Assignment_1

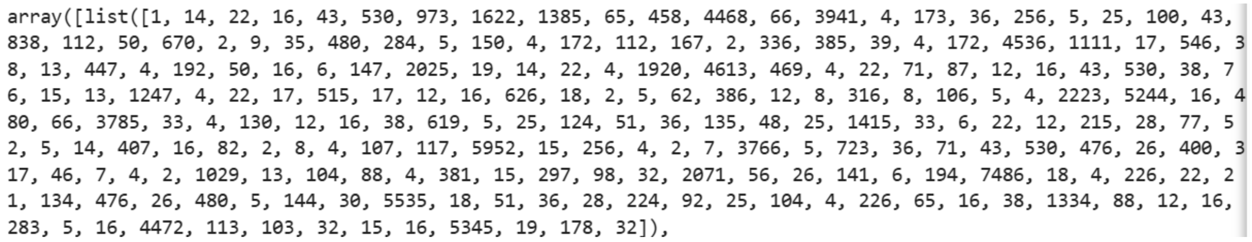
Introduction:

A **neural network** is a computer program that learns from examples, kind of like how our brain learns from experience. It helps with the computer spot patterns and makes guesses or decisions. A **neural network** is a system of **connected nodes**, called **neurons**, that process information. These neurons are organized into **layers**:

- **Input layer** – receives the data.
- **Hidden layers** – do the processing.
- **Output layer** – produces the result.

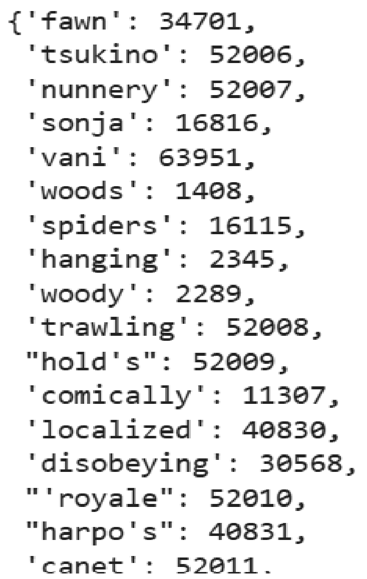
Imagine you're working on a project where you want a computer to understand whether a movie review is positive or negative. To do this, you decide to teach the computer by showing it thousands of movie reviews along with their ratings—either good or bad.

First, you grab a famous dataset called the IMDB reviews. But these reviews aren't written in words anymore; instead, each word has been swapped out with a number as below. This makes it easier for the computer to process. You choose to focus only on the 10,000 most common words, so the computer isn't overwhelmed by rare or complicated words.



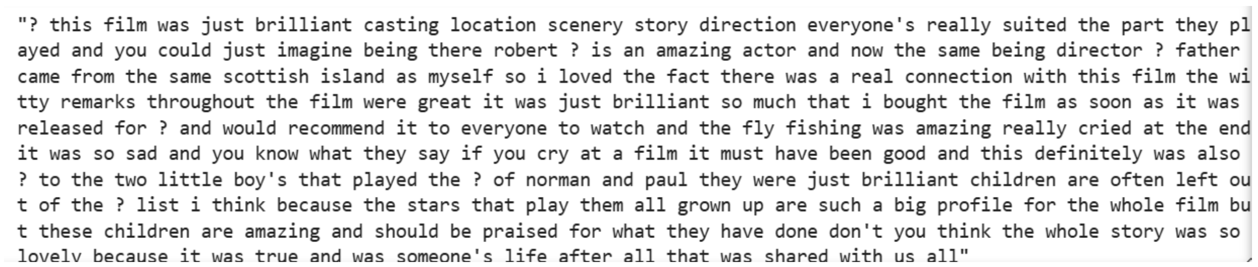
```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43,
838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 3
8, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 7
6, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 4
80, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 5
2, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 3
17, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 2
1, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16,
283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]),
```

Next, you want to peek behind the scenes to see what words those numbers represent. Luckily, there's a special dictionary that maps each word to a number, and you flip this dictionary around so you can translate numbers back into words. This way, you can take a mysterious sequence of numbers and turn it into a readable movie review again, just like reading a story.



