Machine learning 06

Byung Chang Chung

Gyeongsang National University

bcchung@gnu.ac.kr

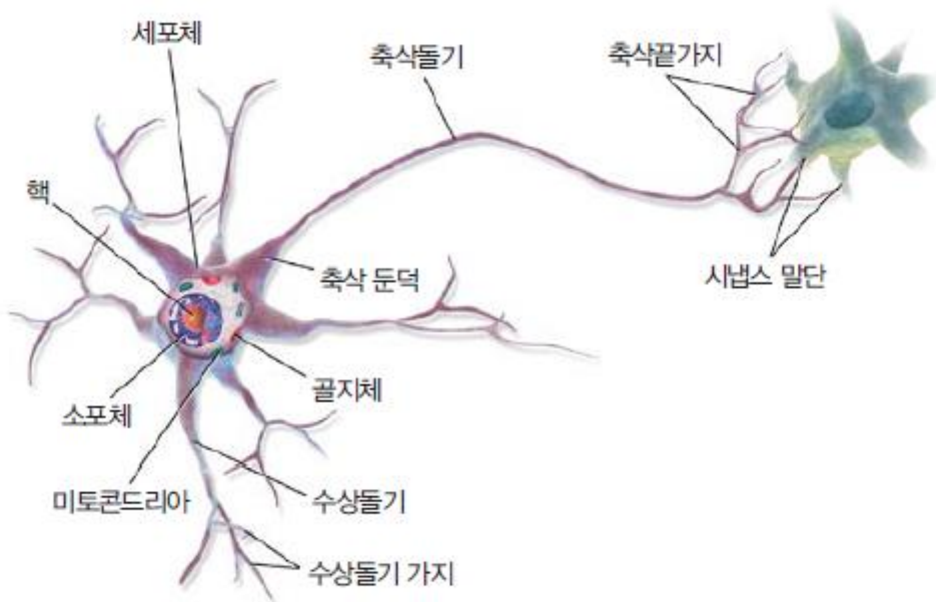
Contents

- Introduction to neural network
- keras application
- hyperparameter tuning

Gyeongsang
National
University
경상국립대학교

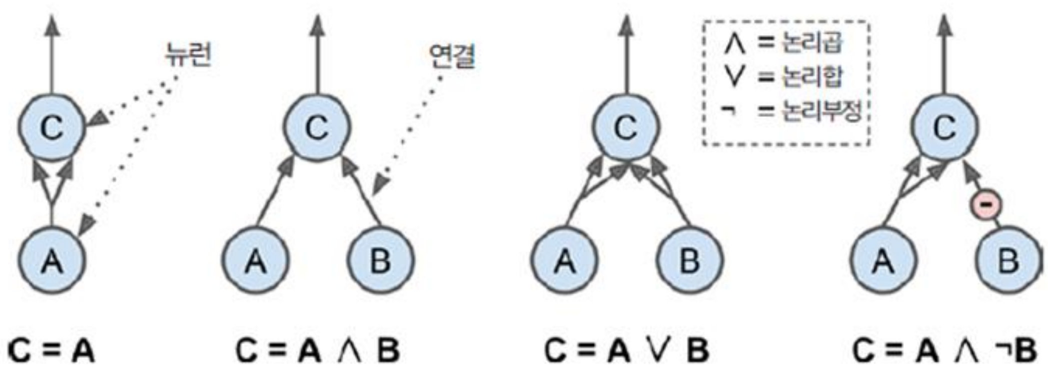
Artificial neural network

- Biological neural network



Artificial neural network

- Artificial neuron
  - Boolean calculation

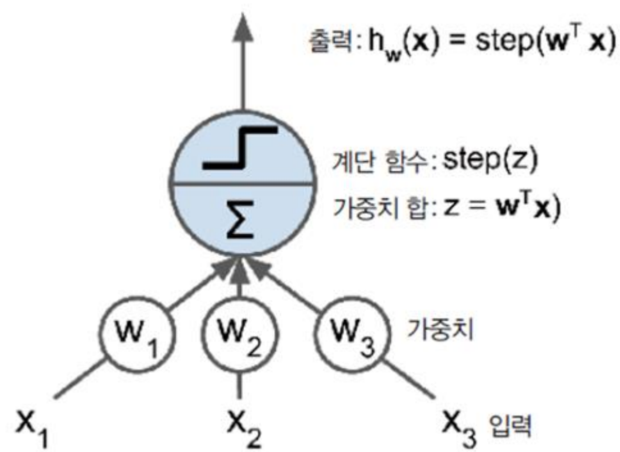


Artificial neural network

- Perceptron
  - one of the simplest artificial neural network structures
  - proposed by Frank Rosenblatt in 1957
  - based on a slightly different type of artificial neuron called threshold logic unit (TLU) or linear threshold unit (LTU)

