Sequence Tagging with LSTMs

CS 780/880 Natural Language Processing Lecture 15

Samuel Carton, University of New Hampshire

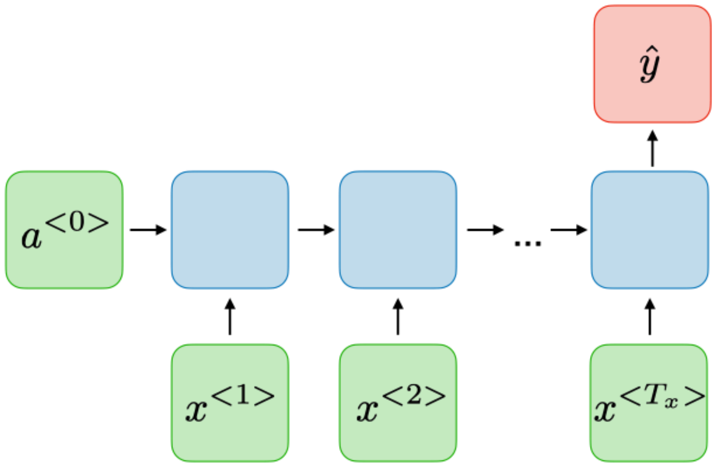
Last lecture

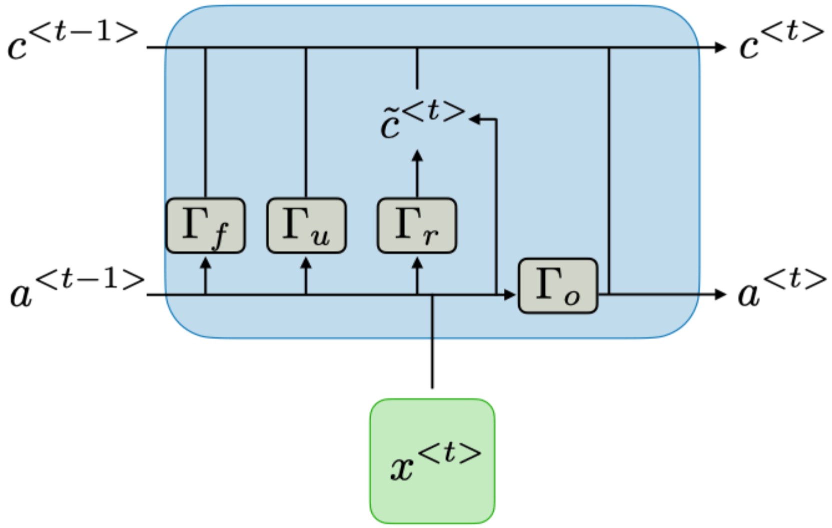
RNNs
- One-to-one
- **Many-to-one**
- Many-to-many

LSTMS

Increasing RNN capacity
- Depth
- Bidirectionality

Dropout

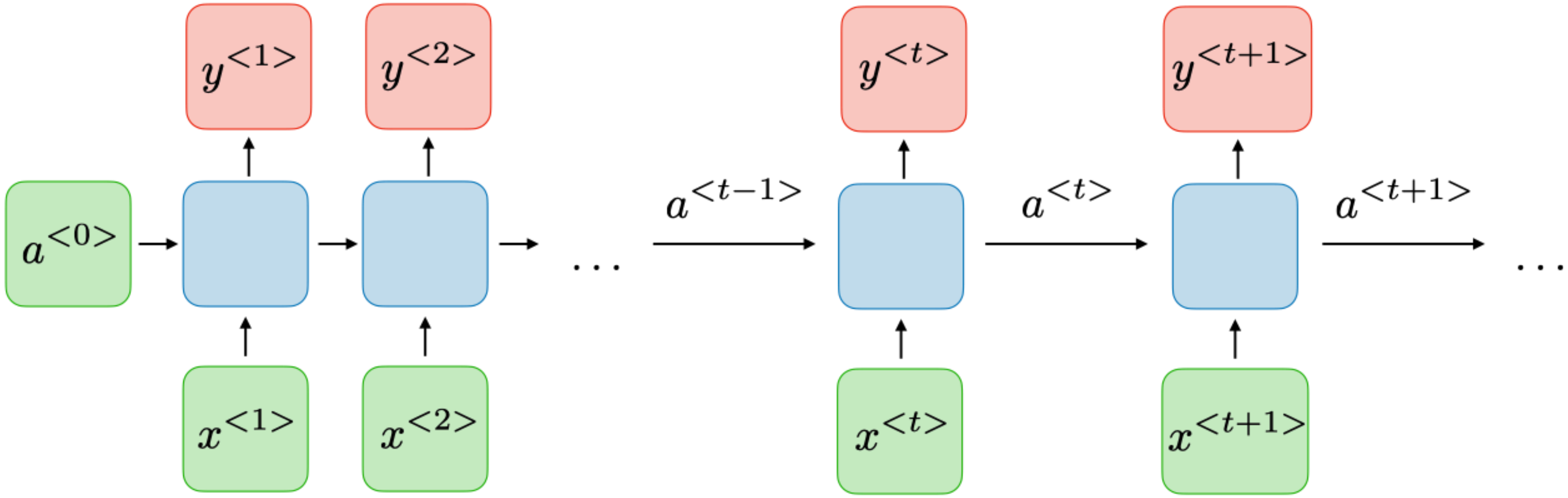




LSTMs: NLP Swiss army knife

LSTMs are exciting for us because they are the Swiss army knife of NLP models.

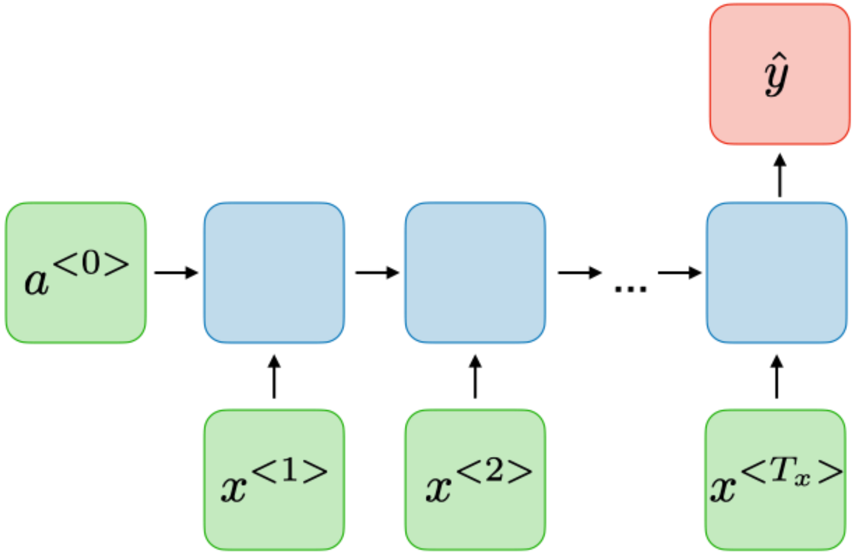
- Sequence classification
- Sequence tagging
- Language modeling
- Text-to-text (e.g. translation)



LSTMs: NLP Swiss army knife

LSTMs are exciting for us because they are the Swiss army knife of NLP models.

