NLP: Hidden Markov Models

AMITDHOMNE

1 Tagging

- Named entities
- Parts of speech

2 Parts of Speech

Tagsets

- Google Universal Tagset, 12: Noun, Verb, Adjective, Adverb, Pronoun, Determiner, Adposition (prepositions and postpositions), Numerals, Conjunctions, Particles, Punctuation, Other
- Penn Treebank, 45. Includes 4 categores of noun, 4 categories of pronoun, and 6 categories of verb.
- Brown Corpus, 87

Deciding what categories

- Auxilliary verbs? "I *had* gone"
- Numbers as adjectives? "I saw 4 cars"
- Count vs. mass nouns? "some apple" vs. "two apples", "some snow" vs. "two snows"

Uses

- Parsing: determiner and noun should connect first, then to verb
- Speech synthesis: OBject (noun) vs. obJECT (verb), CONtent (noun) vs. conTENT (adj)

Rule-based

- "if it ends in '-tion' " → Noun
- "if it ends in '-ize' " → Verb
- "if it starts with 're-' " → Verb
- "if it starts with a capital letter" → Proper Noun
- "if it's preceded by 'the' " → Noun
- "if it's followed by *'s* " → Noun
- "if it's preceded by an adjective" → Noun
- Or just memorize a big list of words and tags?
  - Ambiguity?
  - Picking the most common tag for a word → 90% accuracy (thought state-of-the-art is 97%)

Open vs. Closed

- Open class tags: new words are frequently added
  - noun, verb, adjective, adverb
  - "Twitter", to "tweet", "twitterish"
- Closed class tags: new words are rarely added
  - determiner, preposition, pronouns, ...
  - Don't want to completely close off new words
  - Maybe *alongside* (preposition) wasn't seen in training, only test
  - New domains: "u" as a pronoun

Complexitites:

- Ambiguity:
  - "buy a book" (noun) vs. "book a flight" (verb)
  - "talk over the deal" (particle) vs. "talk over the phone" (prep)
- Phrasal verbs: "turn down", "rule out"
  - "he went on for days"
  - "he went on a boat"

3 Hidden Markov Models

Similar to N-Gram models

- Model the text as a *sequence*
  - Bad assumption, but less sparse
- For ngrams, we modeled the probability of each word conditioned on the previous n-1 words.
- Here, we model each *tag* conditioned on previous *tags*
- Still uses Markov assumption: only look back a few tags
  - Again, bad assumption, but less sparse
- Note that we have to condition words on tags because otherwise

\\ Day 2

Derivation

- We want the most likely tag sequence for a sequence of words:

  $$p(\langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle \mid \langle \text{S} \rangle\ w_1\ w_2\ ...\ w_n\ \langle \text{E} \rangle)$$

  Remember that order matters!
- For simplicity, we'll write this as $t_1^n = \langle t_1, t_2, ..., t_n \rangle$
- So we want

  $$\hat{t}_1^n = \text{argmax}_{t_1^n}\ p(\langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle \mid \langle \text{S} \rangle\ w_1\ w_2\ ...\ w_n\ \langle \text{E} \rangle)$$

  but it's hard to estimate
- Like for ngrams, use Bayes rule:

$$\begin{aligned}\hat{t}_1^n &= \text{argmax}_{t_1^n}\ \frac{p(\langle \text{S} \rangle\ w_1\ w_2\ ...\ w_n\ \langle \text{E} \rangle \mid \langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle) \cdot p(\langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle}{p(\langle \text{S} \rangle\ w_1\ w_2\ ...\ w_n\ \langle \text{E} \rangle)} \\ &= \text{argmax}_{t_1^n}\ p(\langle \text{S} \rangle\ w_1\ w_2\ ...\ w_n\ \langle \text{E} \rangle \mid \langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle) \cdot p(\langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle)\end{aligned}$$

- Two major independence assumptions:
  - Like ngrams, assume probability of a sequence is dependent only on recent past:

$$\begin{aligned}p(\langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle) &\approx p(t_1 \mid \langle \text{S} \rangle) \cdot p(t_2 \mid t_1) \cdot p(t_2 \mid t_2) \cdot ... \cdot p(t_n \mid t_{n-1}) \cdot p(\langle \text{E} \rangle \mid t_n) \\ &= \prod_{i=1}^{n} p(t_i \mid t_{i-1})\end{aligned}$$

– Also assume word is only dependent on its tag:

$$p(\langle \text{S} \rangle\ w_1\ w_2\ ...\ w_n\ \langle \text{E} \rangle \mid \langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle) \approx p(w_1 \mid t_1) \cdot p(w_2 \mid t_2) \cdot ... \cdot p(w_n \mid t_n)$$
$$= \prod_{i=1}^{n} p(w_i \mid t_i)$$

– Together:

$$\begin{aligned}
\hat{t}_1^n &= \text{argmax}_{t_1^n}\ p(\langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle \mid \langle \text{S} \rangle\ w_1\ w_2\ ...\ w_n\ \langle \text{E} \rangle) \\
&= \text{argmax}_{t_1^n}\ p(\langle \text{S} \rangle\ w_1\ w_2\ ...\ w_n\ \langle \text{E} \rangle \mid \langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle) \cdot p(\langle \text{S} \rangle\ t_1\ t_2\ ...\ t_n\ \langle \text{E} \rangle) \\
&\approx \text{argmax}_{t_1^n}\ \prod_{i=1}^{n} p(w_i \mid t_i) \cdot \prod_{i=1}^{n} p(t_i \mid t_{i-1}) \\
&= \text{argmax}_{t_1^n}\ \prod_{i=1}^{n} p(w_i \mid t_i) \cdot p(t_i \mid t_{i-1})
\end{aligned}$$

4 Estimating Parameters: MLE

Two probability distributions to estimate:

- Transitions: probability of a tag, given previous tag, $p(t_i \mid t_{i-1})$
- Emissions: probability of a word, given its tag, $p(w_i \mid t_i)$

MLE

- MLE estimation is just like before (naïve Bayes, ngrams, ...): normalized counts
- Transitions: $p(t_i \mid t_{i-1}) = \frac{C(t_{i-1}\ t_i)}{\sum_x C(t_{i-1}\ x)} = \frac{C(t_{i-1}\ t_i)}{C(t_{i-1})}$
- Emissions: $p(w_i \mid t_i) = \frac{C(t_i, w_i)}{\sum_x C(t_i, x)} = \frac{C(t_i, w_i)}{C(t_i)}$

