

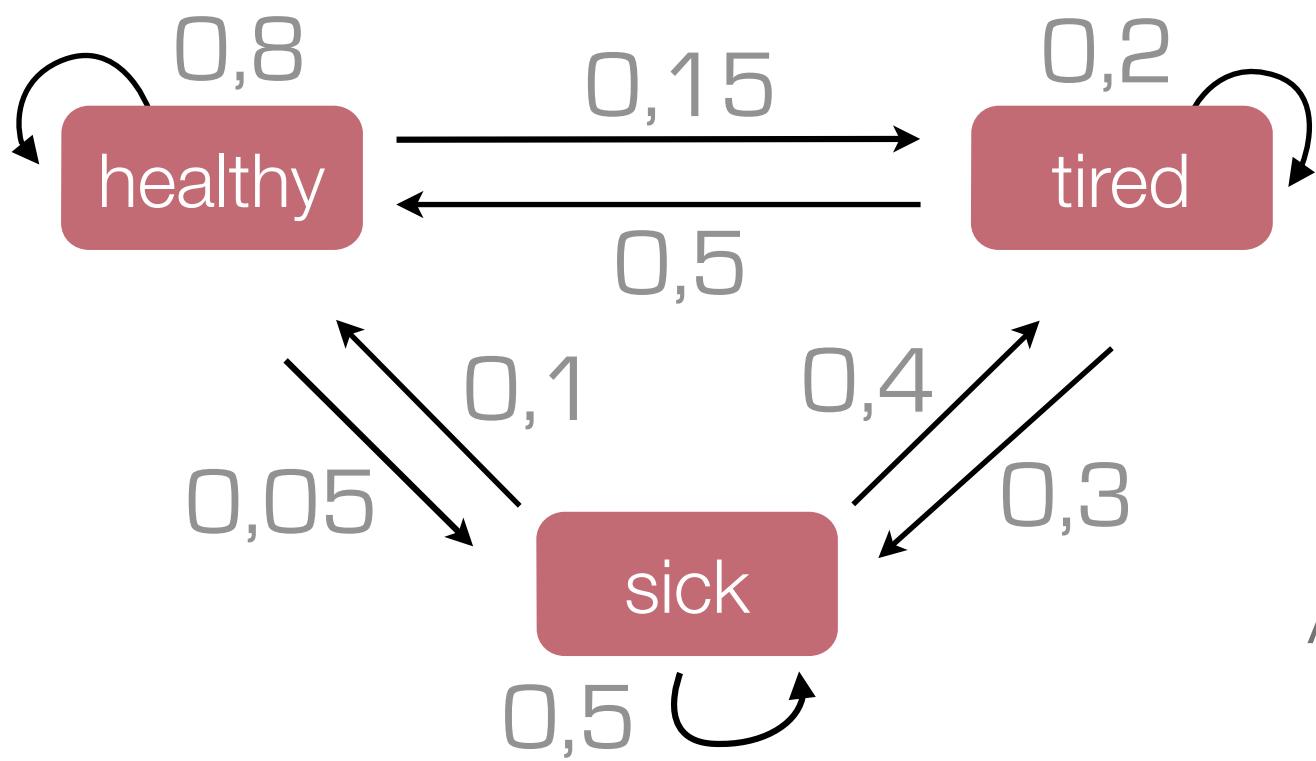
Hidden Markov Models

Thierry Parmentelat — INRIA

thierry.parmentelat@inria.fr

<https://github.com/parmentelat/mig2013>

Markov Models



- model : $\lambda=(A,\pi)$
- A: Transitions
- π : $p(\text{initial state}=i)$

A	<table border="1"><tr><td>0,8</td><td>0,15</td><td>0,05</td></tr><tr><td>0,5</td><td>0,2</td><td>0,3</td></tr><tr><td>0,1</td><td>0,4</td><td>0,5</td></tr></table> $\Sigma=1$	0,8	0,15	0,05	0,5	0,2	0,3	0,1	0,4	0,5
0,8	0,15	0,05								
0,5	0,2	0,3								
0,1	0,4	0,5								
π	<table border="1"><tr><td>0,7</td><td>0,2</td><td>0,1</td></tr></table>	0,7	0,2	0,1						
0,7	0,2	0,1								

Markov Models

- Markov hypothesis
 - behaviour depends only on current state (**not** on history)
- Observation
 - $E_1 E_2 \dots E_t \dots$: sequence of states, E_i in $\{1..N\}$
- Problems
 - (I) Probability of a given sequence
 - (II) Probability to observe state i at time t

Notations

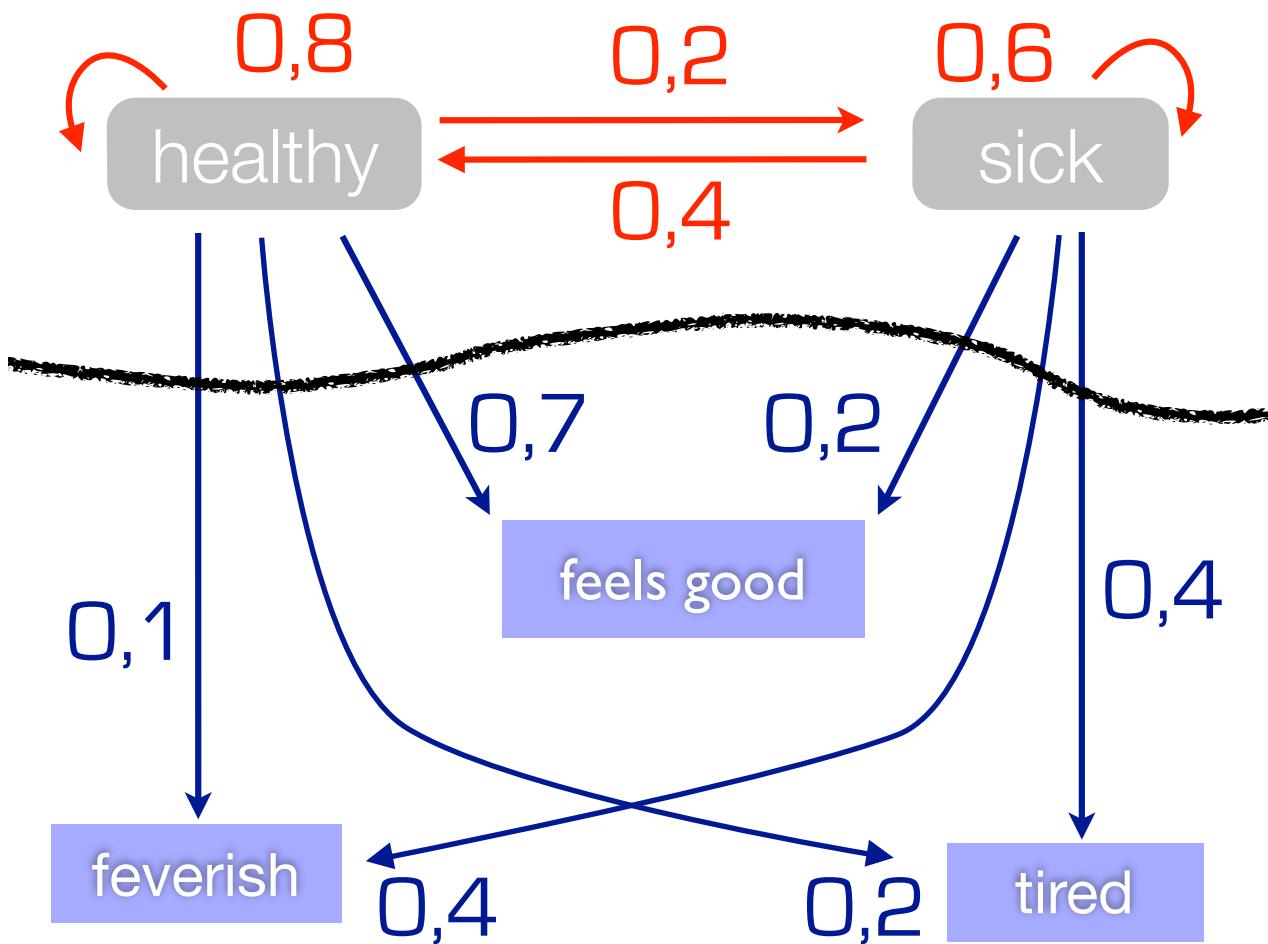
- Mathematical: A_{ij}
- Computer Science: $A[i][j]$
- Example Problem (I)
 - Maths: sequence $E_1 E_2 \dots E_t$
$$\text{proba} = \pi(E_1) * \prod_{i=2..t} A_{(E_{i-1})(E_i)}$$
 - CS: sequence $\text{seq}[t]$
$$\text{proba} = \prod_{i=2..t} (A[\text{seq}[i-1]] [\text{seq}[i]])$$

