Week 7: Sequence Labelling

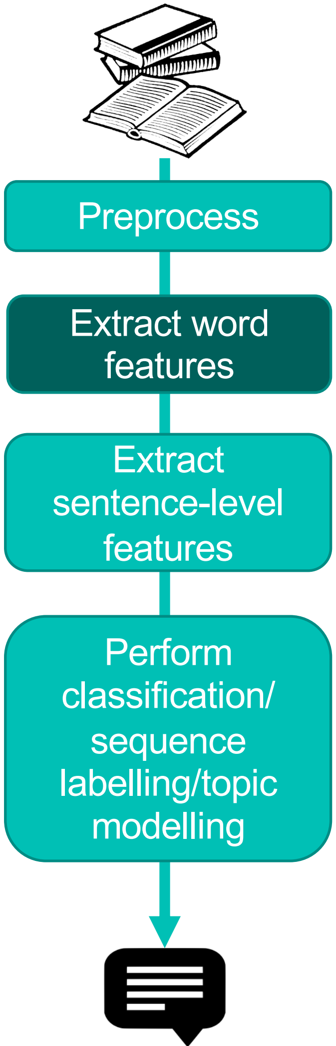
Edwin Simpson

Department of Computer Science,

University of Bristol, UK.

Parts of Speech (POS)

- Nouns, verbs, adjectives, pronouns, prepositions, …
- Important information for downstream tasks:
  - POS help identify which words relate to each other, e.g., the subject of a verb
  - Information extraction – labelling entities and events, identifying their relations from verb phrases.
  - Sentiment analysis -- roles of adjectives in expressing sentiment are very different to verbs.
- Syntactic rather than semantic: they concern how words can be used in a sentence.



POS Tagging in Different Languages

- How does POS tagging differ between languages?
- Think about languages you know in particular and what is distinctive about them.
- Answers please: https://uob.padlet.org/edwinsimpson/3rn4sk6du5hh2ejy

POS Tagging in Different Languages

- English: 55-67% tokens require disambiguation
- Czech, Hungarian, Turkish: Much more information than English in the **morphology** of the word, so use a sequence of tags for each word

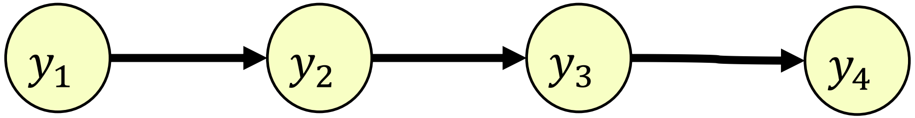
*Yerdeki **izin** temizlenmesi gerek.* → The trace on the floor should be cleaned.
POS tag of *izin* = iz+`Noun+A3sg+Pnon+Gen` [1]

- Chinese: POS tags have to be applied to multiple characters, which may combine to form new words

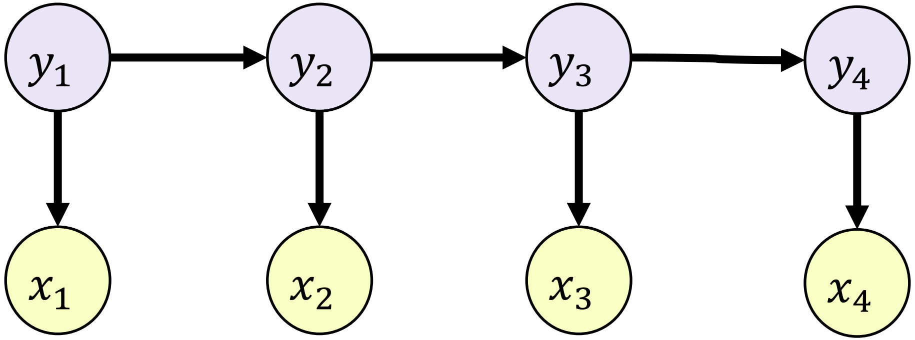
Section 8.7, Speech and Language Processing, 3rd edition draft, Jurafsky & Martin (2021).
[1] Hakkani-Tür, D., et al. (2002). Statistical morphological disambiguation for agglutinative languages.

