Epoch = 100

A screenshot of a computer

Description automatically generated

Epoch = 200

A screenshot of a computer screen

Description automatically generated

Epoch = 290

A screenshot of a computer

Description automatically generated

Epoch = 300

A screen shot of a graph

Description automatically generated

Epoch = 311

A screenshot of a computer

Description automatically generated

Epoch = 356

A screen shot of a graph

Description automatically generated

Epoch = 500

A screen shot of a graph

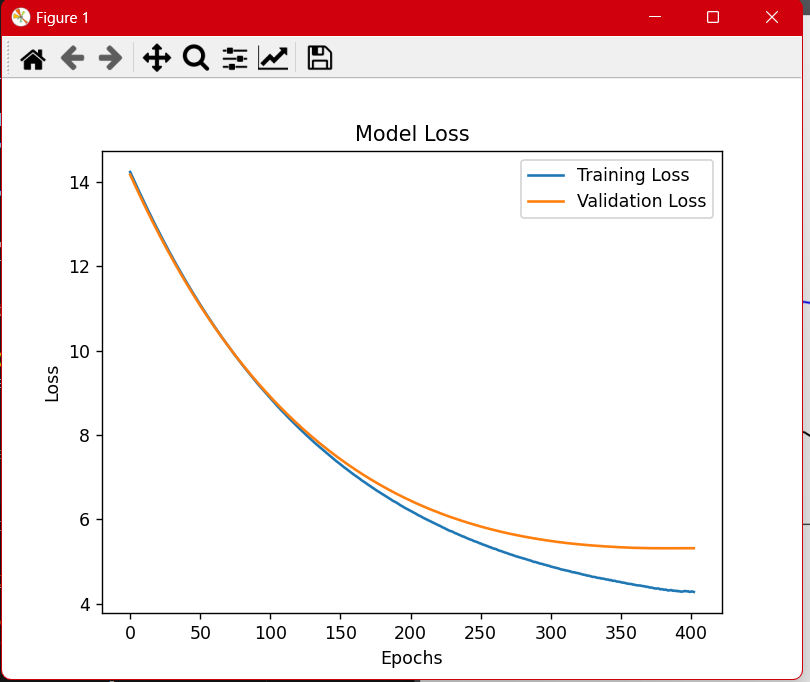
Description automatically generated

Epoch = 800

A screenshot of a computer

Description automatically generated

Epoch = 1,000



A graph on a computer screen

Description automatically generated

A screen shot of a graph

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a graph

Description automatically generated

A graph with a line

Description automatically generated

import json

import numpy as np

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping

from tensorflow.keras.regularizers import l2

import matplotlib.pyplot as plt

# Load data

def load\_data():

    with open('data/sentences\_data.json', 'r', encoding='utf-8') as f:

        data = json.load(f)

    texts = [d['thai'] for d in data['sentences']]

    labels = [d['id'] - 1 for d in data['sentences']]  # Adjust labels to start from 0

    return texts, labels

# Prepare data for training

def prepare\_data(texts, labels):

    tokenizer = Tokenizer()

    tokenizer.fit\_on\_texts(texts)

    sequences = tokenizer.texts\_to\_sequences(texts)

    word\_index = tokenizer.word\_index

    X = pad\_sequences(sequences, padding='post')

    y = np.array(labels)

    # Save the tokenizer

    tokenizer\_json = tokenizer.to\_json()

    with open('data/tokenizer.json', 'w', encoding='utf-8') as f:

        f.write(tokenizer\_json)

    return X, y, tokenizer

# Build the model

def build\_model(vocab\_size, max\_length, output\_dim):

    model = Sequential([

        Embedding(vocab\_size, 256, input\_length=max\_length),

        LSTM(256, return\_sequences=True, kernel\_regularizer='l2'),

        Dropout(0.5),

        LSTM(128, return\_sequences=True, kernel\_regularizer='l2'),

        Dropout(0.5),

        LSTM(64, kernel\_regularizer='l2'),

        Dense(128, activation='relu', kernel\_regularizer='l2'),

        Dropout(0.5),

        Dense(output\_dim, activation='softmax')

    ])

    optimizer = Adam(learning\_rate=0.0001)  # ลด learning rate

    model.compile(optimizer=optimizer, loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

    return model

# Check if the model predictions are correct

def check\_accuracy(model, X, y):

    predictions = model.predict(X)

    predicted\_labels = np.argmax(predictions, axis=1)

    true\_labels = y

    accuracy = np.mean(predicted\_labels == true\_labels)

    print(f"Predictions: {predicted\_labels[:5]}, True Labels: {true\_labels[:5]}")

    return accuracy

# Main training function

def train\_model():

    texts, labels = load\_data()

    X, y, tokenizer = prepare\_data(texts, labels)

    print("Sample X:", X[:5])

    print("Sample y:", y[:5])

    max\_length = X.shape[1]

    vocab\_size = len(tokenizer.word\_index) + 1

    output\_dim = len(set(labels))

    model = build\_model(vocab\_size, max\_length, output\_dim)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=55)

    # Callbacks

    reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=20, min\_lr=0.00001)

    model\_checkpoint = ModelCheckpoint('models/best\_video\_selection\_model.keras', monitor='val\_loss', save\_best\_only=True, mode='min')

    early\_stopping = EarlyStopping(monitor='val\_loss', patience=100, restore\_best\_weights=True)

    # Training with 3000 epochs and batch size 32

    history = model.fit(X\_train, y\_train, epochs=3000, batch\_size=32, validation\_data=(X\_test, y\_test), callbacks=[reduce\_lr, model\_checkpoint, early\_stopping])

    # Check accuracy after initial training

    current\_accuracy = check\_accuracy(model, X\_test, y\_test)

    print(f"Initial accuracy: {current\_accuracy}")

    # Save the final model

    model.save('models/video\_selection\_model\_final.keras')

    # Plotting the loss

    plt.plot(history.history['loss'], label='Training Loss')

    plt.plot(history.history['val\_loss'], label='Validation Loss')

    plt.title('Model Loss')

    plt.xlabel('Epochs')

    plt.ylabel('Loss')

    plt.legend()

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    train\_model()

A screen shot of a graph

Description automatically generated