Label 3 :
Conditional Sequence-to-sequence VAE

Outline

1. Introduction

2. Experiment setups

3. Experiment results

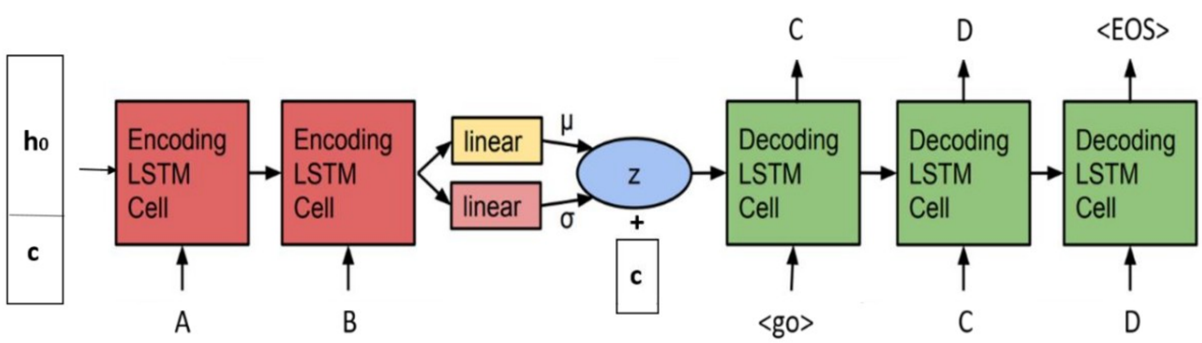
4. Discussion

5. Derivation

1. Introduction

使用 Conditional Variational Autoencoder (C-VAE) 網路,製作將一個單字轉成一串數字, 輸入到 Encoder 搭配特定定義 (型態) , 從 Encoder 輸出後經過平均值, Log 的傅立葉轉換後與高斯分佈 $\mu_2 = 0, \sigma_2^2 = 1$ 運算後與另外一個特定定義 (型態) 一 起輸入到 Decoder, 最後在轉回單字, 做到可以自由轉型態的功能。

網路架構:



更新公式:

$$E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z},c;\boldsymbol{\theta}) - \mathrm{KL}(q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')||p(\boldsymbol{Z}|c))$$

2. Experiment setups

Setups:

RNN: GRU

hidden_szie: RNN 的 hidden 大小。

cond_embed_size: Condition 的 embedding 大小。

linear_output_size: encoder 輸出後接兩個的 linear 輸出大小。

teacher_forcing_ratio: 增強學習效果的係數。

KL_weight: 調整 KL 對更新的影響。

optimizer: SGD 且 momentum = 0.8。

loss: CrossEntropyLoss。

train_cond: 訓練時的順序,且每個區間內都是同個時態的資料。

```
hidden_size = 256
cond_embed_size = 8
input_emb_size = 64
linear_output_size = 32

epoch = 500
learning_rate = 1e-3
teacher_forcing_ratio = 1
```

```
def get_KL_Weight(epoch):
    if annealingType == 0:
        return 0.02
```

```
optimizer = optim.SGD(model.parameters(), lr = learning_rate,  momentum = 0.8)
criterion = nn.CrossEntropyLoss()
train_cond = ['SP', 'TP', 'PP', 'PS']
```

Encoder: 為了將 hidden 得長度一致在設定的數值, 所以在紅框的程式碼中將 Conditional 與 hidden 合併, 以及綠框中 hidden 初始化的時候並非全 0, 而是高斯分佈。

```python
class Encoder(nn.Module):

    def __init__(self, input_size, input_emb_size, hidden_size, cond_embed_size):
        super(Encoder, self).__init__()

        self.vocab_size = input_size
        self.hidden_size = hidden_size
        self.input_emb_size = input_emb_size
        self.cond_embed_size = cond_embed_size

        self.embedding = None
        self.gru = nn.GRU(input_emb_size, self.hidden_size)

    def setEmbedding(self, embedding):
        self.embedding = embedding

    def forward(self, inputData, inputData_lengths, hidden):
        embedded = self.embedding(inputData).view(1, 1, -1)
        outputs, hidden= self.gru(embedded, hidden)
        return outputs, hidden

    def initHidden(self, cond_ver, use_cuda):
        hidden = torch.randn(1, 1, self.hidden_size)
        if use_cuda:
            hidden = hidden.cuda()
        hidden[:, :, self.hidden_size - self.cond_embed_size:] = cond_ver
        return hidden
```

Decoder: 運用兩個不同的 Activation 增加學習效果。

```python
class Decoder(nn.Module):
    def __init__(self, hidden_size, input_emb_size, output_size):
        super(Decoder, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.embedding = None

        self.gru = nn.GRU(input_emb_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, output_size)
        self.activation = nn.ELU()
        self.activation2 = nn.LogSoftmax(dim = 1)

    def setEmbedding(self, embedding):
        self.embedding = embedding

    def forward(self, inputs, hidden):
        output = self.embedding(inputs).view(1, 1, -1)
        output = self.activation (output)
        output, hidden = self.gru(output, hidden)
        output = self.out(output[0])
        output = self.activation2(output)
        return output, hidden
```