Artificial neural network

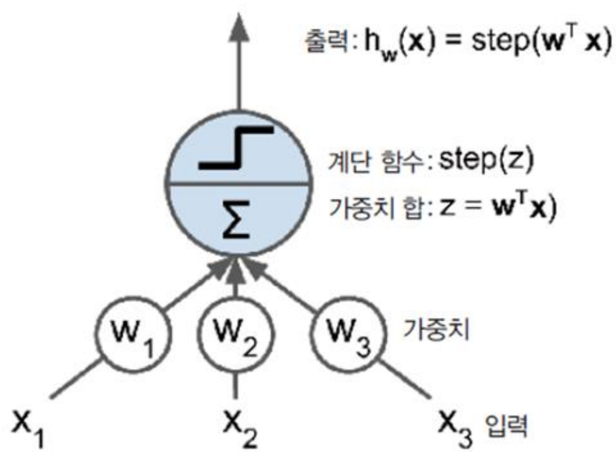
- Perceptron
  - input connection is related to the weight of neuron



Gyeongsang National University 경상국립대학교

Artificial neural network

- Perceptron
  - calculate the sum of the input weights
  - $z = w_1x_1 + w_2x_2 + \cdots + w_nx_n = \mathbf{x}^{\mathrm{T}}\mathbf{w}$
  - $h_{\mathbf{w}}(\mathbf{x}) = step(z)$



Artificial neural network

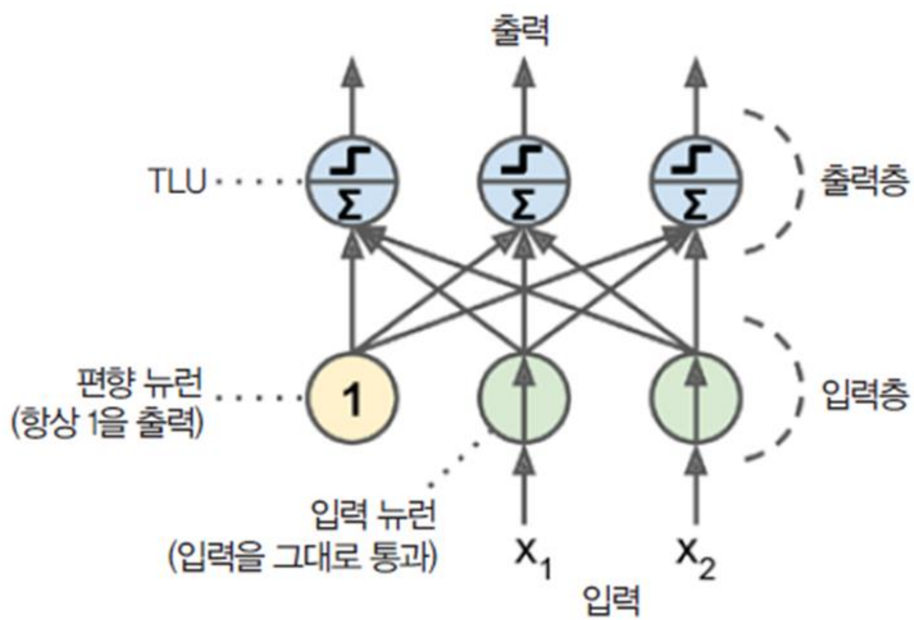
- Perceptron
  - step function

$$\text{heaviside}(z) = \begin{cases} 0 & z < 0\text{일 때} \\ 1 & z \geq 0\text{일 때} \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & z < 0\text{일 때} \\ 0 & z = 0\text{일 때} \\ +1 & z > 0\text{일 때} \end{cases}$$

  - single-layered TLU
    - every TLU is connected to all inputs
    - called as fully connected layer (dense layer)

