COMP90051

# Workshop Week 12

# About the Workshops

- 7 sessions in total
  - Tue 12:00-13:00 AH211
  - Tue 12:00-13:00 AH108 *
  - Tue 13:00-14:00 AH210
  - Tue 16:15-17:15 AH109
  - Tue 17:15-18:15 AH236 *
  - Tue 18:15-19:15 AH236 *
  - Fri 14:15-15:15 AH211

# About the Workshops

- Homepage
  - https://trevorcohn.github.io/comp90051-2017/workshops

- Solutions will be released on next Friday (a week later).

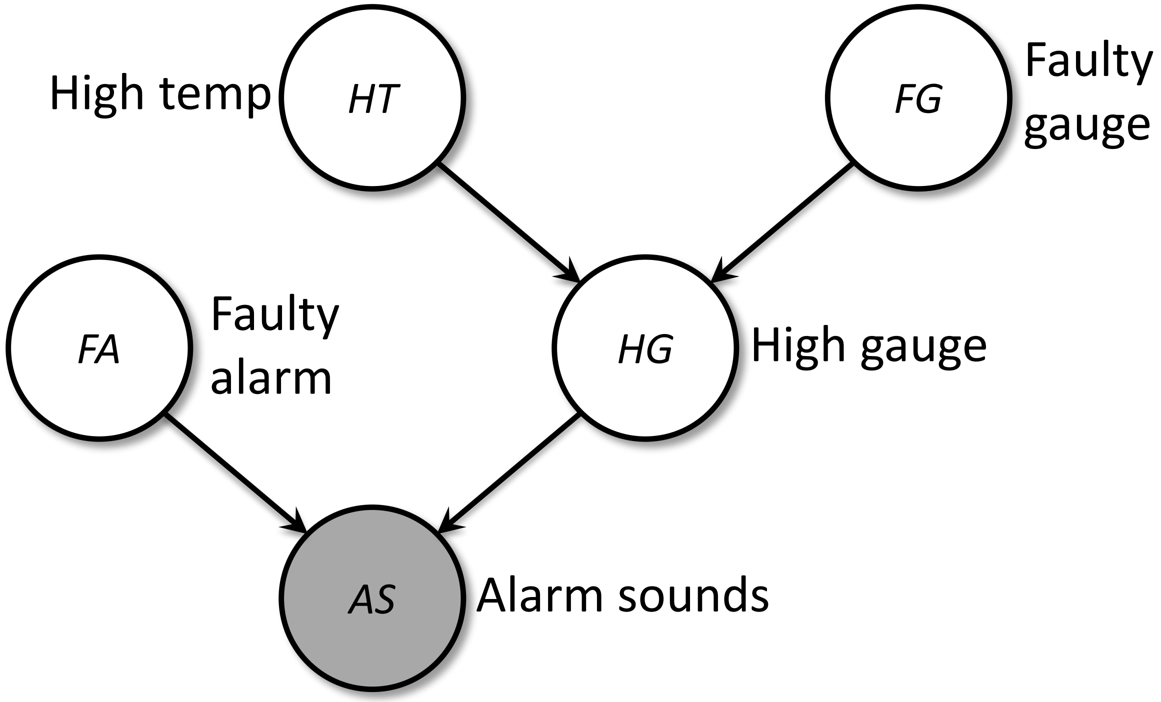
# Syllabus

| | | | |
|---|---|---|---|
| 1 | Introduction;<br>Probability theory | Probabilistic models;<br>Parameter fitting | |
| 2 | Linear regression;<br>Introduction to regularization | Logistic regression;<br>Basis expansion | |
| 3 | Optimization; Regularization | Perceptron | |
| 4 | Backpropagation | CNNs; Auto-encoders | |
| 5 | Hard-margin SVMs | Soft-margin SVMs | |
| 6 | Kernel methods | Ensemble Learning | |
| 7 | Clustering | EM algorithm | |
| 8 | Principal component analysis;<br>Multidimensional Scaling | Manifold Learning;<br>Spectral clustering | |
| 9 | Bayesian inference (uncertainty, updating) | Bayesian inference (conjugate priors) | |
| 10 | PGMs, fundamentals | PGMs, independence | |
| 11 | PGMs, inference | PGMs, EM algorithm | ← |
| 12 | PGMs, HMMs & message passing | Subject review | |

# Outline

- Review the lecture, background knowledge, etc.
  - Elimination algorithm
  - Sampling method
  - EM algorithm

# Nuclear power plant




- **Alarm sounds**; meltdown?!
- $\Pr(HT|AS=t) = \frac{\Pr(HT, AS=t)}{\Pr(AS=t)}$

$$= \frac{\sum_{FG, HG, FA} \Pr(AS=t, FA, HG, FG, HT)}{\sum_{FG, HG, FA, HT'} \Pr(AS=t, FA, HR, FG, HT')}$$

- Numerator (denominator similar)

