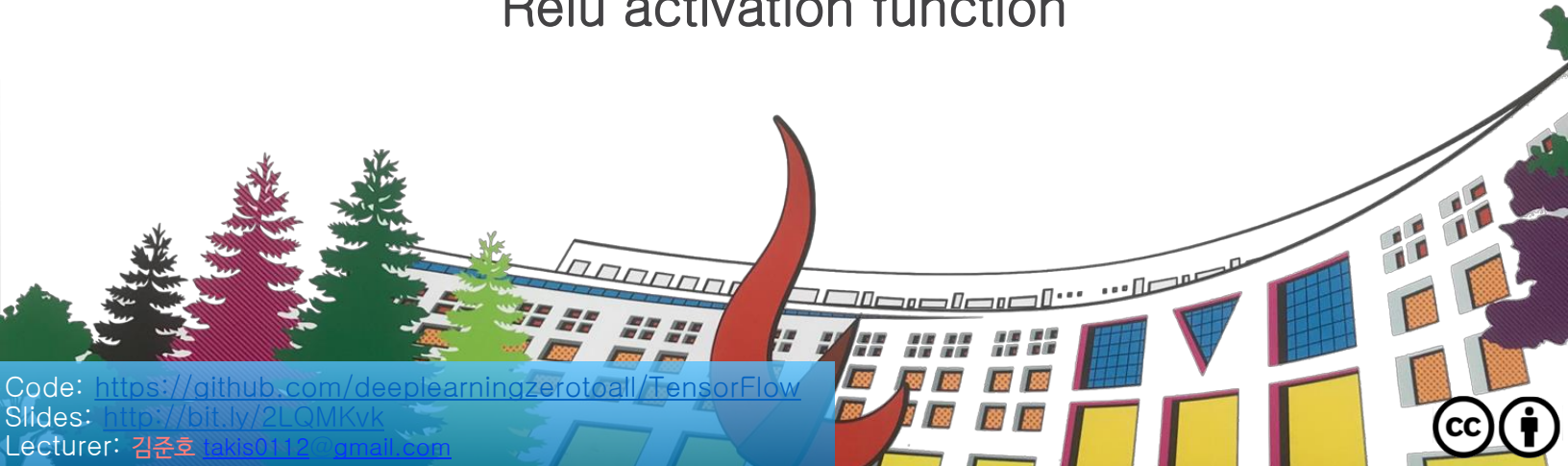
ML/DL for Everyone Season2

with TensorFlow

Lab10-1
Relu activation function



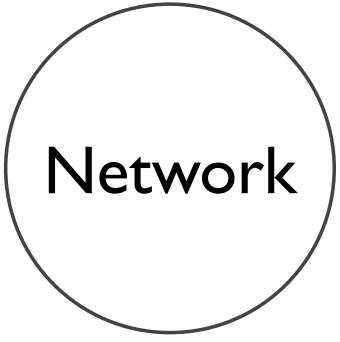
Code: https://github.com/deeplearningzerotoall/TensorFlow
Slides: http://bit.ly/2LQMKvk
Lecturer: 김준호 takis0112@gmail.com

Lab10-1: Relu activation function

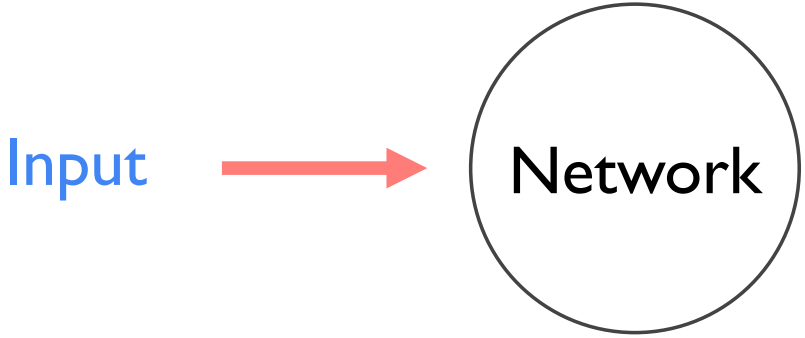
- Problem of Sigmoid sigmoid의 문제점
- Why Relu ?
- Code
  - ○ load dataset
  - ○ create network
  - ○ define loss function
  - ○ experiments
    - parameters
    - model
    - eager mode
- What's Next