Artificial neural network

- Perceptron
  - example



Gyeongsang National University 경상국립대학교

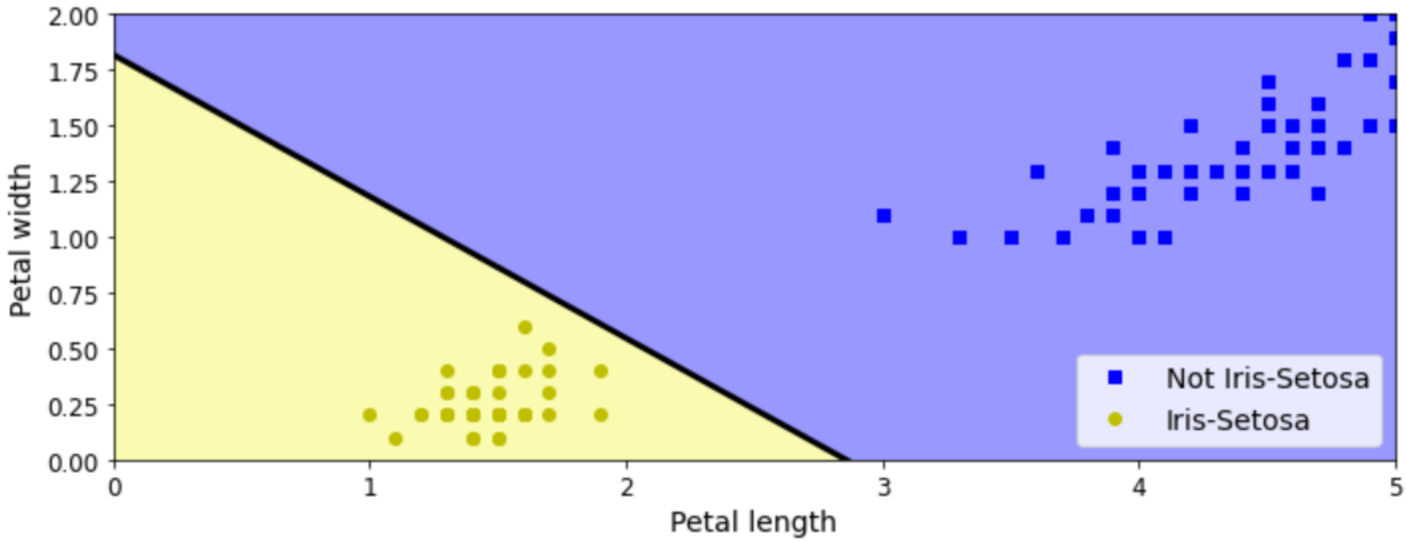
Artificial neural network

- Perceptron
  - weight update
    - $w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$
      - $w_{i,j}$: weight for neuron between i-th input and j-th output
      - x_i: i-th input of training sample
      - $\hat{y}_j$: j-th output of training sample
      - y_j: j-th target value of training sample
      - η: learning rate

GNU Gyeongsang National University 경상국립대학교

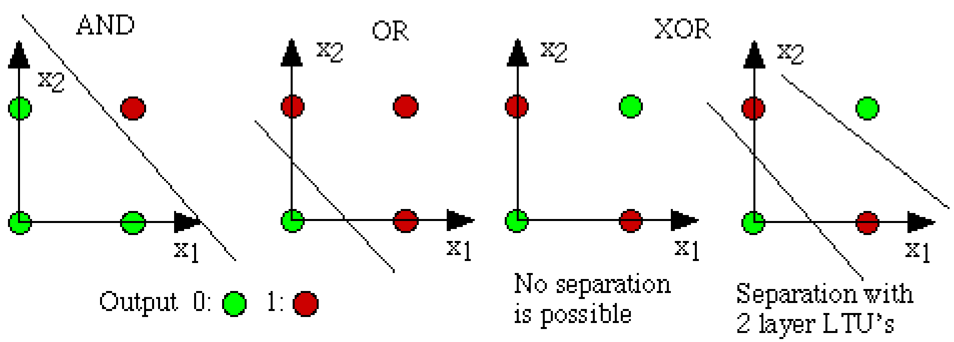
Artificial neural network

- Perceptron
  - implementation in scikit-learn



Artificial neural network

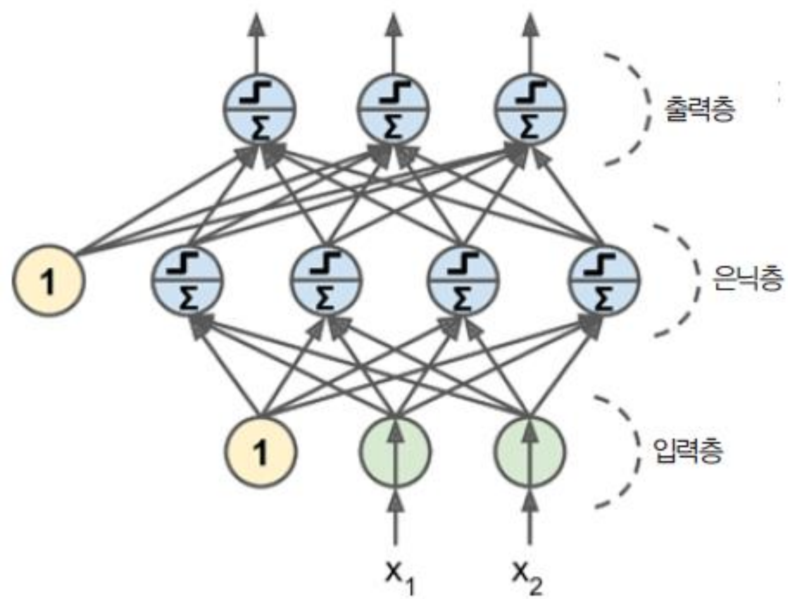
- Perceptron
  - weak points
    - XOR classification



https://erogol.com/brief-history-machine-learning

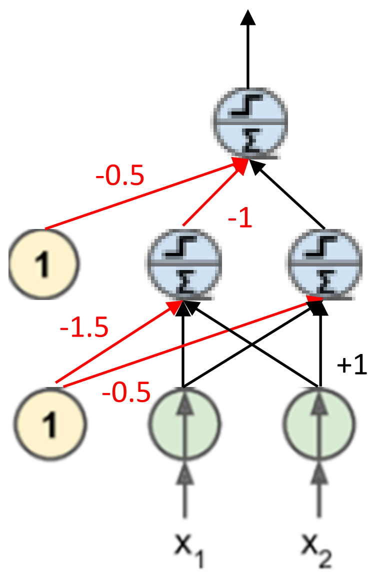
Artificial neural network

- Multi-layer perceptron
  - input layer
  - hidden layer
  - output layer



Artificial neural network

- Multi-layer perceptron
  - XOR classification



Artificial neural network

- Deep neural network
  - how to train deep network?
    - backpropagation
    - suggested by Rumelhart, Hinton, Williams in 1986
    - calculate gradient by chain rule

