Variational Inference with Normalizing Flows

Luca Wellmeier

November 2022

Table of Contents

Basics

Assume we are given a *target distribution* $\mathbf{x} \sim p_x^*(\mathbf{x})$ on $\mathbb{R}^d$ (hard to sample from / hard to evaluate).

Flow-based models require us to supply the following components:

Basics

Assume we are given a *target distribution* $\mathbf{x} \sim p_x^*(\mathbf{x})$ on $\mathbb{R}^d$ (hard to sample from / hard to evaluate).

Flow-based models require us to supply the following components:

- *base distribution* $\mathbf{z} \sim p_z(\mathbf{z})$ on $\mathbb{R}^d$
  - parameterized $\rightarrow$ learn from data
  - easy to sample from and/or easy to evaluate

Basics

Assume we are given a *target distribution* $\mathbf{x} \sim p_x^*(\mathbf{x})$ on $\mathbb{R}^d$ (hard to sample from / hard to evaluate).

Flow-based models require us to supply the following components:

- *base distribution* $\mathbf{z} \sim p_z(\mathbf{z})$ on $\mathbb{R}^d$
  - parameterized $\to$ learn from data
  - easy to sample from and/or easy to evaluate
- *transformation* $T\colon \mathbb{R}^d \to \mathbb{R}^d$ diffeomorphism
  - potentially parameterized $\to$ learn from data
  - operations of interest: evaluation of T and T^{-1}, computation of Jacobian determinant, derivatives of all three

Basics

Assume we are given a *target distribution* $\mathbf{x} \sim p_x^*(\mathbf{x})$ on $\mathbb{R}^d$ (hard to sample from / hard to evaluate).

Flow-based models require us to supply the following components:

- *base distribution* $\mathbf{z} \sim p_z(\mathbf{z})$ on $\mathbb{R}^d$
  - parameterized $\rightarrow$ learn from data
  - easy to sample from and/or easy to evaluate
- *transformation* $T : \mathbb{R}^d \rightarrow \mathbb{R}^d$ diffeomorphism
  - potentially parameterized $\rightarrow$ learn from data
  - operations of interest: evaluation of T and T^{-1}, computation of Jacobian determinant, derivatives of all three

Define the *model distribution* $p_x(\mathbf{x})$ by

$$
\begin{aligned}
\mathbf{x} &= T(\mathbf{z}) \\
\mathbf{z} &= T^{-1}(\mathbf{x})
\end{aligned}
\qquad \text{i.e.} \qquad
\begin{aligned}
p_x(\mathbf{x}) &= p_z(T^{-1}(\mathbf{x}))\,|\det J_{T^{-1}}(\mathbf{x})|^{-1} \\
p_z(\mathbf{z}) &= p_z(\mathbf{z})\,|\det J_T(\mathbf{z})|^{-1}
\end{aligned}
$$

Expressiveness

We can show in a constructive fashion that a large number of distributions can be captured by flow-based models even with very simple base distribution.

Expressiveness

We can show in a constructive fashion that a large number of distributions can be captured by flow-based models even with very simple base distribution.

Theorem

Require both the base $p_z(\mathbf{z})$ *and the model* $p_x(\mathbf{x})$ *to*

- *be positive on* $\mathbb{R}^d$*, and*
- *have conditional probabilities* $\Pr(u_i' \leq u_i \mid \mathbf{u}_{<i}' = \mathbf{u}_{<i})$ *that are differentiable w.r.t.* $\mathbf{u}_{\leq i}$*.*

Then there exists a transformation $T\colon \mathbf{z} \mapsto \mathbf{x}$ *turning* $p_z(\mathbf{z})$ *into* $p_x(\mathbf{x})$*.*

Proof

We first construct an intermediate transformation step $F : \mathbf{x} \mapsto \mathbf{u} \in (0,1)^d$ into the open unit cube

$$
\begin{aligned}
u_i = F_i(\mathbf{x}) = F_i(\mathbf{x}_{\leq i}) &= \int_{-\infty}^{x_i} p_x(x_i' \mid \mathbf{x}_{<i})\, dx_i' \\
&= \Pr(x_i' \leq x_i \mid \mathbf{x}'_{<i} = \mathbf{x}_{<i}) \in (0,1).
\end{aligned}
$$

where $\mathbf{x}'$ is a random variable distributed according to p_x.

Proof

We first construct an intermediate transformation step $F\colon \mathbf{x} \mapsto \mathbf{u} \in (0,1)^d$ into the open unit cube

$$u_i = F_i(\mathbf{x}) = F_i(\mathbf{x}_{\leq i}) = \int_{-\infty}^{x_i} p_x(x_i' \mid \mathbf{x}_{<i})\, dx_i'$$
$$= \Pr(x_i' \leq x_i \mid \mathbf{x}_{<i}' = \mathbf{x}_{<i}) \in (0,1).$$

where $\mathbf{x}'$ is a random variable distributed according to p_x.

- F is clearly differentiable.

