

بسم الله الرحمن الرحيم

پروژه پردازش متن

برای محصولات دیجیکالا

نسخه شماره ۵

تیم ارائه دهنده: داده کاوان فونیکس

ناظر و سرپرست تیم: دکتر علیرضا وفایی صدر

درخواست دهنده: ستاد علوم شناختی IPM

بهار ۱۳۹۹

فهرست مطالب

۲	-----	تعریف پروژه
۲	-----	تولید جملات معنادار از جملات فیک و بدون ساختار
۱۳	-----	جمع بندی

تعريف پروژه

در ادامه فاز دوم و بحث های صورت گرفته، به ادامه کار تولید متن با لایک های بالا پرداختیم.

ولی جملات تولید شده جملاتی بدون گرامر و بی معنی بودند که فقط شامل دنباله ای از لغات هستند.

در این پروژه با استفاده از مدل ترجمه ماشینی Machine Translation کامنت های بی معنی را سعی کردیم معنادار کنیم.

تولید جملات معنادار از جملات فیک و بدون ساختار

برای اینکه مدلی را بسازیم که از کامنت های بی معنا و بدون ساختار به متن هایی دارای مفعوم و معنا بررسد، ابتدا ترتیب کلمات در هر کامنت از دیتاست را به هم ریختیم تا آنرا به مدل داده و مدل کامنت های بی ساختار و به هم ریخته را بشناسد.

برای اینکار هر کامنت را شافل shuffle کرده و جملات هر کامنت ها را به هم ریختیم!

کامنت های به هم ریخته را به عنوان داده های آموزشی و کامنت های اولیه و دارای معنا را هم به عنوان ستون تارگت و هدف در نظر گرفتیم.

داده ها را به دو دسته داده های تست و ولیدیشن تقسیم کرده و کار tokenizer و تبدیل هر کلمه به عدد متناظر را هم برای داده های شافل شده و هم برای داده های منظم در ستون هدف انجام دادیم.

قرار است مدلی را تولید کنیم که با مشاهده هر کامنت شافل شده و به هم ریخته، بتواند به کامنتی منظم و دارای مفهوم دست پیدا کند. برای این روش از مدل ترجمه ماشینی استفاده کردیم که دارای معماری زیر می باشد:

- یک Encoder با یک لایه امبدینگ و یک لایه شبکه بازگشتی GRU
- یک لایه Attention
- یک Decoder با یک لایه امبدینگ و یک لایه شبکه بازگشتی GRU و یک لایه Dense

در ابتدا جملات به هم ریخته به عنوان ورودی به encoder داده شده با State های اولیه، و خروجی آن و های خروجی از encoder را از لایه Attention عبور می دهیم و خروجی آن را به عنوان ورودی state Decoder در نظر می گیریم.

در هر time-step ، خروجی تولید شده در decoder به عنوان ورودی در time-step بعدی به دیگر ورودی های decoder که از لایه Attention میاید نیز اضافه خواهد شد تا هر کلمه تولید شده در جمله خروجی به کلمه قبلی نیز وابسته باشد تا جمله معنا و مفهوم پیدا کند. (اساس کار در ترجمه ماشینی اینگونه است که تولید هر کلمه در جمله به کلمه قبل نیز بستگی دارد).

در تمام مدل های ترجمه ماشینی ما زبانی را به زبان دیگری ترجمه می کنیم (برای مثال ترجمه از زبان فارسی به انگلیسی) اما در این مورد ما زبان فارسی را به فارسی ترجمه می کنیم اما از جملات فارسی بدون معنا به جملات فارسی معنادار و نزدیک به زبان مبدا.

کد encoder را در شکل زیر می بینید:



```
#set Hyper Parameters
BATCH_SIZE = 64
embedding_dim = 256
units = 1024
vocab_inp_size = 53019
vocab_tar_size = 53019

#define Encoder
class Encoder(Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = Embedding(vocab_size, embedding_dim)
        self.gru = GRU(self.enc_units,
                      return_sequences=True,
                      return_state=True,
                      recurrent_initializer='glorot_uniform')

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))

encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)

# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
```

