KAIST Korea Advanced Institute of Science and Technology

Week 3: MNIST using MLP

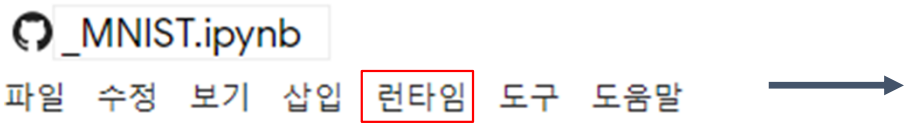
KAIST

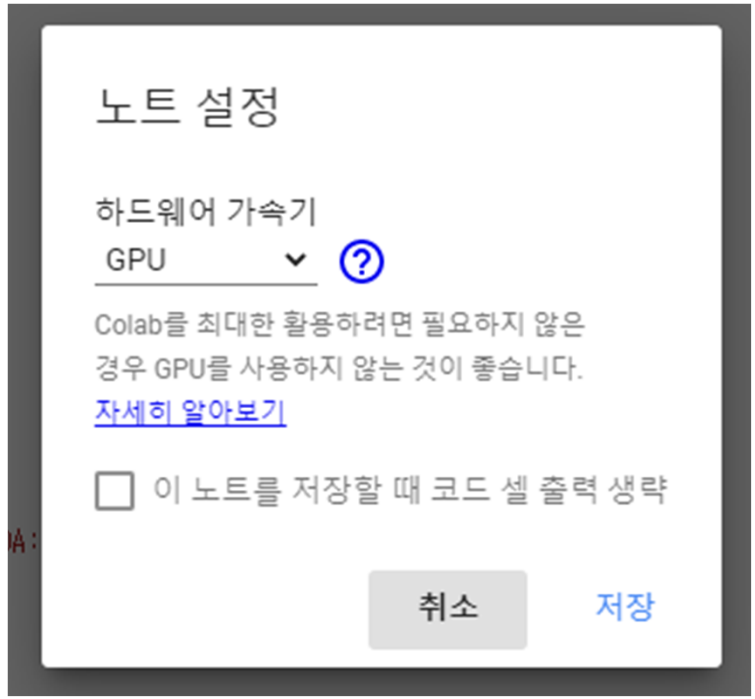
Mobile Robitics & Intelligence Lab

박사과정 김진식

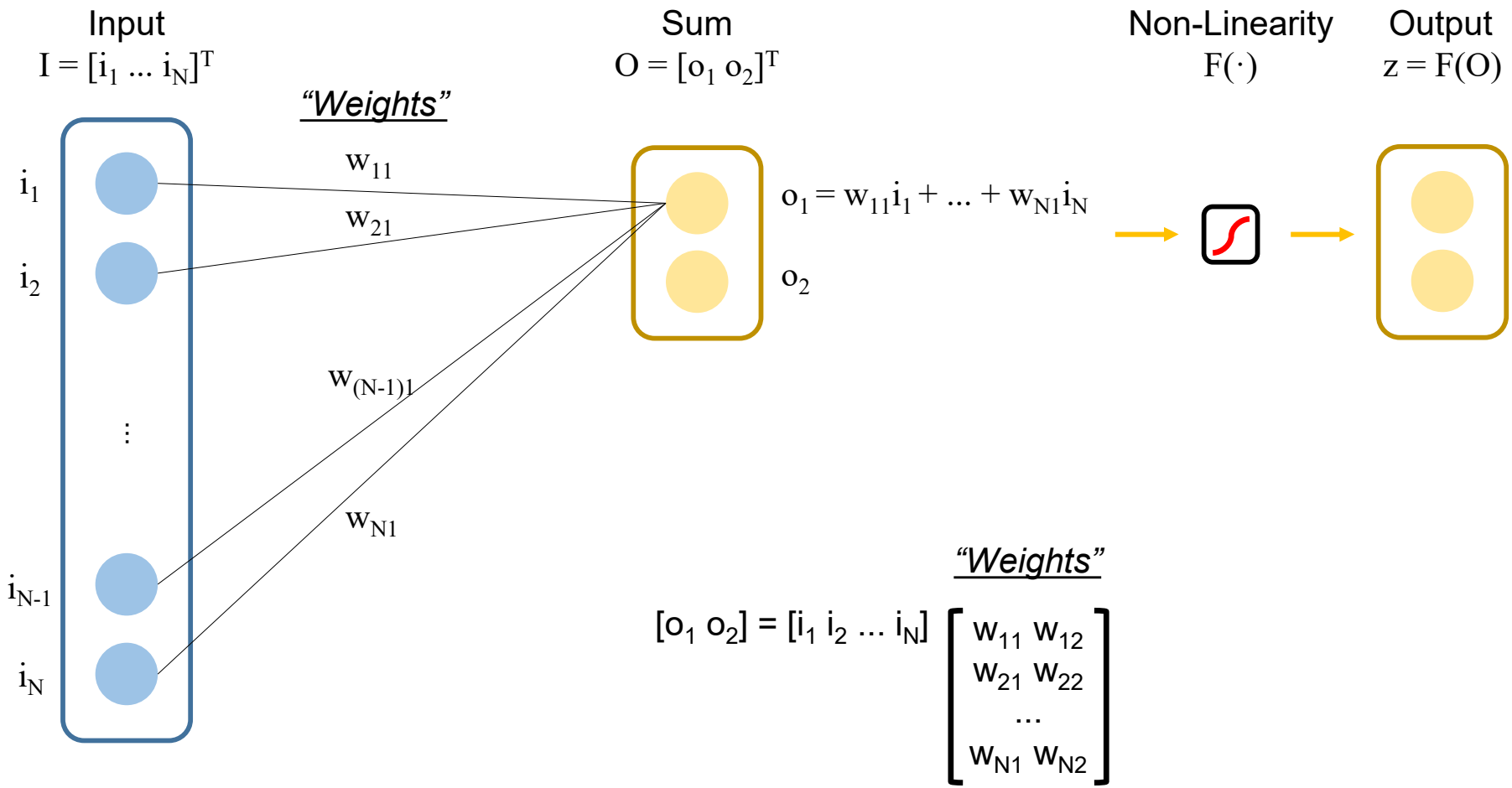
Prerequisite - Colab

런타임 -> 런타임 유형 변경 -> 하드웨어 가속기(GPU)

