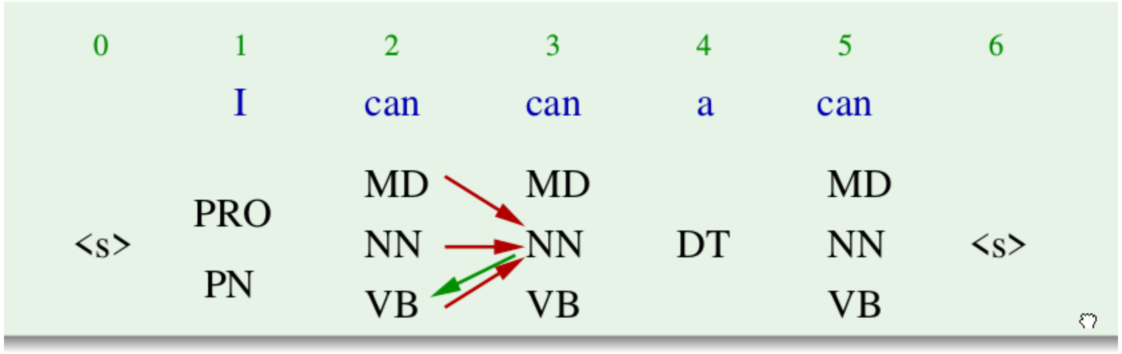
17.01.21

Bitte aus dem Foliensatz Übungsaufgaben_alle_Themen.pdf **slide 85 (Aufgabe 4)** besprechen

Aufgabe 4) Begründen Sie, warum der Viterbi-Algorithmus beim Wortart-Taggen mehr Zeit braucht, wenn die Zahl der möglichen Tags eines Wortes oder die Ordnung des HMMs (= Zahl der relevanten vorhergehenden Tags) erhöht wird.

Steigt die benötigte Rechenzeit schneller bei Erhöhung der Zahl der Tags oder bei Erhöhung der Ordnung des HMMs? (3 Punkte)



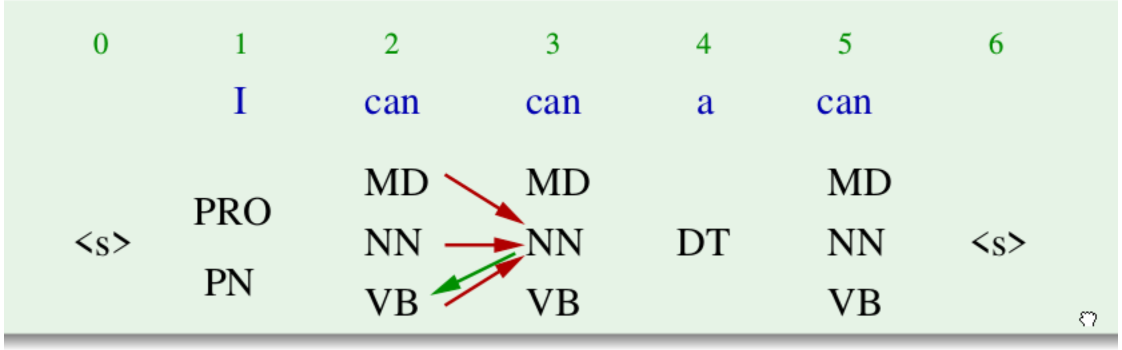
Wenn #Tag erhöht wird, erhöht sich die Anzahl von δ_t (k), die berechnet werden müssen. In jede Position k, berechnet der Algorithmus für jeden Tag t eine viterbi prob δ.

Z.B. Wenn “can” noch einen Tag hat, dann werden für position 2,3,5 jeweils vier viterbi-prob (statt 3 viterbi-prob) berechnet.

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

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

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



Wenn die Ordnung erhöht wird, dann erhöht sich auch die Anzahl von δ_t(k), die berechnet werden müssen.
In jeder Position k, berechnet der Algorithmus für jedes Tagpaar t',t eine viterbi prob δ.

