Machine Learning

Maximum Entropy Markov Models (MEMM)

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**

What we achieved so far

- We are trying to find the sequence which gets the highest score:

$$argmax_{t_1^n} \prod_i P(w_i|t_i) \cdot P(t_i|t_{i-1}) = argmax_{t_1^n} \sum_i \left(s_N(t_i, i) + s_E(t_i, t_{i-1})\right)$$

Edge-score

Node-score

What we achieved so far

- In our most advanced settings, we had:

$$argmax_{t_1^n} \prod_i P(w_i|t_i) \cdot P(t_i|t_{i-1}) = argmax_{t_1^n} \sum_i \left(s_N(t_i, i) + s_E(t_i, t_{i-1})\right)$$

Edge-score: A decision Tree

Node-score: A MaxEnt classifier

What we achieved so far

- But even this model is just an approximation of

$$argmax_{t_1^n}\ P(t_1^n|w_1^n) \sim argmax_{t_1^n} \prod_i P(w_i|t_i) \cdot P(t_i|t_{i-1})$$

- In the same manner, we could approximate as follows:

$$argmax_{t_1^n}\ P(t_1^n|w_1^n) \sim argmax_{t_1^n} \prod_i P(t_i|w_i, t_{i-1})$$

What we achieved so far

- Comparison

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

- Has features between observations w and current tag t_i
- Has features between current tag t_i and previous tag t_{i-1}
- Decoded using Viterbi

$$argmax_{t_1^n} \prod_i P(t_i|w_i, t_{i-1})$$

- Has features between observations w and current tag t_i
- Has features between current tag t_i and previous tag t_{i-1}
- Has features involving all three: w, t_i, t_{i-1}
- Decoded using Viterbi

Maximum Entropy Markov Model

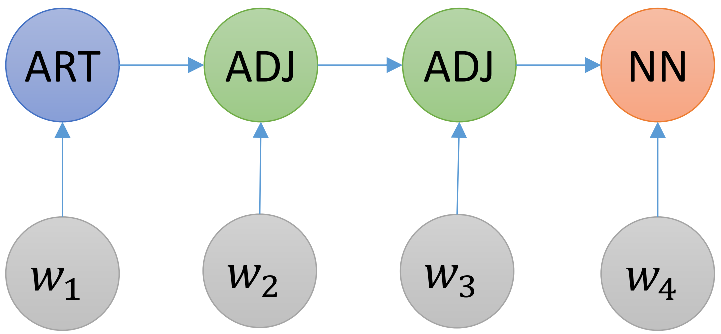
- We use a different view of our approximation
- And build a single Maximum Entropy classifier for this distribution $P(t_i|w_i, t_{i-1})$
- And as you can already recall, this is the same as having a regular Maximum Entropy classifier, but:
  - The labels are now tuples!
  - Same holds for the decoding, but all of our node scores are 0! (But our edge scores are more potent!)

$$argmax_{t_1^n} \prod_i P(t_i|w_i, t_{i-1})$$

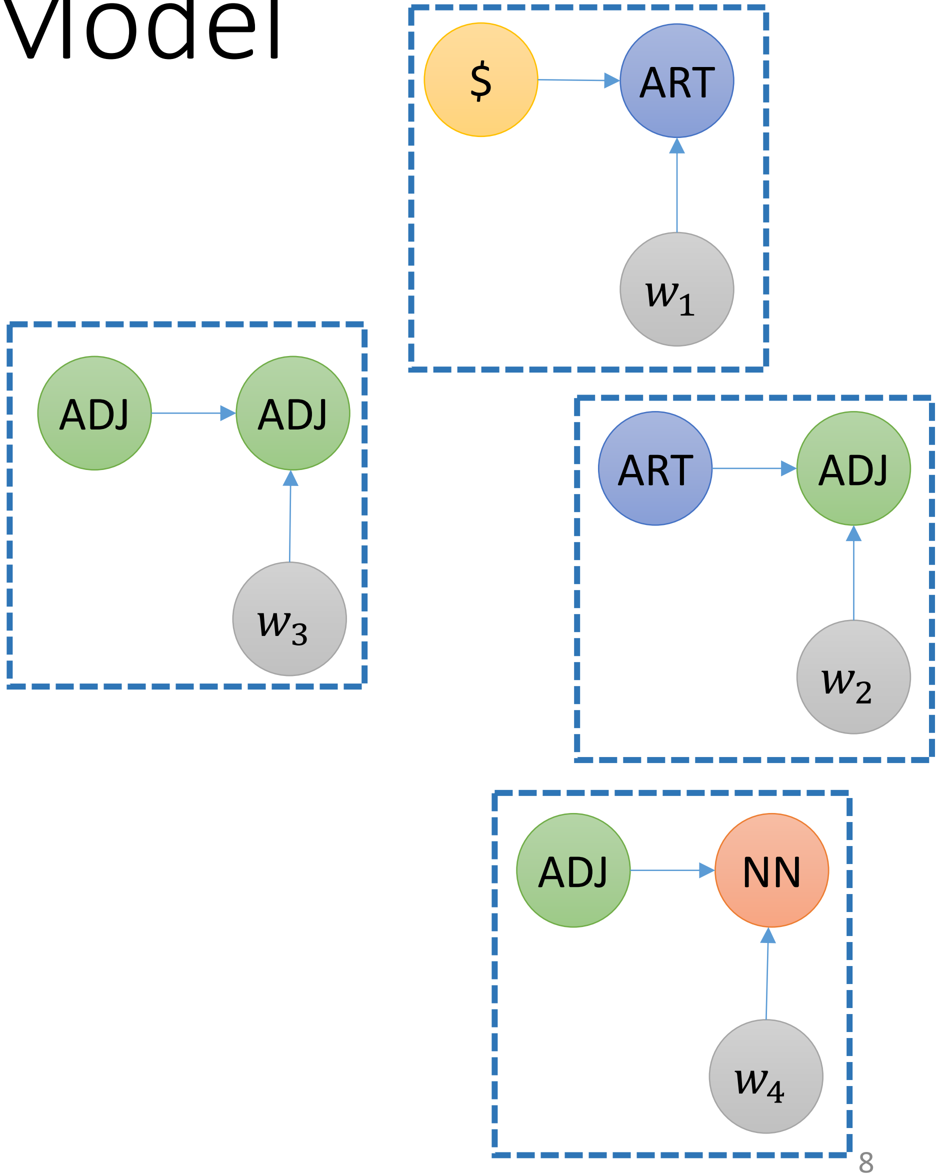
Maximum Entropy Markov Model

- We can now dissect the sequence as follows:
- The score of a sequence is now:

The score of this

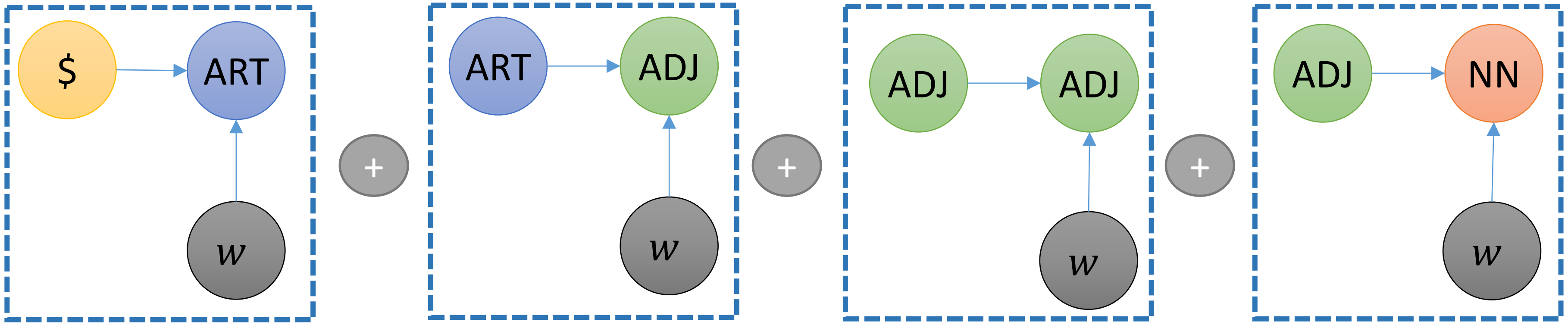


Becomes the sum of this:



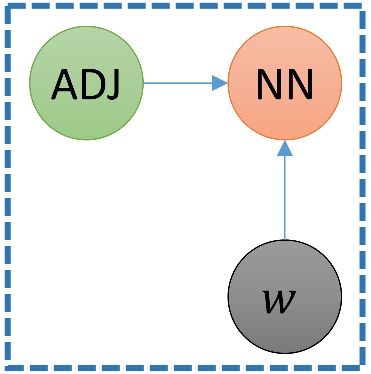
Maximum Entropy Markov Model

- But since we are using the Maximum Entropy framework, we can access the entire input w and use it for feature calculation
- And score as follows:



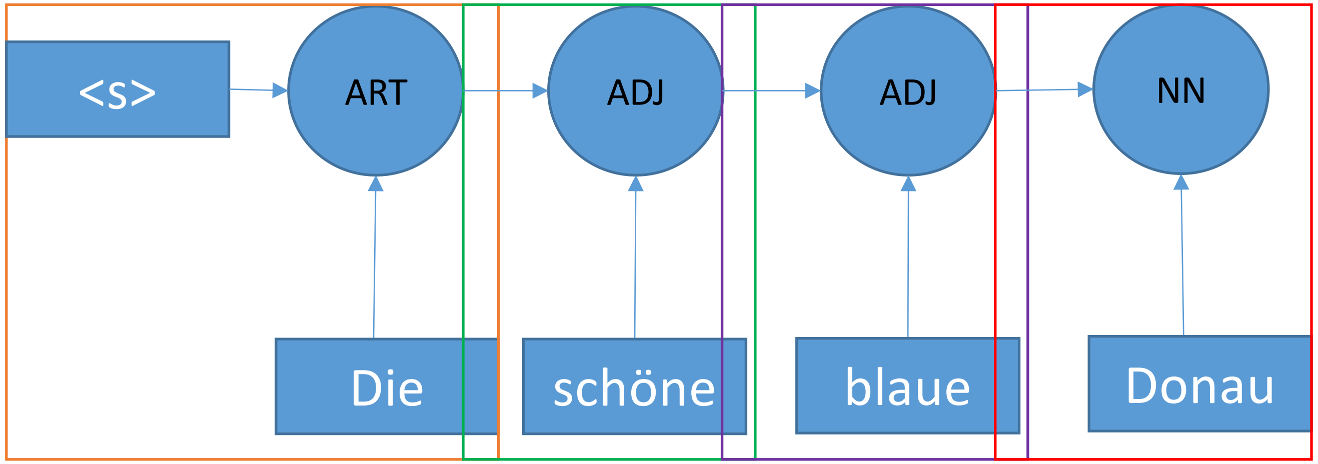
Maximum Entropy Markov Model

- Templates:
  - We can now compare the expressiveness of a model using feature templates $\Phi(w, t)$
  - A classical Maximum Entropy classifier has only one template $\Phi_{Node}(w, t_i)$
  - The presented MEMM has the following templates:
    - $\Phi_{Node}(w, t_i)$
    - $\Phi_{Edge}(w, t_i, t_{i-1})$
  - This constitutes a MEMM of order 1



Example: MEMM

- We are going a bit more global:

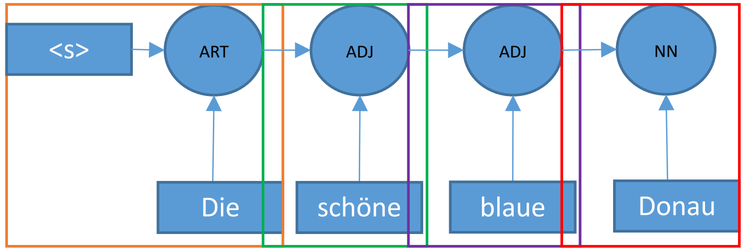


- 4 local models, 2 possible feature templates:

$$\Phi(w, y_i) \text{ and } \Phi(w, y_i, y_{i-1})$$

- Yet again we reuse a single MaxEnt at every stage

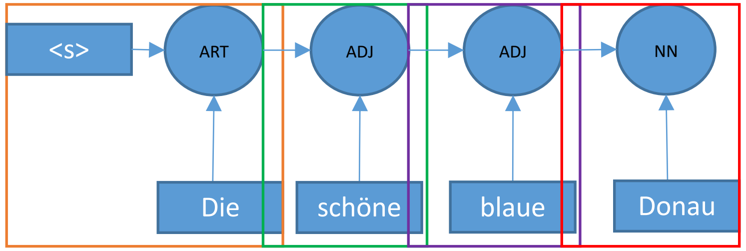
Example: MEMM



- We use the features of the simple MaxEnt from the template $\Phi(w, y_i)$

Feature	λ for ART	λ for ADJ	λ for NN
CurrentWord=Die	0.6	0.1	0.25
CurrentWord=schöne	-0.1	0.8	0.3
CurrentWord=blaue	0.1	0.6	0.2
CurrentWord=Donau	0.1	0.1	1.4

Example: MEMM

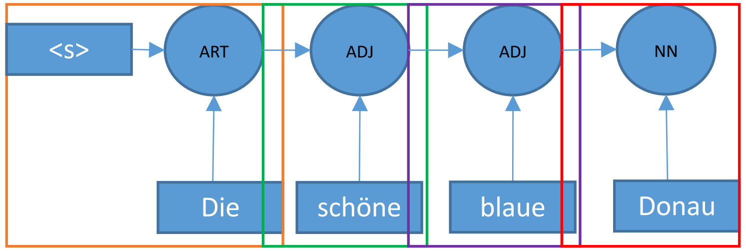


And on top we use features of the template $\Phi(w, y_i, y_{i-1})$

- For y_i=ART

Feature	$\lambda\ ART \rightarrow ART$	$\lambda\ ADJ \rightarrow ART$	$\lambda\ NN \rightarrow ART$	$\lambda < s > \rightarrow ART$
CurrentWord=Die	-0.1	0.1	0.1	0.8
CurrentWord=schöne	-0.3	-0.8	-0.2	0.1
CurrentWord=blaue	-1.2	-0.6	-0.2	-0.5
CurrentWord=Donau	-0.1	-0.1	-1.4	-1

