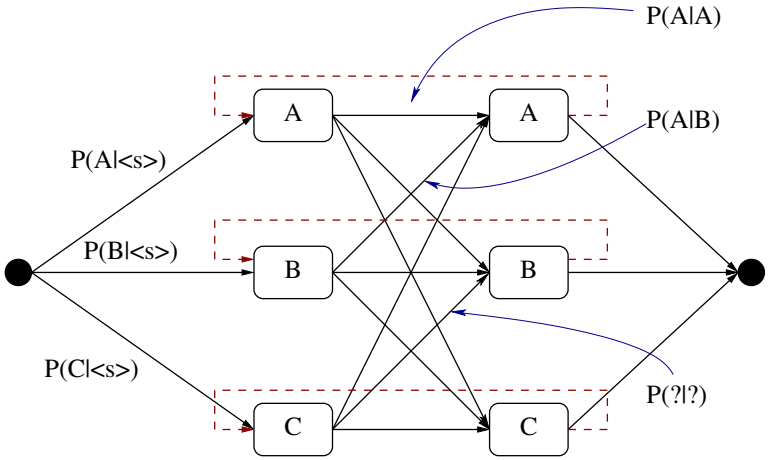
Speech Processing (Module CS5241)

Lecture 7 – Statistical Language Modelling



SIM Khe Chai

School of Computing, National University of Singapore, January 2010

Language Modelling for Speech Recognition

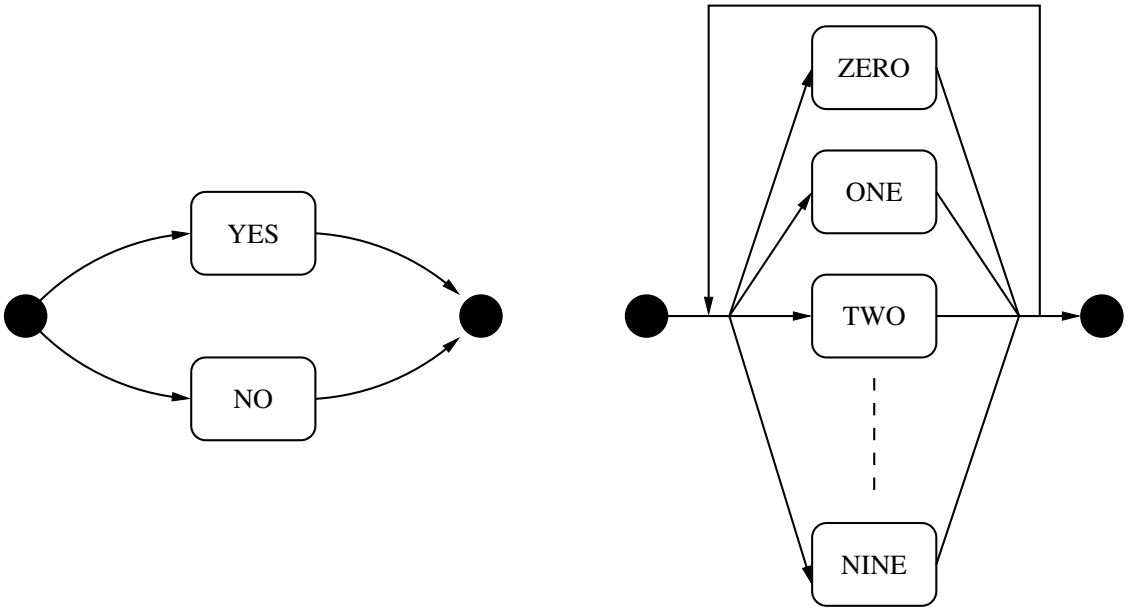
A statistical language model assigns a probability to a sequence of m words by means of a probability distribution. Language modeling is used in many natural language processing applications such as speech recognition, machine translation, part-of-speech tagging, parsing and information retrieval.

In speech recognition, a language model defines the search space rom which the most likely word sequence is decoded. A language model defines a vocabulary (list of words that the recogniser will produce). The size of the vocabulary determines the speech recognition task:

- **Small vocabulary**: less than 1,000 words
- **Medium vocabulary**: between 1,000 – 10,000 words
- **Large vocabulary**: more than 10,000 words

Obviously, the search space grows with the vocabulary size, most of the time *exponentially*. For grammer free language models, there are inevitable words which are not covered by the vocabulary. These are known as Out-Of-Vocabulary (OOV) words. It is desirable to keep the OOV rate as low as possible.

Simple Grammars



For simple tasks, the language model can be represented by a grammar network. A grammar network is a weighted finite state machine. It can be in the form of a word-alternatives, word-loop or more complex grammar structure. Weights (probabilities) can be assigned to arcs to give a higher scores to paths which are more frequently observed.

Statistical Language Model

Sometimes, grammar network are not suitable for certain tasks because:

- It is difficult to hand-craft all the possible grammar rules
- The user may be allowed to speak freely, even *ungrammatically*!

Statistical language models are an attractive alternative because:

- The estimation process is data-driven (automated)
- There is no strict grammar rules

A statistical language model assigns probabilities to sentences (word sequences):

$$\begin{aligned} P(W_1^K|\theta) &= P(w_1, w_2, \ldots, w_K|\theta) \\ &= \prod_{k=1}^{K} P(w_k|w_1, w_2, \ldots, w_{k-1}) \end{aligned}$$

n-gram Language Model

The most widely used statistical language models are the n-gram language models. An n-gram language model approximate the conditional probability of a given word w_k as follows:

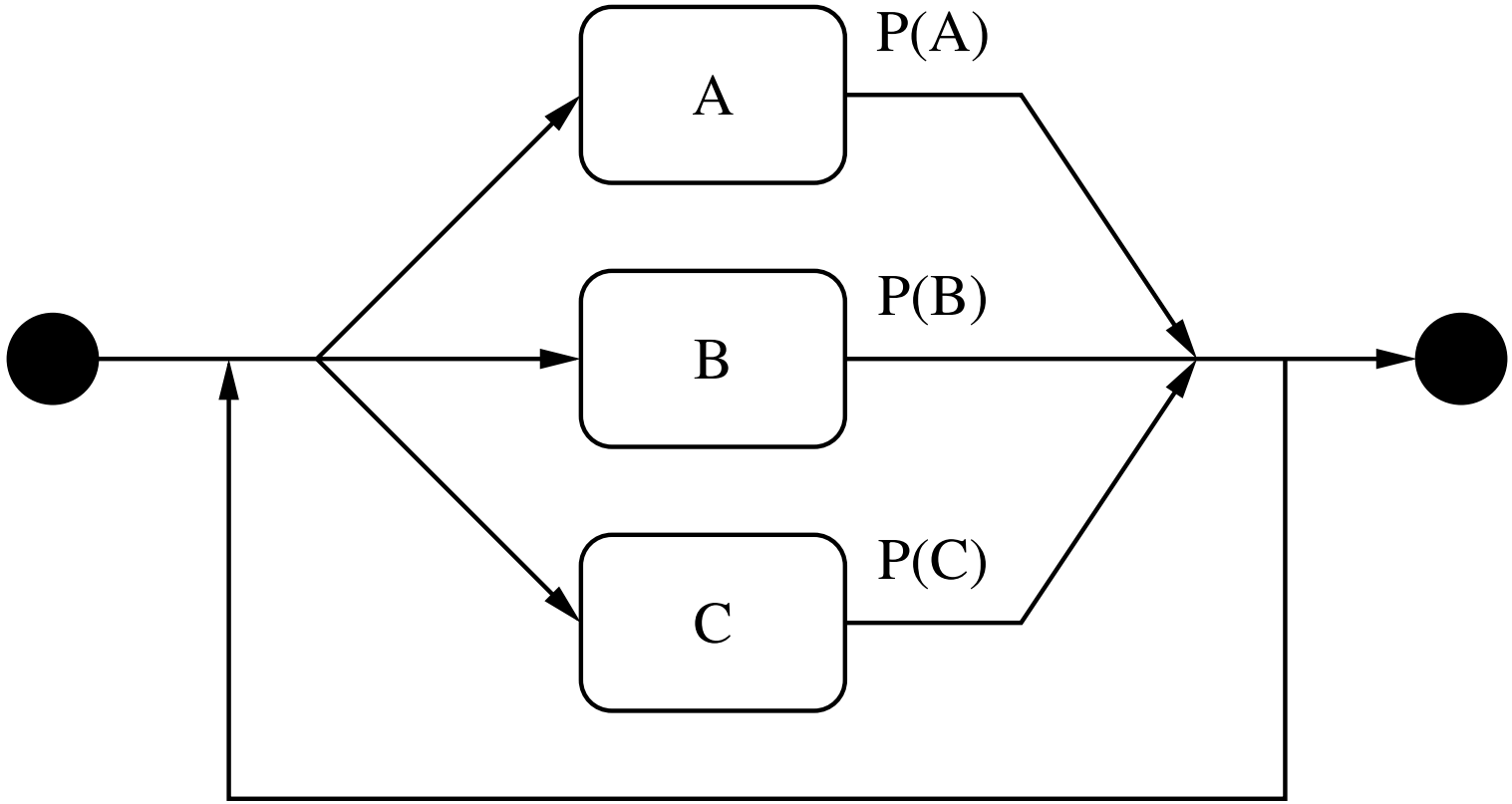
$$P(w_k|w_1, w_2, \ldots, w_{k-1}) \approx P(w_k|w_{k-n+1}, w_{k-n+2}, \ldots, w_{k-1}) \quad (1)$$

Hence, the probability of a given word depends on $n-1$ preceeding words. Commonly used n-gram language models are

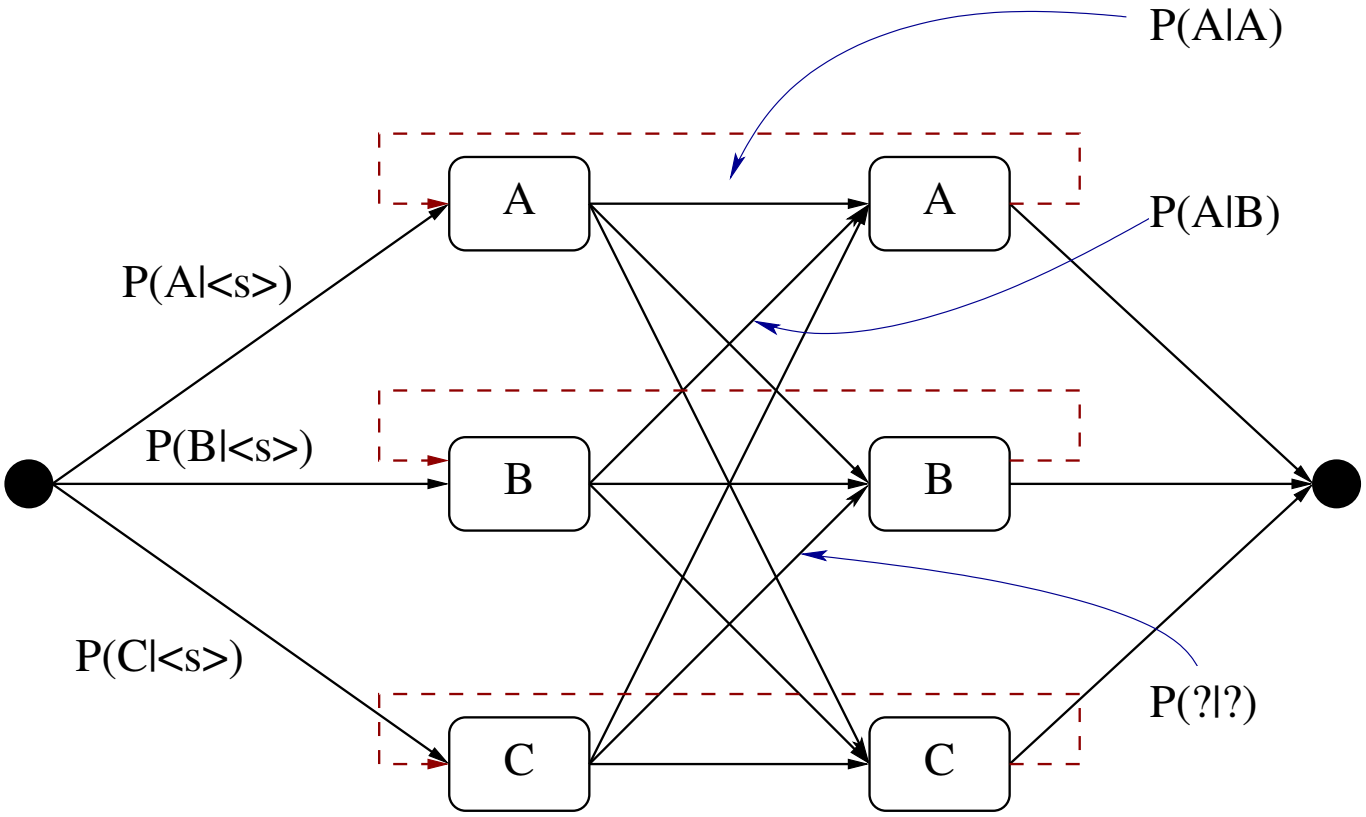
n	Language Model
1	unigram
2	bigram
3	trigram
4	four-gram

Note that for a unigram, the probability of a word is simply given by the frequency of the word itself, *i.e.* independent of the word history.