CVAE: 輸入的文字與 Condition 的 embedding 各一個, 且 encoder, decoder 共用, 和要給 decoder 的 hidden 會多進一次 linear。

```python
class CVAE(nn.Module):

    def __init__(self, encoder, decoder, linear_output_size):
        super(CVAE, self).__init__()
        self.encoder = encoder
        self.decoder = decoder

        self.word_embeddg = nn.Embedding(self.encoder.vocab_size, self.encoder.input_emb_size)
        self.con_embeddg = nn.Embedding(4, self.encoder.cond_embed_size) # 4 condittional

        self.encoder.setEmbedding(self.word_embeddg)
        self.decoder.setEmbedding(self.word_embeddg)

        self.MN = nn.Linear(self.encoder.hidden_size, linear_output_size)
        self.LG  = nn.Linear(self.encoder.hidden_size, linear_output_size)
        self.latent_to_decoder_input = nn.Linear(linear_output_size + self.encoder.cond_embed_size, self.encoder.hidden_size)
```

Vocabulary: 定義字元以及收資料和轉換字元等腳本。

```python
class Vocabulary(object):

    def __init__(self):
        self.char2idx = {'SOS': 0, 'EOS': 1}
        self.idx2char = {0: 'SOS', 1: 'EOS'}
        self.conditional = {'SP': 0, 'TP' : 1, 'PP' : 2, 'PS' : 3}
        self.num_chars = 2
        self.max_length = 0
        self.word_list = {'SP':[], 'TP':[], 'PP':[], 'PS':[]}
        self.test_data_list = []
        self.test_label_list = []

    def build_vocab(self, data_path, test_path):
        vocab = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
                 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

        for char in vocab:
            self.char2idx[char] = self.num_chars
            self.idx2char[self.num_chars] = char
            self.num_chars += 1
```

```python
def sequence_to_indices(self, sequence):
    index_sequence = []

    for char in self.split_sequence(sequence):
        index_sequence.append(self.char2idx[char])

    return index_sequence

def indices_to_sequence(self, indices):
    sequence = ""

    for idx in indices:
        char = self.idx2char[int(idx)]

        if char == "EOS":
            break
        else:
            sequence += char

    return sequence
```

DataTransformer: 分配資料與組合的腳本。

```python
class DataTransformer(object):

    def __init__(self, data_path, test_path, use_cuda):
        self.indices_sequences = {'SP':[], 'TP':[], 'PP':[], 'PS':[]}
        self.use_cuda = use_cuda

        self.vocab = Vocabulary()
        self.vocab.build_vocab(data_path, test_path)
        self.SOS_ID = self.vocab.char2idx['SOS']
        self.EOS_ID = self.vocab.char2idx['EOS']
        self.vocab_size = self.vocab.num_chars

        if data_path != None:
            self._build_training_set()

        self.EOS = Variable(torch.LongTensor([ self.EOS_ID  ] ))
        if use_cuda:
            self.EOS = self.EOS.cuda()

    def _build_training_set(self):
        for cond, word_list in self.vocab.word_list.items():
            for word in word_list:
                data_seq = self.vocab.sequence_to_indices(word)
                self.indices_sequences[cond].append(data_seq)

    def mini_batches(self, cond):
        np.random.shuffle(self.indices_sequences[cond])
        for batch in self.indices_sequences[cond]:
            input_var = torch.LongTensor(batch)
            if self.use_cuda:
                input_var = input_var.cuda()
            yield (input_var, len(input_var))
```

```python
def get_Test_data(self):
    cond_encode_ver =['SP', 'SP', 'SP', 'SP', 'PS', 'SP', 'PS', 'PP', 'PP', 'PP']
    cond_decode_ver =['PS', 'PP', 'TP', 'TP', 'TP', 'PP', 'SP', 'SP', 'PS', 'TP']
    return self.vocab.test_data_list, self.vocab.test_label_list, (cond_encode_ver, cond_decode_ver)

def pad_sequence(self, sequence, max_length):
    sequence += [self.PAD_ID for i in range(max_length - len(sequence))]
    return sequence

def evaluation_batch(self, words):
    evaluation_batch = []
    indices_seq = self.vocab.sequence_to_indices(words)
    indices_seq = torch.LongTensor(indices_seq)

    if self.use_cuda:
        indices_seq = indices_seq.cuda()

    return indices_seq, len(indices_seq)

def getConditional(self, cond):
    cond_var= Variable(torch.LongTensor([ [ self.vocab.conditional[cond] ] ] ))
    if self.use_cuda:
            cond_var = cond_var.cuda()

    return cond_var

def combineEOS(self, data):
    return torch.cat((data, self.EOS), 0)
```

