Tractable Approaches Towards Approximate Inference in Deep Learning

Lecture 2/2: Variational Autoencoders and Normalizing Flows

4DV661, Sebastian Hönel

Approximate Inference

Refers to any approach that simplifies complicated, hard-to-solve or ***intractable*** probabilistic computations in complex models like neural networks or *deep generative models*. Examples are:

- Calculating *posterior* distributions (previous lecture)
- **Variational inference** (previous lecture)
- Markov Chain **Monte Carlo** (MCMC) (previous lecture)
- **Density estimation** (**this** lecture)
- **(Conditional) Sample generation** (**this** lecture)
- …

Contents – Lecture 2

1. Notation (Recap)
2. Variational Inference (Recap)
3. Autoencoders

VAEs (Part 1):

1. VAEs formally
2. VAEs computationally
3. VAEs practically
4. Sampling from VAEs
    - Unconditionally and conditionally

Normalizing Flows (Part 2):

1. Motivation and definition
2. Base mechanics
3. Determinant & transformation mechanics
4. Optimization
    - Using the forward KL-divergence
5. Change of Variables, „Gaussianization“
    - Higher dimensions
6. Sampling
7. Components
    - Bijectors, Coupling laws & -blocks, Conditioners

VAEs backed by NFs (Part 3):

1. Normalizing Flows as variational distribution
2. Implementation details

Notation (Recap)

$$P(\theta|Y) = \frac{P(Y|\theta) \cdot P(\theta)}{P(Y)}$$

- **Bayes‘ theorem** is a tool for relating two probabilities with one another (conditional, joint).
  - For example, $P(Y) = \int_\theta \ P(Y, \theta) = \int_\theta \ P(Y|\theta) \cdot P(\theta)\, d\theta$. The first is the **Sum**-rule, the second the **Product** rule.
  - Also recall the symmetry property: $\mathrm{P}(X, Y) = P(Y, X)$. We can use all these to derive Bayes' theorem!
- **Expected Values** are the result of integrating a function against a probability density (a weighted average).
  - The example above can be expressed as an **expectation**: $\mathrm{P(Y)} = \int_\theta \ P(Y|\theta) \cdot P(\theta) = \mathbb{E}_{\theta \sim P(\theta)}[P(Y|\theta)]$.
  - Here the KL-divergence between Q, P is: $D_{\mathrm{KL}}(Q \,||\, P) = \int_{-\infty}^{\infty} q(x) \cdot \log\left(\frac{q(x)}{p(x)}\right)\, dx = \mathbb{E}_{Q \sim q}[\log(q(x)) - \log(p(x))]$.
- **Jensen's Inequality** for random variables and a concave $f(\cdot)$ is $f(\mathbb{E}[Y]) \geq \mathbb{E}[f(Y)]$.
- The Evidence Lower BOund (**ELBO**) is: $\mathbb{E}_{q_\phi(\theta)}\left[\underbrace{\log(P(Y, \theta))}_{\text{Exp. data likelihood}} - \underbrace{\log\big(q_\phi(\theta)\big)}_{\text{Entropy term}}\right]$.
- We used a **mean-field** approximation as **variational posterior** $q_\phi(\theta)$ so far.

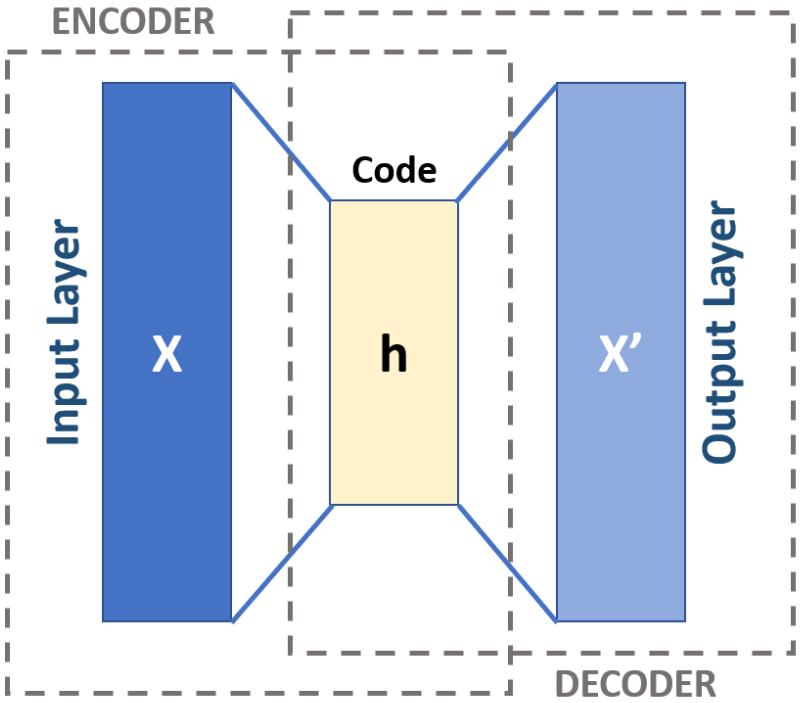
Variational Inference (Recap)

