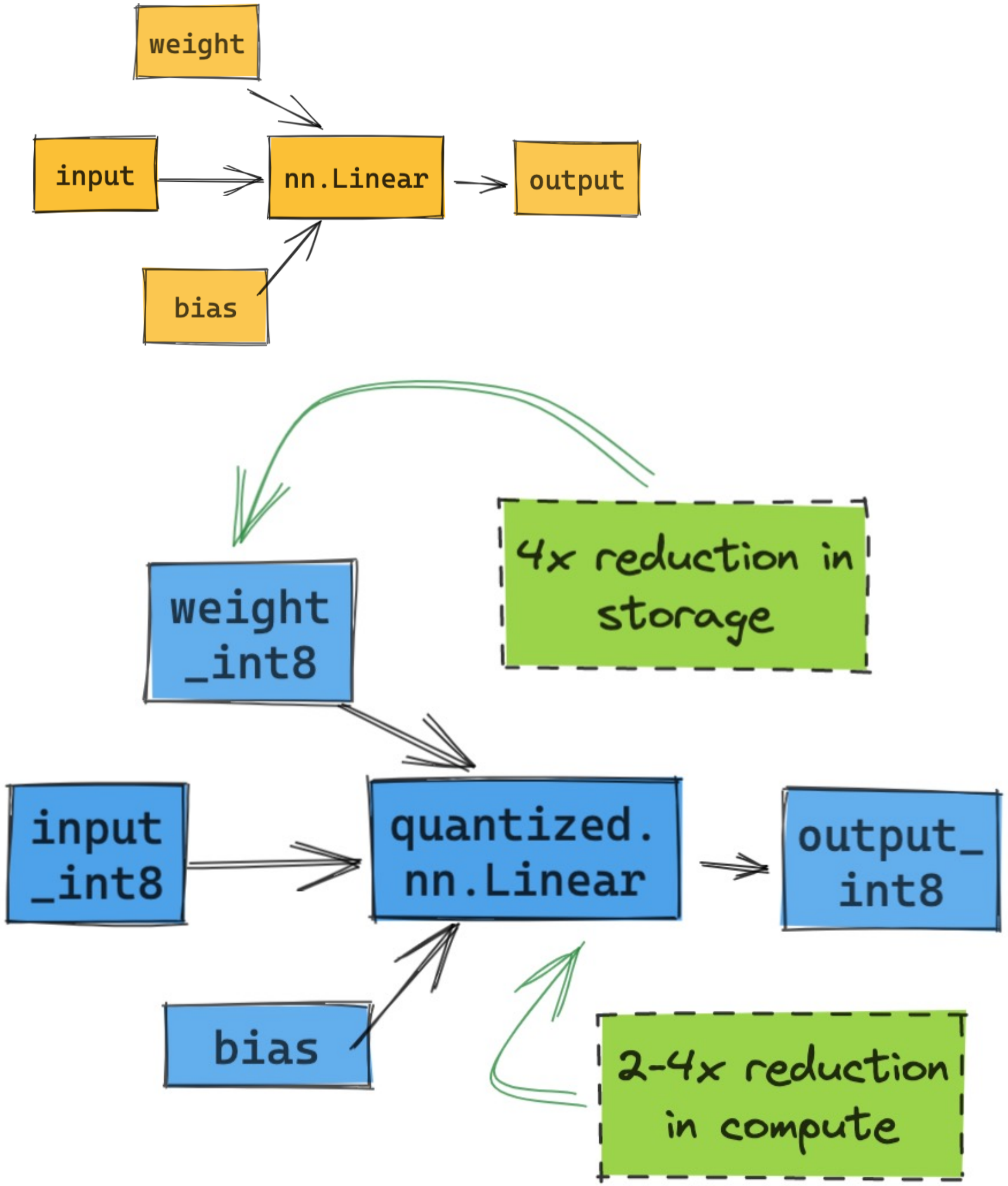
QUANTIZATION 101

INT8 Arithmetic is lighter and faster



QUANTIZATION 101

INT8 Arithmetic is lighter and faster



QUANTIZATION 101

Mapping functions

```
# mapping functions translate floating-point
numbers to integer numbers
```

QUANTIZATION 101

Mapping functions

```python
# mapping functions translate floating-point numbers to integer numbers

# floor, ceil and round are also quantization mapping functions

import math

print(math.floor(3.14159265359))
print(math.ceil(3.14159265359))
print(round(3.14159265359))
```

