

AI Programming: Lecture 5

Advanced Topics in Probabilistic Programming

Markus Böck and Jürgen Cito

Research Unit of Software Engineering

Table of contents

1. Custom Inference
2. Data-Driven Inference
3. Probabilistic Programs as Proposals
4. Deep Probabilistic Programming

Recap

- Probabilistic programs can describe any probabilistic model
- Underlying models can be difficult to describe mathematically
 - Unbounded number of random variables
 - Stochastic branching
 - Dynamic distributions allowed (non-static support)
- Efficient general-purpose inference is hard

Recap

- General-purpose inference algorithms exist
 - importance sampling
 - single-site MH
 - Can be inefficient
- Imposing restrictions on the probabilistic program allows us to optimise inference
 - fixed, finite number of continuous variables
 - gradient-based inference: HMC, ADVI
 - Still work for a large class of models

Outlook

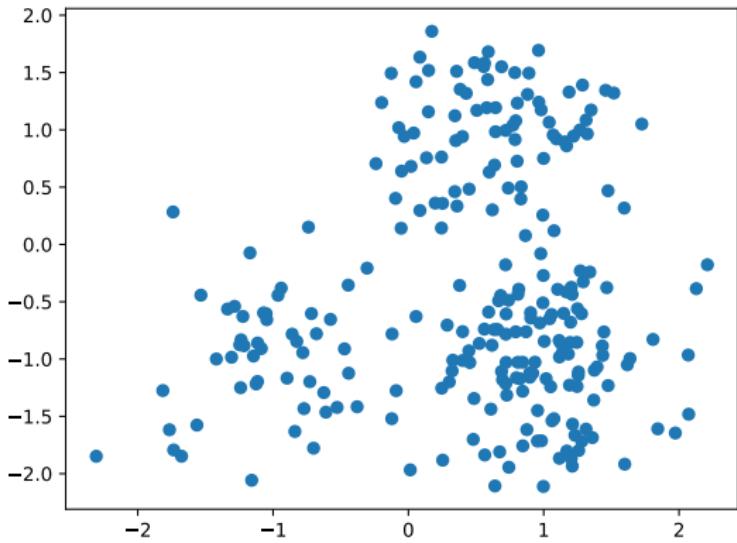
- We can **optimise inference for individual models**
- **Custom Inference:** manually exploit structure of model
- **Data-Driven Inference:** use observed data to improve proposals
- **Probabilistic Programs as Proposals:** convenient way to customise inference
- **Deep Probabilistic Programming:** learning proposals (and models) from data

Custom Inference

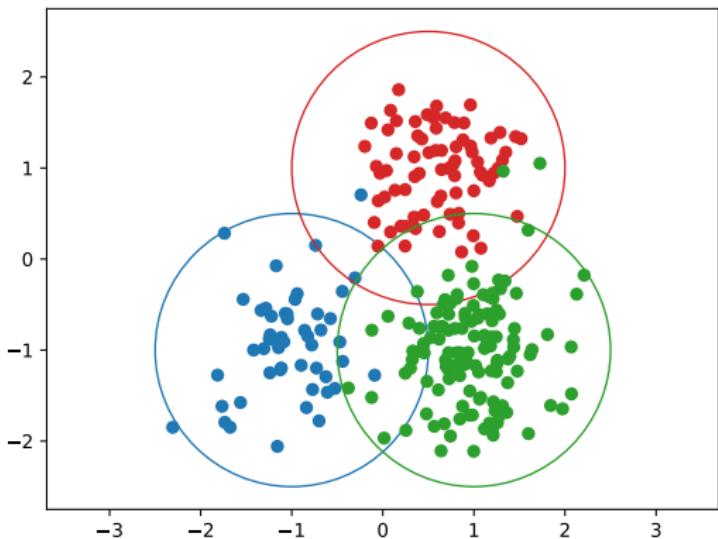
Infinite Mixture Models: Where single-site MH fails

- Number of clusters:
 $K \sim \text{Poisson}(5)$
- Probability of being in cluster k , p_k :
 $p \sim \text{Dirichlet}(1/K)$
- Cluster centers, $k = 0, \dots, K$:
 $\mu_k^x \sim \text{Uniform}(-3,3),$
 $\mu_k^y \sim \text{Uniform}(-3,3)$
- Cluster spread, $k = 0, \dots, K$:
 $\sigma_k^2 \sim \text{InverseGamma}(1,1)$
- Cluster membership, $i = 1, \dots, N$:
 $z_i \sim \text{Categorical}(p)$
- Observed data, $i = 1, \dots, N$:
 $x_i \sim \text{Normal}(\mu_{z_i}, \sigma_{z_i})$
- Unbounded number of random variables
- Discrete variables
- \Rightarrow no HMC / ADVI
- High-dimensional
- \Rightarrow no IS / LW
- but single-site MH is applicable in principle

Data set

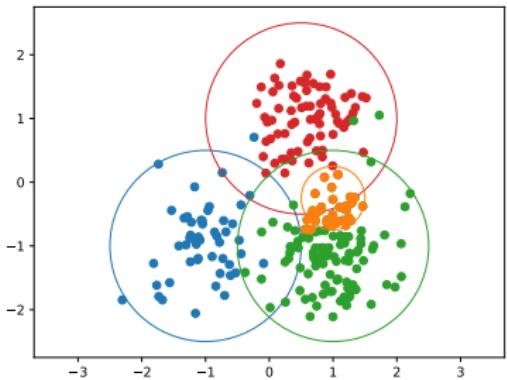


Ground truth



Single-site update

Updating the number of clusters K



- Adding clusters is easy: sample new cluster center and deviation
- How can we **remove the orange cluster?**
- Change K from 4 to 3 (single-site)
- Changes dimension of p (so current p has 0 log-prob?)
- Fix: sample p_k individually
- All memberships $z_i = 4$ have log-prob 0.

In theory, this update can happen, but is very low probability. All $z_i = 4$ have to be changed before setting $K = 3$.

