

HIDDEN MARKOV MODELS

PART 2

CSC401/2511 – Natural Language Computing – Spring 2019
Lecture 5 Frank Rudzicz and Chloé Pou-Prom
University of Toronto

Definition of an HMM

- A **hidden Markov model** (HMM) is specified by the 5-tuple $\{S, W, \Pi, A, B\}$:

- $S = \{s_1, \dots, s_N\}$

: set of **states** (e.g., moods)

- $W = \{w_1, \dots, w_K\}$

: output **alphabet** (e.g., words)

$$\theta \left\{ \begin{array}{l} \Pi = \{\pi_1, \dots, \pi_N\} \\ A = \{a_{ij}\}, i, j \in S \\ B = b_i(w), i \in S, w \in W \end{array} \right.$$

: **initial** state probabilities

: state **transition** probabilities

: state **output** probabilities

yielding

- $Q = \{q_0, \dots, q_T\}, q_i \in S$

: state sequence

- $O = \{\sigma_0, \dots, \sigma_T\}, \sigma_i \in W$

: output sequence

Fundamental tasks for HMMs

1. Given a model with particular parameters $\theta = \langle \Pi, A, B \rangle$, how do we efficiently compute the likelihood of a *particular observation sequence*, $P(\mathcal{O}; \theta)$?

We previously computed the probabilities of word sequences using N -grams.

The probability of a particular sequence is usually useful as a means to some other end.

The Forward procedure

- The trellis is computed **left-to-right** and **top-to-bottom**.
- There are three steps in this procedure:
 - **Initialization:** Compute the nodes in the *first column* of the trellis ($t = 0$).
 - **Induction:** Iteratively compute the nodes in the *rest* of the trellis ($1 \leq t < T$).
 - **Conclusion:** Sum over the nodes in the *last column* of the trellis ($t = T - 1$).

The Backward procedure

- Initialization

$$\beta_i(T - 1) = 1,$$

$$i := 1..N$$

- Induction

$$\beta_i(t) = \sum_{j=1}^N a_{ij} b_j(\sigma_{t+1}) \beta_j(t + 1),$$

$$i := 1..N$$

$$t := T - 1..0$$

- Conclusion

$$P(\mathcal{O}; \theta) = \sum_{i=1}^N \pi_i b_i(\sigma_0) \beta_i(0)$$

Fundamental tasks for HMMs

2. Given an **observation sequence** \mathcal{O} and a **model** θ , how do we choose a **state sequence** $Q = \{q_0, \dots, q_T\}$ that *best explains* the observations?

This is the task of **inference** – i.e., guessing at the best explanation of unknown (‘latent’) variables given our model.

This is often an important part of **classification**.

Fundamental tasks for HMMs

3. Given a large **observation sequence** \mathcal{O} , how do we choose the *best parameters* $\theta = \langle \Pi, A, B \rangle$ that explain the data \mathcal{O} ?

This is the task of **training**.

As before, we want our parameters to be set so that the available training data is maximally likely,
But doing so will involve guessing unseen information.

Fundamental tasks for HMMs

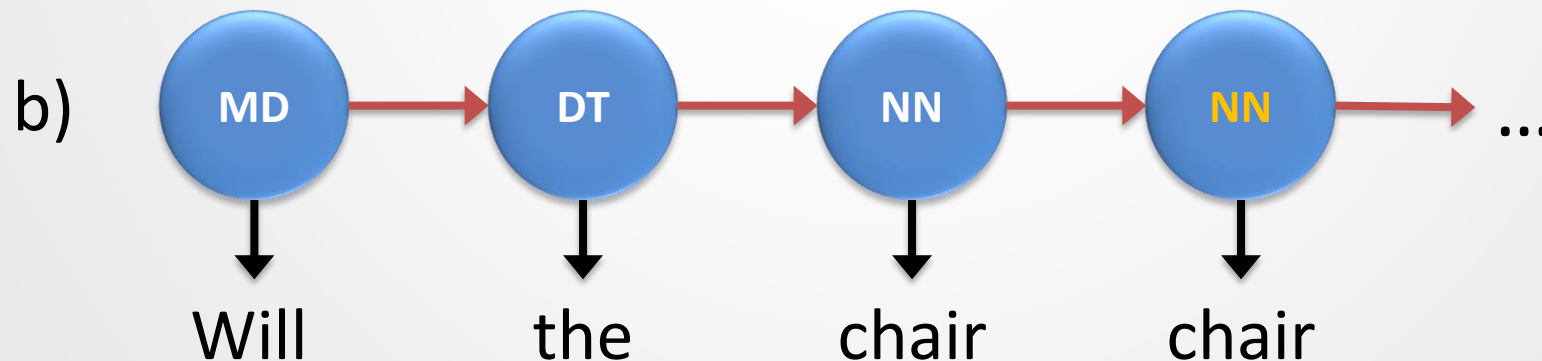
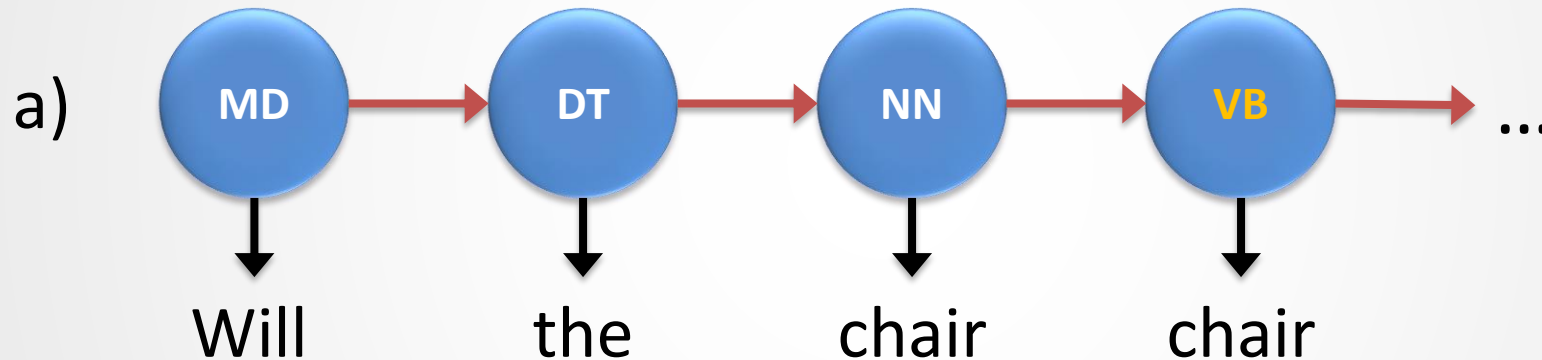
2. Given an **observation sequence** \mathcal{O} and a **model** θ , how do we choose a **state sequence** $Q = \{q_0, \dots, q_T\}$ that *best explains* the observations?

This is the task of **inference** – i.e., guessing at the best explanation of unknown ('latent') variables given our model.

This is often an important part of **classification**.

Example – PoS state sequences

- Will/MD the/DT chair/**NN** chair/?? the/DT meeting/NN from/IN that/DT chair/**NN**?*



Task 2: Choosing $Q = \{q_0 \dots q_T\}$

- The purpose of finding **the best state sequence** Q^* out of all possible state sequences Q is that it tells us what is **most likely** to be going on ‘under the hood’.
 - E.g., it tells us the most likely *part-of-speech* tags,
 - E.g., it tells us the most likely *English words* given *French translations* (*in a very simple model).
- With the **Forward** algorithm, we didn’t care about specific state sequences – we were summing over **all possible** state sequences.

Task 2: Choosing $Q = \{q_0 \dots q_T\}$

- In other words,

$$Q^* = \underset{Q}{\operatorname{argmax}} P(\mathcal{O}, Q; \theta)$$

where

$$P(\mathcal{O}, Q; \theta) = \pi_{q_0} b_{q_0}(\sigma_0) \prod_{t=1}^T a_{q_{t-1}q_t} b_{q_t}(\sigma_t)$$

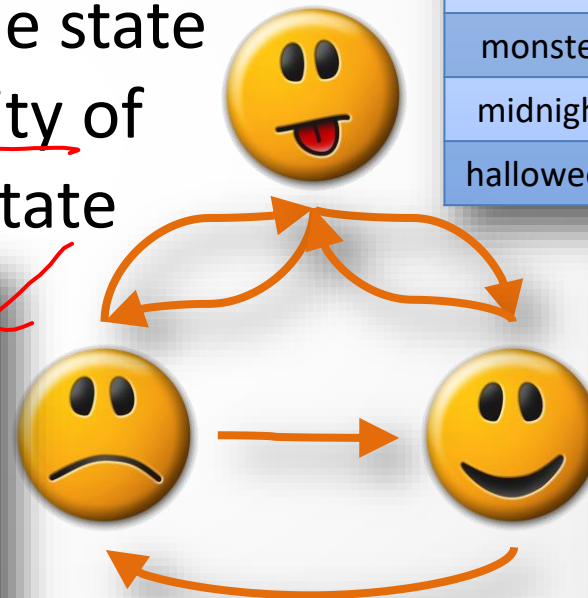
Recall

O = upside, upside, upside

- Observation likelihoods depend on the state, which changes over time
- We **cannot** simply choose the state that maximizes the probability of o_t without considering the state sequence.

word	P(word)
upside	0.1
down	0.05
promise	0.05
friend	0.6
monster	0.05
midnight	0.1
halloween	0.05

word	P(word)
upside	0.25
down	0.25
promise	0.05
friend	0.3
monster	0.05
midnight	0.09
halloween	0.01



word	P(word)
upside	0.3
down	0
promise	0
friend	0.2
monster	0.05
midnight	0.05
halloween	0.4

The Viterbi algorithm

- The Viterbi algorithm is an inductive dynamic-programming algorithm that uses a *new kind* of trellis.
- We define the probability of the most probable path leading to the trellis node at (state i , time t) as

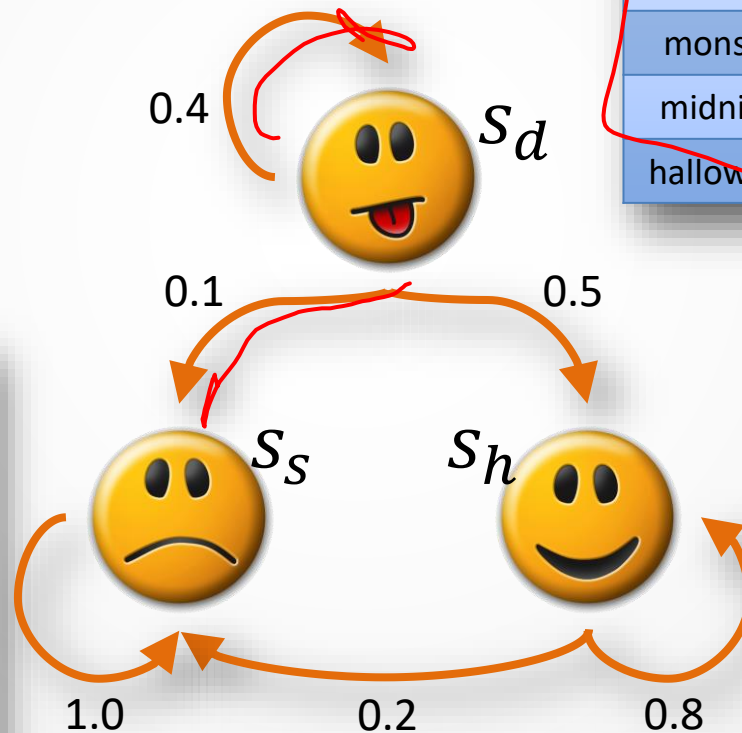
$$\delta_i(t) = \max_{q_0 \dots q_{t-1}} P(q_0 \dots q_{t-1}, \sigma_0 \dots \sigma_{t-1}, \mathbf{q}_t = \mathbf{s}_i; \theta)$$

- $\psi_i(t)$: The best possible previous state, if I'm in state i at time t .

