Beyond VAEs

How can you improve a VAE?

Improve the inference over $\mathbf{z}$, given $\mathbf{x}$?

- Use $q(\mathbf{z}|\mathbf{x})$ as an importance sampling proposal (IWAE; Burda et al., 2015)
- Use $q(\mathbf{z}|\mathbf{x})$ as an initialization for MCMC (Hoffman, 2015)
- Use a much more powerful $q(\mathbf{z}|\mathbf{x})$, instead of a factorized Gaussian

Improve the prior over $\mathbf{z}$?

- Mixture models as priors
- Autoregressive models as priors

Automatically "disentangle" the dimensions of $\mathbf{z}$ to have axis-aligned features?

- Beta-VAE, Higgins et al.
- Total correlation VAE, Chen et al.
- ...

Alternate objectives:
“Wake-Sleep”

The "other" KL divergence (1/3)

We motivated the ELBO objective for approximate inference by minimizing

$$\min_{\phi} D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \min_{\phi} \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z}.$$

The generative model parameters θ were then dealt with by means of (approximate) maximum likelihood, using the ELBO as a lower bound.

The "other" KL divergence (2/3)

An alternative way to motivate this is to directly minimize this KL divergence with respect to θ:

$$\begin{aligned}\min_{\theta} D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) &= \min_{\theta} \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\ &= \min_{\theta} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log q_\phi(\mathbf{z}|\mathbf{x})\right] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{z}|\mathbf{x})\right] \\ &= \max_{\theta} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{z}|\mathbf{x})\right].\end{aligned}$$

That is, we see that minimizing this KL divergence is equivalent to **maximizing the expected log posterior**, under the approximate posterior.

The "other" KL divergence (3/3)

This suggests an alternative approach for estimating ϕ: we find it by minimizing the **other** KL divergence,

$$\begin{aligned}\min_{\phi} D_{KL}(p_\theta(\mathbf{z}|\mathbf{x})\|q_\phi(\mathbf{z}|\mathbf{x})) &= \min_{\phi} \int p_\theta(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\ &= \max_{\phi} \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})} \left[\log q_\phi(\mathbf{z}|\mathbf{x})\right]\end{aligned}$$

This is an expectation under our current generative model, where we maximize the expected log likelihood of our inference network.

This can be evaluated on real data $\mathbf{x}_1, \ldots, \mathbf{x}_N$, or on synthetic data $\mathbf{x} \sim p_\theta(\mathbf{x})$. For the latter, note that

$$\mathbb{E}_{p_\theta(\mathbf{x})}[D_{KL}(p_\theta(\mathbf{z}|\mathbf{x})\|q_\phi(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{p_\theta(\mathbf{x},\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})}\right].$$

Wake-sleep algorithm

This suggests a two-part algorithm:

- **Wake phase**, update θ: Using the real data, draw samples from the encoder $q_\phi(\mathbf{z}|\mathbf{x})$, and use this to estimate θ by minimizing $D_{KL}(q_\phi \| p_\theta)$
- **Sleep phase**, update ϕ: Using the learned model, **dream** synthetic data by sampling $\mathbf{x}, \mathbf{z}$ from $p_\theta(\mathbf{x}, \mathbf{z})$, and use this to estimate ϕ by minimizing $\mathbb{E}_{p_\theta(\mathbf{x})}[D_{KL}(p_\theta \| q_\phi)]$

[Dayan et al., 1995]

Reweighted wake-sleep

Instead of finding $\min_\phi \mathbb{E}_{p_\theta(\mathbf{x})}[D_{KL}(p_\theta \| q_\phi)]$, let $\tilde{p}(\mathbf{x})$ denote the data distribution and instead minimize

$$\begin{aligned}\mathbb{E}_{\tilde{p}(\mathbf{x})}[D_{KL}(p_\theta(\mathbf{z}|\mathbf{x}) \| q_\phi(\mathbf{z}|\mathbf{x}))] &= \mathbb{E}_{\tilde{p}(\mathbf{x})}[\mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{z}|\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]] \\ &= \mathbb{E}_{\tilde{p}(\mathbf{x})}[\mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\left[-\log q_\phi(\mathbf{z}|\mathbf{x})\right]] + \text{const.}\end{aligned}$$

