Reference Answers for Assignment 1

Note: This serves as a reference answer for **Assignment 1**. While the results may exhibit variability due to the random data selection in each batch—essentially, models are trained using different forms of stochastic gradient descent—the overarching conclusion should remain consistent across various model configurations.

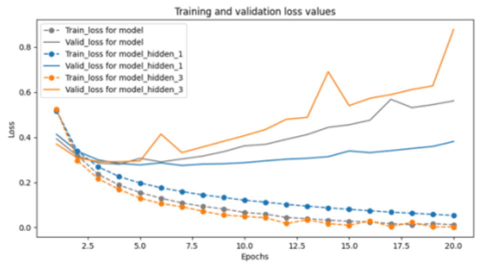
As denoted in the .ipynb file, we train models with epoch=6 to generate accuracy tables.

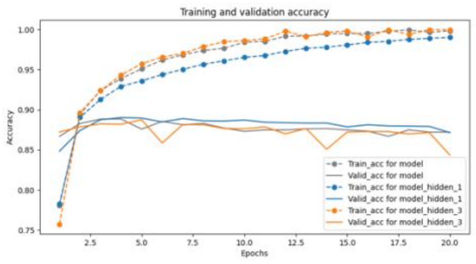
1. You used two hidden layers. Try using one or three hidden layers and see how doing so affects validation and test accuracy.

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model_hidden_1 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_hidden_3 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

Comparison plots on initial run with 20 epochs. Here, the model with gray color is the baseline model.





After changing to 6 epochs and performing 5 runs, we obtain the following average accuracy in the test set:

Model	Validation Accuracy	Test Accuracy
Baseline Model (2 hidden layers)	0.88898	0.8762
Model with 1 hidden layer	0.88912	0.8788
Model with 3 hidden layers	0.88994	0.8739

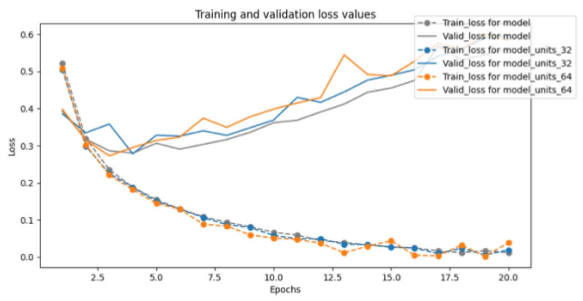
Ans: From the plots, we observed that the loss values and accuracy performance of the smaller model (with **1 hidden layer**) on the validation set **are steadier and less volatile than** the **baseline model**. The smaller model also degrades slowly when it starts overfitting. On the other hand, the bigger model (with **3 hidden layers**) overfits much **more severely** than **baseline model**, resulting in a large difference between the training and validation loss. It exhibits a low training loss but a **much higher and noisier validation loss,** showing relatively low values but **greater volatility and worser performance** in **validation accuracy.**

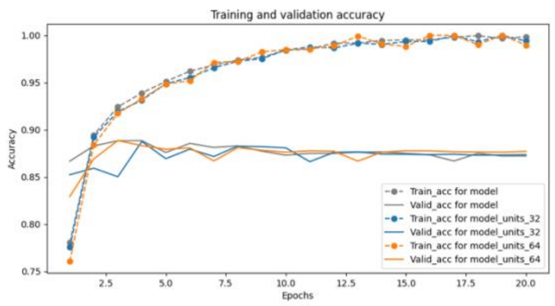
Additionally, the table above also shows that the model with **1 hidden layer** outperforms the models with 2 and 3 hidden layers for **test accuracy,** while model with 3 hidden layers has the highest validation accuracy.

