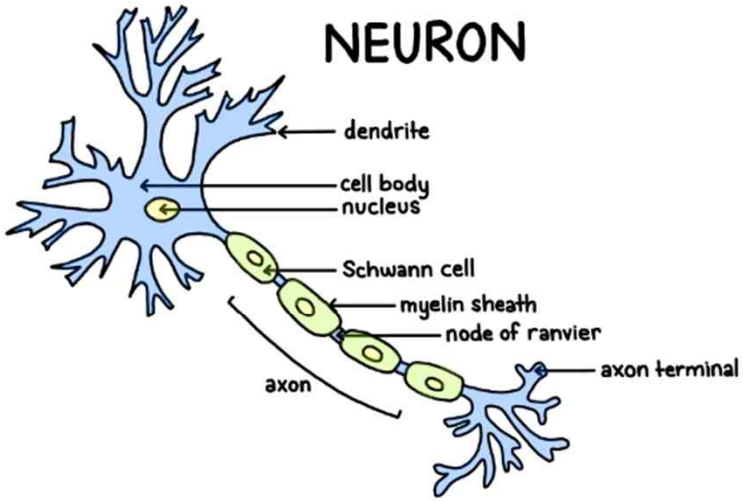
# Week 3

ML/DL General 안태영

# Lecture 08: 딥러닝 기본 개념

**Brain & Neuron**

→ 복잡하다

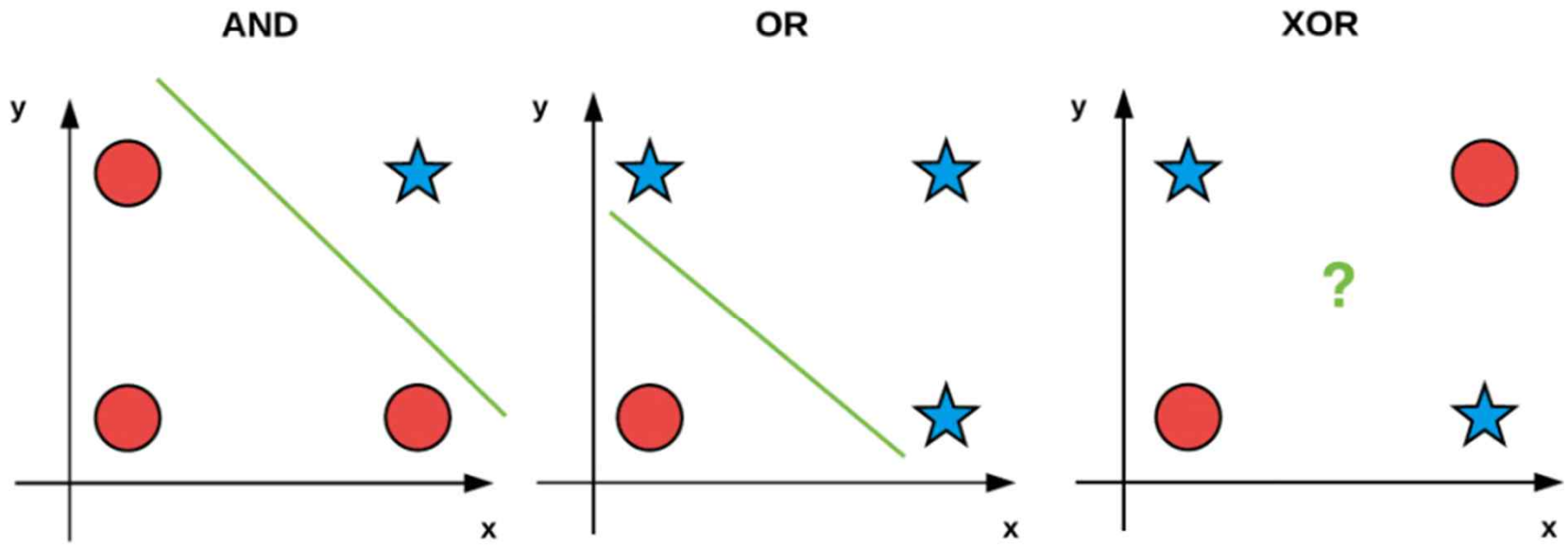



- axon from antoher nueron → $x_0$
- synapse → $w_0$
- cell body → $\sum(w_i x_i + b_i)$ and **activation function**
- output axon → $f(\sum(w_i x_i + b_i))$ and **sigmoid function**