Example dataset (punctuation excluded for simplicity!):

```
<S>|<S> the|D man|N walks|V the|D dog|N <E>|<E>
<S>|<S> the|D dog|N runs|V <E>|<E>
<S>|<S> the|D dog|N walks|V <E>|<E>
<S>|<S> the|D man|N walks|V <E>|<E>
<S>|<S> a|D man|N saw|V the|D dog|N <E>|<E>
<S>|<S> the|D cat|N walks|V <E>|<E>
```

Some probabilities:

- $p(t_i = \text{V} \mid t_{i-1} = \text{N}) = \frac{C(\text{N V})}{\sum_x C(\text{N } x)} = \frac{6}{8} = 0.75$

- $p(t_i = \text{D} \mid t_{i-1} = \text{N}) = \frac{C(\text{N D})}{\sum_x C(\text{N } x)} = \frac{0}{8} = 0.0$
- $p(w_i = \text{dog} \mid t_i = \text{N}) = \frac{C(\text{N,dog})}{\sum_x C(\text{N},x)} = \frac{4}{8} = 0.50$
- $p(w_i = \text{the} \mid t_i = \text{N}) = \frac{C(\text{N,the})}{\sum_x C(\text{N},x)} = \frac{0}{8} = 0.0$

5 Add-λ Smoothing

Again, just like before.

- Transitions: $p(t_i \mid t_{i-1}) = \frac{C(t_{i-1}\ t_i)+\lambda}{\sum_x (C(t_{i-1}\ x)+\lambda)} = \frac{C(t_{i-1}\ t_i)+\lambda}{(\sum_x C(t_{i-1}\ x))+\lambda \cdot |T|} = \frac{C(t_{i-1}\ t_i)+\lambda}{C(t_{i-1})+\lambda \cdot |T|}$
- Emissions: $p(w_i \mid t_i) = \frac{C(t_i,w_i)+\lambda}{\sum_x (C(t_i,x)+\lambda)} = \frac{C(t_i,w_i)+\lambda}{(\sum_x C(t_i,x))+\lambda \cdot |V|} = \frac{C(t_i,w_i)+\lambda}{C(t_i)+\lambda \cdot |V|}$

Some probabilities (with $\lambda = 2$):

- The Vocabulary V is the set of all word types that might be associated with a tag t_i:
  $V = \{a, cat, dog, man, runs, saw, the, walks\}, \quad |V| = 8$
- The Tagset T is the set of all possible tags that could follow a tag t_{i-1}:
  $T = \{D, N, V, \langle \text{E} \rangle\}, \quad |T| = 4$

- $p(t_i = \text{V} \mid t_{i-1} = \text{N}) = \frac{C(\text{N V})+2}{\sum_{x \in T}(C(\text{N } x)+2)} = \frac{C(\text{N V})+2}{(\sum_{x \in T} C(\text{N } x))+(2 \cdot 4)} = \frac{6+2}{8+8} = \frac{8}{16} = 0.50$
- $p(t_i = \text{D} \mid t_{i-1} = \text{N}) = \frac{C(\text{N D})+2}{\sum_{x \in T}(C(\text{N } x)+2)} = \frac{C(\text{N D})+2}{(\sum_{x \in T} C(\text{N } x))+(2 \cdot 4)} = \frac{0+2}{8+8} = \frac{2}{16} = 0.125$
- $p(w_i = \text{dog} \mid t_i = \text{N}) = \frac{C(\text{N,dog})}{\sum_{x \in V}(C(\text{N},x)+2)} = \frac{C(\text{N,dog})}{(\sum_{x \in V} C(\text{N},x))+(2 \cdot 8)} = \frac{4+2}{8+16} = 0.25$
- $p(w_i = \text{the} \mid t_i = \text{N}) = \frac{C(\text{N,the})}{\sum_{x \in V}(C(\text{N},x)+2)} = \frac{C(\text{N,the})}{(\sum_{x \in V} C(\text{N},x))+(2 \cdot 8)} = \frac{0+2}{8+16} = 0.08$

Differences:

- Two distributions, two kinds of smoothing
- Can do add-λ for both, but don't need to use same λ
- Can use totally different smoothing for each.

6 One-Count Smoothing

From Chen and Goodman (1996)

Idea: Smooth conditional distribution by interpolating the MLE with a unigram distribution where the degree of smoothing is proportional to the "openness" of a tag.

$$p(t_i \mid t_{i-1}) = \frac{C(t_{i-1}, t_i) + \alpha(t_{i-1}) \cdot p(t_i)}{C(t_{i-1}) + \alpha(t_{i-1})} \qquad \text{where} \quad \alpha(t_{i-1}) = |t_i : C(t_{i-1}, t_i) = 1|$$

$$p(w_i \mid t_i) = \frac{C(t_i, w_i) + \beta(t_i) \cdot p(w_i)}{C(t_i) + \beta(t_i)} \qquad \text{where} \quad \beta(t_i) = |w_i : C(t_i, w_i) = 1|$$

- For emissions, if a tag t_i appears one time with a large number of word types, then we assume that it is very "open": there is a large number of words that it has only been seen with once, so the likelihood that it *could have* been seen with some other word, even though it didn't *happen* to be seen with it, is high.
- For transitions, the reasoning is the same. Instead of a tag being associated with a wide variety of words once, it's a wide variety of subsequent tags once.
- Note that it is still necessary to smooth the prior distributions $p(t_{i-1})$ and $p(w_i)$ since if the transition or emission count is zero, we need to ensure that we back off to some non-zero value; add-1 smoothing should be sufficient. Additionally, It is necessary to add some small amount to $\alpha(t_{i-1})$ and $\beta(t_i)$ in case there are no singletons

7 Three Tasks for HMM

1. Likelihood: Given a tagged sequence, determine its likelihood
2. Decoding: Given an untagged sequence, determine the best tag sequence for it
3. Learning: Given an untagged sequence, and a set of tags, learn the HMM parameters.

