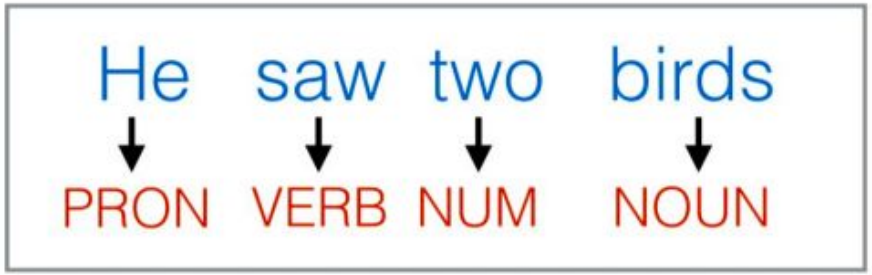
11-737 Multilingual NLP

Assignment 1

Athiya Deviyani Ting-Rui Chiang

Task: Multilingual Parts-of-Speech Tagging

- Parts-of-speech: Lexical Categories or Word Classes or Tags



- Input: a sentence
- Output: a sequence of POS tags

Assignment goals

- Practical introduction to multilingual parts-of-speech (POS) tagging
- Investigate challenges related to the limited availability of labeled data
- Be familiarized with deep learning frameworks (PyTorch), computing resources (AWS), and multilingual datasets

Requirements

Machine with a GPU

- AWS
- Your own computer

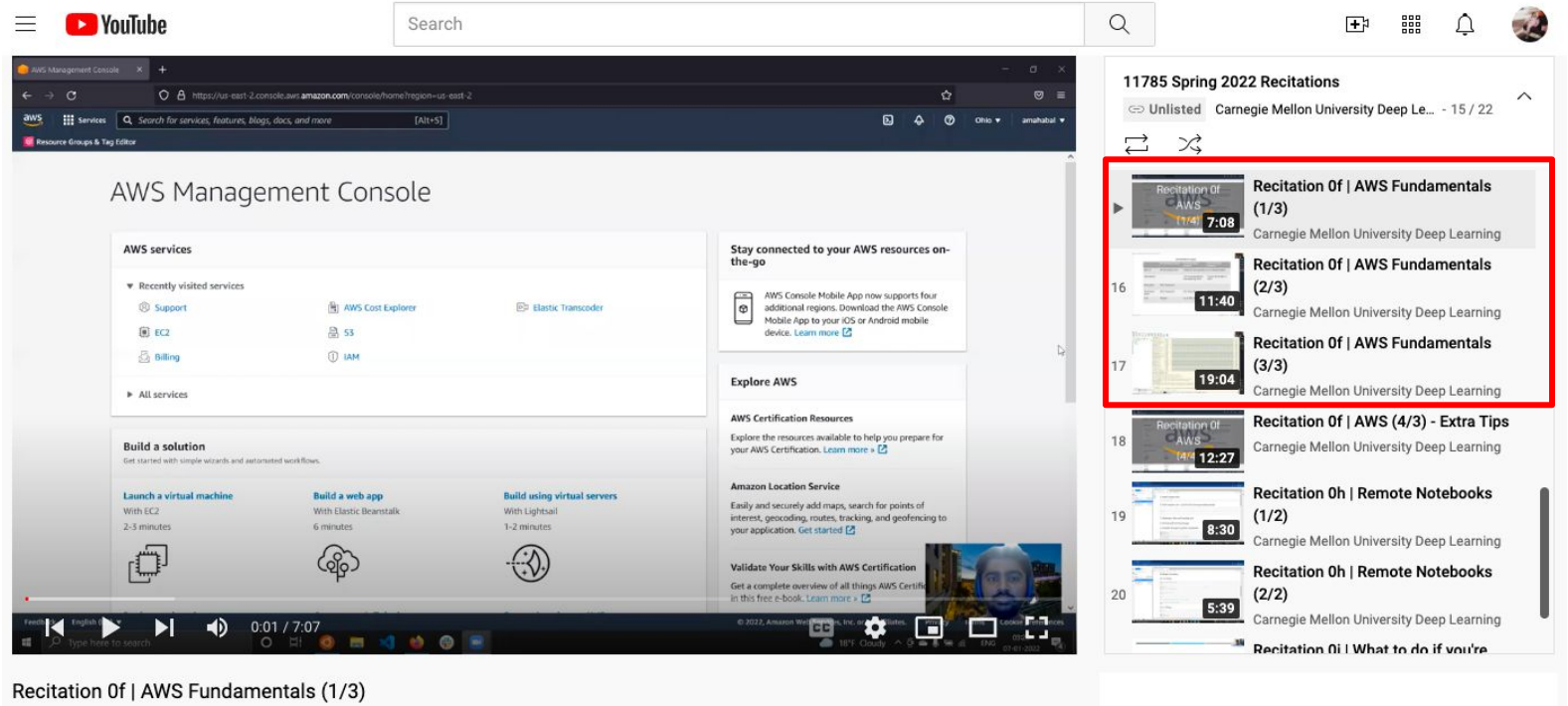
Software/package/libraries

- Conda
- Numpy
- Python=3.8
- PyTorch=1.10.1
- Torchtext=0.11.1

Using AWS

- Request AWS Credit (instructions on Piazza)
- All students should have and use an AWS account using their @andrew.cmu.edu email, and should join AWS educate on this account: https://aws.amazon.com/education/awseducate/
- The assignment is completely doable without a GPU, but *strongly* recommended
  - Training time without GPU can range from 45s - 5 mins per epoch
- Make sure to stop the AWS instance when you’re not using it!

