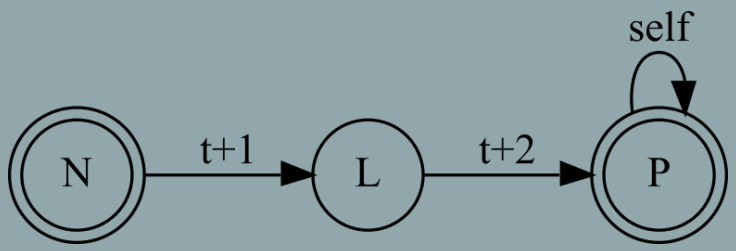
LING-362

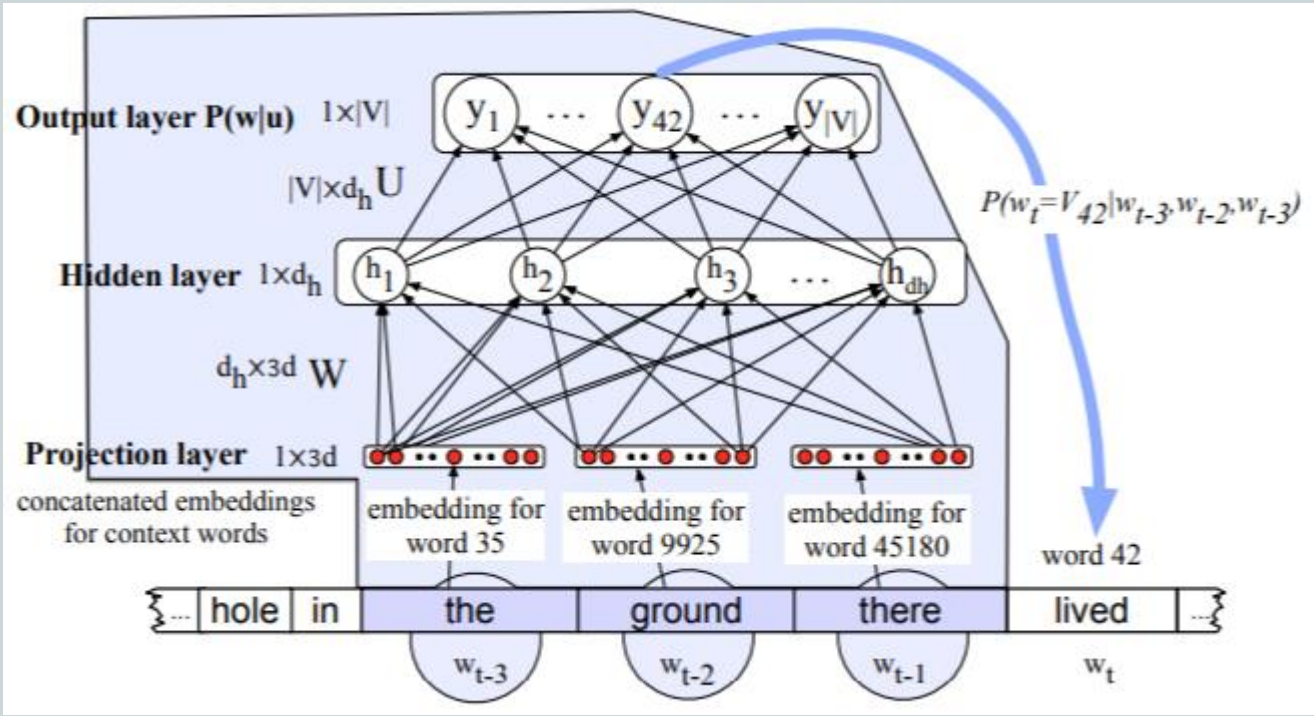
Introduction to Natural Language Processing

Tagging and Hidden Markov Models

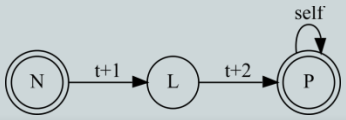


Deep Learning in language models

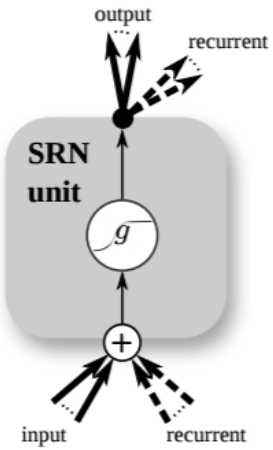
- In a feed forward network we could do:

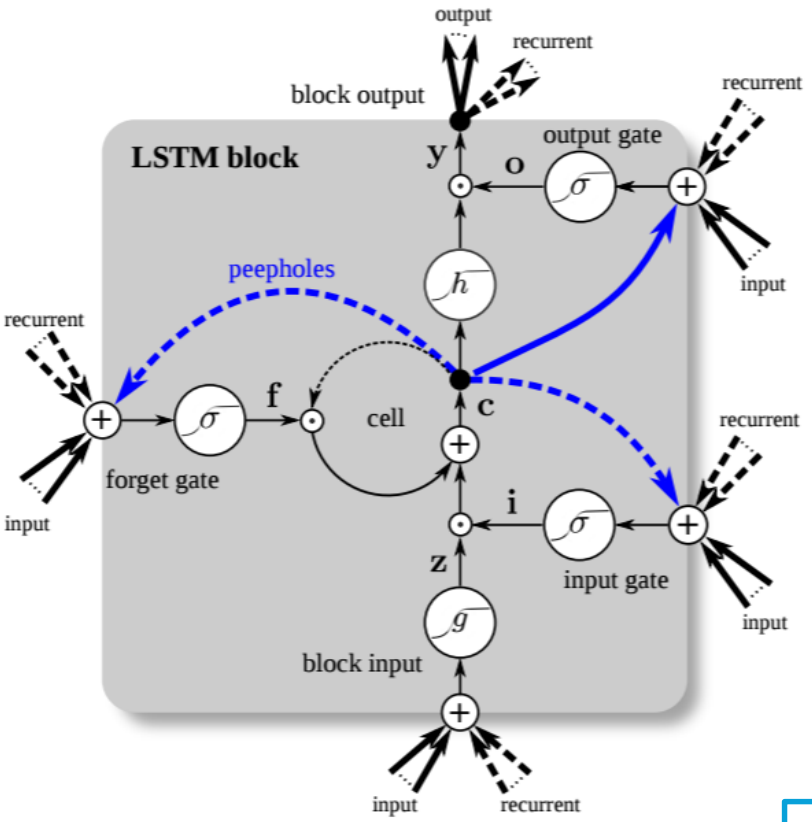


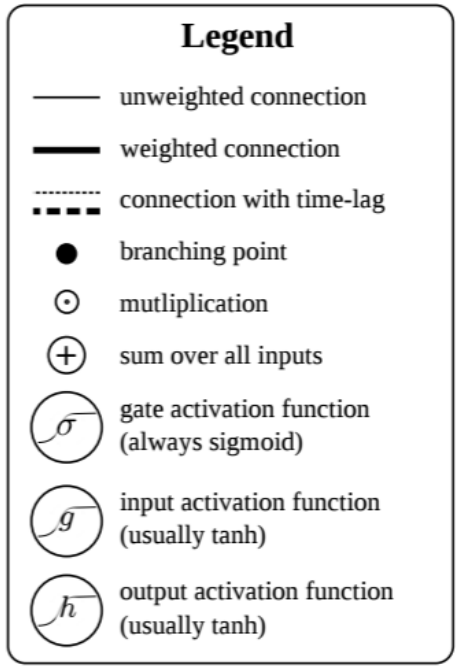
Jurafsky & Martin (2017)



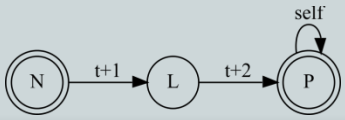
LSTMs are even more complex







Greff et al. (2015)



What are these cells learning?