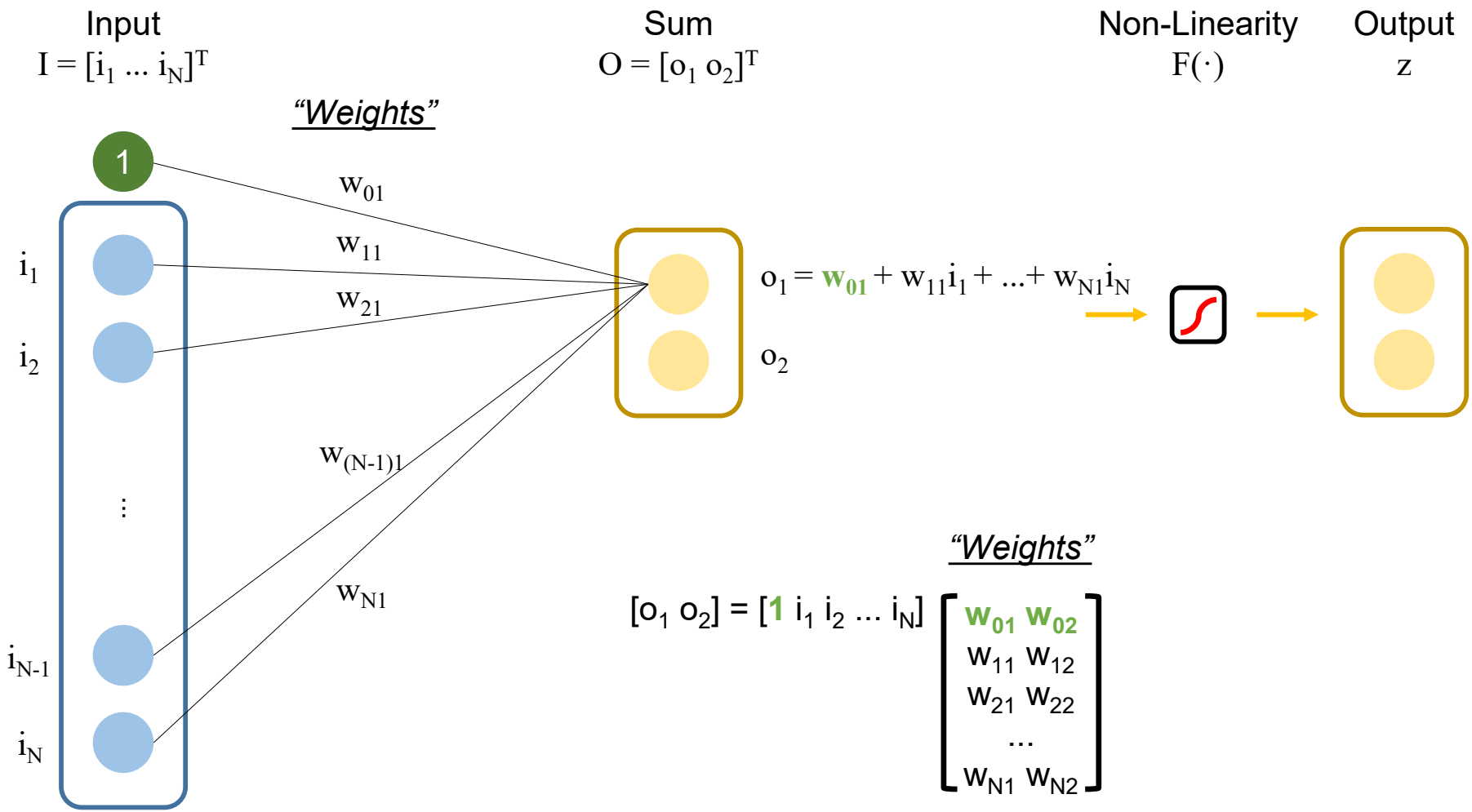




