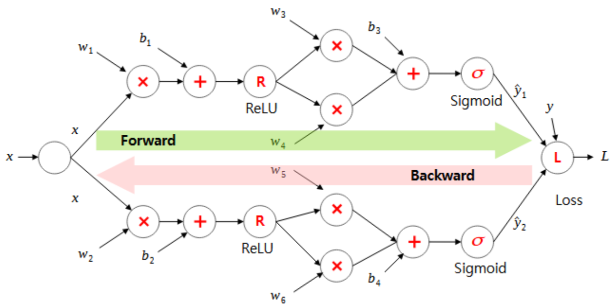
TensorFlow

GradientTape



4. TensorFlow & Keras

Part 1: Ways to build neural networks in TensorFlow

This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

TensorFlow / Keras

1. Multiple ways to build neural networks in TensorFlow and Keras API

[MXDL-4-01]

2. Type-1: Tensorflow's GradientTape() & gradient descent – [Example] Binary Classification
3. Type-2: Tensorflow's GradientTape() & Optimizer – [Example] Multiclass Classification
4. Type-3: Tensorflow's Optimizer – [Example] Nonlinear Regression

[MXDL-4-02]

5. Type-4: Keras Sequential model – [Example] Binary Classification
6. Type-5: Keras Functional API - [Example] Multiclass Classification
7. Type-6: Tensorflow + Keras Functional API - [Example] Nonlinear Regression
8. Type-7: Customizing Keras - [Example] Custom loss (Regularized loss function)

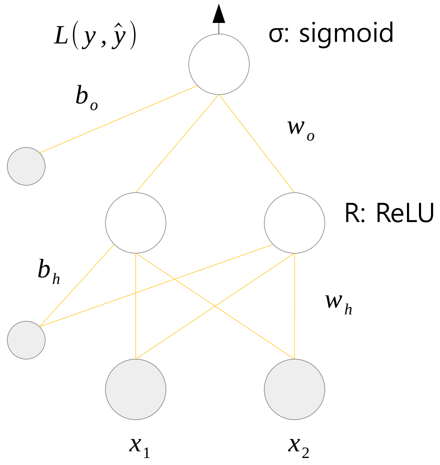
[MXDL-4-01] Deep Learning / TensorFlow & Keras

■ Multiple ways to build neural networks in TensorFlow and Keras API

Create a neural network

```
wh = tf.Variable(np.random.normal(size=(ni, nh)))
bh = tf.Variable(np.zeros(shape=(1, nh)))
wo = tf.Variable(np.random.normal(size=(nh, no)))
bo = tf.Variable(np.zeros(shape=(1, no)))
parameters = [wh, bh, wo, bo]

def predict(x): # forward propagation
    p = parameters
    h = tf.nn.relu(tf.matmul(x, p[0]) + p[1])
    y_hat = tf.sigmoid(tf.matmul(h, p[2]) + p[3])
    return y_hat
```

$$\hat{y}(w_h, b_h, w_o, b_o) = \sigma\left(R(x \cdot w_h + b_h) \cdot w_o + b_o\right)$$



1. GradientTape & Gradient descent

```
with tf.GradientTape() as tape:
     loss = Loss(y, y_hat)

grads = tape.gradient(loss, parameters)

for i, p in enumerate(parameters):
    p.assign_sub(lr * grads[i])
```

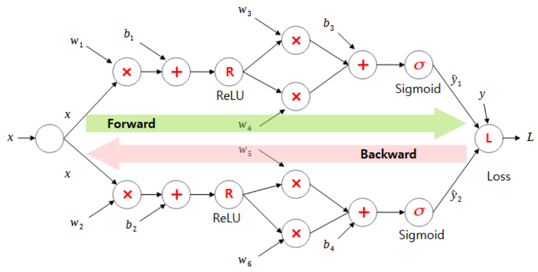
2. GradientTape & Optimizer

```
opt = optimizers.Adam(learning_rate=0.01,
                      beta_1=0.9, beta_2=0.999)

with tf.GradientTape() as tape:
     loss = Loss(y, y_hat)

grads = tape.gradient(loss, parameters)

opt.apply_gradients(zip(grads, parameters))
```

3. Optimizer

```
opt = optimizers.Adam(learning_rate = 0.01)

loss = Loss(y, y_hat)
opt.minimize(loss, parameters)
```

Automatic Differentiation



Adam optimizer

$$m_t = \beta m_{t-1} + (1-\beta) g_t$$

$$G_t = \rho G_{t-1} + (1-\rho) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta^n} \qquad \hat{G}_t = \frac{G_t}{1-\rho^n}$$

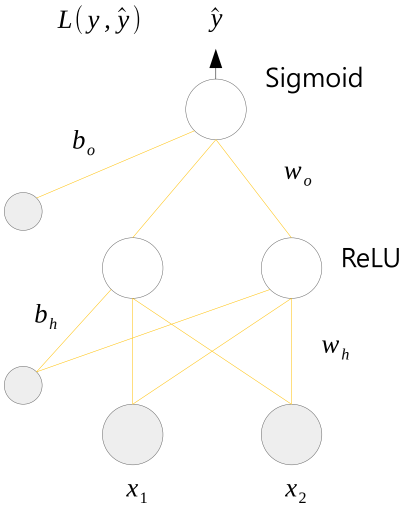
$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\hat{G}_t + \epsilon}} \cdot \hat{m}_t$$

MX-AI

Multiple ways to Build Neural Networks in TensorFlow and Keras API

4. Sequential Model

```
model = Sequential()
model.add(Dense(ni, input_dim=n1, activation='relu'))
model.add(Dense(no, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=adam)

h = model.fit(x, y, epochs=200, batch_size=50)
```



5. Functional API Model

```
x_input = Input(batch_shape=(None, ni))
h = Dense(nh, activation='relu')(x_input)
y_output = Dense(no, activation='sigmoid')(h)
model = Model(x_input, y_output)
model.compile(loss='binary_crossentropy', optimizer=adam)

h = model.fit(x, y, epochs=200, batch_size=50)
```