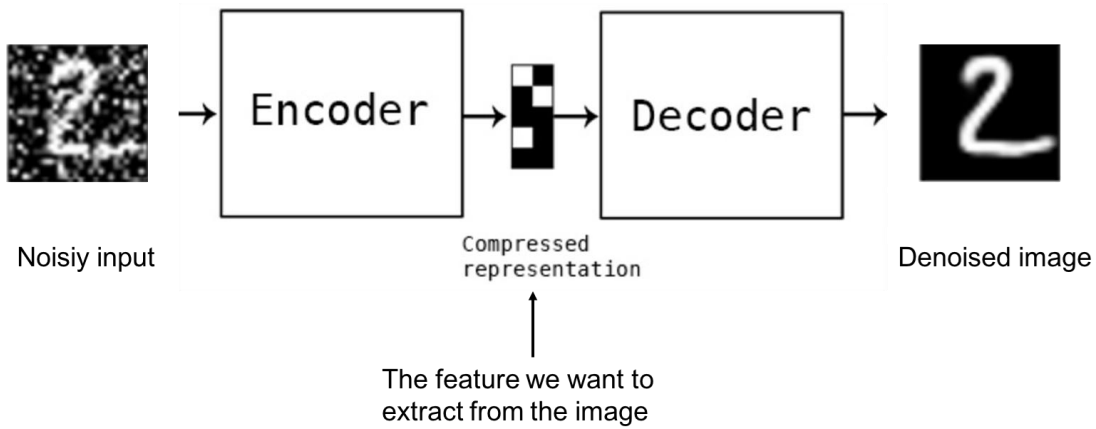
- Exact computation of the posterior is often intractable. Using sampling methods (rejection sampling, importance sampling, MCMC) allow us in some cases to find approximations, some drawbacks remain.
- Variational inference trades accuracy (no exact posterior) for computational efficiency. Traditionally, this was done by using a simpler, parameterized distribution. Then, we would minimize the KL-divergence between the true posterior and the simpler distribution.
- In practice, having samples from the true posterior, we would implement a **Monte Carlo** approach.
- For example, for S samples of the parameters θ, $\{\theta^{(s)}\}_{s=1}^{S}, \forall\, \theta^{(s)} \sim q_\phi(\theta)$, we can maximize:

$$\mathrm{ELBO}(\phi) \approx \frac{1}{S}\sum_{s=1}^{S}\left[\log\left(P(Y,\theta^{(s)}\right) - \log\left(q_\phi\left(\theta^{(s)}\right)\right)\right].$$

Autoencoders (precursor)

- A standard autoencoder (**AE**) is a network with two functions: an encoder and a decoder.
- Inbetween those sits some efficient representation (encoding) that allows the decoder to reconstruct its inputs.
- Frequently-used types of AEs are sparse, contractive, and denoising (figure on the lower right).
- In this lecture, we will take a closer look at a **variational** autoencoder (**VAE**).





(*) Images taken from Wikipedia.

