Chatbot con TensorFlow 2.0

Acecom IA

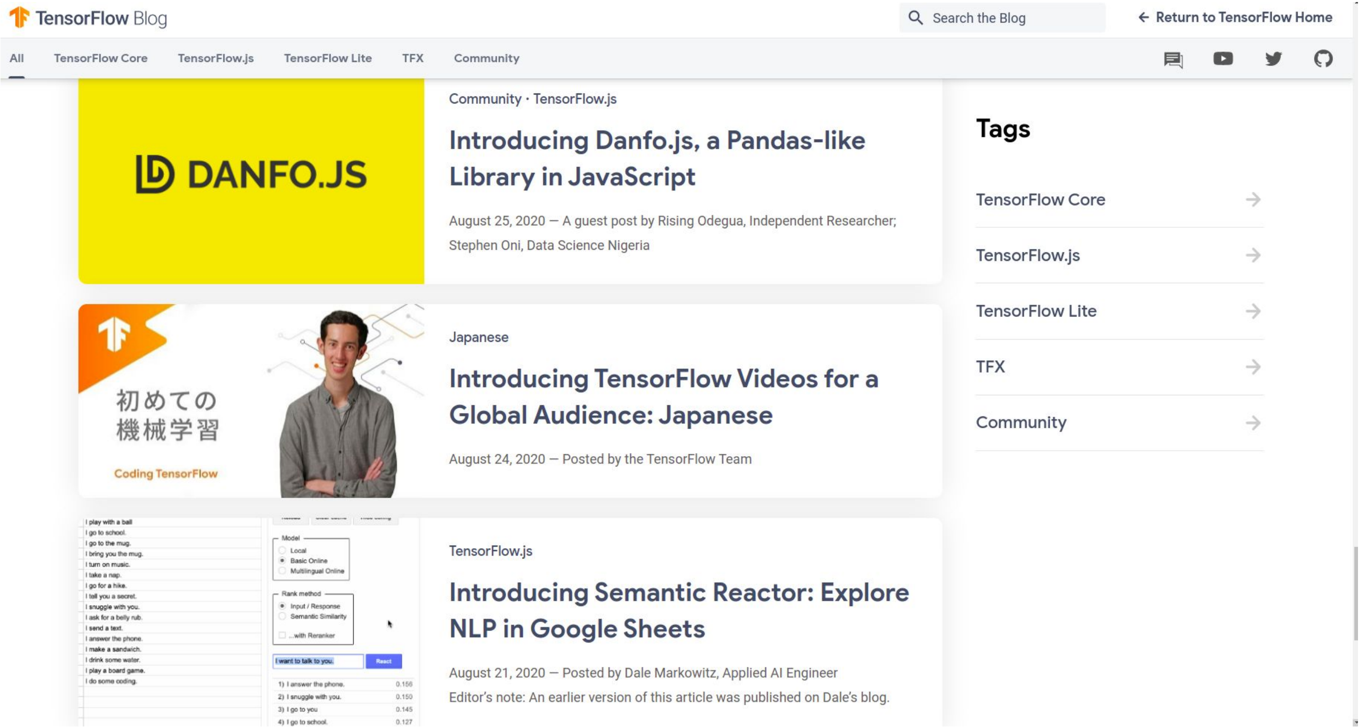
TensorFlow Blog

Artículo: Bryan M. Li, FOR.ai

Código: Abonia1

M. Li, B. (2019, 23 mayo). *A Transformer Chatbot Tutorial with TensorFlow 2.0*. TensorFlow Blog.

TensorFlow Blog



Procesamiento del dataset

Procesamiento del dataset

The Cornell Movie-Dialog

Descripción General:

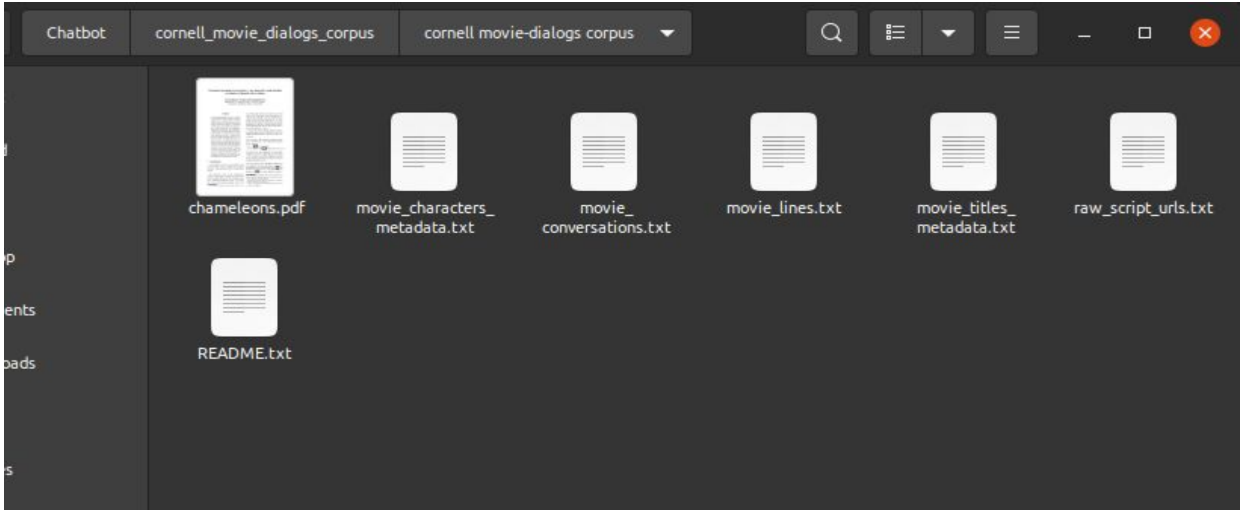
- 220,579 conversaciones intercambiadas entre 10,292 personajes de películas.
- Incluye 9,035 personajes de 617 películas.
- En total 304,713 expresiones.

Niculescu Mizil, C. D., & Lee, L. (2011). *Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs*. Cristian Danescu-Niculescu-Mizil.

Procesamiento del dataset

The Cornell Movie-Dialog

Archivos:



Niculescu Mizil, C. D., & Lee, L. (2011). *Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs*. Cristian Danescu-Niculescu-Mizil.

Procesamiento del dataset

The Cornell Movie-Dialog

1. movie_conversations.txt:

```
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L194', 'L195', 'L196', 'L197']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L198', 'L199']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L200', 'L201', 'L202', 'L203']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L204', 'L205', 'L206']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L207', 'L208']
```

Niculescu Mizil, C. D., & Lee, L. (2011). *Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs*. Cristian Danescu-Niculescu-Mizil.

