

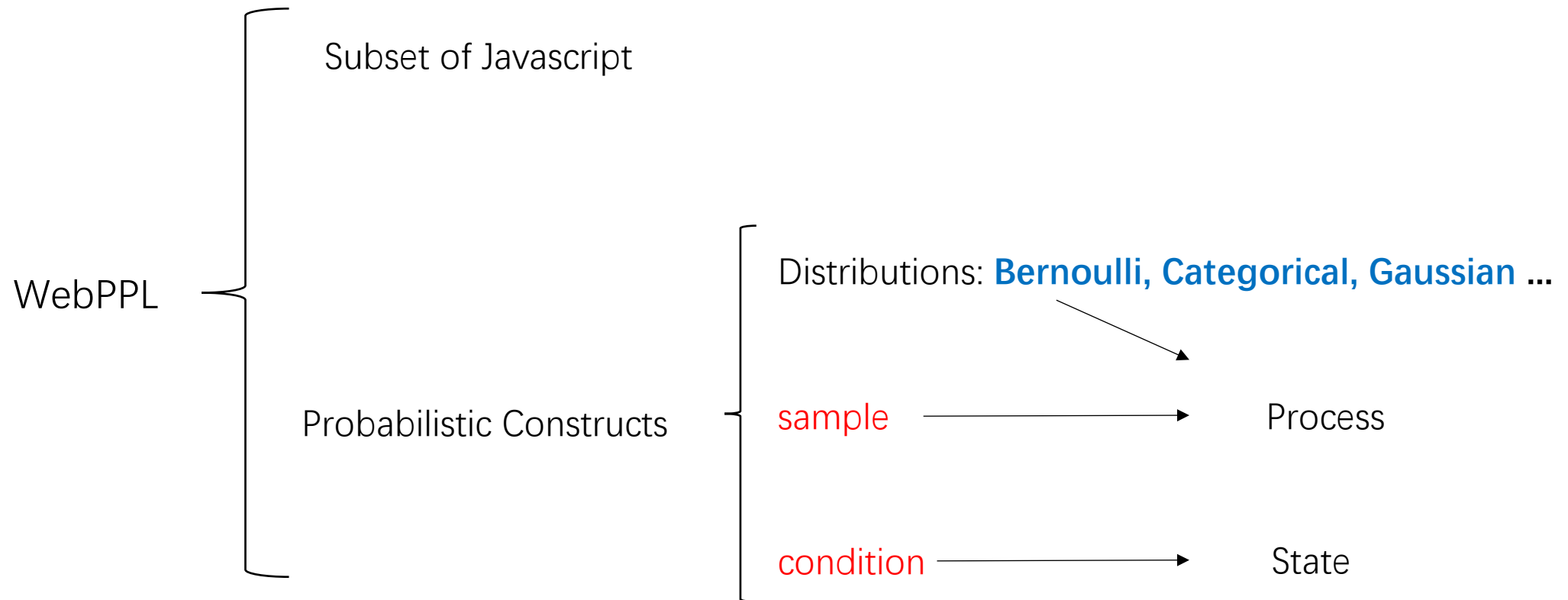
# Probabilistic Programming Using WebPPL

Xin Zhang  
Peking University

# Recap

- Probabilistic programming = Bayesian learning using a general-purpose programming language
  - Express your beliefs and uncertainties to generate data **in programs**
  - Adjust the model based on observed data using **general algorithms**
- Compared to existing Bayesian approaches
  - far more expressive
- Compared to existing programming languages
  - define distributions rather than values

# Recap



Is the following statement right?

Probabilistic programs can express applications that can not be expressed in conventional graphical models.

Is the following statement right?

A Probabilistic programming language can express programs that cannot be expressed in a conventional Turing-complete language.

Is the following statement right?

Suppose the output distribution of a program is discrete, then the probabilities of all its outputs add up to 1.

# Which distribution the function computes?

$A = \dots, B \sim \text{Binomial}(0.5, A+1)$

```
function(A) {  
    var dist = function() {  
        var B = binomial(0.5, A+1)  
        return B  
    }  
  
    return Infer(dist)  
}
```

1. B

2.  $A \mid B$

3.  $B \mid A$

4. B, A

# Which distribution the function computes?

$A \sim \text{Binomial}(0.5, 3)$ ,  $B \sim \text{Binomial}(0.5, A+1)$

```
function() {
  var A = binomial (0.5, 3) // sample(Binomial({p:0.5, n:3}))
  var dist = function() {
    var B = binomial(0.5, A+1)
    condition(A == B)
    return B
  }
  return Infer(dist)
}
```

1.  $B \mid A = B$     2.  $B \mid A$     3.  $B$     4.  $B \mid B = \text{some constant}$



# Which distribution the function computes?

$A \sim \text{Binomial}(0.5, 3)$ ,  $B \sim \text{Binomial}(0.5, A+1)$

```
function() {
  var dist = function() {
    var A = binomial (0.5, 3)
    var B = binomial(0.5, A+1)
    condition(A == B)
    return B
  }

  return Infer(dist)
}
```

1.  $B \mid A = B$     2.  $B \mid A$     3.  $A$     4.  $B \mid B = \text{some constant}$

# This Class

- Finish introduction to WebPPL
  
- Some representative applications

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- **20% of players believe the rate is not as advertised, but only 8%.**
- **To test if the assumption is true, Xiaoming pulled 20 times, and got 4 SSRs.**

What is the chance of the rate being 8%?

```

var gacha = function(){
  var cheated = sample(Bernoulli({p:0.2}));
  var pull = function(){
    if (cheated){
      return Bernoulli({p:0.08});
    }
    else
      return Bernoulli({p:0.1});
  }
  var num_pull_inst = 20;

  var performPull = function(c, num_no_ssr){
    ...
  }
  var num_ssrs = performPull(num_pull_inst, 0)
  condition(num_ssrs == 4)
  return cheated;
};

```

```

var gacha_model = Infer({model: gacha})
display(Math.exp(gacha_model.score(1)))
viz(gacha_model)

