

# **Assignment 2**

## **Advanced Machine learning**

Instructor: Chaojiang (CJ) Wu, Ph.D.  
Department of MIS  
Kent state university.

Student: Lava Kumar Saripudi  
Reg. ID: 811196583

## Contents:

Problem Statement -----	03
Data Analysis-----	04
Conclusion -----	14
Reference-----	14
Appendix-----	15

**Problem Statement:**

As per the instructions of the question given in the assignment 2:

In this assignment, you will accomplish the following:

1. Modify an existing neural network model to improve performance
2. Explain how different approaches affect the performance of the model

For the IMDB example that we discussed in class, do the following:

1. You used two hidden layers. Try using one or three hidden layers and see how doing so. Affects validation and test accuracy.
2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.
3. Try using the mse loss function instead of binary\_crossentropy.
4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.
5. Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.

**Dataset:**

Here we are using the IMDB dataset for executing different types of operations as mentioned above.

```
from tensorflow.keras.datasets import imdb
```

```
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(  
    num_words=10000)
```

The above code snippet loads the IMDB dataset using the Keras API in TensorFlow.

The Internet Movie Database has 50,000 highly polarized reviews that make up the IMDB dataset. There are 25,000 reviews for training and 25,000 reviews for testing, with 50 percent negative and 50 percent positive reviews in each set.

The dataset has already been preprocessed, with each review encoded as a list of integers, each representing a word. The maximum number of words that can be included in the dataset based on their frequency of occurrence is specified by the "num\_words" parameter in the load\_data function. We have set it to 10,000 here, which means that only the top 10,000 words will be kept, and the rest will be deleted.

The encoded reviews can be found in train\_data, test\_data, and their corresponding sentiment labels, where 0 indicates a negative review and 1 indicates a positive one.

The current model has three parts as well as other terminology explained below:

1)**Input layer:** It contains units representing the input fields.

2)**Hidden Layers:** The function applies weights to the inputs and directs them through an activation function as the output in a hidden layer that is situated between the algorithm's input and output. In a nutshell, the network's inputs undergo nonlinear transformations by the hidden layers.

Neural networks with one to two hidden layers would work for data that is less complex and has fewer dimensions or features. Three to five hidden layers can be used to achieve the best possible solution for data with large dimensions or features.

**Hidden Units:** In a neural network, the components that make up the processor hidden layers between the input and output units.

3)**Output Layer:** It is a unit or units representing the target field(s)

There are a variety of connection strengths (or weights) between the units. The values are transmitted from each neuron to every neuron in the next layer as input data are presented to the first layer. The output layer eventually delivers a result.

**Dropout Layer:** By reducing the network's reliance on input features, the dropout layer aids in the prevention of overfitting. The model becomes more robust and better adapts to new data by randomly eliminating some input units during each training iteration.

**Regularization:** Regularization assists with forestalling overfitting by adding a punishment term to the misfortune capability that urges the loads to remain little. The model can better generalize to new data and is less sensitive to minor changes in the input data as a result. You can further enhance the model's capacity for generalization and reduce the likelihood of overfitting by combining regularization with dropout.

The label will be either 0 or 1, where 0 denotes a negative review and 1 denotes a positive review, as the IMDB dataset is a binary classification problem.

Because only the top 10,000 most frequent words were retained and the remainder were discarded, the maximum integer index will be less than or equal to 9999.

### **Data Analysis:**

Here I have executed the code using 1 hidden layer and 3 hidden layers, an example code snippet for 3 hidden layers with 128 hidden unit is as below:

```
from tensorflow import keras  
  
from tensorflow.keras import layers  
  
from tensorflow.keras import regularizers  
  
from keras.layers import Dense  
  
from keras.layers import Dropout
```

```

model = keras.Sequential([
    layers.Dense(128, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(128, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(128, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid", kernel_regularizer=keras.regularizers.l1(0.01))
])

```

A sequential model, which is a linear stack of layers, is what the model is called. The model in this instance consists of a dropout layer, four dense layers, and a final dense layer with a sigmoid activation function for binary output.

Each of the dense layers uses a hyperbolic tangent activation function (tanh), which has 128 units and produces outputs in the range (-1, 1). The first three dense layers also employ L2 regularization with a strength of 0.01, which adds a penalty term to the loss function to discourage large weights and prevents overfitting. The final dense layer promotes sparsity in the weights by employing L1 regularization with a strength of 0.01.

