Variational inference and VAEs

CPSC 440/550: Advanced Machine Learning

cs.ubc.ca/~dsuth/440/24w2

University of British Columbia, on unceded Musqueam land

2024-25 Winter Term 2 (Jan–Apr 2025)

Last time: variational inference

- Finding "best" approximation q from some family to unnormalized target $\tilde{p}$
  - Often, will be $\tilde{p}(\theta \mid \mathbf{X}) = p(\mathbf{X} \mid \theta) p(\theta)$
- Usual objective: $\arg\min_\phi \mathrm{KL}(q_\phi \,\|\, \tilde{p}) = \arg\max_\phi \mathrm{Entropy}[q_\phi] + \mathbb{E}_{X \sim q_\phi}[\log \tilde{p}(X)]$

- To estimate gradients of the objective, we use the reparameterization trick:
  - Write $X \sim q_\phi$ as combination of: $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $X = f(\varepsilon, \phi)$
    - e.g. $f(\varepsilon, \boldsymbol{\mu}, \mathbf{L}) = \mathbf{L}\varepsilon + \boldsymbol{\mu}$ gets $\mathcal{N}(\boldsymbol{\mu}, \mathbf{L}\mathbf{L}^\mathsf{T})$
  - Then, under reasonable regularity conditions,

$$\nabla_\phi \mathop{\mathbb{E}}_{X \sim q_\phi} \log \tilde{p}(X) = \nabla_\phi \mathop{\mathbb{E}}_{\varepsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})} \log \tilde{p}(f(\varepsilon, \phi)) = \mathop{\mathbb{E}}_{\varepsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})} \nabla_\phi \log \tilde{p}(f(\varepsilon, \phi));$$

    now we can take a Monte Carlo sample for ε (often just one sample)
    and use autodiff to evaluate $\nabla_\phi \log \tilde{p}(f(\varepsilon, \phi))$
  - Can also have ε follow some other distribution, as long as it doesn't depend on ϕ

Last time: VAEs

- Deep latent variable model: something like

$$Z \sim \mathcal{N}(\mathbf{0}_k, \mathbf{I}_k) \qquad X \mid (Z = z) \sim \mathcal{N}(g_\theta(z), \sigma^2 \mathbf{I}_d)$$

- Sampling is easy: sample a Z, run it through the network g_θ, add normal noise
- Inference is hard:

$$p(z \mid x) = \frac{p(z)p(x \mid z)}{p(x)} = \frac{p(z)p(x \mid z)}{\int p(z')p(x \mid z')\mathrm{d}z'}$$

  - Finding most likely z: find the (small-norm z) that gets $g_\theta(z) \approx x$
  - Finding the distribution: requires complicated integral over all possible z
- So we decide to do *approximate* inference with a recognition network: use $q_\phi(z \mid x) = \mathcal{N}(z; \boldsymbol{\mu}_\phi(x), \boldsymbol{\Sigma}_\phi(x))$ with $\boldsymbol{\mu}_\phi$, $\boldsymbol{\Sigma}_\phi$ a neural network
- Training based on ELBO (recapped next)

Last time: VAE objective

- We got, similar to ELBO derivation for EM, that

$$\log p_\theta(x) = \mathrm{ELBO}_{\theta,\phi}(x) + \mathrm{KL}(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x))$$

- $\max_\phi \sum_i \mathrm{ELBO}_{\theta,\phi}(x^{(i)})$ aims for $q_\phi(z \mid x) \approx p_\theta(z \mid x) = \dfrac{p_\theta(x \mid z) p_\theta(z)}{\int p_\theta(x \mid z') p_\theta(z') \mathrm{d}z'}$
  - Try to get approximate inference network to be consistent with p_θ on the $x^{(i)}$
- $\max_\theta \max_\phi \sum_i \mathrm{ELBO}_{\theta,\phi}(x^{(i)})$ approximates $\max_\theta \sum_i \log p_\theta(x^{(i)})$
  - So if we max over both θ and ϕ, we should approximately get the MLE
- The objective is based on $\mathrm{ELBO}_{\theta,\phi}(x)$ which is

$$\mathop{\mathbb{E}}_{z \sim q_\phi(z|x)}\left[\log \frac{p_\theta(x, z)}{q_\phi(z \mid x)}\right] = \mathop{\mathbb{E}}_{z \sim q_\phi(z|x)}[\log p_\theta(x \mid z)] - \mathrm{KL}(q_\phi(z \mid x) \,\|\, p_\theta(z))$$

