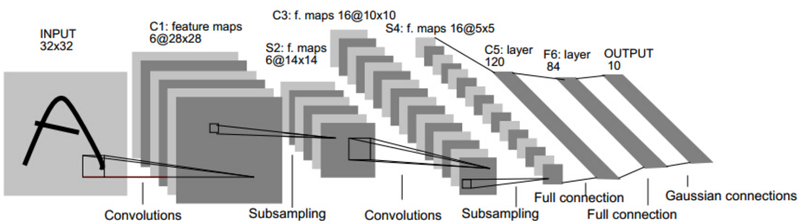
Parallel Parameter Update for Deep Neural Network



CSE633 Parallel Algorithm (Spring 2018)

Instructor: Prof. **Russ Miller**

Presenter: **Xin Ma**

xma24@buffalo.edu

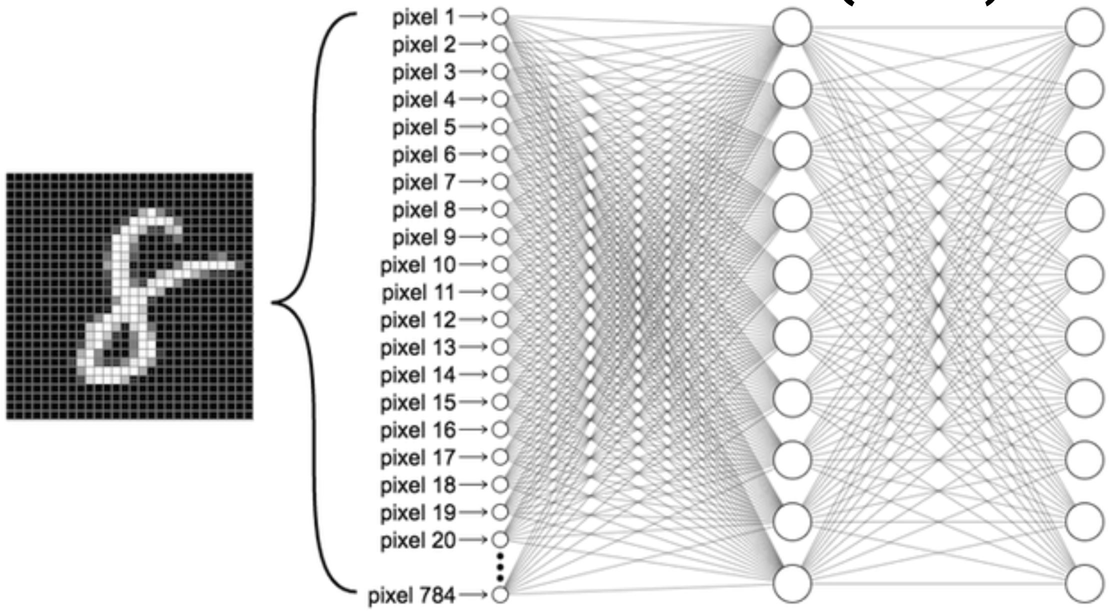
05/03/2018

Outline

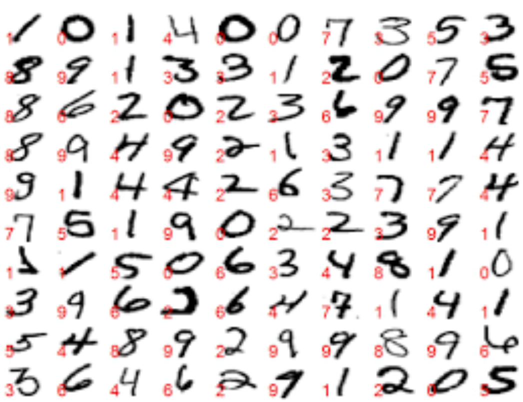
- Background
- Parallel Parameter Update
- Evaluation
- Conclusions

