# WICWIU: 인수인계 자료

날짜: 20190701
작성자: 박천명

# Agenda

- Introduction
- Components of WICWIU
- cuda & cuDNN
- Example
- Suggestion for RNN
- Future Works

# Introduction

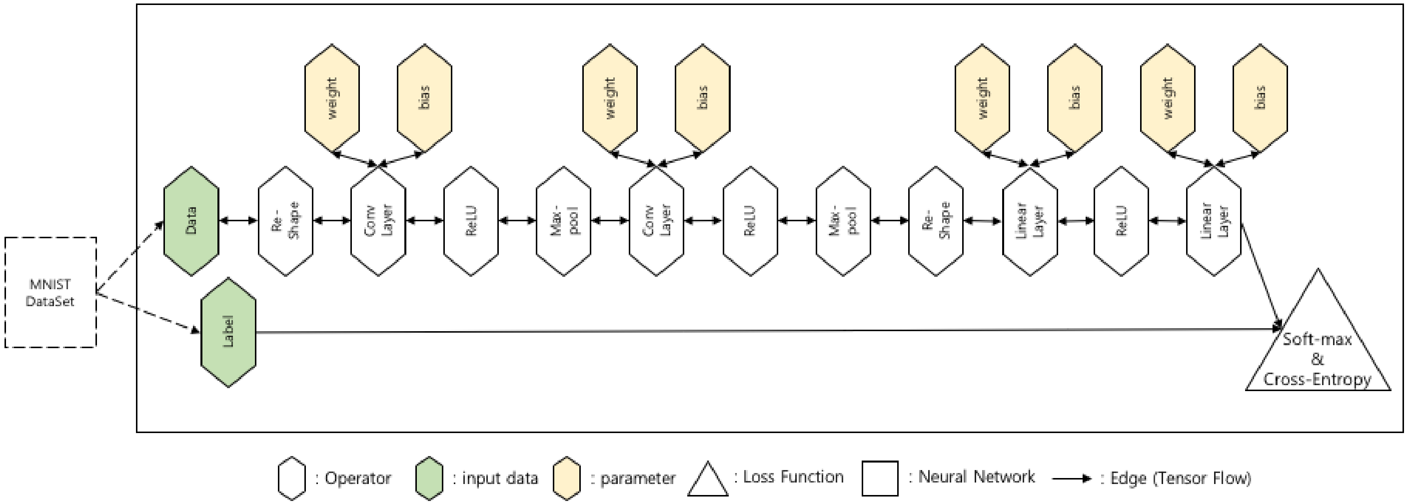
- WICWIU: 국내 대학 최초의 딥러닝 오픈소스 프레임워크
  - 한동대 딥러닝 연구실에서 연속성을 가지고 개발
  - WICWIU = What I Create is What I Understand.
    - "What I cannot create, I do not understand" [Richard Feynman]에서 영감

| | |
|---|---|
| **가독성** | 딥러닝 초보자들을 위한 가독성 높은 코드 (주요 딥러닝 알고리즘) |
| **확장성** | 잘 정의된 API를 통해 자신만의 Operator, Layer 추가 용이 |
| **고성능** | GPU 기반 대규모 병렬처리 (cuDNN) |
| **일반성** | Linear 구조 뿐 아니라 일반적인 Graph 구조의 신경망 지원 |
| **편리성** | Auto-differenciation 지원 (Gradient 계산 자동화) |

# WICWIU Design Overview

- General computational graph
  - Combine building blocks to build any computational graph
  - Auto-differentiation for easy training
    - Operators/Layers automatically compute gradients
- Consistency / simplicity

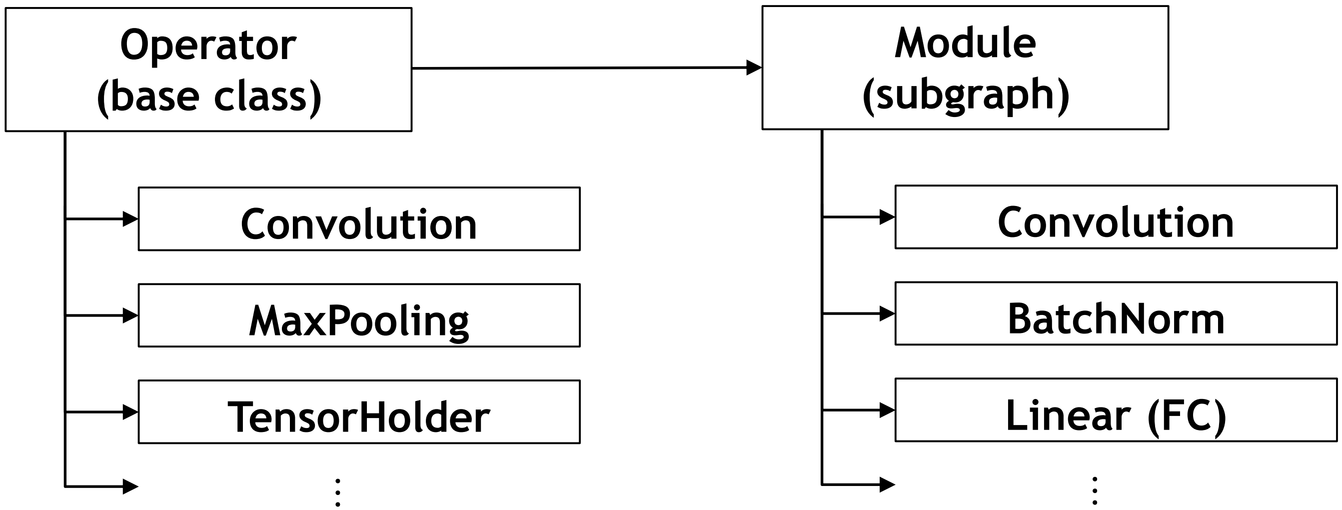
  Ex) All Operators/Layers take input as Operators



# WICWIU Design Overview

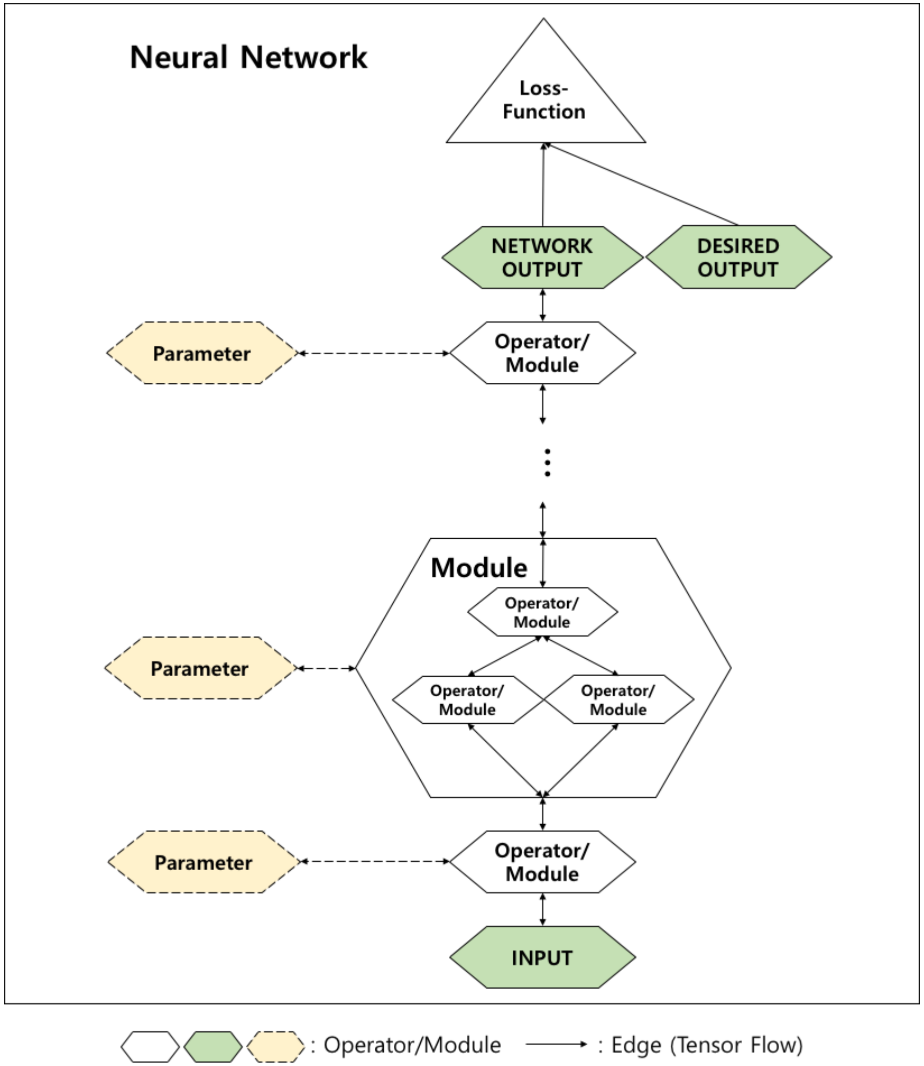
- Data classes
  - Tensor to represent both data(input/hidden) and parameters

- Well-defined Operator / Module class hierarchy
  - All Operators take operators as Input and Output



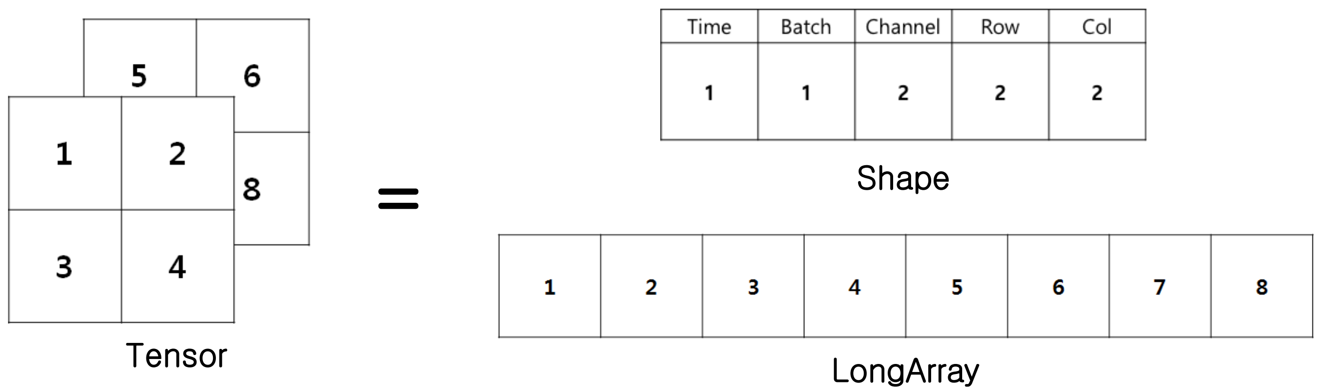
# Components of WICWIU

- Data
  - Tensor
    - Shape
    - Long Array
- Model building block
  - Operators (nodes)
  - Modules (subgraphs)
- Learning components
  - Loss Functions
  - Optimizers
- Computational graph
  - Neural Network



# WICWIU Components: Tensor, etc.

- Tensor class: multi-dimensional Tensor
  - 1D ~ 5D: Time, Batch, Channel, Row, Col
  - Memory synchronization between host and device memory
- Shape class: shape of multi-dimensional Tensors
  - Rank, length of each dimension
- LongArray class – long 1D data array (old name: Data class)
  - Internally, composed of block



# Tensor and Shape classes

- Tensor: class to represent multi-dimensional tensor
  - Shape *m_aShape; // for tensor shape
  - LongArray<DTYPE> *m_aLongArray; // for data
  - Device m_Device; // CPU or GPU

- Shape: class to represent tensor shape
  - int m_Rank; // can be 1D ~ 5D
  - int *m_aDim; // size of each dim.