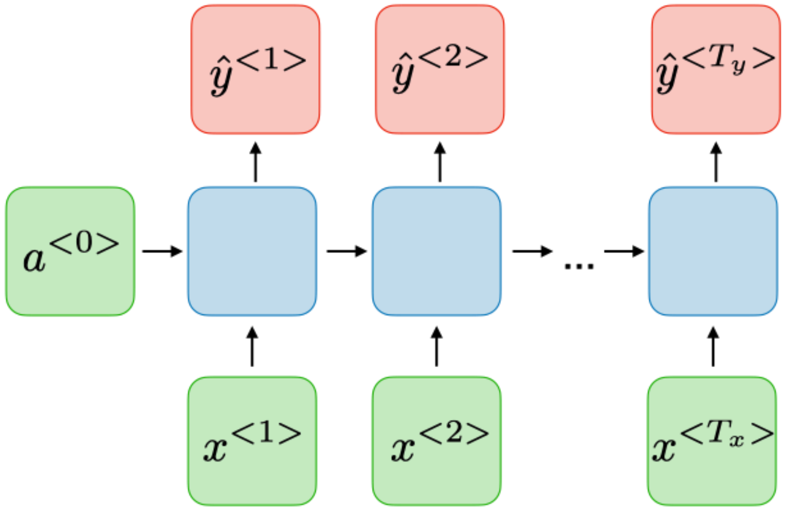
- **Sequence classification**
- Sequence tagging
- Language modeling
- Text-to-text (e.g. translation)



LSTMs: NLP Swiss army knife

LSTMs are exciting for us because they are the Swiss army knife of NLP models.

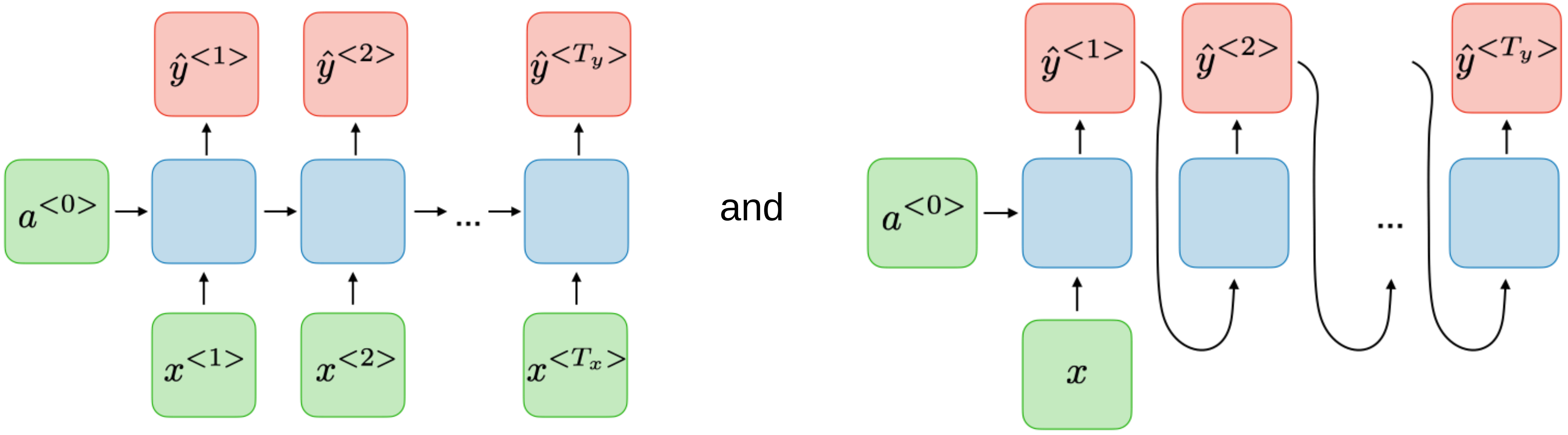
- Sequence classification
- **Sequence tagging**
- Language modeling
- Text-to-text (e.g. translation)



LSTMs: NLP Swiss army knife

LSTMs are exciting for us because they are the Swiss army knife of NLP models.

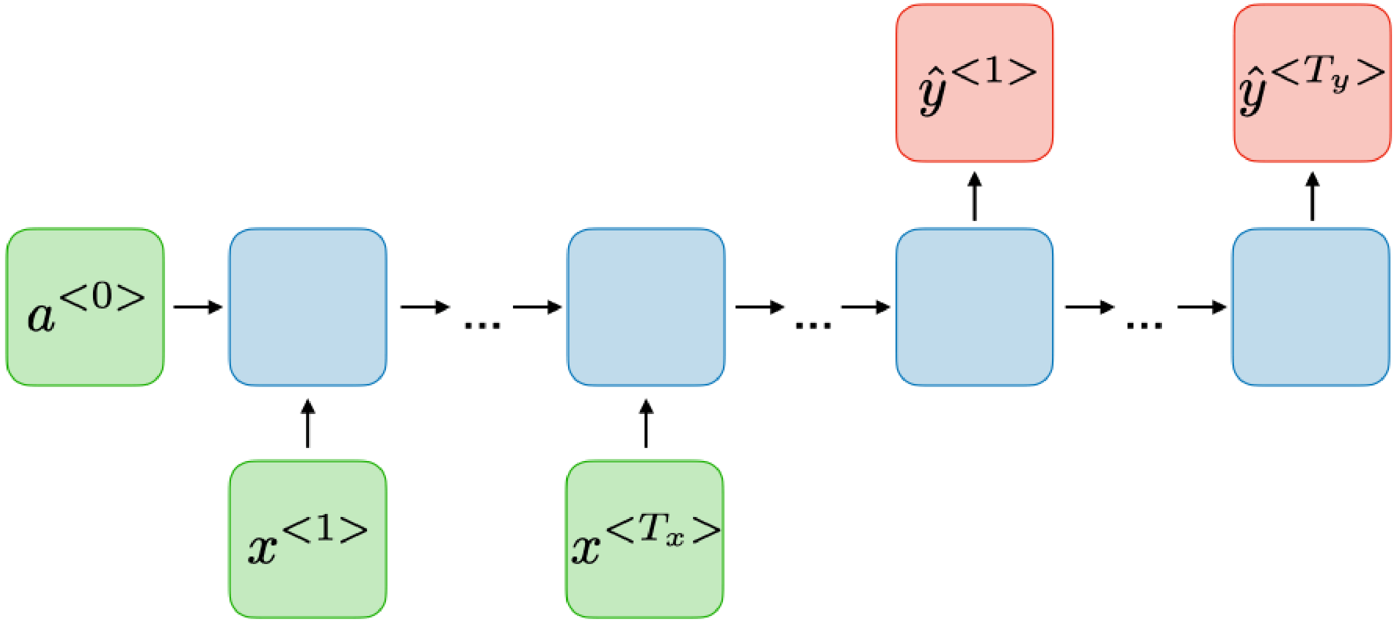
- Sequence classification
- Sequence tagging
- **Language modeling**
- Text-to-text (e.g. translation)