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

Cell that turns on inside quotes:

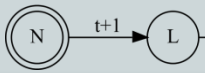
```
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

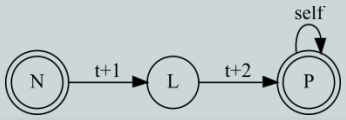
A large portion of cells are not easily interpretable. Here is a typical example:

```
/*  Unpack a filter field's string representation from user-space
 *  buffer.  */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /*  Of the currently implemented string fields, PATH_MAX
   *  defines the longest valid length.
   */
```



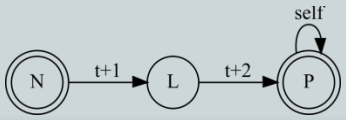
Training your own

- A relatively simple model works out of the box using PyTorch:
  - pip install torch
  - https://github.com/pytorch/examples/tree/master/word_language_model
- Other good libraries: Tensorflow, Keras

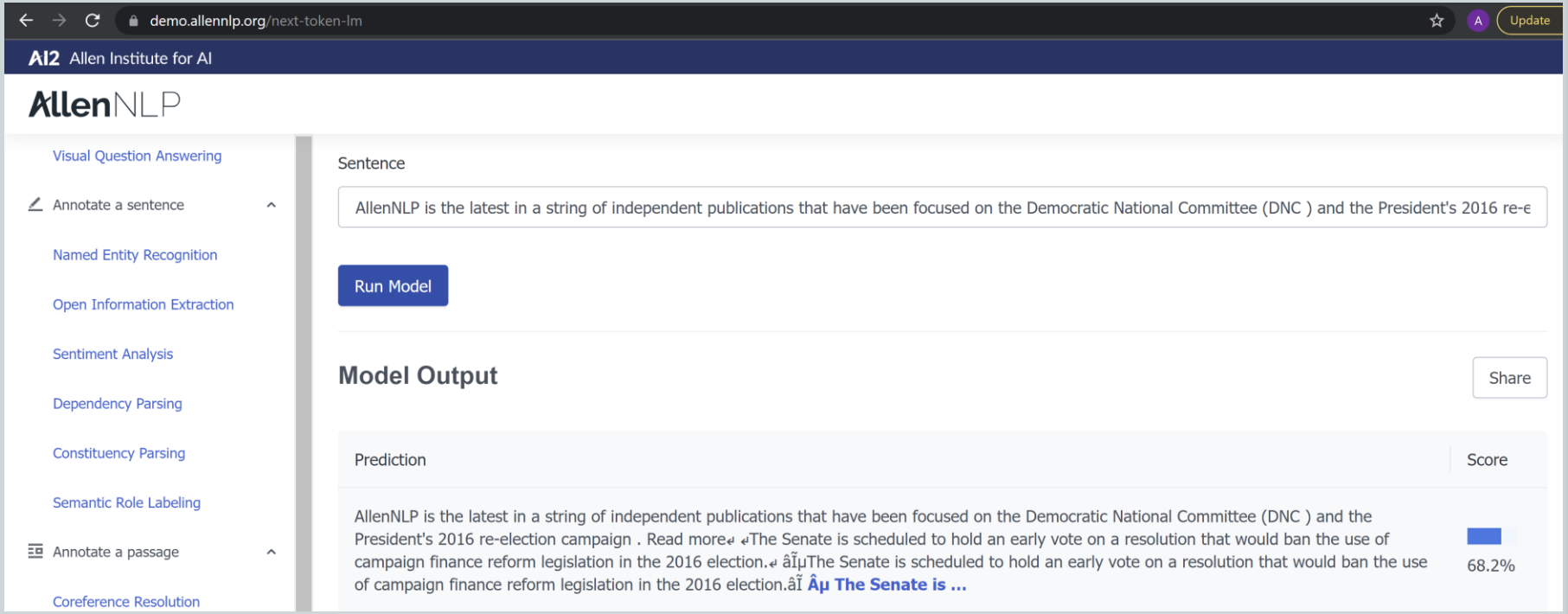


Bonus fun

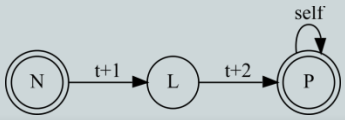
- You can test AllenNLP's neural LM here:
  - https://demo.allennlp.org/next-token-lm
- And you can chat with a neural network trained on conversational pairs
- Example:
  - http://neuralconvo.huggingface.co/
  - (also compare Microsoft's TAY: https://twitter.com/tayandyou)



Do neural LMs solve all problems?

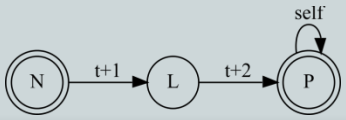


Generated on 2021-10-18



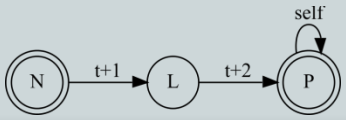
More information

- We will learn more about practical applications of neural networks later
- Learning how neural models work in depth is outside the scope of this course
  - **Jurafsky & Martin 2017, C7** (Canvas) is a good starting point
  - Grad students: once you are confident in coding, consider taking LING-504/COSC-576
- Further reading:
  - *Hands-on Machine Learning with Scikit-Learn and TensorFlow* / A. Geron, 2019 (https://github.com/ageron/handson-ml)



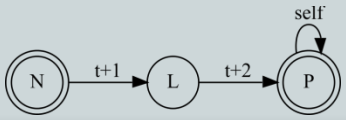
A more abstract view of ngrams

- What do language modes really 'model'?
  - Probabilities of individual words
  - Probabilities of sequences of words
- How is our language model using them?
  - Get **transitional** ~~possibilities~~ probabilities
  - What are the odds of moving **from word X to word Y**?
- *We've seen something like this before…*



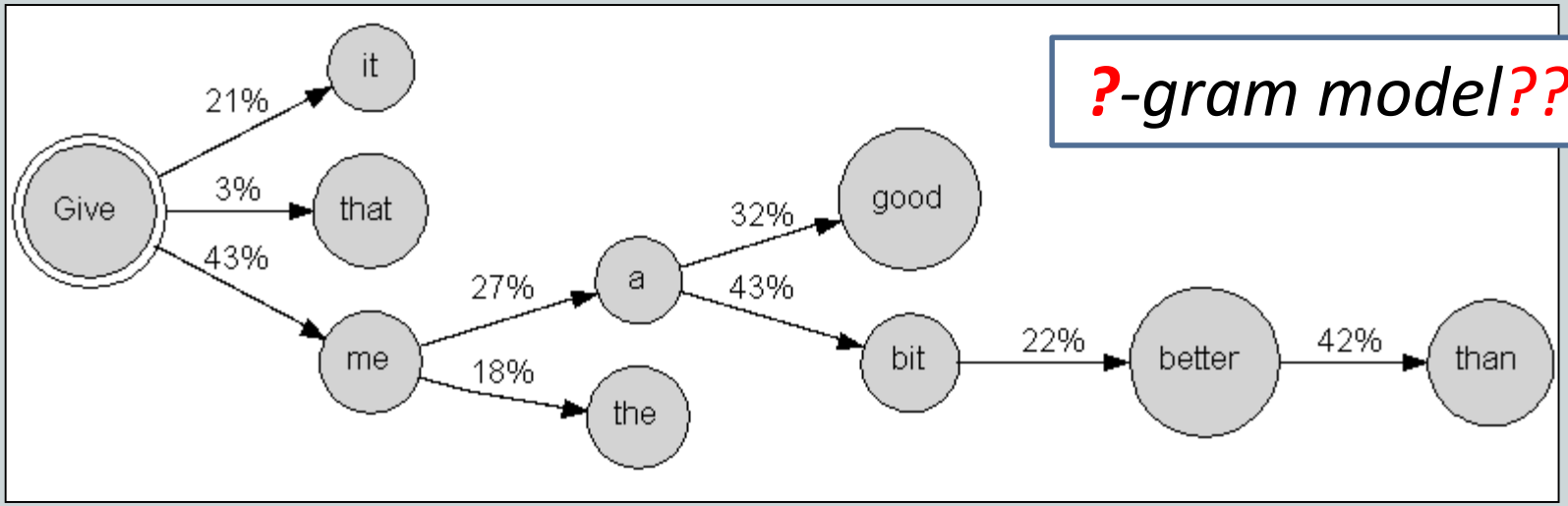
Transitional probabilities

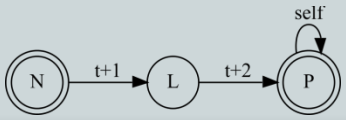
- Moving through a language model is like progressing in a web of words:
  - Let's say I type the word "Give" into my smartphone's message app