کد لایه Attention نیز در شکل زیر دید می شود:



```
#define Attention Layer
class BahdanauAttention(Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = Dense(units)
        self.W2 = Dense(units)
        self.V = Dense(1)

    def call(self, query, values):
        # hidden shape == (batch_size, hidden size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden size)
        # we are doing this to perform addition to calculate the score
        hidden_with_time_axis = tf.expand_dims(query, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying score to self.V
        # the shape of the tensor before applying self.V is (batch_size, max_length, units)
        score = self.V(tf.nn.tanh(
            self.W1(values) + self.W2(hidden_with_time_axis)))

        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

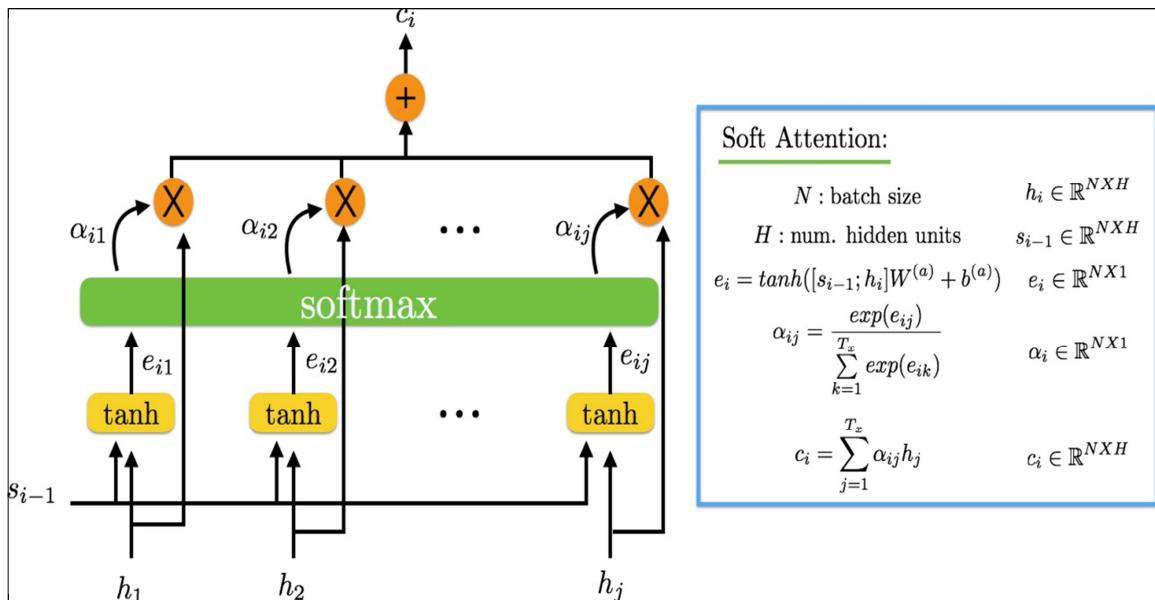
        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights

attention_layer = BahdanauAttention(10)
attention_result, attention_weights = attention_layer(sample_hidden, sample_output)
```

خروجی لایه attention state های آن وارد لایه encoder به همراه وزن های اتنشن و خروجی لایه اتنشن را تشکیل می دهد.

شکل زیر نمایی از لایه attention را نشان می دهد:



کد به صورت زیر است:

```
#define decoder
class Decoder(Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = Embedding(vocab_size, embedding_dim)
        self.gru = GRU(self.dec_units,
                      return_sequences=True,
                      return_state=True,
                      recurrent_initializer='glorot_uniform')
        self.fc = Dense(vocab_size)

        # used for attention
        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)
        context_vector, attention_weights = self.attention(hidden, enc_output)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

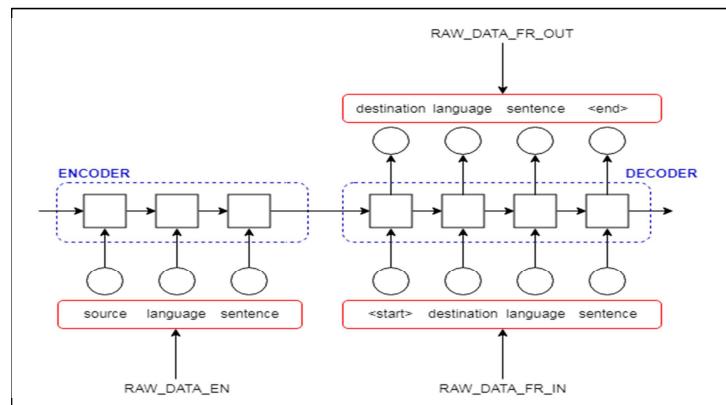
        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # output shape == (batch_size, vocab)
        x = self.fc(output)

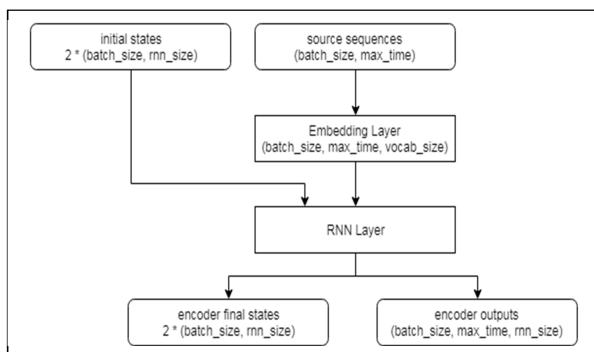
        return x, state, attention_weights

decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)
sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
                                     sample_hidden, sample_output)
```

