N-gram models, hidden Markov models and POS tagging

NLP Week 4

Thanks to Dan Jurafsky for some of the slides and inspiration!

Plan for today

1. **N-gram language models or Markov models**
2. **Latent categories, Hidden Markov models, and Part-of-Speech (POS) tagging**
3. ***Group exercises***

This semester

We will build language models adding to each layer of their complexity:

1. **Bag of words models** (basic statistical models of language)
2. **N-gram models** (+ sequential dependencies)
3. **Hidden Markov models** (+ latent categories)
4. **Recurrent neural networks** (+ distributed representations)
5. **LSTM language models** (+ long distance dependencies)
6. **Transformer language models** (+ attention-based dependency learning)

= Today’s language models!

Sequential dependencies

‘Not only was it the greatest cast’

‘Only it was not the greatest cast’

Sequential dependencies

‘Not only was it the greatest cast’

-> positive

‘Only it was not the greatest cast’

Sequential dependencies

‘Not only was it the greatest cast’

-> positive

‘Only it was not the greatest cast’

-> negative

Sequential dependencies

‘Not only was it the greatest cast’

-> positive

‘Only it was not the greatest cast’

-> negative

BoW features are the same!

[cast, greatest, it, only, not, the, was]

Sequential dependencies

‘Not only was it the greatest cast’

-> positive

‘Only it was not the greatest cast’

-> negative

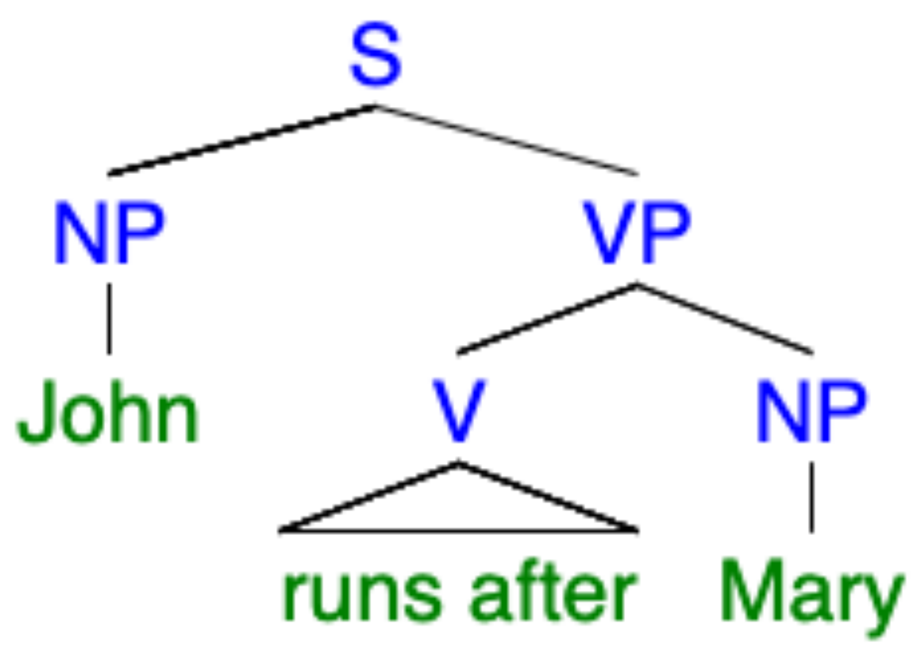
BoW features are the same!

[cast, greatest, it, only, not, the, was]

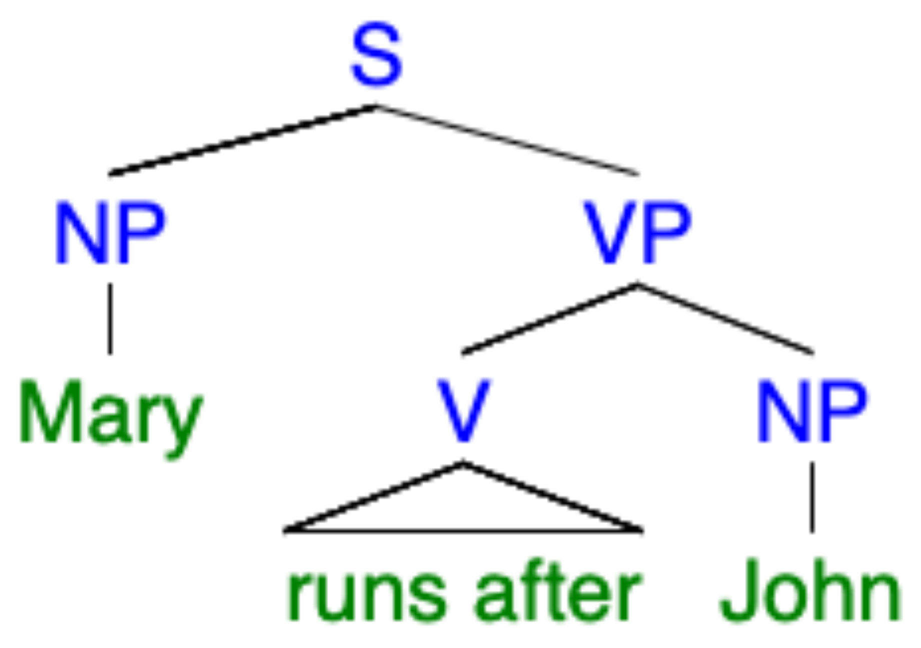
The order of words matters, there is meaning in sequential dependencies.

Linguistic data… why so special?

There exists a mapping between structure and meaning.



vs



N-gram models

- BoW models do not represent sequential dependencies in there distributions.

