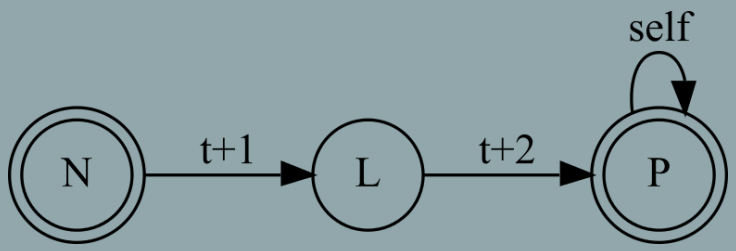
LING-362

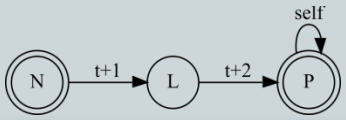
Introduction to **N**atural **L**anguage **P**rocessing

A Shallow Introduction to
Neural Language Models (ctd.)



Talk announcement

- JHU, Center for Language and Speech Processing seminar
  - Friday, October 22 at 12:00pm ET
  - **https://wse.zoom.us/j/93327212503**
  - **Reno Kriz –** Towards a Practically Useful Text Simplification System



Perplexity (PP)

- Measured in general:

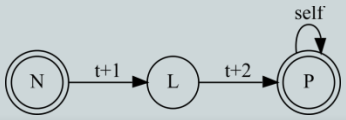
$$PP(text) = \sqrt[N]{\frac{1}{P(w_1 \ldots w_N)}}$$

- For a bigram model:

$$PP(text) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

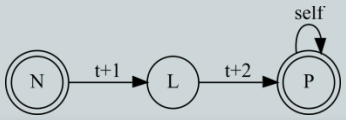
- For a trigram model:

$$PP(text) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})P(w_{i-1}|w_{i-2})}}$$



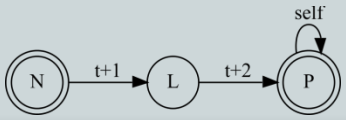
Smoothing

- To avoid multiplication by zero we need some probability for **OOV items**
- Common solutions:
  - Laplace (a.k.a. “add 1”) smoothing (very skewed)
  - Delta smoothing (“add 0.000….1”, a little skewed)
  - Good-Turing Discounting (actual probabilities, based on dataset properties)



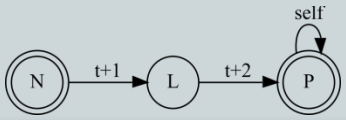
Further reading

- This was a brief introduction to ngram models:
  - We can calculate probabilities for higher order models (bigram, trigram, n-gram model)
  - Our ngrams code could do better smoothing (Good-Turing)
  - For n-grams, we can use shorter grams if longer ones are OOV (a.k.a. ***backoff*** *models*), or incorporate weights from all attested n-gram lengths (***interpolation***)
  - Use **variable length n-grams**
- Recommended reading: Jurafsky & Martin (2017, C4 – [at least] pages 1-16)



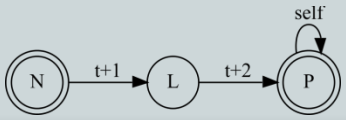
Contemporary language models

- N-gram models were (and are) used for a long time, give reasonable results with small datasets
- But it's 2021 and we need to talk about Neural Networks…
  - Reliance on machine learning to find best model
  - Deep Learning architectures allow special conditions to be learned for huge numbers of interacting features – not just last 2 words
  - Numerical representation of words allows defaulting to 'similar' words
  - Memory based architectures let the computer 'remember' having seen something to trigger different behavior
  - Use of attention weights to prioritize different cues



