ASSGN_3_PART_1

February 20, 2022

[1]:
```
# MLFA Assignment 3

# Team Members -
# Deepansh Agrawal - 19MI10018
# Rohit Ranjan - 20CS30066
# Neha Gupta - 20CH10094
# Gautam Jaju - 20AG30015
# Siddharth Madhupati - 20ME30083
# Madiha Hanifa - 20MF10018
# Himadri Pandya - 20ME10047
```

[2]:
```
# import commands
import numpy as np
import matplotlib.pyplot as plt
import random
import tensorflow as tf
from tensorflow import keras
from keras.layers.core import Dense, Dropout, Activation
from keras.layers import BatchNormalization
from keras.utils import np_utils

# loading the dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

[3]:
```
# reshaping the input data to the range 0 - 1
x_train = x_train.reshape(-1, 28 * 28).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28 * 28).astype("float32") / 255.0
```

[4]:
```
print(x_train.shape)
print(y_train.shape)
```

```
(60000, 784)
(60000,)
```

[5]:
```
# Implementation of the uniform network
# There are 88 neurons in each layer except the first layer and the output layer
# The parameters of the input layer (39250) and first layer (4488) are␣
 ↪discounted
```

```
# The number of parameters is constant after that
# The number of parameters is different in the last hidden layer as specified␣
 ↪in the problem
```

[6]:
```
initializer = tf.keras.initializers.GlorotNormal()
UniformNet = keras.Sequential([
                                keras.layers.InputLayer((784)),
                                keras.layers.Dense(50, activation='relu',␣
 ↪name='first_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(88, activation='relu',␣
 ↪name='second_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(88, activation='relu',␣
 ↪name='third_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(88, activation='relu',␣
 ↪name='fourth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
                                keras.layers.Dense(88, activation='relu',␣
 ↪name='fifth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
                                keras.layers.Dense(88, activation='relu',␣
 ↪name='sixth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
                                keras.layers.Dense(88, activation='relu',␣
 ↪name='seventh_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(88, activation='relu',␣
 ↪name='eighth_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(88, activation='relu',␣
 ↪name='nineth_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(10, activation='softmax',␣
 ↪name='output_layer', kernel_initializer=initializer)
])
```

[7]:
```
UniformNet.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 first_layer (Dense)         (None, 50)                39250

 batch_normalization (BatchN  (None, 50)               200
 ormalization)

 second_layer (Dense)        (None, 88)                4488

 batch_normalization_1 (Batc  (None, 88)               352
 hNormalization)
```

```
 third_layer (Dense)         (None, 88)                7832

 batch_normalization_2 (Batc  (None, 88)               352
 hNormalization)

 fourth_layer (Dense)        (None, 88)                7832

 dropout (Dropout)           (None, 88)                0

 batch_normalization_3 (Batc  (None, 88)               352
 hNormalization)

 fifth_layer (Dense)         (None, 88)                7832

 dropout_1 (Dropout)         (None, 88)                0

 batch_normalization_4 (Batc  (None, 88)               352
 hNormalization)

 sixth_layer (Dense)         (None, 88)                7832

 dropout_2 (Dropout)         (None, 88)                0

 batch_normalization_5 (Batc  (None, 88)               352
 hNormalization)

 seventh_layer (Dense)       (None, 88)                7832

 batch_normalization_6 (Batc  (None, 88)               352
 hNormalization)

 eighth_layer (Dense)        (None, 88)                7832

 batch_normalization_7 (Batc  (None, 88)               352
 hNormalization)

 nineth_layer (Dense)        (None, 88)                7832

 batch_normalization_8 (Batc  (None, 88)               352
 hNormalization)

 output_layer (Dense)        (None, 10)                890

=================================================================
Total params: 102,468
Trainable params: 100,960
Non-trainable params: 1,508
```