```

- 1% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- Xiaoming usually pulls 0-9 times a day

```

var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c, num_no_ssr){
    if( c == 0){
      return 0;
    }
    if (num_no_ssr == 5){
      return 1 + performPull(c-1, 0);
    }
    else{
      var cp = sample(pull);
      if (cp)
        return 1 + performPull(c-1, 0);
      else
        return 0 + performPull(c-1, num_no_ssr+1);
    }
  }
  return performPull(num_pull_inst);
};

```

- 1% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- Xiaoming usually pulls 0-9 times a day, **and because of the pity system, it is more likely he does 6 pulls.**

```

var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c, num_no_ssr){
    if( c == 0){
      return 0;
    }
    if (num_no_ssr == 5){
      return 1 + performPull(c-1, 0);
    }
    else{
      var cp = sample(pull);
      if (cp)
        return 1 + performPull(c-1, 0);
      else
        return 0 + performPull(c-1, num_no_ssr+1);
    }
  }
  return performPull(num_pull_inst);
};

```

- 1% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- Xiaoming usually pulls 0-9 times a day, **and because of the pity system, it is more likely he does 6 pulls.**

```

var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  if (num_pull_inst == 6){
    factor(1);
  }
  var performPull = function(c, num_no_ssr){
    if( c == 0){
      return 0;
    }
    if (num_no_ssr == 5){
      return 1 + performPull(c-1, 0);
    }
    else{
      var cp = sample(pull);
      if (cp)
        return 1 + performPull(c-1, 0);
      else
        return 0 + performPull(c-1, num_no_ssr+1);
    }
  }
}

```

...

- 1% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- Xiaoming usually pulls 0-9 times a day, **and because of the pity system, it is more likely he does 6 pulls.**

**factor(score)**

Add *score* to the log probability of the current execution.

```

var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(
  var num_pull = Categ
  var num_pull_inst =
  if (num_pull_inst == 0)
    factor(1);
}
var performPull = function(c, num_no_ssr){
  if( c == 0){
    return 0;
  }
  if (num_no_ssr == 5){
    return 1 + performPull(c-1, 0);
  }
  else{
    var cp = sample(pull);
    if (cp)
      return 1 + performPull(c-1, 0);
    else
      return 0 + performPull(c-1, num_no_ssr+1);
  }
}

```

Increase the probability of executions that reach here.

# More on **factor**

```
var num_pull = Categorical({vs: [0,...,9]});  
var num_pull_inst = sample(num_pull)
```



	Weight	Probability
num_pull = 0	1	1/10
num_pull = 1	1	1/10
...		
num_pull = 6	1	1/10
...		



# More on **factor**

```
var num_pull = Categorical({vs: [0,...,9]});
var num_pull_inst = sample(num_pull)
```

```
if (num_pull_inst == 6){
  factor(1);
}
```

	Weight	Probability
num_pull = 0	1	$1/(9+e)$
num_pull = 1	1	$1/(9+e)$
...		
num_pull = 6	$1*e$	$e/(9+e)$
...		

# More on Factor

- **Factor** is
  - A “soft” version of condition
  - A more fine-grained way to modify the output distribution
  - Harder to reason with than **condition**
  - Not included in every probabilistic language

- 10% chance to get a SSR card every pull.
- A pull costs 10 RMB.
- If 5 continuous pulls yield 0 SSR, the 6<sup>th</sup> pull guarantees an SSR.
- Xiaoming usually pulls 0-9 times a day.
- **It is more likely that Xiaoming pulls 2 SSR than 0 SSR, and the chance that he pulls 1 SSR is twice of him pulling 2 SSRs.**

```
var gacha = function(){
  var pull = Bernoulli({p:0.1});
  var VS = rangeArray(0,10);
  var num_pull = Categorical({vs: VS});
  var num_pull_inst = sample(num_pull);
  var performPull = function(c, num_no_ssr){
    ...
  };
}
```

```
var gacha1 = function(){
  var num_ssrs1 = gacha();
  if(num_ssrs1 == 2)
    factor(1);
  if(num_ssrs1 == 1){
    factor(Math.log(X));
  }
  return num_ssrs1;
}
```

Need some work to  
get this number

```
var num_ssrs = Infer({model: gacha1})
viz(num_ssrs)
```

# What is the relationship between these two programs?

```
var a = Gaussian({mu:0, sigma:1})

var model = function(){
  var num = categorical({vs:_.range(1,11)})
  var sim = binomial(0.5, num) - 2
  factor(-Math.abs(num+sim - sample(a)))
  return {n:num, s:sim}
}

var m = Infer(model)

viz(marginalize(m,function(x){return x.s}))
```

```
var a = Gaussian({mu:0, sigma:1})

var model = function(){
  var num = categorical({vs:_.range(1,11)})
  var sim = binomial(0.5, num) - 2
  condition(num+sim == sample(a))
  return {n:num, s:sim}
}

var m = Infer(model)

viz(marginalize(m,function(x){return x.s}))
```

# Complete Syntax of WebPPL

- We finish off WebPPL by introducing its complete syntax
- Deterministic part: subset of Javascript
  - Doesn't allow general assignments (i.e., you cannot redefine a variable)
  - Doesn't allow loops (but recursions)
  - Can invoke Javascript functions (must have no side effects)

# Deterministic Constructs in WebPPL

- <https://webppl.readthedocs.io/en/master/language.html>

# Probabilistic Constructs in WebPPL

- Distributions
  - <https://webppl.readthedocs.io/en/master/distributions.html>
- `sample(dist)`
  - Draw a sample from `dist`, which is a built-in distribution or a distribution constructed using `Infer`
- `Infer(f)`
  - `f` is a stochastic function which samples a value from a given distribution
  - It computes the marginal distribution of `f`

# Probabilistic Constructs in WebPPL

- Conditioning

- **condition**(bool): conditions the marginal distribution on a proposition (bool)
- **observe**(dist, value)

