DL ASSIGNMENT 3

Q1) no, it is not okay to initialise with same weights.

Q2) it is okay to initialise bias with zero.

Q3) self normalisation and faster even with batch normalisation, cannot die, enable deep neural networks due to no vanishing gradient problem.

Q4) selu is good when data is explosive and require automatic normalisation with faster neural networks. Leaky relu is good when vanishing gradient problem arises. When data shows linearity, relu is better. For faster learning, tanh is better. Softmax is good to have probabilistic outcomes. Logistic is good for faster training.

Q5) algorithm will overshoot over minimum and may lead to overfitting due to very fast learning.

Q6) one hot encoding, pruning etc can help in creating sparse models.

Q7) yes, dropout slows training. Dropout will also slow down prediction. Mc dropout is much faster.

Q8) import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

cifar10 = tf.keras.datasets.cifar10

(x\_train\_full, y\_train\_full),(x\_test,y\_test) = cifar10.load\_data()

x\_valid, x\_train = x\_train\_full[:25000]/255.0, x\_train\_full[25000:]/255.0

y\_valid, y\_train = y\_train\_full[:25000]/255.0, y\_train\_full[25000:]/255.0

initializer = tf.keras.initializers.HeUniform()

LAYERS = [tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer, input\_shape = [32, 32, 3]),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer),

tf.keras.layers.Dense(100, activation = "elu", kernel\_initializer = initializer)]

model\_cifar = tf.keras.models.Sequential(LAYERS)

LOSS\_FUNCTION = "sparse\_categorical\_crossentropy"

OPTIMIZER = "Nadam"

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2)

model\_cifar.add(layers.Dense(10, activation="softmax"))

model\_cifar.compile(optimizer = OPTIMIZER, loss = LOSS\_FUNCTION)

EPOCHS = 15

VALIDATION\_SET = (x\_valid, y\_valid)

history = model\_cifar.fit(x\_train, y\_train, epochs = EPOCHS, validation\_data = VALIDATION\_SET, callbacks = [callback])