- **The solution: N-gram models, a generalization of BoWs to larger context sizes**

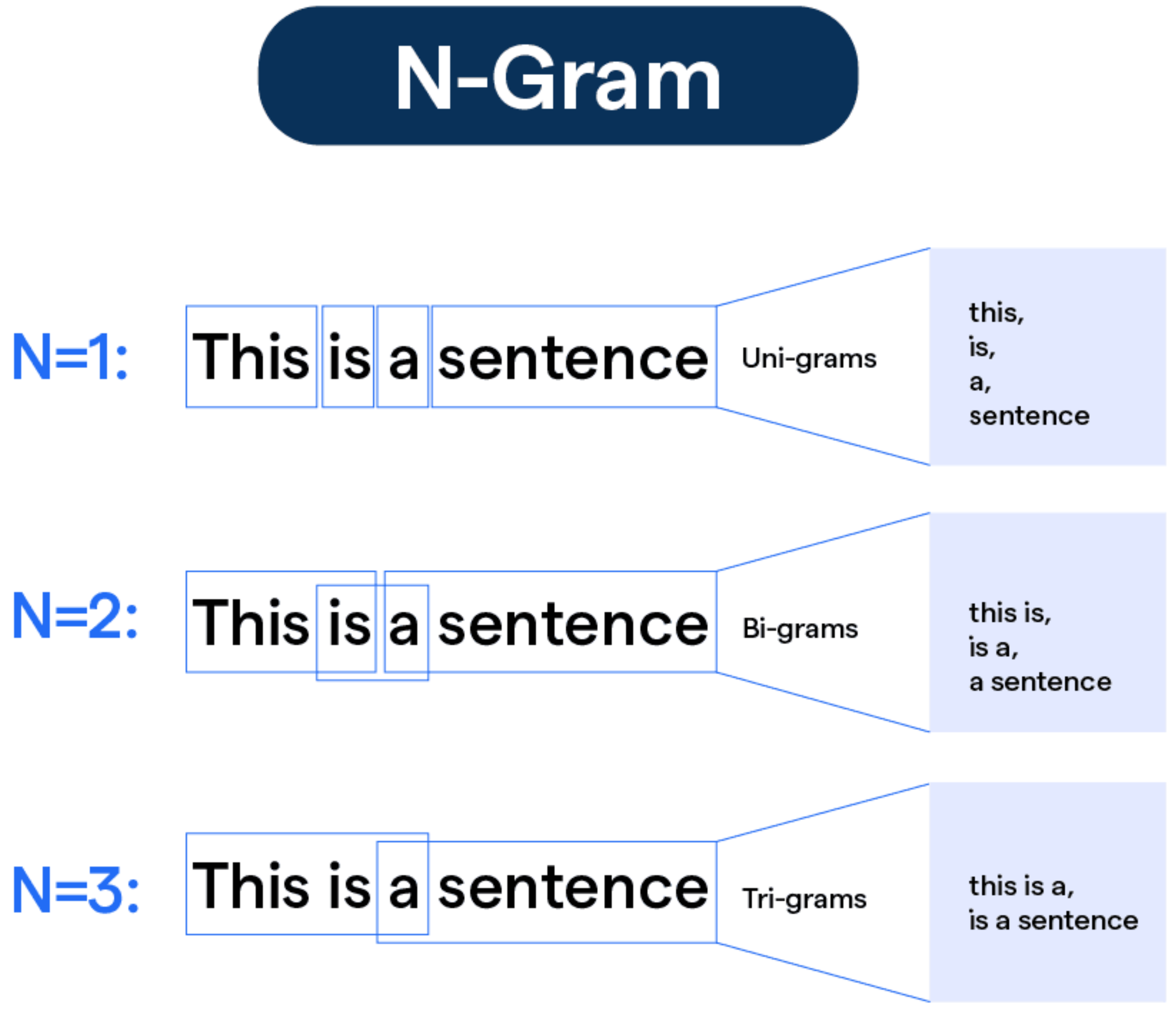
- The 'N' stands for the dependency context size:

    - Uni-gram models consider 1-word contexts a.k.a uni-gram = BoW

    - Bi-gram models consider 2-word dependencies

    - Tri-gram models consider 3-word dependencies

    - …



Language modeling

Language models are systems that can predict upcoming words:

- They can assign a probability to each potential next word
- They can assign a probability to a whole sentence

Language modeling

Why word prediction?

It's how **large language models (LLMs)** work!

LLMs are **trained** to predict words

- Left-to-right (autoregressive) LMs learn to predict next word

LLMs **generate** text by predicting words

- By predicting the next word over and over again

Language modeling

Goal: compute the probability of a sentence or sequence of words W:

$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$

Related task: probability of an upcoming word:

$P(w_5|w_1,w_2,w_3,w_4)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$

An LM computes either of these:

$P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$

Joint distributions

$$P(X_1, \ldots, X_n)$$

Joint distributions

- A distribution over a series of random variables or events

$$P(X_1, \ldots, X_n)$$

Joint distributions

- A distribution over a series of random variables or events
- The support of a joint distribution is the cartesian product of the individual random variables' supports.

$$P(X_1, \ldots, X_n)$$

Joint distributions

- A distribution over a series of random variables or events
- The support of a joint distribution is the cartesian product of the individual random variables' supports.
- The probability mass function will depend on whether or not there are independence assumptions.

$$P(X_1, \ldots, X_n)$$

Marginal distributions

Marginal distributions

Given some joint distribution: $P(X, Y)$

- The support of a marginal distribution $P(Y)$ is simply the set of outcomes for the variable Y

- The marginal probability of some outcome $y \in Y$ is :

$$P(y) = \sum_{x \in X} P(x, y)$$

Conditional distributions

Conditional distributions

- The support of a conditional distribution is the subset of the joint distributions support where the conditioner Y is true.

- The probability mass function will return the renormalized probabilities for the support such that they sum to one (we do so by dividing by the marginal probability of Y)

$$P(X \mid Y) = \frac{P(X, Y)}{P(Y)}$$

Marginalization versus conditioning:

- In marginalization we are asking: what is the probability of the last word in our sentence being “movie”
- In conditioning we are asking: given that the last word in the sentence is “movie”, what is the probability of any of the other words in the sentence.

These are complementary operations in relation to the joint distribution

Variable independence

- **Independence** can be defined in two ways
    - Variables are independent if their joint distribution is the product of the marginal distributions

$$P(X_1, \ldots, X_n) = P(X_1) \cdot \ldots \cdot P(X_n)$$

-
    - Variables are independent if their conditional distributions are equal to their marginal distributions

$$P(X \mid Y) = P(X)$$

The Chain Rule

- A decomposition of joint probability using conditional and marginal probabilities
- Works for both independent and dependent variables
- Is order invariant!

$$\Pr(X_1, \ldots, X_n) = \Pr(X_1) \cdot \Pr(X_2 \mid X_1) \cdot \ldots \cdot \Pr(X_n \mid X_1, \ldots X_{n-1})$$

The Chain Rule

- A decomposition of joint probability using conditional and marginal probabilities
- Works for both independent and dependent variables
- Is order invariant!

$$\Pr(X_1, \ldots, X_n) = \Pr(X_1) \cdot \Pr(X_2 \mid X_1) \cdot \ldots \cdot \Pr(X_n \mid X_1, \ldots X_{n-1})$$

$$= \Pr(X_n) \cdot \Pr(X_{n-1} \mid X_n) \cdot \ldots \cdot \Pr(X_1 \mid X_2, \ldots, X_n)$$

The Chain Rule

The Chain Rule applied to compute joint probability of words in sentence

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \end{aligned}$$

P("The water of Walden Pond") =

P(The) × P(water|The) × P(of|The water)

× P(Walden|The water of) × P(Pond|The water of Walden)

Markov assumption

Andrei Markov

- N-grams models are also known as Markov models, because the follow the Markov assumption.
- **Markov assumption:** The probability of events in a joint distribution are conditionally independent and can be decomposed as such.