Markov Models and Hidden Markov Models

Markov assumption: $P(y_i|\boldsymbol{y}_{-}) \approx P(y_i|y_{i-1})$



Hidden Markov model (HMM): the states (y variables) are hidden and we observe x instead.



Transition Matrix

```python
for sentence_tags in tqdm(train_tags_encoded):
    for i, tag in enumerate(sentence_tags):
        if i==0:
            start_states[tag] += 1
            continue
        ### WRITE YOUR OWN CODE HERE
        transitions[sentence_tags[i-1], tag] += 1
```

```python
### WRITE YOUR CODE HERE
transitions /= np.sum(transitions, 1)[:, None]
start_states /= np.sum(start_states)
```

Maximum likelihood parameter estimate

$$P(y_i = c | y_{i-1} = d) = \frac{\text{num\_tokens\_with\_tag\_c\_preceded\_by\_d}}{\text{total\_num\_toks\_with\_tag\_d}}$$

Learning the Observation Model

Maximum likelihood parameter estimate

$$P(x_i = w | y_i = c) = \frac{count(w \mid c) + 1}{\sum_{w' \in V}(count(w' \mid c) + 1)}$$

```
observations = np.zeros((num_tags, V))

for i, sentence_toks in tqdm(enumerate(train_toks_encoded)):
    sentence_tags = train_tags_encoded[i]
    for j, tok in enumerate(sentence_toks):
        tag = sentence_tags[j]
        # WRITE YOUR OWN CODE HERE
        observations[tag, tok] += 1
```

```
#WRITE YOUR OWN CODE HERE
observations /= np.sum(observations, 1)[:, None]
```

Lin, C. C., et al. (2015, January). Unsupervised POS Induction with Word Embeddings. *HLT-NAACL*.

Learning the Observation Model

- Vector semantics: represent words as numerical vectors called "word embeddings" learned using neural networks
- Embeddings encode similarities between words
- Word embeddings as Gaussian-distributed observations:

Maximum likelihood parameter estimates

$$P(\vec{\boldsymbol{x}}_{\boldsymbol{i}} = w | y_i = c) = \mathcal{N}(\vec{\boldsymbol{x}}_{\boldsymbol{i}}, \boldsymbol{\mu}_{\boldsymbol{c}}, \boldsymbol{\Sigma}_{\boldsymbol{c}})$$

Lin, C. C., et al. (2015, January). Unsupervised POS Induction with Word Embeddings. *HLT-NAACL*.

Decoding as Argmax

Input: a sequence of tokens

Possible sequence of labels

Most probable label sequence

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})}{\operatorname{argmax}} \Psi(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

Viterbi algorithm performs argmax to find the most probable sequence.

Probability **of a sequence**
Which equation do we need to compute the probability of a sequence?

Decoding: Viterbi

```
# probabilities for all the sequences leading to this
seq_prob = V[t-1, :] * transition_probs[:, state]

# Choose the most likely sequence
max_seq_prob = np.max(seq_prob)
best_previous_state = np.argmax(seq_prob)

# Calculate the probability of the most likely sequence leading to this state at time
# Add eps to help with numerical issues.
V[t, state] = (max_seq_prob + eps) * (observation_probs[state, observed_seq[t]] + eps)

backpointer[t, state] = best_previous_state
```