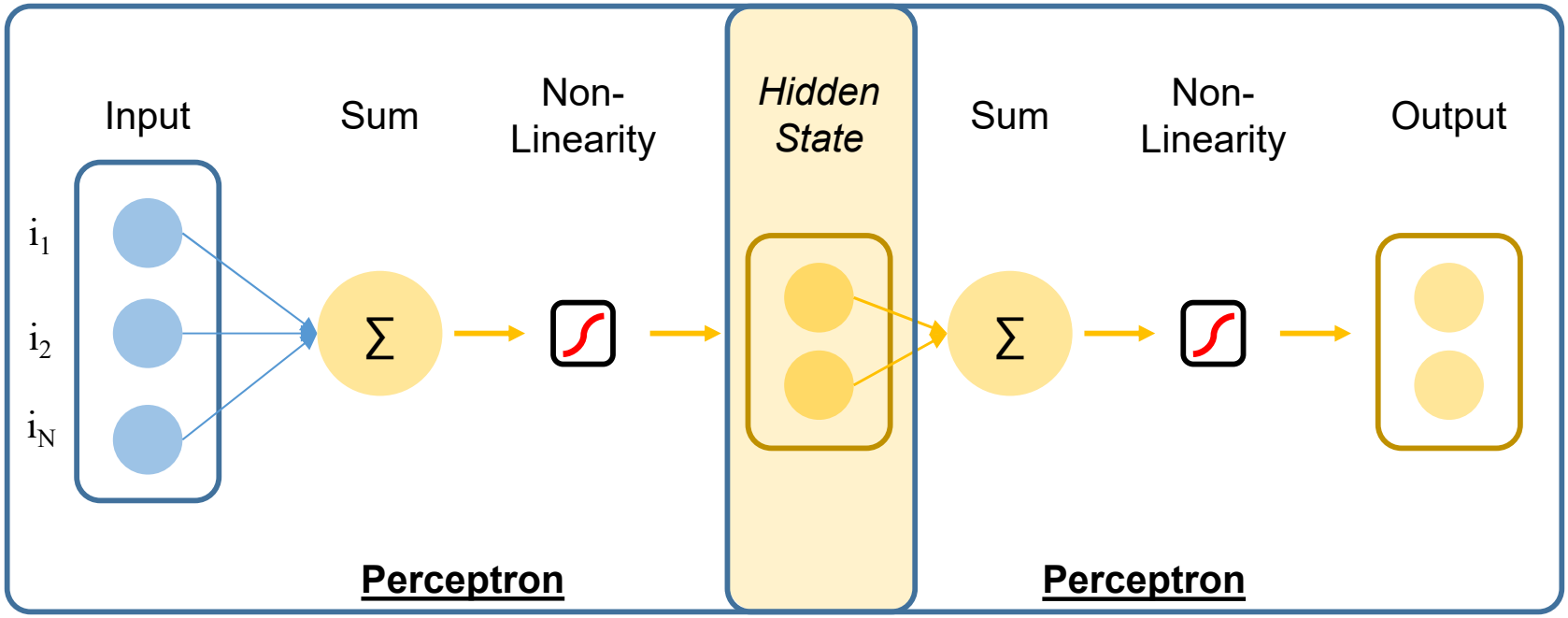
Perceptrons (No bias)