Viterbi example

- For illustration, we assume a simpler state-transition topology:

word	P(word)
upside	0.25
down	0.25
promise	0.05
friend	0.3
monster	0.05
midnight	0.09
halloween	0.01



word	P(word)
upside	0.1
down	0.05
promise	0.05
friend	0.6
monster	0.05
midnight	0.1
halloween	0.05

word	P(word)
upside	0.3
down	0
promise	0
friend	0.2
monster	0.05
midnight	0.05
halloween	0.4

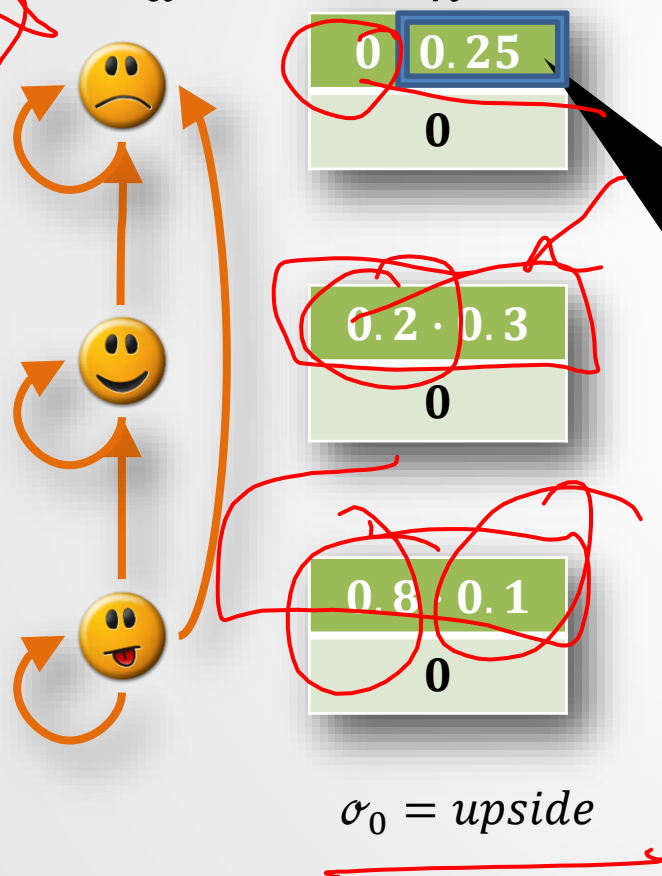
Step 1: Initialization of Viterbi

- Initialize with $\delta_0(i) = \pi_i b_i(\sigma_0)$ and $\psi_i(0) = 0$ for all states.



Step 1: Initialization of Viterbi

- For example, let's assume $\pi_d = 0.8, \pi_h = 0.2$, and $\mathcal{O} = \text{upside, friend, halloween}$



word	P(word)
upside	0.25
down	0.25
promise	0.05
friend	0.3
monster	0.05
midnight	0.09
halloween	0.01

δ : max probability

ψ : backtrace

$\sigma_1 = \text{friend}$

$\sigma_2 = \text{halloween}$

Observations, σ_t

Step 2: Induction of Viterbi



0
0

0.06
0

0.08
0

$\sigma_0 = upside$

The best path to state s_j at time t , $\delta_j(t)$, depends on the best path to each possible previous state, $\delta_i(t-1)$, and their transitions to j , a_{ij}

$$\delta_j(t) = \max_i [\delta_i(t-1) a_{ij}] b_j(\sigma_t)$$

$$\psi_j(t) = \operatorname{argmax}_i [\delta_i(t-1) a_{ij}]$$

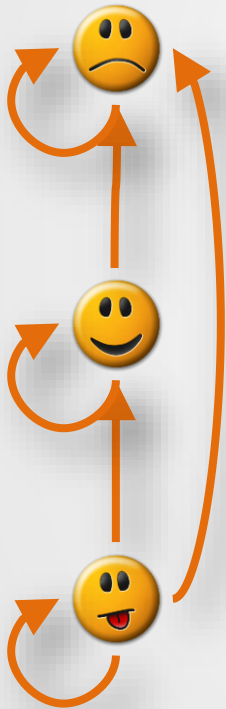
$\sigma_1 = friend$

$\sigma_2 = halloween$

Observations, σ_t

Step 2: Induction of Viterbi

Specifically...



0
0

0.06
0

0.08
0

$\sigma_0 = \text{upside}$

$$\delta_s(1) = \max_i [\delta_i(0) a_{is}] b_s(\sigma_1)$$

$$\psi_s(1) = \operatorname{argmax}_i [\delta_i(0) a_{is}]$$

$$\delta_h(1) = \max_i [\delta_i(0) a_{ih}] b_h(\sigma_1)$$

$$\psi_h(1) = \operatorname{argmax}_i [\delta_i(0) a_{ih}]$$

$$\delta_d(1) = \max_i [\delta_i(0) a_{id}] b_d(\sigma_1)$$

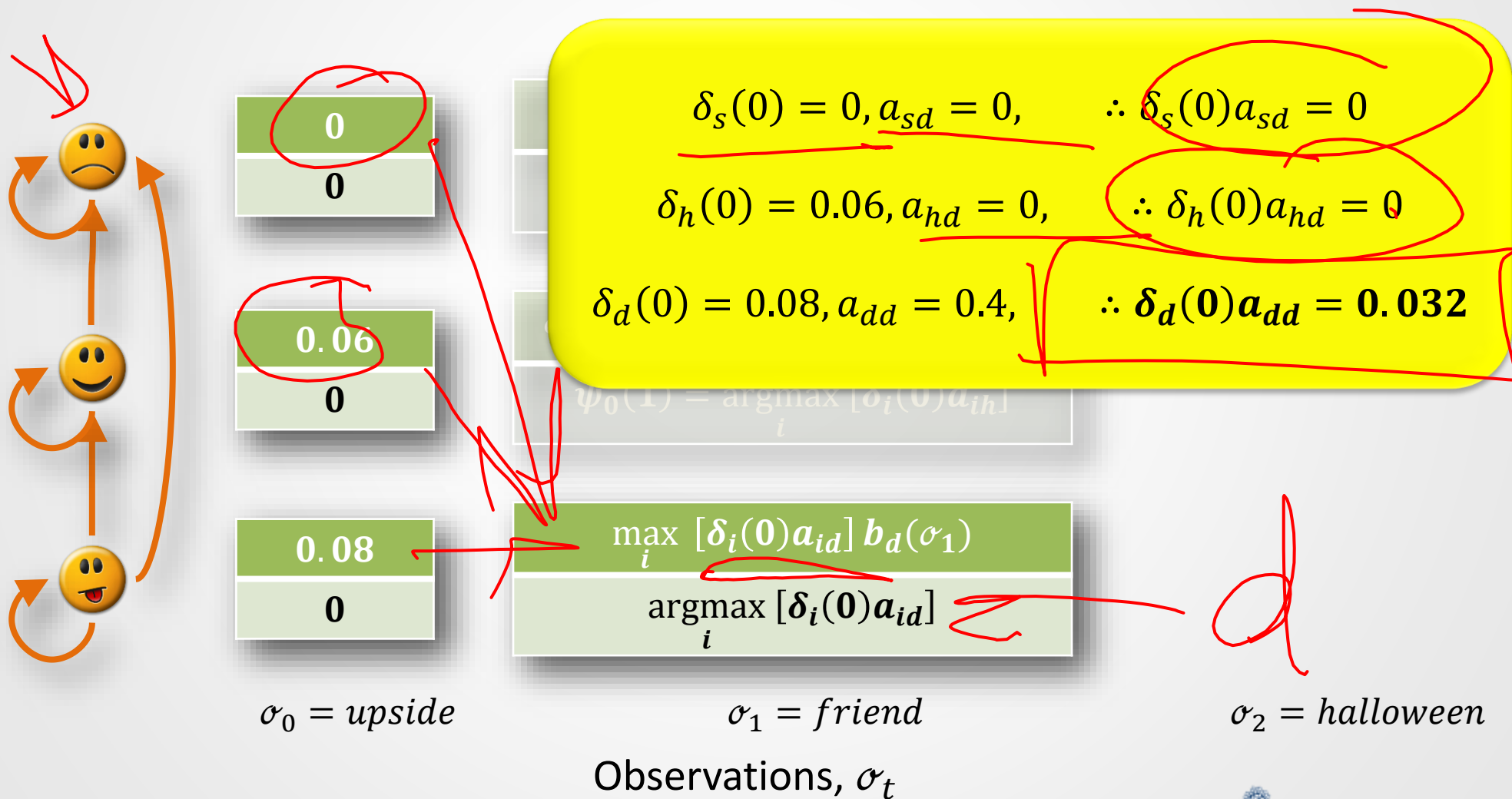
$$\psi_d(1) = \operatorname{argmax}_i [\delta_i(0) a_{id}]$$