8 Likelihood of a tagged sentence

We can compute the likelihood of a particluar sequence of tags for a sentence:

- $p(t_1 ... t_n \mid w_1 ... w_n) = \prod_{i=1}^{n} p(w_i \mid t_i) \cdot p(t_i \mid t_{i-1})$

Example: "the|D dog|N walks|V"

$$\begin{aligned} p(t_1 ... t_n \mid w_1 ... w_n) &\approx \prod_{i=1}^{n} p(w_i \mid t_i) \cdot p(t_i \mid t_{i-1}) \\ &= p(D \mid \langle \text{S} \rangle) \cdot p(the \mid D) \cdot p(N \mid D) \cdot p(dog \mid N) \cdot p(V \mid N) \cdot p(walks \mid V) \cdot p(\langle \text{E} \rangle \mid V) \end{aligned}$$

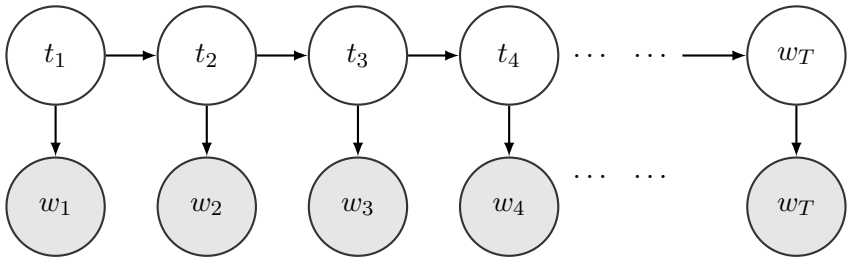

Figure 1: Second-order HMM showing conditional dependencies

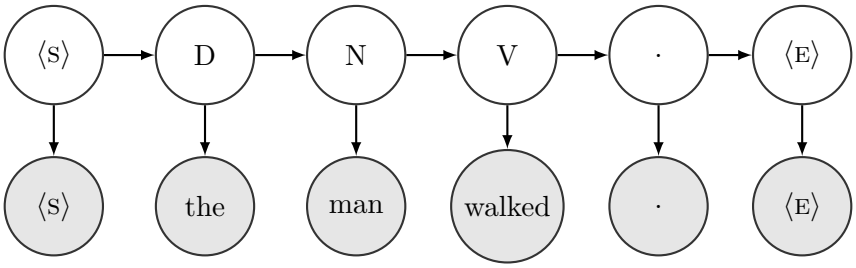

Figure 2: Second-order HMM for the sentence "the man walks ."

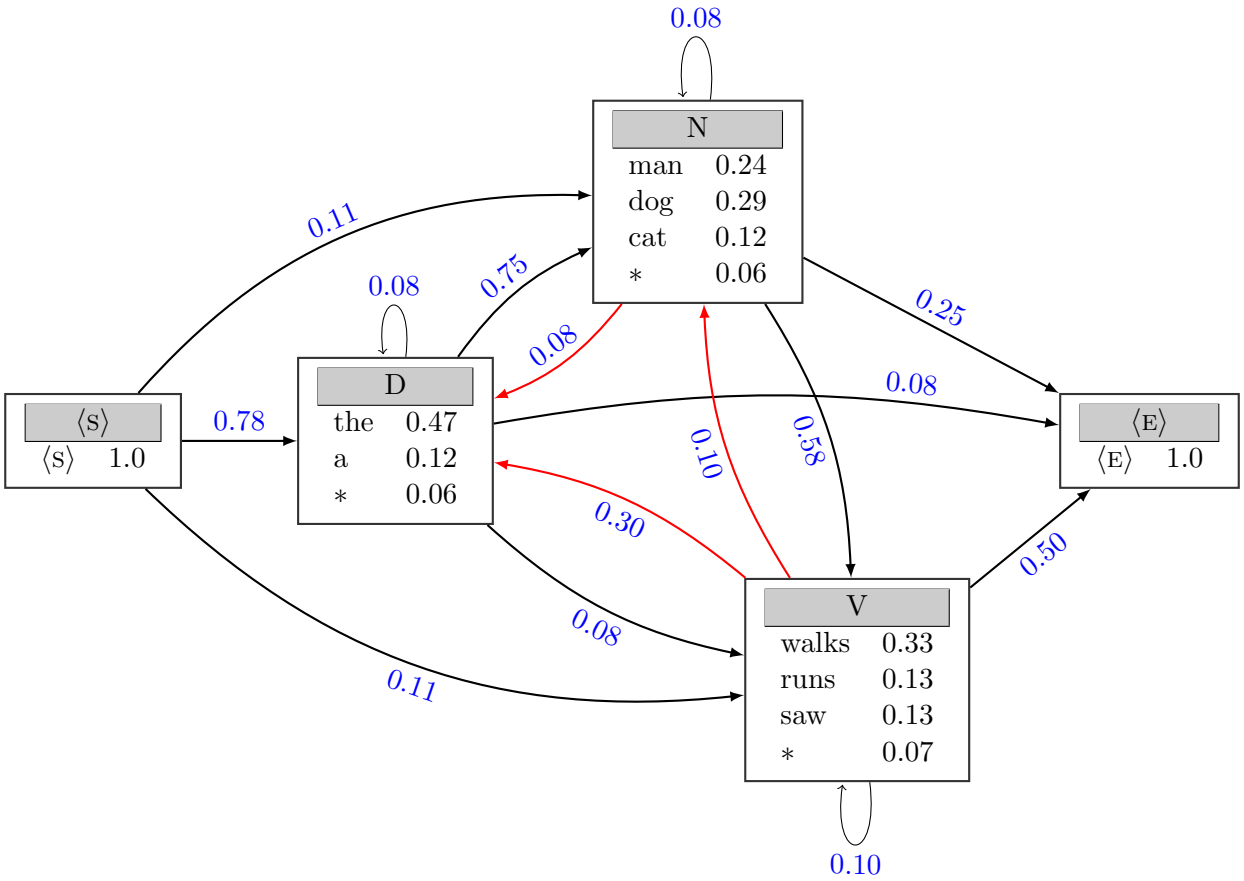

Figure 3: Finite state machine of the data with add-1 smoothing. Missing arrows are assumed to be zero probabilities. With smoothing, there is an arrow in *both directions* between every pair of tags.

- This says that the probability of the tag sequence is the probability of seeing *D* as the first tag, *the* given that the chosen first tag was *D*, *N* given that the previous tag was *D*, *dog* given that the chosen second tag was *N*, etc...
- Transition ($p(t_i \mid t_{i-1})$) and emission ($p(w_i \mid t_i)$) values are calculated as above

9 Decoding

Tagging a sentence

- $\hat{t}_i^n = \text{argmax}_{t_1^n}\ p(t_1 ... t_n \mid w_1 ... w_n) \approx \text{argmax}_{t_1^n}\ \prod_{i=1}^{n} p(w_i \mid t_i) \cdot p(t_i \mid t_{i-1})$
- For "the dog walks", we have to check all combinations:

  ⟨S⟩ D D D ⟨E⟩

  ⟨S⟩ N D D ⟨E⟩

  ⟨S⟩ V D D ⟨E⟩

  ⟨S⟩ . D D ⟨E⟩

  ⟨S⟩ N N D ⟨E⟩

  ⟨S⟩ N V D ⟨E⟩

  ...
- But this is insane.