Proof

We first construct an intermediate transformation step $F : \mathbf{x} \mapsto \mathbf{u} \in (0,1)^d$ into the open unit cube

$$u_i = F_i(\mathbf{x}) = F_i(\mathbf{x}_{\leq i}) = \int_{-\infty}^{x_i} p_x(x_i' \mid \mathbf{x}_{<i})\, dx_i'$$
$$= \Pr(x_i' \leq x_i \mid \mathbf{x}_{<i}' = \mathbf{x}_{<i}) \in (0,1).$$

where $\mathbf{x}'$ is a random variable distributed according to p_x.

- F is clearly differentiable.
- $0 < p_x(\mathbf{x}) = \prod_{i=1}^{d} p_x(x_i \mid \mathbf{x}_{<i})$ so that $p_x(x_i \mid \mathbf{x}_{<i}) > 0$.

Proof

We first construct an intermediate transformation step $F : \mathbf{x} \mapsto \mathbf{u} \in (0,1)^d$ into the open unit cube

$$u_i = F_i(\mathbf{x}) = F_i(\mathbf{x}_{\leq i}) = \int_{-\infty}^{x_i} p_x(x_i' \mid \mathbf{x}_{<i})\, dx_i'$$
$$= \Pr(x_i' \leq x_i \mid \mathbf{x}_{<i}' = \mathbf{x}_{<i}) \in (0,1).$$

where $\mathbf{x}'$ is a random variable distributed according to p_x.

- F is clearly differentiable.
- $0 < p_x(\mathbf{x}) = \prod_{i=1}^d p_x(x_i \mid \mathbf{x}_{<i})$ so that $p_x(x_i \mid \mathbf{x}_{<i}) > 0$.
- $\frac{\partial F_i}{\partial x_i} = p_x(x_i \mid \mathbf{x}_{<i}) > 0$ and $\frac{\partial F_i}{\partial x_j} = 0$ for $i < j$.

Proof

We first construct an intermediate transformation step $F : \mathbf{x} \mapsto \mathbf{u} \in (0,1)^d$ into the open unit cube

$$
\begin{aligned}
u_i = F_i(\mathbf{x}) = F_i(\mathbf{x}_{\le i}) &= \int_{-\infty}^{x_i} p_x(x_i' \mid \mathbf{x}_{<i})\, dx_i' \\
&= \Pr(x_i' \le x_i \mid \mathbf{x}_{<i}' = \mathbf{x}_{<i}) \in (0,1).
\end{aligned}
$$

where $\mathbf{x}'$ is a random variable distributed according to p_x.

- F is clearly differentiable.
- $0 < p_x(\mathbf{x}) = \prod_{i=1}^{d} p_x(x_i \mid \mathbf{x}_{<i})$ so that $p_x(x_i \mid \mathbf{x}_{<i}) > 0$.
- $\frac{\partial F_i}{\partial x_i} = p_x(x_i \mid \mathbf{x}_{<i}) > 0$ and $\frac{\partial F_i}{\partial x_j} = 0$ for $i < j$.
- $F_i(\cdot, \mathbf{x}_{<i})$ has positive derivative $\implies$ invertible

Proof

We first construct an intermediate transformation step $F : \mathbf{x} \mapsto \mathbf{u} \in (0,1)^d$ into the open unit cube

$$\begin{aligned} u_i = F_i(\mathbf{x}) = F_i(\mathbf{x}_{\leq i}) &= \int_{-\infty}^{x_i} p_x(x_i' \mid \mathbf{x}_{<i})\, dx_i' \\ &= \Pr(x_i' \leq x_i \mid \mathbf{x}_{<i}' = \mathbf{x}_{<i}) \in (0,1). \end{aligned}$$

where $\mathbf{x}'$ is a random variable distributed according to p_x.

- F is clearly differentiable.
- $0 < p_x(\mathbf{x}) = \prod_{i=1}^{d} p_x(x_i \mid \mathbf{x}_{<i})$ so that $p_x(x_i \mid \mathbf{x}_{<i}) > 0$.
- $\frac{\partial F_i}{\partial x_i} = p_x(x_i \mid \mathbf{x}_{<i}) > 0$ and $\frac{\partial F_i}{\partial x_j} = 0$ for $i < j$.
- $F_i(\cdot, \mathbf{x}_{<i})$ has positive derivative $\implies$ invertible
- u_i depends only on $\mathbf{x}_{\leq i}$ so that F has component-wise inverse $x_i = (F^{-1})_i(\mathbf{u}) = (F_i(\cdot, \mathbf{x}_{<i}))^{-1}(u_i)$

- J_F is triangular $\implies$ $\det J_F(\mathbf{x}) = \prod_{i=1}^{d} \frac{\partial F_i}{\partial x_i} = p_x(\mathbf{x}) > 0$ everwhere $\implies$ F diffeomorphism

- J_F is triangular $\implies$ $\det J_F(\mathbf{x}) = \prod_{i=1}^{d} \frac{\partial F_i}{\partial x_i} = p_x(\mathbf{x}) > 0$ everwhere $\implies$ F diffeomorphism

Moreover,

$$p_u(\mathbf{u}) = p_x(\mathbf{x}) |\det J_F(\mathbf{x})|^{-1} = 1$$

so that F is a transformation that maps $\mathbf{x}$ from any valid model distribution satisfying the conditions into *uniformly distributed* $\mathbf{u} \in (0, 1)^d$.

