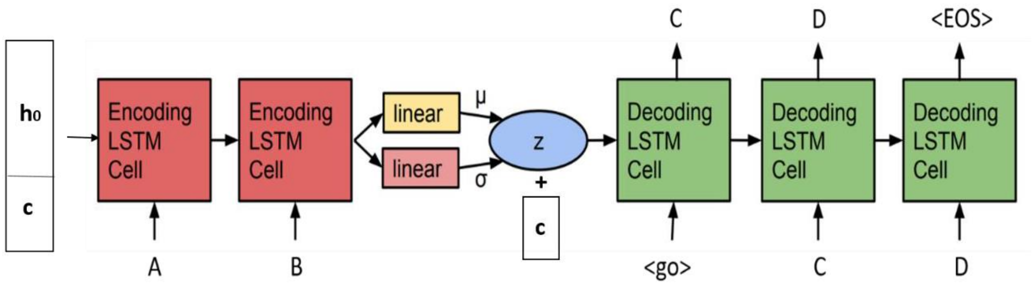
Lab 3: Conditional Sequence-to-sequence VAE

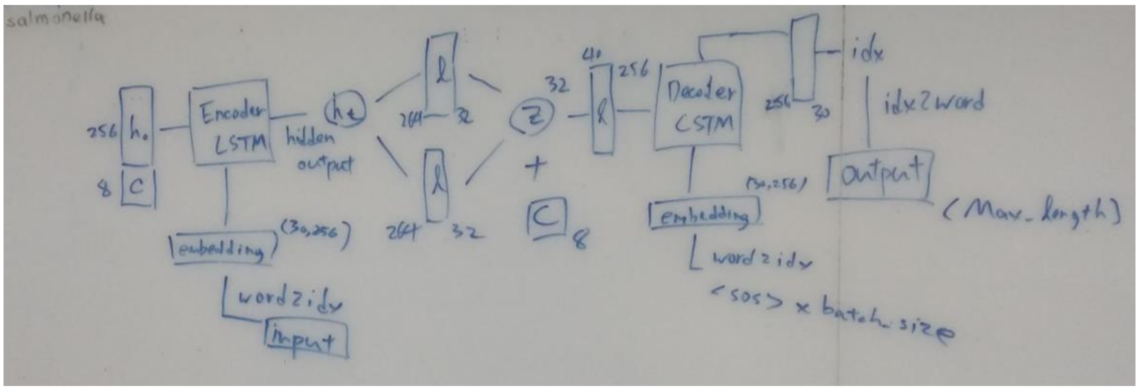
多媒體工程研究所 0756614 林榆軒

·Introduction



When we feed the input word ‘access’ with the tense (the condition) ‘simple present’ to the encoder, it will generate a latent vector z. Then, we feed z with the tense ‘present progressive’ to the decoder and we expect that the output word should be ‘accessing’. In addition, we can also manually generate a Gaussian noise vector and feed it with different tenses to the decoder and generate a word those tenses. The figure above is the overall conditional seq2seq VAE model architecture modified from [Samuel R. Bowman et al. 2016].

