تعداد لایه ها و نورون های مختلف را برای مثال رویترز بررسی کنید.

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import reuters

from tensorflow.keras import models, layers

from tensorflow.keras.utils import to\_categorical

کتابخانه ها را فراخوانی میکنیم.

def vectorize\_sequences(sequences, dimension=10000):

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

    for i, sequence in enumerate(sequences):

        results[i, sequence] = 1.0

    return results

(train\_data, train\_labels), (test\_data, test\_labels) = reuters.load\_data(num\_words=10000)

# Vectorize data

x\_train = vectorize\_sequences(train\_data)

x\_test = vectorize\_sequences(test\_data)

# Convert labels to one-hot encoding

y\_train = to\_categorical(train\_labels)

y\_test = to\_categorical(test\_labels)

داده ها را بارگذاری کرده و انکد میکنیم

# Function to build model with variable layers and units

def build\_model(num\_layers, units\_per\_layer):

    model = models.Sequential()

    # Input layer

    model.add(layers.Dense(units\_per\_layer, activation='relu', input\_shape=(10000,)))

    # Additional hidden layers

    for \_ in range(num\_layers - 1):

        model.add(layers.Dense(units\_per\_layer, activation='relu'))

    # Output layer

    model.add(layers.Dense(46, activation='softmax'))

    # Compile model

    model.compile(optimizer='rmsprop',

                  loss='categorical\_crossentropy',

                  metrics=['accuracy'])

    return model

برای ساخت مدل های مختلف با تعداد لایه ها و نورون های مختلف یک تابع تعریف کرده‌ایم .

# Configurations to test

configurations = [

    {'num\_layers': 1, 'units': 16},

    {'num\_layers': 1, 'units': 64},

    {'num\_layers': 1, 'units': 128},

    {'num\_layers': 2, 'units': 16},

    {'num\_layers': 2, 'units': 64},  # Baseline

    {'num\_layers': 2, 'units': 128},

    {'num\_layers': 3, 'units': 16},

    {'num\_layers': 3, 'units': 64},

    {'num\_layers': 3, 'units': 128},

]

ترکیبات مختلف از تعداد لایه ها و نورون ها را به صورت دیکشنری ذخیره کرده‌ایم.

history\_dict = {}

# Train and evaluate each configuration

for config in configurations:

    print(f"\nTesting model with {config['num\_layers']} layers and {config['units']} units")

    model = build\_model(config['num\_layers'], config['units'])

    # Split training data into training and validation sets

    x\_val = x\_train[:1000]

    partial\_x\_train = x\_train[1000:]

    y\_val = y\_train[:1000]

    partial\_y\_train = y\_train[1000:]

    # Train model

    history = model.fit(partial\_x\_train, partial\_y\_train,

                       epochs=20, batch\_size=512,

                       validation\_data=(x\_val, y\_val),

                       verbose=1)

    # Store history

    history\_dict[f"{config['num\_layers']}\_layers\_{config['units']}\_units"] = history.history

    # Evaluate on test set

    test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=0)

    print(f"Test accuracy: {test\_acc:.4f}")

مدل ها را آموزش و ارزیابی کرده و روند آموزش را داخل دیکشنری ذخیره میکنیم. نتیجه ارزیابی را چاپ میکنیم.

plt.figure(figsize=(12, 8))

for config\_name, history in history\_dict.items():

    val\_acc = history['val\_accuracy']

    plt.plot(range(1, len(val\_acc) + 1), val\_acc, label=config\_name)

plt.title('Validation Accuracy for Different Model Configurations')

plt.xlabel('Epochs')

plt.ylabel('Validation Accuracy')

plt.legend()

plt.show()

نتایج ذخیره شده در دیکشنری را به صورت نمودار رسم میکنیم

