Part of Speech Tagging

Probabilistic Models

Task description – more formal

- Given a sentence and its tokens t_i, assign a single label $l \in L$ with L being the tagset (e.g. Penn Treebank, STTS) to every t_i
- This is a **structural problem**, where our input is a sequence (“a list of tokens”) and the output is a sequence (“a list of labels”), and:
  - Both sequences have the same length
    - (This is not the case for OCR or speech recognition)

WORD	tag
the	**DET**
koala	**N**
put	**V**
the	**DET**
keys	**N**
on	**P**
the	**DET**
table	**N**

Probabilistic POS-Tagging – Probabilistic View

Let us formalize the task:

- We are given a sentence (an “observation” or “sequence of observations”)
  - Secretariat is expected to race tomorrow
- What is the best sequence of tags that corresponds to this sequence of observations?
- Probabilistic view:
  - Consider all possible sequences of tags
  - Out of this universe of sequences, choose the tag sequence which is most probable given the observation sequence of n words $w_1 \ldots w_n$.

Probabilistic POS-Tagging – Probabilistic View

- We want, out of all sequences of n tags $t_1 \ldots t_n$ the single tag sequence such that $P(t_1 \ldots t_n | w_1 \ldots w_n)$ is highest.

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

- hat ^ means “our estimate of the best one”
- Argmax_x f(x) means “the x such that f(x) is maximized”

Getting to HMMs

- This equation is guaranteed to give us the best tag sequence

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

- But how to make it operational? How to compute this value?
- Intuition of Bayesian classification:
  - Use Bayes rule to transform this equation into a set of other probabilities that are easier to compute

Using Bayes Rule

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

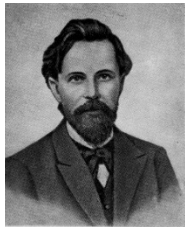
$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n|t_1^n)P(t_1^n)$$

Likelihood and Prior

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \overbrace{P(w_1^n|t_1^n)}^{\text{likelihood}} \overbrace{P(t_1^n)}^{\text{prior}}$$

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$



$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1})$$

Two Kinds of Probabilities

- Tag transition probabilities $p(t_i|t_{i-1})$
  - Determiners likely to precede adjectives and nouns
    - That/DT flight/NN
    - The/DT yellow/JJ hat/NN
  - Compute $P(NN|DT)$ by counting in a labelled corpus:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

Two Kinds of Probabilities

- Word likelihood probabilities $p(w_i|ti)$
  - VBZ (3sg Pres verb) likely to be “is”
  - Compute $P(is|VBZ)$ by counting in a labelled corpus:

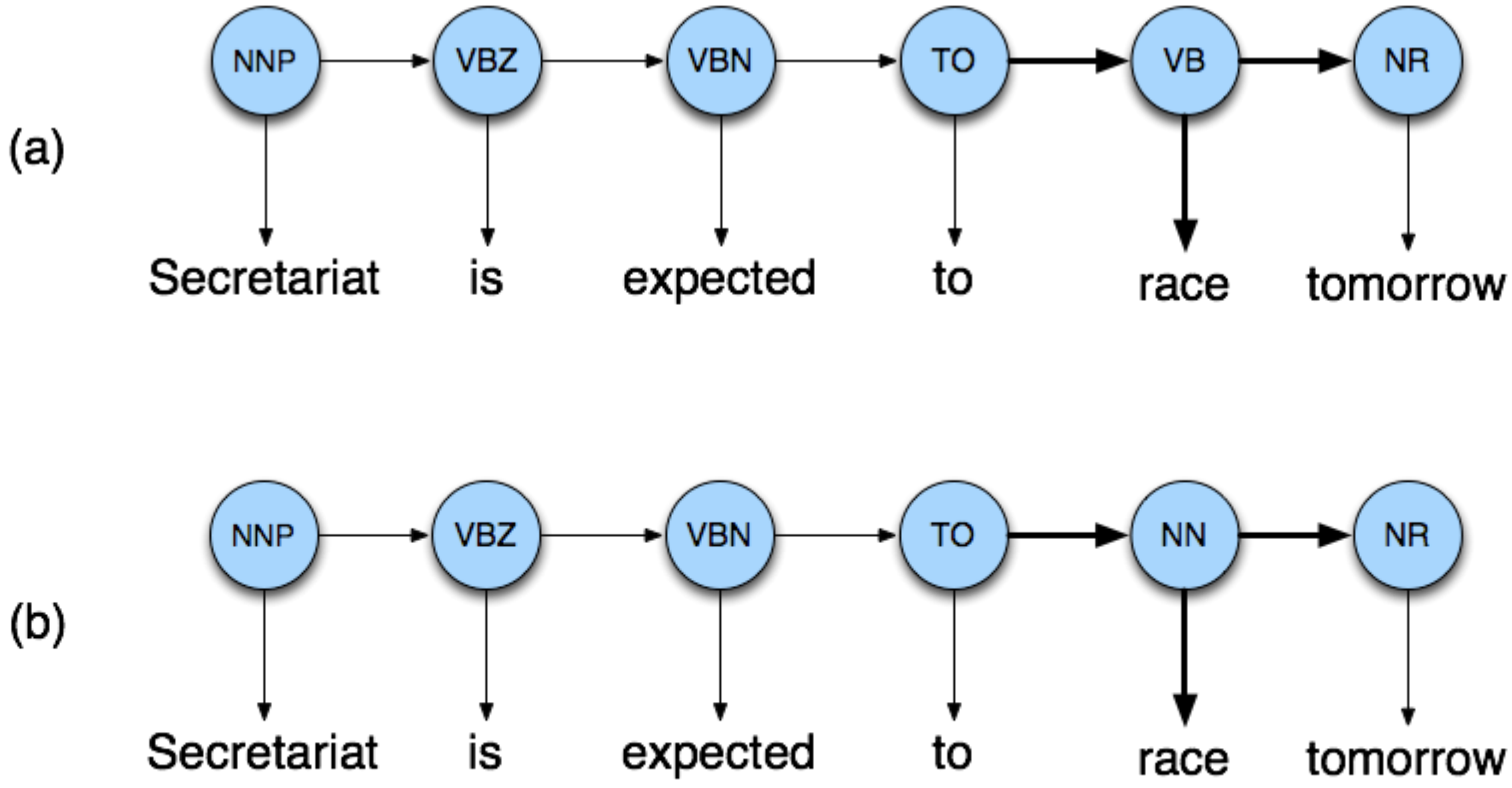
$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47$$

Example: The Verb "race"

- Secretariat/NNP is/VBZ expected/VBN to/TO **race**/VB tomorrow/NR
- People/NNS continue/VB to/TO inquire/VB the/DT reason/NN for/IN the/DT **race**/NN for/IN outer/JJ space/NN
- How do we pick the right tag?

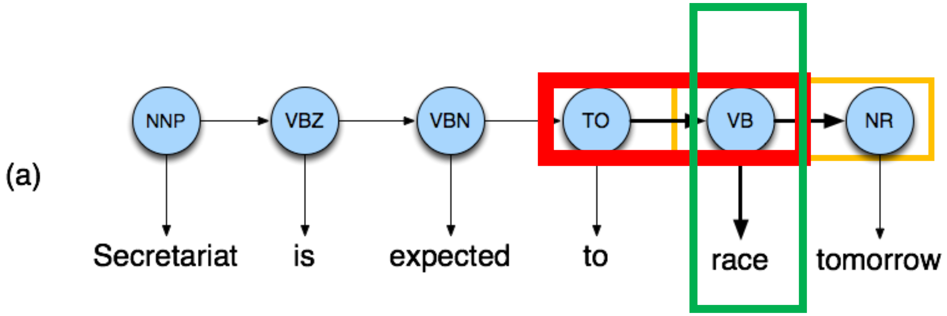
Disambiguating “race”



Example

- P(VB|TO) = .83
- P(race|VB) = .00012
- P(NR|VB) = .0027
- P(NN|TO) = .00047
- P(race|NN) = .00057
- P(NR|NN) = .0012
- $P(VB|TO) \cdot P(NR|VB) \cdot P(race|VB) = 0.00000027$
- $P(NN|TO) \cdot P(NR|NN) \cdot P(race|NN) = 0.00000000032$

➔So we (correctly) choose the verb reading



Hidden Markov Models

- What we’ve described with these two kinds of probabilities is a Hidden Markov Model (HMM)