Example: MEMM

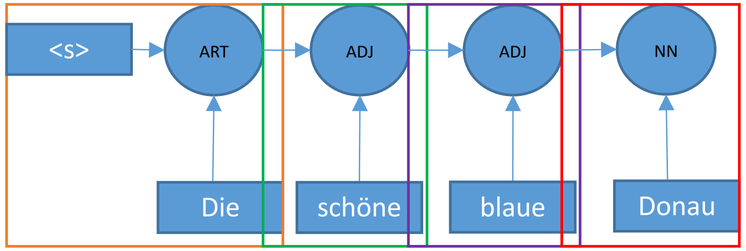


And on top we use features of the template $\Phi(w, y_i, y_{i-1})$

- For y_i=ADJ

Feature	$\lambda\ ART \rightarrow ADJ$	$\lambda\ ADJ \rightarrow ADJ$	$\lambda\ NN \rightarrow ADJ$	$\lambda <s> \rightarrow ADJ$
CurrentWord=Die	-0.3	0.1	-0.2	0.8
CurrentWord=schöne	0.9	-0.8	-0.2	0.1
CurrentWord=blaue	-1.2	0.7	-0.2	-0.5
CurrentWord=Donau	-0.1	-0.1	-1.4	-1

Example: MEMM



And on top we use features of the template $\Phi(w, y_i, y_{i-1})$

- For y_i=NN

Feature	$\lambda\, ART \rightarrow NN$	$\lambda\, ADJ \rightarrow NN$	$\lambda\, NN \rightarrow NN$	$\lambda < s > \rightarrow NN$
CurrentWord=Die	-0.1	0.25	-0.45	0.2
CurrentWord=schöne	0.1	-0.8	-0.2	-0.1
CurrentWord=blaue	-0.5	-0.3	-0.2	-0.5
CurrentWord=Donau	0.5	1.5	0.2	0.3

Example: MEMM

- By changing our model we can now use 48 additional features, so in total we got 60 features
- This is all done by the model, we only defined 4 features involving x:

Feature
CurrentWord=Die
CurrentWord=schöne
CurrentWord=blaue
CurrentWord=Donau

- Maximum Entropy combines those with every possible label, generating 12 features
- A local model with the current template bloats every feature an additional 12 times for every combination of $y_{i-1} \rightarrow y_i$

Example: MEMM

- The more expressive our feature templates get, the more sparse our observations in the training data

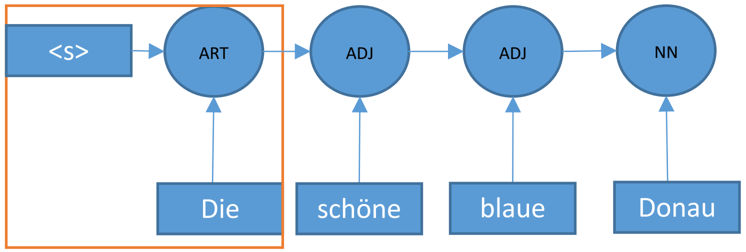
- We can therefore say that our model uses ***highly specific*** features.

➔One usually keeps all smaller templates to „back-off“ to the less specific features in case we face something we have never seen

Example: MEMM

- Back to calculating…

Example: MEMM



- Example calculation for $y_1 = ART\ and\ y_0 = start$

- $p(ART|w, y_0 =< s >) =$

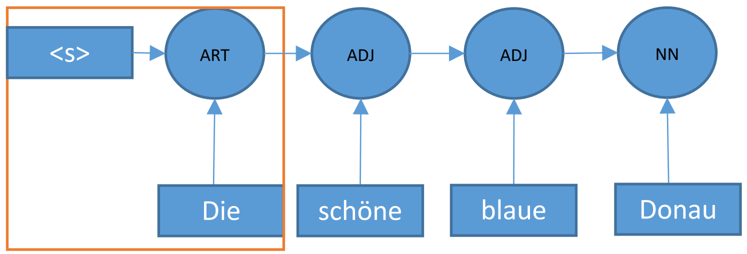
$$\frac{\exp(\sum_{f_i \in featuretable} \lambda_i f_i\,(w, y = ART, y_0 =< s >))}{\sum_{\hat{y} \in \{ART, ADJ, NN\}} exp\left(\sum_{f_i \in featuretable} \lambda_i f_i\,(w, y = \hat{y}, y_0 =< s >)\right)}$$

- $p(ART|w, y_0 =< s >) = \frac{\exp(0.6+0.8)}{\exp(0.6+0.8)+\exp(0.1+0.8)+\exp(0.25+0.2)} = 0.5$

- $p(ADJ|w, y_0 =< s >) = \frac{\exp(0.1+0.8)}{\exp(0.6+0.8)+\exp(0.1+0.8)+\exp(0.25+0.2)} = 0.3$

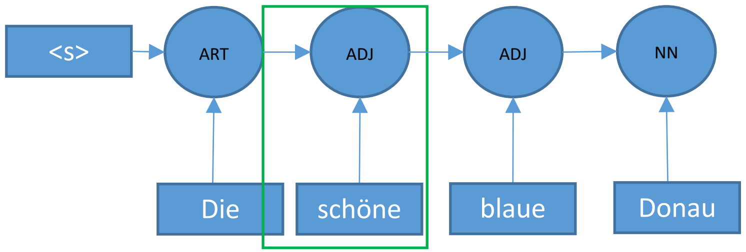
- $p(NN|w, y_0 =< s >) = \frac{\exp(0.25+0.2)}{\exp(0.6+0.8)+\exp(0.1+0.8)+\exp(0.25+0.2)} = 0.2$

Example: MEMM



- Now we could continue calculating and we would end up with 27 additional probabilities, 9 for every transition
- I'm only giving the probabilities for the next time step since the rest is calculated in the same manner

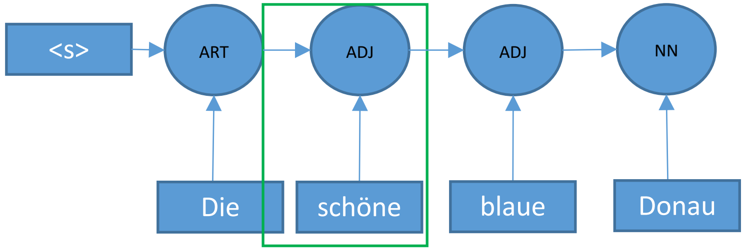
Example: MEMM



Assume that the previous Word has the tag ART

- $p(ART|x, y_1 = ART) = \frac{\exp(-0.3-0.1)}{\exp(-0.3-0.1)+\exp(0.8+0.9)+\exp(0.3+0.1)} \approx 0.087$
- $p(ADJ|x, y_1 = ART) = \frac{\exp(0.8+0.9)}{\exp(-0.3-0.1)+\exp(0.8+0.9)+\exp(0.3+0.1)} \approx 0.716$
- $p(NN|x, y_1 = ART) = \frac{\exp(0.3+0.1)}{\exp(-0.3-0.1)+\exp(0.8+0.9)+\exp(0.3+0.1)} \approx 0.195$