- J_F is triangular $\implies$ $\det J_F(\mathbf{x}) = \prod_{i=1}^{d} \frac{\partial F_i}{\partial x_i} = p_x(\mathbf{x}) > 0$ everwhere $\implies$ F diffeomorphism

Moreover,

$$p_u(\mathbf{u}) = p_x(\mathbf{x}) |\det J_F(\mathbf{x})|^{-1} = 1$$

so that F is a transformation that maps $\mathbf{x}$ from any valid model distribution satisfying the conditions into *uniformly distributed* $\mathbf{u} \in (0,1)^d$.

Therefore, we can find another such transformation $G \colon \mathbf{z} \mapsto \mathbf{u} \in (0,1)^d$ from any valid base distribution and obtain the desired $T = F^{-1} \circ G \colon \mathbf{z} \mapsto \mathbf{x}$.

Table of Contents

Examples of Transformations

- $h\colon \mathbb{R} \to \mathbb{R}$ diffeomorphism applied element-wise
  - all computations easy but ...
  - ... no intermingling of dimensions

Examples of Transformations

- $h\colon \mathbb{R} \to \mathbb{R}$ diffeomorphism applied element-wise
  - all computations easy but ...
  - ... no intermingling of dimensions
- $A\colon \mathbb{R}^d \to \mathbb{R}^d$ (affine) linear and bijective
  - intermingles dimensions but ...
  - ... determinant and inversion have time complexity $O(d^3)$
  - use triangular, QR, PLU (also easier to enforce invertibility)
  - limited expressiveness: linear flows of exponential base distributions generate exponential distributions

Examples of Transformations

- $h \colon \mathbb{R} \to \mathbb{R}$ diffeomorphism applied element-wise
  - all computations easy but ...
  - ... no intermingling of dimensions
- $A \colon \mathbb{R}^d \to \mathbb{R}^d$ (affine) linear and bijective
  - intermingles dimensions but ...
  - ... determinant and inversion have time complexity $O(d^3)$
  - use triangular, QR, PLU (also easier to enforce invertibility)
  - limited expressiveness: linear flows of exponential base distributions generate exponential distributions
- compose $T = T_K \circ \cdots \circ T_1$
  - steps $\mathbf{z}_0 = \mathbf{z}$, $\mathbf{z}_i = T(\mathbf{z}_{i-1})$, $\mathbf{x} = T_K(\mathbf{z}_{K-1})$
  - inversion and Jacobian determinant are straightforward if the they are for T_i

Autoregressive Flows

As in the proof, focus on transformations with triangular Jacobians:

$$x_i = \tau(z_i; \mathbf{h}_i), \qquad \mathbf{h}_i = c_i(\mathbf{z}_{<i}).$$

The *transformer* $\tau \colon \mathbb{R} \to \mathbb{R}$ is strictly monotonic in z_i.

Autoregressive Flows

As in the proof, focus on transformations with triangular Jacobians:

$$x_i = \tau(z_i; \mathbf{h}_i), \qquad \mathbf{h}_i = c_i(\mathbf{z}_{<i}).$$

The *transformer* $\tau : \mathbb{R} \to \mathbb{R}$ is strictly monotonic in z_i. Common choices include:

- affine linear functions

Autoregressive Flows

As in the proof, focus on transformations with triangular Jacobians:

$$x_i = \tau(z_i; \mathbf{h}_i), \qquad \mathbf{h}_i = c_i(\mathbf{z}_{<i}).$$

The *transformer* $\tau : \mathbb{R} \to \mathbb{R}$ is strictly monotonic in z_i. Common choices include:

- affine linear functions
- multi-layer perceptrons with positive weights and strictly monotonic activations

Autoregressive Flows

As in the proof, focus on transformations with triangular Jacobians:

$$x_i = \tau(z_i; \mathbf{h}_i), \qquad \mathbf{h}_i = c_i(\mathbf{z}_{<i}).$$

The *transformer* $\tau : \mathbb{R} \to \mathbb{R}$ is strictly monotonic in z_i. Common choices include:

- affine linear functions
- multi-layer perceptrons with positive weights and strictly monotonic activations
- monotonic splines that are easily invertible (linear, quadratic, rational, ...)

The *conditioner* does not need to be bijective. Common choices are:

The *conditioner* does not need to be bijective. Common choices are:

- recurrent: sharing parameters across the c_i by using an RNN

The *conditioner* does not need to be bijective. Common choices are:

- recurrent: sharing parameters across the c_i by using an RNN
- masking: zero out the unwanted paths from a single feedforward net

The *conditioner* does not need to be bijective. Common choices are:

- recurrent: sharing parameters across the c_i by using an RNN
- masking: zero out the unwanted paths from a single feedforward net
- coupling: choose a splitting dimension $1 < s < d$; consider $\mathbf{h}_1, \ldots, \mathbf{h}_s$ constants and let $(\mathbf{h}_{s+1}, \ldots, \mathbf{h}_d) = F(\mathbf{h}_{\leq s})$ for some learnable function F

Residual Flows

Consider transformations of the form $\mathbf{x} = \mathbf{z} + g(\mathbf{z})$ (g learnable).
We can enforce invertibility in two ways:

Residual Flows

Consider transformations of the form $\mathbf{x} = \mathbf{z} + g(\mathbf{z})$ (g learnable). We can enforce invertibility in two ways:

- f is invertible if g is contractive by Banach fixed point theorem (which also yields an iteration to find the inverse)

Residual Flows

Consider transformations of the form $\mathbf{x} = \mathbf{z} + g(\mathbf{z})$ (g learnable).
We can enforce invertibility in two ways:

- f is invertible if g is contractive by Banach fixed point theorem (which also yields an iteration to find the inverse)
- by employing the matrix determinant lemma

$$\det(\mathbf{A} + \mathbf{V}\mathbf{W}^T) = \det(\mathbf{I} + \mathbf{W}^T\mathbf{A}^{-1}\mathbf{V}) \det \mathbf{A}$$

  where $\mathbf{A}$ is invertible and $\mathbf{V}$, $\mathbf{W}$ have the same number of rows as $\mathbf{A}$ ... gives rise to Sylvester and radial flow

Sylvester flow: single layer neural net with m hidden units and element-wise activation σ given by

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \mathbf{V}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b}),$$

where $\mathbf{W}, \mathbf{V}$ are $d \times m$ and $\mathbf{b}$ is m-dimensional

Sylvester flow: single layer neural net with m hidden units and element-wise activation σ given by

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \mathbf{V}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b}),$$

where $\mathbf{W}, \mathbf{V}$ are $d \times m$ and $\mathbf{b}$ is m-dimensional

- $J_T(\mathbf{z}) = \mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^T\mathbf{V}$ where $\mathbf{S}(\mathbf{z}) = \frac{d}{d\mathbf{z}}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b})$ is diagonal

Sylvester flow: single layer neural net with m hidden units and element-wise activation σ given by

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \mathbf{V}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b}),$$

where $\mathbf{W}, \mathbf{V}$ are $d \times m$ and $\mathbf{b}$ is m-dimensional

- $J_T(\mathbf{z}) = \mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^T\mathbf{V}$ where $\mathbf{S}(\mathbf{z}) = \frac{d}{d\mathbf{z}}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b})$ is diagonal
- by matrix determinant lemma $\det J_T(\mathbf{z}) = \det(\mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^T\mathbf{V})$ has time complexity $O(m^3 + dm^2) \rightarrow$ linear in d!

Sylvester flow: single layer neural net with m hidden units and element-wise activation σ given by

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \mathbf{V}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b}),$$

where $\mathbf{W}, \mathbf{V}$ are $d \times m$ and $\mathbf{b}$ is m-dimensional

- $J_T(\mathbf{z}) = \mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^T\mathbf{V}$ where $\mathbf{S}(\mathbf{z}) = \frac{d}{d\mathbf{z}}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b})$ is diagonal
- by matrix determinant lemma $\det J_T(\mathbf{z}) = \det(\mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^T\mathbf{V})$ has time complexity $O(m^3 + dm^2) \to$ linear in d!
- further improvement: $\mathbf{V} = \mathbf{QU}$ and $\mathbf{W} = \mathbf{QL}$ with $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ and $\mathbf{L}, \mathbf{U}$ are lower and upper $m \times m$ triangular matrices; then

$$\det J_T(\mathbf{z}) = \det(\mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{L}^T\mathbf{U}) = \prod_{i=1}^{d}(1 + S_{ii}(\mathbf{z})L_{ii}U_{ii}).$$

Sylvester flow: single layer neural net with m hidden units and element-wise activation σ given by

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \mathbf{V}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b}),$$

where $\mathbf{W}, \mathbf{V}$ are $d \times m$ and $\mathbf{b}$ is m-dimensional

- $J_T(\mathbf{z}) = \mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^T\mathbf{V}$ where $\mathbf{S}(\mathbf{z}) = \frac{d}{d\mathbf{z}}\sigma(\mathbf{W}^T\mathbf{z} + \mathbf{b})$ is diagonal
- by matrix determinant lemma $\det J_T(\mathbf{z}) = \det(\mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^T\mathbf{V})$ has time complexity $O(m^3 + dm^2) \to$ linear in d!
- further improvement: $\mathbf{V} = \mathbf{Q}\mathbf{U}$ and $\mathbf{W} = \mathbf{Q}\mathbf{L}$ with $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ and $\mathbf{L}, \mathbf{U}$ are lower and upper $m \times m$ triangular matrices; then

$$\det J_T(\mathbf{z}) = \det(\mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{L}^T\mathbf{U}) = \prod_{i=1}^{d}(1 + S_{ii}(\mathbf{z})L_{ii}U_{ii}).$$

- a sufficient condition for invertibility is $L_{ii}U_{ii} > -\frac{1}{\sup_x \sigma'(x)}$ assuming that σ' is positive and bounded from above

Radial flow: let $\mathbf{z}_0 \in \mathbb{R}^d$, $\alpha > 0$ and $\beta \in \mathbb{R}$ be parameters and $r(\mathbf{z}) = \|\mathbf{z} - \mathbf{z}_0\|$. Define

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \frac{\beta}{\alpha + r(\mathbf{z})}(\mathbf{z} - \mathbf{z}_0).$$