Train:

```python
for e in range(1, epoch + 1):
    model.changeToTrainMode()
    times = 0
    loss_record = 0
    KL_record = 0
    BLEU_record = 0

    for cond in train_cond:
        mini_batches = dataTransformer.mini_batches(cond)
        for input_batch in mini_batches:
            times += 1

            optimizer.zero_grad()
            cond_var = dataTransformer.getConditional(cond)

            mean, log, hidden = model.doEncoder(input_batch, cond_var)
            outputs = model.doDecoder(hidden, input_batch, cond_var)

            loss, KL = get_loss(criterion, outputs, dataTransformer.combineEOS(input_batch[0]), mean, log)

            loss_record += float(loss)
            KL_record += float(KL)

            loss += KL * get_KL_Weight(e)

            loss.backward()
            optimizer.step()
```

```python
def doEncoder(self, inputs, cond_vars):
    input_vars, input_lengths = inputs
    con_emb = self.con_embeddg(cond_vars)
    hidden = self.encoder.initHidden(con_emb, self.use_cuda)
    for t in range(input_lengths):
        output, hidden = self.encoder(input_vars[t], input_lengths, hidden)

    mean = self.MN(output)
    log = self.LG(output)

    return mean, log, self.reparameterize(mean, log)

def reparameterize(self, mean, log):
    ep = torch.exp(log)
    gussion = torch.randn_like(ep)
    if self.use_cuda:
        gussion = gussion.cuda()
    return ep.mul(gussion).add(mean)
```

```python
def doDecoder(self, inputs, targets, cond_vars, train = True):
    if targets != None:
        target_vars, target_lengths = targets
    use_teacher_forcing = (random.random() < self.teacher_forcing_ratio)

    con_emb = self.con_embeddg(cond_vars)
    hidden = torch.cat((inputs, con_emb), 2)
    hidden = self.latent_to_decoder_input(hidden)

    token = torch.LongTensor([self.sos_id])

    length = self.encoder.vocab_size
    if train:
        length = target_lengths + 1

    decoder_outputs = torch.ones(length, self.decoder.output_size)

    if self.use_cuda:
        token = token.cuda()
        decoder_outputs = decoder_outputs.cuda()

    if train and use_teacher_forcing:
        token = torch.cat((token, target_vars), 0)
        for t in range(0, length):
            x = token[t]
            outputs, hidden = self.decoder(x, hidden)
            decoder_outputs[t] = outputs
    else:
        for t in range(0, length):
            outputs, hidden = self.decoder(token, hidden)
            decoder_outputs[t] = outputs

            topv, topi = outputs.topk(1)
            token = topi.squeeze().detach()
            if token == self.eos_id:
                break

    return decoder_outputs
```

Loss:

```
def get_loss(criterion, outputs, targets, mean, log):
    loss = criterion(outputs, targets)
    KL = (-0.5) * torch.sum(1 + 2 * log - mean.pow(2) - log.exp().pow(2))
    return loss, KL
```

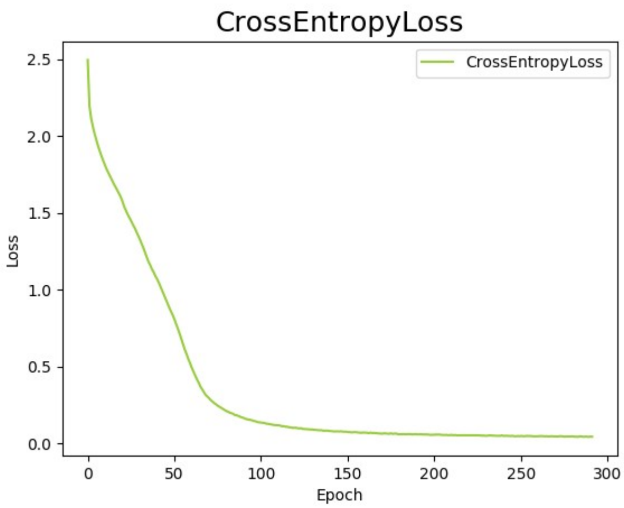
Eval:

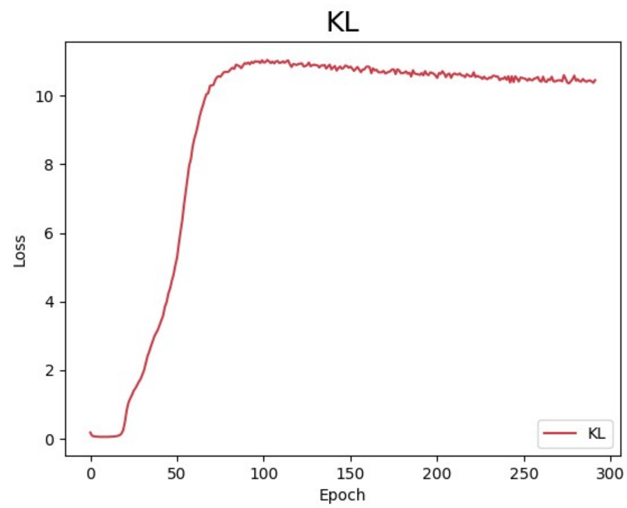
```
result = ''
BLEU = 0
for i in range(len(test_data)):
    test_data_var = dataTransformer.evaluation_batch(test_data[i])
    cond_en_var = dataTransformer.getConditional(cond_encode_ver[i])
    cond_de_var = dataTransformer.getConditional(cond_decode_ver[i])

    predic = model.evaluation( test_data_var, cond_en_var, cond_de_var)
    predic = dataTransformer.vocab.indices_to_sequence(predic[0])
    result += str(predic) +" : " +str(test_label[i]) +"\n"
    BLEU += float(compute_bleu(predic, test_label[i]))

BLEU /= len(test_data)
```

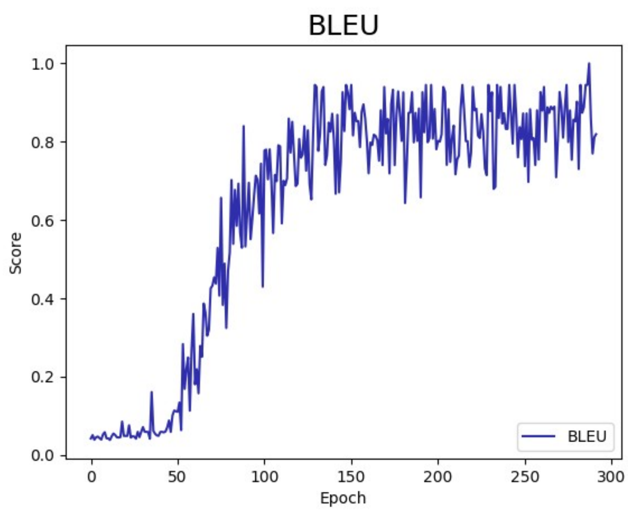
```
def evaluation(self, inputs, cond_encoder_vars, cond_decoder_vars):
    mean, log, encoder_hidden = self.doEncoder(inputs, cond_encoder_vars)
    decoder_outputs = self.doDecoder(encoder_hidden, inputs, cond_decoder_vars, False)
    return self._decode_to_index(decoder_outputs)
```

3. Experiment results

Training situation：







轉型測試：測試的最佳結果, 準確度最高到 **1.0**, 平均落在 0.909 (100Round)

```
abandoned : abandoned
abetting : abetting
begins : begins
expends : expends
sends : sends
splitting : splitting
flare : flare
function : function
functioned : functioned
heals : heals
abandon : abandon
coincide : coincide
depended : depended

BLEU: max: 1.0, average: 0.9089745897467361
```

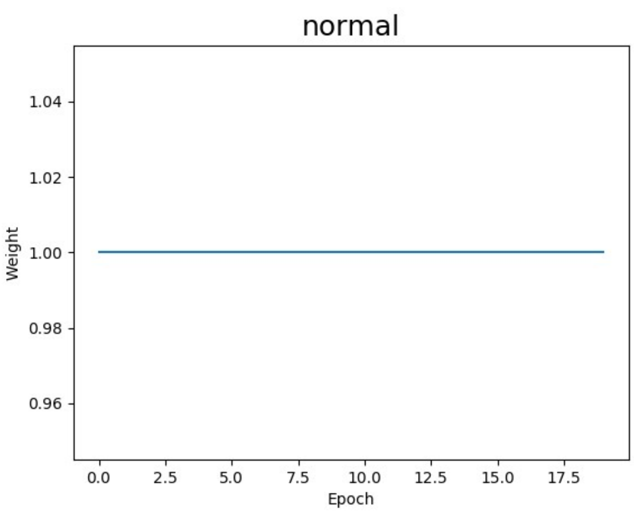
Gaussian 測試：測試 100 組不同的 gaussian noise, 準確度最高到 **0.54**, 平均落在 0.415 (100Round)

