Announcements

HW3 will be out next Tuesday

Project midpoint meetings should be concluded now

Final project presentation will be a poster session. More details: https://cmu-llms.org/project/#final-deliverables-instructions

- Prepare your project poster like conference posters
- Peer feedback: Each of you go through all other posters
- May have a different location, will announce

Final project report due in one month. Start working towards the finish line!

Scaling Up LLM Pretraining: Parallel Training

Chenyan Xiong

11-667

Outline

Optimization

- Optimization Basics
- Numerical Types

Optimization: Recap of Stochastic Gradient Descent

In deep learning, mini-batch learning is the norm and Stochastic Gradient Descent (SGD) is the basis optimizer

$g_t = \nabla_\theta f_t(\theta_{t-1})$ Gradient at step t of loss function $f()$

$\theta_t = \theta_{t-1} - \alpha g_t$ Updating with step size α

Compared to classic convex optimization:

- Each step only uses a small sub sample of data: stochastic sampling
- Non-convex optimization has many local optimal with different effectiveness

Optimization: Challenge of SGD

In deep learning, mini-batch learning is the norm and Stochastic Gradient Descent (SGD) is the basis optimizer

$$g_t = \nabla_\theta f_t(\theta_{t-1})$$ Gradient at step t of loss function $f\,()$

$$\theta_t = \theta_{t-1} - \alpha g_t$$ Updating with step size α

Challenge: How to select the right step size?

- Different parameters have different behaviors:
  - norm, sensitivity, influence to optimization process, etc.
  - thus have different preferences on step size
- No way to manually tune step size per parameter
  - Millions or billions of hyperparameters to tune

Figure 1: SGD on two parameter loss contours [1]

[1] Sebastian Ruder. "An overview of gradient descent optimization Algorithms". arXiv 2017

Optimization: Challenge of SGD

In deep learning, mini-batch learning is the norm and Stochastic Gradient Descent (SGD) is the basis optimizer

$$g_t = \nabla_\theta f_t(\theta_{t-1})$$

Gradient at step t of loss function $f()$

$$\theta_t = \theta_{t-1} - \underline{\alpha} g_t$$

Updating with step size α

Challenge: How to select the right step size?

→Solution: Dynamic learning rate per parameter

Adaptive gradient methods (AdaGrad [2])

$$\theta_t = \theta_{t-1} - \frac{\alpha g_t}{\underline{\sqrt{\sum_{i=1}^{t} g_i^2}}}$$

Reweight per parameter step size by its accumulated past norm

[2] Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization" JMLR 2011

Optimization: Challenge of SGD

In deep learning, mini-batch learning is the norm and Stochastic Gradient Descent (SGD) is the basis optimizer

$$g_t = \nabla_\theta f_t(\theta_{t-1})$$

Gradient at step t of loss function $f()$

$$\theta_t = \theta_{t-1} - \underline{\alpha} g_t$$

Updating with step size α

Challenge: How to select the right step size?

→Solution: Dynamic learning rate per parameter

Adaptive gradient methods (AdaGrad [2])

$$\theta_t = \theta_{t-1} - \frac{\alpha g_t}{\underline{\sqrt{\sum_{i=1}^{t} g_i^2}}}$$

Reweight per parameter step size by its accumulated past norm

- The more a parameter has been updated previously $\sqrt{\sum_{i=1}^{t} g_i^2} \uparrow$, the less its step size
- Sparse features with fewer past gradients $\sqrt{\sum_{i=1}^{t} g_i^2} \downarrow$ get boosted

[2] Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization" JMLR 2011

Optimization: Challenge of SGD

In deep learning, mini-batch learning is the norm and Stochastic Gradient Descent (SGD) is the basis optimizer

$$g_t = \nabla_\theta f_t(\theta_{t-1})$$ Gradient at step t of loss function $f\,()$

$$\theta_t = \theta_{t-1} - \alpha \underline{g_t}$$ Updating with step size α

Challenge: Local updates

- Only uses information from current mini-batch
  - Can easily stuck in local optima

