1. Deep Learning.

a. Build a DNN with five hidden layers of 100 neurons each, He initialization, and the ELU activation function.

b. Using Adam optimization and early stopping, try training it on MNIST but only on digits 0 to 4, as we will use transfer learning for digits 5 to 9 in the next exercise. You will need a softmax output layer with five neurons, and as always make sure to save checkpoints at regular intervals and save the final model so you can reuse it later.

c. Tune the hyperparameters using cross-validation and see what precision you can achieve.

d. Now try adding Batch Normalization and compare the learning curves: is it converging faster than before? Does it produce a better model?

e. Is the model overfitting the training set? Try adding dropout to every layer and try again. Does it help?

Ans: a. Build a DNN with five hidden layers of 100 neurons each, He initialization, and the ELU activation function.

In [1]:

import tensorflow as tf

from functools import partial

In [2]:

*# Define the number of inputs and outputs*

n\_inputs = 28 \* 28 *# MNIST*

n\_outputs = 10

In [3]:

*# Define the number of neurons in each hidden layer*

n\_hidden1 = 100

n\_hidden2 = 100

n\_hidden3 = 100

n\_hidden4 = 100

n\_hidden5 = 100

In [4]:

*# Define the initializer and activation function to use*

he\_init = tf.keras.initializers.VarianceScaling(scale=2., mode='fan\_avg', distribution='uniform')

elu\_activation = tf.keras.activations.elu

In [5]:

*# Create the model using the Sequential API*

model = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=[28, 28]), *# Flatten the input data*

tf.keras.layers.Dense(n\_hidden1, activation=elu\_activation, kernel\_initializer=he\_init), *# First hidden layer*

tf.keras.layers.Dense(n\_hidden2, activation=elu\_activation, kernel\_initializer=he\_init), *# Second hidden layer*

tf.keras.layers.Dense(n\_hidden3, activation=elu\_activation, kernel\_initializer=he\_init), *# Third hidden layer*

tf.keras.layers.Dense(n\_hidden4, activation=elu\_activation, kernel\_initializer=he\_init), *# Fourth hidden layer*

tf.keras.layers.Dense(n\_hidden5, activation=elu\_activation, kernel\_initializer=he\_init), *# Fifth hidden layer*

tf.keras.layers.Dense(n\_outputs) *# Output layer*

])

b. Using Adam optimization and early stopping, try training it on MNIST but only on digits 0 to 4, as we will use transfer learning for digits 5 to 9 in the next exercise. You will need a softmax output layer with five neurons, and as always make sure to save checkpoints at regular intervals and save the final model so you can reuse it later.

Ans:

In [6]:

from tensorflow import keras

import numpy as np

In [7]:

*# Load MNIST dataset*

(X\_train\_full, y\_train\_full), (X\_test, y\_test) = keras.datasets.mnist.load\_data()

In [8]:

*# Scale the data*

X\_train\_full = X\_train\_full / 255.0

X\_test = X\_test / 255.0

In [9]:

*# Create a validation set and training set*

X\_valid, X\_train = X\_train\_full[:5000], X\_train\_full[5000:]

y\_valid, y\_train = y\_train\_full[:5000], y\_train\_full[5000:]

In [10]:

*# Only keep digits 0 to 4 in the training and validation sets*

X\_train = X\_train[y\_train < 5]

y\_train = y\_train[y\_train < 5]

X\_valid = X\_valid[y\_valid < 5]

y\_valid = y\_valid[y\_valid < 5]

In [11]:

*# Build the model*

model = keras.models.Sequential([

keras.layers.Flatten(input\_shape=[28, 28]),

keras.layers.Dense(300, activation="relu"),

keras.layers.Dense(100, activation="relu"),

keras.layers.Dense(5, activation="softmax")

])

In [12]:

*# Compile the model*

model.compile(loss="sparse\_categorical\_crossentropy",

optimizer="adam",

metrics=["accuracy"])

In [13]:

*# Define early stopping*

early\_stopping\_cb = keras.callbacks.EarlyStopping(patience=10,

restore\_best\_weights=True)

In [14]:

*# Train the model*

history = model.fit(X\_train, y\_train, epochs=30,

validation\_data=(X\_valid, y\_valid),

callbacks=[early\_stopping\_cb])

Epoch 1/30

877/877 [==============================] - 6s 6ms/step - loss: 0.1940 - accuracy: 0.9386 - val\_loss: 0.0424 - val\_accuracy: 0.9859

Epoch 2/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0374 - accuracy: 0.9893 - val\_loss: 0.0356 - val\_accuracy: 0.9867

Epoch 3/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0218 - accuracy: 0.9930 - val\_loss: 0.0346 - val\_accuracy: 0.9891

Epoch 4/30

877/877 [==============================] - 4s 4ms/step - loss: 0.0141 - accuracy: 0.9958 - val\_loss: 0.0381 - val\_accuracy: 0.9883

Epoch 5/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0122 - accuracy: 0.9962 - val\_loss: 0.0345 - val\_accuracy: 0.9898

