CS5242 HOMEWORK 6

Chew Kin Whye

Admin

Homework 6 Release Date: 10 Oct (Monday)

Homework 6 Due Date: 23 Oct 23:59 (Sunday)

Homework 7 Release Date: 24 Oct (Monday)

Homework 7 Due Date: 6 Nov 23:59 (Sunday)

15% deducted per day late, no marks given 7 days after deadline

Copying code from the internet is not allowed

Slack clarifications

Question 1

Question: Train a vanilla RNN for language modelling with test perplexity less than 400

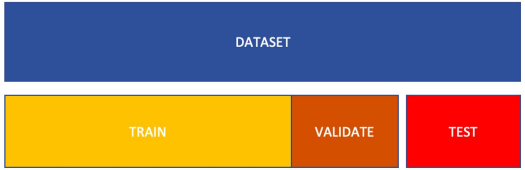
L/O: Identify the key hyperparameter(s) and tune them.

- Perplexity will not be low enough without any tuning

Hints:

- Think about what problem RNNs facing during training, and which hyper-parameter helps to mitigate this problem.

Note: In practice, you should **NEVER** use the test set for tuning, else the test set will be biased, use validation set instead

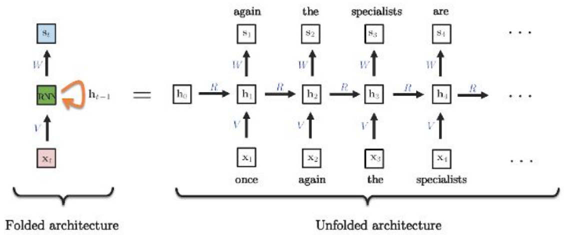


Question 2

Question: Train and **implement** a vanilla RNN

L/O: Deeper understanding of how RNN works through implementation

Hints:



- For a **single** timestep (t)
  - 2 inputs (x_t and h_{t-1})
  - 1 output (h_t)
  - Input to output operation $h_t = \tanh(Ah_{t-1} + a + Bx_t + b)$
- Loop through all timesteps to obtain all hidden states
- Use hidden states to obtain outputs $y_t = Ch_t + c$

Question 3

Goal: Train and **implement** a GRU for language modelling

L/O: Learn how to implement more complicated architectures

Hints:

- Difference: Forward pass operations

  Vanilla RNN: $h_t = \tanh(Ah_{t-1} + a + Bx_t + b)$ GRU:

$$
\begin{aligned}
r_t &= \text{sigmoid}(Ax_t + a + Bh_{t-1} + b) \\
z_t &= \text{sigmoid}(Cx_t + c + Dh_{t-1} + d) \\
n_t &= \tanh(Ex_t + e + r_t \odot (Fh_{t-1} + f)) \\
h_t &= (1 - z_t) \odot n_t + z_t \odot h_{t-1}
\end{aligned}
$$

- Everything else is the same

Hadamard product is simply the element-wise product

$$
\overset{G}{\begin{bmatrix} 3 & 5 & 7 \\ 4 & 9 & 8 \end{bmatrix}} \circ \overset{H}{\begin{bmatrix} 1 & 6 & 3 \\ 0 & 2 & 9 \end{bmatrix}} = \overset{N}{\begin{bmatrix} 3\times1 & 5\times6 & 7\times3 \\ 4\times0 & 9\times2 & 8\times9 \end{bmatrix}}
$$

Question 4

Goal: Language translation model with attention

L/O:

- Understand language preprocessing
- Understand seq2seq training
- Implement attention network

French Input	**English Output**
Ce cours est génial	**This course is awesome**

Question 4 - Language Preprocessing

Pipeline

1) Load Dataset
2) Tokenization
3) Token to index
4) BOS and EOS tokens
5) Padding

