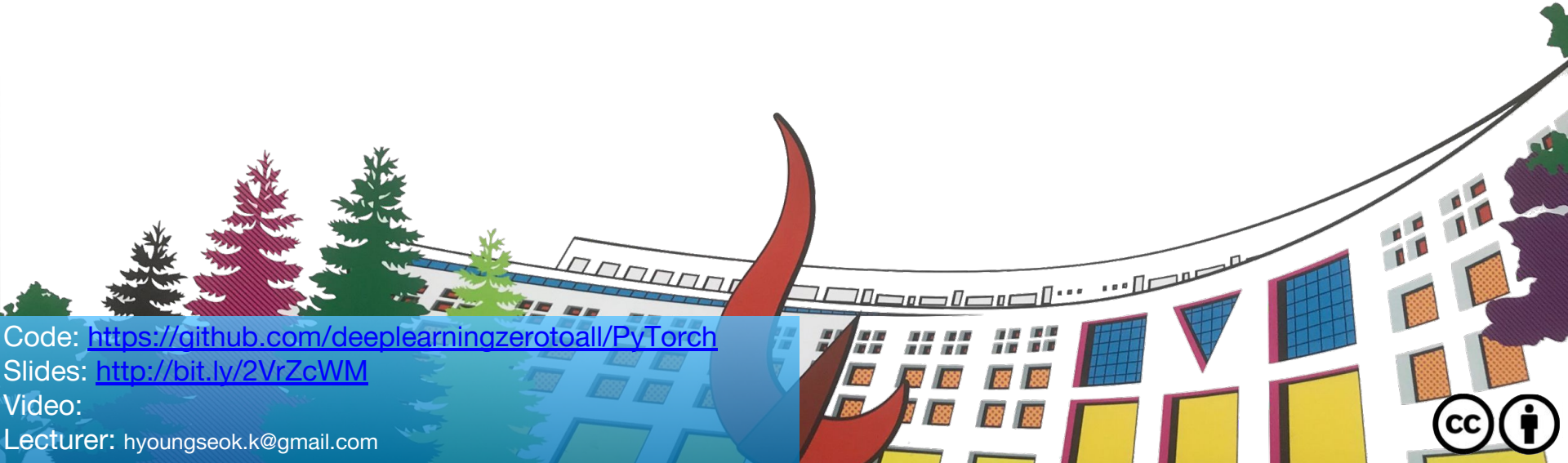
ML/DL for Everyone Season2

with PYTORCH

RNN - Seq2Seq



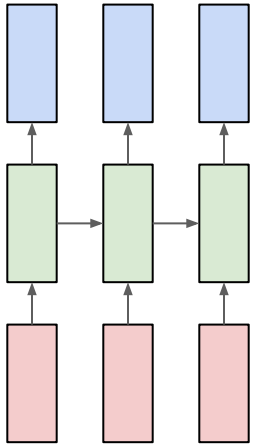
Code: https://github.com/deeplearningzerotoall/PyTorch
Slides: http://bit.ly/2VrZcWM
Video:
Lecturer: hyoungseok.k@gmail.com

RNN - Seq2Seq

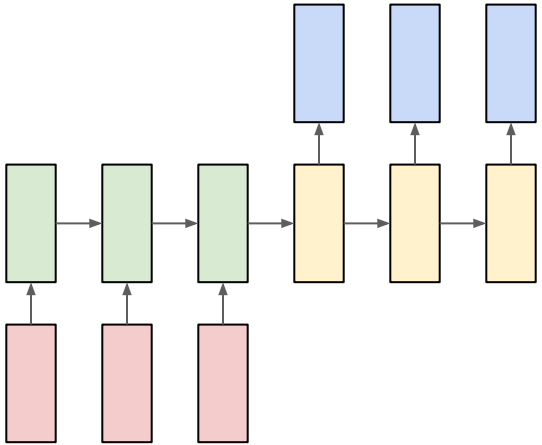
- Seq2Seq
- Apply Seq2Seq
  - Encoder - Decoder
  - Data Preprocessing
  - Neural Net Setting
  - Training
  - Evaluation

Seq2Seq

What is the difference between general RNN model and Seq2Seq model?