Markov assumption

Andrei Markov

- N-grams models are also known as Markov models, because the follow the Markov assumption.

- **Markov assumption:** The probability of events in a joint distribution are conditionally independent and can be decomposed as such.

Markov assumption in bi-gram model

$$P(\texttt{blue}|\texttt{The water of Walden Pond is so beautifully})$$

$$\approx \; P(\texttt{blue}|\texttt{beautifully})$$

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

N-gram models

- **Uni-gram LM:** $$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$
- **Bi-gram LM:** $$P(w_1 w_2 \dots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-1})$$
- **Tri-gram LM:** $$P(w_1 w_2 \dots w_n) \approx \prod_{k=1}^{n} P(w_k \,|\, w_{k-2} w_{k-1})$$

N-gram models

P('John chased Mary') =

- **Uni-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

- **Bi-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-1})$$

- **Tri-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-2} w_{k-1})$$

N-gram models

P('John chased Mary') =

- **Uni-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

P('John') x P('chased') x P('Mary')

- **Bi-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-1})$$

- **Tri-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-2} w_{k-1})$$

N-gram models

P('John chased Mary') =

- **Uni-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

P('John') x P('chased') x P('Mary')

- **Bi-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-1})$$

P('\PAD John') x P('John chased') x P('chased Mary') x P('Mary \PAD')

- **Tri-gram LM:**

$$P(w_1 w_2 \ldots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-2} w_{k-1})$$

N-gram models

P(‘John chased Mary’) =

- **Uni-gram LM:**

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

P(‘John’) x P(‘chased’) x P(‘Mary’)

- **Bi-gram LM:**

$$P(w_1 w_2 \dots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-1})$$

P(‘\PAD John’) x P(‘John chased’) x P(‘chased Mary’) x P(‘Mary \PAD’)

- **Tri-gram LM:**

$$P(w_1 w_2 \dots w_n) \approx \prod_{k=1}^{n} P(w_k | w_{k-2} w_{k-1})$$

P(‘\PAD \PAD John’) x P(‘\PAD John chased’) x P(‘John chased Mary’) x P(‘chased Mary \PAD’) x P(‘Mary \PAD \PAD’)

Evaluations

How can we evaluate the ‘goodness’ or ‘accuracy’ of a language model?

- **Extrinsic evaluations:** Evaluating models on their external behaviour, such as their performance on downstream tasks (eg. If we use an LM to generate features for classification)

- **Intrinsic evaluations:** Evaluating the models internal behaviour, a.k.a. the goodness of its contextual probability distributions. But how?

Perplexity

- Is the common intrinsic metric used to evaluate LMs.

- Based on the intuition that LMs should mimic our grammaticality judgements, such that grammatical sentences should be more likely than ungrammatical ones… or sentences we think are more likely should also be more likely for the model.

- **Perplexity:** A measure of how likely *or surprising* a test set of sentences is under a model.

Perplexity

Perplexity = the inverse probability of the test set (one over the probability of the test set), normalized by the number of words (or tokens)

$$\begin{aligned}\text{perplexity}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}\end{aligned}$$

Or we can use the chain rule to expand the probability of W:

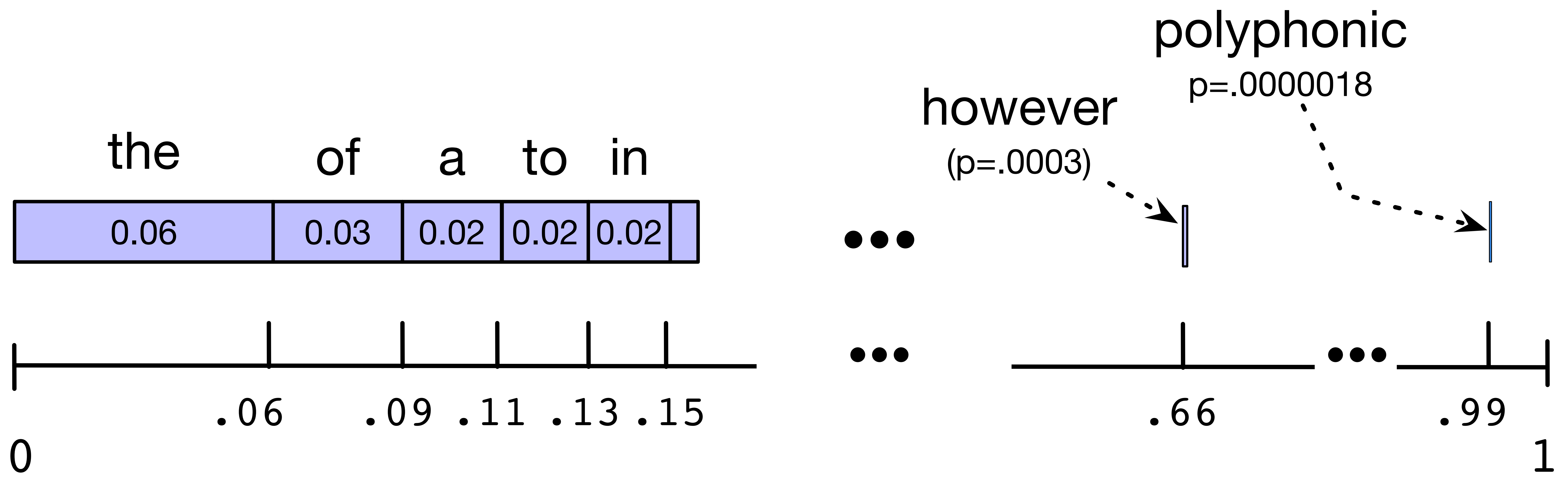
$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Sampling/Generating

We have seen how to **score** a sentence under our language model — i.e. return its probability — but how can we use a language model to **generate** new likely sentences under our model?