Epoch 6/30

877/877 [==============================] - 4s 4ms/step - loss: 0.0086 - accuracy: 0.9973 - val\_loss: 0.0272 - val\_accuracy: 0.9918

Epoch 7/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0083 - accuracy: 0.9971 - val\_loss: 0.0312 - val\_accuracy: 0.9918

Epoch 8/30

877/877 [==============================] - 4s 4ms/step - loss: 0.0034 - accuracy: 0.9992 - val\_loss: 0.0277 - val\_accuracy: 0.9941

Epoch 9/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0062 - accuracy: 0.9982 - val\_loss: 0.0251 - val\_accuracy: 0.9937

Epoch 10/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0055 - accuracy: 0.9983 - val\_loss: 0.0398 - val\_accuracy: 0.9906

Epoch 11/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0083 - accuracy: 0.9978 - val\_loss: 0.0324 - val\_accuracy: 0.9922

Epoch 12/30

877/877 [==============================] - 4s 4ms/step - loss: 0.0047 - accuracy: 0.9983 - val\_loss: 0.0278 - val\_accuracy: 0.9930

Epoch 13/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0042 - accuracy: 0.9987 - val\_loss: 0.0393 - val\_accuracy: 0.9898

Epoch 14/30

877/877 [==============================] - 3s 4ms/step - loss: 0.0051 - accuracy: 0.9983 - val\_loss: 0.0398 - val\_accuracy: 0.9918

Epoch 15/30

877/877 [==============================] - 3s 4ms/step - loss: 5.5724e-04 - accuracy: 1.0000 - val\_loss: 0.0409 - val\_accuracy: 0.9930

Epoch 16/30

877/877 [==============================] - 4s 4ms/step - loss: 9.6116e-05 - accuracy: 1.0000 - val\_loss: 0.0396 - val\_accuracy: 0.9930

Epoch 17/30

877/877 [==============================] - 3s 4ms/step - loss: 1.7608e-05 - accuracy: 1.0000 - val\_loss: 0.0384 - val\_accuracy: 0.9934

Epoch 18/30

877/877 [==============================] - 4s 4ms/step - loss: 9.7778e-06 - accuracy: 1.0000 - val\_loss: 0.0385 - val\_accuracy: 0.9934

Epoch 19/30

877/877 [==============================] - 3s 4ms/step - loss: 6.5843e-06 - accuracy: 1.0000 - val\_loss: 0.0383 - val\_accuracy: 0.9934

In [15]:

*# Save the final model*

model.save("my\_mnist\_model.h5")

c. Tune the hyperparameters using cross-validation and see what precision you can achieve.

Ans:

In [16]:

from tensorflow import keras

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import precision\_score, make\_scorer

import numpy as np

In [17]:

*# Load MNIST dataset*

(X\_train\_full, y\_train\_full), (X\_test, y\_test) = keras.datasets.mnist.load\_data()

In [18]:

*# Scale the data*

X\_train\_full = X\_train\_full / 255.0

X\_test = X\_test / 255.0

In [19]:

*# Create a validation set and training set*

X\_valid, X\_train = X\_train\_full[:5000], X\_train\_full[5000:]

y\_valid, y\_train = y\_train\_full[:5000], y\_train\_full[5000:]

In [20]:

*# Only keep digits 0 to 4 in the training and validation sets*

X\_train = X\_train[y\_train < 5]

y\_train = y\_train[y\_train < 5]

X\_valid = X\_valid[y\_valid < 5]

y\_valid = y\_valid[y\_valid < 5]

In [21]:

*# Build a function to create the model*

def build\_model(n\_hidden=1, n\_neurons=30, learning\_rate=3e-3):

model = keras.models.Sequential()

model.add(keras.layers.Flatten(input\_shape=[28, 28]))

for layer in range(n\_hidden):

model.add(keras.layers.Dense(n\_neurons, activation="relu"))

model.add(keras.layers.Dense(5, activation="softmax"))

optimizer = keras.optimizers.Adam(lr=learning\_rate)

model.compile(loss="sparse\_categorical\_crossentropy", optimizer=optimizer,

metrics=["accuracy"])

return model

In [22]:

*# Create a KerasClassifier object*

keras\_clf = keras.wrappers.scikit\_learn.KerasClassifier(build\_model)

In [23]:

*# Define the hyperparameters to search over*

param\_grid = {

"n\_hidden": [1, 2],

"n\_neurons": [10, 30],

"learning\_rate": [3e-4, 3e-3]

}

In [24]:

*# Create a GridSearchCV object*

grid\_search\_cv = GridSearchCV(keras\_clf, param\_grid,

cv=3,

scoring=make\_scorer(precision\_score,

average='weighted'))

In [25]:

*# Fit the GridSearchCV object to the data*

grid\_search\_cv.fit(X\_train, y\_train,

validation\_data=(X\_valid, y\_valid),

callbacks=[keras.callbacks.EarlyStopping(patience=10)])