Hidden Markov Models

- States Q = $q_1, q_2 \ldots q_N$ (our POS-Tags)
- Observations O= $o_1, o_2 \ldots o_N$ (the incoming words)
  - Each observation is a symbol from a vocabulary

$$V = \{v_1, v_2, \ldots v_V\}$$

- Transition probabilities
  - Transition probability matrix $A = \{a_{ij}\}$

$$a_{ij} = P(q_t = j \mid q_{t-1} = i) \quad 1 \le i, j \le N$$

- Observation likelihoods
  - Output probability matrix $B = \{b_i(k)\}$

$$b_i(k) = P(X_t = o_k \mid q_t = i)$$

- Special initial probability vector π

$$\pi_i = P(q_1 = i) \quad 1 \le i \le N$$

Hidden Markov Models

- States Q = q_1, q_2…q_N (our POS-Tags) — Usually defined (e.g. STTS)
- Observations O= o_1, o_2…o_N (the incoming words)
  - Each observation is a symbol from a vocabulary

$$V = \{v_1, v_2, \ldots v_V\}$$

Measured in a labelled corpus, all word „types“

- Transition probabilities
  - Transition probability matrix $A = \{a_{ij}\}$ — Measured in a labelled corpus

$$a_{ij} = P(q_t = j \mid q_{t-1} = i) \quad 1 \le i, j \le N$$

- Observation likelihoods
  - Output probability matrix $B = \{b_i(k)\}$ — Measured in a labelled corpus

$$b_i(k) = P(X_t = o_k \mid q_t = i)$$

- Special initial probability vector π — Measured in a labelled corpus or intuition

$$\pi_i = P(q_1 = i) \quad 1 \le i \le N$$

Decoding

- Ok, now we have a complete model that can give us what we need. Recall that we need to get

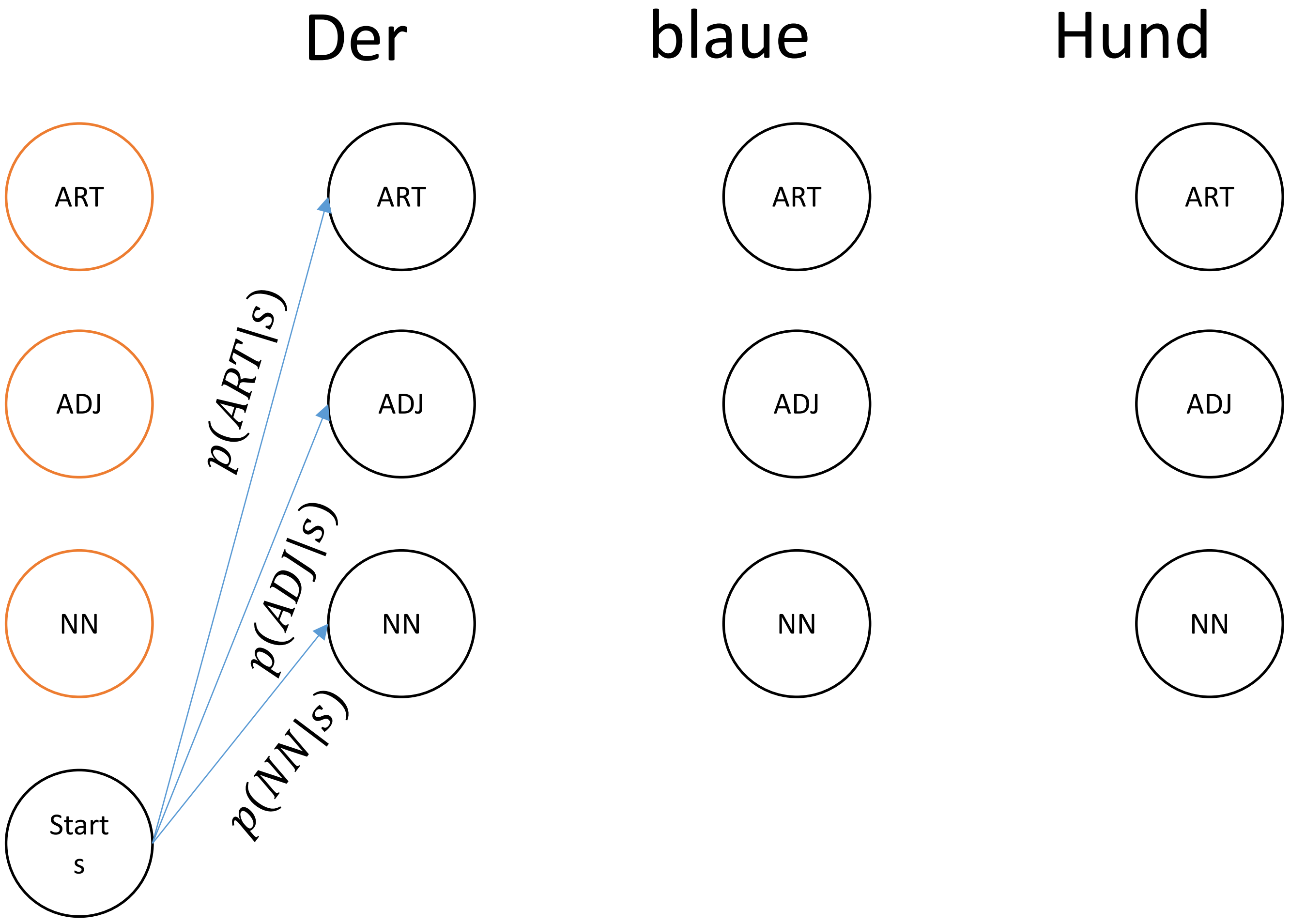
$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

- We could just enumerate all paths given the input and use the model to assign probabilities to each.
  - Not a good idea.
  - Luckily dynamic programming helps us here

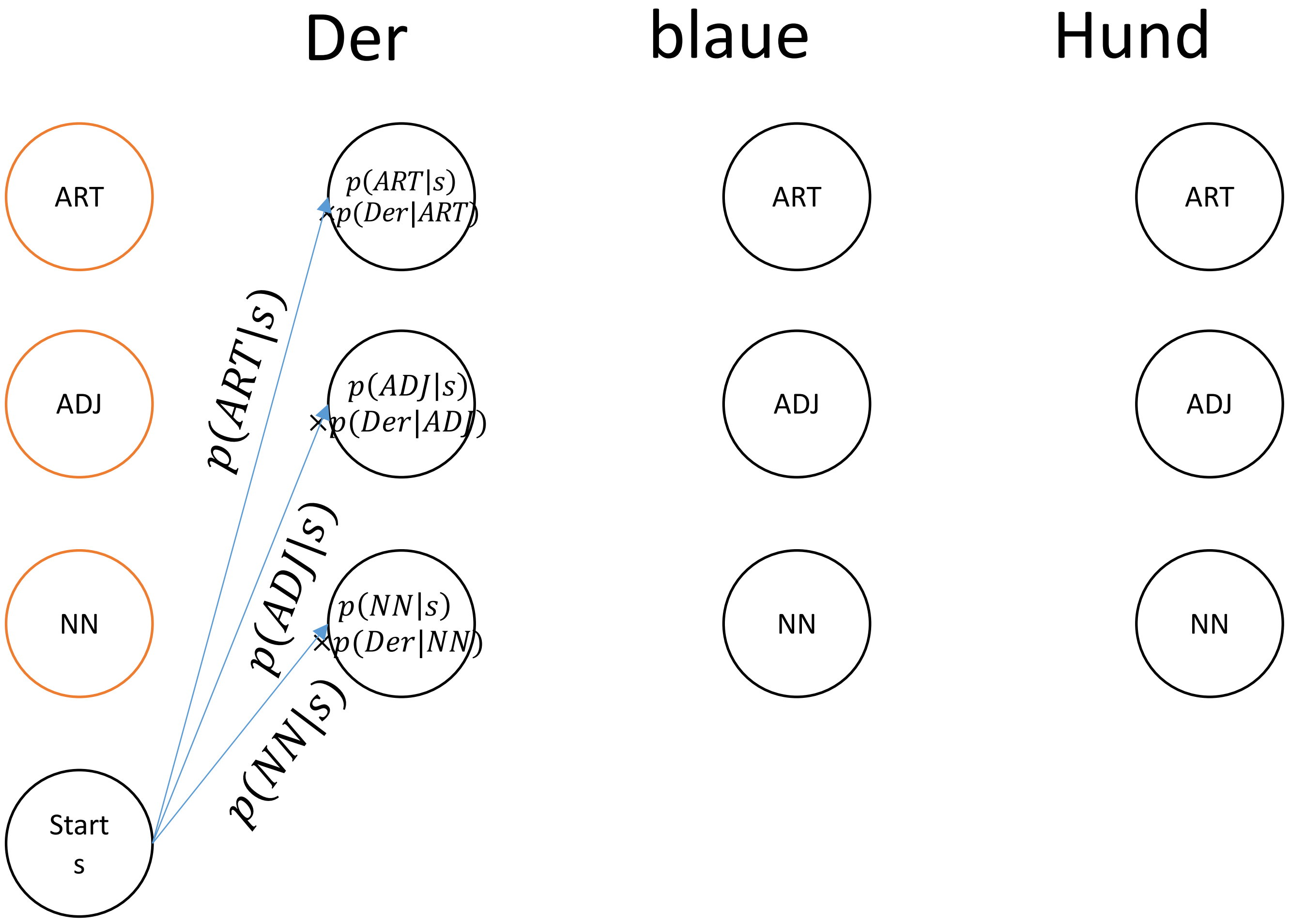
The Viterbi Algorithm

- I am now starting with an example of the algorithm, to get you all familiarized with the algorithm
- We will then proceed to generalize the algorithm