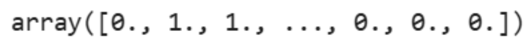
```
{'fawn': 34701,
 'tsukino': 52006,
 'nunnery': 52007,
 'sonja': 16816,
 'vani': 63951,
 'woods': 1408,
 'spiders': 16115,
 'hanging': 2345,
 'woody': 2289,
 'trawling': 52008,
 "hold's": 52009,
 'comically': 11307,
 'localized': 40830,
 'disobeying': 30568,
 "'royale": 52010,
 "harpo's": 40831,
 'canet': 52011,
```

Now, the computer still can't work well with lists of different lengths because reviews vary in how many words they have. So, you come up with a clever trick: you create a big sheet with 10,000 columns, one for each word. For each review, you mark a column with a 1 if that word appears and leave it 0 if it doesn't. This turns every review into a neat, fixed-size checklist of words, making it easier for the computer to analyze. Below is the sample review.



```
"? this film was just brilliant casting location scenery story direction everyone's really suited the part they pl
ayed and you could just imagine being there robert ? is an amazing actor and now the same being director ? father
came from the same scottish island as myself so i loved the fact there was a real connection with this film the wi
tty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was
released for ? and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end
it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also
? to the two little boy's that played the ? of norman and paul they were just brilliant children are often left ou
t of the ? list i think because the stars that play them all grown up are such a big profile for the whole film bu
t these children are amazing and should be praised for what they have done don't you think the whole story was so
lovely because it was true and was someone's life after all that was shared with us all"
```

Finally, you prepare the labels—the ratings—by making sure they're in a format the computer understands. With your data transformed into neat vectors and clear labels, the computer is now ready to learn. It can start recognizing patterns: which words often show up in positive reviews, and which in negative ones.



```
array([0., 1., 1., ..., 0., 0., 0.])
```

And just like that, you've set the stage for a machine to understand the language of movie reviews and decide whether they're praising or criticizing a film.

Method:

First, it takes list of movie reviews (turned into numbers) and splits it into two parts: one part to actually teach the computer, and a smaller part to check how well it's learning while it practices.

Then, it builds a simple "brain" for the computer with just one hidden layer — think of it as a small team inside the computer that looks for patterns in the reviews. This team has 16 little workers (called neurons) who use a special way to think called "tanh" (which helps them understand more complicated stuff).

After that, the computer uses another tiny worker at the end that decides if the review is positive or negative by giving a number between 0 and 1.

Next, the computer gets instructions on how to learn—how to improve its guesses step by step—using a method called "rmsprop" and checking how far off its guesses are with something called "mean squared error." Don't worry about the fancy names; just know it's a way to help the computer learn better.

The computer then practices by looking at chunks of reviews for 20 rounds. After each round, it tests itself on the smaller set it kept aside earlier, to see if it's actually getting better.

Finally, after practicing, the computer checks how well it does on two sets: the practice-check set and a completely new set of reviews it's never seen before. It tells you how often it gets the right answer and how far off it was when it made mistakes.

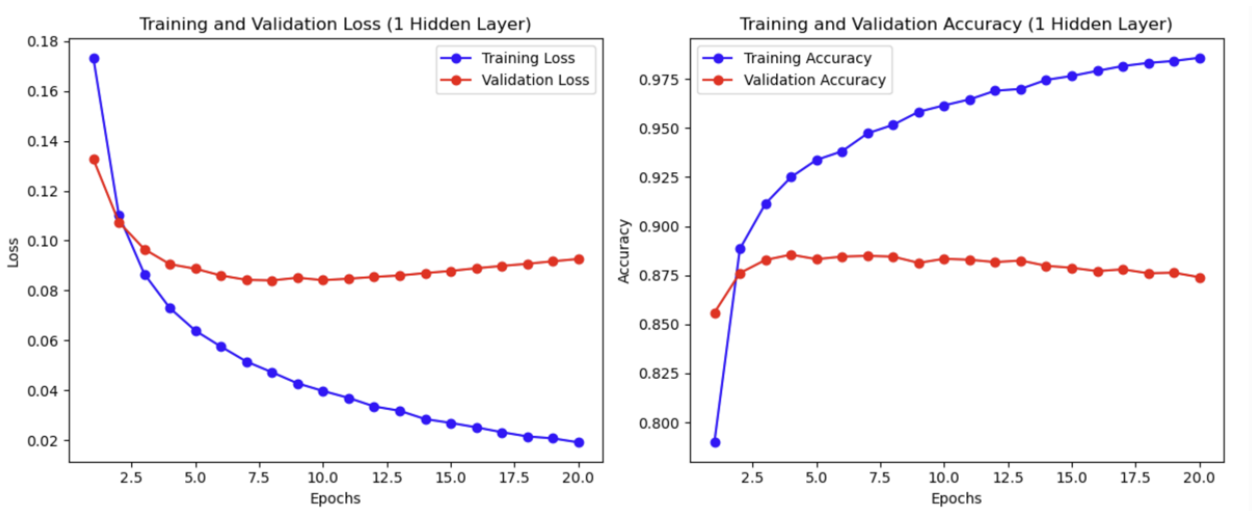
Results:

1)Below are the metrics when trained with 1 hidden layer with 16 units followed by 3 layers 16 each and 32 units as well and their respective plots.

Model	Test Loss	Test Accuracy	Val Loss	Val Accuracy
1-layer, 16 units (MSE, tanh)	0.0994	86.76%	0.0927	87.40%
3-layer, 16 units (MSE, tanh)	0.1239	85.94%	0.1127	87.24%
1-layer, 32 units (MSE, tanh)	0.1030	86.56%	0.0954	87.40%
3-layer, 32 units (MSE, tanh)	0.1237	85.87%	0.1128	87.14%

The above table compares models with 1 and 3 hidden layers using either 16 or 32 units. We observe that **1-layer models consistently perform better** than 3-layer models in terms of test accuracy and loss. This indicates that increasing depth with the same number of units can lead to **slightly worse performance**, possibly due to overfitting or unnecessary complexity for this task. Interestingly, increasing the number of units from 16 to 32 in the single-layer models provides **no significant improvement**, showing that **a simple architecture is sufficient** for good performance.

a)1 Hidden layer-16 units

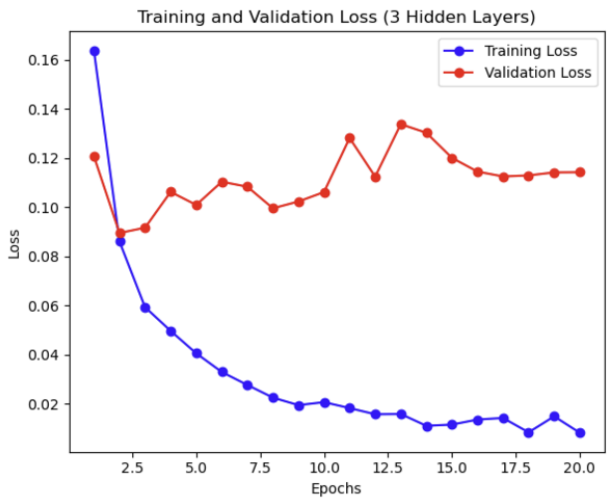


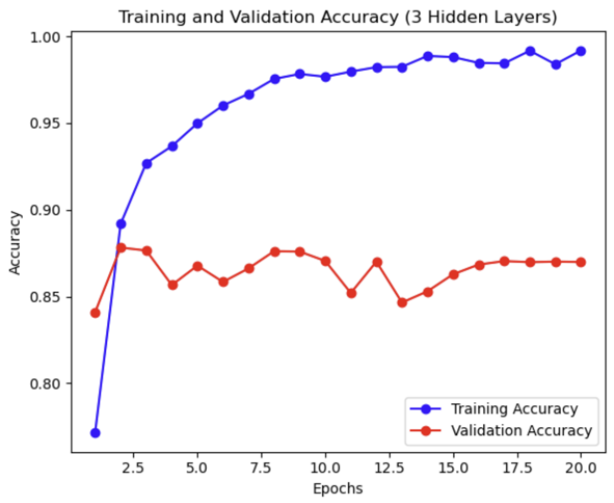
- The model with **1 hidden layer** is learning effectively, as shown by the decreasing training loss and increasing training accuracy.
- However, the **validation metrics plateau**, suggesting that the model may not benefit from further training beyond a certain point.
- This could be a sign of **limited model capacity** or **early overfitting**, and might be improved with techniques like regularization, dropout, or early stopping.
- While both models perform comparably, the regularized model provides better generalization and long-term reliability, making it the preferred choice in production or real-world settings where unseen data can vary significantly as below.