# Lecture 08: 딥러닝 기본 개념

**Linear separation**

- AND / OR can linearly separable
- XOR can not linearly separable



# Lecture 08: 딥러닝 기본 개념

**Perceptron**

- Made by Morkin Minsky
- MLP(Multiple perceptron ) can help calculate XOR, but we can not find the weigth of MLP

**Backprogation**

- weigth 값 수정을 앞에서 차례대로 바꿔서 하는 것이 아니고, 결과 값에 error가 생기면 wieght에 수정을 해줘서 찾는 방식이다

**Convolutional Neural Network, CNN**

- 부분적으로 문자나 사진을 나눠서 인식하여 output을 사용하여 나타내는 것

**Big Problem**

- Backpropagation just did not work well for normal neural nets
- other risng machine learning Algrithm such as SUM, RandomForest is better than CNN

**Back Through**

- If weights are initialized in a clear way
- Rebranding to **Deep Learning**

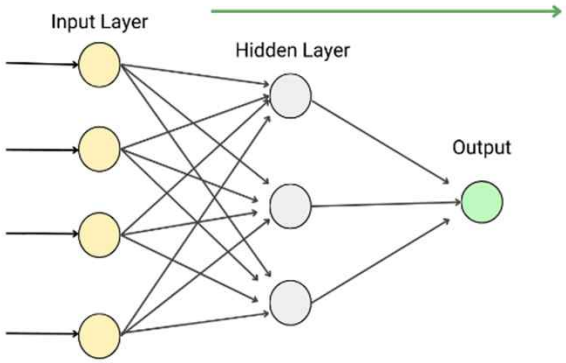
# Lecture 09: XOR 문제 딥러닝으로 풀기

**XOR**

- One logistic Regression is unit can not seperate
- Exclusive or gate
- $W_1 x_1 + b_1, W_1 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, b_1 = -8$
- $W_2 x_2 + b_2, W_2 = \begin{bmatrix} -7 \\ -7 \end{bmatrix}, b_2 = 3$
- $W_3 x_3 + b_3, W_3 = \begin{bmatrix} -11 \\ -11 \end{bmatrix}, b_3 = 6$
- Can create XOR gate

# Lecture 09: XOR 문제 딥러닝으로 풀기

**Forward Propagation**




- 여러가지 gate를 만들어서 여러층으로 만든다

**Neural Network**

- Forward propagation에서 여러층으로 된 Input layer를 행렬 형태로 변환 시킨 네트워크
- $k(X) = sigmoid(XW_1 + B_1)$
- $\overline{Y} = H(X) = sigmoid(k(X)W_2 + b_2)$

**Backpropagation(chain rule)**

- How can we learn W and B from data?
- 중간에 어떤 데이터들이 있더라도 처음 값과 마지막 값만 있으면 chain rule을 이용하여 미분할 수 있다

# Lab 09-1: Neural Net for XOR

```
x_data = [[0, 0],
          [0, 1],
          [1, 0],
          [1, 1]]
y_data = [[0],
          [1],
          [1],
          [0]]

dataset = tf.data.Dataset.from_tensor_slices((x_data, y_data)).batch(len(x_data))

#Data for feature and label
def preprocess_data(features, labels):
    features = tf.cast(features, tf.float32)
    labels = tf.cast(labels, tf.float32)
    return features, labels

#Weight and Bias for hypothesis
W = tf.Variable(tf.zeros((2,1)), name='weight')
b = tf.Variable(tf.zeros((1,)), name='bias')
print("W = {}, B = {}".format(W.numpy(), b.numpy()))

#Hypothesis for sigmoid
def logistic_regression(features):
    hypothesis  = tf.divide(1., 1. + tf.exp(tf.matmul(features, W) + b))
    return hypothesis

#Cost Function for loss
def loss_fn(hypothesis, features, labels):
    cost = -tf.reduce_mean(labels * tf.math.log(logistic_regression(features)) + (1 - labels) * tf.math.log
    return cost
```