Bornschein & Bengio (2015)

Reweighted wake-sleep

Instead of finding $\min_\phi \mathbb{E}_{p_\theta(\mathbf{x})}[D_{KL}(p_\theta \| q_\phi)]$, let $\tilde{p}(\mathbf{x})$ denote the data distribution and instead minimize

$$\begin{aligned}\mathbb{E}_{\tilde{p}(\mathbf{x})}[D_{KL}(p_\theta(\mathbf{z}|\mathbf{x}) \| q_\phi(\mathbf{z}|\mathbf{x}))] &= \mathbb{E}_{\tilde{p}(\mathbf{x})}[\mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{z}|\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]] \\ &= \mathbb{E}_{\tilde{p}(\mathbf{x})}[\mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\left[-\log q_\phi(\mathbf{z}|\mathbf{x})\right]] + \text{const.}\end{aligned}$$

This inner expectation over $p_\theta(\mathbf{z}|\mathbf{x})$ is tricky (it's over the posterior!), but can be handled by importance sampling, using $q_\phi(\mathbf{z}|\mathbf{x})$ as a proposal distribution and with importance weights

$$w(\mathbf{z}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}.$$

This is sometimes called the "wake-wake" algorithm, since it uses the data for each update.

Bornschein & Bengio (2015)

Why or why not?

- A big point in favor: discrete latent variables $\mathbf{z}$ pose no problem!
  - Wake/sleep (and re-weighted wake-sleep) sample from one distribution to compute gradient estimates of another. There is no need to apply the reparameterization trick:

$$\nabla_\phi \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})} \left[\log q_\phi(\mathbf{z}|\mathbf{x}) \right] = \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})} \left[\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x}) \right]$$

Why or why not?

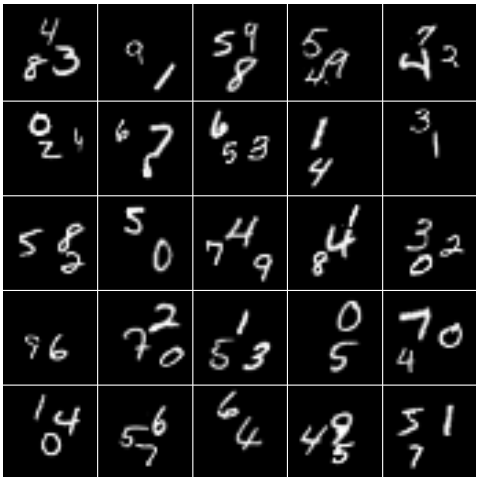
- A big point in favor: discrete latent variables $\mathbf{z}$ pose no problem!
  - Wake/sleep (and re-weighted wake-sleep) sample from one distribution to compute gradient estimates of another. There is no need to apply the reparameterization trick:

$$\nabla_\phi \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})} \left[\log q_\phi(\mathbf{z}|\mathbf{x}) \right] = \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})} \left[\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x}) \right]$$

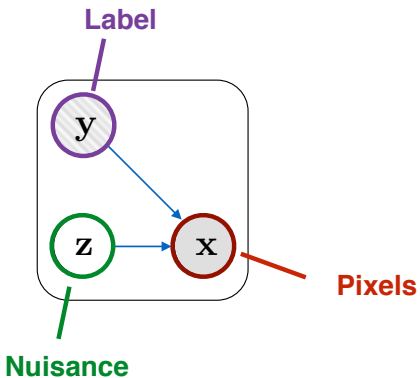
- A potential downside: there is no longer a single objective, and no guarantee of converging to a fixed point
  - In practice, reweighted wake-sleep seems to converge fine
  - Wake-sleep can suffer if the "dream" samples from $p_\theta(\mathbf{x}, \mathbf{z})$ are far from the real data, particularly early in training
- Due to importance sampling, "wake" gradient estimates for ∇_ϕ are biased (so multiple samples are required)