Designing a Custom Inference Algorithm

In each iteration, we pick one type of move at random

1. Updating cluster centers μ_k and deviations σ_k
2. Reweighting clusters – updating p
3. Updating the memberships z_i
4. Merging two randomly selected clusters.
5. Splitting one random cluster

Designing a Custom Inference Algorithm - 1

Updating cluster centers μ_k and deviations σ_k

We can simply do random walk Metropolis Hastings updates.

Slightly perturbing the current values.

Designing a Custom Inference Algorithm - 2

Reweighting clusters – updating p

Let n_k be the number of data points allocated to cluster k .

We expect that

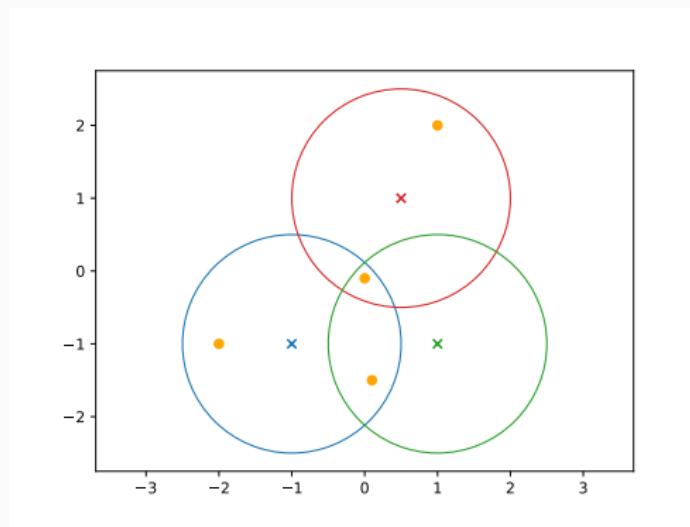
$$\frac{n_k}{N} \approx p_k.$$

We can update p reflecting this relationship:

$$p \sim \text{Dirichlet}(n_1, \dots, n_K)$$

Designing a Custom Inference Algorithm - 3

Updating the memberships z_i



$$\tilde{w}_k := \mathcal{N}(x_i; \mu_k, \sigma_k) \propto \exp\left(-\frac{1}{2\sigma_k^2}(x_i - \mu_k)^\top(x_i - \mu_k)\right), \quad w_k := \frac{\tilde{w}_k}{\sum_{k=1}^K \tilde{w}_k}$$
$$z_i \sim \text{Categorical}(w_1, \dots, w_K)$$

Designing a Custom Inference Algorithm - 4

Merging two randomly selected clusters

Choose two "neighbouring" clusters with weights p_i , means μ_i and deviations σ_i at random, such that

$$\|\mu_1 - \mu_2\|_2 \leq \|\mu_1 - \mu_j\|_2, \quad \text{for } j = 1, \dots, K.$$

Match moments for isotropic Normals of dimension d :

$$p_* = p_1 + p_2 \tag{1}$$

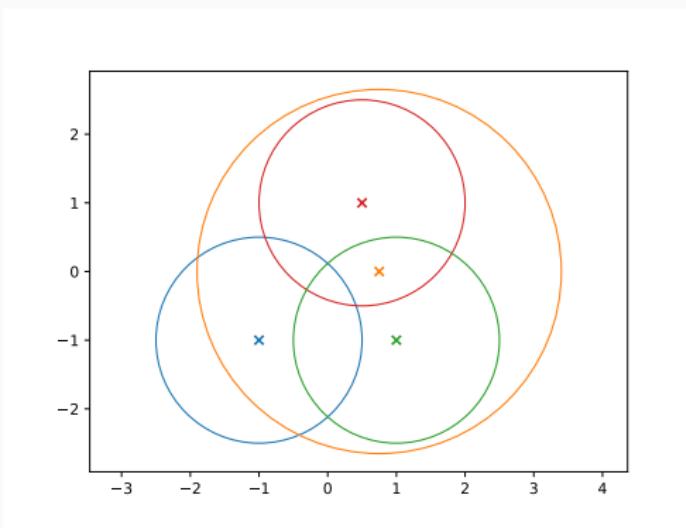
$$p_* \mu_* = p_1 \mu_1 + p_2 \mu_2 \tag{2}$$

$$p_*(\mu_*^\top \mu_* + d\sigma_*^2) = p_1(\mu_1^\top \mu_1 + d\sigma_1^2) + p_2(\mu_2^\top \mu_2 + d\sigma_2^2) \tag{3}$$

and update memberships z_i .

Designing a Custom Inference Algorithm - 4

Merging two randomly selected clusters



Merge red and green cluster to orange.

Designing a Custom Inference Algorithm - 5

Splitting one random cluster

Select cluster at random with weight p_* , mean μ_* and deviation σ_* .

Draw auxiliary variables:

$$u_1 \sim \text{Beta}(2, 2), u_2 \sim \text{Dirichlet}(2, \dots, 2) \in \mathbb{R}^d, u_3 \sim \text{Beta}(1, 1)$$