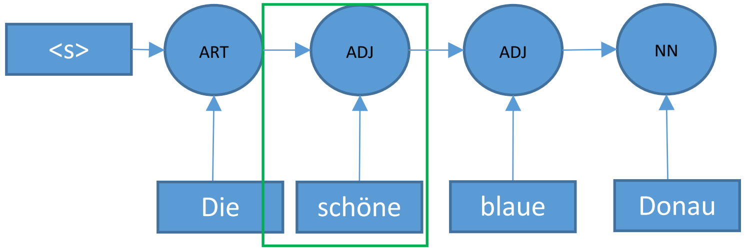
Example: MEMM



Assume that the previous Word has the tag ADJ

- $p(ART|x, y_1 = ADJ) = \frac{\exp(-0.1-0.8)}{\exp(-0.1-0.8)+\exp(0.8-0.8)+\exp(0.3-0.8)} = 0.2$
- $p(ADJ|x, y_1 = ADJ) = \frac{\exp(0.8-0.8)}{\exp(-0.1-0.8)+\exp(0.8-0.8)+\exp(0.3-0.8)} = 0.5$
- $p(NN|x, y_1 = ADJ) = \frac{\exp(0.3-0.8)}{\exp(-0.1-0.8)+\exp(0.8-0.8)+\exp(0.3-0.8)} = 0.3$

