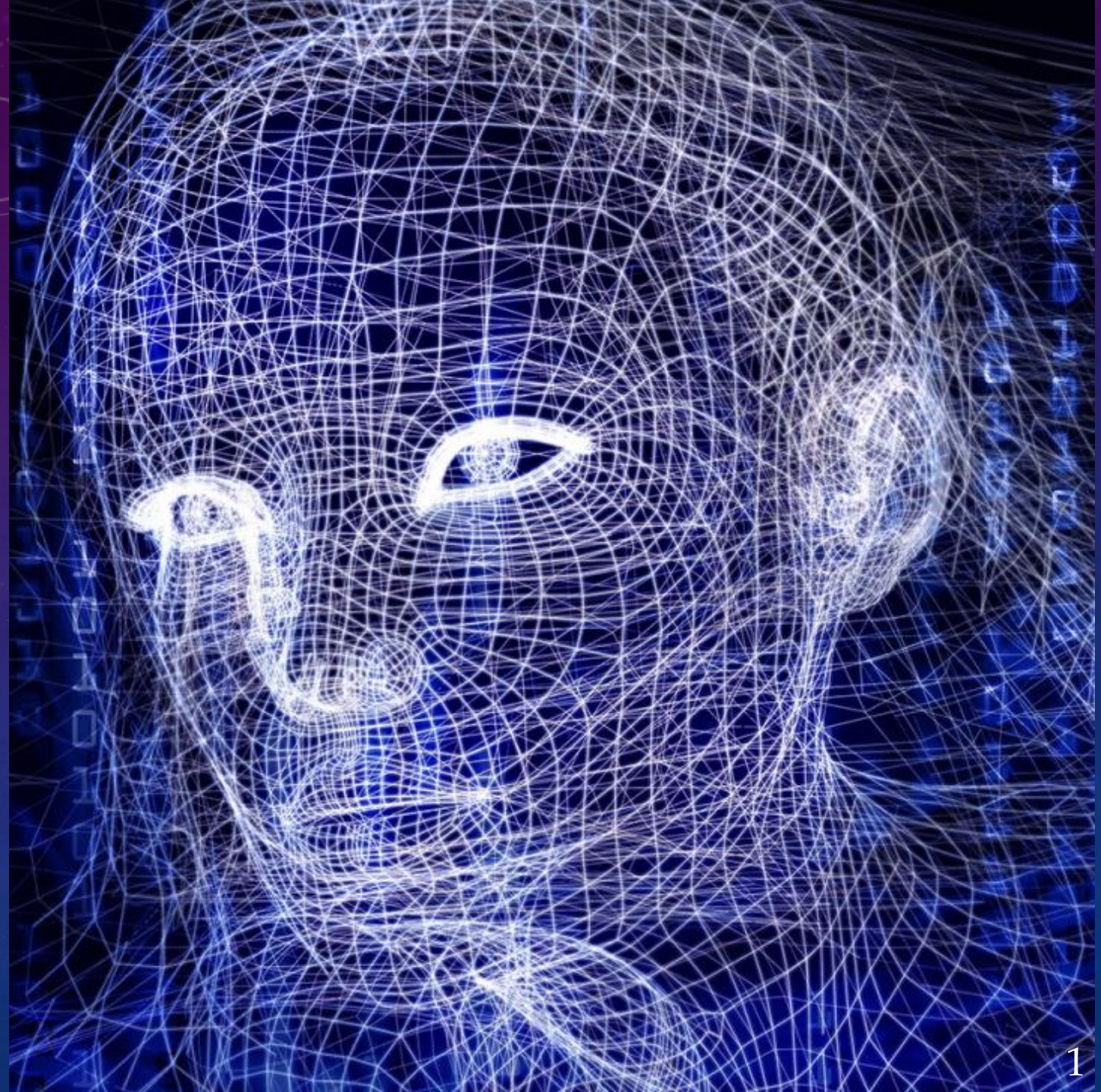
DIGITAL SURVEILLANCE SYSTEMS AND APPLICATION

CH 2_EXERCISE

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology



Example 2.1 Build A Neural Network

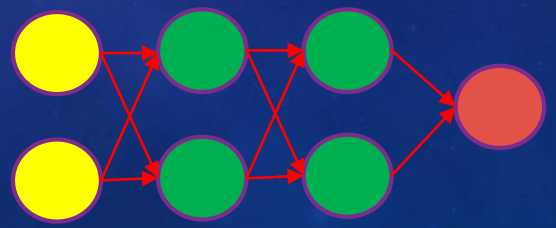
Please download 2-1_log_regression_visualize.zip and unzip it.

- Given an architecture with input size=2, hidden layer=2 and output size=1.
- Each layer is followed by ReLU activation function.
- In this case, we can know how to use “Autograd” to use the loss to compute the gradient; and how to establish a neural network in Pytorch.