LSTMs: NLP Swiss army knife

LSTMs are exciting for us because they are the Swiss army knife of NLP models.

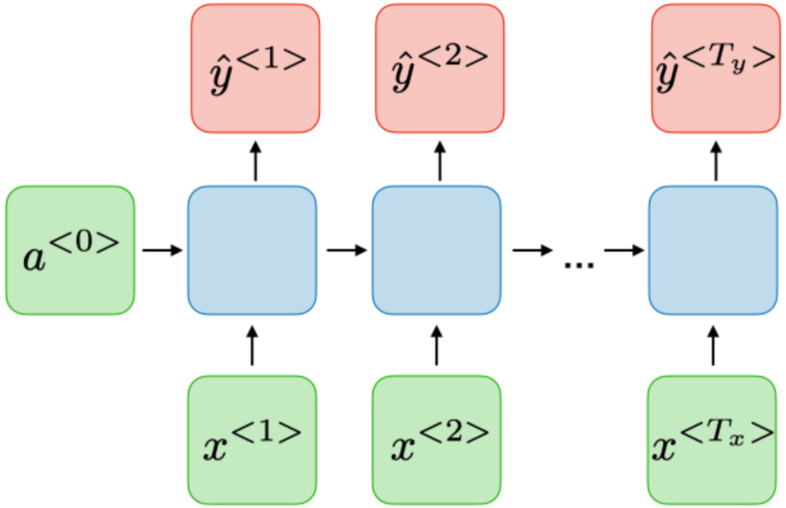
- Sequence classification
- Sequence tagging
- Language modeling
- **Text-to-text (e.g. translation)**



LSTMs: NLP Swiss army knife

LSTMs are exciting for us because they are the Swiss army knife of NLP models.

- Sequence classification
- **Sequence tagging**
- Language modeling
- Text-to-text (e.g. translation)



Sequence tagging

Basic idea: Given a corpus of text where each **word** has a label, learn to predict word labels for unseen texts

- Part-of-speech tagging
- Named entity recognition
  - "In his speech to the UN today, **George Bush** addressed the rising problems of…"
- Explanations
  - "You are a real **piece of garbage** human being." → Predicted toxic

Evaluation: Use the same metrics as for classification (Acc/P/R/F1)

- Two choices for aggregation:
  - **Calculate score for each text and then take mean**
  - Concatenate all texts together and calculate over one long sequence
- F1 preferable for very unbalanced tasks

Named-entity recognition (NER)

Goal: Identify the **named entities** (people, corporations, etc) in a piece of text.

Important for large-scale text analysis

- E.g. Extracting structured information from scientific literature
- E.g. Performing market research over social media

Usually treated as sequence tagging task, where each word is tagged as (1) part of an entity or (2) not part of an entity

F1 preferable as a metric because usually unbalanced

P2- Na2/3Ni1/4TixMn3/4-xO2 was prepared through a simple solid state method. The precursor solution was prepared by mixing desirable amount of Ni(CH3COO)2*4H2O, Mn(CH3COO)2*4H2O and CH3COONa and titanium citrate solution. The obtained mixture was heated at 400 degC for 12 h. The ground powder was ball-milled for 1 h and was subsequently calcinated at 900 degC in air for 12 h to synthesize Na2/3Ni1/4TixMn3/4-xO2 (x=0, 0.05, 0.10, 0.15, 0.20, 0.30).

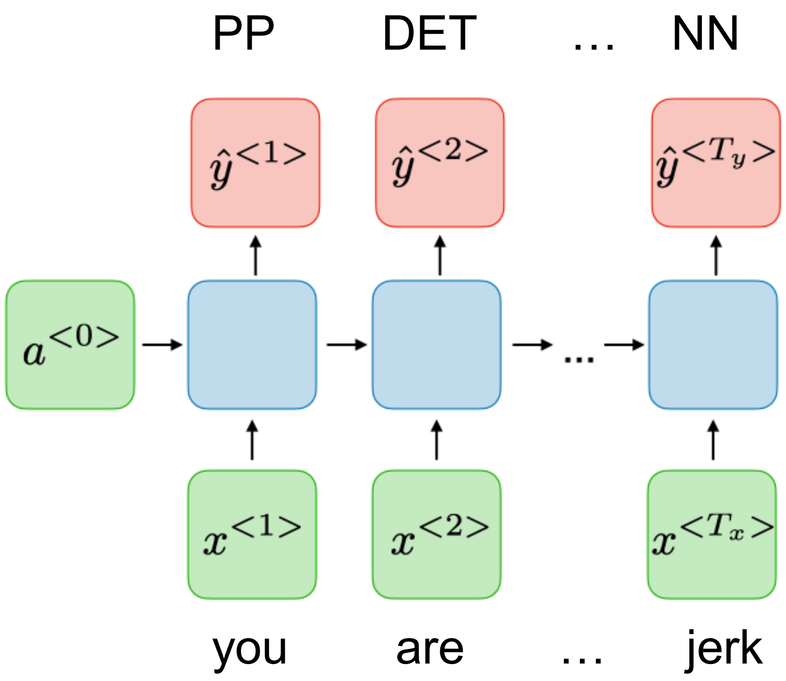
Figure 1: Part of an example synthesis procedure included in the dataset with entity annotations from Zhao et al. (2015). Colors represent entity types and underlines represent span boundaries. Colors: Target, Nonrecipe-operation, Unspecified-Material, Operation, Material, Condition-Unit, Number.