Larger compositional models

From one digit to many digits

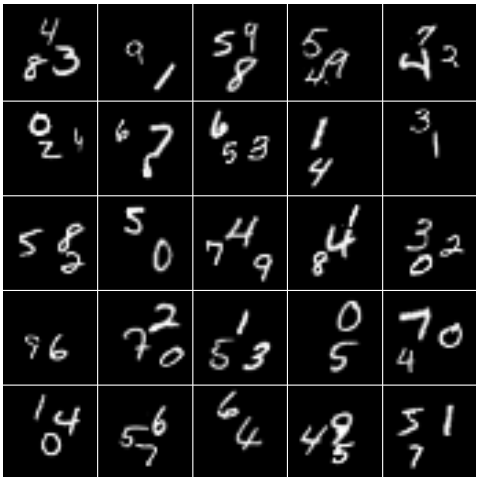


Generative model

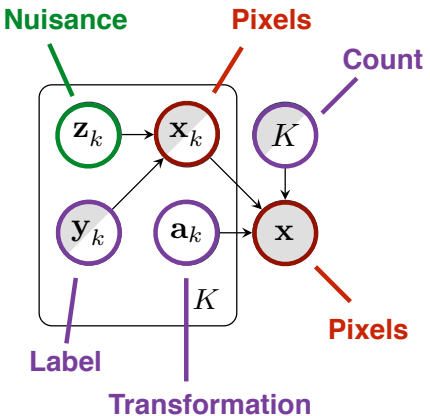


Siddharth et al., 2017

From one digit to many digits



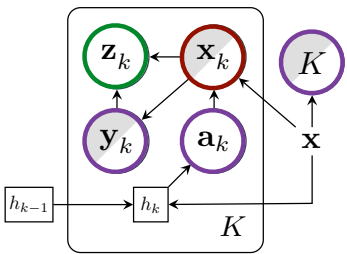
Generative model



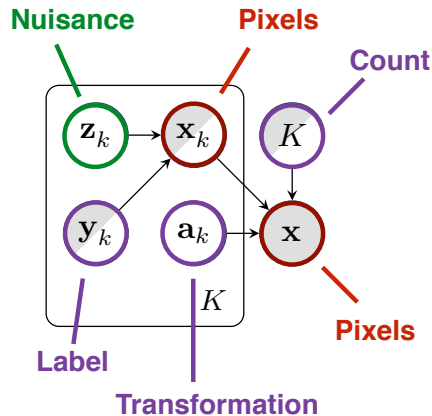
Siddharth et al., 2017

How do we build these models?

Inference model
(recurrent neural network)



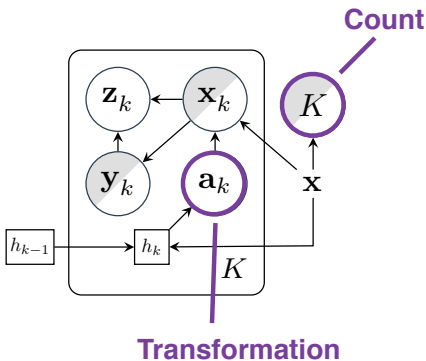
Generative model

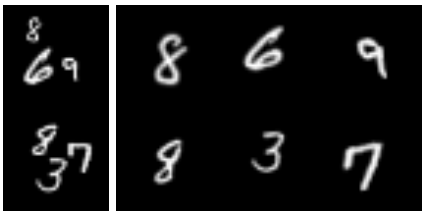


Siddharth et al., 2017

Inference: counting and locating

Inference model
(recurrent neural network)