6. Functional API Model + Tensorflow

```
x_input = Input(batch_shape=(None, ni))
h = Dense(nh, activation='relu')(x_input)
y_output = Dense(no, activation='sigmoid')(h)
model = Model(x_input, y_output)

for i in range(epochs):
    with tf.GradientTape() as tape:
           loss = Loss(y, y_hat)

    grads = tape.gradient(loss, model.trainable_variables)
    opt.apply_gradients(zip(grads, model.trainable_variables))
```

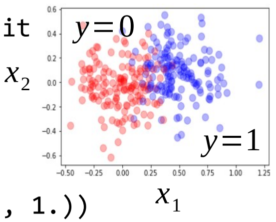
7. Customizing

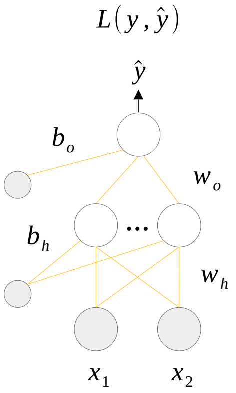
- Customizing Layer
- Customizing Loss function
- Customizing Model fit()

MX-AI

■ **Type-1.** Tensorflow's GradientTape & Gradient descent – [Example] Binary Classification

```python
# [MXDL-4-01] 1.tf_binary_class.py
# Binary classification
import numpy as np
import tensorflow as tf
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Generate a data set
x, y = make_blobs(n_samples=300, n_features=2,
                  centers=[[0., 0.], [0.5, 0.1]],
                  cluster_std=0.2, center_box=(-1., 1.))
y = y.reshape(-1,1)
x_train, x_test, y_train, y_test = train_test_split(x, y)

# Visually see the data.
plt.figure(figsize=(6,4))
color = [['red', 'blue'][a] for a in y.reshape(-1,)]
plt.scatter(x[:,0], x[:,1],s=100,c=color,alpha=0.3)
plt.show()

# Create an ANN with a hidden layer
n_input = x.shape[1]  # number of input neurons
n_output = 1          # number of output neurons
n_hidden = 8          # number of hidden neurons
lr = 0.05             # learning rate

# Initialize the parameters
wh = tf.Variable(np.random.normal(size=(n_input, n_hidden)))
bh = tf.Variable(np.zeros(shape=(1, n_hidden)))
wo = tf.Variable(np.random.normal(size=(n_hidden, n_output)))
bo = tf.Variable(np.zeros(shape=(1, n_output)))
parameters = [wh, bh, wo, bo]
```






```python
# loss function
def binary_crossentropy(y, y_hat):
    return -tf.reduce_mean(y * tf.math.log(y_hat) + \
                           (1. - y) * tf.math.log(1. - y_hat))

def predict(x, proba=True):
    p = parameters
    o_hidden = tf.nn.relu(tf.matmul(x, p[0]) + p[1])
    o_output = tf.sigmoid(tf.matmul(o_hidden, p[2]) + p[3])

    if proba:
        return o_output   # return sigmoid output as is
    else:
        return (o_output.numpy() > 0.5) * 1  # return class

def fit(x_trn, y_trn, x_val, y_val, epochs, batch_size):
    trn_loss = []
    val_loss = []
    for epoch in range(epochs):
        # Training with mini-batch
        for batch in range(int(x_trn.shape[0] / batch_size)):
            idx = np.random.choice(x_trn.shape[0], batch_size)
            x_bat = x_trn[idx]
            y_bat = y_trn[idx]

            # Automatic differentiation
            with tf.GradientTape() as tape:
                loss = binary_crossentropy(y_bat, predict(x_bat))

            # Find the gradients of loss w.r.t the parameters
            grads = tape.gradient(loss, parameters)

            # update parameters by the gradient descent
            for i, p in enumerate(parameters):
                p.assign_sub(lr * grads[i])  # p= p - lr * gradient
```

MX-AI

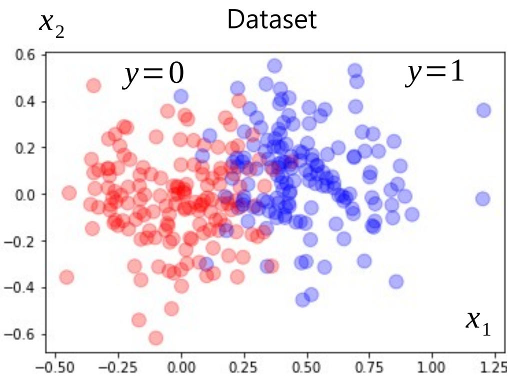
■ **Type-1.** Tensorflow's GradientTape & Gradient descent – [Example] Binary Classification

```
        # loss history
        loss = binary_crossentropy(y_trn, predict(x_trn))
        trn_loss.append(loss.numpy())

        loss = binary_crossentropy(y_val, predict(x_val))
        val_loss.append(loss.numpy())

        if epoch % 10 == 0:
            print("{}: train_loss={:.4f}, val_loss={:.4f}".\
                   format(epoch, trn_loss[-1], val_loss[-1]))

    return trn_loss, val_loss

# training
trn_loss, val_loss = fit(x_train, y_train, x_test, y_test,
                         epochs=200, batch_size=50)

# Visually see the loss history
plt.plot(trn_loss, c='blue', label='train loss')
plt.plot(val_loss, c='red', label='validation loss')
plt.legend()
plt.show()

# Check the accuracy of the test data
y_pred = predict(x_test, proba=False)
acc = (y_pred == y_test).mean()
print("\nAccuracy of the test data = {:4f}".format(acc))
```

