ADVANCED MACHINE LEARNING

(BA – 64061)

<u>ASSIGNMENT – 1</u>

NEURAL NEWORKS – IMBD DATASET

DIVYA CHANDRASEKARAN

ID: 811284790

EXECUTIVE SUMMARY

The IMDB dataset presents a binary classification task aimed at discerning whether a movie review conveys a positive or negative review. With a dataset containing 50,000 reviews, each reduced to 150 words, our study delves into various sample sizes for training (100, 500, 1000, and 10,000), validating on 10,000 samples, and considering only the top 10,000 words. Pre-processing techniques were applied to refine the data. Subsequently, we explored two approaches: direct embedding layer utilization and integration with a pre-trained embedding model. By adopting different methodologies, we assessed performance to ascertain the most effective strategy.

PROBLEM STATEMENT

The objective of this project is to use Keras to build a deep neural network to predict whether a review will be positive or negative in python.

DATASET

The IMDB dataset contains 50,000 highly polarized movie reviews that have been split into two equal groups of 25,000 reviews each for training and testing. There are an equal amount of favorable and negative evaluations in each set. Keras includes this dataset, which contains reviews and the labels that go with them, with 0 indicating a negative review and 1 reflecting a good review. The evaluations take the shape of a series of words that have been pre-processed into a series of numbers, each integer signifying a different word from the dictionary.

METHODOLOGY

To create our neural network using Python's Keras framework, we first split the dataset into training and test sets. Then, we loaded the data, retaining only the 10,000 most frequent terms. Next, we used a lexicon mapping to decode the evaluations back to their original text.

The sequential model was constructed with an embedding layer, which translates each word from the integer-encoded vocabulary into a feature vector of a specified size. This was followed by a dense layer with 32 hidden units and the ReLU activation function, connected to an output layer with a sigmoid activation function, generating a probability between 0 and 1.

For training, we employed binary cross-entropy as the loss function, the RMSprop optimizer, and accuracy as the evaluation metric. This model architecture aims to effectively process and classify textual data for sentiment analysis tasks.

FINDINGS

For this task, to ensure our neural network operates effectively, we initially obtained the requisite libraries. Based on my research and analysis, TensorFlow demonstrates superior implementation and support compared to alternative deep learning frameworks such as PyTorch.

List of imports are:

from tensorflow import keras from tensorflow keras import layers from keras.layers import Dense from kent.layers import Dropout.

For the purpose of this assignment, we have imported keras, keras.layers, Dense and Dropouts.

Each of its own has a significant importance in its implementation. Keras is the high-level API of TensorFlow 2: an approachable, highly productive interface for solving machine learning problems, with a focus on modern deep learning.

The core data structures of Keras are layers and models. The simplest type of model is the Sequential model, a linear stack of layers. ***Dense:*** represents the number of hidden units in the neural network. ***Dropout***: The significance of dropout is taking a bunch of inputs or hidden layer input, random remove the connections. model = keras.Sequential().

The sequential model is the simplest mode of keras, which is stack up the layers in the sequences. model.add(Dense(32,activation='tanh')) Stacking layers is easy using the. add function. Also 32 represents the number of hidden units and we use the activation function tanh.

Contents of neural networks

1. **Input Layer**: Receives input data, typically in vector form such as the IMDB dataset. (Here, we provide the vector representation of IMBD data)

2. **Hidden Layers**: Comprise dense units and can be stacked as needed based on requirements.

(It contains the number of dense units, and we can stack up as many layers as we want depending on the requirement.)

3. **Output Layer**: Usually contains one dense unit, producing the final output.

For this task, a three-layered approach was implemented as per assignment instructions, utilizing the `keras.Sequential` model.

Three layers with 32 dense units each and a hyperbolic tangent (tanh) activation function were stacked. The model was compiled using the Adagrad optimizer with mean squared error (MSE) loss. Although initially, binary cross-entropy loss is common for IMDB data, exploring regression loss like MSE was considered. Optimizers like Adam are commonly used, but Adagrad was chosen for this task.

model = keras.Sequential([layers.Dense(32, activation="tanh"), layers.Dense(32, activation="tanh"), layers.Dense(32, activation="tanh"), layers.Dense(1, activation="sigmoid")])

The above code model initialized as sequential. And we stack up three layers with 32 dense units and tanh activation function. In the task, I implemented tanh instead of relu as suggested in the assignment.

model.compile(optimizer="adagrad", loss="mean_squared_error", metrics=["accuracy"]) The above piece of code uses an optimizer adagrad with mse loss.

The IMDB dataset typically employs binary cross-entropy loss, which is suited for binary classification tasks. Switching to a regression loss like mean squared error (MSE) could alter the model's behavior, potentially affecting its performance on sentiment analysis.

Optimizers play a crucial role in minimizing errors during training. While Adam optimizer is commonly favored, Adagrad was chosen for this task. Data was partitioned into training and validation sets for model evaluation. This partitioning enables training the neural network over multiple epochs while simultaneously validating against a separate validation set, aiding in monitoring model performance and preventing overfitting.