After each dense layer, a 0.5-rate dropout layer is added, removing half of the input units at random during training to avoid overfitting.

The model is intended to output a single binary value expressing either positive or negative sentiment from the 10,000-line binary vector representation of movie reviews.

Here I have used the optimizers SGD, adam and adagrad. The best among the three optimizers is used for the final code execution.

The loss function used is mse which is mean square error as instructed in the question.

### **Training the model:**

```

x_val = x_train[:10000]

partial_x_train = x_train[10000:]

y_val = y_train[:10000]

partial_y_train = y_train[10000:]

```

The x\_train and y\_train datasets were previously defined as the binary vector representations of the movie reviews and their corresponding labels, respectively. The x\_train dataset has a shape of (25000, 10000), indicating that there are 25,000 reviews in the training set, each represented as a binary vector

with length 10,000. The `y_train` dataset has a shape of (25000,), indicating that there are 25,000 labels corresponding to the 25,000 reviews in the `x_train` dataset.

The first 10,000 reviews and labels in this code have been chosen as the validation set, which will be used to evaluate the model's performance during training. The remaining 15,000 reviews and labels have been chosen to serve as the partial training set for the model.

The first 10,000 reviews and labels from `x_train` and `y_train` are selected to create the `x_val` and `y_val` datasets, respectively. The remaining 15,000 reviews and labels from `x_train` and `y_train` are selected to create the `partial_x_train` and `partial_y_train` datasets, respectively.

We can detect overfitting and adjust the model accordingly by using a validation set during training to monitor the model's performance on data it has never seen before.

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

The partial training set, which consists of 15,000 reviews and labels, was defined in the previous code block as the `partial_x_train` and `partial_y_train` datasets, respectively. These datasets serve as the model's training data.

The number of times to iterate over the entire training dataset during training is specified by the `epochs` argument, which is set to 20. An epoch is an iteration over the entire dataset.

The number of samples used in each training batch is specified by setting the `batch_size` argument to 512. The gradients determined by the loss function for each batch will be used by the model to adjust its weights during training.

The validation set that will be utilized during training is specified by setting the `validation_data` argument to (`x_val`, `y_val`). In the previous code block, the first 10,000 reviews and labels from the training set were defined as the validation set.

### **Retraining a model from scratch:**

```
model = keras.Sequential([
    layers.Dense(128, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(128, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(128, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
```

```

layers.Dropout(0.5),

layers.Dense(1, activation="sigmoid", kernel_regularizer=keras.regularizers.l1(0.01))

])

model.compile(optimizer="adam",

              loss="mse",

              metrics=["accuracy"])

model.fit(x_train, y_train, epochs=4, batch_size=512)

results = model.evaluate(x_test, y_test)

```

This code block trains the model with the same model architecture as before but for fewer epochs (epochs=4) and the full training data. The evaluation method is used to evaluate the trained model on the test set, returning the test loss and accuracy.

```
model.predict(x_test)
```

The predicted sentiment scores for the test dataset can be obtained using the model.predict() function. The function will give you an array of predicted sentiment scores, with each score representing the predicted sentiment score for the review in the test dataset that corresponds to it.

The sentiment scores range from 0 to 1, with scores closer to 0 representing negativity and scores closer to 1 representing optimism.

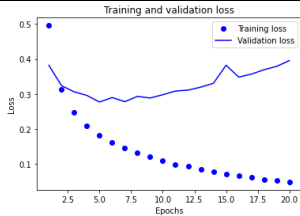
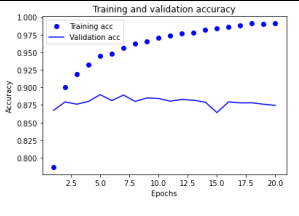
### Observations:

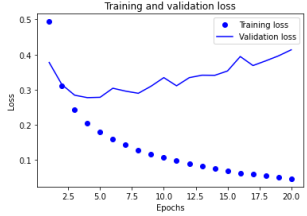
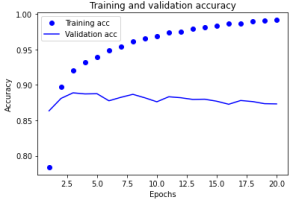
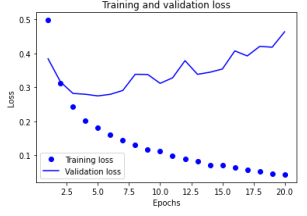
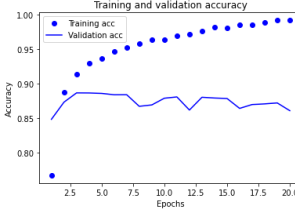
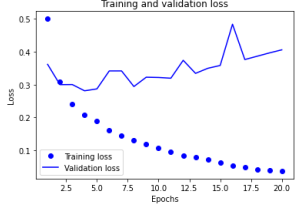
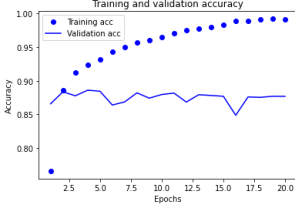
For this question I have executed the code in different steps by adding the features in a stepwise manner and observed the loss, accuracy, train validation accuracy, test accuracy.

### Case 1: Hidden layer 1 – Hidden units -32,64,128

I have used the code provided which is using IMDB dataset with 2 hidden layers with 16 hidden units and modified it for the 1 hidden layer and hidden units of 32,64 and 128.

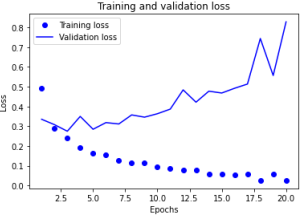
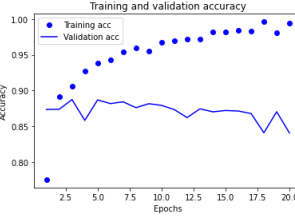
**Step 1:** I have executed the code after modifying it for 1 hidden layer and got the results as in below table:

Parameters	Loss	Accuracy	Validation accuracy	Test accuracy	Plot Validation loss	Plot Validation accuracy
Hidden layer 1 Hidden units 16 Dropout: No Regularization: No Tanh: NO Mse: No	0.0487	0.9911	0.3955	0.8744	 <p>Training and validation loss plot showing Training loss (blue dots) and Validation loss (blue line) over 20 epochs. Training loss decreases from 0.5 to approximately 0.05, while validation loss decreases from 0.4 to approximately 0.35.</p>	 <p>Training and validation accuracy plot showing Training acc (blue dots) and Validation acc (blue line) over 20 epochs. Training accuracy increases from 0.8 to approximately 0.99, while validation accuracy increases from 0.4 to approximately 0.87.</p>

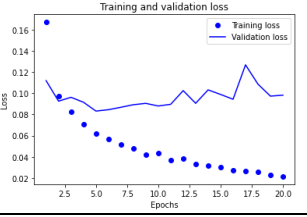
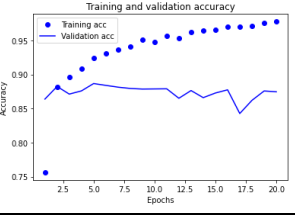
Hidden units 32 Dropout: No Regularization: No Tanh: NO Mse: No	0.0466	0.9915	0.4136	0.8732		
Hidden units 64 Dropout: No Regularization: No Tanh: NO Mse: No	0.0439	0.9919	0.8611	0.8681		
Hidden units 128 Dropout: No Regularization: No Tanh: NO Mse: NO	0.0373	0.9917	0.8769	0.8757		

From above table we can observe that test accuracy is more which is of 0.8757 while using it with 128 hidden units. Therefore, I performed remaining steps on this.