Perceptrons (with bias)



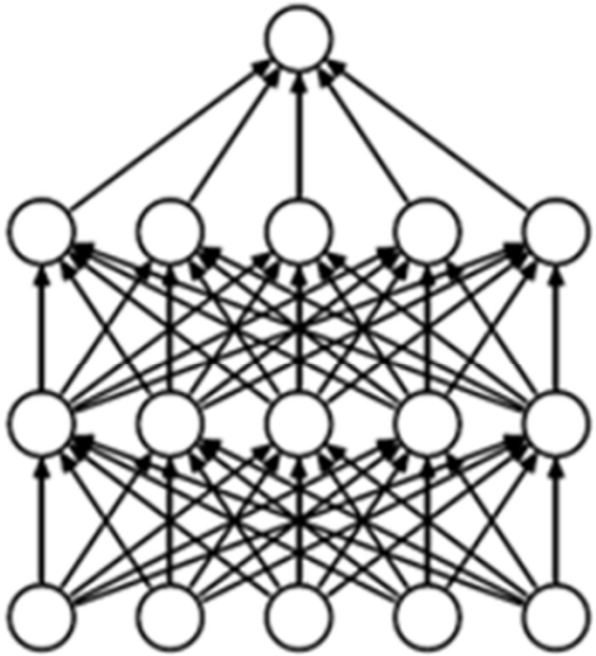
Multi-layer perceptron



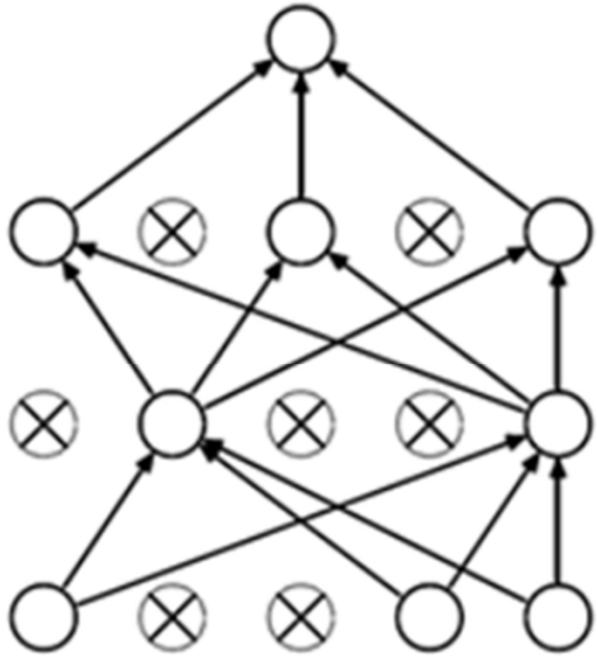
+ Dropout

Multi-layer perceptron – dropout

- One of the biggest problems of deep learning is overfitting to training data.
- *Dropout rate* defines probability of each neuron to be *zeroed out*.
- We can partially solve the overfitting problem through dropout method.