  - First, encoding then decoding should be consistent; use reparamaterization trick
  - Second, encoding shouldn't be "too weird"; closed-form function of ϕ for Gaussians

Outline

Convolutions and transposed convolutions

- A VAE for images has two parts: typically

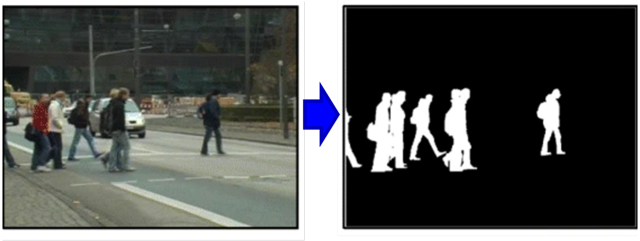
$$\begin{array}{lrcl} \text{encoder:} & \text{image } X & \Longrightarrow & \text{latent } Z \sim \mathcal{N}\left(\boldsymbol{\mu}_\phi(X), \boldsymbol{\Sigma}_\phi(X)\right) \\ \text{decoder:} & \text{latent } Z & \Longrightarrow & \text{image } \hat{X} \sim \mathcal{N}\left(g_\theta(Z), \sigma^2 \mathbf{I}\right) \end{array}$$

- Image is maybe $256 \times 256 \times 3$ (196,608 dimensions)
- We want the latent to be lower-dimensional (tens, maybe thousands, depending)
- Going from high-dimensional image to low-dimensional output is familiar
  - Baseline approach: convolutional and pooling layers
- What kind of layers should we use to output an image?

Related problem: pixel-level classification

- Sometimes you want to apply a label to each pixel in an image:

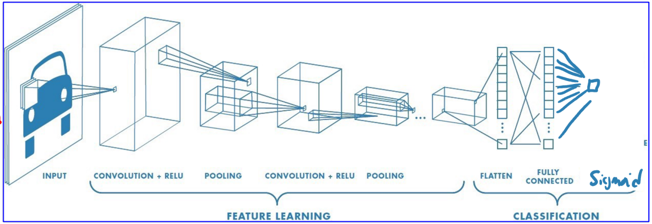
Is this a pedestrian?



Is this a car, a sidewalk, a building, . . . ? (semantic segmentation)

Naive approach: sliding window

- Train a CNN (or whatever) that predicts a pixel's label given its neighbourhood



- Apply it to each pixel, given its neighbourhood
  - Turns the problem into familiar image classification
  - Easy to apply to images with different sizes
  - Slow: need to run the CNN once per pixel in the image!
  - Need to choose the right window size, might not be as good at "sharing information"

Another approach: multi-label classification

- Reduce to some low-dimensional latent, treat latent like multi-label classification

$$\underbrace{\text{image } X}_{256\times256\times3} \xRightarrow{\text{conv, . . . , conv, dense}} \underbrace{\text{latent } Z}_{1024} \xRightarrow{\text{dense, dense}} \underbrace{\text{classifications } \hat{Y}}_{256\times256\times\#\text{ classes}}$$

  Y_1: "is top-left pixel a pedestrian?", $Y_{65,536}$: "is bottom-right pixel a pedestrian?"
- Looks exactly like typical multi-label architecture:

$$\underbrace{\text{image } X}_{256\times256\times3} \xRightarrow{\text{conv, . . . , conv, dense}} \underbrace{\text{latent } Z}_{1024} \xRightarrow{\text{dense, dense}} \underbrace{\text{classifications } \hat{Y}}_{\#\text{ of labels}}$$