# Lab 09-1: Neural Net for XOR

```
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

#Accuracy Fuction
def accuracy_fn(hypothesis, labels):
    predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
    accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, labels), dtype=tf.float32))
    return accuracy

#Gradient Descent and Fitting
def grad(hypothesis, features, labels):
    with tf.GradientTape() as tape:
        loss_value = loss_fn(logistic_regression(features),features,labels)
    return tape.gradient(loss_value, [W,b])

EPOCHS = 1001

for step in range(EPOCHS):
    for features, labels  in dataset:
        features, labels = preprocess_data(features, labels)
        grads = grad(logistic_regression(features), features, labels)
        optimizer.apply_gradients(grads_and_vars=zip(grads,[W,b]))
        if step % 100 == 0:
            print("Iter: {}, Loss: {:.4f}".format(step, loss_fn(logistic_regression(features),features,labe
print("W = {}, B = {}".format(W.numpy(), b.numpy()))
x_data, y_data = preprocess_data(x_data, y_data)
test_acc = accuracy_fn(logistic_regression(x_data),y_data)
print("Testset Accuracy: {:.4f}".format(test_acc))
```

# Lab 09-1: Neural Net for XOR

```
dataset = tf.data.Dataset.from_tensor_slices((x_data, y_data)).batch(len(x_data))
nb_classes = 10

class wide_deep_nn():
    def __init__(self, nb_classes):
        super(wide_deep_nn, self).__init__()

        self.W1 = tf.Variable(tf.random.normal((2, nb_classes)), name='weight1')
        self.b1 = tf.Variable(tf.random.normal((nb_classes,)), name='bias1')

        self.W2 = tf.Variable(tf.random.normal((nb_classes, nb_classes)), name='weight2')
        self.b2 = tf.Variable(tf.random.normal((nb_classes,)), name='bias2')

        self.W3 = tf.Variable(tf.random.normal((nb_classes, nb_classes)), name='weight3')
        self.b3 = tf.Variable(tf.random.normal((nb_classes,)), name='bias3')

        self.W4 = tf.Variable(tf.random.normal((nb_classes, 1)), name='weight4')
        self.b4 = tf.Variable(tf.random.normal((1,)), name='bias4')

        self.variables = [self.W1,self.b1,self.W2,self.b2,self.W3,self.b3,self.W4,self.b4]

    def preprocess_data(self, features, labels):
        features = tf.cast(features, tf.float32)
        labels = tf.cast(labels, tf.float32)
        return features, labels

    def deep_nn(self, features):
        layer1 = tf.sigmoid(tf.matmul(features, self.W1) + self.b1)
        layer2 = tf.sigmoid(tf.matmul(layer1, self.W2) + self.b2)
        layer3 = tf.sigmoid(tf.matmul(layer2, self.W3) + self.b3)
        hypothesis = tf.sigmoid(tf.matmul(layer3, self.W4) + self.b4)
        return hypothesis