$$w_1 = w_* u_1, \tag{4}$$

$$w_2 = w_*(1 - u_1) \tag{5}$$

$$\mu_1 = \mu_* - u_2 \sigma_* \sqrt{d \frac{w_2}{w_1}} \tag{6}$$

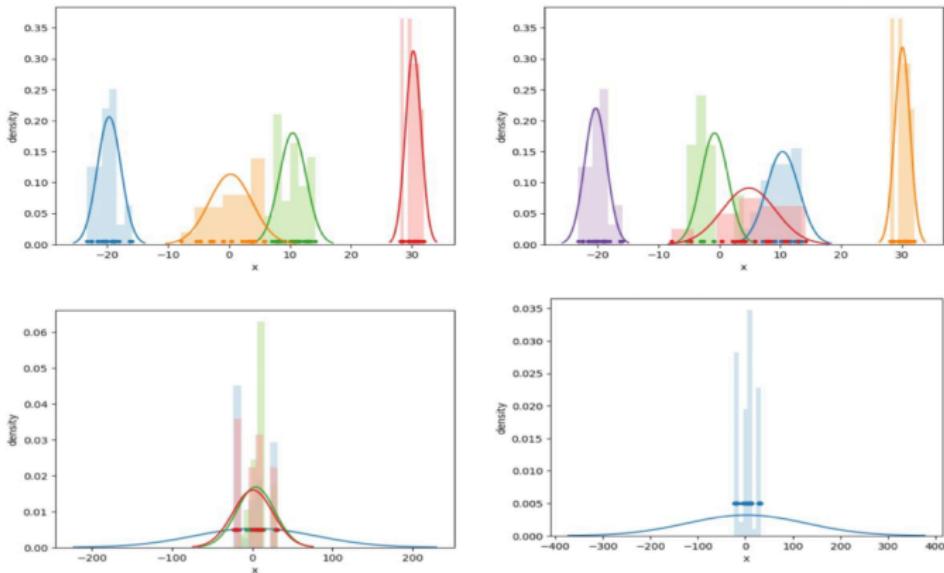
$$\mu_2 = \mu_* + u_2 \sigma_* \sqrt{d \frac{w_1}{w_2}} \tag{7}$$

$$\sigma_1 = u_3 (1 - u_2^\top u_2) \sigma_*^2 \frac{w_*}{w_1} \tag{8}$$

$$\sigma_2 = (1 - u_3) (1 - u_2^\top u_2) \sigma_*^2 \frac{w_*}{w_2} \tag{9}$$

These variables satisfy equations (1) - (3). Thus, merging the two randomly created clusters results in the original cluster (p_*, μ_*, σ_*) .

Designing a Custom Inference Algorithm - Results



(b) Two Samples from the Inferred Posterior:
Richardson & Green's Data-driven MCMC (top),
BLOG Ancestral Sampling (bottom)

Designing a Custom Inference Algorithm

- In the proposal, we make use of **auxiliary random variables**
- This makes **computing the acceptance probability non-trivial**
- It is key to be able to "undo" moves, e.g. merge – join
- This is called **reversible-jump MCMC**
- It is a special case of **involutive MCMC**
- More details in: On Bayesian Analysis of Mixtures with an Unknown Number of Components (with discussion)
https://academic.oup.com/jrsssb/article-pdf/59/4/731/49588858/jrsssb_59_4_731.pdf

Data-Driven Inference

Data-Driven Proposals = Biased Inference?

- It is often good practice to chose *uninformative* priors, i.e. we do not prefer any values for the latent variables *a-priori*
- However, with the proposals, we want to stir inference towards high probability areas of the *posterior*
- We can **use the observed data to construct proposals as close to the posterior as possible**

However, to ensure convergence to the true posterior proposals have to satisfy following properties:

- **Unconditional proposals** $Q(x)$: if a state x is possible according to the model $P(x) > 0$, then it has to be possible according to the proposal $Q(x) > 0$
- **Conditional proposals** $Q(x'|x)$: any state should be reachable from any other state in any number of steps less or equal to a fixed number N .

Common strategy:

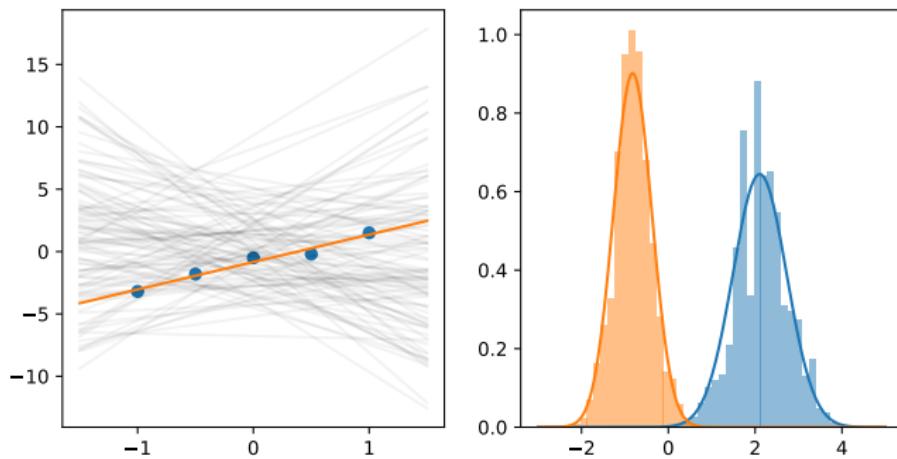
One way of constructing data-driven proposals is to use a **heuristic to estimate the mode** of the target distribution (or one of its conditional distributions) and to sample values near the estimate of the mode, but with **noise added**.

With enough data:

mode of posterior \approx maximum likelihood estimator

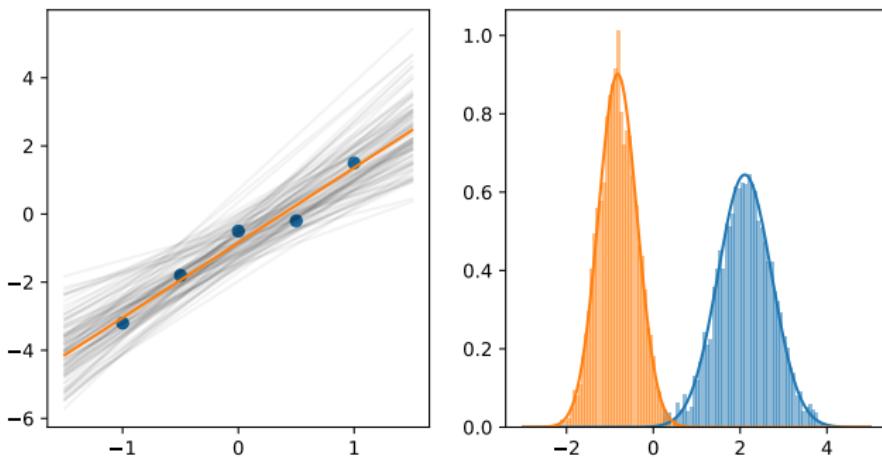
Data-Driven Inference

Linear regression:
propose from prior



Data-Driven Inference

Linear regression: propose from Normals centered at ordinary least squares (OLS) solution