Radial flow: let $\mathbf{z}_0 \in \mathbb{R}^d$, $\alpha > 0$ and $\beta \in \mathbb{R}$ be parameters and $r(\mathbf{z}) = \|\mathbf{z} - \mathbf{z}_0\|$. Define

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \frac{\beta}{\alpha + r(\mathbf{z})}(\mathbf{z} - \mathbf{z}_0).$$

- Jacobian

$$J_T(\mathbf{z}) = \left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)\mathbf{I} - \frac{\beta}{r(\mathbf{z})(\alpha + r(\mathbf{z}))^2}(\mathbf{z} - \mathbf{z}_0)(\mathbf{z} - \mathbf{z}_0)^T.$$

Radial flow: let $\mathbf{z}_0 \in \mathbb{R}^d$, $\alpha > 0$ and $\beta \in \mathbb{R}$ be parameters and $r(\mathbf{z}) = \|\mathbf{z} - \mathbf{z}_0\|$. Define

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \frac{\beta}{\alpha + r(\mathbf{z})}(\mathbf{z} - \mathbf{z}_0).$$

- Jacobian

$$J_T(\mathbf{z}) = \left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)\mathbf{I} - \frac{\beta}{r(\mathbf{z})(\alpha + r(\mathbf{z}))^2}(\mathbf{z} - \mathbf{z}_0)(\mathbf{z} - \mathbf{z}_0)^T.$$

- by matrix determinant lemma

$$\det J_T(\mathbf{z}) = \left(1 + \frac{\alpha\beta}{(\alpha + r(\mathbf{z}))^2}\right)\left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)^{d-1}$$

Radial flow: let $\mathbf{z}_0 \in \mathbb{R}^d$, $\alpha > 0$ and $\beta \in \mathbb{R}$ be parameters and $r(\mathbf{z}) = \|\mathbf{z} - \mathbf{z}_0\|$. Define

$$\mathbf{x} = T(\mathbf{z}) = \mathbf{z} + \frac{\beta}{\alpha + r(\mathbf{z})}(\mathbf{z} - \mathbf{z}_0).$$

- Jacobian

$$J_T(\mathbf{z}) = \left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)\mathbf{I} - \frac{\beta}{r(\mathbf{z})(\alpha + r(\mathbf{z}))^2}(\mathbf{z} - \mathbf{z}_0)(\mathbf{z} - \mathbf{z}_0)^T.$$

- by matrix determinant lemma

$$\det J_T(\mathbf{z}) = \left(1 + \frac{\alpha\beta}{(\alpha + r(\mathbf{z}))^2}\right)\left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)^{d-1}$$

- sufficient condition for invertibility: $\beta > -\alpha$

Table of Contents

Training

The *forward KL divergence* of a distribution q from a reference distribution q_0 is

$$D_{\mathrm{KL}}(q_0 \parallel q) = \mathbb{E}_{x \sim q_0(x)}\left[\frac{\log q_0(x)}{\log q(x)}\right] = -\mathbb{E}_{x \sim q_0(x)}[\log q(x)] + C.$$

Training

The *forward KL divergence* of a distribution q from a reference distribution q_0 is

$$D_{\mathrm{KL}}(q_0 \,\|\, q) = \mathbb{E}_{x \sim q_0(x)}\left[\frac{\log q_0(x)}{\log q(x)}\right] = -\mathbb{E}_{x \sim q_0(x)}[\log q(x)] + C.$$

The *reverse KL divergence* of a distribution q from a reference distribution q_0 is

$$\begin{aligned} D_{\mathrm{KL}}(q \,\|\, q_0) &= \mathbb{E}_{x \sim q(x)}\left[\frac{\log q(x)}{\log q_0(x)}\right] \\ &= \mathbb{H}[q] - \mathbb{E}_{x \sim q(x)}[\log q_0(x)]. \end{aligned}$$

Loss w.r.t. forward KL divergence (dropping parameters from notation):

$$\begin{aligned}\mathcal{L}_{\text{forw}} &= D_{\text{KL}}(p_x^*(\mathbf{x}) \,\|\, p_x(\mathbf{x})) \\ &= -\mathbb{E}_{\mathbf{x}\sim p_x^*}[\log p_z(T^{-1}(\mathbf{x})) + \log|\det J_{T^{-1}}(\mathbf{x})|] + C\end{aligned}$$

Loss w.r.t. forward KL divergence (dropping parameters from notation):

$$\begin{aligned}\mathcal{L}_{\text{forw}} &= D_{\text{KL}}(p_x^*(\mathbf{x}) \parallel p_x(\mathbf{x})) \\ &= -\mathbb{E}_{\mathbf{x}\sim p_x^*}[\log p_z(T^{-1}(\mathbf{x})) + \log|\det J_{T^{-1}}(\mathbf{x})|] + C\end{aligned}$$

Minimizing it ...

- ... requires: sampling from p_x^*, evaluation of p_z, T^{-1} and its Jacobian determinant

Loss w.r.t. forward KL divergence (dropping parameters from notation):

$$\begin{aligned}\mathcal{L}_{\text{forw}} &= D_{\text{KL}}(p_x^*(\mathbf{x}) \,\|\, p_x(\mathbf{x})) \\ &= -\mathbb{E}_{\mathbf{x}\sim p_x^*}[\log p_z(T^{-1}(\mathbf{x})) + \log|\det J_{T^{-1}}(\mathbf{x})|] + C\end{aligned}$$

Minimizing it ...