Step:

- Autograd
- Build A Neural Network

```
Feedforward(
  (input): Linear(in_features=2, out_features=2, bias=True)
  (relu1): ReLU()
  (fc1): Linear(in_features=2, out_features=2, bias=True)
  (relu2): ReLU()
  (fc2): Linear(in_features=2, out_features=2, bias=True)
  (relu3): ReLU()
  (output): Linear(in_features=2, out_features=1, bias=True)
)
```



Yellow : Input layer
Green : Hidden layer
Orange : Output layer

Example 2.1 Build A Neural Network

- Autograd, which is an automatic differentiation package provided by PyTorch.
- The gradient can only be calculated if your variable is a function of the variable you want to differentiate.

```
import torch

x = torch.ones(1, requires_grad=True)

print(x.grad)    # returns None
```

Example 2.1 Build A Neural Network

Example:

```
import torch

x = torch.ones(1, requires_grad=True)
y = x + 2
z = y * y * 2

z.backward()     # automatically
calculates the gradient
print(x.grad)    # ∂z/∂x = 12
```

.backward() : performs the backpropagation automatically

$$\frac{\partial z}{\partial x} = \frac{\partial 2y^2}{\partial x}$$

$$= 4y\frac{\partial y}{\partial x}$$

$$= 4(x+2)\frac{\partial (x+2)}{\partial x}$$

$$= 4(x+2)$$

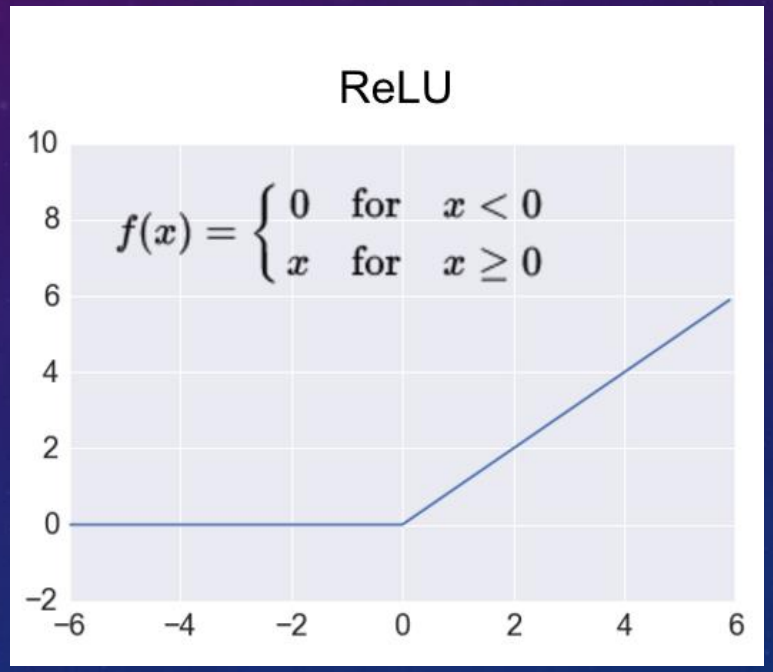
$$\rightarrow x = 1$$

$$\rightarrow \frac{\partial z}{\partial x} = 12$$

Example 2.1 Build A Neural Network

Example:

```
class Perceptron(torch.nn.Module):
    def __init__(self):
        super(Perceptron, self).__init__()
        self.fc = nn.Linear(1,1)
        self.relu = torch.nn.ReLU()
    def forward(self, x):
        output = self.fc(x)
        output = self.relu(x)
      return output
```



self.fc = outputs linear information
self.relu = makes it non-linear

Example 2.1 Build A Neural Network

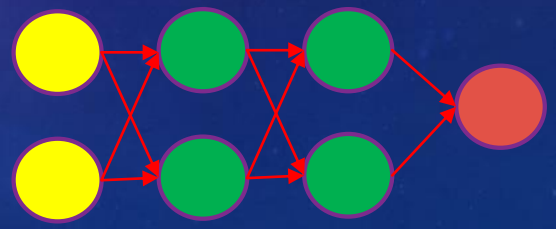
```
class Fedforward(torch.nn.Module):
        def __init__(self):
            super(Feedforward, self).__init__()
            self.input_size = 2
            self.hidden_size  = 2
            self.output_size  = 1
            self.input = torch.nn.Linear(self.input_size,
self.hidden_size)
            self.fc1 = torch.nn.Linear(self. hidden_size,
self.hidden_size)
            self.relu = torch.nn.ReLU()
            self.fc2 = torch.nn.Linear(self.hidden_size, self.
hidden_size)
            self.output = torch.nn.Linear(self.hidden_size,
self.output_size)
        def forward(self, x):
            input = self.input(x)
            relu1 = self.relu(input)
            hidden1 = self.fc1(relu1)
            relu2 = self.relu(hidden1)
            hidden2 = self.fc2(relu2)
            output = self.relu(hidden2)
            output = self.output(output)
```