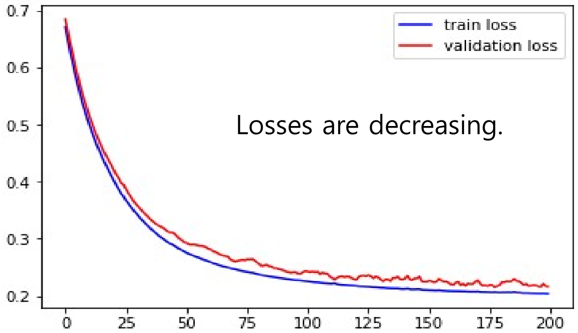
```
  0: train_loss=0.6704, val_loss=0.6838
 10: train_loss=0.4978, val_loss=0.5131
 20: train_loss=0.4015, val_loss=0.4194
 30: train_loss=0.3399, val_loss=0.3568
 40: train_loss=0.3006, val_loss=0.3195
 50: train_loss=0.2754, val_loss=0.2931
 60: train_loss=0.2590, val_loss=0.2813
 70: train_loss=0.2466, val_loss=0.2599
 80: train_loss=0.2374, val_loss=0.2559
 90: train_loss=0.2305, val_loss=0.2448
100: train_loss=0.2254, val_loss=0.2425
110: train_loss=0.2216, val_loss=0.2319
120: train_loss=0.2176, val_loss=0.2284
...
160: train_loss=0.2085, val_loss=0.2196
170: train_loss=0.2078, val_loss=0.2275
180: train_loss=0.2062, val_loss=0.2242
190: train_loss=0.2052, val_loss=0.2211
```

Accuracy of the test data = 0.92

Dataset

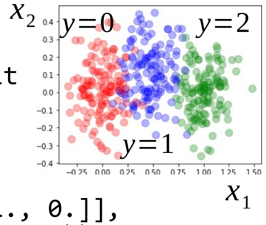


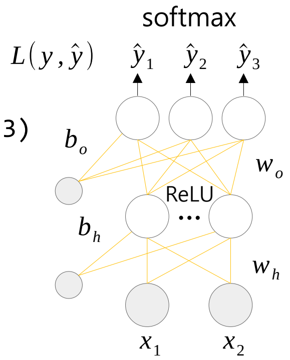
Loss history



■ **Type-2.** Tensorflow's GradientTape & Optimizer – [Example] Multiclass Classification

```python
# [MXDL-4-01] 2.tf_multi_class.py: Multiclass classification
import numpy as np
import tensorflow as tf
from tensorflow.keras import optimizers
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Generate a dataset
x, y = make_blobs(n_samples=400, n_features=2,
                  centers=[[0., 0.], [0.5, 0.1], [1., 0.]],
                  cluster_std=0.15, center_box=(-1., 1.))
n_class = np.unique(y).shape[0]  # the number of classes

# one-hot encode class y, y = [0,1,2]
y_ohe = np.eye(n_class)[y]
x_train, x_test, y_train, y_test = train_test_split(x, y_ohe)

# Visually see the data.
plt.figure(figsize=(5,4))
color = [['red', 'blue', 'green'][a] \
         for a in y.reshape(-1,)]
plt.scatter(x[:,0],x[:,1],s=100,c=color,alpha=0.3)
plt.show()

# Create an ANN with a hidden layer
n_input = x.shape[1] # number of input neurons
n_output = n_class   # number of output neurons
n_hidden = 8         # number of hidden neurons
lr = 0.05            # learning rate

# Initialize the parameters
wh = tf.Variable(np.random.normal(size=(n_input, n_hidden)))
bh = tf.Variable(np.zeros(shape=(1, n_hidden)))
wo = tf.Variable(np.random.normal(size=(n_hidden, n_output)))
bo = tf.Variable(np.zeros(shape=(1, n_output)))
parameters = [wh, bh, wo, bo]
```






```python
opt = optimizers.Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999)

# loss function
def crossentropy(y, y_hat):
    ce = -tf.reduce_sum(y * tf.math.log(y_hat),
                        axis=1)
    return tf.reduce_mean(ce)

def predict(x, proba=True):
    p = parameters
    o_hidden = tf.nn.relu(tf.matmul(x, p[0]) + p[1])
    o_output = tf.nn.softmax(tf.matmul(o_hidden, p[2]) + p[3])

    if proba:
        return o_output   # return softmax output as is
    else:
        return tf.math.argmax(o_output, axis=1)  # return class

def fit(x_trn, y_trn, x_val, y_val, epochs, batch_size):
    trn_loss, val_loss = [], []
    for epoch in range(epochs):
        # Training with mini-batch
        for batch in range(int(x_trn.shape[0] / batch_size)):
            idx = np.random.choice(x_trn.shape[0], batch_size)
            x_bat = x_trn[idx]
            y_bat = y_trn[idx]

            # Automatic differentiation
            with tf.GradientTape() as tape:
                loss = crossentropy(y_bat, predict(x_bat))

            # Find the gradients and update the parameters
            grads = tape.gradient(loss, parameters)
            opt.apply_gradients(zip(grads, parameters))
```

Adam optimizer

$$m_t = \beta m_{t-1} + (1-\beta) g_t$$

$$G_t = \rho G_{t-1} + (1-\rho) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta^n} \qquad \hat{G}_t = \frac{G_t}{1-\rho^n}$$

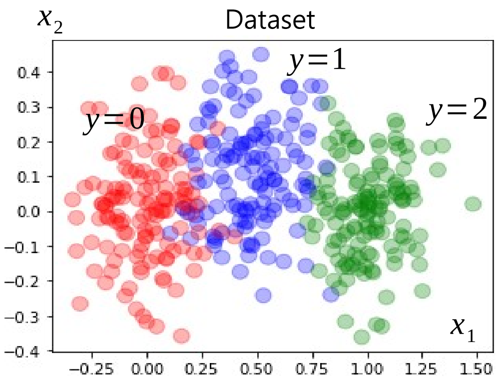
$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\hat{G}_t + \epsilon}} \cdot \hat{m}_t$$