Ex) m_rank = 5

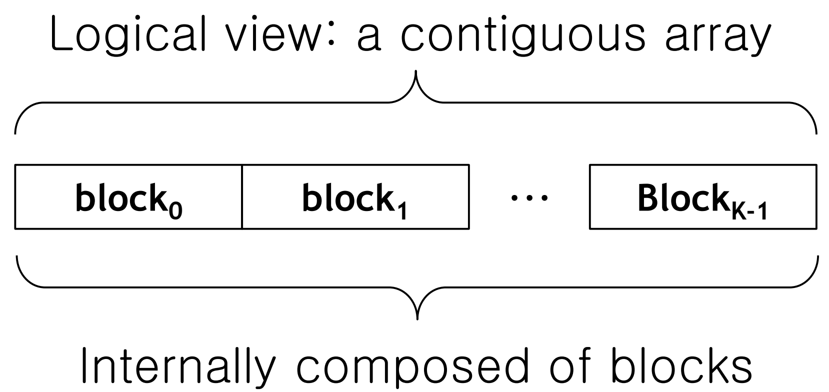
m_aDim = {1, 1, 2, 2, 2}

| Time | Batch | Channel | Row | Col |
| --- | --- | --- | --- | --- |
| 1 | 1 | 2 | 2 | 2 |

Shape

# LongArray class

- LongArray: very long 1D array
  - Logically – a contiguous array
  - Physically – divided into blocks (correspond to planes)
    - Each block contains a batch at each time (maximum 4D)

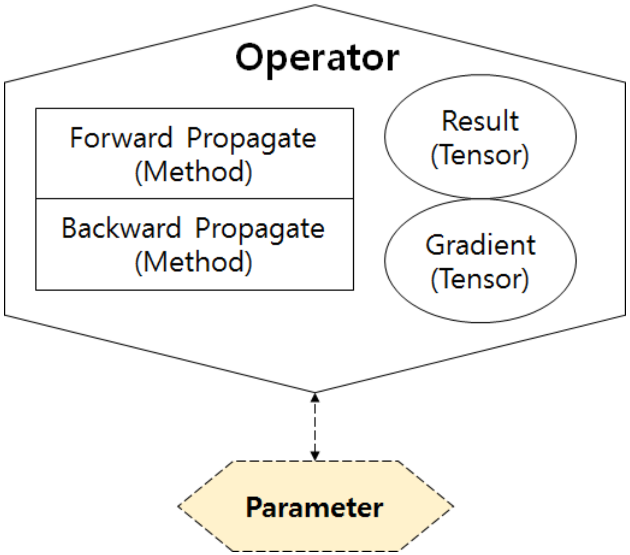



  - Data Transfer between host and device memory
    - int AllocOnGPU(), void DeleteOnGPU();
    - int MemcpyCPU2GPU(), int MemcpyGPU2CPU();

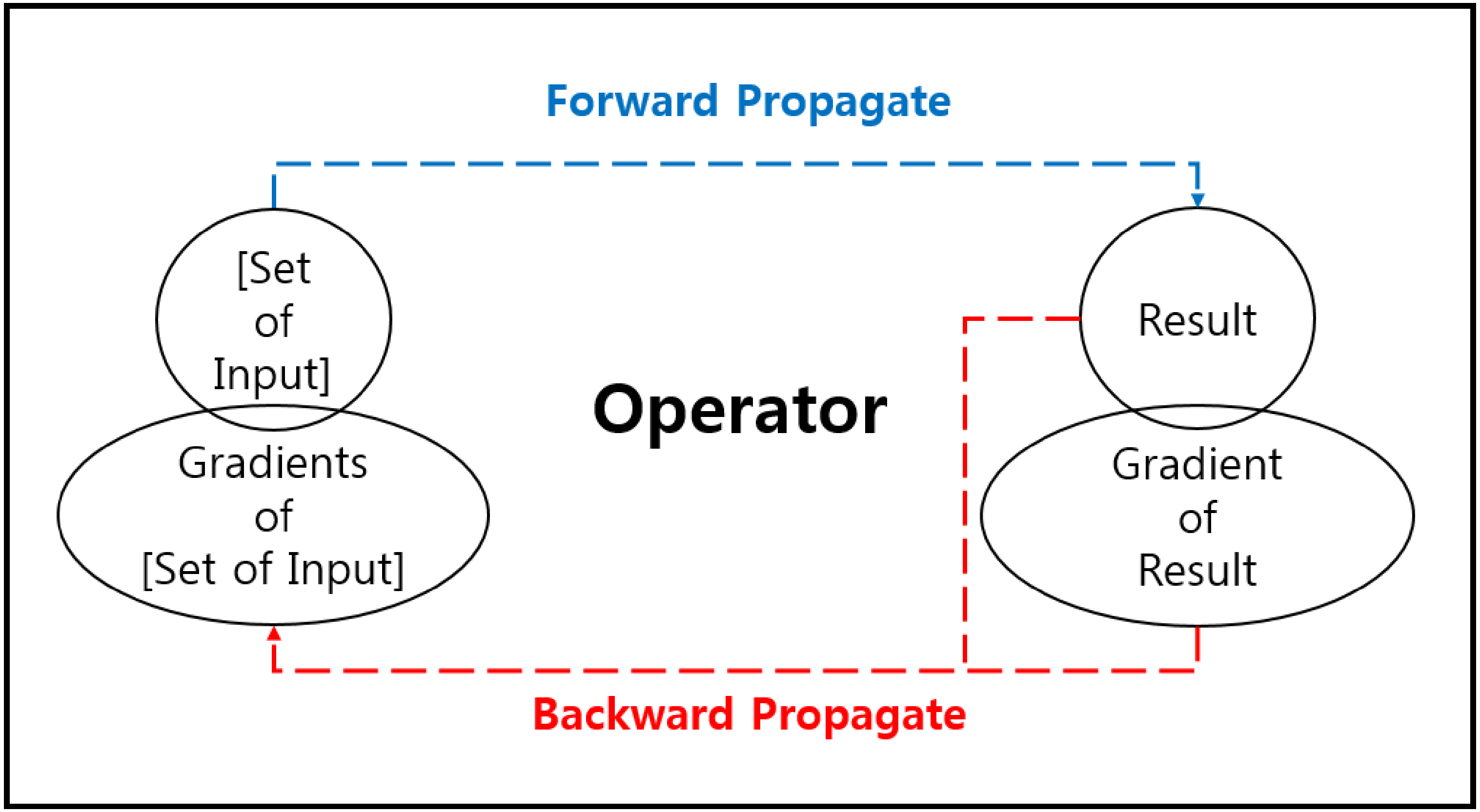
# WICWIU Components: Operators

- Operators classes: low-level operators
  - Common interface of Operators
    - Result (activation), Gradient
    - ForwardPropagation(), BackwardPropagation()
  - Connected to parameters as separate node

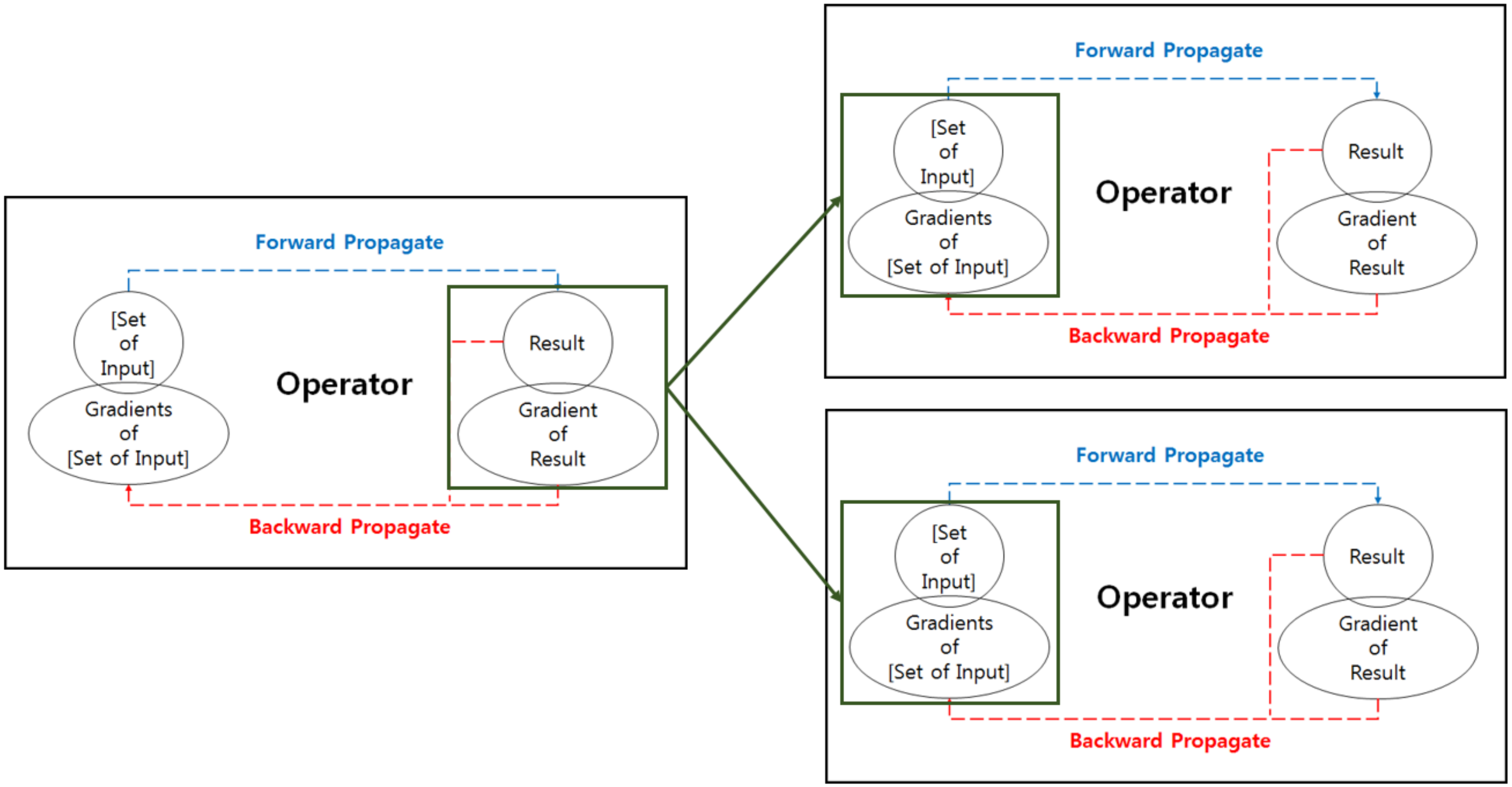
- Built-in Operators (Subclasses)
  - TensorHolder
  - Add
  - MatMul
  - ReLU
  - Sigmoid
  - Tanh
  - Convolution
  - Max-pooling
  - Average-pooling
  - Batch-Normalization



