Integrating Theory-Driven and Data-Driven Approaches to Affective Computing via Deep Probabilistic Programming

[redacted]

Desmond Ong
National University of Singapore
& A*STAR Singapore

Together with Zhi-Xuan Tan (A*STAR->MIT), Harold Soh (NUS),
Jamil Zaki (Stanford), & Noah Goodman (Stanford)

Reference: https://arxiv.org/abs/1903.06445
Tutorial website: https://desmond-ong.github.io/pplAffComp/

Tutorial Outline

- What is & Why (deep) probabilistic programming?
  - Intro to probabilistic programming concepts
  - Pyro
- Model Building vs. Model Solving
- Worked Examples (in Affective Computing)
  - Illustrate with a simple dataset, and simple "building-block" models
  - Designed to be easy to compare different theories

Tutorial Outline

- What is & Why (deep) probabilistic programming?
  - Intro to probabilistic programming concepts
  - Pyro
- Model Building vs. Model Solving

1 hr 20 mins (9am-10:20am)

20 min break (until 10:40am)

- Worked Examples (in Affective Computing)
  - Illustrate with a simple dataset, and simple "building-block" models
  - Designed to be easy to compare different theories

Google Colab: No installation required!

1 hr 20 mins (until 12pm)

Learning Objectives

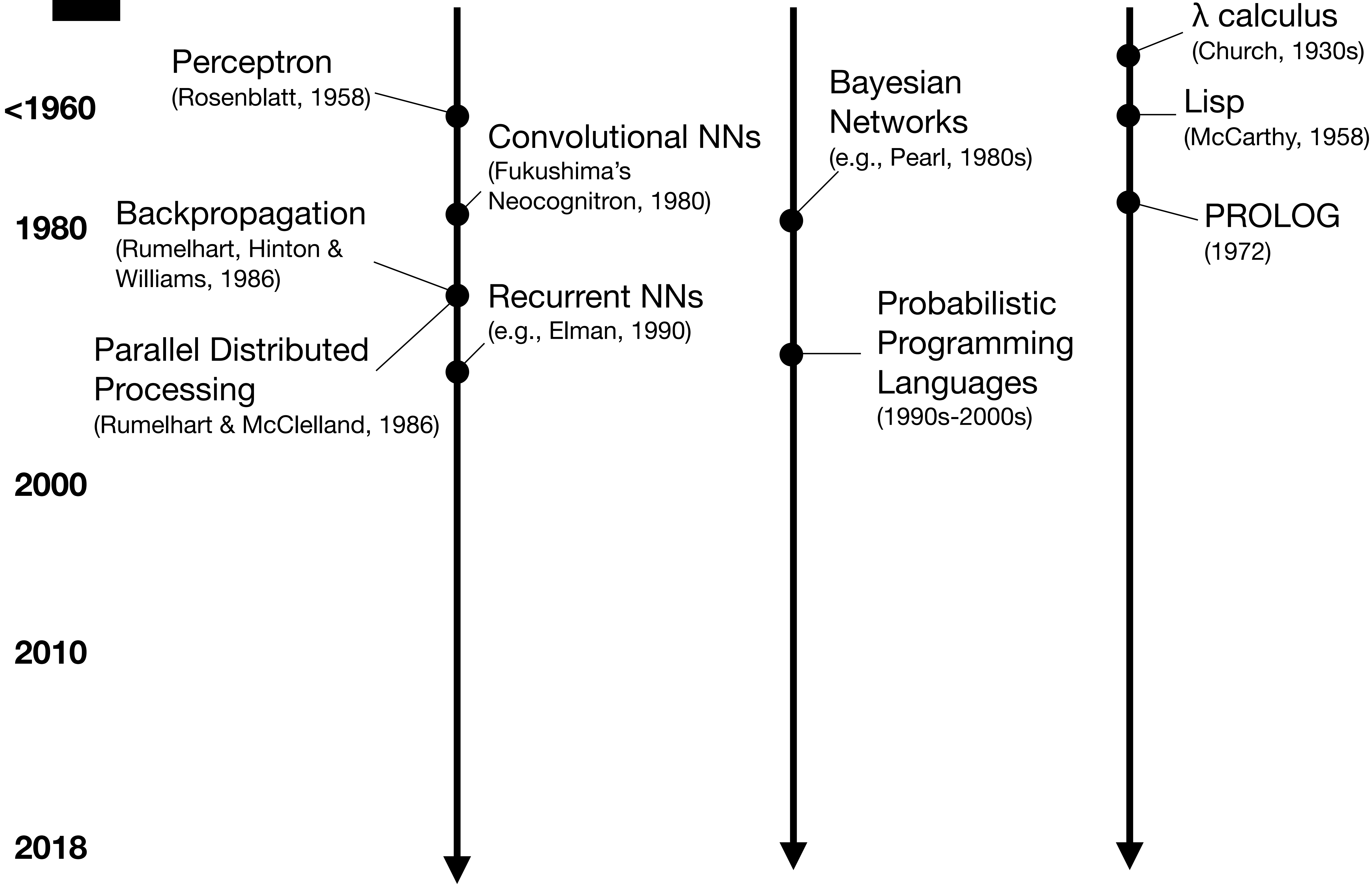
By the end of the tutorial,

Tutorial participants will be introduced to **deep probabilistic programming** as a novel paradigm for building affective computing models, via worked examples.

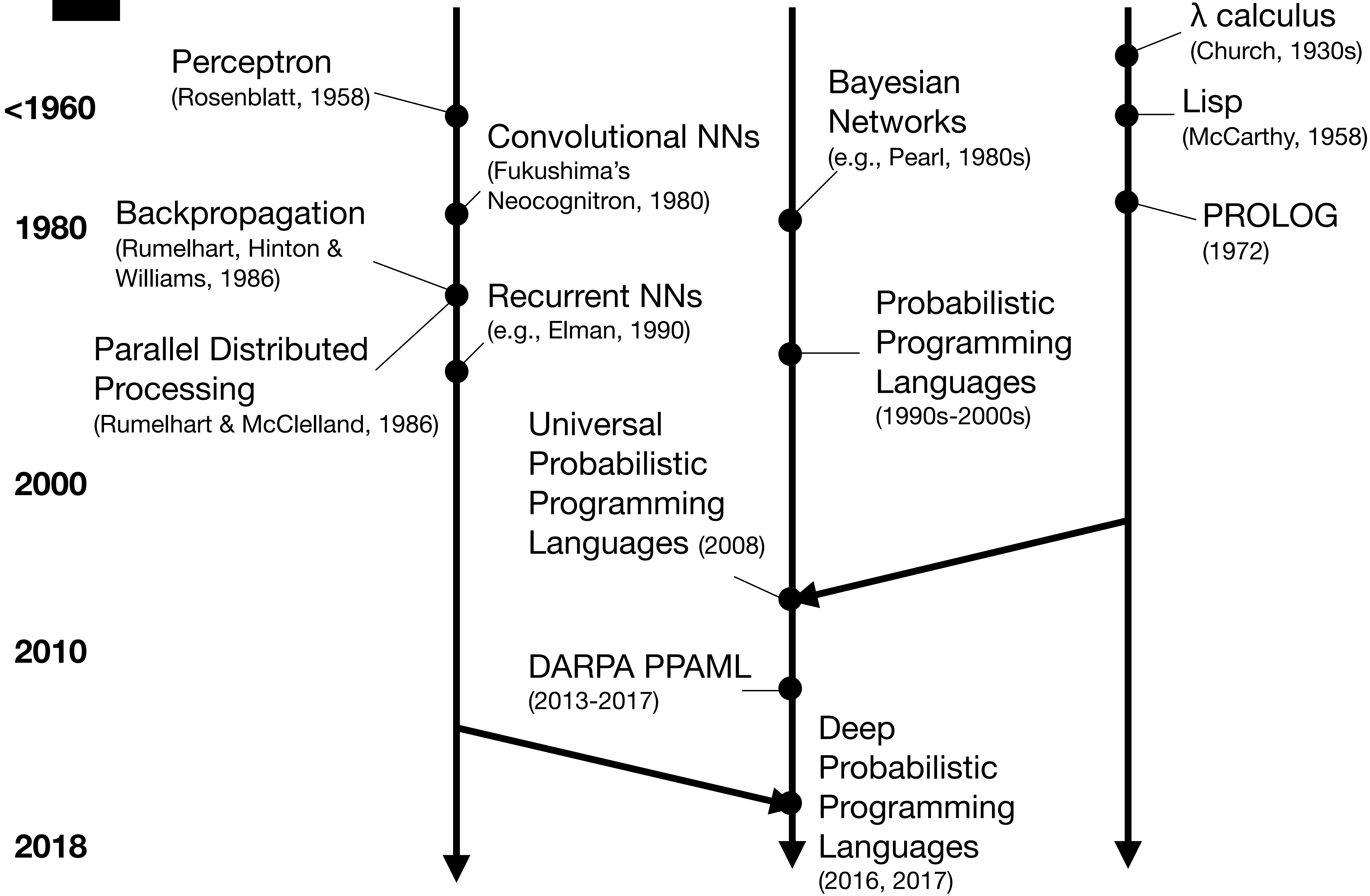
Tutorial participants will be introduced to several key concepts in probabilistic programming, such as stochastic functions, compositionality and recursion, and non-deterministic control flow.