Part 1: Variational Autoencoders

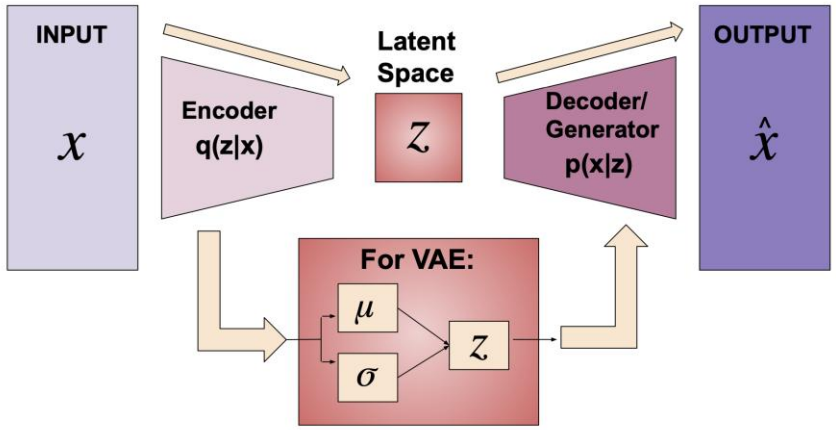


Image taken from https://www.math.emory.edu/site/cmds-reuret/projects/2024-vae/

- In the examples so far, we have **quantified uncertainty** about model parameters. If, after optimization, we wanted to use these models **generatively**, we could have sampled sets of new parameters, make predictions with each, and analyze the resulting distribution of predictions (we cannot sample new data).
- In a **VAE**, we **quantify uncertainty per individual data point** (in terms of latent variables). Typically, we use amortized inference, that is, the encoder predicts conditional parameters for the latent variables (the variational distribution).
- Compared to a standard (denoising) AE, here the bottleneck has N units, each of which shall be trained to follow a distribution. In a vanilla VAE, we choose the same mean-field approximation (standard isotropic Gaussian) for that (as earlier).

VAEs (formally)

For observable variables $\boldsymbol{x}$ and latent variables $\boldsymbol{z}$, a vanilla VAE has **three** major components:

Encoder network: Takes data $\boldsymbol{x}$ and predicts parameters of the chosen mean-field approximation, $q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$. We'll be using a Gaussian with independent components.

Variational distribution: $q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$, sits in the bottleneck. The true and more expressive distribution $p(\boldsymbol{z}|\boldsymbol{x})$ is approximated by a simple parametric family.

Decoder network: $p_{\theta}(\boldsymbol{x}|\boldsymbol{z})$, learns to reconstruct the original inputs using the latent variables only.

The **ELBO** then is: $\mathcal{L}(\theta, \phi; \boldsymbol{x}) = \mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\big[\log(p_{\theta}(\boldsymbol{x}|\boldsymbol{z}) - D_{KL}\big(q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z})\big]$.

Tip: The closed-form analytical KL-divergence between two Gaussians is: $D_{KL}\left(\left(q_{\mu,\sigma}(\boldsymbol{z}) \,||\, \mathcal{N}(0, I)\right)\right) = \frac{1}{2}\sum_j \left(\sigma_j^2 + \mu_j^2 - 1 - \log\left(\sigma_j^2\right)\right)$.

VAEs (computationally)

1. Inputs x → 2. Encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ →

3. Sample from ϕ-parameterized variational distribution: $z \sim q_\phi(z|x)$ → 4. Decode z: $x' \approx p_\theta(x|z)$.

Reconstruction loss, e.g., MSE: $|| x - x'||^2$. KL-divergence (previous slide).

As is obvious, if we were to sample $z \sim q_\phi$ in step 3 above, we would not be able to differentiate our model any longer. Therefore, z needs to be created using the **reparameterization** trick:

– (3a) Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(0,1)$ first.

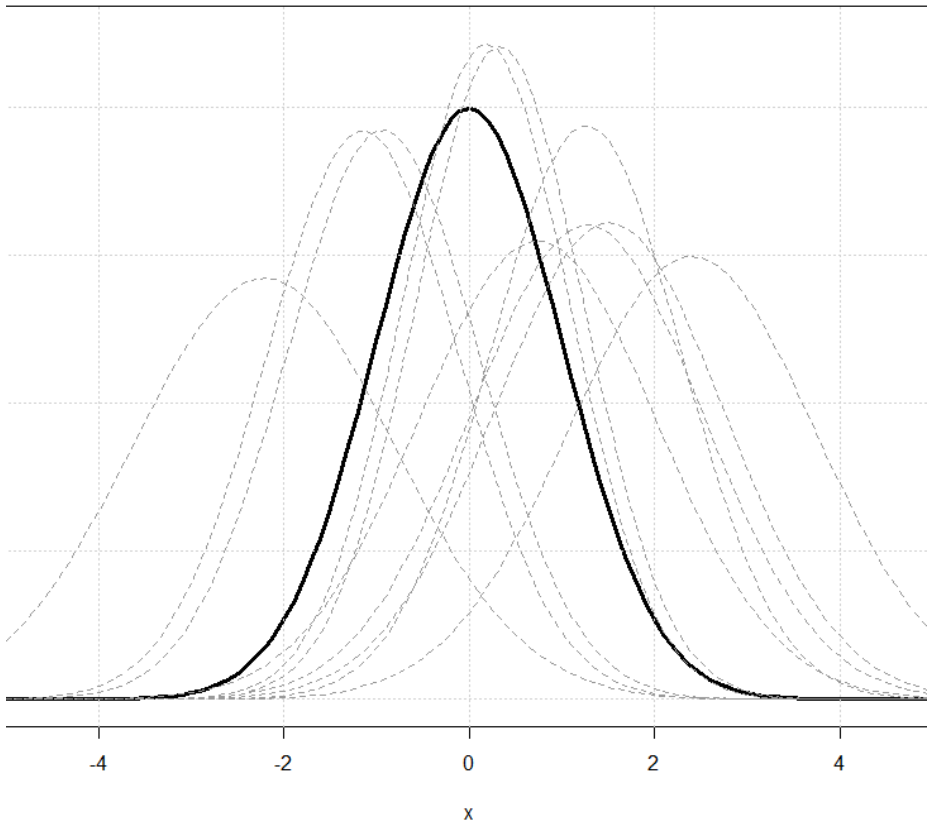
– (3b) $\boldsymbol{\mu}, \boldsymbol{\sigma}$ are data-conditional predictions from the encoder, so we construct $\boldsymbol{z} = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}(\boldsymbol{x}) \cdot \boldsymbol{\epsilon}$.