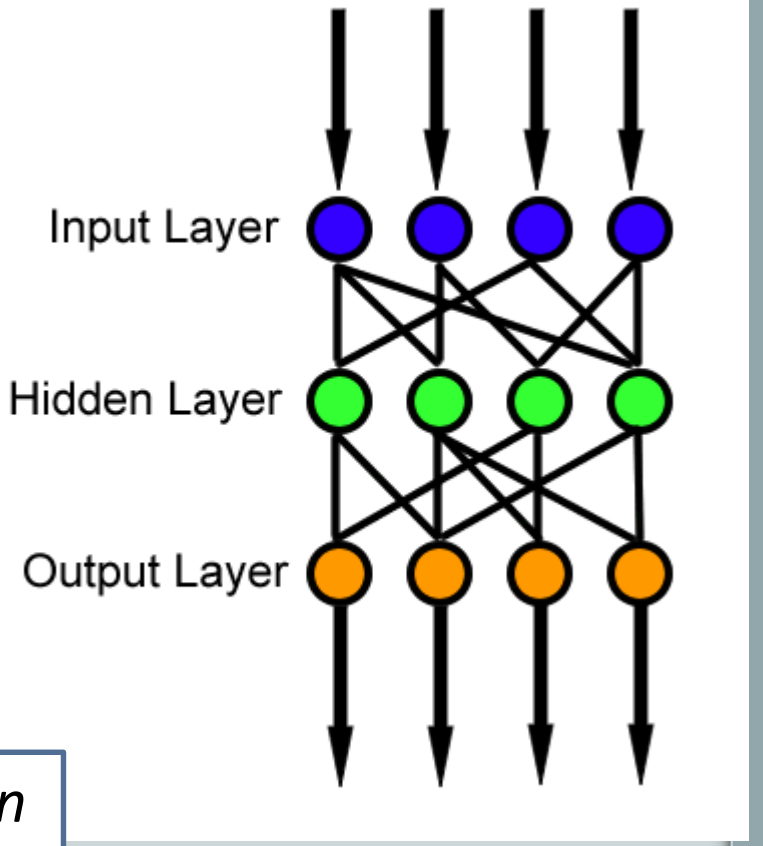
Contemporary language models

- Our discussion will be necessarily **shallow**
- Theoretical overview available in Jurafsky & Martin (2017, C7)
- For more with practical examples in Python I recommend working through:
  - *Hands-on Machine Learning with Scikit-Learn and TensorFlow* / A. Geron
  - https://github.com/ageron/handson-ml
- For grad students especially: consider more advanced ML courses (LING-504 in spring)

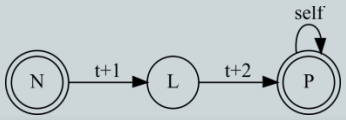


Feed forward networks

- Basic neural networks:
  - Take a bunch of inputs
  - Activate ‘synapses’
  - Propagate activation forward
  - Fire some output
- Input and output can be anything
- Word example:
  - Input: ***the***
  - Output: ***cat***

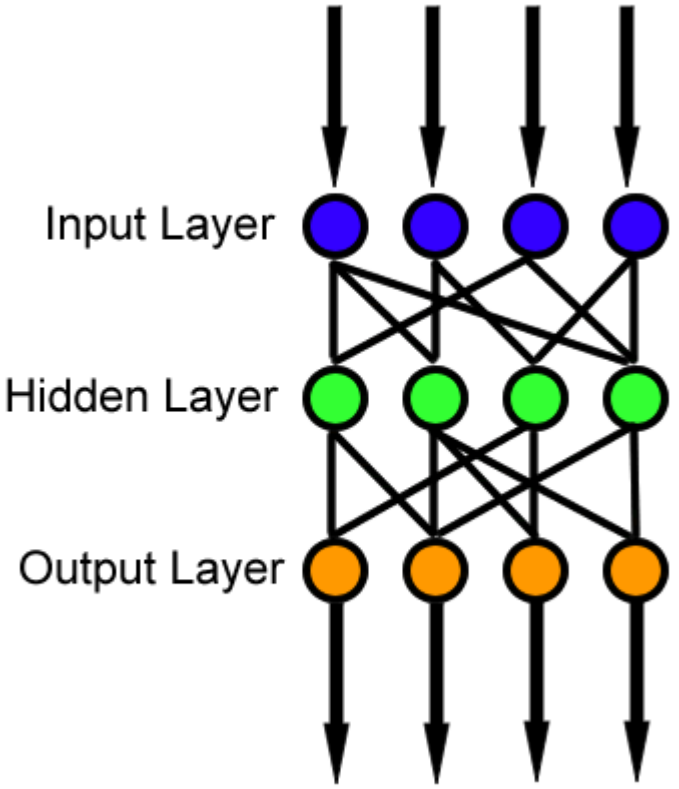


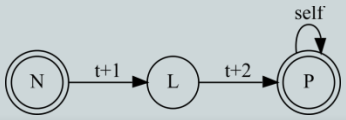
I saw the cat on the mat…



Back-propagation

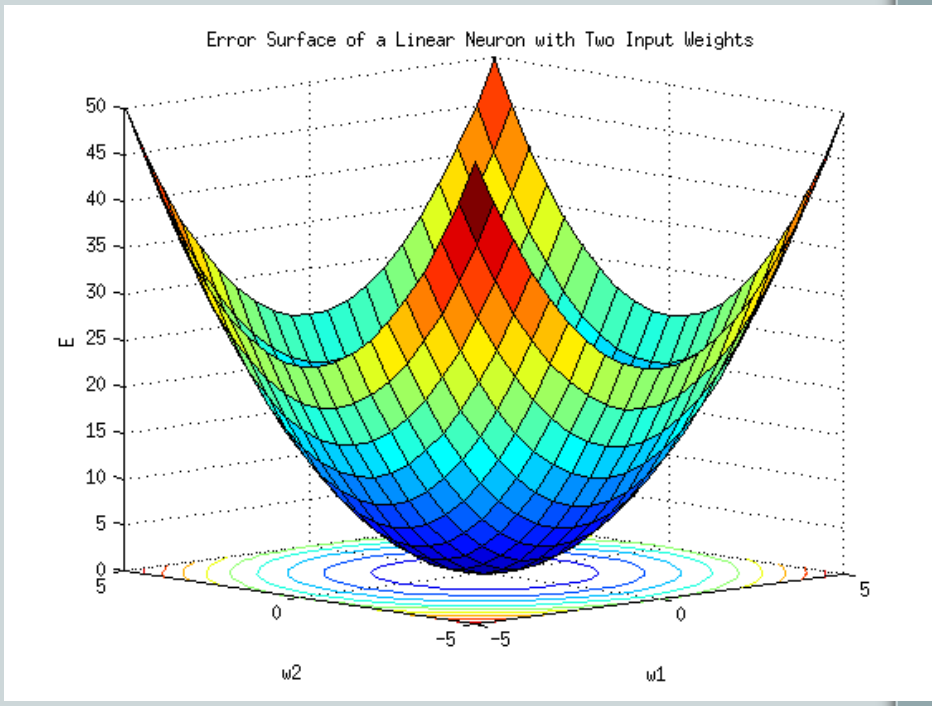
- How do they learn?
  - Whenever the output is wrong we apply a cost to all activating inputs
  - Propagate back in the network
  - Weights are modified
- Word example:
  - Input: ***the***
  - Output: ***the*** -> all active input links to ***the*** are penalized

