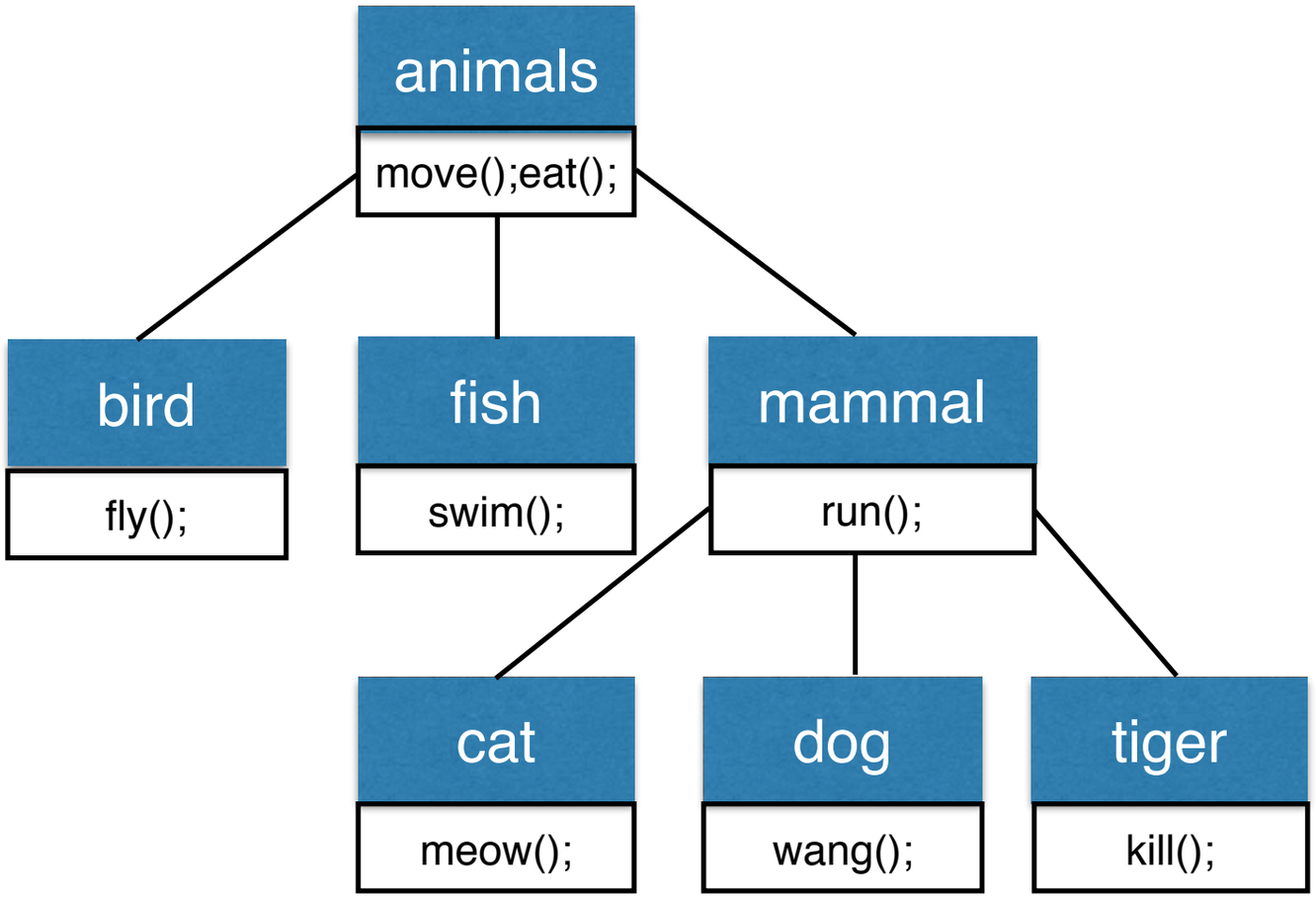
Probabilistic Programming Languages (PPL)