Model Variant	Dataset	Loss	Accuracy
Without Regularization	Test	0.0994	86.76%
	Validation	0.0927	87.40%

Model Variant	Dataset	Loss	Accuracy
With L2 + Dropout	Test	0.1180	86.88%
	Validation	0.1129	87.36%

b)3 Hidden layers-16 units each



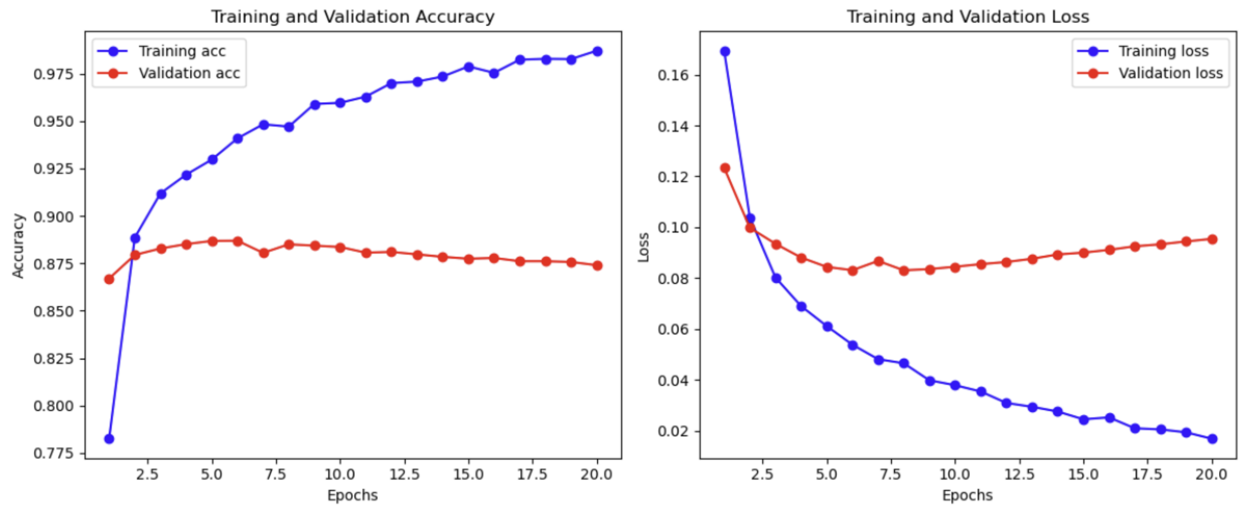


- The model with **3 hidden layers** is highly expressive and learns the training data very well.
- However, the **validation performance plateaus**, indicating that the model may be **too complex** for the dataset or lacks regularization.
- Techniques like **dropout**, **early stopping**, or **reducing model complexity** could help improve generalization

Model Variant	Dataset	Loss (MSE)	Accuracy
Without Regularization	Test	0.1239	85.94%
	Validation	0.1127	87.24%
With L2 Regularization + Dropout	Test	0.1329	86.73%
	Validation	0.1268	87.50%