Z.B. bigram vs trigram

Bei 2-gram
Für Position 3, berechnet der Algorithmus
vit_MD(3)
vit_NN(3)
vit_VB(3)

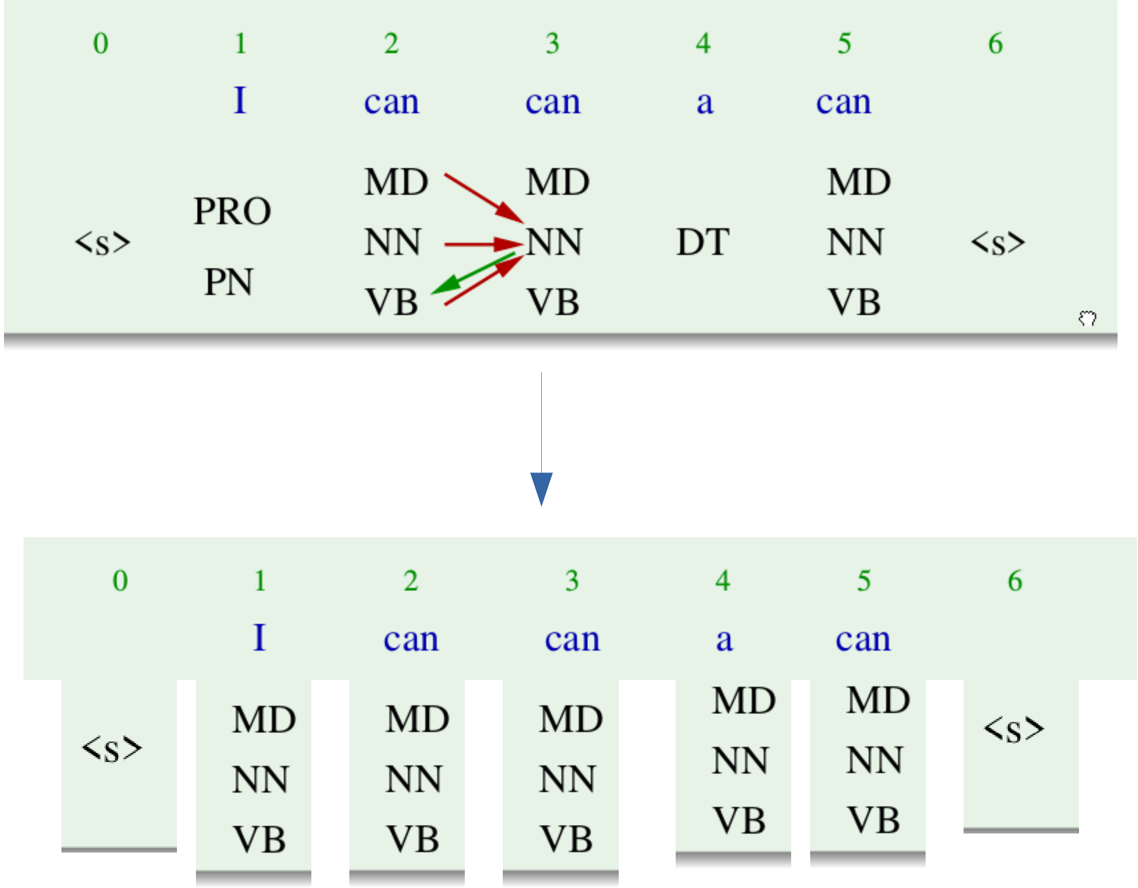
Bei 3-gram
Für Position 3, der Algorithmus
vit_MD,MD(3), vit_NN,MD(3),vit_VB,MD(3)
vit_MD,NN(3),vit_NN,NN(3),vit_VB,NN(3)
vit_MD,VB(3),vit_NN,VB(3),vit_VB,VB(3)

Aufgabe 4) Begründen Sie, warum der Viterbi-Algorithmus beim Wortart-Taggen mehr Zeit braucht, wenn die Zahl der möglichen Tags eines Wortes oder die Ordnung des HMMs (= Zahl der relevanten vorhergehenden Tags) erhöht wird.

Steigt die benötigte Rechenzeit schneller bei Erhöhung der Zahl der Tags oder bei Erhöhung der Ordnung des HMMs? (3 Punkte)

Antwort: Bei Erhöhung der Ordnung des HMMs

Wie prüfen wir das?
Wir vergleichen die Rechenzeit von beiden Fällen.



This graphic shows that the algorithm already knows the possible tags für each word.
However, in the practice, all words have the same set of possible tags. In the implementation we use a function that filters out the unlikely tags before running the viterbi algorithm. This we have to count this step in the time complexity as well.