Background

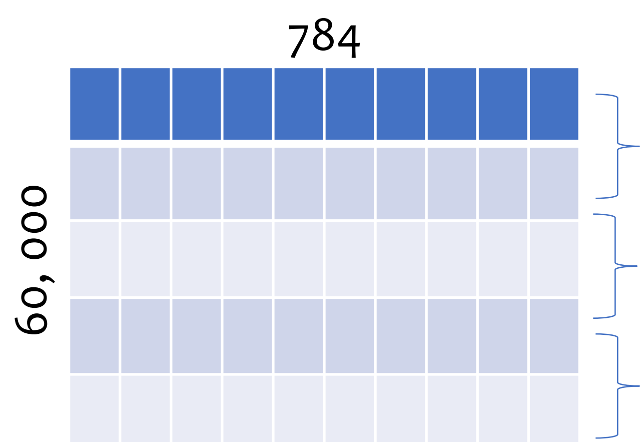
Artificial Neural Network (ANN) model



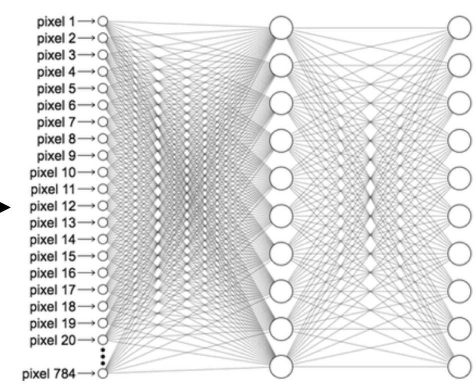
MNIST data set



60,000
training samples

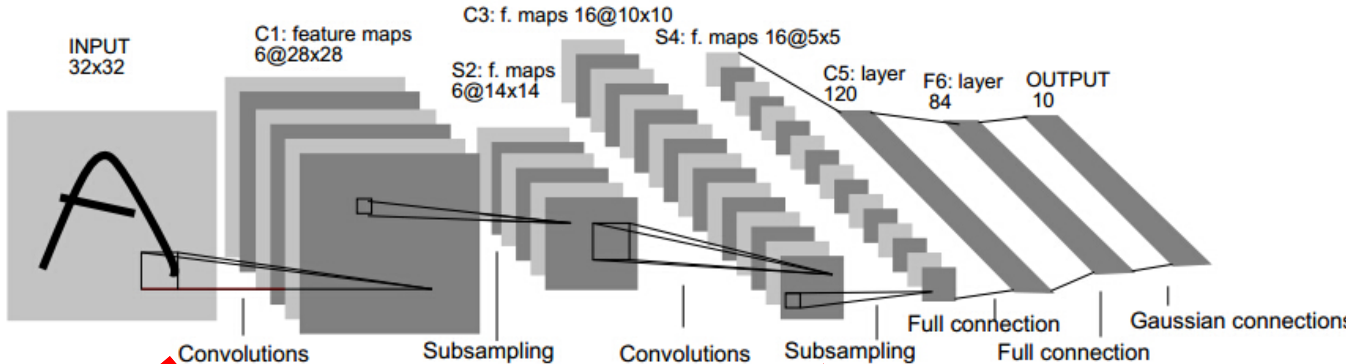


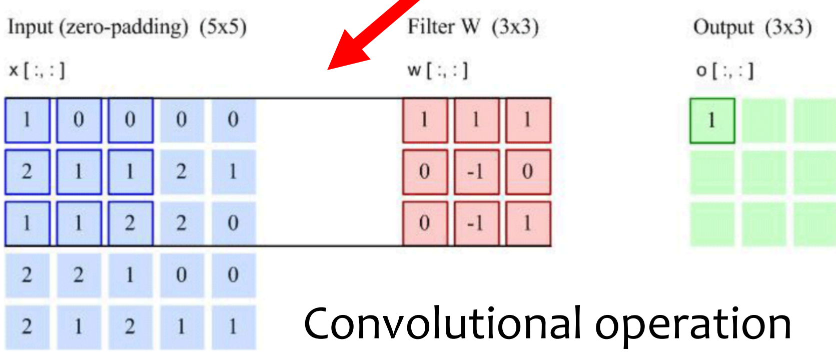
13.3 MB



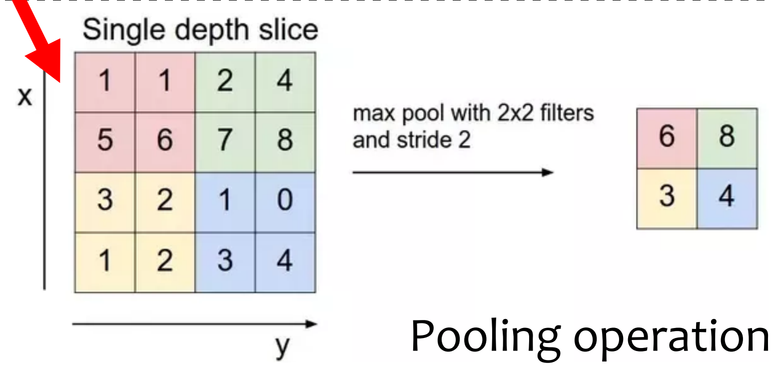
Background