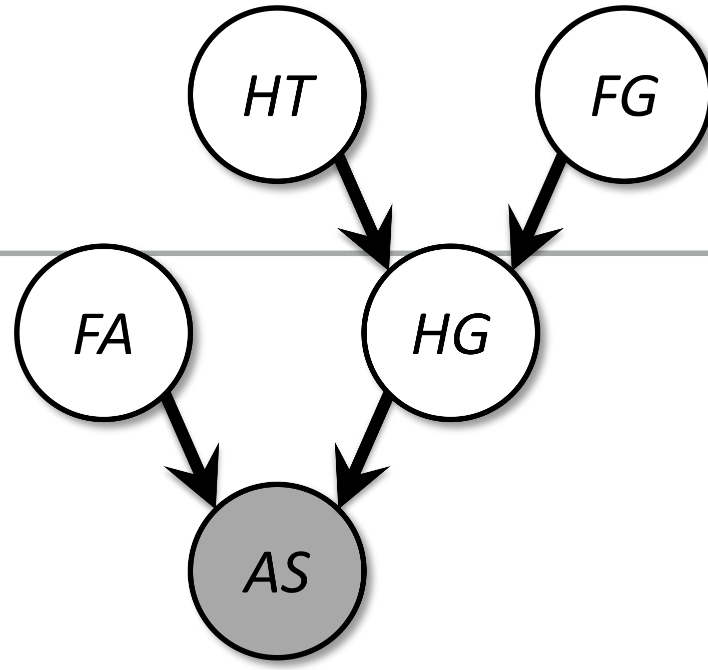
expanding out sums, joint *summing once over $2^5$ table*

$$= \sum_{FG} \sum_{HG} \sum_{FA} \Pr(HT) \Pr(HG|HT, FG) \Pr(FG) \Pr(AS=t|FA, HG) \Pr(FA)$$

distributing the sums as far down as possible *summing over several smaller tables*

$$= \Pr(HT) \sum_{FG} \Pr(FG) \sum_{HG} \Pr(HG|HT, FG) \sum_{FA} \Pr(FA) \Pr(AS=t|FA, HG)$$

# To calculate $P(HT|AS=1)$




- Joint

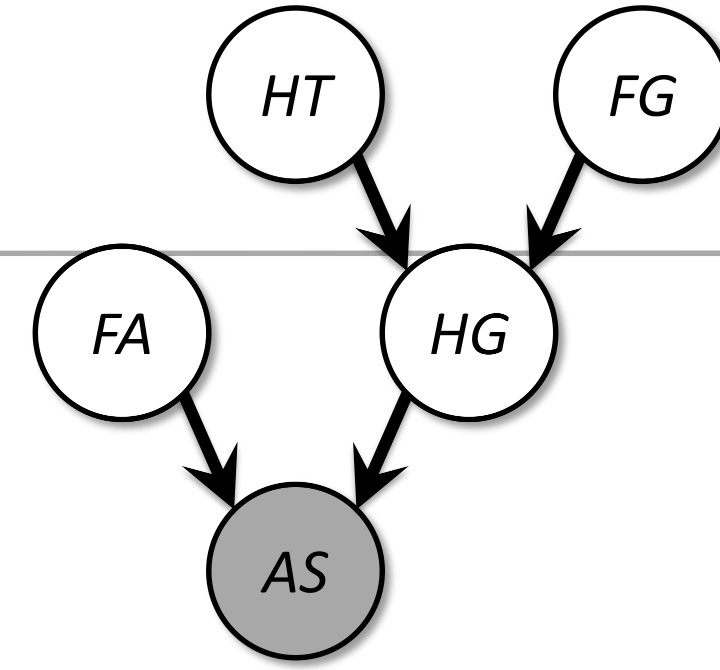
$$P(AS, FA, HG, HT, FG) = P(AS|FA, HG)P(FA)P(HG|HT, FG)P(HT)P(FG)$$

- Step 1. $P(HT|AS=1) \propto P(AS=1, HT)$
- Step 2. $P(AS=1, HT) = \sum_{FG,HG,FA} P(AS=1, FA, HG, HT, FG)$
- Step 3. Normalize $P(AS=1, HT) \rightarrow P(HT|AS=1)$

- $P(AS = 1, HT)$ has two numbers
  - $P(AS = 1, HT = 0)$ and $P(AS = 1, HT = 1)$
  - can be calculated together

