Probabilistic Models of Cognition: Generative models

Lucca Hellriegel

Table of Contents

Chapter Content

Generative Model

A Generative Model describes a process that generates data, which hopefully encodes knowledge about the causal structure of the world

Example: Plinko Machine

The Plinko machine is a working model for physical processes (e.g. leafs falling from a tree)

Working Model

can be used
for simulation

captures some
structure of the world
in useful way

Plinko Machine Demo

- Simulate outcomes (data) many times, shape emerges
- Reason about ‘shape of expected outcomes’ (with probabilistic concepts)
- How to formally describe simulations/working models?

Building Generative Models

… with programming languages (WebPPL).

WebPPL allows us to describe probabilistic computation with stochastic operations.

Examples with Flip

Flip

- Sample random choice (true/false) with *flip()*
- Get visualization of (uniform) distribution with *viz(repeat(1000,flip))*

Flip Sum

- More complex process, adds 0 and 1s:
    - *var sumFlips = function() {*

      return flip() + flip() + flip()

      }
- *viz(repeat(100, sumFlips))*

Flipping Coins Bend

- *var makeCoin = function(weight) {
return function() { flip(weight) ? 'h' : 't' } };*

Flipping Coins Bend

- ```
  var bend = function(coin) {
    return function() {
      (coin() == 'h') ?
  makeCoin(0.7)() : makeCoin(0.1)()
    }
  }
  ```
  ```

Flipping Coins Bend

- *var fairCoin = makeCoin(0.5)*
- *var bentCoin = bend(fairCoin)*
- *viz(repeat(100,bentCoin))*
- Bending a fairCoin randomly in one direction

Flipping Coins Repeat Sum

- *var coin = makeCoin(0.8)*
- *var data = repeat(1000, function()*
  { sum(repeat(10, coin)) })
- *viz(data, {xLabel: '# heads'})*
- Around 80% of coin flips are true, sum is most often 8
- Distribution as expected

Causal Models in Medical Diagnosis

- *var lungCancer = flip(0.01);*
- *var cold = flip(0.2);*
- *var cough = cold || lungCancer;*
- *cough;*

Advanced Causal Models in Medical Diagnosis

- *var cough = cold || lungCancer;*
- *var cough = ((cold && flip(0.5)) ||*
  (lungCancer && flip(0.3)) ||
  (TB && flip(0.7)) ||
  (other && flip(0.01)))
- *Many illnesses and symptoms*

Probability Concepts and WebPPL

Probability

- Predict outcome value of *[flip(), flip()]*?
- A probability is a number between 0 and 1, degree of belief of specific outcomes (e.g. [true, false])
- The probability of an event A (e.g. [true,false]) is usually written as: P(A)

Probability Distribution

- A probability distribution is the probability of each possible outcome of an event
- Inspect it by sampling
- *var randomPair = function () { return [flip(), flip()]; };*
- *viz.hist(repeat(1000, randomPair), 'return values');*

Distributions in WebPPL

- The Bernoulli distribution is a coin flip with probability p for heads
- *var b = Bernoulli({p: 0.5})*
- *sample(b)*
- *viz(b)*

Distributions in WebPPL

- *var g = Gaussian({mu: 0, sigma: 1})*
- *sample(g)*
- *gaussian(0,1))*
- *var foo = function(){return gaussian(0,1)*gaussian(0,1)}*

Constructing marginal distributions: Infer

- *var foo = function(){gaussian(0,1)*gaussian(0,1)}*
- Make distribution explicit?
- *var d = Infer({method: 'forward', samples: 1000}, foo)*
- *sample(d)*
- *viz(d)*

Constructing marginal distributions: Infer

- Two views: Sampling Perspective and Distributional Perspective
- With suitable restrictions:
  - Any WebPPL program represents a distribution
  - Any distribution can be represented by WebPPL program

- With Infer: Build distribution from complicated programs
- But also: derive distributions with the “rules of probability”, for simple programs at least

The Rules of Probability

Product Rule

- *var A = flip();*
- *var B = flip();*
- *var C = [A, B];*
- 4 cases, so probability for each case is 0.25
- How to calculate this?
  - Using the product rule of probabilities

Product Rule

- *var A = flip();*
- *var B = flip();*
- *var C = [A, B];*
- Product Rule: The probability of two random choices is the product of their individual probabilities.

Product Rule

- *var A = flip();*
- *var B = flip();*
- *var C = [A, B];*
- Using the Product Rule: P(C=[true,true])=0.5*0.5=0.25
- Joint Probability: The probability of several random choices together, written as P(A,B)

Product Rule

- *var A = flip();*
- *var B = flip(A ? 0.3 : 0.7);*
- Dependent Random Choice!
- How to compute the probability?

Product Rule

- In general, the joint probability of sequential events A (first) and B (second) is:
    - $P(A,B) = P(A)*P(B|A)$
    - $P(B|A)$ is "B given A"
- Independent Choice: $P(A,B) = P(A)*P(B|A)=P(A)*P(B)$

Product Rule

- *var A = flip();*
- *var B = flip(A ? 0.3 : 0.7);*
- Dependent Random Choice!
- Calculation of P(B) needs Sum Rule!
- Examples in Exercises

Sum Rule

- *var C = flip() || flip()*
- Product Rule? Sequence?
- Cases: C == true, if [true,true] or [true,false] or [false,true]
- Calculate with Sum Rule

Sum Rule

- Sum Rule:
  - The sum of probabilities of alternative sequences of choices that lead to the same return value, is the probability of this return value
  - P(A) = \sum_{B} P(A,B)
  - Event B is sequence, event A is endresult

Sum Rule

- *var C = flip() || flip()*
- Cases: C == true, if [true,true] or [true,false] or [false,true]
- Cases is equal to sequences that lead to return value true
- $P(C) = \sum_{B} P(C,B) = 0.25 + 0.25 + 0.25 = 0.75$

Sum Rule and Product Rule

- Distribution View: The final distribution is the marginal distribution on final values

Sum Rule and Product Rule

- Distribution View: The final distribution is the marginal distribution on final values
- Sampling View: Summing up the result values of the sampled random sequences which may include joint and dependent probabilities, ignoring values in between

Sum Rule and Product Rule

- Distribution View: The final distribution is the marginal distribution on final values
- Sampling View: Summing up the result values of the sampled random sequences which may include joint and dependent probabilities, ignoring values in between

Advanced WebPPL

Stochastic recursion

- var geometric = function (p) {
  flip(p) ? 0 : 1 + geometric(p);
  };
- Adding a random number of 1s
- Stop has to be reached with 100% probability

Persistent Randomness: mem

- var eyeColor = function (person) {
  return uniformDraw(['blue', 'green', 'brown']); };
- eyeColor('bob');
- eyeColor('bob');
- Bob's eye color can change each time we ask about it!

Persistent Randomness: mem

- Solution: eye color is random, but persistent
- *mem* takes a procedure and produces a memoized version
- Memoized stochastic procedure: sample a random value the first time, then always return that same value

Persistent Randomness: mem

- *var eyeColor = mem(function (person) {*