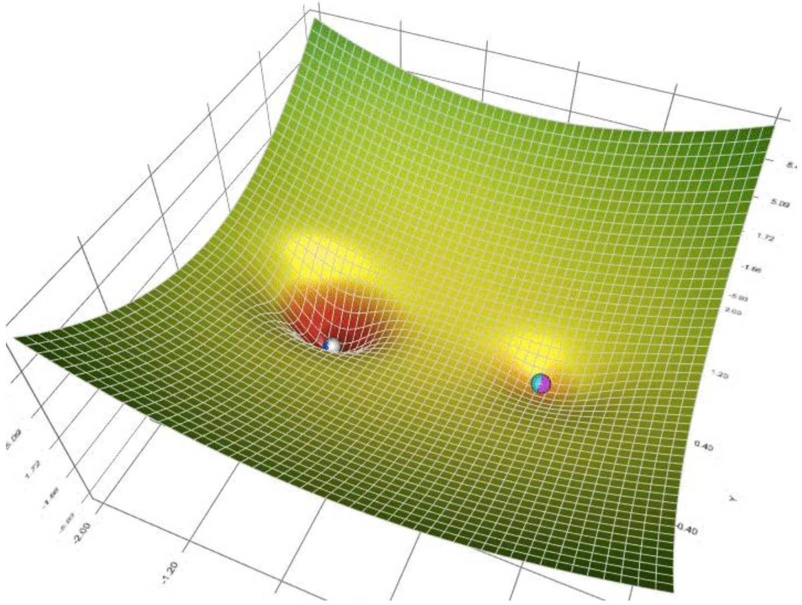


Figure 2: Optimization with Local Optima [3]

Optimization: Challenge of SGD

In deep learning, mini-batch learning is the norm and Stochastic Gradient Descent (SGD) is the basis optimizer

$g_t = \nabla_\theta f_t(\theta_{t-1})$ Gradient at step t of loss function $f()$

$\theta_t = \theta_{t-1} - \alpha g_t$ Updating with step size α

Challenge: Local updates

→ Solution: Momentum [4]

$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta f_t(\theta_{t-1})$ Momentum of Gradient

$\theta_t = \theta_{t-1} - \alpha m_t$ Updating with gradient momentum

[1] Sebastian Ruder. "An overview of gradient descent optimization Algorithms". arXiv 2017

Optimization: Challenge of SGD

In deep learning, mini-batch learning is the norm and Stochastic Gradient Descent (SGD) is the basis optimizer

$g_t = \nabla_\theta f_t(\theta_{t-1})$ — Gradient at step t of loss function $f()$

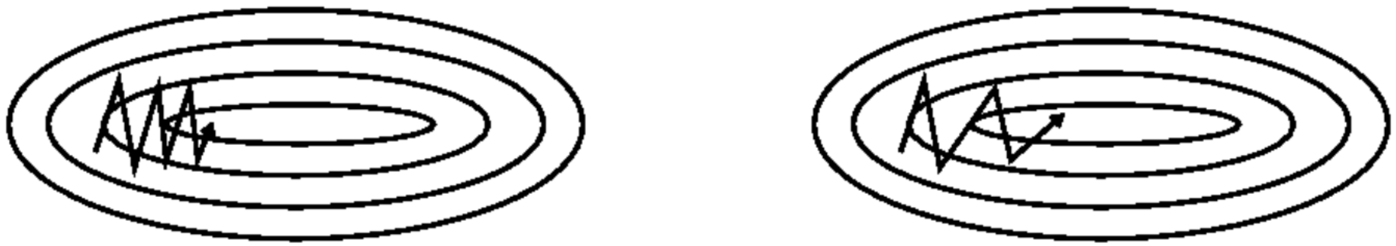
$\theta_t = \theta_{t-1} - \alpha g_t$ — Updating with step size α

Challenge: Local updates

→ Solution: Momentum [4]

$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta f_t(\theta_{t-1})$ — Momentum of Gradient