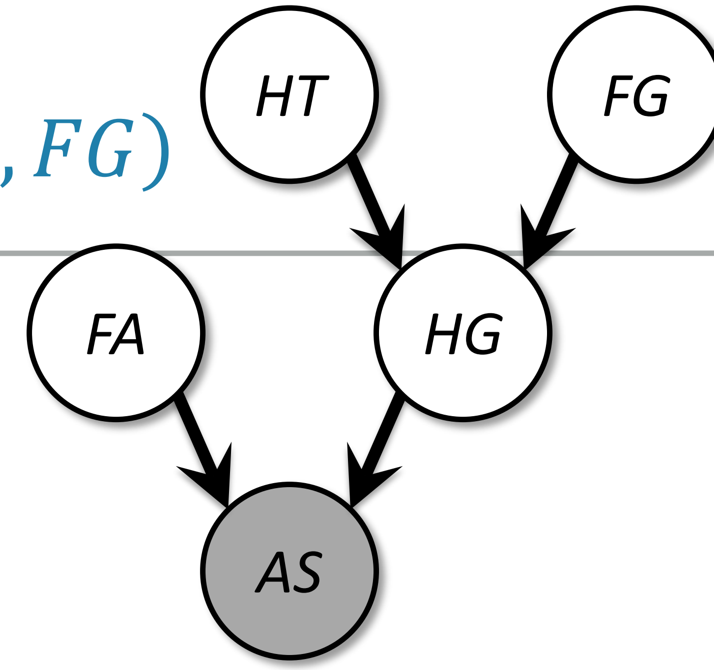
- We will first see a Naive way to calculate them

# Define some tables




```
table_FG = np.asarray([0.1, 0.9]) ← P(FG)
table_HT = np.asarray([0.2, 0.8]) ← P(HT)
table_FA = np.asarray([0.3, 0.7]) ← P(FA)

table_HG_HT_FG = np.empty((2, 2, 2))       ← P(HG|HT,FG)
table_HG_HT_FG[:, 0, 0] = [0.35, 0.65]
table_HG_HT_FG[:, 0, 1] = [0.25, 0.75]
table_HG_HT_FG[:, 1, 0] = [0.15, 0.85]
table_HG_HT_FG[:, 1, 1] = [0.05, 0.95]

table_AS_FA_HG = np.empty((2, 2, 2))       ← P(AS|FA,HG)
table_AS_FA_HG[:, 0, 0] = [0.45, 0.55]
table_AS_FA_HG[:, 0, 1] = [0.55, 0.45]
table_AS_FA_HG[:, 1, 0] = [0.65, 0.35]
table_AS_FA_HG[:, 1, 1] = [0.75, 0.25]
```

$$P(AS = 1, HT) = \sum_{FG,HG,FA} P(AS = 1, FA, HG, HT, FG)$$




```
AS = 1

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            for FA in [0, 1]:
                prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                                table_HG_HT_FG[HG, HT, FG] *
                                table_FA[FA] *
                                table_AS_FA_HG[AS, FA, HG]
                               )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

```
m_AS = table_AS_FA_HG[1, :, :]

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            for FA in [0, 1]:
                prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                                table_HG_HT_FG[HG, HT, FG] *
                                table_FA[FA] *
                                m_AS[FA, HG]
                               )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Any ideas to reduce #multiplications?

```
m_AS = table_AS_FA_HG[1, :, :]

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            for FA in [0, 1]:
                prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                                table_HG_HT_FG[HG, HT, FG] *
                                table_FA[FA] * m_AS[FA, HG]
                               )
print(prob_HT)
---------------
[ 0.0672  0.2528]
```

# Loop unrolling

```
m_AS = table_AS_FA_HG[1, :, :]

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                            table_HG_HT_FG[HG, HT, FG] *
                            table_FA[0] * m_AS[0, HG] ←
                           )
            prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                            table_HG_HT_FG[HG, HT, FG] *
                            table_FA[1] * m_AS[1, HG] ←
                           )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Rearranging the parentheses

```
m_AS = table_AS_FA_HG[1, :, :]

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                            table_HG_HT_FG[HG, HT, FG] *
                            (table_FA[0] * m_AS[0, HG] + ←
                             table_FA[1] * m_AS[1, HG]   ←
                            )
                           )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Define a message function for $FA$

```
m_AS = table_AS_FA_HG[1, :, :]

def m_FA(HG): return (table_FA[0] * m_AS[0, HG] +
                      table_FA[1] * m_AS[1, HG])

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                            table_HG_HT_FG[HG, HT, FG] *
                            m_FA(HG)
                           )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Better to precompute m_FA

```
m_AS = table_AS_FA_HG[1, :, :]

m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]


prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                            table_HG_HT_FG[HG, HT, FG] *
                            m_FA[HG]
                           )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# $FA$ is removed, then remove $HG$

```
m_AS = table_AS_FA_HG[1, :, :]
m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                            table_HG_HT_FG[HG, HT, FG] *
                            m_FA[HG]
                           )
print(prob_HT)
---------------
[ 0.0672  0.2528]
```

# Loop unrolling, rearranging the parentheses

```
m_AS = table_AS_FA_HG[1, :, :]
m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                        (table_HG_HT_FG[0, HT, FG] * m_FA[0] +
                         table_HG_HT_FG[1, HT, FG] * m_FA[1]
                        )
                       )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Define a message function for *HG*

```
m_AS = table_AS_FA_HG[1, :, :]
m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]

def m_HG(HT, FG): return (table_HG_HT_FG[0, HT, FG] * m_FA[0] +
                          table_HG_HT_FG[1, HT, FG] * m_FA[1])

