# HW 4 Report

**2016112083 김연웅**

# About My Model Design

6]:
```
class LSTM(nn.Module):
    def __init__(self, input_size=5, hidden_size=5, num_layers=1, output_size=5):
        super(LSTM, self).__init__()
        self.num_layers = num_layers #number of layers
        self.input_size = input_size #input size
        self.hidden_size = hidden_size #hidden state
        self.output_size = output_size
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True) #lstm
        self.fc_1 =  nn.Linear(hidden_size, 128) #fully connected 1
        self.fc = nn.Linear(128, output_size) #fully connected last layer

        self.relu = nn.ReLU()

    def forward(self,x):
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size)) #hidden state
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size)) #internal state
        # Propagate input through LSTM
        output, (hn, cn) = self.lstm(x, (h_0, c_0)) #lstm with input, hidden, and internal state
        hn = hn.view(-1, self.hidden_size) #reshaping the data for Dense layer next
        out = self.relu(hn)
        out = self.fc_1(out) #first Dense
        out = self.relu(out) #relu
        out = self.fc(out) #Final Output
        return out
```

## Model Layers

1. LSTM

- Input size: 5
- hidden size: 5
- num_layers: 1
- output size: 5

2. **Activation function** : ReLU
3. Fully Connected Layer

   1st : 128

   2nd(output) : 128

## Number of Parameters

In [7]:
```
model = LSTM()
loss_function = MAPE()
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)


params = list(model.parameters())
print("The number of parameters:", sum([p.numel() for p in model.parameters() if p.requires_grad]), "elements")
```

```
The number of parameters: 1653 elements
```

In [8]:
```
print(model)
```

```
LSTM(
  (lstm): LSTM(5, 5, batch_first=True)
  (fc_1): Linear(in_features=5, out_features=128, bias=True)
  (fc): Linear(in_features=128, out_features=5, bias=True)
  (relu): ReLU()
)
```

**The number of parameters: 1653 elements.**

# accuracy in terms of the mean absolute percentage error (MAPE)

```
accuracy in terms of the mean absolute percentage error (MAPE): 0.4647793471813202
```

In [76]:
```
# Load best performance model
checkpoint = torch.load('./bestModel.pt')
trained_model = LSTM()

trained_model.load_state_dict(checkpoint['state_dict'])

# evaluate
trained_model.eval()
with torch.no_grad():
    test_data = np.load('test_series.npy')
    # load test dataset
    test_x = test_data[:,:,0]
    test_y = test_data[:,:,1]


    test_x=torch.from_numpy(test_x).float()
    test_y=torch.from_numpy(test_y).float()
    test_x = test_x.reshape(-1,1,5)
    # evaluate using weighted f1 score.
    y_pred = trained_model(test_x)
    loss = loss_function(y_pred, test_y)
    print("accuracy in terms of the mean absolute percentage error (MAPE):", loss.item())
```

```
accuracy in terms of the mean absolute percentage error (MAPE): 0.4647793471813202
```

## Hyper parameter setting

num epochs : 100

Learnint rate :  0.001

batch size : 32

## Random Seeds Configuration

```
# Configure Random seeds
torch.manual_seed(777)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(777)
```