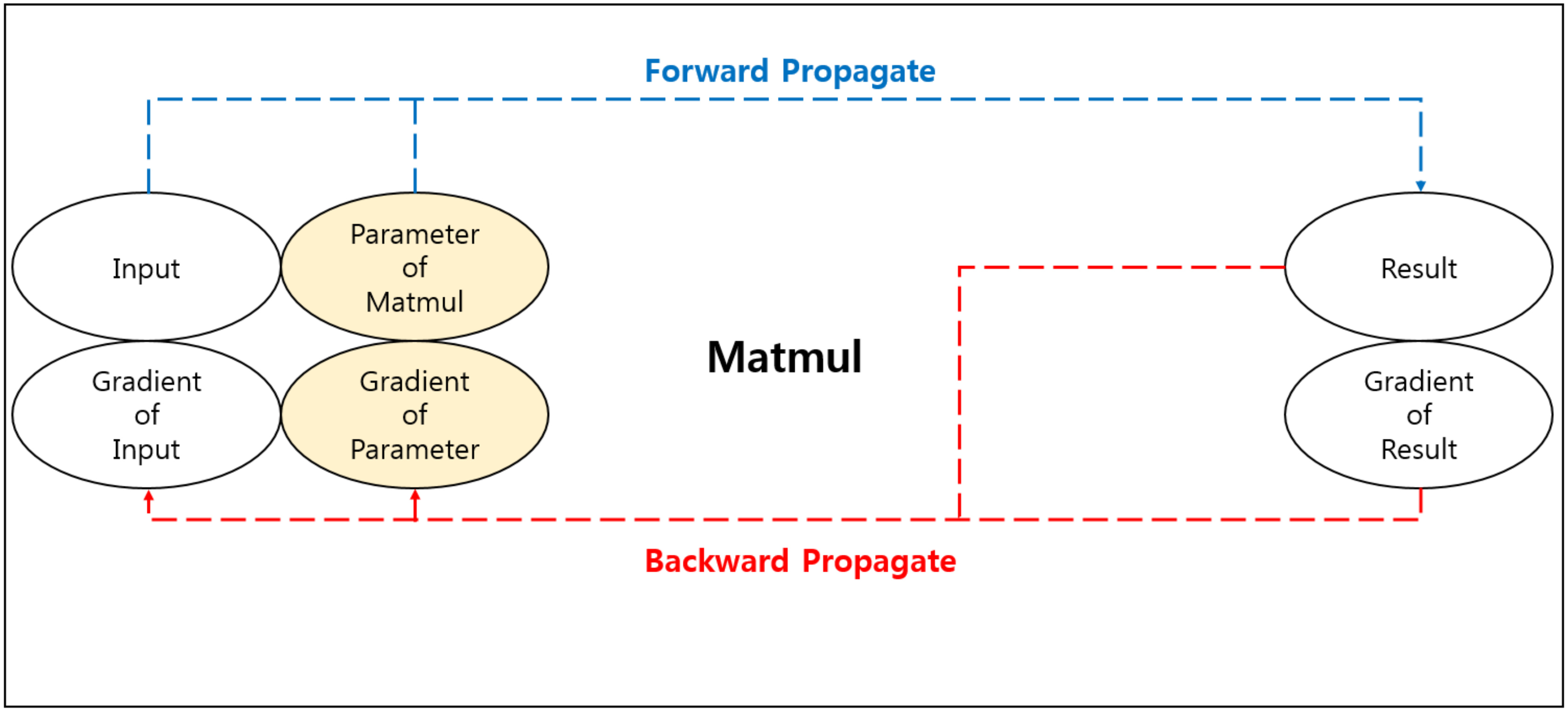
# WICWIU Components: Operators



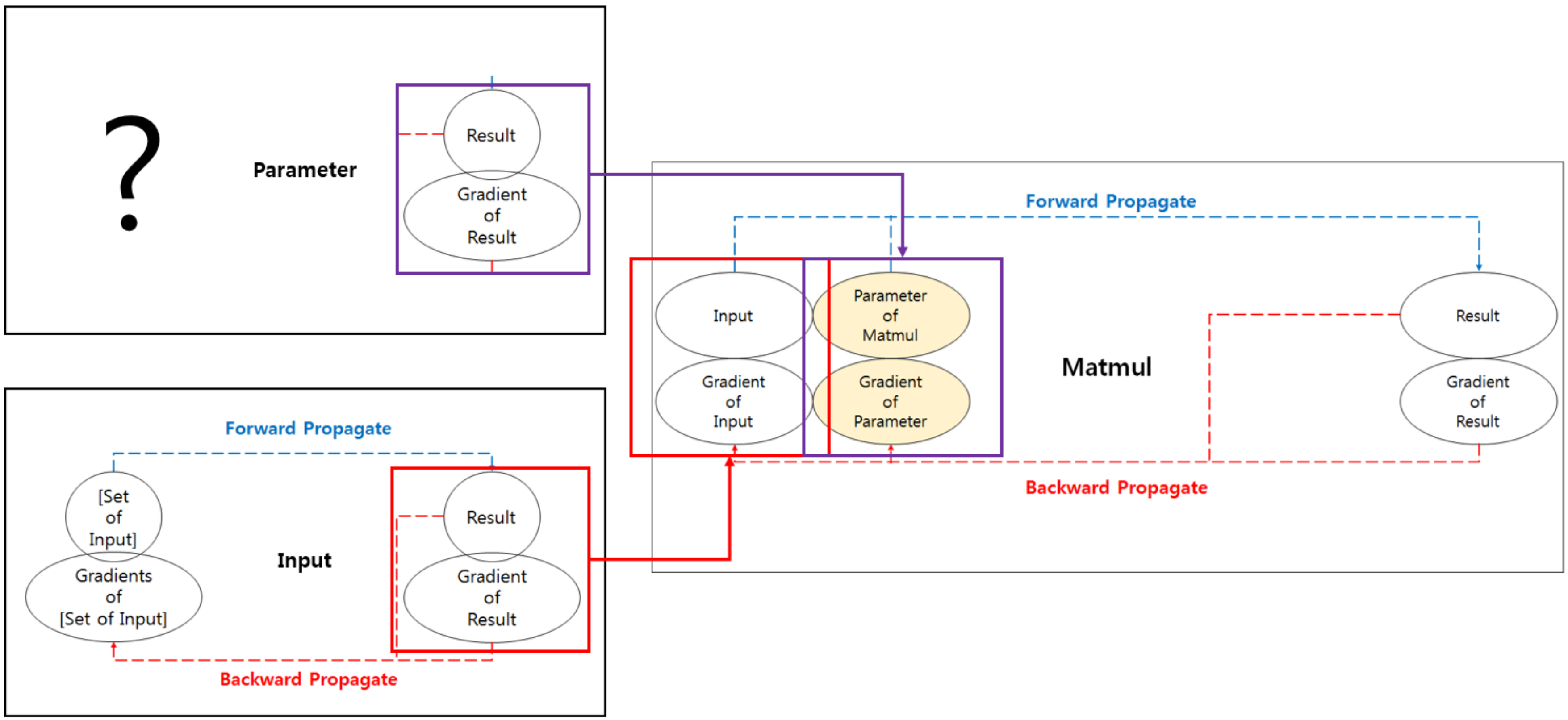
# WICWIU Components: Operators



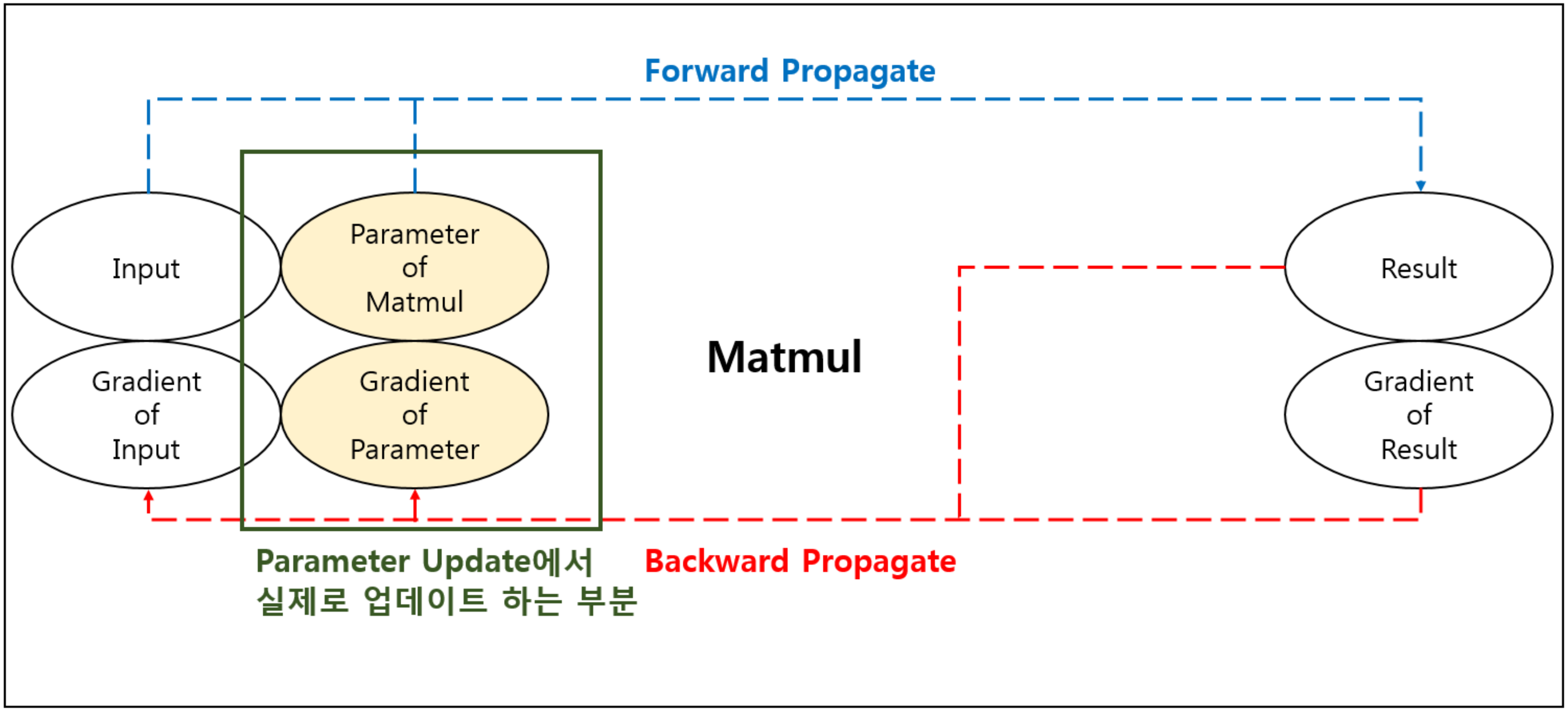
# WICWIU Components: Operators



# WICWIU Components: Operators



# WICWIU Components: Operator



# Operator class

- Operator: base class of all Operators
  - Major fields
    - Container<Operator<DTYPE> *> *m_apOutput;
    - Container<Operator<DTYPE> *> *m_apInput;
    - Container<Tensor<DTYPE> *> *m_aaResult;
    - Container<Tensor<DTYPE> *> *m_aaGradient;
  - Propagation functions
    - virtual int ForwardPropagate(int pTime = 0, int pThreadNum = 0);
      - Can run on multiple threads
    - virtual int ForwardPropagateOnGPU(int pTime = 0);
    - virtual int BackPropagate(int pTime = 0, int pThreadNum = 0);
    - virtual int BackPropagateOnGPU(int pTime = 0);

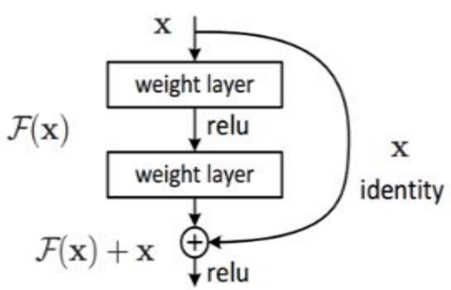
# Operator class

- Operator: base class of all low-level operators
  - Other fields
    - std::string m_name;
    - Device m_Device; // CPU or GPU
    - Int m_idOfDevice; // -1 = CPU, 0~n = ID of GPU Device
    - Mode m_Mode; // TRAINING, ACCUMULTING, INFERENCING