From now on,
we will have
all indices start at 0

Python features, and modules of interest

- range $\text{range}(5) \rightarrow [0, 1, 2, 3, 4]$ $\text{range}(1,5) \rightarrow [1, 2, 3, 4]$
- [function(x) for x in inputs] ou [function(x) for x in inputs if condition(x)]
- built-in sum $\text{sum}(\text{range}(5)) \rightarrow 10$
- module operator from operator import add, mul
- reduce $\text{reduce}(\text{add}, \text{range}(5), 0) \rightarrow 10$ $\text{reduce}(\text{mul}, \text{range}(1,5), 1) \rightarrow 24$
- numpy http://wiki.scipy.org/Tentative_NumPy_Tutorial
- matplotlib <http://matplotlib.org>

Hidden Markov Models



- Model : $\lambda = (A, \pi, E)$
- decoupling state-output

E

0,1	0,2	0,7
0,4	0,2	0,4

$\Sigma = 1$

Two major kinds of Hidden Markov Model

- Discrete: Emission among a **finite** set of observations
- Continuous: Emission in \mathbb{R}^d - second chapter

More on notations

- **sequence** : a list of observations (a.k.a. **seq**)
- **path** : a list of states
- indices:
 - **i,j** : for denoting states (**N**: nb of states)
 - **t** : for denoting time (**T**: max time for a given seq/path)
 - **o** : for denoting outputs/observations (**O**: nb of signals)

Classical problems in HMM's

- Reference paper - Lawrence Rabiner
A tutorial on Hidden Markov Models...
<http://www.cs.ubc.ca/~murphyk/Bayes/rabiner.pdf>
- (I) Probability to observe a given sequence
algos : **forward** and **backward** (a.k.a. alpha and beta)
- (II) Decoding: estimate (most probable) sequence of states that produce a given sequence - algo: **Viterbi**
- (III) Learning: given observed sequences, and from a given HMM, find an HMM that is more likely to produce them
algo: **Baum-Welch**

Probability for a given sequence of observations

- Naive algorithm:
 - probability for observing sequence along a given path
 - sum on all state paths

$$P(\text{seq}/\text{path}) = \pi(\text{path}_0) * \prod_{t=1}^{T-1} A_{\text{path}_{t-1}, \text{path}_t} * \prod_{t=0}^{T-1} E_{\text{path}_t, \text{seq}_t}$$

$$P(\text{seq}) = \sum_{\text{path} \in N^T} P(\text{seq}/\text{path})$$

- unpractical, we have N^T paths to sum on

Forward

- greedy variation
- **alpha[i][t]**
probability to observe seq **until t** and end up in **state i**
- initialization
$$\alpha_{i,0} = \pi_i * E_{i,seq_0}$$
- for i in range(N):
$$\text{alpha}[i][0] = P[i] * E[i][seq[0]]$$

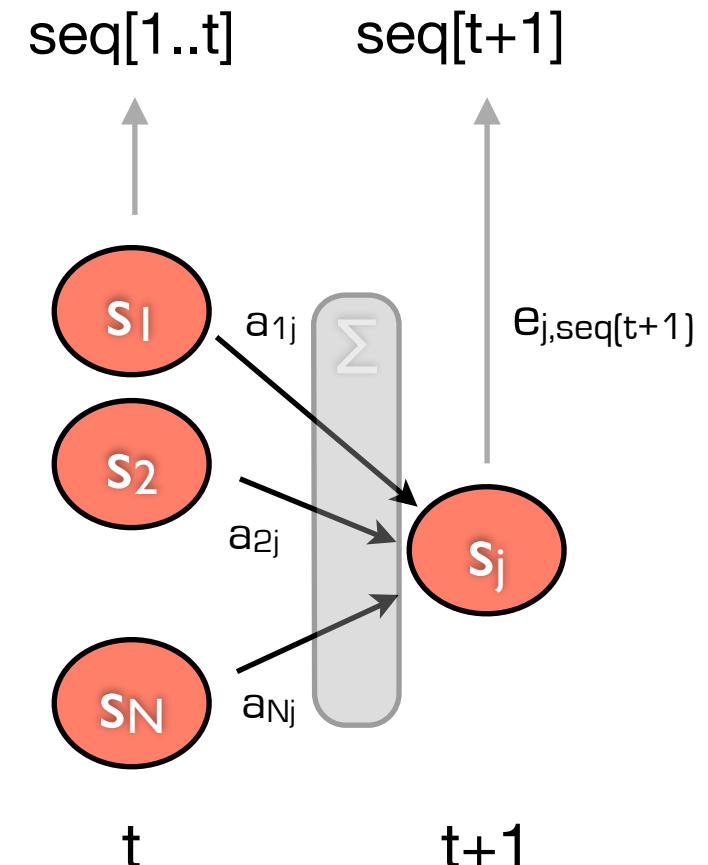
$$\alpha_{i,0} = \pi_i * E_{i,\text{seq}_0}$$

Forward - continued

- A: NxN
E: NxO
Pi: N
sequence: T

- Recurrence

$$\alpha_{j,t+1} = \sum_{i=0}^{N-1} \alpha_{i,t} * A_{i,j} * E_{j,\text{seq}_{t+1}}$$



- $\text{alpha}[j,t+1] = \backslash$
reduce (add, [$\text{alpha}[i,t] * A[i,j] * E[j,\text{seq}[t+1]]$] for i in range(N))

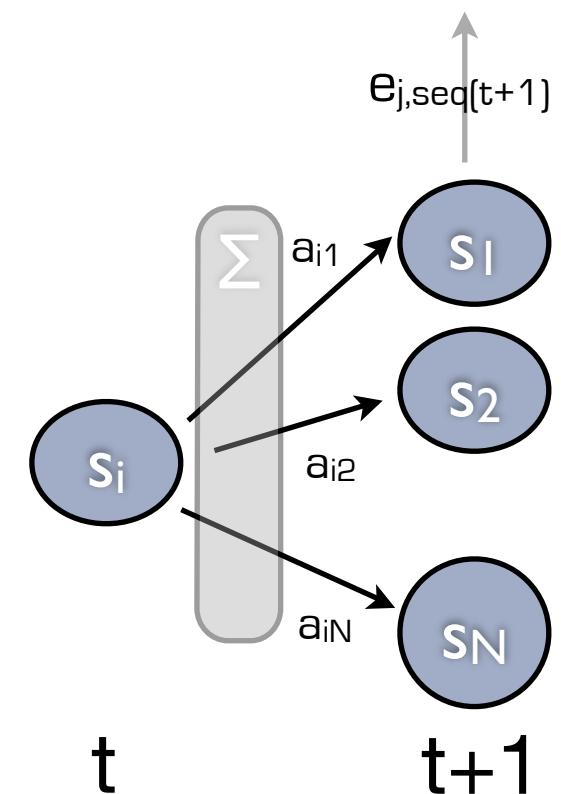
Forward - finalization

- Original purpose : compute proba (seq)
 - obtained by summing all $\alpha[i][T]$
- Note that the details of α are needed as well
 - as they will be re-used in the other algorithms
- Complexity of this algorithm
 - $\Theta(N^2 * T)$ in space and time

Backward

- Same idea as *forward*, except .. the other way around
- **beta[i][t]**
probability to be in **i** at **t** and observe seq **from t+1** and on
- $\text{beta}[i][T-1] = 1$
- $\text{beta}[i][t] = <\text{yours to say}>$
- final result

$$P(\text{seq}) = \sum_{i=0}^{N-1} \pi_i * E_{i,\text{seq}_0} * \text{beta}(i, 0)$$



Implementation

- Use your locally installed python (v2 recommended ?)
- To implement a function that can be used like this
 $(\text{proba}, \text{alpha}) = \text{forward}(\text{Pi}, \text{A}, \text{E}, \text{sequence})$
- Same for backward
- Using numpy or not - your call