Sampling/Generating

Sampling a word from a distribution



Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x)
according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>) `<s> I`

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

```
<s> I
    I want
```

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

```
<s> I
    I want
      want to
```

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

```
<s> I
    I want
      want to
           to eat
```

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
```

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
```

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food </s>
```

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

Sampling/Generating

To sample a sentence, you recursively sample from the conditional N-gram distribution. Here's an example with a bi-gram model.

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

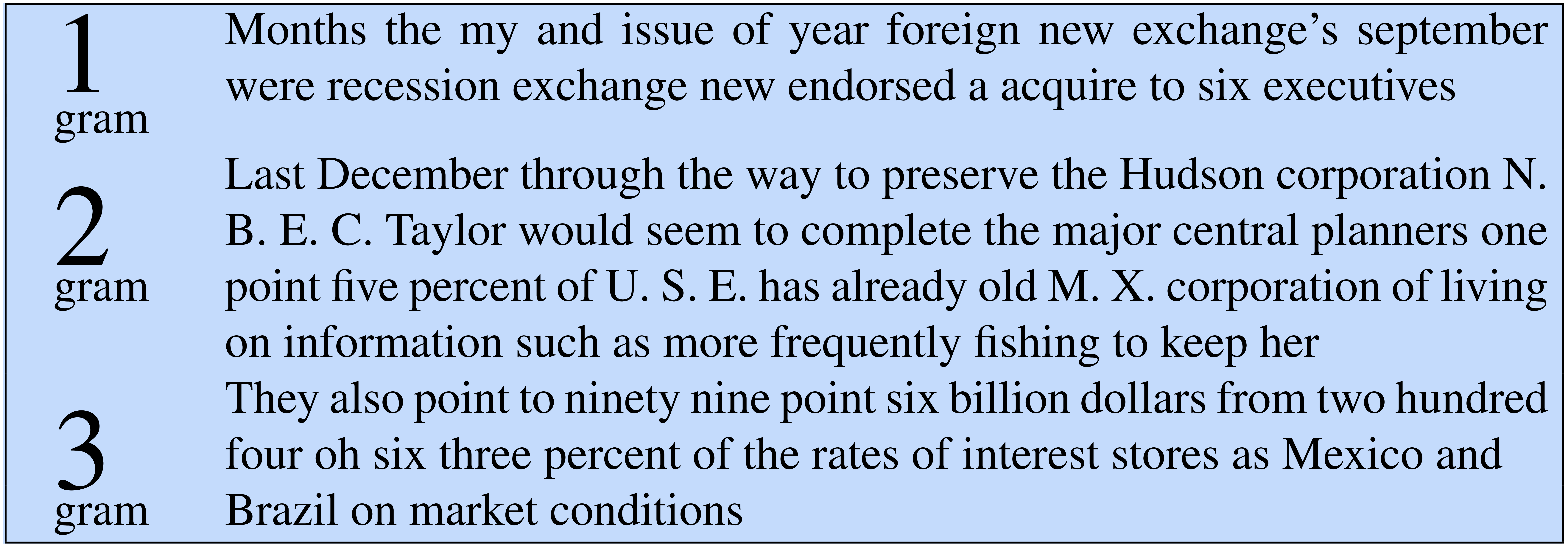
And so on until we choose </s>

Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food </s>
I want to eat Chinese food
```

Sampling/Generating

Samples from different N-gram models fit to the Wall Street Journal corpus



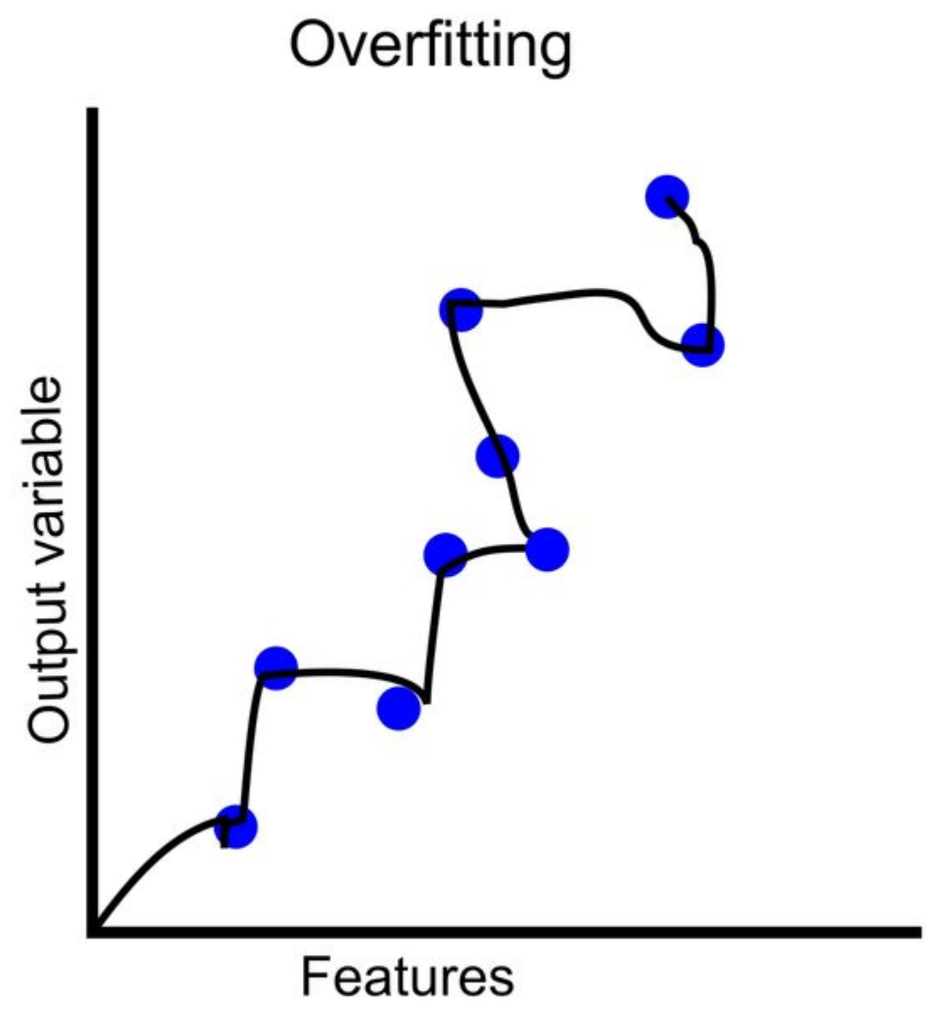
1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