    - int m_numOfThread;
    - int m_isParameter;
    - Int m_isTrainable;

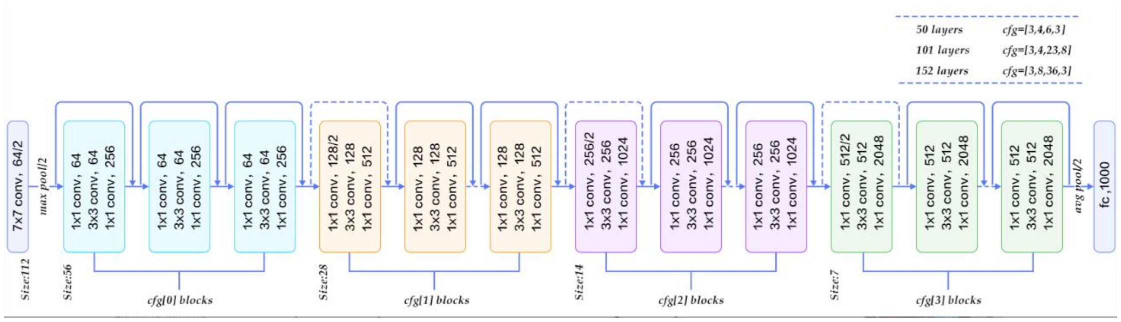
# WICWIU Components: Module

- 필요성
  - 반복되는 구조가 존재하는 경우, 복잡한 그래프를 좀 더 간단하게 구현하는 것이 가능

- 예시 (Resnet)



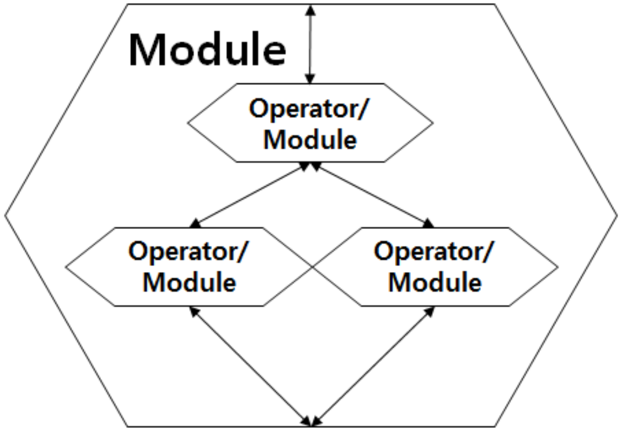

<Resnet block>



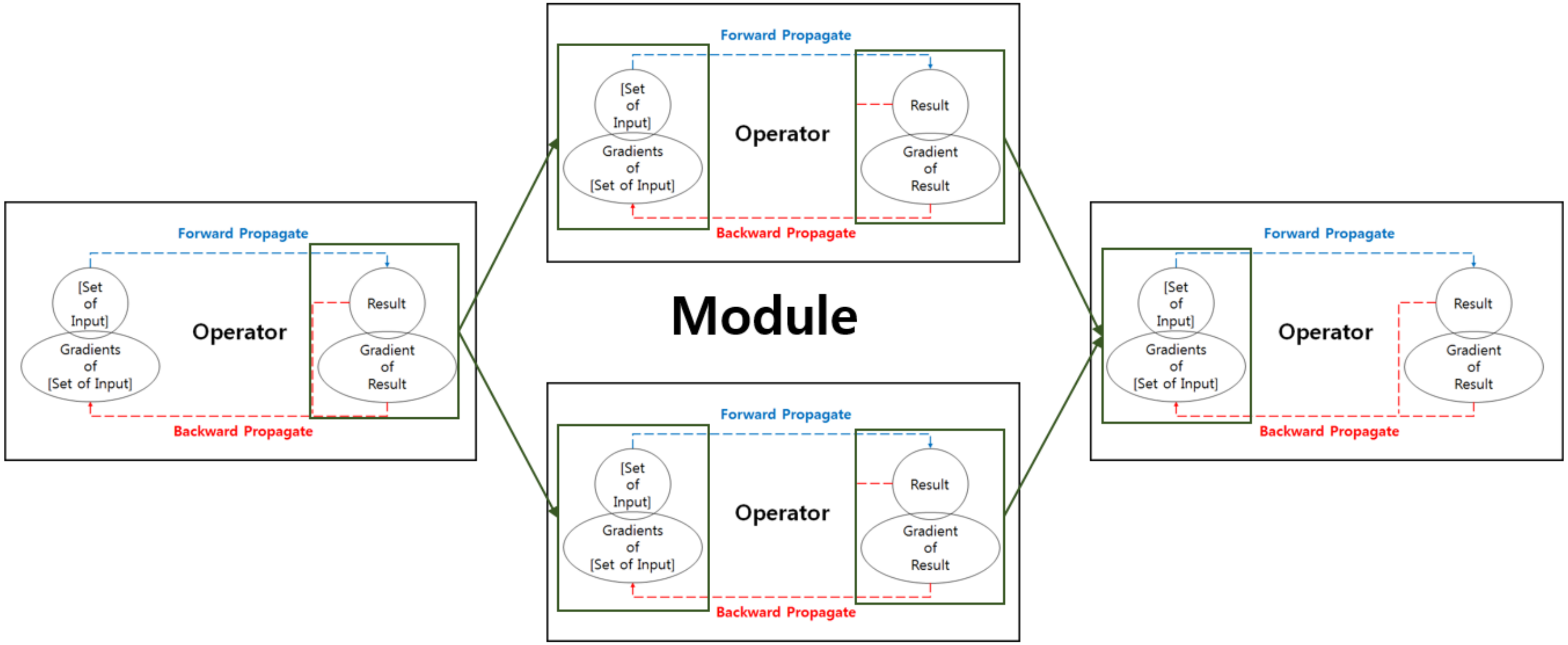

<Resnet>

# WICWIU Components: Module

- Module classes: high-level composite operations
  - Combine operators to build neural network layer or modules
  - Common interface for Module
    - Combines operators to build subgraph
    - Forward / backward propagation
    - Recursive structure (can contain another Layer)
- Built-in Layers (Module) (Subclasses)
  - Convolution Layer
  - Batch-Normalization Layer
  - Linear(fully-connected) Layer
- Example Modules
  - Resnet Basic Block
  - Densenet Basic Block (under construction)



# WICWIU Components: Module



# Module class

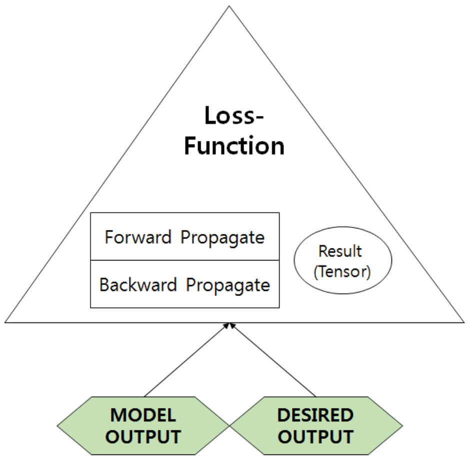
- Module: base class of high-level layers
  - Subgraph composed of Operators
  - Supports nested Module
    - Module inherits Operator (Module can contain another Layer)
  - Easy to extend to define custom Layer classes