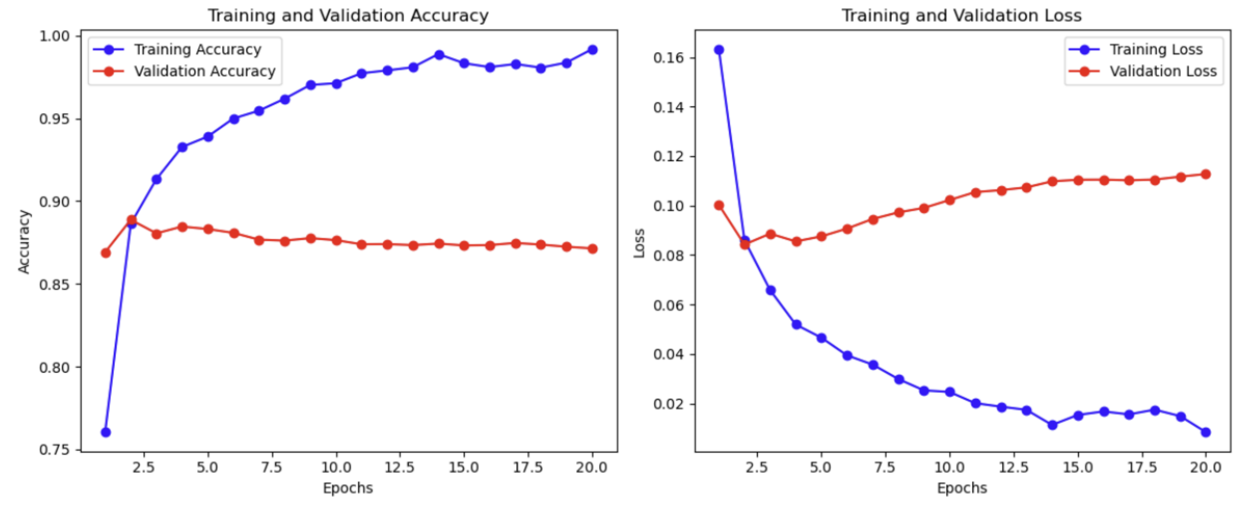
While the unregularized model achieves slightly lower training/validation loss, the model with **L2 regularization and Dropout** delivers **better validation and test accuracy**, indicating **stronger generalization** and **robustness** to overfitting. For real-world applications, the regularized model is the more reliable choice, especially as dataset complexity or noise increases.

c)1 Hidden layer-32 units



- The model achieves **high training accuracy** and **low training loss**, but **validation metrics plateau**, indicating **overfitting**.

d)3 Hidden layers-32 units each



- The model learns the training data extremely well, but its performance on validation data **plateaus** and even **worsens slightly**.
- This indicates **overfitting**, where the model memorizes training data but fails to generalize.

2)Below is the table which shows various metrics for 1 and 3 hidden layers with 64 units.

Model	Test Loss	Test Accuracy	Val Loss	Val Accuracy
1-layer, 64 units	0.1053	86.55%	0.0981	87.46%
3-layer, 64 units	0.1238	85.92%	0.1132	87.16%

Here, we explore the effect of increasing the number of units to **64**. While the **1-layer, 64-unit model performs well**, the **3-layer version does not show improvement** and performs slightly worse. This supports the idea that adding more layers may lead to **overfitting** without offering real benefits, even when the number of units is increased. The **best validation accuracy** is achieved by the 1-layer model, showing that **a shallow and wide network can outperform a deeper one**.

3)Below is the table which shows various metrics for 3 hidden layers with and without dropout and L2 regularization.

Model Description	Test Accuracy	Validation Accuracy
Baseline (3 layers, 64 units, tanh + MSE)	85.92%	87.16%
+ Dropout + L2 Regularization (0.3 dropout, L2 = 0.001)	86.74%	87.70%

The above table highlights the impact of applying **regularization techniques**—specifically, **dropout (0.3)** and **L2 regularization (0.001)**—to a deeper model. Compared to the baseline deep model without any regularization, the use of these techniques results in a **notable improvement in both test and validation accuracy**. This shows that adding regularization helps the model **generalize better** and **reduces overfitting**, especially in deeper networks.

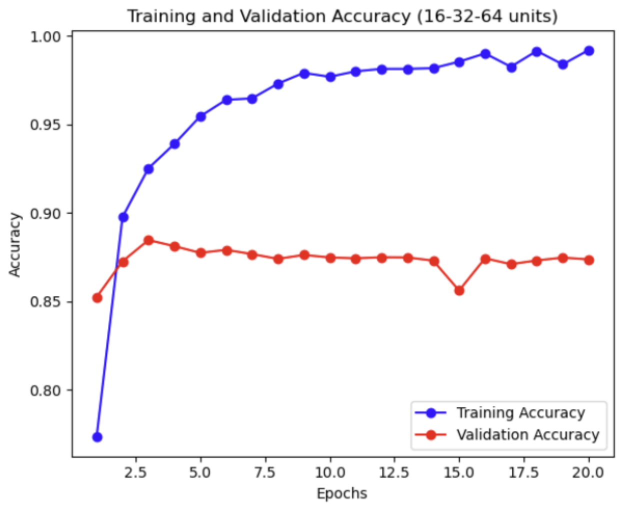
4)Below is the table with 3 layers (16-32-64 units) with and without dropout and regularization.