**Step 2:** tanh activation: I have changed the activation from relu to tanh as per the question.

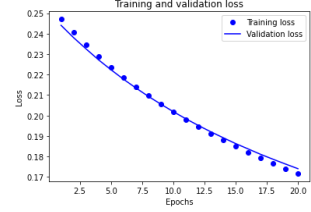
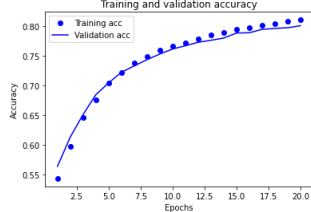
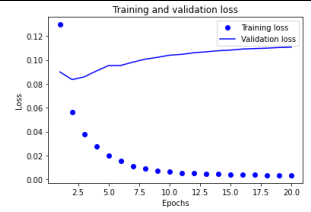
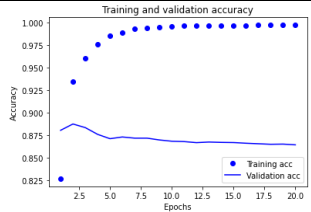
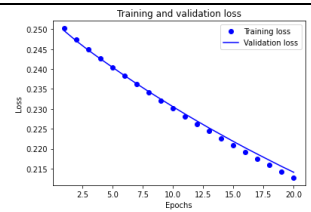
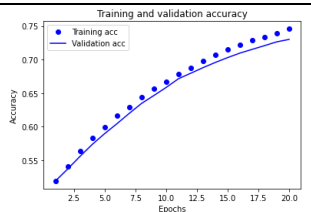
Activation: tanh (128) Dropout: No Regularization: No Tanh: Yes Mse: NO	0.0268	0.9945	0.8406	0.8828		
---	--------	--------	--------	--------	--	---

**Step 3:** I have changed the loss to mse as per the question.

Loss: mse Dropout: No Regularization: No Tanh: Yes Mse: Yes	0.0214	0.9775	0.8745	0.8828		
--	--------	--------	--------	--------	--	---

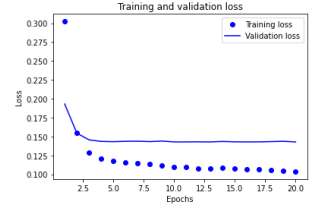
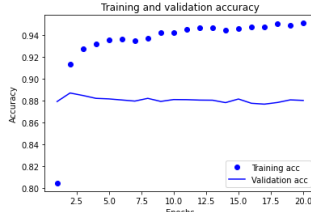


**Step 4:** I have changed to optimizers to sgd, adm, adagrad and observed the accuracy results which provides more accuracy.

Optimizer: SGD Dropout: No Regularization: No Tanh: Yes Mse: Yes	0.1717	0.8105	0.8006	0.7359		
Optimizer: adam Dropout: No Regularization: No Tanh: Yes Mse: Yes	0.0034	0.9969	0.8641	0.8714		
Optimizer: adagrad Dropout: No Regularization: No Tanh: Yes Mse: Yes	0.2127	0.7459	0.7301	0.6052		

As observed from the above table best accuracy is of 0.8714 is obtained when adam is used as optimizer. Therefore, regularization and dropout are executed by using the same.

**Step 5:** I have included the regularization using l1 and l2 regularizes and a dropout of 0.5 which further improved the accuracy.

Regularization & Dropout 128 final Dropout: Yes Regularization: Yes Tanh: Yes Mse: Yes	0.1037	0.9507	0.8800	0.8820		
---	--------	--------	--------	--------	--	---

The above table results are obtained by using the following features:

Hidden layer:1

Hidden units:128

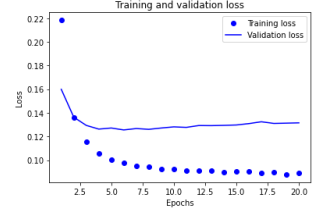
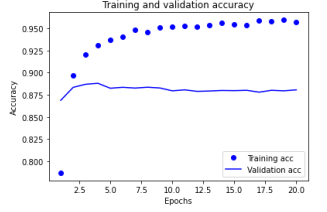
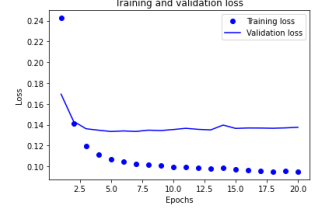
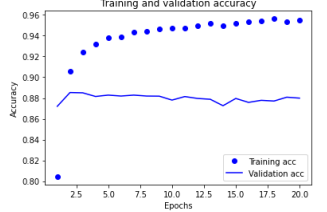
Activation: tanh

Loss: mse

Optimizer: Adam

Dropout:0.5

Similarly below are results while using 32 and 64 hidden units after performing the required modifications.