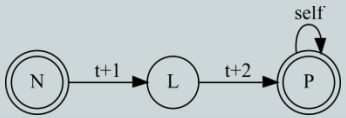




Gradient descent

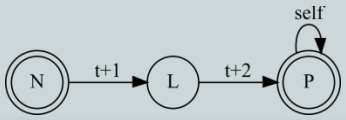
- How can we find the best weights?
  - Make fewest mistakes
  - Track derivative of cost (loss function)
  - If cost is getting less, all is well
  - If cost is getting higher, correct in other direction, until network converges on a minimal error





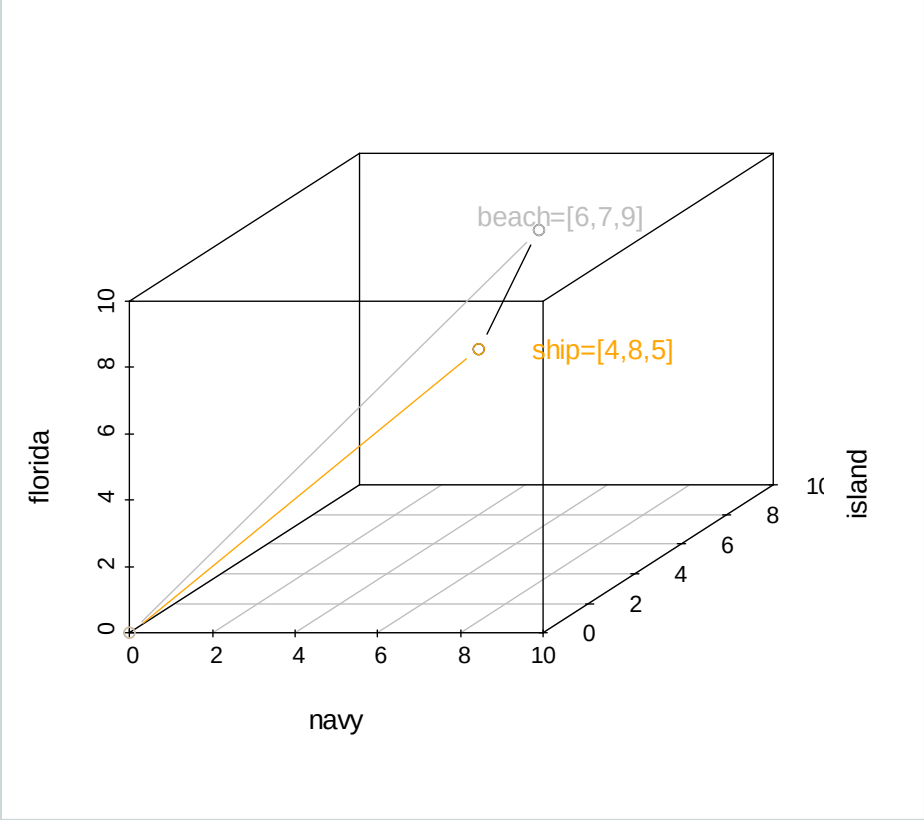
Is this a glorified lookup table?

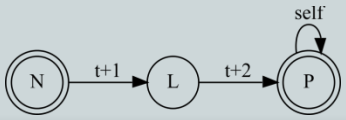
- Although neural networks are very impressive, they are only as good as input/output features
  - Mapping words to words is not that amazing
  - Getting from words to sentences is still a problem
  - Many words will be **OOV** (out of vocabulary)
- State of the art neural LMs use many tricks to solve these problems



Vector space models / embeddings

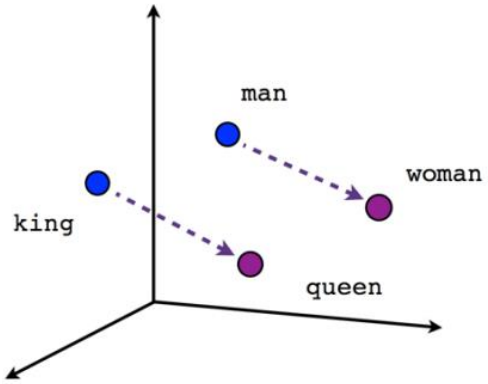
- Co-occurrence frequencies (or transformations thereof) make a vector space
- Allows similarity metrics for words and documents
- Models of meaning based on neighboring words