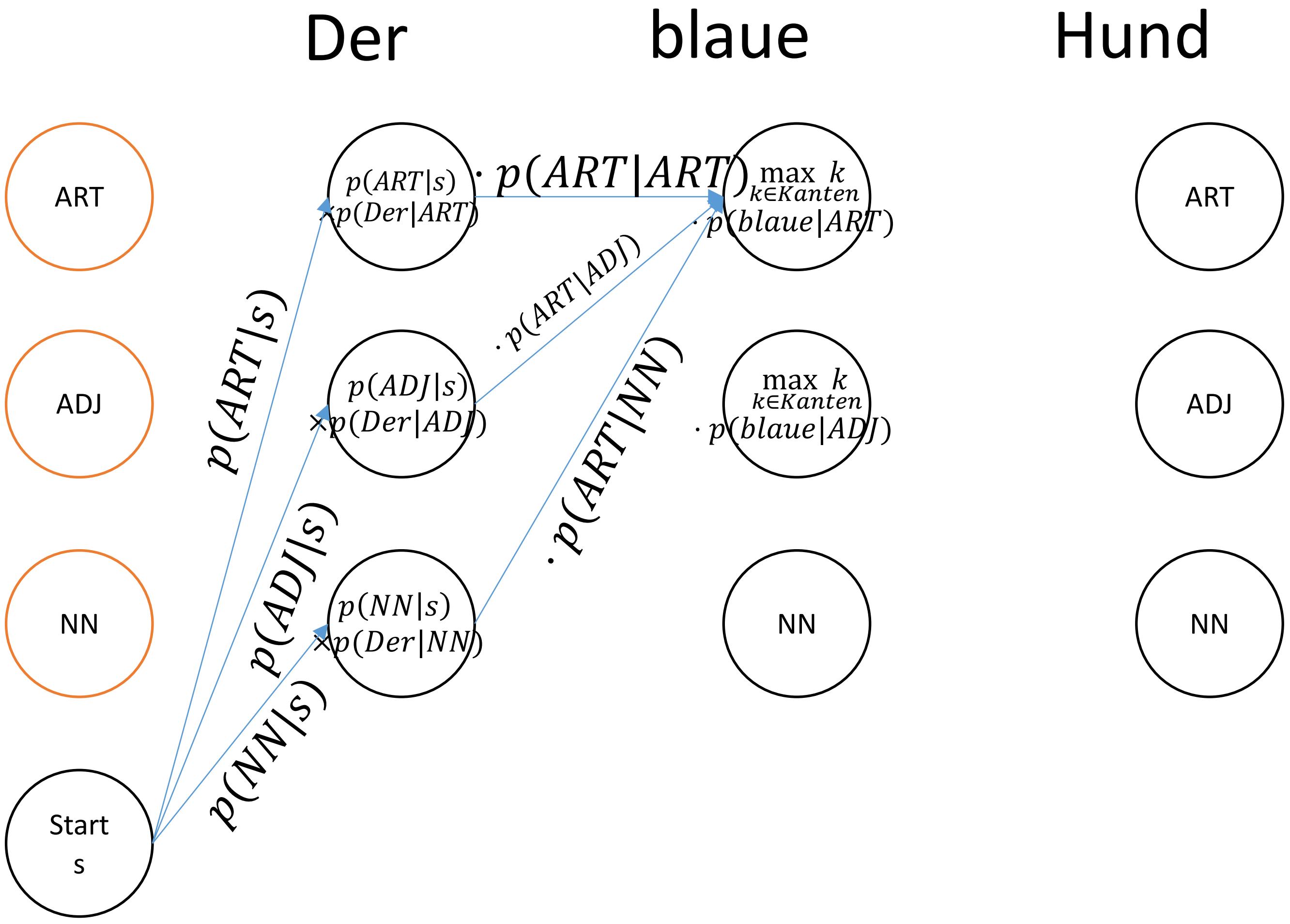
Viterbi Example-Trellis



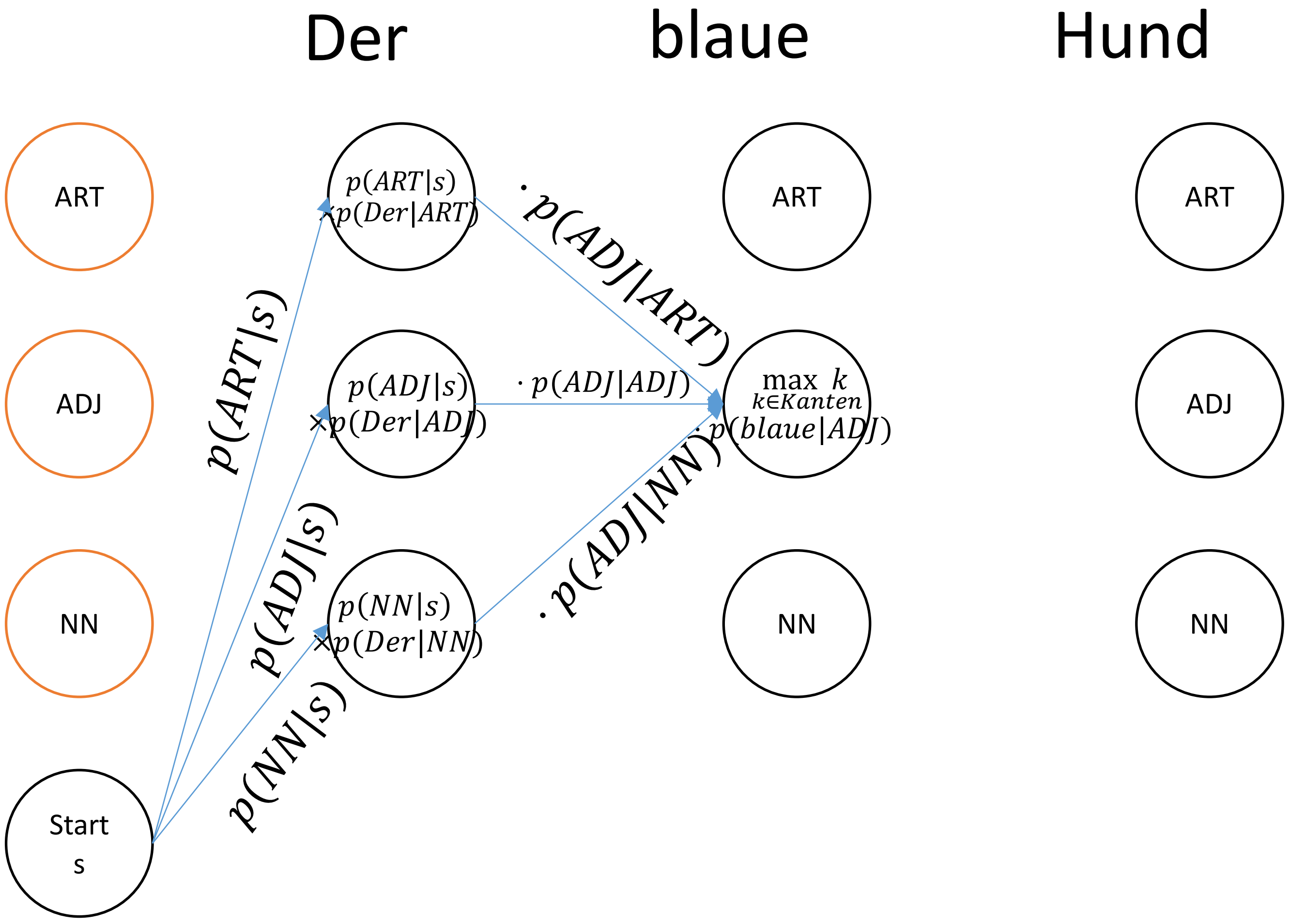
Viterbi Example-Trellis



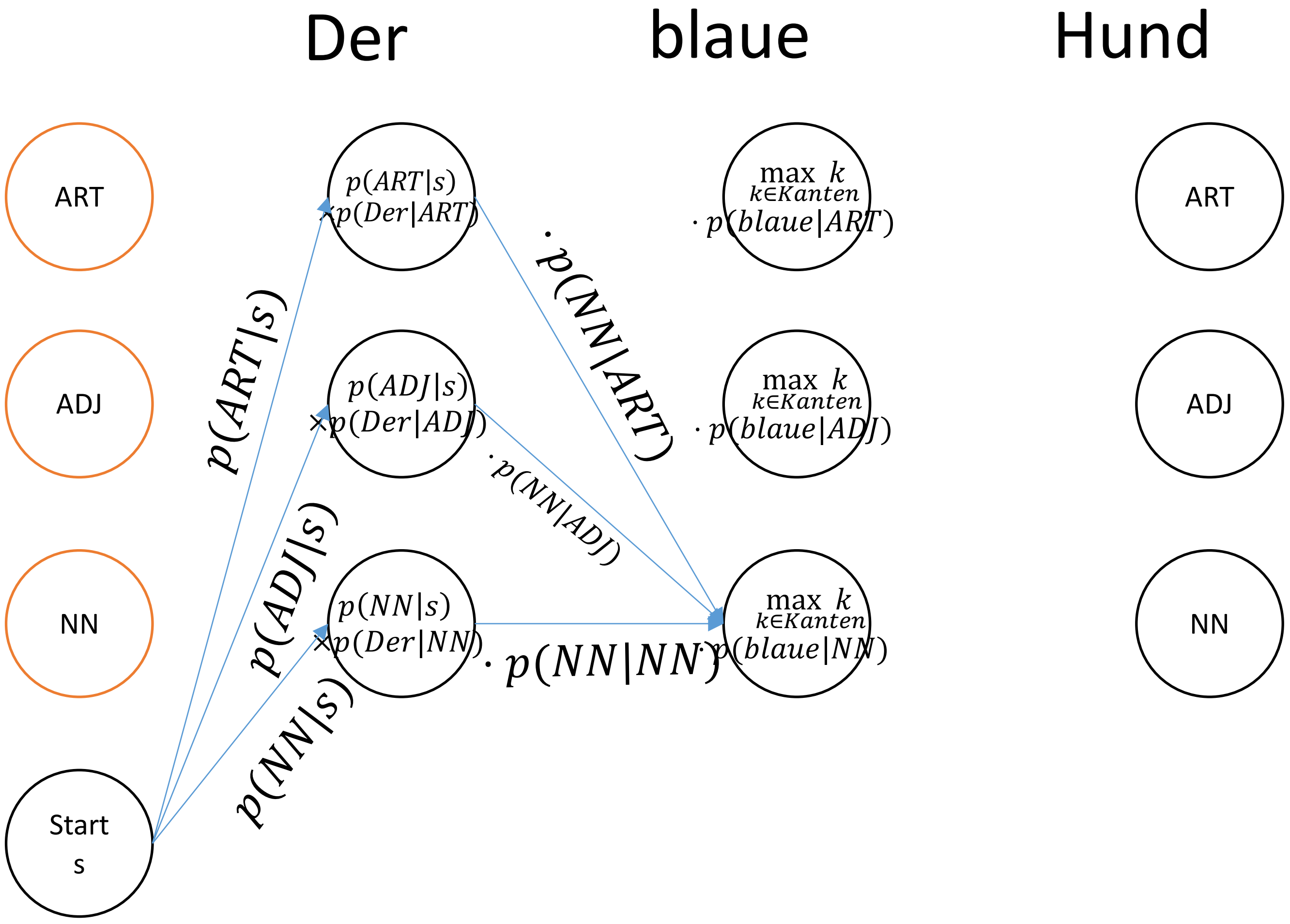
Viterbi Example-Trellis