QUANTIZATION 101

Mapping functions

$$Q(r) = round(r/S + Z)$$

```python
# mapping functions translate floating-point numbers to integer numbers


# floor, ceil and round are also quantization mapping functions

import math

print(math.floor(3.14159265359))
print(math.ceil(3.14159265359))
print(round(3.14159265359))


# affine mapping function
import torch

def scale_transform(x, S, Z):
  x_q = 1/S * x + Z
  x_q = torch.round(x_q).to(torch.int8)
  return x_q
```

QUANTIZATION 101

Mapping functions

$$Q(r) = round(r/S + Z)$$

$$y = mx + c$$

```
# mapping functions translate floating-point
numbers to integer numbers


# floor, ceil and round are also quantization
mapping functions

import math

print(math.floor(3.14159265359))
print(math.ceil(3.14159265359))
print(round(3.14159265359))


# affine mapping function
import torch

def scale_transform(x, S, Z):
  x_q = 1/S * x + Z
  x_q = torch.round(x_q).to(torch.int8)
  return x_q
```

QUANTIZATION 101

Quantization Parameters

Quantization Parameters

Scaling Factor

Ratio of input-range to output-range

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

$$\frac{[0, \ldots, 99]}{[-2, 2]} = 20$$

QUANTIZATION 101

Quantization Parameters

Scaling Factor

Ratio of input-range to output-range

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

TITLE IN ALL CAPS

Quantization Parameters

Scaling Factor

Ratio of input-range to output-range

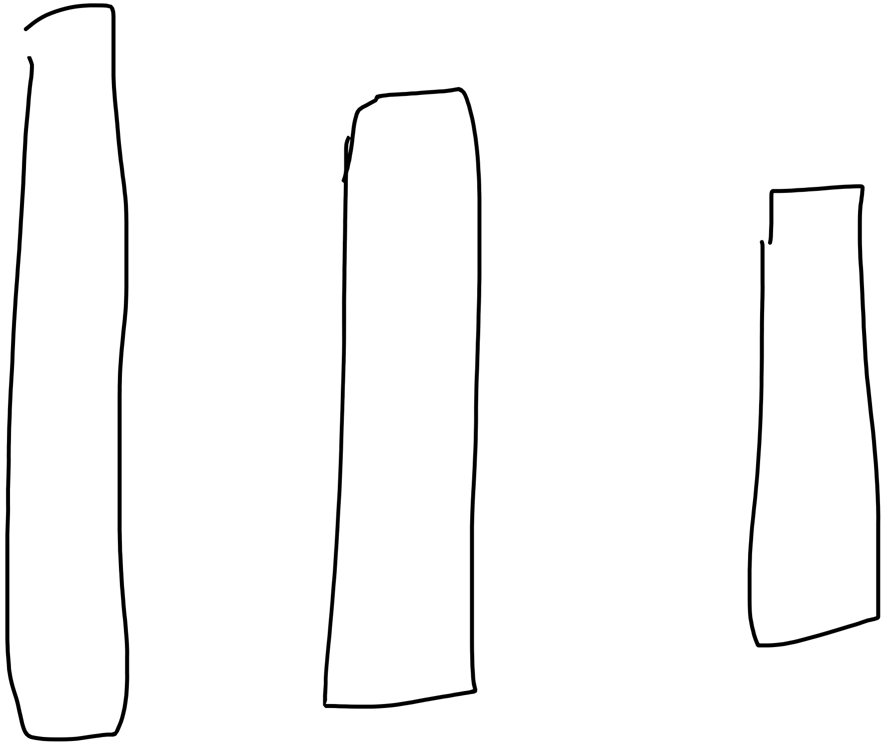
$$S = \frac{\beta - \alpha}{\beta_q - \alpha_q}$$