```

```
    def loss_fn(self, hypothesis, features, labels):
        cost = -tf.reduce_mean(labels * tf.math.log(hypothesis) + (1 - labels) * tf.math.log(1 - hypothesis
        return cost

    def accuracy_fn(self, hypothesis, labels):
        predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
        accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, labels), dtype=tf.float32))
        return accuracy

    def grad(self, hypothesis, features, labels):
        with tf.GradientTape() as tape:
            loss_value = self.loss_fn(self.deep_nn(features),features,labels)
        return tape.gradient(loss_value,self.variables)

    def fit(self, dataset, EPOCHS=20000, verbose=500):
        optimizer =  tf.keras.optimizers.SGD(learning_rate=0.01)
        for step in range(EPOCHS):
            for features, labels  in dataset:
                features, labels = self.preprocess_data(features, labels)
                grads = self.grad(self.deep_nn(features), features, labels)
                optimizer.apply_gradients(grads_and_vars=zip(grads, self.variables))
                if step % verbose == 0:
                    print("Iter: {}, Loss: {:.4f}".format(step, self.loss_fn(self.deep_nn(features),feature

    def test_model(self,x_data, y_data):
        x_data, y_data = self.preprocess_data(x_data, y_data)
        test_acc = self.accuracy_fn(self.deep_nn(x_data),y_data)
        print("Testset Accuracy: {:.4f}".format(test_acc))
```

# Lab 09-2: TensorBoard(Neural Net for XOR)

```
log_path = "./logs/xor"
writer = tf.summary.create_file_writer(log_path)
```

```
W1 = tf.Variable(tf.random.normal((2, 10)), name='weight1')
b1 = tf.Variable(tf.random.normal((10,)), name='bias1')

W2 = tf.Variable(tf.random.normal((10, 10)), name='weight2')
b2 = tf.Variable(tf.random.normal((10,)), name='bias2')

W3 = tf.Variable(tf.random.normal((10, 10)), name='weight3')
b3 = tf.Variable(tf.random.normal((10,)), name='bias3')

W4 = tf.Variable(tf.random.normal((10, 1)), name='weight4')
b4 = tf.Variable(tf.random.normal((1,)), name='bias4')

def neural_net(features, step):
    layer1 = tf.sigmoid(tf.matmul(features, W1) + b1)
    layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)
    layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)
    hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)
```

```
def neural_net(features, step):
    layer1 = tf.sigmoid(tf.matmul(features, W1) + b1)
    layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)
    layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)
    hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)

    with writer.as_default():
        tf.summary.histogram("weights1", W1, step=step)
        tf.summary.histogram("biases1", b1, step=step)
        tf.summary.histogram("layer1", layer1, step=step)

        tf.summary.histogram("weights2", W2, step=step)
        tf.summary.histogram("biases2", b2, step=step)
        tf.summary.histogram("layer2", layer2, step=step)

        tf.summary.histogram("weights3", W3, step=step)
        tf.summary.histogram("biases3", b3, step=step)
        tf.summary.histogram("layer3", layer3, step=step)

        tf.summary.histogram("weights4", W4, step=step)
        tf.summary.histogram("biases4", b4, step=step)
        tf.summary.histogram("hypothesis", hypothesis, step=step)
    return hypothesis
```

# Lab 09-2: TensorBoard(Neural Net for XOR)

```
EPOCHS = 3000

for step in range(EPOCHS):
    for features, labels  in dataset:
        features, labels = preprocess_data(features, labels)
        grads = grad(neural_net(features, step), features, labels, step)
        optimizer.apply_gradients(grads_and_vars=zip(grads,[W1, W2, W3, W4, b1, b2, b3, b4]))
        if step % 50 == 0:
            loss_value = loss_fn(neural_net(features, step),labels)
            print("Iter: {}, Loss: {:.4f}".format(step, loss_value))
x_data, y_data = preprocess_data(x_data, y_data)
test_acc = accuracy_fn(neural_net(x_data, step),y_data)
print("Testset Accuracy: {:.4f}".format(test_acc))
```

