[model\_fn]

example: (the multiple layer perceptron cnn model for mnist)

def model\_fn(features, labels, mode, params):

# connect the model with input layers

net = tf.feature\_column.input\_layer(features, params[“feature\_column”])

# define model

net = tf.keras.layers.Flatten()(net)

for i in range(params[“n\_hidden\_layer”]):

net = tf.keras.layers.Dense(units=params[“n\_neurons”], activation=tf.nn.relu)(net)

logits = tf.keras.layers.Dense(units=params[“n\_class”], activation=None)(net)

# define prediction classes

predicted\_classes = tf.argmax(logit, axis=1)

# define prediction mode

if mode == tf.estimator.ModeKeys.PREDICT:

prediction = {

“class\_ids”: predicted\_classes[:, tf.newaxis],

“probability”: tf.nn.softmax(logits),

“logits”: logits

}

return tf.estimator,EstimatorSpec(mode, prediction)

# define accuracy

accuracy = tf.metrics.accuracy(labels=labels,

prediction=predicted\_classes,

name= “acc\_op”)

# add accuracy in tensorboard

tf.summary.scalar(“accuracy”, accuracy[1])

# define evaluation mode

if mode == tf.estimator.ModeKeys.EVAL:

metrics = {“accuracy”: accuracy}

return tf.estimator.EstimatorSpec(mode, metrics)

# define loss

loss = tf.losses.sparse\_softmax\_cross\_entropy(logits=logits, labels=labels)

# define training operation

optimizer = tf.train.AdamOptimizer(learning\_rate=params[“lr”])

train\_op = optimizer.minimize(loss, global\_step=tf.train.get\_global\_step())

# define train mode

if mode == tf.estimator.ModeKeys.TRAIN:

return tf.estimator.EstimatoSpec(mode, loss=loss, train\_op=train\_op)

* variable chart