Zero-point

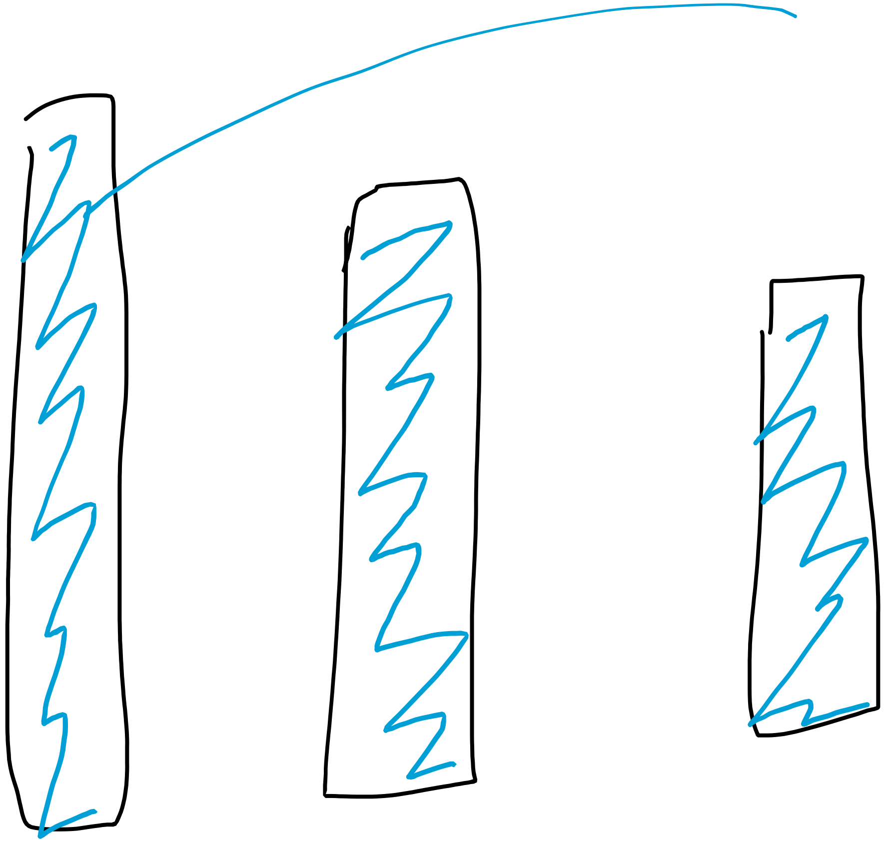
"Bias" term that maps 0 in the input space to 0 in the output space