Problem of Sigmoid

Problem of Sigmoid

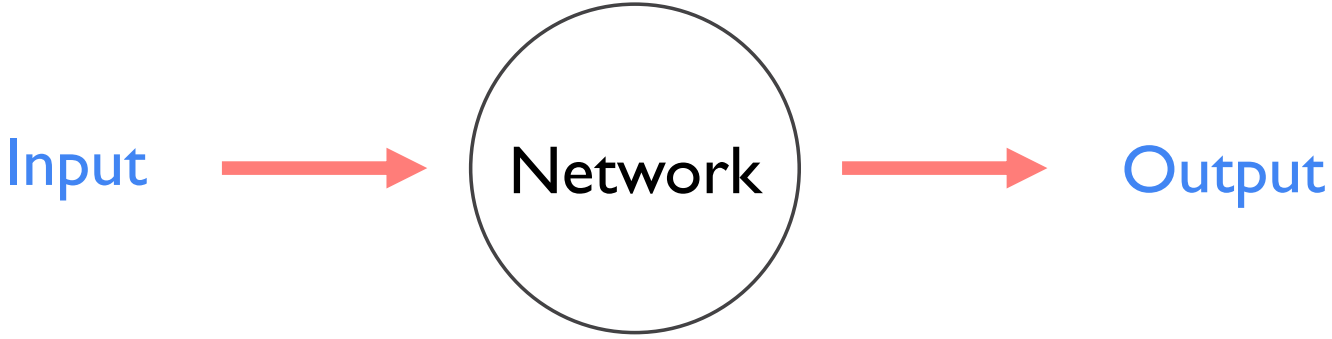


Problem of Sigmoid

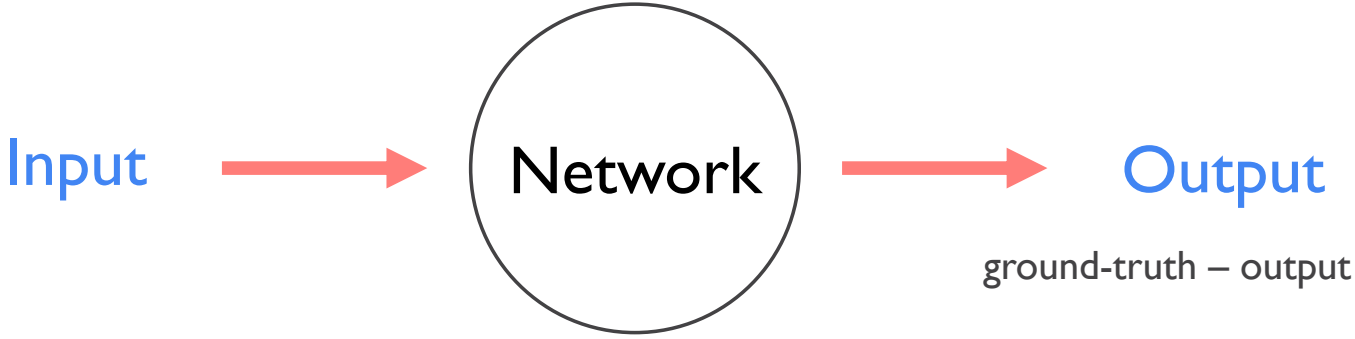


NEURAL NETWORK

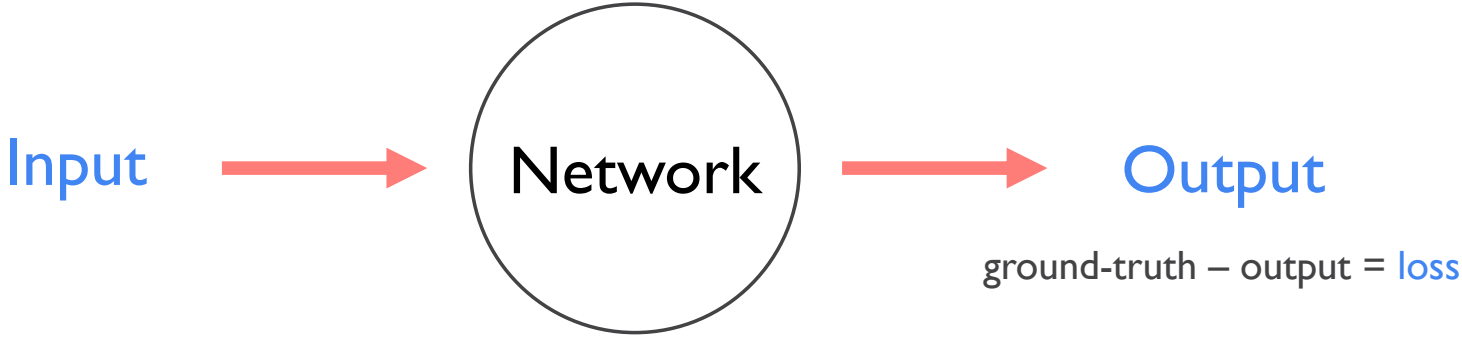
Problem of Sigmoid



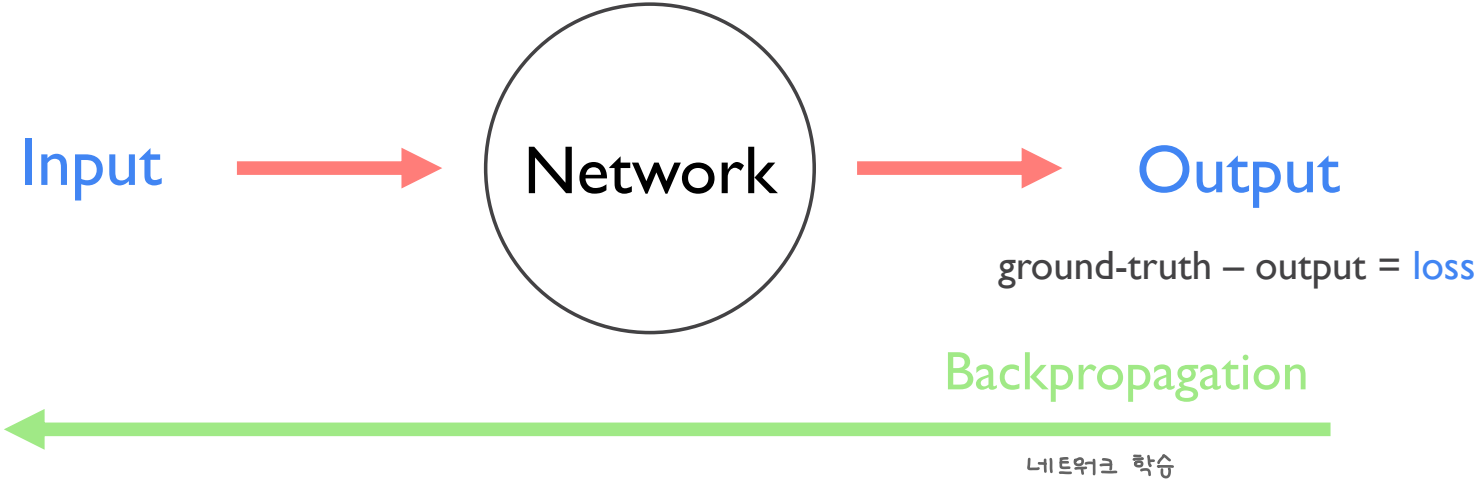
Problem of Sigmoid