  - This is a job for the auto SMS wizard! ☺
  - Here's what happens when I click next, next…
    - *Give me a bit better than the bus and should be there in a couple of days ago…*



Language models as FSAs

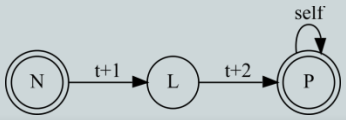
- Language model choices are probabilistic – different from deterministic FSAs
- But we can represent them as **weighted** automata:





Markov Chains

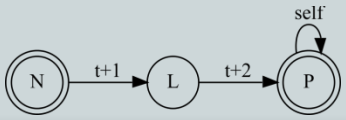
- A set of ordered variables with probabilities following the **Markov Property:**
  - The probability of each value of $\boldsymbol{X_i}$ in the sequence depends **only** on $\boldsymbol{X_{i-1}}$
  - (Or in variants: some other sufficiently small number: ***second order Markov Model***, ***third order***… etc.)
  - In other words: context effects are limited, but can chain
  - Formally: $P(\boldsymbol{X_i}=x \mid \boldsymbol{X_{i-1}}=x_1, \boldsymbol{X_{i-2}}=x_2,\ldots) = P(\boldsymbol{X_i}=x \mid \boldsymbol{X_{i-1}}=x_1)$
- This is a shameless, but very useful simplification! ☺



An example

- Suppose the difficulty of a homework assignment is influenced by the previous one
  - If the last assignment was easy, this next one will be hard with 70% probability (but 30%: easy)
  - If the last one was hard, 60% that the next will be easy (but 40%: still hard)
- Results are uncertain, but depend **only on last time**
- Globally we still model a process where difficulty alternates **across the chain**

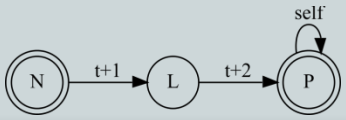
➢ **This works not just for words!!**



The Markov property assumption

- Note n% transition depends only on the current state
- We get a 'plausible' sequence overall
- What we need for this:
  - Probabilities of each $P(w_k|w_{k-1})$
  - Smoothing for missing values

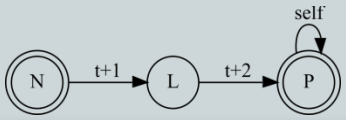
- We know how to get these for words, but what about other categories?



Beneath the surface

- Token n-gram models represent transitions between **actually observed characters/words**
- We can call them **Visible Markov Models (VMMs)**

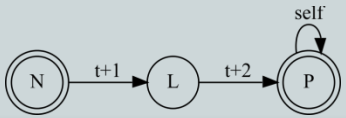
- Besides properties that are overt, we are often interested in the probabilities of **hidden** categories
- These will require **Hidden Markov Models (HMMs)**



Invisible categories

- Which categories are hidden?
  - We may not be interested in a specific adjective like ***number (than)***
  - We might want to know the likelihood of **any** comparative adjective at this position