  Y_1: "is this a selfie?", Y_2: "is this a screenshot?", Y_3: "is this NSFW?"
- Faster than sliding window: only run through the network once
- Requires fixed image sizes
- Many labels
- Each problem is hard since spatial information in Z is all mixed up

Fully-convolutional networks

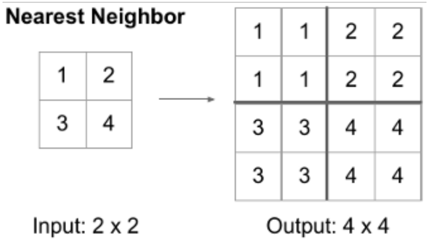
- Make sure that the latent keeps spatial structure

$$\underbrace{\text{image } X}_{256\times 256\times 3} \xrightarrow{\text{conv, } \ldots, \text{ conv}} \underbrace{\text{latent } Z}_{16\times 16\times 4} \xrightarrow{\text{up-sample}} \underbrace{\text{classifications } \hat{Y}}_{256\times 256\times \#\text{ classes}}$$

- Still fast since it's all one pass through a single network
- Problems are easier since we still have local structure
- Everything is convolutional: works on different sizes of images
- Long, Shelhamer, Darrell (2014); quickly became default approach after

How to up-sample?

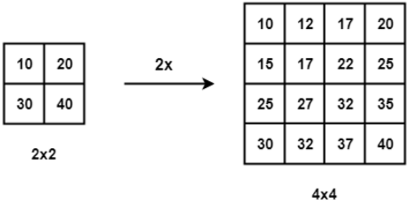
- Goal of up-sampling/decoder is to go from small image to bigger image
- Simplest approach: nearest-neighbour interpolation



https://towardsdatascience.com/
transposed-convolution-demystified-84ca81b4baba

How to up-sample?

- Goal of up-sampling/decoder is to go from small image to bigger image
- Simplest approach: nearest-neighbour interpolation
- Slightly fancier: bilinear interpolation takes weighted combination of corners



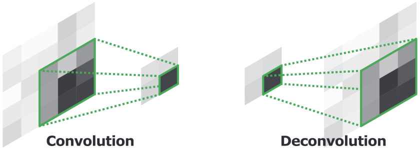
https://towardsdatascience.com/
transposed-convolution-demystified-84ca81b4baba

How to up-sample?

- Goal of up-sampling/decoder is to go from small image to bigger image
- Simplest approach: nearest-neighbour interpolation
- Slightly fancier: bilinear interpolation takes weighted combination of corners
- There are of course even fancier traditional methods (bicubic, splines, . . .)
- Instead, let's learn our upsampling operation

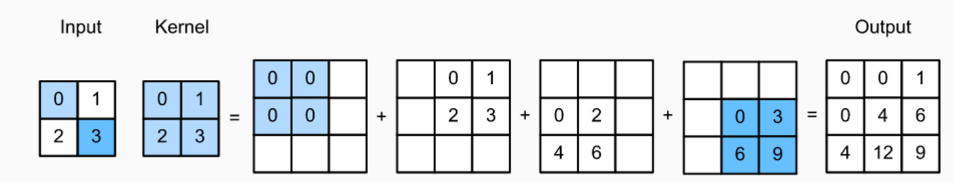
Transposed convolution

- Usual base layer: transposed convolution / deconvolution
  - Note: different thing from signal processing's notion of deconvolution!



https://arxiv.org/abs/1505.04366

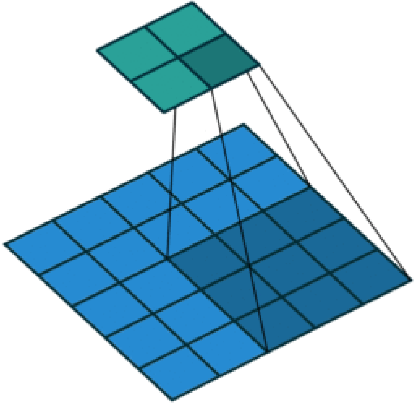
- Convolution layer: output pixel is a linear combination of input window
- Transposed convolution: each input pixel produces several output pixels, which overlap and are added up to make the output image