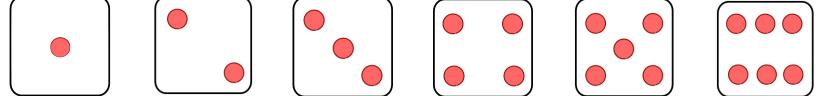
unfaircasino — a study model

2 states

A	<table border="1"><tr><td>0,9</td><td>0,1</td></tr><tr><td>0,1</td><td>0,9</td></tr></table>	0,9	0,1	0,1	0,9
0,9	0,1				
0,1	0,9				

fair dice

biased dice

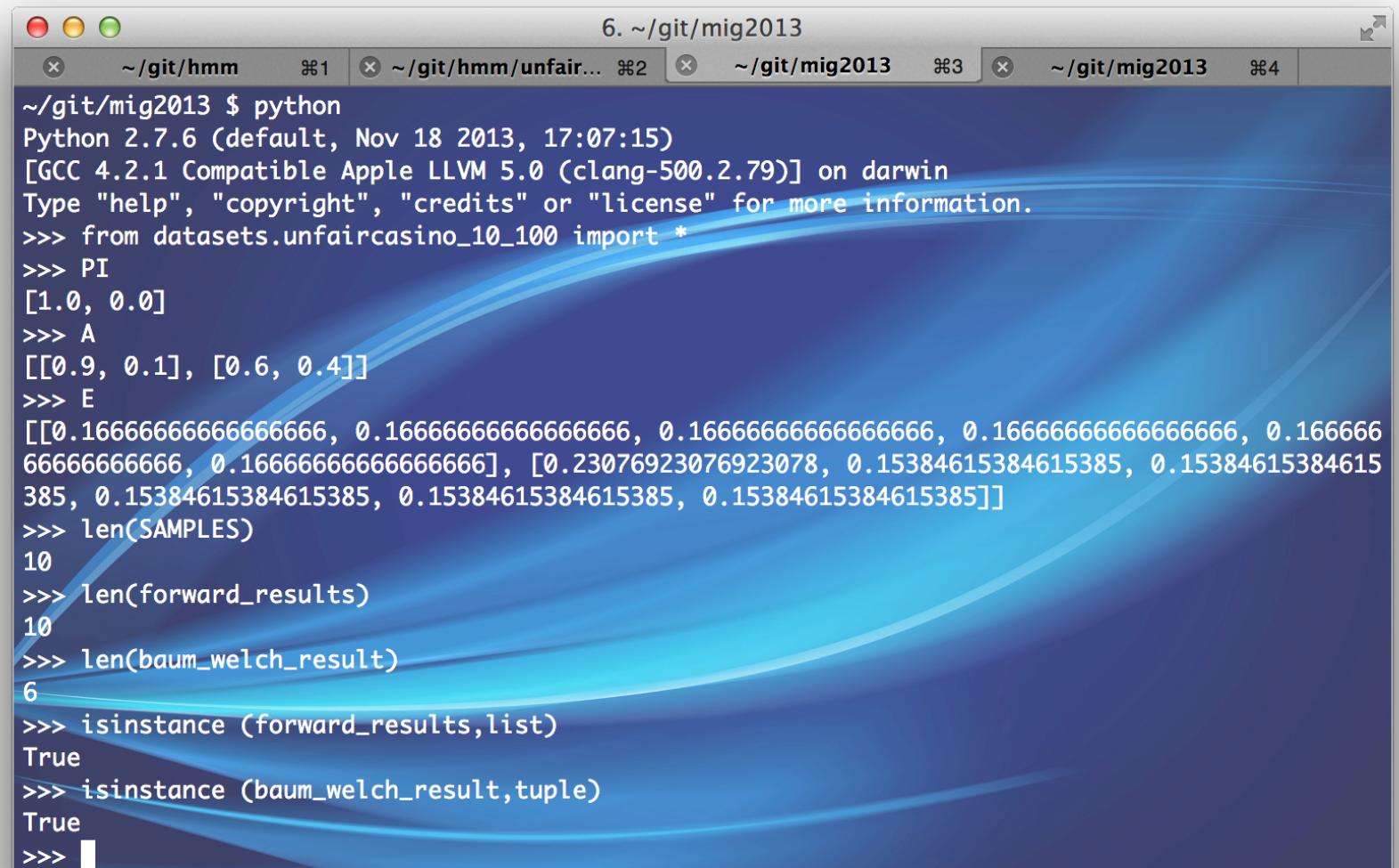
E	<table border="1"><tr><td>1/6</td><td>1/6</td><td>1/6</td><td>1/6</td><td>1/6</td><td>1/6</td></tr><tr><td>3/13</td><td>2/13</td><td>2/13</td><td>2/13</td><td>2/13</td><td>2/13</td></tr></table>	1/6	1/6	1/6	1/6	1/6	1/6	3/13	2/13	2/13	2/13	2/13	2/13	
1/6	1/6	1/6	1/6	1/6	1/6									
3/13	2/13	2/13	2/13	2/13	2/13									

starting with the fair dice

π	<table border="1"><tr><td>1</td><td>0</td></tr></table>	1	0
1	0		

Available datasets

- For comparing your results: see datasets/



The screenshot shows a Mac OS X terminal window titled "6. ~/git/mig2013". The window has four tabs at the top: "~/git/hmm" (⌘1), "~/git/hmm/unfair..." (⌘2), "~/git/mig2013" (selected, ⌘3), and "~/git/mig2013" (⌘4). The main pane displays the following Python session:

```
~/git/mig2013 $ python
Python 2.7.6 (default, Nov 18 2013, 17:07:15)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.2.79)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from datasets.unfaircasino_10_100 import *
>>> PI
[1.0, 0.0]
>>> A
[[0.9, 0.1], [0.6, 0.4]]
>>> E
[[0.1666666666666666, 0.1666666666666666, 0.1666666666666666, 0.1666666666666666, 0.1666666666666666, 0.1666666666666666, 0.1666666666666666], [0.23076923076923078, 0.15384615384615385, 0.15384615384615385, 0.15384615384615385, 0.15384615384615385, 0.15384615384615385, 0.15384615384615385]]
>>> len(SAMPLES)
10
>>> len(forward_results)
10
>>> len(baum_welch_result)
6
>>> isinstance (forward_results,list)
True
>>> isinstance (baum_welch_result,tuple)
True
>>> █
```

Two specific variables are highlighted with blue boxes: "SAMPLES[0]" and "forward_result[0]".

Viterbi – decoding

- Given an observation sequence
find out most likely path that has caused this sequence
a.k.a. **Viterbi path**
- **delta[i][t]**
best probability of path **until t** that **ends up at i**
(and of course emits sequence until t)
- **psi[i][t]**
state at t-1 for the path that corresponds to *delta*
 $\psi[i][0]$ undefined / does not matter

Viterbi – details

- Initialization

$$\delta[i][0] = \pi[i] * E[i][\text{sequence}[0]]$$

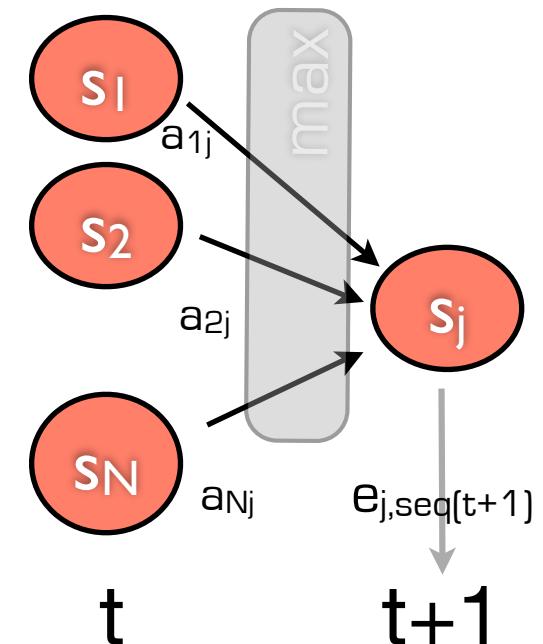
$$\psi[i][0] = \text{None}$$

- Recurrence

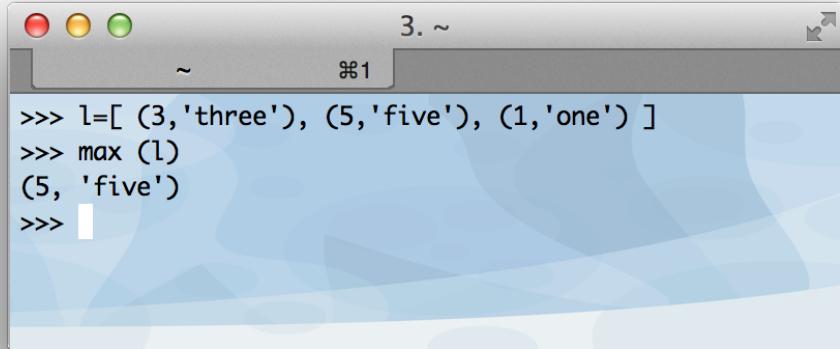
$$\delta[j][t+1] = \backslash$$

max ([$\delta[i][t] * A[i][j]$] for i in range(N)) * $E[j][\text{seq}[t+1]]$

$$\psi[j][t] = \text{argmax} (\text{same expression})$$



Viterbi – a useful trick

A screenshot of a terminal window titled "3. ~" with the number "⌘1" in the top right corner. The window has three colored status icons (red, yellow, green) in the top left. The terminal contains the following Python code:

```
>>> l=[(3,'three'), (5,'five'), (1,'one')]  
>>> max(l)  
(5, 'five')  
>>> |
```

The cursor is positioned at the end of the last line.