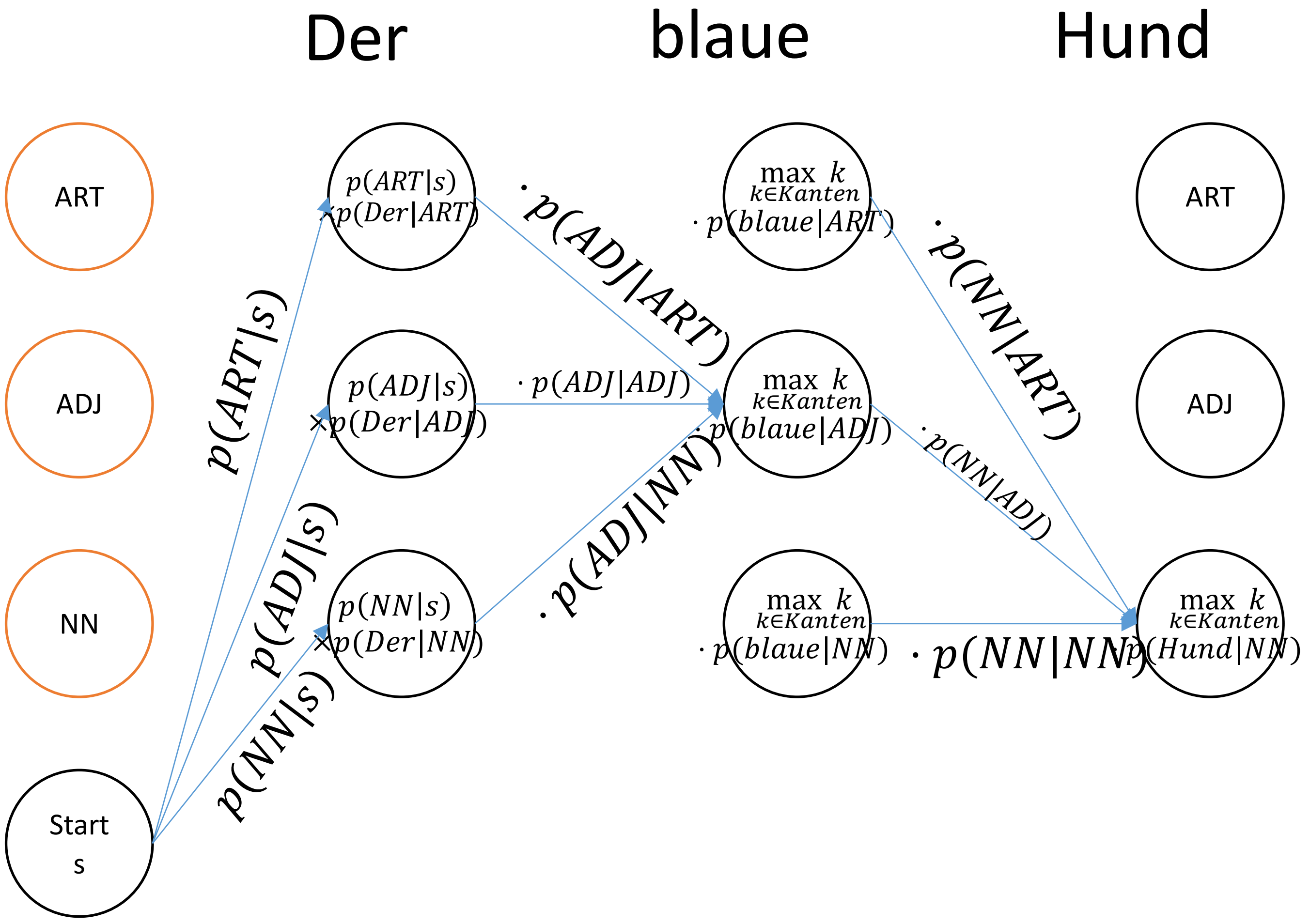
Viterbi Example-Trellis



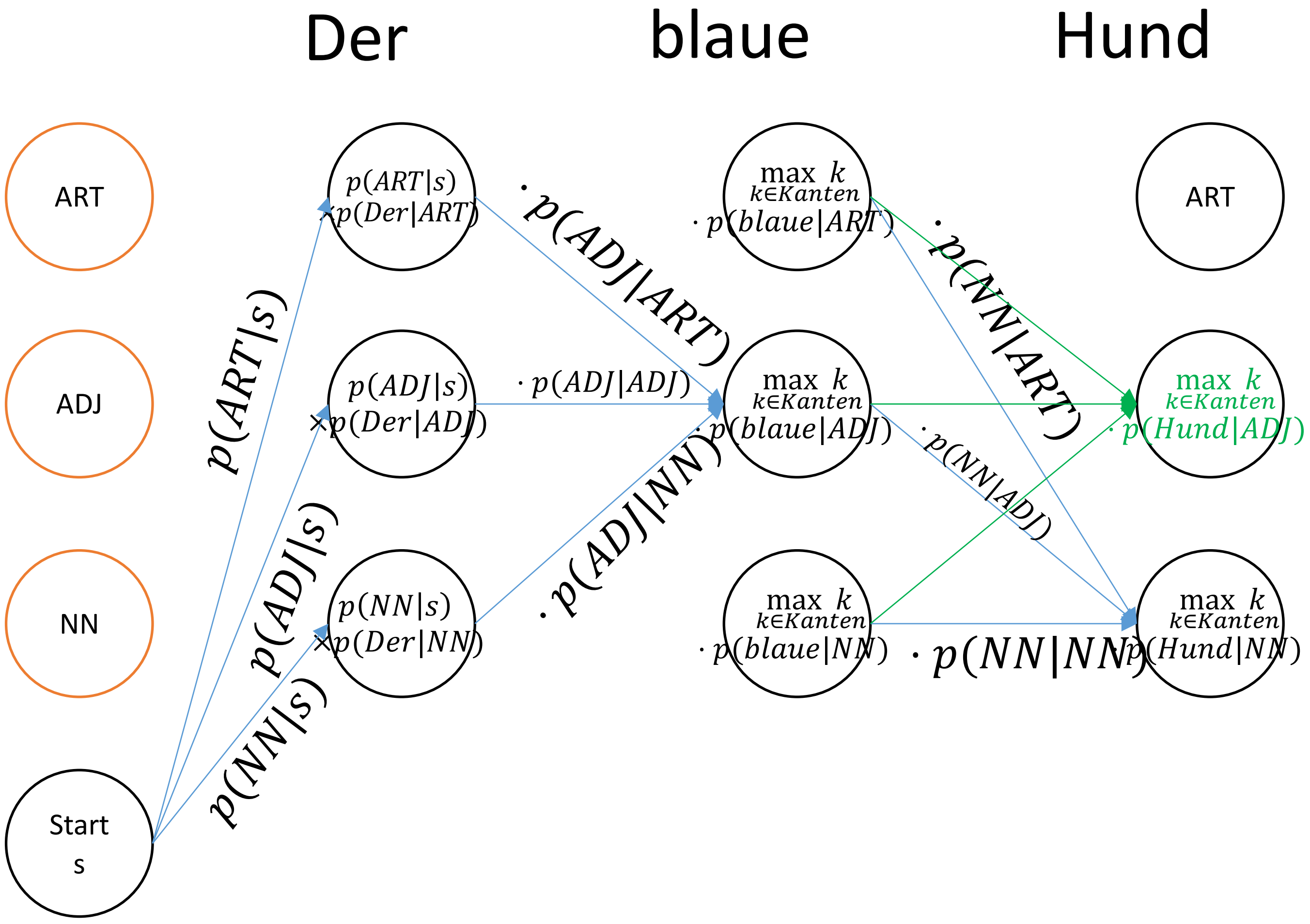
Viterbi Example-Trellis



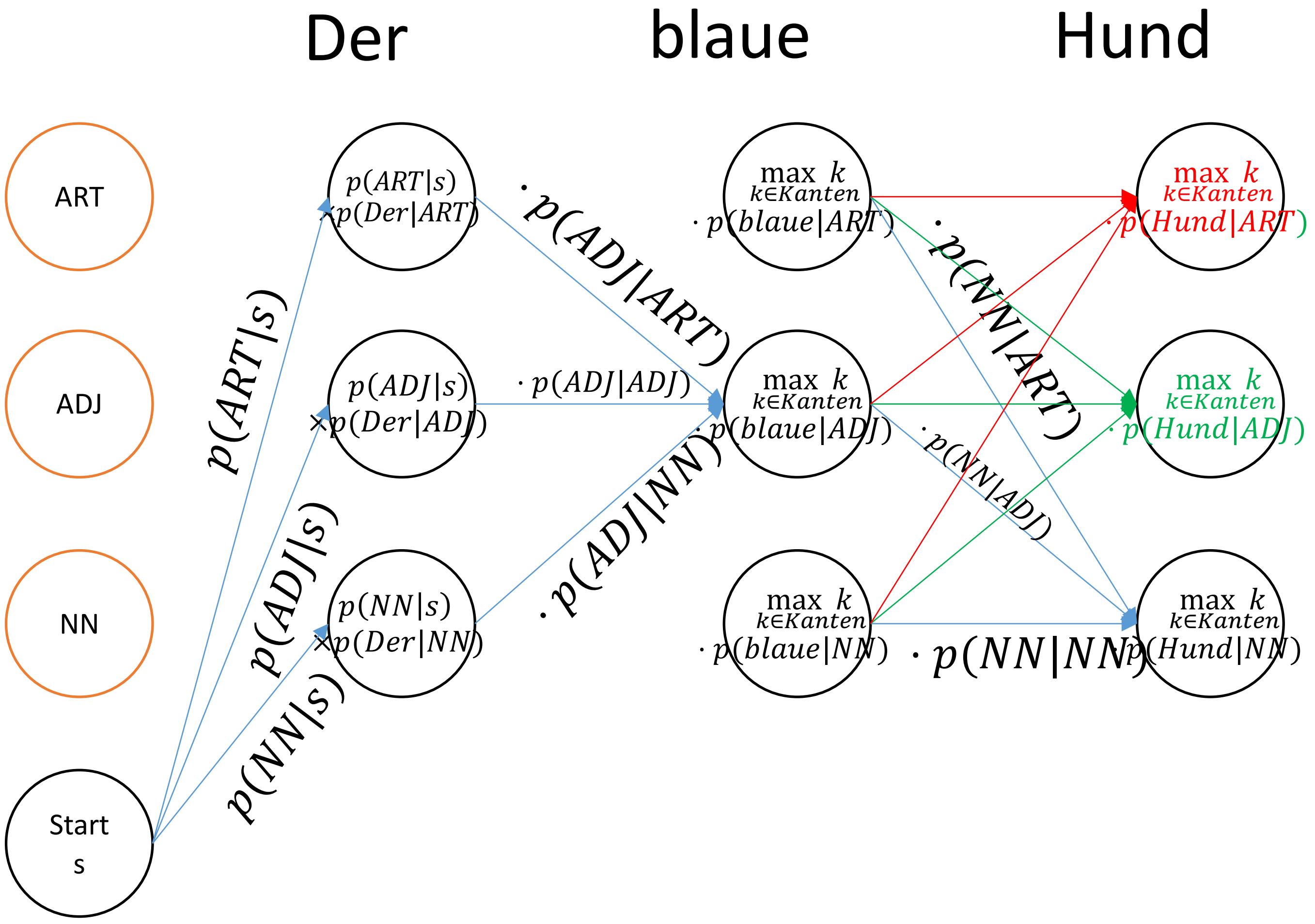
Viterbi Example-Trellis



