ANN with TensorFlow: Theory and Practice

Junghwan Lee, Ph.D. (jhrrlee@gmail.com)

3rd lecture

- Analysis of an example code
- Code: Custom_model (https://github.com/jhrrlee/mlcode.git)

Analysis of example code

```
bs = pd.read_csv("basic_sample.csv") #z=x*2+y*3+2
```

We need to import panda as below

```
import pandas as pd
```

Check if sample data is loaded as below code

```
bs.sample(5)
```

Results will be shown as below

	x	y	z
6	6	21	77
2	2	2	12
15	7	1	19
12	4	6	28
5	5	5	27

"x" and "y" data are input data

"z" data is the results from "x" and "y"

So we need to separate input data from result data as below code

```
inputs = bs.drop('z', axis='columns')
```

We can check the separated data as below

```
print(inputs)
print(inputs.shape)
```

```
    x   y
0   1   1
1   2   2
2   2   2
3   3   3
4   4   4
5   5   5
6   6  21
7   7   1
8   1   2
9   1   3
10  2   4
11  3   5
12  4   6
13  5   7
14  6   2
15  7   1
16  1   2
17  2   3
18  3   4
```

- But inputs are still not the array.
- So, we convert inputs to array type as below

```
inputs = inputs.values.reshape(bs.shape[0],2)
```

- Now, we can see they are converted to the array type

```
array([[ 1,  1],
       [ 2,  2],
       [ 2,  2],
       [ 3,  3],
       [ 4,  4],
       [ 5,  5],
       [ 6, 21],
       [ 7,  1],
       [ 1,  2],
       [ 1,  3],
       [ 2,  4],
       [ 3,  5],
       [ 4,  6],
       [ 5,  7],
       [ 6,  2],
       [ 7,  1],
       [ 1,  2],
       [ 2,  3],
       [ 3,  4]], dtype=int64)
```

- We load result data into z as below

```
z = bs['z'].values.reshape(bs.shape[0],1)
```

- We can check

```
array([[ 7],
       [12],
       [12],
       [17],
       [22],
       [27],
       [77],
       [19],
       [10],
       [13],
       [18],
       [23],
       [28],
       [33],
       [20],
       [19],
       [10],
       [15],
       [20]], dtype=int64)
```

```
import time

class custom_train:


    def __init__(self, model, optimizer_fn, loss_fn, metric_fn):
        self.model = model
        self.optimizer = optimizer_fn
        self.loss_fn = loss_fn
        self.metrics = metric_fn

    #tf.function
    def train_step(self, x, y):
        with tf.GradientTape() as tape:
            #print(f'x:{x}, y:{y} {type(x)}')
            logits = self.model(x, training=True)
            loss_value = self.loss_fn(y, logits)
        grads = tape.gradient(loss_value, self.model.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.model.trainable_weights))
        self.metrics.update_state(y, logits)
        return loss_value

    def train(self, inputs, outputs, epochs, batch_size, batch_log, loss_threshold):
        loss_list = []
        epoch_list = []
        start_time = time.time()
        for epoch in range(epochs):
            loss_value = self.train_step(inputs, outputs)

            # Display metrics at the end of each epoch.
            train_acc = self.metrics.result()

            if epoch%50==0:
                loss_list.append(loss_value)
                epoch_list.append(epoch)
                print(
                    "Training acc over epoch: %.4f Training loss (for one batch) at step %d: %.4f Time taken: %.2fs "
                    % (float(train_acc), epoch, float(loss_value), (time.time() - start_time))
                )
            if loss_value <= loss_threshold:
                break
            # Reset training metrics at the end of each epoch
            self.metrics.reset_states()

        return loss_list, epoch_list
```

We import “time” to print execution time of each epoch.

```
import time
```

We will make the “custom_train” class get a neural network model, optimizer function, loss function, and metric function.