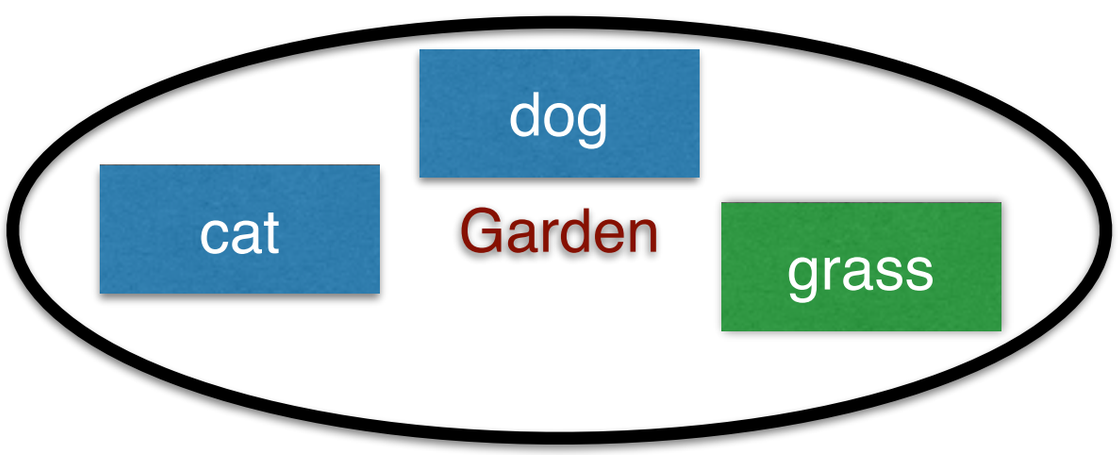
church code examples from probsmods.org

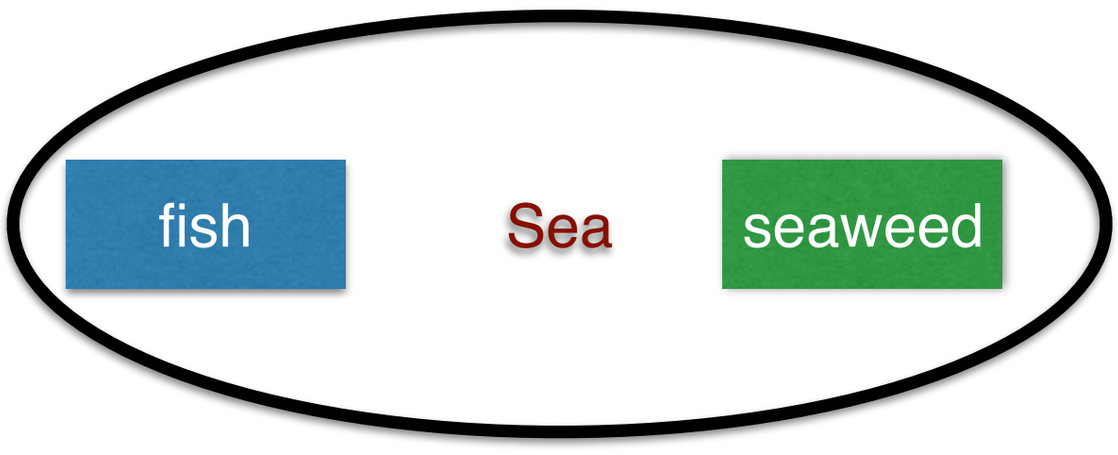
Zhirong Wu
March 11th 2015

Languages For Objects

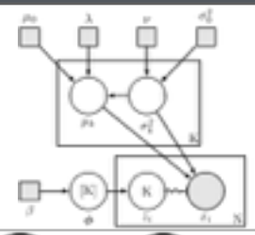
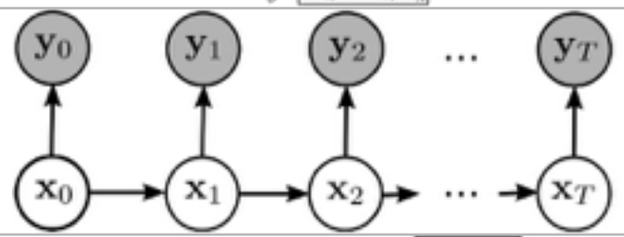
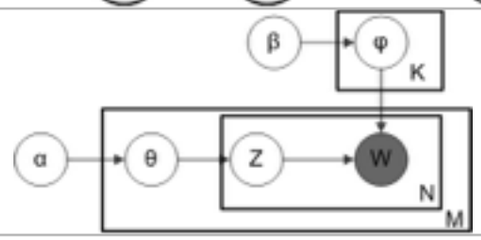
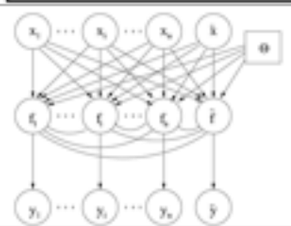
effectively describes the world through abstraction and composition. (c++, java, python)







Languages For Distributions

probablistic models	Graphical model	Inference/Learning
Mixture of Gaussian		EM
Hidden Markov Model		Baum-Welch Algorithm
Topic Model(LDA)		Variational Bayes Approximation
Gaussian Process		exact/approximate

- a lot of papers for each model.
- several implementations(with hacks) for each model.
- difficult to manipulate and do surgery over these models.

Languages For Distributions

In analogy to languages for objects, can we have a language for distributions that emphasis reusability, modularity, completeness, descriptive clarity, and generic inference?

Central Tasks:

- generative process
  compositional means for describing complex probability distributions.
- inference/learning
  generic inference engines: tools for performing efficient probabilistic inference over an arbitrary program.

Not really a programming language
A general framework for implementing probabilistic models.

Related Topics

Generative Process:

probabilistic generative models

lambda calculus

a universal language to describe any computable function.