With this setting, we want to compare the runtime of bigram HMM after increasing a tag versus after increasing the HMM order.

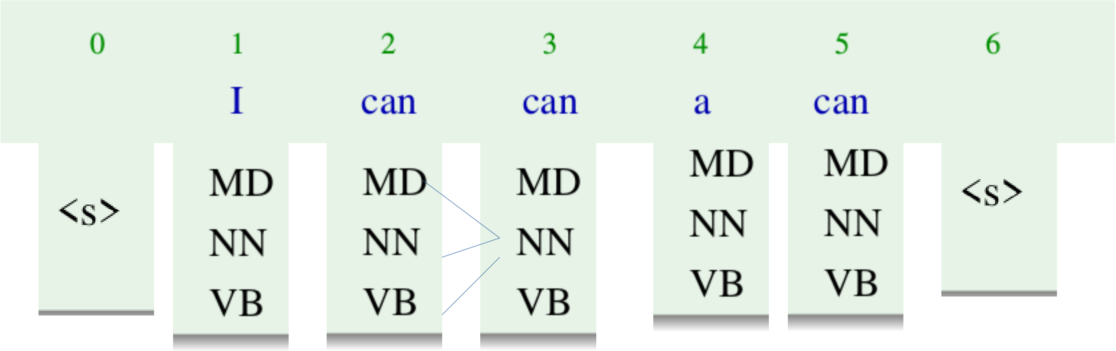
To get a better understanding of the question, we consider a bigram model, what is the runtime complexity of it?

Next we ask, what is the runtime after increasing 1 tag (or more) ?

Next, what is the runtime after increasing the order of the bigram HMM ? (= trigram model)

v	w_1=the	w_2=doctor	w_3=is	w_4=in	</s>
Noun	0	.0756	.001512	0	
Verb	0	.00021	.027216	0	
Det	.21	0	0	0	.0000272
Prep	0	0	0	.005443	
Adv	0	0	0	.000272	
	Det	Noun	Verb	Prep	

ref: http://people.cs.georgetown.edu/nschneid/cosc572/s18/12_viterbi_slides.pdf



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

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

What is the runtime complexity of a bigram model with 3 tags?

To calculate the runtime we count the number of operations done in the algorithm.

Let’s look at how many caculations the algorithm has to do to compute one viterbi prob.

Um vit_NN(3) zuberechnen, braucht der Algorihmus 3 Berechnungen/Operationen (Wenn wir die Max-Operation nicht mitberechnet).

An Position 3 werden Vit-probs für all tags berechnen.
D.h. Die Rechenzeit für Position 3 ist Anzahl der Tags(Position 3) * Anzahl der Tags (Position 2).

Da alle Wörter den gleichen Tagset haben ist die Rechenzeit für Position 3 dann T^2 (wobei Tfür die Anzahl der Tags steht).

Also, die Rechenzeit für jede Position T^2. Für die gesamte Wortfolge ist es dann **T^2 * N** (wenn N die Anzahl der Wörter ist).

Nächste Frage: wie ändert sich die Komplexität (Rechenzeit), wenn die Anzahl der Tags erhöht wird?

Wie ändert sich die Komplexität (Rechenzeit), wenn die **Anzahl der Tags erhöht wird?**

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

Bigram-HMM mit Anzahl der Tag = 3

Die Rechenzeit = O(T^2 * N)

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

Jetzt erhöhen wir die Anzahl der Tags um 1. Wir haben dann Bigram-HMM mit Anzahl der Tag = 4.