  - Or the probabilities that words refer to a company, or have positive sentiment, or …
  - How can we look at categories that are not in the data explicitly?

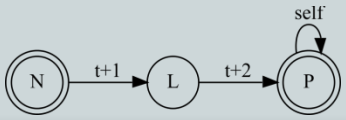
- Let's look at an example



POS tagging

- Probably the most widely used ‘hidden’ category in NLP
  - Assume each token has exactly 1 correct part of speech
  - We can’t see it, but it’s there
  - If we knew the POS tags of a text, we could create n-gram models describing them:
    - ART ADJ N → NP trigram!
    - TO ADV V → split infinitive!

- What tags are there?



Tag sets for English

Common in the US:

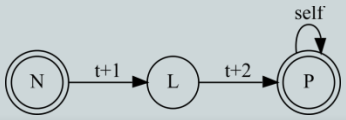
- Penn Treebank Tagset (PTB) — 36 Tags
- Extended PTB (AMALGAM/TT) — 57 Tags

Common in the UK:

- CLAWS 5 — 62 Tags
- CLAWS 7 — 137 Tags

Other notable mentions:

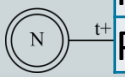
- Brown tag set — 85 Tags
- Google “Universal Tags” (V2) — 17 Tags



The PTB tag set (vanilla)

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

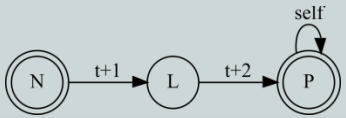
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person sg. present
VBZ	Verb, 3rd person sg. present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb



Tagging exercise

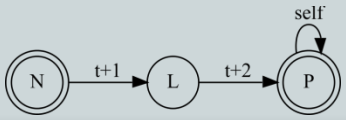
- Tag the text: ***Is ISIS Going Broke?***
  - What's easy and what's hard?
  - How do we determine the correct tag?
  - How can a computer do it?

- If you have a printer handy or can annotate on screen easily (e-pen) – use the PDF
- Otherwise try the Excel spreadsheet and put tags in the second column!



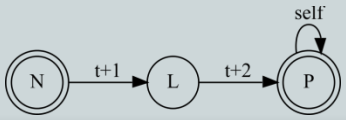
The PTB tag set

- There is a lot to be said about the PTB tag set
  - Successes and shortcomings
  - Extensions since its inception – notably through the AMALGAM project (2001), TreeTagger, OntoNotes, English Web Treebank…
- We don't have time to discuss these…
- For this course: PTB (a.k.a. vanilla PTB) will be our only tag set for English (more: LING-367)



How does a POS tagger work?

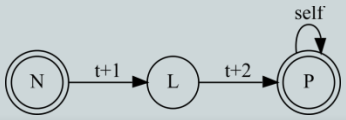
- To decide what POS tags to assign, an automatic tagger consults training data
  - Known POS distributions
  - Known conditional probabilities *P(POS2|POS1)*
  - …
- We can get initial probabilities for a single word…
  - but the tagger can also continue a plausible sequence
  - First let's see what we can do with a tagger using NLTK



Tagging with NLTK – tagging.py