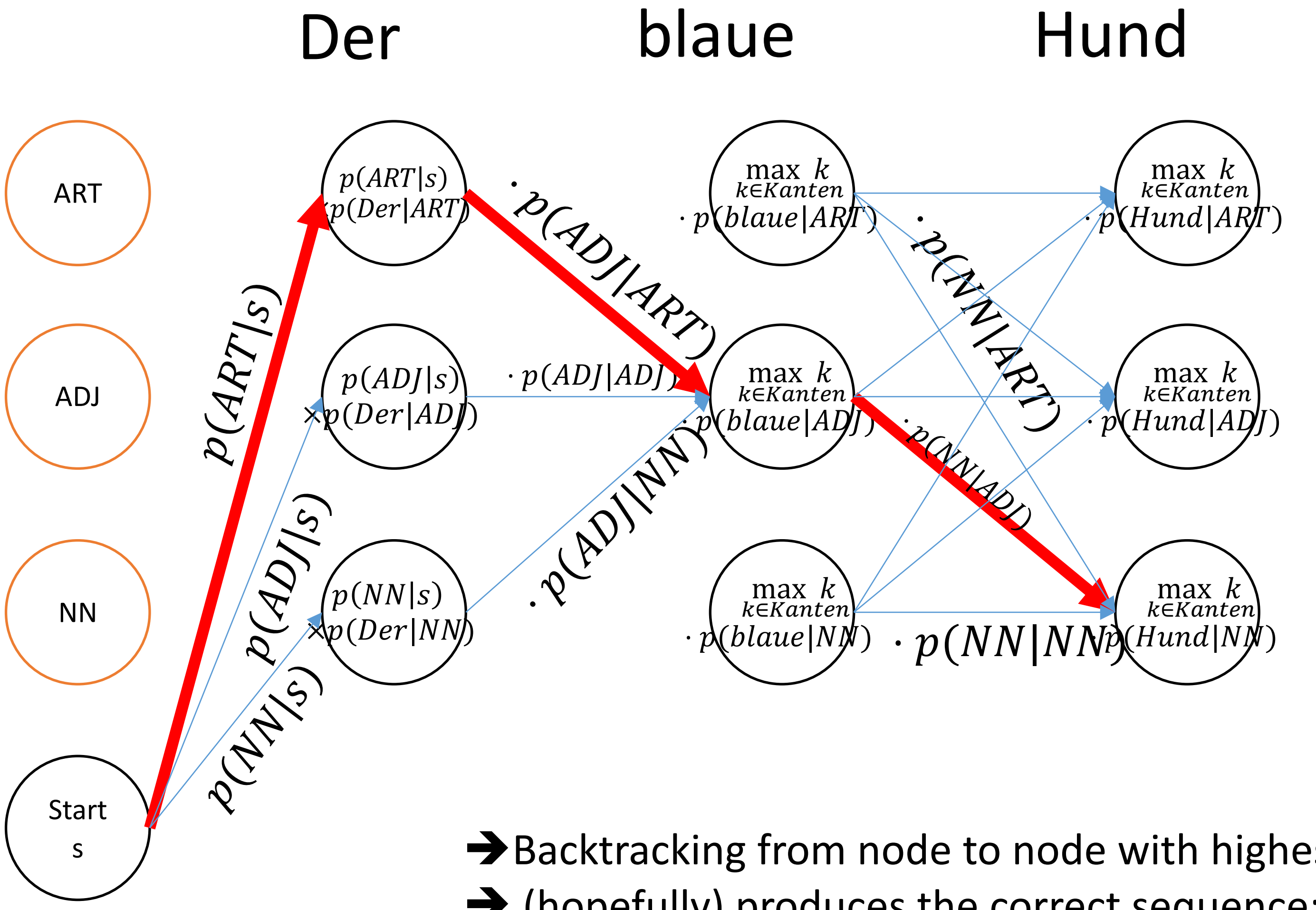
Viterbi Example-Trellis



Viterbi Example-Trellis



Viterbi Example-Trellis: Maximum



➔ Backtracking from node to node with highest probability.