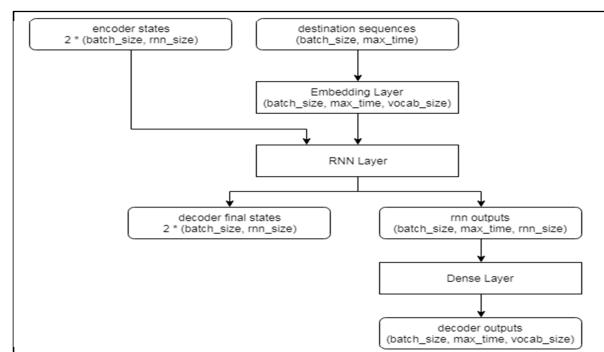
شماتیکی کلی Encoder-Decoder به صورت زیر است:



شماتیکی کلی Decoder و Encoder بدون لایه Attention را در تصاویر زیر مشاهده می کنید:

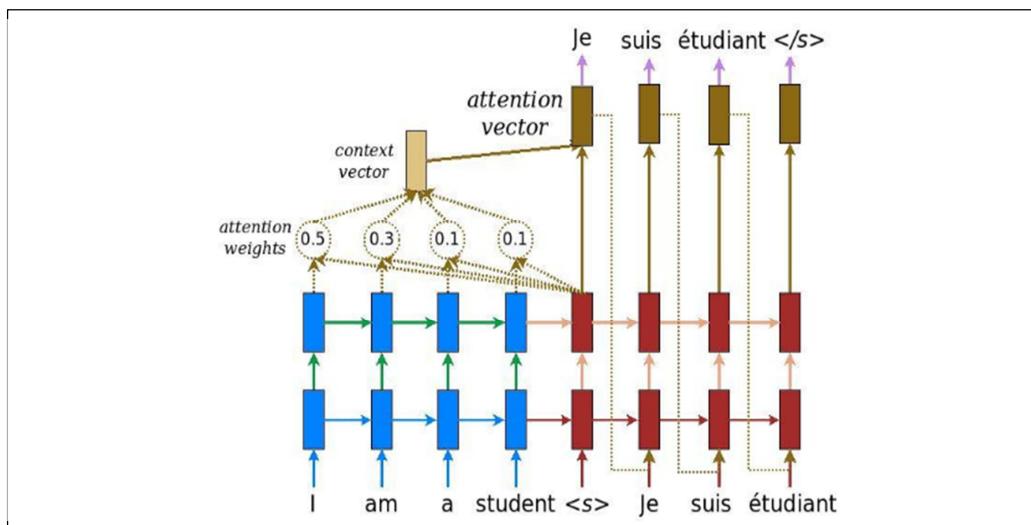


Encoder Schema



Decoder Schema

شماتیکی کلی Attention encoder-decoder برای ترجمه ماشینی:



تابع loss را به صورت زیر نوشته ایم:

```
#Define the optimizer and the loss function
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_*= mask

    return tf.reduce_mean(loss_)
```

مدل را به صورت زیر آموزش دادیم و ترین کردیم:

```
def train_step(inp, targ, enc_hidden):
    loss = 0

    with tf.GradientTape() as tape:
        enc_output, enc_hidden = encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([tokenizer_y.word_index['start']] * BATCH_SIZE, 1)

        # Teacher forcing - feeding the target as the next input
        for t in range(1, targ.shape[1]):
            # passing enc_output to the decoder
            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output)

            loss += loss_function(targ[:, t], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(targ[:, t], 1)

    batch_loss = (loss / int(targ.shape[1]))

    variables = encoder.trainable_variables + decoder.trainable_variables
    gradients = tape.gradient(loss, variables)
    optimizer.apply_gradients(zip(gradients, variables))

    return batch_loss
```