```
import nltk
text = "Mr. Pickwick turned azure."

# Get a list of tokens
tokenized = nltk.word_tokenize(text)

# Make it a list of (token, pos) tuples
tagged = nltk.pos_tag(tokenized)

print(tagged)
# Note the error!
```

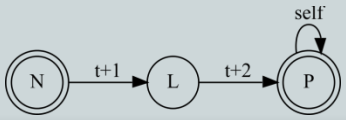
```
[('Mr.', 'NNP'), ('Pickwick', 'NNP'), ('turned', 'VBD'),
('azure', 'NN'), ('.', '.')]
```



What can we do with ‘tagged’?

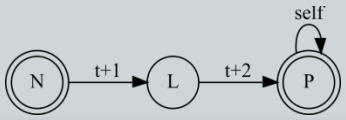
- Tagged data is a **list** of **tuples**
- Can be separated into **two lists** for processing just tags:

```
words = []
tags = []
for word, tag in tagged:
    words.append(word)
    tags.append(tag)
```



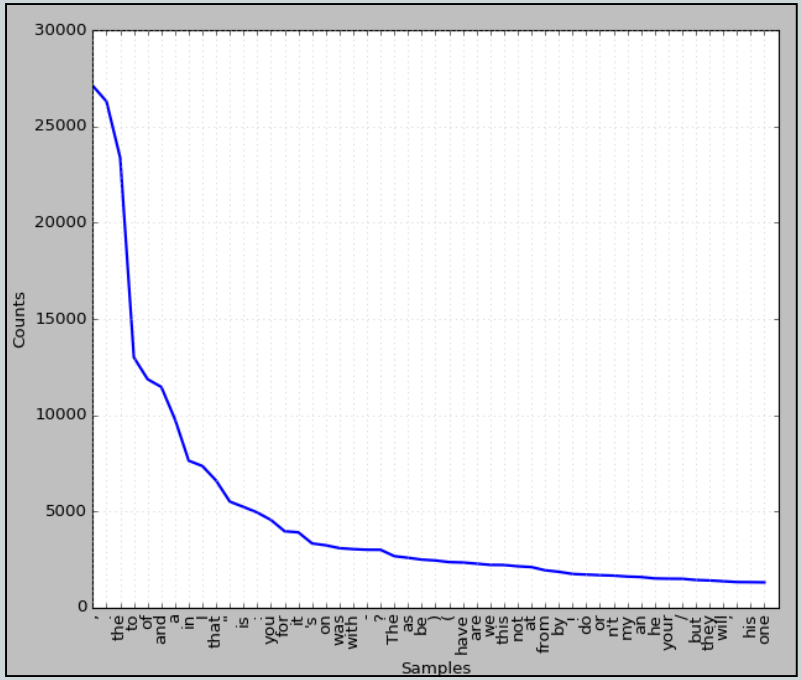
Inspecting frequencies