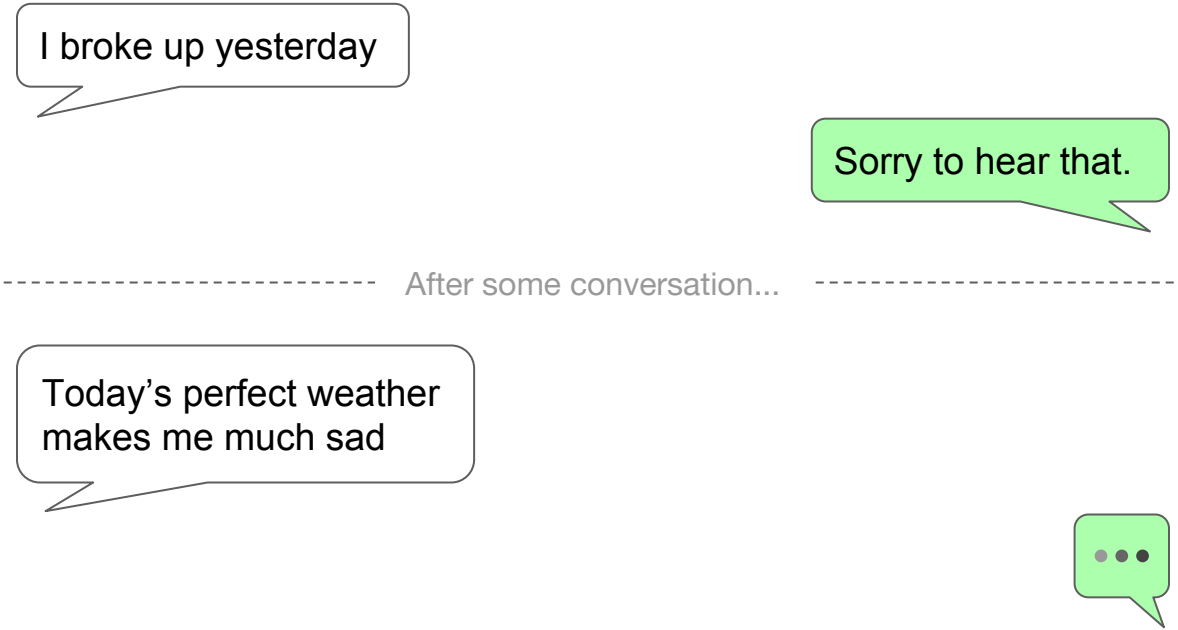


RNN

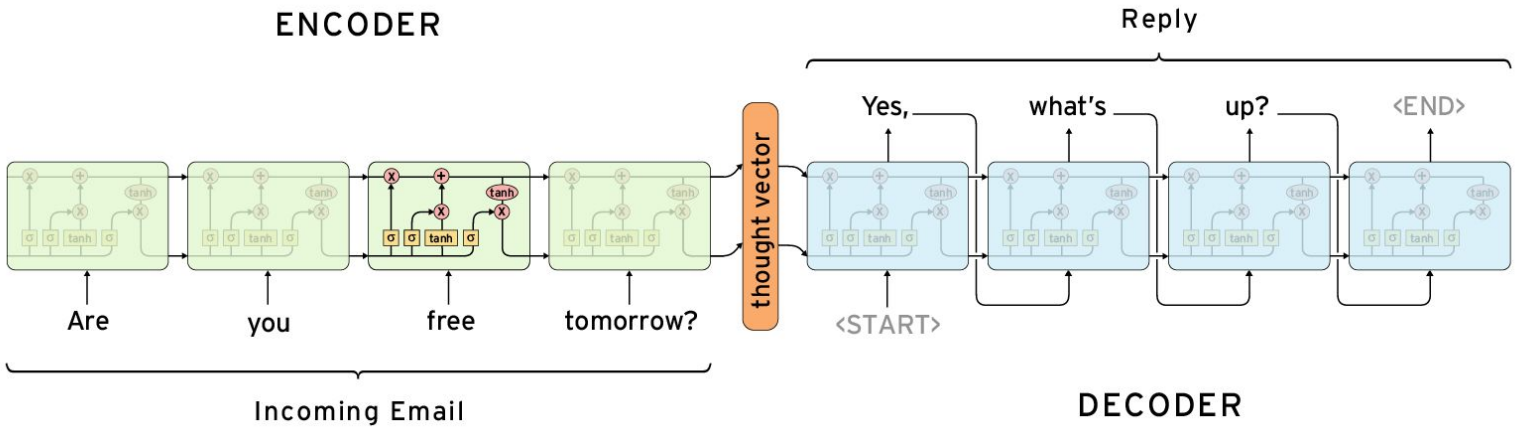


Seq2Seq

Example : Chatbot



Apply Seq2Seq : Encoder-Decoder



https://towardsdatascience.com/sequence-to-sequence-tutorial-4fde3ee798d8

Apply Seq2Seq : Encoder-Decoder

```
4   import random
5   import torch
6   import torch.nn as nn
7   import torch.optim as optim
```

```
184 SOURCE_MAX_LENGTH = 10
185 TARGET_MAX_LENGTH = 12
186 load_pairs, load_source_vocab, load_target_vocab = preprocess(raw, SOURCE_MAX_LENGTH, TARGET_MAX_LENGTH)
187 print(random.choice(load_pairs))
188
189 enc_hidden_size = 16
190 dec_hidden_size = enc_hidden_size
191 enc = Encoder(load_source_vocab.n_vocab, enc_hidden_size).to(device)
192 dec = Decoder(dec_hidden_size, load_target_vocab.n_vocab).to(device)
193
194 train(load_pairs, load_source_vocab, load_target_vocab, enc, dec, 5000, print_every=1000)
195 evaluate(load_pairs, load_source_vocab, load_target_vocab, enc, dec, TARGET_MAX_LENGTH)
```