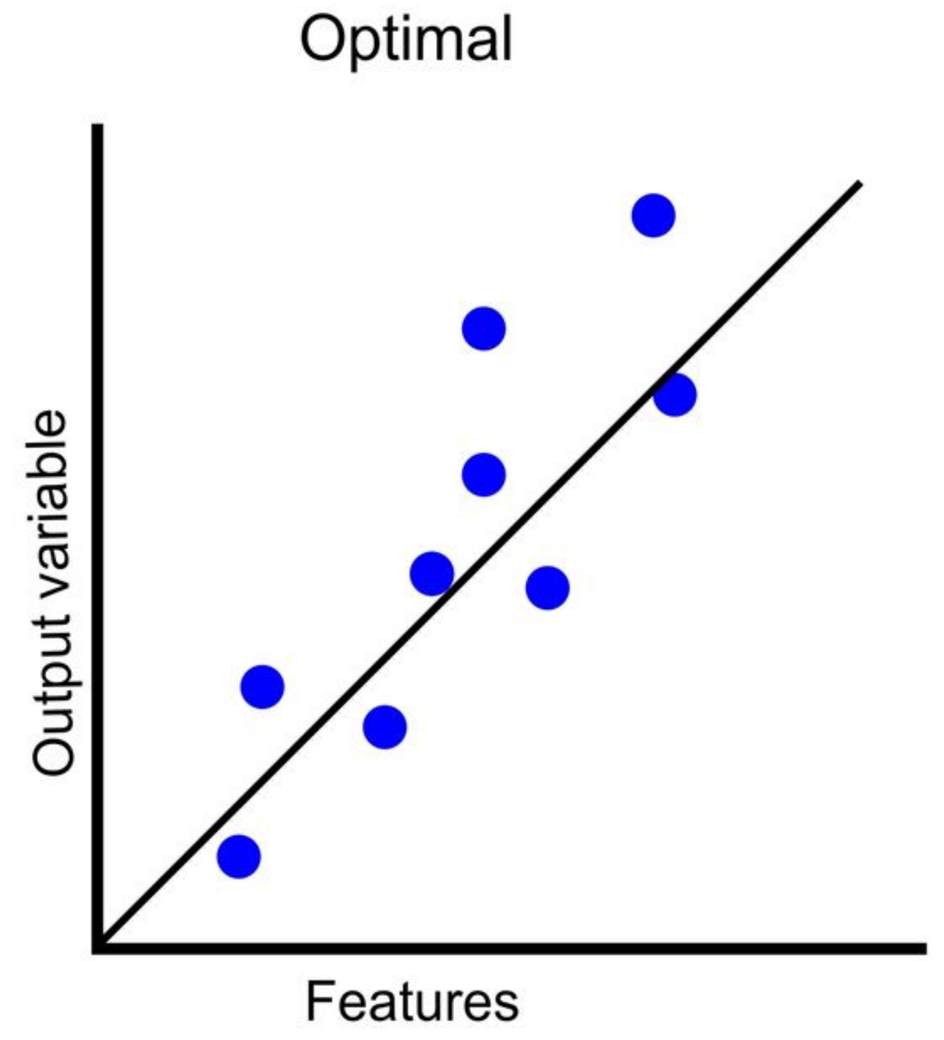
Overfitting versus generalisation

- Language models are trained/fit to a training dataset — i.e. they learn to represent the contextual distribution of the training data.

- If the true train data distribution diverges from the true test data distribution:

  ➡**The model can be *overfit* to the train data distribution and be unable to *generalise* to unseen test data.**