$\sigma_1 = \text{friend}$

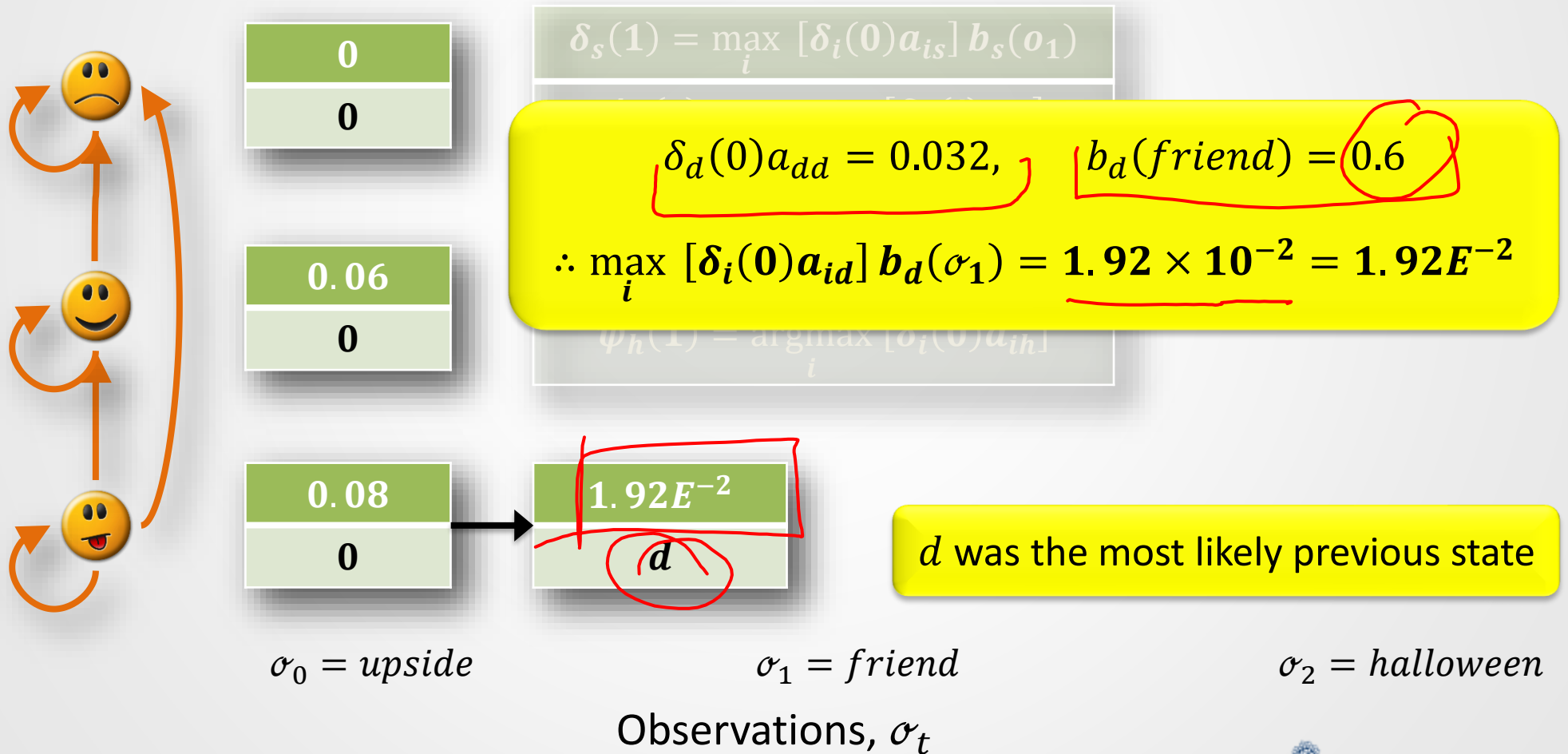
$\sigma_2 = \text{halloween}$

Observations, σ_t

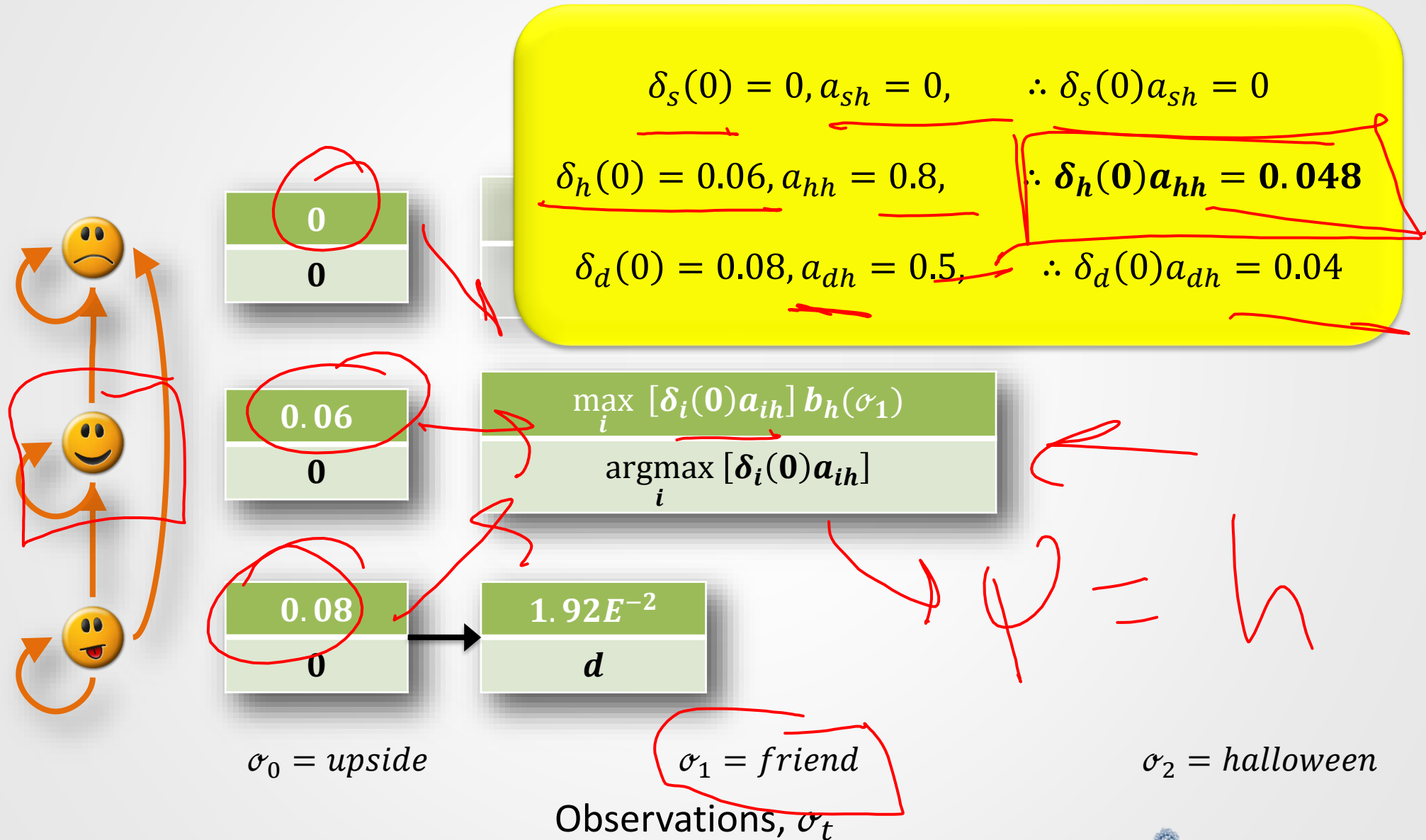
Step 2: Induction of Viterbi



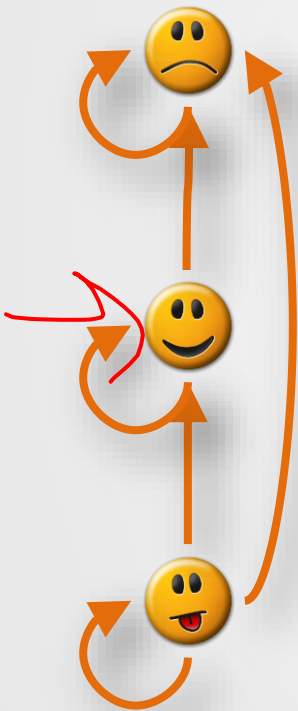
Step 2: Induction of Viterbi



Step 2: Induction of Viterbi



Step 2: Induction of Viterbi



0
0

0.06
0

0.08
0

9.6E ⁻³
<i>h</i>

1.92E ⁻²
<i>d</i>

$\sigma_0 = \text{upside}$

$\sigma_1 = \text{friend}$

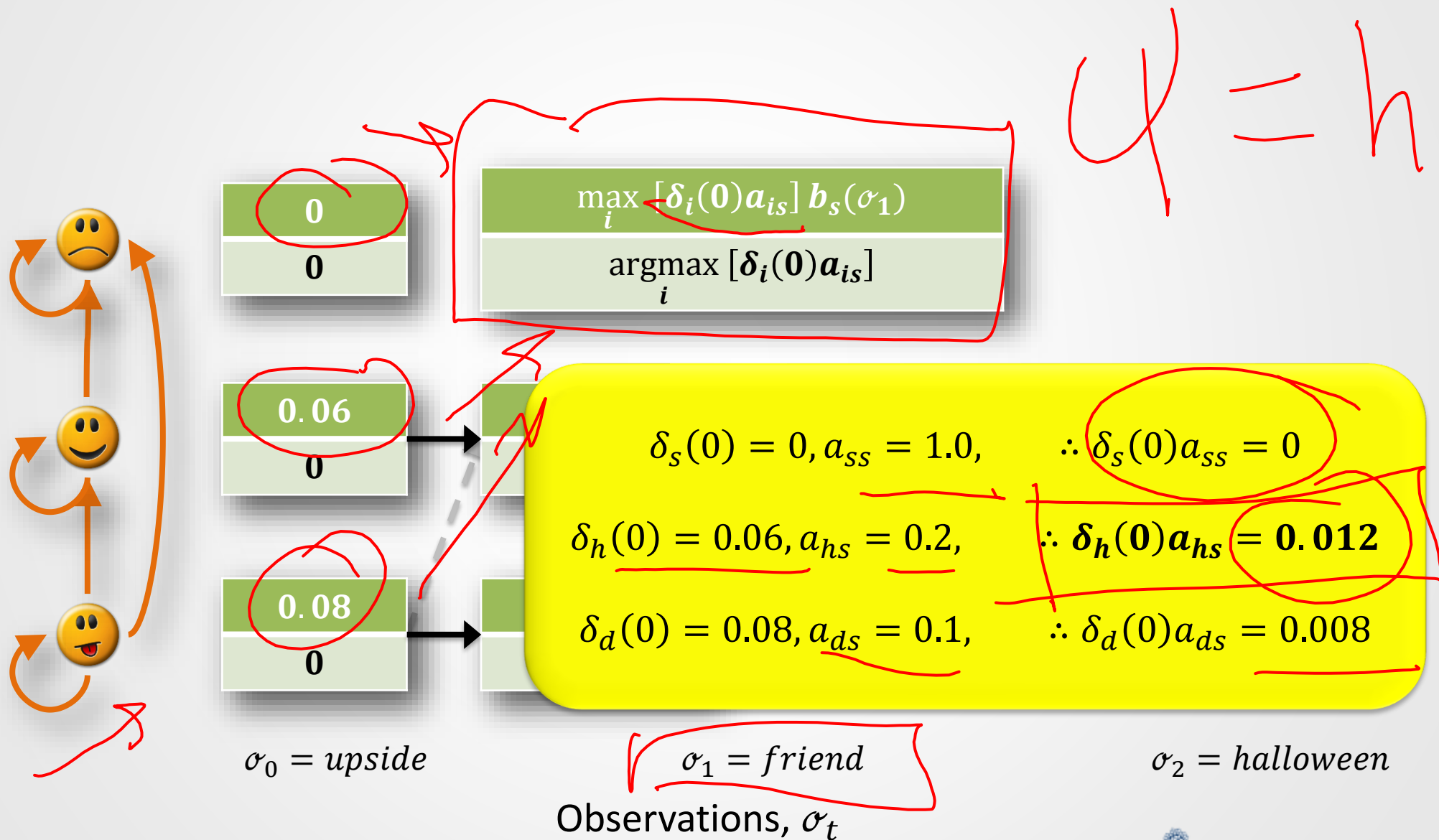
$\sigma_2 = \text{halloween}$

Observations, σ_t

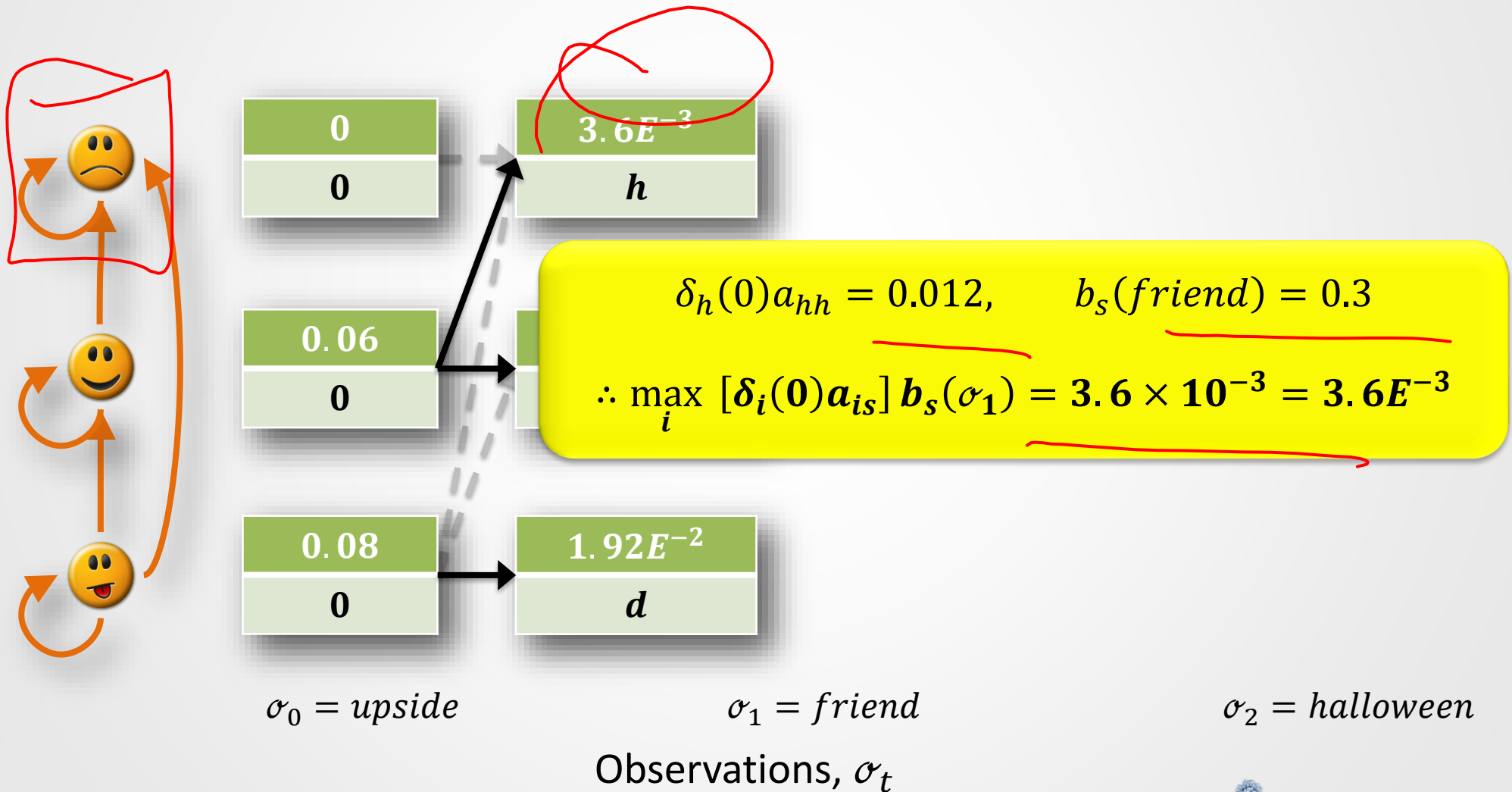
$$\delta_h(0)a_{hh} = 0.048, \quad b_h(\text{friend}) = 0.2$$

$$\therefore \max_i [\delta_i(0)a_{ih}] b_h(\sigma_1) = 9.6 \times 10^{-3} = 9.6E^{-3}$$