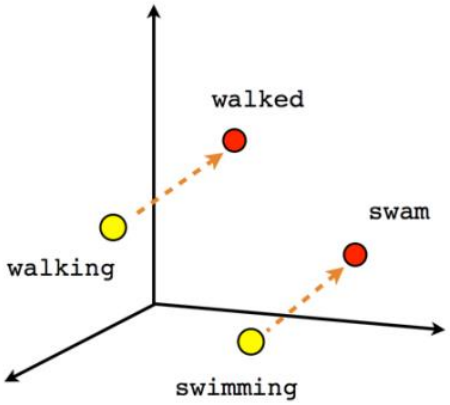


Applications

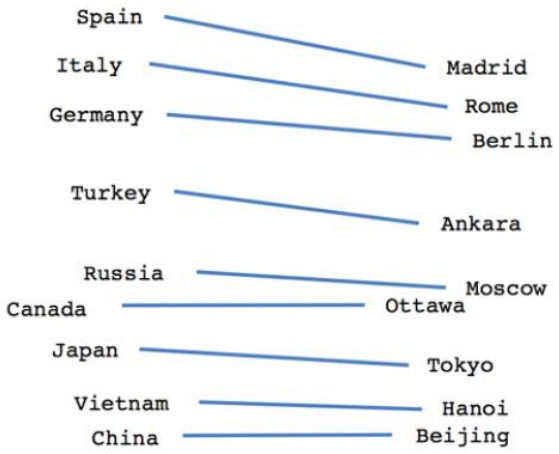
- Projecting vectors to lower dimensions
  - Reveal systematic relationships
  - Word level similarity



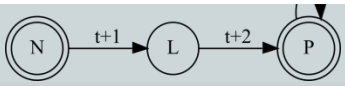
Male-Female



Verb tense



Country-Capital



Word2Vec

Don't count, predict! (see Baroni et al. 2014)

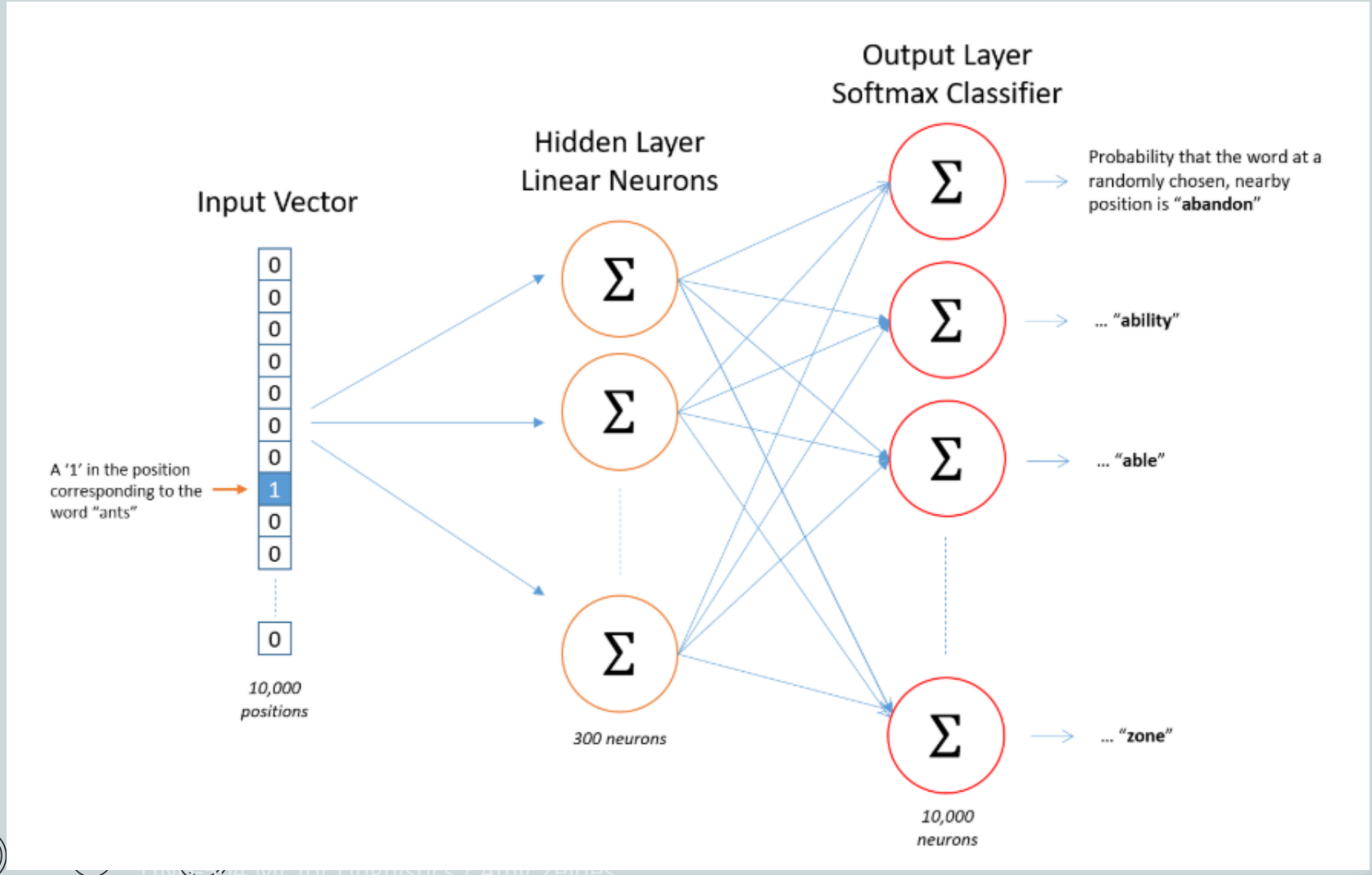
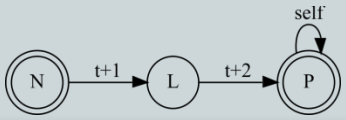