Tutorial participants will be introduced to stochastic variational inference as a powerful optimisation algorithm to perform approximate inference, and how to use SVI in deep probabilistic programs.

A brief (and selective) history



A brief (and selective) history



Why (deep) Probabilistic Programming?

Probabilistic Modelling: Representing and handling uncertainty

+

Programming Languages: "Universality", Expressivity

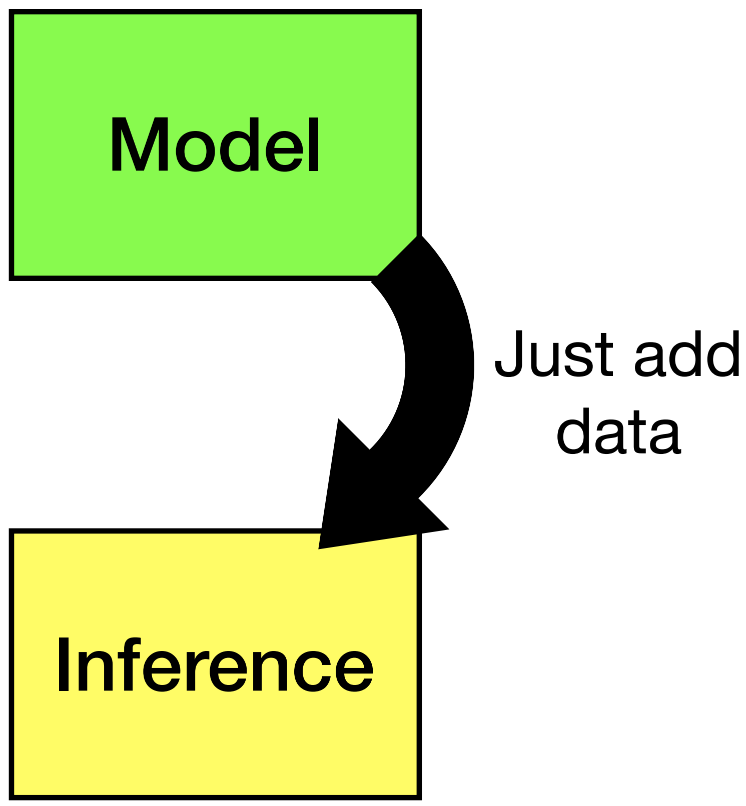
+

Deep Learning: Scalability, Flexibility

- Probabilistic Programs capture abstract knowledge about the world, represented as executable *programs*.
  - Model different sources of uncertainty (more on that later…)
  - **see also Ghahramani (Nature 2015) for an argument for probabilistic machine learning*

Why (deep) Probabilistic Programming?

- **Abstraction away from inference**
  - Modeller can focus on modelling, call libraries to do inference.
    - e.g., How PyTorch, Tensorflow abstracts out backprop
  - Many PPLs come with general-purpose approximate inference algorithms: Variational inference; MCMC, etc



Why (deep) Probabilistic Programming?

- Today: http://www.probabilistic-programming.org lists over 30+ PPLs. Some examples:
  - Church, 2008, Lisp/Scheme

```
(flip 0.5)
```

  - WebPPL (2014), webppl.org, (subset of) Javascript

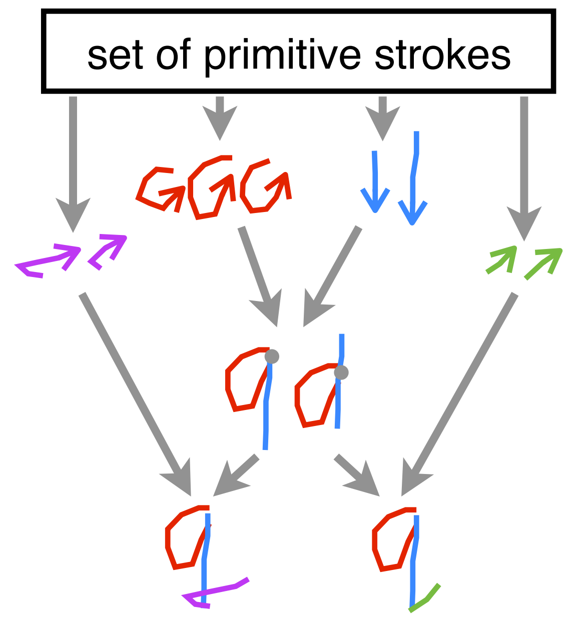
```
var geometric = function() {
  return flip(.5) ? 0 : geometric() + 1;
}
```

- **Deep** PPLs combine theory-driven (“probabilistic”) and data-driven (“deep”) approaches,
  - allowing probabilistic models to learn from high-dimensional, unstructured data (e.g. video).
- Natively integrated with deep learning libraries.
  - Pyro (Uber AI labs 2017; integrated into PyTorch),
  - Edward (2016) / Tensorflow Probability (Google; 2018)

Examples of Probabilistic Programming (i)

Handwriting recognition (Lake, Salakhutdinov, & Tenenbaum, 2015)