$\theta_t = \theta_{t-1} - \alpha m_t$ — Updating with gradient momentum



(a) SGD without momentum (b) SGD with momentum

Figure 3: SGD with and without Momentum [1]

Optimization: Adam Optimizer

Adam: Adaptive Moment Estimation [4]

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

---

[4] Kingma and Ba. “Adam: A Method for Stochastic Optimization”. ICLR 2015

Optimization: Adam Optimizer

Adam: Adaptive Moment Estimation [4]

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Hyperparameters that you can/should tune
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
Initializations
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

[4] Kingma and Ba. “Adam: A Method for Stochastic Optimization”. ICLR 2015

Optimization: Adam Optimizer

Adam: Adaptive Moment Estimation [4]

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Hyperparameters that you can/should tune

Initializations

Standard back-propagation for raw gradients

[4] Kingma and Ba. "Adam: A Method for Stochastic Optimization". ICLR 2015

Optimization: Adam Optimizer

Adam: Adaptive Moment Estimation [4]

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Hyperparameters that you can/should tune

Initializations

Standard back-propagation for raw gradients

Get 1st and 2nd order momentum of gradient

Optimization: Adam Optimizer

Adam: Adaptive Moment Estimation [4]

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Hyperparameters that you can/should tune

Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)

Initializations

  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

Standard back-propagation for raw gradients

    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

Get 1st and 2nd order momentum of gradient

    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

Correct momentum bias

    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

[4] Kingma and Ba. “Adam: A Method for Stochastic Optimization”. ICLR 2015

Optimization: Adam Optimizer

Adam: Adaptive Moment Estimation [4]

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Hyperparameters that you can/should tune

Initializations

Standard back-propagation for raw gradients

Get 1st and 2nd order momentum of gradient

Correct momentum bias

Dynamic per-parameter step size by 2nd order momentum

Update by 1st order momentum

Optimization: Illustrations

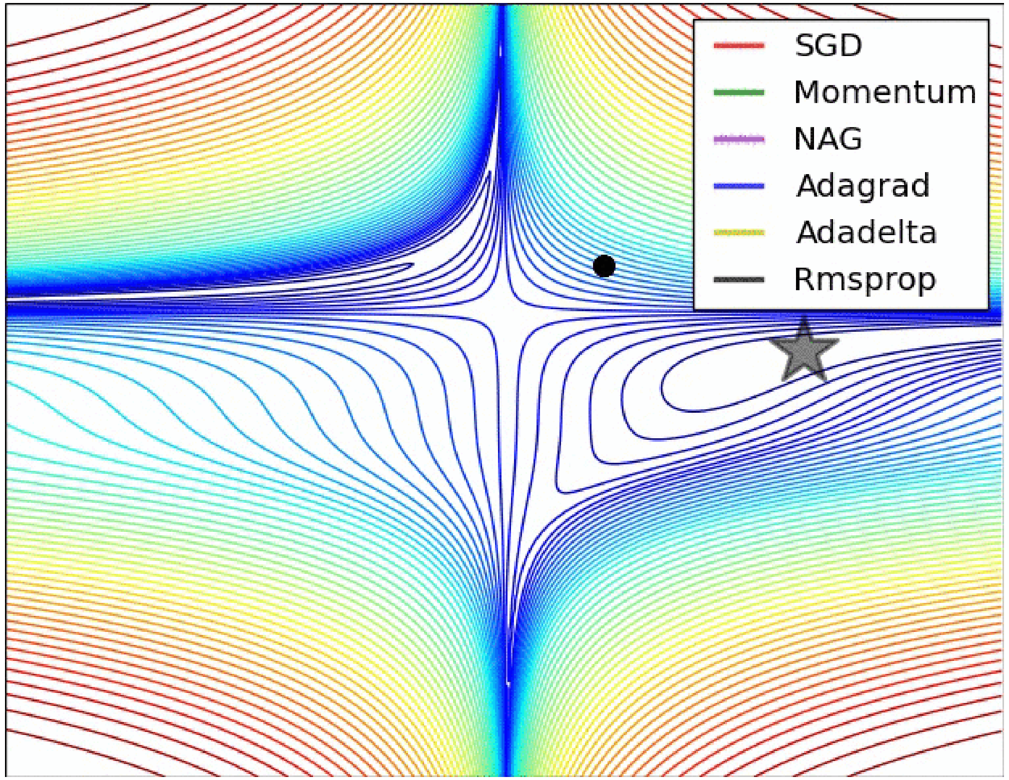


Figure 4: SGD optimization on loss surface contours [1]

