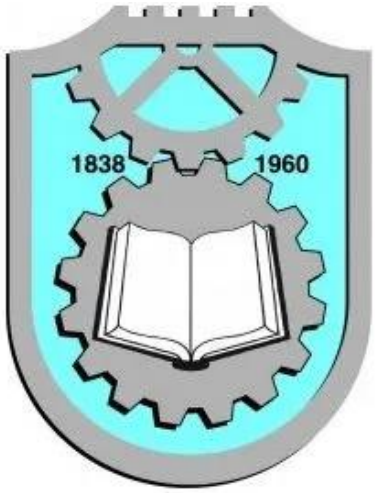
Основи дубоког учења – систем за превођење језика (са енглеског на српски)

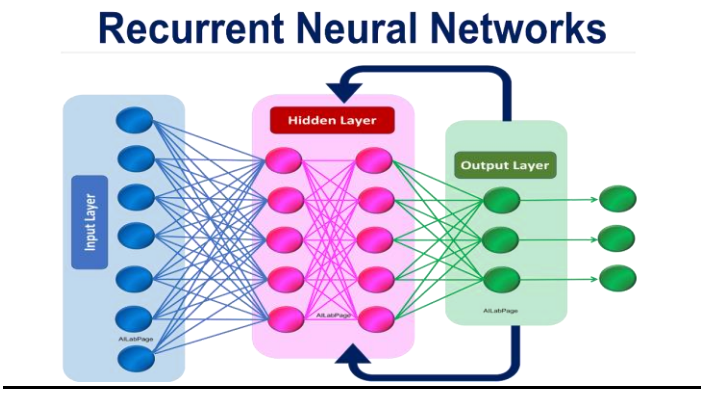


Универзитет у Крагујевцу, 2022.

Студент: Павле Зорић, 641/2019

1. Увод у систем за превођење

Основни циљ ове теме је да направимо систем за превођење са енглеског на српски језик. За ову тему нам је неопходан скуп података који ћемо да истренирамо коришћењем рекурентне неуронске мреже (**Recurrent Neural Network**) и учењем по секвенцама. Учење по секвенцама подразумева конвертовање секвенци из једног у други домен. Како бисмо превели са енглеског на српски, **Keras** библиотека игра значајну улогу у томе. Учитаћемо податке из .txt фајла **srp.txt** и исте те податке ћемо да истренирамо. Подаци су преузети са следећег линка: http://www.manythings.org/anki/.



2. Дефинисање параметара

Почетни параметри:
1. Величина скупа података
2. Број пута колико ћемо скуп података да истренирамо
3. Латентни простор
4. Број података унутар скупа за тренирање
5. .txt фајл у коме су складиштени подаци за тренирање

```
batch_size = 128  #velicina grupe podataka koju cemo da istreniramo
epochs = 100  #broj puta koliko cemo citav skup podataka da istreniramo
latent_dim = 256
num_samples = 10000  #broj podataka koje cemo da treniramo
#ulazni podaci
data_path = 'srp.txt'
```

3. Векторизација података

Да бисмо извршили векторизацију података које треба истренирати, потребно је прво да учитамо податке из srp.txt фајла. Затим, све податке учитане из .txt фајла конвертујемо у нумеричке векторе

```
# Vektorizacija podataka
input_texts = []
target_texts = []
input_characters = set()
target_characters = set()
with open(data_path, 'r', encoding='utf-8') as f:
    lines = f.read().split('\n')
```

4. Токенизација

Након што смо векторизовали податке, груписаћемо податке у више секвенци, при чему свака секвенца представља једну реч једне реченице коју треба да преведемо. Тиме ћемо да извршимо токенизацију, при чему једна секвенца представља један токен. Генерално, токенизација игра значајну улогу у раду са текстуалним подацима.

```
#Tokenizacija
for line in lines[: min(num_samples, len(lines) - 1)]:
    input_text, target_text, _ = line.split('\t')
    # Koristimo tab kao pocetnu sekvencu
    # i "\n" as krajnja sekvenca.
    target_text = '\t' + target_text + '\n'
    input_texts.append(input_text)
    target_texts.append(target_text)
    for char in input_text:
        if char not in input_characters:
            input_characters.add(char)
    for char in target_text:
        if char not in target_characters:
            target_characters.add(char)
```