Procesamiento del dataset

The Cornell Movie-Dialog

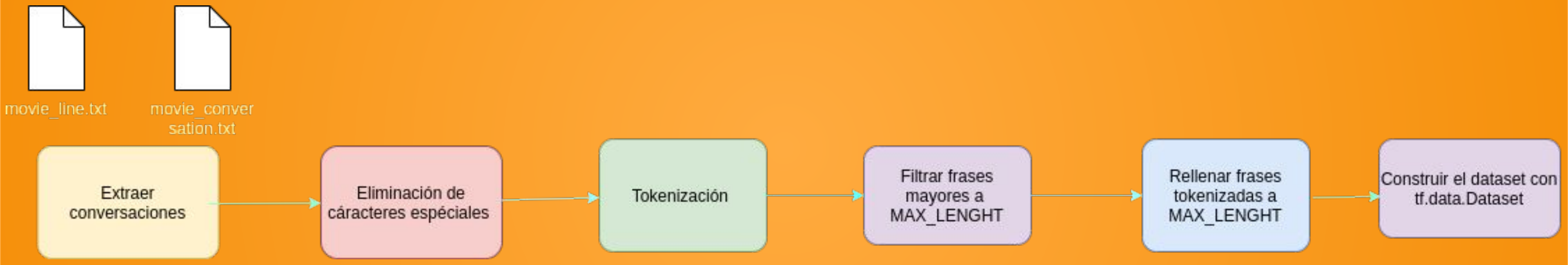
1. movie_lines.txt:

```
L901 +++$+++ u5 +++$+++ m0 +++$+++ KAT +++$+++ He said everyone was doing it. So I did it.
L900 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ As in…
L899 +++$+++ u5 +++$+++ m0 +++$+++ KAT +++$+++ Now I do. Back then, was a different story.
L898 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ But you hate Joey
L897 +++$+++ u5 +++$+++ m0 +++$+++ KAT +++$+++ He was, like, a total babe
```

Niculescu Mizil, C. D., & Lee, L. (2011). *Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs*. Cristian Danescu-Niculescu-Mizil.

Procesamiento del dataset

Pipeline



Procesamiento del dataset

- Path_line: ruta de movie_line.txt
- Path_movie_conversation: ruta de movie_conversation.txt
- Max_Samples: Max cantidad de ejemplos de entrenamiento(50000)

```python
def preprocess_sentences(frase):
    frase = frase.lower().strip()
    frase = re.sub(r"([?.!,])", r"\1 ", frase)
    frase = re.sub(r'[" "]+', " ", frase)
    frase = re.sub(r"[^a-zA-Z?.!,]+", " ", frase)
    frase = frase.strip()
    return frase

def load_sentence():
    id2line = {}
    with open(path_lines, errors="ignore") as file:
        lines = file.readlines()
    for line in lines:
        parts = line.replace("\n", "").split(" +++$+++ ")
        id2line[parts[0]] = parts[4]

    inputs, outputs = [], []
    with open(path_movie_conversions, "r") as file:
        lines = file.readlines()
    for line in lines:
        parts = line.replace("\n", "").split(" +++$+++ ")
        conversation = [line[1:-1] for line in parts[3][1:-1].split(", ")]
        for i in range(len(conversation) - 1):
            inputs.append(preprocess_sentences(id2line[conversation[i]]))
            outputs.append(preprocess_sentences(id2line[conversation[i + 1]]))
            if len(inputs) >= MAX_SAMPLES:
                return inputs, outputs
    return inputs, outputs
```

Procesamiento

del dataset

Which means unknown word pieces will be encoded one character at a time. It's best understood through an example. Let's suppose you build a `SubwordTextEncoder` using a very large corpus of English text such that most of the common words are in vocabulary.

```
vocab_size = 10000
tokenizer = tfds.features.text.SubwordTextEncoder.build_from_corpus(
    corpus_sentences, vocab_size)
```

Let's say you try to tokenize the following sentence.

```
tokenizer.encode("good badwords badxyz")
```

It will be tokenized as:

1. good
2. bad
3. words
4. bad
5. x
6. y
7. z

As you can see, since the word piece "xyz" is not in vocabulary it is tokenized as characters.

```
# Build tokenizer using tfds for both questions and answers
tokenizer = tfds.features.text.SubwordTextEncoder.build_from_corpus(
    inputs + outputs, target_vocab_size=2**13)
```

```
START_TOKEN, END_TOKEN = [tokenizer.vocab_size], [tokenizer.vocab_size + 1]
VOCAB_SIZE = tokenizer.vocab_size + 2
```

```
print(tokenizer.encode("hello there!!"))
print(tokenizer.decode(tokenizer.encode("hello there!!")))
print(tokenizer.vocab_size)
```

```
[2276, 126, 8110, 8110]
hello there!!
8333
```

```
print(START_TOKEN, END_TOKEN)
```

```
[8333] [8334]
```

Procesamiento del dataset

- Max_LENGHT: Max longitud de la frase.(40)

```
def tokenize_and_filter(input, output, MAX_LENGHT):
    tokenize_inputs, tokienize_outputs = [], []
    for(sentence1, sentence2) in zip(input, output):
        sentence1 = START_TOKEN + tokenizer.encode(sentence1) + END_TOKEN
        sentence2 = START_TOKEN + tokenizer.encode(sentence2) + END_TOKEN
        if len(sentence1) <= MAX_LENGHT and len(sentence2) <= MAX_LENGHT:
            tokenize_inputs.append(sentence1)
            tokienize_outputs.append(sentence2)
    tokenize_inputs = tf.keras.preprocessing.sequence.pad_sequences(
        tokenize_inputs, maxlen= MAX_LENGHT, padding= 'post')
    tokienize_outputs = tf.keras.preprocessing.sequence.pad_sequences(
        tokienize_outputs, maxlen= MAX_LENGHT, padding= 'post')
    return tokenize_inputs, tokienize_outputs
```

Procesamiento del dataset

```python
BATCH_SIZE = 64
BUFFER_SIZE = 20000

dataset = tf.data.Dataset.from_tensor_slices((
    {
        'inputs': inputs,
        'dec_inputs': outputs[:, :-1]
    },
    {
        'outputs': outputs[:,1:]
    },
))

dataset = dataset.cache()
dataset = dataset.shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE)
dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
```

Python

```python
dataset
```

Python

```
<PrefetchDataset shapes: ({inputs: (None, 40), dec_inputs: (None, 39)}, {outputs: (None, 39)}), types: ({inputs: tf.int32, dec_inputs: tf.int32}, {outputs: tf.int32})>
```