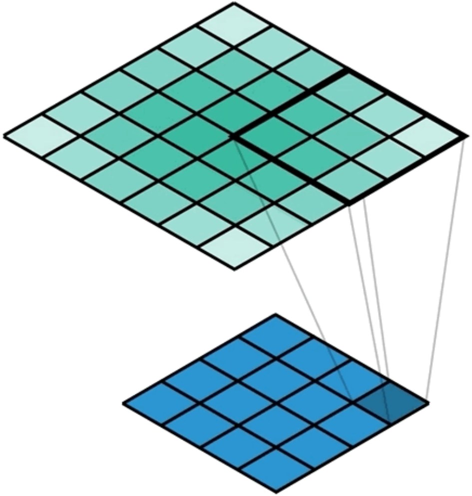
https://d2l.ai/chapter_computer-vision/transposed-conv.html

Transposed convolution

- Convolution:



- Transposed convolution:



Why "transposed" convolution?

- We can write convolution as a matrix multiplication:

$$\begin{bmatrix}1 & 2 & 3\\ 6 & 5 & 3\\ 1 & 4 & 1\end{bmatrix} \star \begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix} = \begin{bmatrix}22 & 21\\ 22 & 20\end{bmatrix} \qquad \begin{bmatrix}1 & 2 & 0 & 2 & 1 & 0 & 0 & 0 & 0\\ 0 & 1 & 2 & 0 & 2 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1 & 0\\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1\end{bmatrix} \begin{bmatrix}1\\ 2\\ 3\\ 6\\ 5\\ 3\\ 1\\ 4\\ 1\end{bmatrix} = \begin{bmatrix}22\\ 21\\ 22\\ 20\end{bmatrix}$$

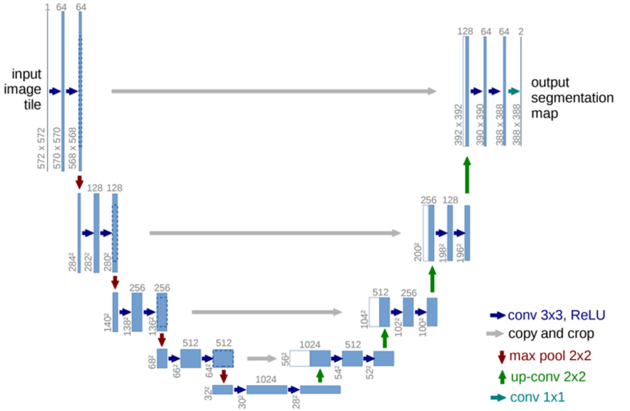
- We can also write write transposed convolution as a matrix multiplication:

$$\begin{bmatrix}1 & 2\\ 2 & 4\end{bmatrix} \otimes \begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix} = \begin{bmatrix}1 & 4 & 4\\ 4 & 13 & 10\\ 4 & 10 & 4\end{bmatrix} \qquad \begin{bmatrix}1 & 2 & 0 & 2 & 1 & 0 & 0 & 0 & 0\\ 0 & 1 & 2 & 0 & 2 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1 & 0\\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1\end{bmatrix}^{\mathsf{T}} \begin{bmatrix}1\\ 2\\ 2\\ 4\end{bmatrix} = \begin{bmatrix}1\\ 4\\ 4\\ 4\\ 13\\ 10\\ 4\\ 10\\ 4\end{bmatrix}$$

- With the same filter, get the transpose of the corresponding matrix

U-Nets

- Convolutions, pooling lose a lot of information
  - If the latent is $16 \times 16 \times k$, hard to remember/guess where the original boundaries in the 256×256 image were *exactly*
- Various approaches to let the network see the original image to "check"
- U-Nets connect back to when they processed at the same resolution



Getting labels for semantic segmentation

bonus!

