Міністерство освіти і науки

України Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

ЗВІТ

лабораторної роботи №8

з курсу «Програмні засоби проєктування і реалізації неромережевих систем»

Тема: «Рекурентні нейронні мережі LSTM»

Перевірив:

Шимкович В. М.

Виконав:

Студент Гр. ІП-01 Шпилька В.С.

Київ 2023

Завдання: Написати програму, що реалізує нейронну мережу типу CNN-bi-LSTM для розпізнавання мови в текст.

1. Реалізація нейронної мережі:

```
def Speech2Text(input_dim, output_dim, rnn_layers=5, rnn_units=128):
    input = Input((None, input_dim))
    x = Reshape((-1, input_dim, 1))(input)
    x = Conv2D(
        filters=32,
        kernel_size=[11, 41],
        strides=[2, 2],
        padding="same",
        use_bias=False
    )(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = Conv2D(
        filters=32,
        kernel_size=[11, 21],
        strides=[1, 2],
        padding="same",
        use_bias=False
    )(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = Reshape((-1, x.shape[-2] * x.shape[-1]))(x)
    for i in range(1, rnn_layers + 1):
        recurrent = GRU(
            units=rnn_units,
            recurrent_activation="sigmoid",
            return_sequences=True,
            reset_after=True
        )
        x = Bidirectional(
            recurrent, merge_mode="concat"
        )(x)
        if i < rnn_layers:
            x = Dropout(rate=0.5)(x)
    x = Dense(units=rnn_units * 2)(x)
    x = ReLU()(x)
    x = Dropout(rate=0.5)(x)
    output = Dense(units=output_dim + 1, activation="softmax")(x)
    model = Model(input, output)
    return model
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, 193)]       0

reshape (Reshape)            (None, None, 193, 1)      0

conv2d (Conv2D)              (None, None, 97, 32)      14432

batch_normalization (BatchN  (None, None, 97, 32)     128
ormalization)

re_lu (ReLU)                 (None, None, 97, 32)      0

conv2d_1 (Conv2D)            (None, None, 49, 32)      236544

batch_normalization_1 (Batc  (None, None, 49, 32)     128
hNormalization)

re_lu_1 (ReLU)               (None, None, 49, 32)      0

reshape_1 (Reshape)          (None, None, 1568)        0

bidirectional (Bidirectiona  (None, None, 1024)       6395904
l)

dropout (Dropout)            (None, None, 1024)        0

bidirectional_1 (Bidirectio  (None, None, 1024)       4724736
nal)

dropout_1 (Dropout)          (None, None, 1024)        0

bidirectional_2 (Bidirectio  (None, None, 1024)       4724736
nal)

dropout_2 (Dropout)          (None, None, 1024)        0

bidirectional_3 (Bidirectio  (None, None, 1024)       4724736
nal)

dropout_3 (Dropout)          (None, None, 1024)        0

bidirectional_4 (Bidirectio  (None, None, 1024)       4724736
nal)

dense (Dense)                (None, None, 1024)        1049600

re_lu_2 (ReLU)               (None, None, 1024)        0

dropout_4 (Dropout)          (None, None, 1024)        0

dense_1 (Dense)              (None, None, 32)          32800

=================================================================
Total params: 26,628,480
Trainable params: 26,628,352
Non-trainable params: 128
_________________________________________________________________
```

Нейрона мережа складається з 2 згорткових шарів та 5 рекурентних. Це дозволяє на першому етапі виділити корисні риси зі спектрограми, а потім за допомогою рекурентних шарів розпізнати звук на конкретному кроці.

Як функція втрат було реалізовано CTCLoss:

```
import tensorflow as tf

def CTCLoss(y_true, y_pred):
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

    loss = tf.keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
    return loss
```

2. Створення пайплану для завантаження даних.

Для тренування було використано LJSpeech dataset. Для цього було створено tf dataset з шляхів до файлів та їх транскрипцій.