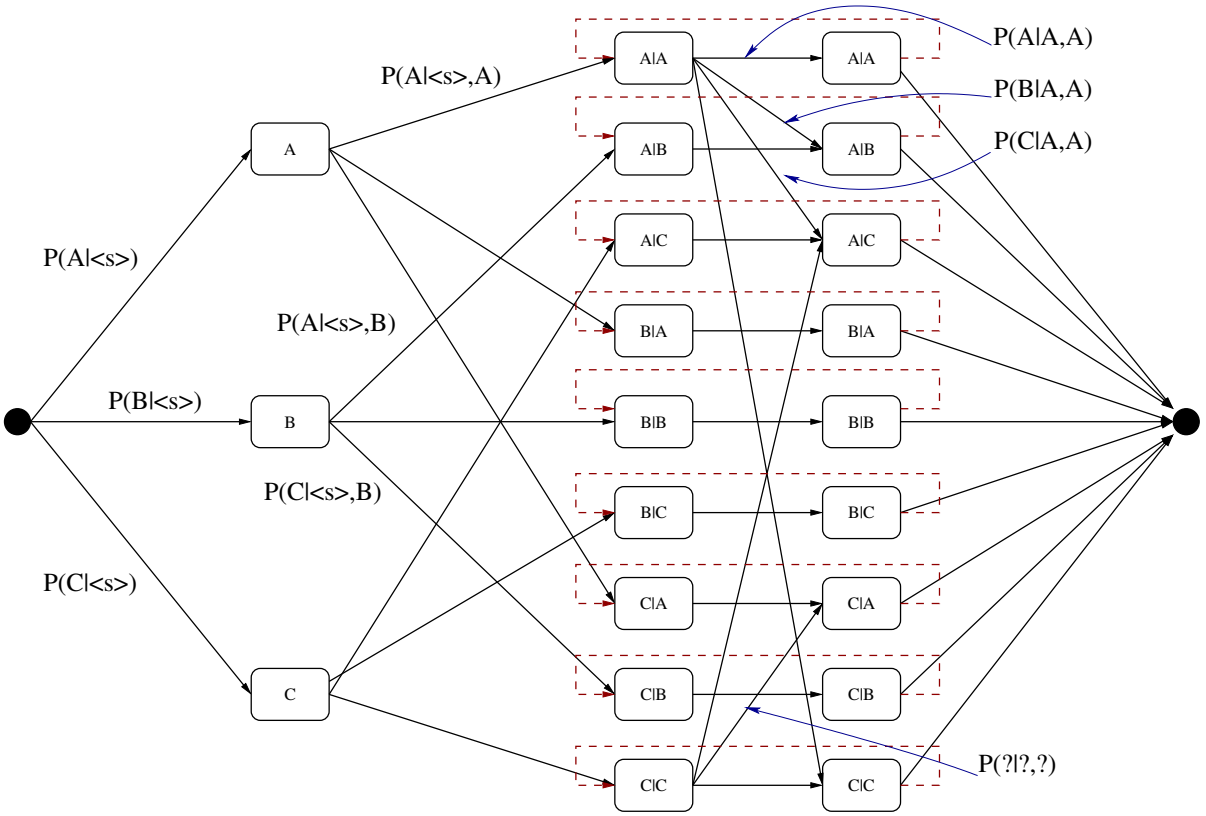
Unigram Language Model



Bigram Language Model



Trigram Language Model



Text Normalisation

Text normalisation is an important step when preparing training data to build language models for speech recognition. Typically, training data are collected from various resources which may include information which may need to be processed into useful formats. Text normalisation aims at converting texts into its spoken forms and removing tokens which are not associated with speech. Typical text normalisations including:

- **Date, currency & number normalisation**:
  - 10/4/1998 $\Rightarrow$ TENTH OF APRIL NINETEEN NINTY EIGHT
  - \$12.50 $\Rightarrow$ TWELVE DOLLARS AND FIFTY CENTS
  - 3.67% $\Rightarrow$ THREE POINT SIX SEVEN PERCENT
- **Abbreviation normalisation**:
  - CNN $\Rightarrow$ C. N. N.
- **Removal of punctuations**
- **Expansion of shorthards**:
  - WE'LL $\Rightarrow$ WE WILL
  - CAN'T $\Rightarrow$ CANNOT

Maximum Likelihood Estimation of n-gram Language Model Parameters

The log likelihood of function is given by:

$$\mathcal{L}(\theta) = \frac{1}{K} \sum_{k=1}^{K} \log P\left(w_k | W_{k-n+1}^{k-1}\right)$$

If the vocabulary is defined by $\mathcal{V} = \{v_1, v_2, \ldots, v_R\}$, then the objective function to be maximised can be rewritten as:

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{K} \sum_{v \in \mathcal{V}} \sum_{h \in \mathcal{H}} \sum_{k=1, w_k = v, h = W_{k-n+1}^{k-1}}^{K} \log P\left(v|h\right) \\ &= \frac{1}{K} \sum_{v \in \mathcal{V}} \sum_{h \in \mathcal{H}} C(h, v) \log P\left(v|h\right) \end{aligned}$$

where $\mathcal{H}$ is the set of possible n-gram history. $C(h, v)$ denotes the number of times v comes after h in the training text. How to find $P(v|h)$?

Optimising the Objective Function

The objective is to find $P(v|h)$:

$$\hat{P}(v'|h') = \arg \max_{P(v'|h')} \sum_{v \in \mathcal{V}} \sum_{h \in \mathcal{H}} C(h,v) \log P\left(v|h\right)$$

$$s.t. \quad \sum_{v \in \mathcal{V}} P(v|h) = 1$$

Hence, the function to be maximised is (including the Language Multiplier for sum-to-one constraint):

$$\mathcal{Q}(P(v|h)) = C(h,v) \log P\left(v|h\right) + \lambda \left(\sum_{v \in \mathcal{V}} P(v|h) - 1 \right)$$

Differentiating this w.r.t $P(v|h)$ and equating to zero yields:

$$P(v|h) = \frac{C(h,v)}{C(h)} \qquad \text{where} \qquad C(h) = \sum_{v \in \mathcal{V}} C(h,v)$$

Likelihood & Perplexity (I)

In order to evaluate the true performance of a language model for speech recognition task, it is neccesary to apply the language model to the actual recognition task. Unfortunately, this is very costly and a quicker way of evaluating LM performance is needed. Typically, an LM is evaluated based on its prediction power on some held out text data. Usually, the perplexity is used as the predictivity measure:

$$PP = 2^{H(P,Q)} = 2^{-\mathcal{L}(\theta)}$$

where $H(\theta)$ is the **cross-entropy** measure between the probability distributions P and Q. This is associated with the average log likelihood scores, where a base-2 log is used:

$$\begin{aligned} H(P,Q) &= -\sum_{v\in\mathcal{V}}\sum_{h\in\mathcal{H}} Q(v|h)\log_2 P(v|h) \approx -\frac{1}{K}\sum_{v\in\mathcal{V}}\sum_{h\in\mathcal{H}} C(h,v)\log_2 P\left(v|h\right) \\ &= \approx -\frac{1}{K}\sum_{k=1}^{K}\log_2 P\left(w_k|W_{k-n+1}^{k-1}\right) = -\mathcal{L}(\theta) \end{aligned}$$

where P is the n-gram model to be evaluated. Q is represented by some held out data.