Therefore, we will make the constructor as below

```
    def __init__(self, model, optimizer_fn, loss_fn, metric_fn):
        self.model = model
        self.optimizer = optimizer_fn
        self.loss_fn = loss_fn
        self.metrics = metric_fn
```

Our train function performs 1) prediction calculation from an input model as below

```
logits = self.model(x, training=True)
```

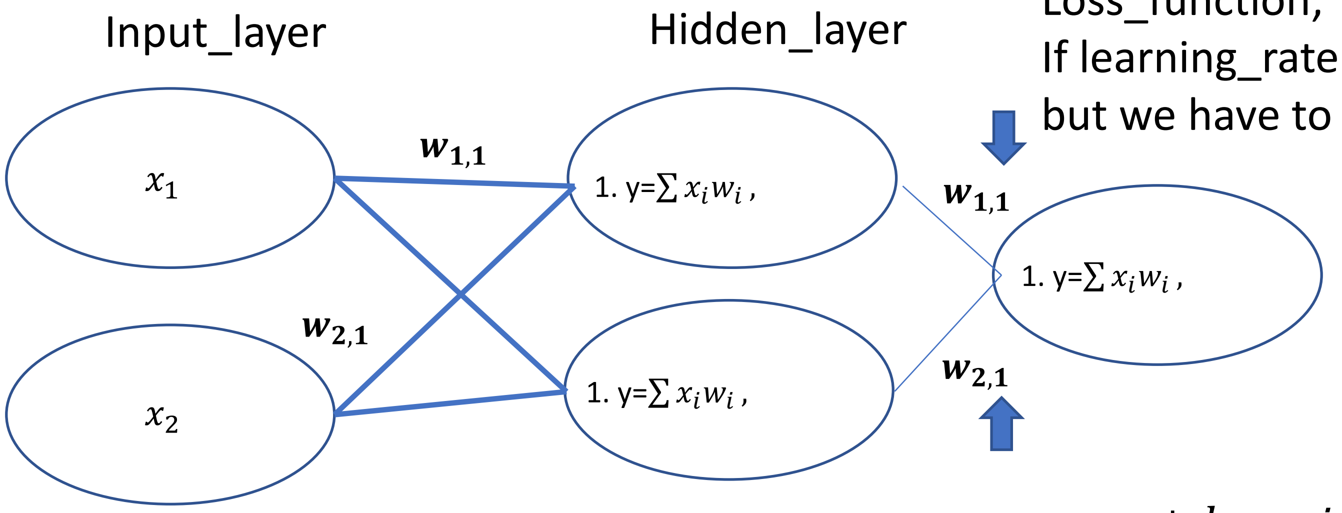
The logits have prediction values from the input model

“Training = true” will record trainable data “weights”

$$\boldsymbol{w_{1,1}} = \boldsymbol{w_{1,1}} + learning\_rate * \frac{\partial f(x_1,x_2)}{\partial x_1},\ f(x_1,x_2) =$$

Loss_function,

If learning_rate has large value then learning _speed will be fast, but we have to consider diffusion.



$$\hat{y}_i = \boldsymbol{w_{1,1}} * \boldsymbol{x_1} + \boldsymbol{w_{2,1}} * \boldsymbol{x_2} + \textbf{bias}$$

Loss_function = $\frac{1}{2m}\sum(\hat{y}_i - z_i)^2$: z_i in data set

$$w_{2,1} = w_{2,1} + learning\_rate * \frac{\partial f(x_1,x_2)}{\partial x_2}$$

2) Loss calculation

loss_value = self.loss_fn(y, logits)

The “y” is input argument of this functions. So we will give “z” for “y” argument

The loss function is as below.

Loss_function = $\frac{1}{2m}\sum(\hat{y}_i - z_i)^2$: z_i in data set

3) Gradient

“tape.gradient” perform gradient with the recorded data in previous “self.model(x, training=True)“

https://github.com/tensorflow/tensorflow/blob/v2.11.0/tensorflow/python/eager/backprop.py#L751-L1391

```
grads = tape.gradient(loss_value, self.model.trainable_weights)
```

4) Stochastic gradient descent

We will use “Adam optimization” which is a stochastic gradient descent method.