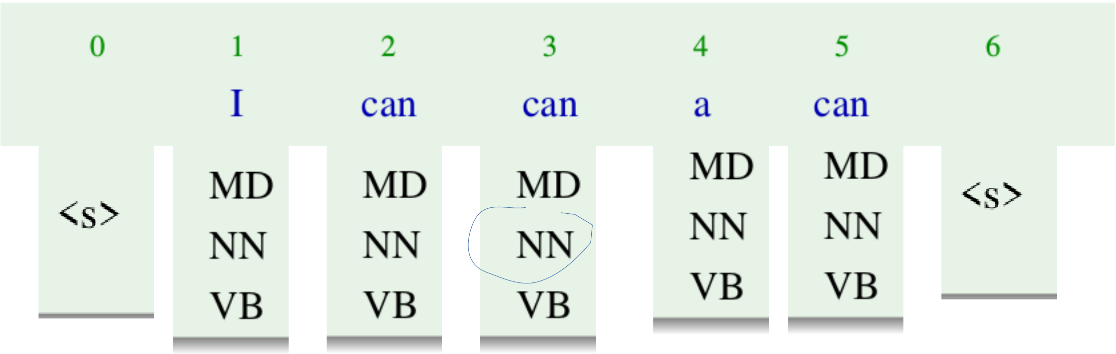
Für eine Position, braucht der Algorithmus 4*4 Operationen(Berechnungen) um alle Vit-Prob für alle Tags (an der Position) zu berechnen.

Die Rechenzeit = O((T+1)^2 * N)
Wenn wir die Anzahl der Tags um m erhöht, schreiben wir dann **O((T+m)^2 * N)**

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

Wir wollen als nächstes diese Rechenzeit mit der von einem 3-gram-HMM (mit 3 tags) vergleichen.

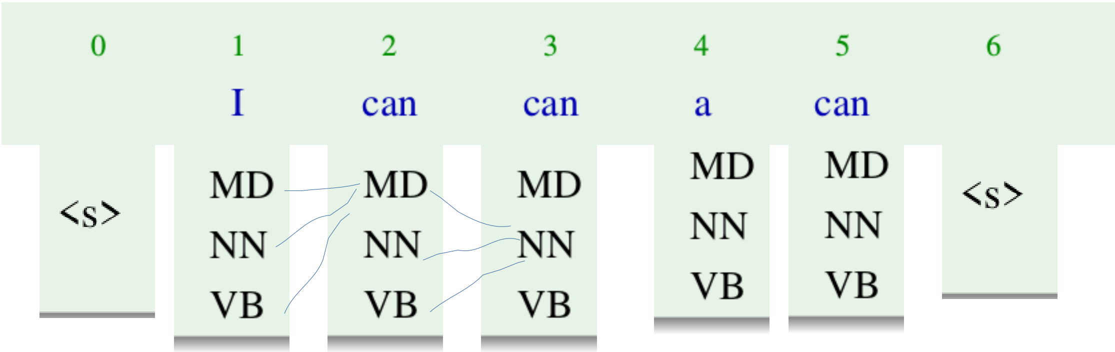
bigram HMM mit 3 tags



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

Komplexität = O(T^2 * N)

trigram HMM mit 3 tags



Komplexität von Trigram = O(T^3 * N)

In Trigramm HMM berechnet der Algorihmus
δ MD,MD(3), δ NN,MD(3),δ VB,MD(3)
δ MD,NN(3),δ NN,NN(3),δ VB,NN(3)
δ MD,VB(3),δ NN,VB(3),δ VB,VB(3)
Das ist 3 * 3 viterbi prob. (3 ist Anzahl der Tags)

Um eine Viterbiprob zu berechnen, braucht der Algo noch 3 Berechnungen. Z.B. für δ MD,NN (3)

δ MD,NN (3) = max[
δ MD,MD(2)p(NN| MD,MD)p(can|NN),
δ NN,MD(2)p(NN| NN,MD)p(can|NN),
δ VB,MD(2)p(NN| VB,MD)p(can|NN)]

Für Position 3 werden also 3 * 3 * 3 Berechnungen gemacht.