5. Даљи поступак са улазним и излазним параметрима

Увозимо два нова параметара, енкодер и декодер. Енкодер представља низ података које треба истренирати и касније их превести, при чему резултат превођења се складишти у низ података којом управља декодер. Како бисмо могли лакше да енкодирамо и декодирамо податке који су секвенцијално груписани претходно токенизацијом, сваки токен (секвенцу) ћемо да индексирамо. Улазне низове енкодера и декодера постављамо на почетну вредност 0. Када по петљи пролазимо кроз енкодовани и декодовани низ, сваки токен који смо прошли кроз петљу га постављамо на 1. И током пролажења кроз петљу, декодер циљаних података је за један корак даљи од декодера улазних података.

```
input_token_index = dict(
    [(char, i) for i, char in enumerate(input_characters)])
target_token_index = dict(
    [(char, i) for i, char in enumerate(target_characters)])

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.
    encoder_input_data[i, t + 1:, input_token_index[' ']] = 1.
    for t, char in enumerate(target_text):
        # decoder_target_data je za jedan korak dalji od decoder_input_data
        decoder_input_data[i, t, target_token_index[char]] = 1.
        if t > 0:
            decoder_target_data[i, t - 1, target_token_index[char]] = 1.
    decoder_input_data[i, t + 1:, target_token_index[' ']] = 1.
    decoder_target_data[i, t:, target_token_index[' ']] = 1.
```

Затим је неопходно да сваку секвенцу истренирамо користећи LSTM (**Long short-term memory**) и да конвертујемо нумеричке излазне векторе декодера у векторе вероватноћа користећи **softmax** ф-ју. Након тога, истренираћемо модел и његове резултат тренирања ћемо да сачувамо.

```
# Definisanje ulazne sekvence (enkodera)
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]

# Definisanje ulazne sekvence (dekodera)
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)

#Definisanje modela koji pretvara ulazne podate enkodera i dekodera u ciljane podatke dekodera
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

#Treniranje modela
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)
#Cuvanje modela
model.save('s2s.h5')
```

Тај модел који смо сачували након тренирања, ћемо да искористимо како бисмо поново дефинисали тренутна стања енкодера и декодера.

```
#definisanje stanja enkodera i dekodera nakon treniranja modela
encoder_model = Model(encoder_inputs, encoder_states)

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)

reverse_input_char_index = dict(
    (i, char) for char, i in input_token_index.items())
reverse_target_char_index = dict(
    (i, char) for char, i in target_token_index.items())
```

6. Финална обрада података

```
def decode_sequence(input_seq):
    #Enkodiranje ulaza
    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)

        #treniranje tokena
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char

        #nalazenje poslednjeg karaktera cime se petlja prekida
        if (sampled_char == '\n' or
           len(decoded_sentence) > max_decoder_seq_length):
            stop_condition = True

        #apdejtovanje ciljanih sekvenci i njihovo postavljanje na 1
        target_seq = np.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.

        #apdejtovanje stanja
        states_value = [h, c]

    return decoded_sentence

for seq_index in range(50):
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Decoded sentence:', decoded_sentence)
```

Прво енкодујемо улазну секвенцу векторским путем. Циљану секвенцу постављамо на вредност 1. Да бисмо предвидели вероватноћу за наредни карактер, укључујемо улазну секвенцу и векторе стања које смо дефинисали на почетку. За предвиђање вероватноће наредног карактера, користимо фју **argmax()**. Након што смо предвидели вероватноћу карактера, додајемо га у низ којом располаже циљана секвенца и њему додељујемо вредност 1.