Likelihood & Perplexity (II)

Note that the perplexity can be rewritten as

$$
\begin{aligned}
PP = 2^{-\mathcal{L}(\theta)} &= 2^{-\frac{1}{K}\sum_{k=1}^{K}\log_2 P\left(w_k|W_{k-n+1}^{k-1}\right)} \\
&= \left(2^{-\sum_{k=1}^{K}\log_2 P\left(w_k|W_{k-n+1}^{k-1}\right)}\right)^{\frac{1}{K}} \\
&= \left(\prod_{k=1}^{K} 2^{-\log_2 P\left(w_k|W_{k-n+1}^{k-1}\right)}\right)^{\frac{1}{K}} \\
&= \left(\prod_{k=1}^{K} \frac{1}{P\left(w_k|W_{k-n+1}^{k-1}\right)}\right)^{\frac{1}{K}}
\end{aligned}
$$

Therefore, the perplexity is the *geometry mean* of the inverse probabilities of the n-grams. It can be interpreted as the *average branching factor*. Hence, higher perplexity leads to higher branching factors, *i.e.* lower prediction power.

n-gram Language Model – Unigram Example

Training sentences:

<S> I HAVE A RED CAR </S>
<S> I BUY A NEW CAR </S>
<S> THEY HAVE A NEW BOOK </S>

Vocabulary:

A	BOOK	BUY	CAR	HAVE	I	NEW	RED	THEY

Unigram language model:

$P(\text{A})$	3/15	$P(\text{BOOK})$	1/15	$P(\text{BUY})$	1/15
$P(\text{CAR})$	2/15	$P(\text{HAVE})$	2/15	$P(\text{I})$	2/15
$P(\text{NEW})$	2/15	$P(\text{RED})$	1/15	$P(\text{THEY})$	1/15

Test sentence: <S> I BUY A NEW BOOK </S>

Probability:

$$\begin{aligned} P(W) &= P(\text{I})P(\text{BUY})P(\text{A})P(\text{NEW})P(\text{BOOK}) \\ &= 2/15 \times 1/15 \times 3/15 \times 2/15 \times 1/15 = 12/15^5 = 1.58 \times 10^{-5} \\ \text{PP} &= 2^{-\frac{1}{5}\log_2(P(W))} \approx 9.13 \end{aligned}$$

n-gram Language Model – Bigram Example

Training sentences:

<S> I HAVE A RED CAR </S>
<S> I BUY A NEW CAR </S>
<S> THEY HAVE A NEW BOOK </S>

Bigram language model:

$P(\mathrm{I}\mid <\mathrm{S}>)$	$2/3$	$P(\mathrm{THEY}\mid <\mathrm{S}>)$	$1/3$	$P(\mathrm{NEW}\mid \mathrm{A})$	$2/3$
$P(\mathrm{RED}\mid \mathrm{A})$	$1/3$	$P(\mathrm{A}\mid \mathrm{BUY})$	$1/1$	$P(\mathrm{A}\mid \mathrm{HAVE})$	$2/2$
$P(\mathrm{HAVE}\mid \mathrm{I})$	$1/2$	$P(\mathrm{BUY}\mid \mathrm{I})$	$1/2$	$P(\mathrm{CAR}\mid \mathrm{NEW})$	$1/2$
$P(\mathrm{BOOK}\mid \mathrm{NEW})$	$1/2$	$P(\mathrm{CAR}\mid \mathrm{RED})$	$1/1$	$P(\mathrm{HAVE}\mid \mathrm{THEY})$	$1/1$

Test sentence: <S> I BUY A NEW BOOK </S>

Probability:

$$
\begin{aligned}
P(W) &= P(\mathrm{I}\mid <\mathrm{S}>)P(\mathrm{BUY}\mid \mathrm{I})P(\mathrm{A}\mid \mathrm{BUY})P(\mathrm{NEW}\mid \mathrm{A})P(\mathrm{BOOK}\mid \mathrm{NEW}) \\
&= 2/3 \times 1/2 \times 1/1 \times 2/3 \times 1/2 = 1/9 = 1.11 \times 10^{-1} \\
\mathrm{PP} &= 2^{-\frac{1}{5}\log_2(P(W))} \approx 1.55
\end{aligned}
$$

Coping With Rare Events

So, higher n-gram order yields better prediction power. However, as n increases, the amount of training data required to obtain a robot estimation of the probabilities also increases. There will be issues of **unseen** n-grams. In our previous examples, the 4-gram `BUY A NEW BOOK` in the test sentence is not seen in any of the training sentences. **How to cope with this problem?**

Possible solutions to handle unseen n-grams and improve robustness of estimation:

- **Discounting**:
  – Discounting modifies the original counts so as to redistribute the probability mass from the more commonly observed events to the less frequent and unseen events.

$$C(W) \Rightarrow d(C(W))C(W)$$

  where $d(n)$ is known as the discount ratio.
- **Back-off modelling**:
  – A back-off strategy recursively backs-off to lower n-gram order until a robust probability estimation is available

Good Turing Discounting

The idea of Good-Turing Discounting is to reallocate the probability mass of n-gram that occur r+1 times the in training data to the n-gram that occur r times. More importantly, it reallocate the probability mass of n-grams that occur one time to the n-grams that have zero occurrence (unseen). Hence, the Good-Turing discounting can be expressed as

$$r^* = (r+1)\frac{N_{r+1}}{N_r}$$

where N_r is the count of counts, *i.e.* the number of n-gram seen exactly r times. Note that $N = \sum_{r=0}^{\infty} r^* N_r = \sum_{r=1}^{\infty} r N_r$ is the total number of observed n-grams. Hence, the probability of an n-gram is given by:

$$P(x : C(x) = r) = \frac{r^*}{N}$$

Usually, smoothing/adjustment is applied to N_r before Good-Turing discounting.

Good Turing Discounting – Example

Given the following counts:

Symbol	A	B	C	D	E	F	G	H	I	J
Counts	1	1	0	2	1	2	2	0	1	3

Frequency table: (applying $r^* = (r+1)\frac{N_{r+1}}{N_r}$)

r	Symbols	N_r	rN_r	$P(x)$
0	C, H	2	0	0
1	A, B, E, I	4	4	4/13
2	D, F, G	3	6	6/13
3	J	1	3	3/13
Total			13	1

$\rightarrow$

r^*	Symbols	N_r	r^*N_r	$P(x)$
2	C, H	2	4	4/13
3/2	A, B, E, I	4	6	6/13
1	D, F, G	3	3	3/13
1	J	1	0	0
Total			13	1

Note that the probabilities for those occured r times have been relocated to those occured $r-1$ times.

Jelinek-Mercer Smoothing (Interpolation)

Under Good-Turing discounting, unseen n-grams are given the same modified counts. So,

$$\left.\begin{array}{rcl} C(\text{BE,HAPPY}) & = & 0 \\ C(\text{BE,NONCHALANT}) & = & 0 \end{array}\right\} \Rightarrow P(\text{HAPPY}|\text{BE}) = P(\text{NONCHALANT}|\text{BE})$$

This doesn't seem right because $P(\text{HAPPY}) > P(\text{NONCHALANT})$. Solution: **Interpolate** between unigram and bigram:

$$P^*(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i)$$

Can also apply interpolation recursively:

$$P^*(w_i|W_{i-n+1}^{i-1}) = \lambda_n P(w_i|W_{i-n+1}^{i-1}) + (1-\lambda_n)P^*(w_i|W_{i-n+2}^{i-1})$$

Stop recursion unigram model. Learn λ_n on *held-out* data.