Transformers

Arquitectura Transformer

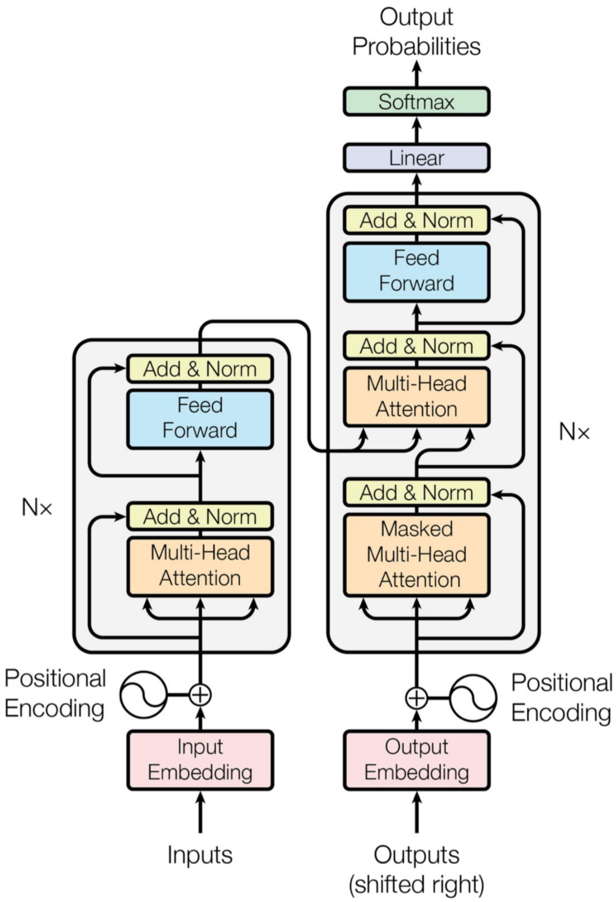


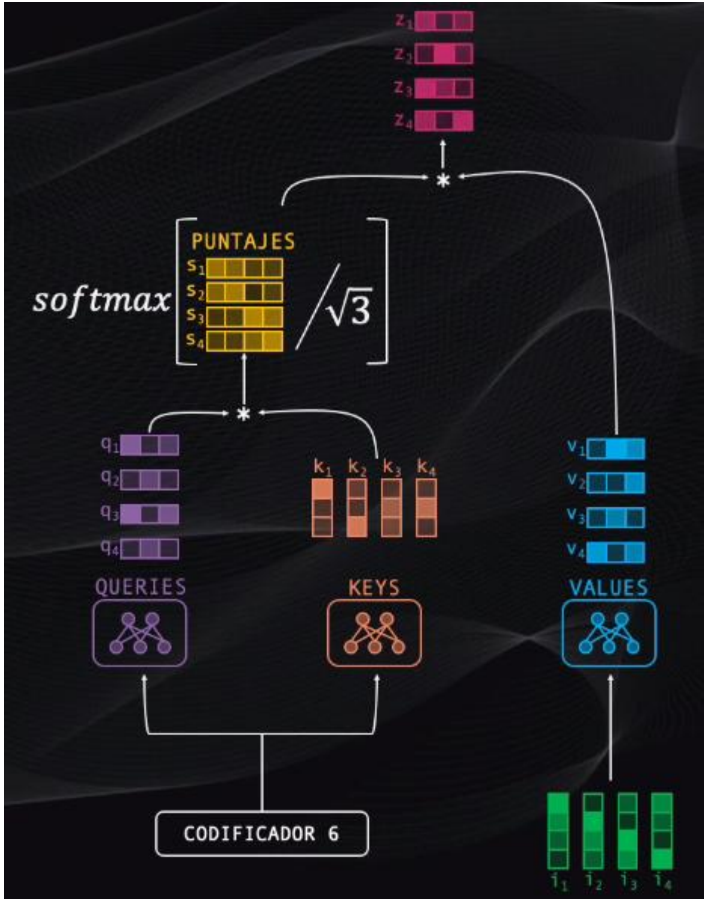
Figure 1: The Transformer - model architecture.

Implementación Multi-Head Attention

Implementación de Atención

scaled

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\boxed{\sqrt{d_k}}}\right)V$$



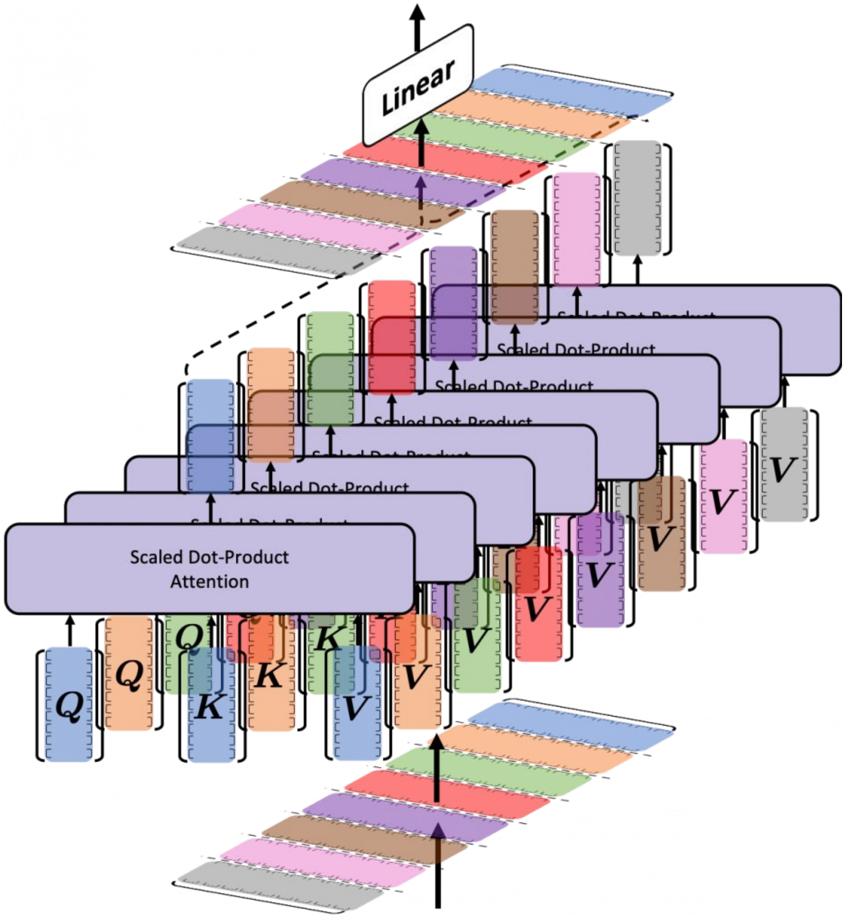
Implementación de Atención

scaled

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\boxed{\sqrt{d_k}}})V$$

```python
def scaled_dot_product_attention(query, key, value, mask):
  matmul_qk = tf.matmul(query, key, transpose_b=True)

  depth = tf.cast(tf.shape(key)[-1], tf.float32)
  logits = matmul_qk / tf.math.sqrt(depth)

  # add the mask zero out padding tokens.
  if mask is not None:
    logits += (mask * -1e9)

  attention_weights = tf.nn.softmax(logits, axis=-1)

  return tf.matmul(attention_weights, value)
```