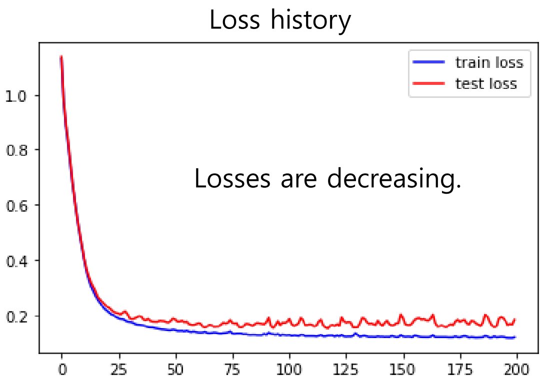
MX-AI

■ **Type-2.** Tensorflow's GradientTape & Optimizer – [Example] Multiclass Classification

```
        # loss history
        loss = crossentropy(y_trn, predict(x_trn))
        trn_loss.append(loss.numpy())

        loss = crossentropy(y_val, predict(x_val))
        val_loss.append(loss.numpy())

        if epoch % 10 == 0:
            print("{}: train_loss={:.4f}, val_loss={:.4f}".\
                  format(epoch, trn_loss[-1], val_loss[-1]))

    return trn_loss, val_loss

# training
train_loss, test_loss = fit(x_train, y_train, x_test, y_test,
                           epochs=200, batch_size=50)

# Visually see the loss history
plt.plot(train_loss, c='blue', label='train loss')
plt.plot(test_loss, c='red', label='test loss')
plt.legend()
plt.show()

# Check the accuracy of the test data
y_pred = predict(x_test, proba=False).numpy()
acc = (y_pred == np.argmax(y_test, axis=1)).mean()
print("Accuracy of test data = {:4f}".format(acc))
```

```
  0: train_loss=1.1298, val_loss=1.1364
 10: train_loss=0.3957, val_loss=0.4014
 20: train_loss=0.2158, val_loss=0.2312
 30: train_loss=0.1754, val_loss=0.1883
 40: train_loss=0.1559, val_loss=0.1764
 50: train_loss=0.1465, val_loss=0.1847
 60: train_loss=0.1371, val_loss=0.1708
 70: train_loss=0.1333, val_loss=0.1656
 80: train_loss=0.1306, val_loss=0.1596
 90: train_loss=0.1282, val_loss=0.1748
100: train_loss=0.1285, val_loss=0.1864
110: train_loss=0.1243, val_loss=0.1652
120: train_loss=0.1223, val_loss=0.1607
...
160: train_loss=0.1198, val_loss=0.1816
170: train_loss=0.1194, val_loss=0.1623
180: train_loss=0.1204, val_loss=0.1793
190: train_loss=0.1233, val_loss=0.1569

Accuracy of the test data = 0.92
```

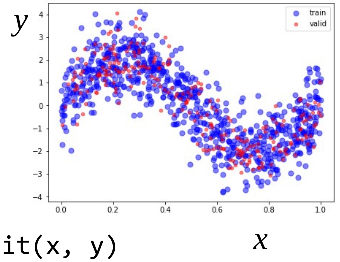




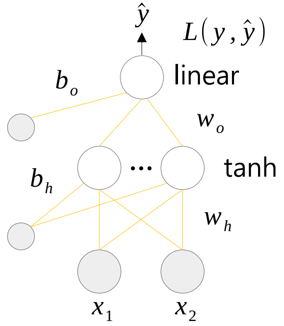
MX-AI

■ **Type-3.** Tensorflow's Optimizer – [Example] Nonlinear Regression

```python
# [MXDL-4-01] 3.tf_regression.py Nonlinear Regression
import numpy as np
import tensorflow as tf
from tensorflow.keras import optimizers
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Generate a data set
x = np.random.random((1000, 1))
y = 2.0 * np.sin(2.0 * np.pi * x) + \
    np.random.normal(0.0, 0.8, (1000, 1))

# Generate training, test data set
x_train, x_test, y_train, y_test = train_test_split(x, y)
x_pred = np.linspace(0, 1, 200).reshape(-1, 1)

# Visually see the data.
plt.figure(figsize=(7,5))
plt.scatter(x_train, y_train, s=20, c='blue', alpha=0.3, label='train')
plt.scatter(x_test, y_test, s=20, c='red', alpha=0.3, label='valid')
plt.legend()
plt.show()
```




```python
# Create an ANN with a hidden layer
n_input = x.shape[1]   # number of input neurons
n_output = 1           # number of output neurons
n_hidden = 8           # number of hidden neurons
lr = 0.05              # learning rate
```




```python
# Initialize the parameters
wh = tf.Variable(np.random.normal(size=(n_input, n_hidden)))
bh = tf.Variable(np.zeros(shape=(1, n_hidden)))
wo = tf.Variable(np.random.normal(size=(n_hidden, n_output)))
bo = tf.Variable(np.zeros(shape=(1, n_output)))
parameters = [wh, bh, wo, bo]
opt = optimizers.Adam(learning_rate = 0.01)

# loss function: mean squared error
def mse(y, y_hat):
    return tf.reduce_mean(tf.math.square(y - y_hat))

def predict(x, proba=True):
    p = parameters
    o_hidden = tf.math.tanh(tf.matmul(x, p[0]) + p[1])
    o_output = tf.matmul(o_hidden, p[2]) + p[3]
    return o_output

def fit(x_trn, y_trn, x_val, y_val, epochs, batch_size):
    trn_loss = []
    val_loss = []
    for epoch in range(epochs):
        # Training with mini-batch
        for batch in range(int(x_trn.shape[0] / batch_size)):
            idx = np.random.choice(x_trn.shape[0], batch_size)
            x_bat = x_trn[idx]
            y_bat = y_trn[idx]

            # Automatic differentiation and update parameters
            loss = lambda: mse(y_bat, predict(x_bat))
            opt.minimize(loss, parameters)
```