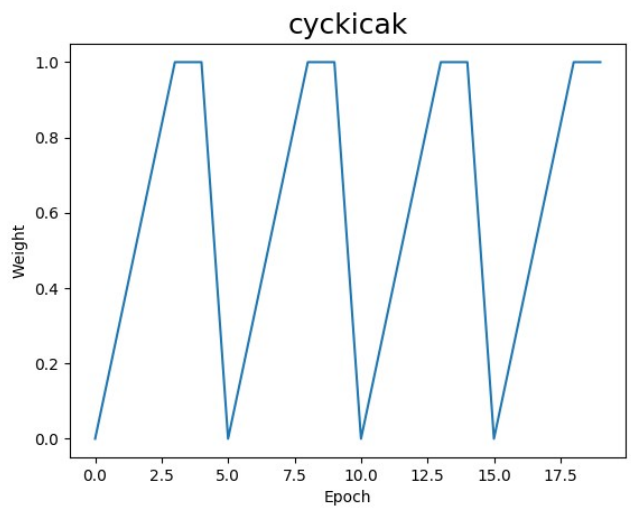
```
['fester', 'festers', 'festering', 'festered'] : 1.0
['sidle', 'sidles', 'sidling', 'sidled'] : 1.0
['skip', 'skips', 'skipping', 'skipped'] : 1.0
['presume', 'presumes', 'presuming', 'presumed'] : 1.0

Gaussian_score: max: 0.54, average: 0.41459999999999999
```