\\ Day 3

Choosing a tag

- Because of our independence assumptions, the probability of each tag is dependent only on its "Markov blanket": the previous tag, emitted word, and next tag.

$$p(t_i \mid \langle S \rangle\ w_1\ w_2\ ...\ w_n\ \langle E \rangle, \langle S \rangle\ t_1\ t_2\ ...\ \mathbf{t_{i-1}}\ \mathbf{t_{i+1}}\ ...\ t_n\ \langle E \rangle) = p(t_i \mid t_{i-1}, w_i, t_{i+1})$$
$$= p(t_i \mid t_{i-1}) \cdot p(w_i \mid t_i) \cdot p(t_{i+1} \mid t_i)$$

Tagging a sentence

- We want to make global decisions about tags.
- Due to independence, global decisions are made in terms of local decisions
- Tagging "forward" through the sentence allows us to predict tags based on previous decisions.
- But we can't calculate picking a tag changes the probabilities of previous tags.

The Viterbi Algorithm

- We need to check all possible tag combinations *efficiently*: dynamic programming
- We do this by making two passes through the sentence: one forward through the sentence and one backward
- The forward pass
  - Compute, for each potential tag value k of t_i in the sentence, the highest probability sequence of tags for the previous i-1 words and ending with $t_i = k$, given *all possible ways of tagging the i-1 previous words*. This can be simplified with our independence assumptions.

$$\begin{aligned} v_i(k) &= \max_{t_1,...,t_{i-1}} p(w_1, ..., w_i, t_1, ..., t_{i-1}, t_i = k) \\ &= p(w_i \mid k) \cdot \max_{k' \in T} v_{i-1}(k') \cdot p(k \mid k') \end{aligned}$$

  - $v_i(k)$ is the probability of the most likely subsequence of tags that accounts for the first i words and ends with tag $t_i = k$.
  - So k is the potential tag for the current token i
  - k' is a potential tag for the previous token i-1.
  - In terms of the **Markov blanket**, this covers the connections to the previous tag and the emitted word
- The backward pass
  - Starting at the end of the sentence with $\hat{t}_N = \langle \text{E} \rangle$, use each chosen "next tag" along with the viterbi scores (probability of assigning a tag k to t_i given the best possible sequence of tags), to figure out the most likely tag k for t_i.
  - So:

$$\hat{t}_i = \text{argmax}_{k \in T} \; v_i(k) \cdot P(\hat{t}_{i+1} \mid t_i = k)$$

  - So the chosen tag is one that has a maximally plausible preceding sequence of tags and best connects to the just-selected "next" tag.
  - In terms of the **Markov blanket**, this adds in the connection to the next tag (since it has now been decided) to the forward probability which is made up of connections to the previous tag and the emitted word

10 Tag Dictionary

- Mapping from words to potential tags
- Get from train, assume it works on test
- Unseen words usually assumed to be any possible tag. Could, instead, assume only open-class tags.
- Prune low-probability tags.

11 Semi-Supervised Learning: The Forward-Backward Algorithm

- Given some unlabeled text, and a set of labels, learn the best set of parameters for an HMM tagger.
- Usually needs a tag dictionary to constrain the search space.
- Similar to Viterbi, but instead of finding the max-probability path, we estimate the probability of each tag for each token.
- forward$_i(k)$: probability of being in state k after seeing the first i words (by summing over all paths leading to k)
- backward$_i(k)$: probability of seeing sequence of tags from $i{+}1$ to N, given that $t_i = k$

$$forward_i(k) = p(w_i \mid k) \cdot \sum_{k' \in T} p(k \mid k') \cdot forward_{i-1}(k')$$

$$backward_i(k) = \sum_{k' \in T} p(k' \mid k) \cdot p(w_{i+1} \mid k') \cdot backward_{i+1}(k')$$

expected transitions from $t_i = k_1$ to $t_{i+1} = k_2$:

$$\frac{forward_i(k_1) \cdot p(k_2 \mid k_1) \cdot p(w_{i+1} \mid k_2) \cdot backward_{i+1}(k_2)}{p(w_{1:N})}$$

expected emissions of w_i when $t_i = k$

$$\frac{forward_i(k) \cdot backward_i(k)}{p(w_{1:N})}$$

Re-estimating transitions:

$$p(t_2 \mid t_1) = \frac{\text{expected number of transitions from } t_1 \text{ to } t_2}{\text{expected number of transitions from } t_1}$$

Re-estimating emissions:

$$p(w \mid t) = \frac{\text{expected number of times word } w \text{ occurs with tag } t}{\text{expected number of times tag } t \text{ occurs}}$$

The Forward-Backward Algorithm (Expectation-Maximization):

- E Step: Use $forward$ and $backward$ to compute the expected transition and emission counts
- M Step: Re-estimate transition and emission distributions from expected counts