Reference:

D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” arXiv, Jan. 29, 2017. Accessed: Nov. 27, 2022. [Online]. Available: http://arxiv.org/abs/1412.6980

https://keras.io/api/optimizers/adam/

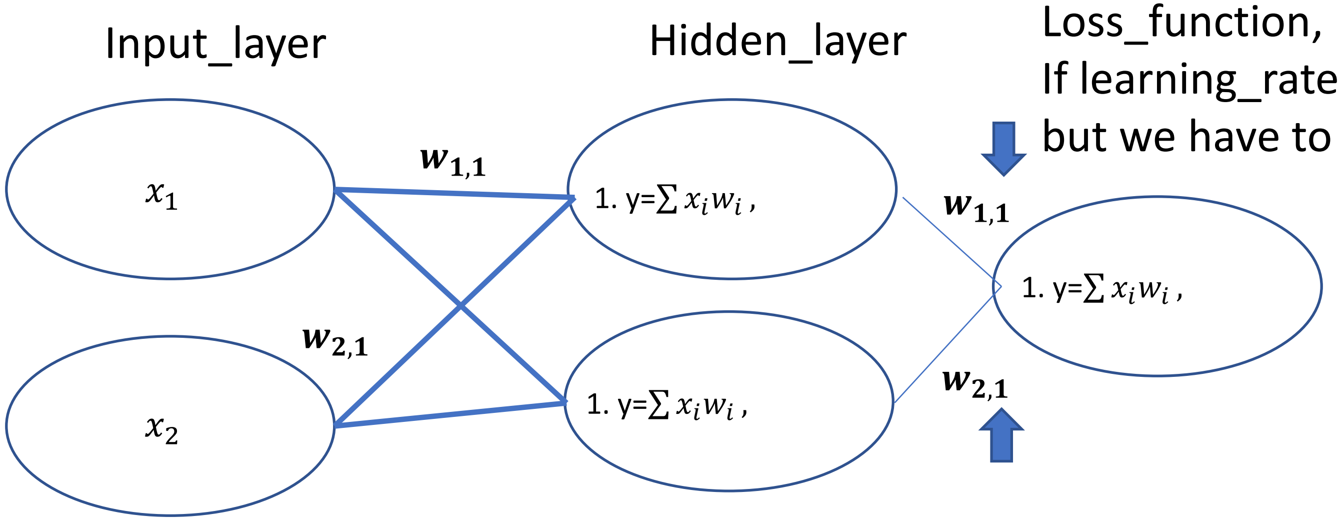
```
self.optimizer.apply_gradients(zip(grads, self.model.trainable_weights))

    self.metrics.update_state(y, logits)
    return loss_value
```

$\boldsymbol{w_{1,1}} = \boldsymbol{w_{1,1}} + learning\_rate * \frac{\partial f(x_1,x_2)}{\partial x_1}$, $f(x_1,x_2)$ =

Loss_function,

If learning_rate has large value then learning _speed will be fast,

but we have to consider diffusion.



$\hat{y}_i$ = $\boldsymbol{w_{1,1}}$*$\boldsymbol{x_1}$ + $\boldsymbol{w_{2,1}}$*$\boldsymbol{x_2}$ + **bias**

Loss_function = $\frac{1}{2m}\sum(\hat{y}_i - z_i)^2$: z_i in data set

$w_{2,1} = w_{2,1} + learning\_rate * \frac{\partial f(x_1,x_2)}{\partial x_2}$

5) Metric

The accuracy of prediction values can be calculated with prediction value and real values (“z”)

“y” is real values “z” and “logits” is prediction values

In this example, we will use MAE(mean absolute error).

```
self.metrics.update_state(y, logits)
```

We will make the train function using the train_step function.

```
def train(self, inputs, outputs, epochs, batch_size, batch_log, loss_threshold):
    loss_list = []
    epoch_list = []
    start_time = time.time()
    for epoch in range(epochs):
        loss_value = self.train_step(inputs, outputs)

        # Display metrics at the end of each epoch.
        train_acc = self.metrics.result()

        if epoch%50==0:
            loss_list.append(loss_value)
            epoch_list.append(epoch)
            print(
                "Training acc over epoch: %.4f Training loss (for one batch) at step %d: %.4f Time taken: %.2fs "
                % (float(train_acc), epoch, float(loss_value), (time.time() - start_time))
            )
        if loss_value <= loss_threshold:
            break
        # Reset training metrics at the end of each epoch
        self.metrics.reset_states()
```