Implementación de MultiHead Attention



Implementación de Multi Head Attention

```python
class MultiHeadAttention(tf.keras.layers.Layer):

  def __init__(self, d_model, num_heads, name="multi_head_attention"):
    super(MultiHeadAttention, self).__init__(name=name)
    self.num_heads = num_heads
    self.d_model = d_model

    assert d_model % self.num_heads == 0

    self.depth = d_model // self.num_heads

    self.query_dense = tf.keras.layers.Dense(units=d_model)
    self.key_dense = tf.keras.layers.Dense(units=d_model)
    self.value_dense = tf.keras.layers.Dense(units=d_model)

    self.dense = tf.keras.layers.Dense(units=d_model)

  def split_heads(self, inputs, batch_size):
    inputs = tf.reshape(
        inputs, shape=(batch_size, -1, self.num_heads, self.depth))
    return tf.transpose(inputs, perm=[0, 2, 1, 3])
```

Implementación de Multi Head Attention

```python
def call(self, inputs):
    query, key, value, mask = inputs['query'], inputs['key'], inputs[
        'value'], inputs['mask']
    batch_size = tf.shape(query)[0]

    # linear layers
    query = self.query_dense(query)
    key = self.key_dense(key)
    value = self.value_dense(value)

    # split heads
    query = self.split_heads(query, batch_size)
    key = self.split_heads(key, batch_size)
    value = self.split_heads(value, batch_size)

    scaled_attention = scaled_dot_product_attention(query, key, value, mask)

    scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])

    concat_attention = tf.reshape(scaled_attention,
                                  (batch_size, -1, self.d_model))

    outputs = self.dense(concat_attention)

    return outputs
```

Enmascaramiento

create_paddind_mask

```python
def create_padding_mask(x):
  mask = tf.cast(tf.math.equal(x, 0), tf.float32)
  # (batch_size, 1, 1, sequence length)
  return mask[:, tf.newaxis, tf.newaxis, :]
```

```python
def create_look_ahead_mask(x):
  seq_len = tf.shape(x)[1]
  look_ahead_mask = 1 - tf.linalg.band_part(tf.ones((seq_len, seq_len)), -1, 0)
  padding_mask = create_padding_mask(x)
  return tf.maximum(look_ahead_mask, padding_mask)
```

```python
print(create_padding_mask(tf.constant([[1, 2, 0, 3, 0], [0, 0, 0, 4, 5]])))
```

```
tf.Tensor(
[[[[0. 0. 1. 0. 1.]]]


 [[[1. 1. 1. 0. 0.]]]], shape=(2, 1, 1, 5), dtype=float32)
```

Ver Repositorio de Abonia1 click aquí

Enmascaramiento

create_look_ahead_mask

```python
def create_padding_mask(x):
  mask = tf.cast(tf.math.equal(x, 0), tf.float32)
  # (batch_size, 1, 1, sequence length)
  return mask[:, tf.newaxis, tf.newaxis, :]
```

```python
def create_look_ahead_mask(x):
  seq_len = tf.shape(x)[1]
  look_ahead_mask = 1 - tf.linalg.band_part(tf.ones((seq_len, seq_len)), -1, 0)
  padding_mask = create_padding_mask(x)
  return tf.maximum(look_ahead_mask, padding_mask)
```

```python
print(create_look_ahead_mask(tf.constant([[1, 2, 0, 4, 5]])))
```

```
tf.Tensor(
[[[[0. 1. 1. 1. 1.]
   [0. 0. 1. 1. 1.]
   [0. 0. 1. 1. 1.]
   [0. 0. 1. 0. 1.]
   [0. 0. 1. 0. 0.]]]], shape=(1, 1, 5, 5), dtype=float32)
```

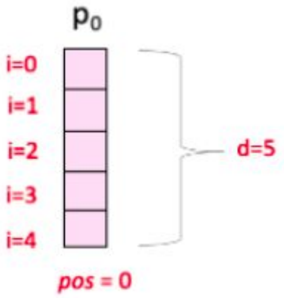
Ver Repositorio de Abonia1 click aquí

Positional encoding

- d es la dimensión de los word embedding
- pos es la posición de la palabra.
- i se refiere a cada una de las diferentes dimensiones de los positional encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

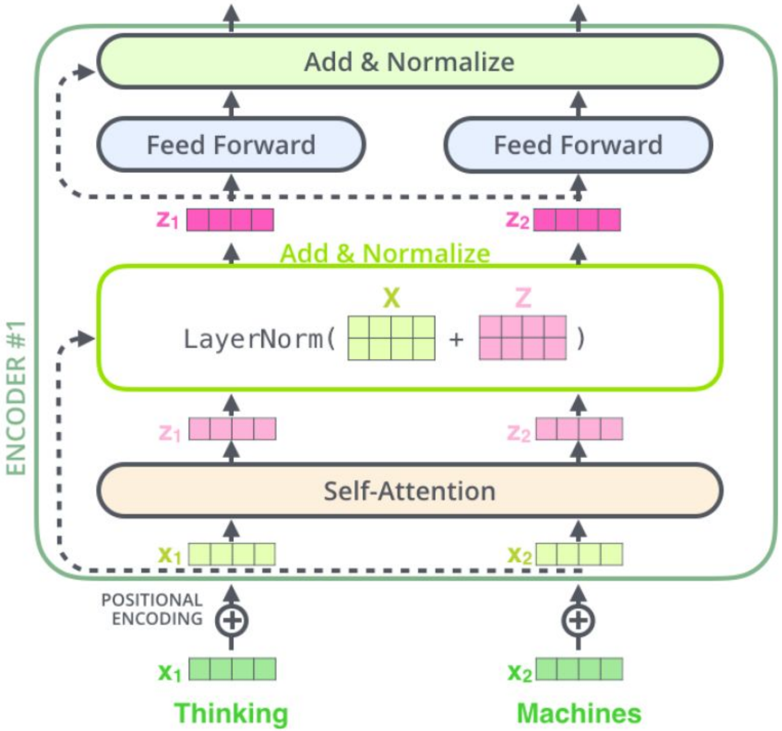


What is the positional encoding in the transformer model?