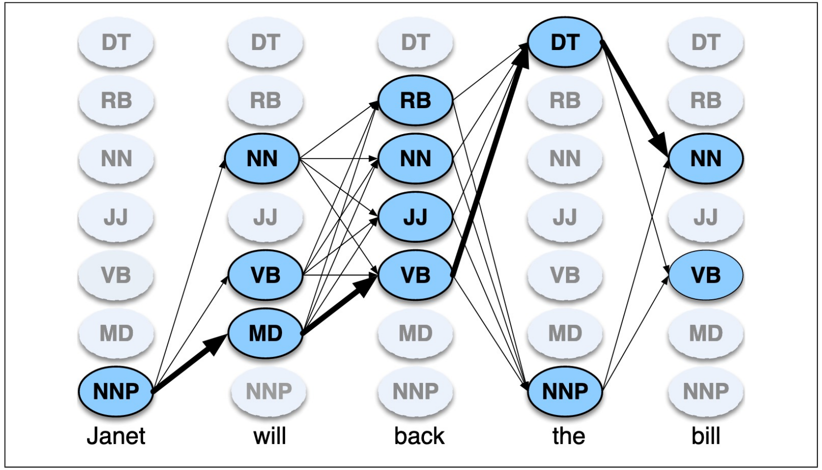
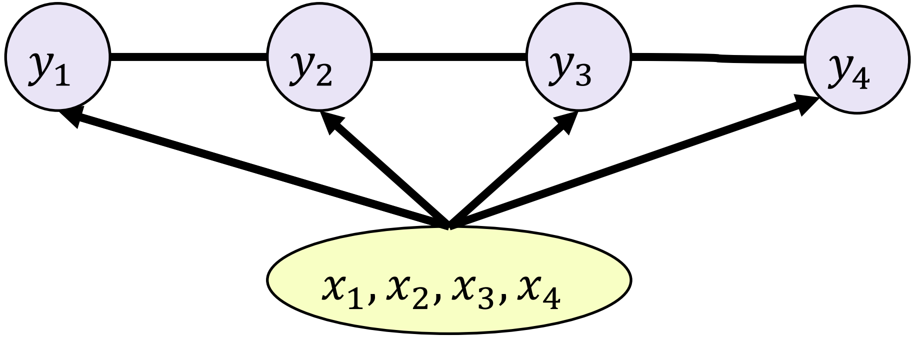


Figure 8.6 A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the *B* matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

Section 8.4, Speech and Language Processing (3rd edition draft), Jurafsky & Martin (2021).

Discriminative Models: CRF

- Conditional Random Field (CRF) is **discriminative**:
  - Optimises predictive distribution $p(\boldsymbol{y}|\boldsymbol{x})$
  - Related discriminative approach for classification: Logistic regression



Section 8.5, Speech and Language Processing (3rd edition draft), Jurafsky & Martin (2019).

CRF Prediction Function

Directly computes probability of the sequence

Weights like in logistic regression

$$P(\boldsymbol{y}|\boldsymbol{x}) \propto \exp\left(\sum_{k=1}^{K} \theta_k F_k(\boldsymbol{x}, \boldsymbol{y})\right)$$

Global feature function to compute the feature from the sequence ***x***

Section 8.5, Speech and Language Processing (3rd edition draft), Jurafsky & Martin (2019).

Feature Functions

Global function for feature *k*

$$F_k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{N} f_k(y_{i-1}, y_i, \boldsymbol{x}, i)$$

Local function can use previous tag, current tag, whole token sequence, and current position.

```python
# Capitalization
if token[0].isupper():
    feature_list.append("CAPITALIZATION")

# Number
if re.search(self._pattern, token) is not None:
    feature_list.append("HAS_NUM")

# Punctuation
punc_cat = {"Pc", "Pd", "Ps", "Pe", "Pi", "Pf", "Po"}
if all(unicodedata.category(x) in punc_cat for x in token):
    feature_list.append("PUNCTUATION")

# Suffix up to length 3
if len(token) > 1:
    feature_list.append("SUF_" + token[-1:])
if len(token) > 2:
    feature_list.append("SUF_" + token[-2:])
if len(token) > 3:
    feature_list.append("SUF_" + token[-3:])


# Current word
feature_list.append("WORD_" + token)

### WRITE YOUR OWN CODE HERE ###
if idx > 0:
    feature_list.append("PREVWORD_" + tokens[idx-1])
if idx < len(tokens)-1:
    feature_list.append("NEXTWORD_" + tokens[idx+1])

####
```

University of BRISTOL

Quiz