Convolutional Neural Network (CNN) model

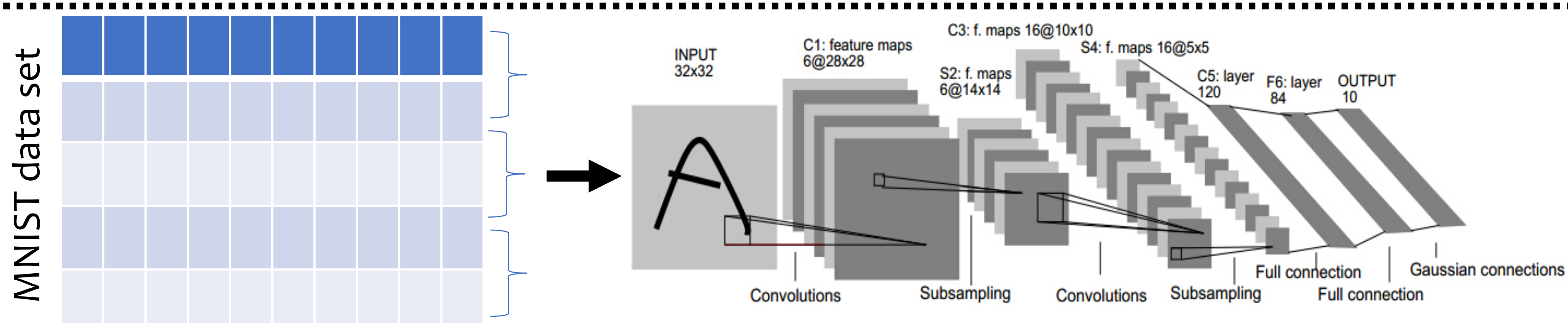




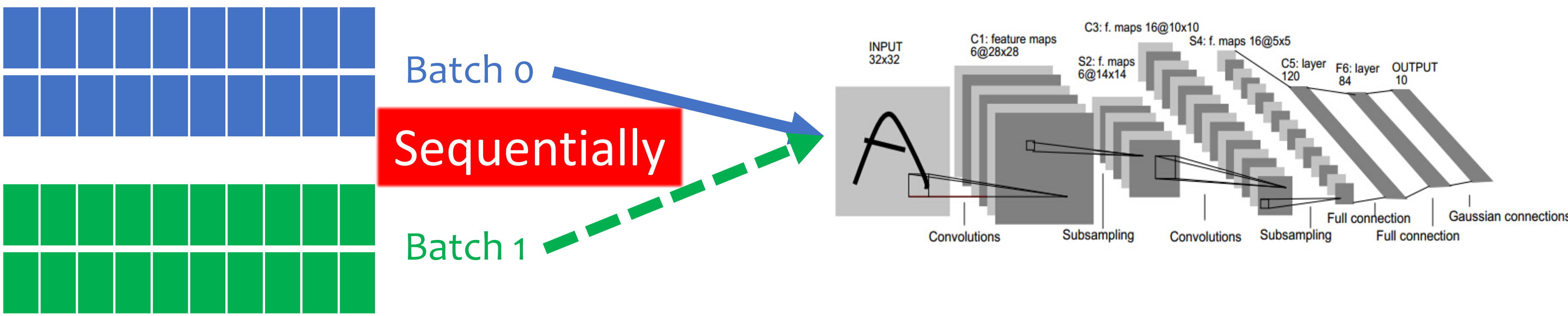
Convolutional operation



Pooling operation



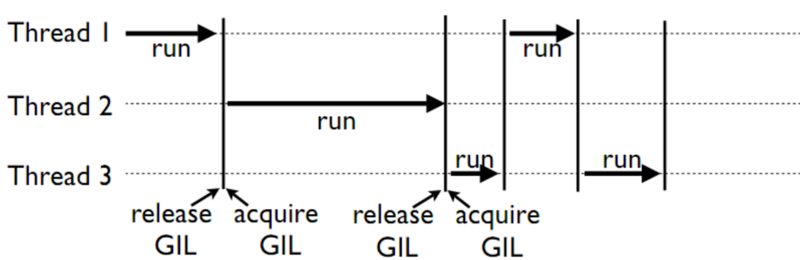
Parallel Parameter Update



How to improve it?

Is it possible to run the program in a parallel way?