Positional encoding

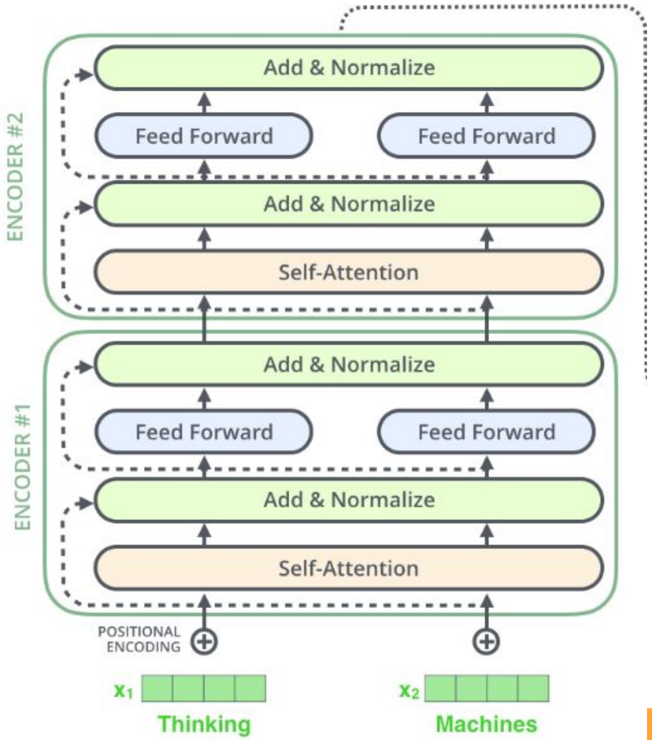
```python
class PositionalEncoding(tf.keras.layers.Layer):

  def __init__(self, position, d_model):
    super(PositionalEncoding, self).__init__()
    self.pos_encoding = self.positional_encoding(position, d_model)

  def get_angles(self, position, i, d_model):
    angles = 1 / tf.pow(10000, (2 * (i // 2)) / tf.cast(d_model, tf.float32))
    return position * angles

  def positional_encoding(self, position, d_model):
    angle_rads = self.get_angles(
        position=tf.range(position, dtype=tf.float32)[:, tf.newaxis],
        i=tf.range(d_model, dtype=tf.float32)[tf.newaxis, :],
        d_model=d_model)
    # apply sin to even index in the array
    sines = tf.math.sin(angle_rads[:, 0::2])
    # apply cos to odd index in the array
    cosines = tf.math.cos(angle_rads[:, 1::2])

    pos_encoding = tf.concat([sines, cosines], axis=-1)
    pos_encoding = pos_encoding[tf.newaxis, ...]
    return tf.cast(pos_encoding, tf.float32)

  def call(self, inputs):
    return inputs + self.pos_encoding[:, :tf.shape(inputs)[1], :]
```

Encoding layer



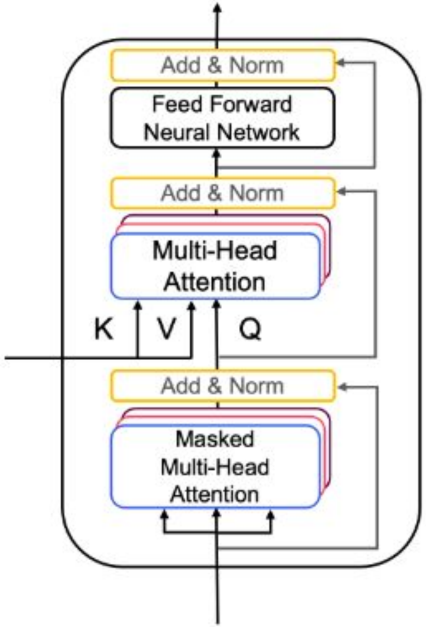

```python
def encoder_layer(units, d_model, num_heads, dropout, name="encoder_layer"):
  inputs = tf.keras.Input(shape=(None, d_model), name="inputs")
  padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")

  attention = MultiHeadAttention(
      d_model, num_heads, name="attention")({
          'query': inputs,
          'key': inputs,
          'value': inputs,
          'mask': padding_mask
      })
  attention = tf.keras.layers.Dropout(rate=dropout)(attention)
  attention = tf.keras.layers.LayerNormalization(
      epsilon=1e-6)(inputs + attention)

  outputs = tf.keras.layers.Dense(units=units, activation='relu')(attention)
  outputs = tf.keras.layers.Dense(units=d_model)(outputs)
  outputs = tf.keras.layers.Dropout(rate=dropout)(outputs)
  outputs = tf.keras.layers.LayerNormalization(
      epsilon=1e-6)(attention + outputs)

  return tf.keras.Model(
      inputs=[inputs, padding_mask], outputs=outputs, name=name)
```

Encoder



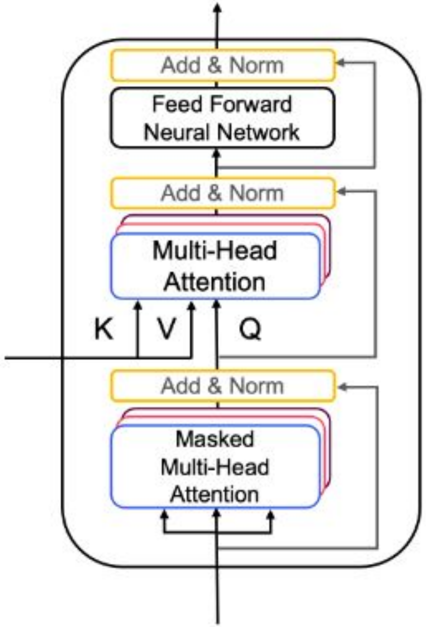

```python
def encoder(vocab_size,
            num_layers,
            units,
            d_model,
            num_heads,
            dropout,
            name="encoder"):
  inputs = tf.keras.Input(shape=(None,), name="inputs")
  padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")

  embeddings = tf.keras.layers.Embedding(vocab_size, d_model)(inputs)
  embeddings *= tf.math.sqrt(tf.cast(d_model, tf.float32))
  embeddings = PositionalEncoding(vocab_size, d_model)(embeddings)

  outputs = tf.keras.layers.Dropout(rate=dropout)(embeddings)

  for i in range(num_layers):
    outputs = encoder_layer(
        units=units,
        d_model=d_model,
        num_heads=num_heads,
        dropout=dropout,
        name="encoder_layer_{}".format(i),
    )([outputs, padding_mask])

  return tf.keras.Model(
      inputs=[inputs, padding_mask], outputs=outputs, name=name)
```