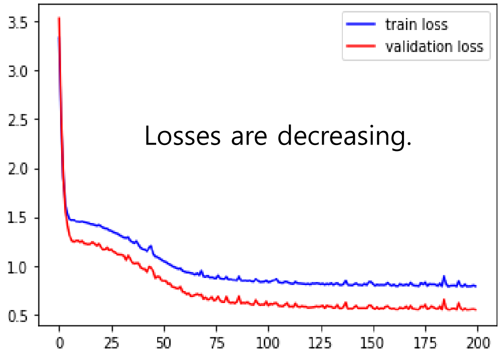
■ **Type-3.** Tensorflow's Optimizer – [Example] Nonlinear Regression

```
        # loss history
        loss = mse(y_trn, predict(x_trn))
        trn_loss.append(loss.numpy())

        loss = mse(y_val, predict(x_val))
        val_loss.append(loss.numpy())

        if epoch % 10 == 0:
            print("{}: train_loss={:.4f}, val_loss={:.4f}".\
                   format(epoch, trn_loss[-1], val_loss[-1]))

    return trn_loss, val_loss

# training
trn_loss, val_loss = fit(x_train, y_train, x_test, y_test,
                           epochs=200, batch_size=50)

# Visually see the loss history.
plt.plot(trn_loss, c='blue', label='train loss')
plt.plot(val_loss, c='red', label='validation loss')
plt.legend()
plt.show()

# Visually check the prediction result.
y_pred = predict(x_pred)
plt.figure(figsize=(7,5))
plt.scatter(x_train, y_train, s=20, c='blue', alpha=0.3,
            label='train')
plt.scatter(x_test, y_test, s=20, c='red', alpha=0.3,
            label='validation')
plt.scatter(x_pred, y_pred, s=5, c='red', label='test')
plt.legend()
plt.show()
```

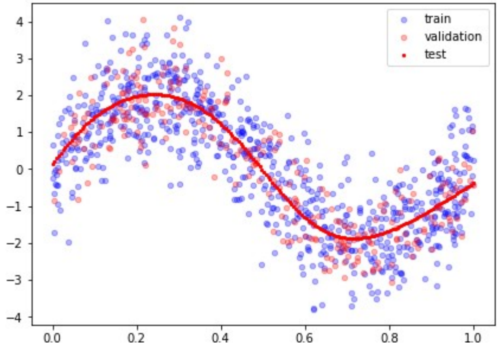
```
  0: train_loss=3.3347, val_loss=3.5312
 10: train_loss=1.4471, val_loss=1.2408
 20: train_loss=1.4016, val_loss=1.1910
 30: train_loss=1.2989, val_loss=1.1085
 40: train_loss=1.1695, val_loss=0.9670
 50: train_loss=1.0439, val_loss=0.8442
 60: train_loss=0.9381, val_loss=0.7275
 70: train_loss=0.8866, val_loss=0.6739
 80: train_loss=0.8866, val_loss=0.6811
 90: train_loss=0.8418, val_loss=0.6271
100: train_loss=0.8269, val_loss=0.5914
110: train_loss=0.8194, val_loss=0.5796
120: train_loss=0.8037, val_loss=0.5726
130: train_loss=0.8178, val_loss=0.5819
140: train_loss=0.7953, val_loss=0.5569
150: train_loss=0.8094, val_loss=0.5655
160: train_loss=0.8048, val_loss=0.5616
170: train_loss=0.7917, val_loss=0.5553
180: train_loss=0.8019, val_loss=0.5596
190: train_loss=0.7834, val_loss=0.5459
```

x_2

Loss history



Prediction



4. TensorFlow & Keras

Part 2: Ways to build neural networks in Keras

This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

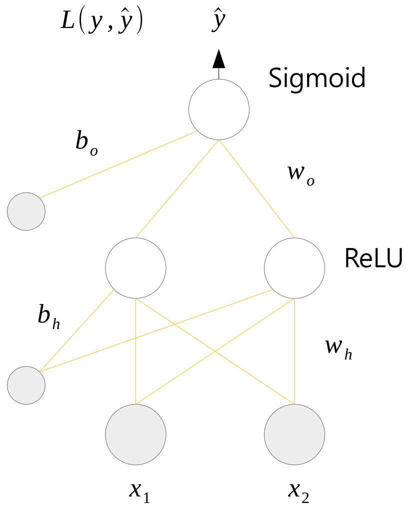
www.youtube.com/@meanxai

MX-AI

Multiple ways to Build Neural Networks in Keras

1. Sequential API

```
model = Sequential()
model.add(Dense(ni, input_dim=n1, activation='relu'))
model.add(Dense(no, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=adam)

h = model.fit(x, y, epochs=200, batch_size=50)
```



2. Functional API

```
x_input = Input(batch_shape=(None, ni))
h = Dense(nh, activation='relu')(x_input)
y_output = Dense(no, activation='sigmoid')(h)
model = Model(x_input, y_output)
model.compile(loss='binary_crossentropy', optimizer=adam)

h = model.fit(x, y, epochs=200, batch_size=50)
```

3. Functional API + Tensorflow

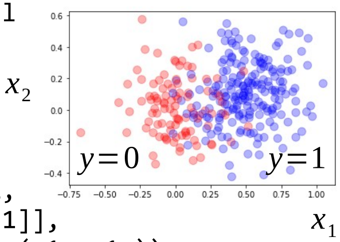
```
x_input = Input(batch_shape=(None, ni))
h = Dense(nh, activation='relu')(x_input)
y_output = Dense(no, activation='sigmoid')(h)
model = Model(x_input, y_output)

for i in range(epochs):
    with tf.GradientTape() as tape:
           loss = Loss(y, y_hat)

    grads = tape.gradient(loss, model.trainable_variables)
    opt.apply_gradients(zip(grads, model.trainable_variables))
```

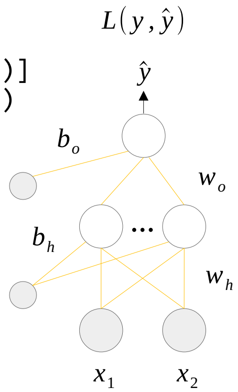
4. Customizing

- Custom Loss function
- Custom Layer
- Custom Model fit()