Strokes → Parts → Characters

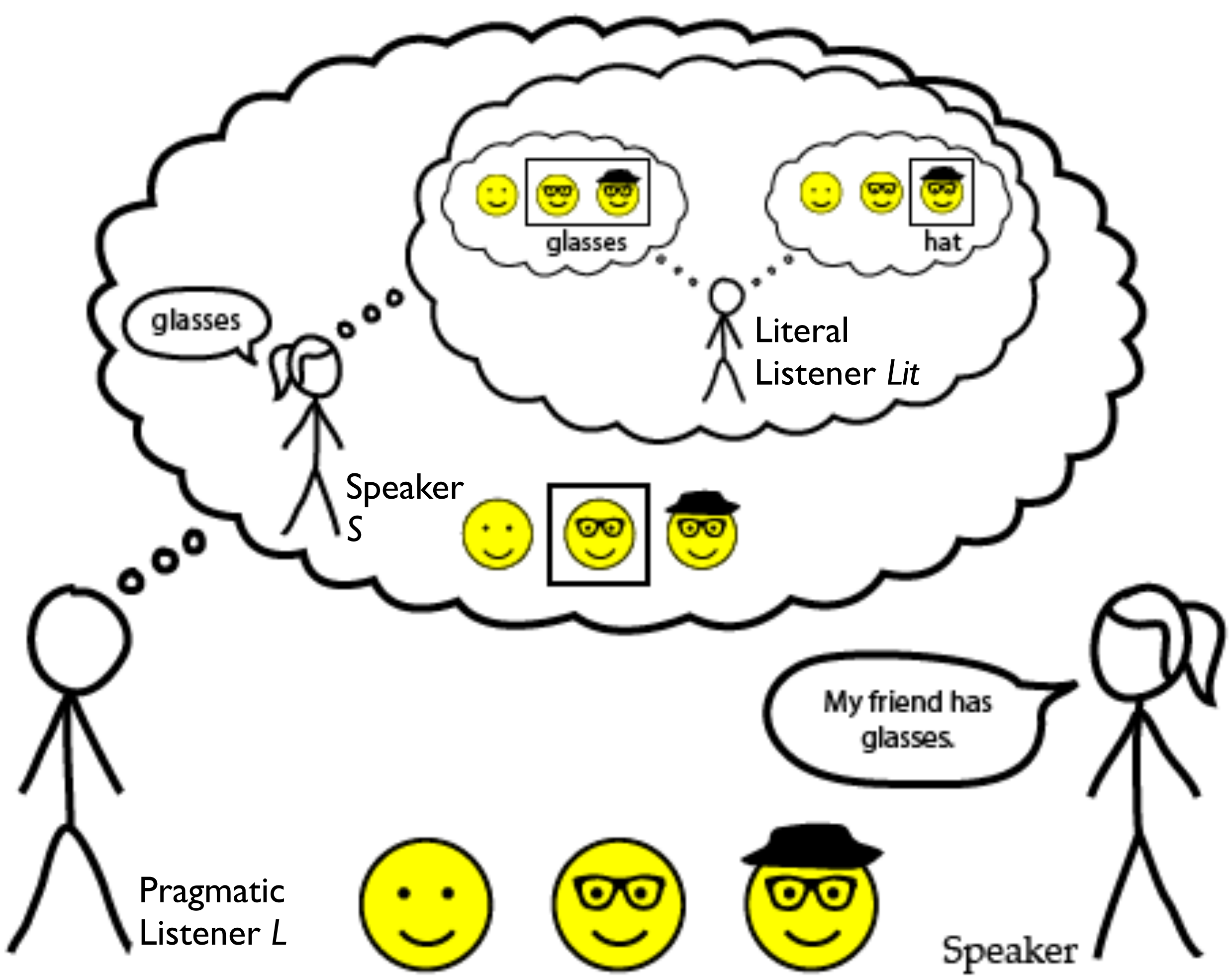



```
generateCharacter() = {
  parts <- Sample_Parts(num_parts)
  // sample with motor variance

  relations <- Sample_Relations(parts)
  // variance in where (sub)parts start
  // and where (sub)parts are joined

  return compose(parts, relations)
  // character concept
}
```

Samples from the Omniglot dataset

Examples of Probabilistic Programming (ii)

Pragmatic understanding in the *Rational Speech Acts* model (Goodman & Frank, 2016)




```
LiteralListener = function(utterance) {
    return worlds consistent with [[utterance]]
}

Speaker = function(world) {
    return utterances proportional to probability
    that LiteralListener(utterance) == world
}

PragmaticListener = function(utterance) {
    return worlds proportional to probability
    that Speaker(world) == utterance
}
```

Frank & Goodman 2012
Goodman & Stuhlmuller, 2013
Goodman & Frank, 2016

Features of Probabilistic Programs (i)

- Stochastic programs => Model different sources of uncertainty
  - Lake et al: Motor variance
  - Rational Speech Acts: Uncertainty in semantics and speaker goals
- More generally, uncertainty can be:
  - (i) incomplete knowledge
    - About the world and others' unobservable mental states; noisy sensor data
  - (ii) incomplete theory
  - (iii) inherent randomness in the generative process.
- Affective Computing:
  - Emotions require inference about latent mental states
  - Individual differences not yet modelled by scientists
  - Inherent randomness

Ong, Soh, Zaki, & Goodman, in press

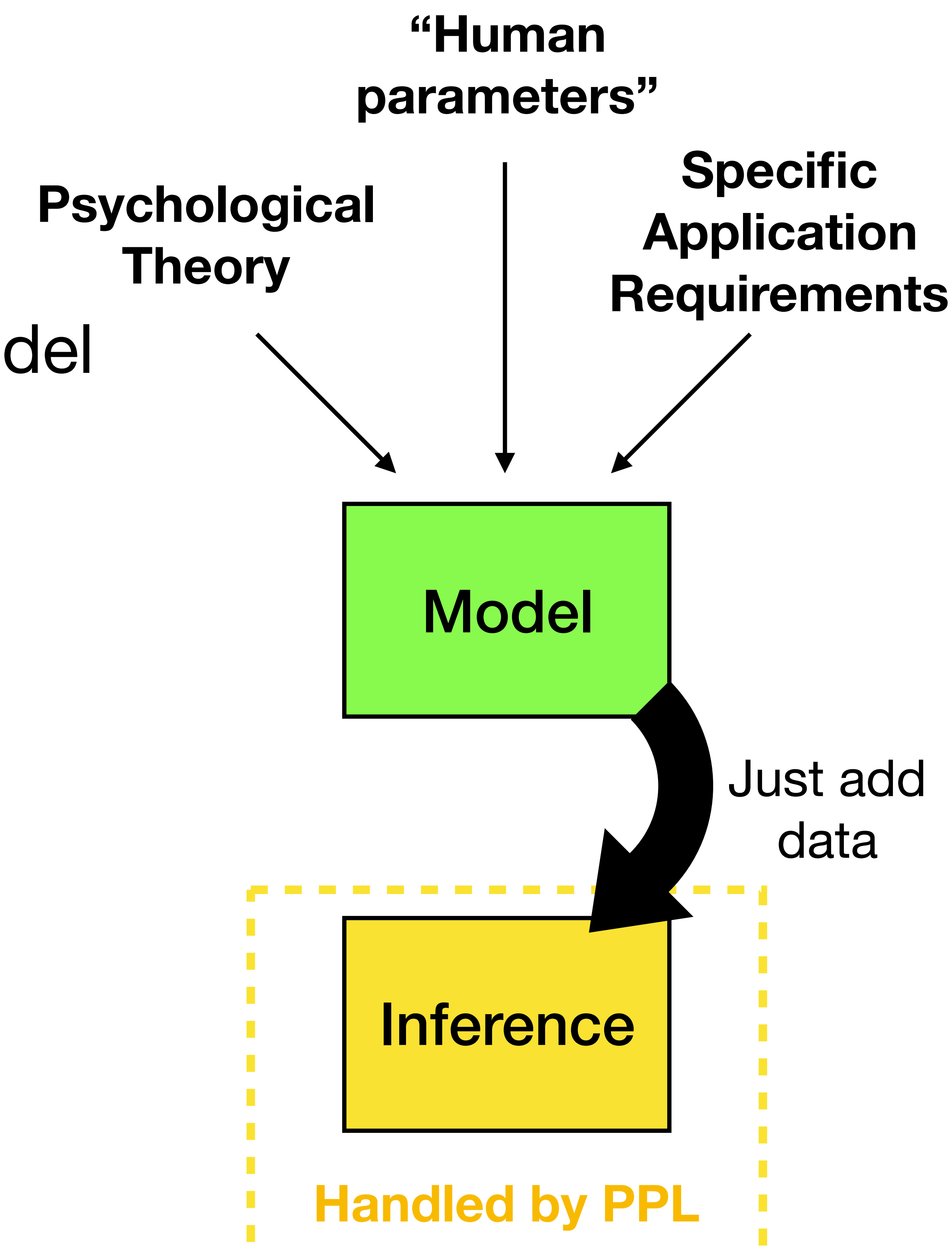
Features of Probabilistic Programs (ii)

- Modularity and compositionality
  - Abstract processes into "modules", re-use modules
  - Build up complexity
  - Examples:
    - Lake et al: Hierarchy of strokes -> sub-parts -> parts -> characters
    - Rational Speech Acts: Nested programs for social reasoning (X thinking about Y thinking about Z)
- Affective Computing:
  - Separate and compose different components
  - Structured reasoning
  - Goal-directed actions [as in POMDPs]
    - e.g., choosing action to maximise Bob's happiness
  - Emotion + Mental States + Norms + ...
- Becoming easier to implement
  - Deep PPL libraries

