**Language Translation with sequence Model**

**Python Code**

**# Step 1: Install TensorFlow (Colab usually has it)**

**!pip install -q tensorflow**

**# Step 2: Import Libraries**

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.models import Model**

**from tensorflow.keras.layers import Input, LSTM, Dense**

**# Step 3: Toy Dataset (English → German)**

**input\_texts = ['hello', 'how are you', 'good morning', 'what are you doing',**

**'good night', 'thank you', 'please', 'I am fine', 'see you later',**

**'what is your name?', 'where are you from?', 'I like it',**

**'I don’t understand', 'can you help me?' ]**

**target\_texts = ['\thallo\n', '\twie geht es dir\n', '\tguten morgen\n', '\tWas machst du?\n',**

**'\tgute Nacht\n', '\tdanke\n', '\tbitte\n', '\tmir geht es gut\n',**

**'\tschönen Tag noch\n', '\twie heißt du?\n', '\twoher kommst du?\n',**

**'\tich mag es\n', '\tich verstehe nicht\n', '\tkannst du mir helfen?\n']**

**# Preprocess Characters**

**input\_chars = sorted(set("".join(input\_texts)))**

**target\_chars = sorted(set("".join(target\_texts)))**

**num\_encoder\_tokens = len(input\_chars)**

**num\_decoder\_tokens = len(target\_chars)**

**max\_encoder\_seq\_length = max([len(txt) for txt in input\_texts])**

**max\_decoder\_seq\_length = max([len(txt) for txt in target\_texts])**

**input\_token\_index = dict([(char, i) for i, char in enumerate(input\_chars)])**

**target\_token\_index = dict([(char, i) for i, char in enumerate(target\_chars)])**

**reverse\_target\_char\_index = dict((i, char) for char, i in target\_token\_index.items())**

**# Vectorize Data**

**encoder\_input\_data = np.zeros((len(input\_texts), max\_encoder\_seq\_length, num\_encoder\_tokens), dtype='float32')**

**decoder\_input\_data = np.zeros((len(input\_texts), max\_decoder\_seq\_length, num\_decoder\_tokens), dtype='float32')**

**decoder\_target\_data = np.zeros((len(input\_texts), max\_decoder\_seq\_length, num\_decoder\_tokens), dtype='float32')**

**for i, (input\_text, target\_text) in enumerate(zip(input\_texts, target\_texts)):**

**for t, char in enumerate(input\_text):**

**encoder\_input\_data[i, t, input\_token\_index[char]] = 1.**

**for t, char in enumerate(target\_text):**

**decoder\_input\_data[i, t, target\_token\_index[char]] = 1.**

**if t > 0:**

**decoder\_target\_data[i, t - 1, target\_token\_index[char]] = 1.**

**# Step 4: Build Model**

**latent\_dim = 256**

**# Encoder**

**encoder\_inputs = Input(shape=(None, num\_encoder\_tokens))**

**encoder = LSTM(latent\_dim, return\_state=True)**

**encoder\_outputs, state\_h, state\_c = encoder(encoder\_inputs)**

**encoder\_states = [state\_h, state\_c]**

**# Decoder**

**decoder\_inputs = Input(shape=(None, num\_decoder\_tokens))**

**decoder\_lstm = LSTM(latent\_dim, return\_sequences=True, return\_state=True)**

**decoder\_outputs, \_, \_ = decoder\_lstm(decoder\_inputs, initial\_state=encoder\_states)**

**decoder\_dense = Dense(num\_decoder\_tokens, activation='softmax')**

**decoder\_outputs = decoder\_dense(decoder\_outputs)**

**# Full Model**

**model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)**

**model.compile(optimizer='rmsprop', loss='categorical\_crossentropy')**

**model.fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data, batch\_size=2, epochs=300)**

**# Step 5: Inference Models**

**# Encoder**

**encoder\_model = Model(encoder\_inputs, encoder\_states)**

**# Decoder**

**decoder\_state\_input\_h = Input(shape=(latent\_dim,))**

**decoder\_state\_input\_c = Input(shape=(latent\_dim,))**

**decoder\_states\_inputs = [decoder\_state\_input\_h, decoder\_state\_input\_c]**

**decoder\_outputs, state\_h, state\_c = decoder\_lstm(**

**decoder\_inputs, initial\_state=decoder\_states\_inputs)**

**decoder\_states = [state\_h, state\_c]**

**decoder\_outputs = decoder\_dense(decoder\_outputs)**

**decoder\_model = Model([decoder\_inputs] + decoder\_states\_inputs,**

**[decoder\_outputs] + decoder\_states)**

**# Step 6: Translation Function**

**def decode\_sequence(input\_seq):**

**states\_value = encoder\_model.predict(input\_seq)**

**target\_seq = np.zeros((1, 1, num\_decoder\_tokens))**

**target\_seq[0, 0, target\_token\_index['\t']] = 1.**

**stop\_condition = False**

**decoded\_sentence = ''**

**while not stop\_condition:**

**output\_tokens, h, c = decoder\_model.predict([target\_seq] + states\_value)**

**sampled\_token\_index = np.argmax(output\_tokens[0, -1, :])**

**sampled\_char = reverse\_target\_char\_index[sampled\_token\_index]**

**decoded\_sentence += sampled\_char**

**if sampled\_char == '\n' or len(decoded\_sentence) > max\_decoder\_seq\_length:**

**stop\_condition = True**

**target\_seq = np.zeros((1, 1, num\_decoder\_tokens))**

**target\_seq[0, 0, sampled\_token\_index] = 1.**

**states\_value = [h, c]**

**return decoded\_sentence.strip()**

**# Step 7: User Input**

**def translate\_user\_input():**

**while True:**

**input\_sentence = input("\nType an English sentence (or 'quit' to exit): ").lower()**

**if input\_sentence == 'quit':**

**break**

**input\_seq = np.zeros((1, max\_encoder\_seq\_length, num\_encoder\_tokens), dtype='float32')**

**for t, char in enumerate(input\_sentence):**

**if char in input\_token\_index:**

**input\_seq[0, t, input\_token\_index[char]] = 1.**

**translation = decode\_sequence(input\_seq)**

**print(f"→ German: {translation}")**

**# Start translation session**

**translate\_user\_input()**