Далі використовується наступна функція препроцесінгу:

а.) Читається файл

б.) Трансформація до потрібного формату

в.) Створення спектрограми

г.) Кодування символів в числа

```python
def encode_single_sample(self, wav_file, label):
    #read and decode audio
    file = tf.io.read_file(self.wavs_path + wav_file + ".wav")
    audio, _ = tf.audio.decode_wav(file)
    audio = tf.squeeze(audio, axis=-1)
    audio = tf.cast(audio, tf.float32)

    #create and normalize spectrogram
    spectrogram = tf.signal.stft(
        audio, frame_length=settings.FRAME_LENGTH, frame_step=settings.FRAME_STEP, fft_length=settings.FFT_LENGTH
    )
    spectrogram = tf.abs(spectrogram)
    spectrogram = tf.math.pow(spectrogram, 0.5)
    means = tf.math.reduce_mean(spectrogram, 1, keepdims=True)
    stddevs = tf.math.reduce_std(spectrogram, 1, keepdims=True)
    spectrogram = (spectrogram - means) / (stddevs + 1e-10)

    #process labels
    label = tf.strings.lower(label)
    label = tf.strings.unicode_split(label, input_encoding="UTF-8")
    label = self.char_to_num(label)

    return spectrogram, label
```

```python
def create_data_pipeline(self, ds, preprocessor):
    ds = (
        ds.map(preprocessor.encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE)
        .padded_batch(self.batch_size)
        .prefetch(buffer_size=tf.data.AUTOTUNE)
    )
    return ds

def create_data_pipelines(self, preprocessor):
    train_ds = self.create_data_pipeline(self.train_ds, preprocessor)
    val_ds = self.create_data_pipeline(self.val_ds, preprocessor)
    test_ds = self.create_data_pipeline(self.test_ds, preprocessor)
    return (train_ds, val_ds, test_ds)
```

3. Постпроцесінг:

Оскільки в результаті виконання ми отримаємо массив ймовірностей послідовності, його треба перетворити назад до тексту. Для цього використовується ctc decoding і його варіант – жадібний.

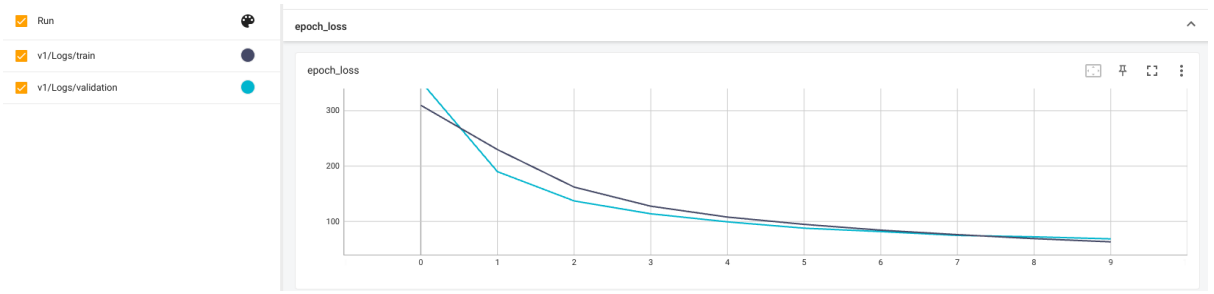
```python
def decode_batch_predictions(self, pred):
    input_len = np.ones(pred.shape[0]) * pred.shape[1]
    results = tf.keras.backend.ctc_decode(pred, input_length=input_len, greedy=True)[0][0]
    output_text = []
    for result in results:
        result = tf.strings.reduce_join(self.num_to_char(result)).numpy().decode("utf-8")
        output_text.append(result)
    return output_text
```

Також було реалізовано простий алгоритм виправлення слів. Для цього парситься деякий великий файл у окремі слова і підраховується кількість кожного слова.