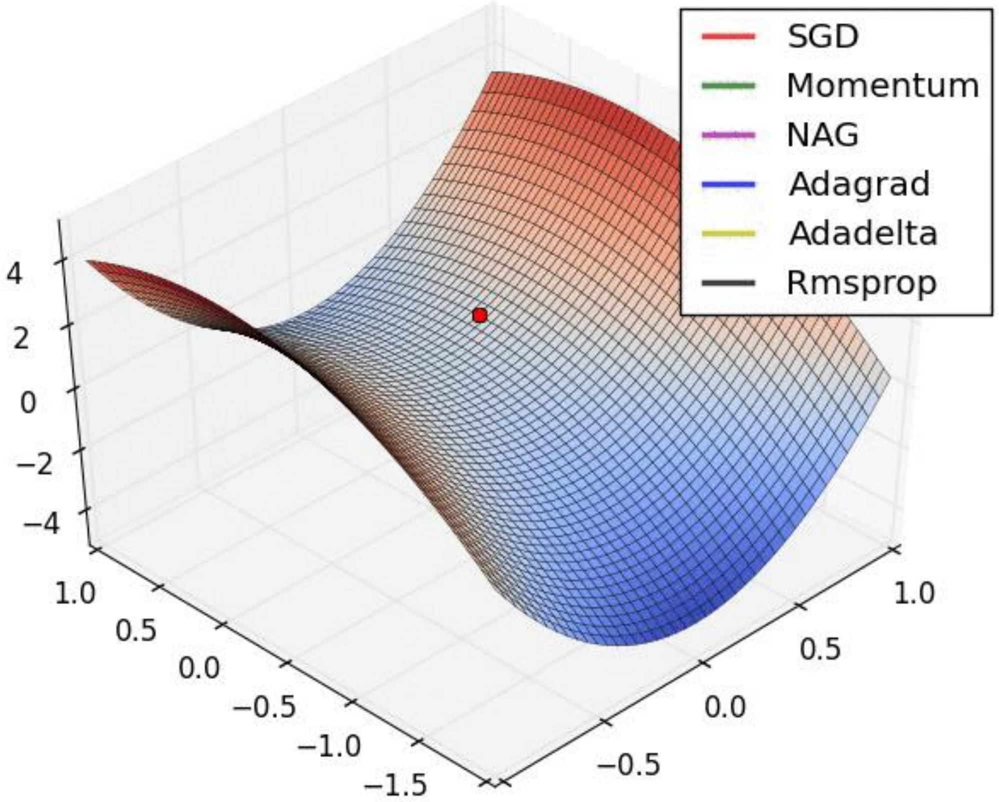


Figure 5: SGD optimization on saddle point [1]

[1] Sebastian Ruder. “An overview of gradient descent optimization Algorithms”. arXiv 2017

Optimization: Extensions of Adams

Adam is the go-to optimizer for deep learning now

- Combines two effective idea: momentum and dynamic learning rates
- Works very well in a large range of network work architectures and tasks
- Many of LLMs are pretrained using Adam or its extensions. (Almost all common ones.)

Optimization: Extensions of Adams

Adam is the go-to optimizer for deep learning now

- Combines two effective idea: momentum and dynamic learning rates
- Works very well in a large range of network work architectures and tasks
- Many of LLMs are pretrained using Adam or its extensions. (Almost all common ones.)