$O((T+m)^2 * N)$ vs $O(T^3 * N)$

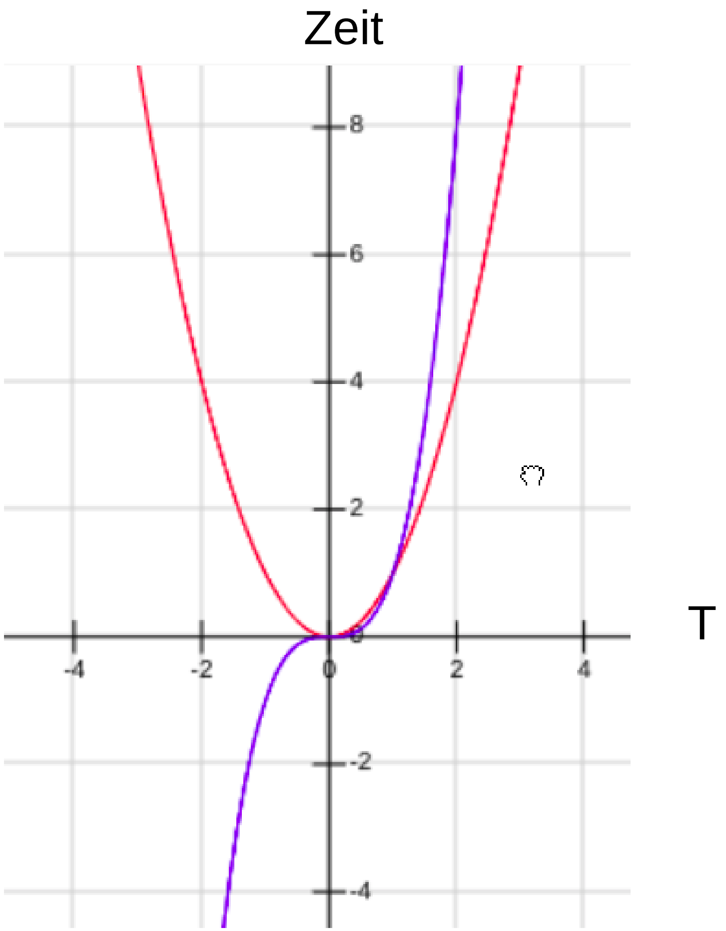
Welche Rechenzeit ist höher?

O((T+m)^2 * N)   vs O(T^3 * N)

N sind gleich bei beiden, kann man es weglassen.
In der Komplexität-Analyse ist m ein Konstant, kann man es auch weglassen.

O(T^2)   vs O(T^3)

Ab T = 1 erhöht sich O(T^3) schneller als O(T^2)



18.01.21
Bitte aus dem Foliensatz Übungsaufgaben_alle_Themen.pdf slide 98 (Aufgabe 10) besprechen

Aufgabe 10) Mit dem **Forward-/Backward-Algorithmus** können Sie die Wahrscheinlichkeit jedes möglichen Tags bei jedem Eingabewort berechnen. Sie wissen dann beispielsweise, dass das Tag *NN* beim 3. Wort eines gegebenen Satzes die Wahrscheinlichkeit 0,47 hat.

Wie könnten Sie auf Basis dieser Wahrscheinlichkeiten (sinnvoll) eine **beste Tagfolge** für den Satz definieren? (Die Lösung kennen Sie nicht aus der Vorlesung.)

Bekommen Sie mit dieser Methode dieselbe Tagfolge wie mit dem Viterbi-Algorithmus? Versuchen Sie, Ihre Antwort zu begründen. (3 Punkte)

Aufgabe 10) Mit dem **Forward-/Backward-Algorithmus** können Sie die Wahrscheinlichkeit jedes möglichen Tags bei jedem Eingabewort berechnen. Sie wissen dann beispielsweise, dass das Tag *NN* beim 3. Wort eines gegebenen Satzes die Wahrscheinlichkeit 0,47 hat.

Diese WK ist die Aposterori-WK

$$P(t_k = t|w_1^n) = \gamma_t(k)$$

Wie könnten Sie auf Basis dieser Wahrscheinlichkeiten (sinnvoll) eine **beste Tagfolge** für den Satz definieren? (Die Lösung kennen Sie nicht aus der Vorlesung.)

Bekommen Sie mit dieser Methode dieselbe Tagfolge wie mit dem Viterbi-Algorithmus? Versuchen Sie, Ihre Antwort zu begründen. (3 Punkte)

$$t_k = \arg\max_t \gamma_t(k) \quad \text{für } 1 \leq k \leq n$$

Nein

With two methods (Viterbi and posterior) to decode, which is more appropriate? When trying to classify each hidden state, the Posterior decoding method is more informative because it takes into account all possible paths when determining the most likely state. In contrast, the Viterbi method only takes into account one path, which may end up representing a minimal fraction of the total probability. At the same time, however, posterior decoding may give an invalid sequence of states! For example, the states identified at time points t and $t+1$ might have zero transition probability between them. As a result, selecting a decoding method is highly dependent on the application of interest.