- Conceptually it is the same as

```
var x = sample(dist)
condition(x == value)
return x
```

- In many cases, especially for continuous distributions, using **observe** is more efficient than using **condition** (see a simple example using **Gaussian**)
- **factor**(score)
  - Adds score to the log probability of the current execution



# This Class

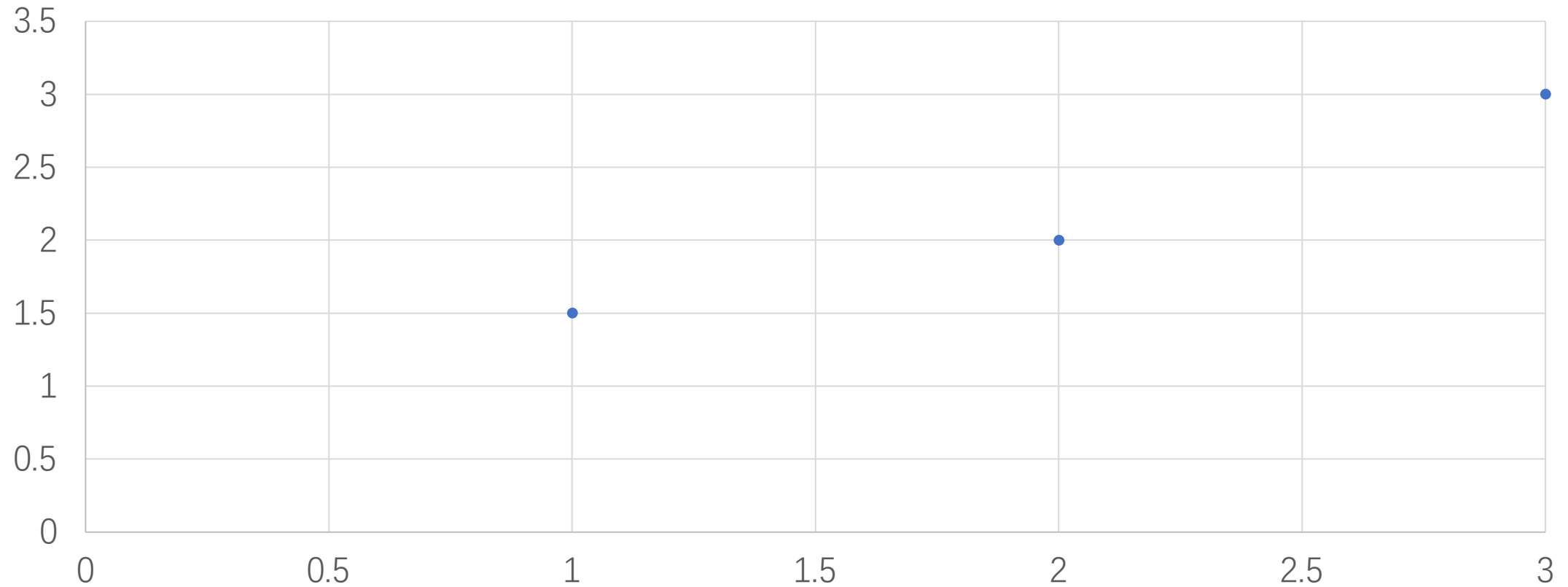
- Finish introduction to WebPPL
  
- Some representative applications

# This Class

- Finish introduction to WebPPL
  
- **Some representative applications**

# Linear Regression

$$y = a*x + b, a =? b=?$$



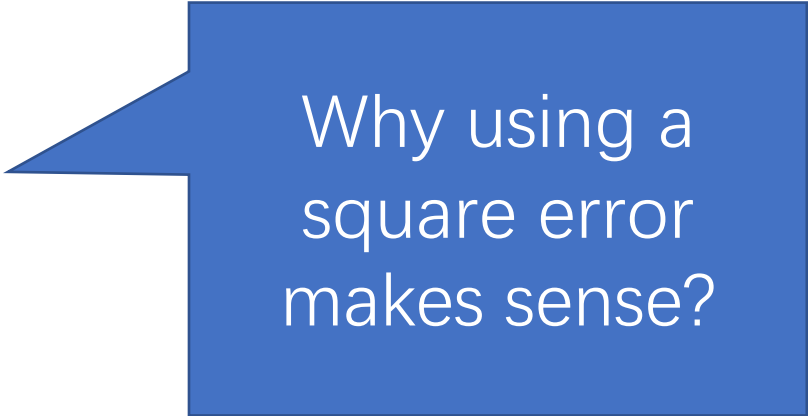
# Linear Regression: Conventional Method

- Data =  $\{(1, 1.5), (2, 2), (3, 3)\}$
- Find  $a$  and  $b$  in  $f(x) = a*x + b$ , such that

$$\sum_{(x,y) \in D} (f(x) - y)^2$$

is minimized.

- In this example,  $a = 0.75$ ,  $b = 0.67$



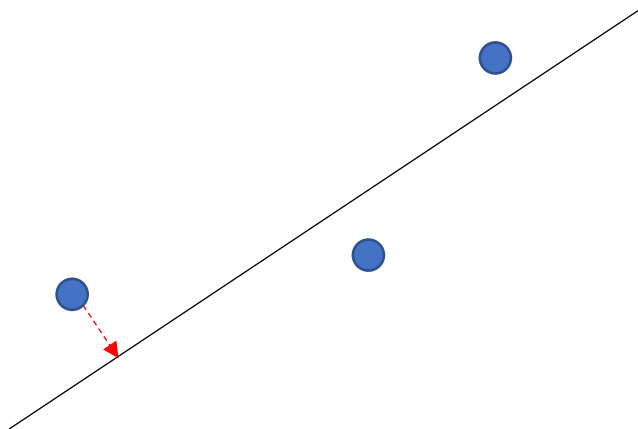
Why using a square error makes sense?