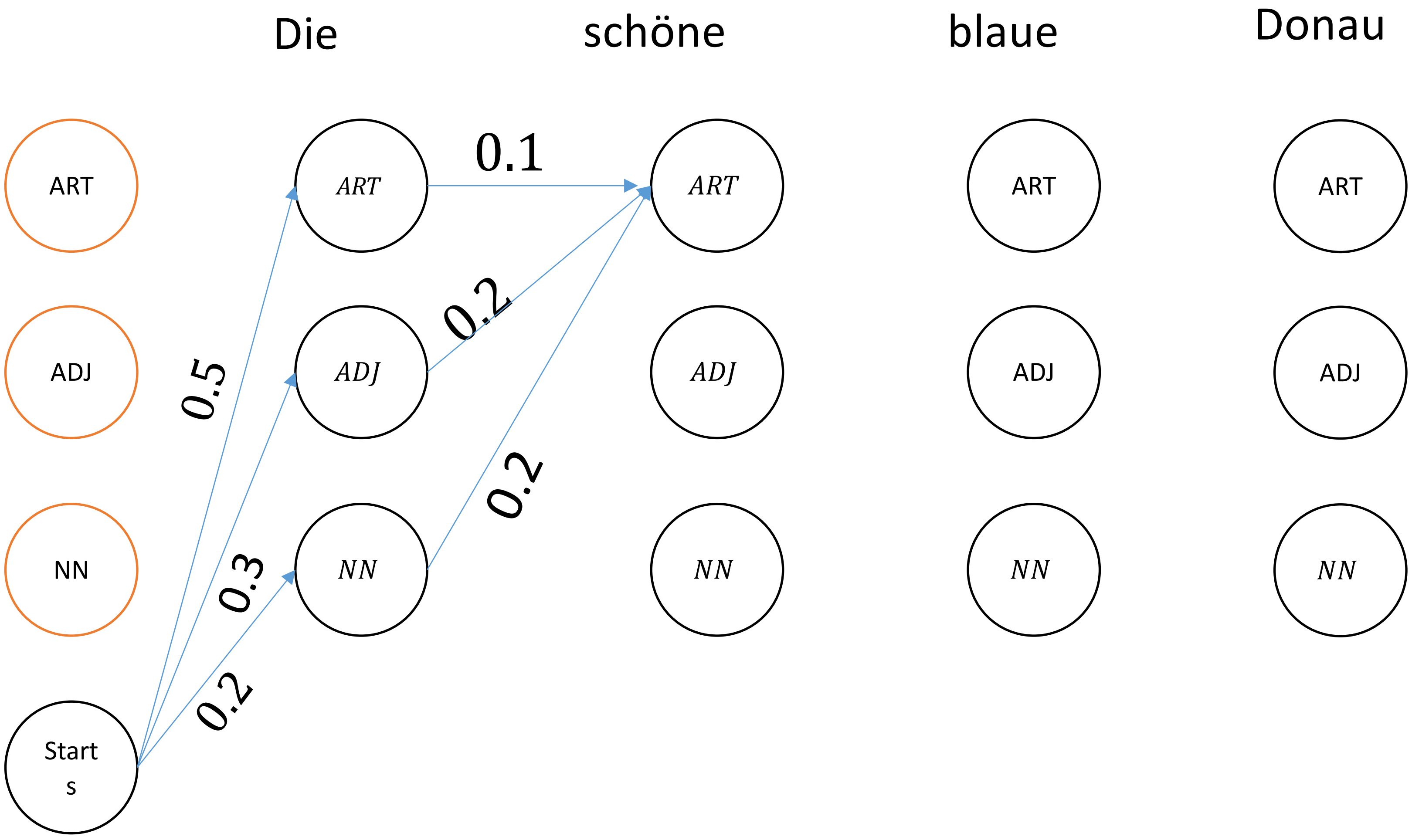
Example: MEMM



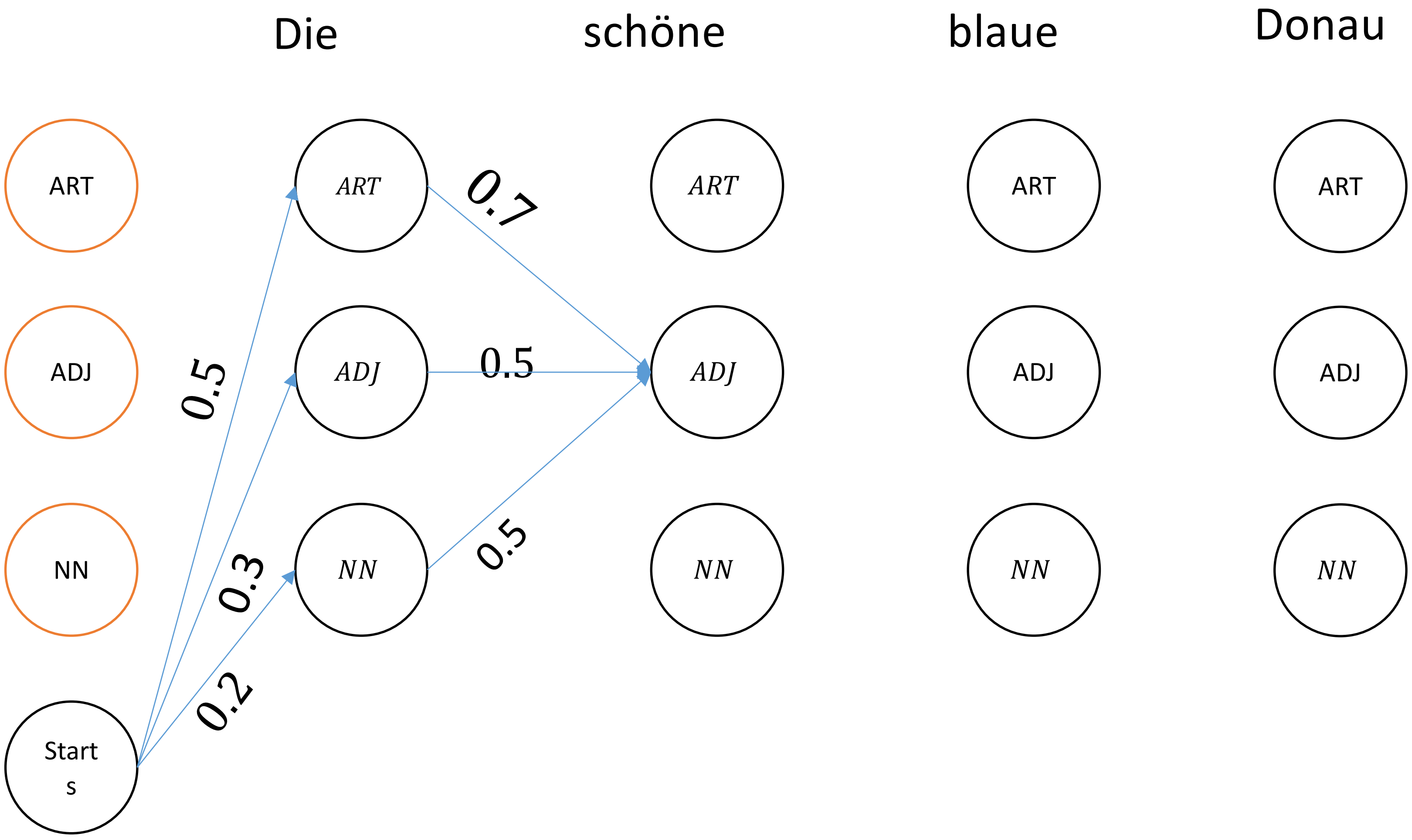
Assume that the previous Word has the tag NN

- $p(ART|x, y_1 = NN) = \frac{\exp(-0.1-0.2)}{\exp(-0.1-0.2)+\exp(0.8-0.2)+\exp(0.3-0.2)} = 0.2$
- $p(ADJ|x, y_1 = NN) = \frac{\exp(0.8-0.2)}{\exp(-0.1-0.2)+\exp(0.8-0.2)+\exp(0.3-0.2)} = 0.5$
- $p(NN|x, y_1 = NN) = \frac{\exp(0.3-0.2)}{\exp(-0.1-0.2)+\exp(0.8-0.2)+\exp(0.3-0.2)} = 0.3$

