Large batch distributed training. CPU offloadings. Quantization

Seminar

Optimization for ML. Faculty of Computer Science. HSE University

Accurate, Large Minibatch SGD. Motivation

Main Pros of Big Data

The increasing data and model scale is rapidly improving accuracy

Accurate, Large Minibatch SGD. Motivation

Main Pros of Big Data

The increasing data and model scale is rapidly improving accuracy

Main Cons of Big Data

As model and data scale grow, so does training time

Accurate, Large Minibatch SGD. Motivation

Main Pros of Big Data

The increasing data and model scale is rapidly improving accuracy

Main Cons of Big Data

As model and data scale grow, so does training time

Solution: Use distributed SGD with large batch size to make **more efficient** iterations

Accurate, Large Minibatch SGD. Problem

Loss function

$$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w)$$

One Iteration of Minibatch SGD (batch size is n)

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

k Iterations of Minibatch SGD (batch size is n)

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_|} \nabla l(x, w_{t+j})$$

One Large Batch Iteration of Minibatch SGD (batch size is kn)

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_|} \nabla l(x, w_t)$$

Desired due to multi-GPU training: $\hat{w}_{t+1} \sim w_{t+k}$

Accurate, Large Minibatch SGD. Main idea

Desired due to multi-GPU training: $\hat{w}_{t+1} \sim w_{t+k}$

Main Paper Asumption

If we could assume $\nabla l(x, w_t) \sim \nabla l(x, w_{t+j})$ for $j < k$, then setting $\hat{\eta} = k\eta$ would yield $\hat{w}_{t+1} \sim w_{t+k}$

Accurate, Large Minibatch SGD. Main idea

Desired due to multi-GPU training: $\hat{w}_{t+1} \sim w_{t+k}$

Main Paper Asumption

If we could assume $\nabla l(x, w_t) \sim \nabla l(x, w_{t+j})$ for $j < k$, then setting $\hat{\eta} = k\eta$ would yield $\hat{w}_{t+1} \sim w_{t+k}$

Question

When is condition $\nabla l(x, w_t) \sim \nabla l(x, w_{t+j})$ clearly not hold?

Accurate, Large Minibatch SGD. Main idea

Desired due to multi-GPU training: $\hat{w}_{t+1} \sim w_{t+k}$

Main Paper Asumption

If we could assume $\nabla l(x, w_t) \sim \nabla l(x, w_{t+j})$ for $j < k$, then setting $\hat{\eta} = k\eta$ would yield $\hat{w}_{t+1} \sim w_{t+k}$

Question

When is condition $\nabla l(x, w_t) \sim \nabla l(x, w_{t+j})$ clearly not hold?

1. The network changes rapidly in initial training
2. Very large k causes very large $\hat{\eta}$ and makes training too unstable

Accurate, Large Minibatch SGD. Solving assumption problems

Question

How would you struggle with assumption problems?

Accurate, Large Minibatch SGD. Solving assumption problems

Question

How would you struggle with assumption problems?

Gradual warmup. Iteration-wise linear scheduler for start value $\hat{\eta} = \eta$ and finish value $\hat{\eta} = k\eta$ after ~ 5 epochs.

- avoids a sudden increase of the learning rate

Constant per-worker sample size. For global batch size kn we keep the *per-worker* sample size n constant when changing the number of workers k.

Accurate, Large Minibatch SGD. Solving assumption problems

Question

How would you struggle with assumption problems?