Apply Seq2Seq : Data Preprocessing

```
4  import random
5  import torch
6  import torch.nn as nn
7  import torch.optim as optim
8
9  torch.manual_seed(0)
10 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
11
12 raw = ["I feel hungry.     나는 배가 고프다.",
13        "Pytorch is very easy.     파이토치는 매우 쉽다.",
14        "Pytorch is a framework for deep learning.     파이토치는 딥러닝을 위한 프레임워크이다.",
15        "Pytorch is very clear to use.   파이토치는 사용하기 매우 직관적이다."]
16
17 SOS_token = 0
18 EOS_token = 1
19
20
```

Apply Seq2Seq : Data Preprocessing

```
43  def preprocess(corpus, source_max_length, target_max_length):
44      print("reading corpus...")
45      pairs = []
46      for line in corpus:
47          pairs.append([s for s in line.strip().lower().split("\t")])
48      print("Read {} sentence pairs".format(len(pairs)))
49
50      pairs = [pair for pair in pairs if filter_pair(pair, source_max_length, target_max_length)]
51      print("Trimmed to {} sentence pairs".format(len(pairs)))
52
53      source_vocab = Vocab()
54      target_vocab = Vocab()
55
56      print("Counting words...")
57      for pair in pairs:
58          source_vocab.add_vocab(pair[0])
59          target_vocab.add_vocab(pair[1])
60      print("source vocab size =", source_vocab.n_vocab)
61      print("target vocab size =", target_vocab.n_vocab)
62
63      return pairs, source_vocab, target_vocab
64
65
```

Apply Seq2Seq : Neural Net Setting

```
66  class Encoder(nn.Module):
67      def __init__(self, input_size, hidden_size):
68          super(Encoder, self).__init__()
69          self.hidden_size = hidden_size
70          self.embedding = nn.Embedding(input_size, hidden_size)
71          self.gru = nn.GRU(hidden_size, hidden_size)
72
73      def forward(self, x, hidden):
74          x = self.embedding(x).view(1, 1, -1)
75          x, hidden = self.gru(x, hidden)
76          return x, hidden
77
78
```

Apply Seq2Seq : Neural Net Setting

```
79  class Decoder(nn.Module):
80      def __init__(self, hidden_size, output_size):
81          super(Decoder, self).__init__()
82          self.hidden_size = hidden_size
83          self.embedding = nn.Embedding(output_size, hidden_size)
84          self.gru = nn.GRU(hidden_size, hidden_size)
85          self.out = nn.Linear(hidden_size, output_size)
86          self.softmax = nn.LogSoftmax(dim=1)
87
88      def forward(self, x, hidden):
89          x = self.embedding(x).view(1, 1, -1)
90          x, hidden = self.gru(x, hidden)
91          x = self.softmax(self.out(x[0]))
92          return x, hidden
93
94
```

Apply Seq2Seq : Training

```
95  def tensorize(vocab, sentence):
96      indexes = [vocab.vocab2index[word] for word in sentence.split(" ")]
97      indexes.append(vocab.vocab2index["<EOS>"])
98      return torch.Tensor(indexes).long().to(device).view(-1, 1)
99
100
101 def train(pairs, source_vocab, target_vocab, encoder, decoder, n_iter, print_every=1000, learning_rate=0.01):
102     loss_total = 0
103
104     encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
105     decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
106
107     training_batch = [random.choice(pairs) for _ in range(n_iter)]
108     training_source = [tensorize(source_vocab, pair[0]) for pair in training_batch]
109     training_target = [tensorize(target_vocab, pair[1]) for pair in training_batch]
110
111     criterion = nn.NLLLoss()
112
```

Apply Seq2Seq : Training

```
101 def train(pairs, source_vocab, target_vocab, encoder, decoder, n_iter, print_every=1000, learning_rate=0.01):


113     for i in range(1, n_iter + 1):
114         source_tensor = training_source[i - 1]
115         target_tensor = training_target[i - 1]
116 
117         encoder_hidden = torch.zeros([1, 1, encoder.hidden_size]).to(device)
118 
119         encoder_optimizer.zero_grad()
120         decoder_optimizer.zero_grad()
121 
122         source_length = source_tensor.size(0)
123         target_length = target_tensor.size(0)
124 
125         loss = 0
126 
127         for enc_input in range(source_length):
128             _, encoder_hidden = encoder(source_tensor[enc_input], encoder_hidden)
129 
```

Apply Seq2Seq : Training

```
101 def train(pairs, source_vocab, target_vocab, encoder, decoder, n_iter, print_every=1000, learning_rate=0.01):


130         decoder_input = torch.Tensor([[SOS_token]]).long().to(device)
131         decoder_hidden = encoder_hidden
132
133         for di in range(target_length):
134             decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
135             loss += criterion(decoder_output, target_tensor[di])
136             decoder_input = target_tensor[di]  # teacher forcing
137
138         loss.backward()
139
140         encoder_optimizer.step()
141         decoder_optimizer.step()
142
143         loss_iter = loss.item() / target_length
144         loss_total += loss_iter
145
146         if i % print_every == 0:
147             loss_avg = loss_total / print_every
148             loss_total = 0
149             print("[{} - {}%] loss = {:05.4f}".format(i, i / n_iter * 100, loss_avg))
150
151
```

What’s Next?

- Sequential data treatment in PyTorch