

## Übung-WS16-17-WH.pdf

**Aufgabe 6)** Ein Bigramm-HMM-Tagger berechnet die beste Tagfolge mit dem Viterbi-Algorithmus. Die Viterbi-Wahrscheinlichkeiten sind wie folgt definiert:

$$\begin{aligned}\delta_t(0) &= \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases} \\ \delta_t(k) &= \max_{t'} \delta_{t'}(k-1) p(t|t') p(w_k|t) \text{ für } 0 < k \leq n+1\end{aligned}$$

Dabei sind  $t, t'$  Tags,  $k$  eine Wortposition,  $\delta_t(k)$  eine Viterbiwahrscheinlichkeit und  $\psi_t(k)$  das beste Vorgängertag von  $t$  an Position  $k$ .

Schreiben Sie eine Funktion *viterbi(words)*, welche die Viterbi-Wahrscheinlichkeiten berechnet. Das Argument *words* ist die Liste der zu annotierenden Wörter. Sie können die Viterbiwahrscheinlichkeiten in einer Liste (Array) von Hashtabellen (Dictionaries) mit Namen *vitprob* speichern. Eine Funktion *lexprobs(word)* liefert eine Hashtabelle (oder Dictionary) mit den möglichen Tags von *word* als Schlüsseln und den entsprechenden Wahrscheinlichkeiten als Werten. Eine Funktion *contextprob(tag1,tag2)* gibt die Wahrscheinlichkeit zurück, dass tag2 auf tag1 folgt. Beide Funktionen sind gegeben und müssen nicht implementiert werden.

```

# bigram viterbi
def viterbi(words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words = [''] + words + [''] # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitprob = [dict() for _ in range(len(words))] # speichert viterbi prob
    bestprev = [dict() for _ in range(len(words))] # speichert die besten Vorgänger
    vitprob[0]['<s>'] = 1
    for i in range(1, len(words)): # iteriert über all Wörter w_i
        lexprobs_word = lexprobs(words[i]) # mögliche tags t an Position i und p(w_i|t)
        for current_tag, lexprob in lexprobs_word.items(): # iteriert über die möglichen tags t an Position i und p(w_i|t)
            for prev_tag in vitprob[i-1]: # iteriert über alle Tags an position i-1
                p = vitprob[i-1][prev_tag] * contextprob(prev_tag, current_tag) * lexprob # vit prob Kandidaten berechnen
                if current_tag not in vitprob[i] or vitprob[i][current_tag] < p: # vit prob Kandidaten maximieren
                    vitprob[i][current_tag] = p # speichert vit prob in der Tabelle
                    bestprev[i][current_tag] = prev_tag # speichert der beste Vorgänger-Tag

    # der beste Tag für Position n+1 definieren
    best_tag = "<s>"
    # beste Tagfolge extrahieren
    result_tags = []
    for i in range(len(words)-1, 1, -1):
        best_tag = bestprev[i][best_tag]
        result_tags.append(best_tag)
    return reversed(result_tags) # Tagfolge umdrehen

```

1. Initialisierung:  $\delta_t(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

3. Ausgabe: (für  $0 < k \leq n$ )

$$t_{n+1} = \langle s \rangle$$

$$t_k = \psi_{t_{k+1}}(k+1)$$

Code in Colab Notebook:

[https://colab.research.google.com/drive/1cxf8rRPSGEk\\_07MMahIIlNo06VjPuWNd8?usp=sharing](https://colab.research.google.com/drive/1cxf8rRPSGEk_07MMahIIlNo06VjPuWNd8?usp=sharing)

