1, Seq2Seq

https://zh.wikipedia.org/wiki/Seq2seq

Avec attention

Encodeur:

```
class Encoder(keras.layers.Layer):
    def __init__(self, enc_v_dim, emb_dim, units):
        super(Encoder, self).__init__()
        self.units = units
        self.enc_embeddings = keras.layers.Embedding(
            input_dim=enc_v_dim, output_dim=emb_dim,  # [enc_n_vocab, emb_dim]
            embeddings_initializer=tf.initializers.RandomNormal(0., 0.1),
        )
        self.encoder = keras.layers.LSTM(units=units, return_sequences=True, return_state=True)
```

A l'aide d'un encodeur, les mots qui ont été numérisés sont vectorisés pour former un espace de mots.Dans cet espace de mots, les relations mot-mot peuvent être formées.

Decodeur:

```
class Decoder(keras.layers.Layer):
    def __init__(self, dec_v_dim, emb_dim, units, attention_layer_size, encoder):
        super(Decoder, self).__init__()
        self.units = units
        self.attention = tfa.seq2seq.LuongAttention(units, memory=None, memory_sequence_length=None)
        self.encoder = encoder
        self.decoder_cell = tfa.seq2seq.AttentionWrapper(
            cell=keras.layers.LSTMCell(units=units),
            attention_mechanism=self.attention,
            attention_layer_size=attention_layer_size,
            alignment_history=True,  # for attention visualization
        )
        self.dec_embeddings = keras.layers.Embedding(
            input_dim=dec_v_dim, output_dim=emb_dim,  # [dec_n_vocab, emb_dim]
            embeddings_initializer=tf.initializers.RandomNormal(0., 0.1),
        )
        self.decoder_dense = keras.layers.Dense(dec_v_dim)  # output layer

        # train decoder
        self.decoder_train = tfa.seq2seq.BasicDecoder(
            cell=self.decoder_cell,
            sampler=tfa.seq2seq.sampler.TrainingSampler(),  # sampler for train
            output_layer=self.decoder_dense
        )
        self.cross_entropy = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
        self.opt = keras.optimizers.Adam(0.05, clipnorm=5.0)
```

Une couche de décodage avec un mécanisme d'attention inspiré du mécanisme de lecture humaine nous permet de parcourir toute la séquence et de prêter attention à certaines informations clés avant de traiter la séquence. pour une meilleure compréhension de la séquence. Le mécanisme d'attention peut améliorer l'efficacité de la formation du modèle.

```
class ModelTrain(keras.Model):
    def __init__(self, enc_v_dim, dec_v_dim, emb_dim, units, attention_layer_size):
        super(ModelTrain, self).__init__()
        self.encoder = Encoder(enc_v_dim, emb_dim, units)
        self.decoder = Decoder(dec_v_dim, emb_dim, units, attention_layer_size, self.encoder)
        self.cross_entropy = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
        self.opt = keras.optimizers.Adam(0.05, clipnorm=5.0)
```