- ... requires: sampling from p_x^*, evaluation of p_z, T^{-1} and its Jacobian determinant
  - when optimizing with gradients all three functions must be differentiated

Loss w.r.t. forward KL divergence (dropping parameters from notation):

$$\begin{aligned}\mathcal{L}_{\mathrm{forw}} &= D_{\mathrm{KL}}(p_x^*(\mathbf{x}) \,\|\, p_x(\mathbf{x})) \\ &= -\mathbb{E}_{\mathbf{x}\sim p_x^*}[\log p_z(T^{-1}(\mathbf{x})) + \log|\det J_{T^{-1}}(\mathbf{x})|] + C\end{aligned}$$

Minimizing it ...

- ... requires: sampling from p_x^*, evaluation of p_z, T^{-1} and its Jacobian determinant
  - when optimizing with gradients all three functions must be differentiated
- ... does not require evaluating p_x^*

Loss w.r.t. forward KL divergence (dropping parameters from notation):

$$\begin{aligned}\mathcal{L}_{\text{forw}} &= D_{\text{KL}}(p_x^*(\mathbf{x}) \,\|\, p_x(\mathbf{x})) \\ &= -\mathbb{E}_{\mathbf{x}\sim p_x^*}[\log p_z(T^{-1}(\mathbf{x})) + \log|\det J_{T^{-1}}(\mathbf{x})|] + C\end{aligned}$$

Minimizing it ...

- ... requires: sampling from p_x^*, evaluation of p_z, T^{-1} and its Jacobian determinant
  - when optimizing with gradients all three functions must be differentiated
- ... does not require evaluating p_x^*

$\implies$ equivalent to maximum likelihood estimation when using Monte Carlo on a set of examples

Loss w.r.t. reverse KL divergence (dropping parameters):

$$\begin{aligned}
\mathcal{L}_{\mathrm{rev}} &= D_{\mathrm{KL}}(p_x(\mathbf{x}) \,\|\, p_x^*(\mathbf{x})) \\
&= \mathbb{E}_{\mathbf{x}\sim p_x}[\log p_x(\mathbf{x}) - \log p_x^*(\mathbf{x})] \\
&= \mathbb{E}_{\mathbf{z}\sim p_z}[\log p_z(\mathbf{z}) - \log|\det J_T(\mathbf{z})| - \log p_x^*(T(\mathbf{z}))]
\end{aligned}$$

Loss w.r.t. reverse KL divergence (dropping parameters):

$$\begin{aligned}\mathcal{L}_{\mathrm{rev}} &= D_{\mathrm{KL}}(p_x(\mathbf{x}) \,\|\, p_x^*(\mathbf{x})) \\ &= \mathbb{E}_{\mathbf{x}\sim p_x}[\log p_x(\mathbf{x}) - \log p_x^*(\mathbf{x})] \\ &= \mathbb{E}_{\mathbf{z}\sim p_z}[\log p_z(\mathbf{z}) - \log |\det J_T(\mathbf{z})| - \log p_x^*(T(\mathbf{z}))]\end{aligned}$$

Minimizing this loss ...

- ... requires: sampling from p_z, evaluating p_x^*, T and the Jacobian determinant of T

Loss w.r.t. reverse KL divergence (dropping parameters):

$$\begin{aligned}\mathcal{L}_{\mathrm{rev}} &= D_{\mathrm{KL}}(p_x(\mathbf{x}) \,\|\, p_x^*(\mathbf{x})) \\ &= \mathbb{E}_{\mathbf{x}\sim p_x}[\log p_x(\mathbf{x}) - \log p_x^*(\mathbf{x})] \\ &= \mathbb{E}_{\mathbf{z}\sim p_z}[\log p_z(\mathbf{z}) - \log|\det J_T(\mathbf{z})| - \log p_x^*(T(\mathbf{z}))]\end{aligned}$$

Minimizing this loss ...

- ... requires: sampling from p_z, evaluating p_x^*, T and the Jacobian determinant of T
- ... does not require sampling from p_x^*

Loss w.r.t. reverse KL divergence (dropping parameters):

$$\begin{aligned}\mathcal{L}_{\mathrm{rev}} &= D_{\mathrm{KL}}(p_x(\mathbf{x}) \,\|\, p_x^*(\mathbf{x})) \\ &= \mathbb{E}_{\mathbf{x}\sim p_x}[\log p_x(\mathbf{x}) - \log p_x^*(\mathbf{x})] \\ &= \mathbb{E}_{\mathbf{z}\sim p_z}[\log p_z(\mathbf{z}) - \log|\det J_T(\mathbf{z})| - \log p_x^*(T(\mathbf{z}))]\end{aligned}$$

Minimizing this loss ...