Definition der Forward-Wahrscheinlichkeiten $\alpha_t(k)$

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

$$\alpha_t(k) = \sum_{t' \in T} \alpha_{t'}(k-1)\, p(t|t')\, p(w_k|t) \quad \text{für } 1 \leq k \leq n+1$$

$$\beta_t(n+1) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$$

$$\beta_t(k) = \sum_{t' \in T} p(t'|t)\, p(w_{k+1}|t')\, \beta_{t'}(k+1) \quad \text{für } 0 \leq k \leq n$$

und die Aposteriori-Wahrscheinlichkeiten

$$\gamma_t(k) = \frac{\alpha_t(k)\, \beta_t(k)}{\alpha_{\langle s \rangle}(n+1)} \quad \text{für } 1 \leq k \leq n$$

Dann extrahieren Sie die besten Tags

$$t_k = \arg\max_t \gamma_t(k) \quad \text{für } 1 \leq k \leq n$$

und geben sie aus.

Forward-Backward-Algorithmus kann auch zum Taggen benutzen (die beste Tagfolge zu berechnen).

Sie verwenden den Forward-Backward-Algorithmus hier also zum Taggen.[1]

[1]Dies ist ein alternativer Tagging-Algorithmus, der den Vorteil hat, dass zu jedem Tag die Aposteriori-Wahrscheinlichkeit als Maß der Zuverlässigkeit ausgegeben werden kann.

2 Posterior Decoding

Unterschied zu Viterbi

2.1 Motivation

Although the **Viterbi decoding** algorithm provides one means of estimating the hidden states underlying a sequence of observed characters, another valid means of inference is provided by **posterior decoding**.

Posterior decoding provides the most likely state at any point in time. To gain some intuition for posterior

Here, we've defined $f_k(t) = P(\pi_t = k, x_1, \ldots, x_t)$ and $b_k(t) = P(x_{t+1}, \ldots, x_n|\pi_t = k)$. As we will shortly see, these parameters are calculated using the **forward algorithm** and the **backward algorithm** respectively. To solve the posterior decoding problem, we merely need to solve each of these subproblems. The forward algorithm has been illustrated in the previous chapter and in the review at the start of this chapter and the backward algorithm will be explained in the next section.

Before explaining the backward algorithm, one may ask if Posterior Decoding can provide a path like the Viterbi algorithm would. The answer is yes; since Posterior Decoding can provide the most likely state for every point, assembling these states provides the path. Later on, we would compare the pros and cons of the paths generated using Viterbi algorithm and Posterior Decoding.

Seite 7: http://web.mit.edu/6.047/book-2012/Lecture08_HMMSII/Lecture08_HMMSII_standalone.pdf

With two methods (Viterbi and posterior) to decode, which is more appropriate? When trying to classify each hidden state, the Posterior decoding method is more informative because it takes into account all possible paths when determining the most likely state. In contrast, the Viterbi method only takes into account one path, which may end up representing a minimal fraction of the total probability. At the same time, however, posterior decoding may give an invalid sequence of states! For example, the states identified at time points t and $t+1$ might have zero transition probability between them. As a result, selecting a decoding method is highly dependent on the application of interest.

FAQ

Q: What does it imply when the Viterbi algorithm and Posterior decoding disagree on the path?

A: In a sense, it is simply a reminder that our model of the world can never be correct. In the genomic context, it might be a result of some 'funky' biology; alternative splicing, for instance. In some cases, the Viterbi algorithm will be close to the Posterior decoding while in some others they may disagree.

Seite 10: http://web.mit.edu/6.047/book-2012/Lecture08_HMMSII/Lecture08_HMMSII_standalone.pdf