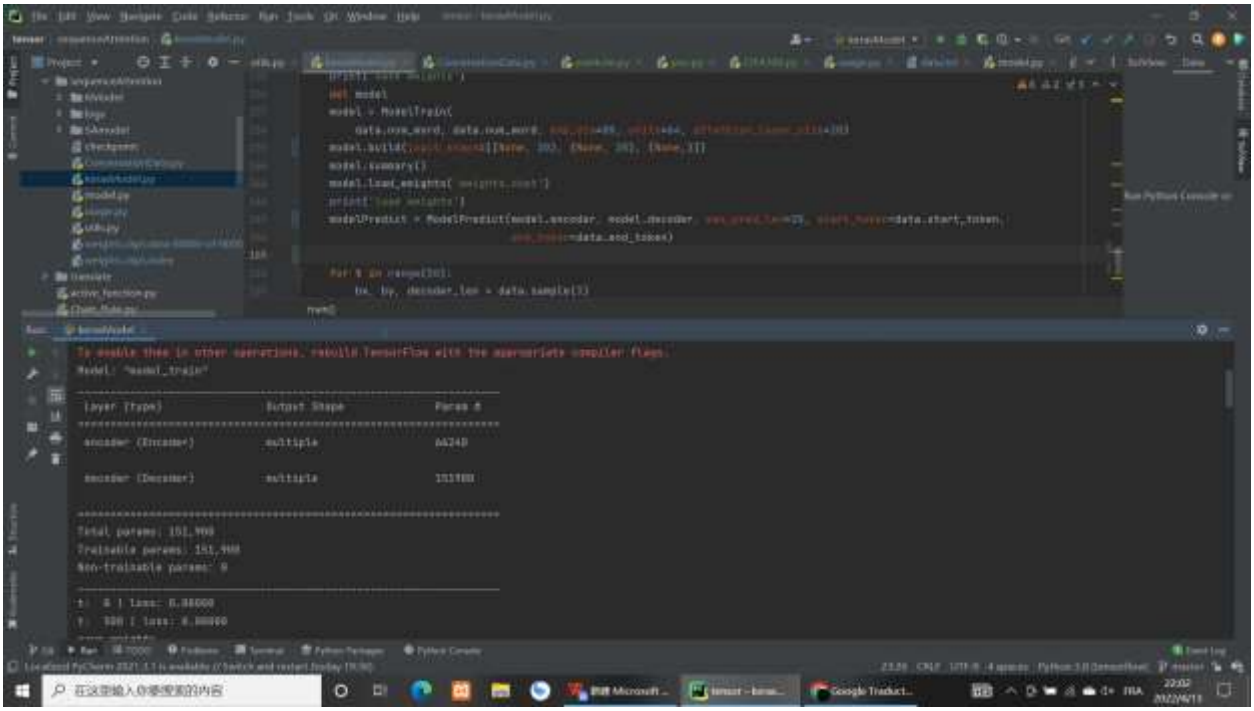
```
class ModelPredict(keras.Model):
    def __init__(self, encoder, decoder, max_pred_len, start_token, end_token):
        super(ModelPredict, self).__init__()
        self.encoder = encoder
        self.decoder = decoder

        self.decoder_eval = tfa.seq2seq.BasicDecoder(
            cell=decoder.decoder_cell,
            sampler=tfa.seq2seq.sampler.GreedyEmbeddingSampler(),  # sampler for predict
            output_layer=decoder.decoder_dense
        )
        self.max_pred_len = max_pred_len
        self.start_token = start_token
        self.end_token = end_token
```

Le modèle d'apprentissage et le modèle de prédiction sont construits séparément.

```
model = ModelTrain(
    data.num_word, data.num_word, emb_dim=80, units=64, attention_layer_size=20)
model.build(input_shape=[(None, 20), (None, 20), (None,)])
```

```
modelPredict = ModelPredict(model.encoder, model.decoder, max_pred_len=20, start_token=data.start_token,
                            end_token=data.end_token)
```

Informations de configuration pour les deux modèles



Informations sur les paramètres du modèle

2,prétraitement des données

Pour le prétraitement des données pour les problèmes de PNL, le plus critique est de convertir les mots en représentations numériques. De plus, pour le mode Seq2Seq, les séquences d'entrée et de sortie sont de longueur égale, un remplissage est donc nécessaire.Et bien sûr la tokenisation la plus basique.

```
class ConversationData:
    def __init__(self):
        np.random.seed(1)
        self.data_question, self.data_answer, self.words = self.getData()

        self.vocab = set(
            [i for i in self.words] + ["<GO>", "<EOS>"])
        self.v2i = {v: i for i, v in enumerate(sorted(list(self.vocab)), start=1)}
        self.v2i["<PAD>"] = PAD_ID
        self.vocab.add("<PAD>")
        self.i2v = {i: v for v, i in self.v2i.items()}
        self.x, self.y = [], []
        for question, answer in zip(self.data_question, self.data_answer):
            self.x.append([self.v2i[v] for v in nltk.word_tokenize(question)])
            self.y.append(
                [self.v2i["<GO>"], ] + [self.v2i[v] for v in nltk.word_tokenize(answer)] + [
                    self.v2i["<EOS>"]])

        # self.x, self.y = np.array(self.x), np.array(self.y)

        self.x = keras.preprocessing.sequence.pad_sequences(self.x, maxlen=Max_length, padding='post')
        self.y = keras.preprocessing.sequence.pad_sequences(self.y, maxlen=Max_length, padding='post')
        self.x, self.y = np.array(self.x), np.array(self.y)
```

```
    self.start_token = self.v2i["<GO>"]
    self.end_token = self.v2i["<EOS>"]
```

Cette classe nous permet de lire un jeu de données sous une forme simple, de construire un corpus et un jeu de données d'apprentissage, et de numériser l'encodage des mots.