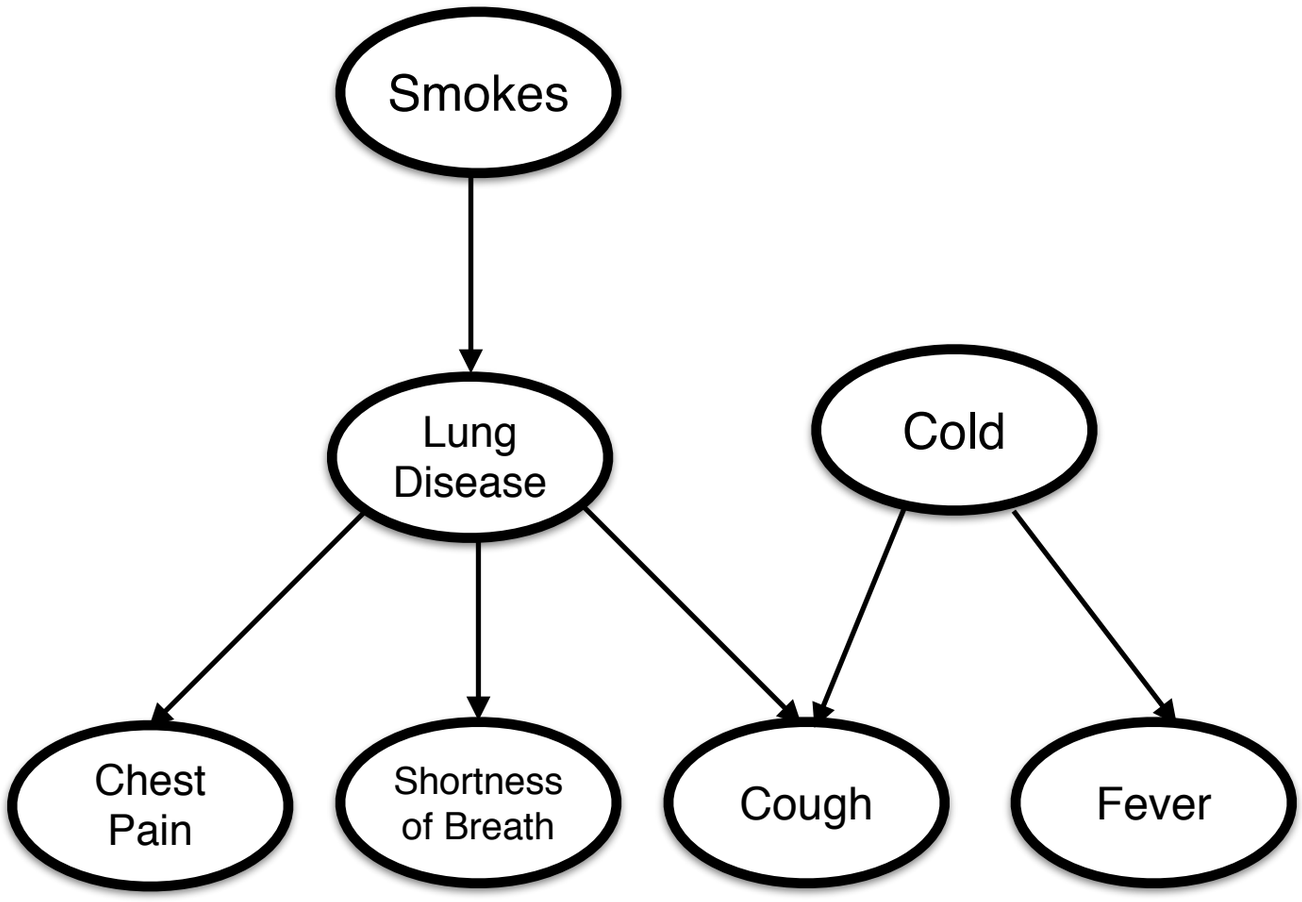
Generic Inference Algorithm:

Metropolis Hastings algorithm

not a powerful one, but generic.

Revisit Probabilistic Generative Models

a generative model describes a process that the observable data is generated. It captures the knowledge about the causal structure of the world.



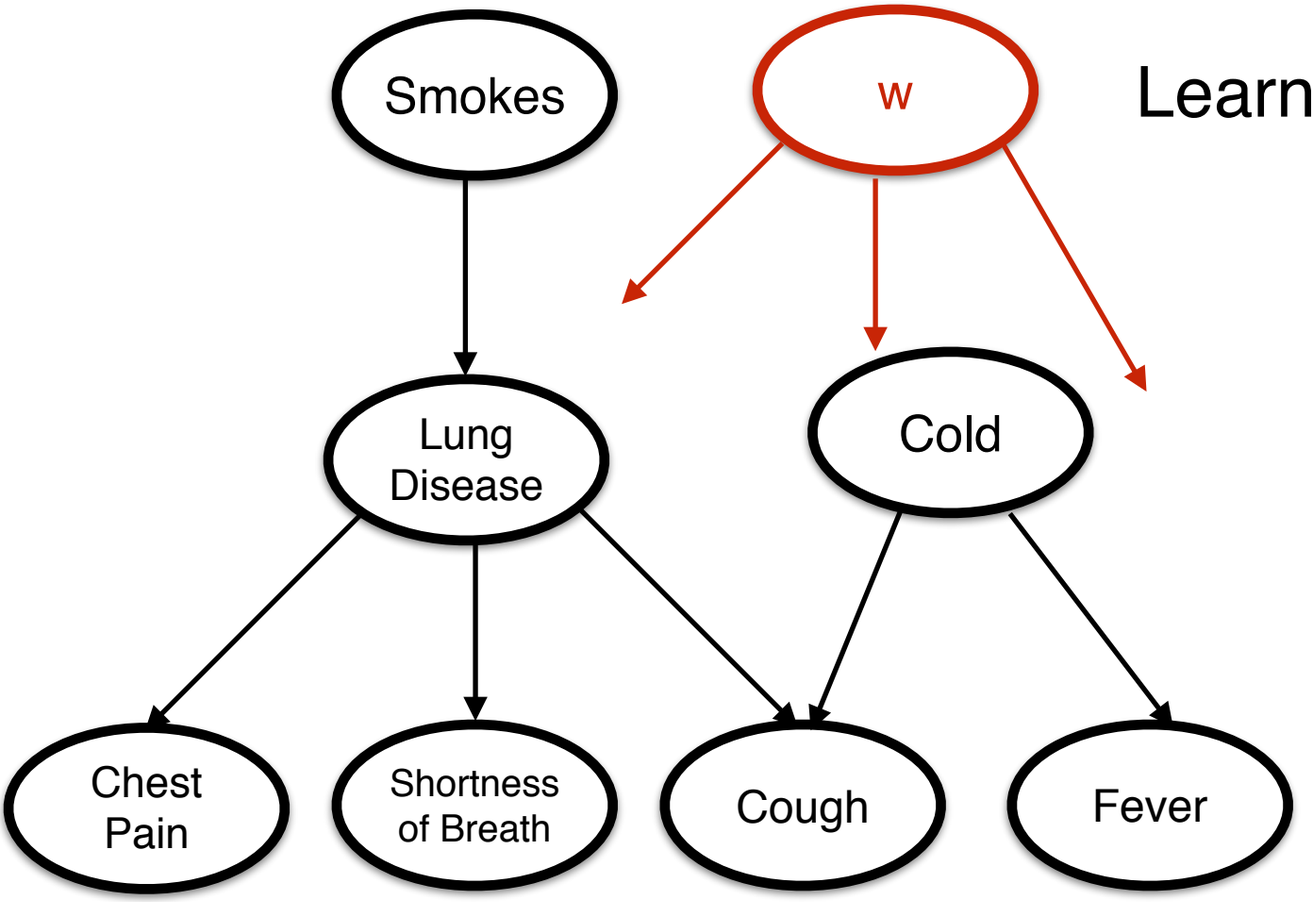
$$P(\text{Data}) = P(\text{Cough}|\text{LD}, \text{Cold}) \cdot P(\text{CP}|\text{LD}) \cdot P(\text{SOB}|\text{LD}) \cdot P(\text{F}|\text{Cold}) \cdot P(\text{LD}|\text{S}) \cdot P(\text{Cold}) \cdot P(\text{S})$$