Parameters

Relu function

When you have more than two hidden layers, the model is also called the deep/multilayer feedforward model or MultiLayer Perceptron (MLP).

```
Feedforward(
  (input): Linear(in_features=2, out_features=2, bias=True)
  (relu1): ReLU()
  (fc1): Linear(in_features=2, out_features=2, bias=True)
  (relu2): ReLU()
  (fc2): Linear(in_features=2, out_features=2, bias=True)
  (relu3): ReLU()
  (output): Linear(in_features=2, out_features=1, bias=True)
)
```



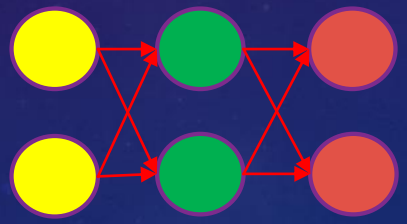
Yellow : Input layer
Green : Hidden layer
Orange : Output layer

Exercise 2.1 Build A Neural Network

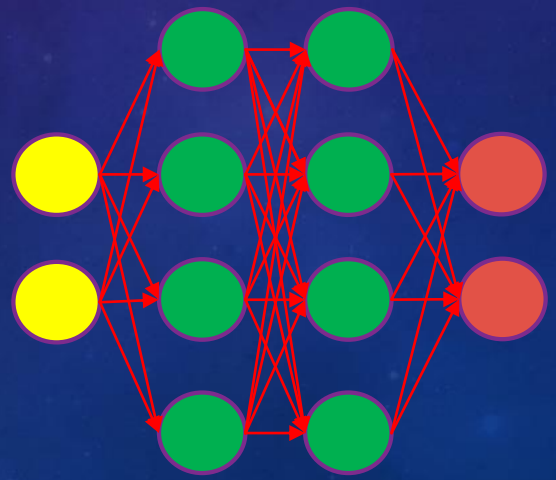
Please download 2-1_log_regression_visualize.zip and unzip it.

- Please modify the settings in the file so that it can be structured as the “Modified”.
- See the configurations below for both networks.
- You need to add the activation function “ReLU” behind each hidden layers
- Please write down the results, codes and your observations in the Word and upload to Moodle.

Original



Modified



Yellow : Input layer
Green: Hidden layer
Orange: Output layer

Example 2.2 Activation Function

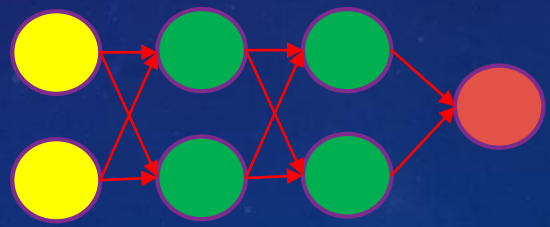
Please download 2-2_Activation_Function.zip and unzip it.

- Given an architecture with input size=2, hidden layer=2 and output size=1.
- Each layer is followed by the Leaky ReLU activation function.
- For more activation functions, please refer to the following link:

  https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity

- In this case, we can know how to establish a neural network in Pytorch; and how to use the activation function.

```
Feedforward(
  (input): Linear(in_features=2, out_features=2, bias=True)
  (Leakyrelu1): LeakyReLU(negative_slope=0.01)
  (fc1): Linear(in_features=2, out_features=2, bias=True)
  (Leakyrelu2): LeakyReLU(negative_slope=0.01)
  (fc2): Linear(in_features=2, out_features=2, bias=True)
  (Leakyrelu3): LeakyReLU(negative_slope=0.01)
  (output): Linear(in_features=2, out_features=1, bias=True)
)
```