```
-----------------------------------------------------------------
```

[8]:
```
UniformNet.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(),
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    metrics=["accuracy"],
)
```

[9]:
```
UniformNet.fit(x_train, y_train, batch_size=128, epochs=20, verbose=1)
UniformNet.evaluate(x_test, y_test, batch_size=128, verbose=1)
```

```
Epoch 1/20
469/469 [==============================] - 10s 15ms/step - loss: 2.3375 -
accuracy: 0.2103
Epoch 2/20
469/469 [==============================] - 4s 9ms/step - loss: 1.6245 -
accuracy: 0.4308
Epoch 3/20
469/469 [==============================] - 4s 9ms/step - loss: 1.1427 -
accuracy: 0.6094
Epoch 4/20
469/469 [==============================] - 4s 9ms/step - loss: 0.8326 -
accuracy: 0.7289
Epoch 5/20
469/469 [==============================] - 4s 8ms/step - loss: 0.6374 -
accuracy: 0.8036
Epoch 6/20
469/469 [==============================] - 4s 9ms/step - loss: 0.5170 -
accuracy: 0.8464
Epoch 7/20
469/469 [==============================] - 4s 9ms/step - loss: 0.4356 -
accuracy: 0.8745
Epoch 8/20
469/469 [==============================] - 4s 9ms/step - loss: 0.3830 -
accuracy: 0.8918
Epoch 9/20
469/469 [==============================] - 4s 9ms/step - loss: 0.3396 -
accuracy: 0.9048
Epoch 10/20
469/469 [==============================] - 4s 9ms/step - loss: 0.3092 -
accuracy: 0.9142
Epoch 11/20
469/469 [==============================] - 4s 9ms/step - loss: 0.2814 -
accuracy: 0.9223
Epoch 12/20
469/469 [==============================] - 4s 8ms/step - loss: 0.2645 -
accuracy: 0.9281
Epoch 13/20
```

```
469/469 [==============================] - 4s 8ms/step - loss: 0.2478 -
accuracy: 0.9330
Epoch 14/20
469/469 [==============================] - 4s 9ms/step - loss: 0.2300 -
accuracy: 0.9379
Epoch 15/20
469/469 [==============================] - 4s 9ms/step - loss: 0.2185 -
accuracy: 0.9418
Epoch 16/20
469/469 [==============================] - 4s 9ms/step - loss: 0.2094 -
accuracy: 0.9442
Epoch 17/20
469/469 [==============================] - 4s 9ms/step - loss: 0.1960 -
accuracy: 0.9481
Epoch 18/20
469/469 [==============================] - 4s 8ms/step - loss: 0.1858 -
accuracy: 0.9499
Epoch 19/20
469/469 [==============================] - 4s 9ms/step - loss: 0.1771 -
accuracy: 0.9525
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.1685 -
accuracy: 0.9547
79/79 [==============================] - 1s 3ms/step - loss: 0.1397 - accuracy:
0.9632
```

[9]:
```
[0.1396992951631546, 0.9631999731063843]
```

[10]:
```
# Implementation of the pyramid network
# There are varying number of neurons in each layer
# The parameters of the input layer (39250) and first layer (200) are discounted
# The number of parameters falls by a factor of two after that
# The number of parameters is different in the last hidden layer as specified␣
 ↪in the problem
```

[11]:
```
initializer = tf.keras.initializers.GlorotNormal()
PyramidNet = keras.Sequential([
                              keras.layers.InputLayer((784)),
                              keras.layers.Dense(50, activation='relu',␣
 ↪name='first_layer', kernel_initializer=initializer), BatchNormalization(),
                              keras.layers.Dense(16, activation='relu',␣
 ↪name='second_layer', kernel_initializer=initializer), BatchNormalization(),
                              keras.layers.Dense(1250, activation='relu',␣
 ↪name='third_layer', kernel_initializer=initializer), BatchNormalization(),
                              keras.layers.Dense(8, activation='relu',␣
 ↪name='fourth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
```