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                         m_HG(HT, FG)
                        )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Again, better to precompute m_HG

```
m_AS = table_AS_FA_HG[1, :, :]
m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]

m_HG = (table_HG_HT_FG[0, :, :] * m_FA[0] +
        table_HG_HT_FG[1, :, :] * m_FA[1])

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                        m_HG[HT, FG]
                       )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Then $FG$, directly define the message func

```
m_AS = table_AS_FA_HG[1, :, :]
m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]
m_HG = (table_HG_HT_FG[0, :, :] * m_FA[0] +
        table_HG_HT_FG[1, :, :] * m_FA[1])

def m_FG(HT): return (table_FG[0] * m_HG[HT, 0] +
                      table_FG[1] * m_HG[HT, 1])

prob_HT = np.zeros(2)
for HT in [0, 1]:
    prob_HT[HT] += m_FG(HT) * table_HT[HT]

print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Finally

```
m_AS = table_AS_FA_HG[1, :, :]

m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]

m_HG = (table_HG_HT_FG[0, :, :] * m_FA[0] +
        table_HG_HT_FG[1, :, :] * m_FA[1])

m_FG = table_FG[0] * m_HG[:, 0] + table_FG[1] * m_HG[:, 1]

prob_HT = m_FG * table_HT

print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Finally (how many multiplications?)

```
m_AS = table_AS_FA_HG[1, :, :] 0

m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]
                    2                           2
m_HG = (table_HG_HT_FG[0, :, :] * m_FA[0] + 4
        table_HG_HT_FG[1, :, :] * m_FA[1])  4

m_FG = table_FG[0] * m_HG[:, 0] + table_FG[1] * m_HG[:, 1]
                    2                           2
prob_HT = m_FG * table_HT
                2
print(prob_HT)                in total 2*2 + 4*2 + 2*2 + 2 = 18
--------------
[ 0.0672  0.2528]
```

# Naive way (how many multiplications?)

```
AS = 1

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for HG in [0, 1]:
            for FA in [0, 1]:
                prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                                table_HG_HT_FG[HG, HT, FG] *
                                table_FA[FA] *
                                table_AS_FA_HG[AS, FA, HG]
                               )    4
print(prob_HT)              in total 4 * 16 = 64
---------------
[ 0.0672  0.2528]
```

# What we have done mathematically?

```
m_AS = table_AS_FA_HG[1, :, :]
```

$$m_{AS}(FA, HG) = P(AS = 1|FA, HG)$$

```
m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]
```

$$m_{FA}(HG) = \sum_{FA} P(FA) m_{AS}(FA, HG)$$

```
m_HG = (table_HG_HT_FG[0, :, :] * m_FA[0] +
        table_HG_HT_FG[1, :, :] * m_FA[1])
```

$$m_{HG}(HT, FG) = \sum_{HG} P(HG|HT, FG) m_{FA}(HG)$$

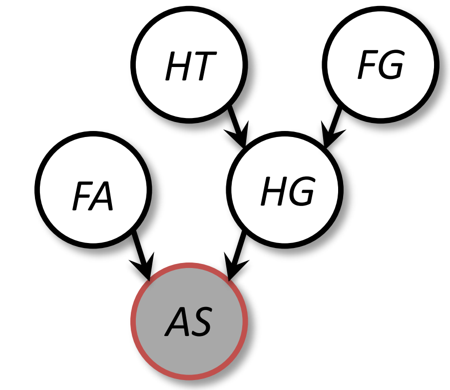
```
m_FG = table_FG[0] * m_HG[:, 0] + table_FG[1] * m_HG[:, 1]
```

$$m_{FG}(HT) = \sum_{FG} P(FG) m_{HG}(HT, FG)$$