Katz's Back-off Model

Katz back-off is a generative n-gram language model that estimates the conditional probability of a word given its history in the n-gram. It accomplishes this estimation by "backing-off" to models with smaller histories under certain conditions. By doing so, the model with the most reliable information about a given history is used to provide the better results.

The equation for Katz's back-off model is

$$P_{\text{bo}}(w_k|W_{k-n+1}^{k-1}) = \begin{cases} d(W_{k-n+1}^{k-1})\frac{C(W_{k-n+1}^{k})}{C(W_{k-n+1}^{k-1})} & C(W_{k-n+1}^{k}) > k \\ \alpha(W_{k-n+1}^{k-1})P_{\text{bo}}(w_k|W_{k-n+2}^{k-1}) & \text{otherwise} \end{cases}$$

k is typically set to zero (back-off only if n-gram is **unseen**). d and α are the discount factor and back-off weights respectively. The Good-Turing discounting is commonly used. α is chosen such that

$$\sum_{k=1}^{K} P_{\text{bo}}(w_k|W_{k-n+1}^{k-1}) = 1$$

Katz's Back-off Model – Example

Given the sequence: `C B C C A B C`

Probabilities:

History	A	B	C
—	1/7	2/7	4/7
A	0	1	0
B	0	0	1
C	1/3	1/3	1/3

$\Rightarrow$

History	d	α	A	B	C
—	–	–	1/7	2/7	4/7
A	5/7	2/5	2/35	5/7	8/35
B	6/7	1/3	1/21	2/21	6/7
C	1	0	1/3	1/3	1/3

Example: consider bigrams with `B` as history, $P(A|B)$ and $P(B|B)$ are unseen. Discount seen bigrams by $d_B = 6/7$. We have:

$$P_{\text{bo}}(A|B) = \alpha_B P(A) \qquad P_{\text{bo}}(B|B) = \alpha_B P(B) \qquad P_{\text{bo}}(C|B) = d_B P(C|B)$$

Equating $P_{\text{bo}}(A|B) + P_{\text{bo}}(B|B) + P_{\text{bo}}(C|B) = 1$ yields $\alpha_B = 1/3$. This gives:

$$P_{\text{bo}}(A|B) = \frac{1}{3} \times \frac{1}{7} = \frac{1}{21} \qquad P_{\text{bo}}(B|B) = \frac{1}{3} \times \frac{2}{7} = \frac{2}{21} \qquad P_{\text{bo}}(C|B) = \frac{6}{7} \times 1 = \frac{6}{7}$$

Interpolation of Language Models

Previously we have examined the Jelinek-Mercer Smoothing technique which interpolates the probabilities of different n-gram orders. Alternatively, it is also possible to interpolate language models of the same n-gram order, but trained on text data collected from different domain. Linear interpolation of multiple n-gram LM can be expressed as:

$$P(w_i|W_{i-n+1}^{i-1}) = \sum_{m=1}^{M} \lambda_m P_m(w_i|W_{i-n+1}^{i-1})$$

The interpolation weights λ_m can be estimated by maximising the log likelihood on some heldout data.

Interpolation also provides a quick way of performing language model adaptation by adjusting the weights. For example, given two language models trained on news and conversation transcriptions, an interpolated LM suitable for scientific articles can be estimated by adjusting the interpolation weights on some small amount of sample data.

Category-based Language Models

Category-based language models use parameter tying approach to obtain more robust estimate of the n-gram probabilities, particularly for high order n-grams. This is achieved by grouping the n-gram history into different categories. Typically, grouping is performed at the word level, either based on expert knowledge (part-of-speech) or automatically determined (*e.g.* based on bigram statistics).

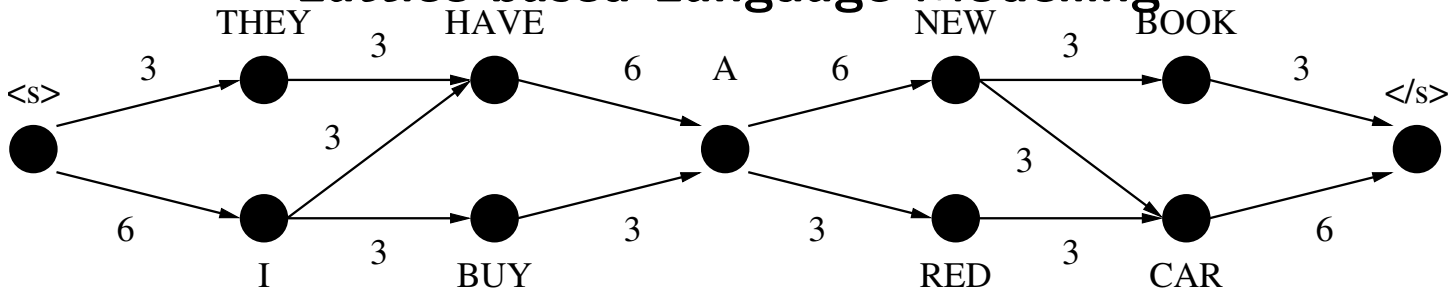
$$P(w_i|W_{i-n+1}^{i-1}) = P(w_i|G_i)P(G_i|G_{i-n+1}^{i-1})$$

where $G(w)$ denote the function which maps the word w_i to its group. Grouping reduces the number of distinct history which in turn reduces the chances of having unseen or rare n-grams.

- `I (NOUN)` `BUY (VERB)` `A (ARTICLE)` `NEW (ADJECTIVE)` `BOOK (NOUN)`
- `HE (NOUN)` `SOLD (VERB)` `A (ARTICLE)` `BLUE (ADJECTIVE)` `CAR (NOUN)`

Usually, category-based language models are interpolated with word-based language models to yield improved performance for speech recognition.