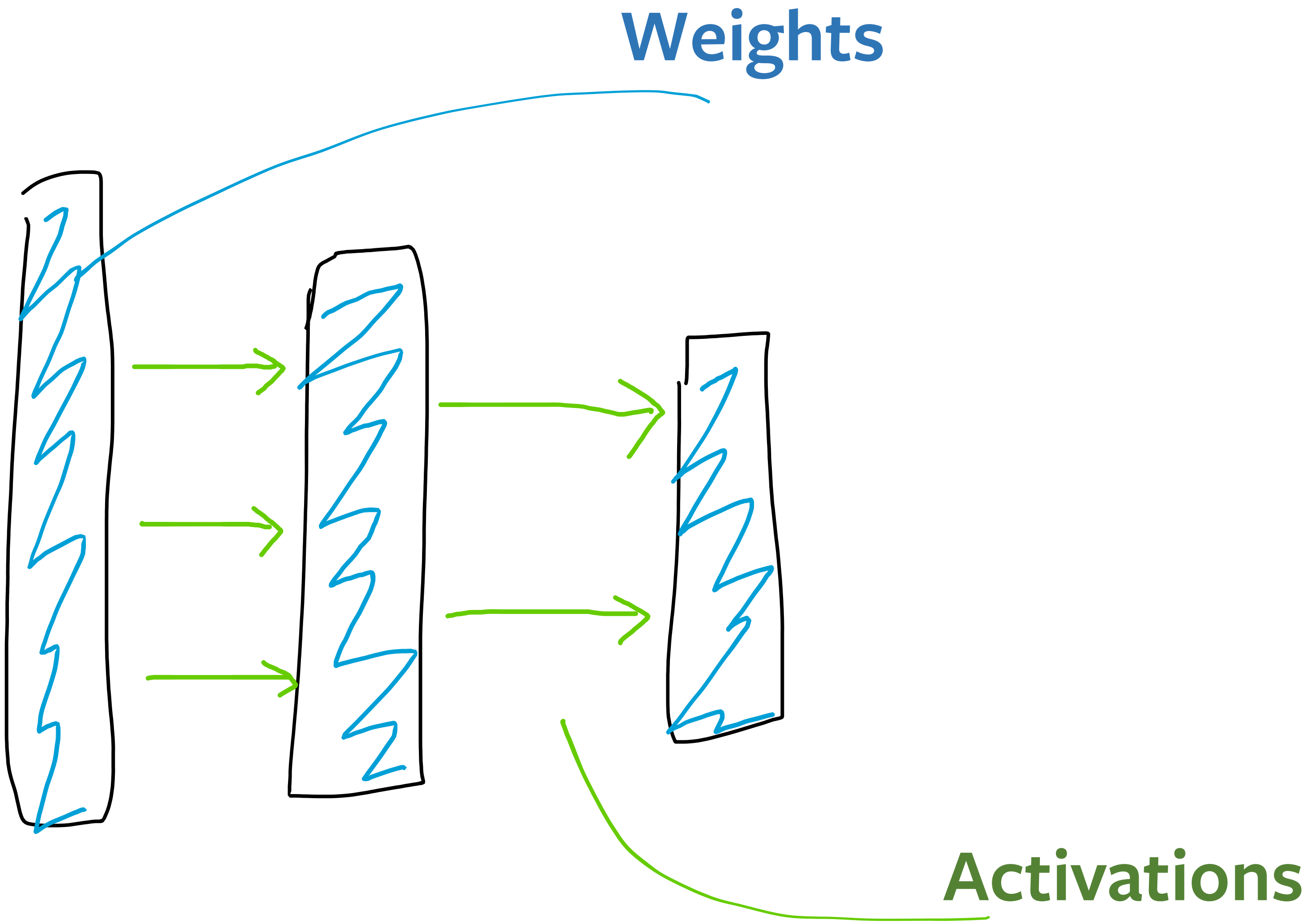
$$Z = -(\frac{\alpha}{S} - \alpha_q)$$

Q U A N T I Z A T I O N T E C H N I Q U E S I N P Y T O R C H



Weights




Weights
Activations

Quantization in PyTorch

Quantization in PyTorch

Post-Training Quantization

Quantization in PyTorch

Post-Training Quantization

Dynamic

(weights only)

QUANTIZATION TECHNIQUES

Dynamic Quantization

- Model weights are pre-quantized
- Activations are quantized on the fly as they come in ("dynamic")
- Good for LSTMs, Transformers, MLPs
- Works well with small batch sizes

- Simple; one-line API call
- Higher overhead

```
# load or train your model
model = WordLanguageModel()
model.load_state_dict(torch.load("model.pt"))

# quantize
qmodel = quantize_dynamic(model,
                          dtype=torch.quint8)

# use or deploy for C++ inference
output = qmodel(input)
torch.jit.script(qmodel).save("scripted.pt")
```

Quantization in PyTorch

Post-Training Quantization

Dynamic

(weights only)

Quantization in PyTorch

Post-Training Quantization

Dynamic

(weights only)

Static

(weights & activations)

QUANTIZATION TECHNIQUES

Static Quantization

- Weights are pre-quantized like dynamic
- Activations are also pre-quantized based on expected inputs (“static”)

- Faster inference than dynamic
- More steps involved

QUANTIZATION TECHNIQUES

Static Quantization Steps

```
import torch
class LinearReLUModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = torch.nn.Linear(5,
10).float()
        self.relu = torch.nn.ReLU()

    def forward(self, x):
        return self.relu(self.linear(x))
```

QUANTIZATION TECHNIQUES

Step 1: Modify the model

Add Stubs

- Surround the quantizable parts of the model

```
import torch
class LinearReLUModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = torch.nn.Linear(5,
10).float()
        self.relu = torch.nn.ReLU()

    def forward(self, x):
        return self.relu(self.linear(x))


class ModifiedLinearReLUModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = torch.nn.Linear(5,
10).float()
        self.relu = torch.nn.ReLU()
        self.quant = QuantStub()
        self.dequant = DeQuantStub()

    def forward(self, x):
        x = self.quant(x)
        x = self.relu(self.linear(x))
        x = self.dequant(x)
        return x
```

QUANTIZATION TECHNIQUES

Step 2: Fuse modules

- Fusing operations before quantizing improves performance

```python
import torch.ao.quantization as quantization
# load or train your model
model = ModifiedLinearReLUModule()
# model.load_state_dict(torch.load("model.pt"))
model.eval()

model = quantization.fuse_modules(model,
  [["linear", "relu"]])

print("fused:", model)
```

```
fused: LinearReLUModule(
  (linear): LinearReLU(
    (0): Linear(in_features=5, out_features=10, bias=True)
    (1): ReLU()
  )
  (relu): Identity()
  (quant): QuantStub()
  (dequant): DeQuantStub()
)
```