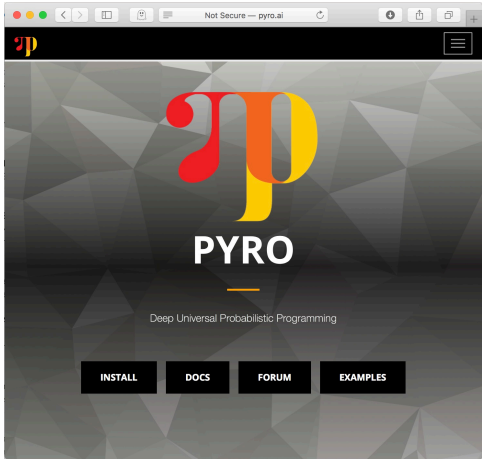


Siddharth et al., 2017

How to make building models less painful

Pyro: deep "probabilistic programming"

- Adds probabilistic modeling on top of PyTorch
- Write two programs: a `model` and a `guide`, defined over the same random variables
- Implements automatic inference by computing and optimizing the ELBO



Other approaches to deep generative models

Generative adversarial networks

Unlike a VAE, a GAN model generates the data **exactly**:

$$\mathbf{z} \sim p(\mathbf{z}), \qquad \mathbf{x} = f_\theta(\mathbf{z}).$$

This is an implicit or "likelihood-free" generative model, and it is no longer straightforward to fit using an ELBO.

Instead, the model can be trained by learning a second network, a **discriminator**, which performs the role of a two-sample test; given an input $\mathbf{x}$, $D(\mathbf{x})$ returns an estimate of the probability that $\mathbf{x}$ is "real" data.

Advantages: excellent realism in generation quality
Disadvantages: mode collapse, training instability

Goodfellow et al. (2014)

Normalizing flows

A normalizing flow also generates the data **exactly**:

$$\mathbf{z} \sim p(\mathbf{z}), \qquad \mathbf{x} = f_\theta(\mathbf{z}).$$

The main difference is that f_θ is restricted to be **bijective** — that is, f_θ is an invertible, one-to-one function.

Note: this means $\mathbf{z} \in \mathbb{R}^D$ and $\mathbf{x} \in \mathbb{R}^D$.

These models can be trained directly by maximum likelihood (no ELBO required), by maximizing

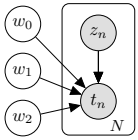
$$\log p_X(\mathbf{x}) = \log p_Z(f_\theta^{-1}(\mathbf{x})) + \log \det \left| \frac{\partial f_\theta^{-1}}{\partial \mathbf{x}} \right|.$$

Advantages: tractable training objective, invertibility
Disadvantages: requires special layer formulations; no dimensionality reduction

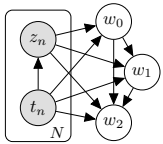
Papamakarios et al. (2019)

Amortized inference without deep generative models

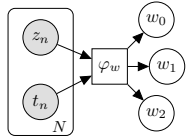
Learning importance sampling proposals



A probabilistic model **generates data**



An inverse model **generates latents**



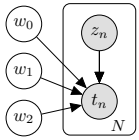
Can we **learn how to sample** from the inverse model?

Learning an importance sampling proposal for a single dataset

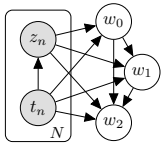
Target density $\pi(\mathbf{x}) = p(\mathbf{x}|\mathbf{y})$, approximating family $q(\mathbf{x}|\lambda)$

Single dataset $\mathbf{y}$: $\underset{\lambda}{\operatorname{argmin}} D_{KL}(\pi||q_\lambda)$ ← fit λ to learn an importance sampling proposal

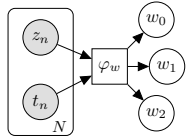
Learning importance sampling proposals



A probabilistic model **generates data**



An inverse model **generates latents**



Can we **learn how to sample** from the inverse model?