1. Keras Sequential API – [Example] Binary Classification

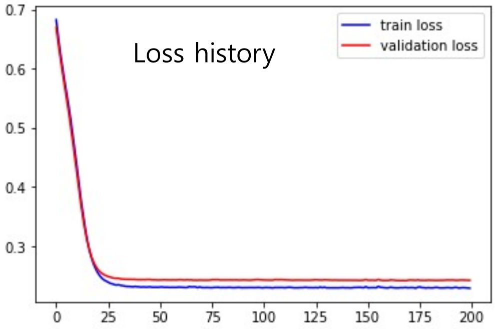
```python
# [MXDL-4-02] 4.keras_binary_class.py - Binary classification
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras import optimizers
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate a data set
x, y = make_blobs(n_samples=300, n_features=2,
                  centers=[[0., 0.], [0.5, 0.1]],
                  cluster_std=0.2, center_box=(-1., 1.))
y = y.reshape(-1,1)
x_train, x_test, y_train, y_test = train_test_split(x, y)

# Visually see the data.
plt.figure(figsize=(6,4))
color = [['red', 'blue'][a] for a in y.reshape(-1,)]
plt.scatter(x[:,0], x[:,1],s=100,c=color,alpha=0.3)
plt.show()

# Create an ANN with a hidden layer
n_input = x.shape[1] # number of input neurons
n_output = 1         # number of output neurons
n_hidden = 8         # number of hidden neurons
adam = optimizers.Adam(learning_rate=0.01)

# Create an ANN model
model = Sequential()
model.add(Dense(n_hidden, input_dim=n_input, activation='relu'))
model.add(Dense(n_output, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=adam)
```






```python
# training
f = model.fit(x_train, y_train,
              validation_data=(x_test, y_test),
              epochs=200, batch_size=50)

# Visually see the loss history
plt.plot(f.history['loss'], c='blue', label='train loss')
plt.plot(f.history['val_loss'], c='red', label='validation loss')
plt.legend()
plt.show()

# Check the accuracy of the test data
y_pred = (model.predict(x_test) > 0.5) * 1
acc = (y_pred == y_test).mean()
print("\nAccuracy of the test data = {:.2f}".format(acc))
```

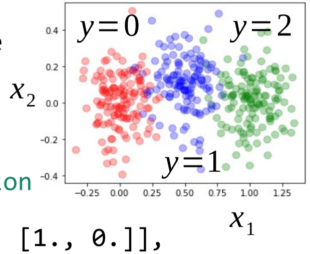
```
Epoch 198/200
5/5 [====================] - 0s 10ms/step - loss: 0.2377 - val_loss: 0.1801
Epoch 199/200
5/5 [====================] - 0s  6ms/step - loss: 0.2377 - val_loss: 0.1799
Epoch 200/200
5/5 [====================] - 0s  7ms/step - loss: 0.2376 - val_loss: 0.1806
```

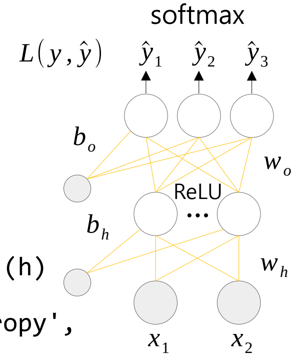


Accuracy of the test data = 0.92

■ 2. Keras functional API – [Example] Multiclass Classification

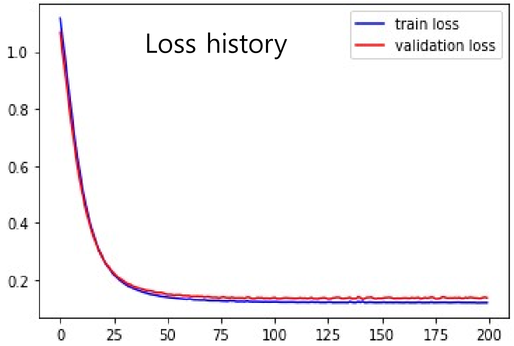
```
# [MXDL-4-02] 5.keras_multi_class.py - Multiclass classification
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_blobs
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt

# Generate a dataset for multiclass classification
x, y = make_blobs(n_samples=400, n_features=2,
                  centers=[[0., 0.], [0.5, 0.1], [1., 0.]],
                  cluster_std=0.15, center_box=(-1., 1.))

x_train, x_test, y_train, y_test = train_test_split(x, y)
n_class = np.unique(y).shape[0]  # the number of classes

# Create an ANN with a hidden layer
n_input = x.shape[1] # number of input neurons
n_output = n_class   # number of output neurons
n_hidden = 8         # number of hidden neurons
adam = optimizers.Adam(learning_rate=0.01)

# Create an ANN model
x_input = Input(batch_shape=(None, n_input))
h = Dense(n_hidden, activation='relu')(x_input)
y_output = Dense(n_output, activation='softmax')(h)
model = Model(x_input, y_output)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=adam)

# training
f = model.fit(x_train, y_train,
              validation_data=(x_test, y_test),
              epochs=200, batch_size=50)
```






```
# Visually see the data.
plt.figure(figsize=(5,4))
color = [['red', 'blue', 'green'][a] for a in y.reshape(-1,)]
plt.scatter(x[:, 0], x[:, 1], s=70, c=color, alpha=0.3)
plt.show()

# Visually see the loss history
plt.plot(f.history['loss'], c='blue', label='train loss')
plt.plot(f.history['val_loss'], c='red', label='validation loss')
plt.legend()
plt.show()

# Check the accuracy of the test data
y_pred = model.predict(x_test)
acc = (np.argmax(y_pred, axis=1) == y_test).mean()
print("Accuracy of test data = {:.2f}".format(acc))
```

```
Epoch 198/200
6/6 [==============================] - 0s 4ms/step - loss: 0.1207 - val_loss: 0.1395
Epoch 199/200
6/6 [==============================] - 0s 4ms/step - loss: 0.1210 - val_loss: 0.1411
Epoch 200/200
6/6 [==============================] - 0s 4ms/step - loss: 0.1209 - val_loss: 0.1385
```