Ong, Soh, Zaki, & Goodman, in press

General Recipe

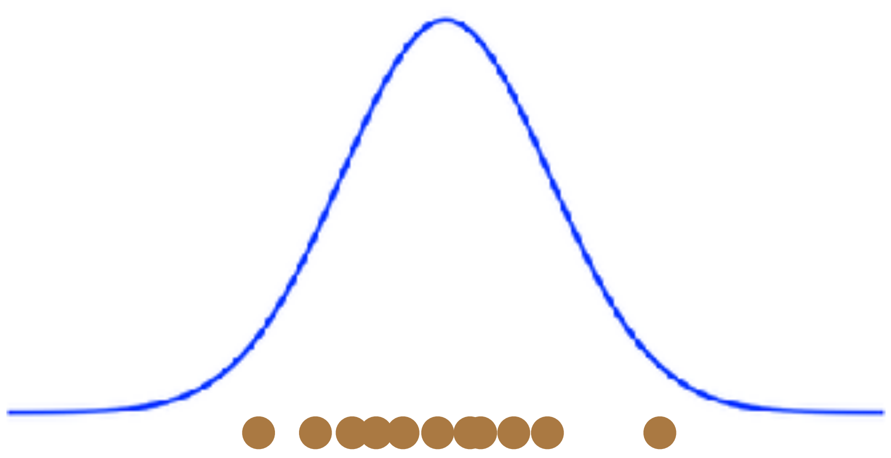
- **Modelling:**
  - Writing down a generative model as a probabilistic program
  - From Theory
  - Has parameters to be learnt
- **Inference**:
  - Learning parameters by conditioning on data
    - Find (or approximate) P(parameters | data)



Stochastic primitives in Pyro (i)

- Stochastic functions:

```
# create a normal distribution object
normal = pyro.distributions.Normal(0, 1)
# draw a sample from N(0,1)
x = pyro.sample("my_sample", normal)
```



https://pyro.ai/examples/intro_part_i.html
https://probmods.org

Stochastic primitives in Pyro (ii)

- Compositionality and Recursion

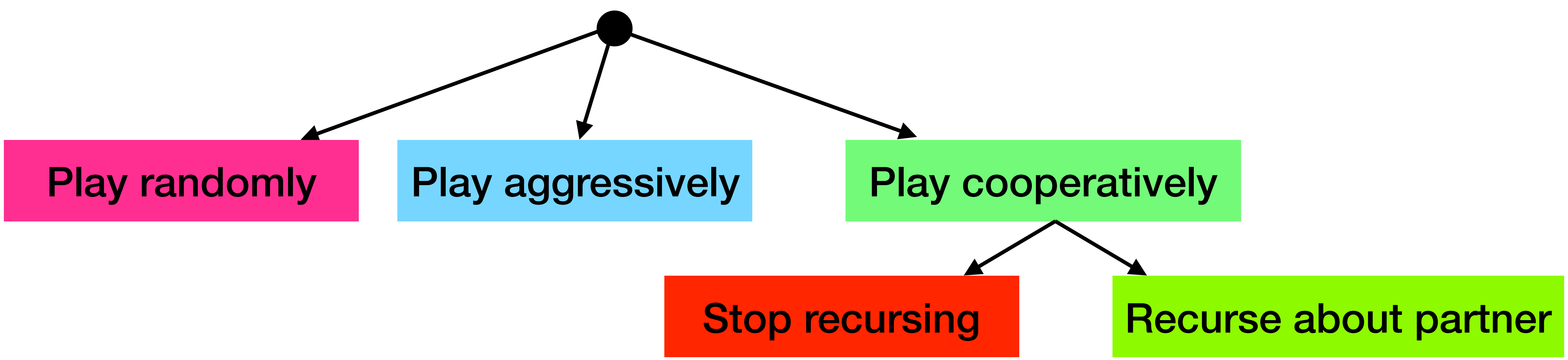
```
# geometric = distribution over number of failures
before the first success
def geometric(probSuccess, t=0):
    x = pyro.sample("x_{}".format(t),
            pyro.distributions.Bernoulli(probSuccess))
    if x.item() == 1:
        return 0
    else:
        return 1 + geometric(probSuccess, t+1)
```

https://pyro.ai/examples/intro_part_i.html
https://probmods.org

Stochastic primitives in Pyro (iii)

- Non-deterministic control flow

```
def leftOrRight(p):
    coinFlip = pyro.sample(pyro.distributions.Bernoulli(p))
    if coinFlip.item() == 1:
        ... # do something: go left
    else:
        ... # do something else: go right
```



Some parts of the code may never get run!

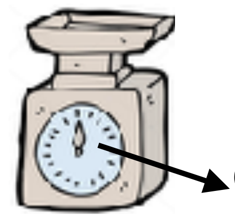
https://pyro.ai/examples/intro_part_i.html
https://probmods.org

Inference as conditioned sampling

Weight??

1) *a priori* guess = density * volume estimate — weight | guess ~ N(guess, 1)

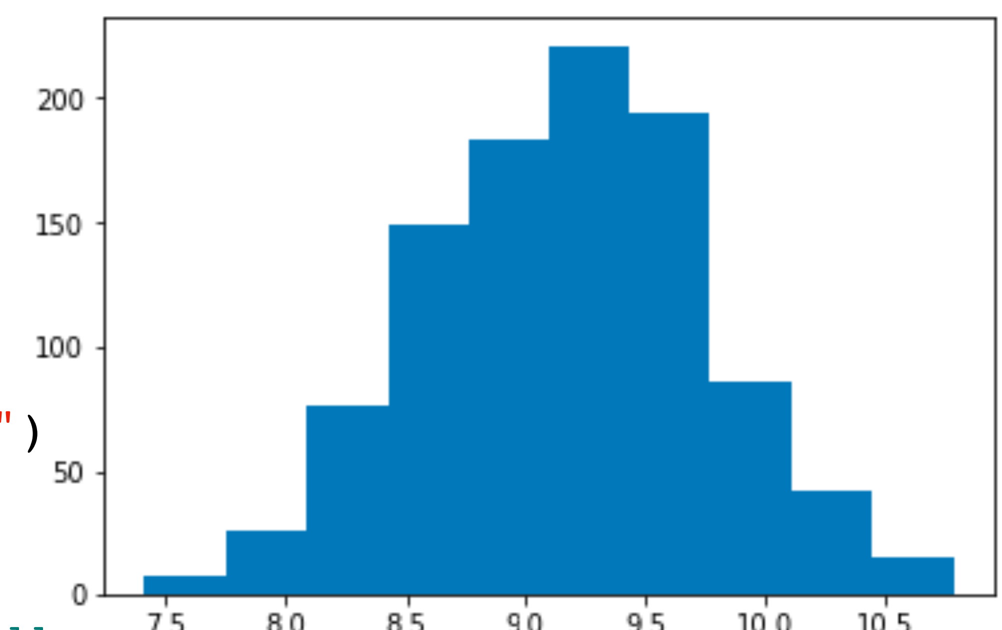
2) Noisy measurement of weight — measurement | weight ~ N(weight, 0.75)



9.5!

```python
def scale(guess):
    weight = pyro.sample("weight", dist.Normal(guess, 1.0))
    return pyro.sample("measurement", dist.Normal(weight, 0.75))

conditioned_scale = pyro.condition(scale, data={"measurement": 9.5})
```

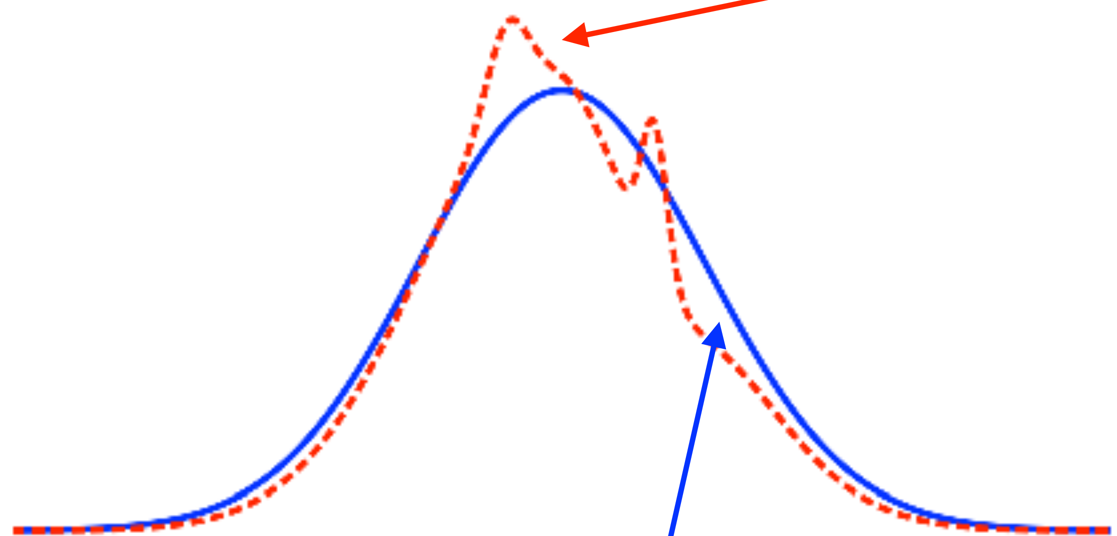
```python
best_guess = 8.5 # let's say
# use importance sampling to infer the posterior
posterior = pyro.infer.Importance(conditioned_scale,
               num_samples=1000).run(best_guess)

# sample from the marginal and plot
marginal = pyro.infer.EmpiricalMarginal(posterior, "weight")
samples = [marginal().detach() for _ in range(1000)]
plt.hist(samples)

## The true posterior, in this case for these values, can be analytically
                                solved: N(9.14, 0.6). See links below.
```