Decoding layer



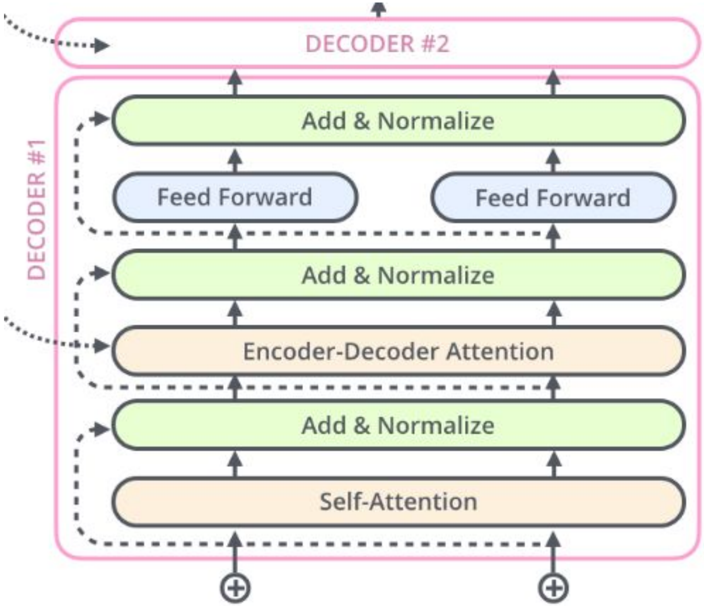

```python
def decoder_layer(units, d_model, num_heads, dropout, name="decoder_layer"):
    inputs = tf.keras.Input(shape=(None, d_model), name="inputs")
    enc_outputs = tf.keras.Input(shape=(None, d_model), name="encoder_outputs")
    look_ahead_mask = tf.keras.Input(
        shape=(1, None, None), name="look_ahead_mask")
    padding_mask = tf.keras.Input(shape=(1, 1, None), name='padding_mask')

    attention1 = MultiHeadAttention(
        d_model, num_heads, name="attention_1")(inputs={
            'query': inputs,
            'key': inputs,
            'value': inputs,
            'mask': look_ahead_mask
        })
    attention1 = tf.keras.layers.LayerNormalization(
        epsilon=1e-6)(attention1 + inputs)

    attention2 = MultiHeadAttention(
        d_model, num_heads, name="attention_2")(inputs={
            'query': attention1,
            'key': enc_outputs,
            'value': enc_outputs,
            'mask': padding_mask
        })
```

Decoding layer



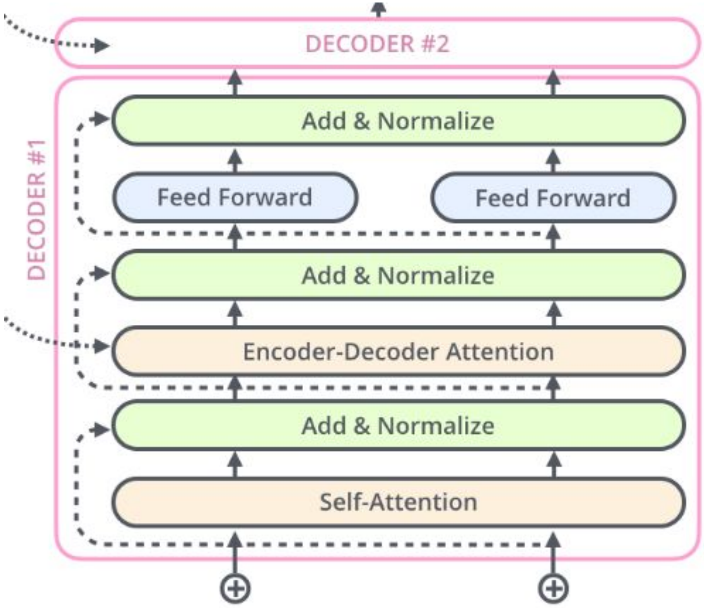

```python
attention1 = tf.keras.layers.LayerNormalization(
    epsilon=1e-6)(attention1 + inputs)

attention2 = MultiHeadAttention(
    d_model, num_heads, name="attention_2")(inputs={
        'query': attention1,
        'key': enc_outputs,
        'value': enc_outputs,
        'mask': padding_mask
    })
attention2 = tf.keras.layers.Dropout(rate=dropout)(attention2)
attention2 = tf.keras.layers.LayerNormalization(
    epsilon=1e-6)(attention2 + attention1)

outputs = tf.keras.layers.Dense(units=units, activation='relu')(attention2)
outputs = tf.keras.layers.Dense(units=d_model)(outputs)
outputs = tf.keras.layers.Dropout(rate=dropout)(outputs)
outputs = tf.keras.layers.LayerNormalization(
    epsilon=1e-6)(outputs + attention2)

return tf.keras.Model(
    inputs=[inputs, enc_outputs, look_ahead_mask, padding_mask],
    outputs=outputs,
    name=name)
```

Decoder



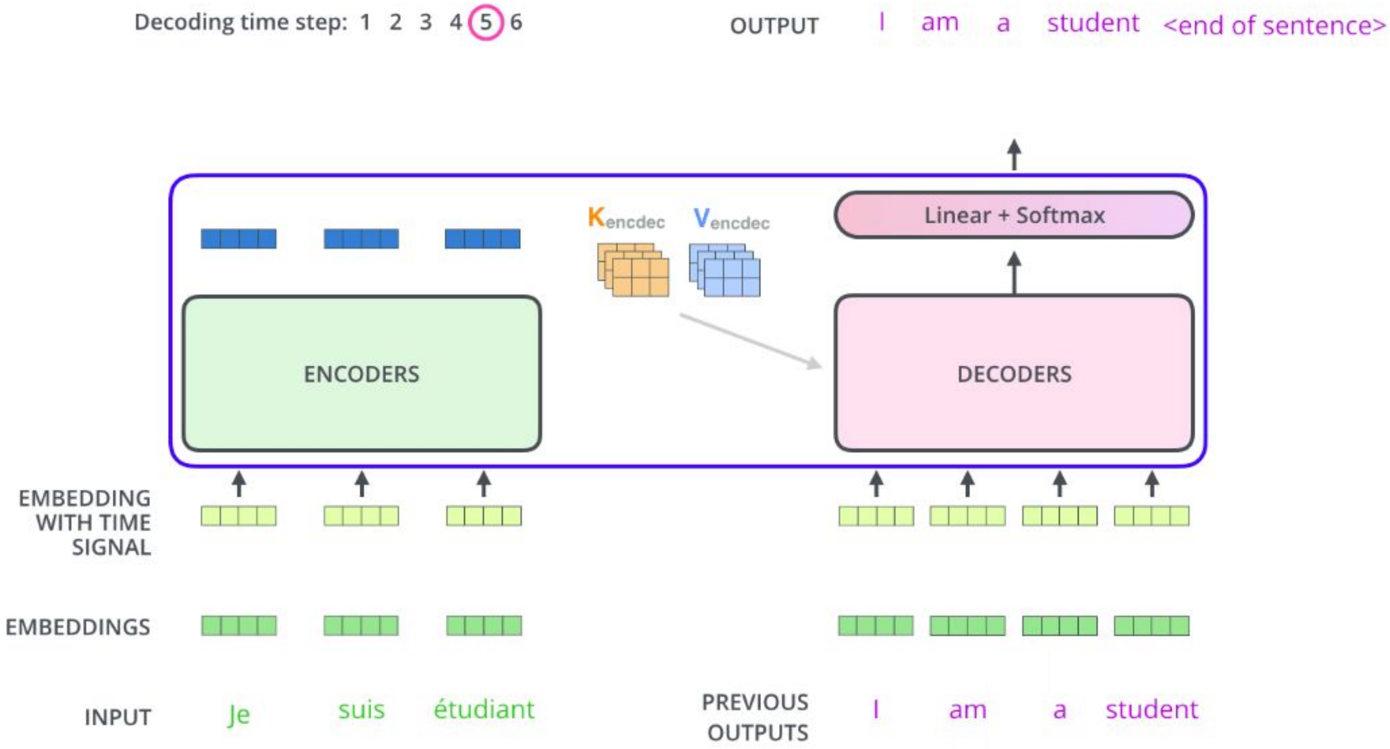

```python
def decoder(vocab_size,
            num_layers,
            units,
            d_model,
            num_heads,
            dropout,
            name='decoder'):
  inputs = tf.keras.Input(shape=(None,), name='inputs')
  enc_outputs = tf.keras.Input(shape=(None, d_model), name='encoder_outputs')
  look_ahead_mask = tf.keras.Input(
      shape=(1, None, None), name='look_ahead_mask')
  padding_mask = tf.keras.Input(shape=(1, 1, None), name='padding_mask')

  embeddings = tf.keras.layers.Embedding(vocab_size, d_model)(inputs)
  embeddings *= tf.math.sqrt(tf.cast(d_model, tf.float32))
  embeddings = PositionalEncoding(vocab_size, d_model)(embeddings)

  outputs = tf.keras.layers.Dropout(rate=dropout)(embeddings)
```