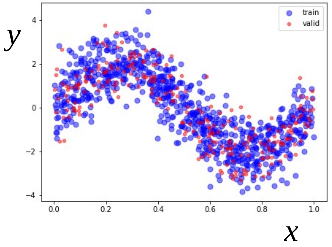


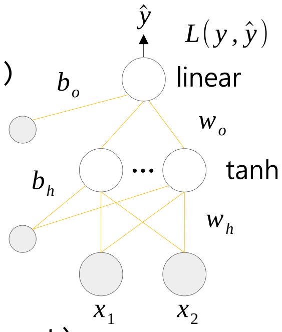
Accuracy of the test data = 0.96

MX-AI

3. Tensorflow + Keras functional API – [Example] Nonlinear Regression

```python
# [MXDL-4-02] 6.tf_keras_regression.py - Nonlinear Regression
import numpy as np
import tensorflow as tf
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt

# Generate a data set
x = np.random.random((1000, 1))
y = 2.0 * np.sin(2.0 * np.pi * x) + \
    np.random.normal(0.0, 0.8, (1000, 1))

# Generate training, test data set
x_train, x_test, y_train, y_test = train_test_split(x, y)
x_pred = np.linspace(0, 1, 200).reshape(-1, 1)

# loss function: mean squared error
def mse(y, y_hat):
    return tf.reduce_mean(tf.math.square(y-y_hat))

# Create an ANN model
n_input = x.shape[1] # number of input neurons
n_output = 1         # number of output neurons
n_hidden = 8         # number of hidden neurons
opt = optimizers.Adam(learning_rate=0.01)

x_input = Input(batch_shape=(None, n_input))
h_hidden = Dense(n_hidden, activation='tanh')(x_input)
y_output = Dense(n_output, activation='linear')(h_hidden)
model = Model(x_input, y_output)
```






```python
# Update parameters using tf.GradientTape() and optimizer
def fit(x_trn, y_trn, x_val, y_val, epochs, batch_size):
    trn_loss = []
    val_loss = []
    for epoch in range(epochs):
        # Training with mini-batch
        for batch in range(int(x.shape[0] / batch_size)):
            idx = np.random.choice(x_trn.shape[0], batch_size)
            x_bat = x_trn[idx]
            y_bat = y_trn[idx]

            # Automatic differentiation
            with tf.GradientTape() as tape:
                loss = mse(y_bat, model(x_bat))

            # Find the gradients of loss w.r.t the parameters
            grads = tape.gradient(loss, model.trainable_variables)

            # update parameters by optimizer
            opt.apply_gradients(zip(grads,
                                    model.trainable_variables))

        # loss history
        loss = mse(y_trn, model(x_trn))
        trn_loss.append(loss.numpy())

        loss = mse(y_val, model(x_val))
        val_loss.append(loss.numpy())

        if epoch % 10 == 0:
            print("{}: train_loss={:.4f}, val_loss={:.4f}".\
                  format(epoch, trn_loss[-1], val_loss[-1]))

    return trn_loss, val_loss
```

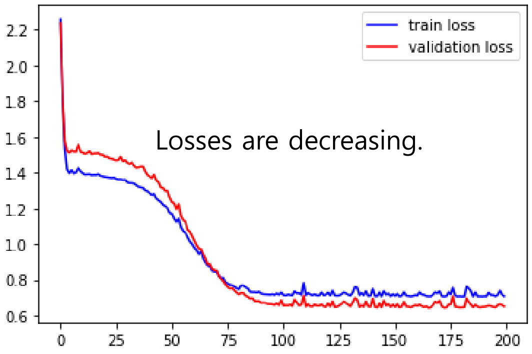
MX-AI

3. Tensorflow + Keras functional API – [Example] Nonlinear Regression

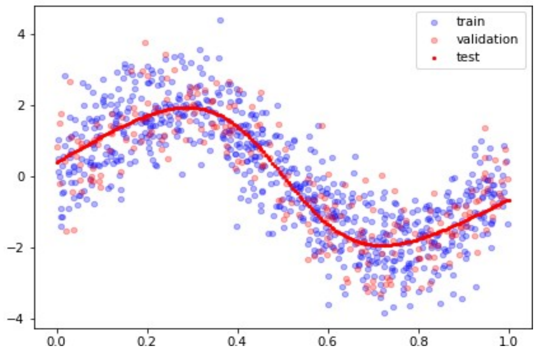
```python
# training
trn_loss, val_loss = fit(x_train, y_train, x_test, y_test,
                         epochs=200, batch_size=50)

# Visually see the data.
plt.figure(figsize=(7,5))
plt.scatter(x_train, y_train, s=50, c='blue', alpha=0.5,
label='train')
plt.scatter(x_test, y_test, s=20, c='red', alpha=0.5,
            label='valid')
plt.legend()
plt.show()

# Visually see the loss history.
plt.plot(trn_loss, c='blue', label='train loss')
plt.plot(val_loss, c='red', label='validation loss')
plt.legend()
plt.show()

# Visually check the prediction result.
y_pred = model.predict(x_pred)
plt.figure(figsize=(7,5))
plt.scatter(x_train, y_train, s=20, c='blue', alpha=0.3,
            label='train')
plt.scatter(x_test, y_test, s=20, c='red', alpha=0.3,
            label='validation')
plt.scatter(x_pred, y_pred, s=5, c='red', label='test')
plt.legend()
plt.show()
```

```
0: train_loss=2.2578, val_loss=2.2365
10: train_loss=1.3973, val_loss=1.5123
20: train_loss=1.3752, val_loss=1.4882
30: train_loss=1.3449, val_loss=1.4547
40: train_loss=1.2845, val_loss=1.3782
50: train_loss=1.1699, val_loss=1.2344
60: train_loss=0.9798, val_loss=1.0186
70: train_loss=0.8442, val_loss=0.8505
80: train_loss=0.7464, val_loss=0.7175
90: train_loss=0.7233, val_loss=0.6700
100: train_loss=0.7139, val_loss=0.6551
110: train_loss=0.7140, val_loss=0.6500
120: train_loss=0.7286, val_loss=0.6685
130: train_loss=0.7109, val_loss=0.6508
140: train_loss=0.7482, val_loss=0.6941
150: train_loss=0.7093, val_loss=0.6477
160: train_loss=0.7108, val_loss=0.6534
170: train_loss=0.7356, val_loss=0.6824
180: train_loss=0.7054, val_loss=0.6454
190: train_loss=0.7078, val_loss=0.6485
```