Tim O'Gorman, Zach Jensen, Sheshera Mysore, Kevin Huang, Rubayyat Mahbub, Elsa Olivetti, and Andrew McCallum. 2021. MS-Mentions: Consistently Annotating Entity Mentions in Materials Science Procedural Text. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*

Sequence tagging

Context sensitive.

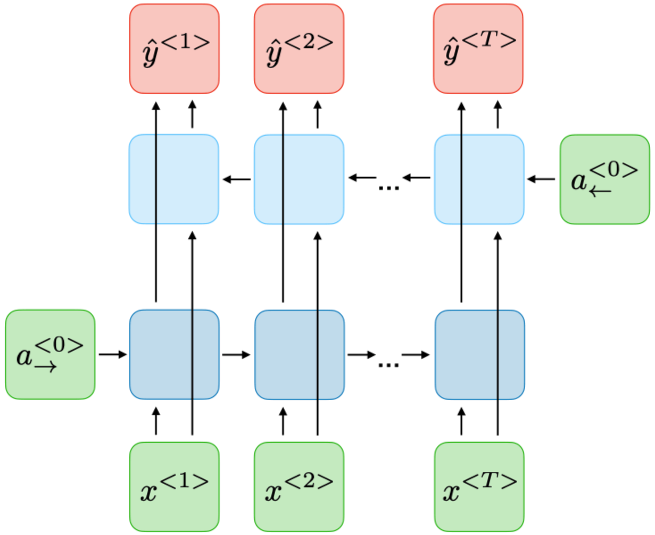
- “You are a real jerk!”
- “I am really craving some Jamaican jerk chicken right now.”

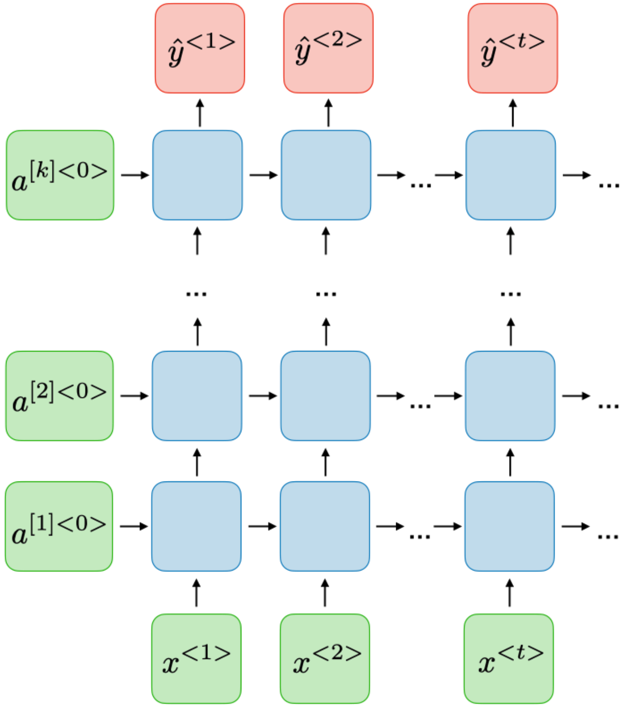


Sequence tagging

Context sensitive.

- “You are a real jerk!”
- “I am really craving some Jamaican jerk chicken right now.”



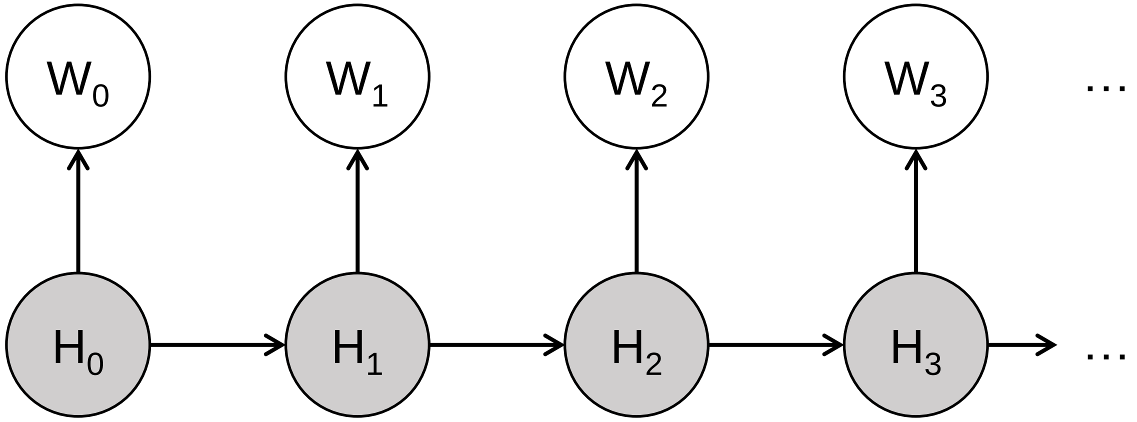


POS tagging with HMM

A popular application of HMMs in NLP is part-of-speech tagging

We imagine a generative story where parts-of-speech occur in a Markov chain, and then they emit words conditioned on their value.