Decoder




```
for i in range(num_layers):
  outputs = decoder_layer(
      units=units,
      d_model=d_model,
      num_heads=num_heads,
      dropout=dropout,
      name='decoder_layer_{}'.format(i),
  )(inputs=[outputs, enc_outputs, look_ahead_mask, padding_mask])

return tf.keras.Model(
    inputs=[inputs, enc_outputs, look_ahead_mask, padding_mask],
    outputs=outputs,
    name=name)
```

Transformer



Transformer

```python
def transformer(vocab_size,
                num_layers,
                units,
                d_model,
                num_heads,
                dropout,
                name="transformer"):
  inputs = tf.keras.Input(shape=(None,), name="inputs")
  dec_inputs = tf.keras.Input(shape=(None,), name="dec_inputs")

  enc_padding_mask = tf.keras.layers.Lambda(
      create_padding_mask, output_shape=(1, 1, None),
      name='enc_padding_mask')(inputs)
  # Enmascaramos los futuros tokens para la entrada del decoder para el primer bloque de
  look_ahead_mask = tf.keras.layers.Lambda(
      create_look_ahead_mask,
      output_shape=(1, None, None),
      name='look_ahead_mask')(dec_inputs)
  # Enmascarmos la salida encoders para el segundo bloque de atención
  dec_padding_mask = tf.keras.layers.Lambda(
      create_padding_mask, output_shape=(1, 1, None),
      name='dec_padding_mask')(inputs)
```

Lambda Layers click aquí

Transformer

```python
enc_outputs = encoder(
    vocab_size=vocab_size,
    num_layers=num_layers,
    units=units,
    d_model=d_model,
    num_heads=num_heads,
    dropout=dropout,
)(inputs=[inputs, enc_padding_mask])

dec_outputs = decoder(
    vocab_size=vocab_size,
    num_layers=num_layers,
    units=units,
    d_model=d_model,
    num_heads=num_heads,
    dropout=dropout,
)(inputs=[dec_inputs, enc_outputs, look_ahead_mask, dec_padding_mask])

outputs = tf.keras.layers.Dense(units=vocab_size, name="outputs")(dec_outputs)

return tf.keras.Model(inputs=[inputs, dec_inputs], outputs=outputs, name=name)
```

Entrenamiento

Entrenamiento

Hiper Parámetros

```
NUM_LAYERS = 2
D_MODEL = 256
NUM_HEADS = 8
UNITS = 512
DROPOUT = 0.1
MAX_LENGTH = 40
model = transformer(
    vocab_size=VOCAB_SIZE,
    num_layers=NUM_LAYERS,
    units=UNITS,
    d_model=D_MODEL,
    num_heads=NUM_HEADS,
    dropout=DROPOUT)
```

Entrenamiento

Función de Pérdida

$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

```python
def loss_function(y_true, y_pred):
    y_true = tf.reshape(y_true, shape=(-1, MAX_LENGTH - 1))

    loss = tf.keras.losses.SparseCategoricalCrossentropy(
        from_logits=True, reduction='none')(y_true, y_pred)

    mask = tf.cast(tf.not_equal(y_true, 0), tf.float32)
    loss = tf.multiply(loss, mask)

    return tf.reduce_mean(loss)
```

Entrenamiento

Diferencia Entre Sparse y Categorical

If your targets are **one-hot encoded**, use `categorical_crossentropy`.

- Examples of one-hot encodings:
  - `[1,0,0]`
  - `[0,1,0]`
  - `[0,0,1]`

But if your targets are **integers**, use `sparse_categorical_crossentropy`.

- Examples of integer encodings (*for the sake of completion*):
  - `1`
  - `2`
  - `3`

Gautam, S. (2020, 27 marzo). *Categorical Cross Entropy vs Sparse Categorical Cross Entropy*. Medium.

Entrenamiento

Radio de aprendizaje

```python
class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):

  def __init__(self, d_model, warmup_steps=4000):
    super(CustomSchedule, self).__init__()

    self.d_model = d_model
    self.d_model = tf.cast(self.d_model, tf.float32)

    self.warmup_steps = warmup_steps

  def __call__(self, step):
    arg1 = tf.math.rsqrt(step)
    arg2 = step * (self.warmup_steps**-1.5)

    return tf.math.rsqrt(self.d_model) * tf.math.minimum(arg1, arg2)
```

Fórmula

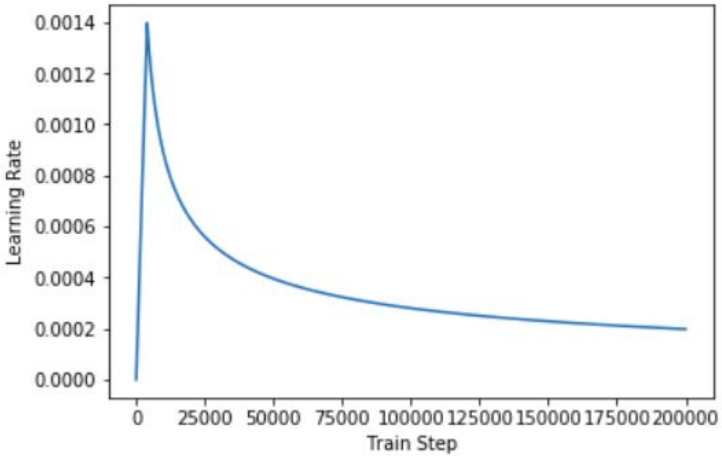
$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