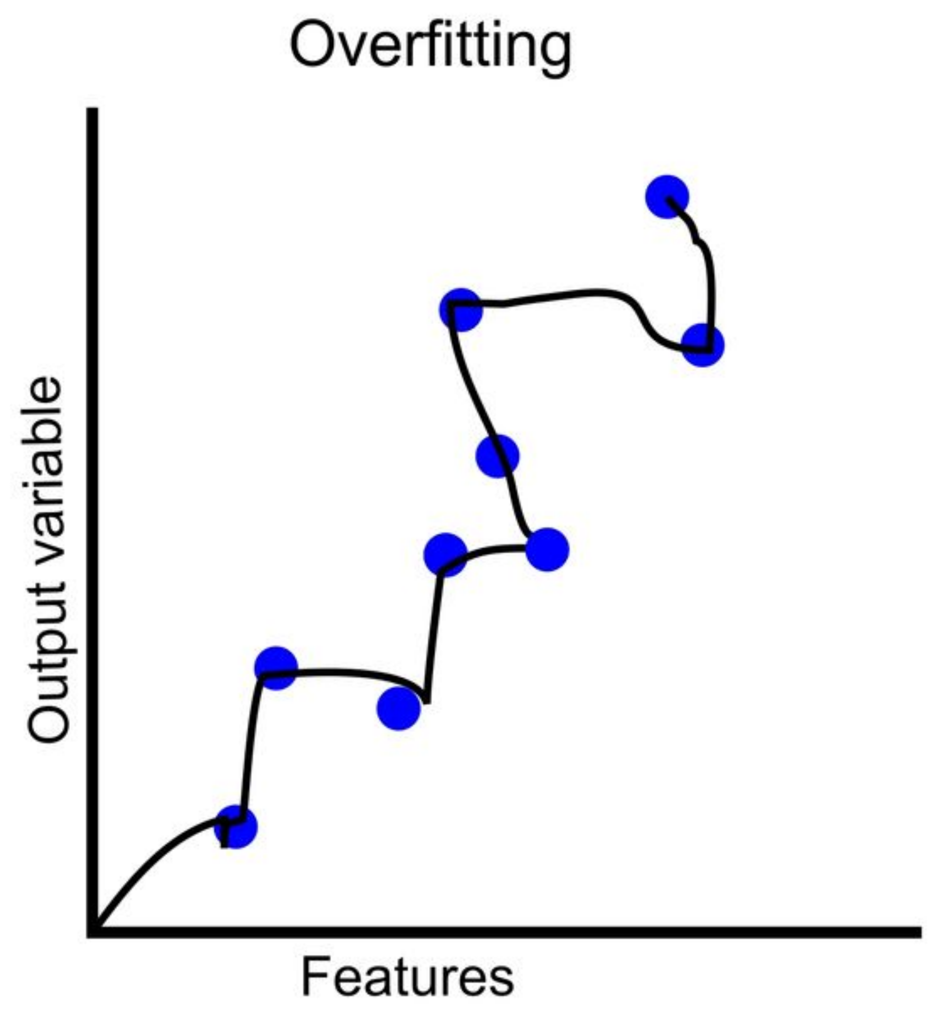


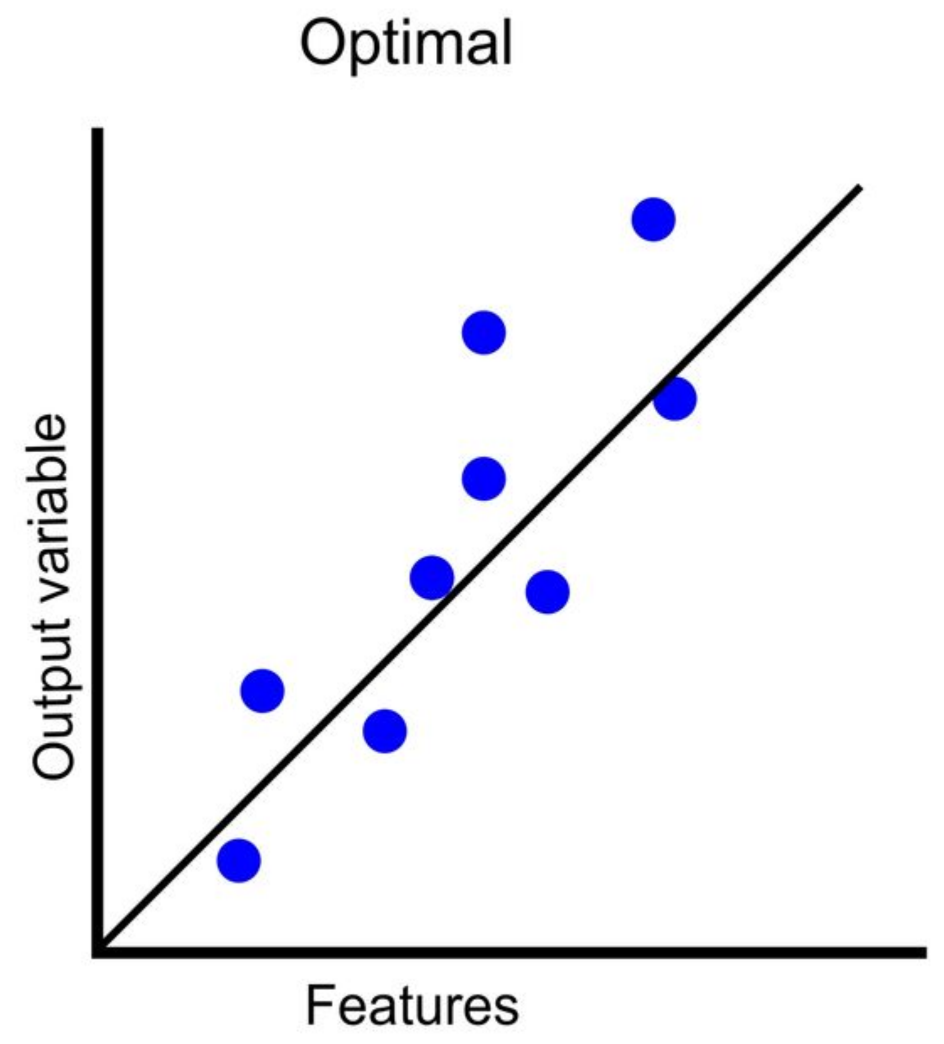
Overfitting versus generalisation

- **For good generalization, we want two things:**

    1. **The train and test data to be representative of one another;**

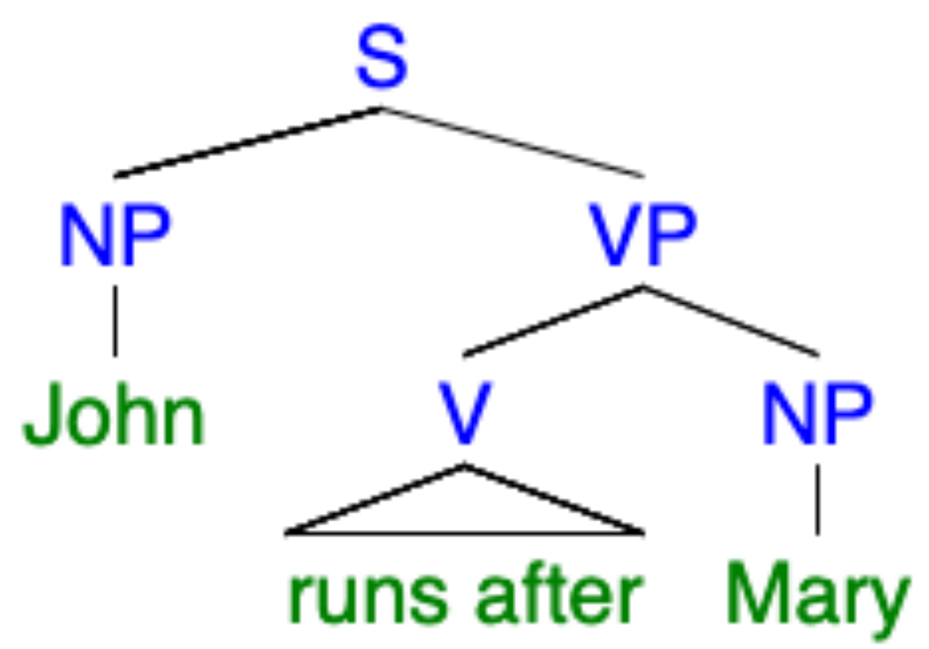
    2. **The model to be optimally trained and not overfit to the data.**



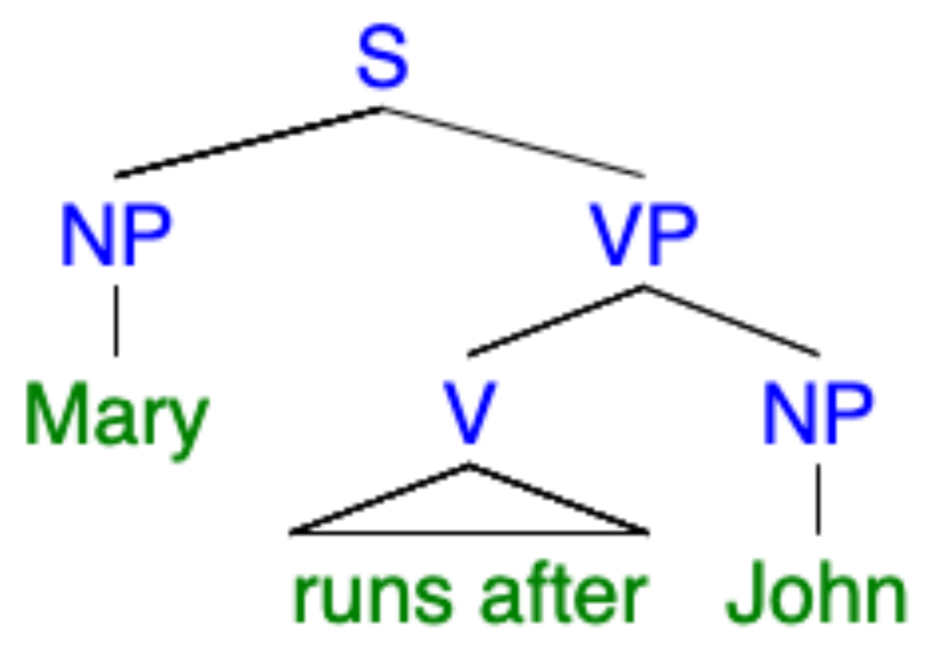


Linguistic data… why so special?