Example 2.2 Activation Function

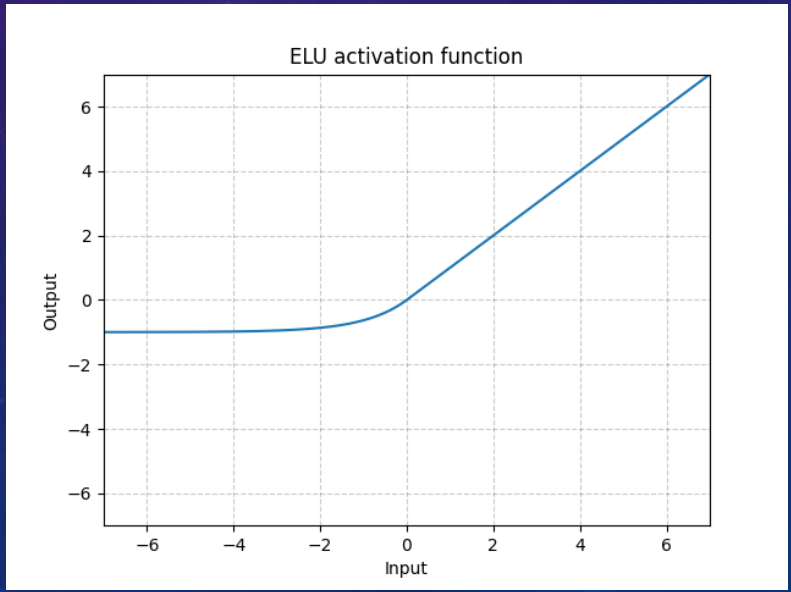
```
class Feedforward(torch.nn.Module):
        def __init__(self):
            super(Feedforward, self).__init__()
            self.input_size = 2
            self.hidden_size  = 2
            self.output_size  = 1
            self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
            self.leakyrelu1 = torch.nn.LeakyReLU()
            self.fc2 = torch.nn.Linear(self.hidden_size, self.hidden_size)
            self.leakyrelu2 = torch.nn.LeakyReLU()
            self.fc3 = torch.nn.Linear(self.hidden_size, self.output_size)
```

Change here as Leaky ReLU activation function

Exercise 2.2 Activation Function

Please download 2-2_Activation_Function.zip and unzip it.

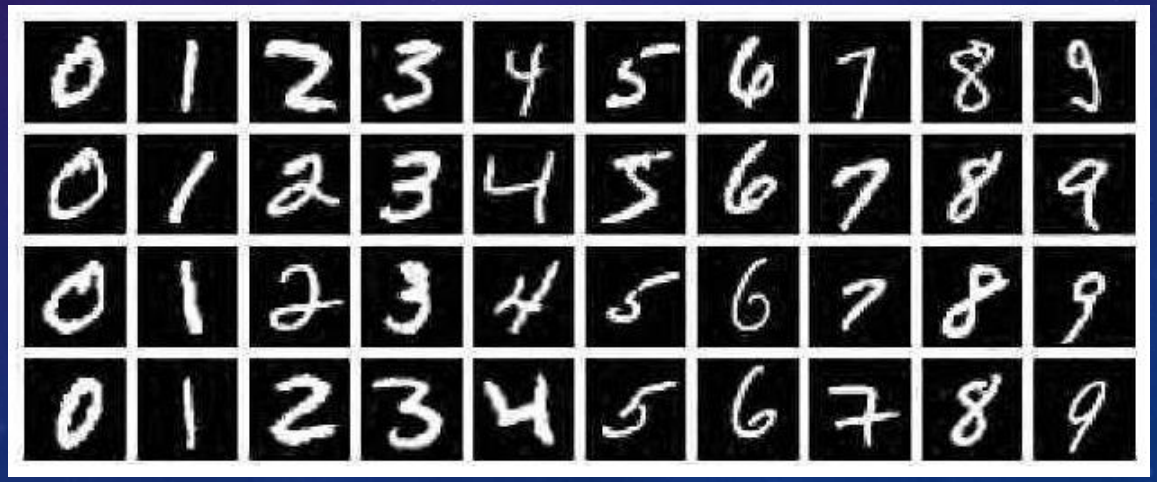
- Given an architecture with input size=2, hidden layer=2 and output size=1.
- Each layer is followed by the ELU activation function.
- Compare the results made by the ReLU and the ELU.
- Please write down the results, codes and your observations in the Word and upload to Moodle.



Example 2.3 Train A MLP Model

Please download 2-3_MLP_MNIST.zip and unzip it.

- Given an architecture with input size=28x28, hidden layer=3 and output size=10.
- The 1st hidden layer includes 500 neurons, the 2nd layer includes 250 neurons and the 3rd layer includes 125 neurons.
- Each layer is followed by the ReLU activation function.
- Use MNIST dataset as the input data: 90% for training and 10% for testing.
- Please follow the instructure to train a MLP base handwritten digits classifier.



Example 2.3 Train A MLP Model

```
def data_transform(x):
    x = np.array(x, dtype = 'float32') / 255
    x = (x - 0.5) /0.5
    x = x.reshape((-1, ))
    x = torch.from_numpy(x)
    return x
```

Normalize the image

Download the MNIST dataset for training and testing

```
trainset = mnist.MNIST('./dataset/mnist', train=True, transform=data_transform, download=True)
testset = mnist.MNIST('./dataset/mnist', train = False, transform=data_transform, download=True)
train_data = DataLoader(trainset, batch_size=64, shuffle=True)
test_data = DataLoader(testset, batch_size=128, shuffle=False)
```

Example 2.3 Train A MLP Model

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28*28, 500)
        self.fc2 = nn.Linear(500, 250)
        self.fc3 = nn.Linear(250, 125)
        self.fc4 = nn.Linear(125, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
mlp = MLP()
```

Define the MLP network

Use the relu activation function

Example 2.3 Train A MLP Model