Artificial neural network

- Deep neural network
  - process of backpropagation
    - 1 epoch: indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Datasets are usually grouped into batches (especially when the amount of data is very large)

Gyeongsang National University
경상국립대학교

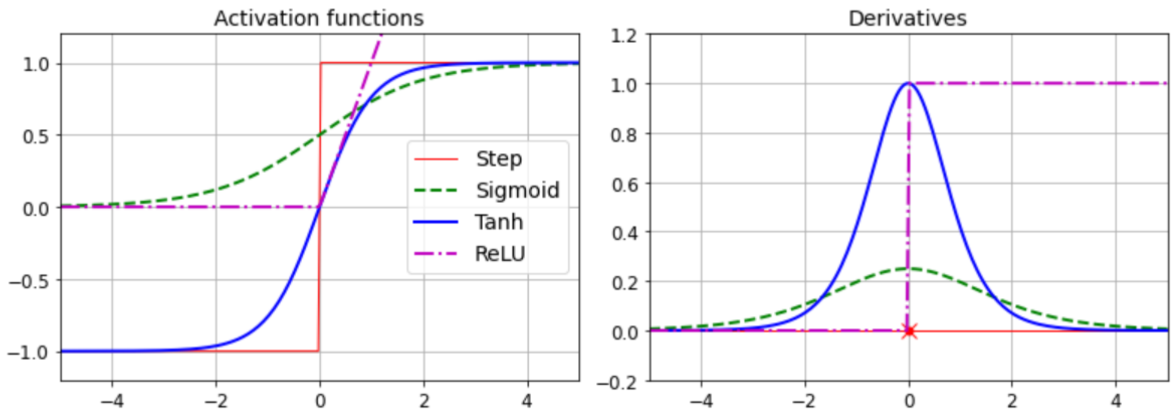
Artificial neural network

- Deep neural network
  - process of backpropagation
    - during an epoch
    - forward pass: every minibatch is injected to deep model to calculate the result of final ouput
    - backpropagation: measure the contribution of weight of previous layer
    - gradient descent: adjust weights to reduce errors

Gyeongsang National University 경상국립대학교

Artificial neural network

- Deep neural network
  - step function
    - logistic function
    - hyperbolic tangent
    - ReLU



Artificial neural network

- Deep neural network
  - regression
    - derive results without activation function in the output neuron
    - loss function is usually mean square error

Gyeongsang
National
University
경상국립대학교

Artificial neural network

- Deep neural network
  - classification
    - derive predictive probabilities for classes
    - loss function is usually cross-entropy function

Gyeongsang
National
University
경상국립대학교

Implementation in keras

- Deep neural network in keras
  - high-level deep learning APIs that easily create, train, validation and run all kinds of neural networks
  - https://keras.io

Implementation in keras

- Image classifier using keras
  - data import

```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
40960/29515 [=========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [===============================================================================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

Implementation in keras

- Image classifier using keras
  - data shape and scaling

```
X_train_full.shape
```

(60000, 28, 28)

```
X_train_full.dtype
```

dtype('uint8')

```
plt.imshow(X_train[0], cmap="binary")
plt.axis('off')
plt.show()
```

```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

Implementation in keras

- Image classifier using keras
  - data example

Gyeongsang National University 경상국립대학교

Implementation in keras

- Image classifier using keras
  - making sequential layer

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

Implementation in keras

- Image classifier using keras
  - model summary

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

Implementation in keras

- Image classifier using keras
  - printing weights

```
weights, biases = hidden1.get_weights()
```

```
weights
```

```
array([[ 0.02448617, -0.00877795, -0.02189048, ..., -0.02766046,
         0.03859074, -0.06889391],
       [ 0.00476504, -0.03105379, -0.0586676 , ...,  0.00602964,
        -0.02763776, -0.04165364],
       [-0.06189284, -0.06901957,  0.07102345, ..., -0.04238207,
         0.07121518, -0.07331658],
       ...,
       [-0.03048757,  0.02155137, -0.05400612, ..., -0.00113463,
         0.00228987,  0.05581069],
       [ 0.07061854, -0.06960931,  0.07038955, ..., -0.00384101,
         0.00034875,  0.02878492],
       [-0.06022581,  0.01577859, -0.02585464, ..., -0.00527829,
         0.00272203, -0.06793761]], dtype=float32)
```

```
weights.shape
```

```
(784, 300)
```