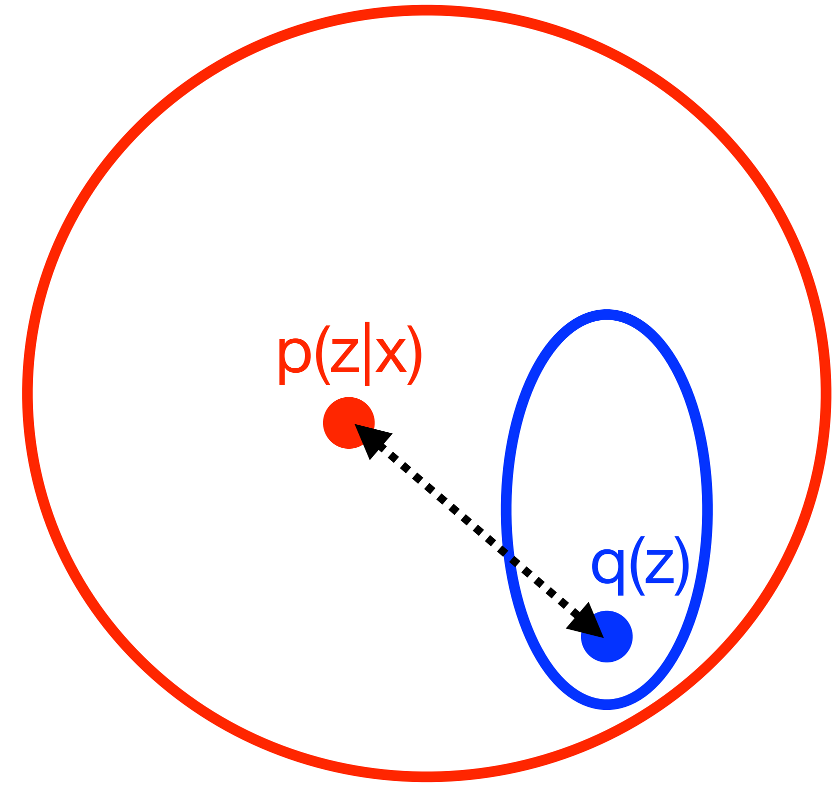
https://pyro.ai/examples/intro_part_ii.html
https://pyro.ai/examples/csis.html

Variational Inference

Don't know the true posterior p(z|x)



Approximate by some ("nice") q(z) that is "close" to p(z|x)



- We define a "Closeness" metric: the Kullback-Leibler Divergence between q and p, $KL\left(q(z)\,||\,p(z\,|\,x)\right) \equiv \mathbb{E}[\log q(z) - \log p(z\,|\,x)]$

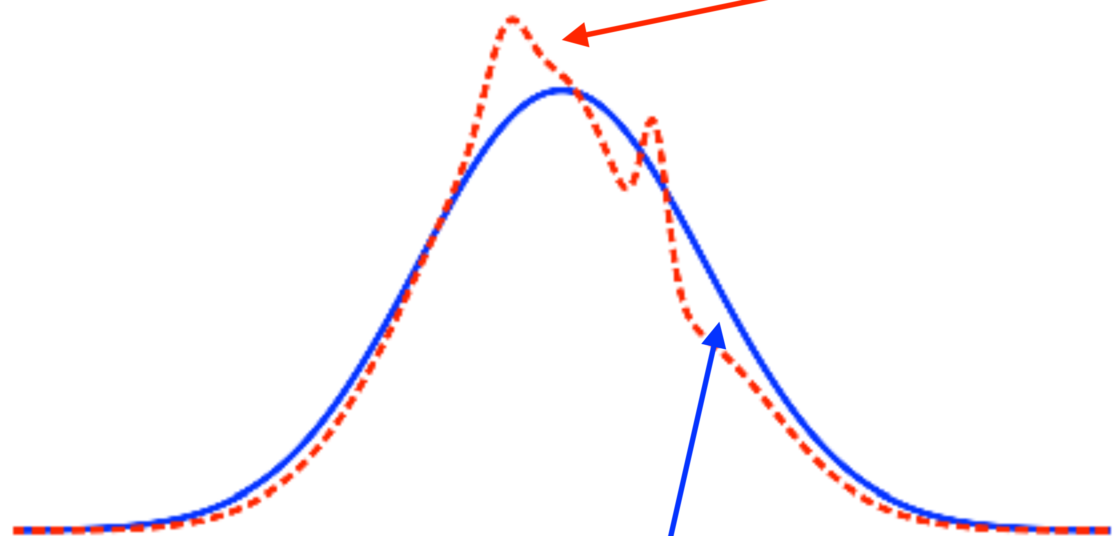
- Want to choose q(z) to minimise the KL:

$$q^*(z) = \arg\min KL\left(q(z)\,||\,p(z\,|\,x)\right)$$

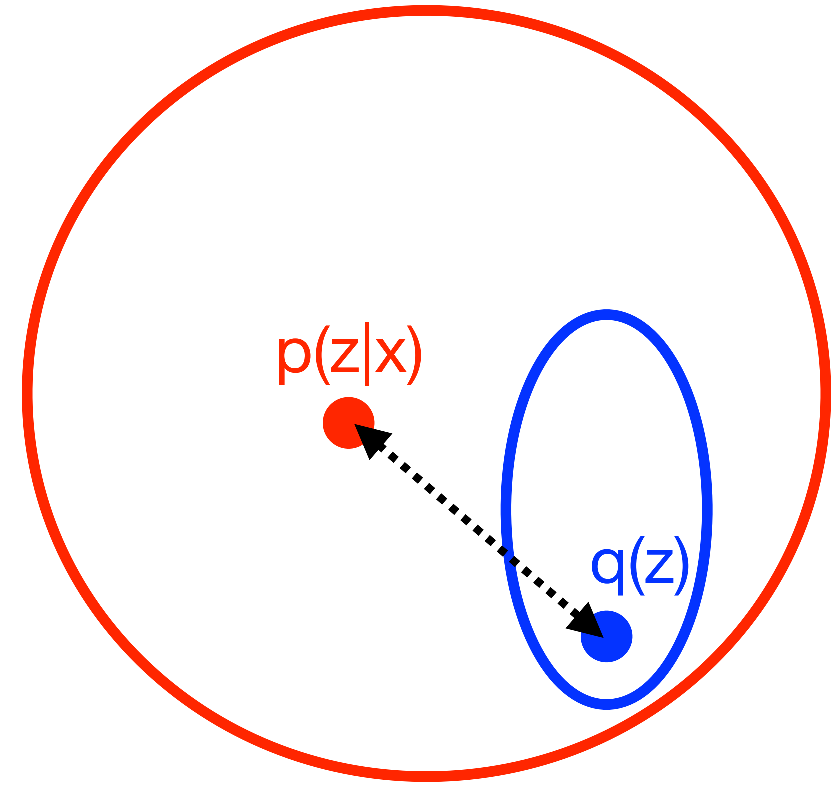
- However, this is still intractable because the KL still contains p(z|x)

Variational Inference

Don't know the true posterior p(z|x)



Approximate by some ("nice") q(z) that is "close" to p(z|x)



- But, some algebra shows:

$$\log p(x) = \underbrace{KL\left(q(z)\,||\,p(z\,|\,x)\right)}_{\geq\, 0} + \underbrace{\mathbb{E}\left[\log p(x,z) - \log q(z)\right]}_{\text{ELBO}(q)}$$

ELBO(q)
Evidence Lower BOund

- Thus, we can maximise the ELBO as a proxy for maximising log p(x)!