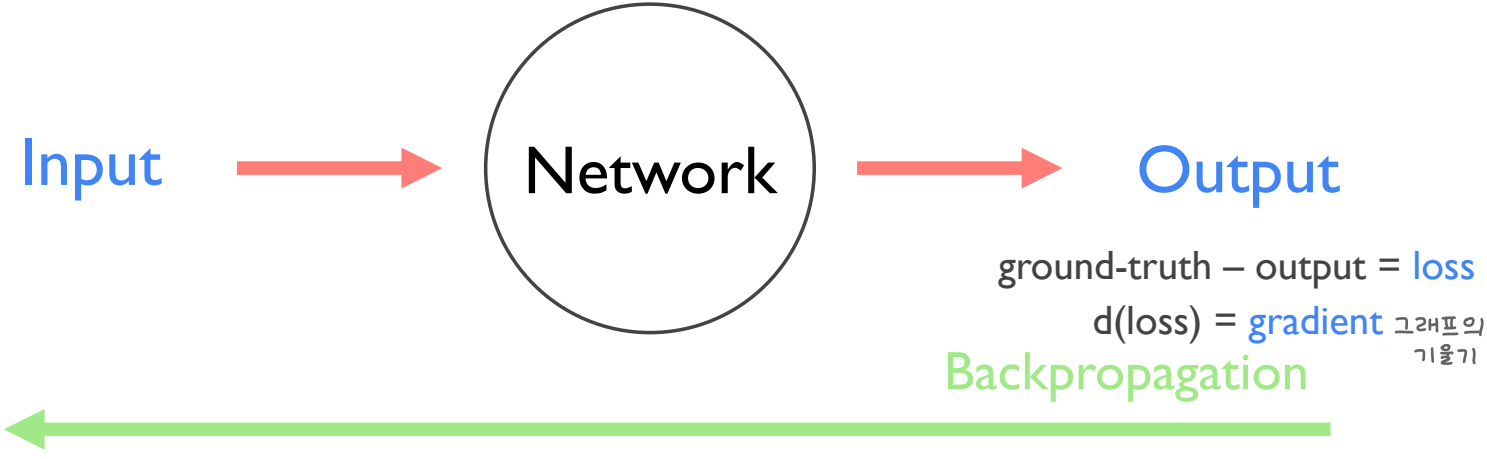
Problem of Sigmoid



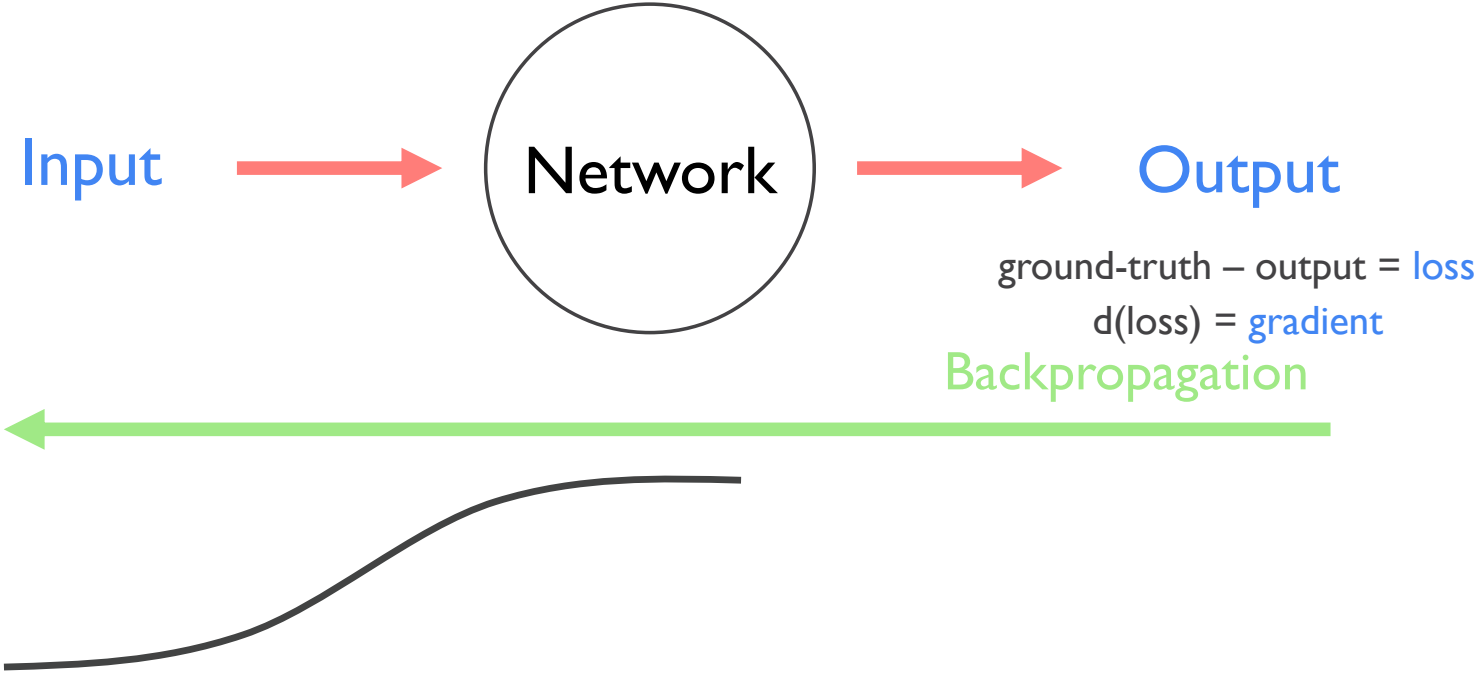
Problem of Sigmoid



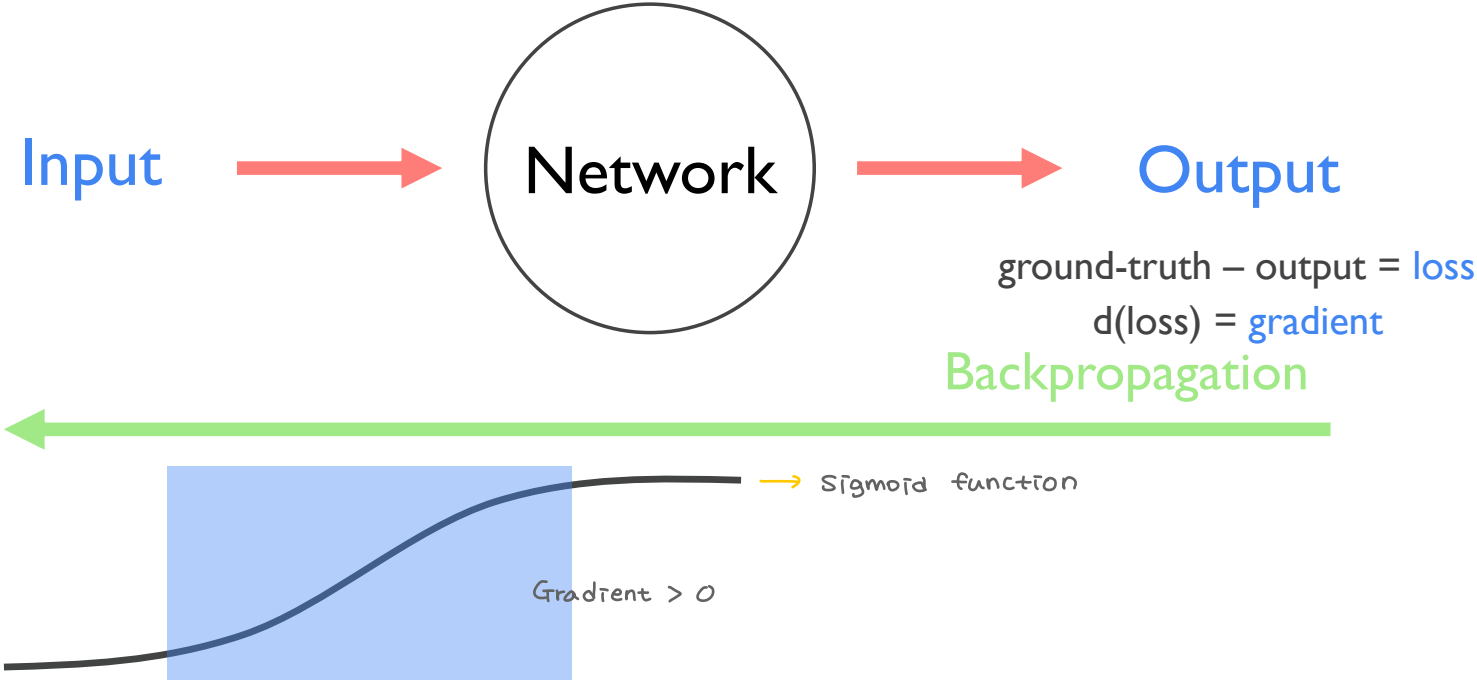
Problem of Sigmoid



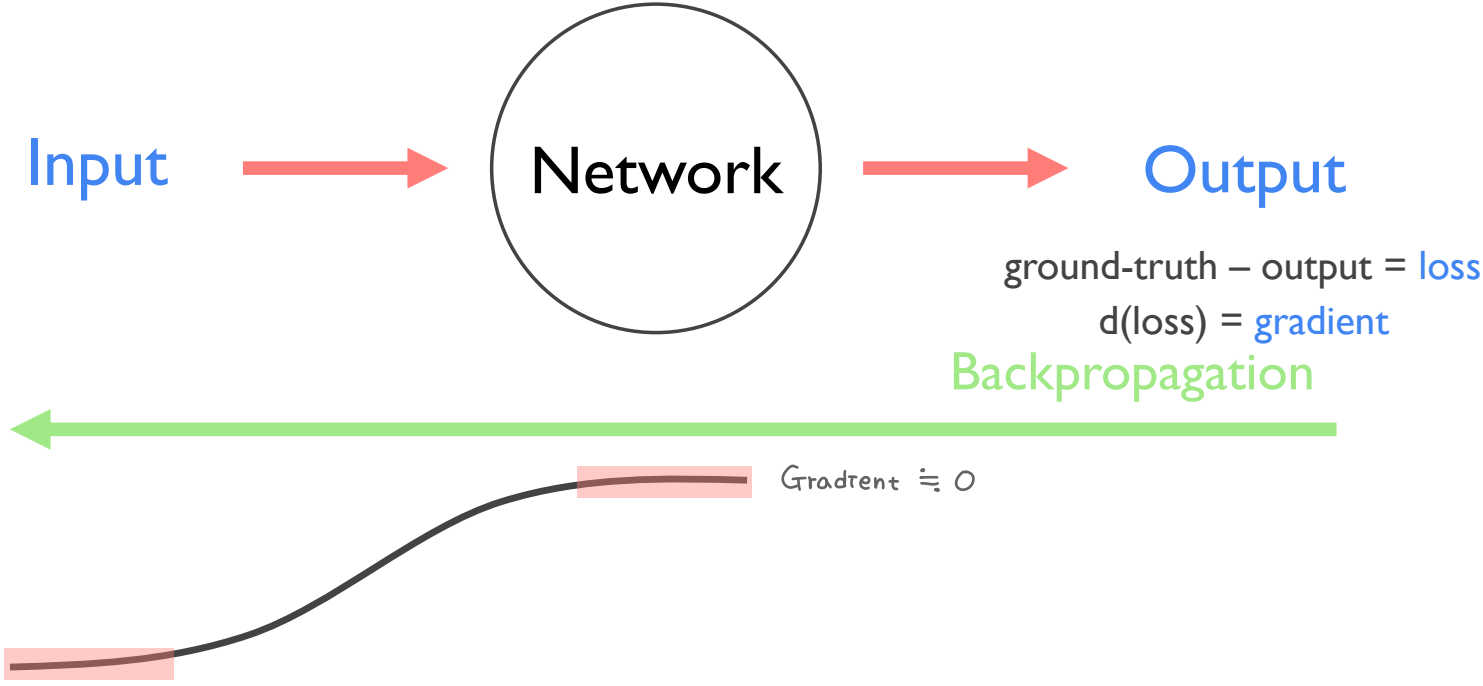
Problem of Sigmoid



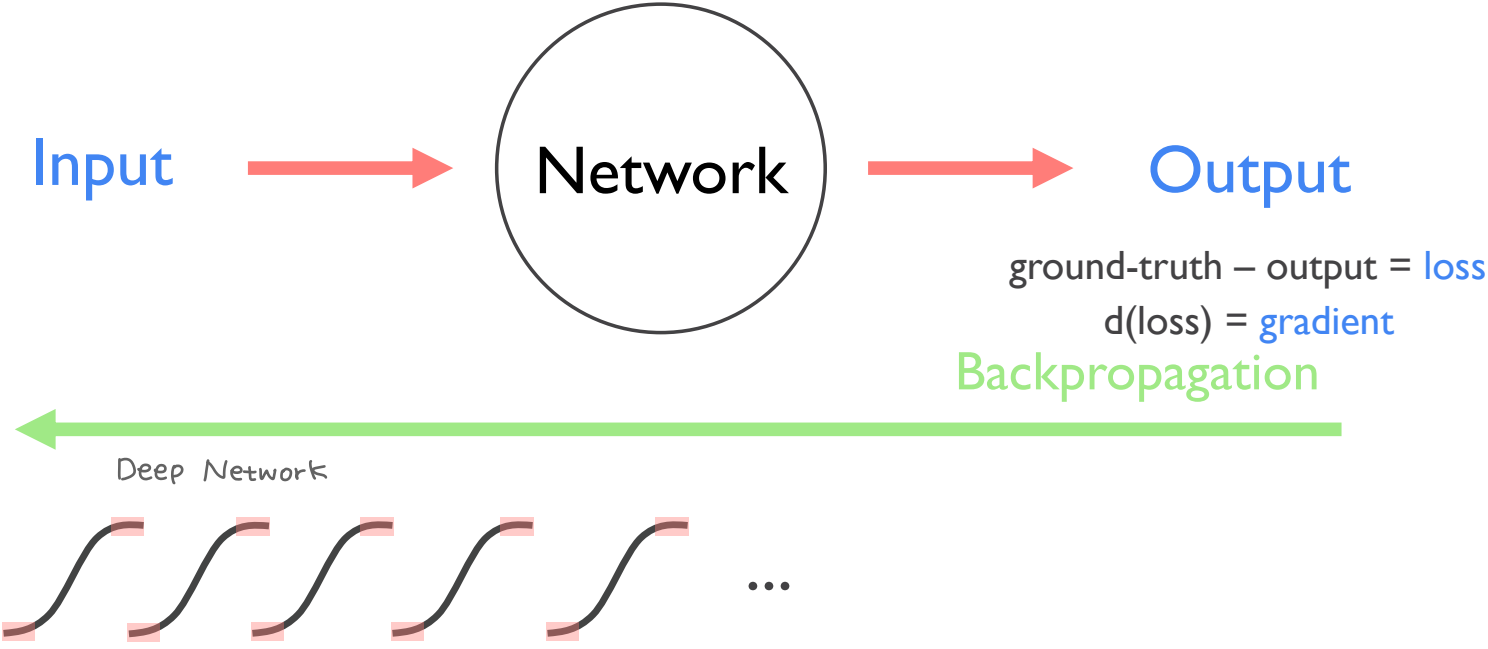