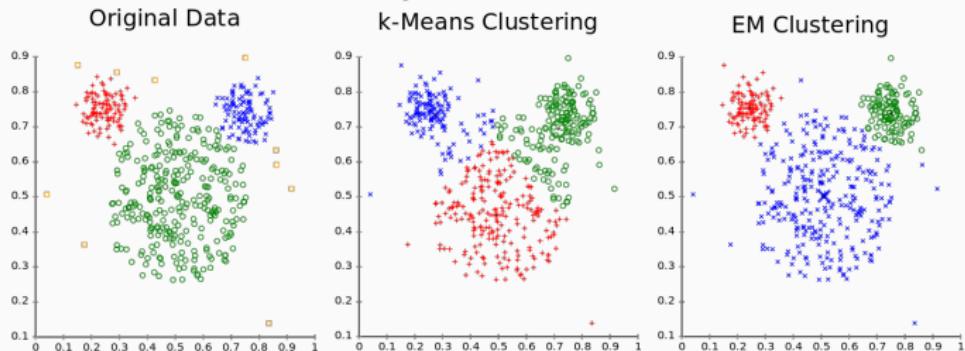
Data-Driven Inference

GMM:

Sample number of clusters $K \sim \text{Poisson}(5)$

Run k-means clustering and perturb the result.

Different cluster analysis results on "mouse" data set:



Probabilistic Programs as Proposals

Probabilistic Programs as Proposals

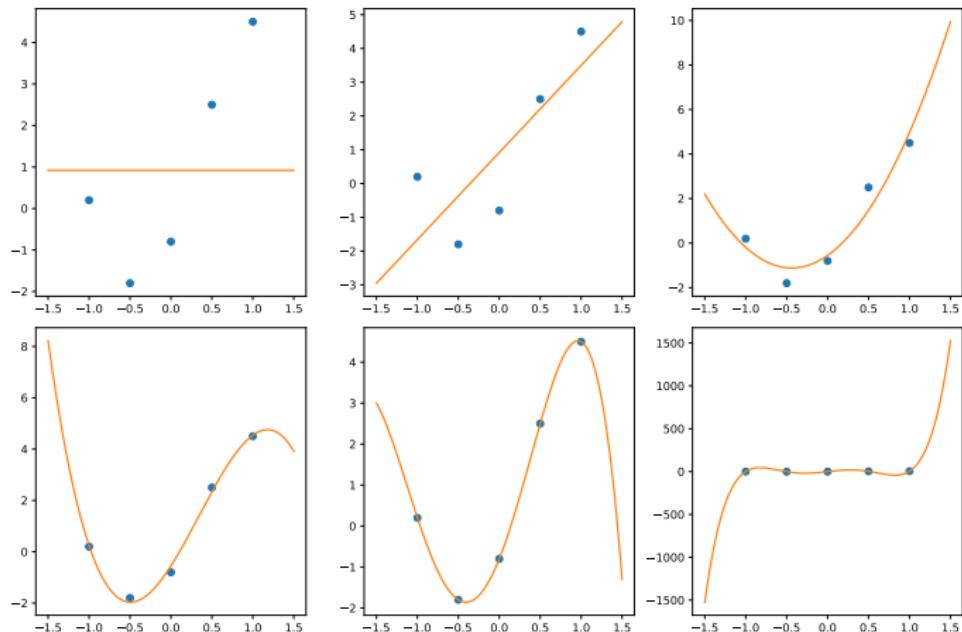
As proposals get more complex it is more convenient to write them programmatically.

Key idea: We can write a probabilistic program and use it for generating proposal in the inference for another program.

These programs are called *guides*.

Gen (and Pyro): programmable inference

Example: Polynomial Regression in Gen.jl



Example: Polynomial Regression in Gen.jl

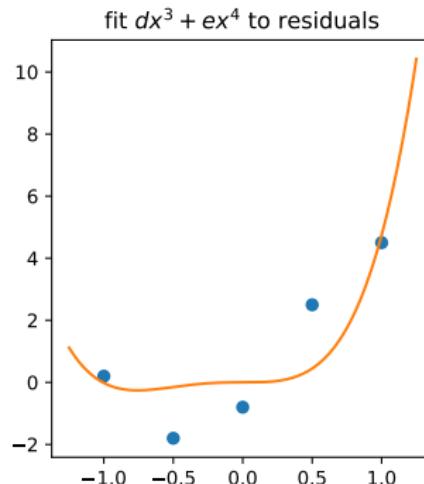
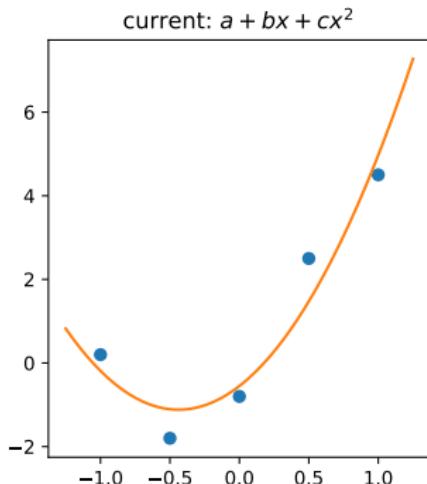
```
1 @gen function poly_model(x_coordinates)
2     degree ~ uniform_discrete(0,4)
3     var ~ inv_gamma(1,1)
4     coefficients = [({{:c,i}} ~ normal(0,1)) for i in 0:degree]
5     for i=1:length(x_coordinates)
6         x = x_coordinates[i]
7         mu = 'coefficients * x.^{0:degree}
8         {(:y,i)} ~ normal(mu, sqrt(var))
9     end
10 end
```

```
1 @gen function poly_proposal_prior(x_coordinates)
2     degree ~ uniform_discrete(0,4)
3     var ~ inv_gamma(1,1)
4     coefficients = [({{:c,i}} ~ normal(0,1)) for i in 0:degree]
5 end
```

Example: Polynomial Regression in Gen.jl

Idea: Iteratively sampling coefficients.

We have currently polynomial of 2nd degree.