Image: mccormickml.com

Fine grained meaning

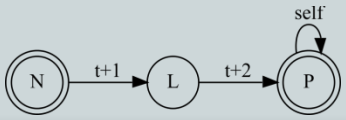
- Vector Space Models can also be used to represent word meaning:
  - Approaches to Distributional Semantics (Harris 1954)
  - *The meaning of a word is its usage in the language* (Wittgenstein 1953)
  - *You shall know a word by the company it keeps* (Firth 1957)

- We now have the data and computing power to realize this
- Recommended advanced readings: Baroni & Zamparelli 2010, Bruni et al. 2012, Baroni et al. 2014



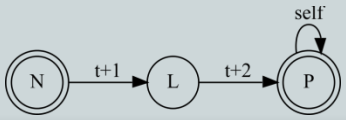
Word2Vec

- Implementation of distributional semantics (Mikolov et al. 2013)
  - Use a window of ±2 tokens
  - Track co-occurrence of target word with its nearest neighbors
  - Vector space as a matrix of each word vs. its potential neighbors:
    - cactus – Christmas : close to no co-occurrences in window
    - Queen – England: many co-occurrences



Skip-gram model

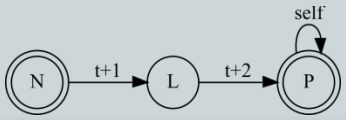
- Train a neural network to use target word vectors to predict context words
  - Problem re-dressed as a binary classification task
  - For some candidate word: is it a neighbor of our word or not?
    - Mix positive examples: Queen -> England? YES
    - And negative ones: Queen -> curry? NO
  - Alter the input vector representation as a result
- Final vectors can be used to rank similarity



Is there one answer?

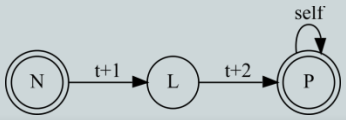
- What would you expect the vector space to tell you?
  - Most similar word to: **Android**
  - **lemon** is to **orange** as **apple** is to …?
  - Which is different?
    lemon orange apple cucumber

- Try the demo here:
  - https://rare-technologies.com/word2vec-tutorial/



Despite some weaknesses…

- **Word vectors** or **embeddings** are a very potent tool
  - Relatively easy to get and use
  - Unparalleled access to arbitrary word meaning
  - Very good at getting similarity in some contexts
  - State of the art for dealing with '**OOV**' items (Chen & Manning 2014) – as long as they're not OOV in the corpus used to train embeddings
  - Allows context to compensate for missing terms (esp. recent architectures, e.g. **ELMo, BERT, XLNet** – Peters et al. 2018, Devlin et al. 2018, Yang et al. 2019)



Examples in Python

- Implementation in Gensim by Radim Řehůřek

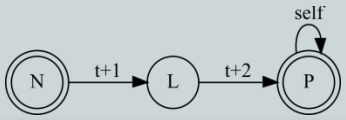
  > pip install scipy

  > pip install gensim

  - For Win 64 install scipy manually if needed from: http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy
    - Download scipy-1.7.1-cp39-cp39-win_amd64.whl

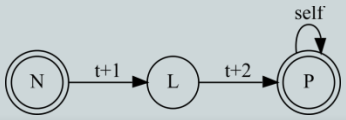
    > pip install scipy-1.7.1-cp39-cp39-win_amd64.whl

  - Or install Visual C++ 2015 Build Tools for compilation



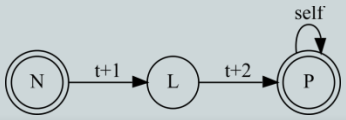
Training a model – toy example

```
from gensim import models

sotu = "sotu_sent_per_line.txt"

with open(sotu,'r',encoding="utf8") as f:
  plain_text = f.read()


sentences = plain_text.split("\n")
```



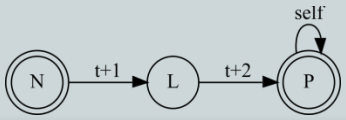
Training a model – toy example

```
tokenized = []

for sentence in sentences:
   tokens = sentence.strip().lower().split(" ")
   tokenized.append(tokens)

model = models.Word2Vec(tokenized, min_count=2, size=50)

print(model['america'])
-- [ 0.17634061  0.58502656  0.27098337 -0.17523931 -
0.24094008 -1.72932017 ...]
```



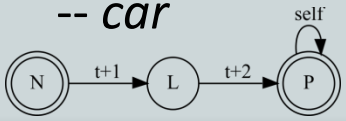
Training a model – toy example

```
# Is 'america' more similar to 'country' or 'goal'?
print(model.wv.similarity('america', 'country') >
model.similarity('america','goal'))

-- True

# Let's find the most similar words to 'america'
print(model.wv.most_similar(positive=['america'],topn=3))

-- [('world', 0.7713112235069275), ('nation', 0.737435519695282), ('freedom', 0.72567957639691)]
-- [('world', 0.7955950498580933), ('freedom', 0.7540770173072815), ('nation', 0.73620635271072)]
-- [('nation', 0.85256427526474), ('best', 0.8303712606430054), ('future', 0.8137349486351013)
...
```