·Implementation details

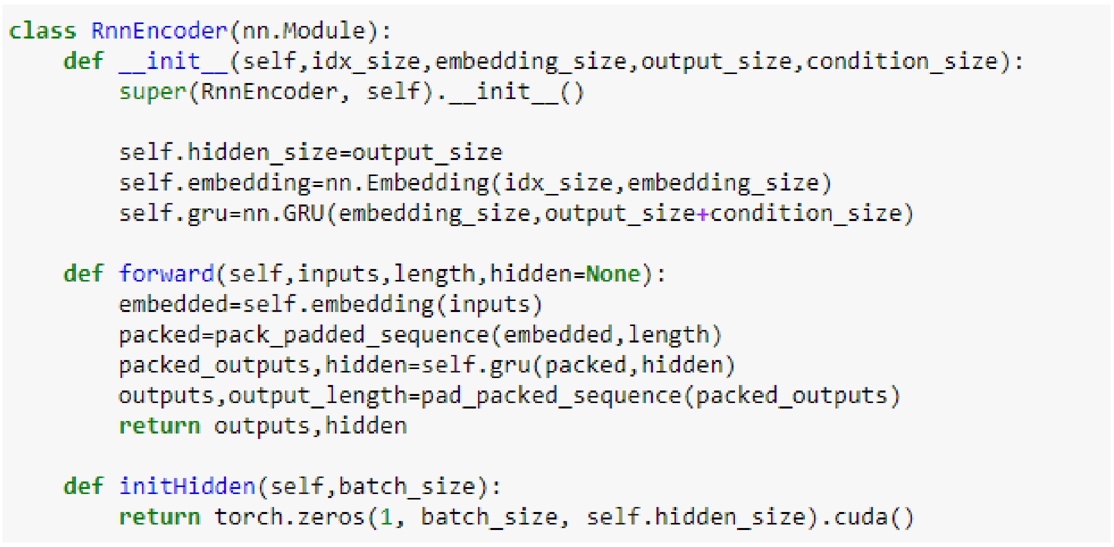


This is the detail of my overall architecture for model, it can divide into three parts:

1. Encoder,
2. Decoder
3. Seq2Seq

【Encoder】

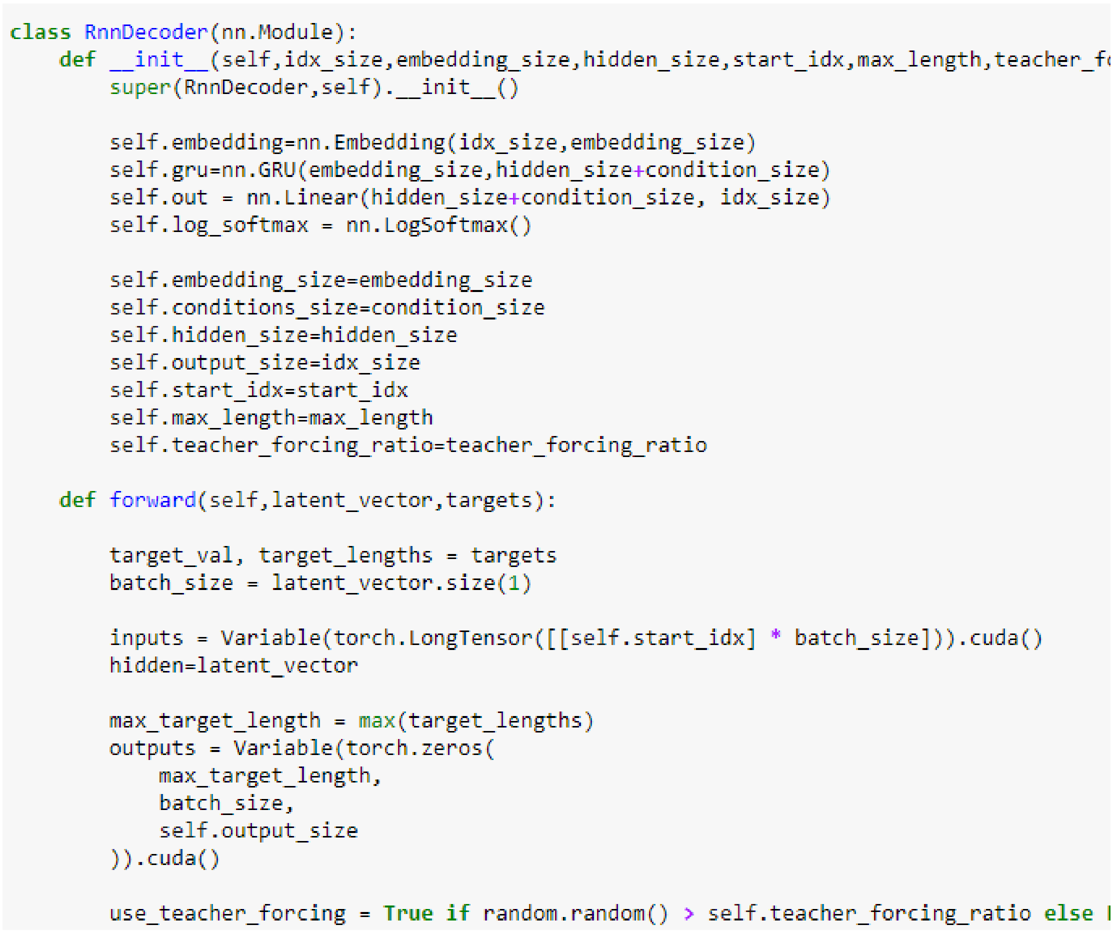
Encoder part is quite sample, only container a single Rnn-layer. The initial of the hidden is a zero tensors concat with embedded conditions.



```python
class RnnEncoder(nn.Module):
    def __init__(self,idx_size,embedding_size,output_size,condition_size):
        super(RnnEncoder, self).__init__()

        self.hidden_size=output_size
        self.embedding=nn.Embedding(idx_size,embedding_size)
        self.gru=nn.GRU(embedding_size,output_size+condition_size)

    def forward(self,inputs,length,hidden=None):
        embedded=self.embedding(inputs)
        packed=pack_padded_sequence(embedded,length)
        packed_outputs,hidden=self.gru(packed,hidden)
        outputs,output_length=pad_packed_sequence(packed_outputs)
        return outputs,hidden

    def initHidden(self,batch_size):
        return torch.zeros(1, batch_size, self.hidden_size).cuda()
```