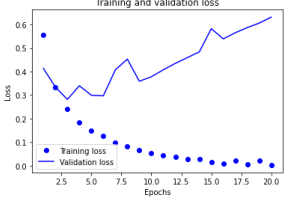
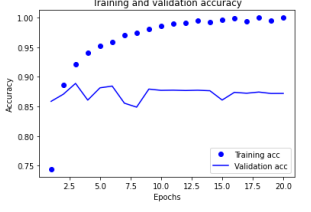
Hidden unit 32 final Dropout: Yes Regularization: Yes Tanh: Yes Mse: Yes	0.0892	0.9575	0.8806	0.8838		
Hidden unit 64 final Dropout: Yes Regularization: Yes Tanh: Yes Mse: Yes	0.0951	0.9549	0.8799	0.8857		

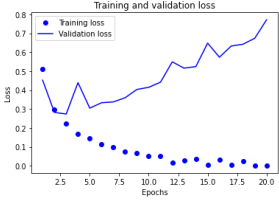
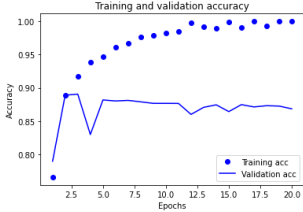
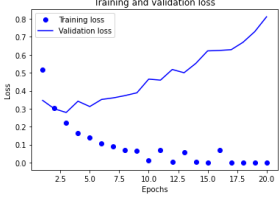
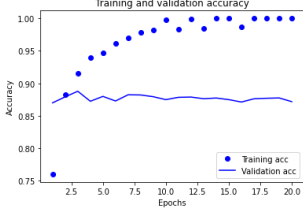
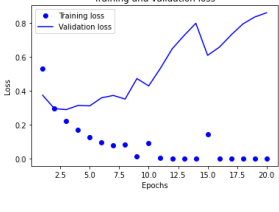
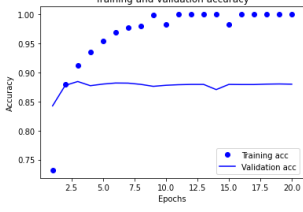
From these results we can observe that the final test accuracy is 88% irrespective of hidden units.

## Case 2: Hidden layer 3 – Hidden units -32,64,128

I have used the code provided which is using IMDB dataset with 2 hidden layers with 16 hidden units and modified it for the 3 hidden layer and hidden units of 32,64 and 128.

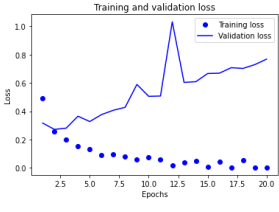
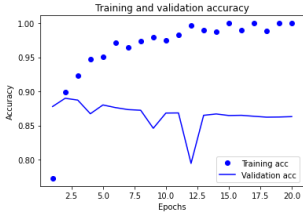
**Step 1:** I have executed the code after modifying it for 3 hidden layer and got the results as in below table:

Parameters	Loss	Accuracy	Validation accuracy	Test accuracy	Plot Validation loss	Plot Validation accuracy
Hidden layer 3 Hidden units 16 Dropout: No Regularization: No Tanh: NO Mse: No	0.0027	0.9999	0.8720	0.8784		

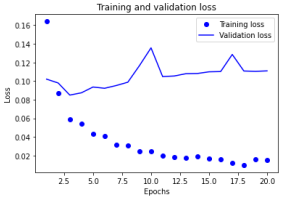
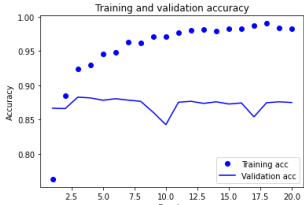
Hidden units 32 Dropout: No Regularization: No Tanh: NO Mse: No	0.0014	0.9999	0.8684	0.8830		
Hidden units 64 Dropout: No Regularization: No Tanh: NO Mse: No	0.0951	0.9999	0.8718	0.8721		
Hidden units 128 Dropout: No Regularization: No Tanh: NO Mse: No	0.0042	0.9998	0.8799	0.7896		

From above table we can observe that test accuracy is more which is of 0.8830 while using it with 32 hidden units. Therefore, I performed remaining steps on this.

