```
# Imports & dataset load
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from tensorflow.keras.datasets import imdb
from sklearn.metrics import confusion_matrix, classification_report

os.makedirs("images", exist_ok=True)

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

print("Loaded:", len(train_data), "train examples and", len(test_data), "
```

```
Loaded: 25000 train examples and 25000 test examples
```

```
# Inspecting and decoding data
word_index = imdb.get_word_index()
reverse_word_index = dict((value, key) for (key, value) in word_index.iter

def decode_review(seq):
    return " ".join([reverse_word_index.get(i - 3, "?") for i in seq])

print("First review token IDs:", train_data[0][:30])
print("First label:", train_labels[0])
print("Decoded review (preview):", decode_review(train_data[0])[:300], ".
```

```
First review token IDs: [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458,
First label: 1
Decoded review (preview): ? this film was just brilliant casting location
```

```
# Vectorizing sequences
def vectorize_sequences(sequences, dimension=num_words):
    results = np.zeros((len(sequences), dimension), dtype="float32")
    for i, sequence in enumerate(sequences):
        for j in sequence:
            if j < dimension:
                results[i, j] = 1.0
    return results

x_train = vectorize_sequences(train_data)
x_test  = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype("float32")
y_test  = np.asarray(test_labels).astype("float32")
```

```
print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
```

```
x_train shape: (25000, 10000)
y_train shape: (25000,)
```

```
# Building model
def build_model(hidden_layers=2, units=16, activation="tanh", output_act=
                l2_weight=0.0, dropout_rate=0.0, input_shape=(num_words,)
    model = keras.Sequential()
    if hidden_layers >= 1:
        model.add(layers.Dense(units, activation=activation,
                               kernel_regularizer=regularizers.l2(l2_weigh
                               input_shape=input_shape))
        if dropout_rate > 0:
            model.add(layers.Dropout(dropout_rate))
    for _ in range(hidden_layers - 1):
        model.add(layers.Dense(units, activation=activation,
                               kernel_regularizer=regularizers.l2(l2_weigh
        if dropout_rate > 0:
            model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(1, activation=output_act))
    return model
```