Inference: $P(\text{S}|\text{Cough})$

credit: probmods

Revisit Probabilistic Generative Models

In bayesian machine learning, we model parameters also with uncertainties.



Learning is just a special case of inference.

learning:

$$P(w|Data) = \frac{P(\text{Data}|w)P(w)}{P(\text{Data})}$$

prediction:

$$P(x|\text{Data}) = \int P(x|\text{Data}, w)P(w|\text{Data})dw$$

$$P(\text{Data}|w) = P(\text{Cough}|\text{LD}, \text{Cold}, w) \cdot P(\text{CP}|\text{LD}, w) \cdot P(\text{SOB}|\text{LD}, w) \cdot P(\text{F}|\text{Cold}, w) \cdot P(\text{LD}|\text{S}, w) \cdot P(\text{Cold}|w) \cdot P(\text{S}|w) \cdot P(w)$$

credit: probmods

lambda calculus

Formulated by Alonzo Church (PhD advisor of Alan Turing) to formalise the concept of effective computability. It is showed that turing machines equates the lambda calculus in their expressiveness.

the key concept: **lambda terms**(expressions)

- a variable, x, is itself a valid lambda term.
- if t is a lambda term, x is a variable,
  then is $(\lambda x.t)$ a valid lambda term. (**Abstraction**)
- if t and s are lambda terms, then (ts) is a lambda term. (**Application**)
- nothing else is a lambda term.

lambda calculus

abstraction: $(\lambda x.t)$

definition of an anonymous function that is capable of taking a single input x and substitute it into expression t. (function that maps input x to output t)

$$(\lambda x.x^2 + 2) \quad \text{for function } f(x) = x^2 + 2$$

currying to handle multiple inputs:

$$f(x, y) = x^2 + y^2 \qquad (x, y) \to x^2 + y^2$$

$$\lambda x.(\lambda y.f(x, y))$$

$$\begin{aligned} f(5, 2) &= ((x \to (y \to x^2 + y^2))(5))(2) \\ &= (y \to 25 + y^2)(2) \\ &= 29 \end{aligned}$$

lambda calculus

applications: $(A)(B)$

functions operate on functions

$$(\lambda x.2x + 1)(3) = 7$$

$$(\lambda x.2x + 1)(y^2 - 1) = 2(y^2 - 1) + 1 = 2y^2 - 1$$

$$(\lambda x.x)(\lambda y.y) = \lambda x.x = \lambda y.y$$

$$\begin{aligned}(\lambda x.(\lambda y.xy))y &= (\lambda x.(\lambda t.xt))y \\ &= \lambda t.yt\end{aligned}$$

functional programming

a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids changing state and mutable data. (no assignment)

sample code of LISP:
(second oldest high-level programming language and the oldest functional programming language)

