Applied Deep Learning

Assignment 1 Brief Tutorial

March 24th, 2020 http://adl.miulab.tw

National Taiwan University

Assignment 1

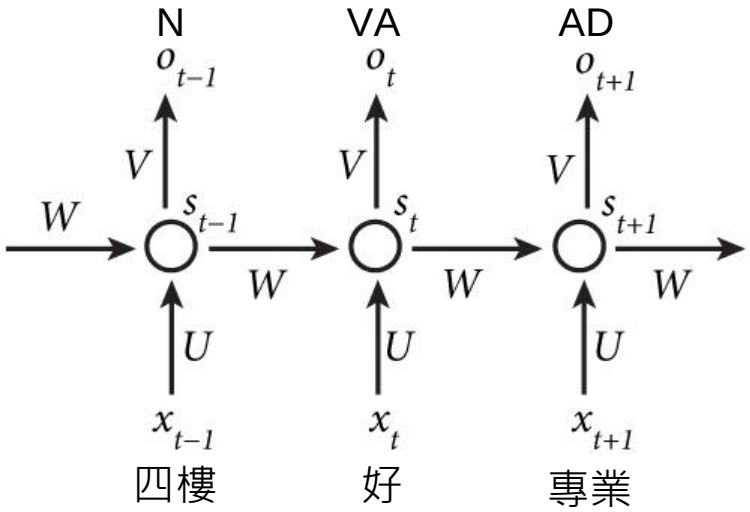
- Extractive Summarization
  - Aligned Sequential Pairs
- Abstractive Summarization
  - Unaligned Sequential Pairs
- PyTorch Tutorial
- Feedback Responses

Assignment 1

- **Extractive Summarization**
  - Aligned Sequential Pairs
- Abstractive Summarization
  - Unaligned Sequential Pairs
- PyTorch Tutorial
- Feedback Responses

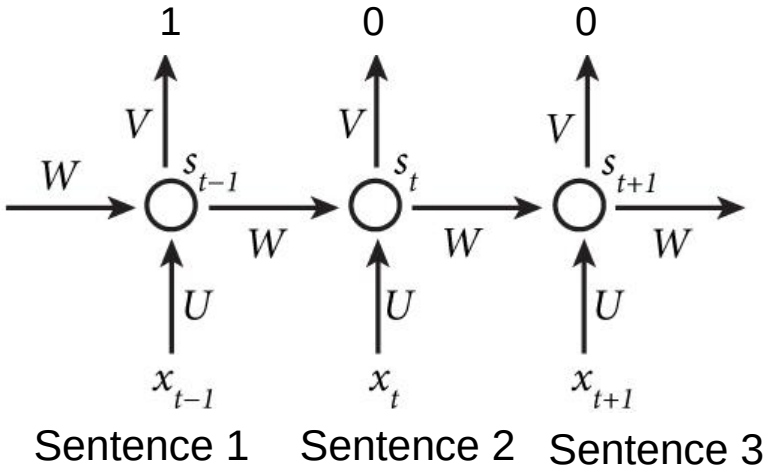
POS Tagging

- Tag a word at each timestamp
  - Input: word sequence
  - Output: corresponding POS tag sequence



Extractive Summarization

- Tag a sentence at each timestamp
  - Input: a document (sentence sequence)
  - Output: corresponding tag indicating whether the sentence should be extracted