# Linear Regression: Conventional Method

- There is a probabilistic explanation!
- Assume that the function is actually written as

$$f'(x) = a^*x + b + \underline{\mathbf{N}(0, \sigma)}$$

Gaussian noise



# Linear Regression: Conventional Method

$$f'(x) = a^*x + b + N(0, \sigma)$$

- We have  $a^*x_1 + b + NV_1 = y_1$   
 $a^*x_2 + b + NV_2 = y_2$   
...  
 $a^*x_n + b + NV_n = y_n$

$NV_k$  can be any value, but we want to find assignments to  $a$  and  $b$  such that their values ( $NV_k$ ) are more likely.

# Linear Regression: Conventional Method

- In other words, we want to maximize

$$\begin{aligned}
 & L(\mathbf{N}(0, \sigma) = NV_1) * L(\mathbf{N}(0, \sigma) = NV_2) * \dots * L(\mathbf{N}(0, \sigma) = NV_n) \\
 = & L(\mathbf{N}(0, \sigma) = y_1 - a * x_1 - b) * \dots * L(\mathbf{N}(0, \sigma) = y_n - a * x_n - b) \\
 = & \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(y_1 - a * x_1 - b)^2}{2\sigma^2}} * \dots * \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(y_n - a * x_n - b)^2}{2\sigma^2}}
 \end{aligned}$$

Which is equivalent to maximizing

$$- \sum (y_k - a * x_k - b)^2$$

Or minimizing  $\sum (y_k - a * x_k - b)^2 = \sum (y_k - f(x_k))^2$

# Problems with Conventional Method

- Overfit – regularization term
- One explanation
- Confidence is unknown



# Linear Regression: Bayesian Way

- Model uncertainties in parameters explicitly
- Make parameters random variables, for example

$$y = f(x) = a * x + b$$
$$a \sim \text{uniform}(-10, 10)$$
$$b \sim \text{uniform}(-10, 10)$$

# What Can We do With the Model?

# What Can We do With the Model?

- Infer the most likely parameter conditioning on the data

$$\begin{aligned} & \operatorname{argmax}_{\omega} P(\omega|D) \\ &= \operatorname{argmax}_{\omega} P(\omega|D) \\ &= \operatorname{argmax}_{\omega} \frac{P(D|\omega)*P(\omega)}{P(D)} \\ &= \boxed{\operatorname{argmax}_{\omega} P(D|\omega)} * P(\omega) \end{aligned}$$

Maximum Likelihood Inference

# What Can We do With the Model?

- Infer the most likely parameter conditioning on the data

$$\begin{aligned} & \operatorname{argmax}_{\omega} P(\omega|D) \\ &= \operatorname{argmax}_{\omega} P(\omega|D) \\ &= \operatorname{argmax}_{\omega} \frac{P(D|\omega)*P(\omega)}{P(D)} \\ &= \operatorname{argmax}_{\omega} P(D|\omega) * P(\omega) \end{aligned}$$

Maximum a posteriori (MAP) Inference = Maximum likelihood + Prior distribution

# What Can We do With the Model?

- Infer the most likely parameter conditioning on the data

$$\begin{aligned}
 & \operatorname{argmax}_{\omega} P(\omega|D) && 1. \text{ What if } P(\omega) \text{ is a uniform distribution?} \\
 = & \operatorname{argmax}_{\omega} P(\omega|D) && 2. \text{ How about Gaussian?} \\
 = & \operatorname{argmax}_{\omega} \frac{P(D|\omega)*P(\omega)}{P(D)} \\
 = & \operatorname{argmax}_{\omega} P(D|\omega) * P(\omega)
 \end{aligned}$$

Maximum a posteriori (MAP) Inference = Maximum likelihood + Prior distribution

# MAP Inference with Uniform Prior

- Assume  $\omega \sim \text{uniform}(-\infty, +\infty)$   
$$\operatorname{argmax}_{\omega} P(D|\omega) * P(\omega)$$
$$= \operatorname{argmax}_{\omega} P(D|\omega)$$

When the prior distribution of parameter is uniform, MAP inference is the same as maximum likelihood inference. And this achieves the same effect as the conventional linear regression.

# MAP Inference with Gaussian Prior

- Assume  $\omega \sim N(0, \sigma)$

$$\begin{aligned} & \operatorname{argmax}_{\omega} P(D|\omega) * P(\omega) \\ = & \operatorname{argmax}_{\omega} P(D|\omega) * \underline{\exp(-\omega^2)} \\ & \text{L2 regularization} \end{aligned}$$

Prior distribution on the parameters helps prevent overfitting.

# MAP Inference with Other Priors

$$\operatorname{argmax}_{\omega} P(D|\omega) * P(\omega)$$

- What if  $\omega \sim \frac{1}{2b} e^{-\frac{|x|}{b}}$  (Laplace distribution)?
- Extended reading: A Probabilistic Interpretation of Regularization (<https://bjlkeng.github.io/posts/probabilistic-interpretation-of-regularization/>)



```
/* Training Data */
var data = [[1,1.5], [2,2], [3,3]];

/* Linear Regression Model. */
var lr = function() {

  /* Prior beliefs. */
  var posterior_a = uniform(-10, 10);
  var posterior_b = uniform(-10, 10);

  /* Condition on training data. */
  mapData({data: data}, function(d){
    var y_pred = posterior_a * d[0] + posterior_b;
    observe(Gaussian({mu:y_pred, sigma:1}), d[1])
  });

  return {a: posterior_a, b: posterior_b,};
};

/* Joint distribution. */
let dist = Infer({method: 'MCMC', samples: 100000, model:lr});
```

WebPPL does not have an adequate support for MAP inference. You can draw many samples and calculate the most frequent values.

In this case, we can calculate the expectations, which is actually more robust.