call a function:
```
(+ (* 2 3) 1)
(function arg1 arg2)
```

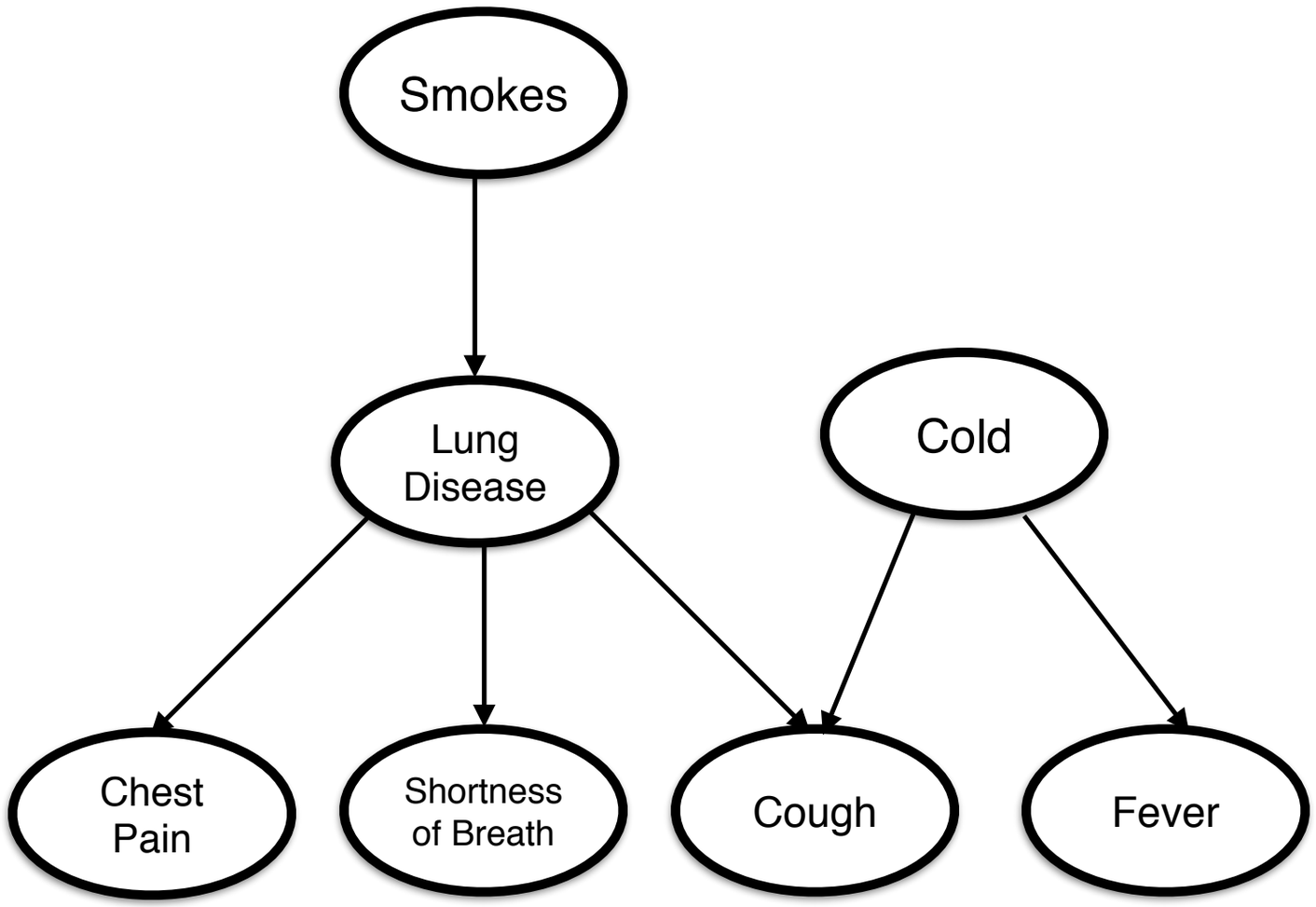
define a function:
```
(defun square (x)
  (* x x))

(defun add (x y)
  (+ x y))
```

functional programming:
```
(mapcar 'square '(1 2 3 4 5))
(remove-if 'oddp '(1 2 3 4 5))
(reduce '+ '(1 2 3 4 5))
```

anonymous function:
```
(mapcar (lambda (x) (* x 2)) '(1 2 3 4 5))
```

example 1 — Generative Process and Inference



exp 1: given the model parameters, generate data.

exp 2: infer disease given symptoms.

example 2 — Learning as Inference

learning is posterior inference:

```
(query
 (define hypothesis (prior))
 hypothesis
 (equal? observed-data (repeat N (lambda () (observe hypothesis)))))
```

exp 1: learning about fair coins

exp 2: learning a continuous parameter

exp 3: learning with priors

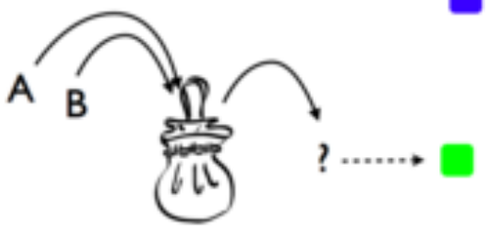
example 3 — Inference about Inference

There are 2 weighted dice.
Both of the teacher and the learner know the weights.

The teacher:
Action: pulls out a die and shows one side of the die.
Goal: successfully teach the hypothesis. Choose examples such that the learner will infer the intended hypothesis.

The learner:
Action: tries to guess which die it is given the side colour.
Goal: Infer the correct hypothesis.

```
(define (teacher die)
  (query
   (define side (side-prior))
   side
   (equal? die (learner side))))
```




```
(define (learner side)
  (query
   (define die (die-prior))
   die
   (equal? side (teacher die))))
```

exp: an agent reasons about another agent

Inference implementations

- **rejection sampling**
  - generate samples unconditionally, and decide whether to accept by checking conditions.
- **MCMC**
  - a Markov Chain make state transitions only depends on the current state and not on the sequence preceded it.
  - a Markov chain can converge to stationery distribution.
  - for any distribution, there is a Markov Chain with that stationery distribution.
  - how to get the right chain?

Let be $p(x)$ the target distribution and $\pi(x \to x')$ be the transition distribution we are interested in.

a sufficient condition is detailed balance:

$$p(x)\pi(x \to x') = p(x')\pi(x' \to x)$$

Inference implementations

Metropolis Hastings

a way to construct transition distribution and verified by detailed balance.

MH starts with a proposal distribution $q(x \to x')$

each time, we accept the new state with probability:

$$\min\left(1, \frac{p(x')q(x' \to x)}{p(x)q(x \to x')}\right)$$

the implied distribution $\pi(x \to x')$ satisfies detailed balance

```
(define (target-distr x) ...)
(define (proposal-fn x) ...)
(define (proposal-distr x1 x2) ...)

(define (accept? x1 x2)
  (flip (min 1 (/ (* (target-distr x2) (proposal-distr x2 x1))
                  (* (target-distr x1) (proposal-distr x1 x2))))))

(define (transition x)
  (let ((proposed-x (proposal-fn x)))
    (if (accept? x proposed-x) proposed-x x)))

(define (mcmc state iterations)
  (if (= iterations 0)
      '()
      (pair state (mcmc (transition state) (- iterations 1)))))
```