**Tensorboard에 있는 값을 jypyter notebook에서 load 하기**

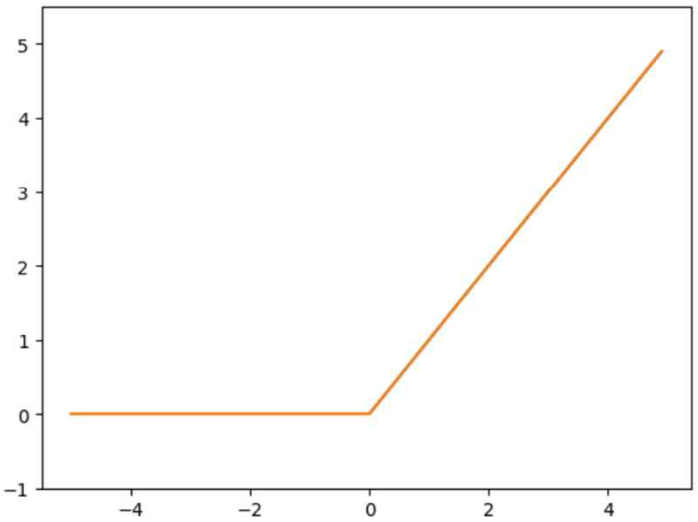
```
# Load the TensorBoard notebook extension
%load_ext tensorboard
%tensorboard --logdir logs/xor
```

# Lab 10-1: Sigmoid 보다 ReLU가 더 좋아

**Problem of sigmoid**

- Vanishing Gradient
  → layer가 많은 network에서 sigmoid 함수가 여러 개 있으면, 기울기가 0이 되는 지점이 많아져 gradient가 소실 되는 현상

**ReLU Function**




- $f(x) = max(0, x)$
- tf.keras.activations에 여러가지 activation 함수가 많다
- sigmoid, tanh, relu, elu, selu등 많다

# Lab 10-1: Sigmoid 보다 ReLU가 더 좋아

```
def create_model_function(label_dim, mode) :
    weight_init = tf.keras.initializers.RandomNormal()

    model = tf.keras.Sequential()
    model.add(flatten())

    if mode == 1:
        for i in range(2) :
            model.add(dense(256, weight_init))
            model.add(sigmoid())
    elif mode == 2:
        for i in range(2) :
            model.add(dense(256, weight_init))
            model.add(relu())

    model.add(dense(label_dim, weight_init))

    return model


```

# Lab 10-1: Sigmoid 보다 ReLU가 더 좋아

**Checkpoint Function**

- 특정 epoch 마다 model의 weight와 bias를 저장해주는 function

```
[ ]
def load(model, checkpoint_dir):
    print(" [*] Reading checkpoints...")

    ckpt = tf.train.get_checkpoint_state(checkpoint_dir)
    if ckpt :
        ckpt_name = os.path.basename(ckpt.model_checkpoint_path)
        checkpoint = tf.train.Checkpoint(dnn=model)
        checkpoint.restore(save_path=os.path.join(checkpoint_dir, ckpt_name))
        counter = int(ckpt_name.split('-')[1])
        print(" [*] Success to read {}".format(ckpt_name))
        return True, counter
    else:
        print(" [*] Failed to find a checkpoint")
        return False, 0

def check_folder(dir):
    if not os.path.exists(dir):
        os.makedirs(dir)
    return dir
```

→ sigmoid는 0.9247

→ relu는 0.9356

**Experiments(parameters)**

```
]
""" dataset """
train_x, train_y, test_x, test_y = load_mnist()

""" parameters """
learning_rate = 0.001
batch_size = 128

training_epochs = 1
training_iterations = len(train_x) // batch_size

label_dim = 10

train_flag = True

""" Graph Input using Dataset API """
train_dataset = tf.data.Dataset.from_tensor_slices((train_x, train_y)).₩
    shuffle(buffer_size=100000).₩
    prefetch(buffer_size=batch_size).₩
    batch(batch_size, drop_remainder=True)

test_dataset = tf.data.Dataset.from_tensor_slices((test_x, test_y)).₩
    shuffle(buffer_size=100000).₩
    prefetch(buffer_size=len(test_x)).₩
    batch(len(test_x))
```

# Lab 10-2: Weight Initialization

**Xavier Initialization**

- Local minimum에 빠질 수도 있고 saddle point에빠질 위험도 있다
- 기존의 RandomNormalization Initialization에서는평균이 0 분산이 1인 초기 weight 값이었다
- Xavier은 평균은 0이다

