Les 13 Hidden Markov modellen

In deze les zullen we nader op Hidden Markov modellen ingaan, in het bijzonder op de technieken en algoritmen die bij het omgaan met dit soort modellen belangrijk zijn. Om de notaties helder te hebben, spreken we nu af dat we een Hidden Markov model als volgt beschrijven:

Een Hidden Markov model (vanaf nu afgekort als HMM) λ is gegeven door $\lambda = \lambda(\mathcal{S}, \mathcal{X}, A, B, \pi)$, waarbij de parameters de volgende betekenis hebben:

- $\mathcal{S} = \{S_1, \ldots, S_N\}$ is een verzameling van states;
- $\mathcal{X} = \{x_1, \ldots, x_M\}$ is een verzameling van uitkomsten, die door de states geproduceerd worden;
- $A = (a_{ij})$ is de matrix van overgangskansen tussen de states, d.w.z. $a_{ij} = p(q_t = S_j \mid q_{t-1} = S_i)$ is de kans voor de overgang van state S_i naar state S_j (onafhankelijk van het tijdstip t);
- $B = b_i(k)$ is de matrix van emissiekansen voor de gebeurtenissen vanuit de states, d.w.z. $b_i(k) = p(o_t = x_k \mid q_t = S_i)$ is de kans dat de state S_i de uitkomst x_k produceert (onafhankelijk van t);
- $\pi = (\pi(1), \ldots, \pi(N))$ is de beginverdeling van de states.

Vaak behoren de states en de gebeurtenissen tot de algemene opzet van een probleem, in dit geval staan alleen maar de verschillende kansverdelingen ter discussie. In zo'n geval wordt een HMM iets korter door $\lambda = \lambda(A, B, \pi)$ beschreven.

Er zijn in feite drie fundamentele vragen, waarmee we ons moeten bemoeien:

(1) Gegeven een rij $O = o_1 o_2 \ldots o_T$ van waarnemingen en een HMM $\lambda = \lambda(A, B, \pi)$, hoe vinden we de kans $p(O \mid \lambda)$ op deze waarnemingen, gegeven het model λ? Deze kans kan men ook interpreteren als maat, hoe goed het model bij de waarnemingen past.

(2) Gegeven een rij $O = o_1 o_2 \ldots o_T$ van waarnemingen en een HMM $\lambda = \lambda(A, B, \pi)$, hoe vinden we de rij $q = q_1 q_2 \ldots q_T$ van states die de rij waarnemingen het beste kan verklaren?

(3) Hoe kunnen we de parameters van het HMM $\lambda = (A, B, \pi)$ zo aanpassen dat $p(O \mid \lambda)$ voor een (vaste) rij O van waarnemingen maximaal wordt?

De eerste vraag gaat over het evalueren van een gegeven model op een rij waarnemingen, de tweede over het onthullen van de verborgen states en de derde over het vinden van de parameters van een HMM, zo dat het model goed bij een gegeven rij waarnemingen past. Het laatste noemt men ook het *training* van een HMM. We zullen deze vragen nu apart bekijken.

13.1 Evalueren met behulp van een HMM

Stel we hebben een rij waarnemingen $O = o_1 o_2 \dots o_T$ en een HMM $\lambda = \lambda(A, B, \pi)$ en we willen de kans $p(O \mid \lambda)$ op de rij waarnemingen, gegeven het model, berekenen. Een typische situatie waar men dit probleem tegen komt is de classificatie van de waarneming O. Stel dat verschillende klassen $C_1, \dots, C_r$ door verschillende HMM's $\lambda_1, \dots, \lambda_r$ gekarakteriseerd zijn, dan is het een voor de hand liggende idee de waarneming O aan degene klasse C_k toe te wijzen, waarvoor $p(O \mid \lambda_k)$ maximaal is. Deze aanpak noemt men ook de *maximum likelihood* methode.

Om de kans $p(O \mid \lambda)$ te berekenen moeten we in principe voor elke rij $q = q_1 q_2 \dots q_T \in \mathcal{S}^T$ van states de kans $p(O, q \mid \lambda)$ berekenen en deze kansen voor alle mogelijke rijen q van states bij elkaar optellen. Volgens de definitie van de voorwaardelijke kans geldt

$$p(O, q \mid \lambda) = p(O \mid q, \lambda) \cdot p(q \mid \lambda)$$

en dus

$$p(O \mid \lambda) = \sum_{q \in \mathcal{S}^T} p(O, q \mid \lambda) = \sum_{q \in \mathcal{S}^T} p(O \mid q, \lambda)\, p(q \mid \lambda).$$

Met behulp van de laatste uitdrukking kunnen we de kans $p(O \mid \lambda)$ inderdaad uitrekenen: Aan de ene kant is $p(q \mid \lambda)$ juist het product van de kansen voor de overgangen tussen de states in de rij $q = q_1 q_2 \dots q_T$, dus

$$p(q \mid \lambda) = \pi(q_1) \cdot \prod_{t=1}^{T-1} a_{q_t q_{t+1}}.$$

Aan de andere kant is voor een gegeven rij van states de kans $p(O \mid q, \lambda)$ het product van de emissiekansen van de enkele states, dus

$$p(O \mid q, \lambda) = \prod_{t=1}^{T} b_{q_t}(o_t).$$

Bij elkaar genomen krijgen we zo:

$$\begin{aligned} p(O \mid \lambda) &= \sum_{q = q_1 \dots q_T} \pi(q_1) b_{q_1}(o_1) \prod_{t=1}^{T-1} a_{q_t q_{t+1}} b_{q_{t+1}}(o_{t+1}) \\ &= \sum_{q = q_1 \dots q_T} \pi(q_1) b_{q_1}(o_1) a_{q_1 q_2} b_{q_2}(o_2) \dots a_{q_{T-1} q_T} b_{q_T}(o_T). \end{aligned}$$

Het probleem hierbij is, dat we voor een rij van lengte T over N^T mogelijke rijen van states moeten lopen, en dit is al voor kleine waarden van T (bijvoorbeeld $T = 100$) ondoenlijk.