# Viterbi-Algorithmus (Bigramm-Tagger)

1. Initialisierung:  $\delta_t(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

2. Berechnung: (für  $0 < k \leq n + 1$ )

$$\delta_t(k) = \max_{t'} \delta_{t'}(k-1) p(t|t') p(w_k|t)$$

$$\psi_t(k) = \arg \max_{t'} \delta_{t'}(k-1) p(t|t') p(w_k|t)$$

3. Ausgabe: (für  $0 < k \leq n$ )

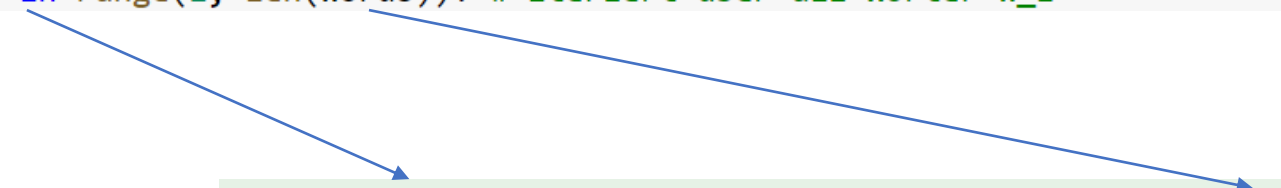
$$t_{n+1} = \langle s \rangle$$

$$t_k = \psi_{t_{k+1}}(k+1)$$

$t, t'$  sind Tags,  $k$  sind Wortpositionen,  $\delta_t(k)$  sind die Viterbiwahrscheinlichkeiten und  $\psi_t(k)$  sind die besten Vorgängertags

```
# bigram viterbi
def viterbi(words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words = [''] + words + [''] # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitprob = [dict() for _ in range(len(words))] # speichert viterbi prob
    bestprev = [dict() for _ in range(len(words))] # speichert die besten Vorgänger
    vitprob[0]['<s>'] = 1
    for i in range(1, len(words)): # iteriert über all Wörter w_i
```



The diagram shows two blue arrows originating from the code in the block above. One arrow points from the word 'can' at index 2 to the word 'can' in the table. The other arrow points from the word 'can' at index 5 to the word 'can' in the table. Additionally, red and green arrows within the table indicate transitions between tags: a red arrow from MD to NN, a green arrow from NN to VB, and a red arrow from VB to NN.

0	1	2	3	4	5	6
	I	can	can	a	can	
<s>	PRO	MD	MD		MD	
	PN	NN	NN	DT	NN	<s>
		VB	VB		VB	

```

# bigram viterbi
def viterbi(words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words = [''] + words + [''] # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitprob = [dict() for _ in range(len(words))] # speichert viterbi prob
    bestprev = [dict() for _ in range(len(words))] # speichert die besten Vorgänger
    vitprob[0]['<s>'] = 1
    for i in range(1, len(words)): # iteriert über all Wörter w_i
        lexprobs_word = lexprobs(words[i]) # mögliche tags t an Position i und p(w_i|t)
        for current_tag, lexprob in lexprobs_word.items(): # iteriert über die möglichen tags t an Position i und p(w_i|t)

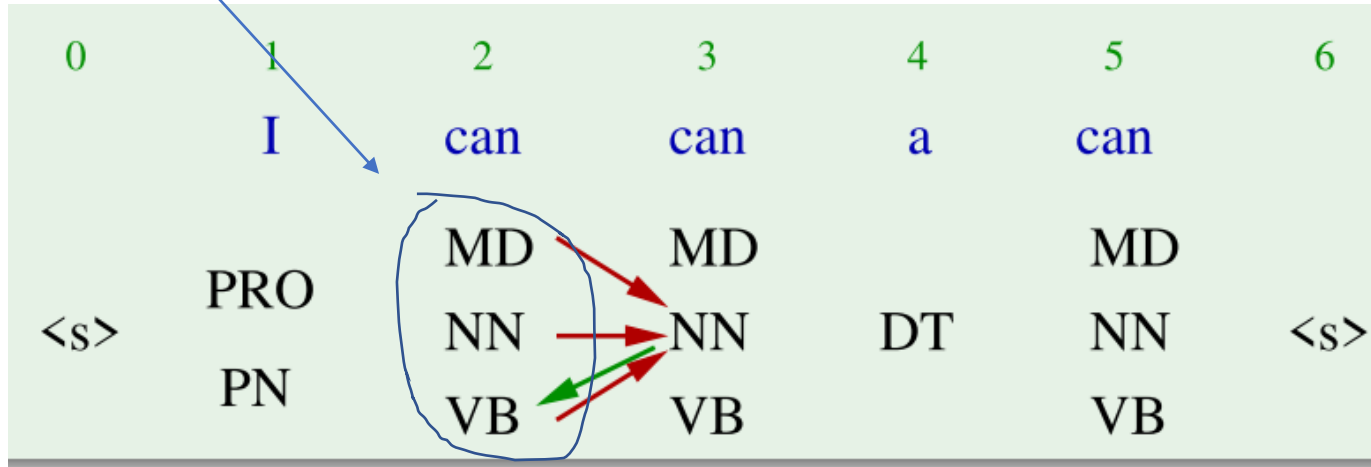
```

0	1	2	3	4	5	6
	I	can	can	a	can	
		MD	MD		MD	
<s>	PRO	NN	NN	DT	NN	<s>
	PN	VB	VB		VB	

Mit i=3

```
# bigram viterbi
def viterbi(words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words = [''] + words + [''] # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitprob = [dict() for _ in range(len(words))] # speichert viterbi prob
    bestprev = [dict() for _ in range(len(words))] # speichert die besten Vorgänger
    vitprob[0]['<s>'] = 1
    for i in range(1, len(words)): # iteriert über all Wörter w_i
        lexprobs_word = lexprobs(words[i]) # mögliche tags t an Position i und p(w_i|t)
        for current_tag, lexprob in lexprobs_word.items(): # iteriert über die möglichen tags t an Position i und p(w_i|t)
            for prev_tag in vitprob[i-1]: # iteriert über alle Tags an position i-1
```



Mit i=3

```
# bigram viterbi
def viterbi(words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words = [''] + words + [''] # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitprob = [dict() for _ in range(len(words))] # speichert viterbi prob
    bestprev = [dict() for _ in range(len(words))] # speichert die besten Vorgänger
    vitprob[0]['<s>'] = 1
    for i in range(1, len(words)): # iteriert über all Wörter w_i
        lexprobs_word = lexprobs(words[i]) # mögliche tags t an Position i und p(w_i|t)
        for current_tag, lexprob in lexprobs_word.items(): # iteriert über die möglichen tags t an Position i und p(w_i|t)
            for prev_tag in vitprob[i-1]: # iteriert über alle Tags an position i-1
                p = vitprob[i-1][prev_tag] * contextprob(prev_tag,current_tag) * lexprob # vit prob Kandidaten berechnen
```

$$\delta_t(k) = \max_{t'} \delta_{t'}(k-1) p(t|t') p(w_k|t)$$

$$\delta_{NN}(3) = \max \begin{pmatrix} \delta_{MD}(2) p(NN|MD) p(can|NN), \\ \delta_{NN}(2) p(NN|NN) p(can|NN), \\ \delta_{VB}(2) p(NN|VB) p(can|NN) \end{pmatrix}$$

0	1	2	3	4	5	6
	I	can	can	a	can	
<s>	PRO	MD	MD		MD	
	PN	NN	NN	DT	NN	<s>
		VB	VB		VB	

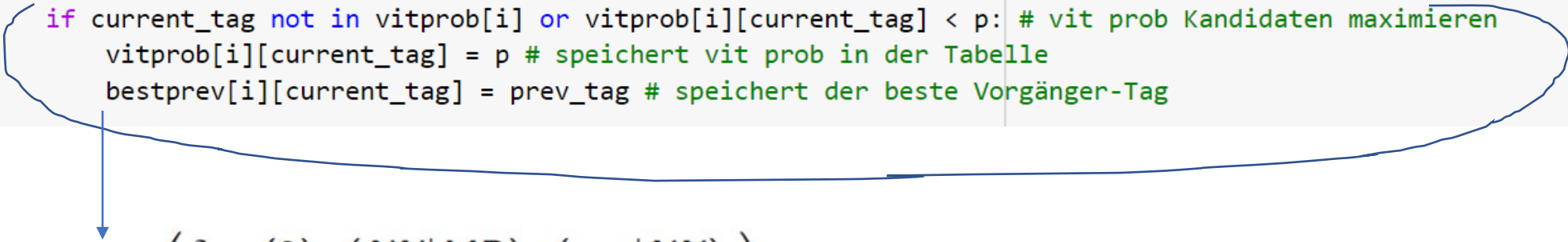


```

# bigram viterbi
def viterbi(words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words = [''] + words + [''] # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitprob = [dict() for _ in range(len(words))] # speichert viterbi prob
    bestprev = [dict() for _ in range(len(words))] # speichert die besten Vorgänger
    vitprob[0]['<s>'] = 1
    for i in range(1, len(words)): # iteriert über all Wörter w_i
        lexprobs_word = lexprobs(words[i]) # mögliche tags t an Position i und p(w_i|t)
        for current_tag, lexprob in lexprobs_word.items(): # iteriert über die möglichen tags t an Position i und p(w_i|t)
            for prev_tag in vitprob[i-1]: # iteriert über alle Tags an position i-1
                p = vitprob[i-1][prev_tag] * contextprob(prev_tag, current_tag) * lexprob # vit prob Kandidaten berechnen
                if current_tag not in vitprob[i] or vitprob[i][current_tag] < p: # vit prob Kandidaten maximieren
                    vitprob[i][current_tag] = p # speichert vit prob in der Tabelle
                    bestprev[i][current_tag] = prev_tag # speichert der beste Vorgänger-Tag

```



$$\delta_{NN}(3) = \max \begin{pmatrix} \delta_{MD}(2) p(NN|MD) p(can|NN), \\ \delta_{NN}(2) p(NN|NN) p(can|NN), \\ \delta_{VB}(2) p(NN|VB) p(can|NN) \end{pmatrix}$$



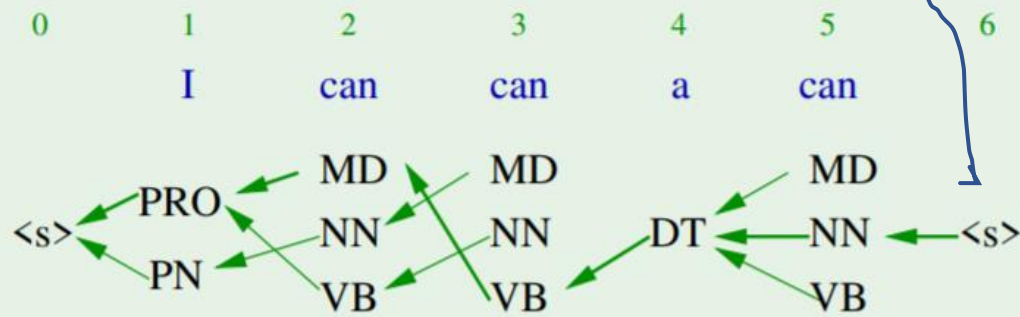
```

# bigram viterbi
def viterbi(words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words = [''] + words + [''] # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitprob = [dict() for _ in range(len(words))] # speichert viterbi prob
    bestprev = [dict() for _ in range(len(words))] # speichert die besten Vorgänger
    vitprob[0]['<s>'] = 1
    for i in range(1, len(words)): # iteriert über all Wörter w_i
        lexprobs_word = lexprobs(words[i]) # mögliche tags t an Position i und p(w_i|t)
        for current_tag, lexprob in lexprobs_word.items(): # iteriert über die möglichen tags t an Position i und p(w_i|t)
            for prev_tag in vitprob[i-1]: # iteriert über alle Tags an position i-1
                p = vitprob[i-1][prev_tag] * contextprob(prev_tag, current_tag) * lexprob # vit prob Kandidaten berechnen
                if current_tag not in vitprob[i] or vitprob[i][current_tag] < p: # vit prob Kandidaten maximieren
                    vitprob[i][current_tag] = p # speichert vit prob in der Tabelle
                    bestprev[i][current_tag] = prev_tag # speichert der beste Vorgänger-Tag

    # der beste Tag für Position n+1 definieren
    best_tag = "<s>"
    # beste Tagfolge extrahieren
    result_tags = []
    for i in range(len(words)-1, 1, -1):
        best_tag = bestprev[i][best_tag]
        result_tags.append(best_tag)
    return reversed(result_tags) # Tagfolge umdrehen

```



## Viterbi Trigram

```
def viterbi(words):  
    ''' Trigram | berechnet die beste Tagfolge für eine gegebene Wortfolge '''  
    words = [''] + words + [''] # Grenztokens hinzufügen  
    # Initialisierung der Viterbi-Tabelle  
    vitprob = [dict() for _ in range(len(words))] # speichert viterbi prob  
    bestprev = [dict() for _ in range(len(words))] # speichert die besten Vorgänger  
    vitprob[0][('<s>', '<s>')] = 1  
    for i in range(1, len(words)): # iteriert über all Wörter w_i  
        lexprobs_word = lexprobs(words[i]) # mögliche tags t an Position i und p(w_i|t)  
        for current_tag, lexprob in lexprobs_word.items(): # iteriert über die möglichen tags t an Position i und p(w_i|t)  
            for tagpair in vitprob[i-1]: # iteriert über alle Tags an position i-1  
                tag1, tag2 = tagpair # Kontext-Tags  
                p = vitprob[i-1][tagpair] * contextprob(tagpair, current_tag) * lexprob # vit prob Kandidaten berechnen  
                newtagpair = (tag2, current_tag)  
                if newtagpair not in vitprob[i] or vitprob[i][newtagpair] < p: # vit prob Kandidaten maximieren  
                    vitprob[i][newtagpair] = p # speichert vit prob in der Tabelle  
                    bestprev[i][newtagpair] = tagpair # speichert der beste Vorgänger-Tag  
  
    # in der letzten Spalte das Tagpaar mit der höchsten Bewertung suchen  
    best_tagpair = max(vitprob[-1], key=vitprob[-1].get)  
    # beste Tagfolge extrahieren  
    result_tags = []  
    for i in range(len(words)-1, 1, -1):  
        result_tags.append(best_tagpair[0])  
        best_tagpair = bestprev[i][best_tagpair]  
    return reversed(result_tags) # Tagfolge umdrehen
```

Code in Colab Notebook:

[https://colab.research.google.com/drive/1h9c\\_SSKDPhxTmNLngGsDn4uCsYoPV95?usp=sharing](https://colab.research.google.com/drive/1h9c_SSKDPhxTmNLngGsDn4uCsYoPV95?usp=sharing)