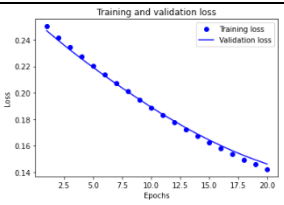
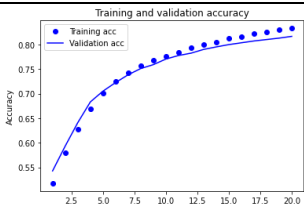
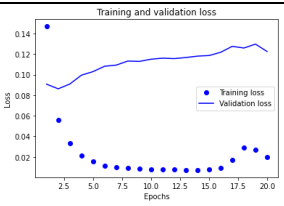
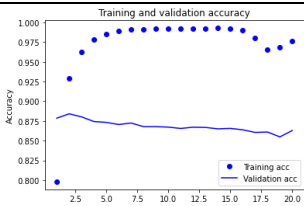
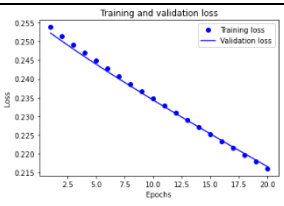
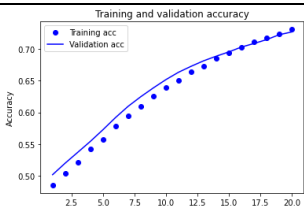
**Step 2:** tanh activation: I have changed the activation from relu to tanh as per the question.

Activation: tanh (32) Dropout: No Regularization: No Tanh: Yes Mse: No	0.0024	0.9998	0.8630	0.8778		
--	--------	--------	--------	--------	--	---

**Step 3:** I have changed the loss to mse as per the question.

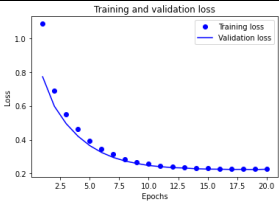
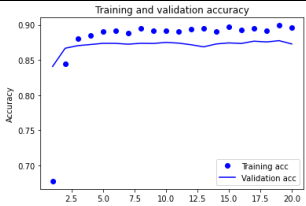
Loss: mse Dropout: No Regularization: No Tanh: Yes Mse: Yes	0.0155	0.9831	0.8748	0.8818		
---	--------	--------	--------	--------	--	---

**Step 4:** I have changed to optimizers to sgd, adm, adagrad and observed the accuracy results which provides more accuracy.

Optimizer: SGD Dropout: No Regularization: No Tanh: Yes Mse: Yes	0.1423	0.8330	0.8166	0.7435		
Optimizer: adam Dropout: No Regularization: No Tanh: Yes Mse: Yes	0.0201	0.9763	0.8629	0.8673		
Optimizer: adagrad Dropout: No Regularization: No Tanh: Yes Mse: Yes	0.2160	0.7309	0.7269	0.6140		

As observed from the above table best accuracy is of 0.8673 is obtained when adam is used as optimizer. Therefore, regularization and dropout are executed by using the same.

**Step 5:** I have included the regularization using l1 and l2 regularizes and a dropout of 0.5 which further improved the accuracy.

Regularization & Dropout 32 final Dropout: Yes Regularization: Yes Tanh: Yes Mse: Yes	0.2266	0.8961	0.8728	0.8730		
--	--------	--------	--------	--------	--	---

The above table results are obtained by using the following features:

Hidden layer: 3

Hidden units: 32

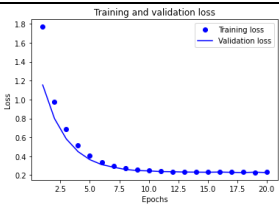
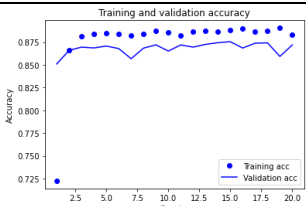
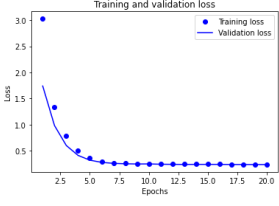
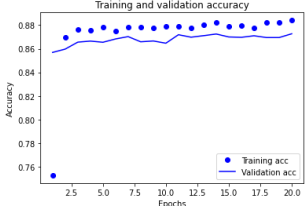
Activation: tanh

Loss: mse

Optimizer: Adam

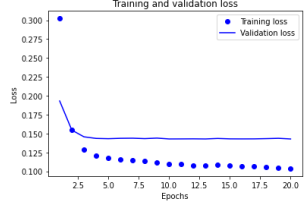
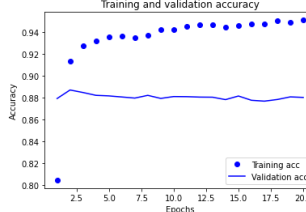
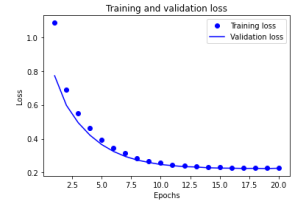
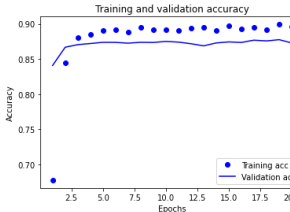
Dropout: 0.5

Similarly, below are results while using 64 and 128 hidden units after performing the required modifications.