Method 1: Run the python program using multiple threads at the same time

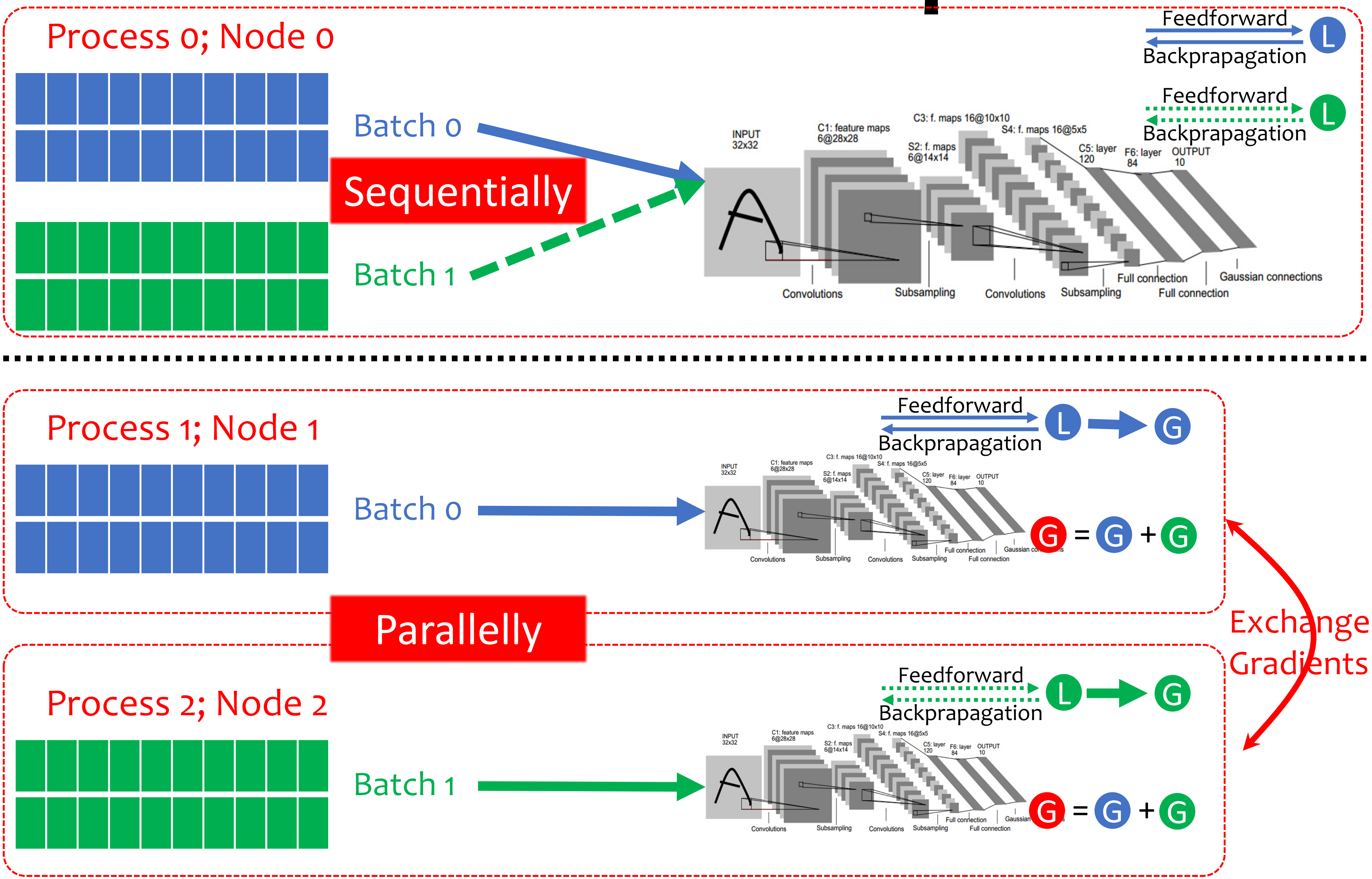


GIL - Global Interpreter Lock

Method 2: Run the python program with different processes (Nodes)

- Assume each node with one available process
- Using massage passing to cooperative those python programs

Parallel Parameter Update



Experiment Setup

Settings:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \frac{1}{2}\lambda||\mathbf{w}||_2^2$$

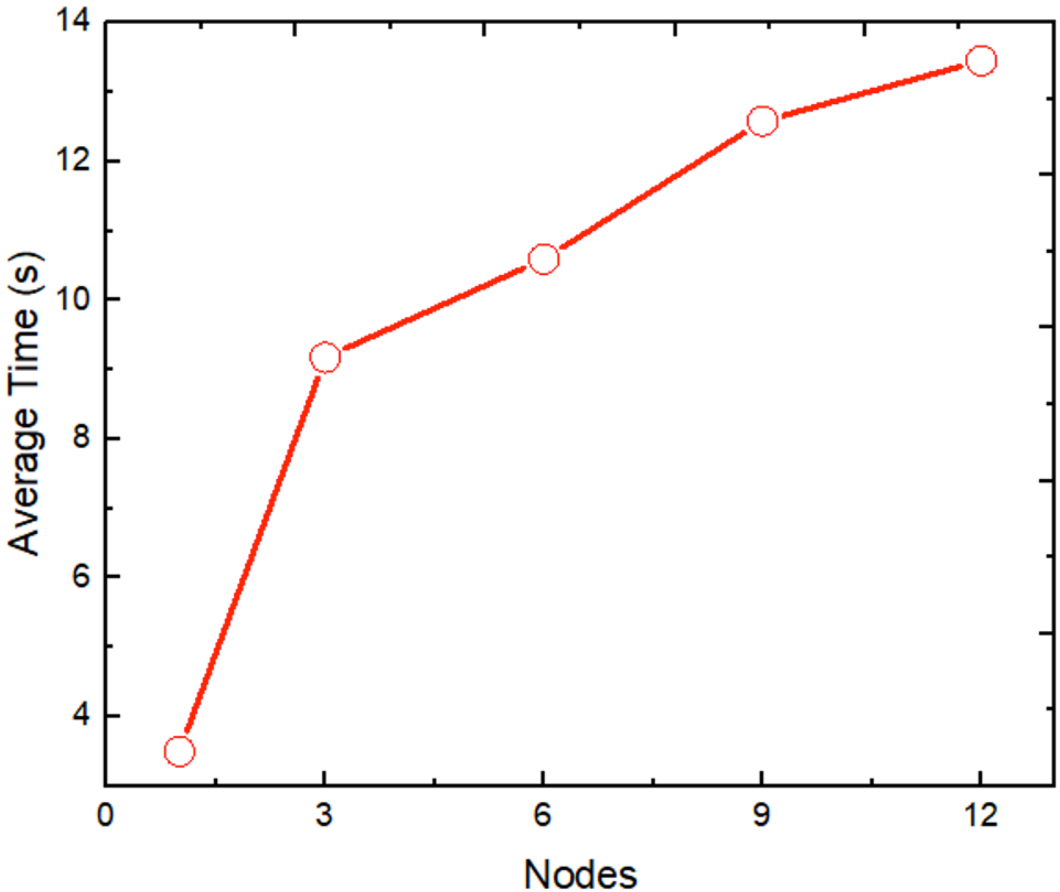
- ANN model:
  - hidden nodes: [10,50]; regularization parameter: 5; # of hidden layer: 1; # of params: 795.2*#_hidden.
- CNN model:

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  )
  (out): Linear(in_features=1568, out_features=10, bias=True)
)
```

  - # of parameters: 28938.
- MPI:
  - Scatter() and Allgather().

Experiment Results

Average run time of each iteration with different # of nodes:

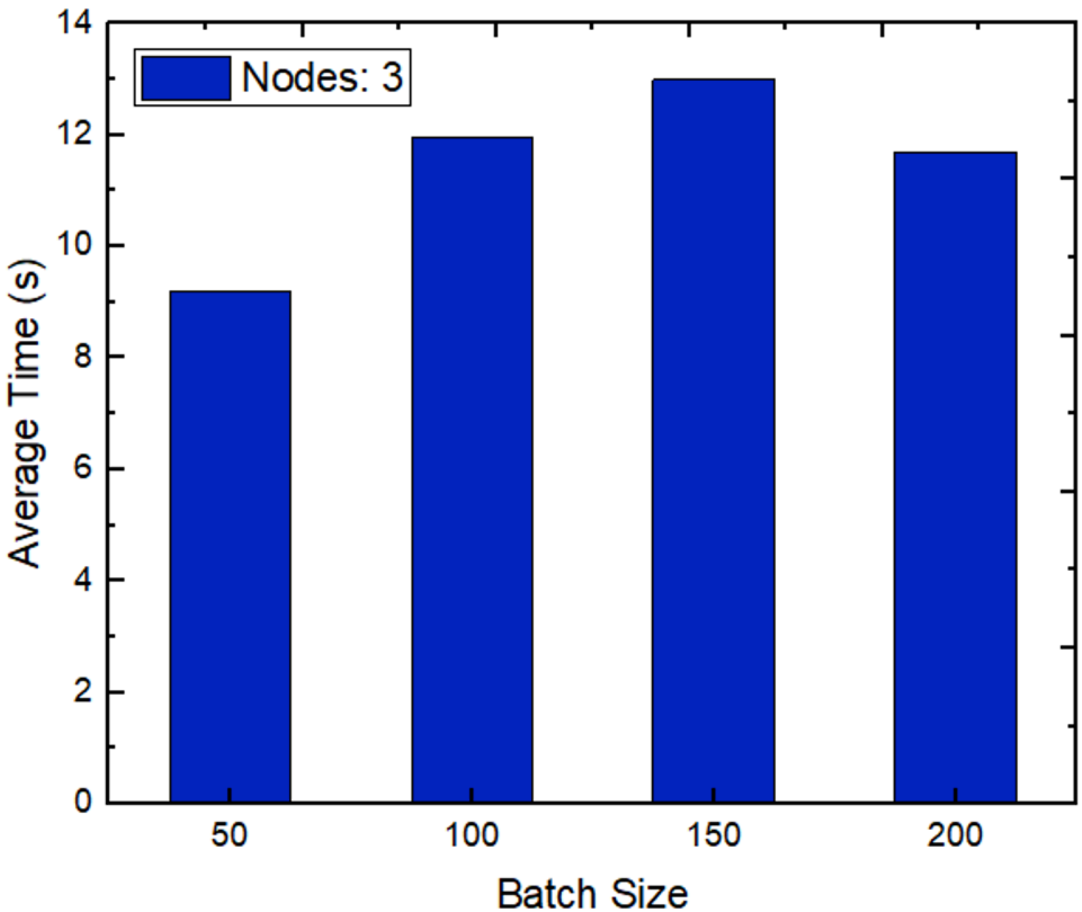


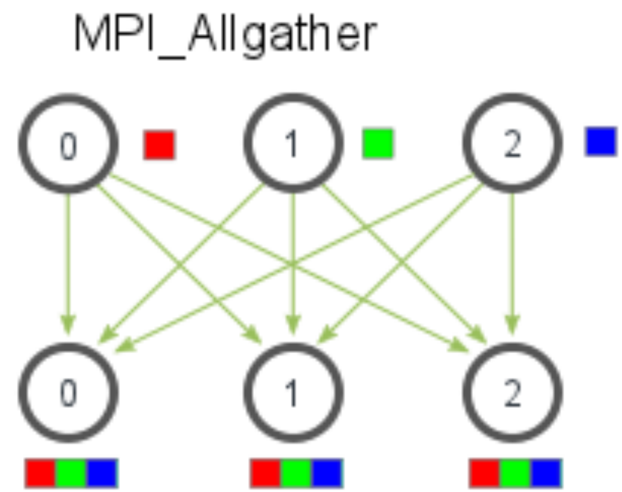
More parallel nodes ---> larger data exchange overhead

Node number	1	3	6	9	12