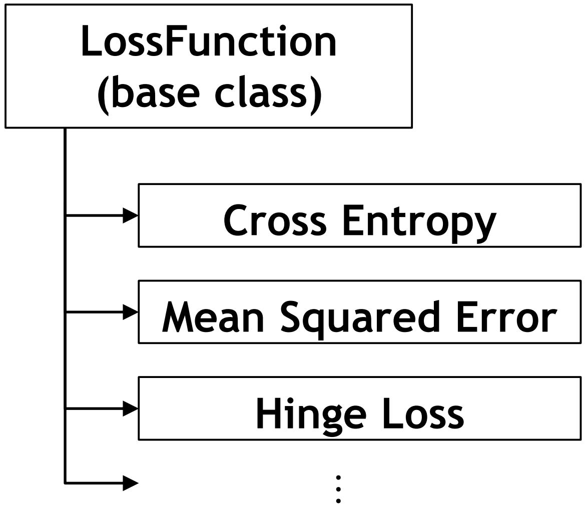
  - Major fields
    - Container<Operator<DTYPE> *> *m_aaExcutableOperator;
    - int m_numOfExcutableOperator;
    - Operator<DTYPE> *m_pLastOperator;

  - Major operations
    - int ForwardPropagate(int pTime = 0, int pThreadNum = 0);
    - int  ForwardPropagateOnGPU(int pTime = 0);
    - int BackPropagate(int pTime = 0, int pThreadNum = 0);
    - int  BackPropagateOnGPU(int pTime = 0);

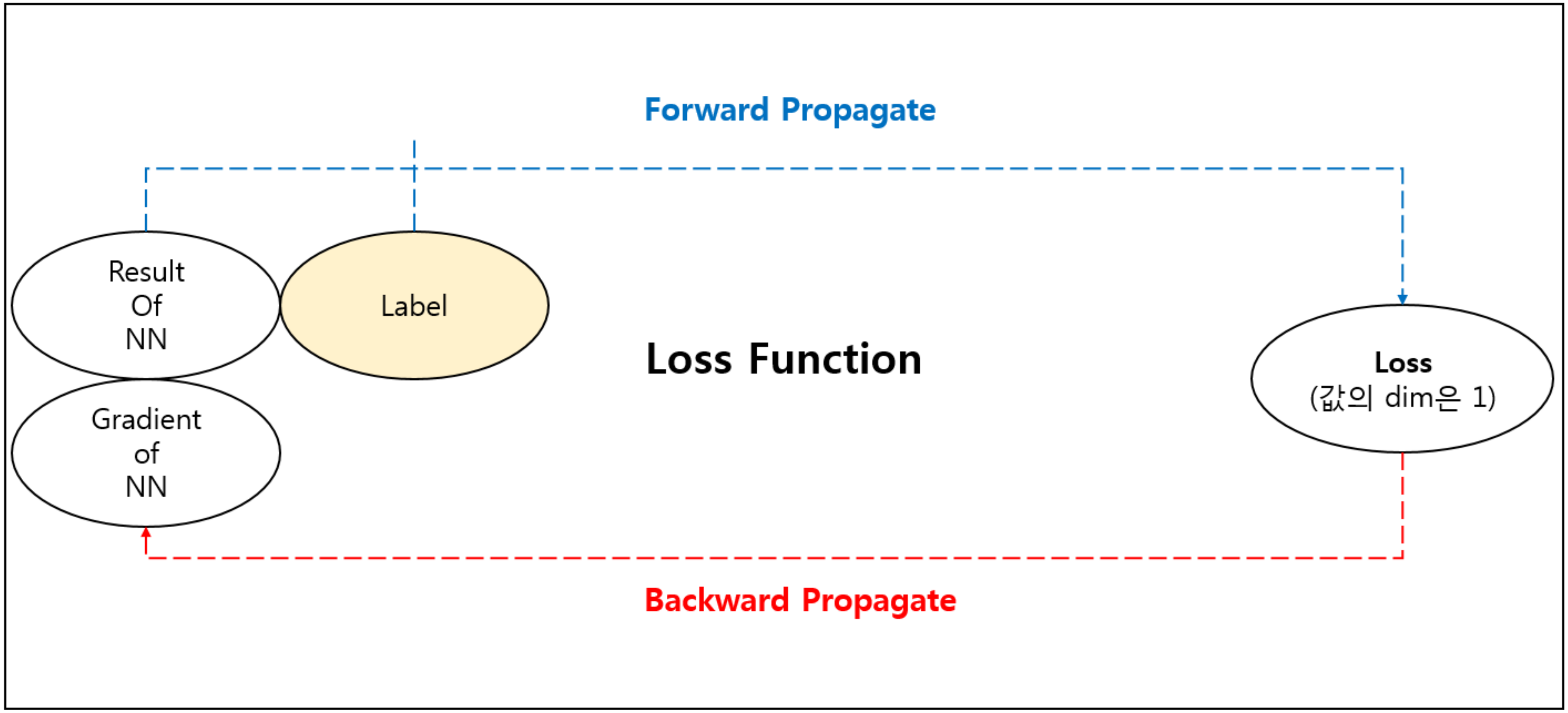
# WICWIU Components: Loss Function

- Loss Function class: loss functions for learning
  - Base class of Loss Function classes
  - Contains forward/backward propagation
  - Easy to extend to define custom LossFunction classes



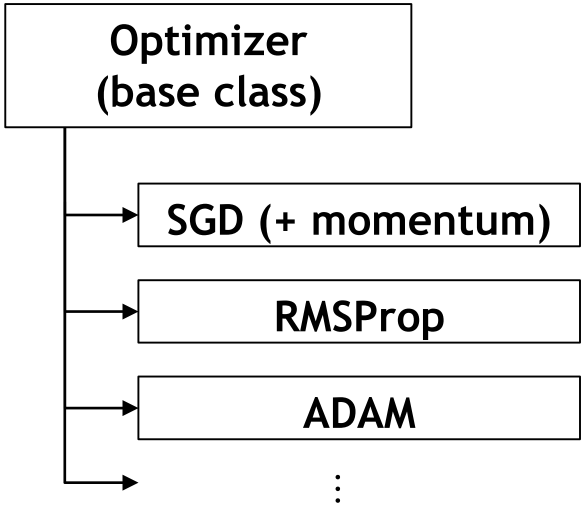



# WICWIU Components: Loss Function

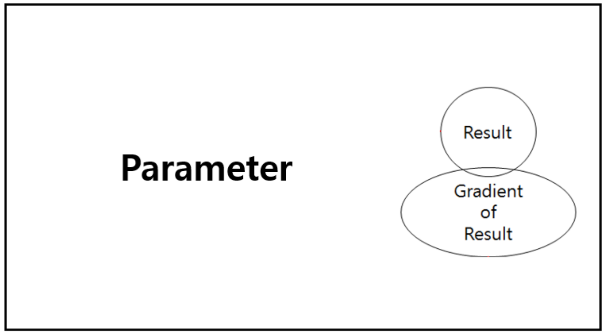


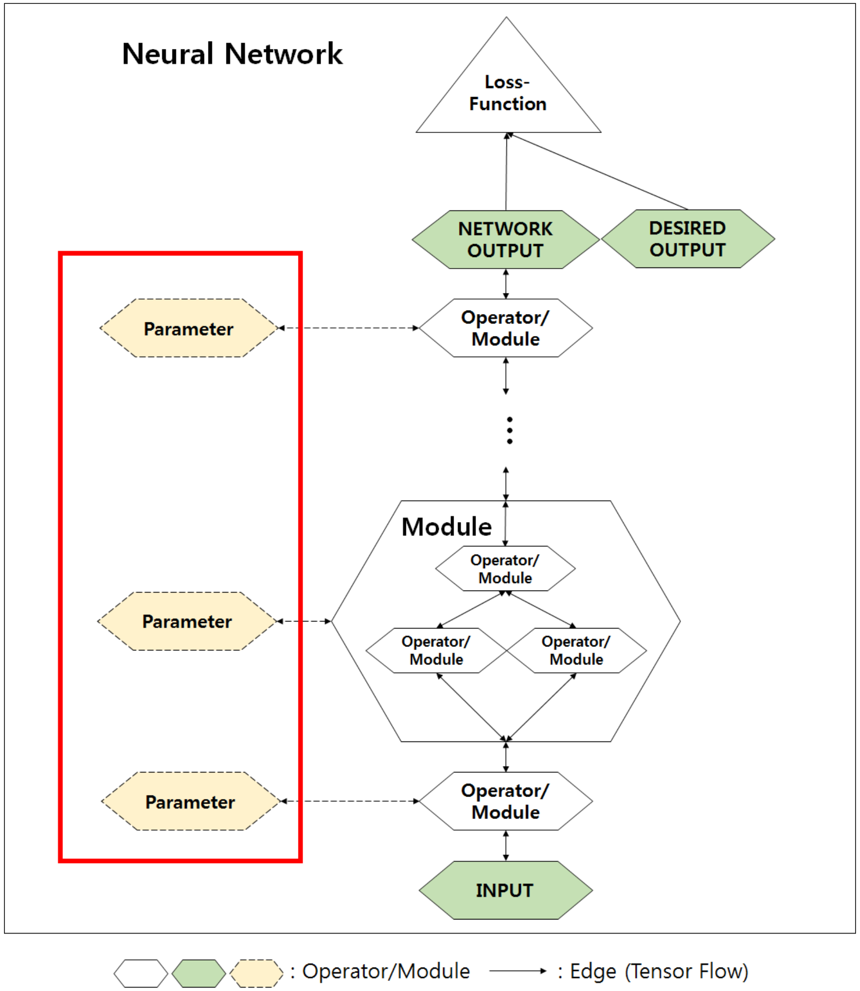
# WICWIU Components: Optimizer

- Optimizer class: 경사도 벡터(gradient)를 이용한 모델 파라미터를 최적화 알고리즘
  - Optimizer 공통 인터페이스
  - Parameter update using gradient vectors
  - Easy to extend to define custom Optimizer class
  - Built-in Optimizers