Problem of Sigmoid



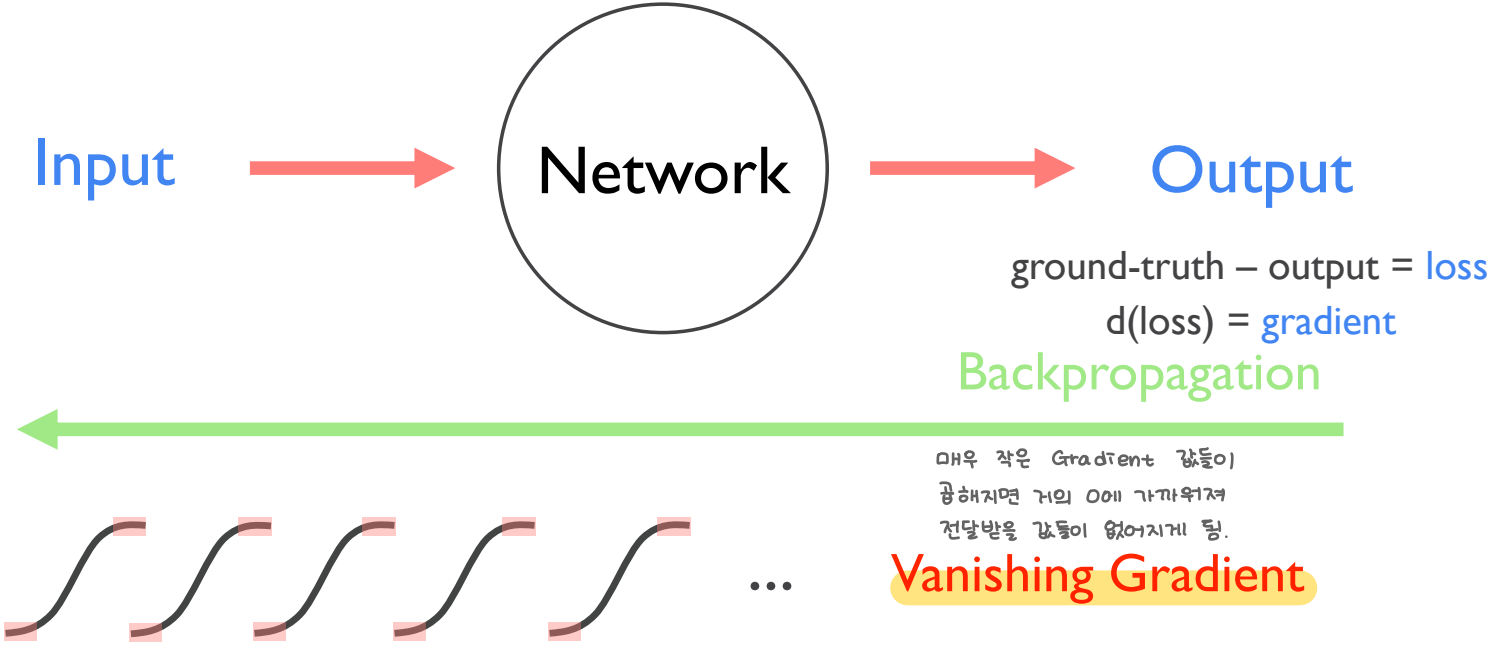
Problem of Sigmoid



Problem of Sigmoid



Problem of Sigmoid



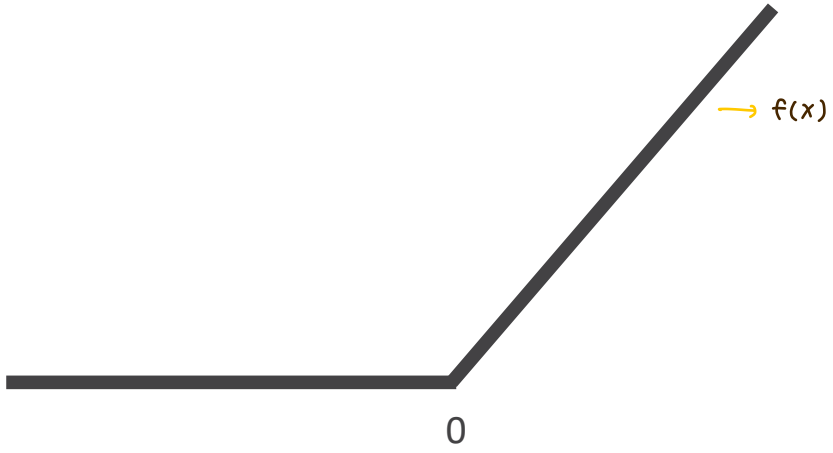
Why Relu ?

Why Relu ?

$$f(x) = \max(0, x)$$

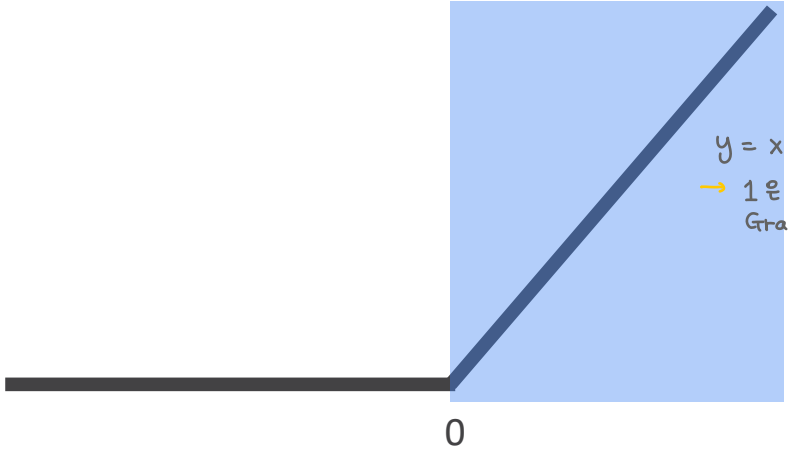
Why Relu ?

$$f(x) = \max(0, x)$$



