**Makine Çevirisi Projesi: İngilizce-Türkçe Çeviri Sistemi**

Bu proje, bir sequence-to-sequence (seq2seq) modeli kullanarak İngilizce'den Türkçe'ye makine çevirisi yapan bir sistem geliştirmektedir. Proje TensorFlow ve Keras kütüphaneleri kullanılarak gerçekleştirilmiştir.

**Projenin Ana Bileşenleri**

**1. Gerekli Kütüphanelerin Yüklenmesi**

import tensorflow as tf

import numpy as np

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, GRU, Embedding

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.preprocessing.text import Tokenizer

Bu kısımda projede kullanılacak temel kütüphaneler ve modüller import edilmiştir. TensorFlow derin öğrenme için, numpy sayısal işlemler için, Keras ise model oluşturmak için kullanılmıştır.

**2. Veri Setinin Yüklenmesi ve Ön İşleme**

mark\_start = 'ssss ' *# decoder için başlangıç tokeni*

mark\_end = ' eeee' *# decoder için bitiş tokeni*

data\_src = [] *# İngilizce cümleler*

data\_dest = [] *# Türkçe cümleler*

for line in open('tur.txt', encoding='UTF-8'):

en\_text, tr\_text = line.rstrip().split('\t')

tr\_text = mark\_start + tr\_text + mark\_end

data\_src.append(en\_text)

data\_dest.append(tr\_text)

* tur.txt dosyasından İngilizce-Türkçe cümle çiftleri okunur.
* Türkçe cümlelerin başına ssss ve sonuna eeee tokenleri eklenir. Bunlar decoder'ın ne zaman başlayıp ne zaman duracağını bilmesi için kullanılır.
* Toplamda 473,035 cümle çifti yüklenmiştir.

**3. Tokenizer Sınıfı**

class TokenizerWrap(Tokenizer):

def \_\_init\_\_(self, texts, padding, reverse=False, num\_words=None):

*# Metinleri tokenlara çevirme ve kelime indeksi oluşturma*

*# Padding işlemi (cümleleri aynı uzunluğa getirme)*

*# Kelime indeksini ters çevirme (reverse=True ise)*

Bu özel sınıf, Keras'ın Tokenizer sınıfını genişleterek daha fazla işlevsellik ekler:

* Metinleri tokenlara çevirir
* Padding uygular (cümleleri aynı uzunluğa getirir)
* Kelime indeksi oluşturur
* Tokenlerden tekrar metin oluşturma yeteneği sağlar

**4. Tokenizer'ların Oluşturulması**

tokenizer\_src = TokenizerWrap(texts=data\_src, padding='pre', reverse=True, num\_words=None)

tokenizer\_dest = TokenizerWrap(texts=data\_dest, padding='post', reverse=False, num\_words=None)

* İngilizce (kaynak) cümleler için tokenizer: padding='pre' ve reverse=True
  + Cümleler ters çevrilir ve başa padding eklenir
* Türkçe (hedef) cümleler için tokenizer: padding='post' ve reverse=False
  + Cümleler normal sırada kalır ve sona padding eklenir

**5. Verilerin Hazırlanması**

encoder\_input\_data = tokens\_src

decoder\_input\_data = tokens\_dest[:, :-1] *# decoder girişi (başlangıç tokeni dahil)*

decoder\_output\_data = tokens\_dest[:, 1:] *# decoder çıkışı (bitiş tokeni dahil)*

* Encoder girişi: Ters çevrilmiş İngilizce cümleler
* Decoder girişi: Türkçe cümleler (son token hariç)
* Decoder çıkışı: Türkçe cümleler (ilk token hariç)

**6. Kelime Gömme (Word Embedding)**

word2vec = {}

with open('/content/drive/MyDrive/glove.6B.100d.txt', encoding='UTF-8') as f:

for line in f:

values = line.split()

word = values[0]

vec = np.asarray(values[1:], dtype='float32')

word2vec[word] = vec

embedding\_matrix = np.random.uniform(-1, 1, (num\_encoder\_words, embedding\_size))

for word, i in tokenizer\_src.word\_index.items():

if i < num\_encoder\_words:

embedding\_vector = word2vec.get(word)

if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

* GloVe önceden eğitilmiş kelime vektörleri yüklenir
* Her kelime için 100 boyutlu bir vektör oluşturulur
* Rastgele başlatılan bir embedding matrisi oluşturulur ve GloVe vektörleriyle doldurulur