i	sentence	you	to	read	this	sentence	.
PP	V	PP	PREP	V	DET	NN	PUNCT



Transition matrix

$P(H_t|H_{t-1})$

		h_0	h_1	…
	h_0	…	…	…
H_{t-1}	h_1	…	…	..
	…	...	…	…

Emission matrix

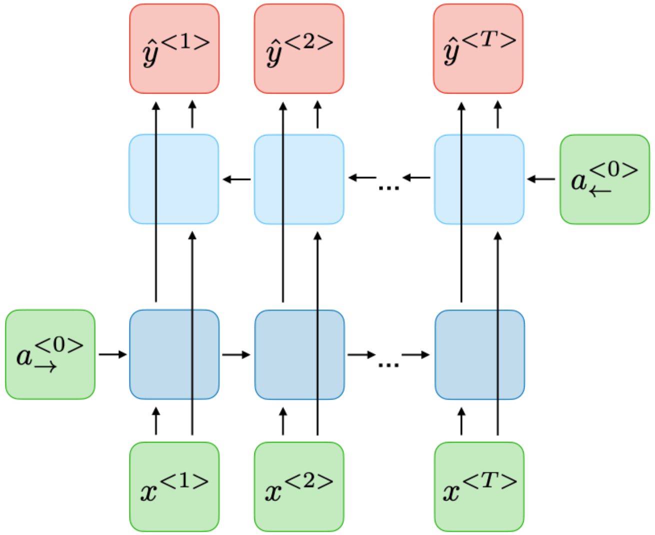
$P(W_t|H_t)$

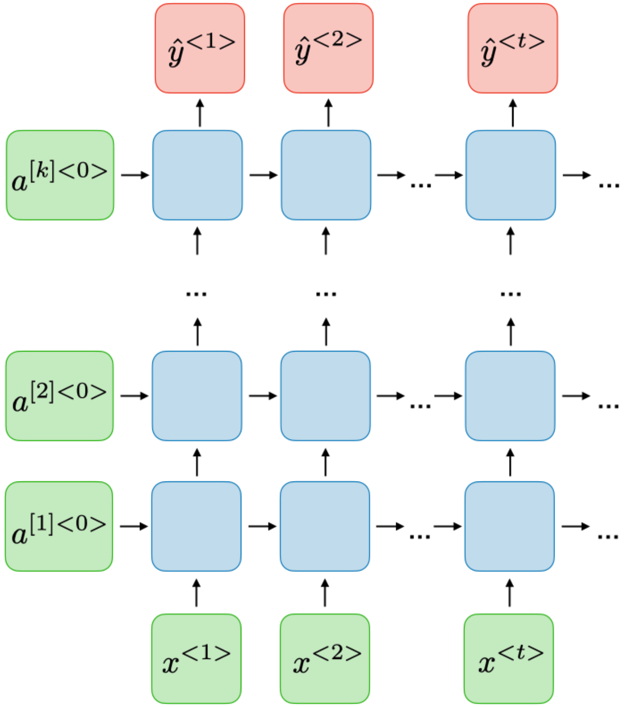
		w_0	w_1	…
	h_0	…	…	…
H_t	h_1	…	…	..
	…	...	…	…

Sequence tagging

Context sensitive.

- “You are a real jerk!”
- “I am really craving some Jamaican jerk chicken right now.”





Loading GloVe vectors with Gensim

```python
import gensim.downloader as api
```

```python
# We're gonna be working with Twitter text today, so we'll use a Twitter-specific set of
# pretrained word embeddings
vector_model = api.load('glove-twitter-100')
```

```
[=================================================-] 99.9% 386.6/387.1MB downloaded
```

Loading GloVe vectors with Gensim

```python
# We can see that this particular model has special embeddings for
# various kinds of things you'll find in tweets
vector_model.index_to_key[0:15]
```

```
['<user>',
 '.',
 ':',
 'rt',
 ',',
 '<repeat>',
 '<hashtag>',
 '<number>',
 '<url>',
 '!',
 'i',
 'a',
 '"',
 'the',
 '?']
```

```python
# It's also apparently got multilingual stuff in it
vector_model.index_to_key[-15:]
```

```
['ｇａｍｅ',
 'ｱﾑﾈｼｱ',
 'ｴｴ',
 'ｶﾞﾘｶﾞﾘ',
 'ｷｲ',
 'ｹﾞｼｯ',
 'ﾃﾍﾍﾞﾛｯ',
 'ﾃﾞﾓ',
 'ﾊﾞｲﾊﾞｰｲ',
 'ﾊﾞﾝﾁ',
 'ﾔﾒﾀﾏｴ',
 'ﾖｲｼｮｯ',
 'ﾞﾞﾞｵﾔｽﾐｰ',
 '<unk>',
 '<pad>']
```

Twitter POS tagging dataset

Named Entity Recognition in Tweets: An Experimental Study (Ritter et al., 2011)

https://raw.githubusercontent.com/aritter/twitter_nlp/master/data/annotated/pos.txt

```
@paulwalk USR
It PRP
's VBZ
the DT
view NN
from IN
where WRB
I PRP
'm VBP
living VBG
for IN
two CD
weeks NNS
. .
Empire NNP
State NNP
Building NNP
= SYM
ESB NNP
. .
Pretty RB
bad JJ
storm NN
here RB
last JJ
evening NN
. .
```

```
@MiSS_SOTO USR
I PRP
think VBP
that DT
's VBZ
when WRB
I PRP
'm VBP
gonna VBG
be VB
there RB

On IN
Thanksgiving NNP
after IN
you PRP
done VBN
eating VBG
its PRP
#TimeToGetOut HT
unless IN
you PRP
wanna VBP
help VB
with IN
the DT
dishes NNS
```

Reading and preprocessing POS data

```
@paulwalk USR
It PRP
's VBZ
the DT
view NN
from IN
where WRB
I PRP
'm VBP
living VBG
for IN
two CD
weeks NNS
. .
Empire NNP
State NNP
Building NNP
= SYM
ESB NNP
. .
Pretty RB
bad JJ
storm NN
here RB
last JJ
evening NN
. .
```

```python
import pandas as pd
import csv
raw_pos_df = pd.read_csv(pos_url, sep=' ', quoting=csv.QUOTE_NONE, names=['token', 'tag'])
display(raw_pos_df)
```

	token	tag
0	@paulwalk	USR
1	It	PRP
2	's	VBZ
3	the	DT
4	view	NN
...	...	...
15180	wanna	VBP
15181	talk	VB
15182	to	TO
15183	u	PRP
15184	!!!!!	.

15185 rows × 2 columns

Reading and preprocessing POS data

```python
tagged_tweets = []
current_tweet = {'tokens':[], 'tags':[]}

for row_index, row in raw_pos_df.iterrows(): # this will yield each row as a pd.Series object
    if row['token'].startswith('@'): # if we hit a new tweet...
        if len(current_tweet['tokens']) > 0: # add the current tweet to the list if it isn't empty
            tagged_tweets.append(current_tweet)
        current_tweet = {'tokens':[], 'tags':[]} #then reset the current tweet

    current_tweet['tokens'].append(row['token']) # then begin accumulating into current tweet
    current_tweet['tags'].append(row['tag'])

if len(current_tweet['tokens']) > 0: # add the current tweet to the list if it isn't empty
    tagged_tweets.append(current_tweet)

#Pandas knows how to create a DataFrame from a list of dictionaries
pos_df = pd.DataFrame(tagged_tweets)
display(pos_df)
```