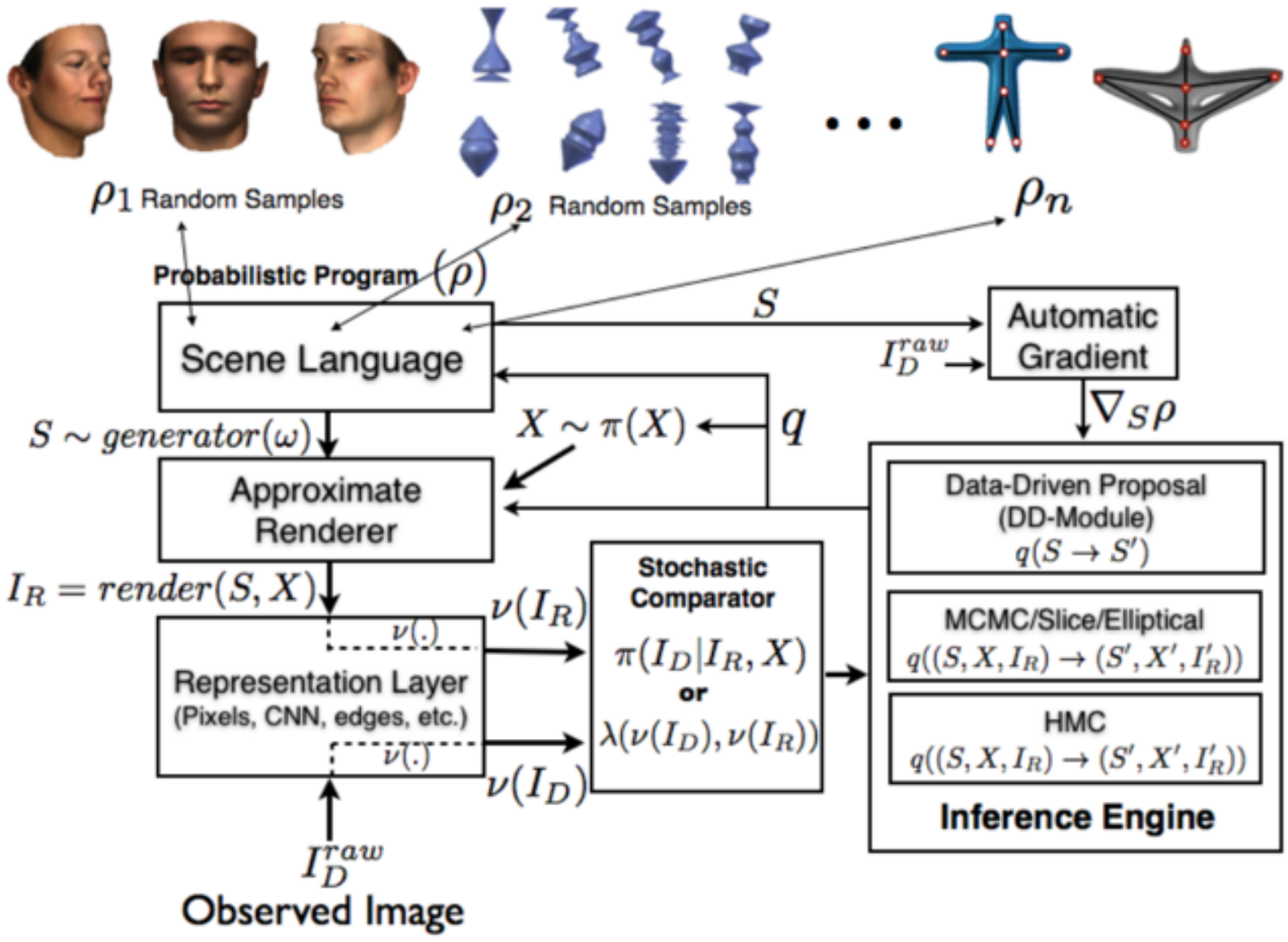
Applications

Picture: A probabilistic programming language for scene perception, CVPR2015

— 50 lines of code to get a CVPR oral paper.

vision as inverse graphics

graphics : CAD models —> images ; vision : images —> CAD models



Applications

pseudo code

```
function PROGRAM(MU, PC, EV, VERTEX_ORDER)
  # Scene Language: Stochastic Scene Gen
  face=Dict();shape = []; texture = [];
  for S in ["shape", "texture"]
    for p in ["nose", "eyes", "outline", "lips"]
      coeff = MvNormal(0,1,1,99)
      face[S][p] = MU[S][p]+PC[S][p].*(coeff.*EV[S][p]
    end
  end
  shape=face["shape"][:]; tex=face["texture"][:];

  #camera:[rotation, translation]
  camera = [Normal(0,5,1,3) Uniform(-1,1,1,2)]
  light = Uniform(-1,1,1,2)

  # Approximate Renderer
  openGLInit()
  rendered_img=openGLRender(shape,tex,light,camera)

  # Representation Layer
  ren_ftrs = getFeatures("CNN_FC8", rendered_img)

  # Stochastic Comparator
  #Using Pixel as Summary Statistics
  observe(MvNormal(0,0.01), rendered_img-obs_img)
  #Using CNN FC-8 as Summary Statistics
  observe(MvNormal(0,10), ren_ftrs-obs_cnn)
end
```

learning and testing:

```
global obs_img = imread("test.png")
global obs_cnn = getFeatures("CNN_FC8", img)
#Load args from file
TR = trace(PROGRAM,args=[MU,PC,EV,VERTEX_ORDER])
# Data-Driven Learning
learn_proposals(TR,100000,"CNN_FC8")
load_proposals(TR)
# Inference
infer(TR,callback,100,["ELLIPTICAL","DATA-DRIVEN"])
```