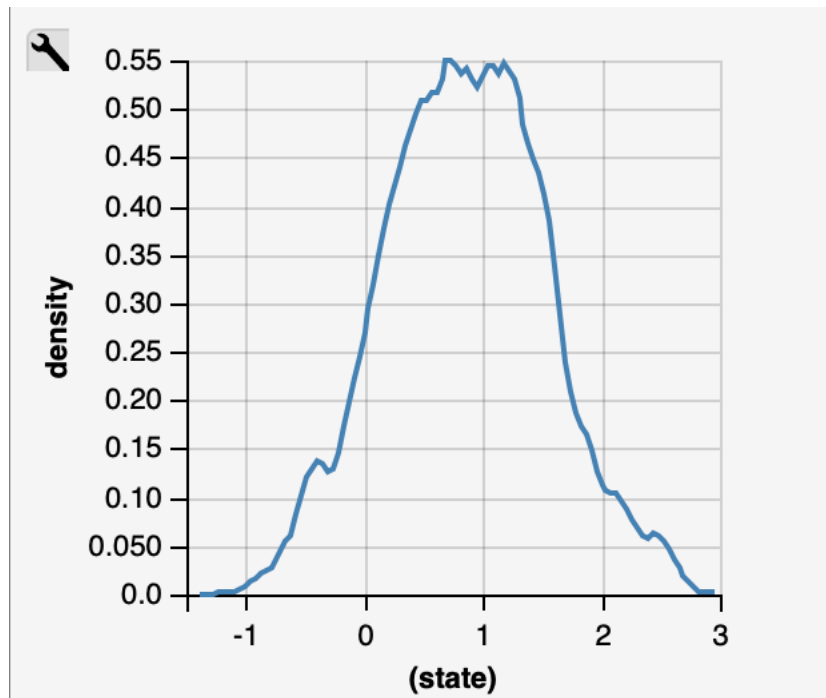
# Linking between Conventional & Bayesian Linear Regression

Conventional with Mean Square Loss  
=  
MAP(Bayesian+ uniform priors on the parameters + gaussian noise)

Conventional with Mean Square Loss + L2 Regularization  
=  
MAP(Bayesian + gaussian priors on the parameters + gaussian noise)

**The Bayesian method is at least as expressive as the conventional method!**

# There is More Bayesian Can Do!



- Seeing the the full distribution lets us know how well the model fits the data.
- Assigns a confidence to each parameter value.
- Smooth out the possible parameters using expectations

# The Full Bayesian Approach

- Instead of evaluating

$$\operatorname{argmax}_{\omega} P(D|\omega) * P(\omega)$$

- Giving a input  $x$ , we can estimate

$$P(y|D, x) = \int P(\omega|D)P(y|x, \omega)d\omega$$

**Now the prediction is a distribution!**

```
/* Training Data */
var data = [[1,1.5], [2,2], [3,3]];

var x = 1.5

/* Linear Regression Model. */
var lr = function() {

  /* Prior beliefs. */
  var posterior_a = uniform(-10, 10);
  var posterior_b = uniform(-10, 10);

  /* Condition on training data. */
  mapData({data: data}, function(d){
    observe(Gaussian({mu:y_pred, sigma:1}), d[1]) });

  var pred = posterior_a * x + posterior_b;
  return pred;
};

/* Joint distribution. */
var dist = Infer({method: 'MCMC', samples: 100000, model:lr});
```

We now have a confidence in the prediction!

# Basic Ideas of Making a Model Bayesian

- Make parameters random variables
- Add random noises to fit the model to the training data
- Make predictions by conditioning on the data

# Basic Ideas of Making a Model Bayesian

- You can make really complex models Bayesian
- A recent popular model is Bayesian Neural Networks



“I don’t know.” *instead of* “Frog!”

# More on Bayesian Neural Networks

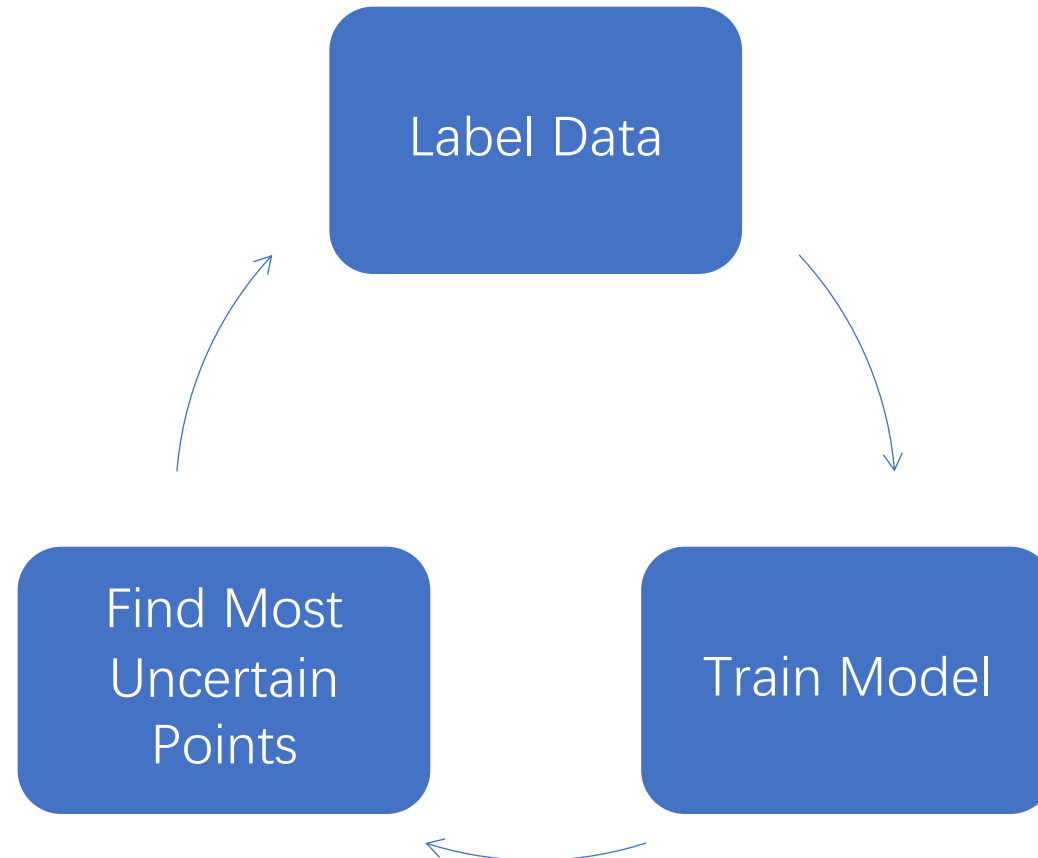
- People have thought hard on how to assign confidences to predictions
  - Previous: by looking at final layers (i.e., softmax)
  - Bayesian neural nets offer a much more principled way!
- But inferences on Bayesian Neural Network are really challenging
  - Variational Inference