- We will make the train function get input data, output data, the number of epochs, epoch log, and loss threshold.
- “epoch_log” is to see some results during training.
- The training is stopped if loss < “loss_threshold”.

```
def train(self, inputs, outputs, epochs, batch_size, batch_log, loss_threshold):
```

- Initial value set to record loss, epoch, and time.

```
loss_list = []
epoch_list = []
start_time = time.time()
```

- The training is run as many as the number of epochs.

```
for epoch in range(epochs):
```

- The Initial values as the array types are defined to record loss, epoch, and time.

```
loss_list = []
epoch_list = []
start_time = time.time()
```

- The training is run as many as the number of epochs.

```
for epoch in range(epochs):
```

- Loss value and accuracy are stored from the train_step function and metric.

```
loss_value = self.train_step(inputs, outputs)
train_acc = self.metrics.result()
```

- Record and print accuracy, loss, and execution time at the count of epoch log.

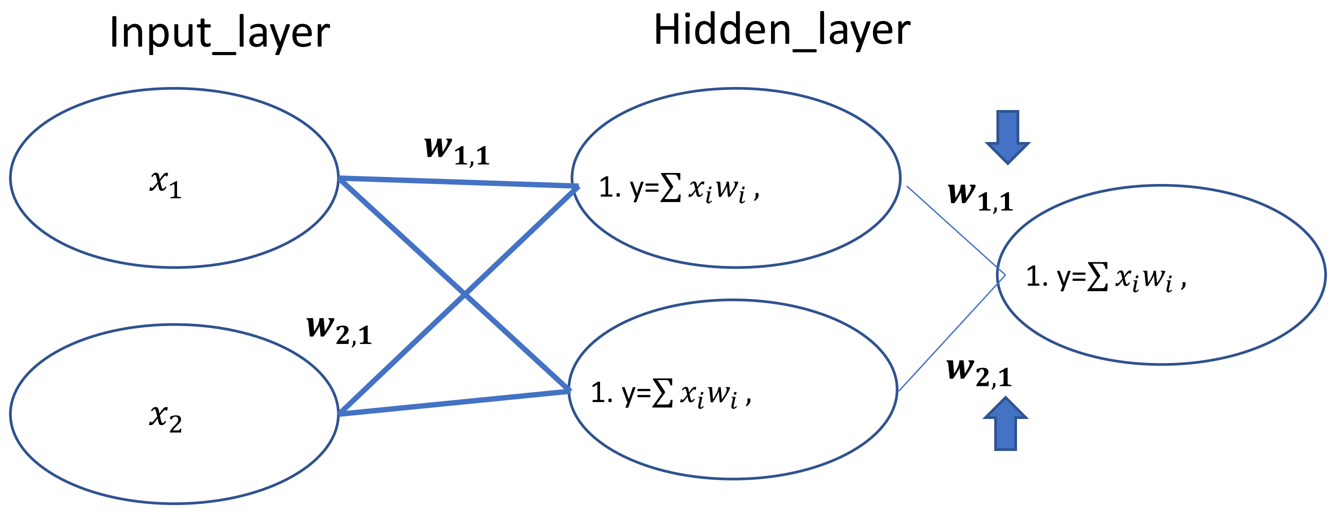
```
 if epoch%epoch_log==0:
        loss_list.append(loss_value)
        epoch_list.append(epoch)
        print(
            "Training acc over epoch: %.4f Training loss (for one batch) at step %d: %.4f Time taken: %.2fs "
            % (float(train_acc), epoch, float(loss_value), (time.time() - start_time))
        )
```

- Training will be stopped if loss_value <= loss_threshold

```
 if loss_value <= loss_threshold:
        break
```

- Now, we set our model as below

```
input_layer = keras.Input(shape=(2,))
hidden_layer = keras.layers.Dense(2)(input_layer)
output_layer = keras.layers.Dense(1)(hidden_layer)
```