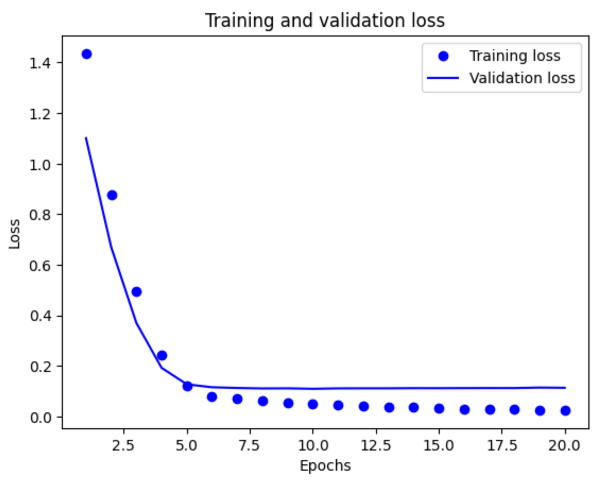
x_val = x_train[:10000] partial_x_train = x_train[10000:] y_val = y_train[:10000] partial_y_train = y_train[10000:]

Training the data history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=256, validation_data=(x_val, y_val))

Data was split into training and validation sets. The training process involved training the neural network for ***<u>20 epochs with a batch size of 256</u>***, validating against a separate validation set. While L1 and L2 regularizers were experimented with, they did not significantly impact validation accuracy.

INTERPRETATION OF GRAPHS

Graph 1 (Training and Validation Loss):

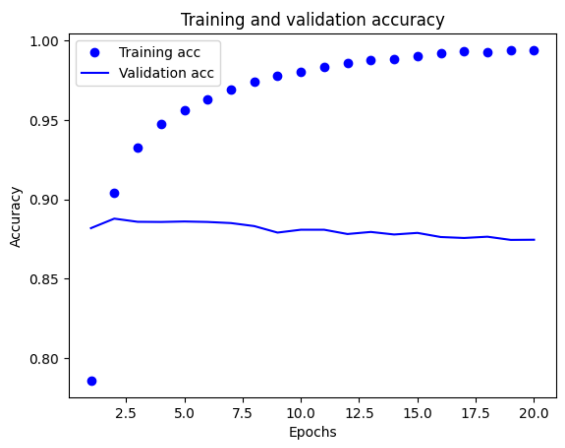


- Training loss decreases with each epoch as model fits the training data better.

- Validation loss decreases initially but stagnates and increases slightly after epoch 3. Indicates overfitting.

- Minimum validation loss occurs at epoch 3.

Graph 2 (Training and Validation Accuracy):



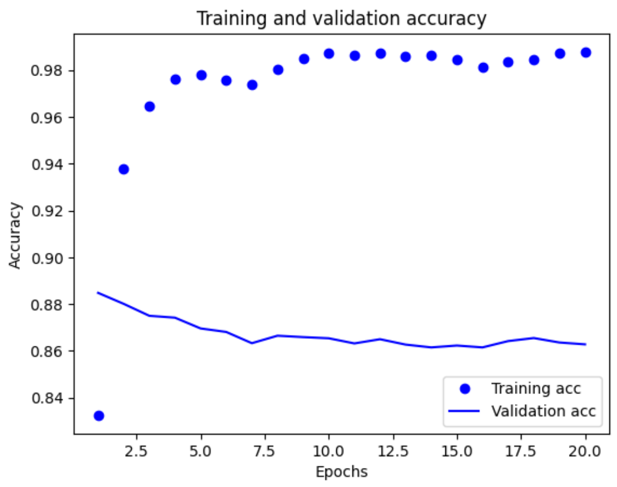
- Training accuracy increases with epochs as model learns patterns in training data.

- Validation accuracy increases initially but stagnates after epoch 3.

- Maximum validation accuracy occurs at epoch 3.

- Training accuracy continues increasing while validation accuracy stalls - overfitting.

Graph 3 (Training and Validation Accuracy with Regularization):



- Training accuracy increases while validation accuracy increases then plateaus.

- Validation accuracy is higher overall compared to Graph 2.

- Maximum validation accuracy at epoch 5.

- Less gap between training and validation accuracy indicates regularization prevents overfitting.

Key Takeaways:

- Validation metrics are key to identify overfitting and ideal number of epochs.
- Regularization helps prevent overfitting, resulting in lower validation loss and higher validation accuracy.
- For both models, 3-5 epochs are ideal before overfitting occurs.

CONCLUSIONS

1. **Neural network designed with 3 layers.**

- A neural network typically consists of multiple layers, including input, hidden, and output layers. In this case, the network architecture involves three layers.

- The input layer receives the input data, while the hidden layers process and transform the data, extracting relevant features.

- Adding more layers allows the network to learn more complex patterns in the data, potentially improving its performance.

2. Activation Functions: Tanh instead of ReLU:

- Activation functions introduce non-linearity into the neural network, enabling it to learn and represent complex relationships in the data.

- Tanh (hyperbolic tangent) and ReLU (rectified linear unit) are common activation functions. While ReLU is widely used, tanh squashes the output to the range [-1, 1], which may help in certain scenarios where the data distribution is centered around zero.

3. **Optimizer: Adam instead of RMSprop:**

- Optimizers are algorithms that adjust the weights of the neural network during training to minimize the loss function.

- Adam (Adaptive Moment Estimation) optimizer combines the advantages of RMSprop and momentum optimization, often leading to faster convergence and better performance compared to RMSprop.

4. **L1 & L2 Regularizers:**

- Regularization techniques help prevent overfitting by adding a penalty term to the loss function, discouraging large weights in the network.

- L1 and L2 regularization are two common methods. L1 regularization adds the absolute values of the weights to the loss function, while L2 regularization adds the squared values.

- By incorporating L1 and L2 regularizers, the model is encouraged to learn simpler patterns, reducing the risk of overfitting to the training data.

5. **Dropout Layer:**

- Dropout is a regularization technique where randomly selected neurons are ignored during training, effectively dropping them out of the network.

- A dropout layer with a dropout rate of 0.5 means that during training, each neuron has a 50% chance of being dropped out. That means we are dropping 50 percent of inputs during the training.

- Dropout helps prevent overfitting by introducing noise and redundancy, forcing the network to learn more robust features.

In summary, these design choices aim to enhance the neural network's performance, improve its generalization ability, and prevent overfitting to the training data.