QUANTIZATION TECHNIQUES

Step 3: Prepare model

- Insert observers around the quantizable parts of the model

```
model = quantization.prepare(model)
print("prepared:", model)
```

```
prepared: LinearReLUModule(
  (linear): LinearReLU(
    (0): Linear(in_features=5, out_features=10, bias=True)
    (1): ReLU()
    (activation_post_process): MinMaxObserver(min_val=inf,
max_val=-inf)
  )
  (relu): Identity()
  (quant): QuantStub(
    (activation_post_process): MinMaxObserver(min_val=inf,
max_val=-inf)
  )
  (dequant): DeQuantStub()
)
```

QUANTIZATION TECHNIQUES

Step 4: Calibrate the model

- Feed the model with sample data
- Sample data should be representative of the test workload

```
# collect calibration statistics
for _ in range(10):
    model(torch.randn(5, 5))

print("calibrated:", model)
```

```
calibrated: LinearReLUModule(
  (linear): LinearReLU(
    (0): Linear(in_features=5, out_features=10, bias=True)
    (1): ReLU()
    (activation_post_process): MinMaxObserver(min_val=0.0,
max_val=1.6580328941345215)
  )
  (relu): Identity()
  (quant): QuantStub(
    (activation_post_process):
MinMaxObserver(min_val=-2.481177568435669,
max_val=2.7160568237304688)
  )
  (dequant): DeQuantStub()
)
```

QUANTIZATION TECHNIQUES

Step 5: Convert the model

- Swap all FP modules with their quantized version

```
# get the quantized model
model = quantization.convert(model)
print("quantized:", model)
```

```
quantized: LinearReLUModule(
  (linear): QuantizedLinearReLU(in_features=5,
out_features=10, scale=0.013055376708507538, zero_point=0,
qscheme=torch.per_tensor_affine)
  (relu): Identity()
  (quant): Quantize(scale=tensor([0.0409]),
zero_point=tensor([61]), dtype=torch.quint8)
  (dequant): DeQuantize()
)
```

QUANTIZATION TECHNIQUES

Step 6: Deploy the model

- Script the model first
- Save/load as you would save any scripted module

```
# use or deploy for C++ inference

torch.jit.script(model).save("quantized.pt")
```

Quantization in PyTorch

Post-Training Quantization

Dynamic

(weights only)

Static

(weights & activations)

Quantization in PyTorch

Post-Training Quantization

Dynamic
(weights only)

Static
(weights & activations)

Quantization-Aware Training
(finetuning with quantization error)

QUANTIZATION TECHNIQUES

Quantization-Aware Training

- Model is optimized on training loss + quantization error
- Best suited to CNNs and MLPs

- Provides best accuracies among all techniques
- High computational costs of retraining (possibly few hundred epochs to learn the QE)

QUANTIZATION TECHNIQUES

	FP32 Accuracy	INT8 Accuracy change	CPU speedup	Technique
Resnet50	**76.1** Imagenet	**-0.2** 75.9	**2x** 214 → 102 ms, Intel Skylake-DE	Static
MobileNet v2	**71.9** Imagenet	**-0.4** 71.5	**4x** 75 → 18 ms, Snapdragon 835	QAT
BERT	**90.2** F1 (GLUE MRPC)	**0.0** 90.2	**1.6x** 581 → 313 ms, Intel Skylake-DE	Dynamic