```
                                keras.layers.Dense(625, activation='relu',␣
 ↪name='fifth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
                                keras.layers.Dense(4, activation='relu',␣
 ↪name='sixth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
                                keras.layers.Dense(312, activation='relu',␣
 ↪name='seventh_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(2, activation='relu',␣
 ↪name='eighth_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(156, activation='relu',␣
 ↪name='nineth_layer', kernel_initializer=initializer), BatchNormalization(),
                                keras.layers.Dense(10, activation='softmax',␣
 ↪name='output_layer', kernel_initializer=initializer)
])
```

[12]:
```
PyramidNet.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 first_layer (Dense)         (None, 50)                39250

 batch_normalization_9 (Batc  (None, 50)               200
 hNormalization)

 second_layer (Dense)        (None, 16)                816

 batch_normalization_10 (Bat  (None, 16)               64
 chNormalization)

 third_layer (Dense)         (None, 1250)              21250

 batch_normalization_11 (Bat  (None, 1250)             5000
 chNormalization)

 fourth_layer (Dense)        (None, 8)                 10008

 dropout_3 (Dropout)         (None, 8)                 0

 batch_normalization_12 (Bat  (None, 8)                32
 chNormalization)

 fifth_layer (Dense)         (None, 625)               5625

 dropout_4 (Dropout)         (None, 625)               0
```

```
batch_normalization_13 (Bat  (None, 625)              2500
chNormalization)

sixth_layer (Dense)          (None, 4)                2504

dropout_5 (Dropout)          (None, 4)                0

batch_normalization_14 (Bat  (None, 4)                16
chNormalization)

seventh_layer (Dense)        (None, 312)              1560

batch_normalization_15 (Bat  (None, 312)              1248
chNormalization)

eighth_layer (Dense)         (None, 2)                626

batch_normalization_16 (Bat  (None, 2)                8
chNormalization)

nineth_layer (Dense)         (None, 156)              468

batch_normalization_17 (Bat  (None, 156)              624
chNormalization)

output_layer (Dense)         (None, 10)               1570

=================================================================
Total params: 93,369
Trainable params: 88,523
Non-trainable params: 4,846
_________________________________________________________________
```

[13]:
```
PyramidNet.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(),
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    metrics=["accuracy"],
)
```

[14]:
```
PyramidNet.fit(x_train, y_train, batch_size=128, epochs=20, verbose=1)
PyramidNet.evaluate(x_test, y_test, batch_size=128, verbose=1)
```

```
Epoch 1/20
469/469 [==============================] - 10s 16ms/step - loss: 2.2005 -
accuracy: 0.1786
Epoch 2/20
469/469 [==============================] - 7s 16ms/step - loss: 1.8831 -
```

```
accuracy: 0.2492
Epoch 3/20
469/469 [==============================] - 7s 15ms/step - loss: 1.6977 -
accuracy: 0.3248
Epoch 4/20
469/469 [==============================] - 7s 16ms/step - loss: 1.5016 -
accuracy: 0.3735
Epoch 5/20
469/469 [==============================] - 7s 15ms/step - loss: 1.4055 -
accuracy: 0.4181
Epoch 6/20
469/469 [==============================] - 7s 15ms/step - loss: 1.3172 -
accuracy: 0.4679
Epoch 7/20
469/469 [==============================] - 7s 15ms/step - loss: 1.2528 -
accuracy: 0.5031
Epoch 8/20
469/469 [==============================] - 7s 15ms/step - loss: 1.2146 -
accuracy: 0.5209
Epoch 9/20
469/469 [==============================] - 7s 15ms/step - loss: 1.1708 -
accuracy: 0.5349
Epoch 10/20
469/469 [==============================] - 7s 16ms/step - loss: 1.1302 -
accuracy: 0.5505
Epoch 11/20
469/469 [==============================] - 7s 15ms/step - loss: 1.0979 -
accuracy: 0.5633
Epoch 12/20
469/469 [==============================] - 7s 15ms/step - loss: 1.0546 -
accuracy: 0.5766
Epoch 13/20
469/469 [==============================] - 7s 15ms/step - loss: 1.0418 -
accuracy: 0.5830
Epoch 14/20
469/469 [==============================] - 7s 15ms/step - loss: 1.0223 -
accuracy: 0.5849
Epoch 15/20
469/469 [==============================] - 7s 15ms/step - loss: 0.9996 -
accuracy: 0.5920
Epoch 16/20
469/469 [==============================] - 7s 15ms/step - loss: 0.9871 -
accuracy: 0.5903
Epoch 17/20
469/469 [==============================] - 7s 16ms/step - loss: 0.9787 -
accuracy: 0.5960
Epoch 18/20
469/469 [==============================] - 7s 16ms/step - loss: 0.9618 -
```