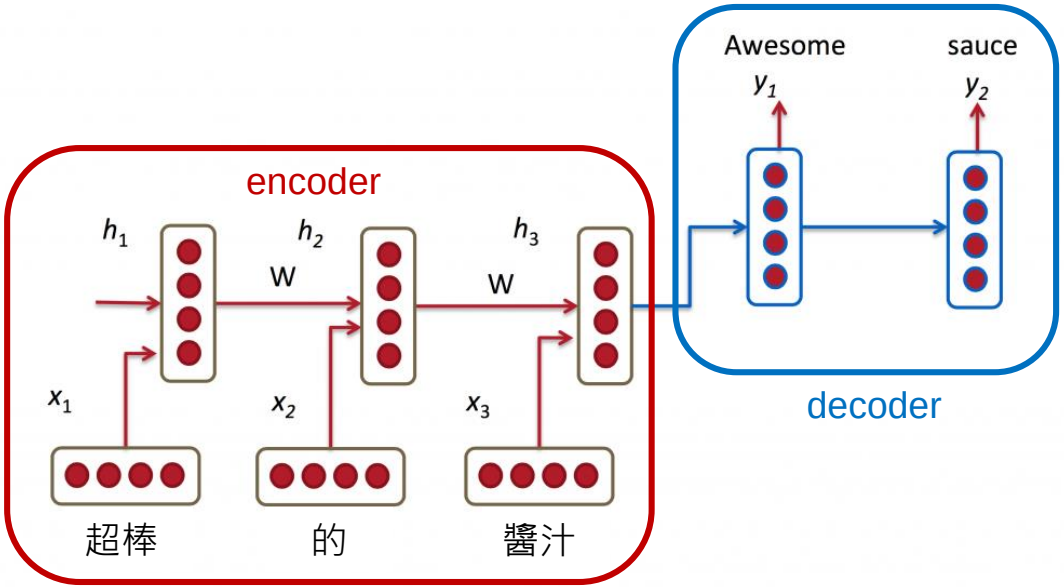


Assignment 1

- Extractive Summarization
  - Aligned Sequential Pairs
- **Abstractive Summarization**
  - Unaligned Sequential Pairs
- PyTorch Tutorial
- Feedback Responses

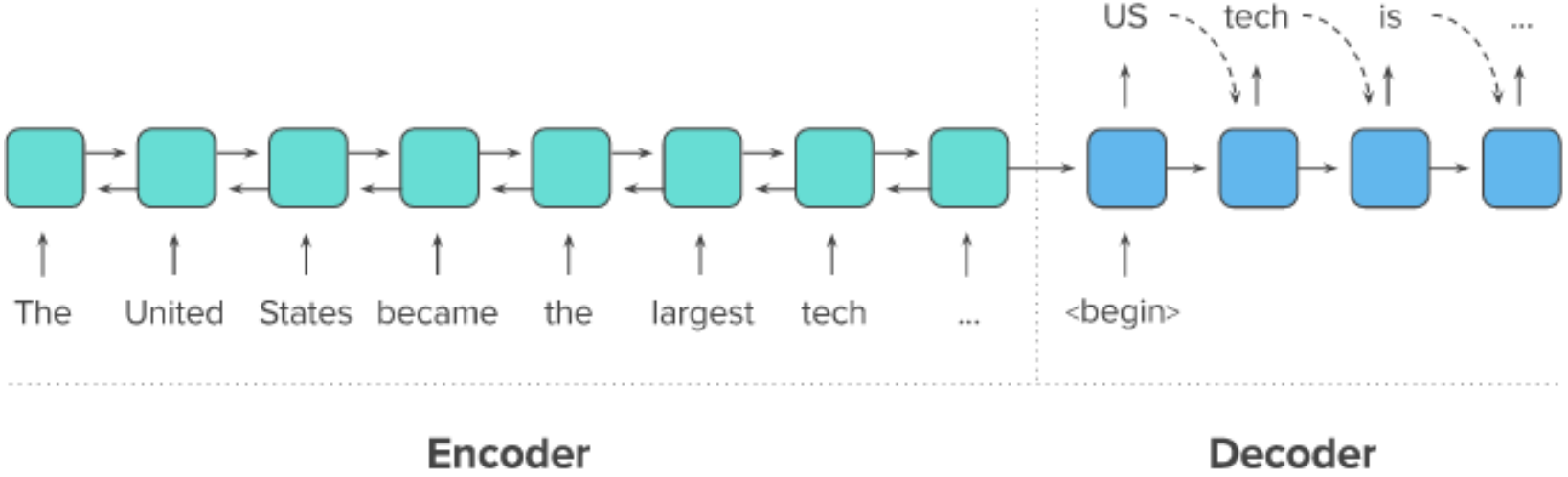
Machine Translation

- Cascade two RNNs, one for encoding and one for decoding
  - Input: word sequences in the source language
  - Output: word sequences in the target language



Abstractive Summarization

- Cascade two RNNs, one for encoding and one for decoding
  - Input: word sequences in the document
  - Output: word sequences in the summary



Assignment 1

- Extractive Summarization
  - Aligned Sequential Pairs
- Abstractive Summarization
  - Unaligned Sequential Pairs
- **PyTorch Tutorial**
- Feedback Responses