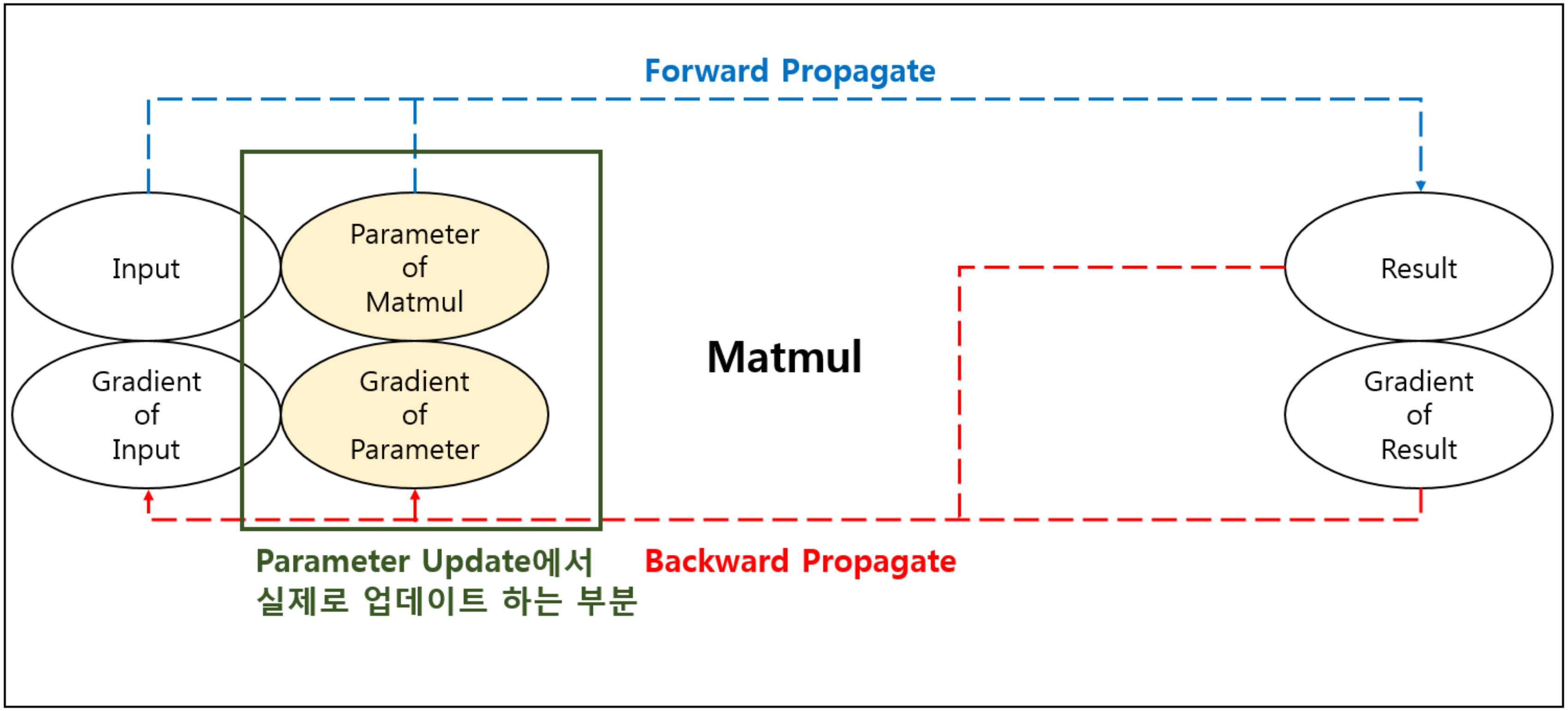


# WICWIU Components: Optimizer



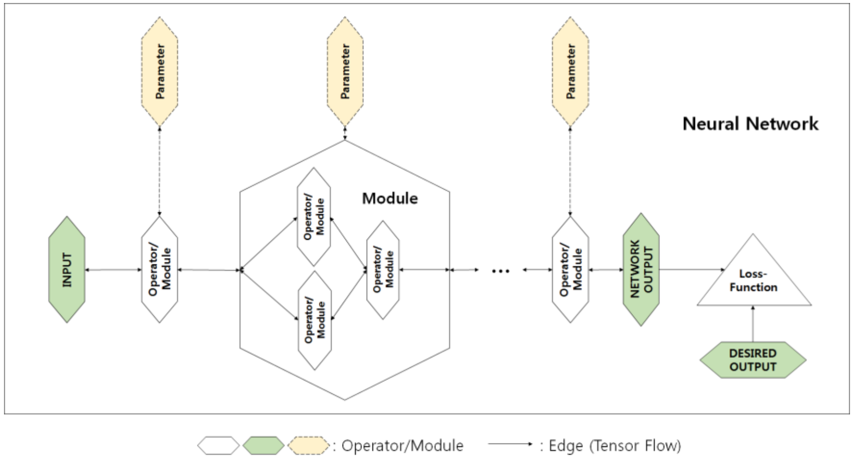



# WICWIU Components: Optimizer



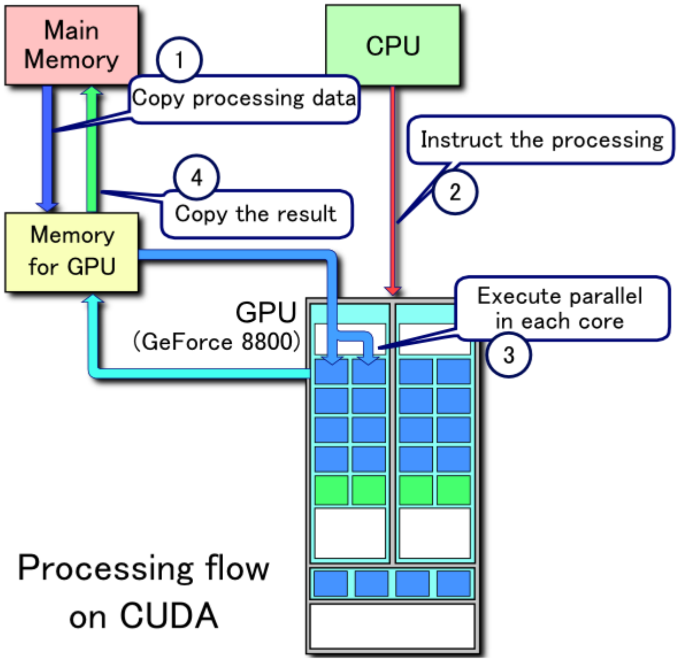
# WICWIU Components: Neural Network

- Neural Network class: general computational graph
  - Graph nodes (Operators and Modules)
  - Training components (LossFunction, Optimizer)
  - Operations
    - Add, connect Operators and Modules
    - Decides computing order by BFS
    - Provides Train / Test functions



# cuda & cuDNN

- 동작 원리
  - GPU에 필요한 자원을 따로 할당한다
  - 본래 CPU에서 실행하던 연산 대신 GPU 연산을 사용할 수 있도록 한다.
  - CPU(Host)와 GPU(Device)의 정보가 교차되는 순간에는 무조건 동기화 작업을 거치도록 한다.




Processing flow on CUDA

# cuda & cuDNN

```
int BackPropagateOnGPU(int pTime) {
    Container<Operator<DTYPE> *> *input_contatiner = this->GetInputContainer();

    Tensor<DTYPE> *left_grad  = (*input_contatiner)[0]->GetGradient();
    Tensor<DTYPE> *right_grad = (*input_contatiner)[1]->GetGradient();
    Tensor<DTYPE> *this_grad  = this->GetGradient();

    m_pDevLeftDelta  = left_grad->GetGPUData(pTime);
    m_pDevRightDelta = right_grad->GetGPUData(pTime);
    m_pDevDelta      = this_grad->GetGPUData(pTime);

    checkCUDNN(cudnnAddTensor(this->GetCudnnHandle(),
                              &m_alpha, deltaDesc, m_pDevDelta,
                              &m_alpha, leftDeltaDesc, m_pDevLeftDelta));

    checkCUDNN(cudnnAddTensor(this->GetCudnnHandle(),
                              &m_alpha, deltaDesc, m_pDevDelta,
                              &m_alpha, rightDeltaDesc, m_pDevRightDelta));



    return TRUE;
}
```

# cuda & cuDNN

```
#ifdef __CUDNN__
    void InitializeAttributeForGPU(unsigned int idOfDevice) {
        m_alpha = 1;
        m_beta  = 0;

        checkCUDNN(cudnnCreateTensorDescriptor(&leftTensorDesc));
        checkCUDNN(cudnnCreateTensorDescriptor(&rightTensorDesc));
        checkCUDNN(cudnnCreateTensorDescriptor(&outputTensorDesc));
        checkCUDNN(cudnnCreateTensorDescriptor(&leftDeltaDesc));
        checkCUDNN(cudnnCreateTensorDescriptor(&rightDeltaDesc));
        checkCUDNN(cudnnCreateTensorDescriptor(&deltaDesc));
```

# cuda & cuDNN

```
checkCUDNN(cudnnSetTensor4dDescriptor(outputTensorDesc, CUDNN_TENSOR_NCHW, CUDNN_DATA_FLOAT,
                                      m_batchsize, m_channelsize, m_rowsize, m_colsize));

checkCUDNN(cudnnSetTensor4dDescriptor(leftDeltaDesc, CUDNN_TENSOR_NCHW, CUDNN_DATA_FLOAT,
                                      m_batchsize, m_channelsize, m_rowsize, m_colsize));

checkCUDNN(cudnnSetTensor4dDescriptor(rightDeltaDesc, CUDNN_TENSOR_NCHW, CUDNN_DATA_FLOAT,
                                      m_batchsize, m_channelsize, m_rowsize, m_colsize));

checkCUDNN(cudnnSetTensor4dDescriptor(deltaDesc, CUDNN_TENSOR_NCHW, CUDNN_DATA_FLOAT,
                                      m_batchsize, m_channelsize, m_rowsize, m_colsize));
```

# cuda & cuDNN

```
int ForwardPropagate(int pTime = 0) {
    Container<Operator<DTYPE> *> *input_contatiner = this->GetInputContainer();

    Tensor<DTYPE> *left   = (*input_contatiner)[0]->GetResult();
    Tensor<DTYPE> *right  = (*input_contatiner)[1]->GetResult();
    Tensor<DTYPE> *result = this->GetResult();

    int m_ti = pTime;

    for (int m_ba = 0; m_ba < m_batchsize; m_ba++) {
        for (int m_ch = 0; m_ch < m_channelsize; m_ch++) {
            for (int m_ro = 0; m_ro < m_rowsize; m_ro++) {
                for (int m_co = 0; m_co < m_colsize; m_co++) {
                    (*result)[Index5D(m_pLeftTenShape, m_ti, m_ba, m_ch, m_ro, m_co)]
                        = (*left)[Index5D(m_pLeftTenShape, m_ti, m_ba, m_ch, m_ro, m_co)]
                          + (*right)[Index5D(m_pRightTenShape, m_ti, m_ba, m_ch, m_ro, m_co)];
                }
            }
        }
    }
```