Gyeongsang National University 경상국립대학교

Implementation in keras

- Image classifier using keras
  - compile (setting loss function and optimizer)

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

Gyeongsang National University 경상국립대학교

Implementation in keras

- Image classifier using keras
  - training

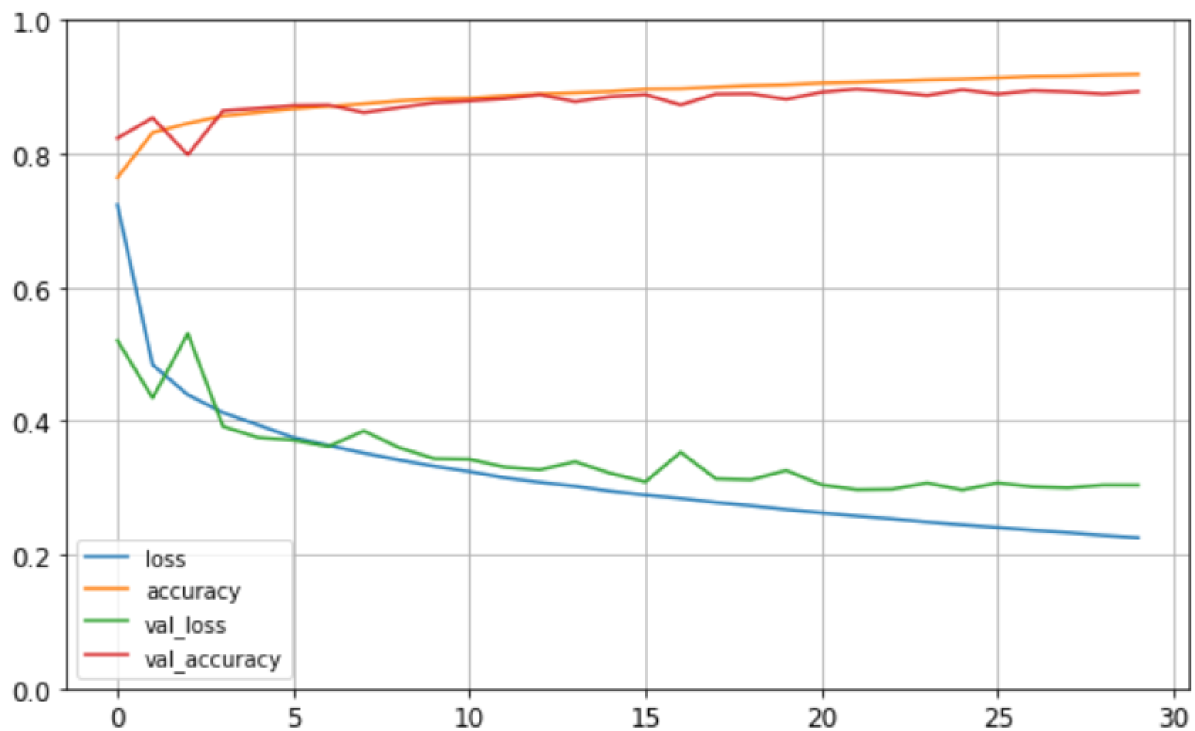
```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Epoch 1/30
1719/1719 [==============================] - 5s 2ms/step - loss: 0.7237 - accuracy: 0.7644 - val_loss: 0.5207 - val_accuracy: 0.8234
Epoch 2/30
1719/1719 [==============================] - 3s 2ms/step - loss: 0.4843 - accuracy: 0.8318 - val_loss: 0.4345 - val_accuracy: 0.8538
Epoch 3/30
1719/1719 [==============================] - 3s 2ms/step - loss: 0.4393 - accuracy: 0.8455 - val_loss: 0.5310 - val_accuracy: 0.7986
Epoch 4/30
1719/1719 [==============================] - 3s 2ms/step - loss: 0.4126 - accuracy: 0.8566 - val_loss: 0.3918 - val_accuracy: 0.8644
Epoch 5/30
1719/1719 [==============================] - 3s 2ms/step - loss: 0.3940 - accuracy: 0.8621 - val_loss: 0.3753 - val_accuracy: 0.8680
Epoch 6/30
1719/1719 [==============================] - 3s 2ms/step - loss: 0.3753 - accuracy: 0.8675 - val_loss: 0.3713 - val_accuracy: 0.8724
Epoch 7/30
1719/1719 [==============================] - 3s 2ms/step - loss: 0.3635 - accuracy: 0.8710 - val_loss: 0.3620 - val_accuracy: 0.8730
```