Why Relu ?

$$f(x) = \max(0, x)$$



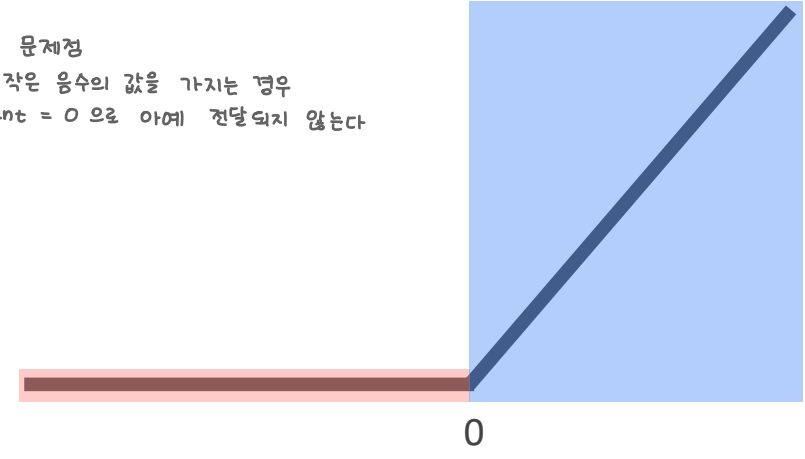
y = x , Gradient = 1

→ 1은 아무리 곱해도 1이기 때문에 Gradient 전달 잘 된다

Why Relu ?

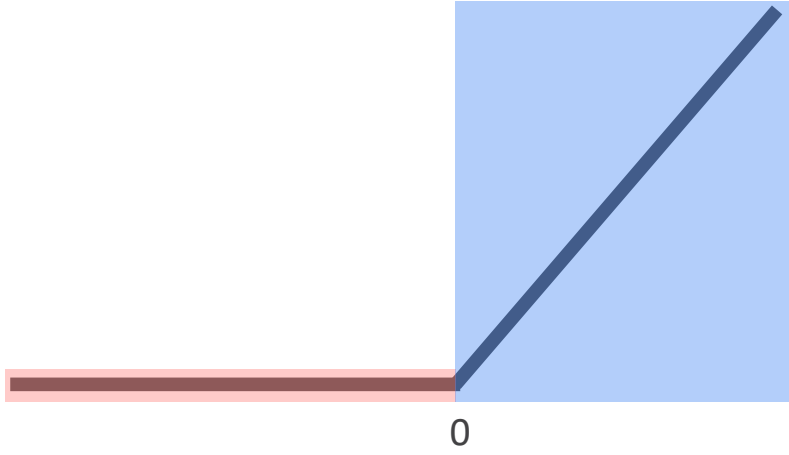
$$f(x) = \max(0, x)$$

* Relu의 문제점
  0보다 작은 음수의 값을 가지는 경우
  Gradient = 0 으로 아예 전달되지 않는다



Why Relu ?