VAEs (practically)

ELBO: $\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log(p_\theta(\boldsymbol{x}|\boldsymbol{z}) - D_{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z})\right]\right.$

In practice, when maximizing the **ELBO**, one could think that the encoder tends to maximize the first term (expected data likelihood) by collapsing all conditional means and variances to zero and one (the prior), respectively.

- **Case 1**: If the conditional distributions would become very close or equal to the prior, then encoding would be uninformative. The decoder could never reconstruct well.
- **Case 2**: If we had no KL-loss, the focus would be on reconstruction. The variational posterior $q_\theta(z|x)$ would collapse to very different distributions for each data point (overfitting each). The latent space would not exhibit structured, smooth, or regularized behavior.
- **Case 3**: (Figure) In the ideal, balanced scenario, there is a trade-off. While the individual variational posteriors can differ from the prior enough to encode important discriminative information. But on average, these variational posterior will cluster around the prior distribution.



VAEs (sampling)

In a well-trained VAE, individual variational distributions $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ are typically distinct from the prior, but when averaging over all data-induced variational distributions, that aggregate $q_\phi(\boldsymbol{z})$ will look close to the prior.

Recall that the KL-divergence encourages our variational distribution to be close to our prior. This allows us to sample **unconditionally** directly from the prior: $\boldsymbol{z} \sim p(\cdot)$ and then to decode $\boldsymbol{x} = p_\theta(\boldsymbol{x}|\boldsymbol{z})$.

If we wish to sample conditionally, then we will have to encode a sample first. Once it is in the latent space, we could modify it there, then decode it. Some more suggestions:

1. Latent space interpolation: interpolate between two or more existing latent vectors $z^{(1)}, z^{(2)}$.
2. Reconstruction sampling: Encode a sample and obtain mean & variance. Use these to draw a new sample from this particular posterior distribution.
3. Temperature scaling: modify prior: $\mathcal{N}(0, T \cdot I)$, then sample z from it. Typically, a lower temperature ($T < 1$) produces more conservative / less diverse samples, while $T > 1$ often yields more creative (but less realistic) ones.

Part 2: Normalizing Flows (NFs)

A **Normalizing Flow** (NF) is a clever method that exploits the change-of-variable law of probabilities to approximate a complex, latent distribution by learning a (often invertible) transformation.

In practice, this transformation is a **diffeomorphism** – an **invertible** function that maps one differentiable manifold to another such that both the function and its inverse are continuously differentiable (making it a flow-based **generative** model).

For a Ddimensional vector $\boldsymbol{x}$, suppose we want to define a joint distribution over $\boldsymbol{x}$. We can express it using a transformation to some real vector $\boldsymbol{u}$ that we sample from the *known base distribution* $p_u(\boldsymbol{u})$.

