Neural Machine Translation from the English to Chinese

1. Corpus

The corpus from the English to Chinese is provided by the `Niutrans`, which has more than 100000 sents containe the Chinese corresponding English.

2. Notebooks about `de-en-s2s` project

1. `utils.py`

   This part of code (only have one function named `load_dataset`) try to load the dataset into the training pipeline, using the `spacy` to tokenize the dataset which provided by the `torchtext`. More details can be found in the `de-en-s2s/utils.py`, I just record the main messages in this file.

   - Input parameter is the size of the batch
   - Output parameters are `train_iter`, `val_iter`, `test_iter`, `DE`, `EN`.
   - `DE`, `EN` can provide some important messages
     - the number of the words in the special language
     - `DE.vocab.stoi[word]`method can get the index from the word.
     - `DE.vocab.itos[index]`methos can get the word from the index.
   - `train_iter`, `val_iter`, `test_iter` have also been changed into the index from the sequence of the words. And the index message is also important, here are some special index need to consider.
     - `<pad>`: index is 1 (in `DE` and `EN`), means the short sequence need to be padded to the max length of this batch.
     - `<sos>`: index is 2 (in `DE` and `EN`), means the start of sequence token.
     - `<eos>`: index is 3, means the end of the sequence token.
     - `<unk>`: index is 0, this token means the unknown word in the dictionary.
   - The size of the `train_iter`, `val_iter`, `test_iter` is `[sequence_length, batch_size]`which can be sent as the input further in the `de-en-s2s/model.py`

2. `model.py`

   - Class `Encoder`

     2 layers bidirentional GRU Encoder.

     - `input_size`: the size of the `DE` dictionary, 8043
     - `embed_size`: the word embedding size, 256 (without pretrain parameters)
     - `hidden_size`: the number of the hidden units in GRU Encoder, 512

     Weights in this encoder.

     ```
     Encoder(
         (embed): Embedding(8043, 256)
         (gru): GRU(256, 512, num_layers=2, dropout=0.5,bidirectional=True)
     )
     ```

     Forward process

     - Embed the `[seq_len, batch_size]`input to the `[seq_len, batch_size, n_embed]` from the lookup table matrix.

- GRU the embedded input (default the hidden is all zero) to get the `encoder_output([seq_len, batch_size, 2 * 512]`and `hidden([2 * 2, batch_size, 512])`. And the `encoder_output`use the sum operater to rescale to `[seq_len, batch_size, 512]`
- return `encoder_output`to attention, and hidden to decoder.

- Class `Attention`

  The Attention module for the Decoder, only have one parameter named `hidden_size`, 512

  - `v`: the tensor parameter in the model, which can be found in Neural Machine Translation by jointly Learning to align and translate.
  - `Linear`: the attention weights

```
Attention(
    (attn): Linear(in_features=1024, out_features=512, bias=True)
)
```

  Forward process

  `forward(self, hidden, encoder_outputs)`the input parameter is the same as the paper. (`encoder_outputs([seq, batch, hidden_size]`,`hidden([hidden_size])`), Use the `score` function calculate the energy weight of the `encoder_outputs`.

  Score process

  `score(self, hidden, encoder_outputs)`, before the `score` function, the `Forward` process have changed the tensor `hidden`, `encoder_outputs`into `hidden([1(batch), seq, hidden_size])`(by repeat the hidden) and `encoder_outputs([batch, seq, hidden_size]`.)

  The score function `torch.cat`the `hidden`and `encoder_outputs`into the energy vector which size is `[batch, seq, 2 * hidden_size]`. Then use the `attn` Linear matrix and transpose to get the new energy vector `[batch, hidden_size, seq]`.

  And according to the formula in the paper.

$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

  Noted that `v` is the paramter used to change the vector to the weight.

- Class `Decoder`

  1 layer GRU with Attention

  - `embed_size`: 256
  - `hidden_size`: 512
  - `output_size`: 10004
  - `embed`: embed the word from previous tiemstamp to the input in current time unit.
  - `attention`: Attention module above
  - `gru`: noted that the input contain the embedded word and the context vector calculated by the Attention.

```
Decoder(
    (embed): Embedding(10004, 256)
    (dropout): Dropout(p=0.5, inplace)
    (attention): Attention(
        (attn): Linear(in_features=1024, out_features=512, bias=True)
        )
    (gru): GRU(768, 512, dropout=0.5)
    (out): Linear(in_features=1024, out_features=10004, bias=True)
)
```

Forward process

1. embed the current input into `[1, B, N]`, once decode one word.
2. get the attention weights `[1, B, N]`
3. cat the embedded input and attention weight into `[1, B, 512+256]` Tensor.
4. input the hidden state and tensor in `(3)` get the output(hidden state, same)
5. cat the output and context to the `output` Linear matrix and softmax to get the output word in English(10004).
6. return `output([B,N])`, `hidden([1,B,N])`, `atten_weights([B,1,T])`

- Class `Seq2Seq`

1. `__init__(encoder, decoder)`
2. `forward(src, tag, teacher_forcing_ratio=0.5)`
   - The meaning of the `teacher_forcing_ratio` is the possibility using the input word in the training sentence to feed into the decoder.
   - Cet the `max_len` in the target sentence and use this as the length of the generating sentence.
   - return the `outputs([T,B,N])`

3. `train.py`

1. `parse_argument`:
   The function that get the arguments for the training, such as `epochs`, `batch_size`, `learning rate`, `grad_clip`
2. `evaluate`:
   Get the average loss for the evaluation (`val_loss`).
3. `train`:
   Training the module with the train dataset
4. `test_by_human`:
   Test some examples from the train dataset
5. `main`:
   - Check the CUDA
   - Load and analyse the dataset
   - Construct the encoder and decoder and Seq2Seq module
   - Start the epoch iterations for the `train` function to traing the module.
   - If the `val_loss` is the best until now, save it.