GNU Gyeongsang National University 경상국립대학교

Implementation in keras

- Image classifier using keras
  - learning curves

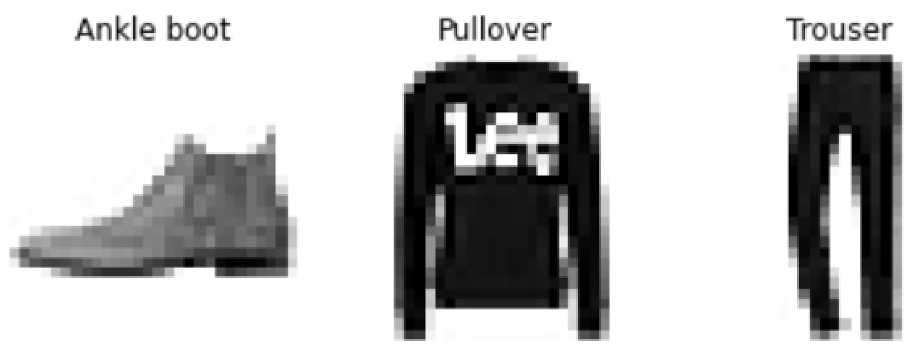


Implementation in keras

- Image classifier using keras
  - checking results of prediction

```
X_new = X_test[:3]
y_proba = model.predict(X_new)
y_proba.round(2)
```

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.03, 0.  , 0.96],
       [0.  , 0.  , 0.99, 0.  , 0.01, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
      dtype=float32)
```



Implementation in keras

- Regressor using keras
  - California housing price

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()

X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data, housing.target, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

Implementation in keras

- Regressor using keras
  - making sequential model and compile

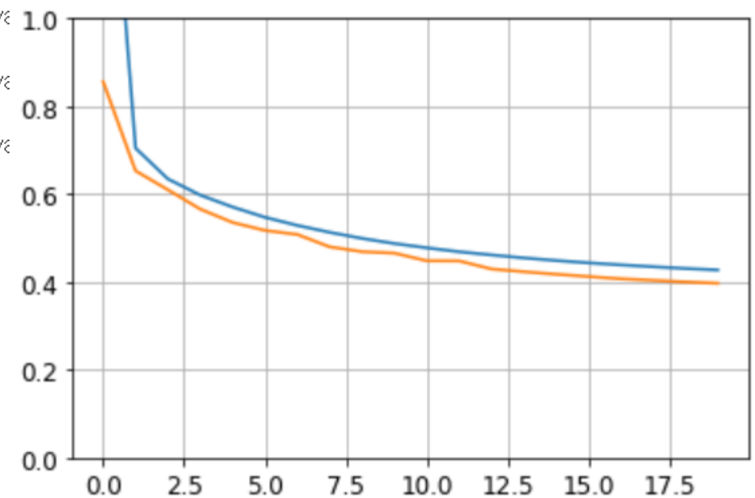
```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=X_train.shape[1:]),
    keras.layers.Dense(1)
])
model.compile(loss="mean_squared_error", optimizer=keras.optimizers.SGD(learning_rate=1e-3))
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid, y_valid))
mse_test = model.evaluate(X_test, y_test)
X_new = X_test[:3]
y_pred = model.predict(X_new)
```

Gyeongsang National University 경상국립대학교

Implementation in keras

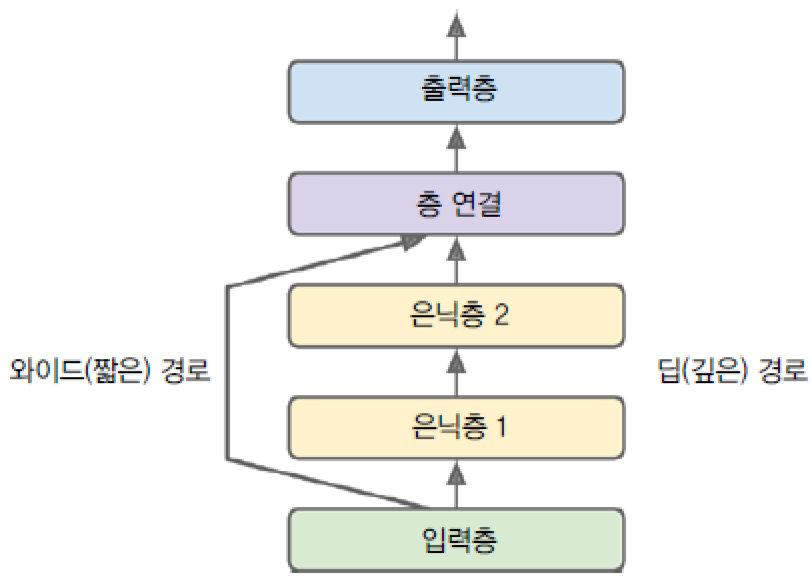
- Regressor using keras
  - training process