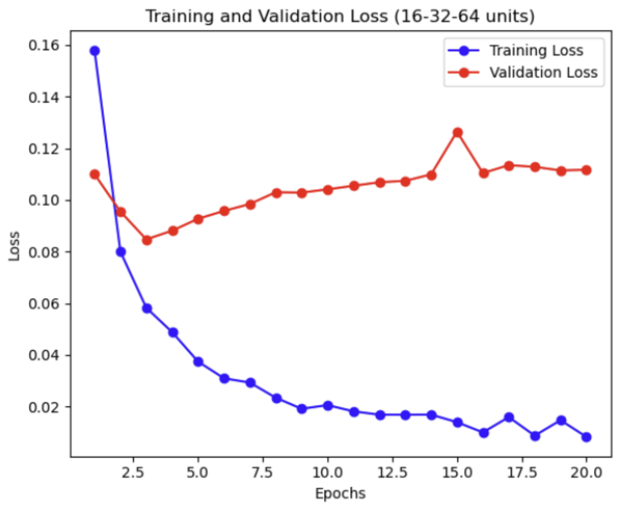
Model	Test Loss	Test Accuracy	Val Loss	Val Accuracy
3-layer, 16-32-64 units	0.1223	86.07%	0.1130	87.12%
3-layer, 16-32-64 units + Dropout + L2	0.1303	86.78%	0.1243	87.41%

In the above table, a model with **increasing units per layer (16 → 32 → 64)** is evaluated with and without regularization. The unregularized model performs decently, but when **dropout and L2** are added, the **test accuracy improves** while validation accuracy becomes **one of**

the highest overall. This suggests that combining a **structured architecture** with **regularization** leads to more reliable generalization, even if test loss increases slightly. It reflects a good trade-off between learning capacity and overfitting control.

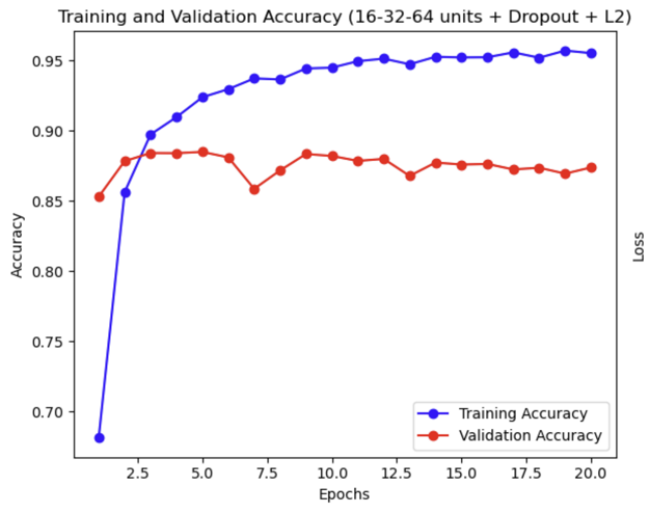
a)3 Layers(16-32-64 units)

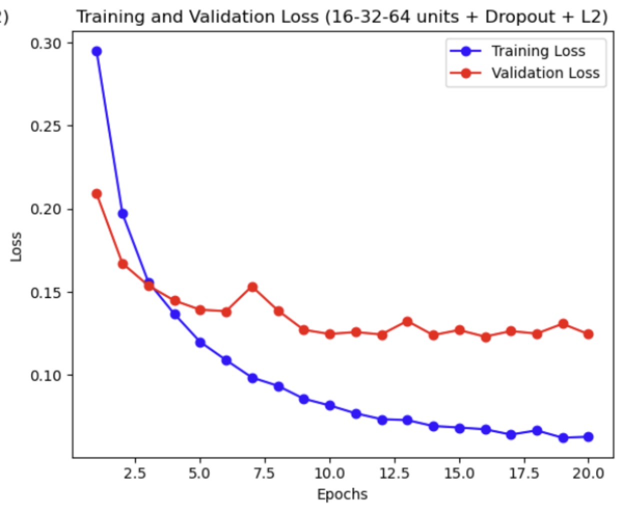




- The model performs well in terms of training accuracy and loss.
- Validation metrics are decent but show some fluctuation, indicating potential for improvement.
- Adding **Dropout or L2 regularization** could help stabilize validation performance and reduce overfitting.

b) 3 Layers(16-32-64 units+Dropout+L2)





- The use of **Dropout** and **L2 regularization** appears to be effective:
    - The model achieves **high training accuracy** without significant overfitting.
    - **Validation accuracy and loss are stable**, indicating good generalization.
- Compared to simpler or deeper models without regularization, this configuration strikes a **better balance** between learning and generalization.