$(\text{proba}, \text{rank}) = \max \left([(\delta[i][t-1] * A[i][j], i) \text{ for } i \in \text{range}(N)] \right)$

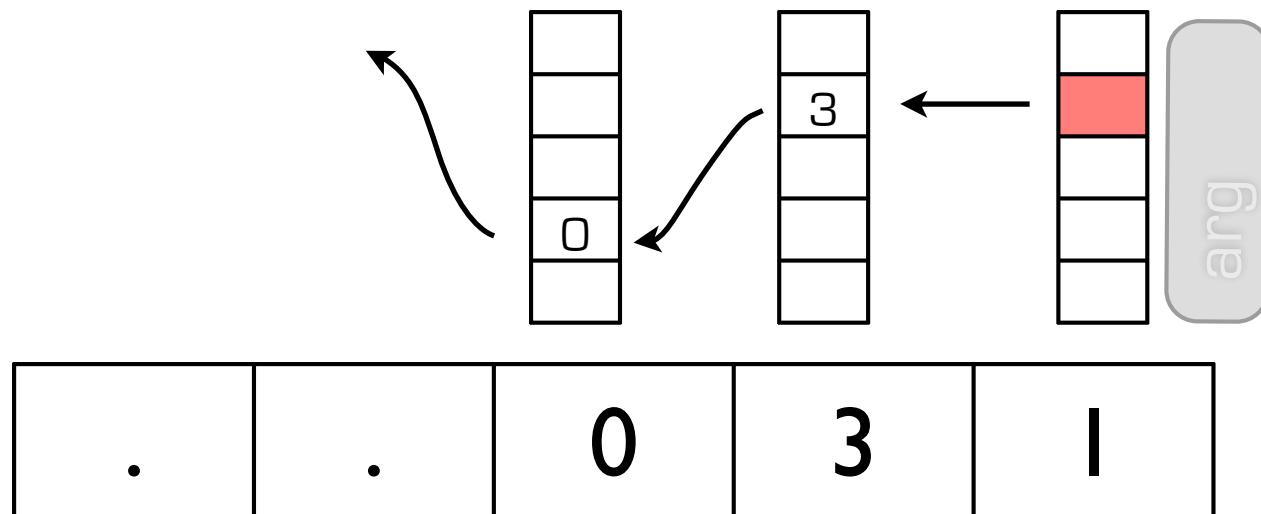
$\delta[j][t] = \text{proba} * E[j][\text{seq}[t]]$

$\psi[j][t] = \text{rank}$

Viterbi - finalization

- retrieve full path

$\psi[T-2] \ \psi[T-1] \ \delta[T-1]$



- Viterbi is a greedy algorithm too

- $\Theta(N^2 * T)$ in space and time

Implementation

- implement a function that can be used like this
 $(\text{proba}, \text{path}, \text{delta}, \text{psi}) = \text{viterbi}(\text{Pi}, \text{A}, \text{E}, \text{sequence})$
- (although delta and psi are not really useful outside)

One possible implementation for forward

```
# iterative style
def forward (Pi, A, E, sequence):
    N = len(A)
    T = len(sequence)
    states = range(N)

    alpha = [ [ 0. for t in range(T) ] for i in states ]

    # time=0
    for i in states: alpha[i][0]=Pi[i] * E[i][sequence[0]]

    # fill in from time=1
    for t in range (1,T):
        for j in states:
            for i in states:
                alpha[j][t] += alpha[i][t-1]*A[i][j]
            # multiply once rather than for each iteration
            alpha[j][t] *= E[j][sequence[t]]
    # resulting proba is a sum from that last time column
    total = 0.
    for i in states: total += alpha[i][T-1]
    return (total, alpha)
```

One possible implementation for backward

```
from operator import add
# return ( proba, beta ) -- beta dimension: beta[i][t]
# functional style
def backward (Pi, A, E, sequence):
    N = len(A)
    T = len(sequence)
    states = range(N)

    beta = [ [ 0. for t in range(T) ] for i in states ]

    # beta starts with 1's on last time
    for i in states: beta[i][T-1]=1.

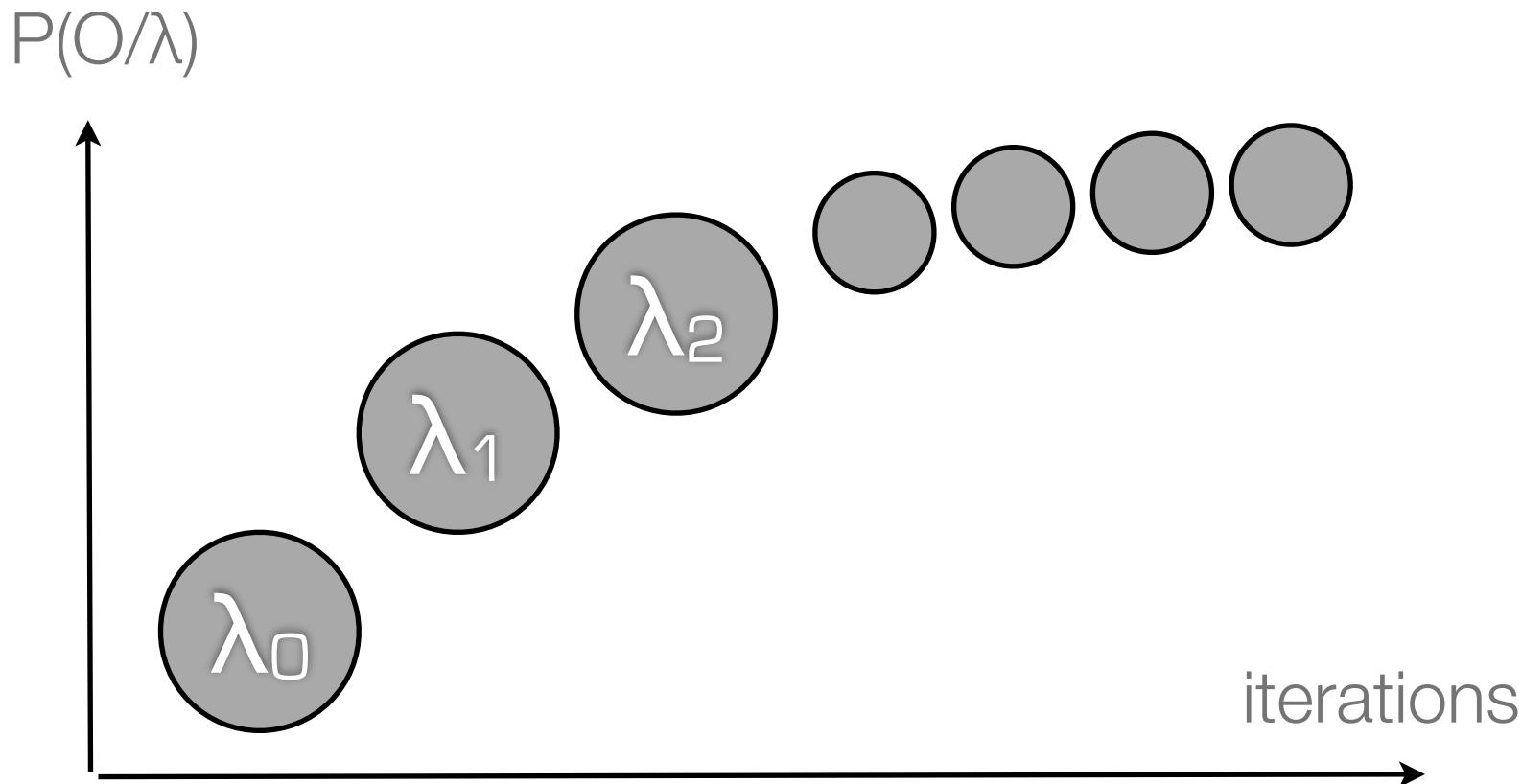
    # starting with time but last, i.e. T-2
    for t in range (T-2, -1, -1):
        for i in range (N):
            beta[i][t] =
                reduce (add, [ beta[j][t+1] * A[i][j] * E[j][sequence[t+1]] for j in states ] )
    # resulting proba, summing on time=0
    total = reduce (add, [ Pi[i]*E[i][sequence[0]]*beta[i][0] for i in states ] )
    return (total,beta)
```

Baum-Welch – learning

- Problem: given a set of output sequences
Find out the “best” HMM that would give this out
- Improve model $\lambda = (A, E, \text{PI})$ by successive iterations
- maximize likelihood over λ
likelihood defined as the **product** of probabilities of all sequences in a sample
- in our case, multiply $P(O/\lambda)$ over sequences

Expectation Maximization (EM)

- Baum-Welch is a special case of “Expectation - Maximization” class of algorithms



Baum Welch basics

- Given sequence seq, and model λ
- $x_{i,j}[t] \xi_t[i,j]$
probability to be in state **i** at time **t**, and in state **j** at **t+1**
- **gamma[i][t]** $\gamma_t[i]$
probability to be in state **i** at time **t**

computing xi and gamma

- $\xi_t(i, j)$ probability to be in **i** at **t**, **j** at **t+1**(knowing seq)

$$\xi_t(i, j) = \frac{\alpha_t(i) A_{i,j} E_{j, \text{seq}_{t+1}} \beta_{t+1}(j)}{P(\text{seq}/\lambda)}$$