Training a model – toy example

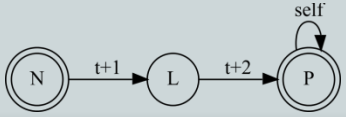
```
# What's a leader/king like?
print(model.wv.most_similar(positive=['leader','king'],topn=2))
-- [('elected', 0.9522648453712463)]
-- [('emperor', 0.851976871490478 5)]


# What if we're looking for words more distant from king?
print(model.wv.most_similar(positive=['american','leader'],
negative=['king'],topn=2))
-- [('freedom', 0.803984224796295 2)]
-- [('human', 0.641267895698547 4)]

# Spot the odd one out
print(model.wv.doesnt_match(["france","germany","car","japan"]))
-- car
```



Much bigger in real life…

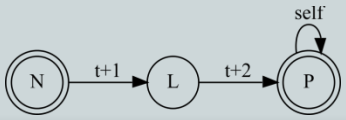
- These examples come from less than 2M tokens / 10 MB of text – often rather bad!
- You can use ready-made examples from Google, Wikipedia, …
  - Google Word2Vec text is 3.6 GB, popular trimmed News version ~80MB
  - GloVe 300D pre-trained embeddings ~1GB (Paddington et al.)
  - BERT Base even larger, BERT Large has 345M parameters based on 3.5G words, takes about 4 days on 16 cloud TPUs (~about 100 mile car drive of electricity!)
- Not trained on the fly – used as saved trained models
- Typically held in memory, not loaded for each function call
- Still slow to load on a laptop, require high GPU RAM

Gensim lets you train your own medium sized models!



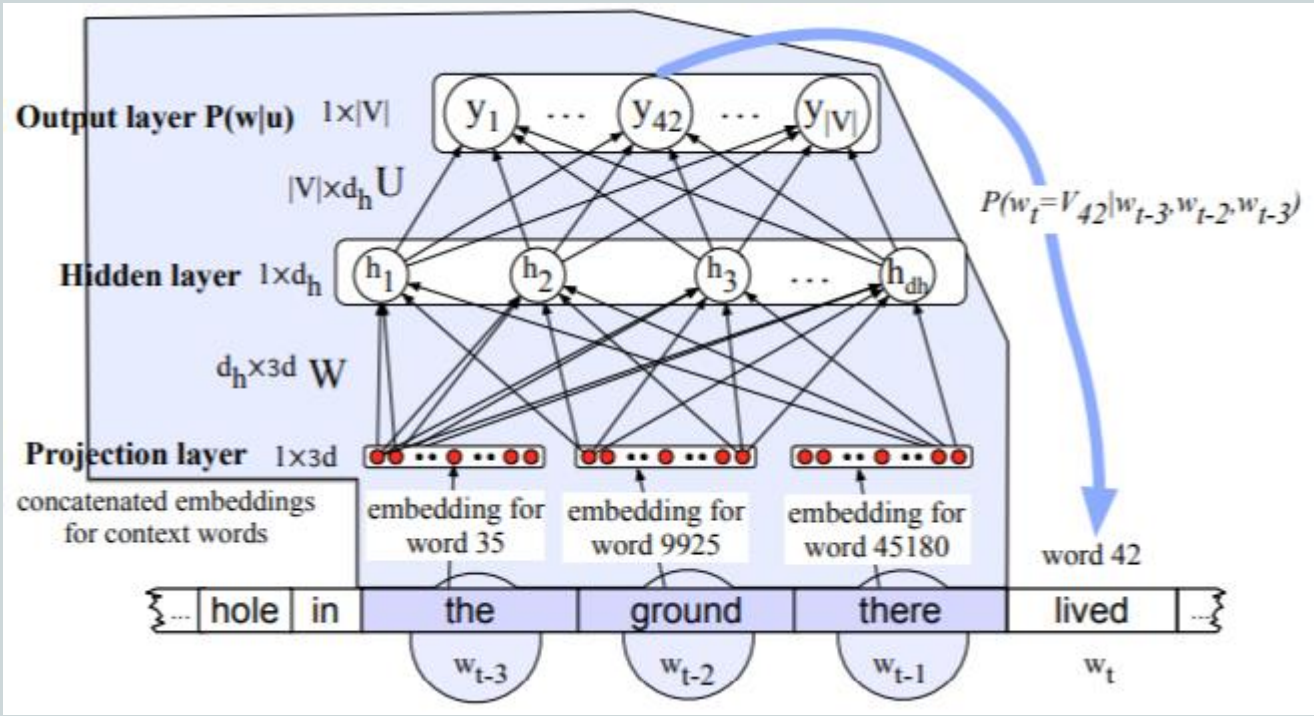
Beyond word similarity

- Word embeddings are now one of the most popular ways of representing text:
  - Feed tools embeddings instead of words
  - Probabilities based on vector dimensions – allows reasonable behavior for OOV items
  - Open to mathematical operations:
    - Sentence meaning = avg. of word vectors?
    - Classify sentence sentiment using vectors?
    - Discourse segmentation via differences in sentence meaning?
    - …