Revisar documentación de tensorflow “click aquí”

Entrenamiento

Radio de aprendizaje

```
In [ ]: sample_learning_rate = CustomSchedule(d_model=128)

        plt.plot(sample_learning_rate(tf.range(200000, dtype=tf.float32)))
        plt.ylabel("Learning Rate")
        plt.xlabel("Train Step")

Out[ ]: Text(0.5, 0, 'Train Step')
```



Fórmula

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

Ver Repositorio de Abonia1 click aquí

Entrenamiento

Optimizador y Métrica

RMSProp

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t}} g_{t-1}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(g_{t-1})^2$$
$$v_1 = (g_0)^2$$

SGDM

$$\theta_t = \theta_{t-1} - \eta m_t$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_2) g_{t-1}$$

Adam

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t}} m_t \text{ (outline)}$$
$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\widehat{v}_t}+\epsilon} \widehat{m}_t \text{ (complete)}$$
$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \widehat{v}_t = \frac{v_t}{1-\beta_2^t}$$

```python
learning_rate = CustomSchedule(D_MODEL)

optimizer = tf.keras.optimizers.Adam(
    learning_rate, beta_1=0.9, beta_2=0.98, epsilon=1e-9)

def accuracy(y_true, y_pred):
  # ensure labels have shape (batch_size, MAX_LENGTH - 1)
  y_true = tf.reshape(y_true, shape=(-1, MAX_LENGTH - 1))
  return tf.keras.metrics.sparse_categorical_accuracy(y_true, y_pred)
```

Entrenamiento

Optimizador y Métrica

```python
def categorical_accuracy(y_true, y_pred):
    return K.cast(K.equal(K.argmax(y_true, axis=-1),
                          K.argmax(y_pred, axis=-1)),
                  K.floatx())


def sparse_categorical_accuracy(y_true, y_pred):
    return K.cast(K.equal(K.max(y_true, axis=-1),
                          K.cast(K.argmax(y_pred, axis=-1), K.floatx())),
K.floatx())
```

```python
learning_rate = CustomSchedule(D_MODEL)

optimizer = tf.keras.optimizers.Adam(
    learning_rate, beta_1=0.9, beta_2=0.98, epsilon=1e-9)

def accuracy(y_true, y_pred):
  # ensure labels have shape (batch_size, MAX_LENGTH - 1)
  y_true = tf.reshape(y_true, shape=(-1, MAX_LENGTH - 1))
  return tf.keras.metrics.sparse_categorical_accuracy(y_true, y_pred)
```

`categorical_accuracy` checks to see if the *index* of the maximal true value is equal to the *index* of the maximal predicted value.

`sparse_categorical_accuracy` checks to see if the maximal true value is equal to the *index* of the maximal predicted value.

Revisar documentación de tensorflow “click aquí”

Entrenamiento

```python
model.compile(optimizer=optimizer, loss=loss_function, metrics=[accuracy])
```

Python

```python
EPOCHS = 15
```

Python

```python
model.fit(dataset, epochs=EPOCHS)
```

Python

Entrenamiento

```
Epoch 6/15
690/690 [==============================] - 715s 1s/step - loss: 1.2432 - accuracy: 0.0973
Epoch 7/15
690/690 [==============================] - 813s 1s/step - loss: 1.1879 - accuracy: 0.1017
Epoch 8/15
690/690 [==============================] - 720s 1s/step - loss: 1.1284 - accuracy: 0.1068
Epoch 9/15
690/690 [==============================] - 628s 911ms/step - loss: 1.0721 - accuracy: 0.1122
Epoch 10/15
690/690 [==============================] - 634s 918ms/step - loss: 1.0196 - accuracy: 0.1180
Epoch 11/15
690/690 [==============================] - 617s 894ms/step - loss: 0.9718 - accuracy: 0.1237
Epoch 12/15
690/690 [==============================] - 620s 899ms/step - loss: 0.9283 - accuracy: 0.1292
Epoch 13/15

show more (open the raw output data in a text editor) ...

690/690 [==============================] - 635s 920ms/step - loss: 0.8881 - accuracy: 0.1344
Epoch 14/15
690/690 [==============================] - 631s 915ms/step - loss: 0.8519 - accuracy: 0.1398
Epoch 15/15
690/690 [==============================] - 657s 952ms/step - loss: 0.8185 - accuracy: 0.1449

<tensorflow.python.keras.callbacks.History at 0x7f78535f9d10>
```

Evaluación

Función evaluate

```
def evaluate(sentence):
  sentence = preprocess_sentences(sentence)

  sentence = tf.expand_dims(
      START_TOKEN + tokenizer.encode(sentence) + END_TOKEN, axis=0)

  output = tf.expand_dims(START_TOKEN, 0)

  for i in range(MAX_LENGTH):
    predictions = model(inputs=[sentence, output], training=False)
    # Seleccionamos la última palabra de la dimesión seq_len
    predictions = predictions[:, -1:, :]
    predicted_id = tf.cast(tf.argmax(predictions, axis=-1), tf.int32)

    # Retornamos el resultado si  predicted_id es igual que el token final.
    if tf.equal(predicted_id, END_TOKEN[0]):
        break

    # Concatenamos el predicted_id a la salida que se le da al decodificador
   # como input.
    output = tf.concat([output, predicted_id], axis=-1)

  return tf.squeeze(output, axis=0)
```

Evaluación

Función predict

```
def predict(sentence, name):
  prediction = evaluate(sentence)

  predicted_sentence = tokenizer.decode(
      [i for i in prediction if i < tokenizer.vocab_size])

  print('{}: {}'.format(name, sentence))
  print('Bot: {}'.format(predicted_sentence))

  return predicted_sentence
```

Resultados

```
output = predict('Hello', 'Alex')
```

```
Alex: Hello
Bot: hi.
```

Resultados

```
output = predict('Where have you been?', "Alex")
```

```
Alex: Where have you been?

Bot: i m a little scared of myself.
```

Guardar Pesos

```
model.save_weights('path_to_my_model.h5')
```

```
model.load_weights('path_to_my_model.h5')
```

Implementación del Chatbot

```
def dialog(name):
    while(True):
        sentence = input('Write a sentences')
        if sentence.lower() == "f":
            break
        predict(sentence, name)
```

Implementación del Chatbot

```
dialog("Alex")
```

```
Alex: hello
Bot: hi.
Alex: how are you?
Bot: i m sorry i didn t have to do anything. i want to be unzip my dress.
Alex: do you study today?
Bot: i m going to be a little girl.
Alex: tell me about you
Bot: you re not going to be late.
Alex: ok, ok
Bot: you re a fucking asshole! you re a fucking bull to you!
Alex: calm down, please
Bot: i m going to give you something.
Alex: ok, see you
Bot: okay.
Alex: good bye
Bot: thank you.
```

Conclusiones

Gracias