- $\gamma_t(i)$ probability to be in **i** at **t** (knowing seq)

$$\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i)(j)$$

Rationale for re-estimation

$$\sum_{t=0}^{T-2} \gamma_t(i) = \text{expected number of transitions from state } i$$

$$\sum_{t=0}^{T-2} \xi_t(i,j) = \text{expected number of transitions from state } i \text{ to } j$$

- note that one instant is not taken into account
we are counting transitions, so there are T-1
will be used for re-estimating PI instead

Re-estimation

$$\bar{\pi}_i = \gamma_0(i)$$

$$\bar{A}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i}$$

$$= \frac{\sum_{t=0}^{T-2} \xi_t(i, j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$$

$$\bar{E}_{jo} = \frac{\text{expected number of times in } j \text{ and observing } o}{\text{expected number of times in } j}$$

$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j)}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

so that $o = seq(t)$

reminder from previous slides

* in **i** at **t** and in **j** at **t+1**

$$\xi_t(i, j) = \frac{\alpha_t(i) A_{i,j} E_{j,seq_{t+1}} \beta_{t+1}(j)}{P(seq/\lambda)}$$

* in **i** at **t**

$$\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i)(j)$$

Convergence criteria

- monitor overall $P(\text{seq})$ at each iteration
- should grow asymptotically
- stop when $(P'/P) < (1+\varepsilon)$
- ε passed as argument

Deal with multiple sample sequences

$$\bar{\pi}_i = \gamma_0(i)$$

Average on sequences

$$\bar{A}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i}$$

Sum over sequences

$$= \frac{\sum_{t=0}^{T-2} \xi_t(i, j)}{\sum_{t=0}^{T-1} \gamma_t(i)}$$

Sum over sequences

$$\bar{E}_{jo} = \frac{\text{expected number of times in } j \text{ and observing } o}{\text{expected number of times in } j}$$

Sum over sequences

$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j) \text{ so that } o = seq(t)}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

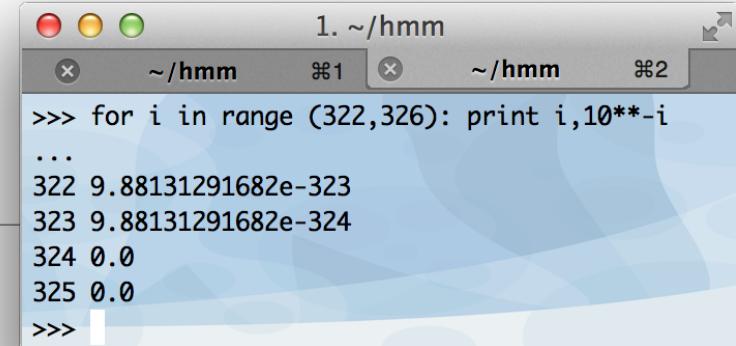
Sum over sequences

Implementation

- implement a function that can be used like this

```
(Pi', A', E', proba, iteration,time) = \  
baum_welch (Pi, A, E,sequenceS,\  
max_iters=100, convergence=1.e-4)
```

Scaling



```
1. ~/hmm
~/hmm ~1 ~hmm ~2
>>> for i in range (322,326): print i,10**-i
...
322 9.88131291682e-323
323 9.88131291682e-324
324 0.0
325 0.0
>>>
```

- If P_{\max} is the maximum of the terms in A and E
- Order of magnitude for e.g. $\alpha[i][t]$: P_{\max}^{2T}
- Even if P_{\max} is, say, 0.9, $T=300$ gives you fairly small values
- Quickly reach limits of hardware implementation
- Solution: for each time t
 - select some relevant factor (e.g. based on sum of values)
 - store in memory real value multiplied by factor

Continuous Hidden Markov Models

Continuous Hidden Markov Model

- Replace finite set of O signals (outputs)
- With a finite set of O gaussian distributions in R^d
- Each of these being defined by
 - an average $\mu = \{\mu_1, \mu_2, \dots \mu_d\}$
 - a covariance matrix σ_{ij} in dimension d
- Then each state is attached a linear stochastic combination of these gaussians (a **mixture**)

Continuous HMM – example

- 3 Outputs:

$$G1 : \mu=(4., 8.), \quad \sigma=(1., 0.25)$$

$$G2 : \mu=(-4., -8.), \quad \sigma=(1., 4.)$$

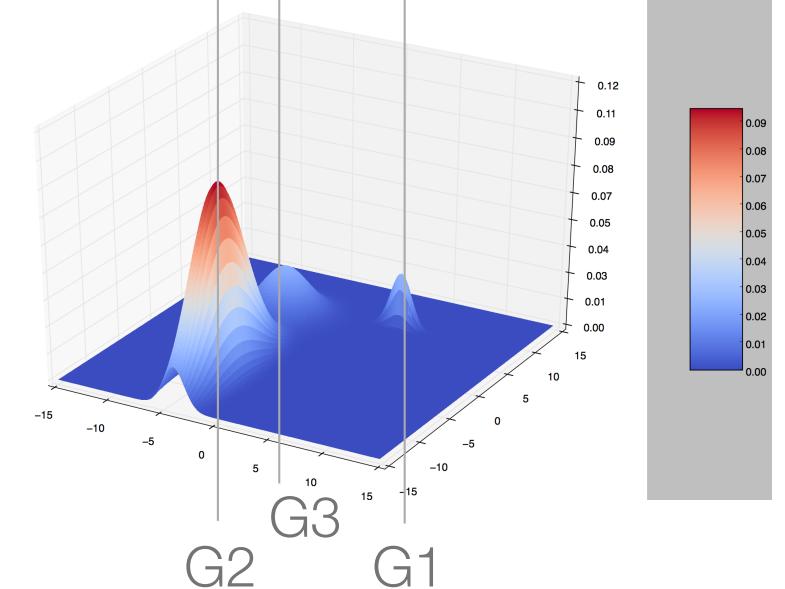
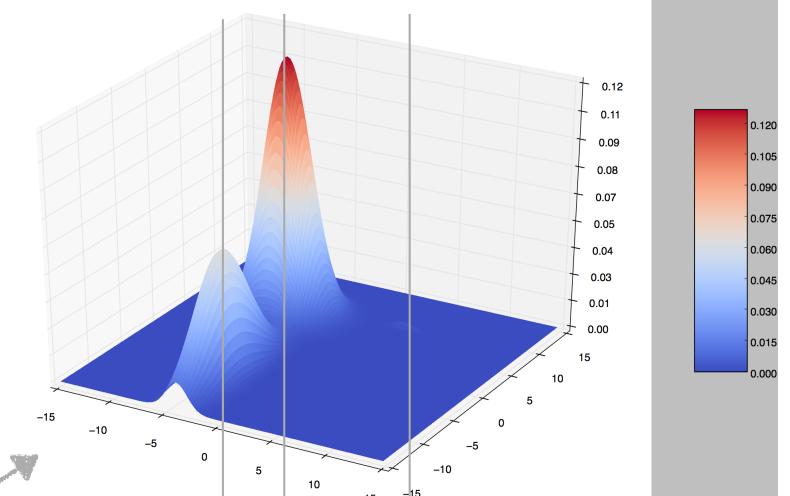
$$G3 : \mu=(-6., 6.), \quad \sigma=(2., 2)$$

say $G1 \sim$ tired,
 $G2 \sim$ feverish, $G3 \sim$ feels good

- 2 states:

$$E[1]: (.1, .1, .8)$$

$$E[2]: (.7, .15, .15)$$



Gaussian distribution - single dimension

- a.k.a. Normal Distribution (wiki)

- basic shape

$$f(x) = e^{-x^2}$$

- normalize

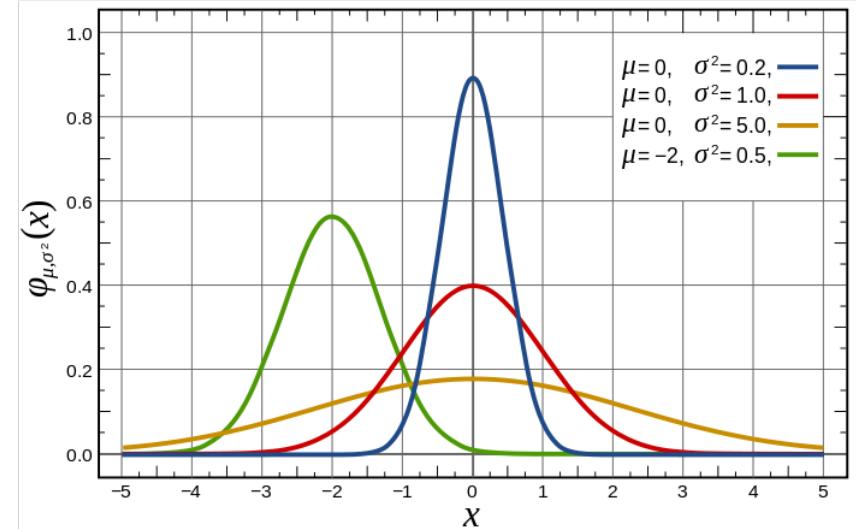
$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