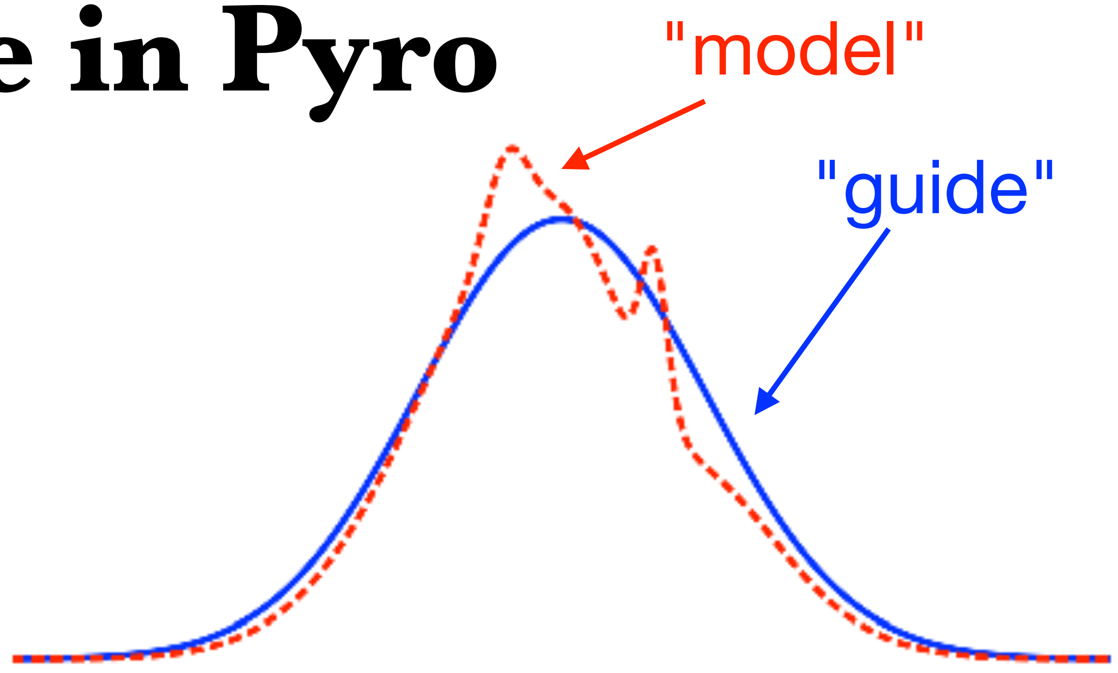
Variational Inference

- Variational Inference replaces the (computationally-intractable) problem of **inference** in a probabilistic model: Solve p(x)

- With a proxy (and computationally-cheaper) **optimisation** problem: Maximize ELBO (also called the variational objective).

- This is the key idea behind recent deep generative models, especially the Variational Autoencoder (VAE; Kingma & Welling, 2014)

Variational Inference in Pyro




```
def model(...):
    ...
    pyro.sample("z", ...)

# "q(z)", e.g. assume Normal with variational parameters mu, sigma
def guide(...):
    ...
    mu = pyro.param("mu", torch.tensor(0.0))
    sigma = pyro.param("sigma", torch.tensor(1.0),
                                constraint=constraints.positive)
    pyro.sample("z", dist.Normal(mu, sigma))

# stochastic variational inference
svi = SVI(model, guide, optimizer, loss_fn)

# abstracted away from model definition
for step in range(num_steps):
    svi.step(data)
```

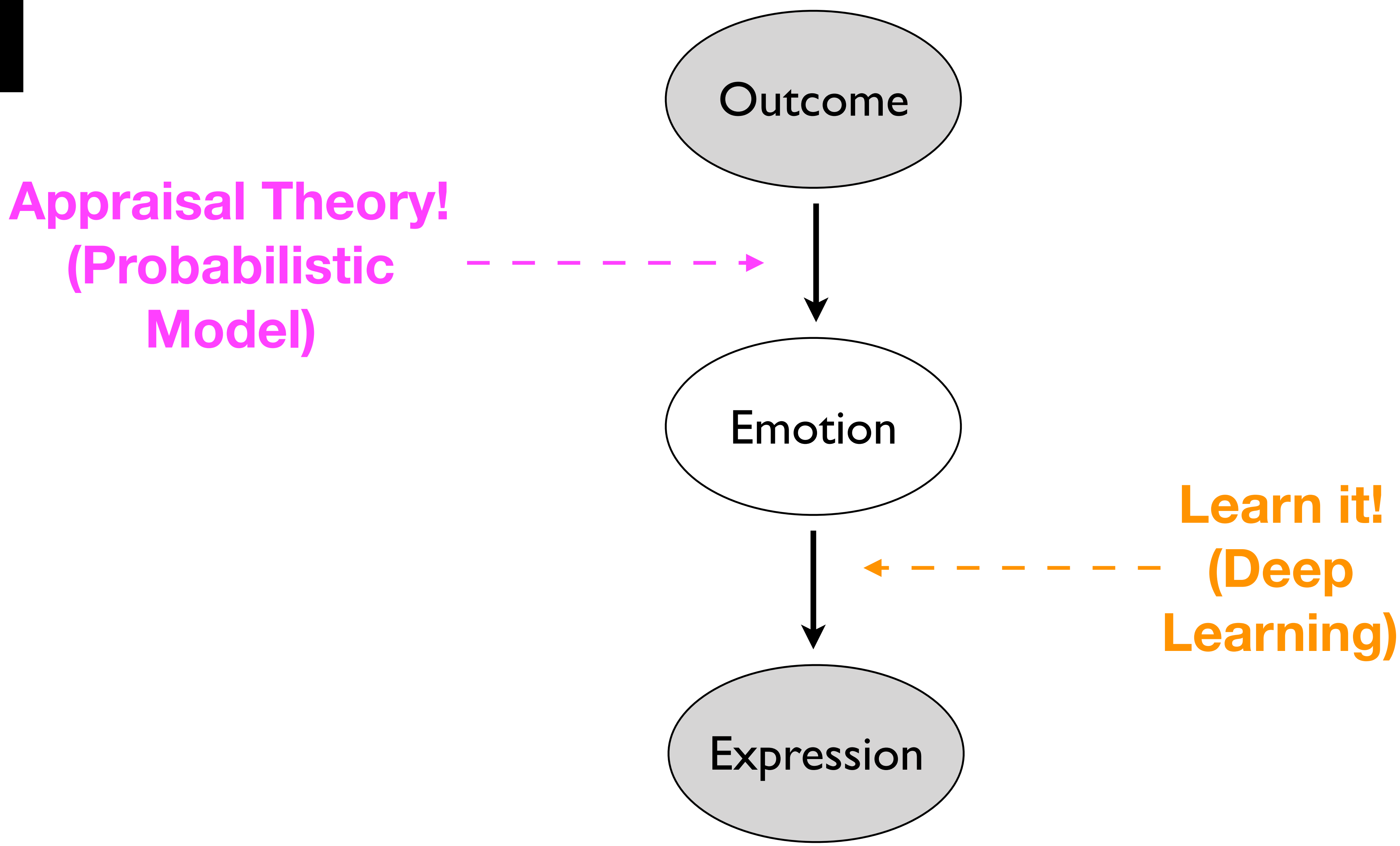
https://pyro.ai/examples/svi_part_i.html

Breaktime!