**7. Model Mimarisi**

Model 3 ana bölümden oluşur:

**a) Encoder**

encoder\_input = Input(shape=(None,), name='encoder\_input')

encoder\_embedding = Embedding(input\_dim=num\_encoder\_words, output\_dim=embedding\_size)

encoder\_gru1 = GRU(state\_size, return\_sequences=True)

encoder\_gru2 = GRU(state\_size, return\_sequences=True)

encoder\_gru3 = GRU(state\_size, return\_sequences=False)

* Giriş katmanı
* Embedding katmanı (kelimeleri vektörlere dönüştürür)
* 3 GRU katmanı (her biri 256 nöronlu)
* Son GRU katmanı encoder'ın durum vektörünü üretir

**b) Decoder**

decoder\_input = Input(shape=(None,), name='decoder\_input')

decoder\_embedding = Embedding(input\_dim=num\_decoder\_words, output\_dim=embedding\_size)

decoder\_gru1 = GRU(state\_size, return\_sequences=True)

decoder\_gru2 = GRU(state\_size, return\_sequences=True)

decoder\_gru3 = GRU(state\_size, return\_sequences=True)

decoder\_dense = Dense(num\_decoder\_words, activation='linear')

* Giriş katmanı
* Embedding katmanı
* 3 GRU katmanı
* Çıkış katmanı (softmax yerine linear kullanılmış, çünkü loss fonksiyonu içinde softmax uygulanacak)

**c) Model Bağlantıları**

def connect\_encoder():

*# Encoder katmanlarını birbirine bağlar*

return encoder\_output

def connect\_decoder(initial\_state):

*# Decoder katmanlarını birbirine bağlar*

return decoder\_output

*# Eğitim modeli*

model\_train = Model(inputs=[encoder\_input, decoder\_input], outputs=[decoder\_output])

*# Encoder modeli (çıkarım için)*

model\_encoder = Model(inputs=[encoder\_input], outputs=[encoder\_output])

*# Decoder modeli (çıkarım için)*

model\_decoder = Model(inputs=[decoder\_input, decoder\_initial\_state], outputs=[decoder\_output])

**8. Model Eğitimi**

def sparse\_cross\_entropy(y\_true, y\_pred):

*# Sparse categorical cross-entropy loss hesaplar*

return tf.reduce\_mean(loss)

optimizer = RMSprop(learning\_rate=1e-3)

model\_train.compile(optimizer=optimizer, loss=sparse\_cross\_entropy)

model\_train.fit(

x=[encoder\_input\_data, decoder\_input\_data],

y=decoder\_output\_data,

batch\_size=64,

epochs=2,

callbacks=[checkpoint]

)

* Özel bir loss fonksiyonu tanımlanmıştır (sparse\_cross\_entropy)
* RMSprop optimizer kullanılmıştır
* Model 2 epoch boyunca eğitilmiştir (batch size=64)

**9. Çeviri Fonksiyonu**

def translate(input\_text, true\_output\_text=None):

*# 1) Girdi cümleyi tokenlara çevir*

*# 2) Encoder'dan state al*

*# 3) Decoder'ı token token çalıştırarak çeviri üret*

*# 4) Sonuçları yazdır*

Bu fonksiyon, eğitilmiş modeli kullanarak İngilizce cümleyi Türkçe'ye çevirir.

**10. Modelin Kaydedilmesi**

model\_train.save('seq2seq\_model.h5')

Eğitilen model HDF5 formatında kaydedilir.

**Projenin Çalışma Mantığı**

1. **Veri Hazırlama**: İngilizce-Türkçe cümle çiftleri yüklenir ve ön işlemden geçirilir.
2. **Tokenizasyon**: Cümleler tokenlara ayrılır ve sayısal indekslere dönüştürülür.
3. **Model Oluşturma**: Encoder-Decoder mimarisiyle bir seq2seq modeli oluşturulur.
4. **Eğitim**: Model veri üzerinde eğitilir.
5. **Çıkarım**: Eğitilen model yeni cümleleri çevirmek için kullanılır.