AWS Setup



Link: https://www.youtube.com/watch?v=Gx5eLtOY10g&list=PLp-0K3kfddPwLUlWMDjMoVfNcVbnjG9m3&index=15

Credits to Intro to DL instructor Prof. Bhiksha Raj and the TAs Ameya Mahabaleswarkar and Zhe Chen

assign1.zip (link)

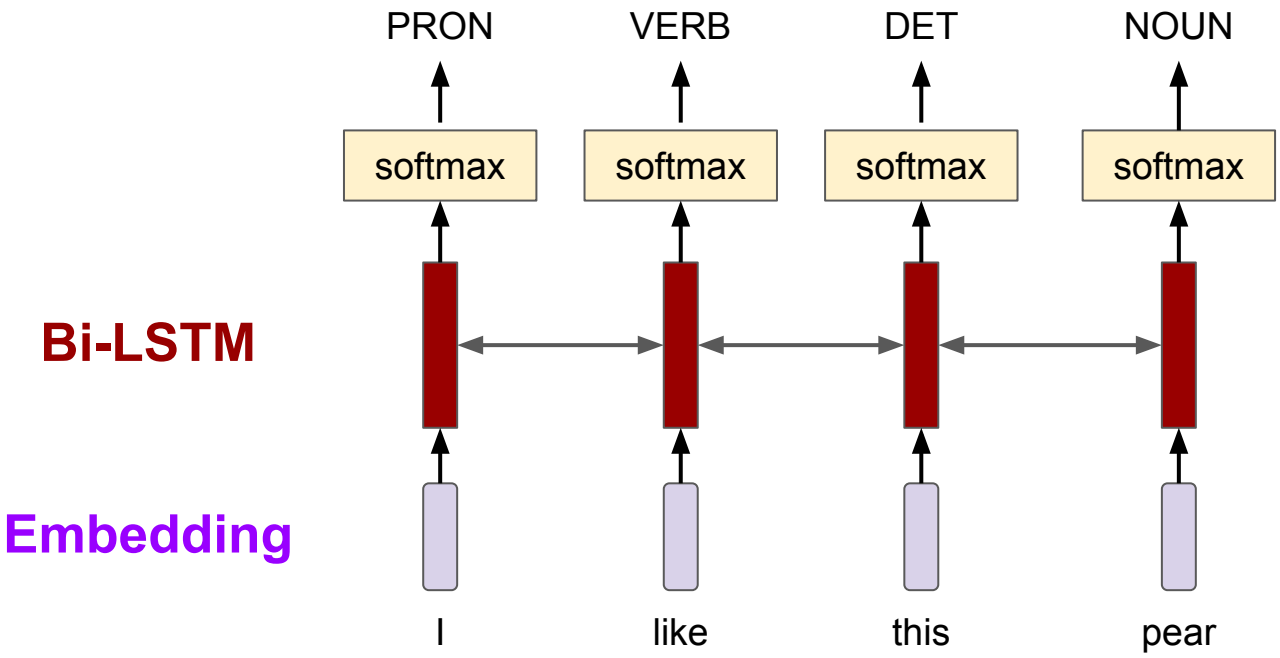
```
assign1
├── config.json
├── data/
├── saved_models/
├── main.py
├── model.py
└── udpos.py
```

Training Data

- 6 languages: en, es, cs, ar, af, lt, hy, ta
  - Germanic: English (en), Afrikaans (af)
  - Slavic: Czech (cs)
  - Romance: Spanish (es)
  - Semitic: Arabic (ar)
  - Baltic: Lithuanian (lt)
  - Armenian: Armenian (hy)
  - Dravidian: Tamil (ta)

```
DPA  PROPN
:    PUNCT
Iraqi   ADJ
authorities   NOUN
announced     VERB
that SCONJ
they PRON
had  AUX
busted  VERB
up   ADP
3    NUM
terrorist     ADJ
cells   NOUN
operating     VERB
in   ADP
Baghdad PROPN
.    PUNCT
```

Baseline Model - BiLSTM



Code Organization - **`config.json`**

```
assign1
├── config.json
├── data/
├── saved_models/
├── main.py
├── model.py
└── udpos.py
```

```
{
  "embedding_dim": 100,
  "hidden_dim":128,
  "n_layers":2,
  "bidirectional":true,
  "dropout":0.25,
  "batch_size": 128,
  "max_epoch": 10,
  "min_freq": 2
}
```