Lattice-based Language Modelling



Lattices encodes multiple sentences. The above lattice represents 9 sentences:

Sentences	Probability
<S> I BUY A RED CAR </S>	1/25
<S> I BUY A NEW CAR </S>	3/25
<S> I BUY A NEW BOOK </S>	1/25
<S> I HAVE A RED CAR </S>	3/25
<S> I HAVE A NEW CAR </S>	9/25
<S> I HAVE A NEW BOOK </S>	3/25
<S> THEY HAVE A RED CAR </S>	1/25
<S> THEY HAVE A NEW CAR </S>	3/25
<S> THEY HAVE A NEW BOOK </S>	1/25

Lattice Forward-backward Algorithm

Compute expected counts using forward-backward algorithm. Define the **forward** and **backward** probabilities as the following recursions:

$$\alpha_j = \sum_{i \in \mathcal{P}_j} a_{ij}\alpha_i \quad \text{and} \quad \beta_i = \sum_{j \in \mathcal{S}_i} a_{ij}\beta_i \begin{cases} \mathcal{P}_j & = & \text{the set of preceeding nodes of } j \\ \mathcal{S}_i & = & \text{the set of succeeding nodes of } i \\ a_{ij} & = & \text{the transition scores from state } i \text{ to state } j \end{cases}$$

The boundary conditions are $\alpha_1 = 1$ abd $\beta_N = 1$ where 1 and N denote the start and end nodes respectively. Hence, the posterior probability of a given node j is given by:

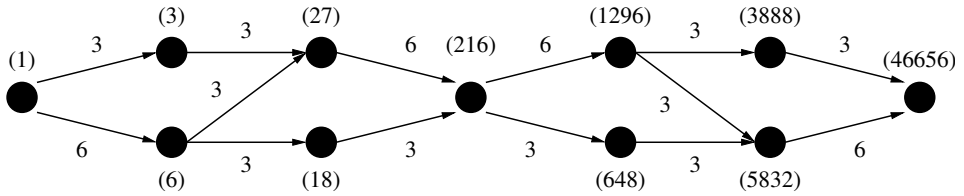
$$\gamma_j = \frac{\alpha_j \beta_j}{\alpha_N} \quad \text{and} \quad \gamma_{ij} = \frac{\alpha_i a_{ij} \beta_j}{\alpha_N}$$

Since each node represents a word, γ_j and γ_{ij} can bew viewed as the probabilistic counts for the unigram i and bigram i, j respectively. Hence,

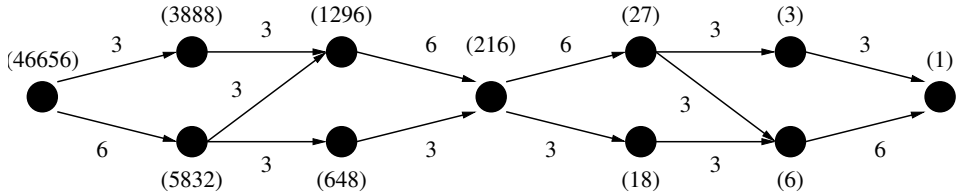
$$P(j) = \frac{\gamma_j}{\sum_j \gamma_j} \quad \text{and} \quad P(j|i) = \frac{\gamma_{ij}}{\sum_j \gamma_{ij}} = \frac{\gamma_{ij}}{\gamma_i}$$

Probabilistic Counts

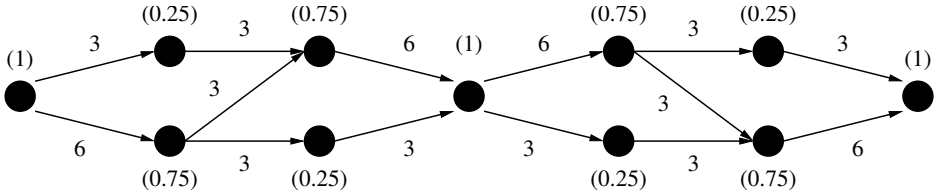
- α_j:



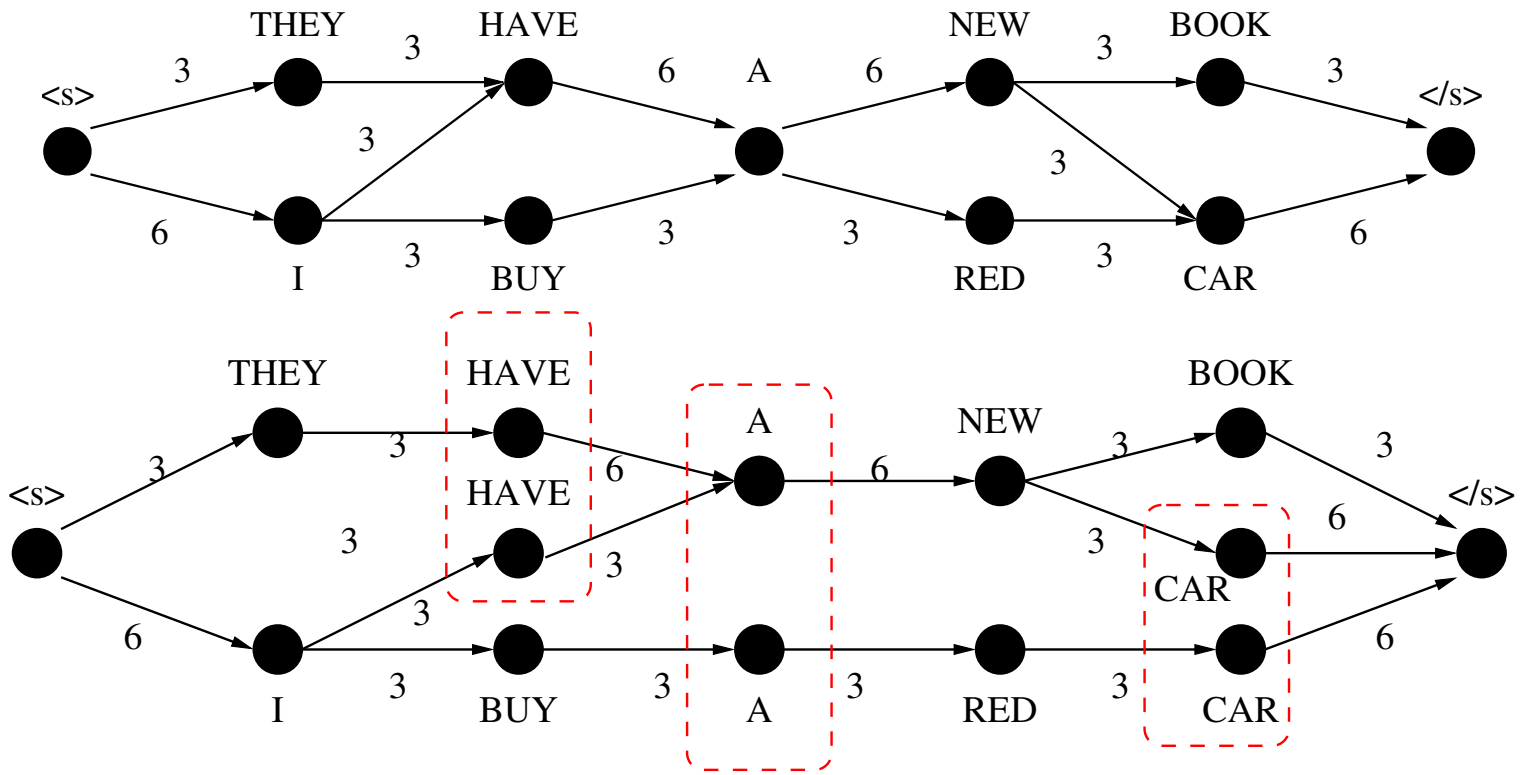
- β_j:



- γ_j:



Lattice Expansion



The previous lattice encodes only the unigram contexts. However, to handle higher order n-grams, it may be necessary to perform lattice expansion to keep each node with different history **distinct**.