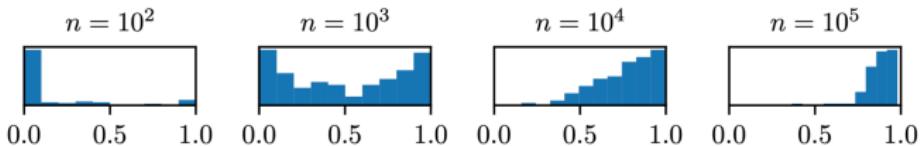
Sample value centered around OLS solution for d .

Example: Polynomial Regression in Gen.jl

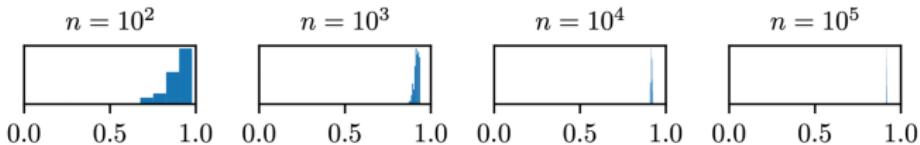
```
1 @gen function poly_proposal_data_driven(x_coords, y_coords)
2     # noise for each coefficient
3     scales = [0.395, 0.242, 0.088, 0.020, 0.007]
4     n = length(x_coords)
5     degree ~ uniform_discrete(0,4)
6     coeffs = [NaN for i in 0:degree]
7     predicted = zeros(n)
8     for i in 0:degree
9         residuals = y_coords .- predicted # elementwise subtraction
10        # fit a polynomial to residuals with coefficients 0..i-1 fixed to zero
11        est_coeffs = least_squares(x_coords, residuals, degree, min_degree=i)
12        coeffs[i+1] = {{(:c,i)} ~ cauchy(est_coeffs[1], scales[i+1])}
13        predicted = [dot(coeffs, x.^{0:i}) for x in x_coords]
14    end
15    # use variance of residuals to get estimate for model noise
16    residuals = y_coords .- predicted
17    var ~ inv_gamma(1 + n/2, 1 + 0.5 * dot(residuals, residuals))
18 end
```

Example: Polynomial Regression in Gen.jl

Estimate for the probability of degree = 3



(a) Estimates from self-normalized importance sampling with a prior proposal.



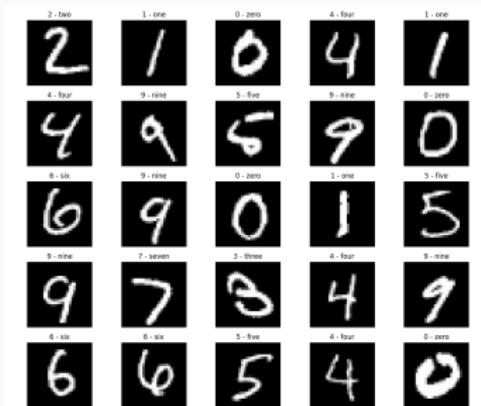
(b) Estimates from self-normalized importance sampling with a data-driven proposal.

Deep Probabilistic Programming

Deep Probabilistic Programming: Motivation

- **Non-programmability:** For many data modalities that are commonly considered in ML and AI, including images and natural language, it is **near-impossible to fully specify a probabilistic program** that defines a sufficiently realistic distribution over data.
- **Scalability:** Models in ML and AI are routinely trained on **very large datasets**. Most inference methods that we have considered so far do not scale to such large datasets without additional modifications.
- These challenges can be addressed by combining inference methods from probabilistic programming with differentiable programming techniques from deep learning research.

Deep Probabilistic Programming: Non-programmability



Non-programmability:

How to implement a probabilistic program that generates 28×28 px images of hand-written digits?

Sample digit $\sim \text{DiscreteUniform}(0, 9)$,

and then ... ??

Deep Probabilistic Programming: Neural Networks

- Neural networks are universal function approximators
- Use neural network η_λ with parameters λ in the program to flexibly model relationship between latents and observes
- latent: digit; observed: image
- $\text{image}[x,y] \sim \text{Bernoulli}(\eta_\lambda(\text{digit})[x,y])$
- probability of pixel being white \cong gray scale value
- Full model specification: $P(\Theta, X|\lambda)$
- Learn λ to fit our data set

Deep Probabilistic Programming

How to learn λ (model parameters)?

- Fully Bayesian treatment: λ are additional latent variables, set prior $P(\lambda)$ and estimate posterior
$$P(\lambda, \Theta | x_1, \dots, x_n) \text{ or MAP } \operatorname{argmax}_{\lambda} P(\lambda, | x_1, \dots, x_n) ?$$
- → Bayesian deep learning
- Challenges: very high-dimensional posterior + choice of prior
- Instead maximise marginal likelihood of training data
$$\operatorname{argmax}_{\lambda} P(x_1, \dots, x_n | \lambda)$$
- → Maximum likelihood estimation (MLE)
- $P(\lambda | x_1, \dots, x_n) \propto P(x_1, \dots, x_n | \lambda)P(\lambda)$
- When there is a lot of data, the likelihood $P(X|\lambda)$ numerically dominates the prior $P(\lambda)$ so effectively the prior can be ignored (formally: Bernstein von Mises theorem)
- MLE \approx MAP if we have a lot of data

Deep Probabilistic Programming: MLE

Find MLE of λ with stochastic gradient ascent

$$\nabla_{\lambda} \log P(X|\lambda) = \mathbb{E}_{\theta \sim P(\cdot|X,\lambda)} [\nabla_{\lambda} \log P(X, \theta|\lambda)]$$

because

$$\begin{aligned} & \mathbb{E}_{\theta \sim P(\cdot|X,\lambda)} [\nabla_{\lambda} \log P(X, \theta|\lambda)] \\ = & \mathbb{E}_{\theta \sim P(\cdot|X,\lambda)} [\nabla_{\lambda} \log P(X|\lambda) + \nabla_{\lambda} \log P(\theta|X, \lambda)] \\ = & \nabla_{\lambda} \log P(X|\lambda) + \underbrace{\mathbb{E}_{\theta \sim P(\cdot|X,\lambda)} [\nabla_{\lambda} \log P(\theta|X, \lambda)]}_{=0} \end{aligned}$$

How to compute $\mathbb{E}_{\theta \sim P(\cdot|X,\lambda)} [\nabla_{\lambda} \log P(X, \theta|\lambda)]$?