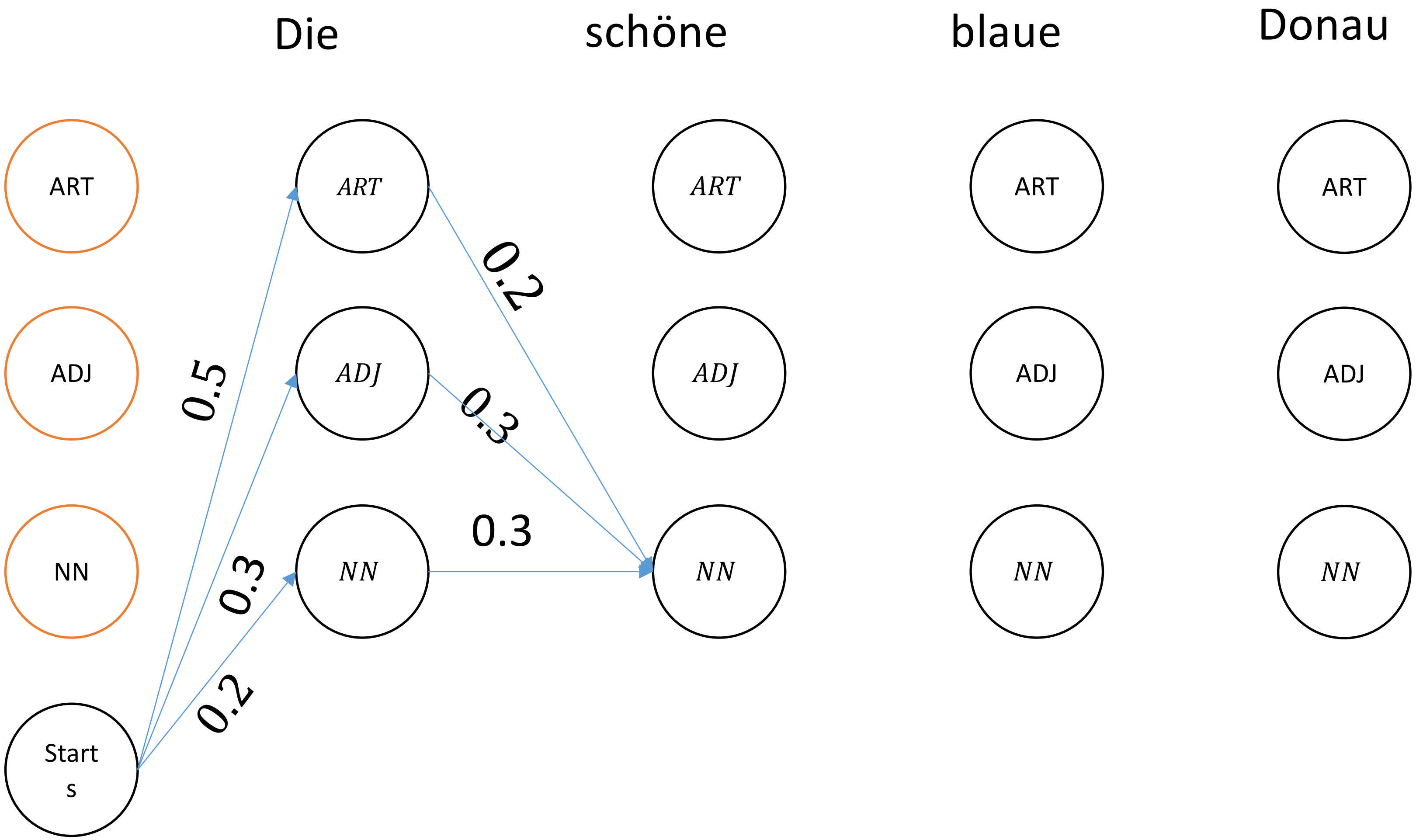
Viterbi Trellis



Viterbi Trellis

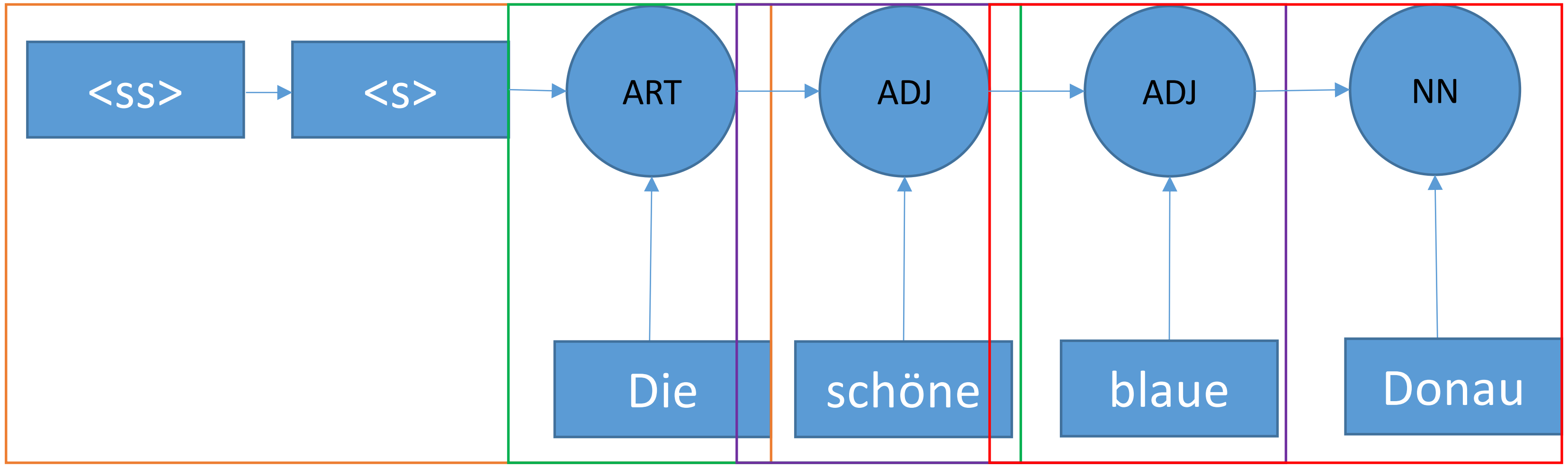


Viterbi Trellis



Maximum Entropy Markov Models (MEMM)

- By decoding with Viterbi we can find the most probable sequence
- We could now create even larger models



- And would get more powerful features (and exponentially more!)

Maximum Entropy Markov Models (MEMM)

- In the end we would include every other label into our local decision, and we would end up with:
- „local models with global feature templates“

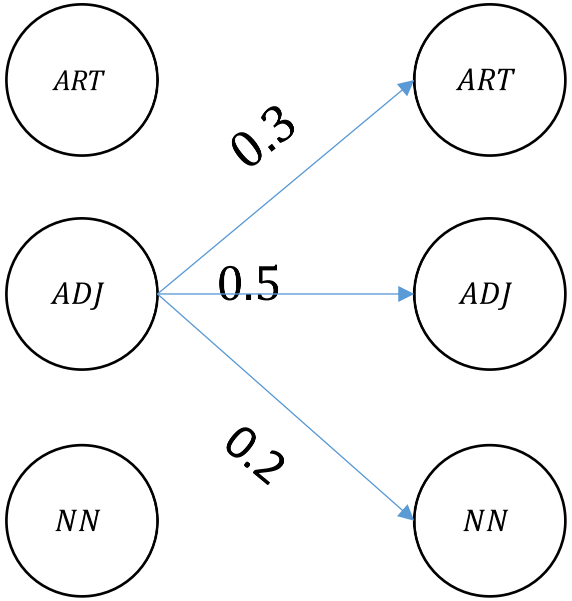
- ➔ is this the best we can do?

Maximum Entropy Markov Models

Label Bias of local models

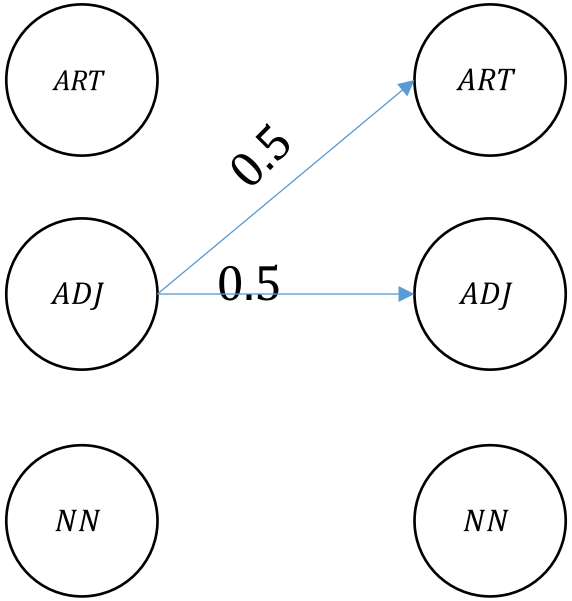
Maximum Entropy Markov Models (MEMM)

- We use the Viterbi to „glue together“ our local decisions
- What if one node can reach all states (artificial numbers)



Maximum Entropy Markov Models (MEMM)