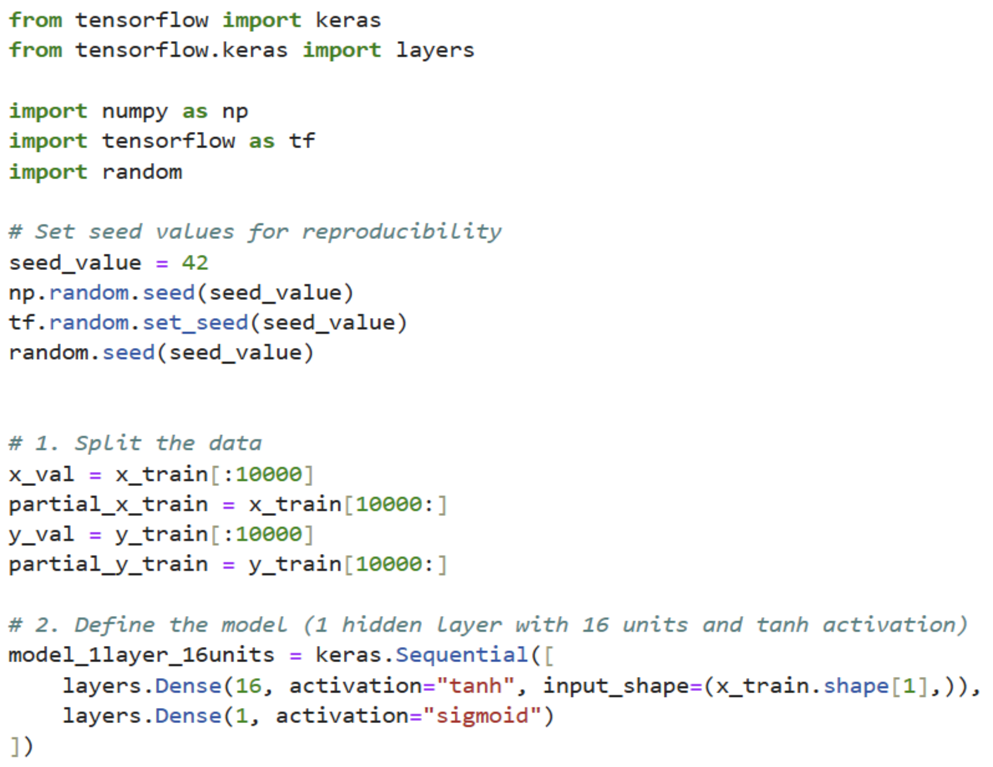
5)The below table shows the impact of increasing the number of hidden layers while keeping the number of units fixed at 16 per layer, using the **ReLU activation function** and **binary_crossentropy loss**, which are typically better suited for classification tasks like this one.

The model with **1 hidden layer performs best**, achieving the **highest test (86.56%) and validation (87.55%) accuracy**. As more layers are added (2 and then 3), both **loss increases** and **accuracy slightly decreases**. This suggests that the deeper models are **overcomplicating** the task, possibly leading to **overfitting** or **training instability**, despite using better-suited functions (ReLU and binary_crossentropy).

Overall, this shows that for this task of deciding, a **simpler model with just one layer** works **better** than models with more layers.