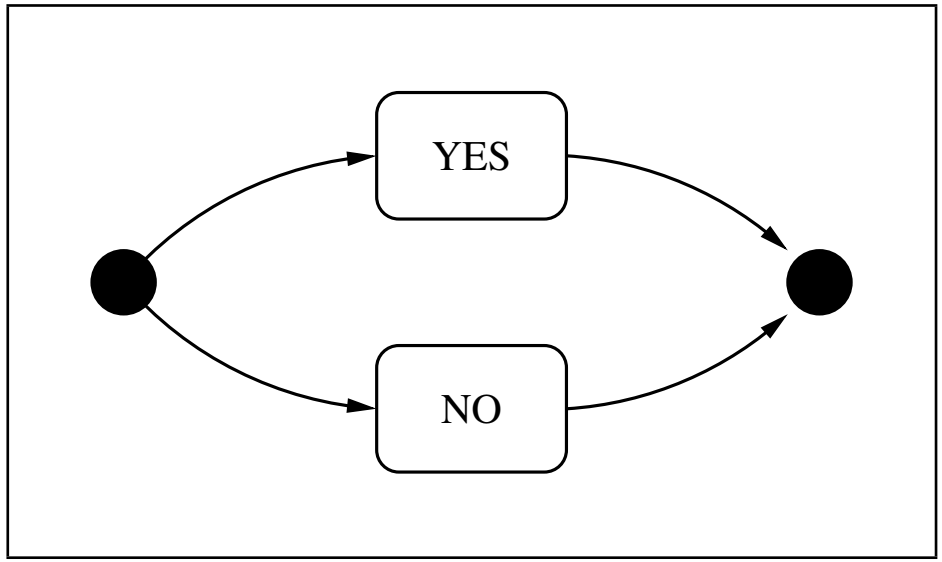
Building decoding network with HTK

HTK provides two tools for building recognition networks:

- `HBuild` – build from word list, bigram matrix
- `HParse` – build from grammar rules

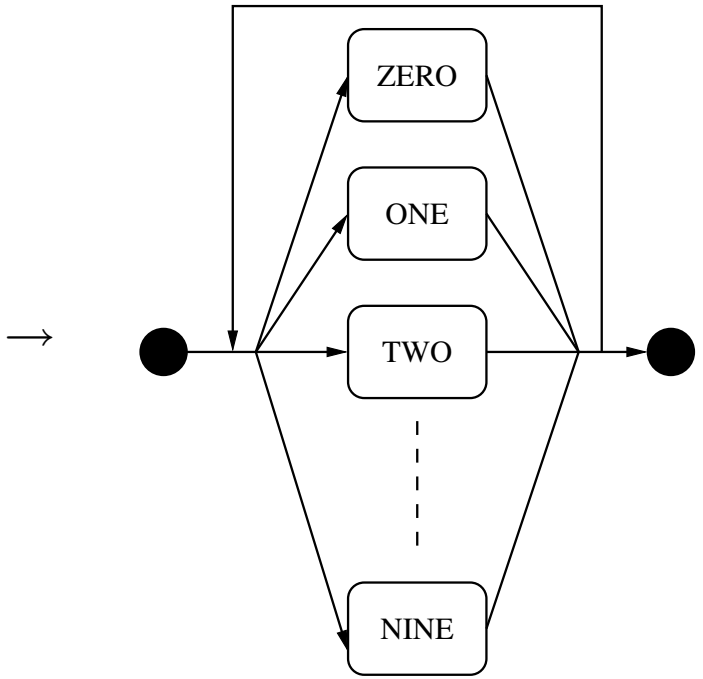
HTK's standard lattice format:

```
N=4 L=4
I=0 W=!NULL
I=1 W=YES
I=2 W=NO
I=3 W=!NULL
J=0 S=0 E=1
J=1 S=0 E=2
J=2 S=1 E=3
J=3 S=2 E=3
```



Simple word loop using HBuild

ZERO
ONE
TWO
...
NINE



Command:

```
> HBuild -t <s> </s> wordlist network
```

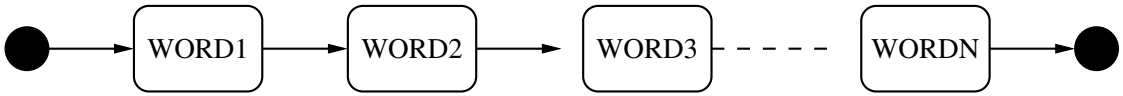
Defining sequences using `HParse`

Grammar:

WORD1 WORD2 WORD3 ... WORDN

↓

Network:



- Space (' ') separated words are joined in sequence
- Command:

```
> HParse grammar network
```

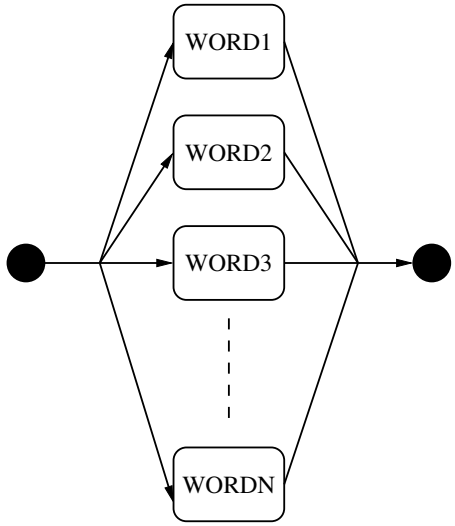
Defining alternatives using `HParse`

Grammar:

```
WORD1 | WORD2 | WORD3 | ... | WORDN
```

→

Network:

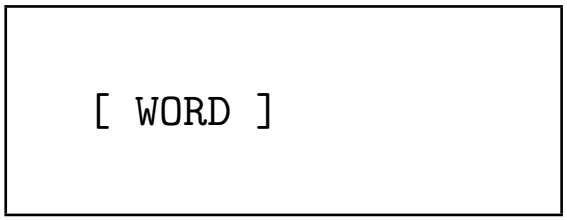


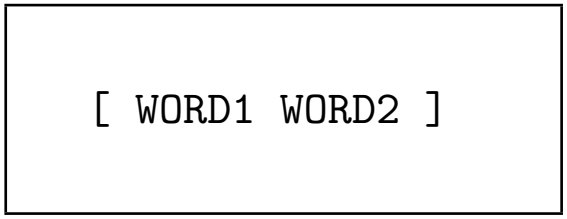
- Vertical bar ('|') separated words are joined in parallel
- Command:

  ```
  > HParse grammar network
  ```