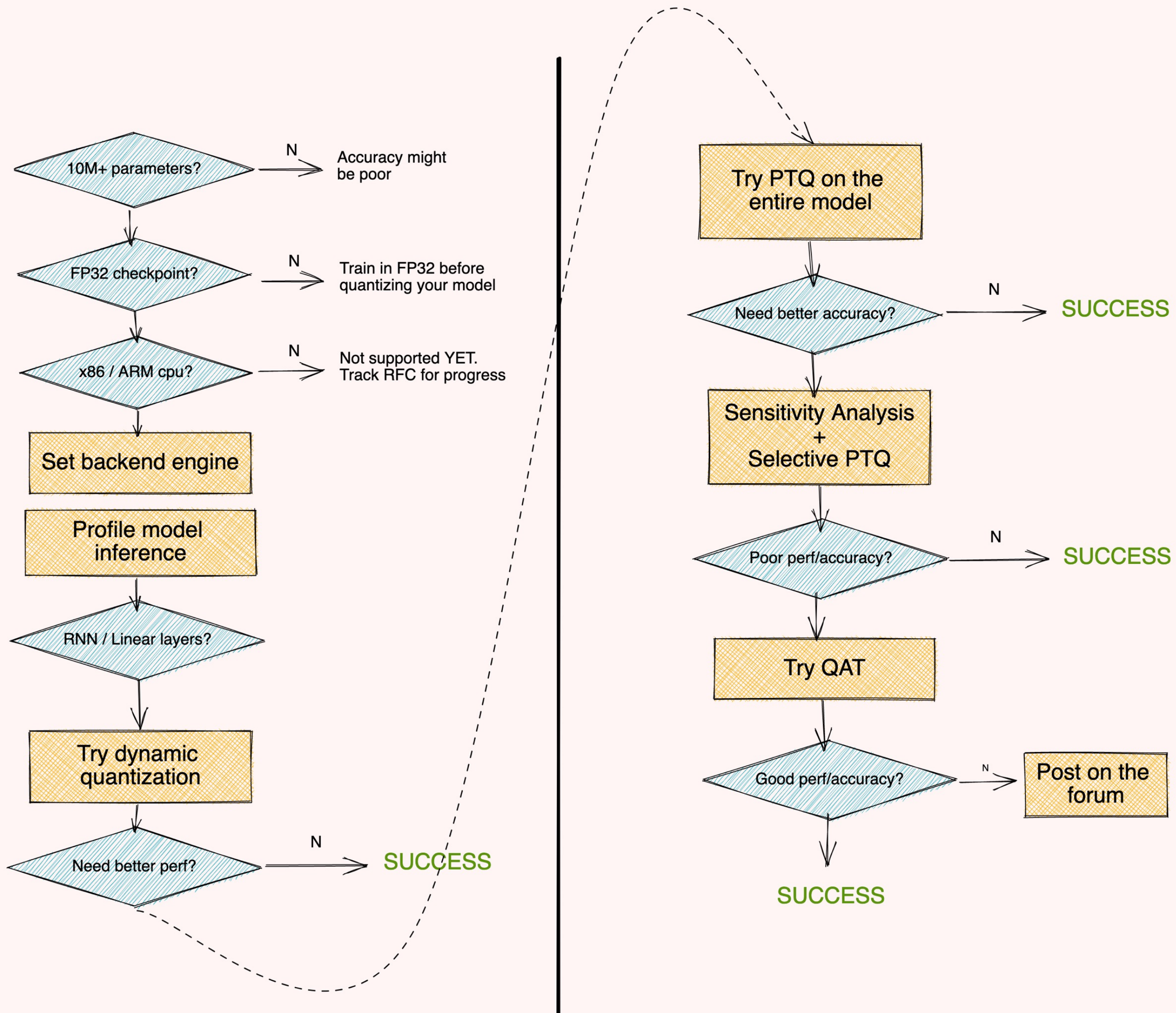
QUANTIZATION TECHNIQUES

Eager Mode vs FX Graph Mode

	Eager Mode	FX Graph Mode
Release Status	Beta	Prototype
Quantizing Modules	Supported	Supported
Operator Fusion	Manual	Automatic
Quant/Dequant Placement	Manual	Automatic
Quantizing Functional/Torch Ops	Manual	Automatic
Support for Customization	Limited	Supported
Input/Output Model Type	nn.Module	nn.Module
Input Model Restrictions	None	Must be Symbolically Traceable (no data dependent control flows)

QUANTIZATION TECHNIQUES

MAKING AN INFORMED CHOICE



HANDS ON KEYBOARD

Quant_Workflow.ipynb

WHAT'S NEXT

WHAT'S NEXT

- FX Graph Mode moving into Beta
- Support for custom backend extensions
- More integration with domain libraries (eg: quantized torchvision models)
- Define-By-Run Quantization

RESOURCES

- [Quantization — PyTorch 1.11.0 documentation]
- FX Graph Mode Quantization User Guide
- Practical Quantization in PyTorch blog
- Latest quantization topics - PyTorch Forums
- Issues · pytorch/pytorch · GitHub

Q & A