PyTorch RNN/LSTM Tutorial

https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

Prepare the data

```
def prepare_sequence(seq, to_ix):
    idxs = [to_ix[w] for w in seq]
    return torch.tensor(idxs, dtype=torch.long)


training_data = [
    ("The dog ate the apple".split(), ["DET", "NN", "V", "DET", "NN"]),
    ("Everybody read that book".split(), ["NN", "V", "DET", "NN"])
]
word_to_ix = {}
for sent, tags in training_data:
    for word in sent:
        if word not in word_to_ix:
            word_to_ix[word] = len(word_to_ix)
print(word_to_ix)
tag_to_ix = {"DET": 0, "NN": 1, "V": 2}

# These will usually be more like 32 or 64 dimensional.
# We will keep them small, so we can see how the weights change as we train.
EMBEDDING_DIM = 6
HIDDEN_DIM = 6
```

```
{'The': 0, 'dog': 1, 'ate': 2, 'the': 3, 'apple': 4, 'Everybody': 5, 'read': 6, 'that': 7,
'book': 8}
```

PyTorch RNN/LSTM Tutorial

https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

- Create the model

```
class LSTMTagger(nn.Module):

    def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size):
        super(LSTMTagger, self).__init__()
        self.hidden_dim = hidden_dim

        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)

        # The LSTM takes word embeddings as inputs, and outputs hidden states
        # with dimensionality hidden_dim.
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)

        # The linear layer that maps from hidden state space to tag space
        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

    def forward(self, sentence):
        embeds = self.word_embeddings(sentence)
        lstm_out, _ = self.lstm(embeds.view(len(sentence), 1, -1))
        tag_space = self.hidden2tag(lstm_out.view(len(sentence), -1))
        tag_scores = F.log_softmax(tag_space, dim=1)
        return tag_scores
```

PyTorch RNN/LSTM Tutorial

https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

- Train the model

```
model = LSTMTagger(EMBEDDING_DIM, HIDDEN_DIM, len(word_to_ix), len(tag_to_ix))
loss_function = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)

# See what the scores are before training
# Note that element i,j of the output is the score for tag j for word i.
# Here we don't need to train, so the code is wrapped in torch.no_grad()
with torch.no_grad():
    inputs = prepare_sequence(training_data[0][0], word_to_ix)
    tag_scores = model(inputs)
    print(tag_scores)

for epoch in range(300):  # again, normally you would NOT do 300 epochs, it is toy data
    for sentence, tags in training_data:
        # Step 1. Remember that Pytorch accumulates gradients.
        # We need to clear them out before each instance
        model.zero_grad()

        # Step 2. Get our inputs ready for the network, that is, turn them into
        # Tensors of word indices.
        sentence_in = prepare_sequence(sentence, word_to_ix)
        targets = prepare_sequence(tags, tag_to_ix)

        # Step 3. Run our forward pass.
        tag_scores = model(sentence_in)

        # Step 4. Compute the loss, gradients, and update the parameters by
        #  calling optimizer.step()
        loss = loss_function(tag_scores, targets)
        loss.backward()
        optimizer.step()
```