【Decoder】

Decoder part is a little bit complicated. The forward step is similar to the encoder part except there is a single linear layer at the end and the forward process will be unfold for each time steps in order to predict all the char in the vocabulary word.



```python
class RnnDecoder(nn.Module):
    def __init__(self,idx_size,embedding_size,hidden_size,start_idx,max_length,teacher_f
        super(RnnDecoder,self).__init__()

        self.embedding=nn.Embedding(idx_size,embedding_size)
        self.gru=nn.GRU(embedding_size,hidden_size+condition_size)
        self.out = nn.Linear(hidden_size+condition_size, idx_size)
        self.log_softmax = nn.LogSoftmax()

        self.embedding_size=embedding_size
        self.conditions_size=condition_size
        self.hidden_size=hidden_size
        self.output_size=idx_size
        self.start_idx=start_idx
        self.max_length=max_length
        self.teacher_forcing_ratio=teacher_forcing_ratio

    def forward(self,latent_vector,targets):

        target_val, target_lengths = targets
        batch_size = latent_vector.size(1)

        inputs = Variable(torch.LongTensor([[self.start_idx] * batch_size])).cuda()
        hidden=latent_vector

        max_target_length = max(target_lengths)
        outputs = Variable(torch.zeros(
            max_target_length,
            batch_size,
            self.output_size
        )).cuda()

        use_teacher_forcing = True if random.random() > self.teacher_forcing_ratio else
        for t in range(max_target_length):
            outputs_on_t, hidden = self.forward_step(inputs,hidden)
            outputs[t] = outputs_on_t
            if use_teacher_forcing:
                input = target_val[t].unsqueeze(0)
            else:
                input = self.Decode2Idx(outputs_on_t)

        return outputs, hidden


    def forward_step(self, inputs, hidden):

        batch_size = inputs.size(1)
        embedded = self.embedding(inputs)
        embedded.view(1, batch_size, self.embedding_size)
        output, hidden = self.gru(embedded, hidden)


        output = output.squeeze(0)
        output = self.log_softmax(self.out(output))
        return output, hidden
```

【Seq2Seq】

This part combined Encoder and Decoder together from end to end. Generate the latent vector z from the output hidden layer of encoder and pass it as initial hidden layer (concat with the conditons) of the decoder.

```
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder,hidden_size,condition_idx,condition_size,latent_size,batch_size
        super(Seq2Seq, self).__init__()
        self.encoder = encoder
        self.decoder = decoder

        self.batch_size=batch_size
        self.embedding=nn.Embedding(condition_idx,condition_size)
        self.hidden2mean=nn.Linear(hidden_size+condition_size,latent_size)
        self.hidden2logv=nn.Linear(hidden_size+condition_size,latent_size)
        self.latent2hidden=nn.Linear(latent_size,hidden_size)


    def forward(self, inputs, targets,conditions):

        input_vals, input_lengths = inputs
        c=self.embedding(conditions)
        c=c.view(1,self.batch_size,-1)
        cz=copy.copy(c)
        initHidden=self.encoder.initHidden(batch_size)

        hidden=torch.cat((initHidden,c),2)

        encoder_outputs, encoder_hidden = self.encoder.forward(input_vals, input_lengths,hidden)

        mu=self.hidden2mean(encoder_hidden)
        logvar=self.hidden2logv(encoder_hidden)

        z=self.reparameterize(mu,logvar)
        z=self.latent2hidden(z)

        z=torch.cat((z,cz),2)

        decoder_outputs, decoder_hidden = self.decoder.forward(z, targets)
        return decoder_outputs, decoder_hidden,logvar,mu
```