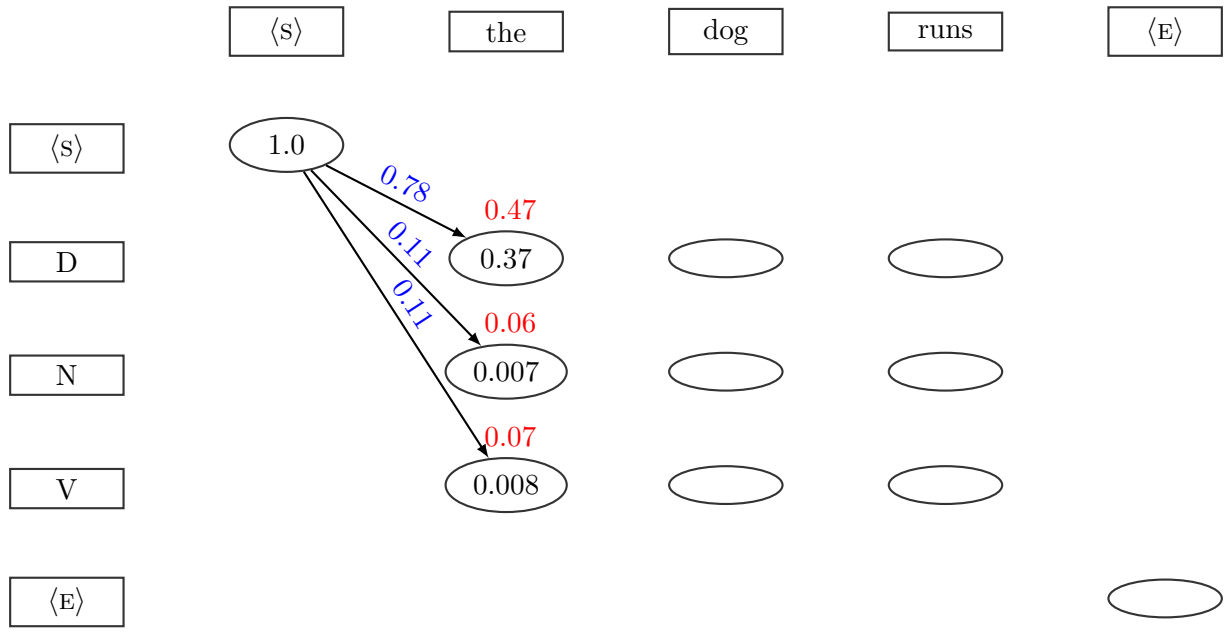

Figure 4: Viterbi Algorithm: Calculating $v_1(D)$, $v_1(N)$, and $v_1(V)$. Since $\langle s \rangle$ is only choice for the previous tag, each score $v_1(k)$ is calculated as $p(\text{the} \mid k) \cdot v_0(\langle s \rangle) \cdot p(k \mid \langle s \rangle)$. So, for example,

$$v_1(D) = p(\text{the} \mid D) \cdot v_0(\langle s \rangle) \cdot p(D \mid \langle s \rangle) = 0.47 \cdot 1.0 \cdot 0.78 = 0.37$$

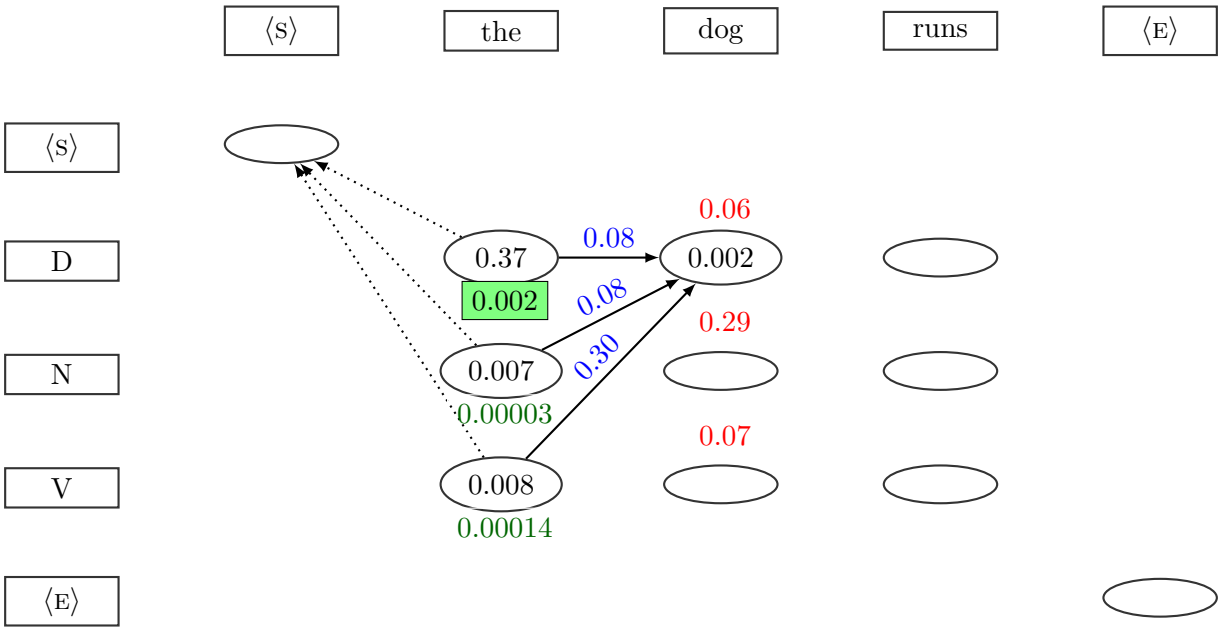

Figure 5: Viterbi Algorithm: Find the maximum probability of all possible tag sequences leading to dog|D at token i. We do so by finding the best previous tag for getting there, given all possible ways of getting to the previous token. Since each viterbi score $v_1(k)$ stores the highest probability path for getting to $t_1 = k$, all we need to do is find the best way of getting to $t_2 = D$ given all the best ways of getting to each possible tag for the previous token, and the probabilities of getting from each possible previous tag to k. For example:

$$\begin{aligned} v_2(D) &= p(\text{dog}|D) \cdot \max(v_1(D) \cdot p(D|D), v_1(N) \cdot p(D|N), v_1(V) \cdot p(D|V)) \\ &= 0.06 \cdot \max(0.37 \cdot 0.08, 0.007 \cdot 0.08, 0.008 \cdot 0.30) \\ &= 0.06 \cdot \max(0.03, 0.0006, 0.002) \\ &= 0.06 \cdot 0.03 \\ &= 0.002 \end{aligned}$$

$t_1 = D$ turns out to be the best previous tag for getting to $t_2 = D$, so $t_1 = D$ is remembered (with a dotted arrow) as the best way to get to $t_2 = D$.