```
Epoch:15, Steps:400, Losses:0.5459817051887512
Epoch:16, Steps:0, Losses:0.5937685966491699
Epoch:16, Steps:100, Losses:0.652214527130127
Epoch:16, Steps:200, Losses:0.5800161361694336
Epoch:16, Steps:300, Losses:0.5280714631080627
Epoch:16, Steps:400, Losses:0.5458921790122986
Epoch:17, Steps:0, Losses:0.5698919892311096
Epoch:17, Steps:100, Losses:0.5695023536682129
Epoch:17, Steps:200, Losses:0.550654947757721
Epoch:17, Steps:300, Losses:0.4742026329040527 3
Epoch:17, Steps:400, Losses:0.4731411635875702
Epoch:18, Steps:0, Losses:0.607618510723114
Epoch:18, Steps:100, Losses:0.5122652053833008
Epoch:18, Steps:200, Losses:0.4506227970123291
Epoch:18, Steps:300, Losses:0.6634547710418701
Epoch:18, Steps:400, Losses:0.4861069619655609
Epoch:19, Steps:0, Losses:0.4804354310035705 6
Epoch:19, Steps:100, Losses:0.521455049514770 5
Epoch:19, Steps:200, Losses:0.4508590698242187 5
Epoch:19, Steps:300, Losses:0.3965365886688232 4
Epoch:19, Steps:400, Losses:0.4753639698028564 5
Performance=86.86%
```

We can see the losses in each epoch, and the performance tested on the test set.

Exercise 2.3 Train A MLP Model

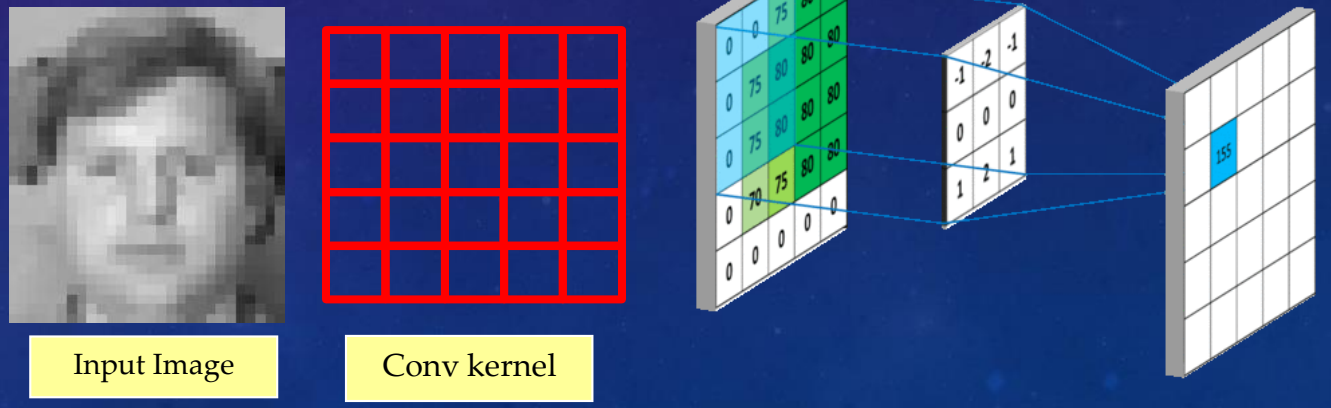
Please download 2-3_MLP_MNIST.zip and unzip it.

- Adjust the network architecture such as the number of hidden layer or the activation function.
- Use MNIST dataset as the input data: 90% for training and 10% for testing.
- Compare the result from example 2.3 to yours.
- Please write down the results, codes and your observations in the Word and upload to Moodle.

Example 2.4 Convolution Filter

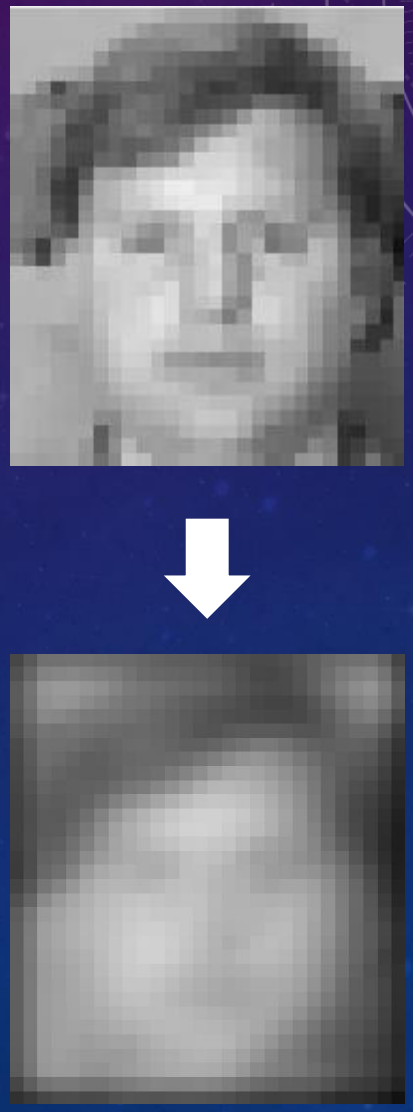
- Please download 2-4_Convolution_Example.zip and unzip it.
- Given a face image I_{img}, please generate F_{conv} , a 5x5 convolution filter with random coefficients in uniform distribution
- Use “same zeros padding” with unit stride.
- Compute the output by convolving I_{img} with F_{conv} and calculate the information loss between original image and the one after convolving.