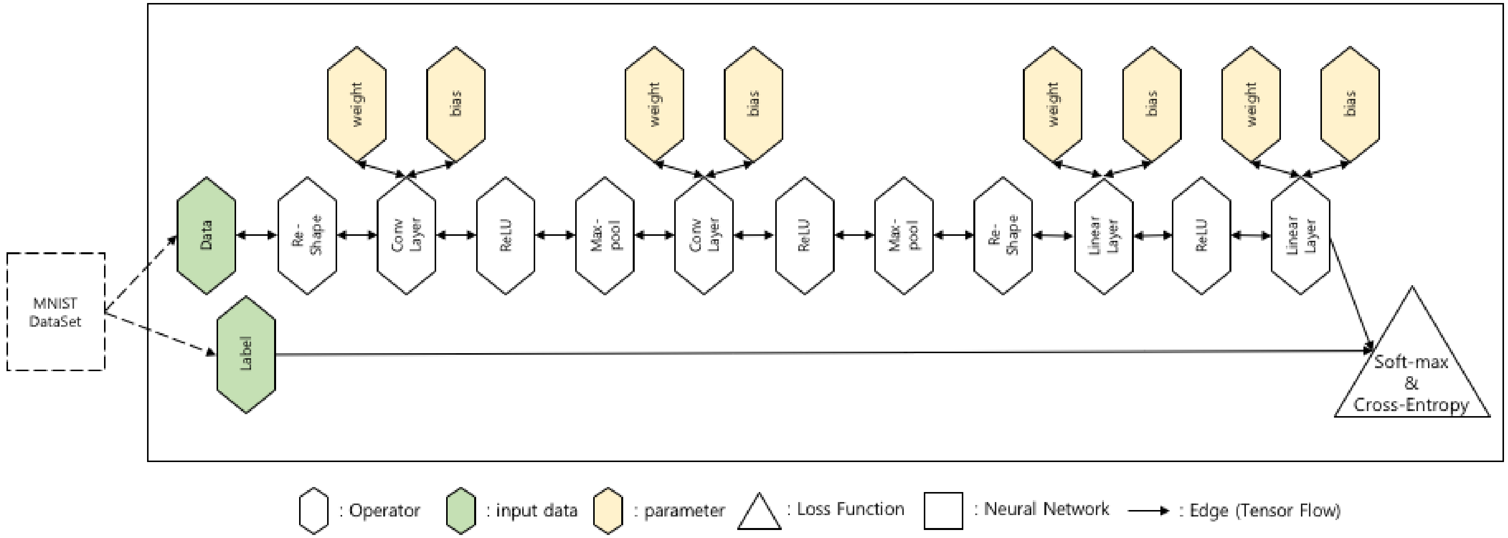
# cuda & cuDNN

```
int ForwardPropagateOnGPU(int pTime) {
    Container<Operator<DTYPE> *> *input_contatiner = this->GetInputContainer();

    Tensor<DTYPE> *left   = (*input_contatiner)[0]->GetResult();
    Tensor<DTYPE> *right  = (*input_contatiner)[1]->GetResult();
    Tensor<DTYPE> *result = this->GetResult();

    m_pDevLeft   = left->GetGPUData(pTime);
    m_pDevRight  = right->GetGPUData(pTime);
    m_pDevOutput = result->GetGPUData(pTime);

    checkCUDNN(cudnnAddTensor(this->GetCudnnHandle(),
                              &m_alpha, leftTensorDesc, m_pDevLeft,
                              &m_alpha, outputTensorDesc, m_pDevOutput));

    checkCUDNN(cudnnAddTensor(this->GetCudnnHandle(),
                              &m_alpha, rightTensorDesc, m_pDevRight,
                              &m_alpha, outputTensorDesc, m_pDevOutput));

    return TRUE;
}
```

# Example

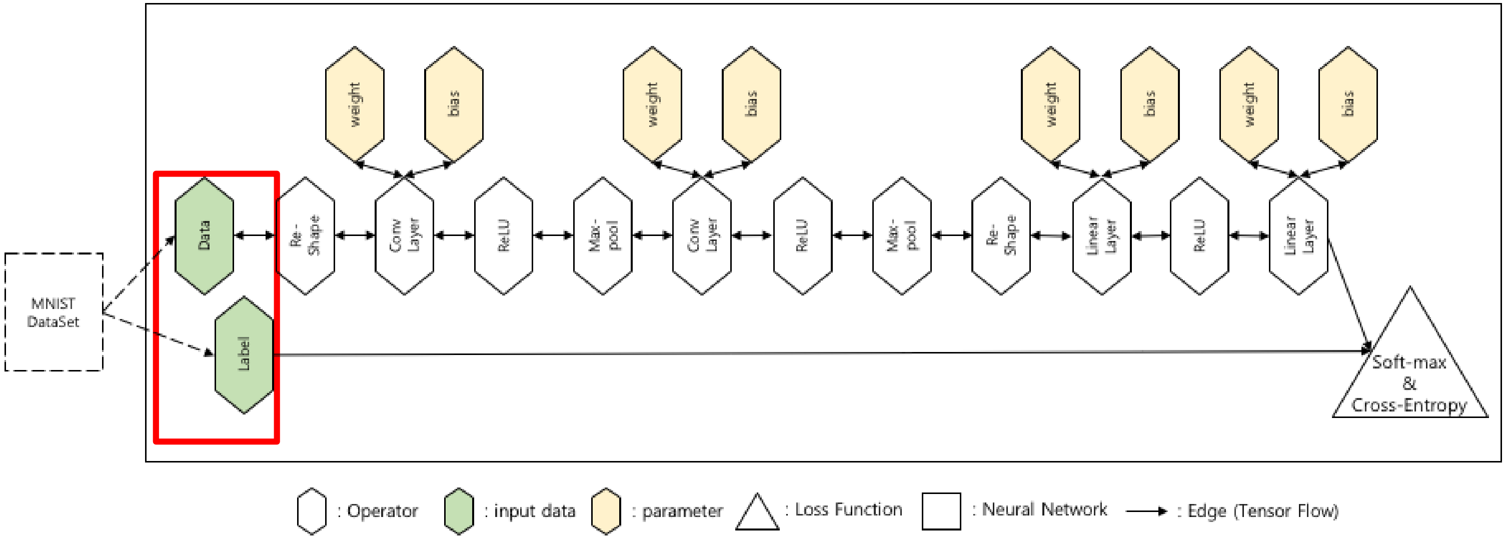
- 튜토리얼 – MNIST 필기 숫자 인식기
  - https://github.com/wicwiu/wicwiu/tutorials/MNIST

MNIST 인식을 위한 CNN 모델



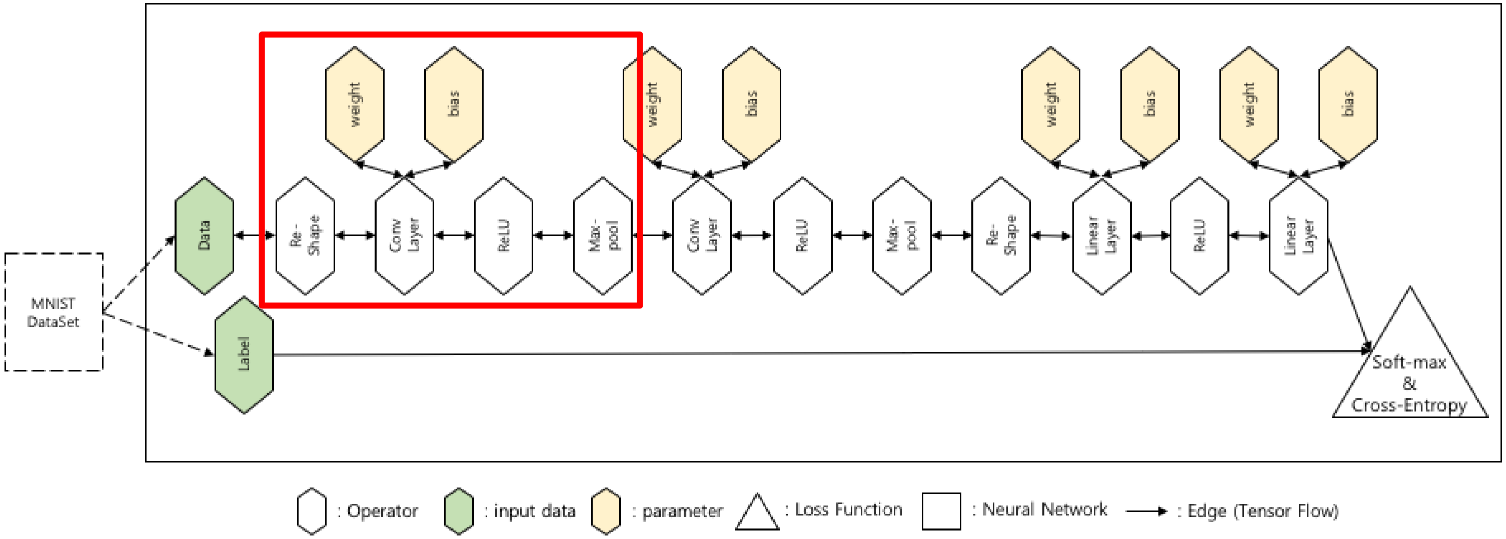
# Example: CNN Model Code

```
class my_CNN : public NeuralNetwork<float>{
private:
public:
    my_CNN(Tensorholder<float> *x, Tensorholder<float> *label) {
        SetInput(2, x, label);
```