Question 4 - Language Preprocessing Dataset

```python
# First, we create a custom dataset to load the data. Each item is a pair of french and english datapoint
class CustomDataset(Dataset):
    def __init__(self, train, train_size=10000, test_size=1000):
        self.en_dir = os.path.join("dataset", "europarl-v7.fr-en.en")
        self.fr_dir = os.path.join("dataset", "europarl-v7.fr-en.fr")
        # First 10000 datapoints for train
        if train:
          with open(self.en_dir, "r", encoding="utf8") as f:
              self.english_data = f.readlines()[:train_size]
          with open(self.fr_dir, "r", encoding="utf8") as f:
              self.french_data = f.readlines()[:train_size]
        # Next 10000 datatpoints for test
        else:
            with open(self.en_dir, "r", encoding="utf8") as f:
                self.english_data = f.readlines()[train_size:train_size+test_size]
            with open(self.fr_dir, "r", encoding="utf8") as f:
                self.french_data = f.readlines()[train_size:train_size+test_size]

    def __len__(self):
        return len(self.english_data)

    def __getitem__(self, idx):
        return self.french_data[idx], self.english_data[idx]
```

Open up the files

Load the data point

```python
train_dataset = CustomDataset(train=True)
train_dataloader = DataLoader(train_dataset, batch_size=bs, shuffle=True)
```

Question 4 - Language Preprocessing Tokenization

```
# Next, we load the tokenizer that transforms the input sentence into tokens
token_transform = {}
token_transform[SRC_LANGUAGE] = get_tokenizer('spacy', language='fr_core_news_sm')
token_transform[TGT_LANGUAGE] = get_tokenizer('spacy', language='en_core_web_sm')
```

	French Input	English Output
Raw	“Ce cours est génial”	“This course is awesome”
Tokenized	[“Ce”, “cours”, “est”, “génial”]	[“This”, “course”, “is”, “awesome”]

Question 4 - Language Preprocessing Token to Index

```python
vocab_transform = {}
for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
    # Training data Iterator
    train_iter = iter(dataset)
    # Create torchtext's Vocab object
    vocab_transform[ln] = build_vocab_from_iterator(yield_tokens(train_iter, ln),
                                                    min_freq=1,
                                                    specials=special_symbols,
                                                    special_first=True)
# Define special symbols and indices
UNK_IDX, PAD_IDX, BOS_IDX, EOS_IDX = 0, 1, 2, 3
# Make sure the tokens are in order of their indices to properly insert them in vocab
special_symbols = ['<unk>', '<pad>', '<bos>', '<eos>']
```

Loop through the dataset to build dictionary, assigning each unique word a unique index

Index 0, 1, 2, 3 are special indices

	French Input	English Output
Raw	**“Ce cours est génial”**	**“This course is awesome”**
Tokenized	**[“Ce”, “cours”, “est”, “génial”]**	**[“This”, “course”, “is”, “awesome”]**
Index	**[4, 5, 6, 7]**	**[4, 5, 6, 7]**

Question 4 - Language Preprocessing BOS and EOS

```
def tensor_transform(token_ids: List[int]):
    return torch.cat((torch.tensor([BOS_IDX]),
                      torch.tensor(token_ids),
                      torch.tensor([EOS_IDX])))
```

Add beginning of sentence and end of sentence indices.

	French Input	English Output
Raw	“Ce cours est génial”	“This course is awesome”
Tokenized	[“Ce”, “cours”, “est”, “génial”]	[“This”, “course”, “is”, “awesome”]
Index	[4, 5, 6, 7]	[4, 5, 6, 7]
BOS and EOS	[2, 4, 5, 6, 7, 3]	[2, 4, 5, 6, 7, 3]

Question 4 - Language Preprocessing Padding

```
from torch.nn.utils.rnn import pad_sequence
src_batch = pad_sequence(src_batch, padding_value=PAD_IDX)
tgt_batch = pad_sequence(tgt_batch, padding_value=PAD_IDX)
```

Pad for same sequence length in batch

Makes the data handling easier

	French Input	English Output
Raw	“Ce cours est génial”	“This course is awesome”
Tokenized	[“Ce”, “cours”, “est”, “génial”]	[“This”, “course”, “is”, “awesome”]
Index	[4, 5, 6, 7]	[4, 5, 6, 7]
BOS and EOS	[2, 4, 5, 6, 7, 3]	[2, 4, 5, 6, 7, 3]
Padding	[2, 4, 5, 6, 7, 3, 1, 1]	[2, 4, 5, 6, 7, 3, 1, 1, 1, 1]