There exists a mapping between structure and meaning.

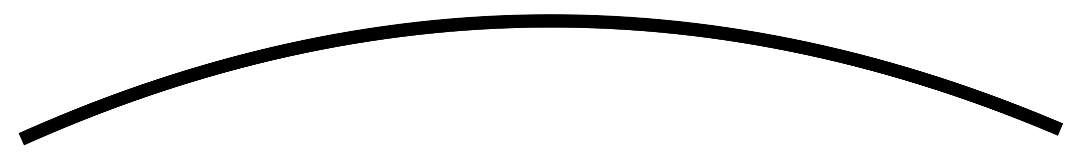


vs



Latent categories

- N-gram models cannot capture certain linguistic structures because they only encode sequential dependencies of fixed length



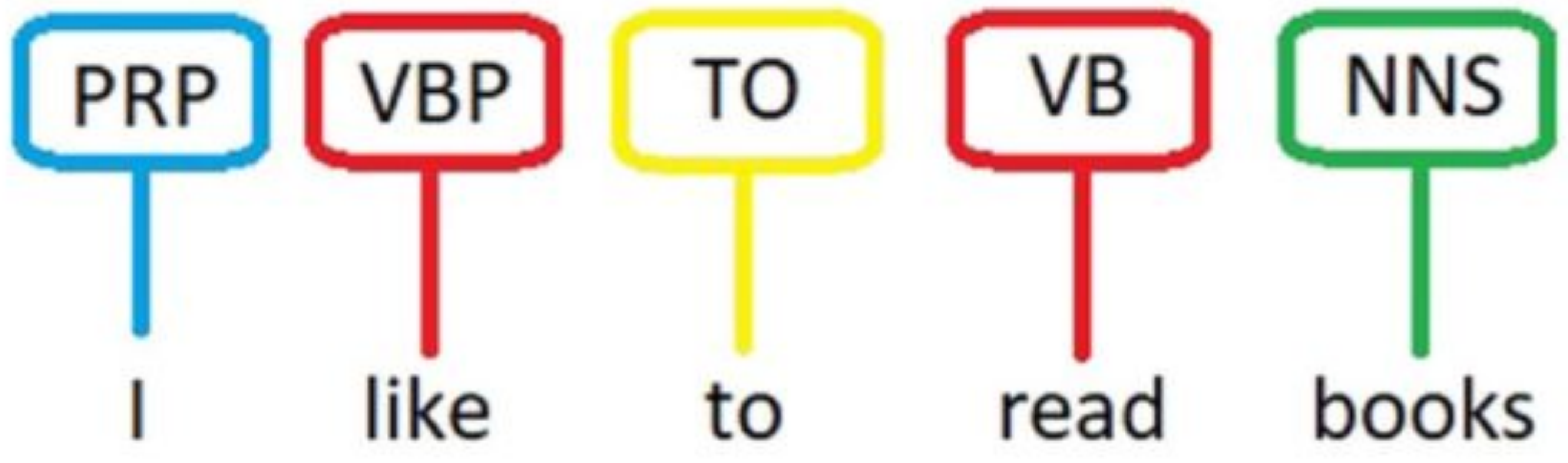
The $\underline{\text{keys}}$ on the table $\underline{\text{were}}$ for the back door.

Latent categories

- A latent variable is a value that is not directly observed.
- A latent category is an unobserved category variable that can be assigned to tokens.
- For example, we often use syntactic categories (verb, noun, adjective…) as latent categories over words.

Part-of-speech (POS)

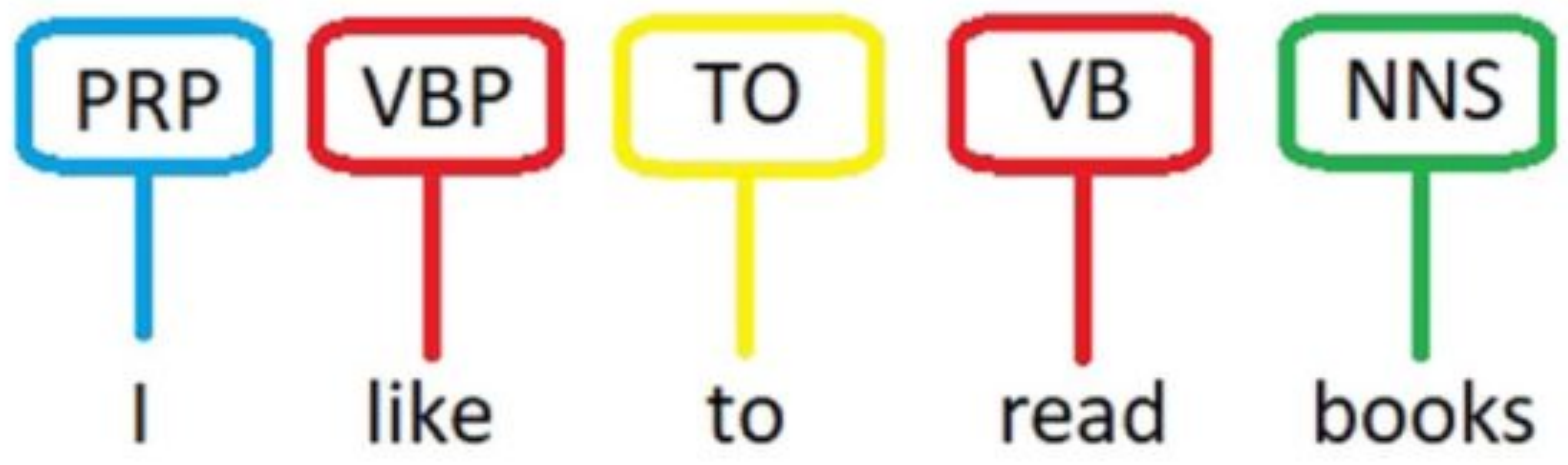
- POS tags are useful latent category labels which we assign to tokens.
- They loosely correspond to syntactic categories
- POS-tagging is the task of assigning POS tags to words. It can be a very useful preprocessing step for downstream NLP tasks.