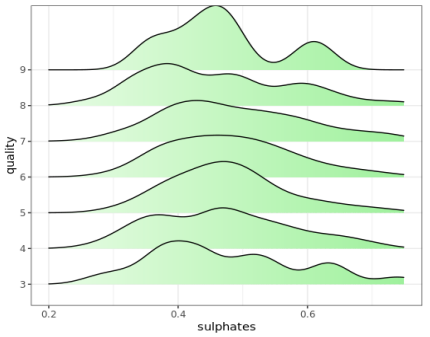
Then, by a change of variables:

$$p_x(\boldsymbol{x}) = p_u(\boldsymbol{u}) \cdot |\det J_{T^{-1}}(\boldsymbol{u})|, \text{ where } \boldsymbol{u} = T^{-1}(\boldsymbol{x}).$$

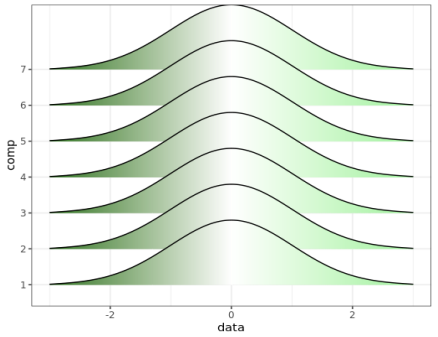
Normalizing Flows

(base mechanics)

Marginal distributions of X.



Marginal distributions of U.



Usually, we would define our transformation as a composition of smaller transformations.

$$z_K = T_K(z_{k-1}), \dots, z_2 = T_2(z_1), z1 = T_1(z_0) = \mathrm{T}_K(z_{k-1}) \circ \dots \circ T_1(z_0).$$

If we want to compute a determinant like: $\det \frac{\partial z_K}{\partial z_0}$, i.e., the determinant of the transformation that outputs z_K and started with z_0 (an initial sample from the base distribution), then it becomes obvious that the dimensionalities must match, as the determinant is a function only defined for square matrices.

If our transformation is, in fact, a diffeomorphism, $\mathbb{R}^X = \mathbb{R}^U$ is required anyways.

Normalizing Flows (determinant mechanics)

The Jacobian is the $D \times D$ matrix of all partial derivatives of T given by: $J_T(\boldsymbol{u}) = \begin{matrix} \frac{\partial T_1}{\partial u_1} & \cdots & \frac{\partial T_1}{\partial u_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_D}{\partial u_1} & \cdots & \frac{\partial T_D}{\partial u_D} \end{matrix}$.

We can also compose transformations:

$$(T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1},$$
$$\det J_{T_2 \circ T_1(\boldsymbol{u})}(\boldsymbol{u}) = \det J_{T_2}\left(T_1(\boldsymbol{u})\right) \cdot \det J_{T_1}(\boldsymbol{u}).$$

But what is the **crux**? Computing the Jacobian determinant is very costly and often not numerically stable. NFs are enabled by using transformations of which the determinant is efficiently and tractably obtained.

In practice, we compose our transformations from functions with a **triangular Jacobian**. The (log) determinant of a triangular matrix is the product (sum) of its diagonal entries!

Normalizing Flows (optimization w/ forward KL-divergence)

Normalizing flows model the likelihood $q_\theta(x)$ explicitly using the change-of-variables formula. The forward KL-divergence is used for when we have samples from $p(x)$, but cannot evaluate it. Recall that the divergence is an **expectation** that we can approximate using a **Monte Carlo** approach.

$$
\begin{aligned}
D_{\mathrm{KL}}\big(p(x) \,||\, q_\theta(x)\big) &= \mathbb{E}_{x\sim p(x)}\left[\log \frac{p(x)}{q_\theta(x)}\right], \\
&= \underbrace{\int p(x)\cdot \log\big(p(x)\big)\; dx}_{\text{constant w.r.t. } \theta = C} - \int p(x)\cdot \log\big(q_\theta(x)\big)\; dx\,, \\
&= -\mathbb{E}_{x\sim p(x)}\big[\log\big(q_\theta(x)\big)\big] + C, \\
&\approx -\mathbb{E}_{x\sim p(x)}\left[\log\left(p_U\left(f_\theta^{-1}(x)\right)\right) + \log\left(\left|\frac{\partial\, f_\theta^{-1}}{x}\right|\right)\right].
\end{aligned}
$$

— $p(x)$: data distribution.

— $p_U(u)$: flow's simple base distribution.

— $q_\theta(x) = \mathrm{p_U(u)} \cdot \left|\det \frac{\partial\, z}{\partial\, x}\right|$, where $z = f_\theta^{-1}(x)$.