```
prob_HT = m_FG * table_HT
```

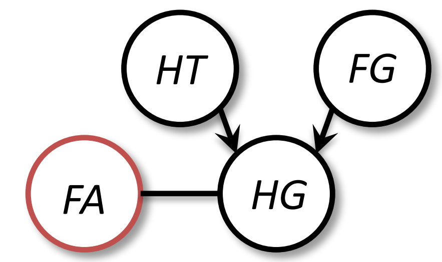
$$P(AS = 1, HT) = m_{FG}(HT) P(HT)$$

# Nuclear power plant (cont.)



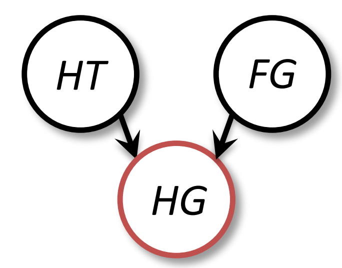

$= \Pr(HT) \sum_{FG} \Pr(FG) \sum_{HG} \Pr(HG|HT,FG) \sum_{FA} \Pr(FA) \Pr(AS = t|FA,HG)$

eliminate *AS*: since *AS* observed, really a no-op



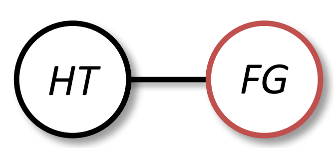

$= \Pr(HT) \sum_{FG} \Pr(FG) \sum_{HG} \Pr(HG|HT,FG) \sum_{FA} \Pr(FA)\, m_{AS}(FA,HG)$

eliminate *FA*: multiplying 1x2 by 2x2




$= \Pr(HT) \sum_{FG} \Pr(FG) \sum_{HG} \Pr(HG|HT,FG)\, m_{FA}(HG)$

eliminate *HG*: multiplying 2x2x2 by 2x1




$= \Pr(HT) \sum_{FG} \Pr(FG)\, m_{HG}(HT,FG)$

eliminate *FG*: multiplying 1x2 by 2x2


$= \Pr(HT)\, m_{FG}(HT)$

Multiplication of tables, followed by summing, is actually matrix multiplication

$m_{FA}(HG)=$

| *FA* | |
|---|---|
| f | t |
| 0.6 | 0.4 |

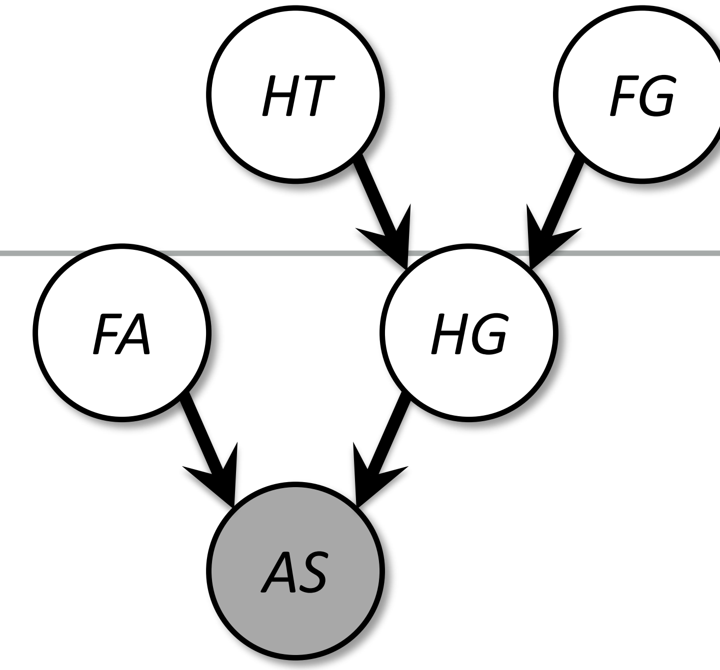
x

| | | *HG* | |
|---|---|---|---|
| | | f | t |
| *F* | f | 1.0 | 0 |
| *A* | t | 0.8 | 0.2 |

# But why the order $FA \rightarrow HG \rightarrow FG \rightarrow HT$?

```
m_AS = table_AS_FA_HG[1, :, :] 0

m_FA = table_FA[0] * m_AS[0, :] + table_FA[1] * m_AS[1, :]
                   2                           2
m_HG = (table_HG_HT_FG[0, :, :] * m_FA[0] + 4
        table_HG_HT_FG[1, :, :] * m_FA[1])  4

m_FG = table_FG[0] * m_HG[:, 0] + table_FG[1] * m_HG[:, 1]
                   2                           2
prob_HT = m_FG * table_HT
               2
print(prob_HT)               in total 2*2 + 4*2 + 2*2 + 2 = 18
--------------
[ 0.0672  0.2528]
```

# Try to eliminate *HG* after *AS*




```
m_AS = table_AS_FA_HG[1, :, :]

prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for FA in [0, 1]:
            for HG in [0, 1]:
                prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                                table_HG_HT_FG[HG, HT, FG] *
                                table_FA[FA] *
                                m_AS[FA, HG]
                               )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# Try to eliminate *HG* after *AS*