Bayesian inference!

- We do not only want to learn the model parameters
- We also want to perform posterior inference over latent variables
- E.g. what is the digit of an unlabeled image?
- **How to combine model learning and posterior inference?**

Variational guide programs

- If we cannot fully specify the model, then we probably also want to specify the proposals with neural networks η_ϕ .
- E.g. mapping images to their digit.
- Thus, we write a variational proposal distribution as a guide program.
- As in ADVI, we can differentiate through the neural networks and maximise the ELBO to minimise the KL-divergence.

Deep Probabilistic Programming

Scalability: Amortised Inference

Instead of learning N variational distributions separately like in ADVI with mean-field approximation,

$$Q(\theta_i|x_i, \phi) = Q(\theta_i|\phi_i),$$

we use the neural network η_ϕ to predict the variational parameters for each observation x_i ,

$$Q(\theta_i|x_i, \phi) = Q(\theta_i|\eta_\phi(x_i)).$$

E.g. for N images of hand-written digits x_i :

Learning N separate distributions over the true latent digits θ_i of x_i versus learning to predict the digit of each image $\eta_\phi(x_i)$ and then build a distribution around it.

Deep Probabilistic Programming

Combining model learning and posterior inference

- Maximising the ELBO w.r.t to ϕ and λ

$$\begin{aligned}\text{ELBO}(X; \lambda, \phi) &= \mathbb{E}_{\theta \sim Q(\cdot | \phi)} [\log P(\theta, X | \lambda) - \log Q(\theta | \phi)] \\ &= \log P(X | \lambda) - D_{\text{KL}}(Q(\Theta | \phi) \| P(\Theta | X, \lambda))\end{aligned}$$

- Justification: assume we have variational distribution with an "infinity capacity" (it can fit every distribution perfectly), then

$$\min_{\phi} D_{\text{KL}}(Q(\Theta | \phi) \| P(\Theta | X, \lambda)) = 0 \text{ and } \max_{\phi} \text{ELBO}(X; \lambda, \phi) = \log P(X | \lambda)$$

- Thus, maximising the ELBO w.r.t to ϕ and λ is equivalent to maximum likelihood estimation,

$$\max_{\lambda} \max_{\phi} \text{ELBO}(X; \lambda, \phi) = \max_{\lambda} \log P(X | \lambda)$$

Deep Probabilistic Programming

Maximising the ELBO w.r.t to ϕ and λ

$$\max_{\lambda} \max_{\phi} \text{ELBO}(X; \lambda, \phi) = \max_{\lambda} \log P(X|\lambda)$$

- In practice, we **will not have an infinite capacity variational distribution**, and we will typically **not fully solve the inner optimization problem** for ϕ at every gradient step for λ .
- We take gradient steps in both λ and ϕ space simultaneously so that the guide and model play chase, with the guide tracking a moving posterior $\log P(\Theta|X, \lambda)$.
- There will be a difference between maximizing the ELBO and maximizing the marginal likelihood. This difference manifests itself as an extra term in the gradient

$$\nabla_{\lambda} \text{ELBO}(X; \lambda, \phi) = \nabla_{\lambda} \log P(X|\lambda) + \nabla_{\lambda} D_{\text{KL}}(Q(\Theta|\phi) \parallel P(\Theta|X, \lambda))$$

Deep Probabilistic Programming

Maximising the ELBO w.r.t to ϕ and λ

$$\nabla_{\lambda} \text{ELBO}(X; \lambda, \phi) = \nabla_{\lambda} \log P(X|\lambda) + \nabla_{\lambda} D_{\text{KL}}(Q(\Theta|\phi) \parallel P(\Theta|X, \lambda))$$

In this gradient, the second term prevents gradient updates to λ from making changes to the model that strongly increase the KL relative to the variational approximation. This is sometimes argued to be beneficial, in the sense that it **acts as a form of regularization that prevents overfitting** in the generative model, or in the sense that it **stabilizes the optimizer**. However, it can also lead to **approximation errors in the learned generative model**.

Optimizing the ELBO will balance maximizing $\log P(X|\lambda)$ against minimizing $D_{\text{KL}}(Q(\Theta|\phi) \parallel P(\Theta|X, \lambda))$. This can be seen as a bias towards learned $P(\Theta|X, \lambda)$ that are "compatible" with performing variational inference in using the variational family $Q(\Theta|\phi)$.

Maximising the ELBO w.r.t to ϕ and λ - Computing Gradients

As

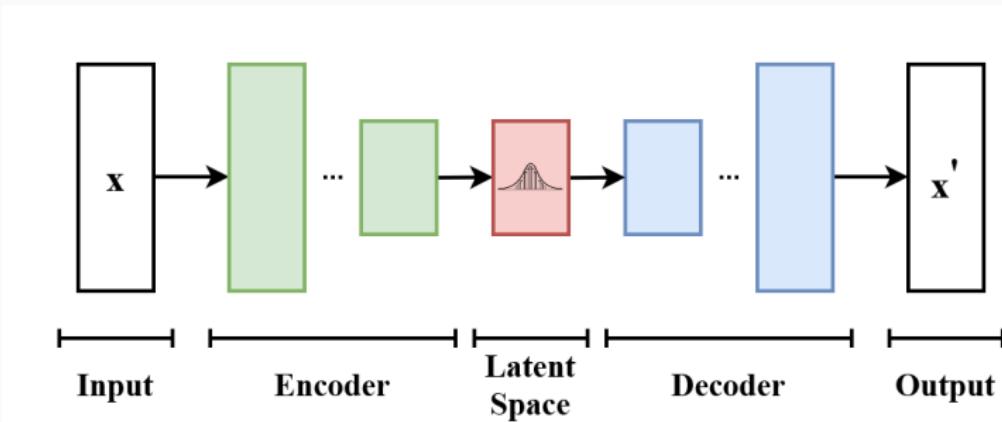
$$\text{ELBO}(X; \lambda, \phi) = \mathbb{E}_{\theta \sim Q(\cdot|\phi)} [\log P(X, \theta|\lambda) - \log Q(\theta|\phi)]$$

is an expectation w.r.t to $Q(\cdot|\phi)$, we can pull ∇_ϕ inside the expectation if we can apply the reparametrisation trick as in ADVI. This allows us to use unbiased lower-variance Monte-Carlo estimates for the gradient.