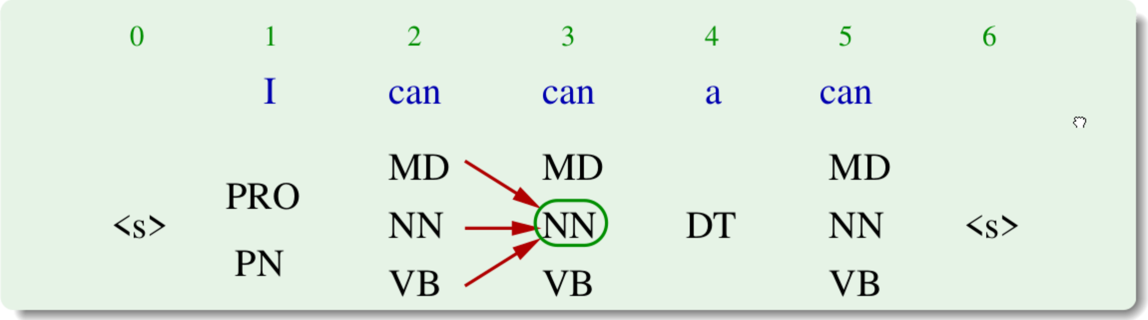
Forward-Backward-Algorithmus

Die (Aposteriori-)Wahrscheinlichkeit $P(t_k = t|w_1^n) = \gamma_t(k)$ der Wortart t an Position k ist die Summe der Wahrscheinlichkeiten aller möglichen Tagfolgen t_1^n mit Tag t an Position k geteilt durch die Gesamtwahrscheinlichkeit aller Tagfolgen.

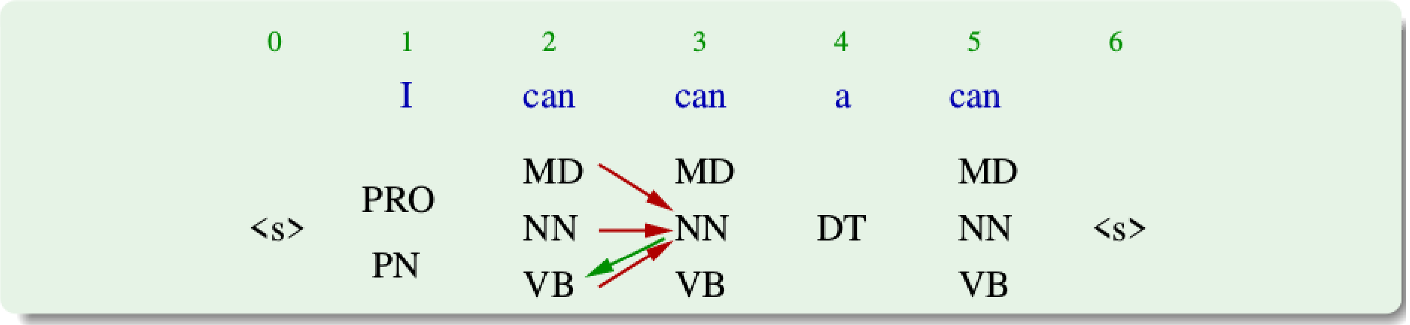
$$\gamma_t(k) = \frac{\sum_{t_1^n : t_k = t} \; p(t_1^n, w_1^n)}{\sum_{t_1^n} \; p(t_1^n, w_1^n)}$$

Sie kann aus den Forward- und Backward-Wahrscheinlichkeiten berechnet werden:

$$\gamma_t(k) = \frac{\alpha_t(k) \; \beta_t(k)}{\alpha_{\langle s \rangle}(n+1)}$$



Beispiel: Viterbi-Algorithmus



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

20.01.21

Könntest du bitte die “Uebung-WS17-18.pdf” und “Uebung-WS16-17-WH.pdf” die Aufgabe 6. aus den Altklausuren lösen? BITTE!!

In “Uebung-WS17-18.pdf” geht es um Perzeptron.

Wir besprechen diese Aufgabe besser nach Übung11 (Perzeptron-Tagger).

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.

Hier ist Pseudocode für die zu implementierende Funktion:

Hier ist Pseudocode für die zu implementierende Funktion:

Viterbitabelle initialisieren
für alle Positionen i von 0 bis zur Länge des Satzes
    für alle möglichen Tags des i-ten Wortes
        für alle Tags in vitprob[i]
            Berechne das Produkt aus lexikalischer Wahrscheinlichkeit,
            Kontextwahrscheinlichkeit und Viterbiwahrscheinlichkeit
            des Vorgängertags und maximiere.

Wie müssten Sie das Programm noch erweitern, um tatsächlich die beste Tagfolge berechnen zu können? (Sie müssen das nicht programmieren.)

(10 Punkte)

Der Lösungvorschlag ist in einer anderen Folie