(a) Standard Neural Net

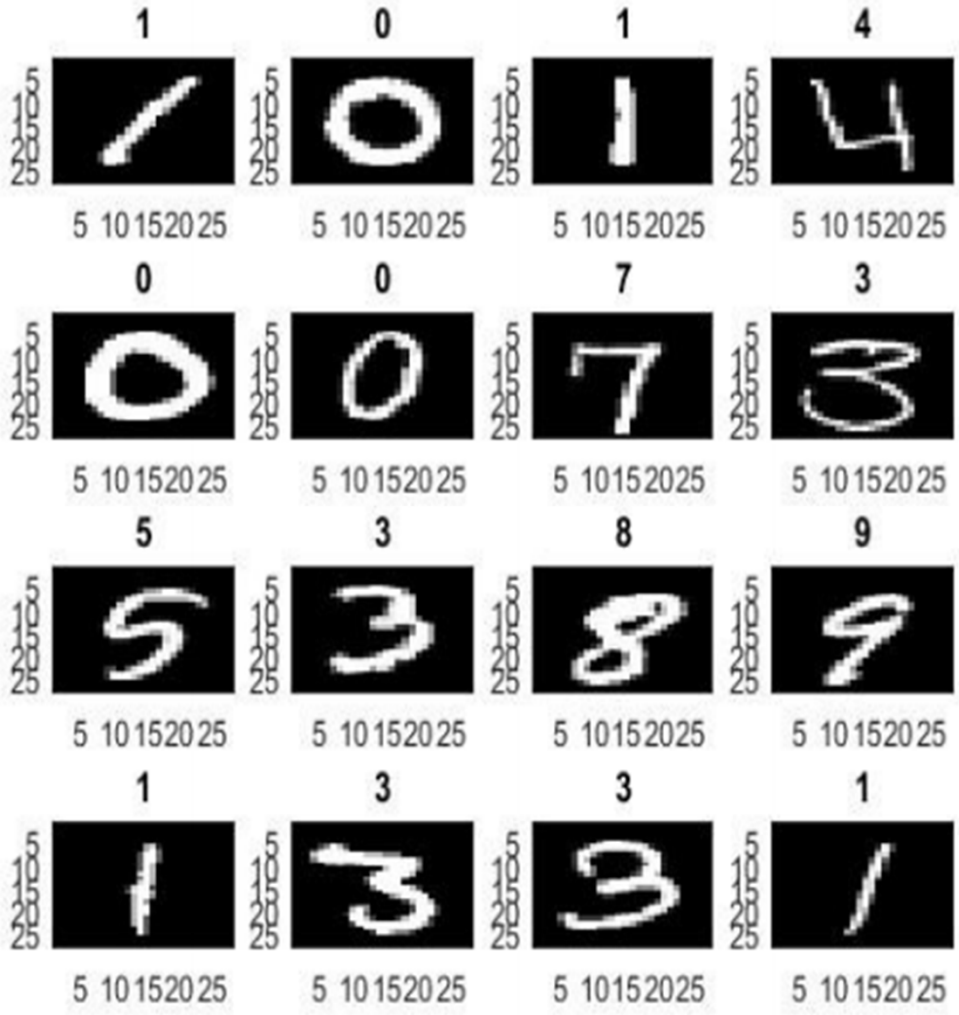


(b) After applying dropout.

Introduction

MNIST Dataset

- MNIST: Large hand written digit classification database
- Format
  - Input: 28 x 28 gray scale image
  - Output: 10 labels(0-9)
  - Centered on center of the mass



Today : Image(28 × 28) classification using MLP

MNIST tutorial

Dataset preparation

```
batch_size = 32

kwargs = {'num_workers': 1, 'pin_memory': True} if cuda else {}

# This will automatically download MNIST data into ../data directory
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=True, **kwargs)

validation_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=False, **kwargs)
```

Training

Validation

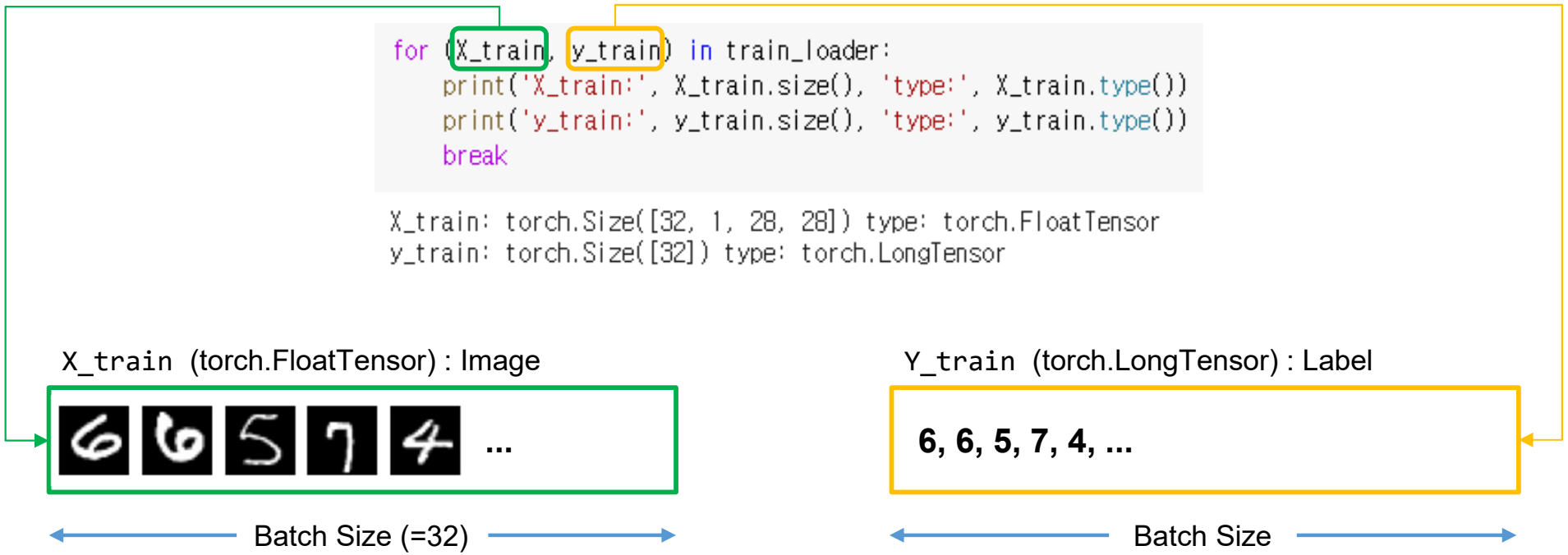
- Load MNIST dataset using data loader
- Input size: 28 x 28 gray scale image
- Output: classes of (“0”, .. “9”) for each training digit