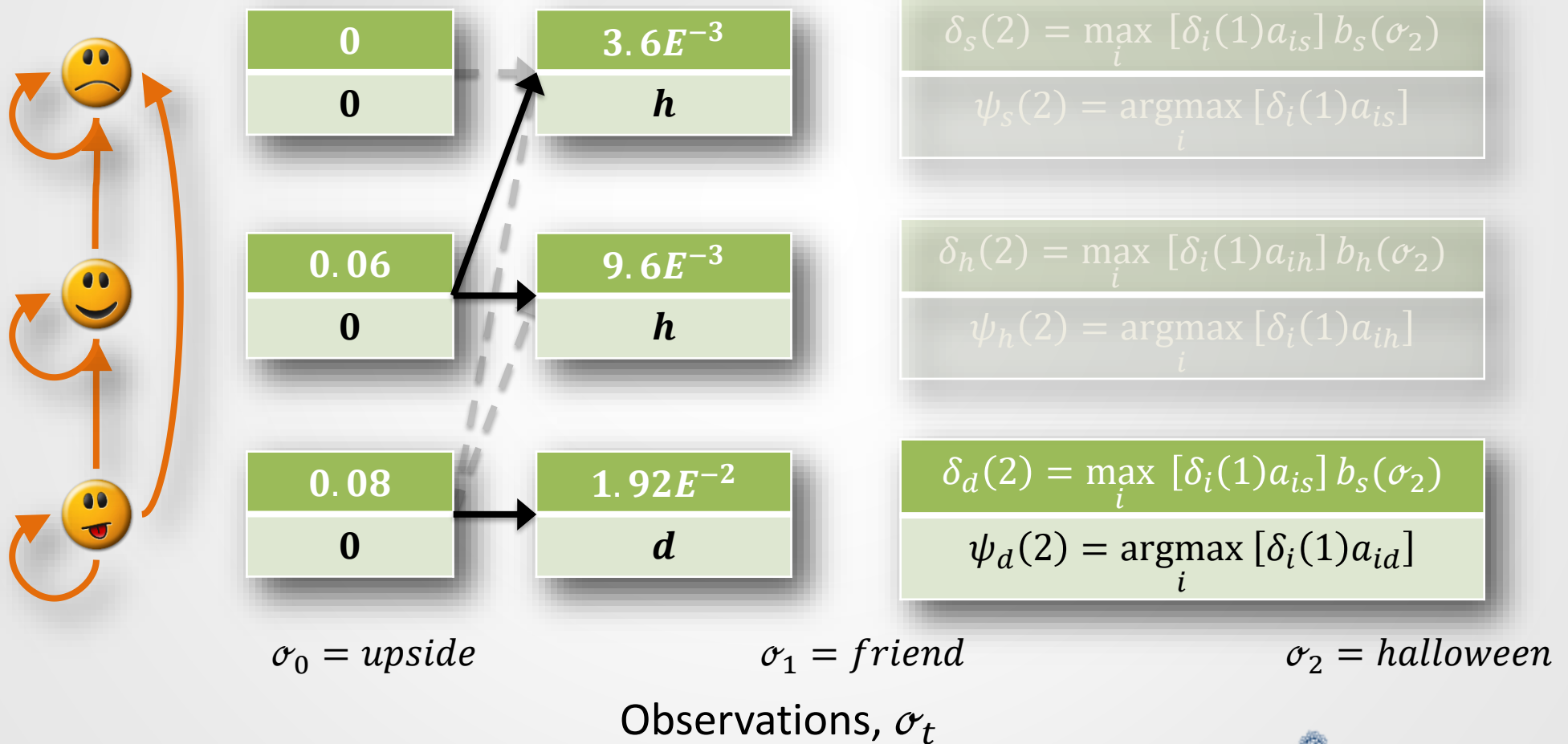
Step 2: Induction of Viterbi



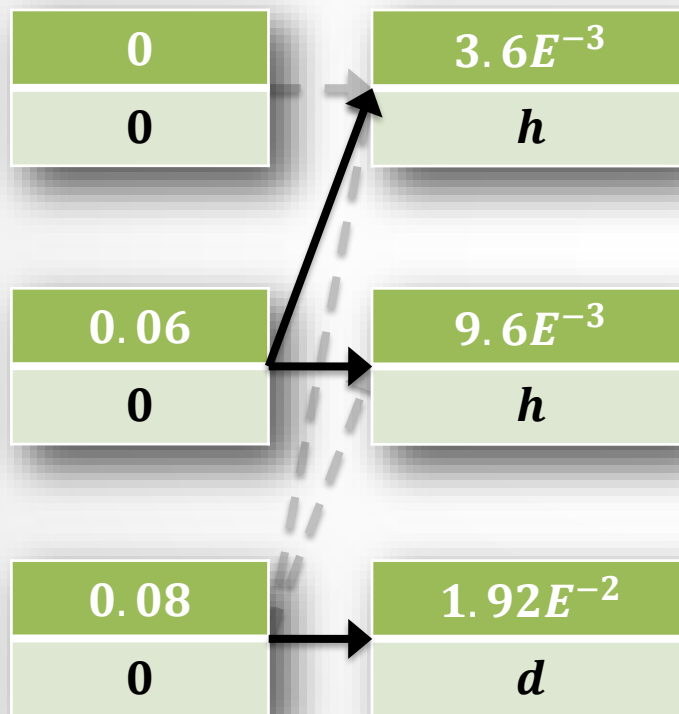
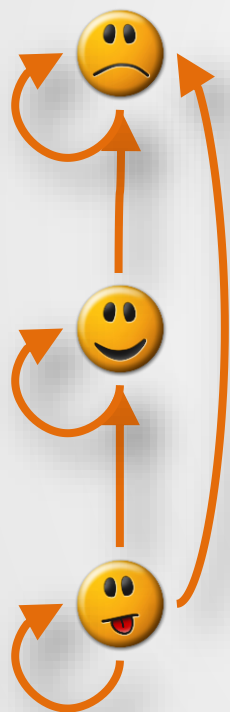
Step 2: Induction of Viterbi



Step 2: Induction of Viterbi



Step 2: Induction of Viterbi



$\sigma_0 = upside$

$\sigma_1 = friend$

$\sigma_2 = halloween$

Observations, σ_t

$$\delta_s(1) = 3.6E^{-3}, a_{sd} = 0, \\ \therefore \delta_s(1)a_{sd} = 0$$

$$\delta_h(1) = 9.6E^{-3}, a_{hd} = 0, \\ \therefore \delta_h(1)a_{hd} = 0$$

$$\delta_d(1) = 1.92E^{-2}, a_{dd} = 0.4, \\ \therefore \delta_d(1)a_{dd} = \mathbf{0.00768}$$

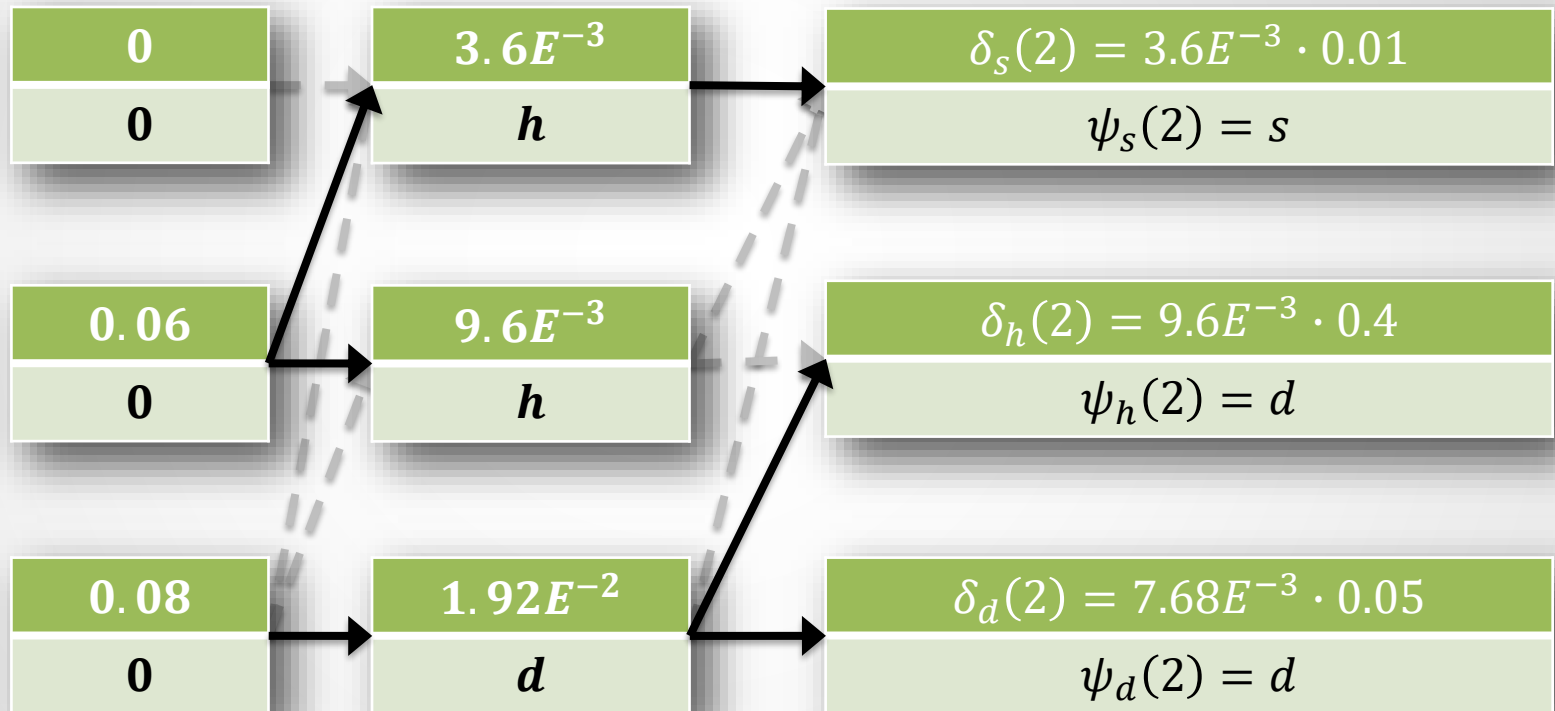
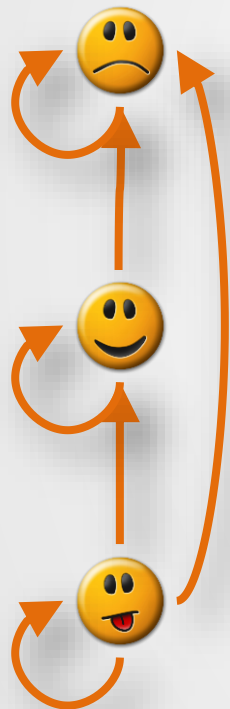
$$\psi_h(2) = \operatorname{argmax}_i [\delta_i(1)a_{ih}]$$

$$\delta_d(2) = \max_i [\delta_i(1)a_{is}] b_s(\sigma_2)$$

$$\psi_d(2) = \operatorname{argmax}_i [\delta_i(1)a_{id}]$$

Step 2: Induction of Viterbi

Continuing...