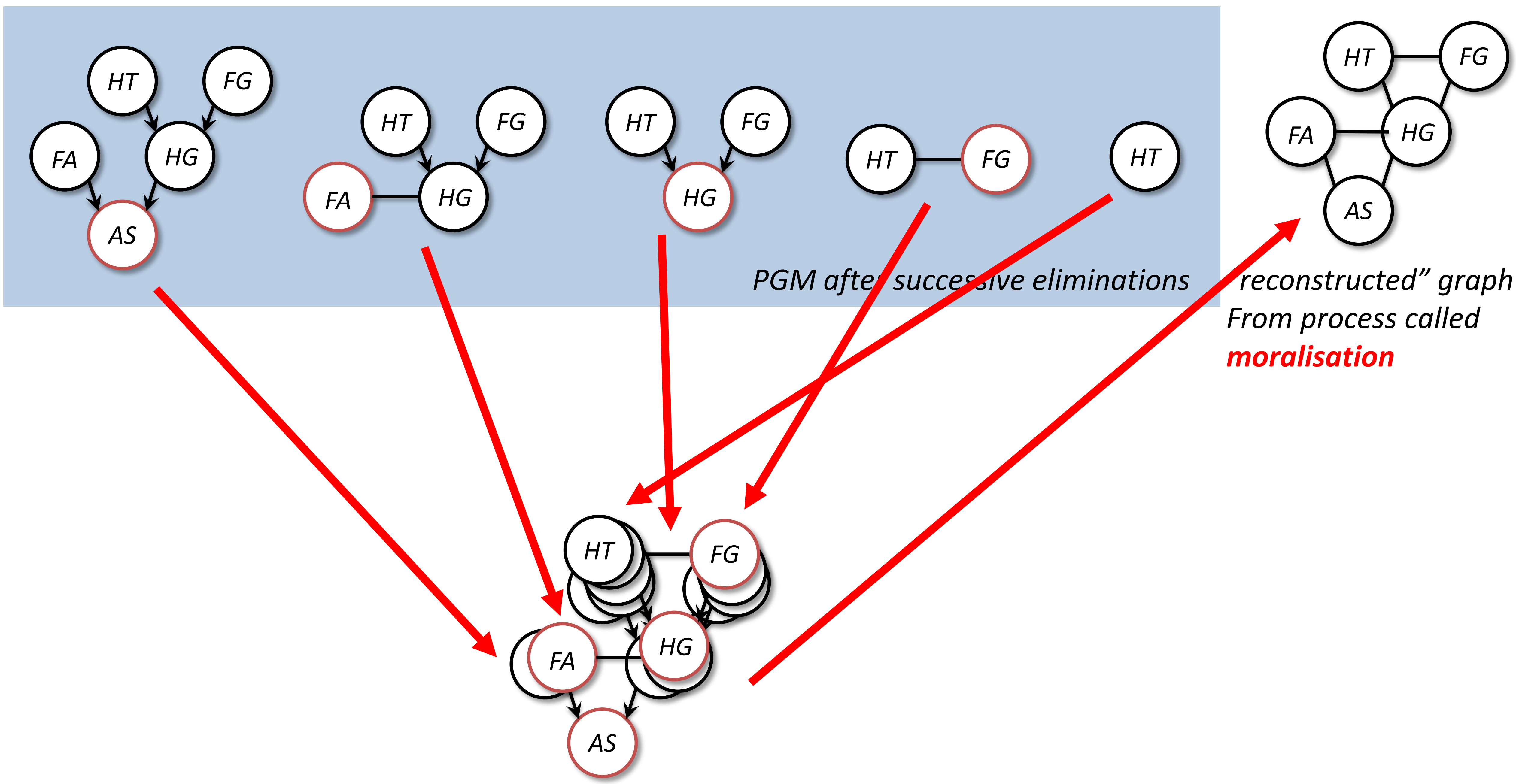
```
m_AS = table_AS_FA_HG[1, :, :]

def m_HG(FA, HT, FG):      already 8*2 = 16 multiplications
    return (m_AS[FA, 0] * table_HG_HT_FG[0, HT, FG] +
            m_AS[FA, 1] * table_HG_HT_FG[1, HT, FG])
                      because HG connected to 3 other nodes
prob_HT = np.zeros(2)
for HT in [0, 1]:
    for FG in [0, 1]:
        for FA in [0, 1]:
            prob_HT[HT] += (table_FG[FG] * table_HT[HT] *
                            table_FA[FA] *
                            m_HG(FA, HT, FG)
                           )
print(prob_HT)
--------------
[ 0.0672  0.2528]
```

# A summary for elimination algorithms

- An efficient way to marginalize random variables

- The order of elimination affects the efficiency
  - Removing a node with many children and parents results in very large clique (message matrix)
  - Time complexity exponential in the largest clique

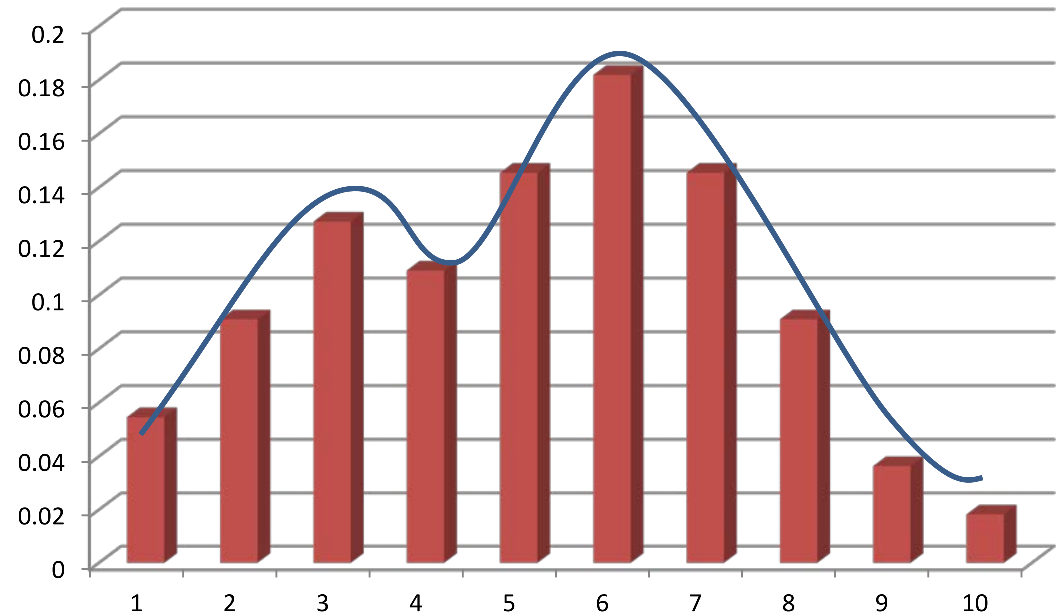
- By the way, what is the reconstructed graph?