Worked examples in affective computing

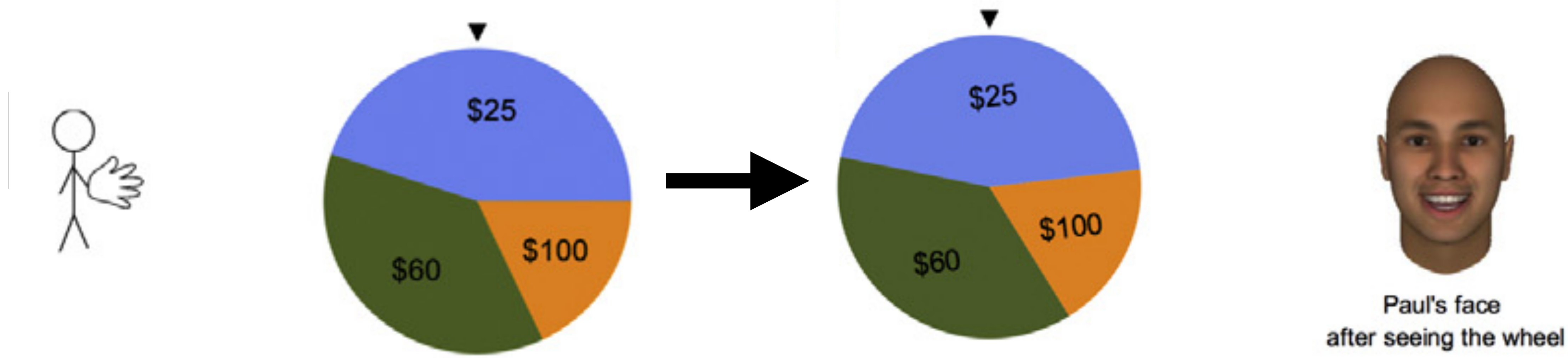
Tutorial website: https://desmond-ong.github.io/pplAffComp/

My Motivating Example



(Ong, Zaki, & Goodman, 2015)

Dataset



+ Participant rating of agent's emotion

Outcome → Emotion → Expression

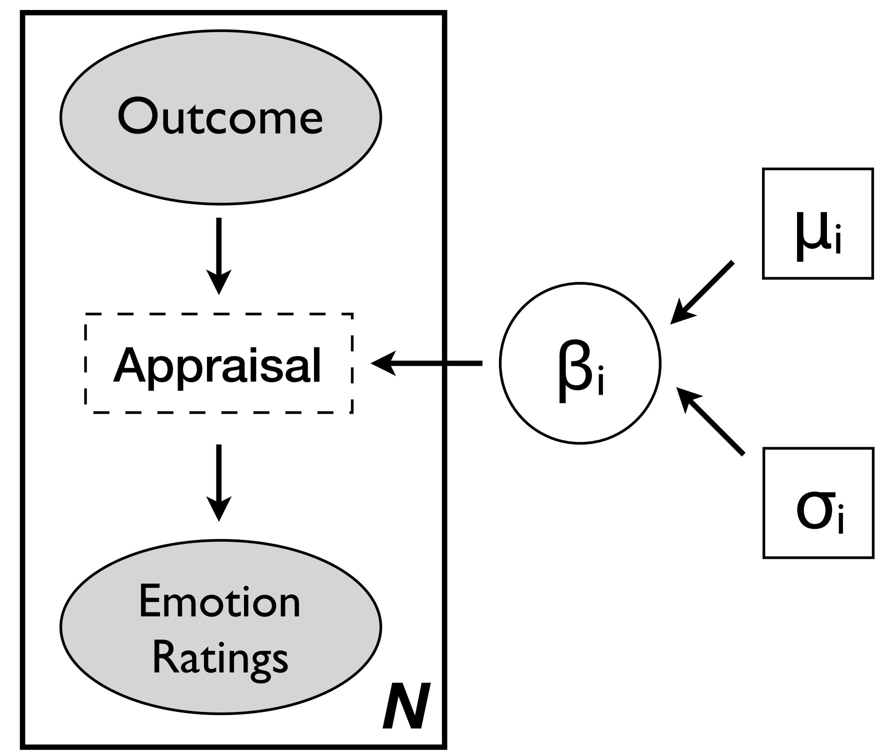
$$P(\text{emotion}|\text{outcome}, \text{face}) \propto \frac{P(\text{emotion}|\text{outcome})P(\text{emotion}|\text{face})}{P(\text{emotion})}$$

(Ong, Zaki, & Goodman, 2015)

Example 1: Modelling Appraisals [via a (Bayesian) Regression]

☆ Incorporating (an instance of) appraisal theory (abstracted into a compute_appraisal() function)


```
class AppraisalRegressionModel():
  def condition(self, outcome, emotion):
    appraisal = self.compute_appraisal(outcome)
    # sample all the b parameters
    b1 = pyro.sample("b1", Normal(mu_1,sd_1))
    ...
    prediction = sum([b_1, ...] * appraisal)
    pyro.sample("observed_emotion",
      Normal(prediction, 1), obs = emotion)
```

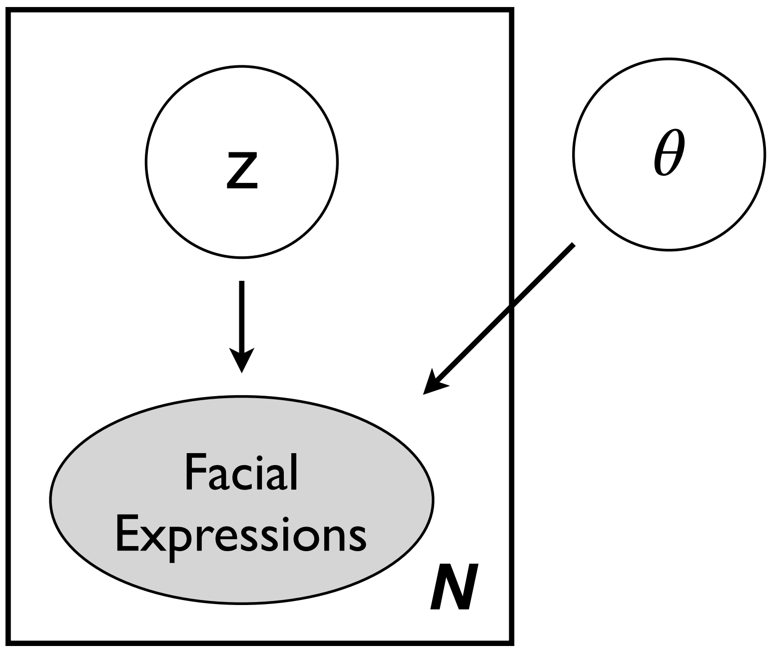


Example 2: Representing/generating faces

☆ Learn from high-dimensional data

(abstracted into a decoder() function, which could be a neural network)

```
class VAE():
  def condition(self, image):
    # sample z given priors
    z = pyro.sample("z",
           Normal(prior_location, prior_scale))
    # generate face from z, conditioned on observed image
    loc = self.Face_Decoder(z)
    pyro.sample("face", dist.Bernoulli(loc), obs=image)