  return uniformDraw(['blue', 'green', 'brown']);
  });

Persistent Randomness: mem

- Represent/reason about an unbounded set of properties of an unbounded set of objects.
- flipAlot maps from an integer to coin flip
- Represent nth flip of a coin, without flipping n times
- *var flipAlot = mem(function (n) {return flip() });*
- *[flipAlot(1), flipAlot(12), flipAlot(47), flipAlot(1548)]*

Example: Intuitive physics

Example: Intuitive physics

- Human Intuition: Generative Model that captures key aspects of physics
- Example: Approximate Newtonian mechanics to imagine future state of rigid bodies

Example: Intuitive physics

- First Demo of Newtonian Physics Simulator: Initial State is presented and then guessing next state
- Is the implementation of the next state(s) congruent with our intuitive model?

Example: Intuitive physics

- Second Demo: Human Intuition about the stability of block towers, first judge whether you think the tower is stable, then simulate to find out if it is

Example: Intuitive physics

- Third Demo: Hamrick et al. think our intuitions of stability are really stability given noise
    - Base Worlds (stable, almostUnstable, unstable)
    - Noisify
    - Idea: Still same intuitive assessment as base state
    - Distribution shows what the model “thinks”

Summary of Chapter Content

- Use Generative Models to describe knowledge about processes in the real world including uncertainty
- Build/simulate Generative Models with WebPPL
- WebPPL offers computation of probability concepts like the coin flip

- Main Viewpoints: Sampling and Distribution
- Important WebPPL computations:
  - Flip and other Distributions, Infer, mem
- Important probability concepts:
  - Distribution, Probability, Probability Distribution, Joint Probability, Dependent Probability
- Important rules for calculation and deriving distributions:
  - Product Rule and Sum Rule

Exercises

Exercise 1 a)

- *flip() ? flip(.7) : flip(.1)* => A ? B : C
- Product Rule (Independent!):
  - P(A,B) = 0.5 * 0.7 = 0.35
  - P(A=false, C) = 0.5*0.1 = 0.05
- Sum Rule:
  - D is end product
  - P(D) = P(A,B) + P(A=false, C) = 0.4

Exercise 1 a)

- *flip(flip() ? .7 : .1)* => $P(A|B) = 0.7$, $P(A|B{=}false) = 0.1$
- Product Rule (Not independent!)
  - $P(A,B) = P(B)*P(A|B) = 0.5 * 0.7 = 0.35$
  - $P(A,B{=}false) = P(B{=}false)*P(A|B{=}false) = 0.5 * 0.1 = 0.05$
- Sum Rule:
  - $P(A) = P(A,B) + P(A,B{=}false) = 0.4$

Exercise 1 a)

- *flip(0.4)* => P(A) = 0.4

Exercise 1 b)

- *viz(repeat(1000,function() {flip() ? flip(.7) : flip(.1)}))*
- *viz(repeat(1000,function() {flip(flip() ? .7 : .1)}))*
- *viz(repeat(1000,function() {flip(0.4)}))*

Exercise 1 c)

- *viz(repeat(1000,function() {(flip(0.6)&&flip(0.6))||flip(0.04)}))*

Exercise 1 c)

- *viz(repeat(1000,function() {(flip(0.6)&&flip(0.6))||flip(0.04)}))*
    - P(A,B) = 0.6*0.6 = 0.36

Exercise 1 c)

- *viz(repeat(1000,function() {(flip(0.6)&&flip(0.6))||flip(0.04)}))*
  - P(A,B) = 0.6*0.6 = 0.36
  - P(C) = 0.04

Exercise 1 c)

- *viz(repeat(1000,function() {(flip(0.6)&&flip(0.6))||flip(0.04)}))*
    - P(A,B) = 0.6*0.6 = 0.36
    - P(C) = 0.04
- Sum Rule (Technically would have to combine A,B,C to one event first):
    - D is the final result
    - P(D) = 0.36+0.04 = 0.4

Exercise 2

a)

Explain why (in terms of the evaluation process) these two programs give different answers (i.e. have different distributions on return values).