Reading and preprocessing POS data

	tokens	tags
0	[@paulwalk, It, 's, the, view, from, where, I,...	[USR, PRP, VBZ, DT, NN, IN, WRB, PRP, VBP, VBG...
1	[@MiSS_SOTO, I, think, that, 's, when, I, 'm, ...	[USR, PRP, VBP, DT, VBZ, WRB, PRP, VBP, VBG, V...
2	[@robmoysey, Eyeopener, vs, ., Ryerson, Quiddi...	[USR, NNP, CC, ., NNP, NN, NN, DT, NNP, IN, CD...
3	[@RUQuidditch, #Rams, RT]	[USR, HT, RT]
4	[@ZodiacFacts, :, #ZodiacFacts, As, an, #Aries...	[USR, :, HT, IN, DT, HT, NN, VBZ, RB, IN, WP, ...
...	...	...
467	[@DailyCaller, tomorrow, !, http://is.gd/fKm4j...	[USR, NN, ., URL, PRP, VBP, RB, VBN, TO, NNP, ...
468	[@DORSEY33, lol, aw, ., i, thought, u, was, ta...	[USR, UH, UH, ., PRP, VBD, PRP, VBD, VBG, IN, ...
469	[@Bibhu2109, No, ., He, 's, gonna, run, out, o...	[USR, UH, ., PRP, VBZ, VBG, VB, IN, IN, NN, DT...
470	[@SincerelyKRenee, but, u, can, just, get, her...	[USR, CC, PRP, MD, RB, VB, PRP, NN, NN, CC, VB...
471	[@MyssLidia, :, If, u, call, someone, 5x, 's, ...	[USR, :, IN, PRP, VBP, NN, NN, VBZ, DT, NN, CC...

472 rows × 2 columns

Reading and preprocessing POS data

49 possible POS tags in this particular dataset

```
# And we can also get a canonical list of all possible tags
tags = raw_pos_df['tag'].unique()
tags
```

```
array(['USR', 'PRP', 'VBZ', 'DT', 'NN', 'IN', 'WRB', 'VBP', 'VBG', 'CD',
       'NNS', '.', 'NNP', 'SYM', 'RB', 'JJ', ':', 'URL', 'HT', 'VB',
       'VBN', 'RT', 'CC', 'TO', 'WP', ',', 'UH', 'RP', 'JJS', 'PRP$',
       'VBD', "''", 'POS', 'JJR', 'MD', 'NNPS', '(', 'WDT', 'VPP', 'EX',
       ')', 'PDT', 'RBR', 'LS', 'TD', 'FW', 'RBS', 'NONE', 'O'],
      dtype=object)
```

Reading and preprocessing POS data

```python
def token_to_ID(token):
    token = token.lower()
    if token.startswith('@'):
        return vector_model.key_to_index['<user>']
    elif token.startswith('#'):
        return vector_model.key_to_index['<hashtag>']
    elif token.startswith('http'):
        return vector_model.key_to_index['<url>']
    elif token in vector_model.key_to_index:
        return vector_model.key_to_index[token]
    else:
        return vector_model.key_to_index['<unk>']

pos_df['input_ids'] = pos_df['tokens'].apply(lambda tokens:[token_to_ID(token) for token in tokens])
```

```python
# We also need to map tags to tag IDs
tag2id = {tag:id for id, tag in enumerate(tags)}
pos_df['tag_ids'] = pos_df['tags'].apply(lambda tags:[tag2id[tag] for tag in tags])
```

Reading and preprocessing POS data

	tokens	tags	input_ids	tag_ids
0	[@paulwalk, It, 's, the, view, from, where, I,...	[USR, PRP, VBZ, DT, NN, IN, WRB, PRP, VBP, VBG...	[0, 33, 41, 13, 3056, 133, 329, 10, 57, 1704, ...	[0, 1, 2, 3, 4, 5, 6, 1, 7, 8, 5, 9, 10, 11, 1...
1	[@MiSS_SOTO, I, think, that, 's, when, I, 'm, ...	[USR, PRP, VBP, DT, VBZ, WRB, PRP, VBP, VBG, V...	[0, 10, 186, 45, 41, 92, 10, 57, 316, 56, 175,...	[0, 1, 7, 3, 2, 6, 1, 7, 8, 19, 14, 5, 12, 5, ...
2	[@robmoysey, Eyeopener, vs, ., Ryerson, Quiddi...	[USR, NNP, CC, ., NNP, NN, NN, DT, NNP, IN, CD...	[0, 519575, 917, 1, 215106, 85242, 302, 53, 12...	[0, 12, 22, 11, 12, 4, 4, 3, 12, 5, 9, 4, 4, 1...
3	[@RUQuidditch, #Rams, RT]	[USR, HT, RT]	[0, 6, 5]	[0, 18, 21]
4	[@ZodiacFacts, :, #ZodiacFacts, As, an, #Aries...	[USR, :, HT, IN, DT, HT, NN, VBZ, RB, IN, WP, ...	[0, 2, 6, 124, 172, 6, 6315, 32, 44, 121, 86, ...	[0, 16, 18, 5, 3, 18, 4, 2, 14, 5, 24, 1, 7, 5...
...	...	...	...	...
467	[@DailyCaller, tomorrow, !, http://is.gd/fKm4j...	[USR, NN, ., URL, PRP, VBP, RB, VBN, TO, NNP, ...	[0, 328, 9, 8, 10, 57, 55, 4968, 16, 218, 110,...	[0, 4, 11, 17, 1, 7, 14, 20, 23, 12, 14, 5, 1,...
468	[@DORSEY33, lol, aw, ., i, thought, u, was, ta...	[USR, UH, UH, ., PRP, VBD, PRP, VBD, VBG, IN, ...	[0, 88, 751, 1, 10, 621, 51, 93, 3427, 734, 59...	[0, 26, 26, 11, 1, 30, 1, 30, 8, 5, 3, 4, 11, ...
469	[@Bibhu2109, No, ., He, 's, gonna, run, out, o...	[USR, UH, ., PRP, VBZ, VBG, VB, IN, IN, NN, DT...	[0, 30, 1, 107, 41, 316, 899, 99, 39, 580, 191...	[0, 26, 11, 1, 2, 8, 19, 5, 5, 4, 3, 4, 26]
470	[@SincerelyKRenee, but, u, can, just, get, her...	[USR, CC, PRP, MD, RB, VB, PRP, NN, NN, CC, VB...	[0, 79, 51, 102, 59, 87, 168, 185, 148, 36, 43...	[0, 22, 1, 34, 14, 19, 1, 4, 4, 22, 7, 14, 19,...
471	[@MyssLidia, :, If, u, call, someone, 5x, 's, ...	[USR, :, IN, PRP, VBP, NN, NN, VBZ, DT, NN, CC...	[0, 2, 74, 51, 462, 238, 1193514, 41, 11, 125,...	[0, 16, 5, 1, 7, 4, 4, 2, 3, 4, 22, 1, 7, 14, ...