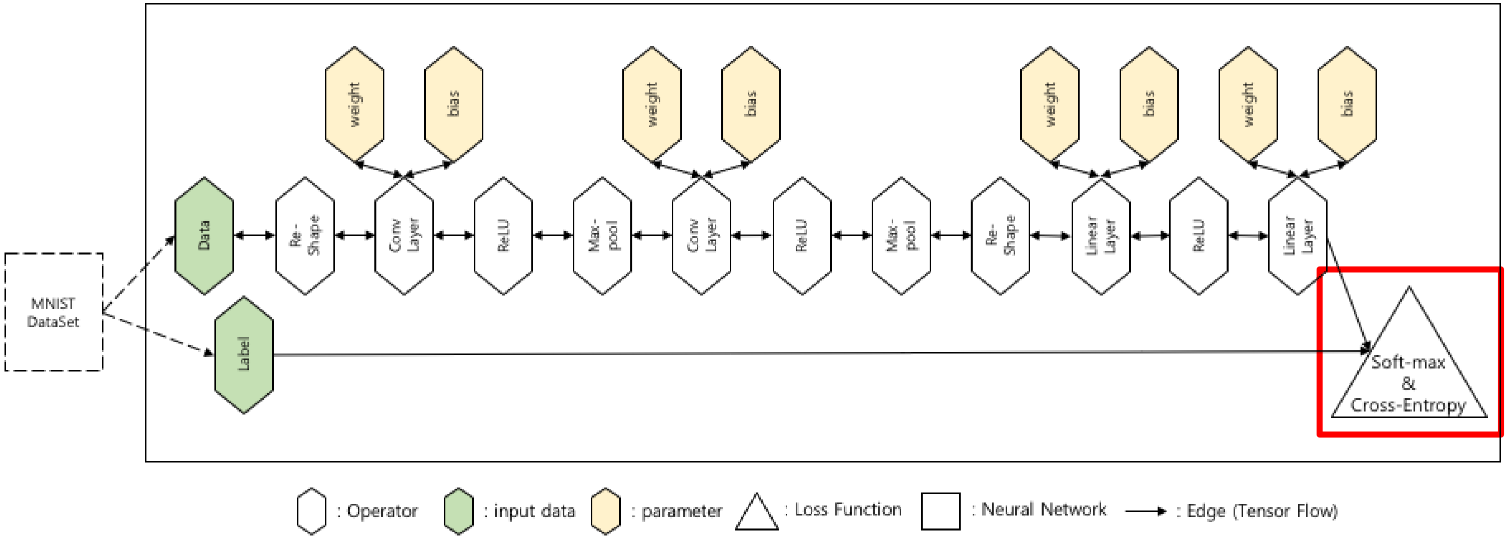
# Example: CNN Model Code

```
out = new ReShape<float>(x, 28, 28, "Flat2Image");

// ======================= layer 1=======================
out = new ConvolutionLayer2D<float>(out, 1, 10, 3, 3, 1, 1, 0, TRUE, "Conv_1");
out = new Relu<float>(out, "Relu_1");
out = new Maxpooling2D<float>(out, 2, 2, 2, 2, "MaxPool_1");
```



# Example: CNN Model Code

```
AnalyzeGraph(out);

// ====================== Select LossFunction Function ===================
SetLossFunction(new SoftmaxCrossEntropy<float>(out, label, "SCE"));

// ====================== Select Optimizer ===================
SetOptimizer(new GradientDescentOptimizer<float>(GetParameter(), 0.04, MINIMIZE));
```

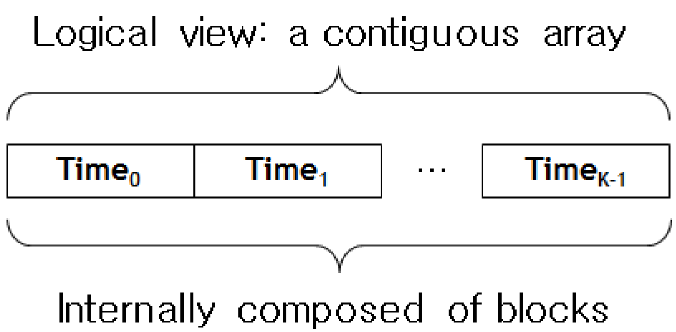


# Example: Training Code

```
// ====================== Select net ===================
NeuralNetwork<float> *net = new my_CNN(x, label);

// ====================== Prepare Data ===================
MNISTDataSet<float> *dataset = CreateMNISTDataSet<float>();

for (int i = 0; i < EPOCH; i++) {

    for (int j = 0; j < LOOP_FOR_TRAIN; j++) {
        dataset->CreateTrainDataPair(BATCH);

        Tensor<float> *x_t = dataset->GetTrainFeedImage();
        Tensor<float> *l_t = dataset->GetTrainFeedLabel();

        net->FeedInputTensor(2, x_t, l_t);
        net->ResetParameterGradient();
        net->Training();
    }
}
```

# Suggestion for RNN

- 현재 RNN과 관련해서 구현된 기능
  - Data per Time
    - Long array block 기준
    - GPU 데이터 블럭별 로드 가능




```
template<typename DTYPE> DTYPE *LongArray<DTYPE>::GetGPUData(unsigned int pTime) {
# if __DEBUG__

    if (m_Device == CPU) {
        printf("Warning! LongArray is allocated in Host(CPU) latest time\n");
        printf("Change mode CPU toGPU\n");
```

# Suggestion for RNN

- 현재 RNN과 관련해서 구현된 기능
  - Operator Time index handling

```
/*!
 * @brief  ForwardPropagate 매소드. 실제 구현은 파생 클래스에서 정의된다.
 * @param pTime ForwardPropagate할 데이터의 Time값.
 * @return 성공 시 TRUE.
 */
template<typename DTYPE> int Operator<DTYPE>::ForwardPropagate(int pTime) {
    #ifdef __DEBUG__

    #endif  // __DEBUG__
    return TRUE;
}

/*!
 * @brief  BackwardPropagate 매소드. 실제 구현은 파생 클래스에서 정의된다.
 * @param pTime forwardPropagate했던 데이터의 Time값.
 * @return 성공 시 TRUE.
 */
template<typename DTYPE> int Operator<DTYPE>::BackPropagate(int pTime) {
    #ifdef __DEBUG__

    #endif  // __DEBUG__
    return TRUE;
}
```

# Suggestion for RNN

- RNN 구현을 위한 제안
  - Neural Network에서 Network를 돌릴 때, 앞서 구현되어 있는 Time 별 Handling 기능을 잘 사용할 것
    - 참고로, Forward가 완벽하게 잘 돌아가는 것을 확신한다면, Backward는 정확히 그 반대 순서로 돌아가게 하면 정확하게 돌아간다.
    - 참고: Module.hpp, NeuralNetwork.hpp

```
template<typename DTYPE> int Module<DTYPE>::ForwardPropagateOnGPU(int pTime) {
    for (int i = 0; i < m_numOfExcutableOperator; i++) {
        (*m_aaExcutableOperator)[i]->ForwardPropagateOnGPU(pTime);
    }
    return TRUE;
}
```

```
template<typename DTYPE> int NeuralNetwork<DTYPE>::TrainOnCPU() {
    // this->ResetOperatorResult();
    // this->ResetOperatorGradient();
    this->ResetResult();
    this->ResetGradient();
    this->ResetLossFunctionResult();
    this->ResetLossFunctionGradient();

    this->ForwardPropagate();
```

# Suggestion for RNN

- RNN 구현을 위한 제안
  - Forward Propagate 정의하는 방식으로 진행
    - 지금은 Module이 자동으로 Forward Propagate를 정의
    - Module의 Forward, Backward 메서드를 virtual로 하면 사용자가 정의 가능
    - 참고:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
       x = F.relu(self.conv1(x))
       return F.relu(self.conv2(x))
```

# Suggestion for RNN

- RNN 구현을 위한 제안
  - Hidden은 Time 0 index 마다 초기화 (혹은 하지 않음)
    - 또한, Hidden에 사용하는 paramete는 save하지 않도록 내부 구현 진행
    - 초기화를 선택적으로 할 수 있도록 구현
    - 참고: https://gist.github.com/spro/ef26915065225df65c1187562eca7ec4

```
def forward(self, inputs, hidden=None, force=True, steps=0):
    if force or steps == 0: steps = len(inputs)
    outputs = Variable(torch.zeros(steps, 1, 1))
    for i in range(steps):
        if force or i == 0:
            input = inputs[i]
        else:
            input = output
        output, hidden = self.step(input, hidden)
        outputs[i] = output
    return outputs, hidden
```

# Suggestion for RNN

- RNN 구현을 위한 제안
  - Operator 혹은 Module의 Output Operator가 하나 이상이 될 수 있도록 수정
    - 현재 하나 이상의 Output node를 연결할 저장 공간은 마련되어 있으나, 내부적으로 사용을 막고 있음. 이 부분을 수정하여 사용할 수 있어야 함
    - 관련 메서드: Operator.hpp

```
Container<Operator<DTYPE> *> *m_apOutput;
///< Operator의 m_aaResult값을 사용할 Operator들의 주소 값.
```

```
int  AddOutputEdge(Operator<DTYPE> *pOutput);
```

```
        input = output
    output, hidden = self.step(input, hidden)
    outputs[i] = output
return outputs, hidden
```

# Future Works

- Loss function을 Operator와 합칠 것
  - Issue. Back Prop시에 맨 마지막에 사용되는 Operator는 this->gradient가 1로 채워진 매트릭스.
  - Link: https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html#gradients
  - Backprop시에 Default로 위의 pytorch link처럼 1로 채워진 tensor를 grad_tensor로 받는 방법 사용하는 방향 가능

- DataLoader branch를 master Branch와 merge하고, ImageNet tutorial에 있는 Image Preprocess class를 WICWIU_src로 옮겨서 다른 tutorial에서도 지원할 수 있도록 수정

# Future Works

- ETRI 관련 Task는 정리해서 일주일에 한번 건네줄 예정 (7월)
  - 현재까지 짜여진 코드 정리
  - 코드 이해 및 수정 요청
  - 필요 기능 구현 및 리뷰 진행

# 참고자료

- 구글 드라이브
  - WICWIU₩내부 공유용 파일₩5. 학교 제출용₩2019_2_4기_공프기₩인수인계자료
    - 시스템 상세 설계서
      - 참고: 일부 Class 이름이 다르게 나와있음
    - 구현 설명서 – 코드 내부 주석 참고
  - 그 이외에 다른 팀에서 작성한 자료들

# 부록1: Rquirement of WICWIU (1)

- 여러 데이터 형태를 모두 표현할 수 있고, 쉽게 확장 가능한 데이터 타입이 필요하다 (텐서)
- 현재 알려진 모든 신경망 모델을 표현할 수 있어야 하며, 확장이 가능해야 한다. (그래프)
- 자동 미분이 가능해야 하며, 이를 이용해서 모델 전체의 미분이 가능해야 한다. (체인룰)
- 모델의 동작을 위해 노드 동작에 확실한 선후 관계를 알고 있어야 한다. (BFS, DFS)

# 부록1: Rquirement of WICWIU (2)

- 속도와 최적화를 위해, 병렬 연산을 최대한 활용해야 한다. (Multi threading, GPU 연산)
- 데이터 전처리에서 시간을 최대한 줄여야 하며, 데이터를 다루기 쉬워야 한다. (생산자-소비자 문제, Data Preprocessing, Data Augmentation)
- 각 요소의 구현이 바뀌게 되더라도 전체 동작에는 영향을 미치지 말아야 한다. (캡슐화)