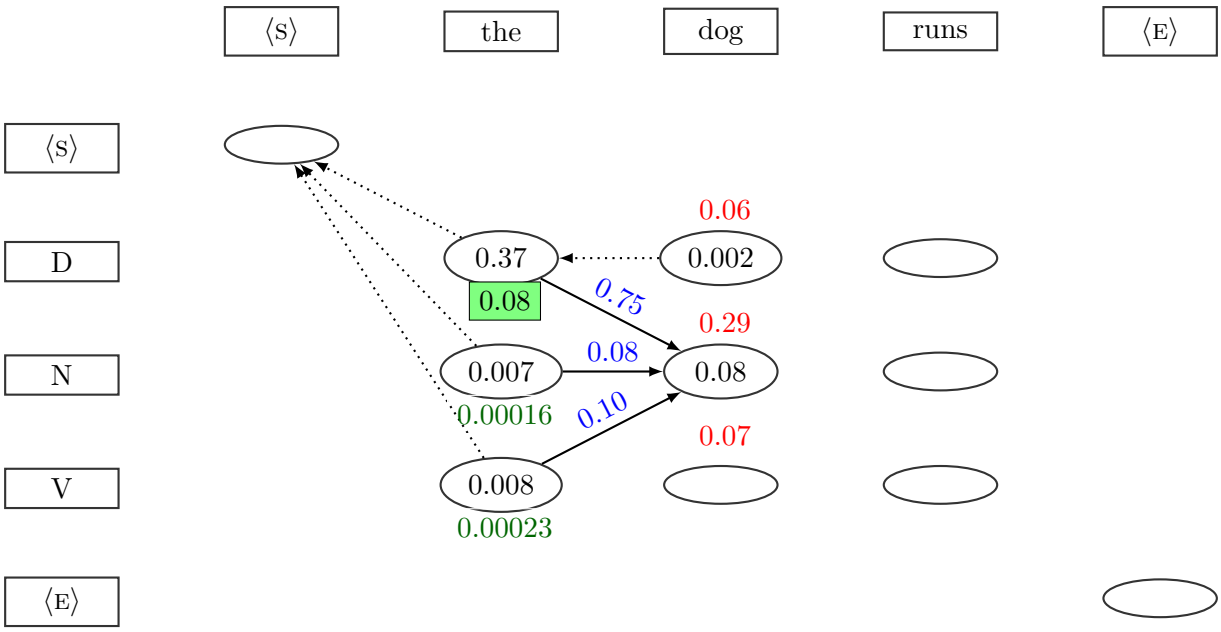

Figure 6: Viterbi Algorithm: Calculate $v_2(N)$ and find the best path to it, which turns out to be the path ending with $t_1 = D$.

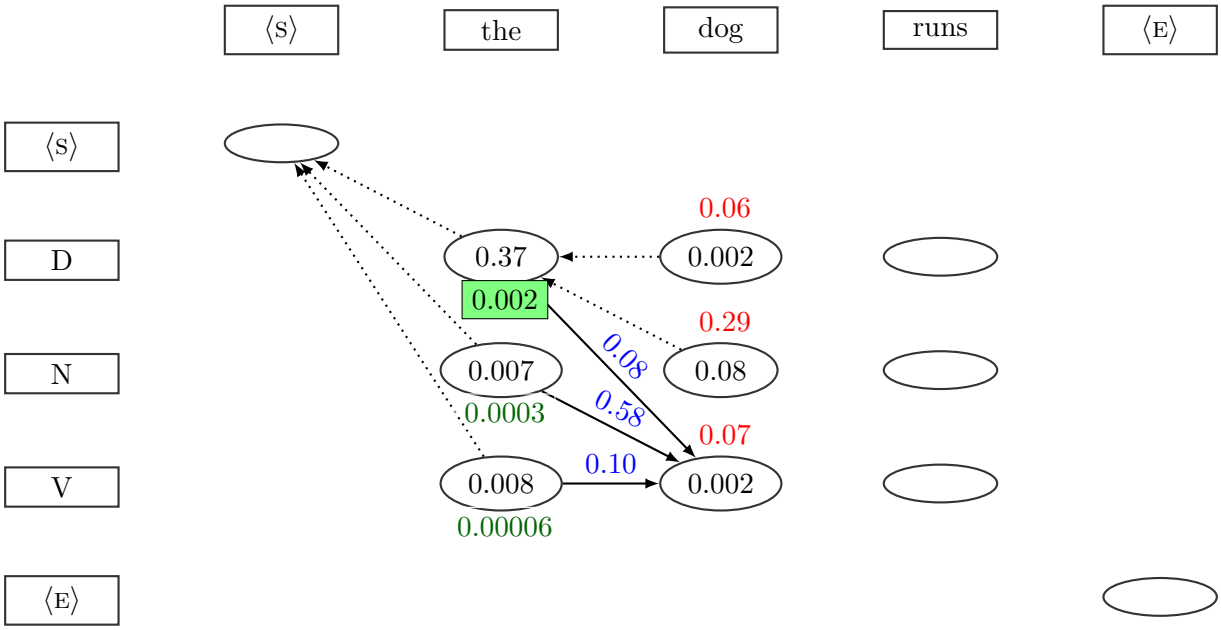

Figure 7: Viterbi Algorithm: Calculate $v_2(V)$ and find the best path to it, which turns out to be the path ending with $t_1 = D$

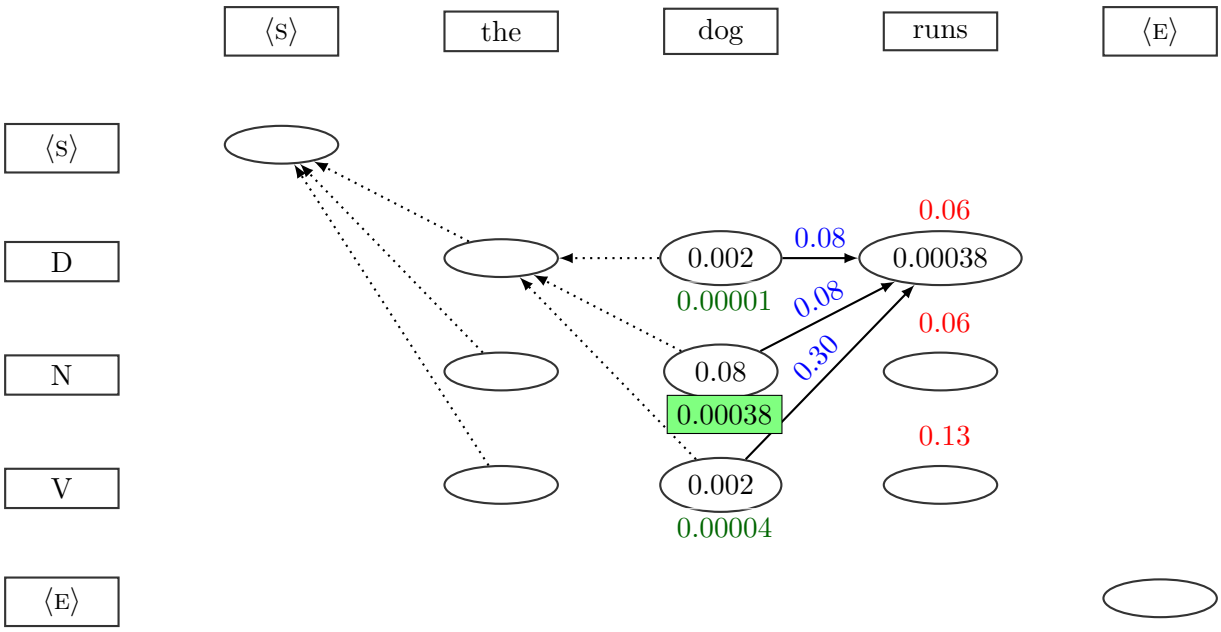

Figure 8: Viterbi Algorithm: Calculate $v_3(D)$ and find the best path to it, which turns out to be the path ending with $t_2 = N$

