UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SECB 4313 BIOINFORMATICS MODELING & SIMULATION

ASSIGNMENT 2

LECTURER: DR. AZURAH BINTI A SAMAH

SECTION 1

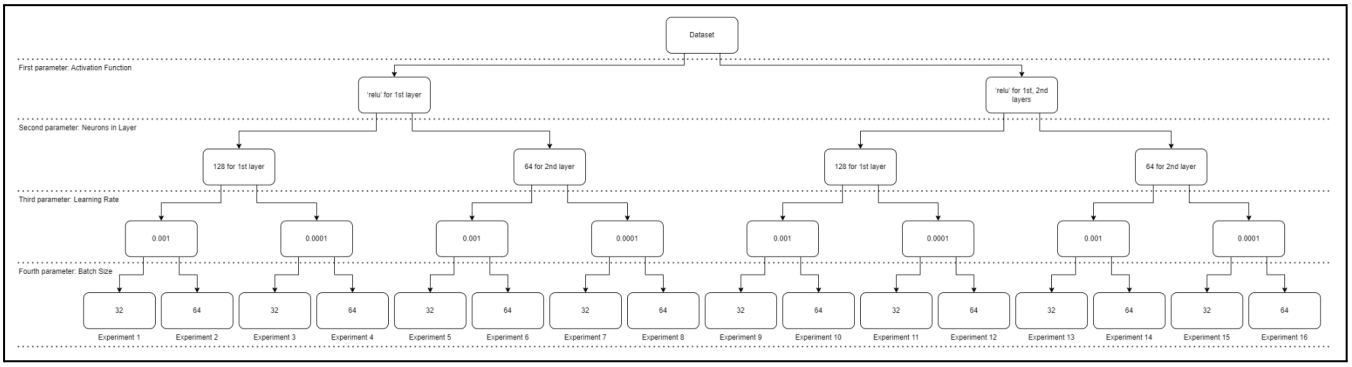
TEAM MEMBERS:

NO.	NAME	MATRIC NUMBER
1.	FELICIA CHIN HUI FEN	A20EC0037
2.	GOH YITIAN	A20EC0038
3.	PHANG CHENG YI	A20EC0131
4.	MOHD FIRDAUS BIN ZAMRI	A20EC0080

1. **Identify 4 hyperparameters and propose two values for each hyperparameter. Justify the selection of the hyperparameters and its corresponding values.**

No.	Hyperparameters	Proposed Value 1	Proposed Value 2	Justification
1.	Choice of activation function	relu' for the first layer	'relu' for the first, two layer	'relu' (Rectified Linear Unit) is more commonly used in hidden layers for its efficiency and capability to mitigate the vanishing gradient problem.
2.	Number of neurons in a layer	128 neurons and for the first dense layer.	64 neurons for the second dense layer.	Increasing the number of neurons allows the network to represent more complex functions.
3.	Learning rate	0.001	0.0001	Lowering it to 0.001 can provide more stable and precise updates, though it may require more epochs to converge.
4.	Batch Size	32	64	Increasing the batch size may help in achieving smoother and faster convergence due to better gradient estimates, although it requires more memory.

2. **Construct a tree diagram that displays the proposed hyperparameters and its corresponding values.**



3. **Tabulate the proposed experimental design based on your tree diagram.**

Experiment No.	**Activation Function**	**Neurons in Layer**	**Learning Rate**	**Batch Size**
1	'relu' for 1st layer	128 for 1st layer	0.001	32
2	'relu' for 1st layer	128 for 1st layer	0.001	64
3	'relu' for 1st layer	128 for 1st layer	0.0001	32
4	'relu' for 1st layer	128 for 1st layer	0.0001	64
5	'relu' for 1st layer	64 for 2nd layer	0.001	32
6	'relu' for 1st layer	64 for 2nd layer	0.001	64
7	'relu' for 1st layer	64 for 2nd layer	0.0001	32
8	'relu' for 1st layer	64 for 2nd layer	0.0001	64
9	'relu' for 1st, 2nd layers	128 for 1st layer	0.001	32
10	'relu' for 1st, 2nd layers	128 for 1st layer	0.001	64

11	‘relu’ for 1st, 2nd layers	128 for 1st layer	0.0001	32
12	‘relu’ for 1st, 2nd layers	128 for 1st layer	0.0001	64
13	‘relu’ for 1st, 2nd layers	64 for 2nd layer	0.001	32
14	‘relu’ for 1st, 2nd layers	64 for 2nd layer	0.001	64
15	‘relu’ for 1st, 2nd layers	64 for 2nd layer	0.0001	32
16	‘relu’ for 1st, 2nd layers	64 for 2nd layer	0.0001	64

4. **Perform *hyperparameter tuning* to improve current result. (Simulate the model and collect the results).**

Original Model:

```
model = Sequential() #Allow us to create model layer by layer
model.add(Dense(64, input_dim=21, activation='softmax')) #Softmax turn number data into probabilities which sum to 1
model.add(Dense(32, activation='softmax'))
model.add(Dense(1, activation='sigmoid')) # produce probability value (number between 0 or 1)
model.summary()
```

```
model.compile(loss='mse',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999,epsilon=1e-07, amsgrad=False,name='Adam'),
              metrics=['acc'])
```

```
output = []
early = EarlyStopping(monitor='val_acc', patience=400, mode='auto')
checkpoint = ModelCheckpoint(model_loc+"heart_disease_best_model.hdf5", monitor='val_acc', verbose=0, save_best_only=True, mode='auto', save_freq='epoch')
reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.01, patience=100, verbose=1, mode='auto', min_lr=0.001)
callbacks_list = [early]

output = model.fit(x_train, y_train,validation_data=(x_val,y_val), epochs=1000, batch_size=16, verbose=1, callbacks=callbacks_list)
```

Figure above shows the original model. The hyperparameter tuning is conducted based on the tabulate proposed experimental design in number 3. The parameter of the model is changed based on the table in 3.