- We set the optimizer (Adam), loss function (MSE), and metric (MAE)

```
model = keras.Model(input_layer, output_layer)
optimizer = tf.keras.optimizers.Adam()
loss_function = tf.keras.losses.MeanSquaredError(reduction="auto", name="mean_squared_error")
metric = tf.keras.metrics.MeanAbsoluteError()
```

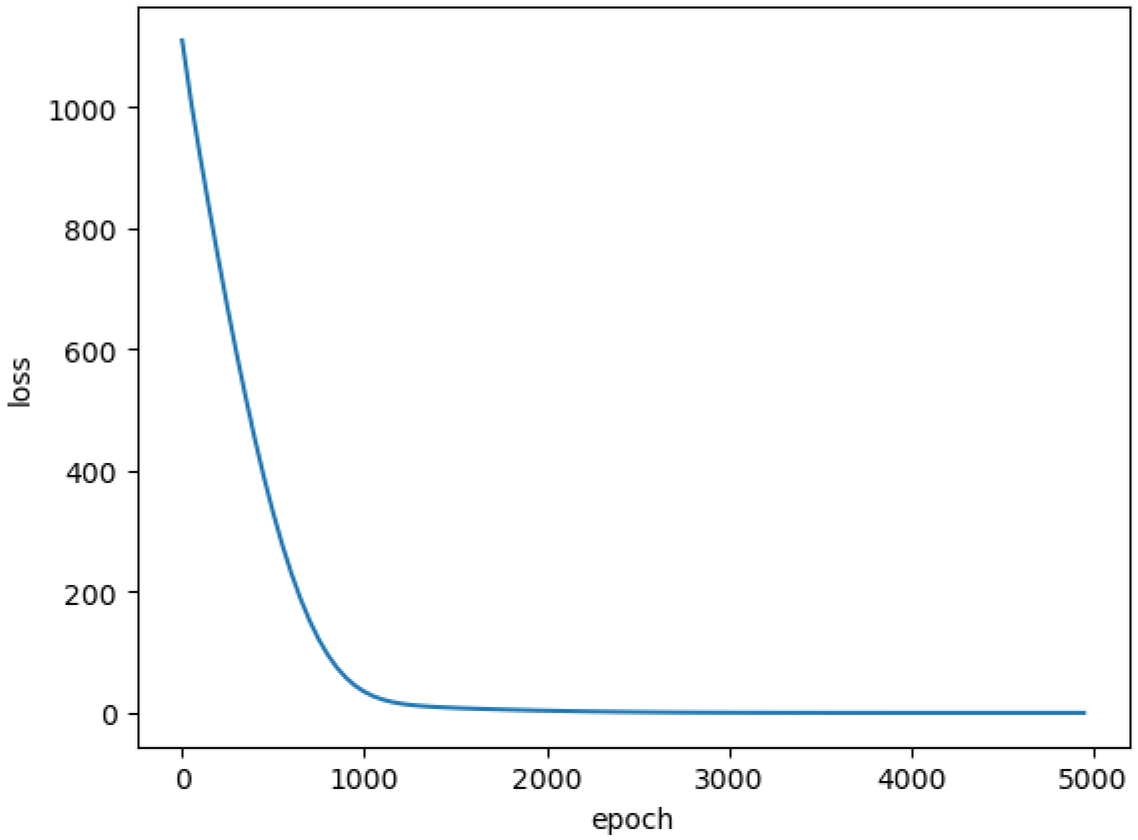
- Run our training function and record loss and epoch to draw a plot.

```
my_train = custom_train(model, optimizer, loss_function, metric)
loss_list, epoch_list = my_train.train(inputs, z, 5000, 50, 0.01)
```

- Draw a plot as below

```
plt.xlabel("epoch")
plt.ylabel("loss")
plt.plot(epoch_list, loss_list)
```

- We can see that the loss is decreased



- Finally, We simply test our trained model with prediction value because we already know the correct output value.

```
temp = np.array([[3, 2]], dtype='int64’)
model.predict(temp)
```

```
1/1 [==============================] - 0s 63ms/step
array([[14.065841]], dtype=float32)
```

- In the following, we will make practical models to predict pictures, numbers, and battery state-of-charge.
- The models will be similar to our example model here.