PyTorch RNN/LSTM Tutorial

https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

- Check the scores after training

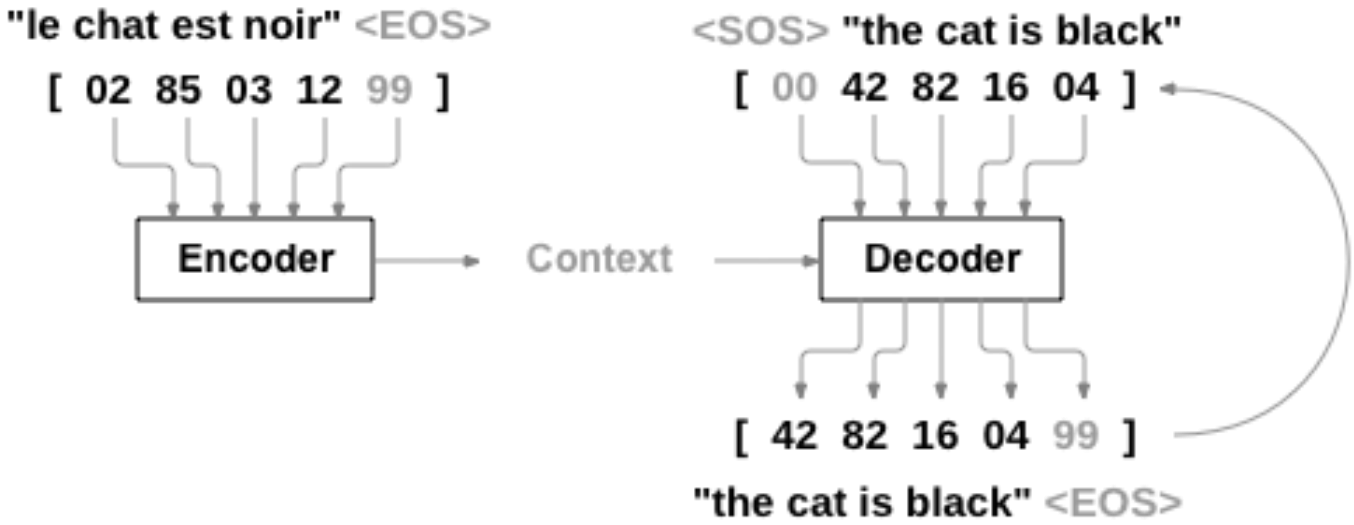
```
# See what the scores are after training
with torch.no_grad():
    inputs = prepare_sequence(training_data[0][0], word_to_ix)
    tag_scores = model(inputs)

    # The sentence is "the dog ate the apple".  i,j corresponds to score for tag j
    # for word i. The predicted tag is the maximum scoring tag.
    # Here, we can see the predicted sequence below is 0 1 2 0 1
    # since 0 is index of the maximum value of row 1,
    # 1 is the index of maximum value of row 2, etc.
    # Which is DET NOUN VERB DET NOUN, the correct sequence!
    print(tag_scores)
```

```
tensor([[-0.0462, -4.0106, -3.6096],
        [-4.8205, -0.0286, -3.9045],
        [-3.7876, -4.1355, -0.0394],
        [-0.0185, -4.7874, -4.6013],
        [-5.7881, -0.0186, -4.1778]])
```

PyTorch Attentional Seq2Seq Tutorial

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

- Seq2Seq (+Attention) for MT



Assignment 1

- Extractive Summarization
  - Aligned Sequential Pairs
- Abstractive Summarization
  - Unaligned Sequential Pairs
- PyTorch Tutorial
- **Feedback Responses**

Feedback Responses

個人覺得作業不是"難"，而是繁、雜、大，建議作業安排可以focus在特定模型，討論主題明確(這次有abstractive, extractive, seq2seq, attention...整個太繁雜巨大，甚至有人連NLP常用的前處理都沒碰過)。或是分組1~3人，報告裡詳述貢獻；又或是拆分成多個較小作業。

- 本作業為RNN練習，所有tasks皆用同一模型完成
- 課程目標：大家可以針對”任務”，找尋適合的formulation

Feedback Responses

老師說這門課是給沒有任何 deep learning 基礎的同學都可以上，但李弘毅老師的作業是可以團體一組(可最多4人)還會直接給sample code，這周作業一裡面要弄attention的model但根本還沒教過，覺的難 QQ

- ML: 15 HW
- ADL: 3 HW (difficulty: 5x)

Feedback Responses

hw1 is for ppl familiar in DL/NLP. Requirement is too much. Guidance is useless for ppl who have no experience(the target students the teacher claimed, ironic.)

- HW1 is to allow students to
  1. practice how to do text preprocessing
  2. find related materials/code for practical usage
- You can wait for the preprocessing code release

Feedback Responses

我個人覺得第一次作業確實繁雜, 但正因為這樣老師才會設一個early bird bonus呀 老師給三週 第一週讓大家試試看, 第二週給sample code（？）, 讓我們能自己先練習preprocessing, 假設真的寫不出來也沒關係, 第二三週有sample code可以follow, 可以專心在RNN的部份

- 謝謝助教貼心的設計，讓大家可以早點開始研究。
- 同學們有任何作業的建議可以留言在COOL助教才可以回復喔！

Concluding Remarks

- Cores of Assignment 1
  - Text Preprocessing (Tokenization, Embeddings, etc.)
  - Vanilla RNN
  - Attention Mechanism
- Tips
  - Using Google to find the related tutorials
  - Asking more questions in COOL
    - You can suggest TAs provide sample codes/pages for reference
- ADL is the advanced course for senior CS students!
  - You should be able to write the code from scratch
  - You should have the capability of finding needed resources