POS tagging

- We can train a form of language model called a Hidden Markov model to learn tags for words and generate new words conditioned on those tags.

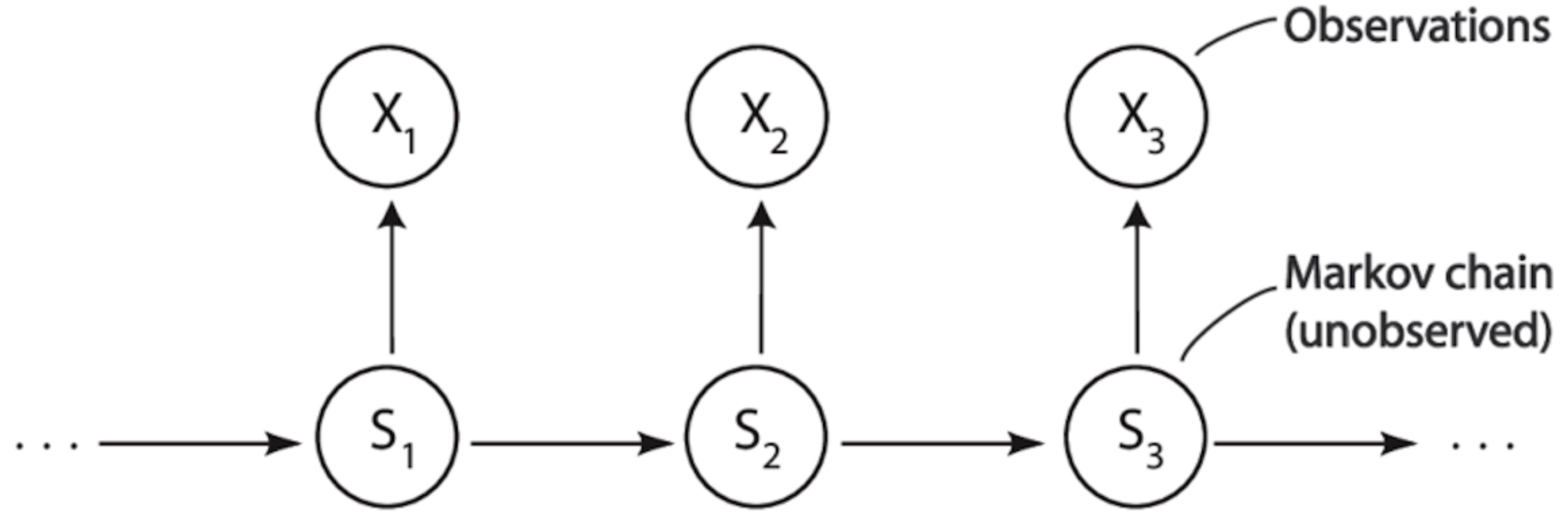
- POS tagging is also possible using recurrent neural networks.



Hidden Markov models

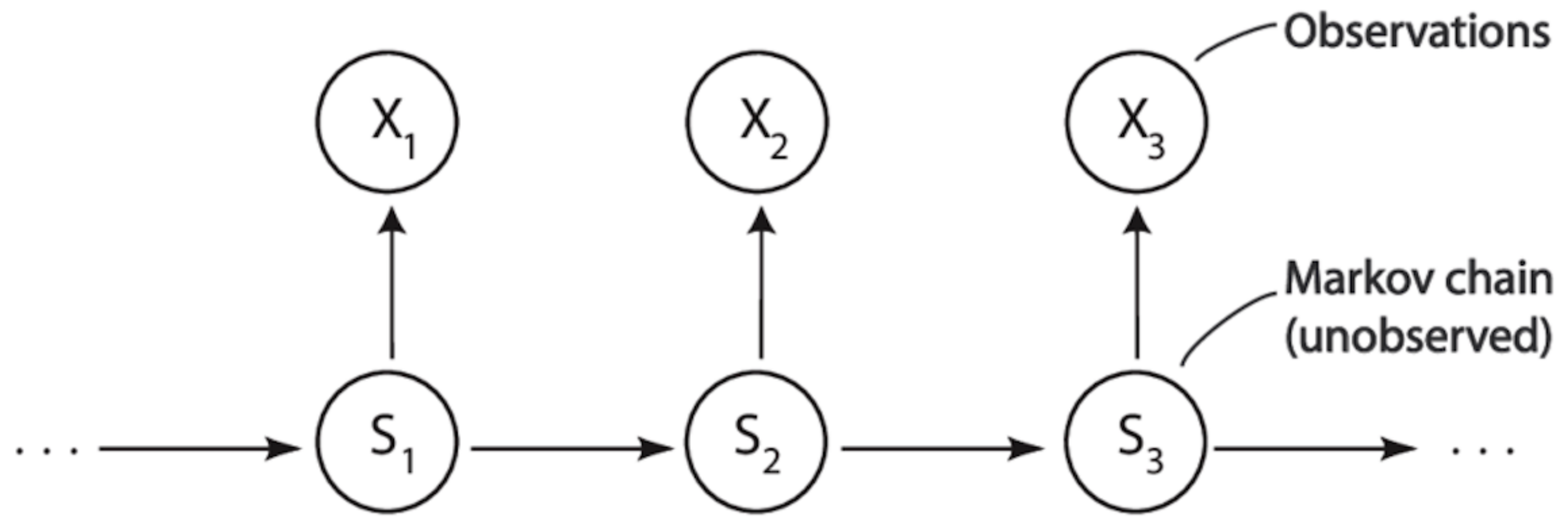
- For every observation (here token), we first sample a latent category and subsequent samples are conditioned on the latent category chain (rather than the previous tokens).

- So, it is a sequential model over latent categories of tokens (hidden Markov model) instead of a sequential model directly over tokens (Markov model, or n-gram!)



Hidden Markov models

- This blog post does a great job explaining how HMMs can be used for POS tagging: https://www.mygreatlearning.com/blog/pos-tagging/
- We will not cover HMM optimization in this class, but if interested read Appendix A, on the forward algorithm and the Viterbi algorithm to learn about how this is done.



[15 minute break]

Working with N-gram models!

Team up!

Open exercises/week 4 in your course folder and start writing/running code!