```
accuracy: 0.5973
Epoch 19/20
469/469 [==============================] - 7s 16ms/step - loss: 0.9544 -
accuracy: 0.6003
Epoch 20/20
469/469 [==============================] - 7s 16ms/step - loss: 0.9478 -
accuracy: 0.5988
79/79 [==============================] - 1s 4ms/step - loss: 2.3110 - accuracy:
0.2793
```

[14]:
```
[2.311028480529785, 0.2793000042438507]
```

[15]:
```
# Implementation of the pyramid network
# There are varying number of neurons in each layer
# The parameters of the input layer (39250) and first layer (255) are discounted
# The number of parameters increases by a factor of two after that
# The number of parameters is different in the last hidden layer as specified␣
 ↪in the problem
```

[16]:
```
initializer = tf.keras.initializers.GlorotNormal()
InvPyramidNet = keras.Sequential([
                                  keras.layers.InputLayer((784)),
                                  keras.layers.Dense(50, activation='relu',␣
 ↪name='first_layer', kernel_initializer=initializer), BatchNormalization(),
                                  keras.layers.Dense(5, activation='relu',␣
 ↪name='second_layer', kernel_initializer=initializer), BatchNormalization(),
                                  keras.layers.Dense(94, activation='relu',␣
 ↪name='third_layer', kernel_initializer=initializer), BatchNormalization(),
                                  keras.layers.Dense(10, activation='relu',␣
 ↪name='fourth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
                                  keras.layers.Dense(188, activation='relu',␣
 ↪name='fifth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
                                  keras.layers.Dense(20, activation='relu',␣
 ↪name='sixth_layer', kernel_initializer=initializer), Dropout(0.3),␣
 ↪BatchNormalization(),
                                  keras.layers.Dense(376, activation='relu',␣
 ↪name='seventh_layer', kernel_initializer=initializer), BatchNormalization(),
                                  keras.layers.Dense(40, activation='relu',␣
 ↪name='eighth_layer', kernel_initializer=initializer), BatchNormalization(),
                                  keras.layers.Dense(750, activation='relu',␣
 ↪name='nineth_layer', kernel_initializer=initializer), BatchNormalization(),
                                  keras.layers.Dense(10, activation='softmax',␣
 ↪name='output_layer', kernel_initializer=initializer)
])
```

[17]:
```
InvPyramidNet.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 first_layer (Dense)         (None, 50)                39250

 batch_normalization_18 (Bat  (None, 50)               200
 chNormalization)

 second_layer (Dense)        (None, 5)                 255

 batch_normalization_19 (Bat  (None, 5)                20
 chNormalization)

 third_layer (Dense)         (None, 94)                564

 batch_normalization_20 (Bat  (None, 94)               376
 chNormalization)

 fourth_layer (Dense)        (None, 10)                950

 dropout_6 (Dropout)         (None, 10)                0

 batch_normalization_21 (Bat  (None, 10)               40
 chNormalization)

 fifth_layer (Dense)         (None, 188)               2068

 dropout_7 (Dropout)         (None, 188)               0

 batch_normalization_22 (Bat  (None, 188)              752
 chNormalization)