- Getting labels for *every pixel in an image* is slow and expensive
- One possibility: simulated environments where you know what everything is
    - Might not match real data well!

Video game　　　　Google street view

https://arxiv.org/abs/1608.01745

Getting labels for semantic segmentation

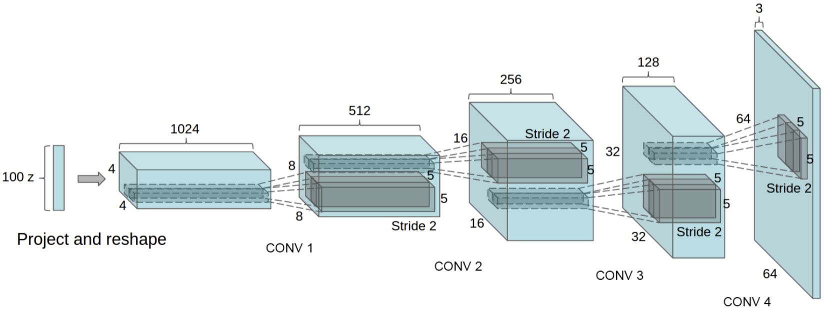
bonus!

- Getting labels for *every pixel in an image* is slow and expensive
- One possibility: simulated environments where you know what everything is
    - Might not match real data well!
- Other options: label one pixel or a scribble per object, guess at boundaries

https://arxiv.org/abs/1807.09856

VAE decoder

- Model *shouldn't* refer back to the original image!
- Typical basic architecture based on transposed convolutions, e.g.:



https://arxiv.org/abs/1511.06434

Outline

Uses of (deep) latent variable models

- Sometimes, what we care about is getting a good $p(x)$
  - Analogy: get a better fit to my data with a Gaussian mixture than just one
- Latent variables Z are just "nuisance variables," can throw them out after fitting

- Sometimes, the Z themselves are useful to get insight about the data
  - Analogy: using GMM as a clustering algorithm and analyzing the clusters
- Another example: can think of PCA as

$$\text{data } X \overset{\text{linear map}}{\Longrightarrow} \text{latents } Z \overset{\text{linear map}}{\Longrightarrow} \text{reconstruction } \hat{X}$$

  and what we really care about is Z
- We hope that Z tells us about structure in the data
- For example, maybe $\|z - z'\|$ is more "semantically meaningful" than $\|x - x'\|$

Autoencoders

- PCA can also be called a linear autoencoder:

$$\min_{\hat{x}} \sum_{i=1}^{n} \|x^{(i)} - \hat{x}^{(i)}\|^2 = \min_{\text{linear } f,g} \sum_{i=1}^{n} \|x^{(i)} - g\left(f\left(x^{(i)}\right)\right)\|^2$$

  - f is the encoder: turns $x \in \mathbb{R}^d$ into a latent $z \in \mathbb{R}^k$
  - g is the decoder: reconstructs $z \in \mathbb{R}^k$ into an original data point $x \in \mathbb{R}^d$
  - If $k \geq d$, can get zero loss by using the identity function $f(x) = x$
  - If $k < d$, can't do the identity function; try to save as much structure as possible
- Suggests nonlinear autoencoders: with deep encoder f_ϕ and decoder g_θ,

$$\min_{\phi,\theta} \sum_{i=1}^{n} \|x^{(i)} - g_\theta\left(f_\phi\left(x^{(i)}\right)\right)\|^2$$