Code Organization - **udpos.py**

```
assign1
├── config.json
├── data/
├── saved_models/
├── main.py
├── model.py
└── udpos.py
```

```python
def UDPOS(root, split):
    if split == 'valid':
        split = 'dev'

    path = os.path.join(
        root,
        _find_match(f'{split}.conll', os.listdir(root))
    )
    data = list(_create_data_from_iob(path))
    return _RawTextIterableDataset('UDPOS', len(data), iter(data))
```

Code Organization - **`model.py`**

```
assign1
├── config.json
├── data/
├── saved_models/
├── main.py
├── model.py
└── udpos.py
```

```python
class BiLSTMPOSTagger(nn.Module):
    def __init__( ... ):
        self.embedding = nn.Embedding( ... )
        self.lstm = nn.LSTM( ... )
        self.fc = nn.Linear( … )
        self.dropout = nn.Dropout(dropout)

    def forward(self, text):
        embedded = self.dropout(self.embedding(text))
        outputs, (hidden, cell) = self.lstm(embedded)
        predictions = self.fc(self.dropout(outputs))
        return predictions
```

Code Organization

```
assign1
├── config.json
├── data/
├── saved_models/
├── main.py
├── model.py
└── udpos.py
```

- Load data
- Build the vocabularies
- Pack samples into a batch
- Train the model

main.py

```
# load the data from the specific path
train_data, valid_data, test_data = UDPOS(
    os.path.join('data', args.lang),
    split=('train', 'valid', 'test'),
)

# building the vocabulary for both text and the labels
vocab_text = torchtext.vocab.build_vocab_from_iterator(
    (line for line, label in train_data), min_freq=params['min_freq'],
    specials=['<unk>', '<PAD>']
)
vocab_text.set_default_index(vocab_text['<unk>'])
vocab_tag = torchtext.vocab.build_vocab_from_iterator(
    (label for line, label in train_data),
    specials=['<unk>', '<PAD>']
)
vocab_tag.set_default_index(vocab_text['<unk>'])
```

main.py

```
# functions that convert words/tags to indices
def transform_text(x):
    return [vocab_text[token] for token in x]
def transform_tag(x):
    return [vocab_tag[tag] for tag in x]

# function that pack (text, label) pairs into a batch
def collate_batch(batch):
    tag_list, text_list = [], []
    for (line, label) in batch:
        text_list.append(torch.tensor(transform_text(line), device=device))
        tag_list.append(torch.tensor(transform_tag(label), device=device))
    return (
        pad_sequence(text_list, padding_value=vocab_text['<PAD>']),
        pad_sequence(tag_list, padding_value=vocab_tag['<PAD>'])
    )
```

main.py

```
# iterators that generate batch with `collate_batch`
train_dataloader = DataLoader(
    train_data, batch_size=params['batch_size'],
    shuffle=True, collate_fn=collate_batch
)
valid_dataloader = DataLoader(
    valid_data, batch_size=params['batch_size'],
    shuffle=False, collate_fn=collate_batch
)
test_dataloader = DataLoader(
    test_data, batch_size=params['batch_size'],
    shuffle=False, collate_fn=collate_batch
)
```

main.py - Training

```
for epoch in range(params['max_epoch']):
    train_loss, train_acc = train( ... )
    valid_loss, valid_acc = evaluate( ... )

    # save the model if valid_loss is better
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save( ... )
```

main.py - Training

```
def train( ... ):
    for batch in iterator:
        text = batch[0]
        tags = batch[1]

        optimizer.zero_grad()

        predictions = model(text)
        predictions = predictions.view(-1, predictions.shape[-1])
        tags = tags.view(-1)
        loss = criterion(predictions, tags)

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

Submission

- Your submission consists of two parts: Code and a write-up
  - Put all your code in a folder named **`code`** including instructions on how to run if you have implemented additional code
  - Include your trained models in **`code/save_models/`**.
  - Rename the write-up as **writeup.pdf**
  - Compress both on them as **assign1.tar.gz**
- This file must be submitted to Canvas (link on the course website)

Grading

- Run the model on the provided English UD data and reproduce the results on its test set: **B**
- Train and reproduce the results on the given multilingual datasets and reproduce desired results: **B+**
- Report with detailed analysis: **A-**
  - Performance across language family, typology, datasize
  - Hyperparameter tuning: config.json
  - Error analysis across tag types: does the model perform better or worse on certain kinds of tags?
- Non-trivial extension which leads to improvement in scores: **A**, **A+**
  - Add a CNN input layer to capture character level features.
  - Pre-train the model parameters with a language modeling objective
  - Add multilingual pre-trained embeddings (Polyglot, mBERT) to your model

Additional Help

Office Hours

Ting-Rui Chiang (Wednesday 11 AM -12 PM): https://cmu.zoom.us/j/9409145401?pwd=SXhDRE1lamJtNXJzbWhrUXV1cjRtUT09

Athiya Deviyani (Thursday 3-4 PM): https://cmu.zoom.us/my/athiya

Contact

Ting-Rui Chiang <tingruic@cs.cmu.edu>

Athiya Deviyani <adeviyan@cs.cmu.edu>