MCTensor: A High-Precision Deep Learning Library with Multi-Component Floating-Point

Tao Yu*, Wentao Guo*, Jianan Canal Li*, Tiancheng Yuan*, Christopher De Sa
*Equal Contribution, Cornell University

Cornell Bowers C·IS
College of Computing and Information Science

High Precision Computations

Applications: dynamic systems (Taylor methods), computational geometry (delaunay triangulation), hyperbolic deep learning …

- **multiple-digit:** a sequence of digits with a single exponent (e.g., Julia BigFloat), represents compactly a much larger range of numbers
- **multiple-component (MCF)** [1]**:** an unevaluated sum of multiple ordinary floating-point numbers (e.g., float16), faster, uses existing floating-point accelerators

Multi-Component Tensor

- **Expansion:** an unevaluated sum of floats (e.g. FloatTensor)

$$x = (x_0, x_1, \cdots, x_{nc-1}) = x_0 + x_1 + \ldots + x_{nc-1}$$

- **MCTensor** Object

$$x\{fc, \text{tensor}, nc\}$$

- **MCTensor** Gradient

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial (x_0 + x_1 + \cdots + x_{nc-1})} = \frac{\partial f}{\partial x_i}$$

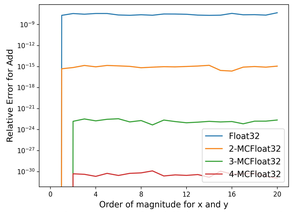
Computing with MCTensor

Operations	MCTensor with PyTorch Tensor	MCTensor with MCTensor
Add	Grow-ExpN	Add-MCN
Multiply	ScalingN	Mult-MCN
Divide	DivN	Div-MCN

Algorithm 1 Grow-ExpN

Input: nc-MCTensor x, PyTorch Tensor v
initialize $Q \leftarrow v$
for $i = 1$ to nc **do**
  $k \leftarrow nc + 1 - i$
  $(Q, h_k) \leftarrow$ **Two-Sum**(x_{k-1}, Q)
end for
$h \leftarrow (Q, h_1, \cdots, h_{nc})$
Return: Simple-Renorm(h, nc)

- **Two-Sum**(Tensor1**,** Tensor2**)** → (result, error);
- **Simple-Renorm**(h, nc**)** → moves zeros backward and outputs a MCTensor with *nc* component



Relative Error of Addition, compared to Julia BigFloat (*precision = 3000*)

Matrix level operations are implemented in a parallel manner using defined basic operations, e.g., Dot-MCN (torch.dot), MV-MCN(torch.mv), MM-MCN(torch.mm), Matmul-MCN(torch.matmul) …

Learning with MCTensor

MCTensor supports learning behaviors in the same way as PyTorch does with same interfaces:

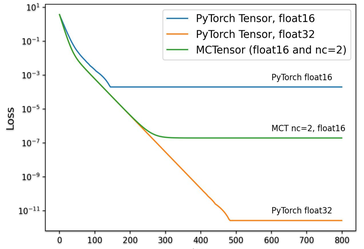
MCModule	MCLinear	MCEmbedding	MCSequential
MCActivation	MC-ReLU	MC-Softmax	MC-GELU
MCOptim	MCAdam	MCSGD	....

MCTensor can be used in the same way as PyTorch Tensor with a few lines of replacement!

```
class MCLinear(MCModule):
    def __init__(self, in_features, out_features, nc, bias=True):
        super(MCLinear, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.nc = nc
        self.weight = MCTensor(out_features, in_features, nc=nc, requires_grad=True)
        if bias:
            self.bias = MCTensor(out_features, nc=nc, requires_grad=True)
        else:
            self.bias = None

    def forward(self, input):
        return F.linear(input, self.weight, self.bias)
```

MCModel Definition Block (MCLinear as an example)



Training loss of Linear Regression on synthetic dataset

```
model = MCModel(nc=nc)
optimizer = MCSGD(mc_model.parameters())
for x, y in train_dataset:
    optimizer.zero_grad()
    y_hat = model(x)
    loss = loss_fn(y_hat, y)
    loss.backward()
    optimizer.step()
```

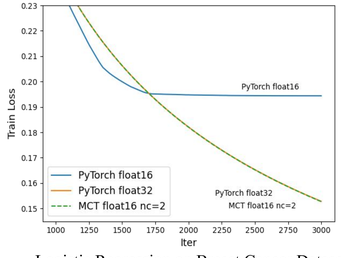
MCModel Training Paradigm

Experiments and Results

We demonstrate the increased precision of MCTensor under the same data type by carrying out experiments in two settings:

A1. high precision learning with low precision numbers, using **Logistic Regression** model on Breast Cancer Dataset (2 classes, 569 samples, 30 features)

With *nc* equals to 2, MCTensor-based model reaches a much lower training loss in Logistic Regression compared to PyTorch HalfTensors



Logistic Regression on Breast Cancer Dataset

Experiments and Results

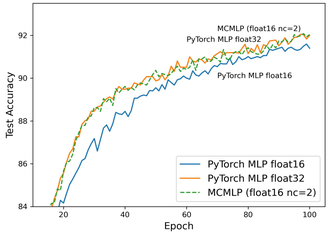
A2. high precision learning with low precision numbers, using **MCTensor-MLP** models, compared with MLP with PyTorch HalfTensors and FloatTensor

Dataset:

– Breast Cancer Dataset – Reduced MNIST (10 classes, 1000 samples/class)

MLP on Breast Cancer Dataset

Model	Training Loss	Testing accuracy
MLP Float16	0.144	90.35
MLP Float32	**0.124**	**91.23**
MLP Float64	**0.124**	**91.23**
MC-MLP (nc=1)	0.144	90.35
MC-MLP (nc=2)	**0.124**	**91.23**



Test accuracy of MLPs on Reduced MNIST

B. high precision hyperbolic embedding reconstruction task

- Following [2], we use Poincaré upper-halfspace (**Halfspace**) model to embed the transitive closure for reconstruction on the WordNet Mammals dataset with 1181 nodes and 6541 edges.
- Contrastive learning approach with loss $\mathcal{L}(\Theta) = \sum_{(\mathbf{x},\mathbf{y})\in D} \log \frac{e^{-d_u(\mathbf{x},\mathbf{y})}}{\sum_{\mathbf{y}'\in N(\mathbf{x})} e^{-d_u(\mathbf{x},\mathbf{y}')}}$

where the hyperbolic distance function is $d_u(\mathbf{x},\mathbf{y}) = \operatorname{arcosh}\left(1 + \frac{||\mathbf{x}-\mathbf{y}||^2}{2x_n y_n}\right)$

Performance of Hyperbolic Models

Model	MAP (mean ± sd)	MR (mean ± sd)
Halfspace (f32)	91.91% ± 0.64%	1.399 ± 0.04
Halfspace (f64)	92.79% ± 0.41%	1.340 ± 0.07
MC-Halfspace (f64 nc=1)	93.02% ± 0.40%	1.296 ± 0.02
MC-Halfspace (f64 nc=2)	92.77% ± 0.28%	1.304 ± 0.02
MC-Halfspace (f64 nc=3)	**93.31**% ± 0.75%	**1.282** ± 0.03

References:
[1] Priest, D. M. Algorithms for arbitrary precision floating point arithmetic. University of California, Berkeley, 1991
[2] Yu, Tao, and Christopher M. De Sa. Representing Hyperbolic Space Accurately using Multi-Component Floats. NeurIPS, 2021