Далі функція приймає на вхід слово, якщо його немає в словнику, то вона генерує всі можливо варіанти даного слова з 2 змінами і шукає в словнику і повертає найімовірніше слово.

```
class SpellCorrection:
    def __init__(self, text_correction_file: str, charlist: str) -> None:
        """ Simple spell correction based on statistical methods """
        self.all_words = Counter(self.words(open(text_correction_file).read()))
        self.charlist = charlist

    def words(self, text: str) -> list:
        """ Preprocess words from file """
        return re.findall(r'\w+', text.lower())

    def P(self, word: str) -> float:
        """ Probability of `word`."""
        N = sum(self.all_words.values())
        return self.all_words[word] / N

    def candidates(self, word: str) -> list:
        """ Generate possible spelling corrections for word."""
        return (self.known([word]) or self.known(self.edits1(word)) or self.known(self.edits2(word)) or [word])

    def known(self, words: set) -> set:
        """ The subset of `words` that appear in the dictionary of WORDS."""
        return set(w for w in words if w in self.all_words)

    def edits1(self, word: str) -> set:
        """ All edits that are one edit away from `word`."""
        letters    = self.charlist
        splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
        deletes    = [L + R[1:]               for L, R in splits if R]
        transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
        replaces   = [L + c + R[1:]           for L, R in splits if R for c in letters]
        inserts    = [L + c + R               for L, R in splits for c in letters]
        return set(deletes + transposes + replaces + inserts)

    def edits2(self, word: str) -> set:
        """ All edits that are two edits away from `word`. """
        return (e2 for e1 in self.edits1(word) for e2 in self.edits1(e1))

    def correction(self, word: str) -> str:
        """ Most probable spelling correction for word."""
        isUpper = word[0].isupper()
        word = word.lower()
        corrected_word =  max(self.candidates(word), key=self.P)
        if(isUpper):
            corrected_word = corrected_word[0].upper() + corrected_word[1:]
        return corrected_word
```

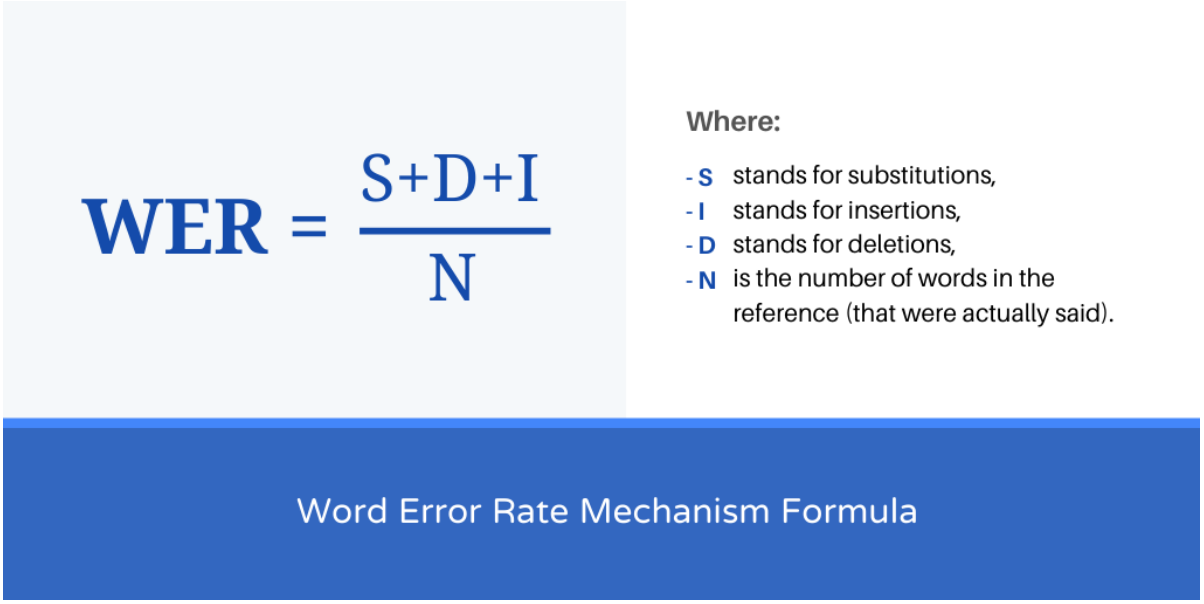
4. Тренування мережі:



Тренування проводилось на 10 епохах.