Question 4 - Language Preprocessing Total Code

```
# Print an example
batch_size = 8
dataset = CustomDataset(train=True)
train_dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
fr_sentence, eng_sentence = next(iter(train_dataloader))
print(f"Raw Inputs: {fr_sentence[0]}\n{eng_sentence[0]}")
# First we split the sentence into tokens
fr_token, eng_token = [token_transform["fr"](i.rstrip("\n")) for i in fr_sentence], [token_transform["en"](i.rstrip("\n")) for i in eng_sentence]
print(f"Tokenized Inputs: {fr_token[0]}\n{eng_token[0]}")
# # Next we transform the tokens into numbers
fr_idx, eng_idx = [vocab_transform["fr"](i) for i in fr_token], [vocab_transform["en"](i) for i in eng_token]
print(f"Tokenized Inputs to indicies: {fr_idx[0]}\n{eng_idx[0]}")
# # Next, we add the beginning of sentence, end of sentence
fr_pad, eng_pad = [tensor_transform(i) for i in fr_idx], [tensor_transform(i) for i in eng_idx]
print(f"Tokenized Indicies with begin (2) and end token (3): {fr_pad[0]}\n{eng_pad[0]}")
# # Lastly, we pad the rest of the sentence
# This also changes the shape from (bs, seq_len) to (seq_len, bs)
fr_pad, eng_pad = pad_sequence(fr_pad, padding_value=PAD_IDX), pad_sequence(eng_pad, padding_value=PAD_IDX)
print(f"After padding (1): {fr_pad[:, 0]}\n{eng_pad[:, 0]}")

# All the above is combined into collate_fn
x, y = collate_fn(fr_sentence, eng_sentence)
print(f"Same Outputs: {x[:, 0]}\n{y[:, 0]}")
```

Question 4 - Language Preprocessing Total Outputs

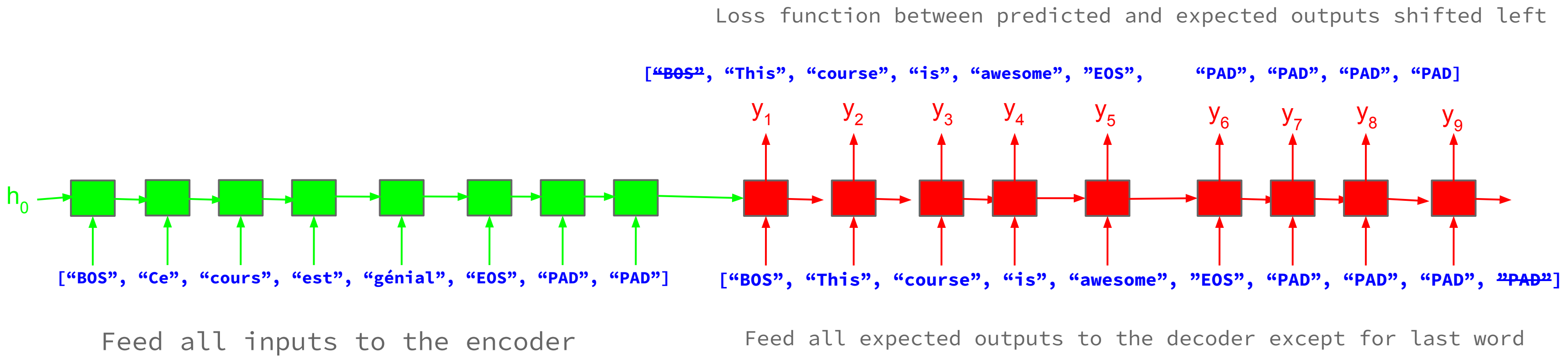
```
Raw Inputs: La France ayant alors très fortement placé l'exemption légale à l'avant-plan, elle avait été dédommagée par le biais de concessions dans la politique agricole.

Since France laid a huge amount of emphasis on legal exemption at the time, it was damaged by concessions in agricultural policy.

Tokenized Inputs: ['La', 'France', 'ayant', 'alors', 'très', 'fortement', 'placé', "l'", 'exemption', 'légale', 'à', "l'", 'avant-plan', ',', 'elle', 'avait', 'été', 'dédommagée', 'pa
['Since', 'France', 'laid', 'a', 'huge', 'amount', 'of', 'emphasis', 'on', 'legal', 'exemption', 'at', 'the', 'time', ',', 'it', 'was', 'damaged', 'by', 'concessions', 'in', 'agricult
Tokenized Inputs to indicies: [69, 681, 670, 314, 92, 1054, 3380, 14, 3001, 3713, 11, 14, 5518, 4, 82, 401, 64, 11189, 34, 10, 851, 5, 5601, 25, 8, 61, 1080, 7]
[1071, 610, 1267, 13, 1045, 954, 8, 1357, 20, 250, 1926, 39, 4, 90, 5, 26, 65, 3711, 32, 4900, 11, 1005, 86, 7]
Tokenized Indicies with begin (2) and end token (3): tensor([    2,    69,   681,   670,   314,    92,  1054,  3380,    14,  3001,
         3713,    11,    14,  5518,     4,    82,   401,    64, 11189,    34,
           10,   851,     5,  5601,    25,     8,    61,  1080,     7,     3])
tensor([   2, 1071,  610, 1267,   13, 1045,  954,    8, 1357,   20,  250, 1926,
          39,    4,   90,    5,   26,   65, 3711,   32, 4900,   11, 1005,   86,
           7,    3])
After padding (1): tensor([    2,    69,   681,   670,   314,    92,  1054,  3380,    14,  3001,
         3713,    11,    14,  5518,     4,    82,   401,    64, 11189,    34,
           10,   851,     5,  5601,    25,     8,    61,  1080,     7,     3,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
            1,     1,     1])
tensor([   2, 1071,  610, 1267,   13, 1045,  954,    8, 1357,   20,  250, 1926,
          39,    4,   90,    5,   26,   65, 3711,   32, 4900,   11, 1005,   86,
           7,    3,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1])
```