2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.

```
model_units_32 = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_units_64 = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

Comparison plots on initial run with 20 epochs. Here, the model with gray color is the baseline model.





After changing to 6 epochs and performing 5 runs, we obtain the following average accuracy in the test set:

Model	Validation Accuracy	Test Accuracy
Baseline Model (with 16 units)	0.88898	0.8762
Model with 32 units	0.88914	0.86758
Model with 64 units	0.88766	0.86715
Model with 4 units	0.88106	0.8723
Model with 512 units	0.88576	0.8748

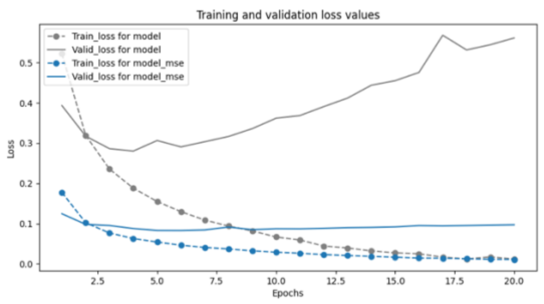
Ans: From the plots, we found that the validation loss and accuracy for **the model with 64 units** are **more volatile and noisier** than **baseline model** (with 16 units). The **relatively large model** (with 32 units) starts overfitting at the same time as the baseline model but is **slightly more diverse** than the latter. Their performance on validation set is very similar.

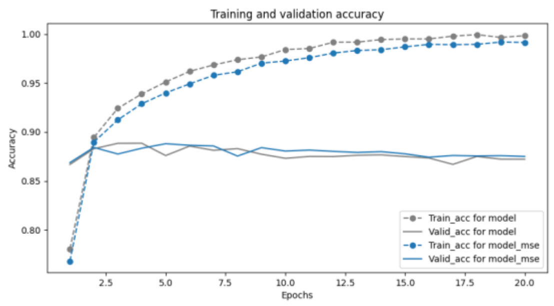
Additionally, the table above shows that the accuracy of the **baseline model with 16 units outperforms** the models with **32 and 64 hidden layers** and is better for more extreme models with 4 and 512 units in the test set. However, in the validation is very similar.

3. Try using the mse loss function instead of binary_crossentropy.

```
model_mse.compile(optimizer="rmsprop",
                  loss="mse",
                  metrics=["accuracy"])
```

Comparison plots on initial run with 20 epochs. Here, the model with gray color is the baseline model.





After changing to 6 epochs and performing 5 runs, we obtain the following average accuracy in the test set:

Model	Validation Accuracy	Test Accuracy
Baseline Model (with binary cross-entropy)	0.88898	0.8762
Model with mse	0.8876	0.875576

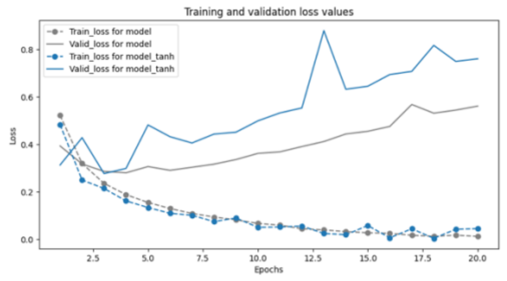
Ans: From the plots, we found when **mse** is used as the loss function, it performs much **worse than binary cross-entropy**. The loss values are worser than the baseline loss for both training and validation, while on the validation set, they are close to each other.

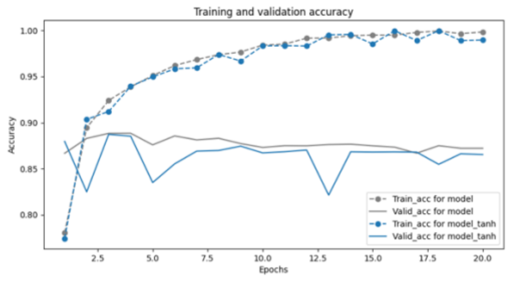
Additionally, the table above shows that the accuracy of the baseline model with **binary cross-entropy** outperforms the models with MSE for both validation and testing set.

4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.