MNIST tutorial

Dataloader usage

```
for (X_train, y_train) in train_loader:
    print('X_train:', X_train.size(), 'type:', X_train.type())
    print('y_train:', y_train.size(), 'type:', y_train.type())
    break
```

```
X_train: torch.Size([32, 1, 28, 28]) type: torch.FloatTensor
y_train: torch.Size([32]) type: torch.LongTensor
```



- "Batch" : Divide dataset into smaller sets

X_train.shape : (Batch Size) × (Channels #) × (Width) × (Height) # 32 × 1 × 28 × 28
Y_train.shape : (Batch Size) # 32

MNIST classification with MLP

Input = 5 x 1 x 28 x 28

MLP

Probability

Output = 5 x 10

Example

MLP

z	0	1	2	3	4	**5**	6	7	8	9
P(z)	0.02	0.01	0.01	0.02	0.01	**0.85**	0.03	0.01	0.02	0.02

- Softmax

$\mathbf{z} = [z_1, ..., z_n]$

$\text{softmax}(\mathbf{z}) = [e^{z1}/\Sigma(e^{zk}), e^{z2}/\Sigma(e^{zk}), ..., e^{zn}/\Sigma(e^{zk})]$

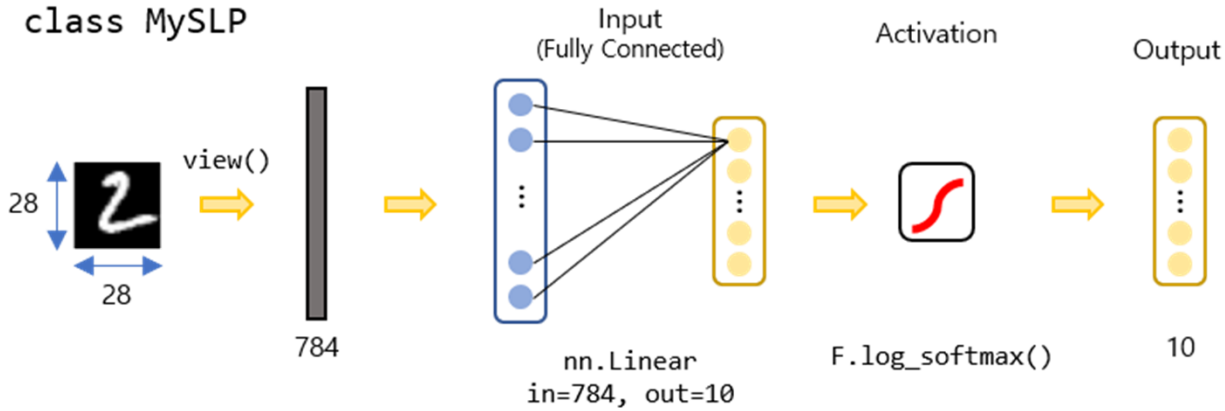
Practice

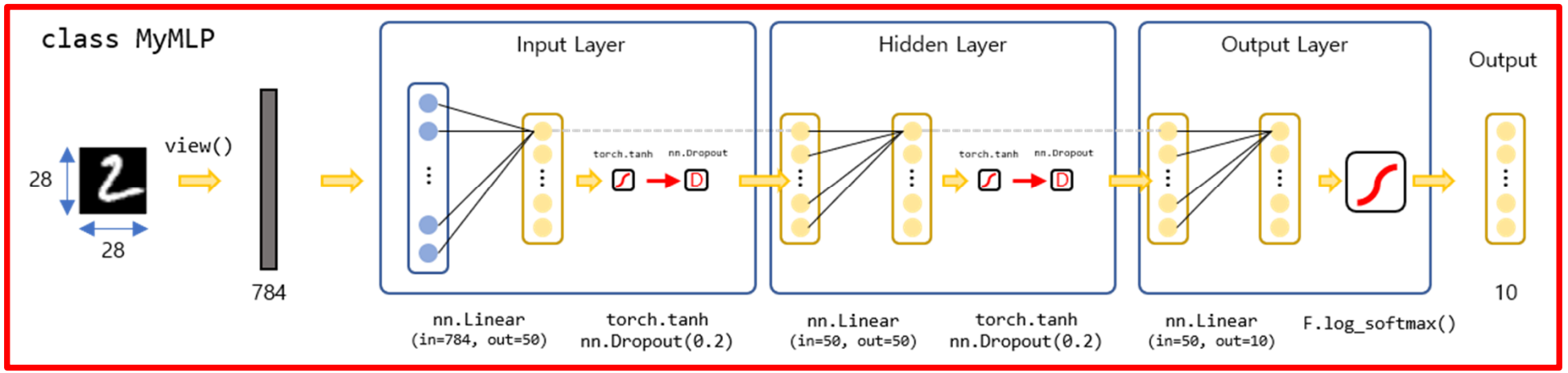
Today, you will be asked to

- Task 1. Implement custom MLP class
- Task 2. Complete training pipeline
  + Visualization

Check out colab notebook for details

Implement MyMLP class





Pytorch MLP Basic – nn.Linear

CLASS `torch.nn.Linear(in_features: int, out_features: int, bias: bool = True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

Shape:

- Input: $(N, *, H_{in})$ where $*$ means any number of additional dimensions and $H_{in} = \text{in\_features}$
- Output: $(N, *, H_{out})$ where all but the last dimension are the same shape as the input and $H_{out} = \text{out\_features}$.

Variables

- **~Linear.weight** – the learnable weights of the module of shape $(\text{out\_features}, \text{in\_features})$. The values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in\_features}}$