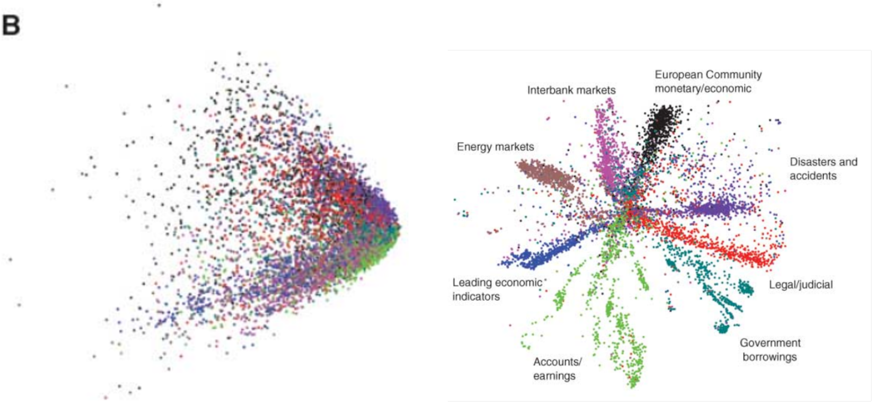
- VAEs make both the encoder and the decoder random distributions
- Regular autoencoder is $\approx$ MLE for the VAE with

$$q_\phi(z \mid x) \propto \mathbb{1}(z = f_\phi(x)) \qquad p_\theta(x \mid z) = \mathcal{N}(x; g_\theta(z), \sigma^2 \mathbf{I}) \qquad p_\theta(z) \propto 1$$

Representation learning with autoencoders

- Some reasons we might want to use an autoencoder:
- Compress a high-dim x into a low-dim z that doesn't lose much information
- Use a two-dimensional z and plot the "latent structure" in the data

B



https://www.cs.toronto.edu/~hinton/science.pdf

these days people usually use t-SNE instead; see distill.pub/2016/misread-tsne/

Latent space interpolation

bonus!

https://arxiv.org/abs/2204.06125

Representation learning for semi-supervised learning

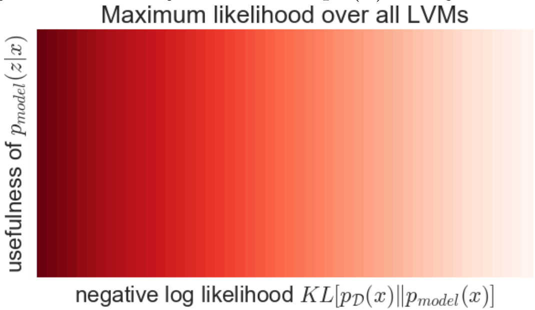
- Common problem: I have tons of unlabeled data but not much labeled data
- Unsupervised pre-training (also called self-supervised):
  1. Find a representation on the unlabeled data
  2. Learn a simple model on the labeled data, using that representation
     - Nearest-neighbour, a linear model, a small network, ...

- We hope that the representation captures the important structure of the data
- If so, then the simple model based on the labeled data can learn quickly
  - With this representation, it's an easy problem!

- Can do this with a "plain autoencoder"
- But the distribution of latents can be very "irregularly shaped," plus can overfit
- VAEs try to make the distribution of latents close to standard normal
- Randomized encoder and decoder also $\approx$ regularization
- Hopefully makes for nicer representations, in addition to allowing sampling

Representation Learning with Latent Variable Models

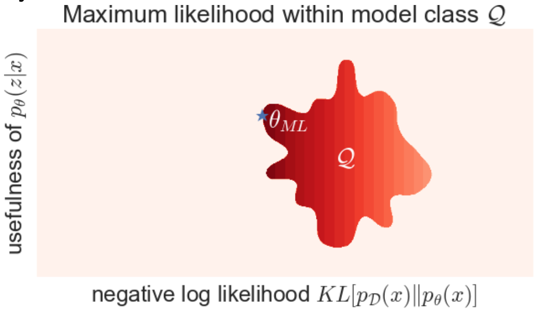
- We'd often like a "useful" distribution for $Z \mid X$
- Maximum likelihood minimizes KL between target and $p_\theta(x) = \int p_\theta(x, z)\mathrm{d}z$
- Objective wants a good fit for $p_\theta(x)$; doesn't care about usefulness at all
  - True for *any* objective that only cares about $p_\theta(x)$, not just MLE in VAEs