Model	Test Loss	Test Accuracy	Val Loss	Val Accuracy
1 hidden layer, 16 units	0.405	86.56%	0.379	87.55%
2 hidden layers, 16 units each	0.603	85.60%	0.554	86.99%
3 hidden layers, 16 units each	0.716	85.59%	0.651	86.97%

Appendix: Source Code



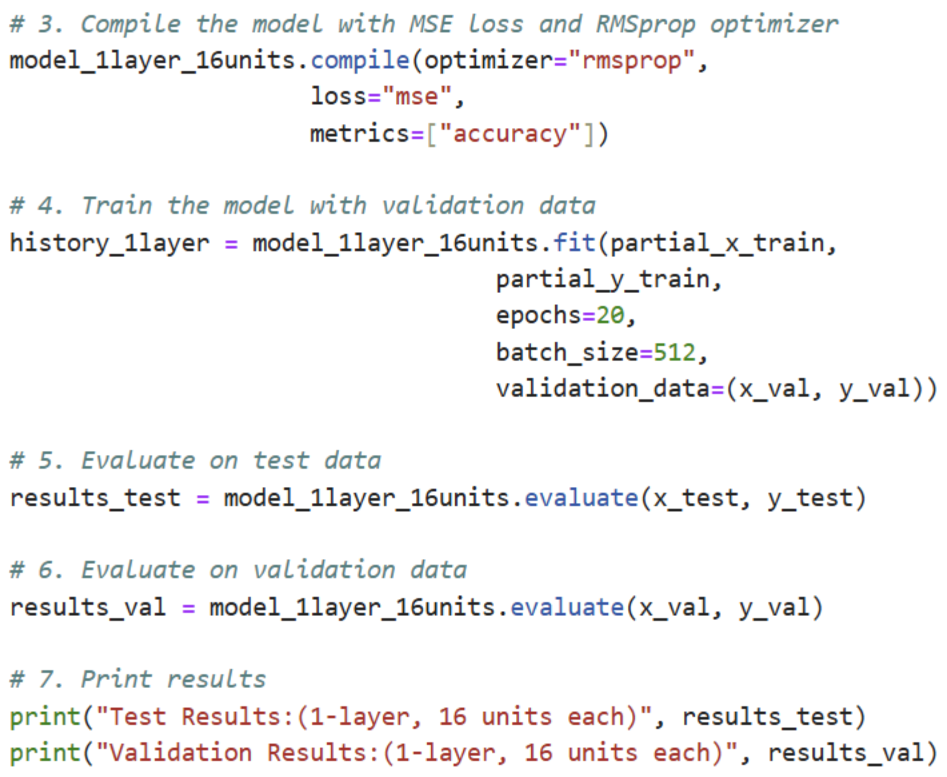
```
from tensorflow import keras
from tensorflow.keras import layers

import numpy as np
import tensorflow as tf
import random

# Set seed values for reproducibility
seed_value = 42
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
random.seed(seed_value)


# 1. Split the data
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]

# 2. Define the model (1 hidden layer with 16 units and tanh activation)
model_1layer_16units = keras.Sequential([
    layers.Dense(16, activation="tanh", input_shape=(x_train.shape[1],)),
    layers.Dense(1, activation="sigmoid")
])
```



```
# 3. Compile the model with MSE loss and RMSprop optimizer
model_1layer_16units.compile(optimizer="rmsprop",
                              loss="mse",
                              metrics=["accuracy"])

# 4. Train the model with validation data
history_1layer = model_1layer_16units.fit(partial_x_train,
                                           partial_y_train,
                                           epochs=20,
                                           batch_size=512,
                                           validation_data=(x_val, y_val))

# 5. Evaluate on test data
results_test = model_1layer_16units.evaluate(x_test, y_test)

# 6. Evaluate on validation data
results_val = model_1layer_16units.evaluate(x_val, y_val)

# 7. Print results
print("Test Results:(1-layer, 16 units each)", results_test)
print("Validation Results:(1-layer, 16 units each)", results_val)
```

```python
import matplotlib.pyplot as plt
def plot_training_history(history, title_suffix=""):
    history_dict = history.history
    epochs = range(1, len(history_dict["loss"]) + 1)

    # Plot Loss
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history_dict["loss"], "bo-", label="Training Loss")
    plt.plot(epochs, history_dict["val_loss"], "ro-", label="Validation Loss")
    plt.title(f"Training and Validation Loss {title_suffix}")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()

    # Plot Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, history_dict["accuracy"], "bo-", label="Training Accuracy")
    plt.plot(epochs, history_dict["val_accuracy"], "ro-", label="Validation Accuracy")
    plt.title(f"Training and Validation Accuracy {title_suffix}")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()

    plt.tight_layout()
    plt.show()
```