- Put them together → the reconstructed graph

# Probabilistic inference by simulation

- Exact probabilistic inference can be expensive/impossible
- Can we approximate numerically?
- Idea: sampling methods
  - Cheaply sample from desired distribution
  - Approximate **distribution** by **histogram of samples**



# A summary for sampling methods

- If we get samples, we can
  - use them to calculate expectations (approximately)
  - approximate distributions by histogram of samples

- Useful when exact inference is expensive or impossible

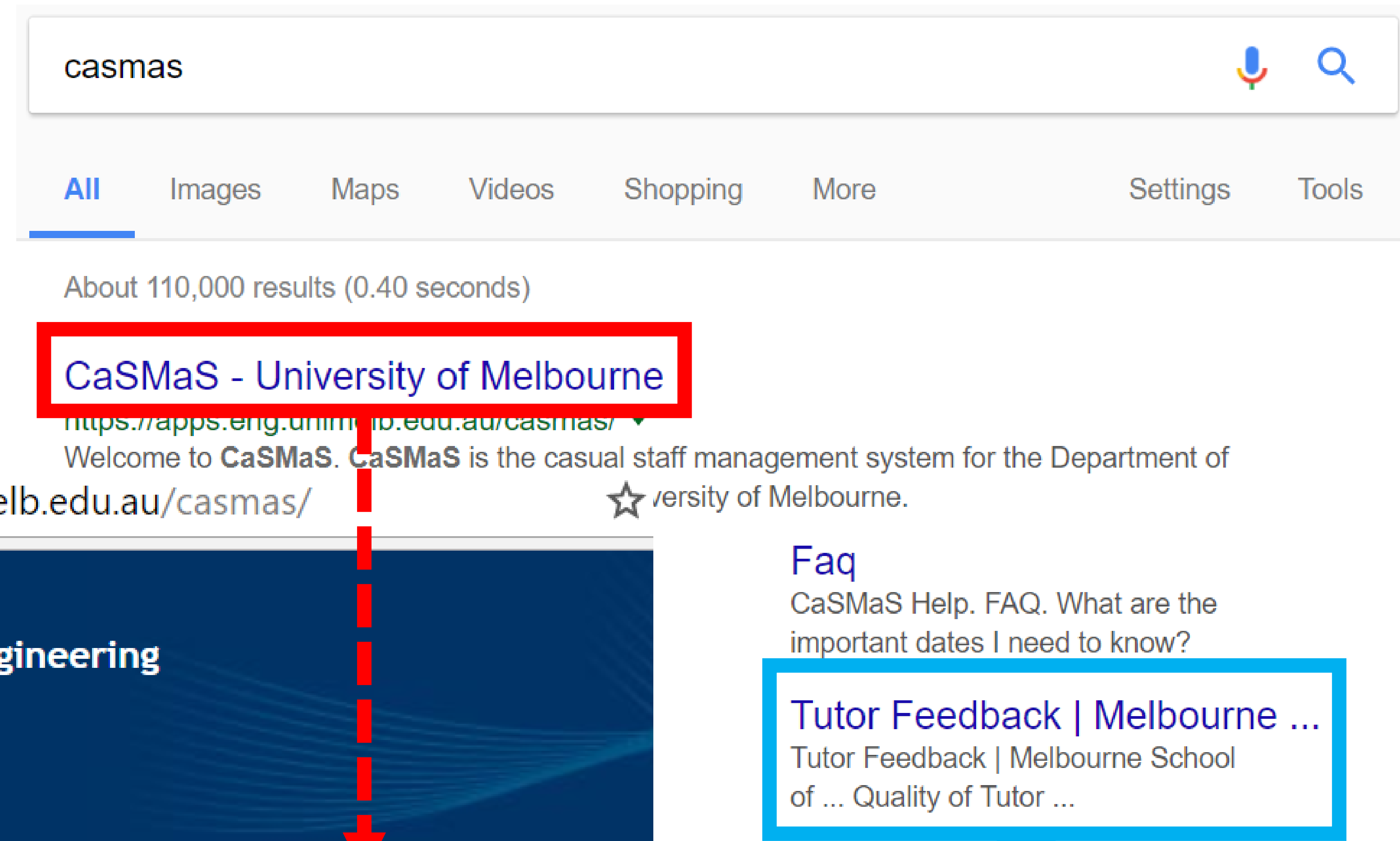
- There are many methods can sample from unnormalized distributions
  - Very useful because normalization is the main challenge for Bayesian inference