```
Epoch 1/20
363/363 [==============================] - 1s 2ms/step - loss: 1.6419 - val_loss: 0.8560
Epoch 2/20
363/363 [==============================] - 1s 2ms/step - loss: 0.7047 - val_loss: 0.6531
Epoch 3/20
363/363 [==============================] - 1s 2ms/step - loss: 0.6345 - va
Epoch 4/20
363/363 [==============================] - 1s 2ms/step - loss: 0.5977 - va
Epoch 5/20
363/363 [==============================] - 1s 2ms/step - loss: 0.5706 - va
```



Implementation in keras

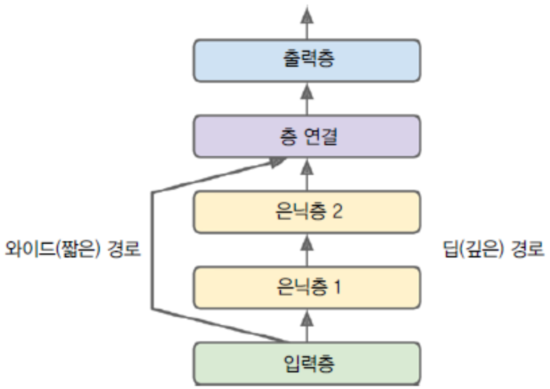
- Functional API model using keras
  - complex structure



Implementation in keras

- Functional API model using keras
  - complex structure

```
input_ = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input_)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input_], outputs=[output])
```

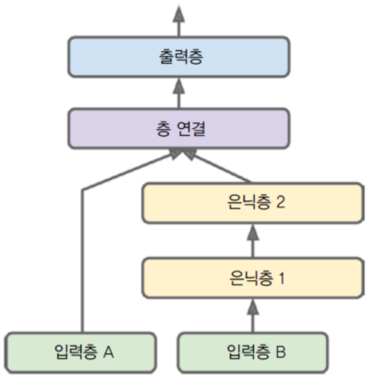


Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 8)]	0	
dense_5 (Dense)	(None, 30)	270	input_1[0][0]
dense_6 (Dense)	(None, 30)	930	dense_5[0][0]
concatenate (Concatenate)	(None, 38)	0	input_1[0][0] dense_6[0][0]
dense_7 (Dense)	(None, 1)	39	concatenate[0][0]

Gyeongsang National University 경상국립대학교

Implementation in keras

- Functional API model using keras
  - complex structure

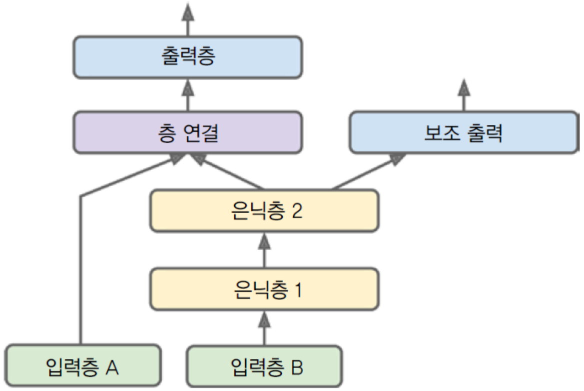



```
input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name="output")(concat)
model = keras.models.Model(inputs=[input_A, input_B], outputs=[output])
```

Gyeongsang National University 경상국립대학교

Implementation in keras

- Functional API model using keras
  - complex structure




```
input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name="main_output")(concat)
aux_output = keras.layers.Dense(1, name="aux_output")(hidden2)
model = keras.models.Model(inputs=[input_A, input_B],
                           outputs=[output, aux_output])
```

Gyeongsang National University 경상국립대학교

Implementation in keras

- Subclassing API model using keras
  - using the concept of object-oriented programming

```
class WideAndDeepModel(keras.models.Model):
    def __init__(self, units=30, activation="relu", **kwargs):
        super().__init__(**kwargs)
        self.hidden1 = keras.layers.Dense(units, activation=activation)
        self.hidden2 = keras.layers.Dense(units, activation=activation)
        self.main_output = keras.layers.Dense(1)
        self.aux_output = keras.layers.Dense(1)

    def call(self, inputs):
        input_A, input_B = inputs
        hidden1 = self.hidden1(input_B)
        hidden2 = self.hidden2(hidden1)
        concat = keras.layers.concatenate([input_A, hidden2])
        main_output = self.main_output(concat)
        aux_output = self.aux_output(hidden2)
        return main_output, aux_output

model = WideAndDeepModel(30, activation="relu")
```

Implementation in keras

- Saving and loading weights using keras

```
model.save("my_keras_model.h5")
```

```
model = keras.models.load_model("my_keras_model.h5")
```

```
model.predict(X_new)
```

```
array([[0.54002357],
       [1.6505971 ],
       [3.009824  ]], dtype=float32)
```