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

print("partial_x_train shape:", partial_x_train.shape)
print("x_val shape:", x_val.shape)
```

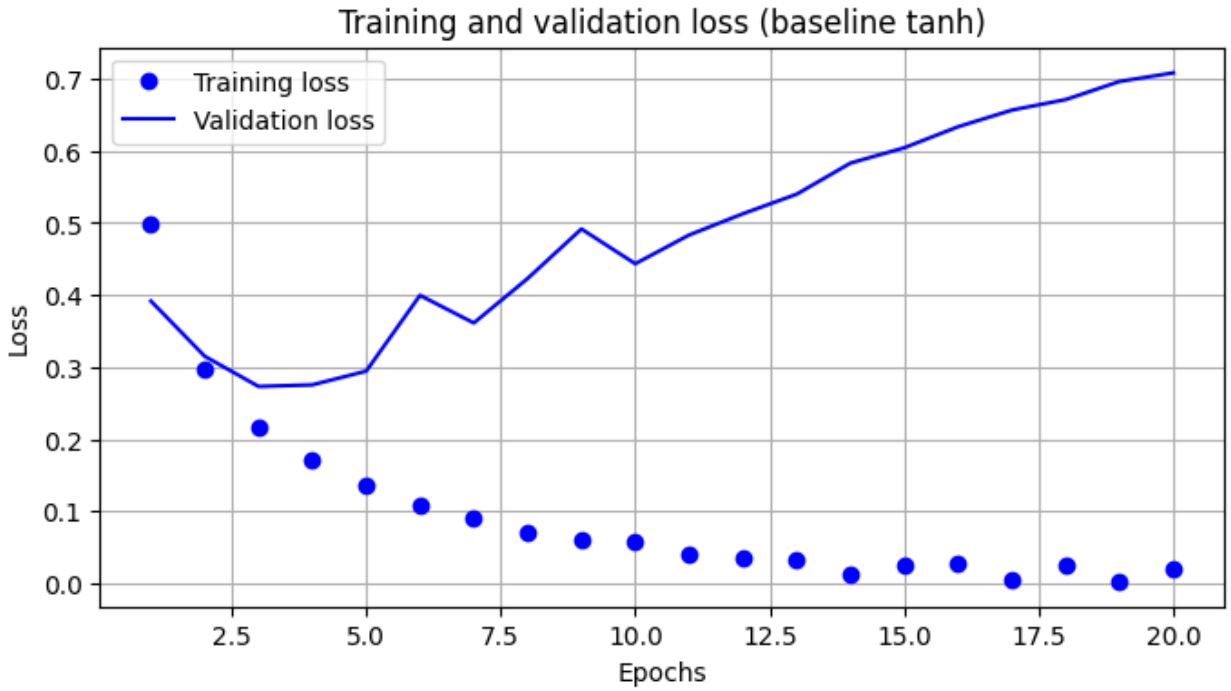
```
partial_x_train shape: (15000, 10000)
x_val shape: (10000, 10000)
```

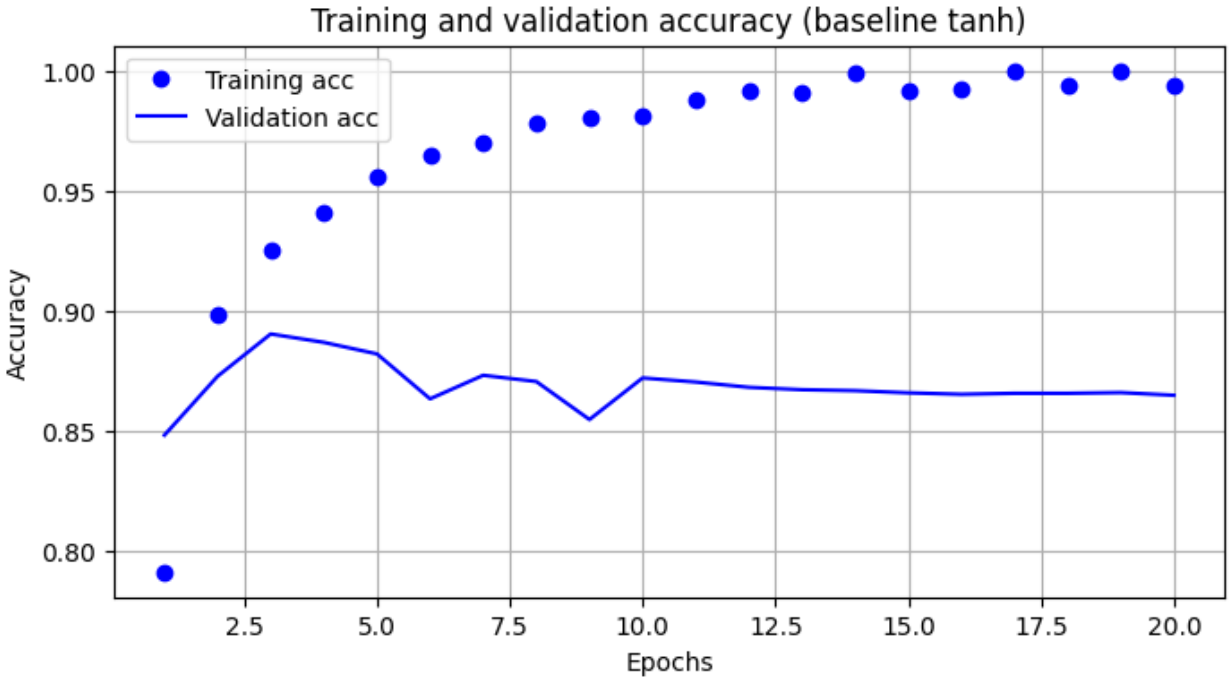
```
# Baseline model
baseline = build_model(hidden_layers=2, units=16, activation="tanh")
baseline.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=