Notable Extensions:

- Reducing the memory footprint of momentum states:
  - AdaFactor
  - 8-Bit Adam
- Better warmup optimizer stage:
  - RAdam
- More information in dynamic learning rate:
  - AdamSAGE (Sensitivity)
  - Sophia (2^{nd} order optimizer approximation)

Outline

Optimization

- Optimization Basics
- **Numerical Types**

Parallel Training

- Data Parallelism
- Pipeline Parallelism
- Tensor Parallelism
- Combination of Combination
- ZeRO Optimizer

Numerical Types: Basic Types

Floating point formats supported by acceleration hardware

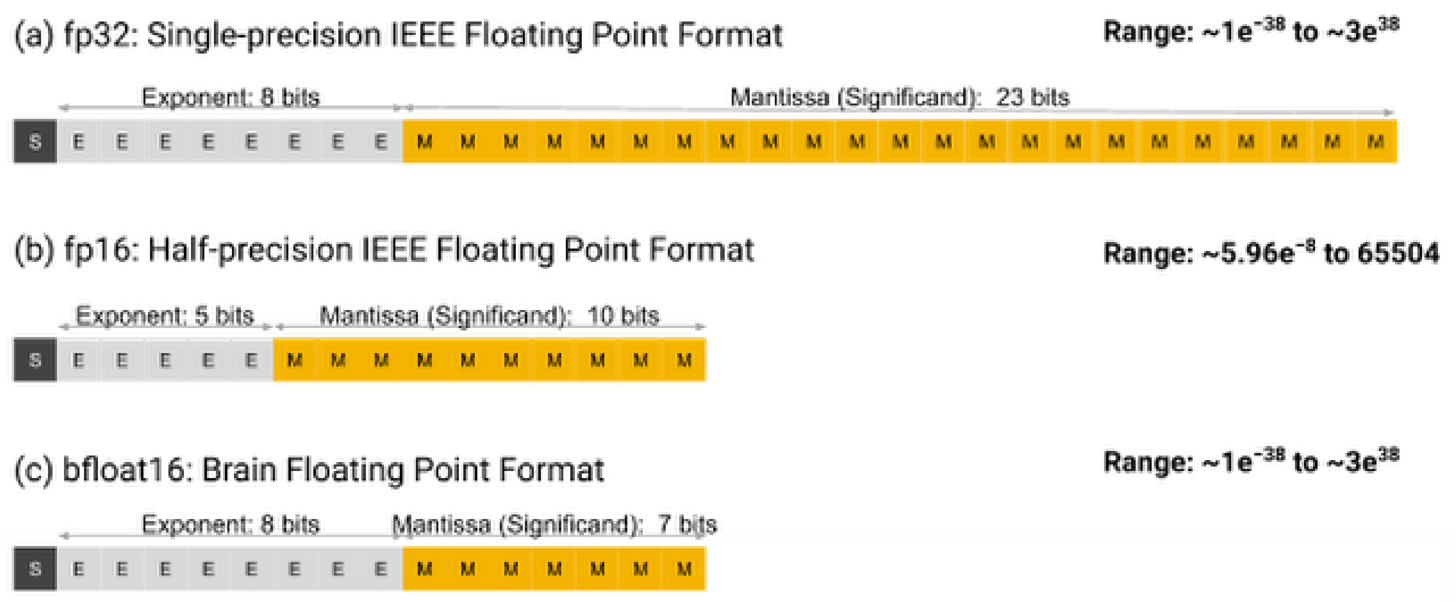


Figure 6: Floating Point Formats [5]

- BF16 is supported on TPU before LLM (2019 or earlier)
- FP32 and FP16 was the only option before A100. BF16 was not supported at hardware level
- BF16 was first supported in GPUs around 2021