- translate

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-\mu)^2}$$

- scale

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



General form of d-dimension gaussian

- a.k.a. multivariate normal distribution (wiki)
- μ average; Σ covariance
 Σ is positive semidefinite and symmetric

$$\mu \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d,d}, G_{\mu,\Sigma} : \mathbb{R}^d \mapsto \mathbb{R}$$

$$G_{\mu,\Sigma}(X) = \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} e^{(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu))}$$

- check out “covariance matrix” in wiki

Degenerate case

- Diagonal covariance matrix (unrelated data)

$$\Sigma = \begin{pmatrix} {\sigma_1}^2 & 0 & \cdot & \cdot & 0 \\ 0 & {\sigma_2}^2 & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & {\sigma_{d-1}}^2 & 0 \\ 0 & \cdot & \cdot & 0 & {\sigma_d}^2 \end{pmatrix}$$

$$G_{\mu, \Sigma}(X) = \frac{1}{\prod_{i=1}^d \sigma_i \sqrt{(2\pi)^d}} e^{(-\frac{1}{2} \sum_{i=1}^d (\frac{x_i - \mu_i}{\sigma_i})^2)}$$

Continuous HMM theory (See Rabiner p. 267)

- each state has a pdf that is a mixture of \mathbf{m} Gaussians
- in the most general framework where an estimation method is known, not all m Gaussians are equal

$$p_j(X) = \sum_{m=0}^{M-1} c_{jm} G_{\mu_{jm}, \Sigma_{jm}}[X] \quad \sum_m c_{jm} = 1$$

that is a total of $\mathbf{n} * \mathbf{m}$ gaussians in the model

- **Note** the example above considers \mathbf{m} gaussians in total

$$p_j(X) = \sum_{m=0}^{M-1} c_{jm} G_{\mu_m, \Sigma_m}[X] \quad \sum_m c_{jm} = 1$$

Model summary - case (2)

- Π (N)
- A (N x N)
- E (N x O)
- seq (T)
- Π (N)
- A (N x N)
- C (M x N)
- G_{μ} (M x D)
- G_{σ} (M x D x D)
- seq (T x D)

Adapting discrete to continuous - α , β & viterbi

- For forward and backward
 - we had terms like “ $E[j][seq[t]]$ ”
 - that corresponds to the probability to emit output $seq[t]$ in state j
- now $seq[t]$ is a d-dimension vector
- we can compute this probability directly from corresponding gaussian mixture

$$p_j(X) = \sum_{m=0}^{M-1} c_{jm} G_{\mu_m, \Sigma_m}[X]$$

Re-estimation in the discrete case



$$\bar{\pi}_i = \gamma_0(i)$$



$$\bar{A}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i}$$

$$= \frac{\sum_{t=0}^{T-2} \xi_t(i, j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$$



$$\bar{E}_{jo} = \frac{\text{expected number of times in } j \text{ and observing } o}{\text{expected number of times in } j}$$

$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j) \quad \text{so that } o = \text{seq}(t)}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

reminder from previous slides

* in i at t and in j at $t+1$

$$\checkmark \quad \xi_t(i, j) = \frac{\alpha_t(i) A_{i,j} E_{j,\text{seq}_{t+1}} \beta_{t+1}(j)}{P(\text{seq}/\lambda)}$$

* in i at t

$$\checkmark \quad \gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i)(j)$$

Adapting discrete to continuous - Baum-Welch

- **gamma[i,t,m]** probability to be in state **i** at time **t** with mixture **m** being responsible for that output

$$\gamma_t(i, m) = \left[\frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)} \right] \left[\frac{c_{im} G_{\mu_{im}, \sigma_{im}}(X_t)}{\sum_{k=0}^{M-1} c_{ik} G_{\mu_{ik}, \sigma_{ik}}(X_t)} \right]$$

Re-estimation rules for continuous Baum-Welch

$$\overline{c_{im}} = \frac{\sum_t \gamma_t(i, m)}{\sum_t \sum_k \gamma_t(i, k)}$$

$$\overline{\mu_{im}} = \frac{\sum_t \gamma_t(i, m) \cdot X_t}{\sum_t \gamma_t(i, m)}$$

$$\overline{\Sigma_{im}} = \frac{\sum_t \gamma_t(i, m) \cdot (X_t - \overline{\mu_{im}}) \cdot (\overline{X_t} - \overline{\mu_{im}})}{\sum_t \gamma_t(i, m)}$$

reminder from previous slide

* in **i** at **t** and in from mixture **m**

$$\gamma_t(i, m) = \left[\frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)} \right] \left[\frac{c_{im} G_{\mu_{im}, \sigma_{im}}(X_t)}{\sum_{k=0}^{M-1} c_{ik} G_{\mu_{ik}, \sigma_{ik}}(X_t)} \right]$$

Source Code Management systems

Source Code Management systems

- purpose
 - collaborative work
 - keep track of changes
 - changes are done concurrently
 - teams do not necessarily know of each other
- git
 - <http://git-scm.com>
 - decentralized
 - used by many open source projects
- svn
 - <http://subversion.tigris.org>
 - deprecated / previous generation
 - simpler mental model, but centralized
- mercurial
 - <http://mercurial.selenic.com> is nice too

git basics

- git init initialize git from a working directory
- git clone create a local working dir from remote
- git add put changes aside for next commit
- git commit create a commit
- git pull get changes from remote
- git push push your commits on remote
- git stash hide local changes under carpet

Check out UI tools : <http://www.sourcetreeapp.com>

example (1) - create repo in github

The screenshot shows a GitHub repository page for 'parmentelat/mig2013'. The repository is public and contains 1 commit, 1 branch, 0 releases, and 1 contributor. The 'master' branch is selected. The repository has a README.md file with the text 'mig2013' and 'Module d'intégration Systèmes Embarqués 2013'. The right sidebar includes links for Code, Issues, Pull Requests, Wiki, Pulse, Graphs, Network, and Settings. It also provides an HTTPS clone URL and options to Clone in Desktop or Download ZIP.