```
model.save_weights("my_keras_weights.ckpt")
```

```
model.load_weights("my_keras_weights.ckpt")
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fe063867690>
```

GNU Gyeongsang National University 경상국립대학교

Implementation in keras

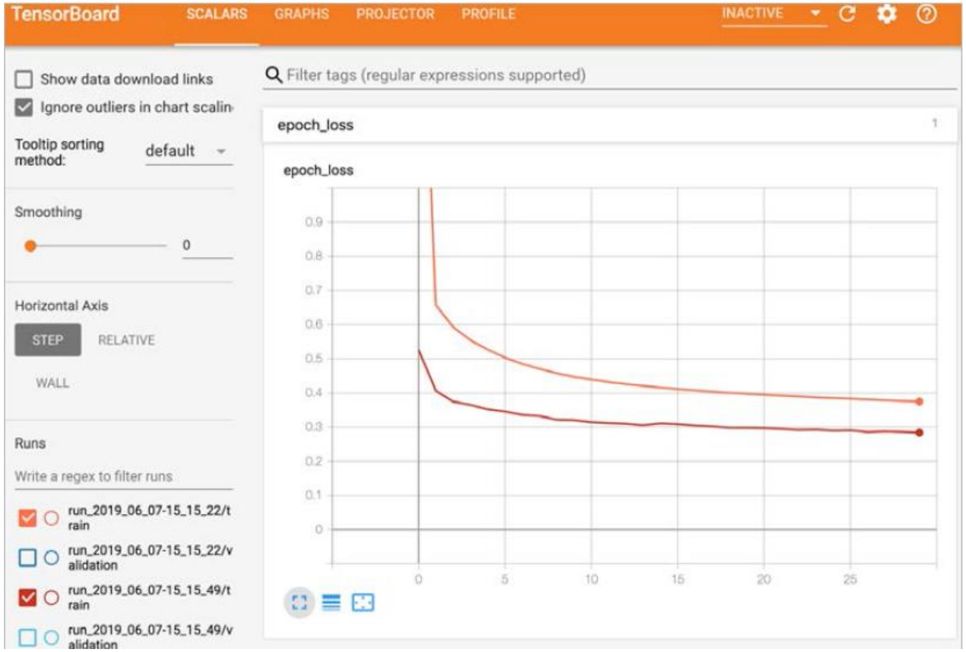
- Callback function using keras
    - specify some processes during training

```
model.compile(loss="mse", optimizer=keras.optimizers.SGD(learning_rate=1e-3))
checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.h5", save_best_only=True)
history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb])
model = keras.models.load_model("my_keras_model.h5") # 최상의 모델로 롤백
mse_test = model.evaluate(X_test, y_test)
```

Gyeongsang National University 경상국립대학교

Implementation in keras

- Visualization using tensorboard
  - specify some processes during training



Hyperparameter tuning

- Flexibility of deep neural network
  - many hyperparameters
    - the number of layer
    - the number of neurons in each layer
    - type of activation function
    - weight initialization

Hyperparameter tuning

- Finding optimal hyperparameter
    - using GridSearchCV or RandomizedSearchCV

```
def build_model(n_hidden=1, n_neurons=30, learning_rate=3e-3, input_shape=[8]):
    model = keras.models.Sequential()
    model.add(keras.layers.InputLayer(input_shape=input_shape))
    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation="relu"))
    model.add(keras.layers.Dense(1))
    optimizer = keras.optimizers.SGD(learning_rate=learning_rate)
    model.compile(loss="mse", optimizer=optimizer)
    return model
```

```
keras_reg = keras.wrappers.scikit_learn.KerasRegressor(build_model)
```

```
keras_reg.fit(X_train, y_train, epochs=100,
              validation_data=(X_valid, y_valid),
              callbacks=[keras.callbacks.EarlyStopping(patience=10)])
```

Gyeongsang National University 경상국립대학교

Hyperparameter tuning

- Finding optimal hyperparameter
  - using GridSearchCV or RandomizedSearchCV

```
from scipy.stats import reciprocal
from sklearn.model_selection import RandomizedSearchCV

param_distribs = {
    "n_hidden": [0, 1, 2, 3],
    "n_neurons": np.arange(1, 100).tolist(),
    "learning_rate": reciprocal(3e-4, 3e-2).rvs(1000).tolist(),
}

rnd_search_cv = RandomizedSearchCV(keras_reg, param_distribs, n_iter=10, cv=3, verbose=2)
rnd_search_cv.fit(X_train, y_train, epochs=100,
                  validation_data=(X_valid, y_valid),
                  callbacks=[keras.callbacks.EarlyStopping(patience=10)])
```

Gyeongsang National University 경상국립대학교

Hyperparameter tuning

- Guidelines for hyperparameters in DNN
  - number of layers
    - complex situation can be modeled by deeper layer with fewer neurons
    - can be designed to fit the physical meaning of dataset
  - number of neurons
    - common to use fewer neurons as the data goes to the upper layers

Hyperparameter tuning

- Guidelines for hyperparameters in DNN
  - the number of layers
    - complex situation can be modeled by deeper layer with fewer neurons
    - can be designed to fit the physical meaning of dataset

Hyperparameter tuning

- Guidelines for hyperparameters in DNN
  - the number of neurons
    - traditionally, it is common to use fewer neurons as the data goes to the upper layers
    - nowadays, the number of neurons is freely determined

Hyperparameter tuning

- Guidelines for hyperparameters in DNN
  - learning rate
  - optimizer
  - the size of batch
  - activation function
  - the number of epochs

Feel free to question

Through e-mail & LMS

본 자료의 연습문제는 수업의 본교재인
한빛미디어, Hands on Machine Learning(2판)에서 주로 발췌함