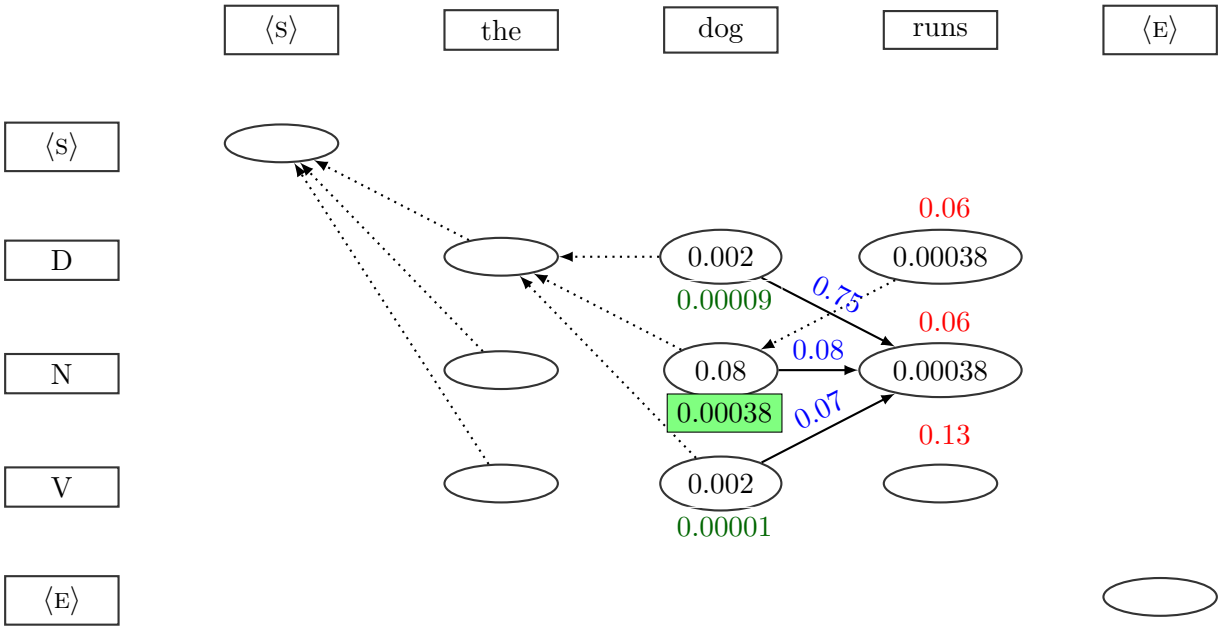

Figure 9: Viterbi Algorithm: Calculate $v_3(N)$ and find the best path to it, which turns out to be the path ending with $t_2 = N$

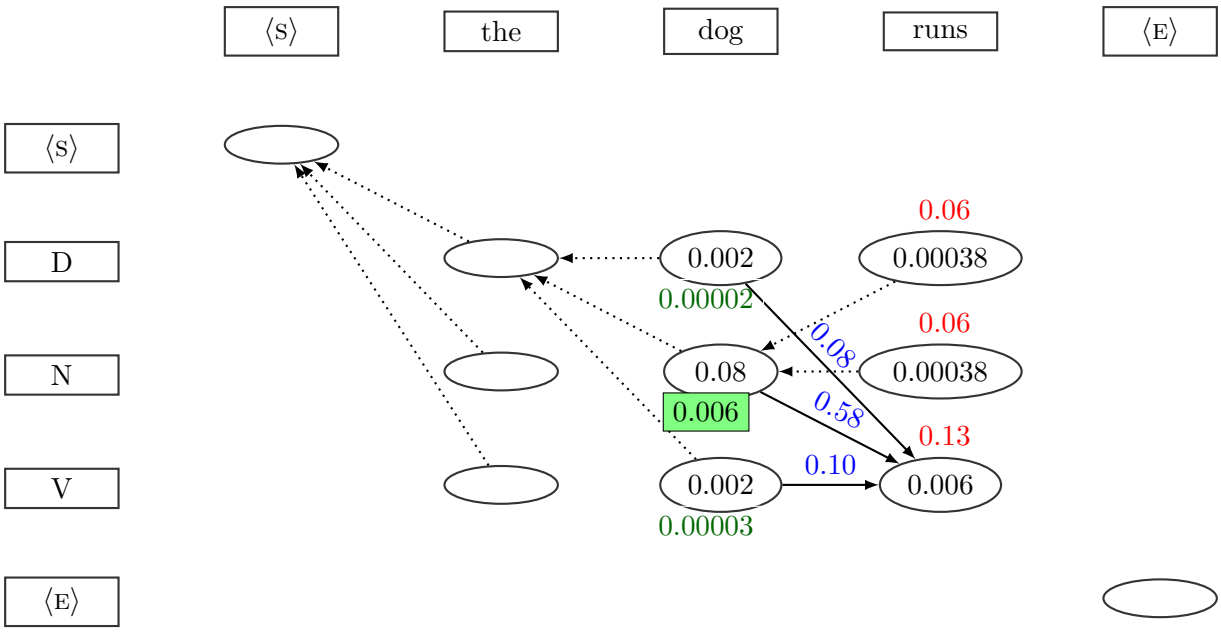

Figure 10: Viterbi Algorithm: Calculate $v_3(V)$ and find the best path to it, which turns out to be the path ending with $t_2 = N$