```
KERNAL_SIZE = 5
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
Conv_Filter = np.random.rand(KERNAL_SIZE,KERNAL_SIZE)
#Conv_Filter =
np.random.normal(mean,std,(KERNAL_SIZE,KERNAL_SIZE))
#Normal distribution
Conv_Filter = Conv_Filter/np.sum(Conv_Filter)
img_F = img
```



Example 2.4 Convolution Filter

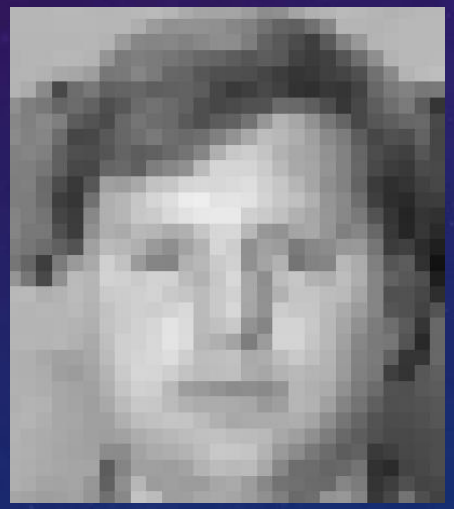
```
for h in range(int((H-KERNAL_SIZE)/STRIDE)+1):
    for w in range(int((W-KERNAL_SIZE)/STRIDE)+1):
        aa = img_F[h*STRIDE:h*STRIDE + (KERNAL_SIZE), w*STRIDE:w*STRIDE +
(KERNAL_SIZE)]*Conv_Filter

        new_feature[h,w] = np.sum(aa)

        img_S = img_F.astype(np.uint8)
        img_new = new_feature.astype(np.uint8)

        cv2.rectangle(img_S, (int(w*STRIDE), int(h*STRIDE)), (int((w*STRIDE +
KERNAL_SIZE)), int((h*STRIDE + KERNAL_SIZE))), (255, 0, 0), 1)

        cv2.namedWindow('Conv_process', cv2.WINDOW_NORMAL)
        cv2.resizeWindow("Conv_process", 300, 300)
        cv2.imshow('Conv_process',img_S)

        cv2.namedWindow('Conv_result', cv2.WINDOW_NORMAL)
        cv2.resizeWindow("Conv_result", 300, 300)
        cv2.imshow('Conv_result',img_new)

        cv2.waitKey(100)
```



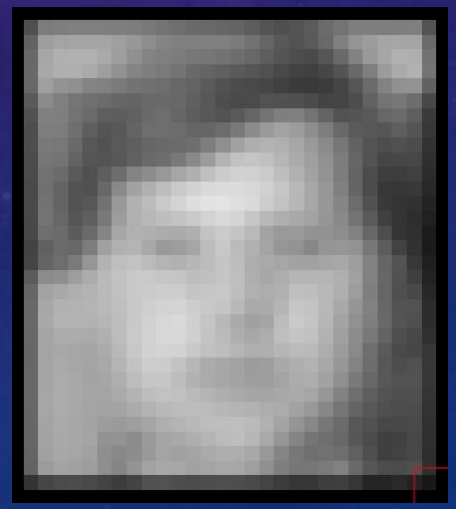
Example 2.4 Convolution Filter

```
%% Percentage of information loss
imgd = reshape(double(img), numel(img),1);
convimgd = reshape(double(Conv_Img), numel(Conv_Img),1);

Diff = sum(sum(abs(double(imgd)./norm(double(imgd)) - double(convimgd)./norm(double(convimgd)))));
PercnetageDiff = Diff/sum(sum(double(imgd)./norm(double(imgd)))*100);

fprintf('The information loss using %dx%d convolution kernel is %.6f%%\n\n', KERNAL_SIZE, KERNAL_SIZE, PercnetageDiff);
```

Original Image



Convolution Processing



Feature Map

Exercise 2.4 Design Convolution Filter

Please download 2-4_Convolution_Example.zip and design your own convolution filter with a

(1) 3x3 convolution filter

(2) 5x5 convolution filter

(3) 7x7 convolution filter

with random coefficients in normal distribution (std=1, mean=5), and compute the output by convolving I_{mg} with F_{conv}. Then, calculate the information loss with original image. Please write down the results, codes and your observations in the Word and upload to Moodle.

Change it in to normal distribution