history = baseline.fit(partial_x_train, partial_y_train,
                       epochs=20, batch_size=512,
                       validation_data=(x_val, y_val), verbose=2)
history_baseline = history.history
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
30/30 - 3s - 103ms/step - accuracy: 0.7913 - loss: 0.4996 - val_accuracy: (
Epoch 2/20
30/30 - 1s - 41ms/step - accuracy: 0.8981 - loss: 0.2981 - val_accuracy: 0
Epoch 3/20
30/30 - 1s - 49ms/step - accuracy: 0.9254 - loss: 0.2177 - val_accuracy: 0
Epoch 4/20
30/30 - 2s - 60ms/step - accuracy: 0.9409 - loss: 0.1703 - val_accuracy: 0
Epoch 5/20
30/30 - 2s - 63ms/step - accuracy: 0.9559 - loss: 0.1359 - val_accuracy: 0
Epoch 6/20
30/30 - 1s - 37ms/step - accuracy: 0.9646 - loss: 0.1085 - val_accuracy: 0
Epoch 7/20
30/30 - 1s - 40ms/step - accuracy: 0.9702 - loss: 0.0916 - val_accuracy: 0
Epoch 8/20
30/30 - 1s - 40ms/step - accuracy: 0.9779 - loss: 0.0710 - val_accuracy: 0
Epoch 9/20
30/30 - 1s - 37ms/step - accuracy: 0.9805 - loss: 0.0614 - val_accuracy: 0
Epoch 10/20
30/30 - 1s - 44ms/step - accuracy: 0.9808 - loss: 0.0576 - val_accuracy: 0
Epoch 11/20
30/30 - 1s - 41ms/step - accuracy: 0.9876 - loss: 0.0414 - val_accuracy: 0
Epoch 12/20
30/30 - 1s - 43ms/step - accuracy: 0.9913 - loss: 0.0340 - val_accuracy: 0
Epoch 13/20
30/30 - 2s - 57ms/step - accuracy: 0.9906 - loss: 0.0330 - val_accuracy: 0
Epoch 14/20
30/30 - 2s - 53ms/step - accuracy: 0.9988 - loss: 0.0127 - val_accuracy: 0
Epoch 15/20
30/30 - 1s - 37ms/step - accuracy: 0.9914 - loss: 0.0259 - val_accuracy: 0
Epoch 16/20
30/30 - 1s - 37ms/step - accuracy: 0.9919 - loss: 0.0267 - val_accuracy: 0
Epoch 17/20
30/30 - 1s - 38ms/step - accuracy: 0.9997 - loss: 0.0053 - val_accuracy: 0
Epoch 18/20
30/30 - 1s - 38ms/step - accuracy: 0.9935 - loss: 0.0245 - val_accuracy: 0
Epoch 19/20
30/30 - 1s - 44ms/step - accuracy: 0.9999 - loss: 0.0031 - val_accuracy: 0
Epoch 20/20
30/30 - 1s - 41ms/step - accuracy: 0.9940 - loss: 0.0201 - val_accuracy: 0
```

```
# Plotting training & validation curves
epochs = range(1, len(history_baseline["loss"]) + 1)
plt.figure(figsize=(8,4))
plt.plot(epochs, history_baseline["loss"], "bo", label="Training loss")
plt.plot(epochs, history_baseline["val_loss"], "b",  label="Validation loss
plt.title("Training and validation loss (baseline tanh)")
plt.xlabel("Epochs"); plt.ylabel("Loss"); plt.legend(); plt.grid(True)
plt.savefig("images/baseline_loss_tanh.png"); plt.show()
```

```
plt.figure(figsize=(8,4))
plt.plot(epochs, history_baseline["accuracy"], "bo", label="Training acc")
plt.plot(epochs, history_baseline["val_accuracy"], "b",  label="Validation
plt.title("Training and validation accuracy (baseline tanh)")
plt.xlabel("Epochs"); plt.ylabel("Accuracy"); plt.legend(); plt.grid(True)
plt.savefig("images/baseline_acc_tanh.png"); plt.show()
```





```
# ✅ Retrainning final model for best epoch
best_epoch = np.argmax(history_baseline["val_accuracy"]) + 1
print("Best epoch (baseline tanh):", best_epoch)

final_model = build_model(hidden_layers=2, units=16, activation="tanh")
final_model.compile(optimizer="rmsprop", loss="binary_crossentropy", metr
final_model.fit(x_train, y_train, epochs=best_epoch, batch_size=512, verb

results = final_model.evaluate(x_test, y_test, verbose=2)
print("Test loss, Test accuracy:", results)
```