- against one node that has only 2 outputs

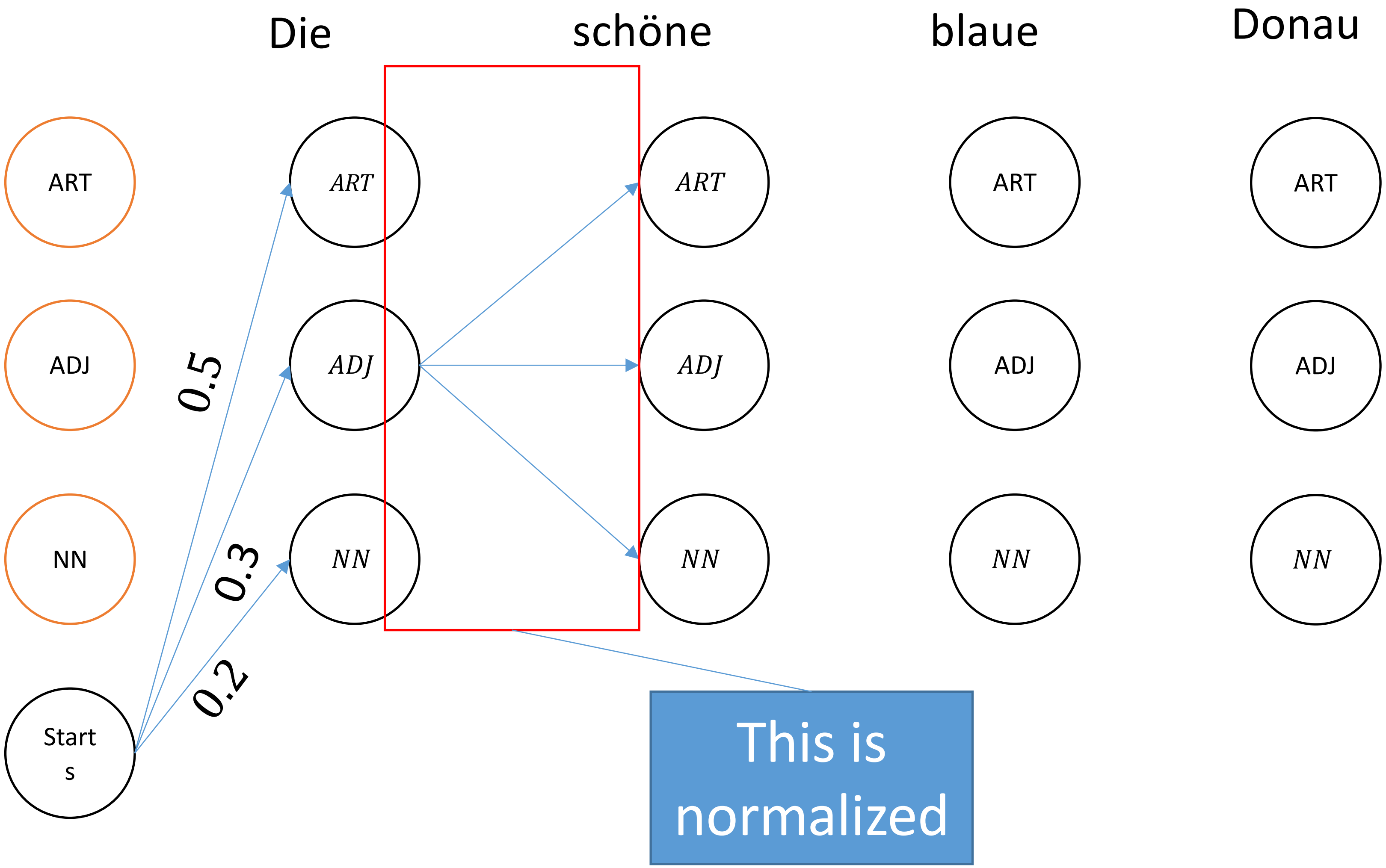


- The path will always prefer states with less outputs, no matter the task („Label Bias“)

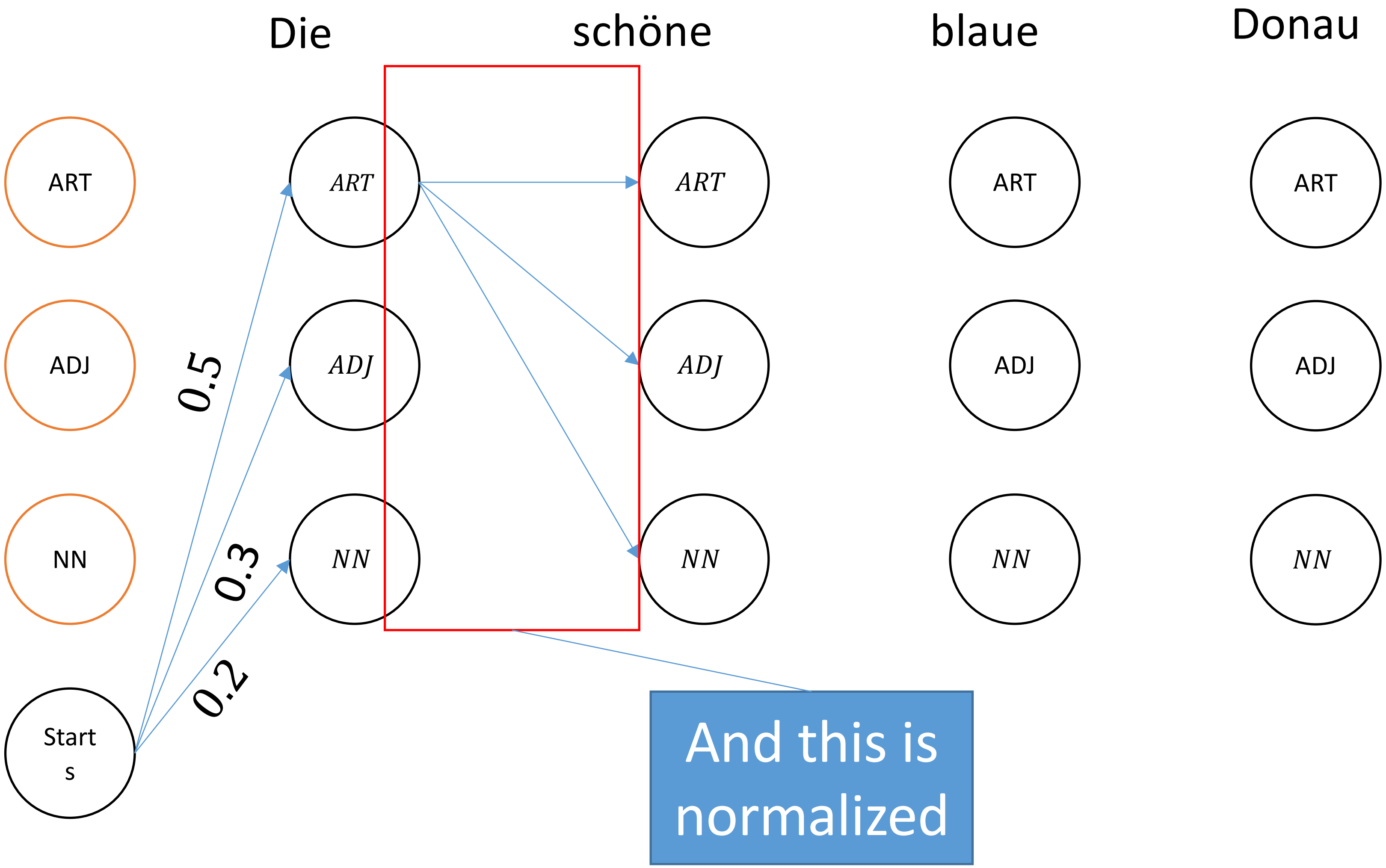
Maximum Entropy Markov Models (MEMM)

- This might happen because our probabilities are normalized locally and do not incorporate this information…