https://www.inference.vc/maximum-likelihood-for-representation-learning-2/

Representation Learning with Latent Variable Models

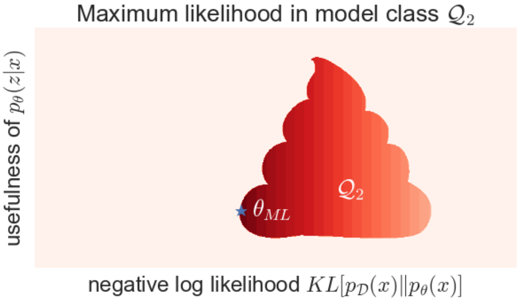
- We'd often like a "useful" distribution for $Z \mid X$
- Maximum likelihood minimizes KL between target and $p_\theta(x) = \int p_\theta(x, z) \mathrm{d}z$
- Objective wants a good fit for $p_\theta(x)$; doesn't care about usefulness at all
  - True for *any* objective that only cares about $p_\theta(x)$, not just MLE in VAEs
- But we don't actually maximize over all latent variable models



https://www.inference.vc/maximum-likelihood-for-representation-learning-2/

Representation Learning with Latent Variable Models

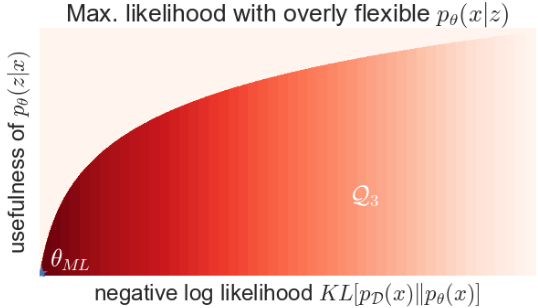
- We'd often like a "useful" distribution for $Z \mid X$
- Maximum likelihood minimizes KL between target and $p_\theta(x) = \int p_\theta(x, z) \mathrm{d}z$
- Objective wants a good fit for $p_\theta(x)$; doesn't care about usefulness at all
  - True for *any* objective that only cares about $p_\theta(x)$, not just MLE in VAEs
- But we don't actually maximize over all latent variable models
- This relies on our model class (or really, learning process. . .) aligning well



https://www.inference.vc/maximum-likelihood-for-representation-learning-2/

Representation Learning with Latent Variable Models

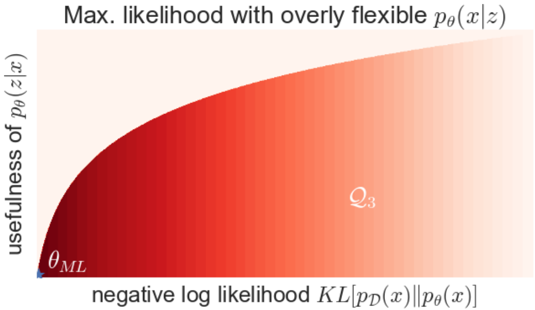
- We'd often like a "useful" distribution for $Z \mid X$
- Maximum likelihood minimizes KL between target and $p_\theta(x) = \int p_\theta(x, z) \mathrm{d}z$
- Objective wants a good fit for $p_\theta(x)$; doesn't care about usefulness at all
  - True for *any* objective that only cares about $p_\theta(x)$, not just MLE in VAEs
- But we don't actually maximize over all latent variable models
- This relies on our model class (or really, learning process...) aligning well
- Real(ish) case: if $p_\theta(x \mid z)$ is too powerful, ignores z, i.e. useless representation



https://www.inference.vc/maximum-likelihood-for-representation-learning-2/

Posterior collapse

- If we use a *really powerful* decoder $p_\theta(x \mid z)$:
- Can in practice get great samples. . . that tend to ignore z entirely



https://www.inference.vc/maximum-likelihood-for-representation-learning-2/

- Remember $\mathrm{ELBO}_{\theta,\phi}(x) = \mathbb{E}_{z \sim q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] - \mathrm{KL}(q_\phi(z \mid x) \parallel p(z))$
  - If $p_\theta(x \mid z)$ ignores z, $q_\phi(z \mid x)$ can be just $p_\theta(z)$ and KL becomes 0
  - Pretty good local max where z is totally useless