برای evaluate کردن مدل، کد زیر به کار رفته است:

```
max_length_targ = 30
max_length_inp = 30

def evaluate(sentence):
    attention_plot = np.zeros((max_length_targ, max_length_inp))

    sentence = add_start_end(sentence)

    inputs = [tokenizer_X.word_index[i] for i in sentence.split(' ')]
    inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
                                                           maxlen=max_length_inp,
                                                           padding='post')
    inputs = tf.convert_to_tensor(inputs)

    result = ''

    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)

    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([tokenizer_y.word_index['start']], 0)

    for t in range(max_length_targ):
        predictions, dec_hidden, attention_weights = decoder(dec_input,
                                                               dec_hidden,
                                                               enc_out)

        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))
        attention_plot[t] = attention_weights.numpy()

        predicted_id = tf.argmax(predictions[0]).numpy()

        result += tokenizer_y.index_word[predicted_id] + ' '

        if tokenizer_y.index_word[predicted_id] == 'end':
            return result, sentence, attention_plot

        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)

    return result, sentence, attention_plot
```

در evaluate کردن، علاوه بر اینکه جملات بی معنا را به مدل داده و خروجی جملات با مفهوم تولید می کنیم، بلکه نمودار attention بین هر دو جمله را نیز به دست می آوریم.

کد تولید نمودار Attention بین جمله ورودی و جمله خروجی تولید شده از مدل :

```
# function for plotting the attention weights
def plot_attention(attention, sentence, predicted_sentence):
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(1, 1, 1)
    ax.matshow(attention, cmap='viridis')

    fontdict = {'fontsize': 14}

    ax.set_xticklabels([''] + sentence, fontdict=fontdict, rotation=90)
    ax.set_yticklabels([''] + predicted_sentence, fontdict=fontdict)

    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    plt.show()
```

در محور X جمله ورودی (کامنت فیک و بی معنا) و در محور Y جمله خروجی (کامنت معنادار) قرار می گیرد.

بعد از انجام این کارها کافیست کدی نوشته که هر بار جمله ای را برای تست به آن داده و جملع معادل ترجمه شده را که قرار است کامنتی با معنا باشد به همراه نمودار attention مربوط به آن ها را تحويل بگیریم. کد زیر همین کار را میکند:

```
def translate(sentence):
    result, sentence, attention_plot = evaluate(sentence)

    print('Input: %s' % (sentence))
    print('Predicted translation: {}'.format(result))
```

برای تولید کامنت های با معنا از کامنت های فیک به صورت زیر عمل می کنیم:

'کامنت فیک و بی معنا'(translate)

در زیر نتایج چند جمله از کامنت های فیک تولید شده و خروجی آن ها را که کامنت هایی با مفهوم و دارای گرامر است و توسط مدل مورد نظر به دست آمده را مشاهده می کنید:

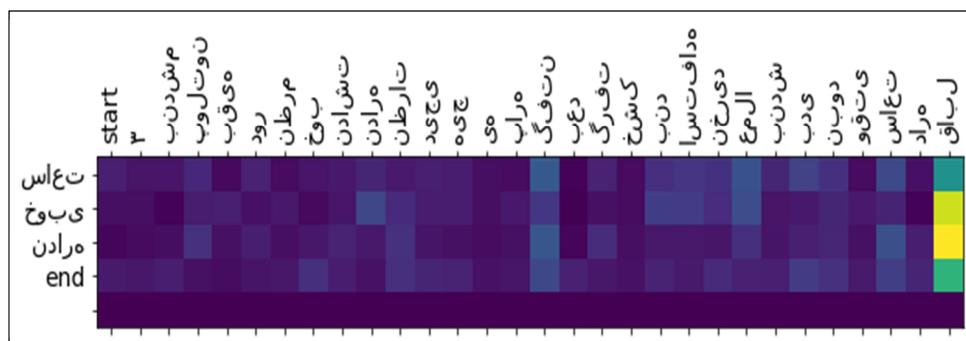
جمله ورودی فیک و بی معنا:

"بندشم پولتون بقیه دور نظرم خوب نداشت نداره نظرات دیجی هیچ یه پاره گفتن بعد گرفت خشک بند استفاده نخاید عملا بندش بدی نبود وقتی ساعت داره قابل سنسوراژ حتی پاره جا ظاهر خوبی توجه ماه دیگه بزرگی یدکی ولی میریزید میشه"^۳

جمله خروجی و معنا دار:

"ساعت خوبی، نداره"

نومدا، Attention :



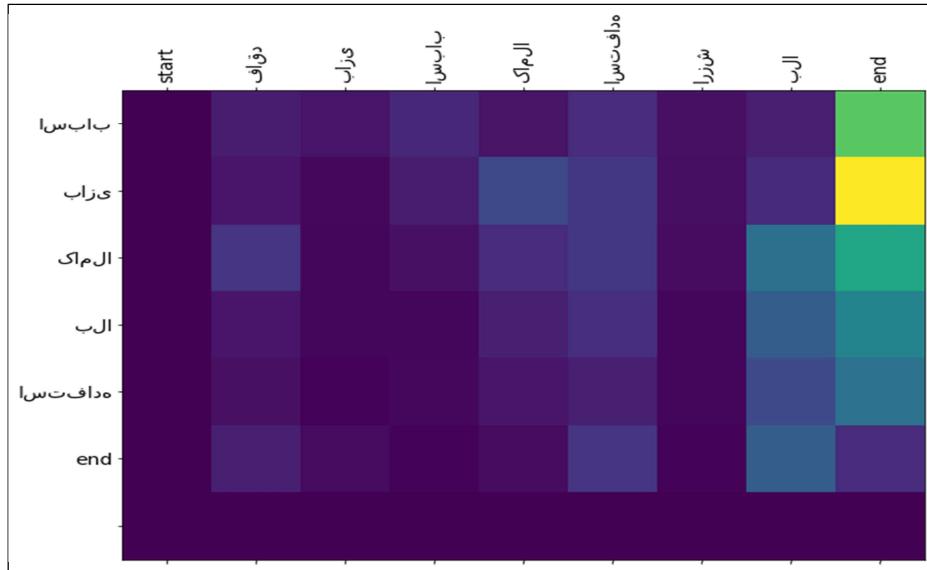
جمله و رودی فک و بے معنا:

"فاقد بازی، اسیاب کاملا استفاده ارزش، نیلا"

حمله خووح، و معنا دار:

"اسباب بازی، کاملاً بلا استفاده"

نمودار : Attention



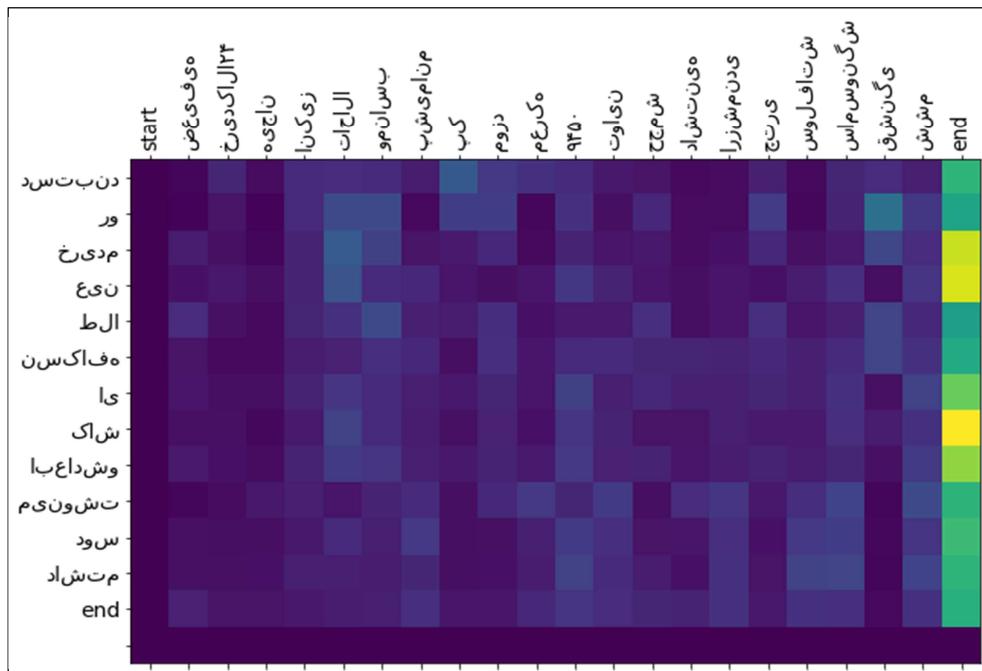
جمله ورودی فیک و بی معنا:

" ضعیفیه خرید کالا ۲۴ هیجان انکیز تاحالا ومناسب پشیمانم پک موزد معركه ۹۴۵۰ توain حجمش داشتنیه ارزشمندی چتری سولفاتش سامسونگش قشنگی ششم "

جمله خروجی و معنا دار:

" دستبند رو خریدم عین طلا نسکافه ای کاش ابعادشو مینوشت دوس داشتم "

نمودار : Attention



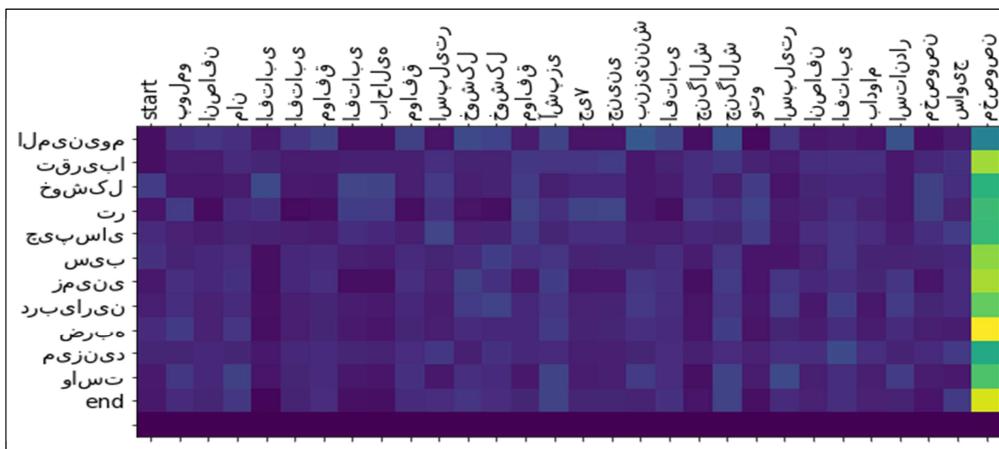
جمله ورودی فیک و بی معنا:

"پولمو انصافن مان افتایی موافق افتایی باحالیه موافق اسپلیت خوشکل خوشکل موافق آشپزی جی ۷ چنینی بنزیننش افتایی چنگالش چنگالش و تو اسپلیت انصافن افتایی بادوام استاندار مخصوصن ساویج مخصوصن و تو خوشکل تفلون"

جمله خروجی و معنا دار:

"المینیوم تقریبا خوشکل تر چیپسای سیب زمینی دربیارین ضربه میزند و است"

: Attention نمودار



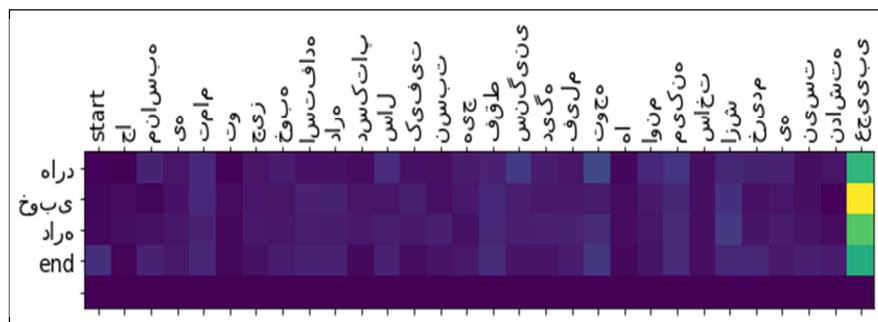
جمله ورودی فیک و بی معنا:

"جا مناسبه یه تمام تو چیز خوبه استفاده داره دسکتاب سال کیفیت نسبت هیچ فقط سنگینی دیگه فیلم توجه ها اونم میکنه ساخت ازش خریدم یه نیست نداشته عجیبی دسکتاب اذیت رو زیاده نزدیک ازش آداتور ساله اینکه انتقال جایی میکنم کرده هارد ایرادی عنوان رو هارد مقدار عکس هارد خوبی هست مشکل سخت نمونه وزن سرعت آپ های استفاده اطلاعاتش ها نداشتمن بک هارد یه اطلاعات"