```
KERNAL_SIZE = 3
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
Conv_Filter = np.random.rand(KERNAL_SIZE,KERNAL_SIZE)
#Conv_Filter = np.random.normal(mean,std,(KERNAL_SIZE,KERNAL_SIZE)) #Normal distribution
Conv_Filter = Conv_Filter/np.sum(Conv_Filter)
img_F = img
```

Example 2.5 Max Pooling

- Given a face image I_{img}, please generate F_{conv} , a 3x3 convolution filter with random coefficients in normal distribution (std = 0, mean=2) , and use “same zeros padding” with unit stride. Use “max pooling” with 3x3 kernel size and unit stride. Please compute the output by convolving I_{img} with F_{conv} and calculate the information loss between original image and the one after convolving.


```
img_new = pooling(img_new, 3, 1, 'max')
img_new = img_new.astype(np.uint8)
```

Use max pooling

Exercise 2.5 Average Pooling

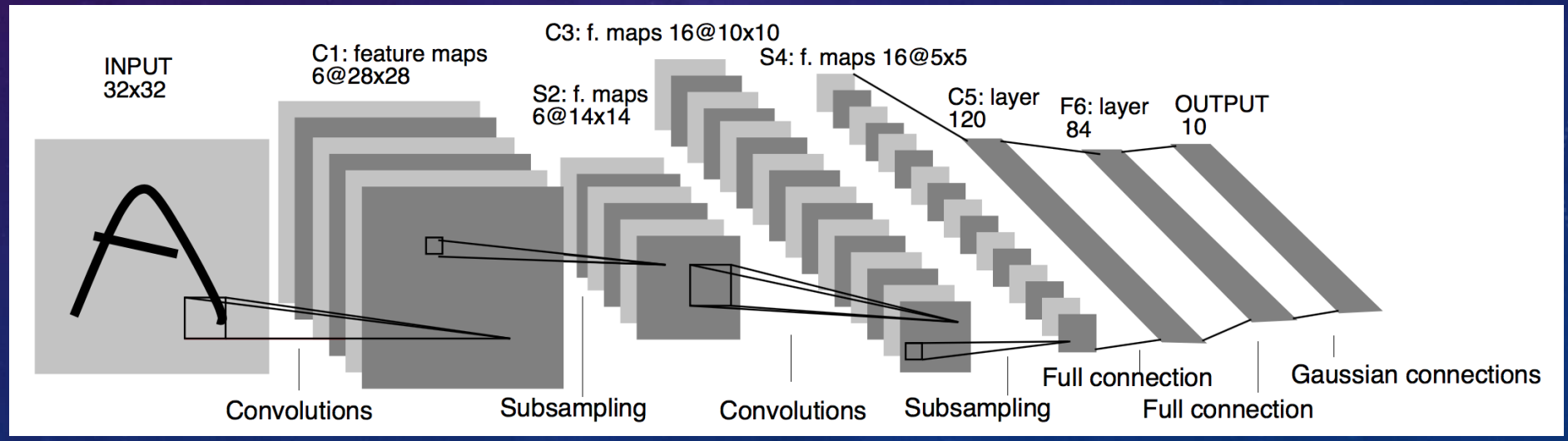
Please download 2-5_Pooling_Example.zip from the Moodle.

- Given a face image I_{img}, please generate F_{conv} , a 3x3 convolution filter with random coefficients in normal distribution (std = 0, mean=2).
- Use “same zeros padding” with unit stride to pool the feature maps.
- Use “average pooling” with 3x3 kernel size and unit stride.
- Compute the output by convolving I_{img} with F_{conv} .
- Please write down the results, codes and your observations in the Word and upload to Moodle.

Example 2.6 Train the LeNet Network

Please download 2-6_CNN_MNIST.zip and unzip it.

- Given a CNN base network architecture in the figure below.
- Consider the MNIST dataset as the input data: 90% for training and 10% for testing.
- Use the "Softmax with Cross Entropy" as objective function.
- Please follow the instruction to train the CNN base handwritten digits classifier.
- With the predicted results and its ground-truth, please compute the confusion matrix for each class.



Example 2.6 Train the LeNet Network

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, padding=2, stride=1)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.fc1 = nn.Linear(in_features=16*5*5, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=84)
        self.fc3 = nn.Linear(in_features=84, out_features=10)
```

→ Define the LeNet network

Example 2.6 Train the LeNet Network

In MNIST dataset, we have 10,000 test images with the ground-truth digits 1 to 10, where the GT denotes the ground-truth of each image, and Pred denotes the predicted digit number.