$\sigma_0 = \text{upside}$

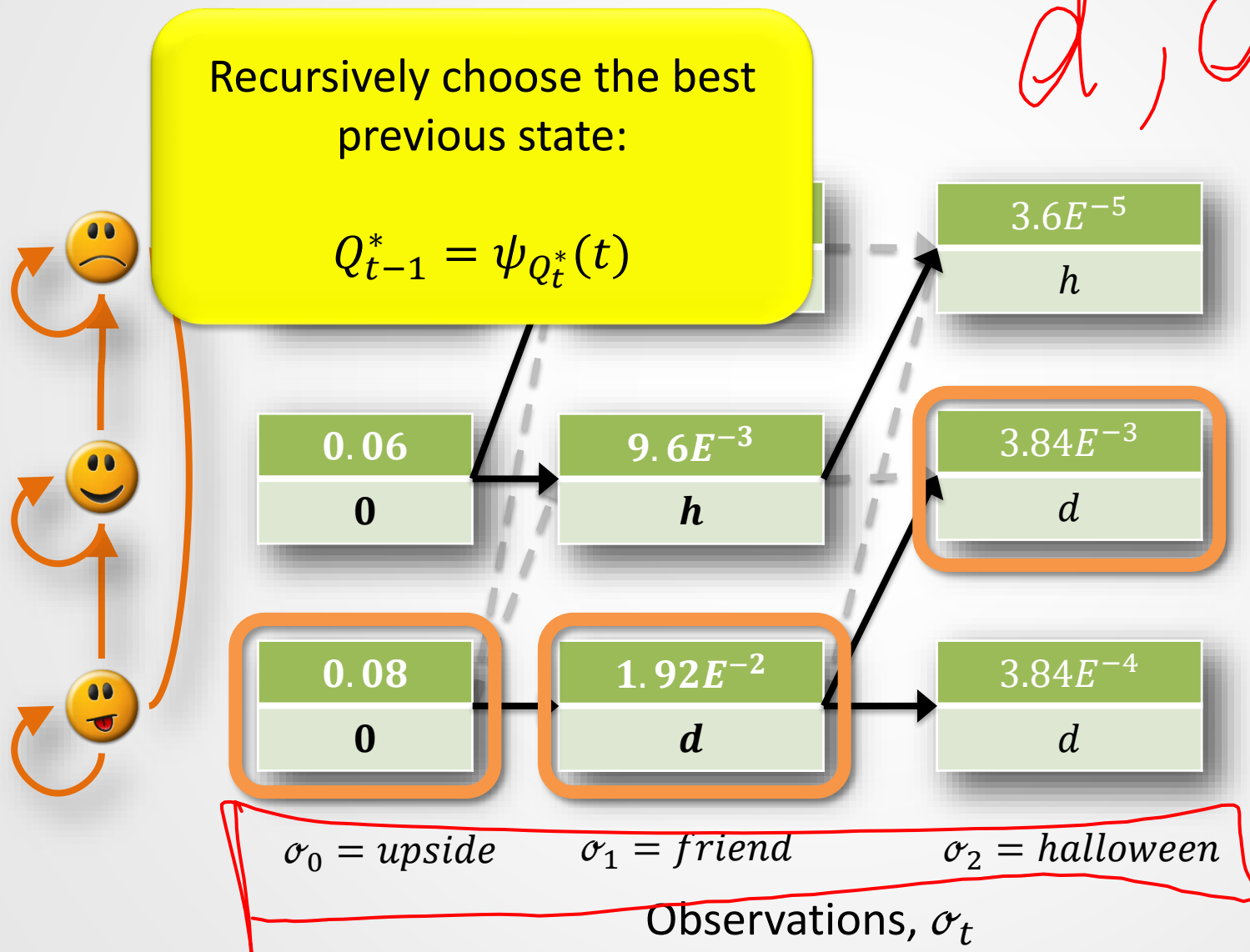
$\sigma_1 = \text{friend}$

$\sigma_2 = \text{halloween}$

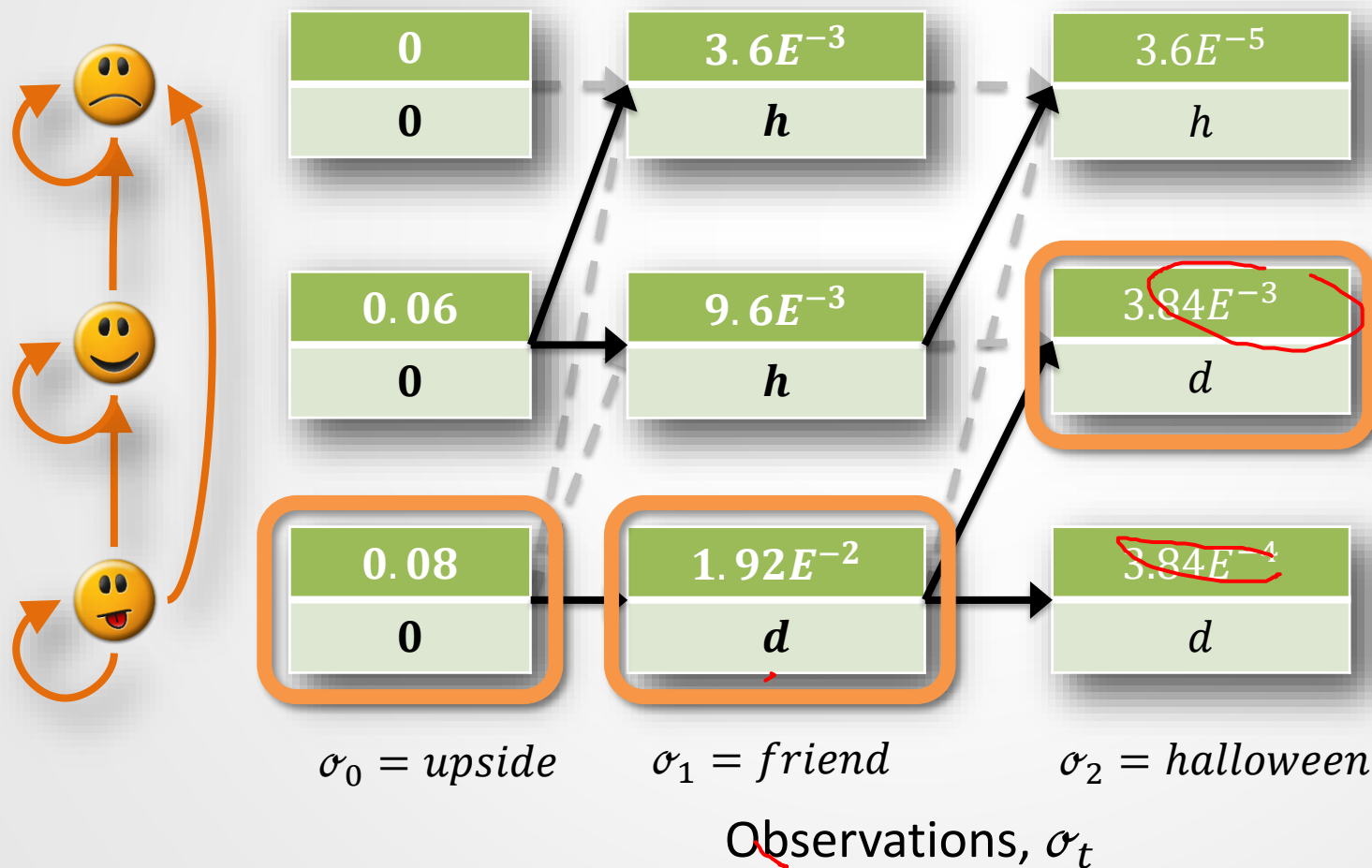
Observations, σ_t

$$q_t = h$$


Step 3: Conclusion of Viterbi

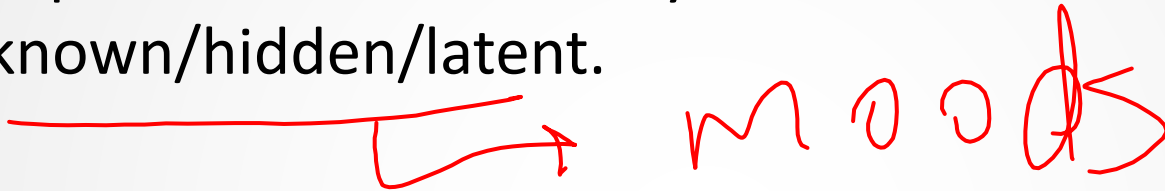


Step 3: Conclusion of Viterbi



Why did we choose $Q^* = \{q_0 \dots q_T\}$?

- Recall the purpose of HMMs:
 - To represent multivariate systems where some variable is unknown/hidden/latent.



- Finding the best hidden-state sequence Q^* allows us to:
 - Identify **unseen parts-of-speech** given **words**,
 - Identify **equivalent English** words given **French words**,
 - Identify **unknown phonemes** given **speech sounds**,
 - Decipher **hidden messages** from **encrypted symbols**,
 - Identify **hidden relationships** from **gene sequences**,
 - Identify **hidden market conditions** given **stock prices**,
 - ...

Working in the log domain

- Our formulation was

$$Q^* = \operatorname{argmax}_Q P(\mathcal{O}, Q; \theta)$$

this is equivalent to

$$Q^* = \operatorname{argmin}_Q -\log_2 P(\mathcal{O}, Q; \theta)$$

where

$$-\log_2 P(\mathcal{O}, Q; \theta)$$

$$= -\log_2 \left(\pi_{q_0} b_{q_0}(\sigma_0) \right) - \sum_{t=1}^T \log_2 \left(a_{q_{t-1}q_t} b_{q_t}(\sigma_t) \right)$$

Fundamental tasks for HMMs

3. Given a large observation sequence \mathcal{O} for **training**, but **not** the state sequence, how do we choose the 'best' parameters $\theta = \langle \Pi, A, B \rangle$ that explain the data \mathcal{O} ?

This is the task of **training**.

As with observable Markov models and **MLE**, we want our parameters to be set so that

the available training data is maximally likely,

But doing so will involve **guessing unseen information...**

Task 3: Choosing $\theta = \langle \Pi, A, B \rangle$

- We want to **modify** the parameters of our model $\theta = \langle \Pi, A, B \rangle$ so that $P(\mathcal{O}; \theta)$ is maximized for some **training data** \mathcal{O} :

$$\hat{\theta} = \operatorname{argmax}_{\theta} P(\mathcal{O}; \theta)$$

- Why? E.g., if we later want to choose the **best state sequence** Q^* for previously unseen **test data**, the parameters of the HMM should be tuned to similar **training data**.

Task 3: Choosing $\theta = \langle \Pi, A, B \rangle$

- $\hat{\theta} = \operatorname{argmax}_{\theta} P(\mathcal{O}; \theta) = \operatorname{argmax}_{\theta} \sum_Q P(\mathcal{O}, Q; \theta)$

Can we do this?