➔ (hopefully) produces the correct sequence: ART→ADJ→NN

Viterbi

- We can now proceed to generalize what we have seen:

  1. We **score** each path by taking the most likely **sub-path** to a node and **adding** a score for the latest transition and the latest observation

Edge-score

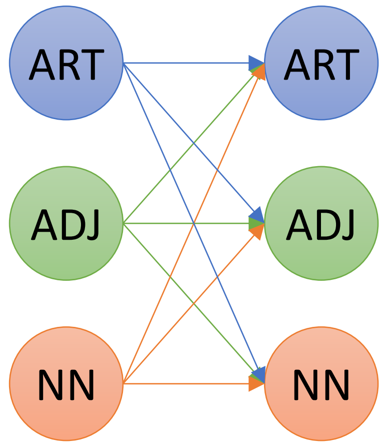
Node-score

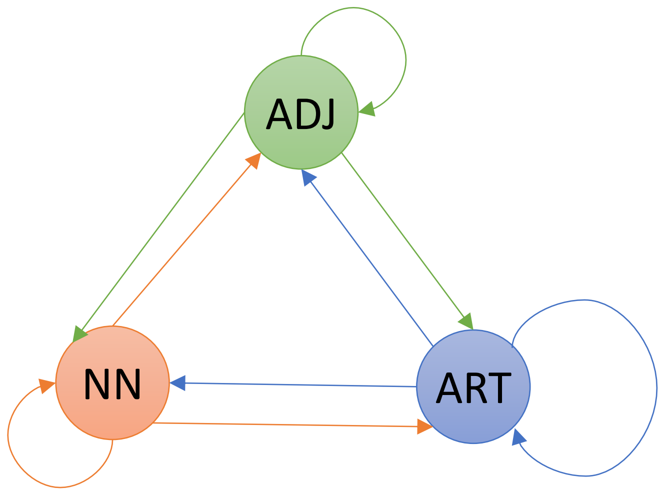
Viterbi

- We can now proceed to generalize what we have seen:

1. We **score** each path by taking the most likely **sub-path** to a node and **adding** a score for the latest transition and the latest observation

2. The states we can reach in between steps could be modelled with a state-machine (which we will call a **Transducer**)

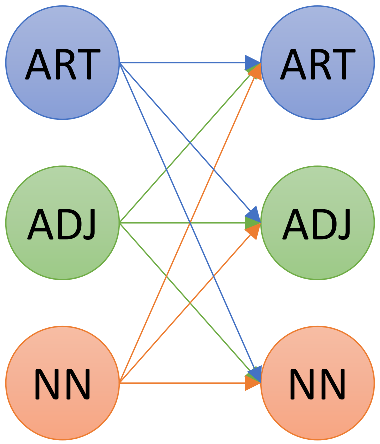


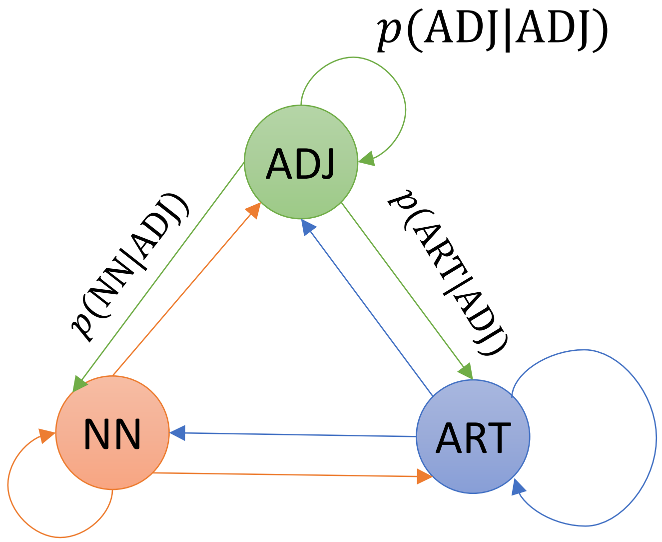


Viterbi

- We can now proceed to generalize what we have seen:

  2. The states we can reach in between steps could be modelled with a state-machine (which we will call a **Transducer**)
     a. And we can store the **edge-scores** on the edges of the Transducer (incompletely drawn)

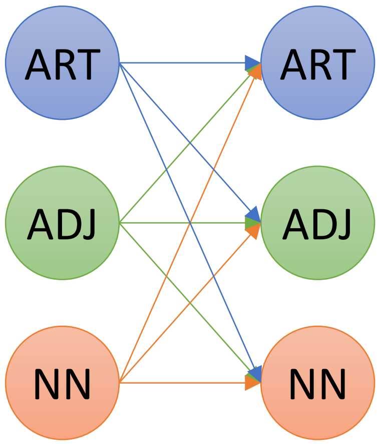


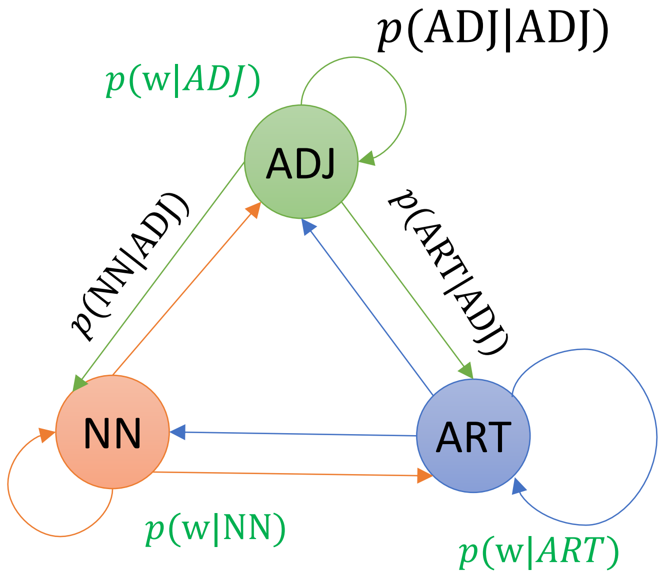


Viterbi

- We can now proceed to generalize what we have seen:

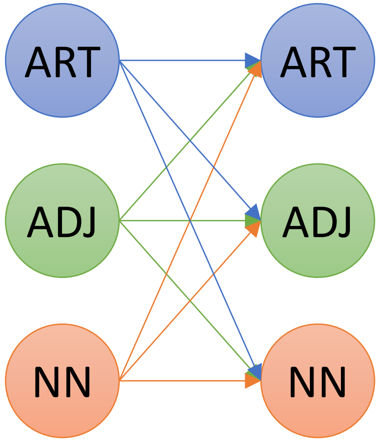
  2. The states we can reach in between steps could be modelled with a state-machine (which we will call a **Transducer**)
     a. And we can store the **edge-scores** on the edges of the Transducer (incompletely drawn)
     b. And the **node-scores** on the nodes of the Transducer