```

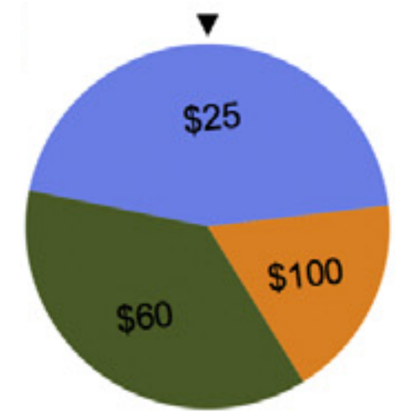


Input

Reconstructions

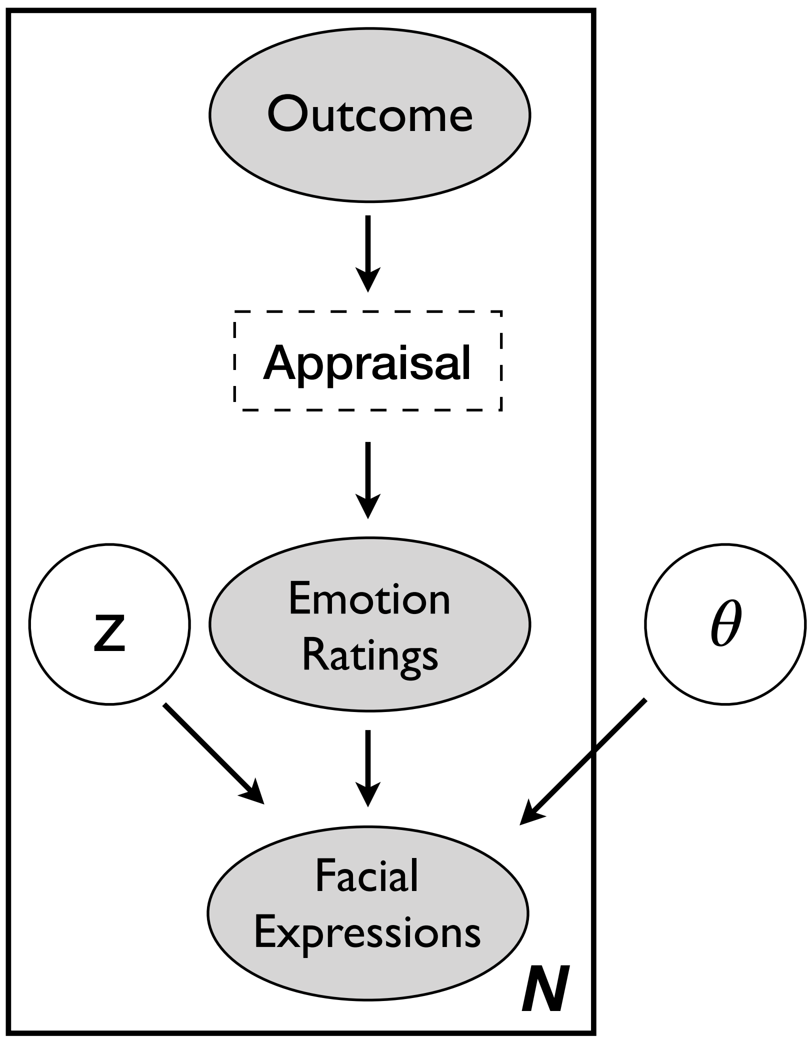
Example 3: Modelling emotion recognition from faces

☆ Learning emotion information in faces



Paul's face
after seeing the wheel

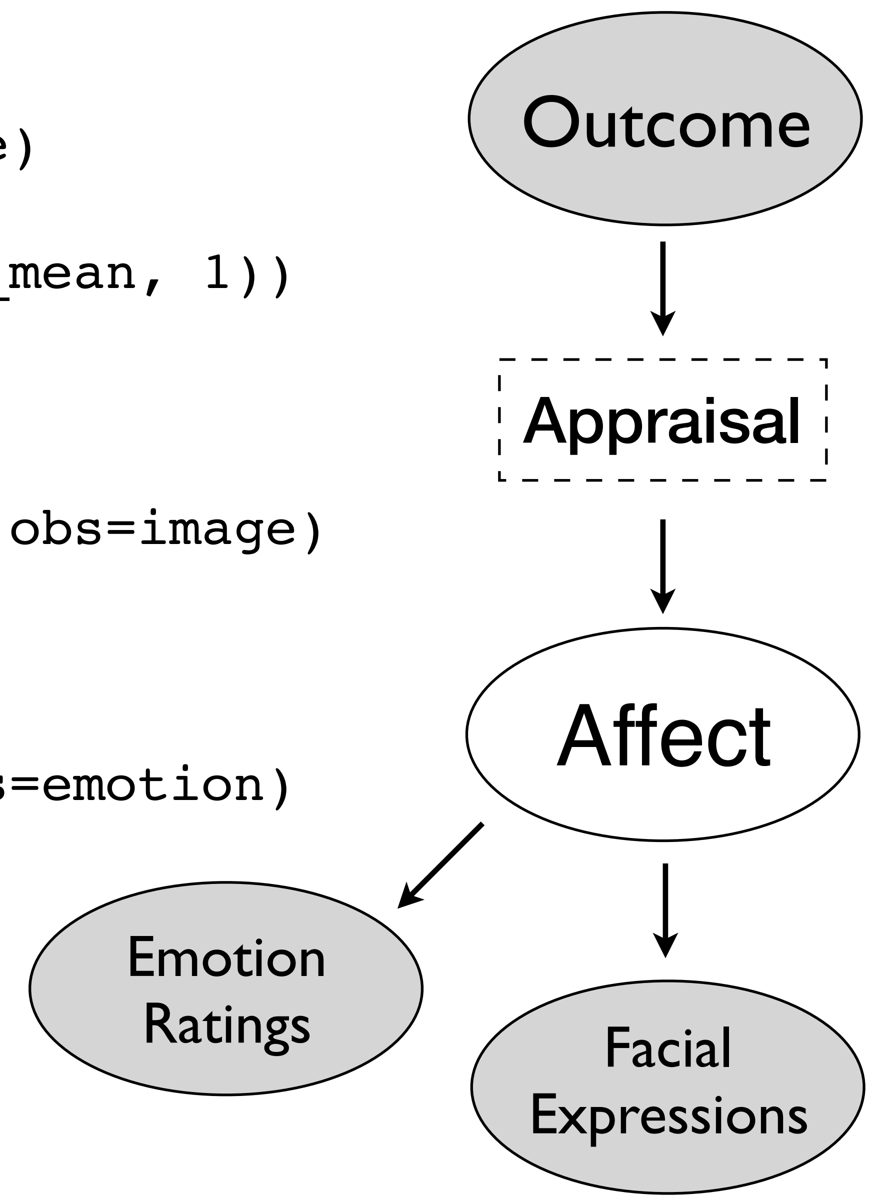
```
class SemiSupervisedVAE():
  def condition(self, outcome, emotion, image):
    # generate emotion from outcome,
      conditioned on observed data
    prediction_mean = self.outcomes_to_emotions(outcomes)
    emo = pyro.sample("emo", Normal(prediction_mean, 1),
                                    obs=emotion)
    # sample z given priors
    z = pyro.sample("z",
           Normal(prior_location, prior_scale))
    # generate face using emotion and z,
      conditioned on observed image
    zEmo = torch.cat((z, emo), 1) # concatenate
    loc = self.zEmoToFace_Decoder(zEmo)
    pyro.sample("face", dist.Bernoulli(loc),
                             obs=image)
```



Example 4: Learning the latent affect space

☆ Learn a latent “affect” variable

```
class MultimodalVAE():
  def condition(self, outcome, emotion, image):
    # generate a new emotion from outcome
    prediction_mean = self.outcomes_to_affect(outcome)
    # sample affect given priors
    affect = pyro.sample("affect", Normal(prediction_mean, 1))
    # generate the facial expression,
      condition on the observed data
    face_mean = self.affectToFace_Decoder(affect)
    face = pyro.sample("face", Bernoulli(face_mean), obs=image)
    # generate the outcome ratings,
      condition on the observed data
    emo_mean = self.affectToRating_Decoder(affect)
    emo = pyro.sample("emo", Normal(emo_mean, 1), obs=emotion)

```



Summary

- Combining theory-based and data-driven approaches.
- <u>**Abstraction**</u>: PPLs abstract away inference, allowing modellers to focus on model building.
- (Deep) PPLs combine the benefits of probabilistic approaches:
  - Encode domain-knowledge
  - Model different sources of uncertainty
- With the benefits of deep learning:
  - Optimized approximate-inference algorithms e.g. variational inference
  - Embed parts that are best learnt via deep learning ("perceptual" tasks)
- And the benefits of programming languages:
  - **Modularity**: Test different theories (of emotion) by substituting out modules
  - **Compositionality**: build up more complex reasoning

Zhi-Xuan Tan
(MIT)

Harold Soh
(Nat'l University of Singapore)

Jamil Zaki
(Stanford)

Noah Goodman
(Stanford, Uber)

JP Chen

Eli Bingham

Uber AI Labs

Thanks!

dco@comp.nus.edu.sg
web.stanford.edu/~dco

Reference paper:

Ong, D. C., Soh, H., Zaki, J., & Goodman, N. D. (in press). Applying Probabilistic Programming to Affective Computing. *IEEE Transactions on Affective Computing* https://arxiv.org/abs/1903.06445

Materials/Code: https://desmond-ong.github.io/pplAffComp/