Idea: amortize inference by learning a map from data to target

Target density $\pi(\mathbf{x}) = p(\mathbf{x}|\mathbf{y})$, approximating family $q(\mathbf{x}|\lambda)$

Averaging over all possible datasets:

$$\lambda = \varphi(\eta, \mathbf{y})$$

learn a mapping from arbitrary datasets to λ

$$\underset{\eta}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{y})}\left[D_{KL}(\pi||q_{\varphi(\eta,\mathbf{y})})\right]$$

Algorithm: "sleep" step from wake-sleep

Learn to invert the generative model, before seeing data:

Averaging over all possible datasets:

$$\lambda = \varphi(\eta, \mathbf{y})$$

$$\underset{\eta}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{y})}\left[D_{KL}(\pi||q_{\varphi(\eta,\mathbf{y})})\right]$$

expectation over any data we might observe

New objective function, upper-level parameters:

$$\mathcal{J}(\eta) = \int D_{KL}(\pi||q_\lambda)p(\mathbf{y})\mathrm{d}\mathbf{y}$$

$$= \int p(\mathbf{y}) \int p(\mathbf{x}|\mathbf{y}) \log\left[\frac{p(\mathbf{x}|\mathbf{y})}{q(\mathbf{x}|\varphi(\eta,\mathbf{y}))}\right] \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{y}$$

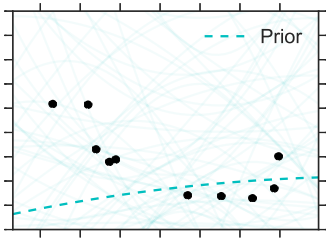
$$= \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\left[-\log q(\mathbf{x}|\varphi(\eta,\mathbf{y}))\right] + const.$$

approximate with samples from the joint distribution

Tractable gradient! Can train entirely offline:

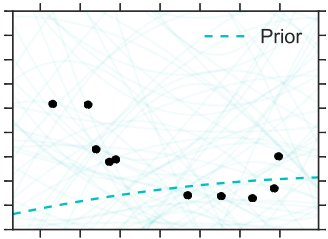
$$\nabla_\eta \mathcal{J}(\eta) = \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\left[-\nabla_\eta \log q(\mathbf{x}|\varphi(\eta,\mathbf{y}))\right]$$

Non-conjugate polynomial regression

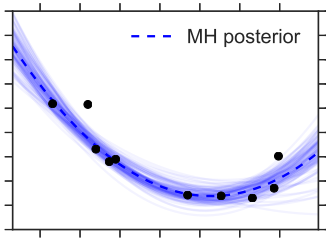


Samples from prior

Non-conjugate polynomial regression

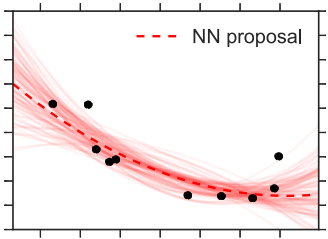


Samples from prior

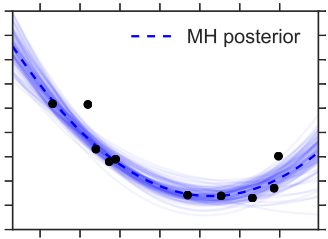


Metropolis-Hastings

Non-conjugate polynomial regression

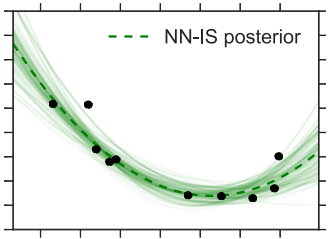


Samples from proposal

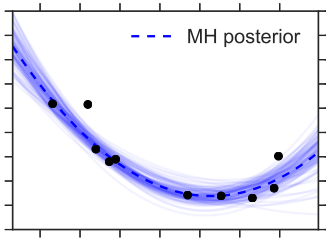


Metropolis-Hastings

Non-conjugate polynomial regression



After importance weighting



Metropolis-Hastings

Non-conjugate polynomial regression