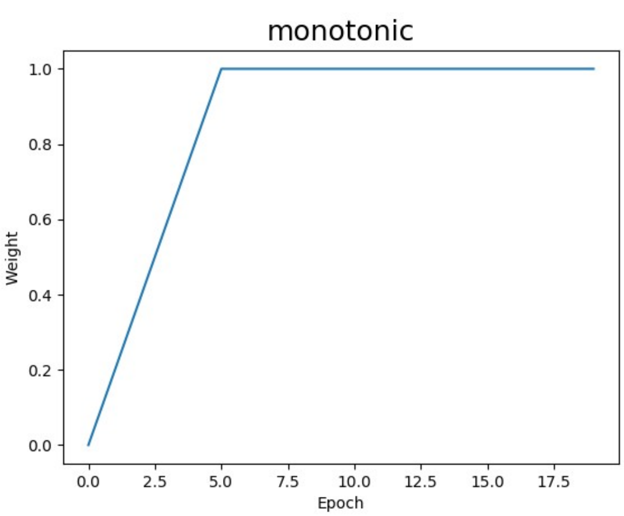
4. Discussion

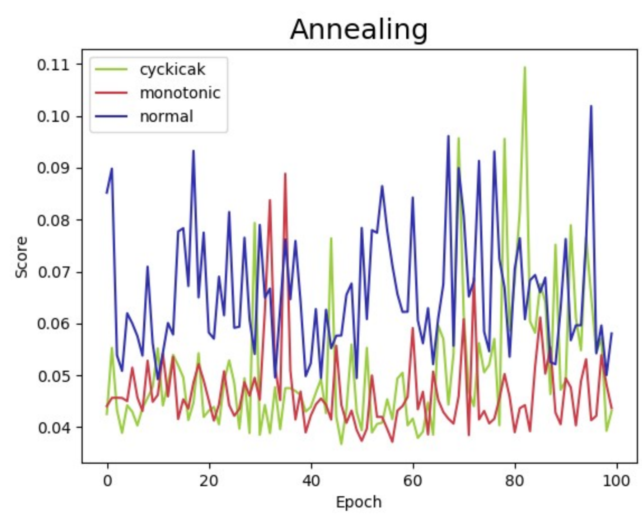
這部份的訓練結果, 會以 BLEU 搭配轉型測試為主要衡量標準。

Annealing: 三種不同的 KL weight 對訓練的情形, normal 的表現比較好。

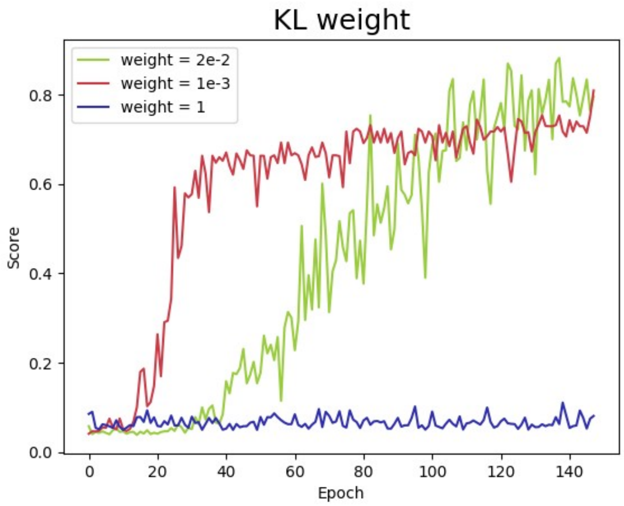




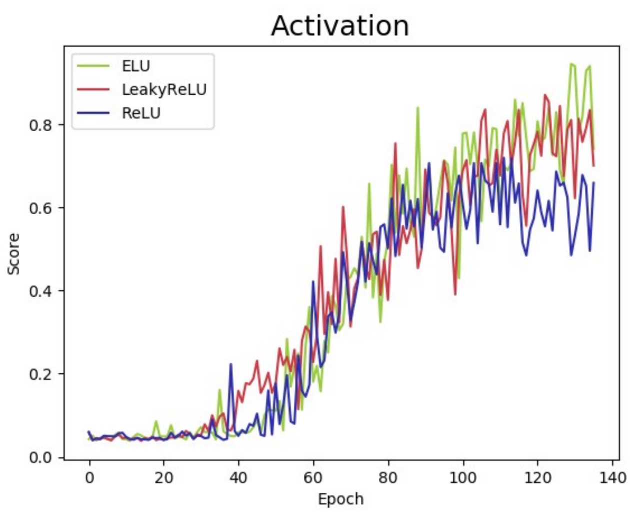




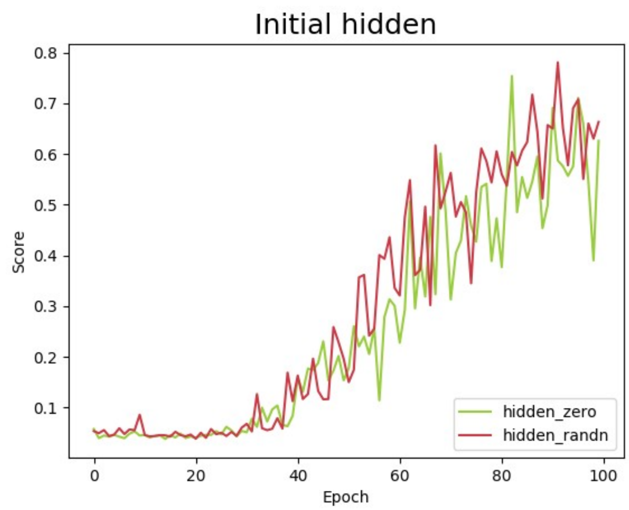
KL weight: 分別調整固定得 weight 為 1, 0.02, 0.001, 雖然 0.001 前面狀況非常好, 但就一直維持在 0.7 左右排回, 0.02 後面直接超越 0.001,甚至直接滿分。



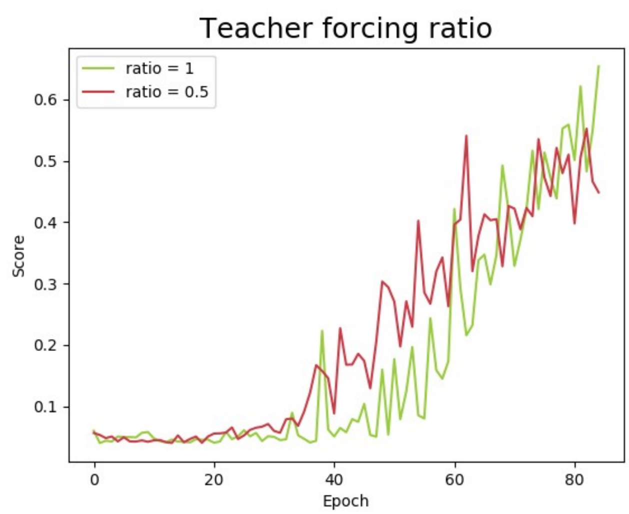
Activation of decoder: 將 decoder 中的變數 **self.activation** 換成 ReLU, LeakyReLU, ELU 做測試, ELU 表現良好。



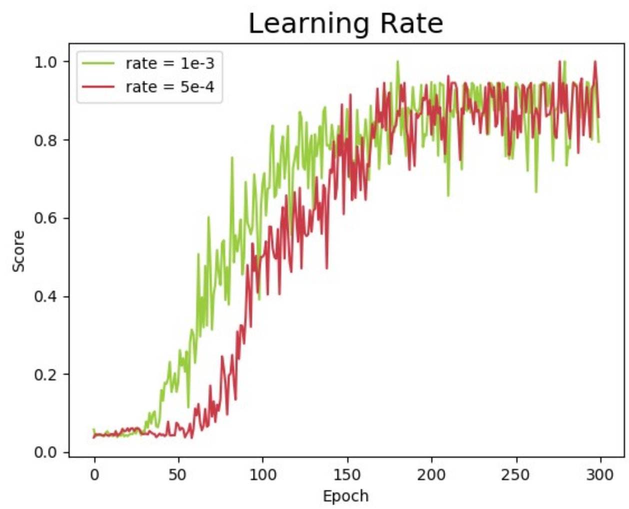
Initial hidden: hidden 得初始化分別為全 0 或高斯分佈, 高斯的會比練 0 的更好更快。