```
var foo = flip();
display([foo, foo, foo]);
```

Just one execution of flip

run

```
var foo = function() {return flip()};
display([foo(), foo(), foo()]);
```

run

Exercise 2 b)

- *var foo = mem(function() {return (flip())});*

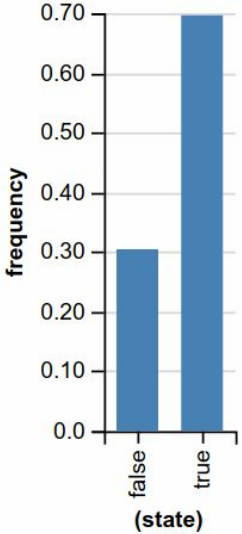
Exercise 2 c)

- *var foo = mem(function(x) {return (flip(x))});*
- *display([foo(0), foo(0), foo(1)]);*

Exercise 3

a)

Which of these programs would be more likely to generate the following proportions for 100 values of C? Justify your response.



Exercise 3 a)

- Answer: B
- P(A) = 0.9
- P(B) = P(A) * 0.9 = 0,81
- P(C) = P(B) * 0.9 = 0.729

Exercise 3 b)

- Answer: Yes, even better
- *var C = D ? A && B : A || B;*
- P(A&&B)=0.5*0.9 = 0.45
- P(A&&B,D)=0.45*0.5=0.225
- P(A||B)=
  P(A,B)+P(A=false,B)+P(A,B=false)=0.5*0.9+0.5*0.9+0.5*0.1=0.95
- P(A||B,D)=0.95*0.5=0.475
- P(C) = P(A&&B,D) + P(A||B,D) = 0.225 + 0.475 = 0.7

Exercise 4 a)

- P(allergies) = 0.3, P(cold) = 0.2
- P(sneeze) = P(allergies, cold) + P(allergies, cold=false) + P(allergies=false, cold)

  = 0.2*0.3 + 0.3*0.2 + 0.7*0.2

  = 0.06 + 0.24 + 0.14

  = 0.44
- P(sneeze, fever) = P(sneeze) - P(allergies, cold=false) = 0.44 - 0.24 = 0.2

Exercise 4 b)

- *viz.hist(Infer({method: "forward", samples: 1000}, function() {*
  var allergies = flip(0.3);
  var cold = flip(0.2);
  var sneeze = cold || allergies;
  var fever = cold;
  return [sneeze, fever];
  }))

Exercise 4 c)

- *var fever = function(person) {return cold(person)}*
- *viz.hist(Infer({method: "forward", samples: 1000}, function() { return [sneeze('bob'),fever('bob')]}))*

Exercise 4 c)

- Fix with mem
- *var allergies = mem(function(person) {return flip(.3)});*
- *var cold = mem(function(person) {return flip(.2)});*

Exercise 5

- *var makeCoin = function(weight) {   return function() {   return flip(weight) ? 'h' : 't' }}*
- Bend returns a function that samples a 0.7 or 0.1 coin based on a coin-flip:
  - *var bend = function(coin) {*
    return function() {
    return coin() == 'h' ? makeCoin(.7)() : makeCoin(.1)()
    }}

Exercise 5 a)

- *fairCoin() == 'h' ? makeCoin(.7)() : makeCoin(.1)()*
- First Product Rule, then Sum Rule:
  - P(bentCoin) = 0.5*0.7 + 0.5*0.1 = 0.4

Exercise 5 a)

- Check your answer using infer:
  - *viz.hist(Infer({method: "forward", samples: 1000}, bentCoin))*

Exercise 6 a)

- Product Rule:
  - P(geom=5)= 0.5*0.5*0.5*0.5*0.5 = 0.03125

Exercise 6 b)

- Check your answer by using Infer
    - *viz.hist(Infer({method: "forward", samples: 1000}, function() { return geometric()==5}))*

Exercise 7 a)

- *var a = flip(0.8) //0.4 + 0.4*
- *var b = flip(a ? 0.5 : 0.3) // 0.4/0.8=0.5, 0.06/0.2=0.3*

Exercise 7 b)

- *viz.hist(Infer({method: "forward", samples: 1000}, function() {*
  var a = flip(0.8) //0.4 + 0.4
  var b = flip(a ? 0.5 : 0.3) // 0.4/0.8=0.5, 0.06/0.2=0.3
  return [a,b]}))

Exercise 8 a)

- Qualitative change: Frequency of heads is higher than would be expected from just the coin
  - Reason: More likely to get two tails in the successful cases , example with 0.7 coin
  - P(2Heads)=0.7*0.7 = 0.49, P(2Tails)= 0.3*0.3 = 0.09
  - 0.49/0.58 = 0.8448
    - As seen in histogram, more probability in one coin toss leads to even higher probability in two coin tosses

Exercise 8 b)

- 0.5 would lead to equal frequency