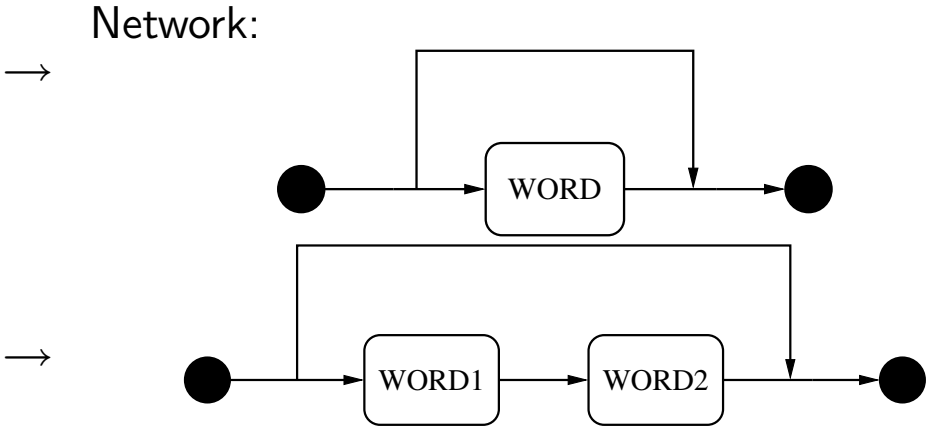
Defining skips using `HParse`

Grammar:





Network:

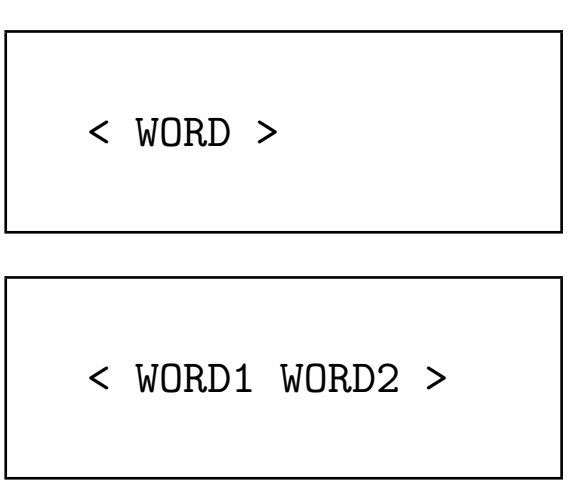


- Square brackets '[...]' are used to enclose **optional** words/phrases (skips)
- Command:

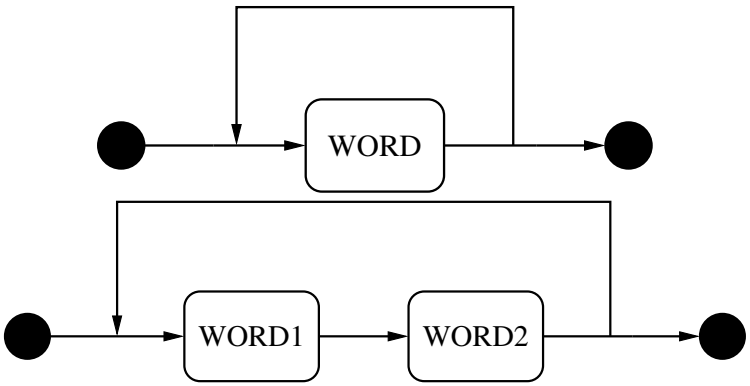
```
> HParse grammar network
```

Defining repetitions using `HParse`

Grammar:



Network:



- Angle brackets '$<$ ... $>$' are used to enclose **repeating** words/phrases
- Command:

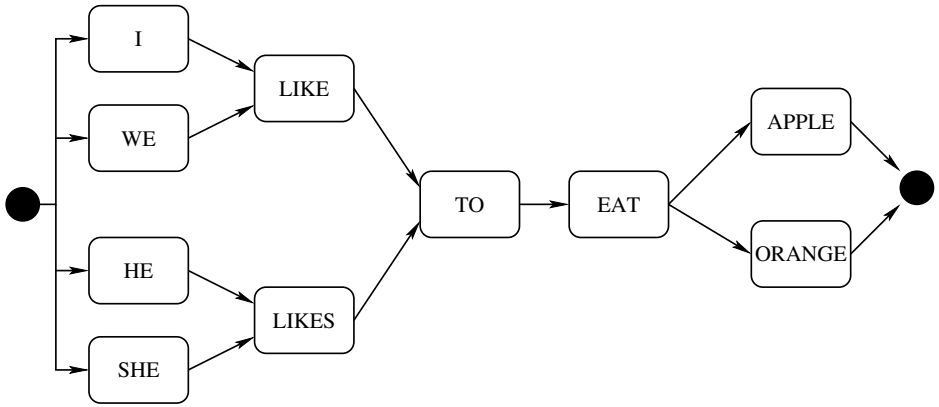
```
> HParse grammar network
```

Combining multiple rules using `HParse`

Grammar:

```
( (I | WE) LIKE) | ( (HE | SHE) LIKES) TO EAT (APPLE | ORANGE)
```

Network:



- Round brackets '(...)' are used to enclose **groups** of words/phrases
- Command:

  ```
  > HParse grammar network
  ```

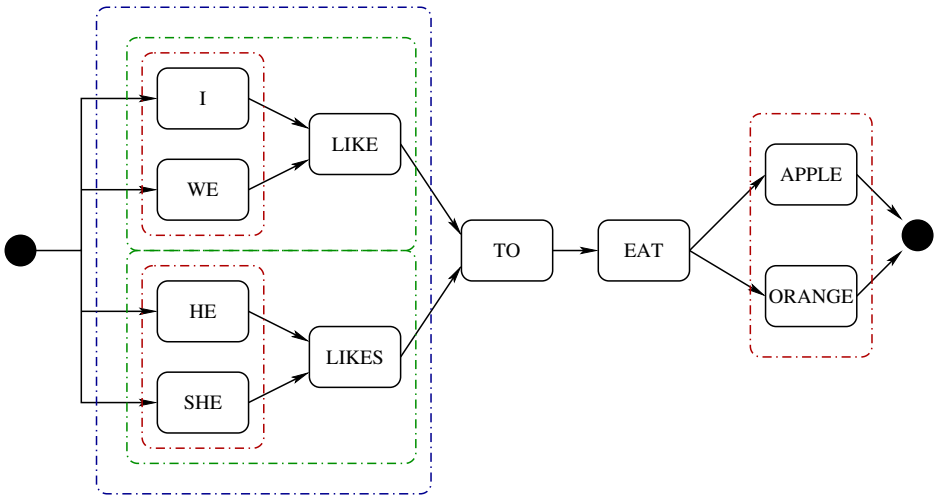
Defining and using subnets using `HParse`

Grammar:

```
$plural = I|WE;
$singular = HE|SHE;
$prefix1 = $plural LIKE;
$prefix2 = $singular LIKES;
$fruits = APPLE|ORANGE;
$prefix = $prefix1|$prefix2;

( $prefix TO EAT $fruit )
```

Network:



- Define subnets using '`$subnet = ... ;`'
- Command:

  `> HParse grammar network`

Recap

- Language model for speech recognition
  - Grammar network
  - Statistical language model (n-gram)
- Parameter estimation
- Evaluation – perlexity
- Discounting & backoff
- Category-based LM
- Lattice-based LM
- Network building using HTK