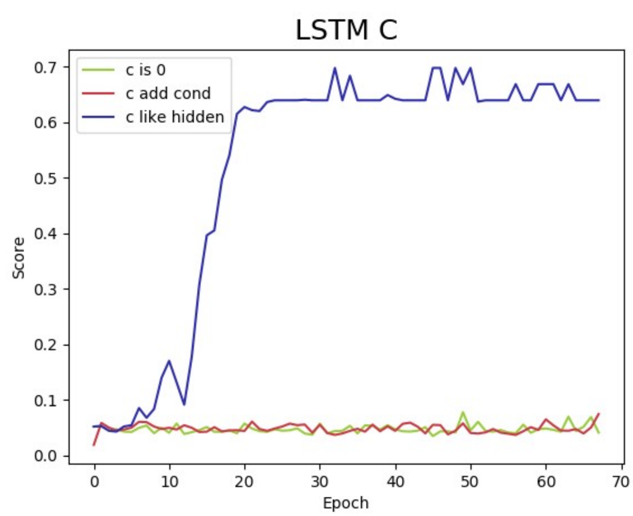
Teacher forcing ratio: 比率分別是 1 與 0.5, 0.5 雖然練的快, 可是很不穩定, 且沒有辦法衝到一個高分。



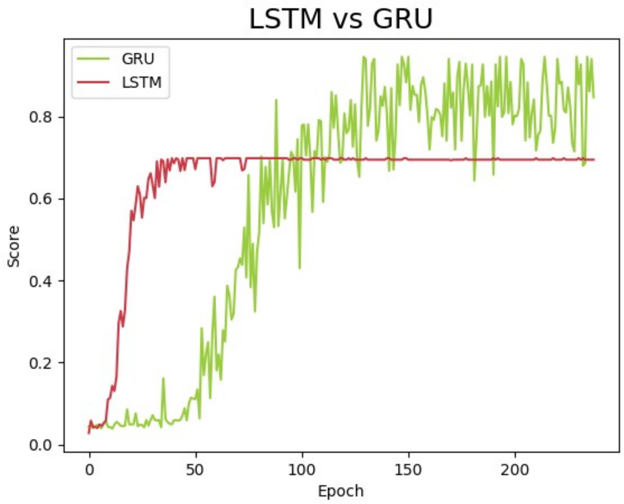
Learning rate: 分別是 5e-4, 1e-3, 顯然 1e-3 效果比較好。



LSTM: 分別試了三種方式針對 LSTM 的 c 參數做設計, 分別是 1. 初始皆為 0, 2. c 與 hidden 一樣最後 8 個資料換成 condition, 3. decoder 的 c 沿用 encoder 的 c 並與 hidden 一樣會做 linear 等轉換, 發現像 hidden 一樣的效果很好。



LSTM vs GRU: LSTM 雖然學很快, 但是不像 GRU 可以有更好的表現, 但 LSTM 在覆現結果上會比 GRU 穩定很多。



5. Derivation

DL Label 3作業推導

2019年8月10日 星期六　　下午8:34

RNN – BPTT

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}, \quad o^{(t)} = c + Vh^{(t)}$$

$$h^{(t)} = \tanh(a^{(t)}), \quad \hat{y}^{(t)} = \mathrm{softmax}(o^{(t)})$$

$$\sigma(x) = \tanh(x) \Rightarrow \sigma'(x) = 1 - \tanh^2(x), \quad H^{(t)} = \mathrm{diag}[\sigma'(h^{(t)})]$$

$$\nabla_{o^{(t)}} L = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \cdot \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = 1 \cdot \frac{\partial -\log \hat{y}_i^{(t)}}{\partial o_i^{(t)}}$$

$$= -\frac{\partial \log \mathrm{softmax}(o_i^{(t)})}{\partial o_i^{(t)}} = -\frac{1}{\hat{y}_i^{(t)}} \cdot \hat{y}_i^{(t)}(1 - \hat{y}_i^{(t)}) = \hat{y}_i^{(t)} - 1$$

$$\nabla_{h^{(t)}} L = V^T \nabla_{o^{(t)}} L = \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}}\right)^T (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial o^{(t)}}{\partial h^{(t)}}\right)^T (\nabla_{o^{(t)}} L)$$

$$= W^T \mathrm{diag}[\sigma'(h^{(t+1)})](\nabla_{h^{(t+1)}} L) + V^T(\nabla_{o^{(t)}} L)$$

$$= W^T \mathrm{diag}(1 - (h^{(t+1)})^2)(\nabla_{h^{(t+1)}} L) + V^T(\nabla_{o^{(t)}} L)$$

$$= W^T H^{(t+1)}(\nabla_{h^{(t+1)}} L) + V^T(\nabla_{o^{(t)}} L)$$

$$\nabla_c L = \sum_t \left(\frac{\partial o^{(t)}}{\partial c}\right)^T (\nabla_{o^{(t)}} L) = \sum_t \nabla_{o^{(t)}} L$$

$$\nabla_b L = \sum_t \left(\frac{\partial h^{(t)}}{\partial b}\right)^T \nabla_{h^{(t)}} L = \sum_t \mathrm{diag}(\sigma'(h^{(t)})) \nabla_{h^{(t)}} L$$

$$= \sum_t \mathrm{diag}(1 - (h^{(t)})^2)(\nabla_{h^{(t)}} L) = \sum_t H^{(t)}(\nabla_{h^{(t)}} L)$$

$$\nabla_V L = \sum_t \left(\frac{\partial o^{(t)}}{\partial V}\right) \nabla_{o^{(t)}} L = \sum_t (\nabla_{o^{(t)}} L) \cdot (h^{(t)})^T$$

$$\nabla_W L = \sum_t \frac{\partial L}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial W} = \sum_t \mathrm{diag}(1 - (h^{(t)})^2)(\nabla_{h^{(t)}} L)(h^{(t)})^T$$