```
tag_dist = nltk.FreqDist(tags)
word_dist = nltk.FreqDist(words)
print(tag_dist.most_common(4))
print(word_dist.most_common(100)[90:])
```



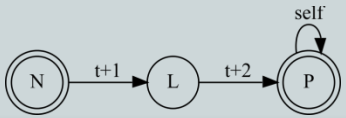
And plotting

- To plot you must install Matplotlib and NumPy from the command line:
  - ***> pip install numpy***
  - ***> pip install matplotlib***



- Then in Python:

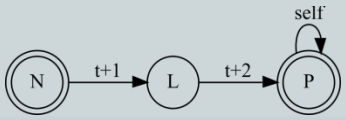
```
word_dist.plot(50)

word_dist.plot(50, cumulative=True)
```



Tagged data from nltk

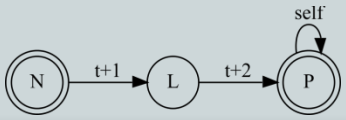
- You can also get some ready corpora from NLTK:

```
from nltk.corpus import masc_tagged

words = []
tags = []
for word, tag in masc_tagged.tagged_words():
    words.append(word)
    tags.append(tag)
```



Homework – for Wednesday, Nov 3

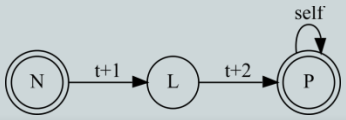
- Write a program that:
  - Takes a text file input
  - Tokenizes and tags the text with NLTK
  - Goes through the result and saves **only common nouns**
  - Prints the most frequent 10
  - Plots the top 100 using nltk's FreqDist
- Run your program on some text
- Find one word mistagged as a noun and write as a comment – why do you think this happened?



Exercise – verb proportions

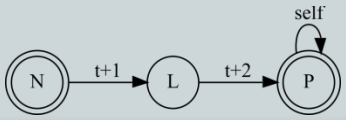
- Try to get the proportion of verbs in some text:
  - Read a text file saved from the Web
  - Tokenize and tag it
  - Loop through data and get:
    - Length in tokens
    - Amount of verbs
    - Print proportions with a verbal tag using one of:

```
if tag in ["VB","VBZ",...]:
        ...
if re.match(r'V.*', tag) is not None:
        ...
if tag.startswith("V"):
```



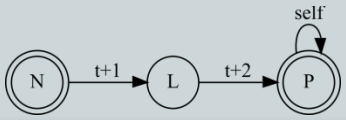
We can also get this by sentence

```
import argparse
from nltk import word_tokenize, pos_tag,
sent_tokenize
```



We can also get this by sentence

```
parser = argparse.ArgumentParser()
parser.add_argument("file")
options = parser.parse_args()

with open(options.file, 'r') as f:
   text = f.read()

sentences = sent_tokenize(text)
sent_num = 0
```



We can also get this by sentence

```
for sentence in sentences:
    tokens = nltk.word_tokenize(sentence)
    tagged = nltk.pos_tag(tokens)
    length = len(tokens)
    verbs = 0
    sent_num += 1
    for token, tag in tagged:
        if tag.startswith('V'):
            verbs += 1
    print("Verb ratio for S" + str(sent_num) + ": " +
                str(float(verbs)/length))
```