```
Best epoch (baseline tanh): 3
Epoch 1/3
49/49 - 3s - 63ms/step - accuracy: 0.8198 - loss: 0.4432
Epoch 2/3
49/49 - 2s - 36ms/step - accuracy: 0.9066 - loss: 0.2555
Epoch 3/3
49/49 - 2s - 36ms/step - accuracy: 0.9270 - loss: 0.1971
782/782 - 3s - 4ms/step - accuracy: 0.8859 - loss: 0.2807
Test loss, Test accuracy: [0.28068244457244873, 0.8858799934387207]
```

```
# Predictions & confusion matrix
y_prob = final_model.predict(x_test, verbose=0).flatten()
y_pred = (y_prob > 0.5).astype("int32")
cm = confusion_matrix(y_test.astype("int32"), y_pred)
print("Confusion matrix:\n", cm)
print("\nClassification report:\n", classification_report(y_test.astype("
```

```
Confusion matrix:
 [[11038  1462]
 [ 1391 11109]]

Classification report:
              precision    recall  f1-score   support

           0     0.8881    0.8830    0.8856     12500
           1     0.8837    0.8887    0.8862     12500

    accuracy                         0.8859     25000
   macro avg     0.8859    0.8859    0.8859     25000
weighted avg     0.8859    0.8859    0.8859     25000
```

```
# ✅ Required experiments
configs = [
    {"id":"1layer_16", "hidden_layers":1, "units":16, "activation":"tanh"
    {"id":"3layer_16", "hidden_layers":3, "units":16, "activation":"tanh"
    {"id":"2layer_32", "hidden_layers":2, "units":32, "activation":"tanh"
    {"id":"2layer_64", "hidden_layers":2, "units":64, "activation":"tanh"
```

```
    {"id":"2layer_64", "hidden_layers":2, "units":64, "activation":"tanh"
    {"id":"2layer_mse", "hidden_layers":2, "units":16, "activation":"tanh"
    {"id":"2layer_dropout", "hidden_layers":2, "units":16, "activation":"t
    {"id":"2layer_l2", "hidden_layers":2, "units":16, "activation":"tanh"
]

results_list = []
for cfg in configs:
    print("\nRunning config:", cfg["id"])
    model_cfg = build_model(hidden_layers=cfg["hidden_layers"],
                            units=cfg["units"],
                            activation=cfg["activation"],
                            l2_weight=cfg["l2"],
                            dropout_rate=cfg["dropout"])
    model_cfg.compile(optimizer="rmsprop", loss=cfg["loss"], metrics=["ac
    hist = model_cfg.fit(partial_x_train, partial_y_train, epochs=10, bat
                         validation_data=(x_val, y_val), verbose=0)
    best_val_acc = max(hist.history["val_accuracy"])
    best_epoch = np.argmax(hist.history["val_accuracy"]) + 1
    model_cfg_full = build_model(hidden_layers=cfg["hidden_layers"],
                                 units=cfg["units"],
                                 activation=cfg["activation"],
                                 l2_weight=cfg["l2"],
                                 dropout_rate=cfg["dropout"])
    model_cfg_full.compile(optimizer="rmsprop", loss=cfg["loss"], metrics=
    model_cfg_full.fit(x_train, y_train, epochs=best_epoch, batch_size=51
    test_loss, test_acc = model_cfg_full.evaluate(x_test, y_test, verbose=
    results_list.append({
        "config": cfg["id"],
        "hidden_layers": cfg["hidden_layers"],
        "units": cfg["units"],
        "activation": cfg["activation"],
        "loss": cfg["loss"],
        "dropout": cfg["dropout"],
        "l2": cfg["l2"],
        "best_val_acc": float(best_val_acc),
        "best_epoch": int(best_epoch),
        "test_acc": float(test_acc),
        "test_loss": float(test_loss),
    })
    print("  best_val_acc=%.4f, best_epoch=%d, test_acc=%.4f" % (best_val_

results_df = pd.DataFrame(results_list)
results_df.to_csv("images/assignment2_results.csv", index=False)
results_df
```

```
Running config: 1layer_16
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
  best_val_acc=0.8883, best_epoch=4, test_acc=0.8860
```