Numerical Types: Neural Network Preferences

Neural networks prefer bigger range than better precision

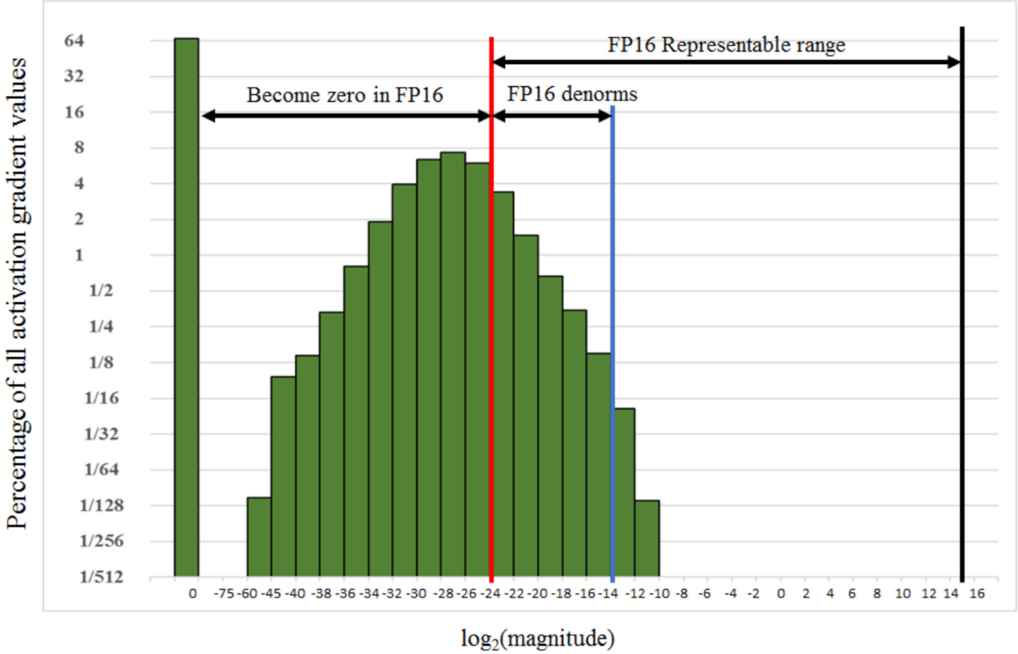


Figure 6: Histogram of gradient values in a FP32 training [6]

- Many computation needs bigger range than FP16

[6] Narang et al. "Mixed Precision Training ". ICLR 2018

Numerical Types: Mixed Precision Training

Using different numerical types at different part of the training process

- Parameters, activations, and gradients often use FP16
- Optimizer states often needs FP32

Maintaining main copies of FP32 for calculations

Dynamically scaling up loss to fit gradients etc. in FP16 range

Numerical Types: Mixed Precision Training

Using different numerical types at different part of the training process

- Parameters, activations, and gradients often use FP16
- Optimizer states often needs FP32

Maintaining main copies of FP32 for calculations

Dynamically scaling up loss to fit gradients etc. in FP16 range

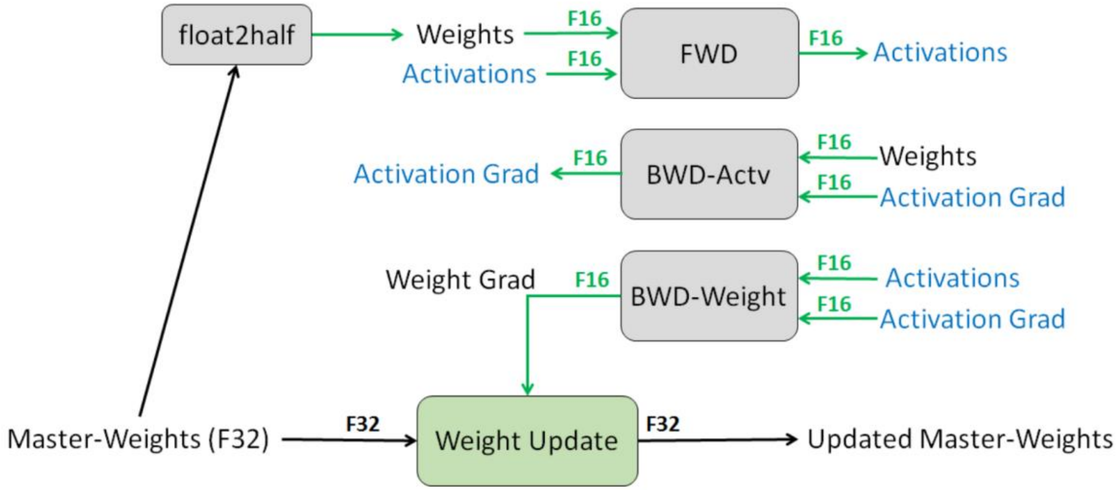


Figure 7: An Example Mixed Precision Training Set up [6]