- **~Linear.bias** – the learnable bias of the module of shape (out_features). If `bias` is `True`, the values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{in\_features}}$

https://pytorch.org/docs/master/generated/torch.nn.Linear.html

Pytorch MLP Basic – nn.Dropout

CLASS `torch.nn.Dropout(p: float = 0.5, inplace: bool = False)` [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Parameters

- **p** – probability of an element to be zeroed. Default: 0.5
- **inplace** – If set to `True`, will do this operation in-place. Default: `False`

Shape:

- Input: $(*)$. Input can be of any shape
- Output: $(*)$. Output is of the same shape as input

https://pytorch.org/docs/master/generated/torch.nn.Dropout.html

Complete training pipeline (2-1)

Training

```
def train(model, optimizer, epoch, log_interval=100):
    model.train()   # Set model to training mode

    for batch_idx, (data, target) in enumerate(train_loader):
        if cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)

        # 2-1. Clear remaining gradients by setting to zero

        # 2-2. Call forward() of model

        # 2-3. Compute loss (use F.nll_loss)

        # 2-4. Compute gradients

        # 2-5. Backpropagation by updating weights

        pass

    ## CLI Visualization (Optional) ##
        if (((batch_idx % log_interval) == 0) or (batch_idx == len(train_loader)-1)):
```

Complete training pipeline (2-1)

Training

```
def train(epoch, log_interval=100):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```

Set to train mode

Get output

Calculate loss

Backpropagation using optimizer

Print training log

Validation (Complete)