Average Time (s)	3.50104	9.1768	10.5955	12.57973	13.45111

Experiment Results

Average run time of each iteration with different batch size:



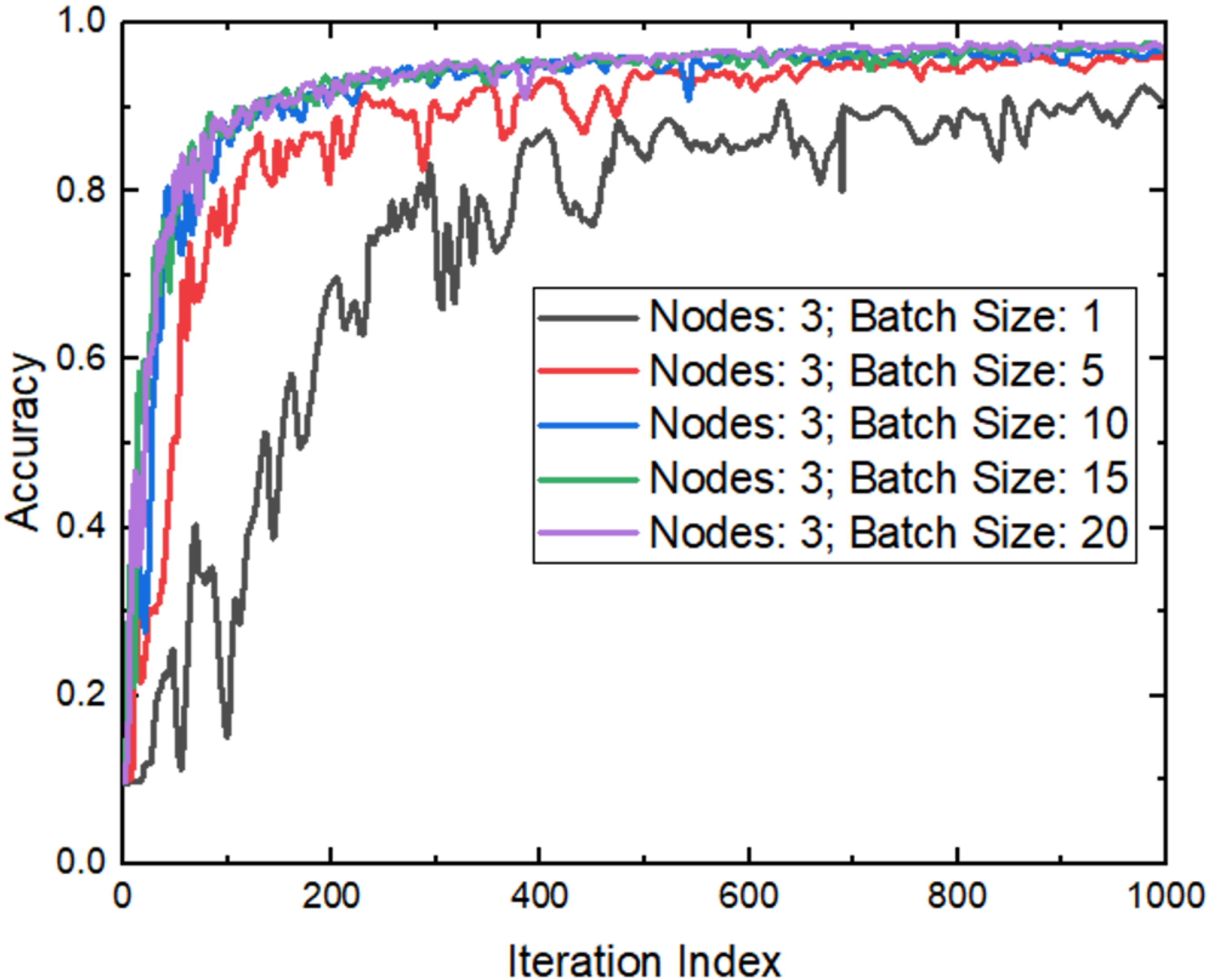


- **Parameters -> Same**
- **One-hop connection**

Batch Size	50	100	150	200
Average Time (s)	9.1768	11.94646	12.99369	11.68846

Experiment Results

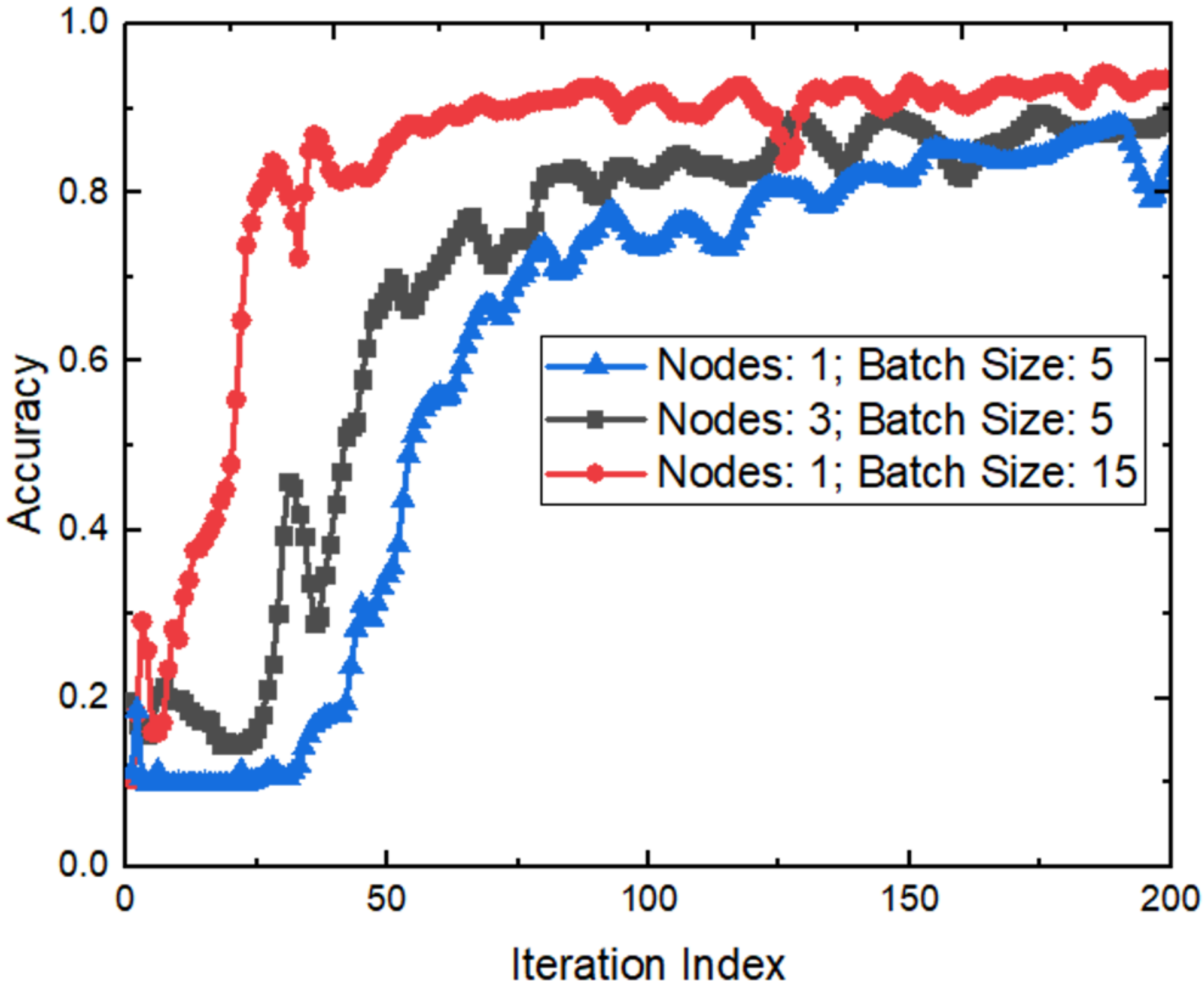
Recognition accuracy with different batch size:



Large batch size ---> good for model convergence