Extended reading: <https://towardsdatascience.com/making-your-neural-network-say-i-dont-know-bayesian-nns-using-pyro-and-pytorch-b1c24e6ab8cd>



# More on Bayesian Models

- Exploration based on uncertainties



# Optimal Experiment Design

- Coin flipping
- There is a user. They either assume the coin has an unknown bias or the coin is unbiased
- We want to know the user's mental model by
  - Showing them the results of 4 flips (i.e., HHHH, HTHT, ...)
  - Asking them to predict the next flip
- Is HHHH or HHTT better?

*Reference: Long Ouyang, Michael Henry Tessler, Daniel Ly, Noah D. Goodman: **webppl-oed: A practical optimal experiment design system.** CogSci 2018*

# Optimal Experiment Design

- HHHH is better because when the user answers T, we know likely they believe the coin is unbiased
- We want to choose an experiment that gives us most information
  - How do we measure information gain?

# Optimal Experiment Design

- We can measure information gain by looking at how much the distribution of the hypotheses has changed

$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left( \frac{P(x)}{Q(x)} \right)$$

## Kullback–Leibler divergence

How much information is gained by going from Q to P

# Optimal Experiment Design

- Suppose the random variable of hypotheses is  $\mathbf{m}$ , the experiment is  $\mathbf{x}$ , the user's answer is  $\mathbf{y}$ , we want to find an experiment  $\mathbf{X}$ , such that

$$X \in \operatorname{argmax}_X \mathbf{E}_{p(X,Y)} (D_{KL}(m | x = X, y = Y || m))$$

- The above expectation is a random variable, we can use WebPPL to compute its distribution

```

// WebPPL cannot model distributions of functions, so
// models: the distribution of model functions
// mSample: returns an index of models
var OED = function(mSample , xSample , ySample, models){
  var mPrior = mcmcInfer(mSample)
  mcmcInfer(function(){
    var x = xSample() // Draw x
    var KLDistrib = mcmcInfer(function(){ // calculate KL | x
      var y = ySample() // Draw y
      var mPosterior = mcmcInfer(function() {
        var m = mSample()
        var am = models[m]
        observe(am(x),y) // condtion on model(x) == y
        return m
      })
      return KL(mPosterior, mPrior)
    })
    var EIG = expectation(KLDistrib)
    factor(Math.log(EIG / maxEIG)) // Optional
    return {x: x, EIG: EIG}
  })
}

```

*Reference: Ouyang et al, CogSci 2018*

# The Models

```
var fairCoin = function(seq) {
  Infer(function(){ flip(0.5) })
}
```

Given the observation of seq, what is the prediction of the next flip?

```
var biasCoin = function(seq) {
  Infer(function(){
    var w = uniform(0,1)
    var flipCoin = function(){ return flip(w) }
    var sampledSeq = repeat(seq.length , flipCoin)
    condition(arrayEquals(seq,sampledSeq))
    return flipCoin ()
  })
}
```

# Prior and Calculating OED

```
var models = [fairCoin, biasCoin]

var mSample = function(){
  return categorical({vs:_.range(0,models.length)})
}

var numFlips = 4

var xSample = function(){
  return repeat(numFlips, flip)
}

var ySample = flip

var oed = OED(mSample,xSample,ySample, models)
```



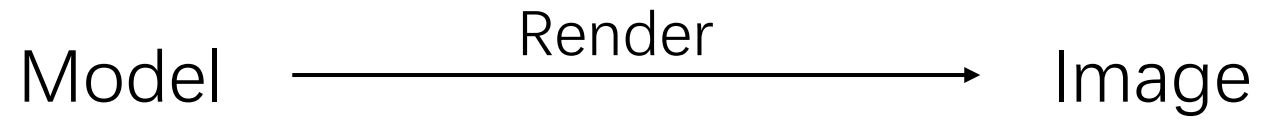
# More on Optimal Experiment Design

- Very useful in designing experiments to test theories. Used in Psychology and other fields.
- You can make our simple experiments more complex by adding more models and more participants
- See more at
  - *Long Ouyang, Michael Henry Tessler, Daniel Ly, Noah D. Goodman: **webppl-oed: A practical optimal experiment design system.** CogSci 2018*
  - <https://github.com/mhtess/webppl-oed>

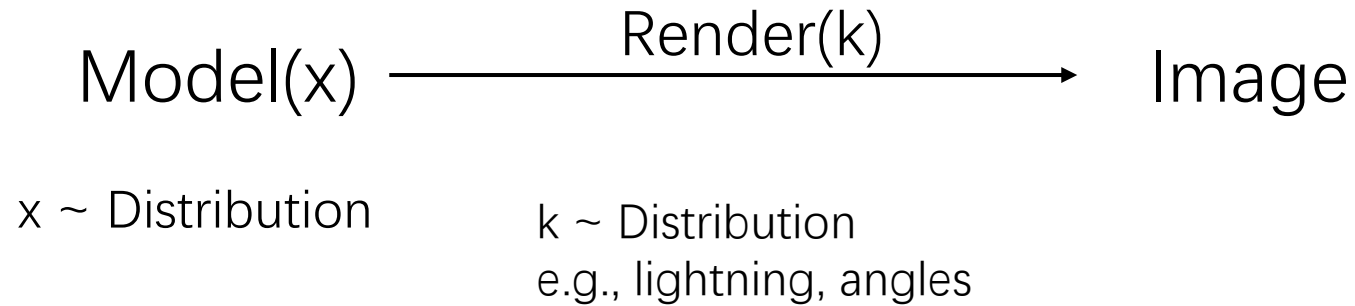
# Inverse Graphics

- More application oriented
- One of the killer applications that made probabilistic programming popular
- Applications:
  - Scene understanding
  - Generating data
  - ...