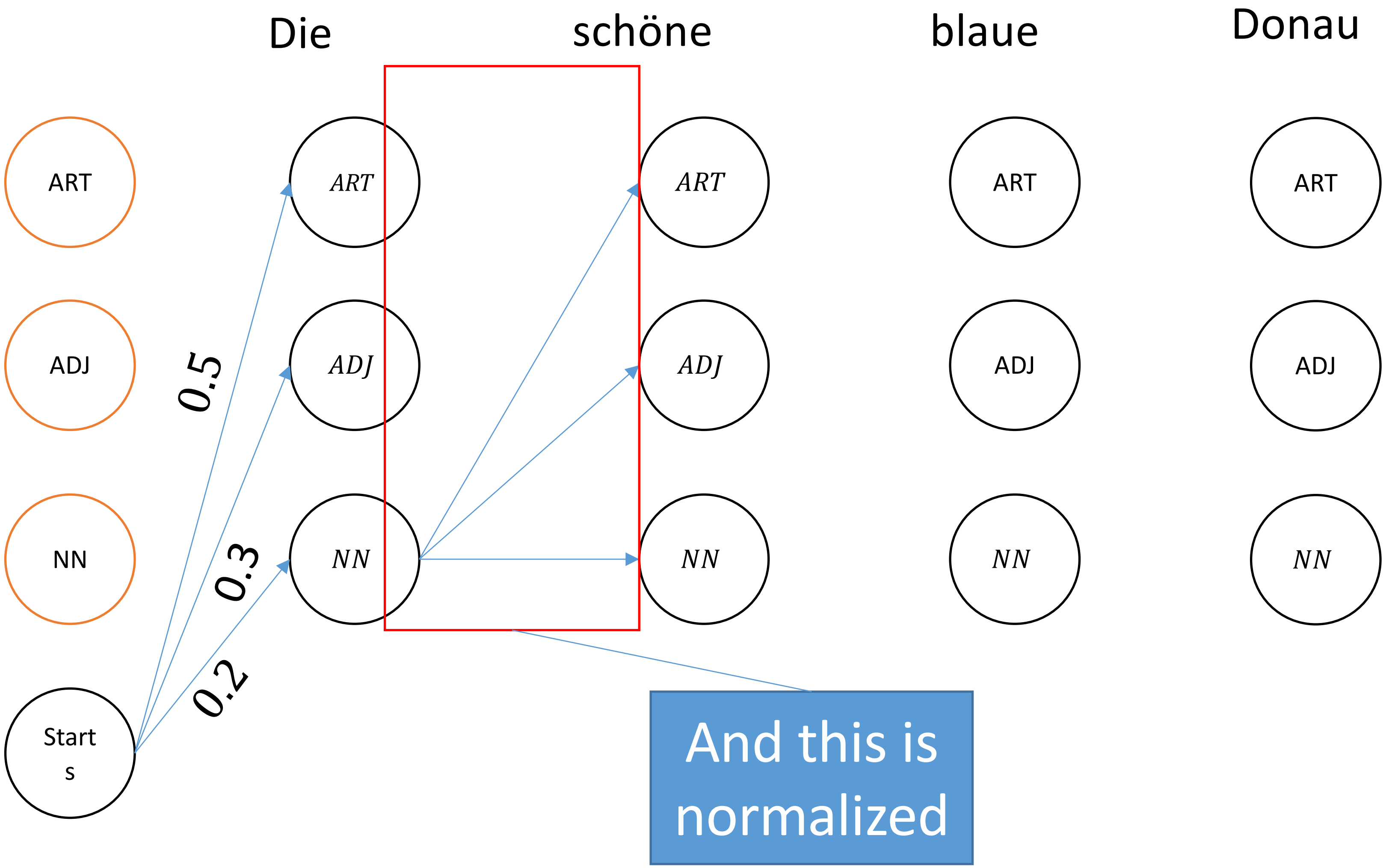
Viterbi Trellis



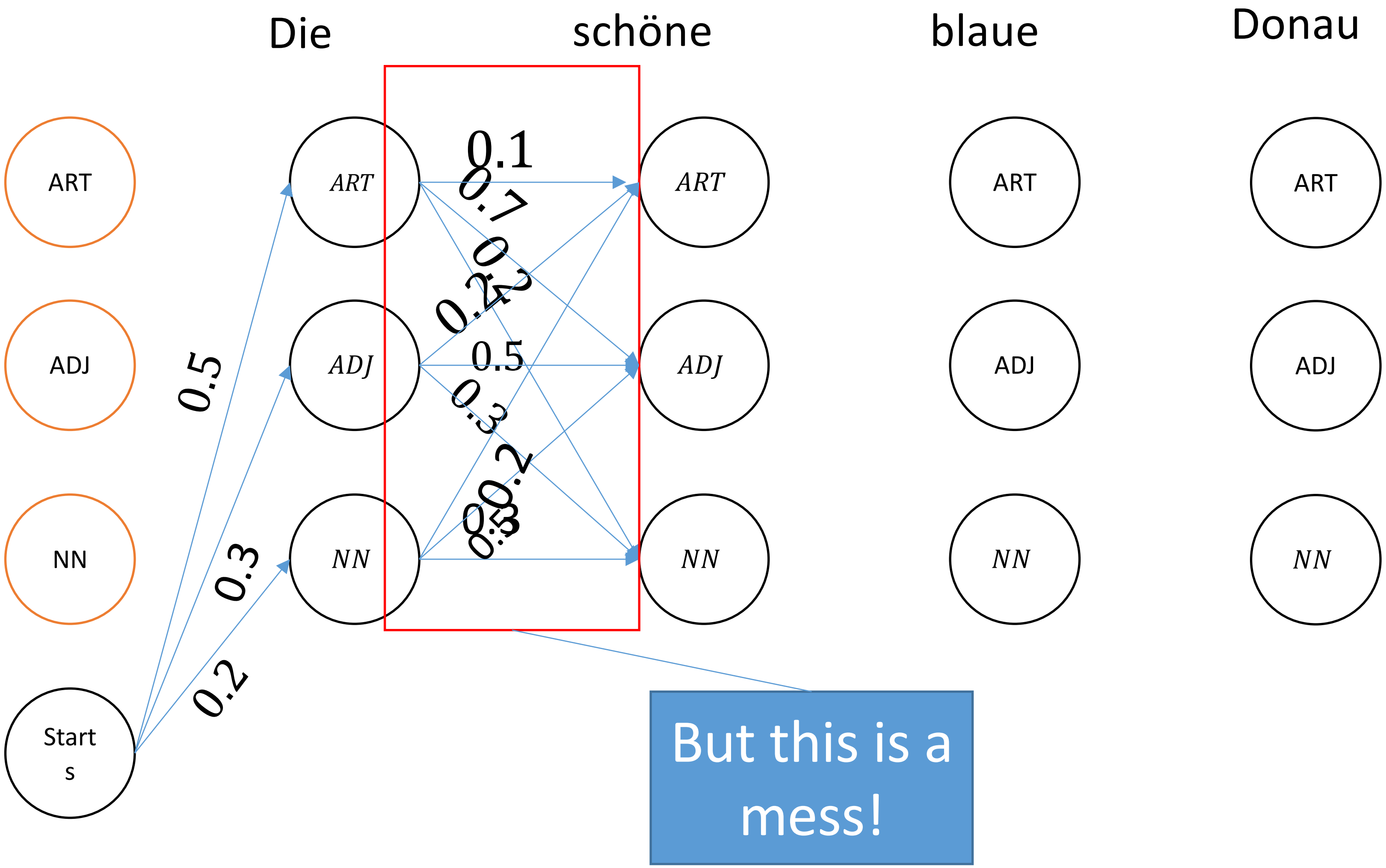
Viterbi Trellis



Viterbi Trellis



Viterbi Trellis



Maximum Entropy Markov Models (MEMM)

- Since the local normalizations do not know anything about each other they are „explaining themselves away“

➔ We have to normalize the right way

➔ This idea is incorporated into Conditional Random Fields!

Recap

- A MEMM is just a Maximum Entropy classifier, with different features (additional template Φ_{Edge})
- We decode with Viterbi, but the scores originate from a single source
- The MEMM normalizes locally, and is therefore prone to the Label-Bias