# A summary for EM algorithm

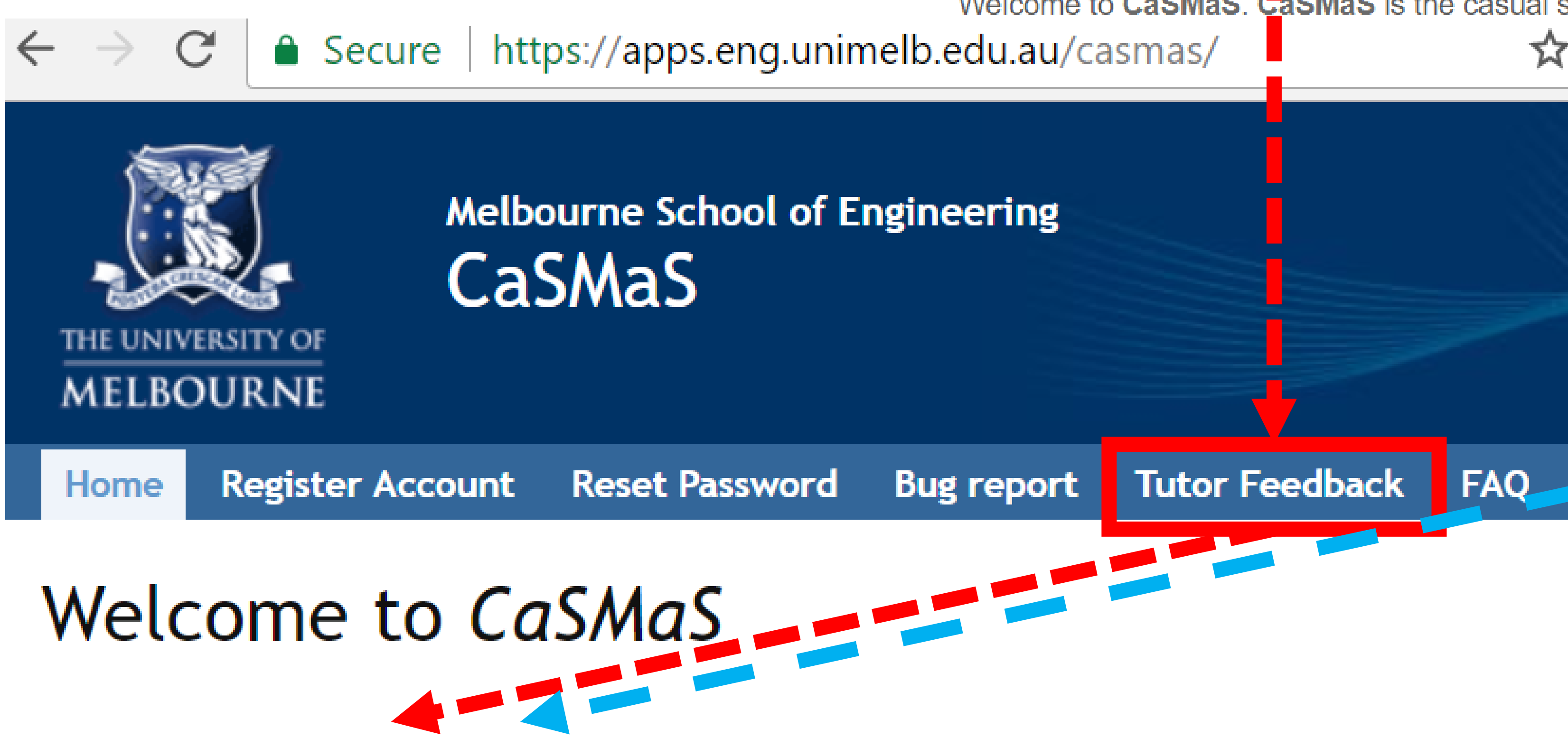
- Designed for MLE when there are latent variables
- The joint $P(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{w})$, $\boldsymbol{X}$ observed, $\boldsymbol{Z}$ unobserved, $\boldsymbol{w}$ paras
- MLE for $\boldsymbol{w}$: $\max_{\boldsymbol{w}} \log P(\boldsymbol{X}|\boldsymbol{w}) = \log \sum_{\boldsymbol{Z}} P(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{w})$
- Due to the marginalization for $Z$, $P(\boldsymbol{X}|\boldsymbol{w})$ is complicated
  - Gradients are often difficult to calculate
- EM can deal with $P(\boldsymbol{X}|\boldsymbol{w})$ by
  - E-step: estimate $P(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{w})$
  - M-step: MLE for $\boldsymbol{w}$ using $P(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{w})$, $\max_{\boldsymbol{w}} E_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{w}}[\log P(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{w})]$
    - Beneficial because $P(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{w})$ can be factorized
- Sensitive to initialization, may converge to different results

# Tutor Feedback

- Search for “casmas”




- Tutor feedback




- COMP90051 → select a class → 5:15pm or 6:16pm → …

- Good luck on your exams!