# Inverse Graphics



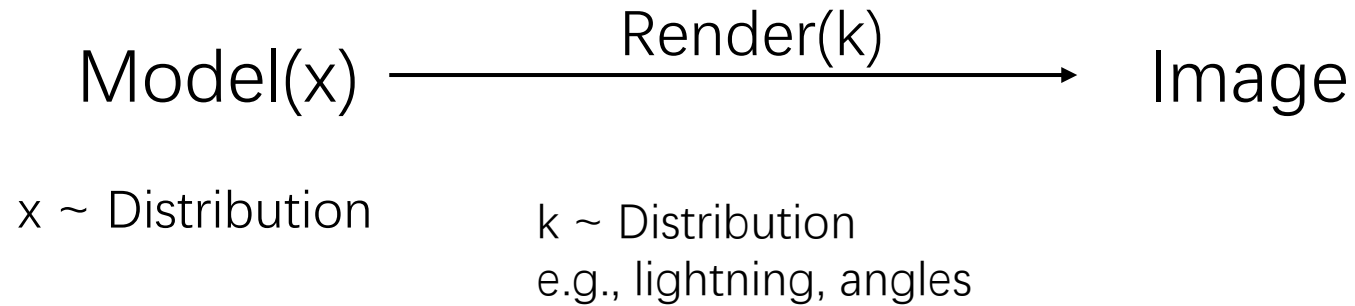
# Inverse Graphics



Given an observed image, we can

- Infer  $x$
- Generating different images by changing  $k$

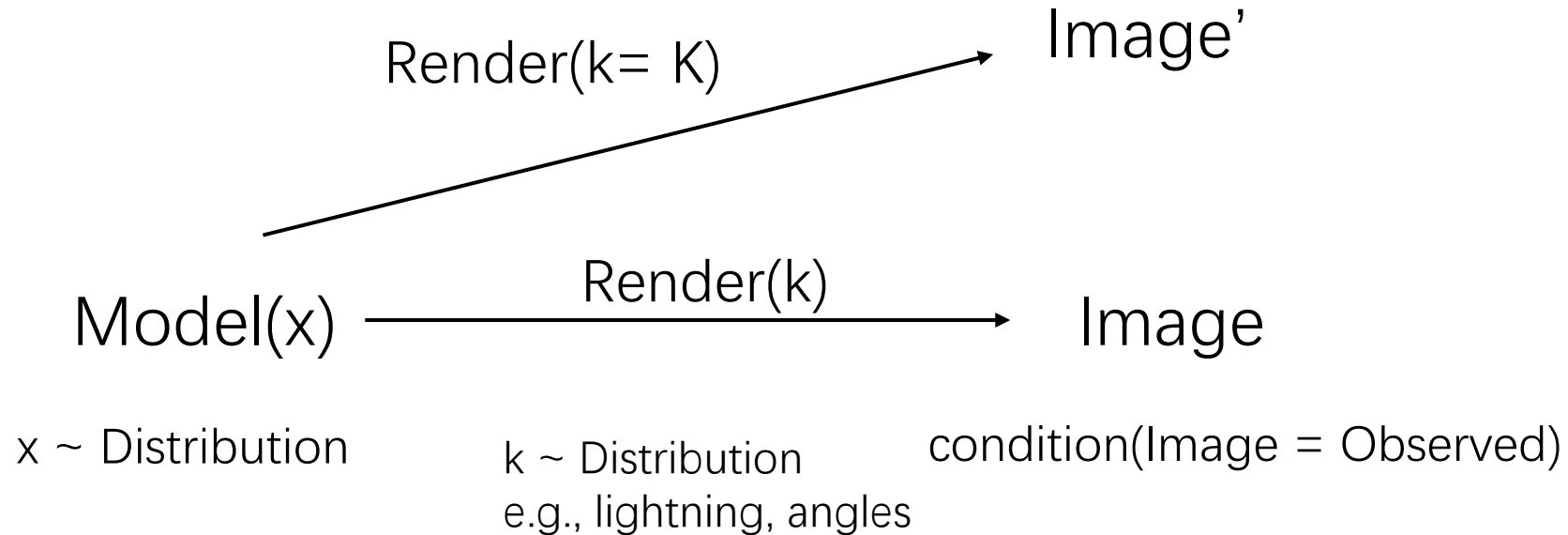
# Inverse Graphics



$P(x \mid \text{Image} = \text{Observed})$

$P(\text{Image}' \mid \text{Image} = \text{Observed}), \text{Image}' = \text{Render}(k = K, \text{Model})$