The reparameterize is using Gaussians to sample the latent vector z

```
def reparameterize(self, mu, logvar):
    std = torch.exp(0.5*logvar).cuda()
    eps = Variable(torch.randn_like(std).cuda())
    return mu + eps*std
```

I didn’t fine tune the hyperparameters for this lab, all the value is exactly same as what the spec has provided.

-hidden size=256

-latent size=32

-condition size=8

-learning rate 0.05

-teacher forcing ratio: 0.5

-epochs: 50

-batch size: 32

-initial beta(KL weight):0

I had try to implement the Monotonic method and Cyclical method, but my model is untrainable after I add the KL loss.

·Results and discussion

【Result】

Sadly, I didn’t make it. I only successful reconstruct the word from the input, but my model is unable to generate the word with different tense. The main reason I fail the lab is because I was too overconfidence to believe that I can finish the whole lab in short time. The coding part is quite easy, while but the fine tune part is really tedious. There are too many hyperparameters must to be fine tune, also the RNN model is much harder to train than the CNN model. I didn’t realize there are many details need to be handle before I start it, I should spend more time on it.

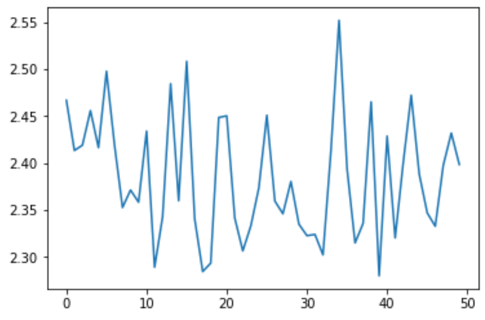
【Discussion】

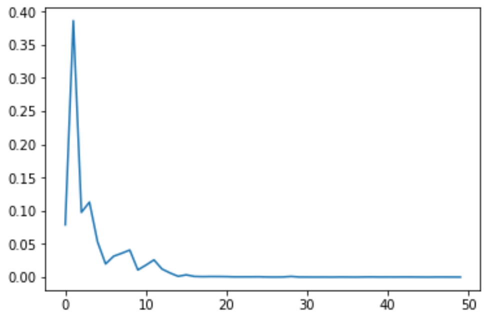
As the mentioned above, my model is able to predict the word from A to A, but is unable to generate a new word from A to B. My model is successfully trained when there are only reconstruction loss, by once I add the KL loss to it, it just complete fail.

A to A reconstruct:

```
fictionalizing    fictiiaaizing
participates      partiiaates
committing        committng
enlighten         enllitteh
revealing         revealing
adjoining         adjonning
fumbles           fumbles
spurred           spurred
pursues           pursues
scrape            scrape
belies            belies
hoped             poped
balks             balks
crash             crash
hears             hears
stole             stole
```

I try to plot the two loss:





The left-hanb side is the reconsturct loss and right hand-side is the KL loss.

It seem like he KL loss part is fine, but the reconsturct loss part is a little bit werid. I done some research for it. This is completely reasonable. When the weight of the KLD loss is large, the model may not be able successfully reconstruct the inuput, in constrast, if the weight is low, the model can clearily reconstruct the input, but tend to be overfitting. So, as the hint spec provided, I try to implement Monotonic method and Cyclical method to overcome this situation, or pre-train the model with reconstruct loss first and then add KL loss, but neither of them works. I guess I my mess up some part of it or maybe I just didn’t choice the right hyperparameters for it. Anyway, I am run out of time to figure it out.