Representation Learning with VAEs

- Maximizing the ELBO isn't *just* MLE...

$$\max_\phi \sum_i \mathrm{ELBO}_{\theta,\phi}(x^{(i)}) = \log p_\theta(\mathbf{X}) - \min_\phi \sum_i \mathrm{KL}(q_\phi(z^{(i)} \mid x^{(i)}) \parallel p_\theta(z^{(i)} \mid x^{(i)}))$$

  - If ϕ is perfect, it's just the MLE
  - Otherwise, we prefer the kinds of distributions that q_ϕ can successfully reconstruct

- And training a VAE isn't just minimizing the ELBO
  - We don't find the actual maximizer in this architecture; we run SGD
  - The implicit bias of the SGD training procedure likely plays a very important role
  - Likely even more true for complex models, e.g. transformer-based

VQ-VAE

bonus!

- vector quantized VAE has discrete latent space, avoids(ish) posterior collapse
- Encoder maps to a single discrete value of the latent; learn a prior on them
- Autoregressive decoder is encouraged to “commit” to a latent

- VQ-VAE-2 uses hierarchical latents
  - Autoregressive prior on the latents, but a fast feed-forward decoder

Figure 1: Class-conditional 256x256 image samples from a two-level model trained on ImageNet.

https://arxiv.org/pdf/1906.00446.pdf

Summary

- Transposed convolutions for going from low dimensions to high
  - Fully-convolution networks for pixel-level labeling
  - Default architectural basis for VAE decoder on images
- Representation learning: sometimes a useful scheme
  - Autoencoders: non-probabilistic version of VAEs
  - Whether VAEs/etc give useful representations: sometimes, but it's tricky!
- Next up: the next thing in sequence, sequential data

β-VAE

bonus!

- Put a weight $\beta > 1$ in front of the KL term in the ELBO

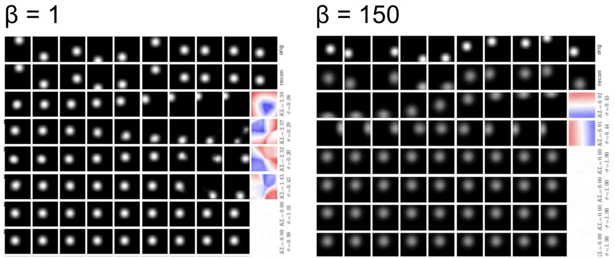


Figure 2: **Entangled versus disentangled representations of positional factors of variation learnt by a standard VAE ($\beta = 1$) and β-VAE ($\beta = 150$) respectively.** The dataset consists of Gaussian blobs presented in various locations on a black canvas. Top row: original images. Second row: the corresponding reconstructions. Remaining rows: latent traversals ordered by their average KL divergence with the prior (high to low). To generate the traversals, we initialise the latent representation by inferring it from a seed image (left data sample), then traverse a single latent dimension (in $[-3, 3]$), whilst holding the remaining latent dimensions fixed, and plot the resulting reconstruction. Heatmaps show the 2D position tuning of each latent unit, corresponding to the inferred mean values for each latent for given each possible 2D location of the blob (with peak blue, -3; white, 0; peak red, 3).

https://arxiv.org/pdf/1804.03599.pdf

- Refined version: see TC-VAE

Wasserstein Auto-Encoder

bonus!

- Different framing for an auto-encoder-based generative model
- Avoids "motivation" for posterior collapse
- Simple version with deterministic encoder/decoder:

$$\min_{\theta,\phi} \frac{1}{n}\sum_{i=1}^{n} \|x^i - \mathrm{dec}_\theta(\mathrm{enc}_\phi(x^i))\|^2 + \lambda D\left(\text{prior}(z), \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\left(z = \mathrm{enc}_\phi(x^i)\right)\right)$$

  where D is some distance between probability distributions (kernel MMD, GAN)
- Only makes marginal distribution of zs match the prior, not each one like VAEs
- Can show approximately minimizes Wasserstein distance between model and data