Change-of-Variable (for Random Variables)

If we apply a (non-linear) transformation $g(x)\colon \mathbb{R} \to \mathbb{R}$ to a random variable X, then its PDF $p_X(x)$ changes. For example, assume $g(x) = x^2$ with inverse $\sqrt{x}$. We will define a new variable $Y = g(X)$.

For a monotonic $g(x)$ the PDF $\mathrm{p}_Y(y)$, the transformed random variable Y can be computed:

$$p_Y(y) = p_X\big(g^{-1}(y)\big) \cdot \left|\frac{d\,z}{d\,y}\right|, \text{ where } z = g^{-1}(y).$$

We need to introduce this scaling to correct for distortions in the volume. In other words, if we want the mass between X, Y to be the same, we need to consider how our transformation contracts and expands the volume.

In higher dimensions, this **generalizes** to the determinant of the Jacobian of the transformation.

I used this tutorial here: https://www.mariushobbhahn.com/2021-05-20-Change_of_variable/

Normalizing Flows (sampling)

Since an NF models the likelihood directly, it is straightforward to evaluate the likelihood of a sample coming from the data distribution $p(\boldsymbol{x}) = p_U\left(f^{-1}(\boldsymbol{x})\right) \cdot \left|\det \frac{\partial f^{-1}(\boldsymbol{x})}{\boldsymbol{x}^\top}\right|$.

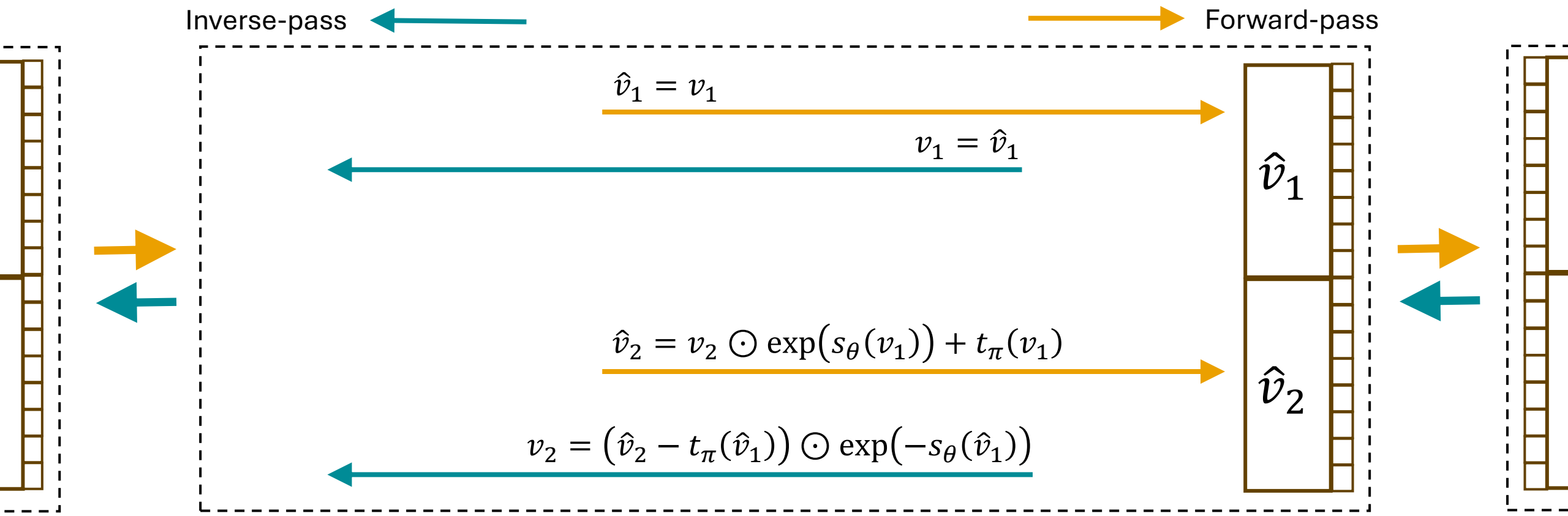
Unconditional sample generation is straightforward, too:

1. Draw a sample from the base distribution: $\boldsymbol{u} \sim p_U(\boldsymbol{u})$.
2. Transform sample to data space: $\boldsymbol{x} = f(\boldsymbol{u})$.

We can draw **conditional** samples, too. For that, we should first train a conditional normalizing flow. There are different ways to do this. Most commonly, a vector of conditions $\boldsymbol{c}$ is supplied along each sample and passed to both, the (inverse-)transformation and the (then-conditional) base distribution.