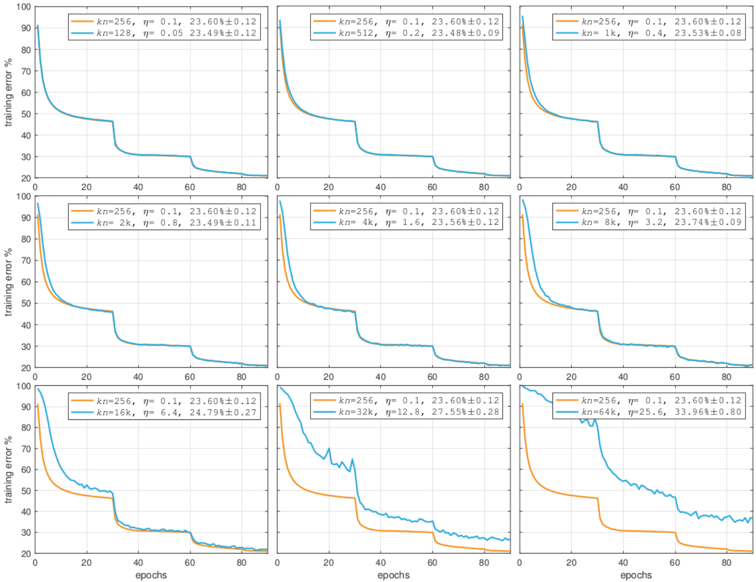
Gradual warmup. Iteration-wise linear scheduler for start value $\hat{\eta} = \eta$ and finish value $\hat{\eta} = k\eta$ after ~ 5 epochs.

- avoids a sudden increase of the learning rate

Constant per-worker sample size. For global batch size kn we keep the *per-worker* sample size n constant when changing the number of workers k.

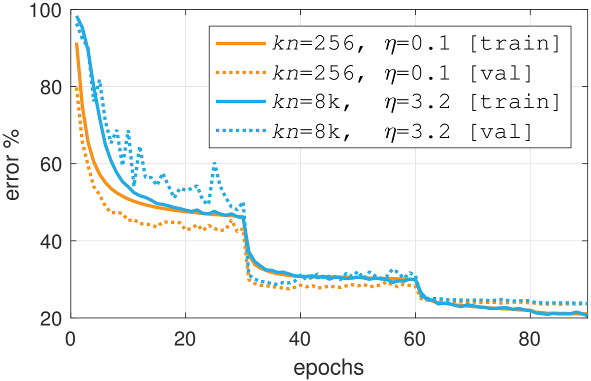
- extremly important for Batch Normalization!

Accurate, Large Minibatch SGD. Results on ImageNet



The training curves closely match the baseline (aside from the warmup period) up through $8k$ minibatches.

Accurate, Large Minibatch SGD. Results on ImageNet



Both sets of curves match closely after training for sufficient epochs.
Note that the BN statistics (for inference only) are computed using running average, which is updated less frequently with a large minibatch and thus is noisier in early training (this explains the larger variation of the validation error in early epochs).

Reduce memory usage. CPU Offloading

- Offloading the weights to the CPU and only loading them on the GPU when performing the forward pass
- CPU offloading works on submodules rather than whole models.
- Inference is much slower due to the iterative uploading and offloading.
- Colab Example Open in Colab.

Reduce memory usage. Model Offloading

- CPU Offloading makes inference slower because *submodules* are moved to GPU as needed, and they're immediately returned to the CPU when a new module runs.
- Full-model offloading is an alternative that moves whole models to the GPU, instead of handling each model's constituent *submodules*.
- During model offloading, only one of the main components of the pipeline (typically the text encoder, UNet or VAE) is placed on the GPU while the others wait on the CPU.
- Colab Example Open in Colab.

Reduce memory usage. Quantization

- Quantization maps a floating point value $x \in [\alpha, \beta]$ to a b-bit integer $x_q \in [\alpha_q, \beta_q]$.
- The quantization process is defined as

$$x_q = \text{clip}\Big(\text{round}\big(\frac{1}{s}x + z\big), \alpha_q, \beta_q\Big)$$

  And the de-quantization process is defined as

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

  The value of scale s and zero point z are

$$s = \frac{\beta - \alpha}{\beta_q - \alpha_q} \tag{1}$$

$$z = \text{round}\big(\frac{\beta\alpha_q - \alpha\beta_q}{\beta - \alpha}\big) \tag{2}$$

$$\tag{3}$$

  Note that z is an integer and s is a positive floating point number.
- Quantization allows to perform a lot of heavy DL-operations (e.g. matrix maltiplication) in integer scope using efficient integer hardware (`NVIDIA Tensor Core` or `Tensor Core IMMA operations`) and algorithms.

Reduce memory usage. Quantization

- For more theory look at *Quantization for Neural Networks* , Lei Mao: **git**.
- Colab Example Open in Colab.