Question 4 - Seq2Seq Training

Simple RNN encoder-decoder architecture without attention

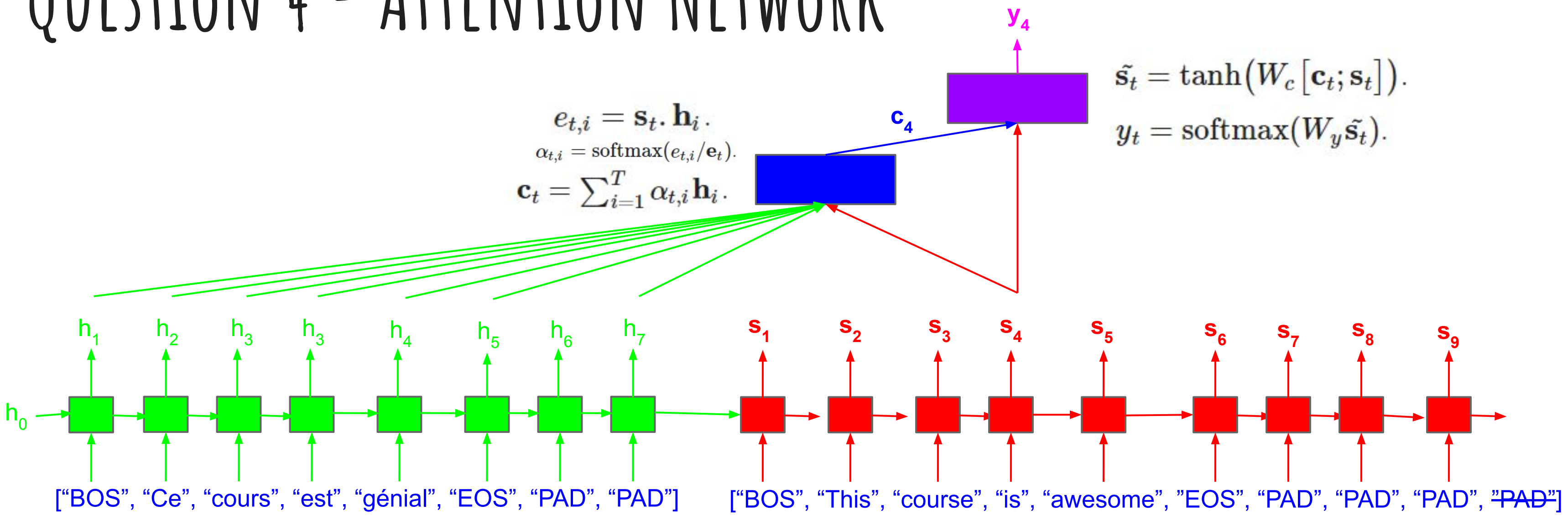


The decoder has to predict the next timestep given the previous timesteps.