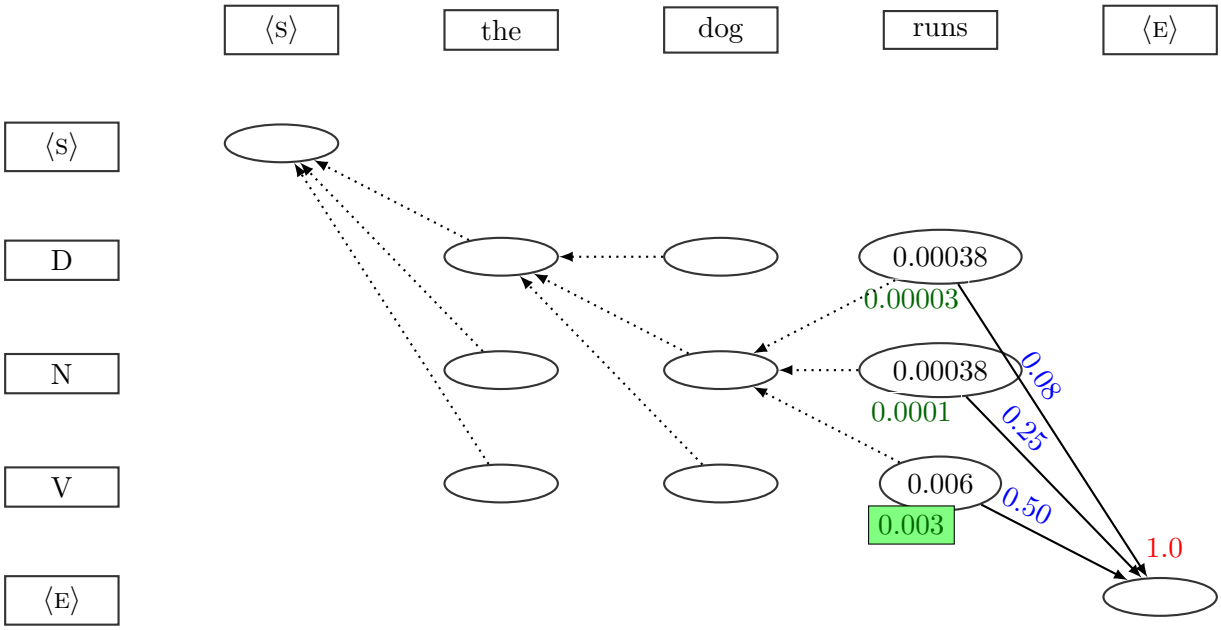

Figure 11: Viterbi Algorithm: Calculate $v_4(\langle E \rangle)$ and find the best path to it, which turns out to be the path ending with $t_3 = V$

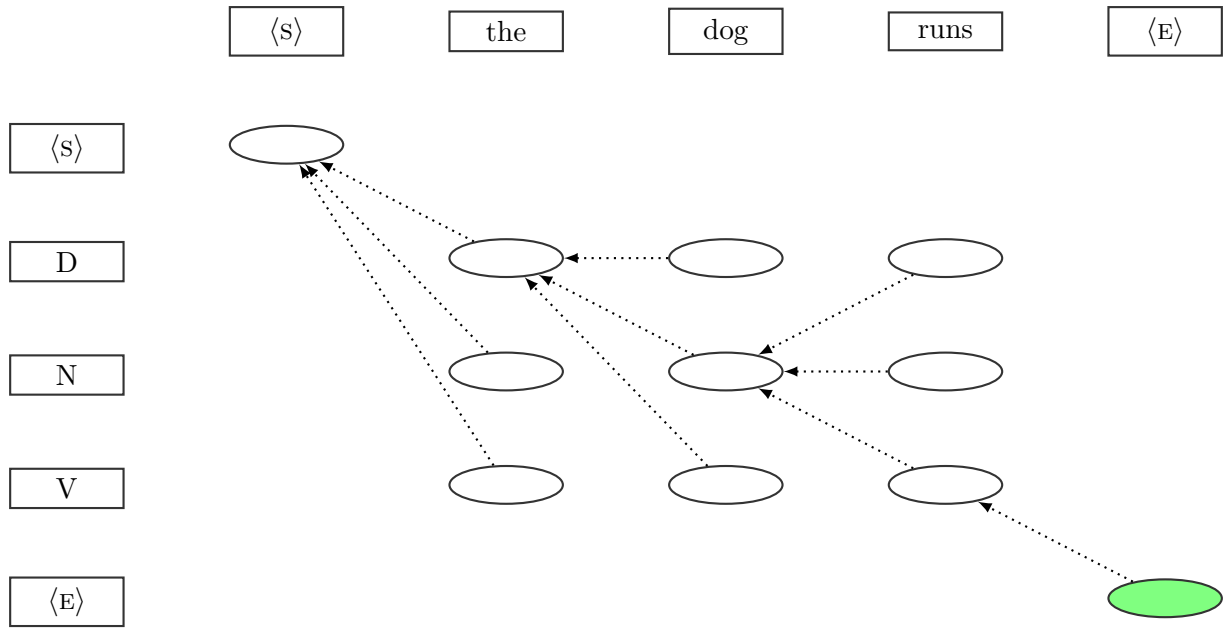

Figure 12: Viterbi Algorithm: The tag for ⟨E⟩ is always ⟨E⟩, so we can always start by selecting that tag.

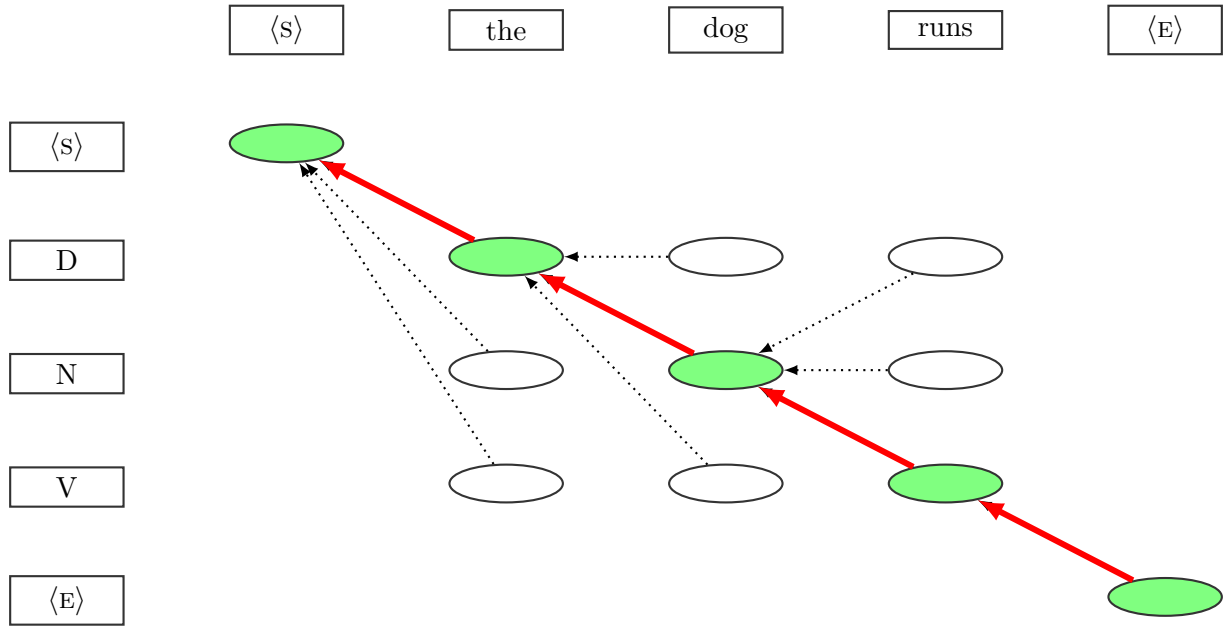

Figure 13: Viterbi Algorithm: For each tag t_{i+1}, we examine the backpointer that indicates the way to the best path to t_{i+1}, and choose t_i accordingly.