PUBLIC parmentelat / mig2013

Module d'intégration Systèmes Embarqués 2013 — Edit

1 commit 1 branch 0 releases 1 contributor

branch: master mig2013 / +

Initial commit

parmentelat authored 20 hours ago latest commit 8b71341318

.gitignore Initial commit 20 hours ago

LICENSE Initial commit 20 hours ago

README.md Initial commit 20 hours ago

README.md

mig2013

Module d'intégration Systèmes Embarqués 2013

Code Issues Pull Requests Wiki Pulse Graphs Network Settings

HTTPS clone URL https://github.com/

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop Download ZIP

© 2013 GitHub, Inc. Terms Privacy Security Contact

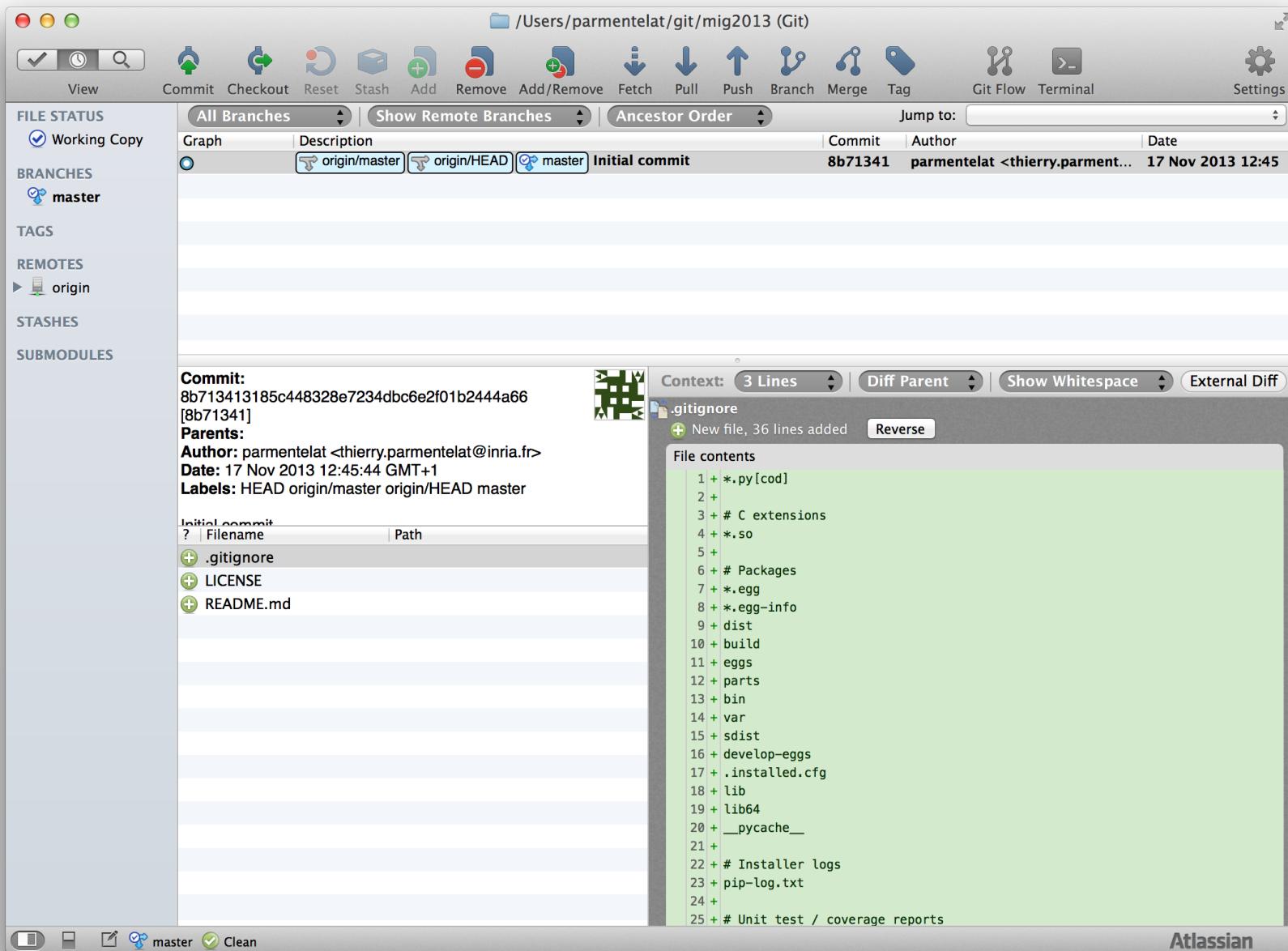
Status API Training Shop Blog About

example (2) - clone repo locally

The image shows a terminal window titled "3. ~/git/mig2013". The window has three tabs at the top, with the third tab active. The title bar also displays the path "3. ~/git/mig2013". The terminal window contains the following text:

```
~/git $ git clone https://github.com/parmentelat/mig2013.git
Cloning into 'mig2013'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
Checking connectivity... done
~/git $ cd mig2013/
~/git/mig2013 $ ls
LICENSE README.md
~/git/mig2013 $
```

example (3) - visualize local repo with sourcetree

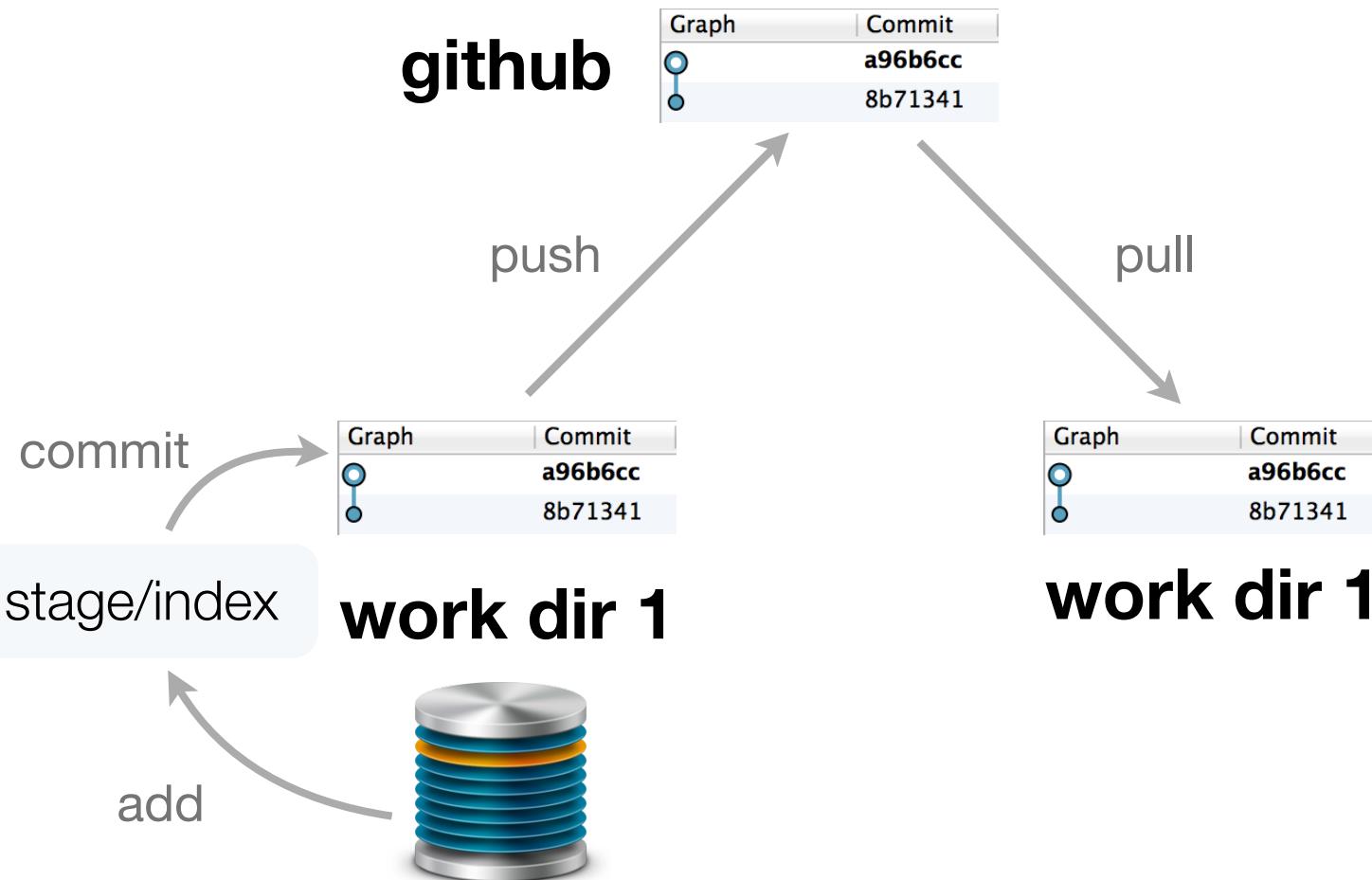


example (4) - two working directories

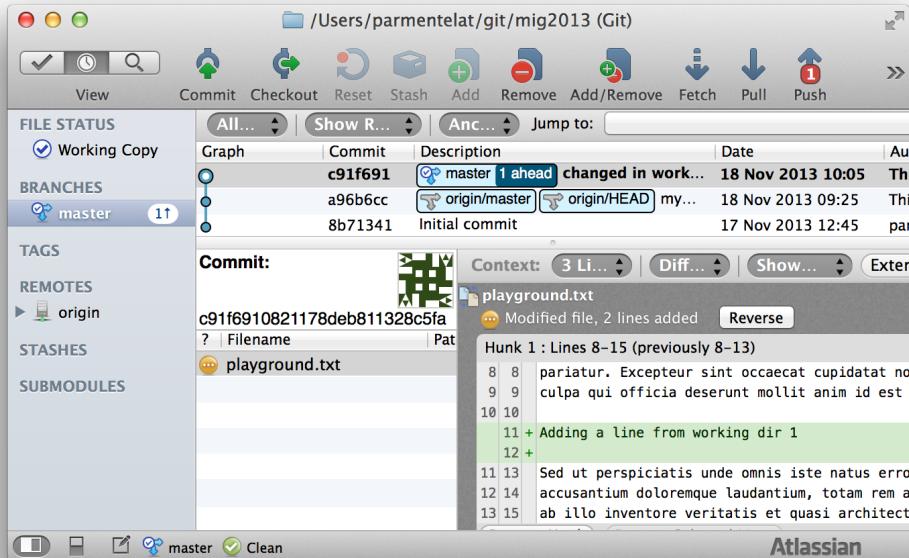
```
$ git ls-files playground.txt  
$ git add playground.txt  
$ git commit -m "my message"  
$ git ls-files playground.txt  
playground.txt  
$ git push
```

```
$ git ls-files playground.txt  
$  
$  
$ git ls-files playground.txt  
$  
$  
$ git pull  
$ git ls-files playground.txt  
$ playground.txt
```