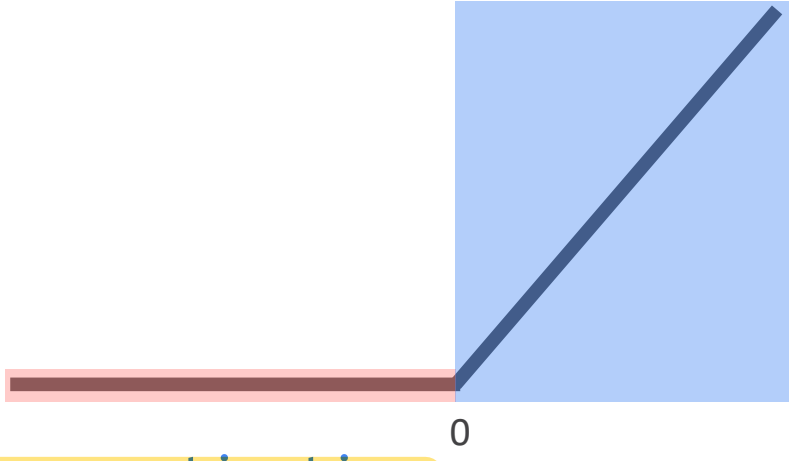
$$f(x) = \max(0, x)$$



sigmoid, tanh
relu, elu, selu

Why Relu ?

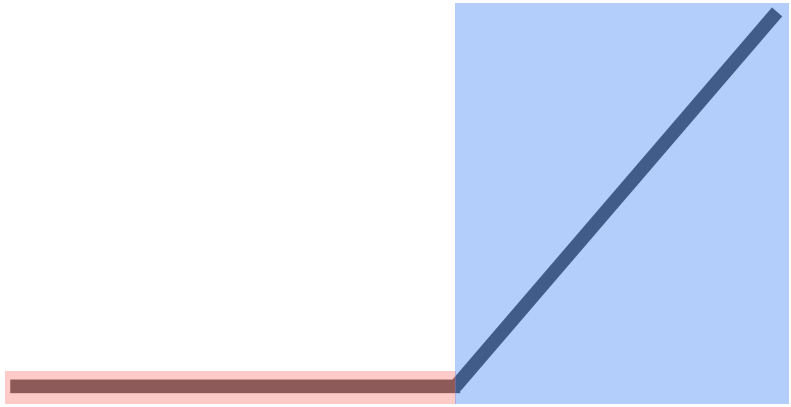
$$f(x) = \max(0, x)$$



`tf.keras.activations` → sigmoid, tanh
relu, elu, selu

Why Relu ?

$$f(x) = \max(0, x)$$

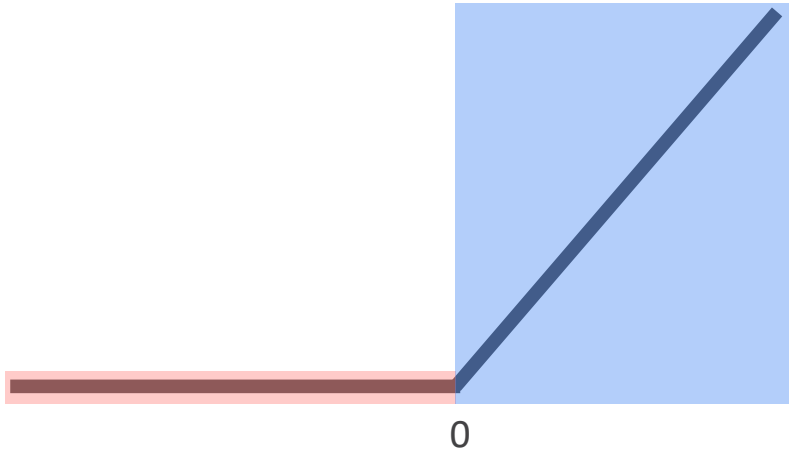


tf.keras.activations → sigmoid, tanh
relu, elu, selu

Relu의 음수에서의 문제점 해결
Relu output = 0 , leaky = $\alpha \times x$ → leaky relu

Why Relu ?

$$f(x) = \max(0, x)$$



`tf.keras.activations` → sigmoid, tanh
relu, elu, selu

`tf.keras.layers` → leaky relu

Code

Load mnist

Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

→ Eager 모드로 텐서플로우 실행

Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

```
def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()
```

↳ 4개의 OUTPUT , 6만개의 train data , 만 개의 test data

Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()

def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
```

채널을 추가할 위치,
-1 = 맨끝

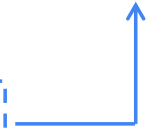
→ 채널 추가
WHY?
텐서플로우가 INPUT으로
받는 Shape의 경우
[batch_size , height ,
width , Channel] 설정

Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

[batch_size, height, width, channel]

```
def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
```



Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

[batch_size, height, width, channel]

```
def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]

    train_data, test_data = normalize(train_data, test_data) # [0 ~ 255] -> [0 ~ 1]
```

Load mnist

```python
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

[batch_size, height, width, channel]

```python
def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]

    train_data, test_data = normalize(train_data, test_data) # [0 ~ 255] -> [0 ~ 1]
```

```python
def normalize(train_data, test_data):
    train_data = train_data.astype(np.float32) / 255.0
    test_data = test_data.astype(np.float32) / 255.0

    return train_data, test_data
```

Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

[batch_size, height, width, channel]

```
def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]

    train_data, test_data = normalize(train_data, test_data) # [0 ~ 255] -> [0 ~ 1]

    train_labels = to_categorical(train_labels, 10) # [N,] -> [N, 10]
    test_labels = to_categorical(test_labels, 10) # [N,] -> [N, 10]
```

우리가 사용하는 데이터셋의 라벨의 총 개수

```
def normalize(train_data, test_data):
    train_data = train_data.astype(np.float32) / 255.0
    test_data = test_data.astype(np.float32) / 255.0

    return train_data, test_data
```

Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

[batch_size, height, width, channel]

```
def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]

    train_data, test_data = normalize(train_data, test_data) # [0 ~ 255] -> [0 ~ 1]

    train_labels = to_categorical(train_labels, 10) # [N,] -> [N, 10]
    test_labels = to_categorical(test_labels, 10) # [N,] -> [N, 10]
```

One hot incoding

```
def normalize(train_data, test_data):
    train_data = train_data.astype(np.float32) / 255.0
    test_data = test_data.astype(np.float32) / 255.0

    return train_data, test_data
```

Load mnist

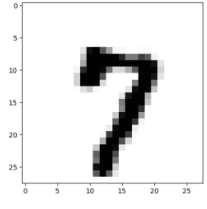
```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

[batch_size, height, width, channel]

```
def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]

    train_data, test_data = normalize(train_data, test_data) # [0 ~ 255] -> [0 ~ 1]

    train_labels = to_categorical(train_labels, 10) # [N,] -> [N, 10]
    test_labels = to_categorical(test_labels, 10) # [N,] -> [N, 10]
```

One hot incoding

```
def normalize(train_data, test_data):
    train_data = train_data.astype(np.float32) / 255.0
    test_data = test_data.astype(np.float32) / 255.0

    return train_data, test_data
```



Load mnist

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()
```

[batch_size, height, width, channel]

```
def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]

    train_data, test_data = normalize(train_data, test_data) # [0 ~ 255] -> [0 ~ 1]

    train_labels = to_categorical(train_labels, 10) # [N,] -> [N, 10]
    test_labels = to_categorical(test_labels, 10) # [N,] -> [N, 10]
```