- $P(\mathcal{O}, Q; \theta) = P(q_{0:t})P(w_{0:t}|q_{0:t}) \approx \prod_{i=0}^t P(q_i|q_{i-1})P(w_i|q_i)$

Recall that we
could use MLE
when Q was known

Task 3: Choosing $\theta = \langle \Pi, A, B \rangle$

- $P(\mathcal{O}, Q; \theta) = P(q_{0:t})P(w_{0:t}|q_{0:t}) \approx \prod_{i=0}^t \underline{P(q_i|q_{i-1})} \underline{P(w_i|q_i)}$
- If the training data contained state sequences, we could simply do **maximum likelihood estimation**, as before:

$$\bullet \quad \underline{P(q_i|q_{i-1})} = \frac{\text{Count}(q_{i-1} q_i)}{\text{Count}(q_{i-1})}$$

$$\underline{P(w_i|q_i)} = \frac{\text{Count}(w_i \wedge q_i)}{\text{Count}(q_i)}$$

- But we **don't** know the states; we **can't** count them.
- However, we **can** use an **iterative hill-climbing** approach if we can **guess** the counts.

What to do with incomplete data?

- When our training data are **incomplete** (i.e., one or more variables in our model is hidden) we **cannot** use maximum likelihood estimation.
- We have **no way** of counting the state-transitions because we don't know which sequence of states generated our observations.
- We can guess the counts if we have some good pre-existing model.

Expecting and maximizing

- If we knew θ , we could make **expectations** such as

- Expected number of times in state s_i ,
- Expected number of transitions $s_i \rightarrow s_j$

- If we knew:

- Expected number of times in state s_i ,
- Expected number of transitions $s_i \rightarrow s_j$

then we could compute the **maximum likelihood estimate** of

$$\theta = \langle \pi_i, \{a_{ij}\}, \{b_i(w)\} \rangle$$

Expectation-maximization

- **Expectation-maximization** (EM) is an **iterative** training algorithm that alternates between two steps:
 - **Expectation (E):** guesses the **expected** counts for the hidden sequence using the current model θ_k .
 - **Maximization (M):** computes a new θ that **maximizes** the likelihood of the data, given the guesses of the E-step. This θ_{k+1} is then used in the next E-step.
- Continue until convergence or stopping condition...

Baum-Welch re-estimation

- **Baum-Welch** (BW): *n.* a specific version of EM for HMMs.
a.k.a. '**forward-backward**' algorithm.

1. Initialize the model.

2. Compute **expectations** for $\alpha_i(t)$ and $\beta_i(t)$ for each state i and time t , given training data \mathcal{O} .

3. Adjust our **start**, **transition**, and **observation** probabilities to **maximize** the likelihood of \mathcal{O} .

4. Go to 2. and repeat until convergence or stopping condition...

Local maxima

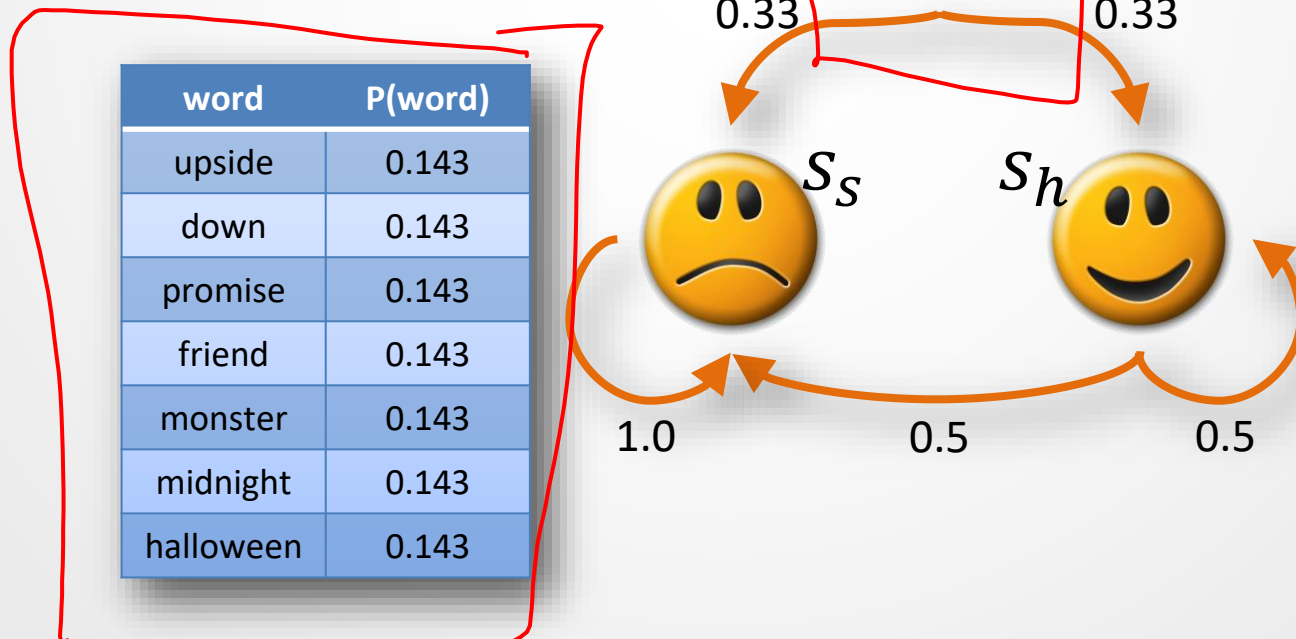
- Baum-Welch changes θ to climb a 'hill' in $P(\mathcal{O}; \theta)$.
 - How we initialize θ can have a big effect.



Step 1: BW initialization

- Our **initial guess** for the parameters, θ_0 , can be:
 - All probabilities are **uniform** (e.g., $b_i(w_a) = b_i(w_b)$ for all states i and words w)

word	P(word)
upside	0.143
down	0.143
promise	0.143
friend	0.143
monster	0.143
midnight	0.143
halloween	0.143



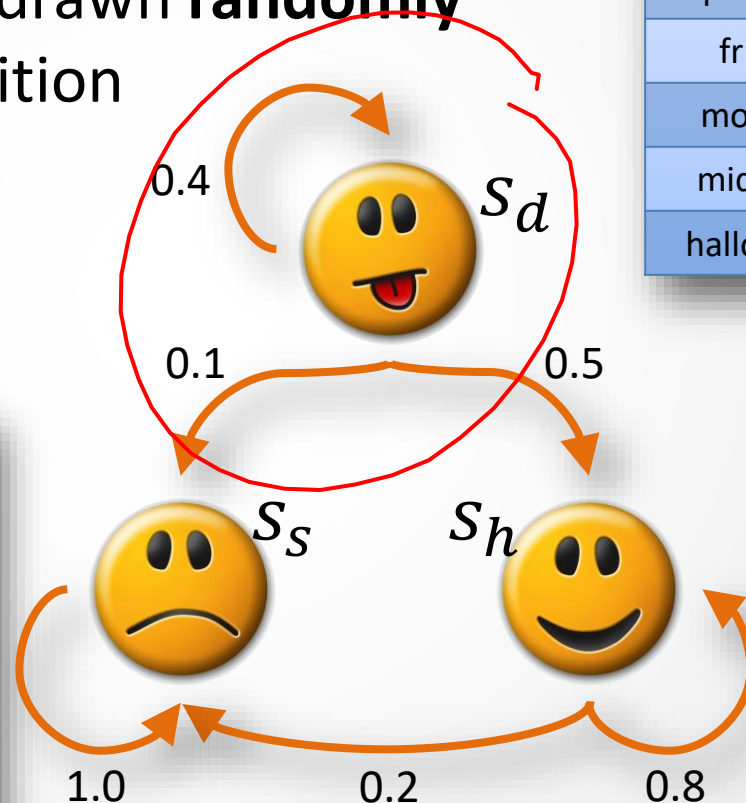
word	P(word)
upside	0.143
down	0.143
promise	0.143
friend	0.143
monster	0.143
midnight	0.143
halloween	0.143

Step 1: BW initialization

- Our **initial guess** for the parameters, θ_0 , can be:
 - b) All probabilities are drawn **randomly** (subject to the condition that $\sum_i P(i) = 1$)

word	P(word)
upside	0.1
down	0.05
promise	0.05
friend	0.6
monster	0.05
midnight	0.1
halloween	0.05

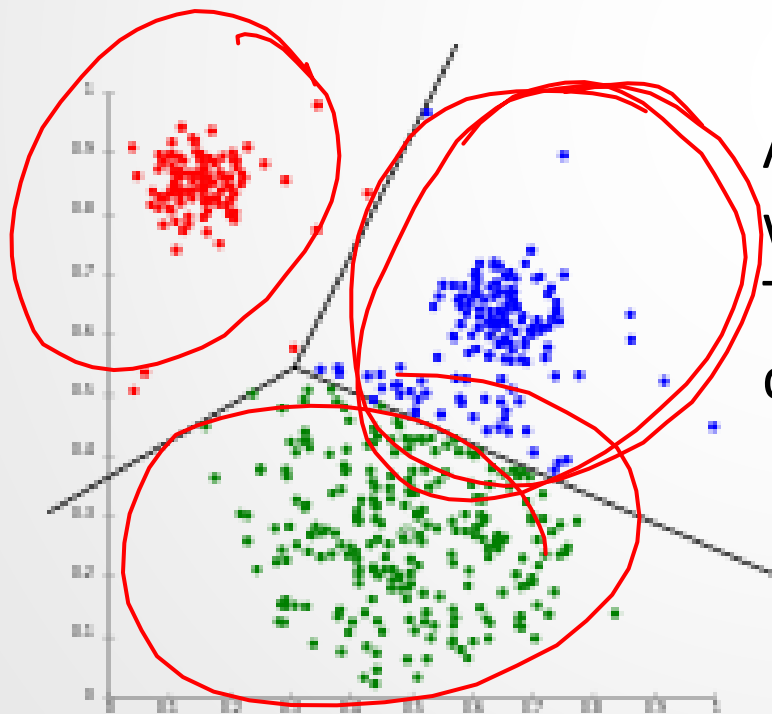
word	P(word)
upside	0.25
down	0.25
promise	0.05
friend	0.3
monster	0.05
midnight	0.09
halloween	0.01



word	P(word)
upside	0.3
down	0
promise	0
friend	0.2
monster	0.05
midnight	0.05
halloween	0.4

Step 1: BW initialization

- Our **initial guess** for the parameters, θ_0 , can be:
 - c) Observation distributions are drawn from prior distributions:
e.g., $b_i(w_a) = P(w_a)$ for all states i .
sometimes this involves pre-clustering, e.g. k-means



All blue dots are words in state BLUE. Their probability distribution is →

word	P(word)
upside	0.2
down	0.1
promise	0.03
friend	0.5
monster	0.07
midnight	0.02
halloween	0.08

What to expect when you're expecting

- If we knew θ , we could estimate **expectations** such as
 - Expected number of times in state s_i ,
 - Expected number of transitions $s_i \rightarrow s_j$

- If we knew:
 - Expected number of times in state s_i ,
 - Expected number of transitions $s_i \rightarrow s_j$then we could compute the **maximum likelihood estimate** of
$$\theta = \langle \{a_{ij}\}, \{b_i(w)\}, \pi_i \rangle$$

BW E-step (occupation)

- We define

$$\gamma_i(t) = P(q_t = i | \mathcal{O}; \theta_k)$$

as the probability of **being** in state i at time t , based on our current model, θ_k , **given** the entire observation, \mathcal{O} .

and rewrite as:

$$\begin{aligned}\gamma_i(t) &= \frac{P(q_t = i, \mathcal{O}; \theta_k)}{P(\mathcal{O}; \theta_k)} \\ &= \frac{\alpha_i(t) \beta_i(t)}{P(\mathcal{O}; \theta_k)}\end{aligned}$$

Remember, $\alpha_i(t)$ and $\beta_i(t)$ depend on values from $\theta = \langle \pi_i, a_{ij}, b_i(w) \rangle$