Hidden unit 64 final Dropout: Yes Regularization: Yes Tanh: Yes Mse: Yes	0.2321	0.8831	0.8719	0.8672		
Hidden unit 128 final Dropout: Yes Regularization: Yes Tanh: Yes Mse: Yes	0.2366	0.8841	0.8727	0.8638		

From these results we can observe that the final test accuracy is 86% irrespective of hidden units.

**Conclusion:** After observing the results we can say that by varying values and functions of loss, optimizer, activation, Hidden layers, hidden units, regularization, and dropout will influence the accuracy of the model. In our model the accuracy increased from 87% to 88% for 1 hidden layer and validation accuracy increased from 86% to 87% after performing the required changes as per the problem statement. The results are as below the table.

Regularization & Dropout Hidden layer:1 128 final Dropout: Yes Regularization: Yes Tanh: Yes Mse: Yes	0.1037	0.9507	0.8800	0.8820		
Regularization & Dropout 32 final Hidden layer:3 Dropout: Yes Regularization: Yes Tanh: Yes Mse: Yes	0.2266	0.8961	0.8728	0.8730		

#### Reference:

- 1) <https://www.ibm.com/docs/en/spss-modeler/18.0.0?topic=networks-neural-model>
- 2) <https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning#:~:text=In%20neural%20networks%2C%20a%20hidden,inputs%20entered%20into%20the%20network.>

**Appendix:** Here I am pasting the final code that I used for the hidden layer 1 with 128 hidden units and hidden layer 3 with 32 hidden units.

### **Hidden layer 1 with 128 hidden units:**

Classifying movie reviews: A binary classification example

#### **The IMDB dataset**

##### **Loading the IMDB dataset**

```
[ ]  
from tensorflow.keras.datasets import imdb  
  
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(  
    num_words=10000)  
  
[ ]  
train_data[0]  
train_labels[0]  
  
[ ]  
max([max(sequence) for sequence in train_data])
```

##### **Decoding reviews back to text**

```
[ ]  
word_index = imdb.get_word_index()  
reverse_word_index = dict(  
    [(value, key) for (key, value) in word_index.items()])  
decoded_review = " ".join(  
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

##### **Preparing the data**

##### **Encoding the integer sequences via multi-hot encoding**

```
[ ]  
import numpy as np
```

```
def vectorize_sequences(sequences, dimension=10000):
```

```
    results = np.zeros((len(sequences), dimension))
```

```
    for i, sequence in enumerate(sequences):
```

```
        for j in sequence:
```

```
            results[i, j] = 1.
```

```
    return results
```

```
x_train = vectorize_sequences(train_data)
```

```
x_test = vectorize_sequences(test_data)
```

```
[ ]
```

```
x_train[0]
```

```
[ ]
```

```
y_train = np.asarray(train_labels).astype("float32")
```

```
y_test = np.asarray(test_labels).astype("float32")
```

## **Building your model**

### **Model definition**

```
[ ]
```

```
from tensorflow import keras
```

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

```
from tensorflow.keras import regularizers
```

```
from keras.layers import Dense
```

```
from keras.layers import Dropout
```

```
model = keras.Sequential([
```

```
    layers.Dense(128, activation="tanh", kernel_regularizer=regularizers.L2(0.001)),
```

```
    layers.Dropout(0.5),
```

```
    layers.Dense(1, activation="sigmoid", kernel_regularizer=regularizers.l1(0.001))
```

```
])
```

### **Compiling the model**



```
[ ]
```

```
model.compile(optimizer="adam",  
              loss="mse",  
              metrics=["accuracy"])
```

Validating your approach

Setting aside a validation set