472 rows × 4 columns

Training a LSTM POS tagger—Dataset

```python
class POSTaggingDataset(Dataset):
  def __init__(self,
               tag_ids=None,
               input_ids=None):

    self.tag_ids = tag_ids
    self.input_ids = input_ids

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

  def __getitem__(self, idx):
    rdict = {
      'tag_ids': torch.tensor(self.tag_ids[idx], dtype=torch.int64),
      'input_ids': torch.tensor(self.input_ids[idx], dtype=torch.int64)
    }
    return rdict
```

```python
train_dataset = POSTaggingDataset(train_df['tag_ids'], train_df['input_ids'])
dev_dataset = POSTaggingDataset(dev_df['tag_ids'], dev_df['input_ids'])
```

Training a LSTM POS tagger—Dataset

```python
from pprint import pprint
pprint(train_dataset[0])
print(train_dataset[0]['input_ids'].shape)
```

```
{'input_ids': tensor([      0,       2,     525,     291,      99,      28,      80,     140,
          10510,      46,    3761,      53,     435,       9,     183,     538,
             16,   12446,      55,    1898,    1417,      15,      68,     291,
              8,      59,     121,     231,     320,       9,     211,      16,
         960077,     389, 1193514,    1352,       1,    2431,      80,     143,
           6404,      41,    3645,      35,      13,    4948,     148,       1]),
 'tag_ids': tensor([ 0, 16, 19,  1, 27, 16,  1,  7, 20,  5, 12,  3,  4, 11, 19, 15, 23, 19,
         5, 12,  2,  1,  7,  1, 17, 14, 14, 14,  4, 11,  8, 23, 12, 14, 26, 26,
        11, 14,  1, 30, 12, 32,  4,  5,  3,  4,  4, 11])}
torch.Size([48])
```

Training a LSTM POS tagger—DataLoader

```python
from torch.utils.data import DataLoader
from typing import List, Dict

# And now we will need to do padding for both the tag IDs and token IDs

def POS_collate(batch:List[Dict[str, torch.Tensor]]):
    tag_id_vectors = [example['tag_ids'] for example in batch]
    tag_id_vector_matrix = torch.nn.utils.rnn.pad_sequence(tag_id_vectors, batch_first=True, padding_value=0)

    input_id_vectors = [example['input_ids'] for example in batch]
    input_id_vector_matrix = torch.nn.utils.rnn.pad_sequence(input_id_vectors, batch_first=True,
                                                             padding_value=vector_model.key_to_index['<pad>'])

    return {
        'tag_ids':tag_id_vector_matrix,
        'input_ids':input_id_vector_matrix
    }
```

```python
batch_size = 10
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, collate_fn = POS_collate, shuffle=True)
dev_dataloader = DataLoader(dev_dataset, batch_size=batch_size, collate_fn = POS_collate, shuffle=False)
```

Training a LSTM POS tagger—DataLoader

```
first_train_batch = next(iter(train_dataloader))
print('First training batch:')
pprint(first_train_batch)

print('First training batch sizes:')
pprint({key:value.shape for key, value in first_train_batch.items()})
```

```
First training batch:
{'input_ids': tensor([[      0,      10,     247,  ..., 1193515, 1193515, 1193515],
        [      0,      10,      64,  ..., 1193514,       8,       5],
        [      0,     277,       6,  ..., 1193515, 1193515, 1193515],
        ...,
        [      0,     122,     524,  ..., 1193515, 1193515, 1193515],
        [      0,     265,      21,  ..., 1193515, 1193515, 1193515],
        [      0,       2,       6,  ..., 1193515, 1193515, 1193515]]),
 'tag_ids': tensor([[ 0,  1,  7,  ...,  0,  0,  0],
        [ 0,  1,  7,  ..., 16, 17, 21],
        [ 0, 29, 18,  ...,  0,  0,  0],
        ...,
        [ 0, 15,  4,  ...,  0,  0,  0],
        [ 0, 19,  1,  ...,  0,  0,  0],
        [ 0, 16, 18,  ...,  0,  0,  0]])}
First training batch sizes:
{'input_ids': torch.Size([10, 66]), 'tag_ids': torch.Size([10, 66])}
```

Training a LSTM POS tagger—Model

```
! pip install --quiet "pytorch-lightning==1.9.4"

# PyTorch Lightning recently released v2.0 (March 15 2023), but it changes some syntax,
# so I am teaching the last 1.9.x version for now.
# https://github.com/Lightning-AI/lightning/releases
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 827.8/827.8 KB 39.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 519.2/519.2 KB 41.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 62.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 264.6/264.6 KB 28.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.2/114.2 KB 14.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 158.8/158.8 KB 18.6 MB/s eta 0:00:00
```

Training a LSTM POS tagger—Model

```python
class LSTMPOSTagger(pl.LightningModule):
  def __init__(self,
               word_vectors:np.ndarray,
               num_classes:int,
               learning_rate:float,
               padding_id:int,
               lstm_hidden_size:int=100, # how big the inner vectors of the LSTM will be,
               lstm_layers:int =2, # how many layers the LSTM will have
               dropout_prob:float=0.1,
               **kwargs):
    super().__init__( **kwargs)

    # The __init__ function will be identical to the classifier version
    self.word_embeddings = torch.nn.Embedding.from_pretrained(embeddings=torch.tensor(word_vectors),
                                                              freeze=True)
    self.lstm = torch.nn.LSTM(input_size = word_vectors.shape[1], # The LSTM will be taking in word vectors
                              hidden_size = lstm_hidden_size,
                              num_layers=lstm_layers,
                              bidirectional=True,
                              dropout=dropout_prob,
                              batch_first=True # This is important. Set to False by default for some reason.
                              )

    # Output layer input size has to be doubled because the LSTM is bidirectional
    self.output_layer  = torch.nn.Linear(2*lstm_hidden_size, num_classes)
    self.lstm_layers = lstm_layers
    self.learning_rate = learning_rate
    self.padding_id = padding_id # we'll need this later
    self.train_accuracy = Accuracy(task='multiclass', num_classes=num_classes)
    self.val_accuracy = Accuracy(task='multiclass', num_classes=num_classes)
```

SAME!