- ... requires: sampling from p_z, evaluating p_x^*, T and the Jacobian determinant of T
- ... does not require sampling from p_x^*
- ... requires knowledge of p_x^* only up to a normalizing factor

Loss w.r.t. reverse KL divergence (dropping parameters):

$$\begin{aligned}
\mathcal{L}_{\text{rev}} &= D_{\text{KL}}(p_x(\mathbf{x}) \,\|\, p_x^*(\mathbf{x})) \\
&= \mathbb{E}_{\mathbf{x}\sim p_x}[\log p_x(\mathbf{x}) - \log p_x^*(\mathbf{x})] \\
&= \mathbb{E}_{\mathbf{z}\sim p_z}[\log p_z(\mathbf{z}) - \log|\det J_T(\mathbf{z})| - \log p_x^*(T(\mathbf{z}))]
\end{aligned}$$

Minimizing this loss ...

- ... requires: sampling from p_z, evaluating p_x^*, T and the Jacobian determinant of T
- ... does not require sampling from p_x^*
- ... requires knowledge of p_x^* only up to a normalizing factor

$\implies$ useful for variational inference but p_x^* is problematic in practice... ELBO

Table of Contents

Variational Inference

Assume we have latent variables $\mathbf{u}$ and observations $\mathbf{x}$ with distribution $p^*(\mathbf{x}, \mathbf{u})$. We want the posterior $p^*(\mathbf{u} \mid \mathbf{x})$:

$$p^*(\mathbf{u} \mid \mathbf{x}) = \frac{p^*(\mathbf{x}, \mathbf{u})}{p^*(\mathbf{x})} = \frac{p^*(\mathbf{x}, \mathbf{u})}{\int p^*(\mathbf{x}, \mathbf{u})\, d\mathbf{u}}.$$

Variational Inference

Assume we have latent variables $\mathbf{u}$ and observations $\mathbf{x}$ with distribution $p^*(\mathbf{x}, \mathbf{u})$. We want the posterior $p^*(\mathbf{u} \mid \mathbf{x})$:

$$p^*(\mathbf{u} \mid \mathbf{x}) = \frac{p^*(\mathbf{x}, \mathbf{u})}{p^*(\mathbf{x})} = \frac{p^*(\mathbf{x}, \mathbf{u})}{\int p^*(\mathbf{x}, \mathbf{u})\, d\mathbf{u}}.$$

- Marginal $p^*(\mathbf{x})$ in the denominator is usually not computable!
  - not needed necessarily when optimizing for ELBO

Variational Inference

Assume we have latent variables $\mathbf{u}$ and observations $\mathbf{x}$ with distribution $p^*(\mathbf{x}, \mathbf{u})$. We want the posterior $p^*(\mathbf{u} \mid \mathbf{x})$:

$$p^*(\mathbf{u} \mid \mathbf{x}) = \frac{p^*(\mathbf{x}, \mathbf{u})}{p^*(\mathbf{x})} = \frac{p^*(\mathbf{x}, \mathbf{u})}{\int p^*(\mathbf{x}, \mathbf{u})\, d\mathbf{u}}.$$

- Marginal $p^*(\mathbf{x})$ in the denominator is usually not computable!
  - not needed necessarily when optimizing for ELBO
- optimize a parametric distribution $p(\mathbf{u}) \approx p^*(\mathbf{u} \mid \mathbf{x})$ instead

ELBO

The KL divergence is always non-negative.

$$
\begin{aligned}
0 &\leq D_{\mathrm{KL}}(q(\mathbf{u}) \parallel p^*(\mathbf{u} \mid \mathbf{x})) \\
&= \mathbb{E}_{q(\mathbf{u})}[\log q(\mathbf{u}) - (\log p^*(\mathbf{x} \mid \mathbf{u}) + \log p^*(\mathbf{x}) - \log p^*(\mathbf{u}))]
\end{aligned}
$$

ELBO

The KL divergence is always non-negative.

$$\begin{aligned} 0 &\leq D_{\mathrm{KL}}(q(\mathbf{u}) \,\|\, p^*(\mathbf{u} \mid \mathbf{x})) \\ &= \mathbb{E}_{q(\mathbf{u})}[\log q(\mathbf{u}) - (\log p^*(\mathbf{x} \mid \mathbf{u}) + \log p^*(\mathbf{x}) - \log p^*(\mathbf{u}))] \end{aligned}$$

Rearranging,

$$\log p^*(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q(\mathbf{u})}[\log p^*(\mathbf{x} \mid \mathbf{u}) - \log p^*(\mathbf{u}) - \log q(\mathbf{u})]}_{\text{ELBO}}$$

ELBO

The KL divergence is always non-negative.

$$\begin{aligned} 0 &\le D_{\mathrm{KL}}(q(\mathbf{u}) \,\|\, p^*(\mathbf{u} \mid \mathbf{x})) \\ &= \mathbb{E}_{q(\mathbf{u})}[\log q(\mathbf{u}) - (\log p^*(\mathbf{x} \mid \mathbf{u}) + \log p^*(\mathbf{x}) - \log p^*(\mathbf{u}))] \end{aligned}$$

Rearranging,

$$\log p^*(\mathbf{x}) \ge \underbrace{\mathbb{E}_{q(\mathbf{u})}[\log p^*(\mathbf{x} \mid \mathbf{u}) - \log p^*(\mathbf{u}) - \log q(\mathbf{u})]}_{\text{ELBO}}$$

Instead of minimizing the KL divergence we can instead maximize ELBO (evidence lower bound) which doesn't require knowledge about $p^*(\mathbf{x})$.