The experimental results are then evaluated using performance metrics such as accuracy, precision, recall, and F1-Score.

5. **Tabulate the results based on your proposed experimental design.**

No.	Model Loss	Accuracy	Precision	Recall	F1-Score
1.	0.15	0.84	0.84	0.84	0.84
2.	0.15	0.84	0.84	0.84	0.84
3.	0.24	0.55	0.28	0.50	0.36
4.	0.25	0.55	0.28	0.50	0.36
5.	0.15	0.84	0.84	0.84	0.84
6.	0.14	0.84	0.84	0.84	0.84
7.	0.25	0.55	0.28	0.50	0.36
8.	0.25	0.55	0.28	0.50	0.36
9.	0.18	0.77	0.79	0.79	0.77
10.	0.16	0.77	0.78	0.77	0.77
11.	0.17	0.84	0.84	0.84	0.84
12.	0.18	0.81	0.81	0.81	0.81
13.	0.17	0.71	0.71	0.71	0.71
14.	0.17	0.77	0.77	0.78	0.78
15.	0.17	0.81	0.81	0.81	0.81
16.	0.18	0.81	0.81	0.81	0.81

6. **Analyse the results. Which combination of hyperparameters generate the most improved result?**

No.	Activation Function	Neurons in Layer	Learning Rate	Batch Size
1.	‘relu’ for 1st layer	128 for 1st layer	0.001	32
2.	‘relu’ for 1st layer	128 for 1st layer	0.001	64
3.	‘relu’ for 1st layer	128 for 1st layer	0.0001	32
4.	‘relu’ for 1st layer	128 for 1st layer	0.0001	64
5.	‘relu’ for 1st layer	64 for 2nd layer	0.001	32

6.	‘relu’ for 1st layer	64 for 2nd layer	0.001	64
7.	‘relu’ for 1st layer	64 for 2nd layer	0.0001	32
8.	‘relu’ for 1st layer	64 for 2nd layer	0.0001	64
9.	‘relu’ for 1st, 2nd layers	128 for 1st layer	0.001	32
10.	‘relu’ for 1st, 2nd layers	128 for 1st layer	0.001	64
11.	‘relu’ for 1st, 2nd layers	128 for 1st layer	0.0001	32
12.	‘relu’ for 1st, 2nd layers	128 for 1st layer	0.0001	64
13.	‘relu’ for 1st, 2nd layers	64 for 2nd layer	0.001	32
14.	‘relu’ for 1st, 2nd layers	64 for 2nd layer	0.001	64
15.	‘relu’ for 1st, 2nd layers	64 for 2nd layer	0.0001	32
16.	‘relu’ for 1st, 2nd layers	64 for 2nd layer	0.0001	64

The original model’s hyperparameters are shown in 4. The accuracy, precision, recall and F1-score for the original model are 0.74, 0.75, 0.75 and 0.74 respectively.

The hyperparameters are then tuned according to the proposed values in 1 to observe the performance of the model whether it is improved or reduced. There are 16 different hyperparameter combinations for the experiment as shown in the table above.

From the results in 5, it can be seen the hyperparameters used in the sixth experiment provide better performance as compared to the original model. The result shows an improvement in accuracy from 0.74 in the original model to 0.84 in Experiment 6. Experiment 6’s model loss of 0.14 is lower as compared to the original model’s model loss of 0.20. Both precision and recall have the same improvement which is from 0.75 to 0.84. The average F1-score also increased from 0.74 to 0.84. Furthermore, among the 16 combinations of hyperparameters, Experiment 6 has the

lowest model loss although it shares the same accuracy, precision, recall and F1-score as in Experiment 1, 2, 5 and 11.

The improvement might be contributed by the changes in these hyperparameters. The first is the activation function. The activation function employed in the original model is Softmax for both layers while the activation function employed in Experiment 1 is Rectified Linear Unit (ReLU) for the first layer. In deep learning models, ReLU activation function is often more effective than the softmax for hidden layers. ReLU introduces non-linearity into the model, which assists in addressing the vanishing gradient problem and enables the model to learn more complicated patterns. On the other hand, since softmax normalises the outputs into probabilities, it is usually employed at the output layer for classification problems.

The neurons in the second layer had increased from 32 to 64, which increased the capacity of the network to learn from data. With only 32 neurons in the second layer, the model might lack the capacity to fully capture the underlying patterns in the data, which may lead to the underfitting of the model. However, with 64 neurons in the second layer, the model has more parameters to capture and learn intricate features in the data. This is why the model loss of the sixth combination is lower than others.

The learning rate decreased from 0.01 to 0.001, which can lead to a more stable convergence process and allows the model to fine-tune the weights more precisely, avoiding the risk of overshooting the minimum of the loss function.

The batch size increased from 16 to 64. A larger batch size provides a better estimate of the gradient by averaging over more examples. This reduces the variance in the gradient estimates, leading to more stable and reliable updates to the model weights.

In conclusion, the results of experiment 6 have the best improvement as it benefits from a combination of improved activation function (ReLU) for learning complex patterns, increased neurons in the second layer from 32 to 64 for greater representational capacity, lower learning rate from 0.01 to 0.001 for finer adjustments to the model and use a larger batch size from 16 to 64 for more stable gradient estimation.