What blend gave you the best accuracy? What blend gave you the worst accuracy? What are some observations you are starting to see about the layers, their complexity, the activation option used, etc? Offer at least 4 observations

**Answer:**

For 2-label dataset:

# 93.1% sgd + meansquareerror  
# 49.3% sgd + categorical\_crossentropy  
# 50.1% adam + categorical\_crossentropy  
# 94.4% adam + meansquareerror   
  
# 3 units in first layer 2 units in the second layer output for sigmoid 94.4  
# 3 units in first layer then output for sigmoid 93.3%  
# 3 units in first layer 3 units in the second layer output for sigmoid 93.4  
# 2 units in first layer 3 units in the second layer output for sigmoid 93.1  
# 2 units in first layer 2 units in the second layer output for sigmoid 93.3

Best: When the loss function is meansquareerror and optimizer is adam, with 2 hidden layer, the first layer relu with 3 units and second layer relu with 2 units

Worst: When the loss function is sgd and optimizer is categorical\_crossentropy, with 2 hidden layer, the first layer relu with 2 units and second layer relu with 3 units

For 3-label dataset:

# 5 layer, relu, relu, sigmoid, sigmoid, softmax; 10, 5, 3, 3, 3 73.2%

# 5 layer, relu, relu, sigmoid, sigmoid, softmax; 18, 12, 8, 6, 3 74.5%

# 4 layer, relu, relu, sigmoid, softmax; 18, 12, 6, 3 73.5%

# 4 layer, relu, relu, sigmoid, softmax; 10, 5, 3, 3 73.3%

# 5 layer, sigmoid, sigmoid, relu, relu, softmax; 18, 12, 8, 6, 3 75.2%

# 59.4% sgd + meansquareerror

# 74.9% sgd + categorical\_crossentropy

# 75.7% adam + categorical\_crossentropy

# 75.2% adam + meansquareerror

Best: When the loss function is categorical\_crossentropy and optimizer is adam, with 5 hidden layer, sigmoid, sigmoid, relu, relu, softmax; 18 units, 12 units, 8 units, 6 units, 3 units

Worst: When the loss function is sgd and optimizer is meansquareerror, with 4 hidden layer, relu, relu, sigmoid, softmax; 10 units, 5 units, 3 units, 3 units

Observation:

1. More hidden layers provide better accuracy but spend more memory and time
2. More hidden units provide better accuracy but spend more memory and time
3. Combination of loss function and optimizer do affect accuracy and need to try different combination of them to find better accuracy
4. The order of layers or, in other words, the activation function do affect accuracy and need to try different activation function to find better accuracy

Table

Description automatically generated

Fig 1 Oranges vs. Grapefruit: diameter, weight, and color data

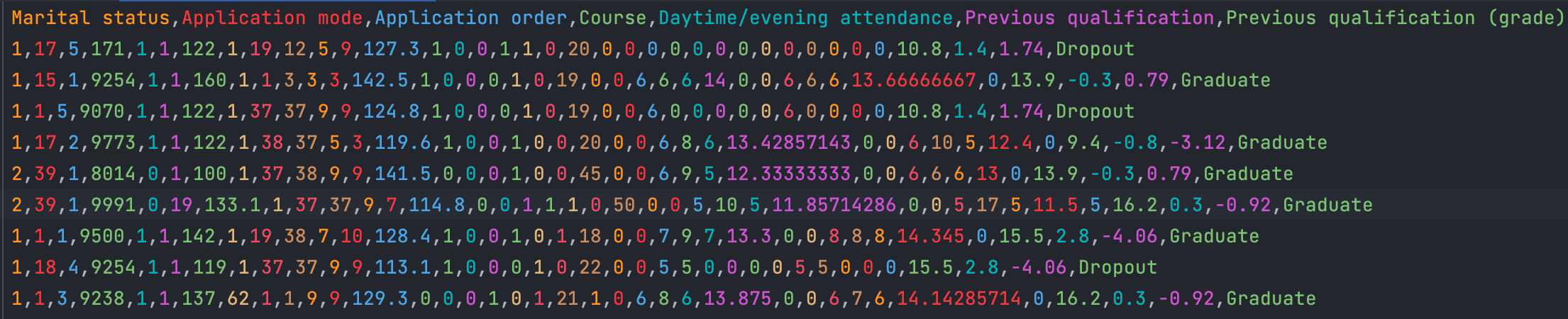


Fig 2 academic success w/ one-hot encoding, labels will be 0, 1, 2 (dropout, graduate, enrolled)

**CODE AND OUTPUT**

# MODEL 1 - for the two-label dataset

model1 = keras.Sequential([

keras.layers.Dense(3, input\_shape=(5,), activation="relu"), # first hidden layer

keras.layers.Dense(2, activation="relu"), # This is the second hidden layer with two units

keras.layers.Dense(1, activation="sigmoid")]) # we are using 0 or 1 here, so we have output size of 1.

model1.compile(

# optimizer='SGD',

optimizer="Adam",

loss=keras.losses.MeanSquaredError(),

# loss="categorical\_crossentropy",

metrics=["accuracy"])

model1.fit(x\_train, y\_train, epochs=50)

Validation\_Loss, Validation\_Accuracy = model1.evaluate(x\_test, y\_test)

Epoch 50/50

250/250 [==============================] - 0s 1ms/step - loss: 0.0471 - accuracy: 0.9362

63/63 [==============================] - 0s 1ms/step - loss: 0.0472 - accuracy: 0.9360

The validation loss and accuracy: 0.04715810716152191 0.9359999895095825

# MODEL 2 - for the three-label dataset

model2 = keras.Sequential([

keras.layers.Dense(18, input\_shape=(36,), activation="relu"), # Hidden layer 1

keras.layers.Dense(12, activation='sigmoid'), # Hidden layer 2

keras.layers.Dense(8, activation='sigmoid'), # Hidden layer 3

keras.layers.Dense(6, activation="relu"), # Hidden layer 4

keras.layers.Dense(3, activation='softmax') # output layer

])

model2.compile(

# optimizer='SGD',

optimizer="Adam",

# loss=keras.losses.MeanSquaredError(),

loss="categorical\_crossentropy",

metrics=["accuracy"])

model2.fit(x\_train, y\_train, epochs=100)

Validation\_Loss, Validation\_Accuracy = model2.evaluate(x\_test, y\_test)

Epoch 100/100

111/111 [==============================] - 0s 3ms/step - loss: 0.4218 - accuracy: 0.8389

28/28 [==============================] - 1s 7ms/step - loss: 0.7446 - accuracy: 0.7525

The validation loss and accuracy: 0.7446030974388123 0.7525423765182495