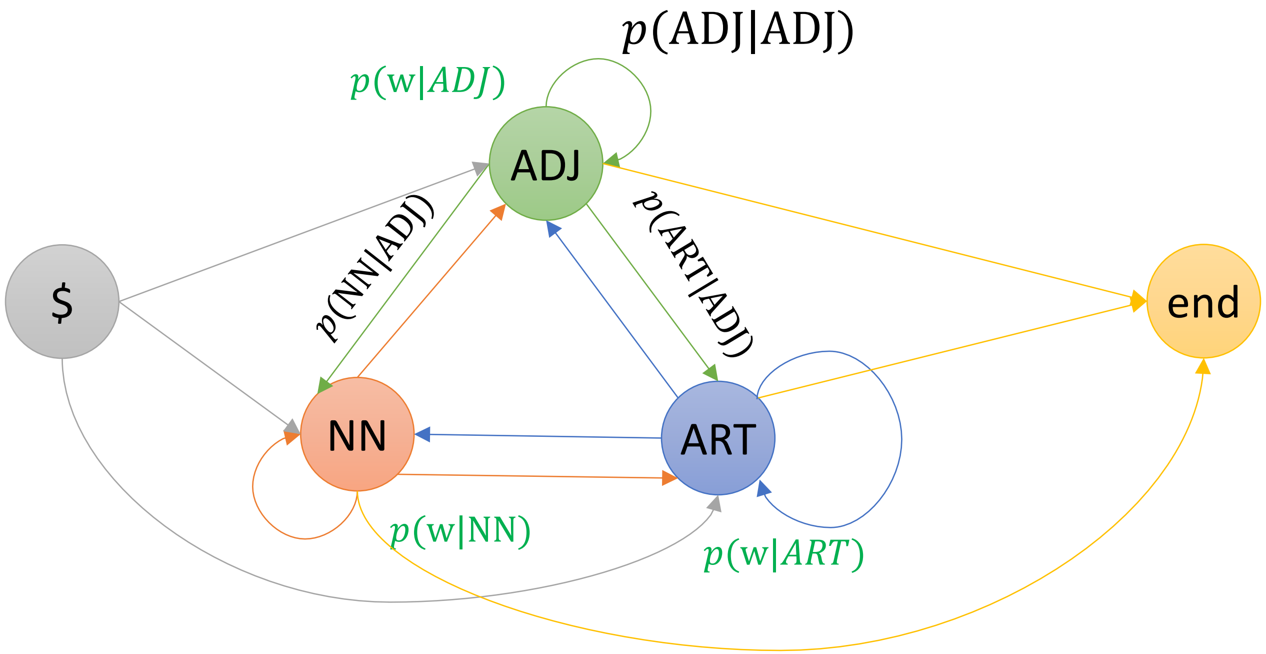




Viterbi

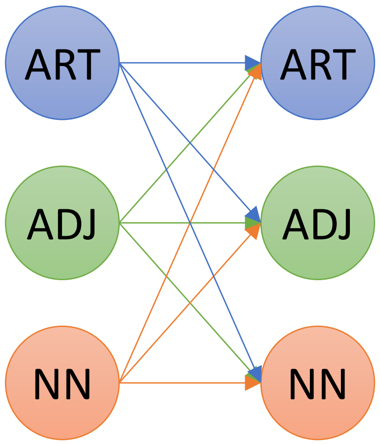
2. The states we can reach in between steps could be modelled with a state-machine (which we will call a **Transducer**)
   a. And we can store the **edge-scores** on the edges of the Transducer (incompletely drawn)
   b. And the **node-scores** on the nodes of the Transducer
3. Add special symbols for the start $< \$ >$ and the end $< \text{end} >$ of a sequence

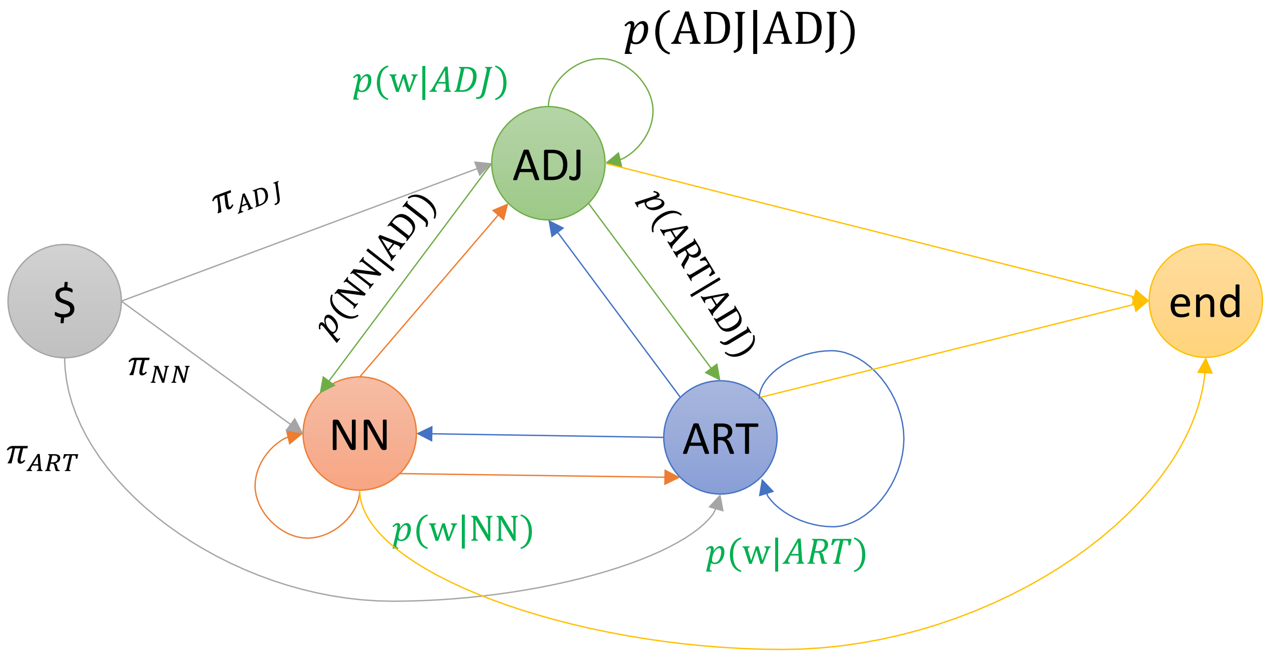




Viterbi

3. Add special symbols for the start $< \$ >$ and the end $< \text{end} >$ of a sequence
   a. On which we can store initial scores π_i

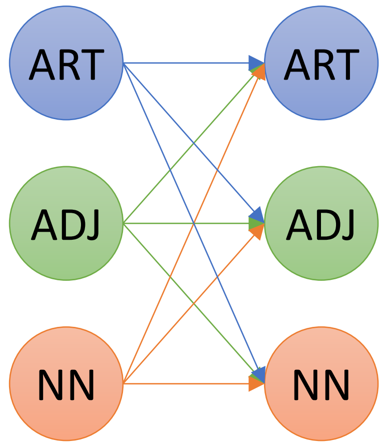


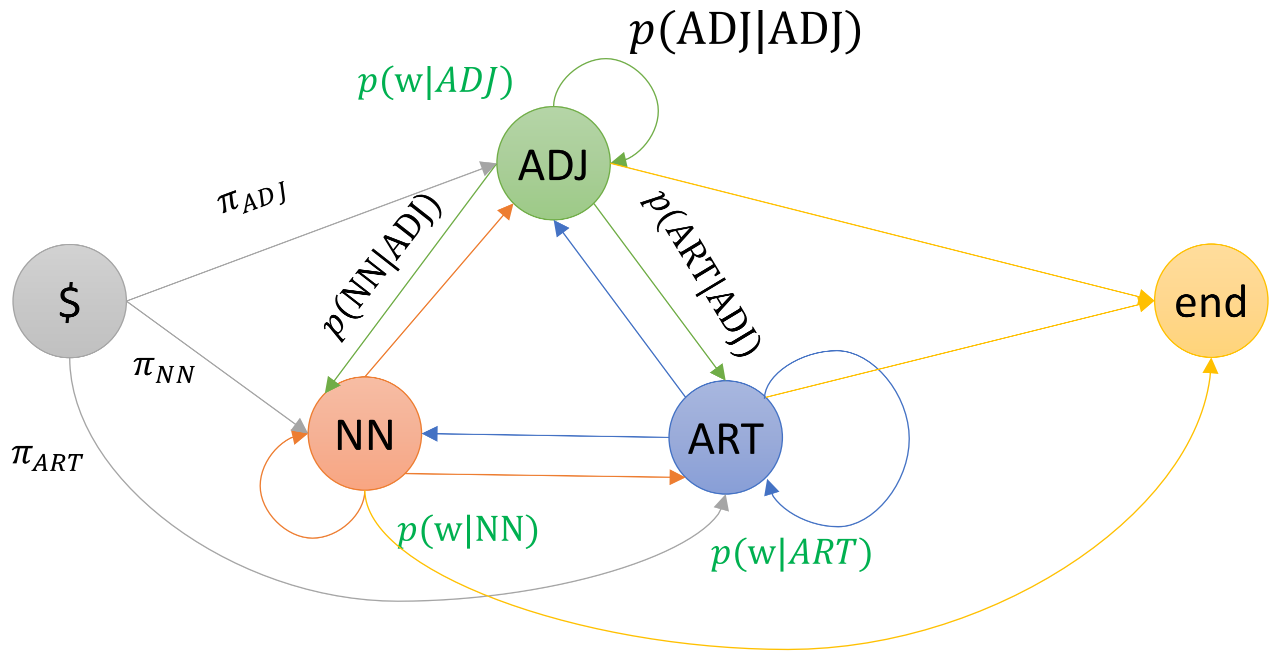


Viterbi

3. Add special symbols for the start $< \$ >$ and the end $< \text{end} >$ of a sequence
   a. On which we can store initial scores π_i