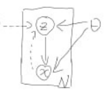
$$= \sum_t H^{(t)}(\nabla_{h^{(t)}} L)(h^{(t)})^T$$

$$\nabla_U L = \sum_t \frac{\partial L}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial U} = \sum_t \mathrm{diag}[1 - (h^{(t)})^2](\nabla_{h^{(t)}} L)(h^{(t)})^T$$

$$= \sum_t H^{(t)}(\nabla_{h^{(t)}} L)(h^{(t)})^T$$

VAE loss

$$\log p(x_1, x_2 \cdots x_N) = \sum_{i=1}^{N} \log p(x_i)$$



$$\log p(x_i) = \int_z q_\phi(z|x_i)\,dz \cdot \log p(x_i)$$

$$= \int_z q_\phi(z|x_i) \cdot \log \frac{p(x_i) \cdot q_\phi(z|x_i) \cdot p_\theta(z|x_i)}{q_\phi(z|x_i) \cdot p_\theta(z|x_i)}\,dz$$

$$= \int_z q_\phi(z|x_i) \cdot \log \frac{p_\theta(x_i, z) \cdot q_\phi(z|x_i)}{q_\phi(z|x_i) \cdot p_\theta(z|x_i)}\,dz$$

$$= \int_z q_\phi(z|x_i) \cdot \log \frac{q_\phi(z|x_i)}{p_\theta(z|x_i)}\,dz + \underline{\int_z q_\phi(z|x_i) \cdot \log \frac{p_\theta(x_i, z)}{q_\phi(z|x_i)}\,dz}$$

$$= D_{KL}(q_\phi(z|x_i) \,\|\, p_\theta(z|x_i)) + \underline{L(\theta, \phi : x_i)}$$

$$L(\theta, \phi; x_i) = \int_z q_\phi(z|x_i) \cdot \log \frac{p_\theta(x_i, z)}{q_\phi(z|x_i)}\,dz$$

$$= -\int_z q_\phi(z|x_i) \cdot \log \frac{q_\phi(z|x_i)}{p_\theta(x_i, z)}\,dz$$

$$= -\int_z q_\phi(z|x_i) \cdot \log \frac{q_\phi(z|x_i)}{p_\theta(x_i|z)\,p_\theta(z)}\,dz$$

$$= -\int_z q_\phi(z|x_i) \cdot \log \frac{q_\phi(z|x_i)}{p_\theta(z)} + \int_z q_\phi(z|x_i) \cdot \log p_\theta(x_i|z)\,dz$$

$$= -D_{KL}(q_\phi(z|x_i) \,\|\, p_\theta(z)) + E_{q_\phi(z|x_i)}[\log p_\theta(x_i|z)]$$

$$N(z; 0, I) \Rightarrow \int_z q_\phi(z) \log p_\theta(z)\,dz = \int N(z; \mu, \sigma) \log(z; 0, I)\,dz$$

$$= \frac{-J}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^{J}(\mu_i^2 + \sigma_i^2)$$

$$\Rightarrow \int_z q_\phi(z) \log q_\phi(z)\,dz = \int N(z; \mu, \sigma) \log(z : \mu, \sigma)\,dz$$

$$= \frac{-J}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^{J}(1 + \log\sigma_i^2)$$

$$-D_{KL}(q_\phi(z|x_i) \,\|\, p_\theta(z)) = \frac{1}{2}\sum_{i=1}^{J}(1 + \log\sigma_i^2 - \mu_i^2 - \sigma_i^2)$$

$$E_{q_\phi(z|x_i)}[\log p_\theta(x_i|z)] = \frac{1}{\chi}\sum_{l=1}^{\chi}\log p_\theta(x_i|z_i^l)$$

$$\chi = 1 \Rightarrow E_{q_\phi(z|x_i)}[\log p_\theta(x_i|z)] = \log_{p_\theta}(x_i|z_i)$$

$$L(\theta, \phi; x_i) = \frac{1}{2}\sum_{i=1}^{J}(1 + \log\sigma_i^2 - \mu_i^2 - \sigma_i^2) + \log p_\theta(x_i|z_i)$$

$$\log P(x_1, x_2 \cdots x_N) = \sum_{i=1}^{N}\log P(x_i) \geq \frac{N}{M}\sum_{i=1}^{M} L(\theta, \phi; x_i)$$