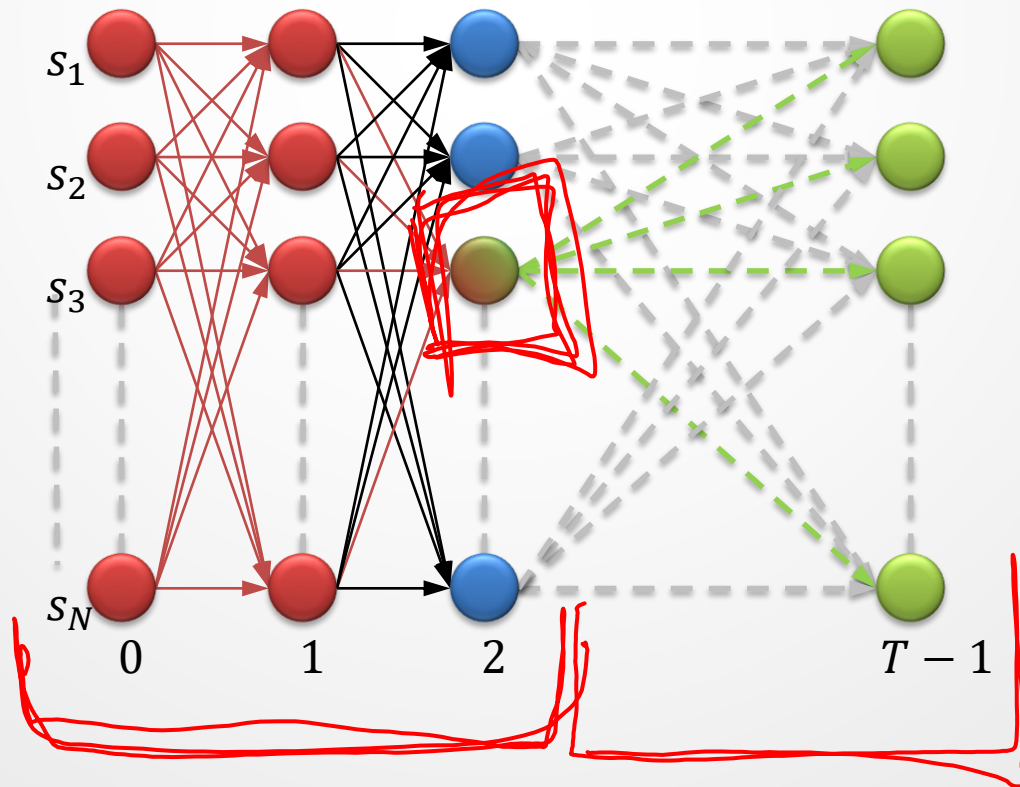
Combining α and β

forward

$$P(\mathcal{O}, q_t = i; \theta) = \alpha_i(t) \beta_i(t)$$

back

$$\therefore P(\mathcal{O}; \theta) = \sum_{i=1}^N \alpha_i(t) \beta_i(t)$$



BW E-step (transition)

- We define

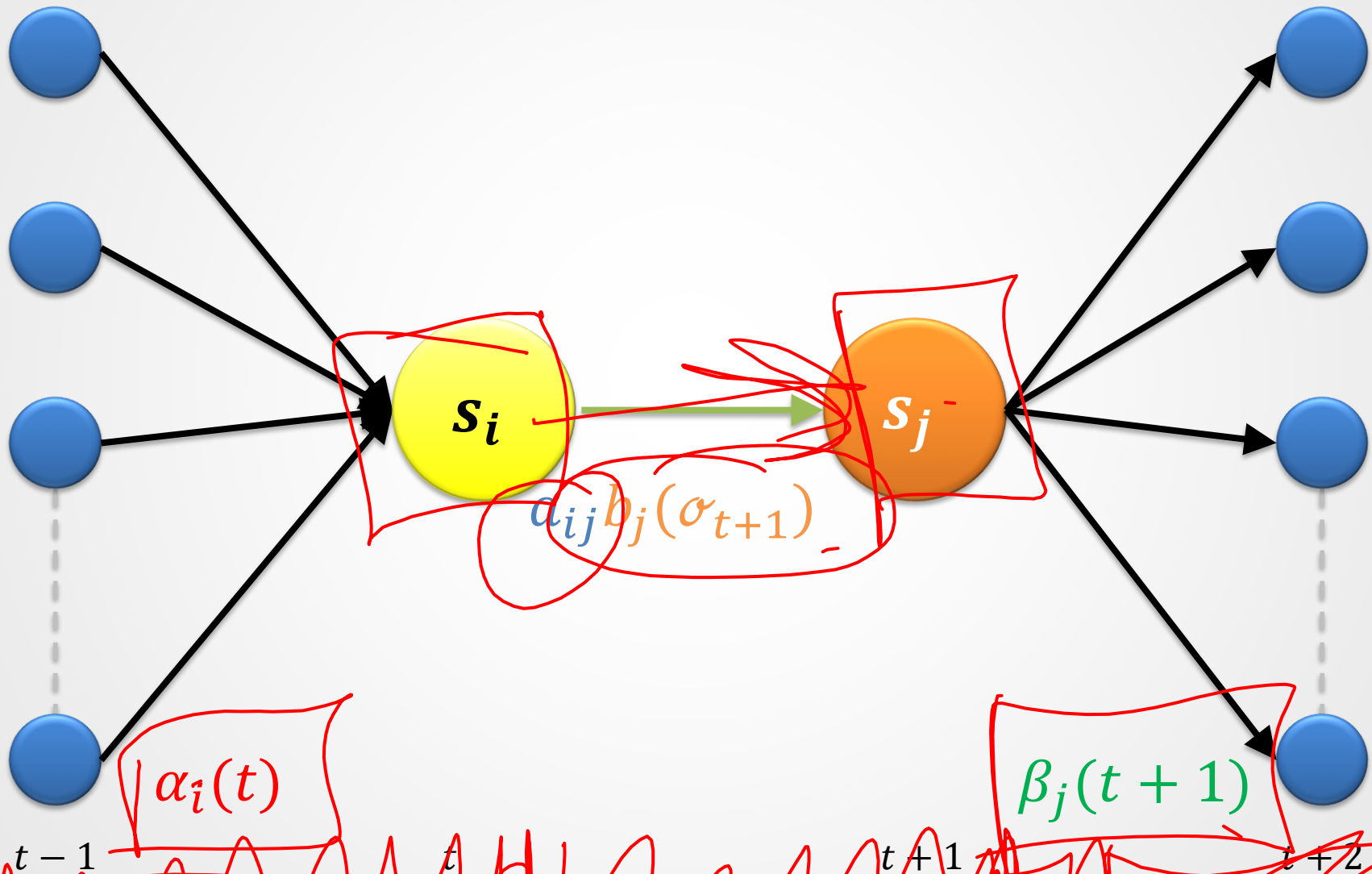
$$\xi_{ij}(t) = P(q_t = i, q_{t+1} = j | \mathcal{O}; \theta_k)$$

as the probability of **transitioning** from state i at time t to state j at time $t + 1$ **based on** our current model, θ_k , and **given** the entire observation, \mathcal{O} . This is:

$$\xi_{ij}(t) = \frac{P(q_t = i, q_{t+1} = j, \mathcal{O}; \theta_k)}{P(\mathcal{O}; \theta_k)}$$
$$= \frac{\alpha_i(t) a_{ij} b_j(\sigma_{t+1}) \beta_j(t+1)}{P(\mathcal{O}; \theta_k)}$$

Again, these estimates come from our model at iteration k , θ_k .

$P(q_t = i, q_{t+1} = j, O, \theta_K)$
BW E-step (transition)



Expecting and maximizing

- If we knew θ , we could estimate **expectations** such as
 - Expected number of times in state s_i ,
 - Expected number of transitions $s_i \rightarrow s_j$

- If we knew:
 - Expected number of times in state s_i ,
 - Expected number of transitions $s_i \rightarrow s_j$

then we could compute the **maximum likelihood estimate** of

$$\theta = \langle \{a_{ij}\}, \{b_i(w)\}, \pi_i \rangle$$

BW M-step

We **update our parameters** as if we were doing MLE:

I. Initial-state probabilities:

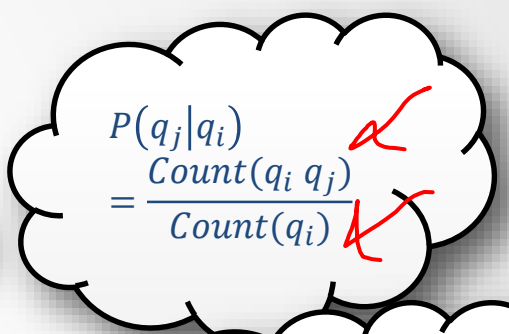
$$\hat{\pi}_i = \gamma_i(0) \quad \text{for } i := 1..N$$

II. State-transition probabilities:

$$\hat{a}_{ij} = \frac{\sum_{t=0}^{T-1} \xi_{ij}(t)}{\sum_{t=0}^{T-1} \gamma_i(t)} \quad \text{for } i, j := 1..N$$

III. Discrete observation probabilities:

$$\hat{b}_j(w) = \frac{\sum_{t=0}^{T-1} \gamma_j(t) |_{\sigma_t=w}}{\sum_{t=0}^{T-1} \gamma_j(t)} \quad \text{for } j := 1..N \text{ and } w \in \mathcal{V}$$


$$\frac{P(q_j|q_i)}{\text{Count}(q_i q_j)} = \frac{\text{Count}(q_i q_j)}{\text{Count}(q_i)}$$


$$\frac{P(w_i|q_i)}{\text{Count}(w_i \wedge q_i)} = \frac{\text{Count}(w_i \wedge q_i)}{\text{Count}(q_i)}$$

Baum-Welch iteration

- We update our parameters after **each iteration**

$$\theta_{k+1} = \langle \hat{\pi}_i, \hat{a}_{ij}, \hat{b}_j(w) \rangle$$

rinse, and repeat until $\theta_k \approx \theta_{k+1}$ (until change almost stops).

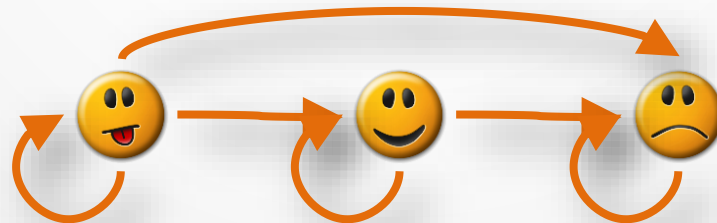
- Baum proved that

$$P(\mathcal{O}; \theta_{k+1}) \geq P(\mathcal{O}; \theta_k)$$

although this method does *not* guarantee a ***global maximum***.