Variational Autoencoders

A generative model to sample from a complicated solution.
Assume we have latent variables u and observations x that we model jointly as $p^*_\psi(\mathbf{x}, \mathbf{u})$ where the parameters are to be learned.

Variational Autoencoders

A generative model to sample from a complicated solution. Assume we have latent variables u and observations x that we model jointly as $p_\psi^*(\mathbf{x}, \mathbf{u})$ where the parameters are to be learned.

- The *decoder* $p_\psi^*(\mathbf{x} \mid \mathbf{u})$ is assumed to be computable.

Variational Autoencoders

A generative model to sample from a complicated solution. Assume we have latent variables u and observations x that we model jointly as $p^*_\psi(\mathbf{x}, \mathbf{u})$ where the parameters are to be learned.

- The *decoder* $p^*_\psi(\mathbf{x} \mid \mathbf{u})$ is assumed to be computable.
- Approximate the *encoder* $p^*(\mathbf{u} \mid \mathbf{x}) \approx q_\phi(\mathbf{u} \mid \mathbf{x})$ with a computable $q_\phi(\mathbf{u} \mid \mathbf{x})$.

Variational Autoencoders

A generative model to sample from a complicated solution. Assume we have latent variables u and observations x that we model jointly as $p^*_\psi(\mathbf{x}, \mathbf{u})$ where the parameters are to be learned.

- The *decoder* $p^*_\psi(\mathbf{x} \mid \mathbf{u})$ is assumed to be computable.
- Approximate the *encoder* $p^*(\mathbf{u} \mid \mathbf{x}) \approx q_\phi(\mathbf{u} \mid \mathbf{x})$ with a computable $q_\phi(\mathbf{u} \mid \mathbf{x})$.
- Instead of learning the latent representation of a sample directly we instead learn the parameters of its distribution.

Variational Autoencoders

A generative model to sample from a complicated solution. Assume we have latent variables u and observations x that we model jointly as $p^*_\psi(\mathbf{x}, \mathbf{u})$ where the parameters are to be learned.

- The *decoder* $p^*_\psi(\mathbf{x} \mid \mathbf{u})$ is assumed to be computable.
- Approximate the *encoder* $p^*(\mathbf{u} \mid \mathbf{x}) \approx q_\phi(\mathbf{u} \mid \mathbf{x})$ with a computable $q_\phi(\mathbf{u} \mid \mathbf{x})$.
- Instead of learning the latent representation of a sample directly we instead learn the parameters of its distribution.
  - We assume $\phi = (\phi_1, \phi_2)$, $\mathbf{u} = T_{\phi_2}\mathbf{z}$ and

$$q_\phi(\mathbf{u}) = p_{\phi_1}(\mathbf{z}) |\det J_{T_{\phi_2}}(\mathbf{z})|$$

Variational Autoencoders

A generative model to sample from a complicated solution. Assume we have latent variables u and observations x that we model jointly as $p_\psi^*(\mathbf{x}, \mathbf{u})$ where the parameters are to be learned.

- The *decoder* $p_\psi^*(\mathbf{x} \mid \mathbf{u})$ is assumed to be computable.
- Approximate the *encoder* $p^*(\mathbf{u} \mid \mathbf{x}) \approx q_\phi(\mathbf{u} \mid \mathbf{x})$ with a computable $q_\phi(\mathbf{u} \mid \mathbf{x})$.
- Instead of learning the latent representation of a sample directly we instead learn the parameters of its distribution.
  - We assume $\phi = (\phi_1, \phi_2)$, $\mathbf{u} = T_{\phi_2}\mathbf{z}$ and

$$q_\phi(\mathbf{u}) = p_{\phi_1}(\mathbf{z}) |\det J_{T_{\phi_2}}(\mathbf{z})|$$

  - If $\phi_1 = (\mu, \sigma^2)$, $p_{\phi_1}(\mathbf{z}) = N(\mu, \sigma^2)$ and $T_{\phi_2} = \text{id}$ we recover the standard variational autoencoder

- Usually both decoder $p^*_\psi(\mathbf{x} \mid \mathbf{u})$ and encoder $q_\phi(\mathbf{u} \mid \mathbf{x})$ are symmetrically looking neural nets

- Usually both decoder $p^*_\psi(\mathbf{x} \mid \mathbf{u})$ and encoder $q_\phi(\mathbf{u} \mid \mathbf{x})$ are symmetrically looking neural nets
- Train by minimizing reverse Kullback-Leibler (i.e. maximizing ELBO instead)

- Usually both decoder $p^*_\psi(\mathbf{x} \mid \mathbf{u})$ and encoder $q_\phi(\mathbf{u} \mid \mathbf{x})$ are symmetrically looking neural nets
- Train by minimizing reverse Kullback-Leibler (i.e. maximizing ELBO instead)

Hypotheses and questions for experiments

- Learning potentially many additional parameters (coming for instance from compositions) should make training quite a bit harder.
- Giving the model a more expressive hypothesis space could (by using a more complicated prior) could result in more interesting generations