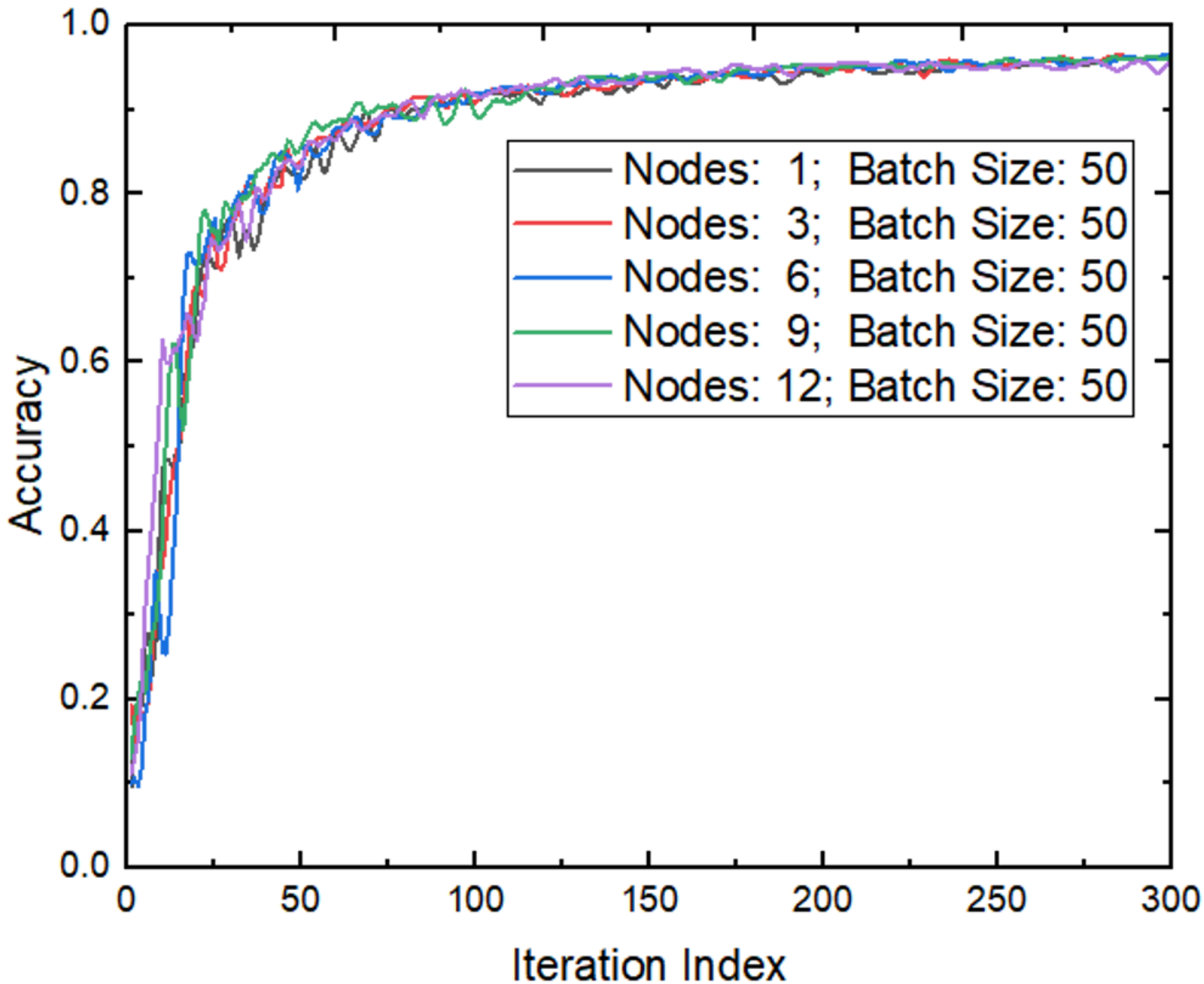
Experiment Results

Recognition accuracy with different batch size:



Experiment Results

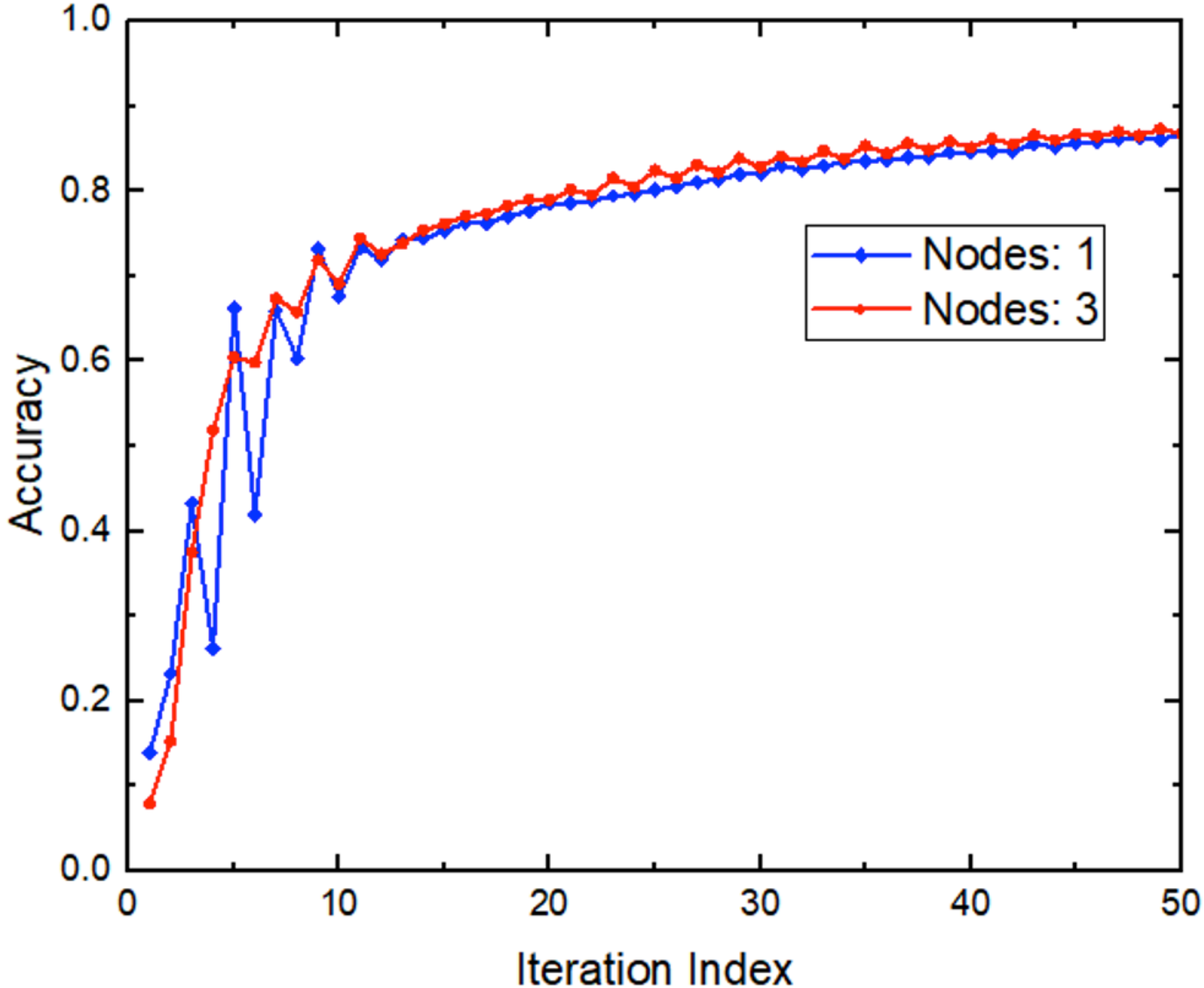
Recognition accuracy with different # of nodes:



If Batch Size is large enough, More parallel nodes ---> may not accelerate the convergence

Experiment Results

Recognition accuracy with different # of nodes:



More parallel nodes ---> good for SGD (Model is easy to train)

Conclusions

- A larger batch size is good for the training of the model (related to convergence).
- When the batch size is large enough, increasing the number of parallel nodes has not had that many benefits.
- When the batch size is quite small, increasing the number of parallel nodes will do good to the training process to some degree. However, this will also increase the overhead of the system.
- Using more parallel nodes can smooth SGD algorithm.

Thanks!

Questions?