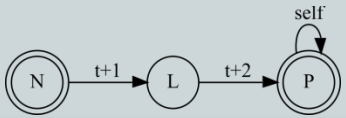


Vectors in language models

- In a feed forward network we could do:

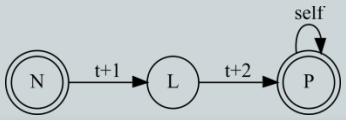


Jurafsky & Martin (2017)

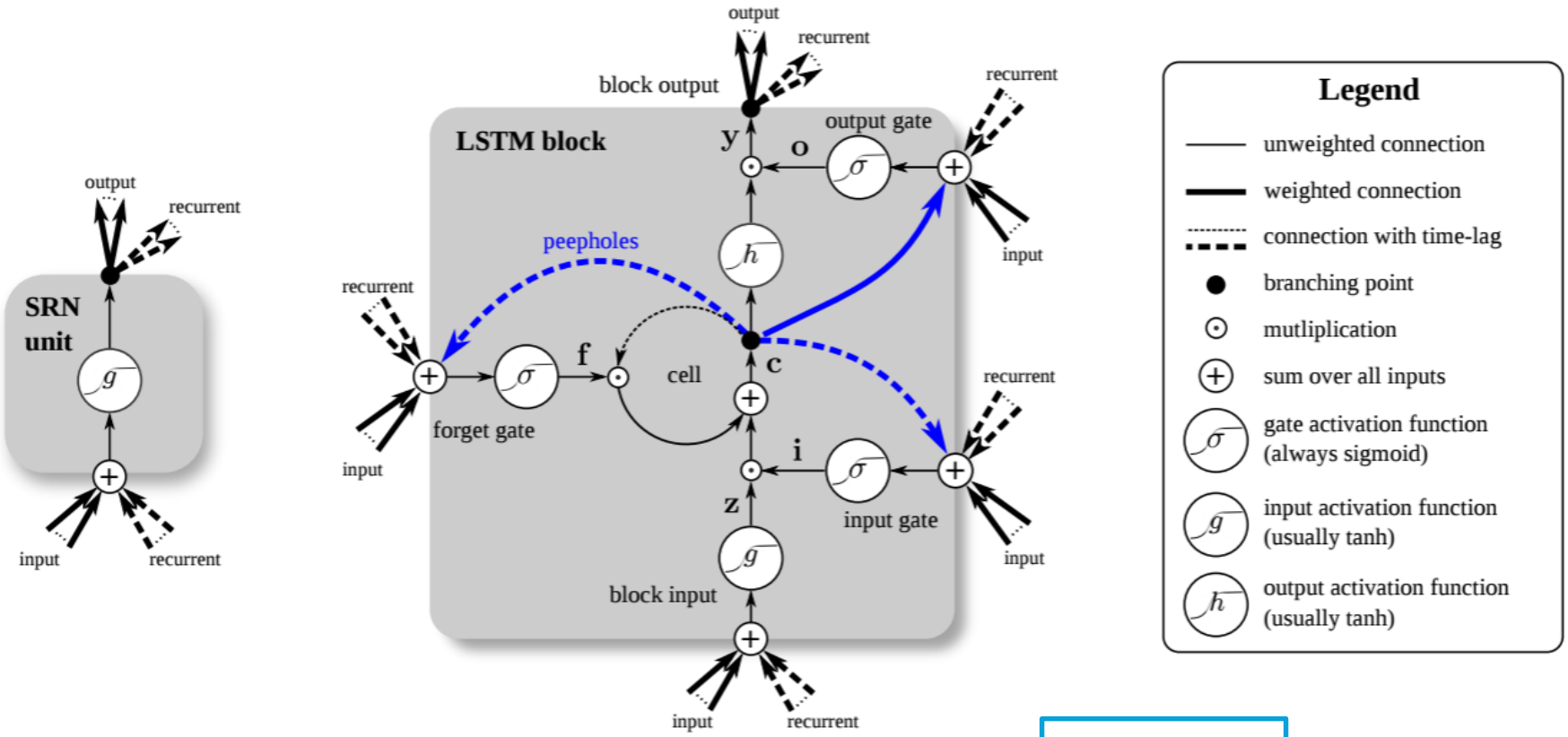


Using memory

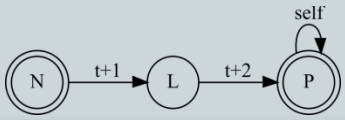
- Recent neural models use **memory** based architectures (Recurrent Neural Networks - RNNs) or attention weights (Transformer)
- Popular type: Long Short Term Memory (LSTM) networks
  - Cells don't just get input 'synapse' weights, but also activate themselves
  - Allows cells to remember previous states
  - In LSTMs: cells also learn when to forget what they've seen



LSTM cell structure



Greff et al. (2015)



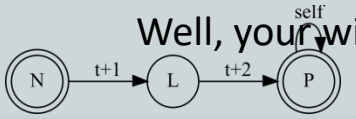
What can LSTMs do?

- Intuitive example: character level sequence to sequence modeling
- Example – trained on Shakespeare:

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

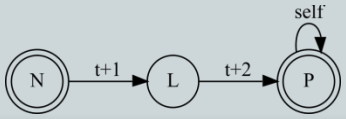


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

What can LSTMs do?