- $Variance = \frac{2}{Channel\_in + Channel\_out}$
- ($Channel\_in$ : input으로 들어가는 채널의 개수, $Channel\_out$ : $output$으로들어가는채널의개수)

**He Initialization**

- ReLU함수에 특화된 weight Initialization 기법
- Xavier 분산에 2를 곱한 값을 취한다

# Lab 10-2: Weight Initialization

**Code**

- weight_init = tf.keras.initializers. 뒷값을 바꿔주면 된다
- 기존 : RandomNormal()
- Xavier : glorot_uniform()
- He : hr_uniform()

```
[ ]
```

```python
def create_model_function(label_dim) :
    weight_init = tf.keras.initializers.glorot_uniform()

    model = tf.keras.Sequential()
    model.add(flatten())

    for i in range(2) :
        model.add(dense(256, weight_init))
        model.add(relu())

    model.add(dense(label_dim, weight_init))

    return model
```

# Lab 10-3: Dropout

> **Dropout**
>
> - underfitting과 overfitting을 막아주는 regularization
> - node 몇개를 끄고 학습시키는 방법
> - 어떤 node를 끄는가는 랜덤으로 설

```
[ ]
```

```python
def dropout(rate) :
    return tf.keras.layers.Dropout(rate)

class create_model_class(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model_class, self).__init__()
        weight_init = tf.keras.initializers.glorot_uniform()

        self.model = tf.keras.Sequential()
        self.model.add(flatten())

        for i in range(4):
            self.model.add(dense(512, weight_init))
            self.model.add(relu())
            ###이부분을 추가해준다
            self.model.add(dropout(rate=0.5))

        self.model.add(dense(label_dim, weight_init))

    def call(self, x, training=None, mask=None):

        x = self.model(x)

        return x
```