Normalizing Flows (here Inside an *affine* (scale & translate) Coupling Block)

$X \in \mathbb{R}^D$ and $v_1, v_2, \hat{v}_1, \hat{v}_2 \in \mathbb{R}^{1:d}$, where $d = \lceil D/2 \rceil$. Parameters θ and π are learned by the conditioner. A coupling block consists of an invertible transformation (a coupling law), parameters for it learned by a so-called conditioner, and data.

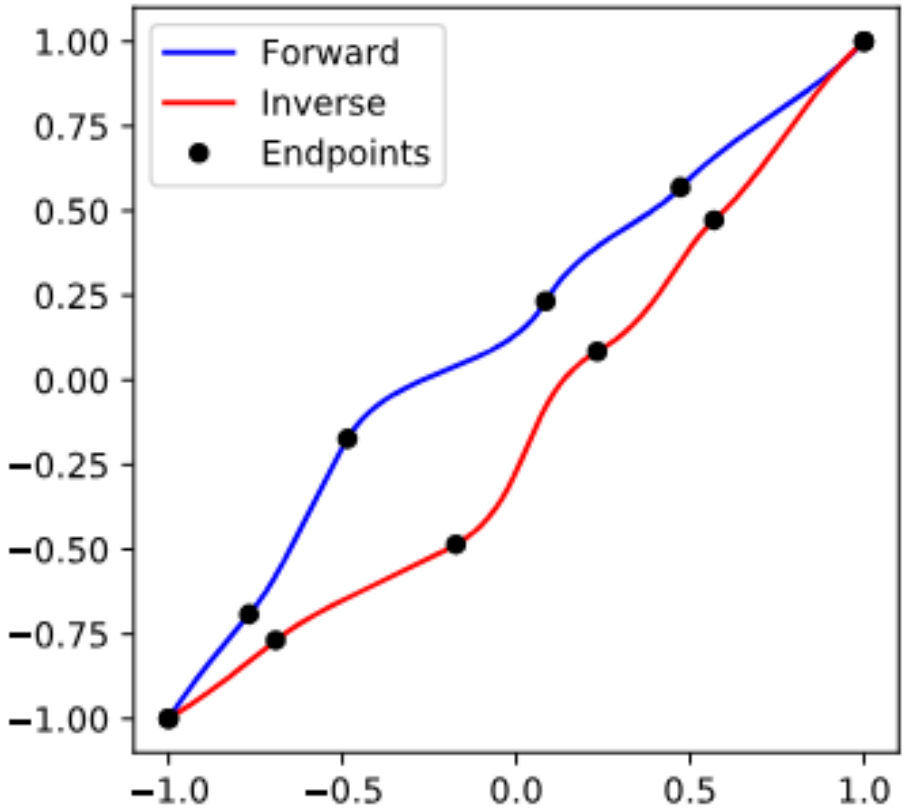


Normalizing Flows (types of bijectors)

In the context of NFs, the composable, invertible transformations are commonly called **bijectors**.

Commonly used bijectors are, for example (cf. [9]):

- Affine
- Combination-based (conic combinations; combinations of monotonic functions)
- Integration-based transformers (no practical use)
- Spline-based transformers (**SotA**), e.g., Rational-Quadratic Splines [18] (Figure on the left).



Invertible Spline Bijector.

Part 3: VAEs with NFs as Approximate Posterior

So far, we have used a mean-field approximation as the variational distribution in a VAE.

1. Inputs x → 2. Encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ → 3. R-Trick: Draw $\epsilon \sim \mathcal{N}(0,1)$ → 4. Define $z_0 = \mu(x) + \sigma(x) \cdot \epsilon$ →
5. Norm. Flow $q_\phi(\boldsymbol{z}_K|\boldsymbol{z}_0)$, where $\boldsymbol{z}_K = \mathrm{T}_K(\boldsymbol{z}_{k-1}) \circ \ldots \circ T_1(\boldsymbol{z_0})$ → 6. Decode z_K: $\boldsymbol{x}' \approx p_\theta(\boldsymbol{x}|\boldsymbol{z}_K)$.