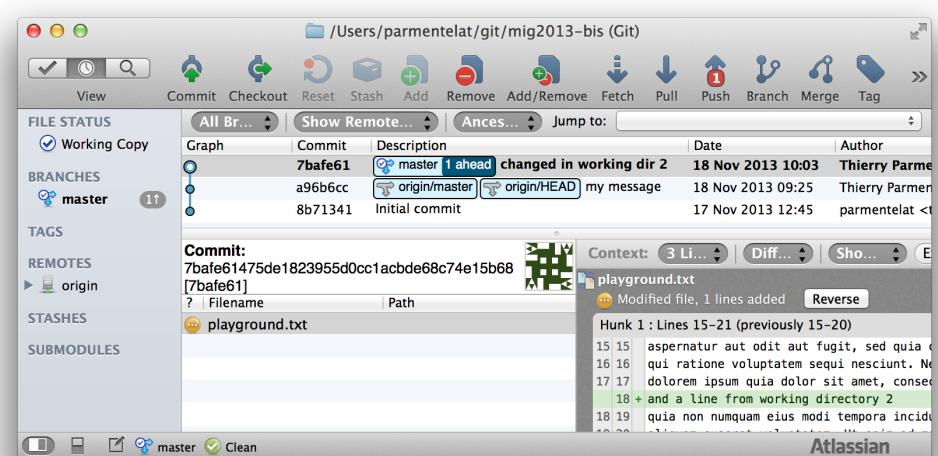
example (5) - the repositories at work



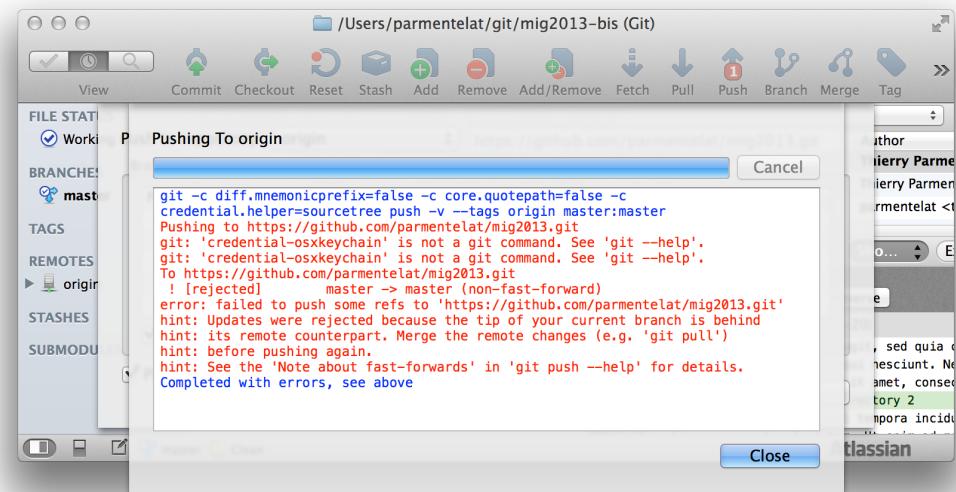
example (6) - merging two concurrent changes



user 1 can push fine

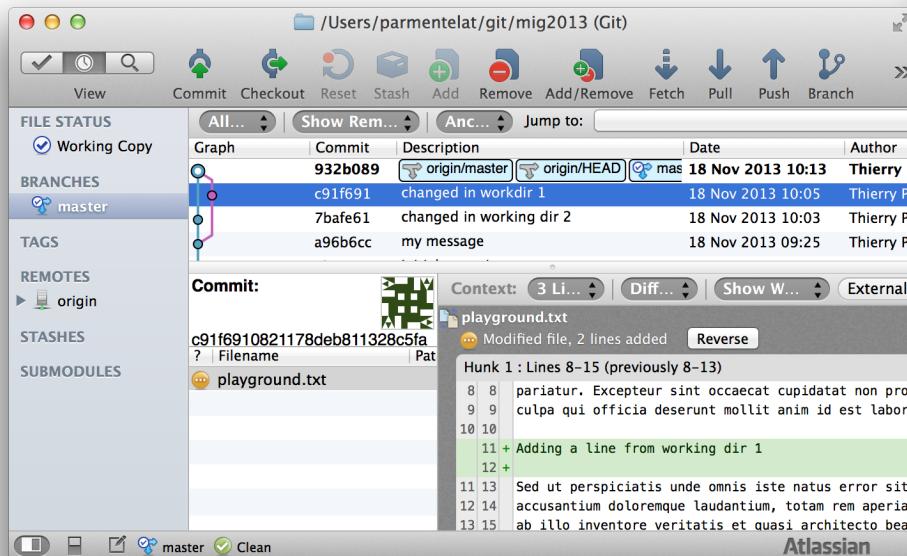


user 2 gets conflicts when pushing

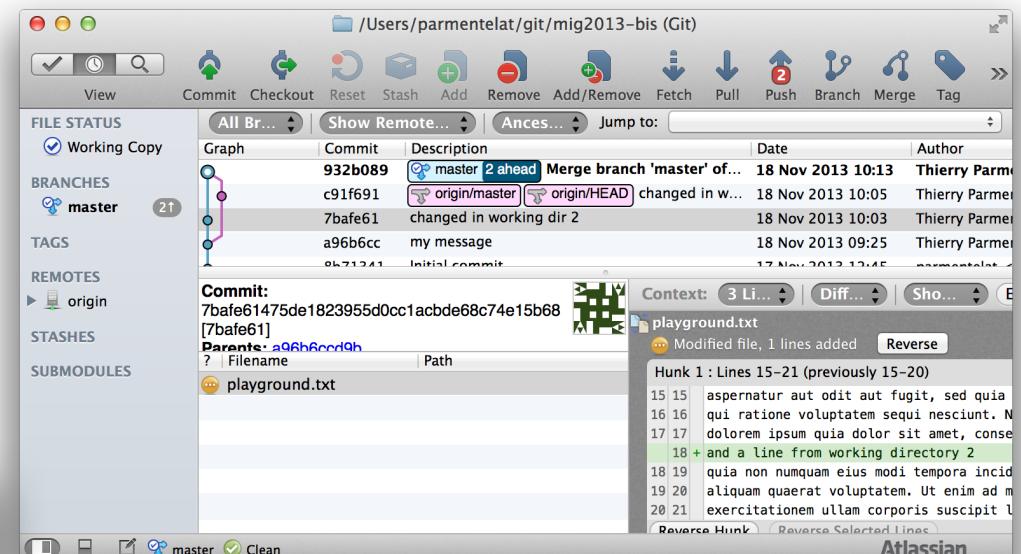


example (7) - merging - continued

user 1 sees once he pulls again



user 2 needs to pull first



user 2 can then push fine

Good practices

- Branches are easy to create, merge, delete..
- Use as many branches as needed
- Typically one per “feature”

The screenshot shows the Atlassian SourceTree application interface for managing a Git repository named 'sfa'. The main window displays a 'Graph' view of the commit history, showing multiple branches (geni-v2 and geni-v3) merging into a single 'geni-v3' branch. The commit list on the right shows numerous commits from various authors, including Tony Mack, Mohamed Larabi, and Sandrine Avakian, spanning from November 2013 to April 2014. The bottom half of the screen shows a detailed view of a specific commit, including the commit message, parents, file changes, and a diff viewer comparing the 'v2_to_v3_adapter.py' file between two versions. The diff highlights changes in lines 12-20, with some lines removed and new code added.

```
Commit: 0584e7e3d0afdaea532d22bda7d0e3f10c9844a [0584e7e]
Parents: 4e2ec73ee3
? Filename Path
v2_to_v3_adapter.py sfa/managers
12.12 class V2ToV3Adapter:
13.13
14.14
15. -     def __init__(self, config):
15. +     def __init__(self, api):
15. +         config = api.config
16.17             flavour = config.SFA_GENERIC_FLAVOUR
17.18             # to be cleaned
18.19             if flavour == "nitos":
19.20                 from sfa.nitns.nitnsdriver import NitnsDriver
```

ssh basics

ssh history and purpose

- successor of (very unsecure) *rsh* (remote shell)
- ssh = secure shell
- basic purpose : remote terminal
- advanced purpose : all sorts of secure tunnels / bouncing
- esp. e.g. rsync (to keep files in sync), git, ...

ssh authentication mechanism(s)

- **password** authentication is **evil**, don't ever enable
- use **public key** authentication only
- ssh-keygen : create a key pair
- keep private key (id_rsa), well... private
 - never out of your computer
 - watch out access rights
 - private keys are password-protected
- expose public key (id_rsa.pub) to your peers

ssh public key authentication - basics

- how to enable access
 - add public key in `~/.ssh/authorized_keys`
 - again watch out for access rights
- how it works
 - $E_{\text{Public}} \circ E_{\text{private}} = E_{\text{private}} \circ E_{\text{public}} = \text{Identity}$
 - challenge remote to “be” X (to have private key X)
 - send $E_{\text{public}}(\text{message})$ over the wire
 - E_{private} is required to decode message