5. Тестування:

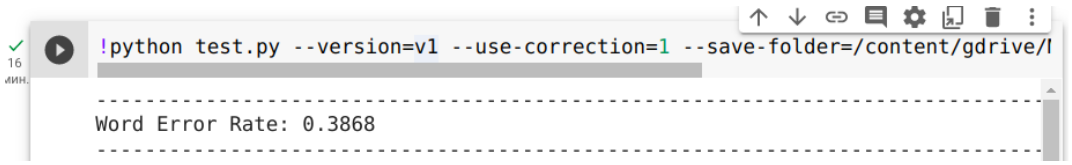
Для тестування було використано метрику wer.



Було проведено тестування без використання виправлення слів:

```
[10] !python test.py --version=v1 --use-correction=0 --save-folder=/content/gdrive/MyD
----------------------------------------------------------------------------------------
Word Error Rate: 0.4819
----------------------------------------------------------------------------------------
```

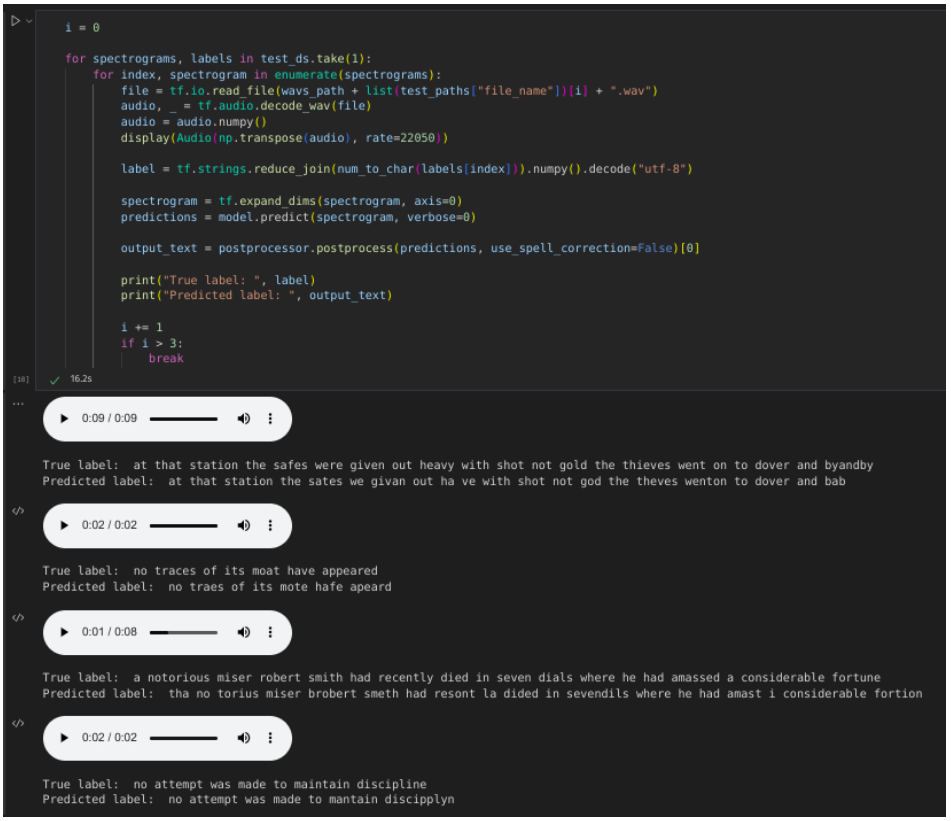
та за допомогою нього:



в результаті отримали покращення в 10 відсотків і фінальну помилку в 0.38. Тобто даний результат можна інтерпретувати як точність в 62 відсотки.

Варто тільки зауважити, що тепер працює тестування на 12 хвилин довше, але дану проблему можна покращити, якщо застосувати паралельний постпроцесінг для результату

Приклад використання:



Висновок: В результаті виконання лабораторної роботи було побудовано нейрону мережу CNN-bi-LSTM для розпізнавання мови в текст. Всього нейрона мережа має 26млн параметрів. Для LJSpeech датасету дана нейрона мережа показала гарні результати, а саме точність в 62 відсотки.