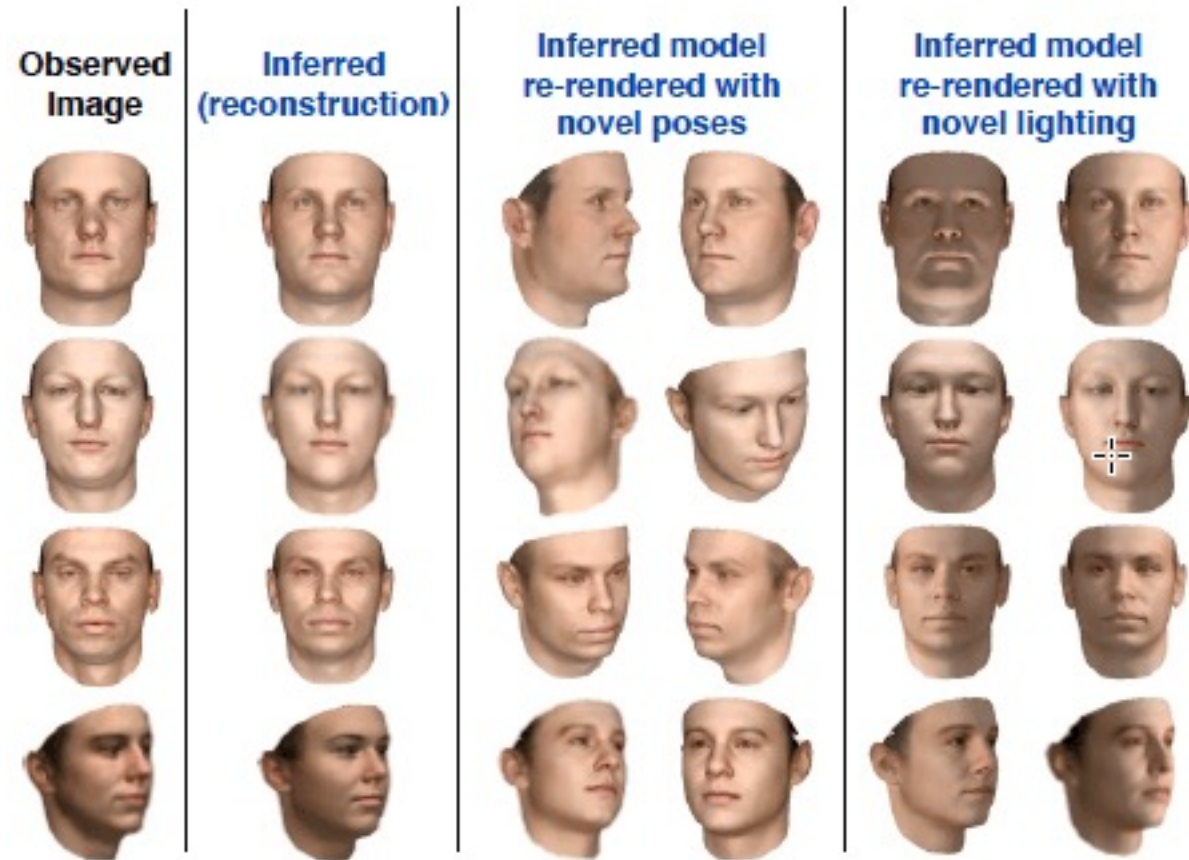
# Inverse Graphics



$P(x \mid \text{Image} = \text{Observed})$

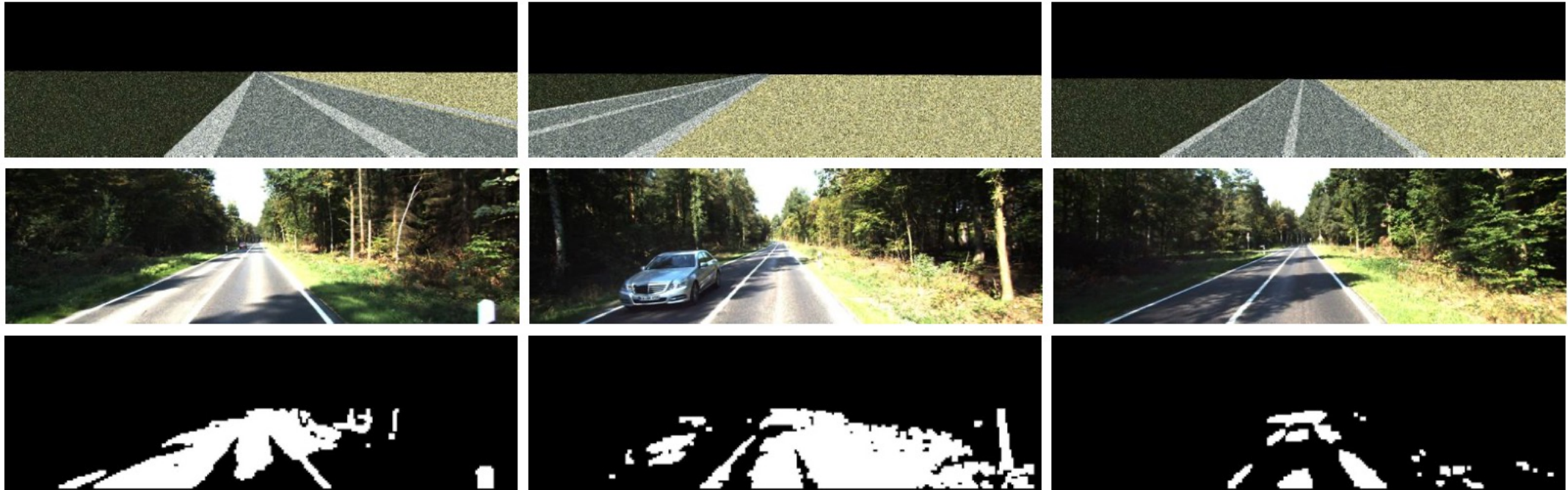
$P(\text{Image}' \mid \text{Image} = \text{Observed}), \text{Image}' = \text{Render}(k = K, \text{Model})$

# Inverse Graphics



**Inverse Graphics** – 3D faces rendered from 2D images using only 50 lines of PPL code.  
Reference: <http://news.mit.edu/2015/better-probabilistic-programming-0413>

# Inverse Graphics



Mansinghka, Vikash K., et al. "Approximate bayesian image interpretation using generative probabilistic graphics programs." *Advances in Neural Information Processing Systems 26* (2013): 1520-1528.



# Inverse Graphics-Simple Example

One of "+", "-", "|"

Render(location offset)



```
00000  
00000  
00010  
00111  
00010
```

See `inverse_graphics.js`.

# Summary

- WebPPL
  - Subset of Javascript
  - Probabilistic constructs: distributions, **sample**, **condition**, **observe**, **factor**
- Applications
  - Bayesian learning models
  - Optimal experiment design
  - Inverse graphics
  - More

# Next Lecture

- You have got an idea of how probabilistic programming looks and what they can do
- We will get into its theory, algorithms, implementation now, starting with its predecessor: probabilistic graphical models