```
def validate(loss_vector, accuracy_vector):
    model.eval()
    val_loss, correct = 0, 0
    for data, target in validation_loader:
        if cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        val_loss += F.nll_loss(output, target).data[0]
        pred = output.data.max(1)[1] # get the index of the max log-probability
        correct += pred.eq(target.data).cpu().sum()

    val_loss /= len(validation_loader)
    loss_vector.append(val_loss)

    accuracy = 100. * correct / len(validation_loader.dataset)
    accuracy_vector.append(accuracy)

    print('\nValidation set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        val_loss, correct, len(validation_loader.dataset), accuracy))
```

Set to validation mode

Calculate validation loss

Calculate accuracy of validation set

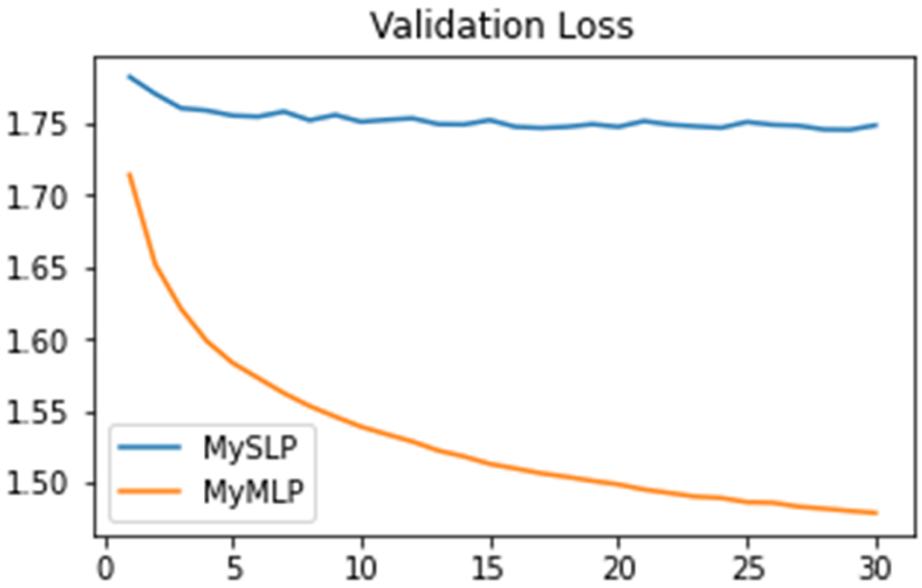
Print log of accuracy of validation set

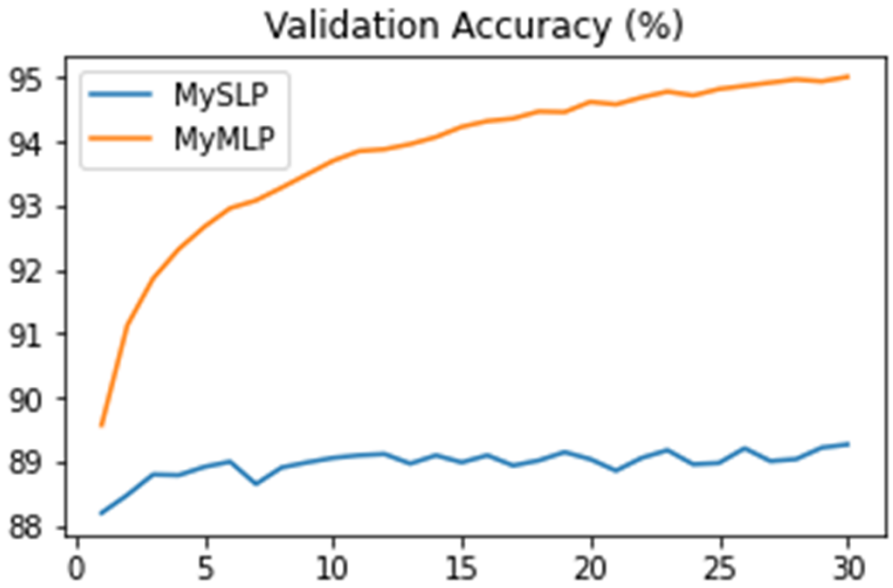
Visualization (2-3)

```
plt.figure(figsize=(5,3))
plt.plot(np.arange(1,epochs+1), lossv)
plt.title('validation loss')

plt.figure(figsize=(5,3))
plt.plot(np.arange(1,epochs+1), accv)
plt.title('validation accuracy');
```

Plot the loss and accuracy graph
We can visualize the performance and the loss of our network in validation sets.





- Today's quiz #1. Complete **~2.3** and submit your outputs (must run by yourself)

Practice Session / Q & A

Task 1. Implement custom MLP class (1)
Task 2. Complete training pipeline + Visualization (2-1)

- Quiz will begin at 5:30