Loss history

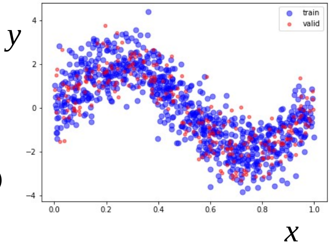


Prediction



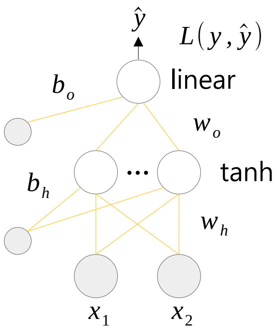
4. Custom Loss : Regularized loss function – [Example] Nonlinear Regression

```python
# [MXDL-4-02] 7.custom_loss.py - Regularized loss function
import numpy as np
import tensorflow as tf
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt

# Generate a data set
x = np.random.random((1000, 1))
y = 2.0 * np.sin(2.0 * np.pi * x) + \
    np.random.normal(0.0, 0.8, (1000, 1))

# Generate training, test data set
x_train, x_test, y_train, y_test = train_test_split(x, y)
x_pred = np.linspace(0, 1, 200).reshape(-1, 1)

# Custom loss: Applying L2 regularization to the loss function
class regularized_loss(tf.keras.losses.Loss):
  def __init__(self, C, h_layer, o_layer):
      super(regularized_loss, self).__init__()
      self.C = C
      self.h_layer=h_layer
      self.o_layer=o_layer

  def call(self, y_true, y_pred):
      mse = tf.reduce_mean(tf.math.square(y_true - y_pred))
      wh = self.h_layer.weights[0] # weights in hidden layer
      wo = self.o_layer.weights[0] # weights in output layer
      mse += self.C * tf.reduce_sum(tf.math.square(wh))
      mse += self.C * tf.reduce_sum(tf.math.square(wo))
      return mse
```



$$L_{reg}=\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2+C\left(\sum_m w_{h,m}^2+\sum_n w_{o,n}^2\right)$$

```python
# Create an ANN model
n_input = x.shape[1]      # number of input neurons
n_output = 1              # number of output neurons
n_hidden = 8              # number of hidden neurons
adam = optimizers.Adam(learning_rate=0.01)

h_layer = Dense(n_hidden, activation='tanh')     # hidden layer
o_layer = Dense(n_output, activation='linear')   # output layer

x_input = Input(batch_shape=(None, n_input))
h = h_layer(x_input)
y_output = o_layer(h)
model = Model(x_input, y_output)

myloss = regularized_loss(0.001, h_layer, o_layer)
model.compile(loss=myloss, optimizer=adam)

# Training
f = model.fit(x_train, y_train,
              validation_data=(x_test, y_test),
              epochs=200, batch_size=50)

# Visually see the data.
plt.figure(figsize=(7,5))
plt.scatter(x_train, y_train, s=50, c='blue', alpha=0.5, label='train')
plt.scatter(x_test, y_test, s=20, c='red', alpha=0.5, label='valid')
plt.legend()
plt.show()

# Visually see the loss history.
plt.plot(f.history['loss'], c='blue', label='train loss')
plt.plot(f.history['val_loss'], c='red', label='validation loss')
plt.legend()
plt.show()
```



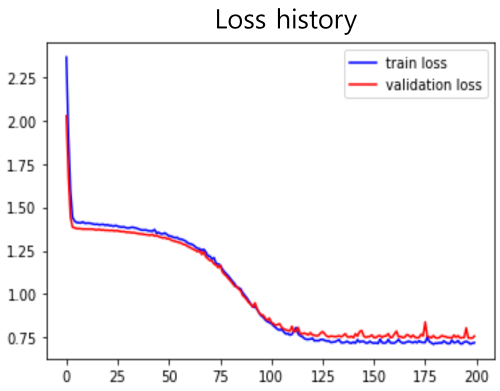
4. Custom Loss : Regularized loss function – [Example] Nonlinear Regression

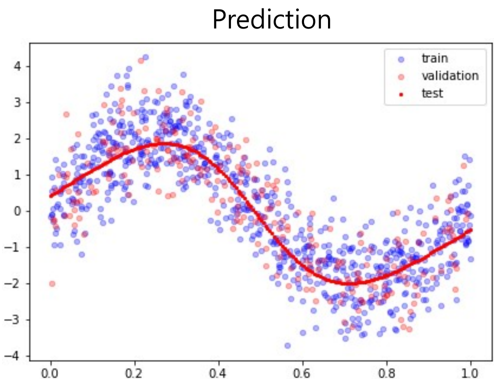
```
# Visually see the prediction result.
y_pred = model.predict(x_pred)
plt.figure(figsize=(7,5))
plt.scatter(x_train, y_train, s=20, c='blue', alpha=0.3,
label='train')
plt.scatter(x_test, y_test, s=20, c='red', alpha=0.3,
label='validation')
plt.scatter(x_pred, y_pred, s=5, c='red', label='test')
plt.legend()
plt.show()
```

$$L_{reg} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + C\left(\sum_m w_{h,m}^2 + \sum_n w_{o,n}^2\right)$$

[C = 0.001]

```
Epoch 198/200
15/15 [==================] - 0s 3ms/step - loss: 0.7438 - val_loss: 0.7287
Epoch 199/200
15/15 [==================] - 0s 3ms/step - loss: 0.7360 - val_loss: 0.7028
Epoch 200/200
15/15 [==================] - 0s 2ms/step - loss: 0.7381 - val_loss: 0.6930
```





[C = 0.01]

```
Epoch 198/200
15/15 [=================] - 0s 2ms/step - loss: 1.1684 - val_loss: 1.1273
Epoch 199/200
15/15 [=================] - 0s 3ms/step - loss: 1.1692 - val_loss: 1.1374
Epoch 200/200
15/15 [=================] - 0s 2ms/step - loss: 1.1658 - val_loss: 1.1238
```