Green: Encoder RNN

Red: Decoder RNN

Question 4 - Attention Network



Obtain h_1 to h_7 sequentially (Use nn.GRU())

Obtain s_1 to s_9 sequentially (Use nn.GRU())

Calculate alignment, attention, and context in parallel

Calculate output in parallel

Green: Encoder RNN
Red: Decoder RNN
Blue: Attention Module
Pink: Output Module

Helpful slide for attention by matrix multiplication

Scaled dot product attention

- **Attention (q, k, v) for a query q and n key-value pairs :**

$$\text{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \in \mathbb{R}^{d_v}$$

$$\mathbf{q} \in \mathbb{R}^{d_k}, \mathbf{K} \in \mathbb{R}^{n \times d_k}, \mathbf{V} \in \mathbb{R}^{n \times d_v}$$

- **Attention for m Queries and n Key-Value pairs**

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \in \mathbb{R}^{m \times d_v}$$

$$\mathbf{Q} \in \mathbb{R}^{m \times d_k}, \mathbf{K} \in \mathbb{R}^{n \times d_k}, \mathbf{V} \in \mathbb{R}^{n \times d_v}$$

Slide 15 from L09

Question 4 - Unroll RNN to make predictions

```python
for x, y in test_dataloader:
    print(x)
    # set the initial h to be the zero vector
    h = torch.zeros(1, 1, hidden_size)

    # send it to the gpu
    h=h.to(device)
    x, y = collate_fn(x, y)
    # send them to the gpu
    minibatch_data=x.type(torch.LongTensor).to(device)
    # The first prediction is the start of sentence index
    start_index = torch.tensor([[2]]).type(torch.LongTensor).to(device)
    predictions=start_index
    # Usually we keep looping till the EOS token. In this case, to prevent the possibility of an infinite loop, we keep it to 20 words.
    # Make 20 words of predictions
    for _ in range(20):
      # At every loop, pass in the previous predictions
      predictions = net.forward(minibatch_data, predictions, h)
      predictions = torch.reshape(predictions, (-1, TGT_VOCAB_SIZE, 1))
      # Get the new predictions shifted right by 1 timestep
      predictions = torch.argmax(predictions, dim=1)
      # Add back the first timestep
      predictions = torch.cat([start_index, predictions], 0)
      if predictions[-1].item() == 3:
          break
    predictions = predictions.reshape(-1)
    # Transform from token to words
    predictions = [vocab_transform[TGT_LANGUAGE].lookup_token(i) for i in predictions]
    print(f"Label: {[vocab_transform[TGT_LANGUAGE].lookup_token(i) for i in y]}")
    print(f"Predicted: {predictions}")
```

Question 4 - Example - Trained Model Performance

```
('Pourquoi cette omission ?\n',)
Label: ['<bos>', 'Why', 'is', 'this', '?', '<eos>']
Predicted: ['<bos>', 'Why', 'this', '?', '<eos>']
('Ce sera ma première question.\n',)
Label: ['<bos>', 'That', 'is', 'my', 'first', 'question', '.', '<eos>']
Predicted: ['<bos>', 'I', 'would', 'like', 'to', 'ask', 'the', 'question', '.', '<eos>']
("Nous sommes d'accord !\n",)
Label: ['<bos>', 'We', 'agree', '.', '<eos>']
Predicted: ['<bos>', 'We', 'agree', '.', '<eos>']
```

Even though output is not that good, all we need to improve the performance is to scale!

- Scale network size
- Scale dataset size
- Scale training epochs
- Improve on model architecture (Transformers)

Key concepts are the same

Questions?