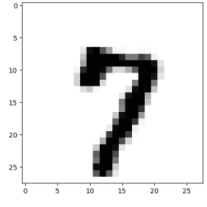
One hot incoding

7

0	0	0	0	0	0	0	1	0	0

TRUE

```
def normalize(train_data, test_data):
    train_data = train_data.astype(np.float32) / 255.0
    test_data = test_data.astype(np.float32) / 255.0

    return train_data, test_data
```



Load mnist

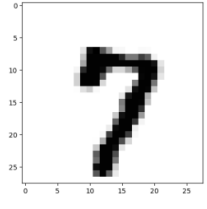
```python
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist # fasion_mnist, cifar10, cifar100
tf.enable_eager_execution()

def load_mnist() :
    (train_data, train_labels), (test_data, test_labels) = mnist.load_data()

    train_data = np.expand_dims(train_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]
    test_data = np.expand_dims(test_data, axis=-1) # [N, 28, 28] -> [N, 28, 28, 1]

    train_data, test_data = normalize(train_data, test_data) # [0 ~ 255] -> [0 ~ 1]

    train_labels = to_categorical(train_labels, 10) # [N,] -> [N, 10]
    test_labels = to_categorical(test_labels, 10) # [N,] -> [N, 10]

    return train_data, train_labels, test_data, test_labels

def normalize(train_data, test_data):
    train_data = train_data.astype(np.float32) / 255.0
    test_data = test_data.astype(np.float32) / 255.0

    return train_data, test_data
```

[batch_size, height, width, channel]

One hot incoding

7

0	0	0	0	0	0	0	1	0	0



Create network

Create network

```
def flatten() :
    return tf.keras.layers.Flatten()
```

Create network

```
def flatten() :
    return tf.keras.layers.Flatten()

def dense(channel, weight_init) :
    return tf.keras.layers.Dense(units=channel, use_bias=True, kernel_initializer=weight_init)
```

TRUE면 bias 사용

OUTPUT으로 나갈 채널 개수

Create network

```
def flatten() :
    return tf.keras.layers.Flatten()

def dense(channel, weight_init) :
    return tf.keras.layers.Dense(units=channel, use_bias=True, kernel_initializer=weight_init)

def relu() :
    return tf.keras.layers.Activation(tf.keras.activations.relu)
```

Create network

```
class create_model(tf.keras.Model):
```

→ Tf. keras. Model 상속하기

Create network

```
class create_model(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model, self).__init__()
```

→ 최종적으로 몇개의 OUTPUT을 낼건지 여기선 10

Create network

```
class create_model(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model, self).__init__()

        weight_init = tf.keras.initializers.RandomNormal()
```

→ 평균이 0, 분산이 1

Create network

```
class create_model(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model, self).__init__()

        weight_init = tf.keras.initializers.RandomNormal()
        self.model = tf.keras.Sequential()
```

↳ 리스트 자료구조 타입

Create network

```
class create_model(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model, self).__init__()

        weight_init = tf.keras.initializers.RandomNormal()
        self.model = tf.keras.Sequential()

        self.model.add(flatten()) # [N, 28, 28, 1] -> [N, 784]
```

Create network

```
class create_model(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model, self).__init__()

        weight_init = tf.keras.initializers.RandomNormal()
        self.model = tf.keras.Sequential()

        self.model.add(flatten()) # [N, 28, 28, 1] -> [N, 784]

        for i in range(2):
            # [N, 784] -> [N, 256] -> [N, 256]
            self.model.add(dense(256, weight_init))
            self.model.add(relu())
```

Create network

```
class create_model(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model, self).__init__()

        weight_init = tf.keras.initializers.RandomNormal()
        self.model = tf.keras.Sequential()

        self.model.add(flatten()) # [N, 28, 28, 1] -> [N, 784]

        for i in range(2):
            # [N, 784] -> [N, 256] -> [N, 256]
            self.model.add(dense(256, weight_init))
            self.model.add(relu())

        self.model.add(dense(label_dim, weight_init)) # [N, 256] -> [N, 10]
```

Create network

```
class create_model(tf.keras.Model):
    def __init__(self, label_dim):
        super(create_model, self).__init__()

        weight_init = tf.keras.initializers.RandomNormal()
        self.model = tf.keras.Sequential()

        self.model.add(flatten()) # [N, 28, 28, 1] -> [N, 784]

        for i in range(2):
            # [N, 784] -> [N, 256] -> [N, 256]
            self.model.add(dense(256, weight_init))
            self.model.add(relu())

        self.model.add(dense(label_dim, weight_init)) # [N, 256] -> [N, 10]

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

        return x
```

Create network

```
def create_model(label_dim) :

    weight_init = tf.keras.initializers.RandomNormal()

    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
```

Define loss

Define loss

```python
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))
    return loss


def accuracy_fn(model, images, labels):
    logits = model(images, training=False)
    prediction = tf.equal(tf.argmax(logits, -1), tf.argmax(labels, -1))
    accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))
    return accuracy


def grad(model, images, labels):
    with tf.GradientTape() as tape:
        loss = loss_fn(model, images, labels)
    return tape.gradient(loss, model.variables)
```

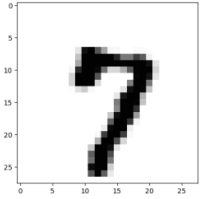
모델에 이미지를 넣어서 이미지의 숫자가 무엇인지 output으로 추출 (모델의)

Define loss

```
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))
    return loss
```

Define loss

```
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))
    return loss
```



0	0	0	0	0	0	0	1	0	0

label

0.1	0.1	0.0	0.2	0.0	0.0	0.0	0.6	0.0	0.0

softmax(logit)

Define loss

```python
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))
    return loss
```

```python
def accuracy_fn(model, images, labels):
    logits = model(images, training=False)
    prediction = tf.equal(tf.argmax(logits, -1), tf.argmax(labels, -1))
    accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))
    return accuracy
```

logits 에서 가장 큰 숫자의 위치 알려준다

Define loss

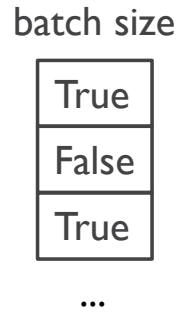
```
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))
    return loss


def accuracy_fn(model, images, labels):
    logits = model(images, training=False)
    prediction = tf.equal(tf.argmax(logits, -1), tf.argmax(labels, -1))
    accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))
    return accuracy
```

[batch size, label_dim]

Define loss

```python
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))
    return loss

def accuracy_fn(model, images, labels):
    logits = model(images, training=False)
    prediction = tf.equal(tf.argmax(logits, -1), tf.argmax(labels, -1))
    accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))
    return accuracy
```

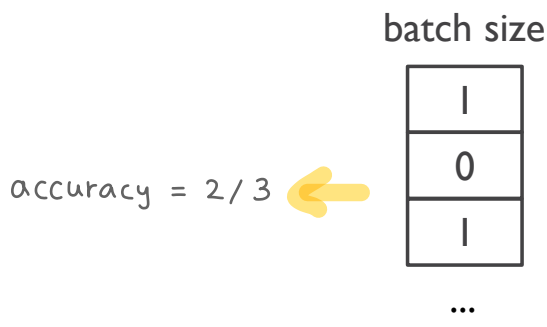
[batch size, label_dim]