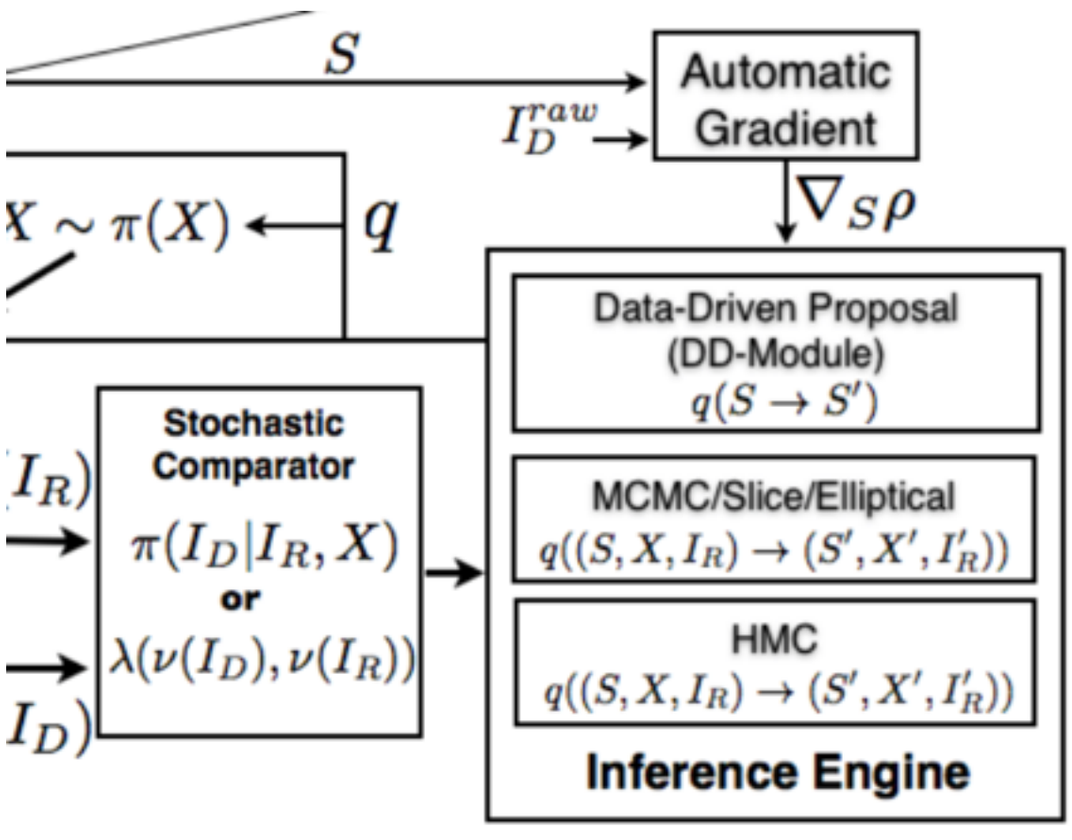
Applications

Stochastic Comparator:

likelihood: $\pi(I_D|I_R, X)$

distance: $\lambda(v(I_D), v(I_R))$

Inference Engine:

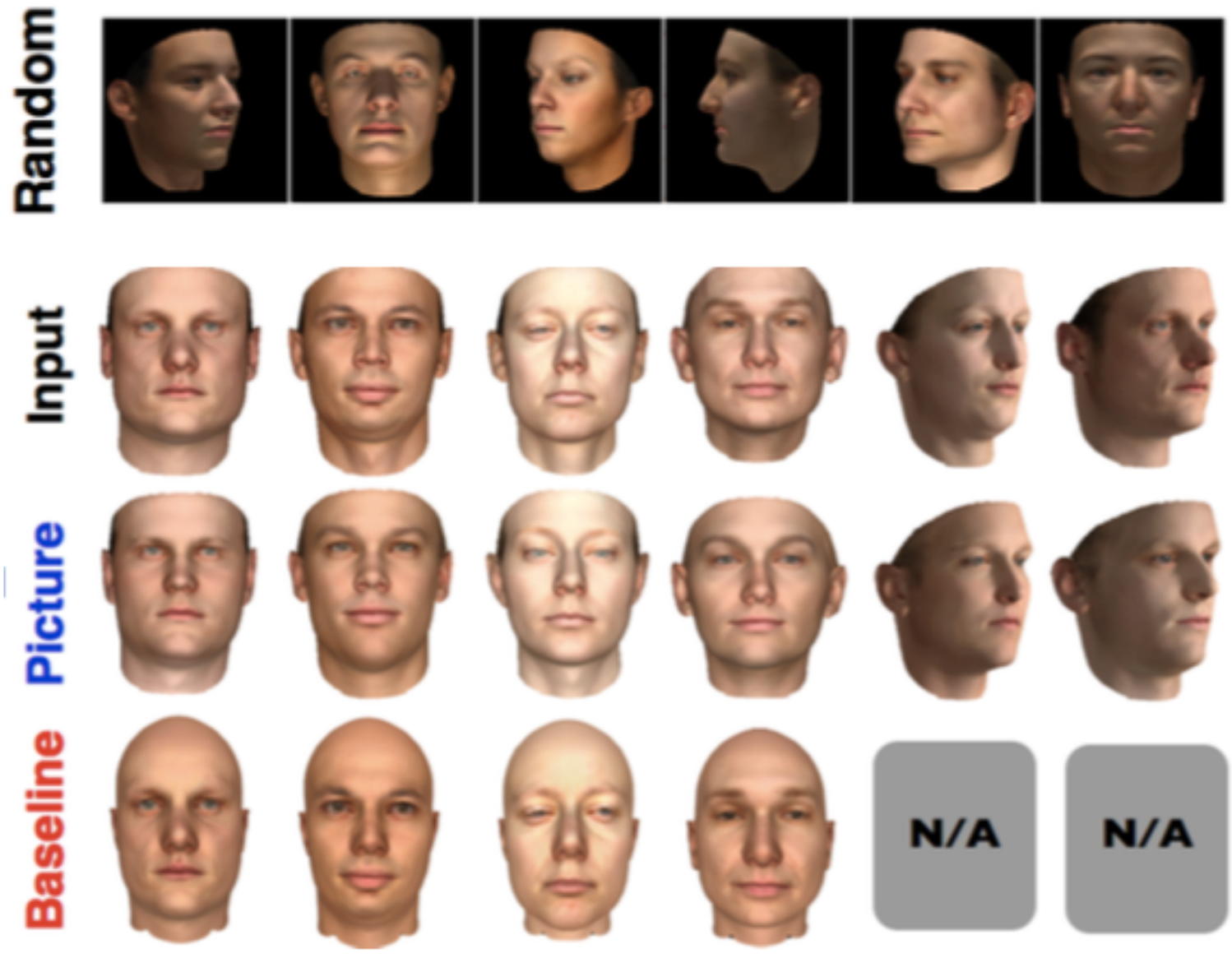


discriminative process: automatic gradient computation with LBFGS, stochastic gradient descend.

generative process: metropolis hasting with data-driven proposals, gradient proposals(Hamiltonian MC).

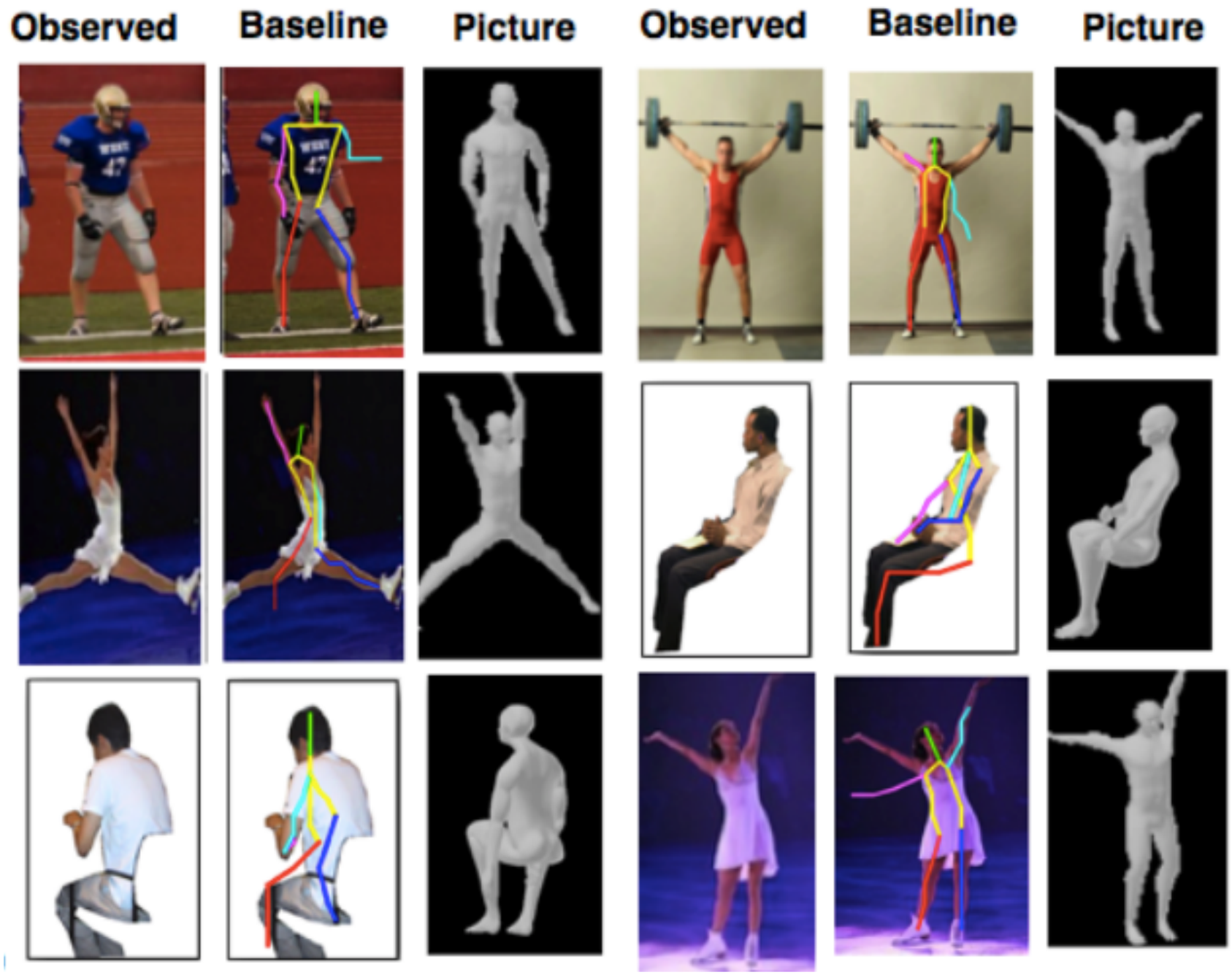
Applications

3D face reconstruction:



Applications

3D human pose estimation:



Summary

- **PPL provides an easy tool for modelling generative process.**
- **still have to design each model according to the problem.**
- **easy manipulation enables the best model design.**