```
model_tanh = keras.Sequential([
    layers.Dense(64, activation="tanh"),
    layers.Dense(64, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])
```

Comparison plots on initial run with 20 epochs. Here, the model with gray color is the baseline model.





After changing to 6 epochs and performing 5 runs, we obtain the following average accuracy in the test set:

Model	Validation Accuracy	Test Accuracy
Baseline Model (with ReLU)	0.88898	0.8762
Model with tanh	0.88684	0.8649

Ans: From the plots, we found the **tanh activation function** exhibits **worse performance** on validation loss and accuracy compared to **ReLU** in the reference model. Additionally, it is **more volatile and diverse** on the validation set.

Meanwhile, the table above shows that the accuracy of the baseline model with **ReLU** outperforms the models with tanh for both validation and test set.

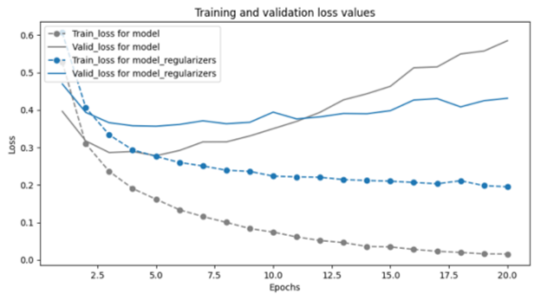
5. Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.

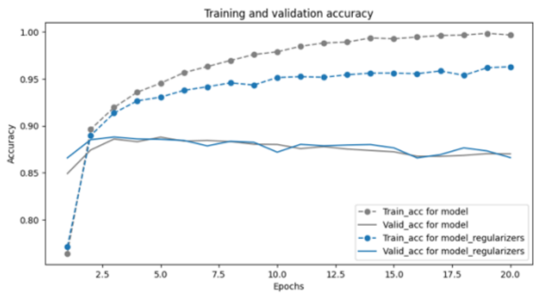
   a. **Weight Regularization** puts constraints on the complexity of a model by forcing its weights to take only small values. This regularization technique makes the distribution of weight values more regular and helps mitigate overfitting. It achieves this by adding a cost to the loss function associated with having large weights. There are two flavors of weight regularization:

      - L1 regularization: The added cost is proportional to the absolute value of the weight coefficients (the L1 norm of the weights).
      - L2 regularization: The added cost is proportional to the square of the value of the weight coefficients (the L2 norm of the weights). In the context of neural networks, L2 regularization is also referred to as weight decay.

```
model_regularizers = keras.Sequential([
    layers.Dense(16, kernel_regularizer=regularizers.l2(0.002),
    activation="relu"),
    # or regularizers.l1(0.001) or regularizers.l1_l2(l1=0.001, l2=0.001)
    layers.Dense(16, kernel_regularizer=regularizers.l2(0.002),
    activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

Comparison plots on initial run with 20 epochs. Here, the model with gray color is the baseline model.





After changing to 6 epochs and performing 5 runs, we obtain the following average accuracy in the test set:

Model	Validation Accuracy	Test Accuracy
Baseline Model	0.88898	0.8762
Model with L2	0.88646	0.866424

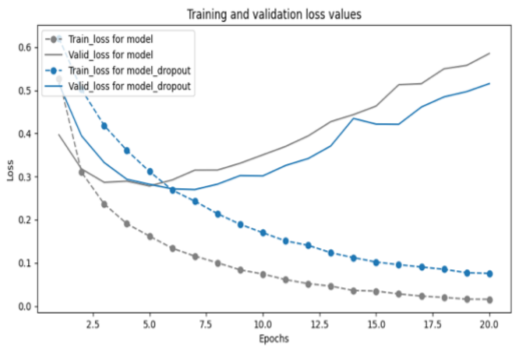
Ans: Above figures show the impact of the L2 regularization penalty. It seems the model with **L2 regularization** has become more resistant to overfitting than the baseline model--- it has smaller difference in training and validation loss values, even though its performance of accuracy on validation set is similar to baseline model.