**1. Seq2Seq Nedir?**

* **Amaç:** Bir diziyi (örneğin İngilizce cümle) başka bir diziye (Türkçe cümle) dönüştürmek.
* **Örnek:**  
  *Girdi:* "I love NLP" → *Çıktı:* "NLP'yi seviyorum"

**2. Seq2Seq'in 3 Ana Bileşeni**

**A) Encoder (Kodlayıcı)**

* **Görevi:** Girdi cümlesini anlamlı bir özet vektörüne (*context vector*) dönüştürür.
* **Nasıl Çalışır?**
  1. Girdi cümlesi tokenlara ayrılır (kelimeler sayısal indekslere dönüşür).
  2. Her token bir **embedding katmanı** ile vektör temsiline çevrilir.
  3. **GRU katmanları** bu vektörleri işleyerek cümlenin anlamını özetler.
  4. Son GRU katmanının çıktısı (encoder\_output), decoder'ın başlangıç durumu olarak kullanılır.
* **Projedeki Kod:**

encoder\_gru1 = GRU(state\_size, return\_sequences=True)

encoder\_gru2 = GRU(state\_size, return\_sequences=True)

encoder\_gru3 = GRU(state\_size, return\_sequences=False) *# Son katman özet vektörü üretir*

**B) Context Vector (Bağlam Vektörü)**

* Encoder'ın ürettiği sabit uzunluklu özet (projede state\_size=256 boyutlu).
* **Önemli:** Tüm cümlenin anlamını içerir (bir nevi "sıkıştırılmış bilgi").

**C) Decoder (Kod Çözücü)**

* **Görevi:** Context vector'ü alıp hedef dili (Türkçe) token token üretir.
* **Nasıl Çalışır?**
  1. Başlangıçta ssss tokeni ile başlar.
  2. Her adımda bir sonraki tokeni tahmin eder (eeee görünce durur).
  3. **Teacher Forcing:** Eğitim sırasında bir önceki token yerine gerçek token kullanılır (daha stabil öğrenme için).
* **Projedeki Kod:**

decoder\_input\_data = tokens\_dest[:, :-1] *# "ssss NLP'yi seviyorum"*

decoder\_output\_data = tokens\_dest[:, 1:] *# "NLP'yi seviyorum eeee"*

**3. Projemdeki Adımlar**

**1. Veri Hazırlığı**

* **Örnek Veri:**  
  tur.txt dosyasından İngilizce-Türkçe cümle çiftleri okunur.
* **Token Ekleme:**  
  Türkçe cümlelerin başına ssss, sonuna eeee eklenir (decoder'ın başlangıç/bitiş sinyali).

**2. Tokenizasyon**

* **Tokenizer:** Her kelimeye bir sayı atanır (örneğin "the" → 5, "kedi" → 10).
* **Padding:** Tüm cümleler aynı uzunlukta olacak şekilde boşluklar doldurulur (encoder için padding='pre', decoder için padding='post').

**3. Embedding Katmanı**

* **GloVe Vektörleri:** Önceden eğitilmiş kelime vektörleri kullanılır (her kelime 100 boyutlu vektörle temsil edilir).
* **Amaç:** Benzer anlamlı kelimeler benzer vektörlere sahip olur (örneğin "king" ve "queen" birbirine yakın).

**4. Model Eğitimi**

* **Loss Fonksiyonu:** sparse\_cross\_entropy (her adımda doğru tokenin tahmin edilmesi).
* **Optimizer:** RMSprop (öğrenme oranı 1e-3).
* **Batch Size:** 64 (her seferinde 64 cümle işlenir).
* **Epochs:** 2 (tüm veri kümesi 2 kez taranır).

**5. Çıkarım (Inference)**

* **Adım Adım Çeviri:**
  1. Encoder, İngilizce cümleyi işler ve context vector üretir.
  2. Decoder, ssss tokeni ile başlar.
  3. Her adımda bir sonraki token tahmin edilir (eeee gelene kadar).
  4. Tokenler tekrar kelimelere dönüştürülür.

**4. Örnek Çalışma**

**Girdi:** "What do you know about ?"  
**Çıktı:** "ne hakkında biliyorsun eeee"

1. Encoder, İngilizce cümleyi işler ve 256 boyutlu vektör üretir.
2. Decoder, bu vektörü kullanarak:
   * Adım 1: ssss → "ne"
   * Adım 2: "ne" → "hakkında"
   * ...
   * Adım N: eeee görünce durur.

**5. Neden GRU Kullanıldı?**

* **LSTM'ye göre daha basit ve hızlı olduğu için.**
* **Uzun bağımlılıkları öğrenebilir** (örneğin cümlenin başındaki kelime ile sonundaki ilişki).