Training a LSTM POS tagger—Model

```python
def forward(self, tag_ids:torch.Tensor, input_ids:torch.Tensor, verbose=False):

    #The first part of the forward() function is the same too
    inputs_embeds = self.word_embeddings(input_ids) #(batch size x sequence length x embedding size)
    padding_mask = (input_ids != self.padding_id).int()
    input_lengths = padding_mask.sum(dim=1).detach().cpu()
    packed_embeddings = pack_padded_sequence(inputs_embeds, input_lengths, batch_first=True, enforce_sorted=False)
    packed_output, (final_hidden, final_state) = self.lstm.forward(packed_embeddings)

    # But now we need to look at all the LSTM output, not just the final hidden state
    # So first we unpack the packed output
    output, _ = pad_packed_sequence(packed_output, batch_first=True, padding_value=0.0, total_length=input_ids.shape[1])

    # output is actually nicely shaped for us: (batch size x sequence length x 2*lstm hidden size)
    py_logits = self.output_layer(output) #(batch size x sequence length x num_classes)
    py = torch.argmax(py_logits, dim=2)

    # We end up with one loss value per token
    # Annoyingly, this function wants the class to be the second dimension
    losses = torch.nn.functional.cross_entropy(py_logits.transpose(1,2), tag_ids, reduction='none')

    # Then the final thing we need to do is zero out the losses for padding
    padded_losses = losses * padding_mask
    loss = padded_losses.mean()

    return {'py':py,
            'loss':loss}
```

Training a LSTM POS tagger—Model

```python
# And then everything else is the same!
def configure_optimizers(self):
  return [torch.optim.Adam(self.parameters(), lr=self.learning_rate)]

def training_step(self, batch, batch_idx):
  result = self.forward(**batch)
  loss = result['loss']
  self.log('train_loss', result['loss'])
  self.train_accuracy.update(result['py'], batch['tag_ids'])
  return loss

def training_epoch_end(self, outs):
  print(f'Epoch {self.current_epoch} training accuracy:', self.train_accuracy.compute())
  self.train_accuracy.reset()

def validation_step(self, batch, batch_idx):
  result = self.forward(**batch)
  self.val_accuracy.update(result['py'], batch['tag_ids'])
  return result['loss']

def validation_epoch_end(self, outs):
  print(f'Epoch {self.current_epoch} validation accuracy:', self.val_accuracy.compute())
  self.val_accuracy.reset()
```

Training a LSTM POS tagger—Model

```
tagger_model = LSTMPOSTagger(word_vectors=vector_model.vectors,
                             num_classes = len(tags),
                             learning_rate = 0.001, #I'll typically start with something like 1e-3 for LSTMs
                             padding_id = vector_model.key_to_index['<pad>'],
                             lstm_hidden_size=100,
                             lstm_layers=2,
                             dropout_prob=0.1)
print('Model:')
print(tagger_model)
```

```
Model:
LSTMPOSTagger(
  (word_embeddings): Embedding(1193516, 100)
  (lstm): LSTM(100, 100, num_layers=2, batch_first=True, dropout=0.1, bidirectional=True)
  (output_layer): Linear(in_features=200, out_features=49, bias=True)
  (train_accuracy): MulticlassAccuracy()
  (val_accuracy): MulticlassAccuracy()
)
```

Training a LSTM POS tagger—Model

```
from pprint import pprint
with torch.no_grad():
  first_train_output = tagger_model(**first_train_batch, verbose=True)

print('First training output:')
pprint(first_train_output)

print('Output item shapes:')
pprint({key:value.shape for key, value in first_train_output.items()})
```

```
First training output:
{'loss': tensor(0.2590),
 'py': tensor([[ 0,  1,  7,  ..., 12, 12, 12],
        [ 0,  1,  7,  ..., 16, 17, 21],
        [ 0, 29, 18,  ..., 12, 12, 12],
        ...,
        [ 0, 12,  4,  ..., 12, 12, 12],
        [ 0, 19,  1,  ..., 12, 12, 12],
        [ 0, 16, 18,  ..., 12, 12, 12]])}
Output item shapes:
{'loss': torch.Size([]), 'py': torch.Size([10, 66])}
```

Training a LSTM POS tagger—Trainer

```python
from pytorch_lightning import Trainer
from pytorch_lightning.callbacks.progress import TQDMProgressBar

pos_trainer = Trainer(
    accelerator="auto",
    devices=1 if torch.cuda.is_available() else None,
    max_epochs=10,
    callbacks=[TQDMProgressBar(refresh_rate=20)],
    val_check_interval = 0.5,
    )
```

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
```

Training a LSTM POS tagger—Trainer

```
pos_trainer.fit(model=tagger_model,
                train_dataloaders=train_dataloader,
                val_dataloaders=dev_dataloader)
```

```
Epoch 0 validation accuracy: tensor(0.0449, device='cuda:0')
Epoch 0 validation accuracy: tensor(0.0723, device='cuda:0')
Epoch 0 training accuracy: tensor(0.0509, device='cuda:0')
Epoch 1 validation accuracy: tensor(0.1089, device='cuda:0')
Epoch 1 validation accuracy: tensor(0.1472, device='cuda:0')
Epoch 1 training accuracy: tensor(0.1122, device='cuda:0')
Epoch 2 validation accuracy: tensor(0.1831, device='cuda:0')
Epoch 2 validation accuracy: tensor(0.2083, device='cuda:0')
Epoch 2 training accuracy: tensor(0.1895, device='cuda:0')
Epoch 3 validation accuracy: tensor(0.2194, device='cuda:0')
Epoch 3 validation accuracy: tensor(0.2348, device='cuda:0')
Epoch 3 training accuracy: tensor(0.2285, device='cuda:0')
Epoch 4 validation accuracy: tensor(0.2426, device='cuda:0')
Epoch 4 validation accuracy: tensor(0.2483, device='cuda:0')
Epoch 4 training accuracy: tensor(0.2610, device='cuda:0')
Epoch 5 validation accuracy: tensor(0.2530, device='cuda:0')
Epoch 5 validation accuracy: tensor(0.2626, device='cuda:0')
Epoch 5 training accuracy: tensor(0.2768, device='cuda:0')
Epoch 6 validation accuracy: tensor(0.2643, device='cuda:0')
Epoch 6 validation accuracy: tensor(0.2719, device='cuda:0')
Epoch 6 training accuracy: tensor(0.2812, device='cuda:0')
Epoch 7 validation accuracy: tensor(0.2737, device='cuda:0')
Epoch 7 validation accuracy: tensor(0.2753, device='cuda:0')
Epoch 7 training accuracy: tensor(0.3019, device='cuda:0')
Epoch 8 validation accuracy: tensor(0.2769, device='cuda:0')
Epoch 8 validation accuracy: tensor(0.2758, device='cuda:0')
Epoch 8 training accuracy: tensor(0.3055, device='cuda:0')
Epoch 9 validation accuracy: tensor(0.2744, device='cuda:0')
Epoch 9 validation accuracy: tensor(0.2781, device='cuda:0')
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=10` reached.
Epoch 9 training accuracy: tensor(0.3073, device='cuda:0')
```

Concluding thoughts

Sequence tagging

- POS tagging

LSTMs as a NLP Swiss army knife

Domain-specific word embeddings

Masked loss