∇_λ can always be pulled inside the expectation.

Deep Probabilistic Programming - Example

Semi-Supervised Variational Auto-Encoders (SSVAE) in Pyro



Objective: Learn generative distribution of hand-written digits and be able to predict the digit of unlabeled images.

Only a fraction of the images are assumed to be labeled.

Deep Probabilistic Programming - SSVAE

```
1 # observation likelihood p(x|y,z)
2 class Decoder(nn.Module):
3     def __init__(self, input_dim, output_dim, hidden_dims):
4         super().__init__()
5         # lambda is parameters of following layers
6         self.fc1 = nn.Linear(input_dim, hidden_dims[0])
7         self.fc2 = nn.Linear(hidden_dims[0], hidden_dims[1])
8         self.fc3 = nn.Linear(hidden_dims[1], output_dim)
9         # activation
10        self.softplus = nn.Softplus()
11
12    def forward(self, y, z):
13        yz = self.concat.forward(y, z)
14        o1 = self.softplus(self.fc1(yz))
15        o2 = self.softplus(self.fc2(o1))
16        loc_img = torch.sigmoid(self.fc3(o2))
17        return loc_img # probabilities of pixels being white
```

Deep Probabilistic Programming - SSVAE

```
1 def model(self, x, y=None):
2     pyro.module("decoder", self.decoder)
3     with pyro.plate("data", x.shape[0]):
4         # setup hyperparameters for prior p(z)
5         z_loc = torch.zeros(x.shape[0], self.z_dim)
6         z_scale = torch.ones(x.shape[0], self.z_dim)
7         # sample from prior p(z)
8         z = pyro.sample("latent", dist.Normal(z_loc, z_scale).to_event(1))
9         # setup hyperparameters for prior p(y)
10        alpha = torch.full(x.shape[0], 1/self.output_size)
11        # sample from prior p(y)
12        y = pyro.sample("y", dist.OneHotCategorical(alpha), obs=y)
13        # sample from p(x|y,z)
14        loc_img = self.decoder.forward(y, z)
15        # sample image
16        pyro.sample(
17            "obs",
18            dist.Bernoulli(loc_img, validate_args=False).to_event(1),
19            obs=x,
20        )
21    return loc_img
```

Deep Probabilistic Programming - SSVAE

```
1 # diagonal gaussian distribution q(z|x,y)
2 class EncoderZ(nn.Module):
3     def __init__(self, input_dim, output_dim, hidden_dims):
4         super().__init__()
5         self.input_dim = input_dim
6         # phi includes parameters of following layers
7         self.fc1 = nn.Linear(input_dim, hidden_dims[1])
8         self.fc2 = nn.Linear(hidden_dims[1], hidden_dims[0])
9         # two heads for mean and std
10        self.fc31 = nn.Linear(hidden_dims[0], output_dim)
11        self.fc32 = nn.Linear(hidden_dims[0], output_dim)
12        # activation
13        self.softplus = nn.Softplus()
14
15    def forward(self, x, y):
16        xy = self.concat.forward(x, y)
17        o1 = self.softplus(self.fc1(xy))
18        o2 = self.softplus(self.fc2(o1))
19        z_loc = self.fc31(o2)
20        z_scale = torch.exp(self.fc32(o2))
21        return z_loc, z_scale
```

Deep Probabilistic Programming - SSVAE

```
1 # diagonal gaussian distribution q(y|x)
2 class EncoderY(nn.Module):
3     def __init__(self, input_dim, output_dim, hidden_dims):
4         super().__init__()
5         self.input_dim = input_dim
6         # phi includes parameters of following layers
7         self.fc1 = nn.Linear(input_dim, hidden_dims[1])
8         self.fc2 = nn.Linear(hidden_dims[1], hidden_dims[0])
9         self.fc3 = nn.Linear(hidden_dims[0], output_dim)
10        self.softplus = nn.Softplus()
11        self.softmax = nn.Softmax(dim=1)
12
13    def forward(self, x):
14        x = self.softplus(self.fc1(x))
15        x = self.softplus(self.fc2(x))
16        y = self.softmax(self.fc3(x)) # returns class probabilities
17        return y
```

Deep Probabilistic Programming - SSVAE

```
1 # define the guide (variational distribution) q(z|x,y) q(y|x)
2 def guide(self, x, y=None):
3     pyro.module("encoder_z", self.encoder_z)
4     pyro.module("encoder_y", self.encoder_y)
5     with pyro.plate("data", x.shape[0]):
6         if y is None:
7             # use the encoder to get the parameters used to define q(y|x)
8             alpha = self.encoder_y.forward(x)
9             # sample q(y|x)
10            y = pyro.sample("y", dist.OneHotCategorical(alpha))
11
12        # amortised inference
13        # use the encoder to get the parameters used to define q(z|x,y)
14        z_loc, z_scale = self.encoder_z.forward(x, y)
15
16        # sample q(z|x,y)
17        z = pyro.sample("latent", dist.Normal(z_loc, z_scale).to_event(1))
```