- Intuitive example: character level sequence to sequence modeling
- Example – trained on Wikipedia:

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. … Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[http://www.humah.yahoo.com/guardian.cfm/7754800786d17551963s89.htm

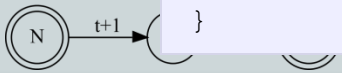


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

What can LSTMs do?

- Example – trained on Linux source code:

```
/*
 * If this error is set, we will need anything right after
that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long)
*FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  return disassemble(info->pending_bh);
}

static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
    disable_single_st_p(dev);
  pci_disable_spool(port);
  return 0;
}
```



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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:

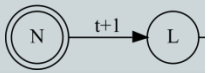
```
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

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;
}
```

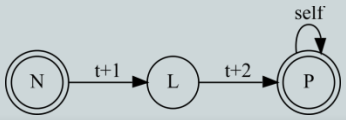
A large portion of cells are not easily interpretable. Here is a typical example:

```
/* 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.
   */
```



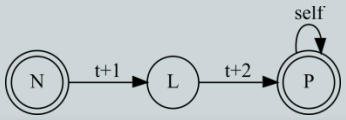
Training your own

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

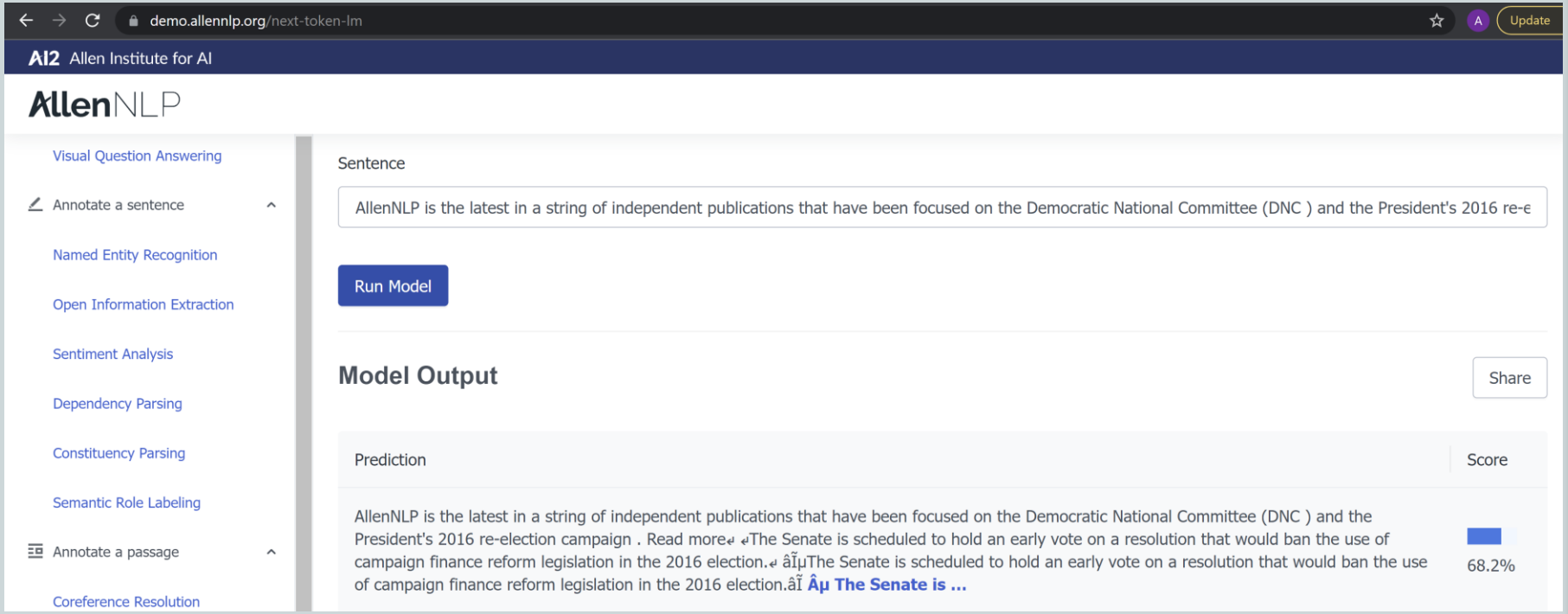


Bonus fun

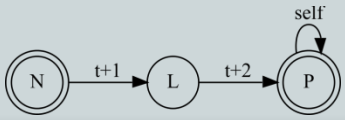
- You can test AllenNLP's neural LM here:
  - https://demo.allennlp.org/next-token-lm
- And you can chat with a neural network trained on conversational pairs
- Example:
  - http://neuralconvo.huggingface.co/
  - (also compare Microsoft's TAY: https://twitter.com/tayandyou)



Do neural LMs solve all problems?

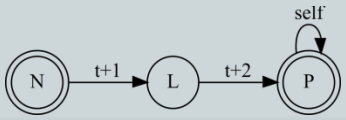


Generated on 2021-10-18



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 is a good starting point
  - Grad students: once you are confident in coding, consider taking LING-504/COSC-576
- Further reading:
  - Jurafsky & Martin (2017, C7)
  - *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**?
- Next time: *Markov Models*