➔ We can now execute the entire Viterbi-algorithm on this data structure





Viterbi-Transducer Version

- As input, we get our observation sequence O and a Transducer T

Function **viterbiDecode(O,T)**

viterbiChart ← ViterbiChart(0,T) // init an empty chart

For each timestep t in O.length:

  validStates ← viterbiChart.validStates($t-1$) // all states that were reachable in the last time step

  reachableStates ← T.reachableStates(validStates) // which states can we reach from the valid states

  For each reachableState in reachableStates:

    (maxScore, prevState) ← *viterbiUpdate(reachableState,t,O[t],viterbiChart,T)* // the viterbi update

    viterbiChart[t][reachableState]← (maxScore,prevState) // store the best score and the pointer

return *followBackpointer(viterbiChart)*

Viterbi-Transducer Version

- The function *viterbiUpdate(state,timeStep,observation,chart,transducer)* does the heavy lifting
- I am not giving an exact pseudo code, however what it should do:

1. Access the scores for all possible incoming states (you get these from the transducer) from the chart (access it at the given timeStep)
2. Add the edge scores for a transition from the previous states to *state*
   - *Yet again, the transducer has that information stored*
3. Find the previous state, which now have the highest score
4. Add the node score (ask the Transducer) to the score
5. Return the tuple consisting of (score,state) for the maximum

➔ This requires O(#S) complexity

Viterbi-Transducer Version

Function **viterbiDecode(O,T)**

viterbiChart ← ViterbiChart(0,T) // init an empty chart

For each timestep t in O.length:

validStates ← viterbiChart.validStates($t-1$) // all states that were reachable in the last time step

reachableStates ← T.reachableStates(validStates) // which states can we reach from the valid states

For each reachableState in reachableStates:

(maxScore,prevState) ← *viterbiUpdate(reachableState,t,O[t],viterbiChart,T)* // the viterbi update

viterbiChart[t][reachableState]← (maxScore,prevState) // store the best score and the pointer

return *followBackpointer(viterbiChart*)

➔ In total the algorithm runs in $O(\#S^2 \cdot O.\text{length})$

Viterbi-Transducer Version

- The Viterbi-Chart is a data-structure (2-dimensional array) which basically stores tuples (score,state)
- The function followBackpointer(viterbiChart) is used at the end to decode the best sequence
- However this version does always **add**, but we initially **multiplied**

➔ **Easy transformation, we just take the log and start to call the resulting values „scores“**

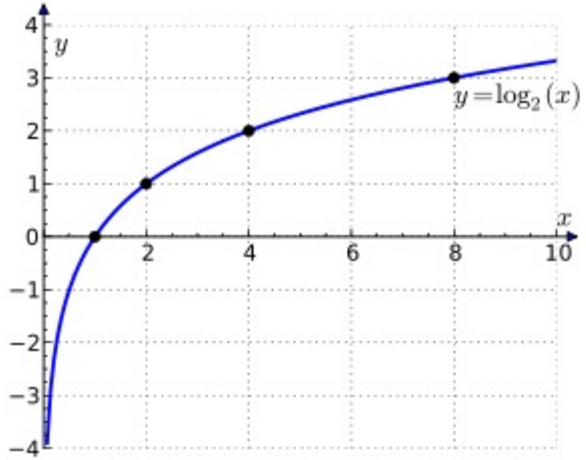
➔ **So each edge/node in the transducer stores the log of a probability instead of the raw probability**

Viterbi-Transducer Version

- Why can we simply take the logarithm?
- Our optimization problem is: $$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$
- So we are looking for a maximum, and the logarithm is a monotonic function:

➔ $x_1 < x_2 \Rightarrow \log_2 x_1 < \log_2 x_2$

➔ The position of the maximum does not change



Viterbi-Transducer Version

- With the logarithm (I'm using the natural, but it doesn't matter), our problem changes into:

$$argmax_{t_1^n} \prod_i P(w_i|t_i) \cdot P(t_i|t_{i-1}) = argmax_{t_1^n} \sum_i (\ln(P(w_i|t_i)) + \ln(P(t_i|t_{i-1})))$$

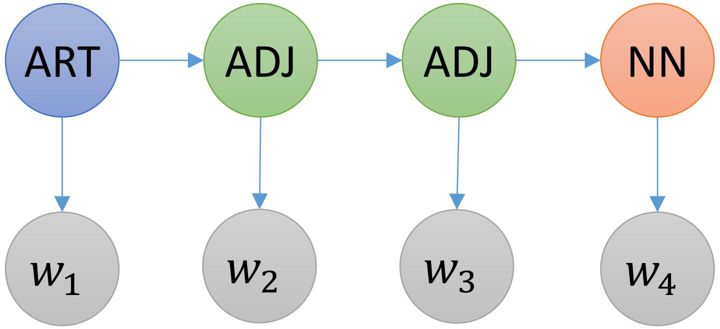
- Let us introduce a symbol for the node-score s_N and for the edge-score s_E

$$argmax_{t_1^n} \sum_i \left(s_N(t_i, i) + s_E(t_i, t_{i-1})\right)$$

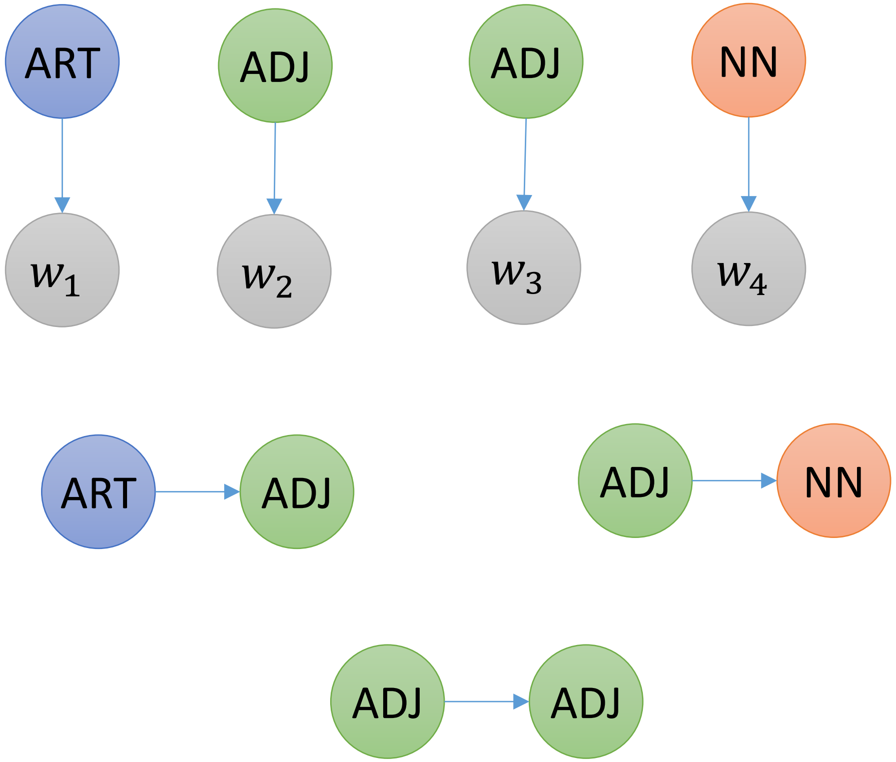
Viterbi-Transducer Version

- Let us interpret what this means: $argmax_{t_1^n} \sum_i \left(s_N(t_i, i) + s_E(t_i, t_{i-1})\right)$

The score of this



Is the sum of this:



➔ We score a sequence by abusing its inner (repetitive) structure

Evaluation of POS Tagging

- Once you have your POS tagger running, how do you evaluate it?
    - Overall error rate
    - Error rates on particular tags
    - Error rates on particular words
    - Confusion-Matrix with respect to a gold-standard (sample solution) test set (Label Accuracy, Label F1)

- Addressed in the chapter "Evaluation"

		Actual class	
		Cat	Dog
Predicted class	Cat	6	3
	Dog	5	4

Viterbi -Recap

- So what did we learn?
  - We started with a probabilistic interpretation of the structured problem
  - Introduced HMMs, which model the problem using two sets of probabilities:
    - State-to-State probabilities
    - Observation probability
  - Learned an efficient way to determine the most probable sequence, given a HMM model
  - Generalized this idea, so that we can implement the Viterbi entirely on a transducer