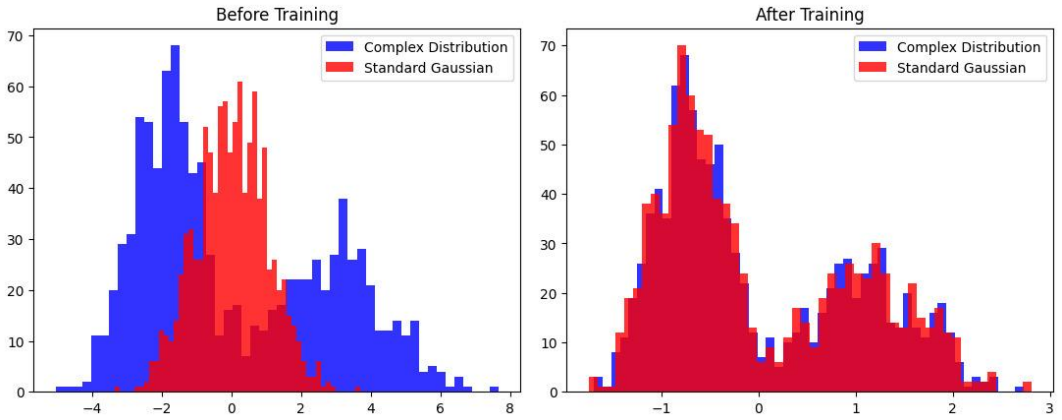


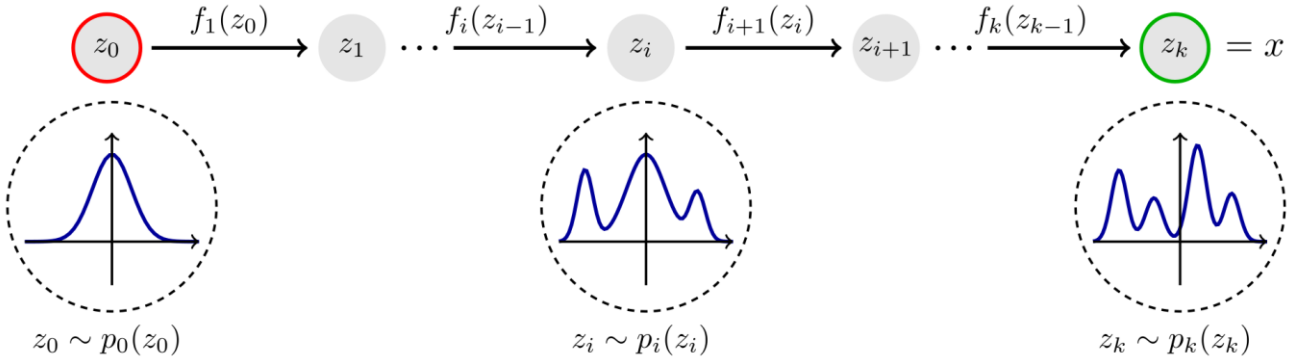
Image taken from https://www.math.emory.edu/site/cmds-reuret/projects/2024-vae/

VAEs with NFs (some details)

Note that in a variational setting, the Normalizing Flow is turned around! (like in the figure). Our data flows through the encoder and then enters the “base space”. Usually, we choose a standard diagonal normal there, too, which goes nicely with the setup we had so far:

$$p_X(x = z_K) = q_\theta(x) = \psi_\phi(z_0) \cdot \left|\det \frac{\partial z_K}{z_0}\right|.$$

Note how the variational parameters $\theta = \langle \mu, \sigma \rangle$ flow back (are applied to) the base distribution. This is important to calculate the KL-divergence correctly.



Assignment

Reading (next lecture will be about LLMs):

- “Attention Is All You Need” by Vaswani et al.; https://doi.org/10.48550/arXiv.1706.03762
- “Retrieval-Augmented Generation for Knowledge-Intensive NLP Task” by Lewis et al.; https://doi.org/10.48550/arXiv.2005.11401

Practical:

- Implement notebook “VAEs and Normalizing Flows”

Deadline: Reading (before next lecture); Practical (Wed., May 28th, 23:59).

Announcements

Students who haven’t received a final grade in the ML course: Please contact the professor and select 1h on Fr 30/5 9-16 or on Mo 2/6 9-16 for an oral retake of the exam.

Next Lecture (LLMs, Transformers, Retrieval-augmented Generation)

Large Language Models are probabilistic models, too. But unlike the models we have seen so far, they do not attempt to capture a probability distribution explictly.

Rather, esp. in the case of next-token-prediction, they present a softmax distribution over all possible tokens. In other words, each globally possible token is assigned a probability to continue the given input.

This and more will be part of the next lecture.