Define loss

```python
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))
    return loss

def accuracy_fn(model, images, labels):
    logits = model(images, training=False)
    prediction = tf.equal(tf.argmax(logits, -1), tf.argmax(labels, -1))
    accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))
    return accuracy
```

[batch size, label_dim]



Define loss

```python
def loss_fn(model, images, labels):
    logits = model(images, training=True)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))
    return loss


def accuracy_fn(model, images, labels):
    logits = model(images, training=False)
    prediction = tf.equal(tf.argmax(logits, -1), tf.argmax(labels, -1))
    accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))
    return accuracy
```

```python
def grad(model, images, labels):
    with tf.GradientTape() as tape:
        loss = loss_fn(model, images, labels)
    return tape.gradient(loss, model.variables)
```

Experiments (parameters)

Experiments (parameters)

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

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
```

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
```

```
""" Graph Input using Dataset API """
train_dataset = tf.data.Dataset.from_tensor_slices((train_x, train_y)).\




test_dataset = tf.data.Dataset.from_tensor_slices((test_x, test_y)).\
```

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
```

```
""" Graph Input using Dataset API """
train_dataset = tf.data.Dataset.from_tensor_slices((train_x, train_y)).\
    shuffle(buffer_size=100000).\
```

↳ 데이터를 잘 섞어라 ↳ INPUT 의 데이터보다 숫자 크면 된다

```
test_dataset = tf.data.Dataset.from_tensor_slices((test_x, test_y)).\
    shuffle(buffer_size=100000).\
```

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
```

```
""" 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).\
```

↳ 네트워크가 batch size 만큼 학습을 하고 있을 때
미리 메모리에 batch size 만큼 올려놔라 → 빠르게 학습

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

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
```

```
""" 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).\


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)).\
```

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
```

```
""" 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).\
    repeat()

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)).\
    repeat()
```

Experiments (model)

Experiments (model)

```
""" Dataset Iterator """
train_iterator = train_dataset.make_one_shot_iterator()
test_iterator = test_dataset.make_one_shot_iterator()
```

Experiments (model)

```
""" Dataset Iterator """
train_iterator = train_dataset.make_one_shot_iterator()
test_iterator = test_dataset.make_one_shot_iterator()
```

```
""" Model """
network = create_model(label_dim)
```

Experiments (model)

```
""" Dataset Iterator """
train_iterator = train_dataset.make_one_shot_iterator()
test_iterator = test_dataset.make_one_shot_iterator()
```

```
""" Model """
network = create_model(label_dim)
```

```
""" Training """
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
```

Experiments (Eager mode)

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()
```

→ 네트워크가 학습을 하다가 중간에 끊겼을 때
재학습을 위해 변경되었던 weight 불러냄
아 테스트 이미지의 정확도를 볼 수 있다

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()

for epoch in range(start_epoch, training_epochs):
    for idx in range(start_iteration, training_iterations):
```

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()

for epoch in range(start_epoch, training_epochs):
    for idx in range(start_iteration, training_iterations):
        train_input, train_label = train_iterator.get_next()
```

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()

for epoch in range(start_epoch, training_epochs):
    for idx in range(start_iteration, training_iterations):
        train_input, train_label = train_iterator.get_next()

        grads = grad(network, train_input, train_label)
        optimizer.apply_gradients(grads_and_vars=zip(grads, network.variables), global_step=global_step)
```

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()

for epoch in range(start_epoch, training_epochs):
    for idx in range(start_iteration, training_iterations):
        train_input, train_label = train_iterator.get_next()

        grads = grad(network, train_input, train_label)
        optimizer.apply_gradients(grads_and_vars=zip(grads, network.variables), global_step=global_step)

        train_loss = loss_fn(network, train_input, train_label)
        train_accuracy = accuracy_fn(network, train_input, train_label)
```

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()

for epoch in range(start_epoch, training_epochs):
    for idx in range(start_iteration, training_iterations):
        train_input, train_label = train_iterator.get_next()

        grads = grad(network, train_input, train_label)
        optimizer.apply_gradients(grads_and_vars=zip(grads, network.variables), global_step=global_step)

        train_loss = loss_fn(network, train_input, train_label)
        train_accuracy = accuracy_fn(network, train_input, train_label)

        test_input, test_label = test_iterator.get_next()
        test_accuracy = accuracy_fn(network, test_input, test_label)
```

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()

for epoch in range(start_epoch, training_epochs):
    for idx in range(start_iteration, training_iterations):
        train_input, train_label = train_iterator.get_next()

        grads = grad(network, train_input, train_label)
        optimizer.apply_gradients(grads_and_vars=zip(grads, network.variables), global_step=global_step)

        train_loss = loss_fn(network, train_input, train_label)
        train_accuracy = accuracy_fn(network, train_input, train_label)

        test_input, test_label = test_iterator.get_next()
        test_accuracy = accuracy_fn(network, test_input, test_label)

        print("Epoch: [%2d] [%5d/%5d], train_loss: %.8f, train_accuracy: %.4f, test_Accuracy: %.4f" \
            % (epoch, idx, training_iterations, train_loss, train_accuracy, test_accuracy))
        counter += 1
```

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()

for epoch in range(start_epoch, training_epochs):
    for idx in range(start_iteration, training_iterations):
        train_input, train_label = train_iterator.get_next()

        grads = grad(network, train_input, train_label)
        optimizer.apply_gradients(grads_and_vars=zip(grads, network.variables), global_step=global_step)

        train_loss = loss_fn(network, train_input, train_label)
        train_accuracy = accuracy_fn(network, train_input, train_label)

        test_input, test_label = test_iterator.get_next()
        test_accuracy = accuracy_fn(network, test_input, test_label)

        print("Epoch: [%2d] [%5d/%5d], train_loss: %.8f, train_accuracy: %.4f, test_Accuracy: %.4f" \
            % (epoch, idx, training_iterations, train_loss, train_accuracy, test_accuracy))
        counter += 1

checkpoint.save(file_prefix=checkpoint_prefix + '-{}'.format(counter))
```

Experiments (Eager mode)

```
checkpoint = tf.train.Checkpoint(dnn=network)
global_step = tf.train.create_global_step()

for epoch in range(start_epoch, training_epochs):
    for idx in range(start_iteration, training_iterations):
        train_input, train_label = train_iterator.get_next()

        grads = grad(network, train_input, train_label)
        optimizer.apply_gradients(grads_and_vars=zip(grads, network.variables), global_step=global_step)

        train_loss = loss_fn(network, train_input, train_label)
        train_accuracy = accuracy_fn(network, train_input, train_label)

        test_input, test_label = test_iterator.get_next()
        test_accuracy = accuracy_fn(network, test_input, test_label)

        print("Epoch: [%2d] [%5d/%5d], train_loss: %.8f, train_accuracy: %.4f, test_Accuracy: %.4f" \
            % (epoch, idx, training_iterations, train_loss, train_accuracy, test_accuracy))
        counter += 1

checkpoint.save(file_prefix=checkpoint_prefix + '-{}'.format(counter))
```

Sigmoid : 81.31 %

Relu : 85.35 %

What's Next?

- Weight initialization
  - Xavier
  - He