```
[ ]
```

```
x_val = x_train[:10000]  
partial_x_train = x_train[10000:]  
y_val = y_train[:10000]  
partial_y_train = y_train[10000:]
```

Training your model

```
[ ]
```

```
history = model.fit(partial_x_train,  
                    partial_y_train,  
                    epochs=20,  
                    batch_size=512,  
                    validation_data=(x_val, y_val))
```

```
[ ]
```

```
history_dict = history.history  
history_dict.keys()  
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

**Plotting the training and validation loss**

```
[ ]
```

```
import matplotlib.pyplot as plt
```

```
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

### **Plotting the training and validation accuracy**

```
[ ]
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

### **Retraining a model from scratch**

```
[ ]
```

```

model = keras.Sequential([
    layers.Dense(128, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid", kernel_regularizer=regularizers.l1(0.001))
])
model.compile(optimizer="adam",
              loss="mse",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
[ ]
results
[0.14226676523685455, 0.8819599747657776]

```

### **Using a trained model to generate predictions on new data**

```

[ ]
model.predict(x_test)
782/782 [=====] - 5s 6ms/step
array([[0.2599414 ],
       [0.99504805],
       [0.41712162],
       ...,
       [0.2000265 ],
       [0.14244986],
       [0.43598858]], dtype=float32)

```

### **Hidden layer 3 with 32 hidden units:**

#### **Classifying movie reviews: A binary classification example**

##### **The IMDB dataset**

##### **Loading the IMDB dataset**

```
[ ]
```

```
from tensorflow.keras.datasets import imdb
```

```
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(  
    num_words=10000)
```

```
[ ]
```

```
train_data[0]
```

```
[ ]
```

```
train_labels[0]
```

```
[ ]
```

```
max([max(sequence) for sequence in train_data])
```

### **Decoding reviews back to text**

```
[ ]
```

```
word_index = imdb.get_word_index()
```

```
reverse_word_index = dict(  
    [(value, key) for (key, value) in word_index.items()])
```

```
decoded_review = " ".join(  
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

### **Preparing the data**

#### **Encoding the integer sequences via multi-hot encoding**

```
[ ]
```

```
import numpy as np
```

```
def vectorize_sequences(sequences, dimension=10000):
```

```
    results = np.zeros((len(sequences), dimension))
```

```
    for i, sequence in enumerate(sequences):
```

```
        for j in sequence:
```

```
            results[i, j] = 1.
```

```

    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

[ ]

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

[ ]

y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")

```

## **Building your model**

### **Model definition**

```

[ ]

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras import regularizers

from keras.layers import Dense

from keras.layers import Dropout

model = keras.Sequential([
    layers.Dense(32, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(32, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(32, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid", kernel_regularizer=keras.regularizers.l1(0.01))
])

```

### **Compiling the model**

```
[ ]
```

```
model.compile(optimizer="adam",  
              loss="mse",  
              metrics=["accuracy"])
```

Validating your approach

Setting aside a validation set

```
[ ]
```

```
x_val = x_train[:10000]  
partial_x_train = x_train[10000:]  
y_val = y_train[:10000]  
partial_y_train = y_train[10000:]
```

**Training your model**

```
[ ]
```

```
history = model.fit(partial_x_train,  
                    partial_y_train,  
                    epochs=20,  
                    batch_size=512,  
                    validation_data=(x_val, y_val))
```

```
[ ]
```

```
history_dict = history.history  
history_dict.keys()  
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

**Plotting the training and validation loss**

```
[ ]
```

```
import matplotlib.pyplot as plt
```

```
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

### **Plotting the training and validation accuracy**

```
[ ]
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

### **Retraining a model from scratch**

```
[ ]
```

```

model = keras.Sequential([
    layers.Dense(32, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(32, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(32, activation="tanh", kernel_regularizer=keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid", kernel_regularizer=keras.regularizers.l1(0.01))
])
model.compile(optimizer="adam",
              loss="mse",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)

[ ]
results
[0.30825385451316833, 0.8730400204658508]

```

### Using a trained model to generate predictions on new data

```

[ ]
model.predict(x_test)
782/782 [=====] - 4s 4ms/step
array([[0.41675007],
       [0.9303876 ],
       [0.6515235 ],
       ...,
       [0.2749144 ],
       [0.26769418],
       [0.48538905]], dtype=float32)

```