Numerical Types: BF16

BF16 is the preferred numerical type on A100 and H100

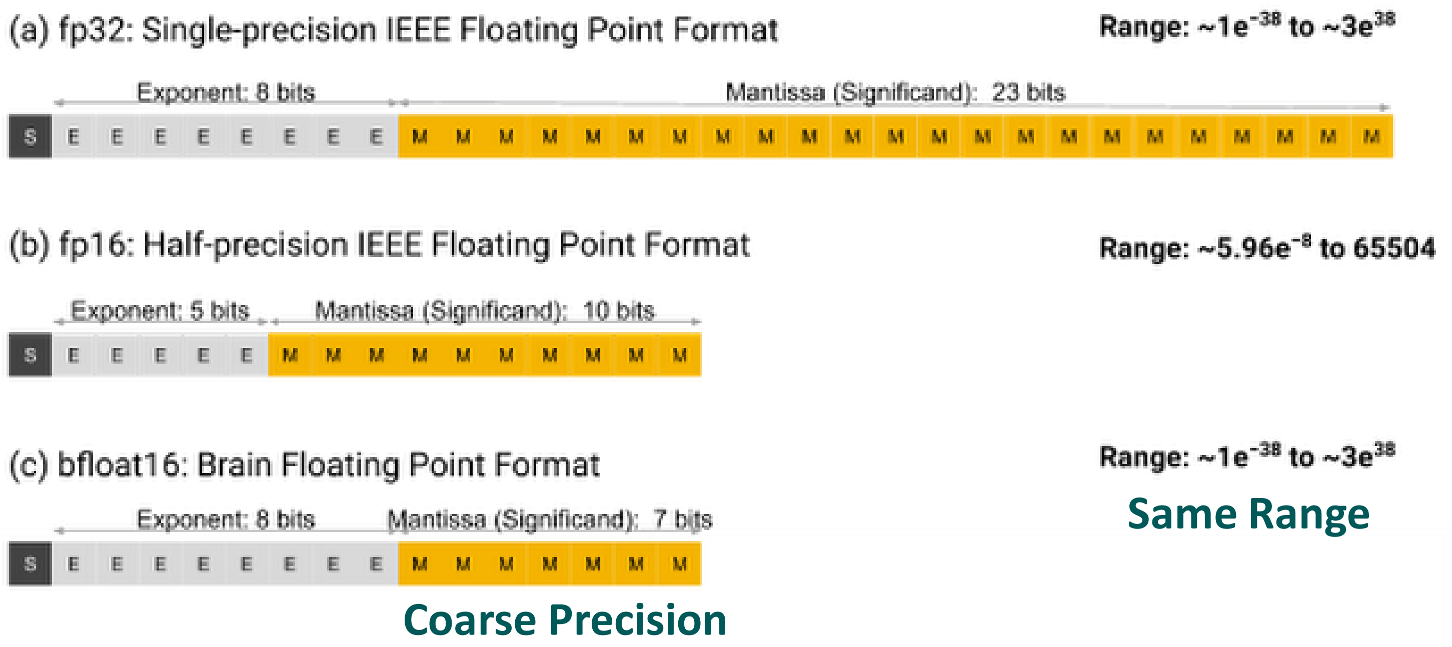


Figure 6: Floating Point Formats [5]

- Same range as FP32: eliminated the needs for mixed precision training while being way more stable
- Coarse precision: mostly fine, only a few places in neural network need more fine-grained precision

Quiz: What layers/operations in Transformers needs FP32 precisions instead of BF16?