جمله خروجی و معنا دار:

"هارد خوبی داره"

: Attention نمودار



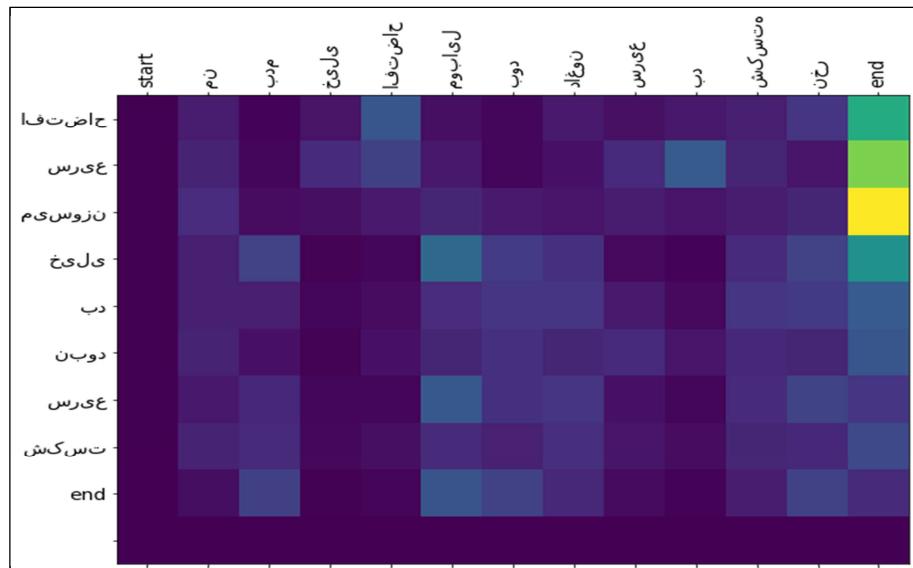
جمله ورودی فیک و بی معنا:

"من بدم خیلی افتضاح موبایل بود داغون سریع بد شکسته نخر"

جمله خروجی و معنا دار:

"افتضاح سریع میسوزن خیلی بد نبود سریع شکست"

: Attention نمودار



جمع بندی

در این گزارش، سعی در تبدیل کامنت های فیک و بدون گرامر به کامنت هایی معنادار داشتیم و از مدلی ترجمه ماشینی attention با لایه encoder-decoder برای این منظور استفاده کردیم.

قابل ذکر است که به دلیل تابع سنگین Train و تولید checkpoint هایی برای ذخیره داده های loss در هر بار ترین شدن مدل، به دلیل کمبود RAM ما مجبور شدیم فقط دو epoch را train کنیم که نتایج آن به صورت زیر است:

Epoch 1 Batch 0 Loss 5.5865

Epoch 1 Batch 200 Loss 4.7869

Epoch 1 Batch 400 Loss 4.1380

Epoch 1 Batch 600 Loss 4.0896

Epoch 1 Batch 800 Loss 3.6118

Epoch 1 Batch 1000 Loss 3.9128

Epoch 1 Batch 1200 Loss 3.7483

Epoch 1 Batch 1400 Loss 3.8761

Epoch 1 Loss 3.9587

Time taken for 1 epoch 1238.7204747200012 sec

Epoch 2 Batch 0 Loss 3.6468

Epoch 2 Batch 200 Loss 3.0730

Epoch 2 Batch 400 Loss 3.1823

Epoch 2 Batch 600 Loss 2.9107

Epoch 2 Batch 800 Loss 2.9296

Epoch 2 Batch 1000 Loss 2.7577

Epoch 2 Batch 1200 Loss 2.6338

Epoch 2 Batch 1400 Loss 2.3632

Epoch 2 Loss 3.0660

Time taken for 1 epoch 1246.1316845417023 sec

امید داریم با سرورهای بهتر و با تعداد epoch های بیشتر بتوان کامنت های با معناتر و شبیه تر به کامنت های ورودی تولید کرد. مدت زمان اموزش در هر epoch با RAM TPU در Google Colab به دست آمده است.

کلیه فایل های این گزارش توسط [آرمیتا رضوی](#) در مسیر Report2-> Armita Razavi-> NMT قابل مشاهده است.

کلیه فایل های مربوط به این گزارش نسخه ۵ در گیت هاب زیر و در فolder Report2 موجود است.

<https://github.com/phoenix-dataminers/Digikala2>