Deep Probabilistic Programming - SSVAE

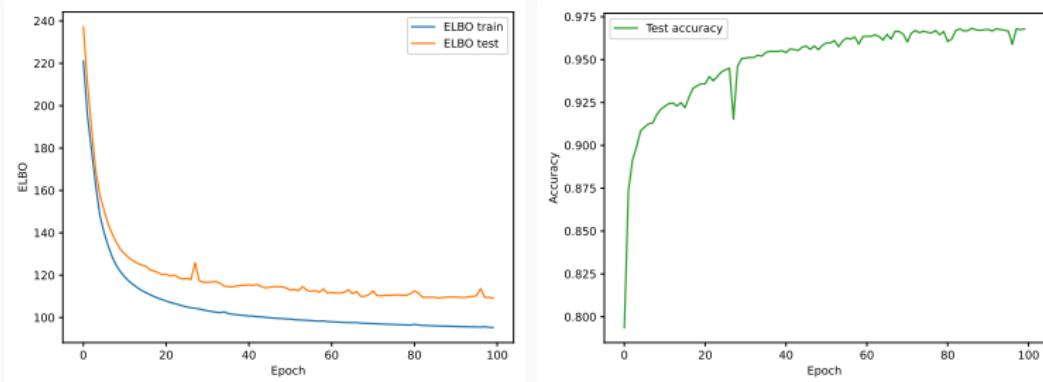
```
1 # auxiliary model
2 def model_classify(self, x, y):
3     pyro.module("encoder_y", self.encoder_y)
4     assert y is not None
5     with pyro.plate("data", x.shape[0]):
6         alpha = self.encoder_y.forward(x)
7         with pyro.poutine.scale(scale=self.aux_loss_multiplier):
8             pyro.sample("y_aux", dist.OneHotCategorical(alpha), obs=y)
9
10 def guide_classify(self, x, y=None):
11     pass
```

Deep Probabilistic Programming - SSVAE

```
1  for epoch in range(1, epochs+1):
2      # perform svi steps on train loader
3      epoch_loss = 0.0
4      # batches are not shuffled
5      for i, (x, y) in enumerate(loaders['train']):
6          x = x.reshape(-1, ssvae.input_size).to(device)
7
8          # alternate between supervised and unsupervised batches
9          if nth_supervised and (i % nth_supervised == 0):
10              y = F.one_hot(y, ssvae.output_size).to(device)
11
12              # perform step on auxiliary model
13              if aux_loss:
14                  epoch_loss += svi_aux.step(x, y)
15              else:
16                  y = None
17
18          epoch_loss += svi.step(x, y)
```

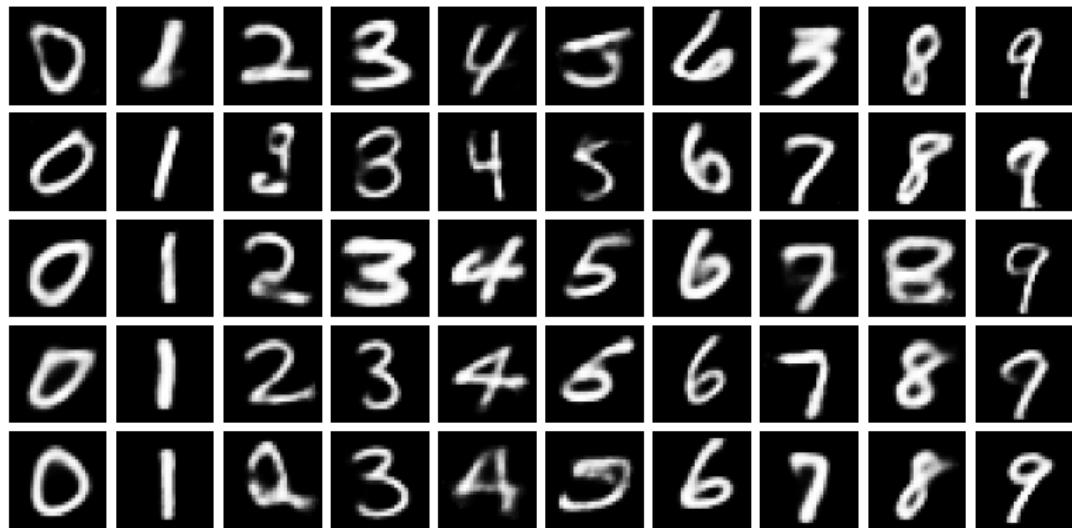
Deep Probabilistic Programming - SSVAE

ELBO + classification accuracy for data set with 10% labeled



Deep Probabilistic Programming - SSVAE

Newly generated digits



Resources

Probabilistic Graphical Models - D Koller, N Friedman - 2009:
Chapter 2.1.4 and 3

Paper: On Bayesian Analysis of Mixtures with an Unknown Number of Components (with discussion)

https://academic.oup.com/jrsssb/article-pdf/59/4/731/49588858/jrsssb_59_4_731.pdf

RJMCMC / Involutive MCMC in Gen Tutorial

<https://www.gen.dev/tutorials/rj/tutorial>

Paper: Transforming Worlds: Automated Involutive MCMC for Open-Universe Probabilistic Models

<https://people.eecs.berkeley.edu/~russell/papers/aabi21-oupm.pdf>

Data-Driven Proposals in Gen Tutorial

<https://www.gen.dev/tutorials/data-driven-proposals/tutorial>

Resources

Paper: Using probabilistic programs as proposals
[*https://arxiv.org/pdf/1801.03612.pdf*](https://arxiv.org/pdf/1801.03612.pdf)

Paper: Pyro: Deep Universal Probabilistic Programming
[*https://arxiv.org/pdf/1810.09538.pdf*](https://arxiv.org/pdf/1810.09538.pdf)

An Introduction to Probabilistic Programming: Chapter 8 Deep
Probabilistic Programming
[*https://arxiv.org/pdf/1809.10756.pdf*](https://arxiv.org/pdf/1809.10756.pdf)

Pyro ELBO Gradients Estimators
[*https://pyro.ai/examples/svi_part_iii.html*](https://pyro.ai/examples/svi_part_iii.html)

Paper: Auto-Encoding Variational Bayes
[*https://arxiv.org/pdf/1312.6114.pdf*](https://arxiv.org/pdf/1312.6114.pdf)

Pyro Semi-Supervised Variational Auto-Encoder
[*https://pyro.ai/examples/ss-vae.html*](https://pyro.ai/examples/ss-vae.html)