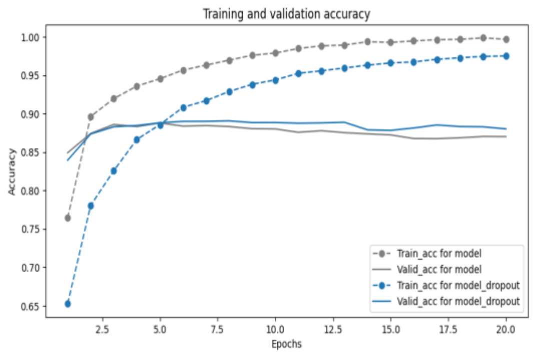
Additionally, the table above shows that the accuracy of the **baseline model** with outperforms the models with L2 for both validation and test set.

b. Dropout, applied to a layer, consists of randomly dropping out (setting to zero) a number of output features of the layer during training.

```
model_dropout = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
```

Comparison plots on initial run with 20 epochs. Here, the model with gray color is the baseline model.





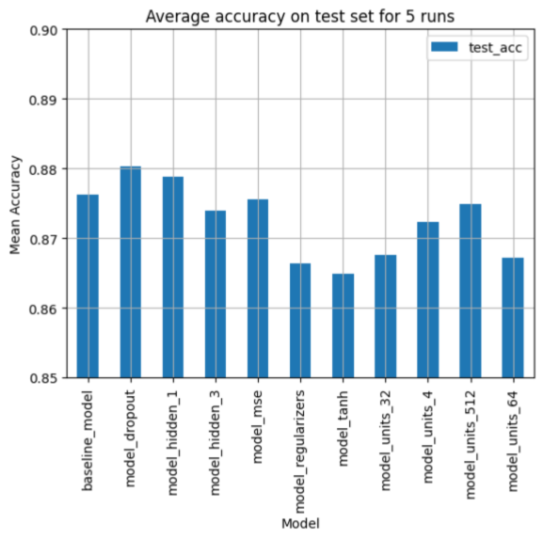
After changing to 6 epochs and performing 5 runs, we obtain the following average accuracy in the test set:

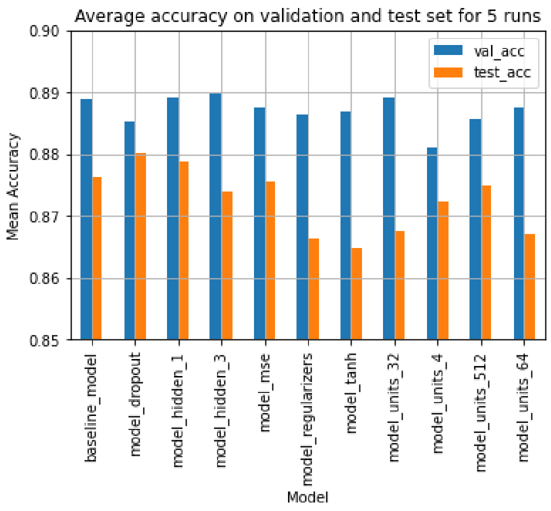
Model	Validation Accuracy	Test Accuracy
Reference Model	0.88898	0.8762
Model with dropout	0.88532	0.8802

Ans: Above figures shows the model with dropout generates **a clear improvement over the baseline model**—it also seems to be working much better than L2 regularization, since the accuracy on validation set is much better than L2 and baseline model.

Additionally, the table above shows that the accuracy of the model with **dropout** is higher than the baseline model for test set but lower in validation set.

Summary:





Several factors impact a neural network model's performance. Experimentation with various hidden units, activation function and loss function revealed a decrease in accuracy, suggesting their necessity only under specific conditions. It's conceivable that when using additional techniques such as optimizers, regularizers and dropouts may enhance model performance. Additionally, while not experiment in above examples, the dataset size also plays a crucial role, the larger datasets tend to yield better accuracy.

Ultimately, for the present dataset, the optimal combination of hyperparameters includes one hidden layer with 16 hidden units, binary cross-entropy loss function, ReLU activation function, RMSprop optimizer and dropout layer. (See final model performance in the .ipynb file)