	GT_1	GT_2	GT_3	GT_4	GT_5	GT_6	GT_7	GT_8	GT_9	GT_10
Pred_1	938	0	15	5	1	19	18	5	7	11
Pred_2	0	1099	27	4	5	6	4	27	16	7
Pred_3	5	3	855	24	5	9	16	23	21	7
Pred_4	2	5	25	874	1	76	1	2	38	9
Pred_5	0	0	21	2	882	12	22	8	15	99
Pred_6	25	1	8	49	2	686	26	2	47	16
Pred_7	8	3	28	1	14	26	869	0	16	1
Pred_8	1	2	14	20	1	13	0	906	10	37
Pred_9	1	22	34	27	6	36	2	8	779	9
Pred_10	0	0	5	4	65	9	0	47	25	813

These numbers denotes the no. of data. (10,000 in total)

GT: ground truth
Pred: prediction

Example 2.6 Train the LeNet Network

To compute the confusion matrix, we take class "1" as the example.

	GT_1	GT_2	GT_3	GT_4	GT_5	GT_6	GT_7	GT_8	GT_9	GT_10
Pred_1	938	0	15	5	1	19	18	5	7	11
Pred_2	0	1099	27	4	5	6	4	27	16	7
Pred_3	5	3	855	24	5	9	16	23	21	7
Pred_4	2	5	25	874	1	76	1	2	38	9
Pred_5	0	0	21	2	882	12	22	8	15	99
Pred_6	25	1	8	49	2	686	26	2	47	16
Pred_7	8	3	28	1	14	26	869	0	16	1
Pred_8	1	2	14	20	1	13	0	906	10	37
Pred_9	1	22	34	27	6	36	2	8	779	9
Pred_10	0	0	5	4	65	9	0	47	25	813

TP = 938 → Number of True Positive

FP = 0 + 15 + 5 + … + 11 = 81

TN = 1099 + 27 + 4 + … + 25 + 813 = 8939

FN = 0 + 5 + 2 + … + 1 + 0 = 42

$$Precision = \frac{TP}{TP + FP} = \frac{938}{938 + 81} = 0.92$$

$$Recall = \frac{TP}{TP + FN} = \frac{938}{938 + 42} = 0.957$$

Example 2.6 Train the LeNet Network

Let's use the class "4" as the example.

	GT_1	GT_2	GT_3	GT_4	GT_5	GT_6	GT_7	GT_8	GT_9	GT_10
Pred_1	938	0	15	5	1	19	18	5	7	11
Pred_2	0	1099	27	4	5	6	4	27	16	7
Pred_3	5	3	855	24	5	9	16	23	21	7
Pred_4	2	5	25	874	1	76	1	2	38	9
Pred_5	0	0	21	2	882	12	22	8	15	99
Pred_6	25	1	8	49	2	686	26	2	47	16
Pred_7	8	3	28	1	14	26	869	0	16	1
Pred_8	1	2	14	20	1	13	0	906	10	37
Pred_9	1	22	34	27	6	36	2	8	779	9
Pred_10	0	0	5	4	65	9	0	47	25	813

TP = 874 → Number of True Positive

FP = 2 + 5 + 25 + … + 38 = 159

TN = 938 + 0 + 15 + … + 25 + 813 = 8831

FN = 5 + 4 + 24 + … + 27 + 4 = 136

$$Precision = \frac{TP}{TP + FP} = \frac{874}{874 + 159} = 0.846$$

$$Recall = \frac{TP}{TP + FN} = \frac{874}{874 + 136} = 0.865$$

Example 2.6 Train the LeNet Network

TP, FP, TN, FN, Precision and Recall distributions in each class.

	No. TP	No. FP	No. TN	No. FN	Precision	Recall
Class 1	938	81	8939	42	0.921	0.957
Class 2	1099	96	8769	36	0.920	0.968
Class 3	855	113	8855	177	0.883	0.828
Class 4	874	159	8831	136	0.846	0.865
Class 5	882	179	8839	100	0.831	0.898
Class 6	686	176	8932	206	0.796	0.769
Class 7	869	97	8945	89	0.900	0.907
Class 8	906	98	8874	122	0.902	0.881
Class 9	779	145	8881	195	0.843	0.800
Class 10	813	155	8836	196	0.840	0.806

Exercise 2.6 Train the LeNet Network

Please download 2-6_CNN_MNIST.zip and unzip it.

- Given a CNN base network architecture in the figure below.
- Adjust the CNN layer, e.g. kernel size, input and output channel.
- Consider the MNIST dataset as the input data: 90% for training and 10% for testing.
- With the predicted results and its ground-truth, please compute the confusion matrix for each class.
- Please write down the results, codes and your observations in the Word and upload to Moodle.