# Lab 10-4: Batch Normalization

**Internal Covariate Shift**

- data가 layer를 지나가면서 distribution이 이상하게 생기는 현상

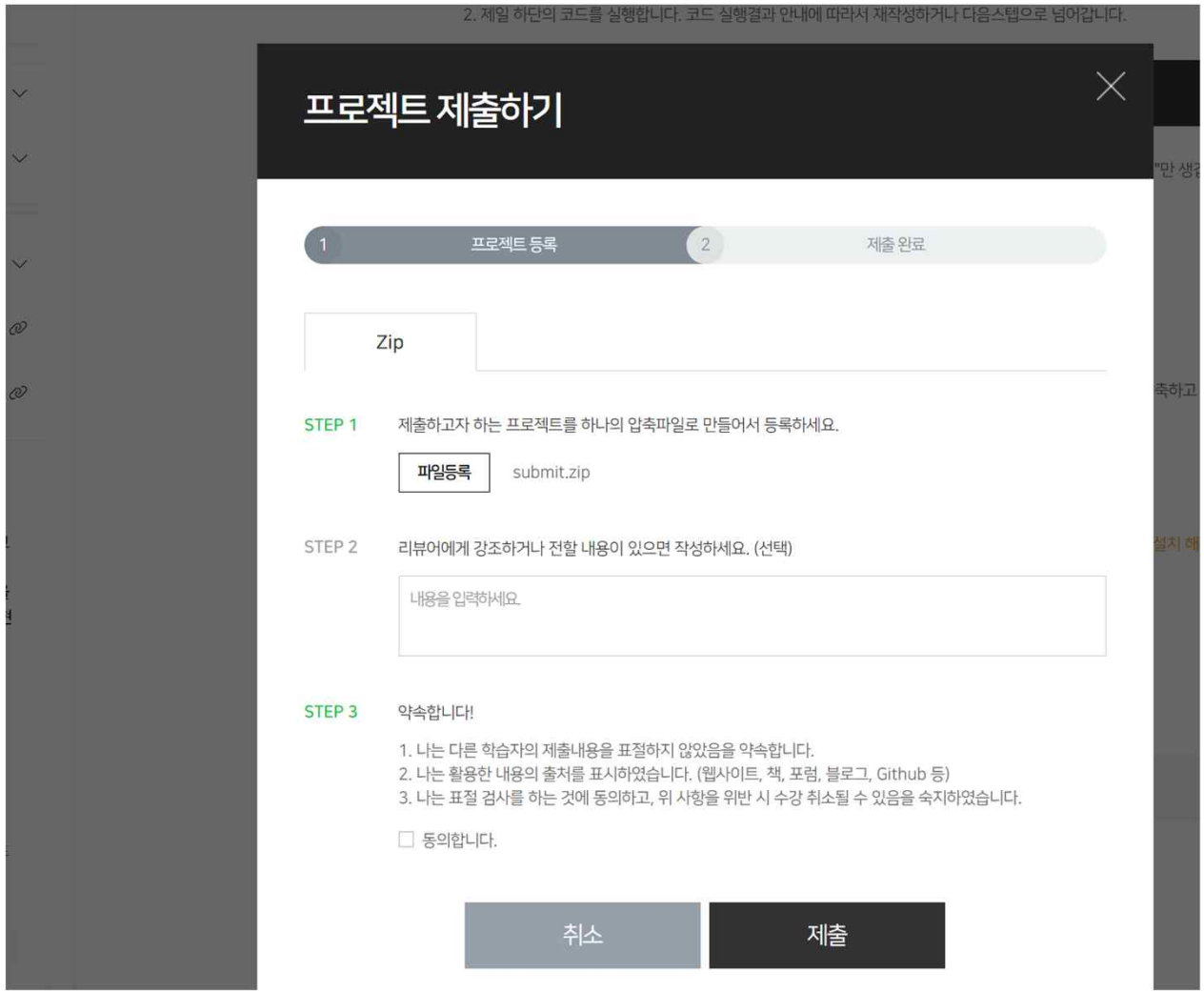
**Batch Normalization**

- internal Covariate shift를 막기 위해 이용한다
- $\bar{x} = \frac{x-\mu_B}{\sqrt{\sigma_B^2+\epsilon}}$ ($x$ : input, $\mu_B$ : Batch들의 평균, $\sqrt{\sigma_B^2+\epsilon}$ : Batch들의 분산
- $\hat{x} = \gamma\bar{x} + \beta$
- 감마와 베타로 어떤 학습이 되는 파라미터들을 이용해서 $\hat{x}$을 다음 input으로 이용한다
- 보통 layer -> norm -> activation 순으로 한다

# Lab 10-4: Batch Normalization

```python
def batch_norm() :
    return tf.keras.layers.BatchNormalization()
```

```python
class create_model_class(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model_class, self).__init__()
        weight_init = tf.keras.initializers.glorot_uniform()

        self.model = tf.keras.Sequential()
        self.model.add(flatten())

        for i in range(4):
            self.model.add(dense(512, weight_init))
            ###이 부분이 바뀐다
            self.model.add(batch_norm())
            self.model.add(relu())

        self.model.add(dense(label_dim, weight_init))

    def call(self, x, training=None, mask=None):

        x = self.model(x)

        return x
```


2. 제일 하단의 코드를 실행합니다. 코드 실행결과 안내에 따라서 재작성하거나 다음스텝으로 넘어갑니다.
프로젝트 제출하기
1
프로젝트 등록
2
제출 완료
Zip
STEP 1
제출하고자 하는 프로젝트를 하나의 압축파일로 만들어서 등록하세요.
파일등록
submit.zip
STEP 2
리뷰어에게 강조하거나 전할 내용이 있으면 작성하세요. (선택)
내용을 입력하세요.
STEP 3
약속합니다!
1. 나는 다른 학습자의 제출내용을 표절하지 않았음을 약속합니다.
2. 나는 활용한 내용의 출처를 표시하였습니다. (웹사이트, 책, 포럼, 블로그, Github 등)
3. 나는 표절 검사를 하는 것에 동의하고, 위 사항을 위반 시 수강 취소될 수 있음을 숙지하였습니다.
동의합니다.
취소
제출