Gelukkig kunnen we het vermijden over alle mogelijke rijen van states te lopen. Bij de brute kracht methode zouden we erg veel dingen herhaaldelijk uitrekenen, namelijk de beginstukken van de rijen waarvoor de eerste t states hetzelfde zijn. Het idee is, de kansen voor de beginstukken stapsgewijs te

berekenen en deze te recyclen. Als we namelijk de kans voor het beginstuk $o_1 o_2 \dots o_t$ al kennen, zijn er maar N mogelijkheden voor de state waarin het systeem op tijdstip t zit, en voor de voortzetting naar o_{t+1} hoeven we alleen maar de overgangen van deze N mogelijkheden naar de N mogelijke states op tijdstip $t+1$ te berekenen. Zo krijgen we slechts $T \cdot N^2$ waarden, die we moeten berekenen. De procedure die we zo net hebben geschetst is zo belangrijk dat ze een eigen naam heeft (ook al is die niet erg karakteristiek), ze heet *forward algoritme*.

Forward algoritme

We willen voor $O = o_1 o_2 \dots o_T$ de kans $p(O \mid \lambda)$ berekenen. Hiervoor definiëren we de *vooruitkans*

$$\alpha_t(i) := p(o_1 o_2 \dots o_t, q_t = S_i \mid \lambda),$$

die de kans aangeeft dat het systeem op tijdstip t in state S_i is en tot dit tijdstip de waarnemingen $o_1, \dots, o_t$ heeft geproduceerd.

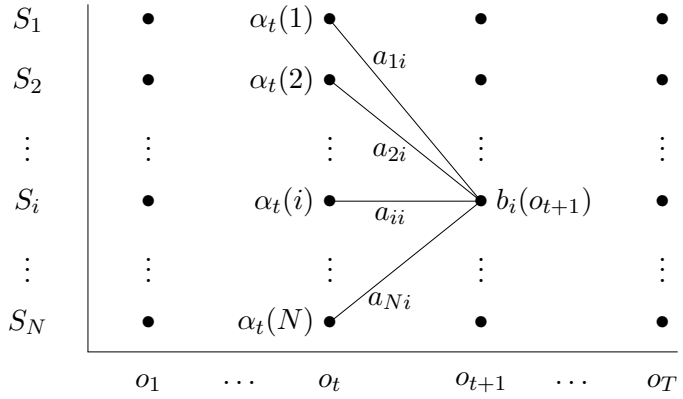
Voor $t = 1$ laten zich de vooruitkansen $\alpha_1(i)$ heel eenvoudig berekenen, er geldt

$$\alpha_1(i) = \pi(i) b_i(o_1).$$

Als we nu van tijdstip t naar tijdstip $t+1$ willen, moeten we over alle N states waarin het systeem op tijdstip t kan zijn lopen en de kans op de overgang naar de verschillende states op tijdstip $t+1$ en de emissie van waarneming o_{t+1} berekenen. Dit geeft de recursie formule:

$$\alpha_{t+1}(i) = \left(\sum_{k=1}^{N} \alpha_t(k) a_{ki} \right) b_i(o_{t+1}).$$

In Figuur III.5 is de berekening van $\alpha_{t+1}(i)$ in een schema aangegeven: De kansen $\alpha_t(1), \dots, \alpha_t(N)$ van de voorafgaande stap worden met de overgangskansen $a_{1i}, \dots, a_{Ni}$ en de emissiekans $b_i(o_{t+1})$ gecombineerd tot de kans $\alpha_{t+1}(i)$.



Figuur III.5: Berekening van $\alpha_{t+1}(i)$ in het *forward algoritme*.

Als we de vooruitkansen $\alpha_t(i)$ voor $t = 1, 2, \dots, T$ berekenen, hoeven we in de laatste stap alleen maar nog de kansen voor de N verschillende states op

tijdstip $t = T$ op te tellen want omdat het systeem in een van de states moet zijn, geeft dit juist de kans op de volledige rij waarnemingen aan. Op deze manier krijgen we

$$p(o_1 o_2 \dots o_T \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

Backward algoritme

Het zou geen verrassing zijn dat er behalve van een forward algoritme ook een *backward algoritme* bestaat, waarbij de kansen op een deel van de waarnemingen van het einde af berekend worden. Men definieert de *achteruitkans* $\beta_t(i)$ als de voorwaardelijke kans

$$\beta_t(i) := p(o_{t+1} \dots o_T \mid q_t = i, \lambda)$$

op de laatste $T-t$ waarnemingen $o_{t+1}, \dots, o_T$, gegeven het feit dat het systeem op tijdstip t in state S_i was.

In dit geval heeft men de initialisering $\beta_T(i) = 1$ want we veronderstellen dat het systeem op tijdstip T in state S_i is.

Om van het tijdstip $t+1$ naar t terug te komen, moeten we over alle states lopen waarin het systeem op tijdstip $t+1$ kan zijn en de overgangen en de emissie van de waarneming o_{t+1} vanuit deze states berekenen. Dit geeft de recursie

$$\beta_t(i) = \sum_{k=1}^{N} a_{ik} b_k(o_{t+1}) \beta_{t+1}(k).$$

Door deze recursie voor $t = T-1, \dots, 2, 1$ te doorlopen, krijgen we uiteindelijk de kans $p(O \mid \lambda)$ door

$$p(O \mid \lambda) = \sum_{i=1}^{N} \pi(i) b_i(o_1) \beta_1(i).$$

We zullen de vooruitkansen $\alpha_t(i)$ en de achteruitkansen $\beta_t(i)$ later in deze les nog eens tegenkomen. Door een slimme combinatie van de $\alpha_t(i)$ en $\beta_t(i)$ laten zich namelijk de parameters van een HMM zo verbeteren dat het systeem een hogere kans voor een gegeven rij waarnemingen oplevert. Op deze manier wordt het HMM beter aan de waarnemingen aangepast, dus getraind.

> De combinatie van vooruit- en achteruitkansen speelt ook bij problemen een rol, waar snel een kandidaat voor een rij states met hoge kans gevonden moet worden. Het idee is, tegelijkertijd aan het begin en aan het eind te beginnen, tot dat $\alpha_t(i)$ en $\beta_t(i)$ in het midden op elkaar stoten. Daarbij worden alleen maar de meestbelovende trajecten meegenomen, d.w.z. de states op tijdstip t die de hoogste vooruit- en achteruitkansen hebben. Deze manier om de *zoekruimte* snel tot de interessante states te beperken staat bekend onder de naam *beam-search*.

13.2 States onthullen

Vaak is het niet genoeg de kans voor een rij waarnemingen, gegeven een HMM, te bepalen, men wil ook een rij states bepalen die bij de waarnemingen past. Maar omdat er verschillende rijen states zijn, die een rij waarnemingen kunnen produceren, moet men hier een criterium hebben, welke states het beste passen. Voor dat we erover na kunnen denken hoe we een optimale rij states kunnen vinden, moeten we dus eerst definiëren, wat we met de *optimale rij states* bij een rij waarnemingen überhaupt bedoelen,

Helaas is er geen *juiste* manier, om een optimaliteitscriterium te definiëren, en afhankelijk van het probleem worden ook verschillende criteria gehanteerd.

Een mogelijkheid is bijvoorbeeld, op elke tijdstip t de state $q_t = S_i$ te kiezen die op dit tijdstip optimaal is. Dat wil zeggen we kiezen q_t zo dat $p(O, q_t = S_i \mid \lambda)$ maximaal wordt. Merk op dat we dit met behulp van de vooruit- en achteruitkansen keurig kunnen formuleren, er geldt namelijk dat

$$p(O, q_t = S_i \mid \lambda) = \alpha_t(i)\beta_t(i)$$

en we hoeven dus voor q_t alleen maar de state S_i te kiezen waarvoor $\alpha_t(i)\beta_t(i)$ maximaal wordt.

Soms willen (of kunnen) we voor de state q_t op tijdstip t alleen maar de waarnemingen $o_1 \dots o_t$ tot op dit tijdstip gebruiken, bijvoorbeeld in een *real-time* systeem. In dit geval zouden we de state $q_t = S_i$ zo kunnen kiezen, dat $p(o_1 o_2 \dots o_t, q_t = S_i \mid \lambda)$ maximaal wordt. Maar dit betekent, dat we voor q_t de state S_i kiezen, waarvoor $\alpha_t(i)$ maximaal is, want dit is precies de definitie van de vooruitkans.

Een probleem bij de genoemde criteria is, dat de overgangen tussen de states enigszins buiten beschouwing blijven, en we zo misschien zelfs een rij van states krijgen die een *verboden* overgang bevat, dus een overgang met kans 0.

Het meest gebruikte criterium dat dit probleem voorkomt is, de optimale rij q_{opt} van states te definiëren als de rij waarvoor de gemeenschappelijke kans over de hele rij states en waarnemingen maximaal is.

Criterium: We noemen een rij $q_{opt} = q_1 q_2 \dots q_T \in \mathcal{S}^T$ van states *optimaal* voor de waarneming $O = o_1 o_2 \dots o_T$ als

$$p(O, q_{opt} \mid \lambda) \geq p(O, q \mid \lambda) \text{ voor alle } q \in \mathcal{S}^T.$$

We staan nu weer voor het probleem dat we in principe de kans $p(O, q \mid \lambda)$ voor alle rijen q van states moeten berekenen. Anders als bij het berekenen van de kans voor de waarneming mogen we nu niet alle mogelijkheden om tot een tussenpunt te komen bij elkaar optellen, dus helpen de vooruitkansen $\alpha_t(i)$ hier niet verder.

Maar een kleine variatie van het forward algoritme geeft ook hier een oplossing, waarbij we niet alle N^T mogelijke rijen moeten bekijken. Het idee wat hier achter zit komt uit het *dynamische programmeren* en is een bijna vanzelfsprekende opmerking, maar is wel zo fundamenteel, dat het de naam *Bellman's principe* draagt.

Bellman's principe

We bekijken een iets algemenere situatie die uit het dynamische programmeren ontleend is: Stel we hebben een rooster met punten (i, j) voor $0 \leq i \leq N$, $0 \leq j \leq M$, en we zijn op zoek naar en pad van $(0, 0)$ naar (N, M). Met elke overgang van een punt naar een andere zijn kosten verbonden, die we als *afstanden* tussen de punten zien, daarbij noteren we de kosten voor de overgang van (i', j') naar (i, j) met $d((i', j'), (i, j))$. Sommige van de kosten kunnen oneindig zijn, om uit te drukken dat deze overgang onmogelijk is.

Voor elk punt (i, j) noemt men de punten (i', j') waarvoor de overgang van (i', j') naar (i, j) mogelijk is (d.w.z. eindige kosten heeft) de *mogelijke voorgangers* een het stelsel van mogelijke voorgangers noemt men de *lokale beperkingen*. In sommige toepassingen kan men bijvoorbeeld alleen maar van $(i - 1, j - 1)$, $(i - 1, j)$ of $(i, j - 1)$ naar (i, j) komen, in andere gevallen zijn alle punten $(i - 1, j')$ mogelijke voorgangers van (i, j). Dit is bijvoorbeeld het geval als de eerste coördinaat tijdstippen en de tweede states aangeeft en we veronderstellen dat we van elke state naar elke andere state kunnen komen.

Het optimale pad van $(0, 0)$ naar (N, M) is natuurlijk het pad waarvoor de som van de kosten van de overgangen minimaal is. Bellman's principe zegt nu het volgende:

Bellman's principe: *Als het optimale pad van* $(0, 0)$ *naar* (N, M) *door het punt* (i, j) *loopt, dan is ook het deelpad van* $(0, 0)$ *tot* (i, j) *een optimaal pad tussen deze twee punten, net als het deelpad van* (i, j) *naar* (N, M) *een optimaal pad tussen deze twee punten is.*

Hier zit alleen maar de vanzelfsprekende opmerking achter dat we de kosten voor het pad van $(0, 0)$ via (i, j) naar (N, M) nog kunnen reduceren, als we de kosten voor een van de deelpaden tussen $(0, 0)$ en (i, j) of tussen (i, j) en (N, M) kunnen reduceren.

Maar als gevolg van Bellman's principe krijgen we een efficiënte manier om het optimale pad te vinden. We moeten (afhankelijk van de lokale beperkingen) stapsgewijs de optimale paden voor de punten (i, j) bepalen, door voor elke mogelijke voorganger (i', j') van (i, j) de kosten voor het optimale pad naar (i', j') bij de kosten voor de overgang van (i', j') naar (i, j) op te tellen en het minimum van deze kosten te kiezen.

Viterbi algoritme

Als we Bellman's principe op het probleem van de optimale rij van states van een HMM toepassen, krijgen we het *Viterbi algoritme*. Bellman's principe zegt in dit geval dat voor de optimale rij $q = q_1 q_2 \dots q_T$ van states voor de waarneming $O = o_1 o_2 \dots o_T$ ook de deelrijen tot en vanaf tijdstip t optimaal zijn, dus $p(o_1 \dots o_t, q_1 \dots q_t \mid \lambda)$ is maximaal en $p(o_t \dots o_T, q_t \dots q_T \mid \lambda)$ is maximaal.

In de opzet van het dynamische programmeren hebben we als roosterpunten de paren (t, i) die aangeven dat $q_t = S_i$ is. Hierbij beginnen we met het (formele) punt $(0, 0)$ en eindigen in een punt (T, i), waarbij we geen beperking op i opleggen. De mogelijke voorgangers van (t, i) zijn $(t - 1, k)$ voor alle

$1 \leq k \leq N$. In plaats van kosten praten we nu over kansen, en natuurlijk willen we voor de kansen niet het minimum maar het maximum vinden. De kans die bij de overgang van $(t-1,k)$ naar (t,i) hoort, is de overgangskans a_{ki} van state S_k naar state S_i en de kans $b_i(o_t)$ om in state S_i op tijdstip t de waarneming o_t te produceren. De totale kans voor de overgang $(t-1,k) \to (t,i)$ is dus

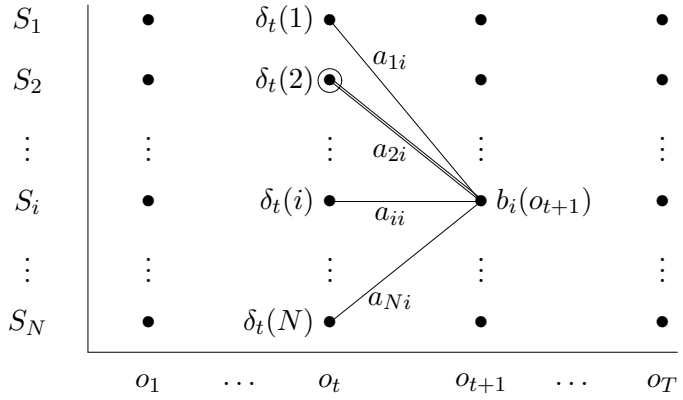
$$p((t-1,k) \to (t,i)) = a_{ki} \cdot b_i(o_t).$$

We definiëren nu $\delta_t(i)$ als de kans van de optimale rij van states voor de deelwaarneming $o_1 o_2 \ldots o_t$, die op tijdstip t in state S_i is.

We krijgen zo de recursie

$$\delta_1(i) = \pi(i) b_i(o_1) \quad \text{en} \quad \delta_{t+1}(i) = \left(\max_{1 \leq k \leq N} \delta_t(k) a_{ki} \right) b_i(o_{t+1})$$

die sterk op de recursie bij het forward algoritme lijkt. Het enige verschil is, dat in plaats van de som over de alle voorgangers nu het maximum over de voorgangers genomen wordt. Maar het schema van het Viterbi algoritme is zo als in Figuur III.6 te zien precies hetzelfde als bij het forward algoritme. Aanvullend moeten we wel bij elke punt (t,i) nog opslaan, vanuit welke voorganger $(t-1,k)$ het maximum bereikt werd, om uiteindelijk het optimale pad terug te kunnen vinden. Dit wordt meestal door een geschakelde lijst geïmplementeerd, in Figuur III.6 is als voorbeeld de overgang $(t,2) \to (t+1,i)$ benadrukt.



Figuur III.6: Berekening van $\delta_{t+1}(i)$ in het Viterbi algoritme.

Om meer efficiëntie bij het evalueren van een rij waarnemingen met verschillende HMM's te bereiken, wordt soms de evaluatie van de kans met behulp van vooruit- of achteruitkansen vervangen door de zogeheten *Viterbi benadering*. Hierbij wordt in plaats van de som over de kansen voor *alle* paden alleen maar de kans voor het beste pad bepaald (en dit natuurlijk met behulp van het Viterbi algoritme). Het idee hierachter is dat bij het evalueren uiteindelijk toch maar heel weinig paden een substantiële bijdrage aan de totale kans leveren en dat de som over de kansen voor hetgeen HMM maximaal wordt waarvoor het optimale pad de hoogste kans heeft.

Er valt nog iets over de implementatie van het Viterbi algoritme (en andere algoritmen in het kader van probabilistische modellen) op te merken:

Omdat er steeds kansen met elkaar vermenigvuldigd worden en deze soms zelf al redelijk klein zijn, worden de waarden van de $\delta_t(i)$ snel erg klein en dalen al gauw onder de rekennauwkeurigheid van een computer. Voor dit probleem bestaat er een heel simpele oplossing: Men rekent met de logaritmen van de kansen.

Merk op: Omdat de logaritme een monotone functie is, is $\delta_t(i)$ maximaal voor de i waarvoor $\tilde{\delta}_t(i) := -\log(\delta_t(i))$ minimaal is.

Als we het Viterbi algoritme op de logaritmen $\tilde{\delta}_t(i) = -\log(\delta_t(i))$ transformeren, krijgen we:

$$\begin{aligned}\tilde{\delta}_1(i) &= -\log(\pi(i)) - \log(b_i(o_1));\\ \tilde{\delta}_{t+1}(i) &= \min_{1\le k\le N}\Big(\tilde{\delta}_t(k) - \log(a_{ki})\Big) - \log(b_i(o_{t+1})).\end{aligned}$$

Natuurlijk worden de logaritmen van de a_{ij} en $b_i(k)$ niet steeds opnieuw berekend, maar ze worden bij het HMM opgeslaan.

> Een soortgelijke opmerking geldt natuurlijk ook voor het forward algoritme. Daarbij is er echter het probleem, dat de kansen voor de verschillende paden bij elkaar opgeteld moeten worden. Dit lost men soms met behulp van de formule $\log(p+q) = \log(p(1+\frac{q}{p})) = \log(p)+\log(1+\frac{q}{p}) = \log(p) + \log(1+e^{\log(q)-\log(p)})$ op, maar vaak wordt hier inderdaad met kansen gerekend die op een geschikte manier geschaald worden.
>
> Ook dit probleem wordt vermeden, als men bij het evalueren het forward algoritme door de Viterbi benadering vervangt.

Toepassing van het Viterbi algoritme

We kijken nu naar de toepassing van het Viterbi algoritme op een HMM met de drie munten, waarvan maar één eerlijk is. De drie munten zijn de drie states S_1, S_2, S_3 en de mogelijke uitkomsten zijn $x_1 = \texttt{K}$ voor *kop* en $x_2 = \texttt{M}$ voor *munt*. Het HMM $\lambda = \lambda(A, B, \pi)$ is gegeven door

$$A = (a_{ij}) := \begin{pmatrix} 0.6 & 0.2 & 0.2\\ 0.4 & 0.2 & 0.4\\ 0.4 & 0.4 & 0.2\end{pmatrix},\quad B = (b_i(k)) := \begin{pmatrix} 0.5 & 0.5\\ 0.75 & 0.25\\ 0.25 & 0.75\end{pmatrix},\quad \pi = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}).$$

We bekijken de waarneming $O = \texttt{KMKMM}$.

Voor de initialisering hebben we:

$$\begin{aligned}\delta_1(1) &= \pi(1)b_1(1) = 0.33 \cdot 0.5 = 0.167,\\ \delta_1(2) &= \pi(2)b_2(1) = 0.33 \cdot 0.75 = 0.25,\\ \delta_1(3) &= \pi(3)b_3(1) = 0.33 \cdot 0.25 = 0.083.\end{aligned}$$

Voor de volgende stap berekenen we nu

$$\begin{aligned}
i=1: \delta_1(1)a_{11}b_1(2) &= 0.167\cdot 0.6\cdot 0.5 = 0.05, \leftarrow \max\\
\delta_1(2)a_{21}b_1(2) &= 0.25\cdot 0.4\cdot 0.5 = 0.05,\\
\delta_1(3)a_{31}b_1(2) &= 0.083\cdot 0.4\cdot 0.5 = 0.0167,\\
i=2: \delta_1(1)a_{12}b_2(2) &= 0.167\cdot 0.2\cdot 0.25 = 0.0083,\\
\delta_1(2)a_{22}b_2(2) &= 0.25\cdot 0.2\cdot 0.25 = 0.0125, \leftarrow \max\\
\delta_1(3)a_{32}b_2(2) &= 0.083\cdot 0.4\cdot 0.25 = 0.0083,\\
i=3: \delta_1(1)a_{13}b_3(2) &= 0.167\cdot 0.2\cdot 0.75 = 0.025,\\
\delta_1(2)a_{23}b_3(2) &= 0.25\cdot 0.4\cdot 0.75 = 0.075, \leftarrow \max\\
\delta_1(3)a_{33}b_3(2) &= 0.083\cdot 0.2\cdot 0.75 = 0.0125.
\end{aligned}$$

Dit geeft voor de $\delta_2(i)$ het volgende:

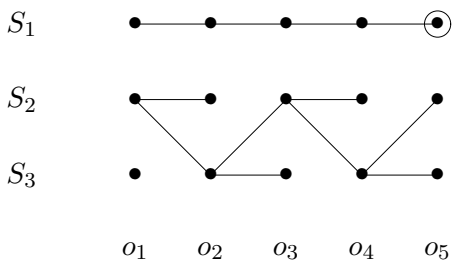
$$\begin{aligned}
&\delta_2(1) = 0.05 \text{ met } k=1 \text{ (of } k=2) \text{ als voorganger},\\
&\delta_2(2) = 0.0125 \text{ met } k=2 \text{ als voorganger},\\
&\delta_2(3) = 0.075 \text{ met } k=2 \text{ als voorganger}.
\end{aligned}$$

Als we zo doorgaan krijgen we voor $\delta_t(i)$ met de voorgangers k:

$$\begin{aligned}
&\delta_3(1) = 0.015, k=1, \quad \delta_3(2) = 0.0225, k=3, \quad \delta_3(3) = 0.00375, k=3,\\
&\delta_4(1) = 0.0045, k=1, \ \delta_4(2) = 0.001125, k=2, \ \delta_4(3) = 0.00675, k=2,\\
&\delta_5(1) = 0.00135, k=1, \delta_5(2) = 0.000675, k=3, \delta_5(3) = 0.0010125, k=3.
\end{aligned}$$

We zien dat $\delta_5(1)$ het maximum van de $\delta_5(i)$ is, daarom eindigt de optimale rij van states in state S_1. Omdat in alle stappen de state S_1 voorganger S_1 heeft, is dus $S_1S_1S_1S_1S_1$ de optimale rij van states. Merk op dat tot $t=4$ de rij $S_2S_3S_2S_3$ optimaal was geweest.

Als we de punten (t,i) als punten van een tralie (of rooster) bekijken en het punt (t,i) met degene voorganger $(t-1,k)$ verbinden die de maximale waarde van $\delta_t(i)$ oplevert, kunnen we hieruit de optimale rij van states makkelijk achterhalen. In Figuur III.7 is dit tralie voor het net besproken voorbeeld te zien, waarbij de optimale eindstate door een extra cirkel benadrukt is.



Figuur III.7: Tralie voor het Viterbi algoritme.

13.3 Training van een HMM

Tot nu toe zijn we ervan uit gegaan dat we de parameters van het HMM al kennen. De vraag is nu, hoe we de parameters $A = (a_{ij})$, $B = (b_i(k))$ en $\pi = (\pi(1), \ldots, \pi(N))$ zo kunnen bepalen, dat het model een gegeven rij $O = o_1 o_2 \ldots o_T$ van waarnemingen zo goed mogelijk beschrijft, dus zo dat de kans $p(O \mid \lambda(A, B, \pi))$ maximaal wordt. Omdat bij deze aanpak de kans gemaximaliseerd wordt, noemt men dit ook de *maximum likelihood schatting* van de parameters.

In Wiskunde 1 hebben we in het kader van de kansrekening naar een soortgelijk, maar veel eenvoudiger probleem gekeken. We wilden toen de parameters van een kansverdeling, bijvoorbeeld een normale verdeling, zo bepalen, dat met deze parameters de kans voor een rij gebeurtenissen maximaal werd. Het idee was toen, de (logaritme van de) kans op de gebeurtenissen als functie van de parameters te interpreteren en een maximum van deze functie te bepalen door de partiële afgeleiden naar de parameters gelijk aan 0 te zetten en deze vergelijkingen op te lossen. Bij de normale verdeling hebben we zo bijvoorbeeld geconcludeerd, dat de beste keuze voor de verwachtingswaarde μ van de normale verdeling het gemiddelde van de gebeurtenissen is – een niet echt verrassend resultaat.

In principe zouden we bij de HMM's een analoge aanpak kunnen kiezen: We schrijven $p(O \mid \lambda)$ als functie van de parameters a_{ij}, $b_i(k)$ en $\pi(i)$, zo als we dat in het begin van deze les al hebben gedaan, dus als

$$p(O \mid \lambda) = \sum_{q=q_1 \ldots q_T} \pi(q_1) b_{q_1}(o_1) a_{q_1 q_2} b_{q_2}(o_2) \ldots a_{q_{T-1} q_T} b_{q_T}(o_T).$$

Vervolgens bepalen we de partiële afgeleiden naar de parameters en proberen de vergelijkingen

$$\frac{\partial}{\partial a_{ij}} p(O \mid \lambda) = 0, \quad \frac{\partial}{\partial b_i(k)} p(O \mid \lambda) = 0, \quad \frac{\partial}{\partial \pi(i)} p(O \mid \lambda) = 0$$

simultaan op te lossen. Dat we eigenlijk nog moeten eisen dat de rijen van de matrices A en B de som 1 hebben, omdat we het over kansverdelingen hebben, vergeten we hierbij even.

Het probleem is dat in alle praktische gevallen het stelsel vergelijkingen dat men zo krijgt niet analytisch oplosbaar is. Maar dit probleem doet zich ook al in veel eenvoudigere vraagstukken voor, want ook bij gewone functies van één veranderlijke kunnen we vaak de nulpunten niet expliciet bepalen. De gebruikelijke manier, om in deze situatie verder te komen, is een *numerieke benaderingsmethode* toe te passen.

Het idee in het kader van HMM's is, startwaarden voor de parameters A, B en π te gokken en vervolgens de parameters stapsgewijs zo aan te passen, dat in elke stap de *likelihood* $p(O \mid \lambda(A, B, \pi))$ toeneemt.

In het algemeen levert zo'n benaderingsmethode alleen maar een lokaal maximum van de functie $p(O \mid \lambda)$ op, en omdat deze functie zo ingewikkeld

is, is er ook geen goede manier om een globaal maximum te vinden. In de praktijk probeert men een paar verschillende stelsels van startwaarden en kiest vervolgens het beste van de gevonden lokale maxima.

Baum-Welch algoritme

We zullen nu een speciale benaderingsmethode bekijken, die de parameters van een HMM stapsgewijs verbetert, namelijk het *Baum-Welch algoritme*. Deze gebruikt de vooruit- en achteruitkansen $\alpha_t(i)$ en $\beta_t(i)$ die we al bij de evaluatie van het HMM tegen gekomen zijn.

Om de methode goed te kunnen formuleren, hebben we eerst nog twee nieuwe uitdrukkingen nodig, die zekere kansen beschrijven:

De voorwaardelijke kans dat het systeem op tijdstip t in state S_i is, gegeven de volledige rij waarnemingen $O = o_1 o_2 \dots o_T$, noemen we $\gamma_t(i)$. Volgens de relatie $p(A \mid B) = \frac{p(A,B)}{p(B)}$ geldt dan:

$$\gamma_t(i) := p(q_t = S_i \mid O, \lambda) = \frac{p(O, q_t = S_i \mid \lambda)}{p(O \mid \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}.$$

Verder definiëren we als $\xi_t(i,j)$ de voorwaardelijke kans dat het systeem tussen de tijdstippen t en $t+1$ van state S_i naar state S_j gaat, gegeven de rij O van waarnemingen. Er geldt

$$\begin{aligned}\xi_t(i,j) := p(q_t = S_i, q_{t+1} = S_j \mid O, \lambda) &= \frac{p(O, q_t = S_i, q_{t+1} = S_j \mid \lambda)}{p(O \mid \lambda)} \\ &= \frac{\alpha_t(i)\ a_{ij}\ b_j(o_{t+1})\ \beta_{t+1}(j)}{p(O \mid \lambda)}.\end{aligned}$$

Tussen de kansen $\xi_t(i,j)$ en $\gamma_t(i)$ bestaat een eenvoudige relatie, want de kans om op tijdstip t in state S_i te zijn is de som over alle j van de kansen, tussen de tijdstippen t en $t+1$ van state S_i naar S_j te gaan. Er geldt dus

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j).$$

Als we nu de kansen $\gamma_t(i)$ over de tijdstippen $t = 1, \dots, T$ optellen, krijgen we het verwachte aantal van waarnemingen die door de state S_i geproduceerd zijn. Net zo kunnen we de kansen $\xi_t(i,j)$ over de tijdstippen $t = 1, \dots, T-1$ optellen, dan krijgen we het verwachte aantal overgangen van state S_i naar state S_j. We hebben dus

$$\begin{aligned}\sum_{t=1}^{T} \gamma_t(i) &= \text{ verwacht aantal emissies vanuit state } S_i; \\ \sum_{t=1}^{T-1} \xi_t(i,j) &= \text{ verwacht aantal overgangen tussen states } S_i \text{ en } S_j.\end{aligned}$$

Maar aan de hand van deze gegevens kunnen we nieuwe parameters A', B' en π' als relatieve frequenties schatten, namelijk door:

$$\begin{aligned}
\pi'(i) &= \text{ verwachte kans op state } S_i \text{ op tijdstip 1} \\
&= \gamma_1(i) = \frac{\alpha_1(i)\ \beta_1(i)}{\sum_{i=1}^{N} \alpha_t(i)\ \beta_t(i)} = \frac{\alpha_1(i)\ \beta_1(i)}{\sum_{i=1}^{N} \alpha_T(i)} \\
a'_{ij} &= \frac{\text{verwacht aantal overgangen van state } S_i \text{ naar state } S_j}{\text{verwacht aantal overgangen vanuit state } S_i} \\
&= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} = \frac{\sum_{t=1}^{T-1} \alpha_t(i)\ a_{ij}\ b_j(o_{t+1})\ \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)} \\
b'_i(k) &= \frac{\text{verwacht aantal emissies vanuit state } S_i \text{ met waarneming } x_k}{\text{verwacht aantal emissies vanuit state } S_i} \\
&= \frac{\sum_{t=1,o_t=x_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} = \frac{\sum_{t=1,o_t=x_k}^{T} \alpha_t(i)\ \beta_t(i)}{\sum_{t=1}^{T} \alpha_t(i)\ \beta_t(i)}
\end{aligned}$$

Merk op hoe de waarneming $O = o_1 o_2 \dots o_T$ bij de berekening van a'_{ij} en $b'_i(k)$ betrokken is, dit is uiteindelijk de reden dat de parameters aan de waarneming aangepast worden.

De grap is nu, dat we met de nieuwe parameters $A' = (a'_{ij})$, $B' = (b'_i(k))$ en $\pi' = (\pi'(1), \dots \pi'(N))$ steeds een *beter* model voor de beschrijving van O krijgen dan met de oude parameters A, B en π, er laat zich namelijk aantonen dat geldt:

$$\lambda' = \lambda(A', B', \pi') \Rightarrow p(O \mid \lambda') \geq p(O \mid \lambda).$$

We kunnen nu de herschatting van de parameters itereren door het nieuwe model $\lambda(A', B', \pi')$ te gebruiken om de vooruit- en achteruitkansen $\alpha_t(i)$ en $\beta_t(i)$ en de kansen $\gamma_t(i)$ en $\xi_t(i,j)$ opnieuw te bepalen en hieruit een verder verbeterd stelsel parameters te verkrijgen. Deze procedure wordt herhaald tot dat de likelihood $p(O \mid \lambda)$ niet meer veranderd of een maximaal aantal iteratie stappen bereikt is.

13.4 Toegift: Levenshtein afstand

Als toegift behandelen we de toepassing van Bellman's principe op een ander belangrijk probleem in de patroonherkenning, namelijk de afstand tussen strings. Dit heeft toepassingen in de verwerking en herkenning van teksten en taal, maar ook in de beeldherkenning.

Een string is hierbij algemeen een keten van symbolen en men wil een afstand tussen twee ketens kunnen berekenen. Bij teksten zijn de symbolen gewoon letters, in de spraakherkenning zijn de symbolen vaak woorden, maar kunnen ook grammaticale etiketten zijn. In de beeldherkenning wordt vaak de omtrek van een element als keten van zekere elementaire symbolen beschreven, lijnstukken, hoeken etc.

Een mogelijke definitie van de afstand tussen twee strings is de *Edit afstand* die naar een van de uitvinders nu meestal *Levenshtein afstand* heet. Het

idee hierbij is, door *elementaire edit operaties* de ene string in de andere te transformeren, waarbij elementaire operaties de volgende zijn:

- vervangen (substitution) van een symbool, bijvoorbeeld *ki***j***ker* → *ki***k***ker*;
- invoegen (insertion) van een symbool, bijvoorbeeld *bouwer* → *b***r***ouwer*.
- weglaten (deletion) van een symbool, bijvoorbeeld *koe***k** → *koe*;

Natuurlijk zijn er verschillende manieren, om van een string door een combinatie van vervangen, invoegen en weglaten naar een andere string te komen, maar het is voor de hand liggend het minimale aantal stappen als edit afstand tussen de strings te definiëren:

Definitie: De *Levenshtein afstand* tussen twee strings is gedefinieerd als het *minimale aantal* van elementaire edit operaties waarmee de eerste string in de tweede string getransformeerd kan worden.

De vraag is nu hoe men het minimale aantal operaties vindt. Dit gebeurt analoog met het Viterbi algoritme door de methode van het dynamische programmeren.

Het idee is dat men voor twee strings $X = x_1x_2 \dots x_N$ en $Y = y_1y_2 \dots y_M$ stapsgewijs kijkt hoe men beginstukken van de twee strings in elkaar kan transformeren. Volgens Bellman's principe hoeft men hierbij alleen maar het minimale aantal operaties op te slaan om van het beginstuk $x_1 \dots x_i$ van lengte i van X naar het beginstuk $y_1 \dots y_j$ van lengte j van Y te komen. Men krijgt zo een rooster van punten (i, j) voor $0 \leq i \leq N$ en $0 \leq j \leq M$ waarbij we het aantal edit operaties als kosten voor de overgang tussen twee punten interpreteren. In dit geval hebben we (tegenover het Viterbi algoritme) sterke lokale beperkingen, want het punt (i, j) heeft slechts drie mogelijke voorgangers:

(1) het punt $(i-1, j-1)$: In het geval $x_i = y_j$ heeft de overgang van $(i-1, j-1)$ naar (i, j) kosten 0, anders kosten 1. Als $x_i \neq y_j$ is deze overgang namelijk het vervangen van x_i door y_j.

(2) het punt $(i, j-1)$: Deze overgang is het invoegen van het symbool y_j en heeft de kosten 1.

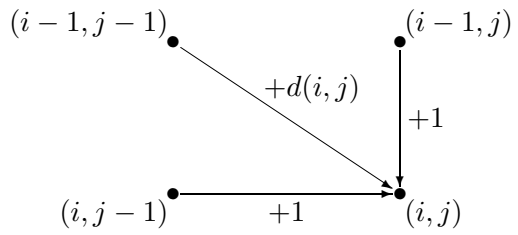
(3) het punt $(i-1, j)$: Deze overgang is het weglaten van het symbool x_i en heeft de kosten 1.

In Figuur III.8 zijn deze overgangen schematisch te zien, waarbij we met $d(i, j)$ de kosten voor het vervangen van x_i door y_j aangeven, hiervoor geldt

$$d(i,j) := \begin{cases} 0 & \text{als} \quad x_i = y_j \\ 1 & \text{als} \quad x_i \neq y_j. \end{cases}$$

Volgens Bellman's principe vinden we de minimale kosten $D(i, j)$ voor de transformatie van het beginstuk $x_1 \dots x_i$ van X naar het beginstuk $y_1 \dots y_j$ van Y als volgt:

We initialiseren $D(i, 0) := i$ voor $0 \leq i \leq N$ (dit is het weglaten van de eerste i symbolen van X) en $D(0, j) := j$ voor $0 \leq j \leq M$ (dit is het invoegen



Figuur III.8: Mogelijke voorgangers van (i,j).

van de eerste j symbolen van Y) en berekenen vervolgens voor $i = 1, 2, \ldots N$ en voor $j = 1, 2, \ldots M$:

$$D(i,j) := \min\{D(i-1,j-1) + d(i,j),\ D(i,j-1) + 1,\ D(i-1,j) + 1\}.$$

Merk op dat op het moment dat we $D(i,j)$ willen berekenen de waarden van $D(i-1,j-1)$, $D(i,j-1)$ en $D(i-1,j)$ al berekend zijn, omdat we i stapsgewijs van 1 t/m N verhogen en voor een vaste i ook met j stapsgewijs van 1 t/m M lopen.

Als we ons de waarden van $D(i,j)$ als elementen van een $N \times M$-matrix voorstellen, vullen we deze matrix rijsgewijs van boven naar beneden en de rijen van links naar rechts. Uiteindelijk zijn we geïnteresseerd in de waarde $D(N,M)$ rechts onder, die de Levenshtein afstand tussen X en Y aangeeft.

Het schema hieronder laat voor het voorbeeld $X =$ `KUNSTMATIGE` en $Y =$ `INTELLIGENTIE` de waarden $D(i,j)$ en een optimaal pad (aangeduid door de hokjes) zien.

		I	N	T	E	L	L	I	G	E	N	T	I	E
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
K	**1**	1	2	3	4	5	6	7	8	9	10	11	12	13
U	2	**2**	3	3	4	5	6	7	8	9	10	11	12	13
N	3	3	**2**	3	4	5	6	7	8	9	9	10	11	12
S	4	4	**3**	3	4	5	6	7	8	9	10	10	11	12
T	5	5	4	**3**	4	5	6	7	8	9	10	10	11	12
M	6	6	5	4	**4**	5	6	7	8	9	10	11	11	12
A	7	7	6	5	5	**5**	6	7	8	9	10	11	12	12
T	8	8	7	6	6	6	**6**	7	8	9	10	10	11	12
I	9	8	8	7	7	7	7	**6**	7	8	9	10	10	11
G	10	9	9	8	8	8	8	7	**6**	**7**	**8**	**9**	**10**	11
E	11	10	10	9	8	9	9	8	7	6	7	8	9	**10**

Merk op dat er verschillende mogelijkheden voor het optimale pad zijn die mogelijk ook verschillende aantallen vervangingen, invoegingen en weglatingen kunnen hebben, maar de *som* van de aantallen vervangingen, invoegingen en weglatingen is bij alle optimale paden natuurlijk hetzelfde. In het voorbeeld is de Levenshtein afstand tussen de twee strings dus 10, het aangegeven pad heeft 4 vervangingen, 4 invoegingen en 2 weglatingen.

Net als bij het Viterbi algoritme moeten we ook hier opslaan vanuit welke voorganger we bij $D(i,j)$ het minimum bereiken om het optimale pad terug te kunnen vinden.

Een iets algemenere versie van de Levenshtein afstand krijgt men, door gewichten aan de verschillende edit operaties te geven, want in sommige toepassingen kan een invoeging erger zijn dan een vervanging. Als we de kosten van een vervanging met k_s, de kosten van een invoeging met k_i en de kosten van een weglating met k_d noteren, berekenen we in dit geval de kosten $D(i,j)$ voor het optimale pad door het punt (i,j) als

$$D(i,j) := \min\{D(i-1,j-1) + d(i,j)\,k_s,\ D(i,j-1) + k_i,\ D(i-1,j) + k_d\},$$

waarbij de initialiseringen $D(i,0) = i\,k_d$ en $D(0,j) = j\,k_i$ zijn.

> In de eerste fase van de spraakherkenning is een soortgelijke techniek ook op spraaksignalen toegepast, er werden namelijk de geluidssignalen in een keten van symbolen omgezet en deze werden door een variatie van de tijdschaal met opgeslagen patronen vergeleken. Deze methode noemt men *dynamic time warping*.

Belangrijke begrippen in deze les

- forward algoritme, backward algoritme
- vooruitkansen, achteruitkansen
- optimale rij van states
- Bellman's principe
- Viterbi algoritme
- training van een HMM, Baum-Welch algoritme
- Levenshtein afstand

Opgaven

99. We beschrijven twee mogelijke uitkomsten `K` en `M` door twee HMM's λ_1, λ_2 met (telkens) twee states. De beginverdelingen voor de states zijn bij beide modellen uniform, dus $\pi = (0.5, 0.5)$. Het model λ_1 heeft de overgangskansen A_1 en emissiekansen B_1, het model λ_2 de overgangskansen A_2 en emissiekansen B_2 gegeven door:

$$A_1 := \begin{pmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{pmatrix},\ B_1 := \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix};\quad A_2 := \begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{pmatrix},\ B_2 := \begin{pmatrix} 0.55 & 0.45 \\ 0.45 & 0.55 \end{pmatrix}.$$

(i) Bepaal voor beide modellen de kansen $p(O \mid \lambda)$ voor de waarnemingen $O_1 =$ `KKK` en $O_2 =$ `MKM`.

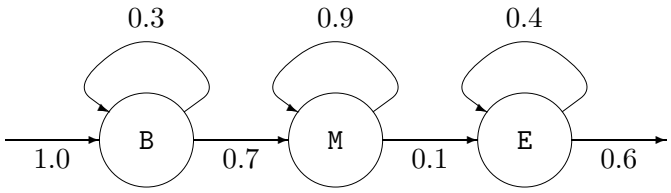
(ii) Bepaal voor beide modellen de optimale rij q van states voor de waarnemingen uit deel (i) en bereken de kansen $p(O, q \mid \lambda)$ voor de combinatie van waarnemingen en states.

100. We kijken nog eens naar het inmiddels bekende HMM met drie munten en parameters:

$$A = (a_{ij}) := \begin{pmatrix} 0.6 & 0.2 & 0.2 \\ 0.4 & 0.2 & 0.4 \\ 0.4 & 0.4 & 0.2 \end{pmatrix}, \quad B = (b_i(k)) := \begin{pmatrix} 0.5 & 0.5 \\ 0.75 & 0.25 \\ 0.25 & 0.75 \end{pmatrix}, \quad \pi = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}).$$

Door een meting weten we, dat bij de eerste en laatste waarneming de eerlijke (eerste) munt geworpen werd. Wat is nu de optimale rij van states die de waarneming $O =$ `KMKMK` voortbrengt?

101. In de spraakherkenning worden fonemen (de kleinste onderscheidbare klanken in een taal) vaak door HMM's met drie states (begin (`B`), midden (`M`), eind (`E`)) gerepresenteerd. Stel de overgangskansen tussen de states zijn door het volgende diagram gegeven:



Het HMM heeft 7 mogelijke uitkomsten $x_1, \ldots, x_7$ en de emissiekansen voor deze uitkomsten zijn gegeven door

state	x_1	x_2	x_3	x_4	x_5	x_6	x_7
`B`	0.5	0.2	0.3	0	0	0	0
`M`	0	0	0.2	0.7	0.1	0	0
`E`	0	0	0	0	0.1	0.5	0.4

Bepaal voor de rij $x_1x_2x_3x_4x_5x_6x_7$ van uitkomsten de optimale rij van states en geef de kans op deze waarneming voor de optimale rij van states aan.

102. Bepaal de Levenshtein afstand tussen de volgende paren van strings (waarbij ook de spatie een symbool is) en geef de edit operaties aan:

(i) $X =$ `ABABAA` en $Y =$ `ABBAA`;

(ii) $X =$ `IK WEET NIETS` en $Y =$ `WEET IK WAT`;

(iii) $X =$ `SINTERKLAAS` en $Y =$ `KERSTMAN`;

(iv) $X =$ `C3POR2D2` en $Y =$ `HAL2001`.