Features of Baum-Welch

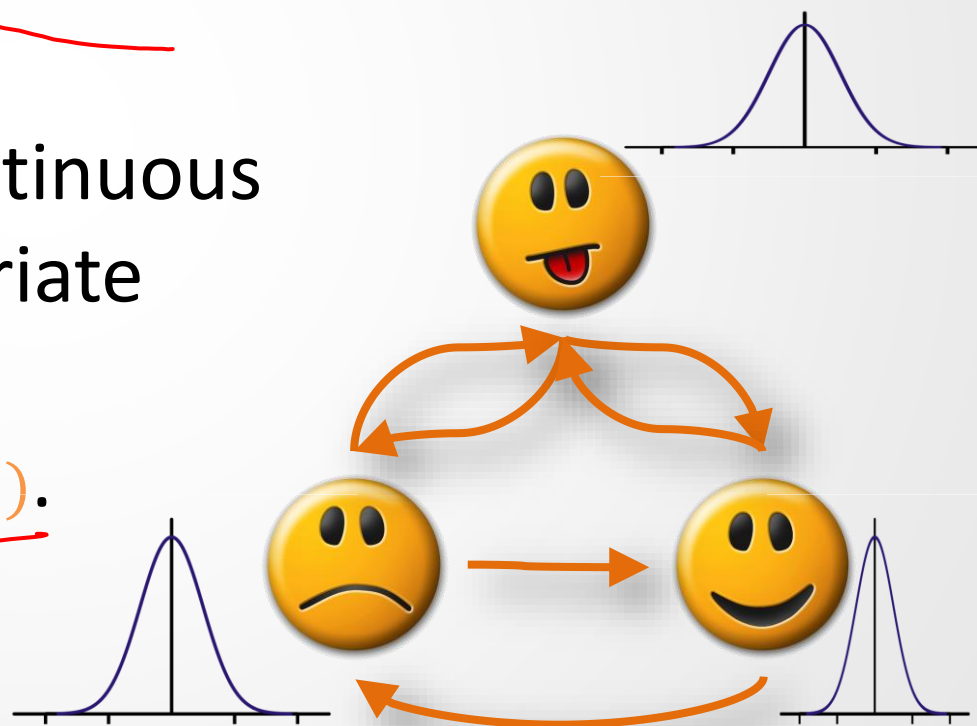
- Although we're **not guaranteed** to achieve a **global optimum**, the **local optima** are often 'good enough'.
- BW does **not** estimate the number of states, which must be 'known' beforehand.
 - Moreover, some constraints on topology are often imposed beforehand to assist training.



Discrete vs. continuous

- If our observations are drawn from a **continuous** space (e.g., speech acoustics), the probabilities $b_i(X)$ must also be continuous.

- HMMs **generalize** to continuous distributions, or multivariate observations, e.g., $b_i([-14.28, 0.85, 0.21])$.



Adaptation

- It can take a LOT of data to train HMMs.
- Imagine that we're given a trained HMM but not the data.
 - Also imagine that this HMM has been trained with data from **many** sources (e.g., many speakers).
- We want to use this HMM with a **particular new source** for whom we have **some** data (but not enough to fully train the HMM properly from scratch).
 - To be **more accurate for that source**, we want to change the original HMM parameters *slightly* given the new data.

Deleted interpolation

- For added robustness, we can combine estimates of a generic HMM, G , trained with **lots of data** from many sources with a specific HMM, S , trained with **a little data** from a single source.

$$P_{DI}(\sigma) = \lambda P(\sigma; \theta_G) + (1 - \lambda) P(\sigma; \theta_S)$$

- This gives us a model tuned to our target source (S), but with some general 'knowledge' (G) built in.
 - How do we pick $\lambda \in [0..1]$?

Deleted interpolation – learning λ

1. Initialize λ with an empirical or guessed estimate.
2. Given \mathcal{O}_a , which is **adaptation data** of which $\mathcal{O}_{a,j}$ is the j^{th} partition, and there are M partitions,
3. Update λ (the weight of model G) according to:



$$\hat{\lambda} = \frac{1}{M} \sum_{j=1}^M \frac{P(\mathcal{O}_{a,j}; \theta_G)}{P_{DI}(\mathcal{O}_a)}$$

We continue until λ and $\hat{\lambda}$ are sufficiently close.

Aside – Maximum a Posteriori (MAP)

- Given adaptation data \mathcal{O}_a , the MAP estimate is
$$\hat{\theta} = \operatorname{argmax}_{\theta} P(\mathcal{O}_a | \theta) P(\theta)$$
- If we can guess some structure for $P(\theta)$, we can use EM to estimate new parameters (or **Monte Carlo**).
- For continuous $b_i(\sigma)$, we use **Dirichlet distribution** that defines the hyper-parameters of the model and the **Lagrange method** to describe the change in parameters $\theta \Rightarrow \hat{\theta}$.

Summary

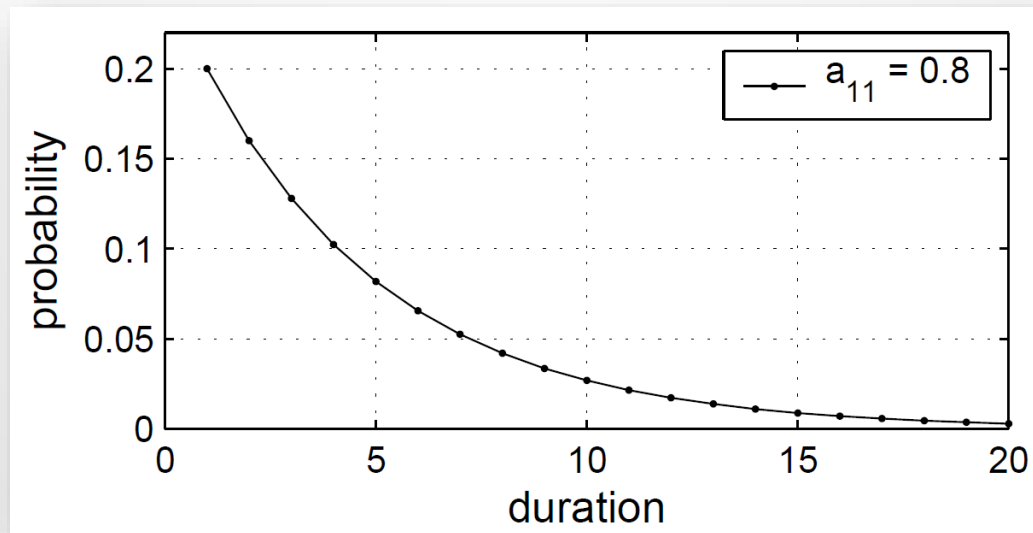
- Important ideas to know:
 - The definition of an HMM (e.g., its parameters).
 - The purpose of the **Forward algorithm**.
 - How to compute $\alpha_i(t)$ and $\beta_i(t)$
 - The purpose of the **Viterbi algorithm**.
 - How to compute $\delta_i(t)$ and $\psi_i(t)$.
 - The purpose of the **Baum-Welch algorithm**.
 - Some understanding of EM.
 - Some understanding of the equations.

Extras

State duration

- The probability of **staying** in a particular state s_i for a specific **period of time**, τ , diminishes exponentially over time, all else being equal.

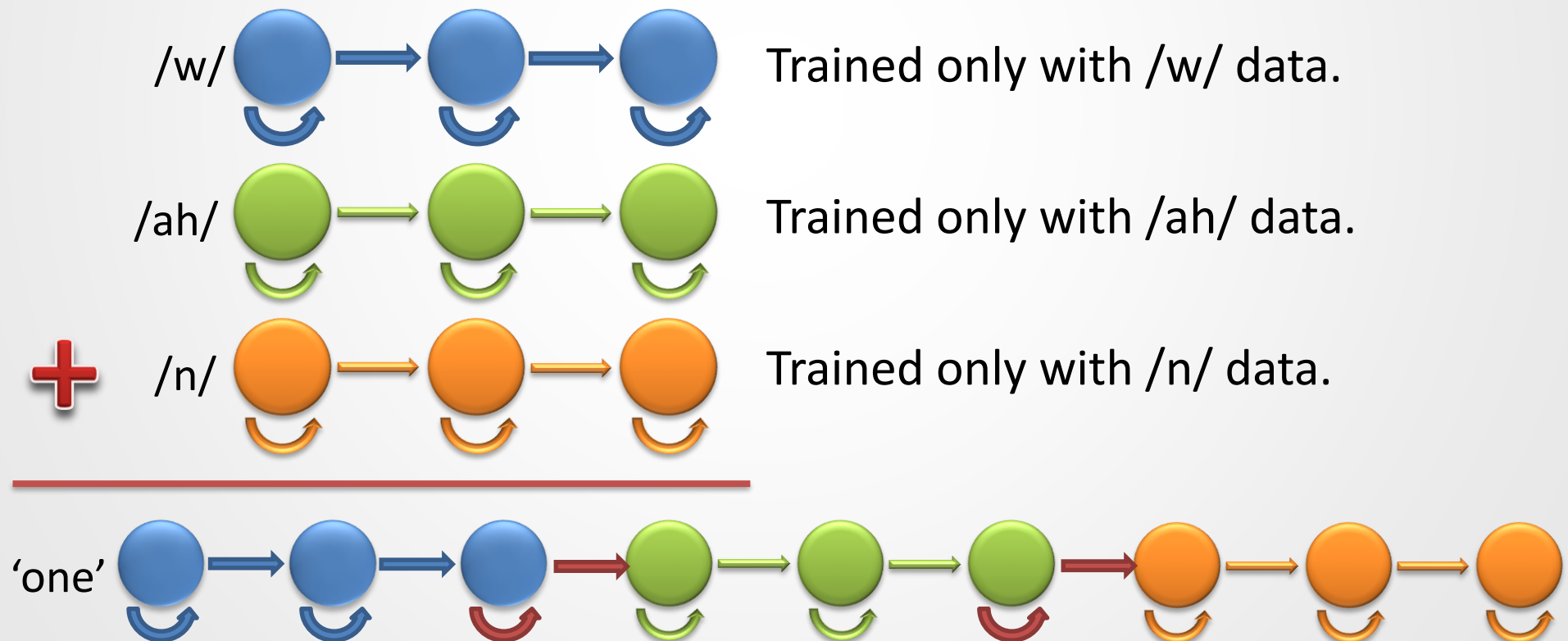
$$(a_{ii})^{\tau-1}(1 - a_{ii})$$



From Philip Jackson at
University of Surrey

Combining HMMs

- Often, we link HMMs together.
 - E.g., we have lots of speech data for /w/, /ah/, and /n/, but almost no data for the word 'one'.



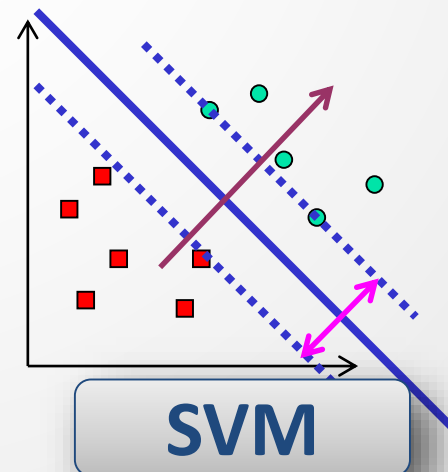
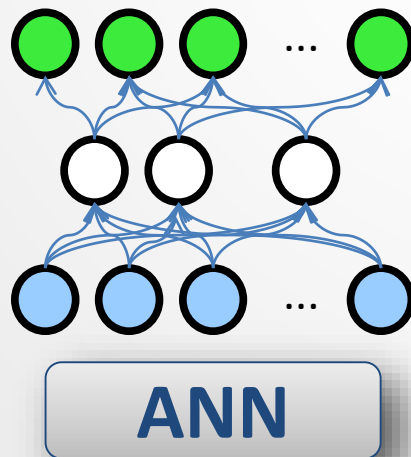
N-best lists

- In our discussion of the **Viterbi** algorithm, we encountered a situation where one state at time t was **equally likely** to have been reached from **two other** states at time $t - 1$.
- Sometimes instead of keeping track of only the **single** best path to state i at time t , we in fact keep track of the **N-best** paths to state i at time t .
 - E.g., in our Viterbi trellis:

δ : max probability	δ : 2 nd max probability	δ : 3 rd max probability
ψ : best backtrace	ψ : 2 nd best backtrace	ψ : 3 rd best backtrace

Generative vs. discriminative

- HMMs are **generative** classifiers. You can **generate** synthetic samples from because they model the phenomenon itself.
- Other classifiers (e.g., artificial neural networks and support vector machines) are **discriminative** in that their probabilities are trained specifically to reduce the error in classification.



Reading

- (optional) Manning & Schütze: Section 9.2—9.4.1
 - Note that they use another formulation...
- Rabiner, L. (1990) A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In: *Readings in speech recognition*. Morgan Kaufmann.
(posted on course website)
- Optional software:
 - Hidden Markov Model Toolkit (<http://htk.eng.cam.ac.uk/>)
 - Sci-kit's HMM (<http://scikit-learn.sourceforge.net/stable/modules/hmm.html>)