 sixth_layer (Dense)         (None, 20)                3780

 dropout_8 (Dropout)         (None, 20)                0

 batch_normalization_23 (Bat  (None, 20)               80
 chNormalization)

 seventh_layer (Dense)       (None, 376)               7896

 batch_normalization_24 (Bat  (None, 376)              1504
 chNormalization)
```

```
 eighth_layer (Dense)        (None, 40)                15080

 batch_normalization_25 (Bat  (None, 40)               160
 chNormalization)

 nineth_layer (Dense)        (None, 750)               30750

 batch_normalization_26 (Bat  (None, 750)              3000
 chNormalization)

 output_layer (Dense)        (None, 10)                7510

=================================================================
Total params: 114,235
Trainable params: 111,169
Non-trainable params: 3,066
_________________________________________________________________
```

[18]:
```
InvPyramidNet.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(),
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    metrics=["accuracy"],
)
```

[19]:
```
InvPyramidNet.fit(x_train, y_train, batch_size=128, epochs=20, verbose=1)
InvPyramidNet.evaluate(x_test, y_test, batch_size=128, verbose=1)
```

```
Epoch 1/20
469/469 [==============================] - 8s 13ms/step - loss: 2.1832 -
accuracy: 0.2240
Epoch 2/20
469/469 [==============================] - 6s 12ms/step - loss: 1.4959 -
accuracy: 0.4511
Epoch 3/20
469/469 [==============================] - 6s 12ms/step - loss: 1.1774 -
accuracy: 0.5608
Epoch 4/20
469/469 [==============================] - 6s 12ms/step - loss: 1.0142 -
accuracy: 0.6153
Epoch 5/20
469/469 [==============================] - 6s 12ms/step - loss: 0.9120 -
accuracy: 0.6561
Epoch 6/20
469/469 [==============================] - 6s 13ms/step - loss: 0.8488 -
accuracy: 0.6820
Epoch 7/20
469/469 [==============================] - 6s 13ms/step - loss: 0.7879 -
accuracy: 0.6998
```

```
Epoch 8/20
469/469 [==============================] - 6s 13ms/step - loss: 0.7367 -
accuracy: 0.7251
Epoch 9/20
469/469 [==============================] - 6s 13ms/step - loss: 0.7038 -
accuracy: 0.7403
Epoch 10/20
469/469 [==============================] - 6s 12ms/step - loss: 0.6601 -
accuracy: 0.7586
Epoch 11/20
469/469 [==============================] - 6s 12ms/step - loss: 0.6385 -
accuracy: 0.7674
Epoch 12/20
469/469 [==============================] - 6s 13ms/step - loss: 0.5991 -
accuracy: 0.7829
Epoch 13/20
469/469 [==============================] - 6s 13ms/step - loss: 0.5774 -
accuracy: 0.7957
Epoch 14/20
469/469 [==============================] - 6s 12ms/step - loss: 0.5558 -
accuracy: 0.8032
Epoch 15/20
469/469 [==============================] - 6s 12ms/step - loss: 0.5327 -
accuracy: 0.8132
Epoch 16/20
469/469 [==============================] - 6s 12ms/step - loss: 0.5107 -
accuracy: 0.8351
Epoch 17/20
469/469 [==============================] - 6s 12ms/step - loss: 0.4901 -
accuracy: 0.8507
Epoch 18/20
469/469 [==============================] - 6s 13ms/step - loss: 0.4682 -
accuracy: 0.8612
Epoch 19/20
469/469 [==============================] - 6s 12ms/step - loss: 0.4508 -
accuracy: 0.8694
Epoch 20/20
469/469 [==============================] - 6s 12ms/step - loss: 0.4383 -
accuracy: 0.8735
79/79 [==============================] - 1s 3ms/step - loss: 1.0184 - accuracy:
0.6292
```

[19]: [1.0183591842651367, 0.6291999816894531]

[19]: