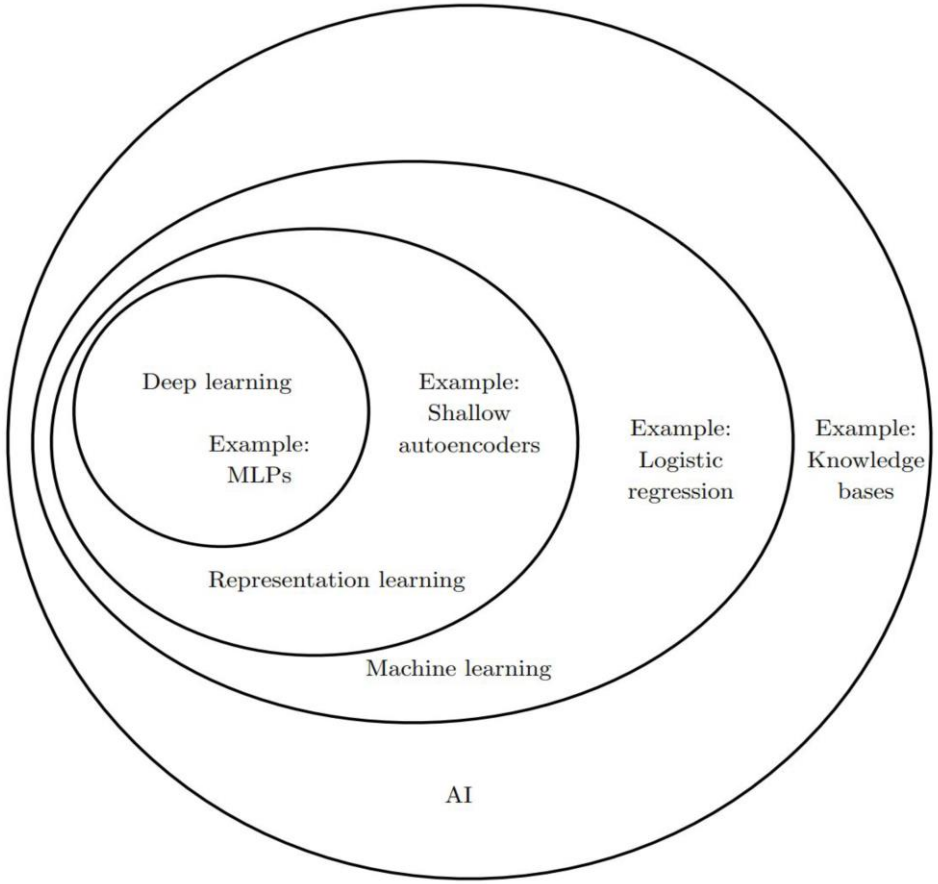
Representation learning

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

Deep Learning is one way to learn features

- Finding all patterns = short description of raw data that can be used in downstream task
  - Recipe is clear: Collect a large labeled dataset, train a model, deploy. Good data and sufficient data are what we need
    - Unfortunately, having plenty of unlabeled data and little labeled data is common. Labeling instances is time-consuming and costly
  - Deep Unsupervised Learning - to learn representations without labels



Why do we learn representation using unsupervised way?

- Expense of producing a new dataset for each task
  - Prepare labeling manuals, categories, hiring humans, creating GUIs, storage pipelines, etc.
- Good supervision may not be cheap (ex: medicine, legal)
  - Take advantage of vast amount of unlabeled data on the internet (images, videos, language)
- Cognitive motivation: How animals / babies learn

> "The brain has about 10^{14} synapses and we only live for about 10^{9} seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning" - Geoffrey Hinton

- Many powerful applications
  - Compression, improve any downstream task, generate novel data, conditional synthesis technology etc.

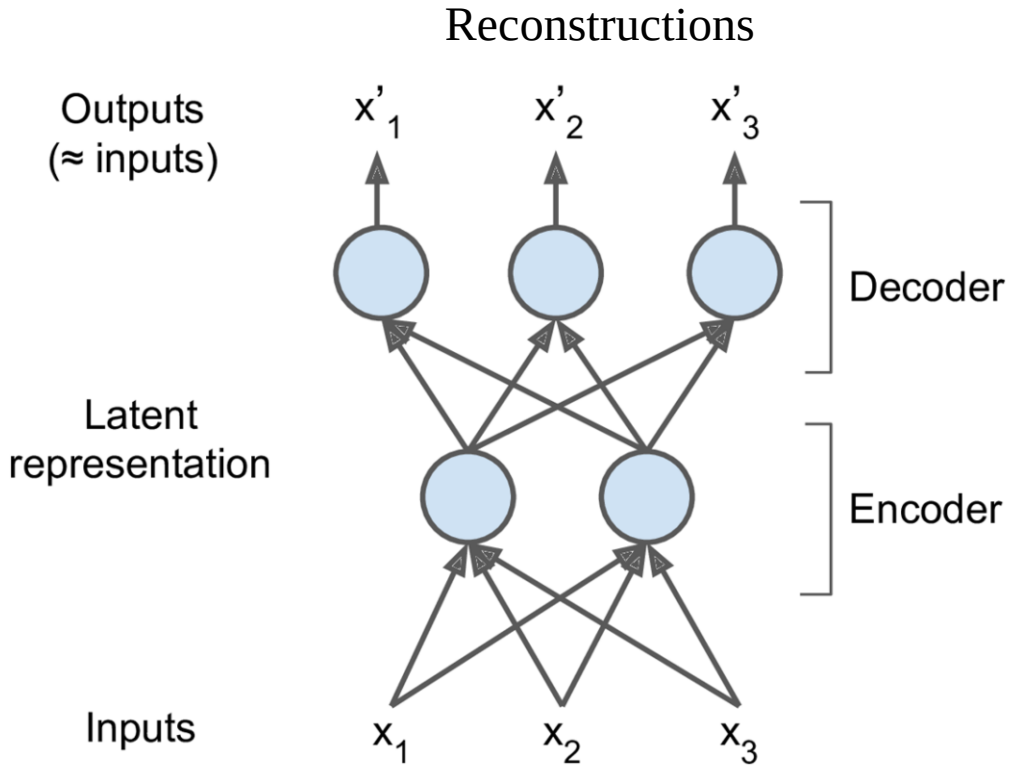
Deep Unsupervised Learning - Generative Learning

- Autoencoders are networks capable of learning dense representations of the input data, called *latent representations*, *embedding or codings*, without any supervision
  - Codings typically have lower dimensionality than the input data
  - Acts as feature detectors used for unsupervised pretraining of deep neural networks
  - Some are generative models
- Generative adversarial networks (GANs) are more powerful in generating realistic data
  - Augmenting a dataset
  - Colorization and image editing
  - Super resolution (increasing the resolution of an image)
  - Generating text, audios, time series…

Efficient data representation is a kind of compression

- Pattern detection
  - 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
  - 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14

- Chess players can memorize the positions of all the pieces in a game by looking at the board for just five seconds
  - This was only the case when the pieces were placed in realistic. They see chess patterns more easily



1. Linear autoencoder are similar to PCA

- Recall that if the data matrix X is column-centered. We have

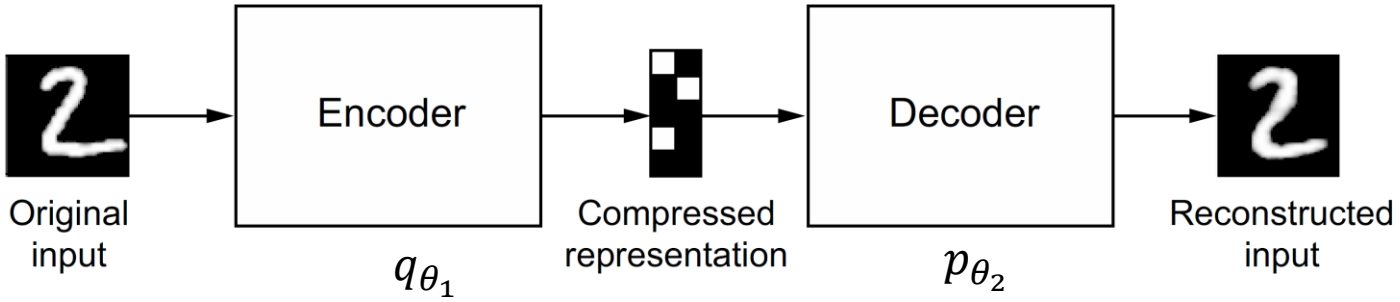
$$\min_{Z\in R^{n\times M},\Phi\in R^{p\times M}}\{\sum_{j=1}^{p}\sum_{i=1}^{n}(x_{ij}-\sum_{m=1}^{M}z_{im}\Phi_{jm})^2\}$$

  - $\hat{Z}$ and $\hat{\Phi}$ are in fact the first M principal components score (coding) and loading vectors (parameters)

- Auto encoder has similar loss

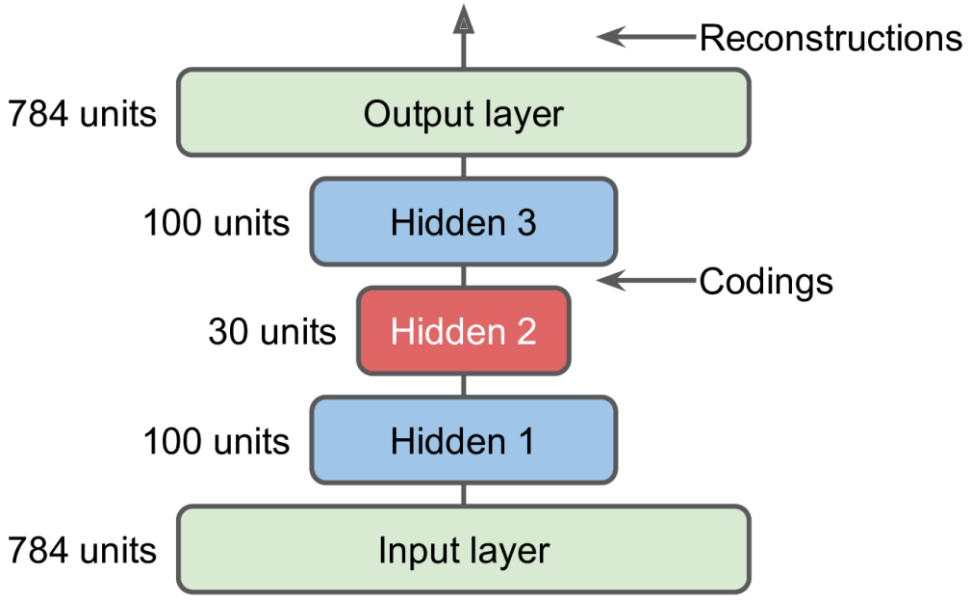
$$\min_{\theta_1\theta_2}\{\sum_{i=1}^{n}(x_i-p_{\theta_2}(q_{\theta_1}(x_i)))^2\}$$

  - Note that it can also use nonlinear activation function



Stacked autoencoders

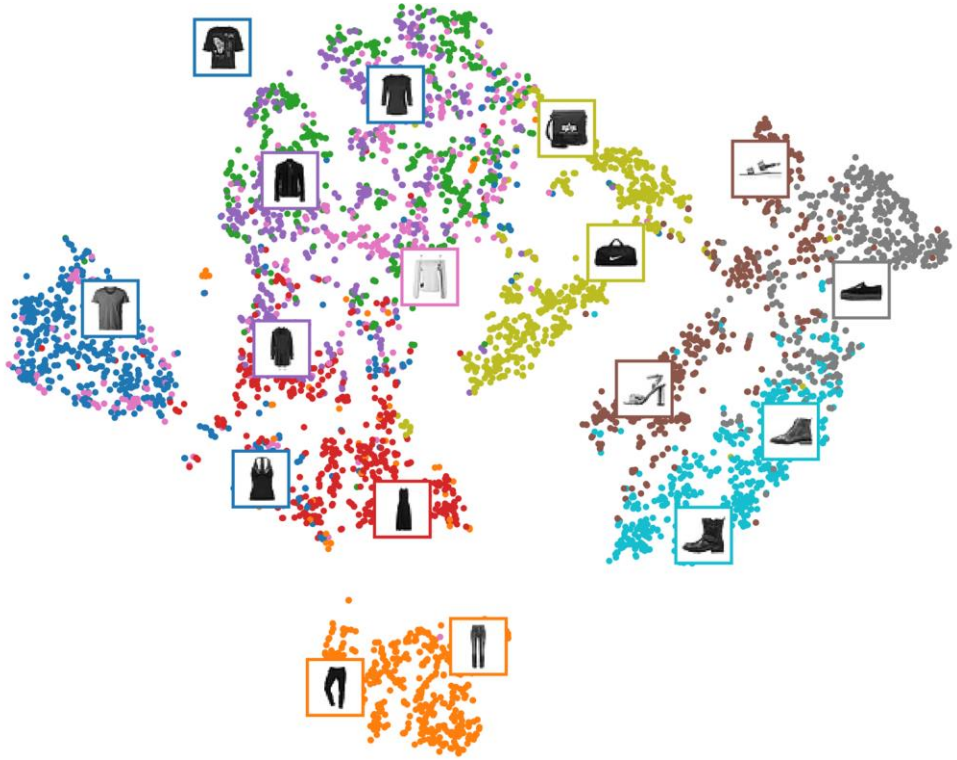
- Autoencoders can have multiple hidden layers called Stacked Autoencoders
  - Be careful not to make the autoencoder too powerful. An encoder that just learns to map each input to a single arbitrary number (decoder learns the reverse mapping) is not useful
  - The architecture is usually symmetric



Reconstructions of Fashion MNIST with MSE loss

Dimensional reduction and visualization

- Visualization the embedding (with the help of t-SNE)
  - Cope with large dataset
  - One strategy is to use an autoencoder to reduce the dimensionality down to a reasonable level, then use another dimensionality reduction algorithm for visualization

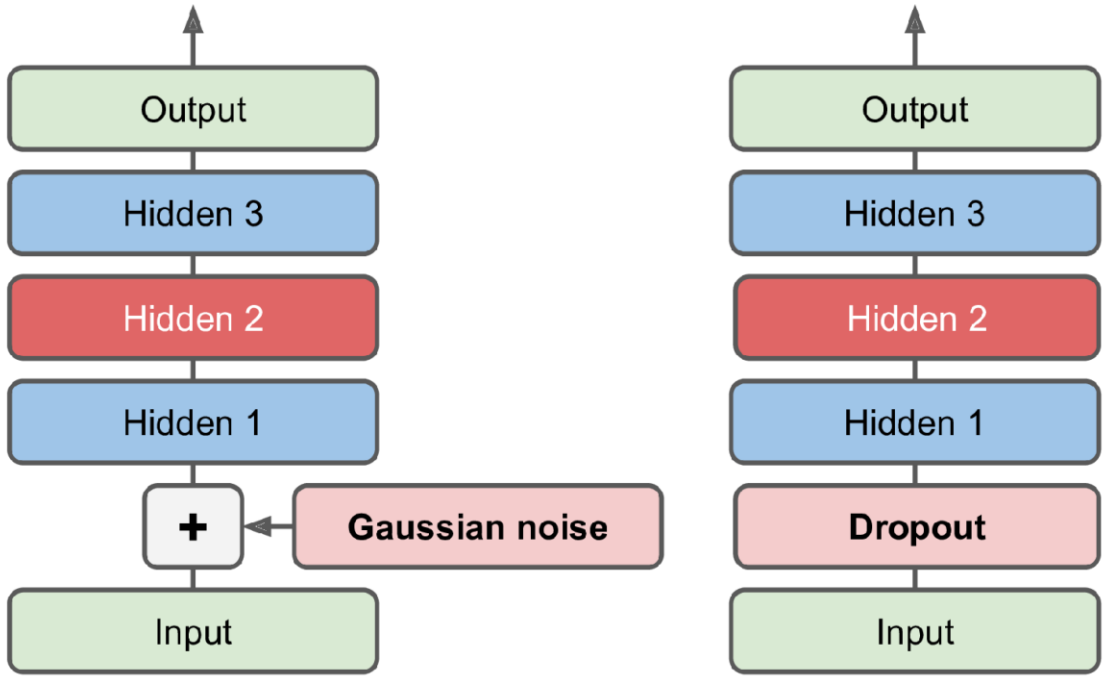


Convolutional/Recurerent Autoencoders

- If you are dealing with images, cnns are far better suited than dense networks to work with images
  - Use convolution layer in the encoder and transpose convolutional layers in the decoder
  - Again, try to use the pyramid design
- If you want to build an autoencoder for sequences, rnns may be better suited
  - The encoder is typically a sequence-to-vector RNN which compresses the input sequence down to a single vector. The decoder is a vector-to-sequence RNN that does the reverse

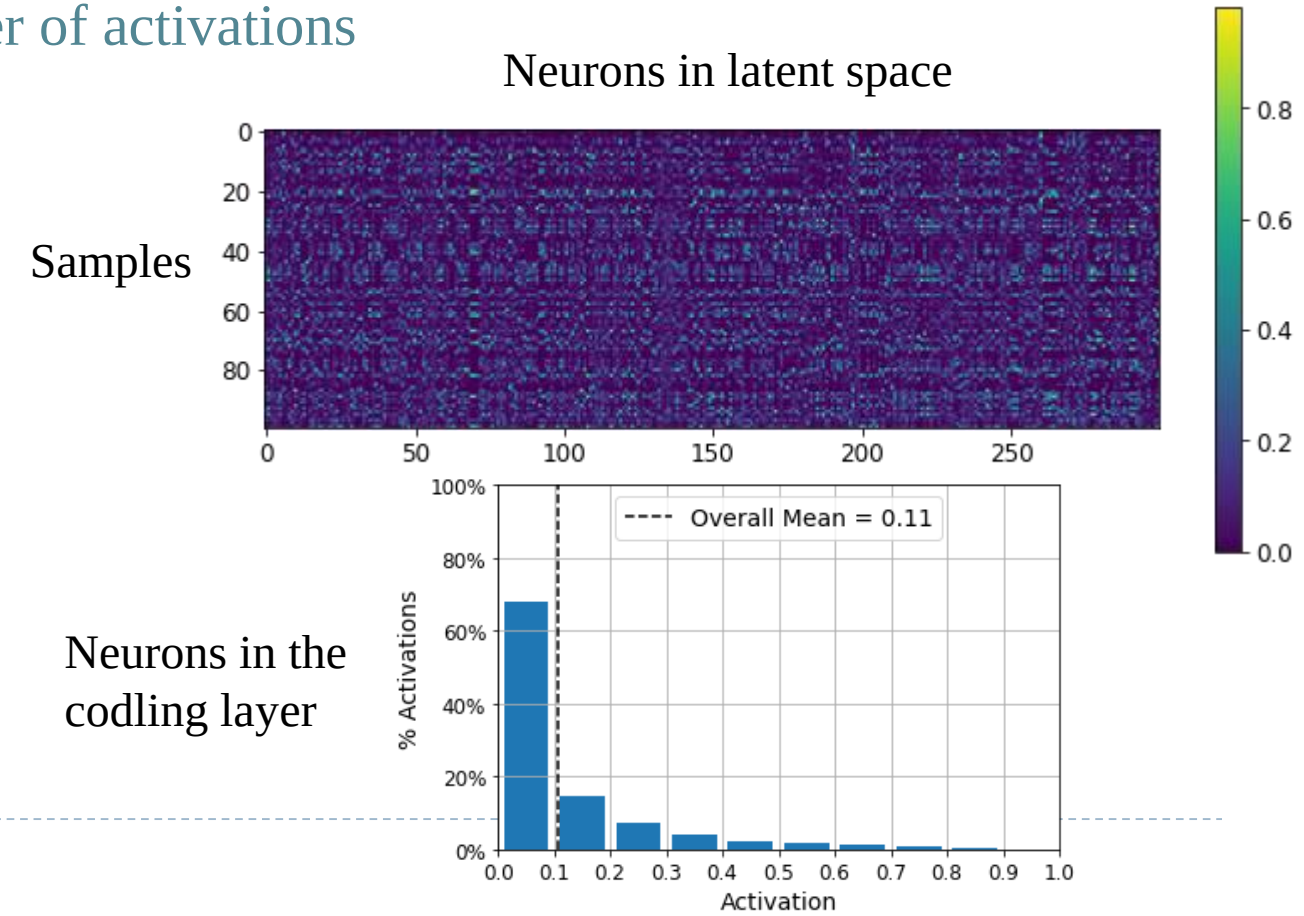
Denoising Autoencoders

- Another way to force the autoencoder to learn useful features is to *add noise* to its inputs, training it to recover the original, noise-free inputs
  - The noise can be pure Gaussian noise added to the inputs, or it can be randomly switched-off inputs, just like in dropout



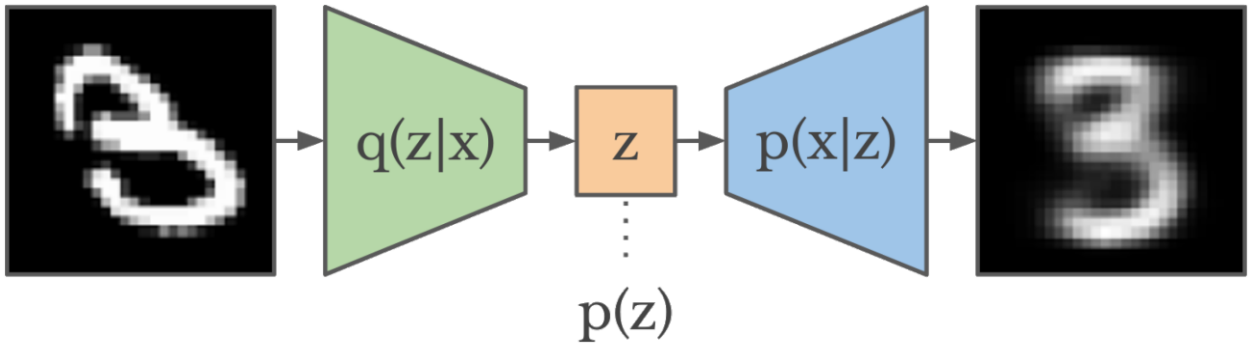
Sparse Autoencoders

- Another constraint that often leads to good *feature extraction is sparsity*
  - By adding an appropriate term to the cost function, the autoencoder is pushed to reduce the number of active neurons in the coding layer. This forces the autoencoder to represent each input as a combination of a small number of activations
  - In practice, you can use sigmod and L1 or L2 regularization in the coding layer
  - Or we can measure the actual sparsity of the coding layer at each training iteration, and penalize the model when the measured sparsity differs from a target sparsity



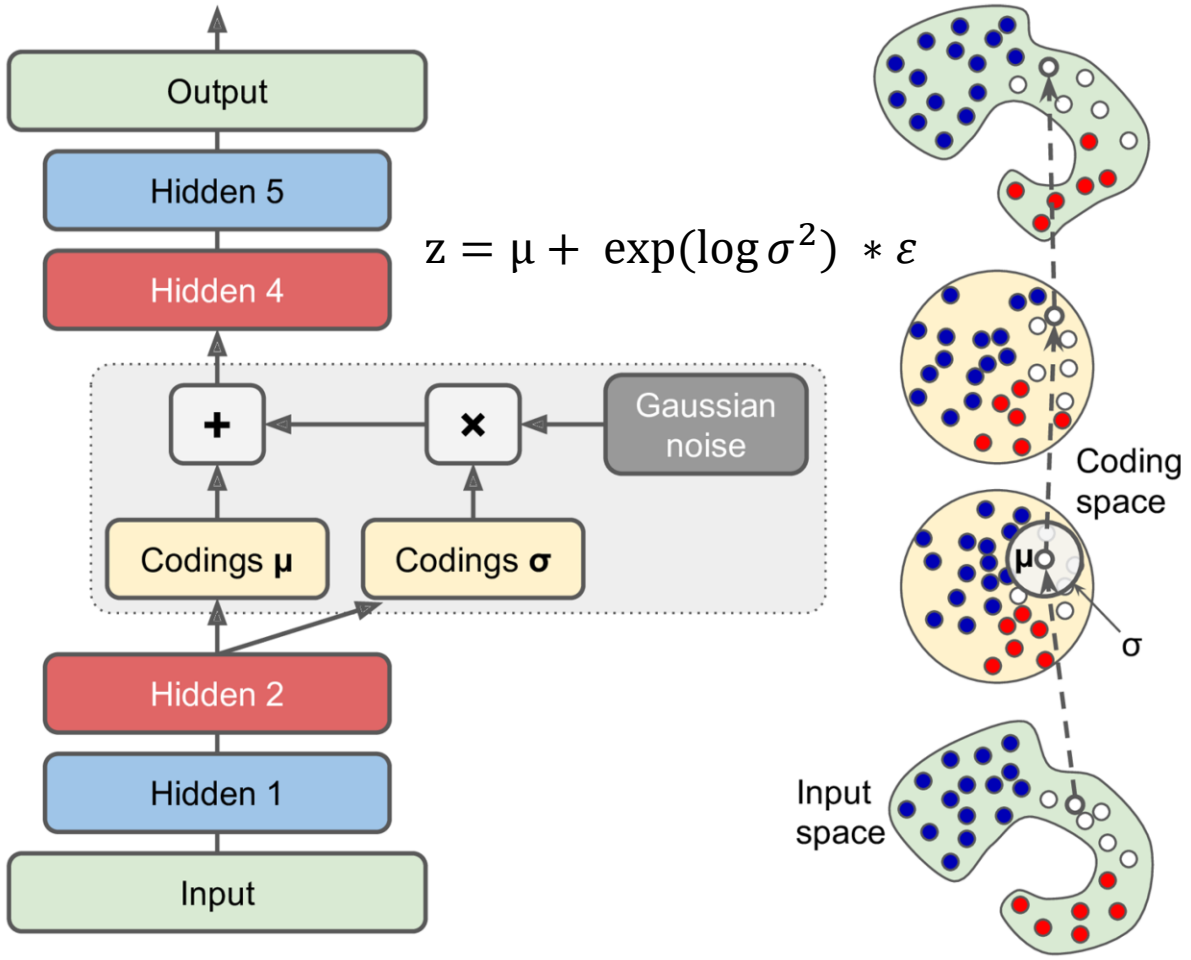
2. Variational Autoencoders (VAE)

- Properties of VAE
  1. They are *probabilistic autoencoders*, their outputs are partly determined by chance, even after training. They turns the input into the parameters of a statistical distribution: a mean and a variance
  2. They can learn smooth latent spaces
  3. They are generative autoencoders, meaning that they can generate new data that look like they were sampled from the training set
  4. It perform variational Bayesian inference , which is an efficient way to perform *approximate Bayesian inference (see appendix)*



Variational Autoencoders

- Instead of directly producing a coding for a given input, the encoder produces a mean coding μ and a variant of standard deviation $\log \sigma^2$ (for numerical stability)
  - The actual coding is then sampled randomly from a Gaussian distribution
  - One great consequence is that after training a variational autoencoder, you can very easily generate a new instance



Variational autoencoder

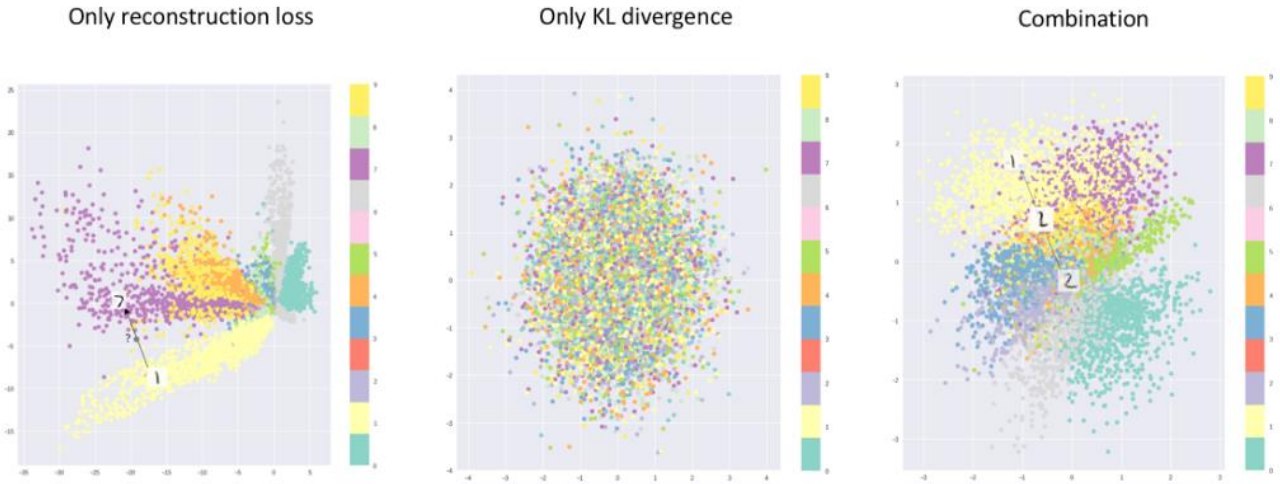
- The loss function

$$\mathcal{L}(x, \hat{x}) + \beta \sum_j KL\left(q_j(z|x) \,||N(0,1)\right)$$

Reconstruction loss

Regularization loss

MNIST Results



Penalizing reconstruction loss encourages the distribution to describe the input

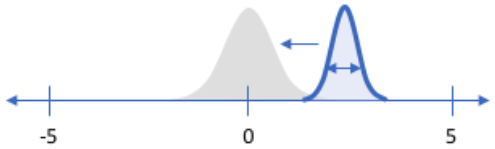
Without regularization, our network can "cheat" by learning narrow distributions

Penalizing KL divergence acts as a regularizing force

Our distribution deviates from the prior to describe some characteristic of the data

With a small enough variance, this distribution is effectively only representing a single value

Attract distribution to have zero mean



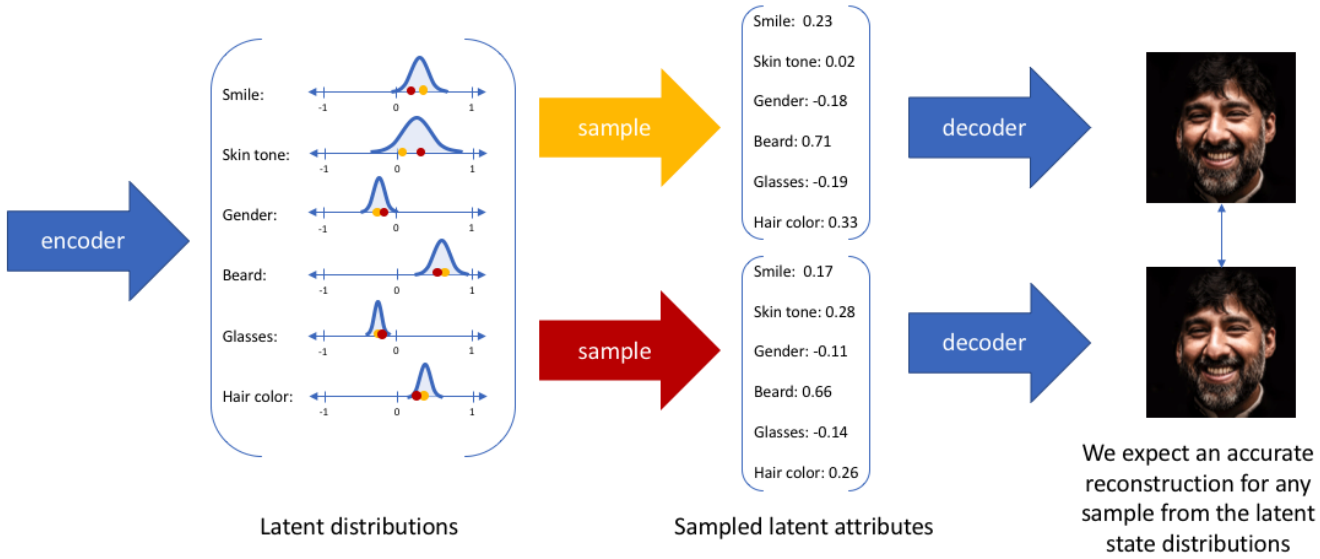
Ensure sufficient variance to yield a smooth latent space

Feature disentangle through VAE

- If we observe that the latent distributions appear to be very tight, we may decide to give higher weight to the KL divergence term with a parameter $\beta > 1$, encouraging the network to learn broader distributions

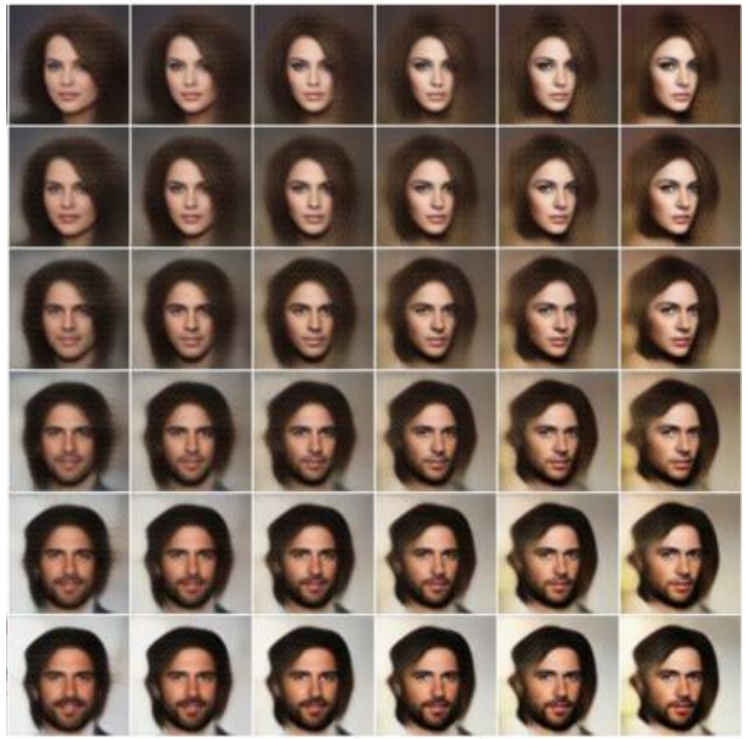
$$\mathcal{L}(x, \hat{x}) + \beta \sum_j KL(q_j(z|x) || N(0,1))$$

  - As it turns out, by placing a larger emphasis on the KL divergence term we're also implicitly enforcing that the learned latent dimensions are uncorrelated (through our simplifying assumption of a diagonal covariance matrix)
  - This is the idea of feature disentangle



Semantic interpolation through VAE

- Variational autoencoders make it possible to perform semantic interpolation: instead of interpolating two images at the pixel level (which would look as if the two images were overlaid), we can interpolate at the codings level.
  - We first run both images through the encoder, then we interpolate the two codings we get, and finally we decode the interpolated codings to get the final image
  - Certain directions in the space may *encode interesting axes of variation* in the original data



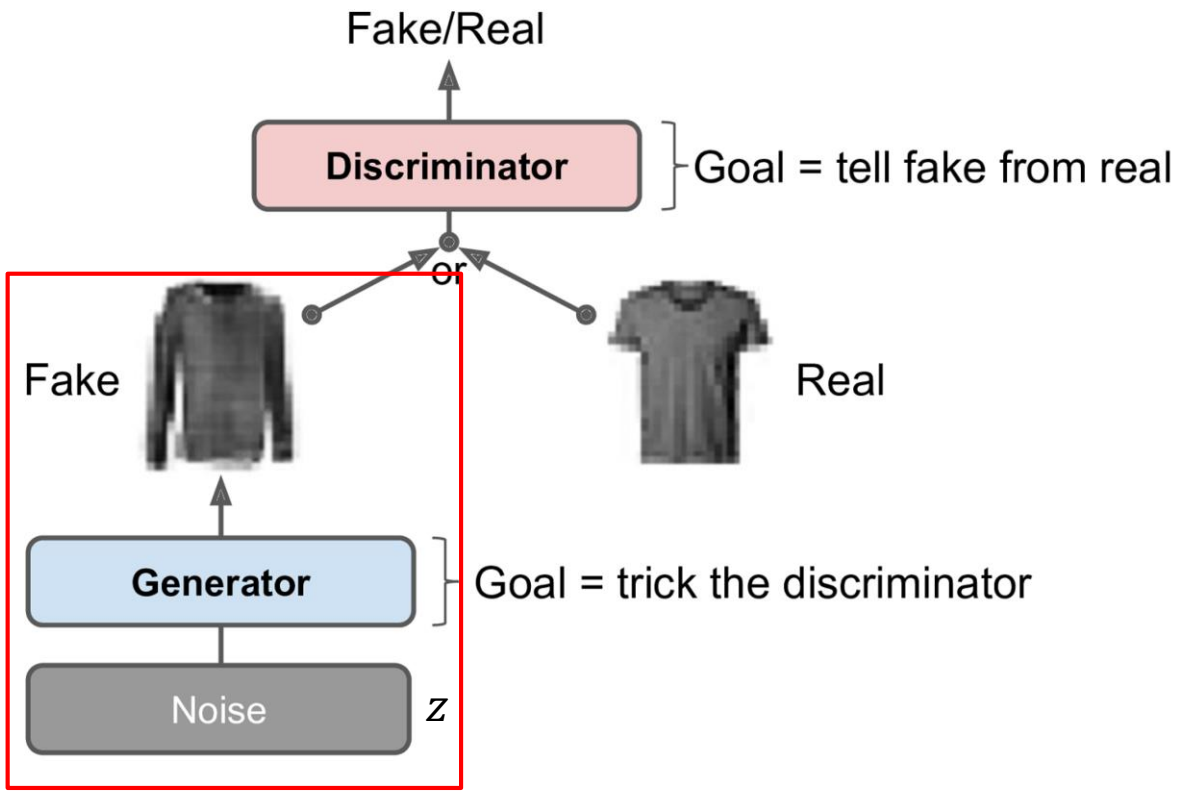
https://medium.com/vitrox-publication/generative-modeling-with-variational-auto-encoder-vae-fc449be9890e

Generative adversarial networks

- Generative adversarial networks (GANs), introduced in 2014 by Goodfellow et al. enable the generation of fairly realistic synthetic images by forcing the generated images to be statistically almost indistinguishable from real ones
  - An intuitive way to understand GANs is to imagine a forger trying to create a fake Picasso painting. At first, the forger is pretty bad at the task. He mixes some of his fakes with authentic Picassos and shows them all to an art dealer
  1. The art dealer makes an authenticity assessment for each painting and gives the forger feedback about what makes a Picasso look like a Picasso
  2. The forger goes back to his studio to prepare some new fake
  3. As time goes on, the forger becomes increasingly competent at imitating the style of Picasso, and the art dealer becomes increasingly expert at spotting fakes. In the end, they have on their hands some excellent fake Picassos

3. Generative Adversarial Networks

- Generator (forger)
  - Takes a random distribution as input (typically Gaussian) and outputs some data. You can think of the random inputs as the latent representations of the image to be generated
  - The generator offers the same functionality as a decoder in a variational autoencoder
- Discriminator (art dealer)
  - Takes either a fake image from the generator or a real image from the training set as input, and must guess whether the input image is fake or real



Training GAN

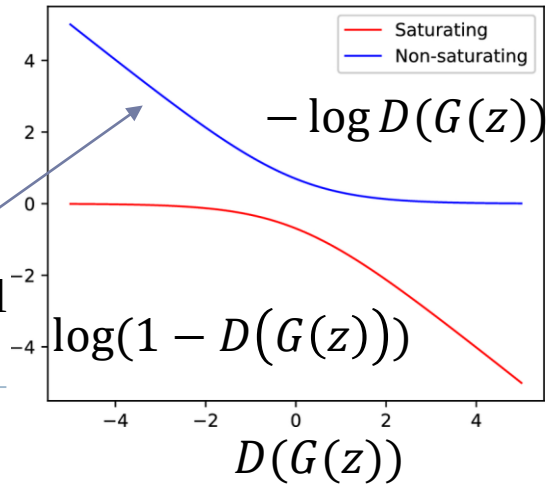
- In the first phase, we train the discriminator (generator are fixed)
  - A batch of real images is sampled from the training set and is completed with an equal number of fake images produced by the generator. *The labels are set to 0 for fake images and 1 for real images*, and the discriminator is trained using the binary cross-entropy loss

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- In the second phase, we train the generator (discriminator are fixed)
  - We first use it to produce another batch of fake images, and once again the discriminator is used to tell whether the images are fake or real. This time all the labels are set to 1 (real). The loss on the right hand side could have larger gradient for bad sample

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \Rightarrow \max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Higher gradient signal



Training GAN

- That's what a GAN is: a forger network and an expert network, each being trained to beat the other
  - Unlike VAEs, this latent space has fewer explicit guarantees of meaningful structure; in particular, it isn't continuous
  - Remarkably, a GAN is a system where the optimization minimum isn't fixed. Normally, gradient descent consists of rolling down hills in a static loss landscape. But with a GAN, *every step taken down the hill changes the entire landscape a little*. It's a dynamic system where the optimization process is seeking not a minimum, but an equilibrium between two forces. For this reason, GANs are notoriously difficult to train

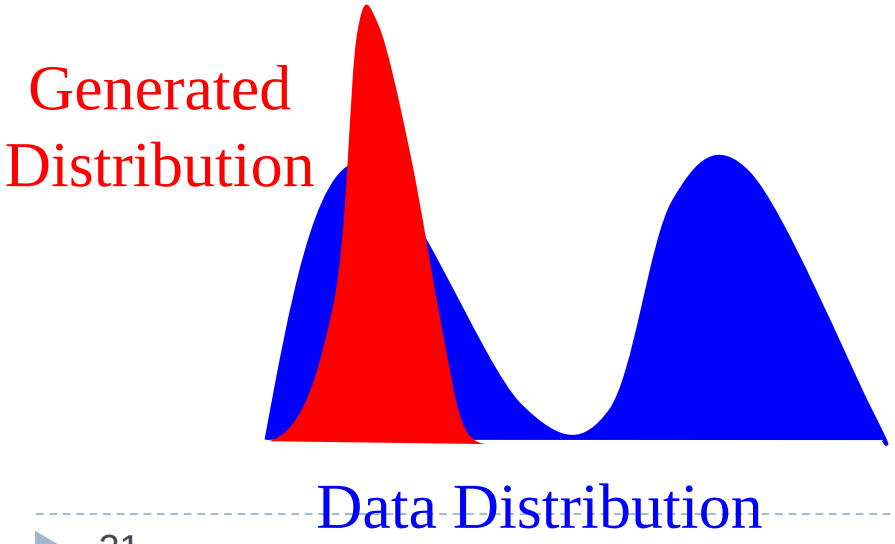
Minimax objective function:

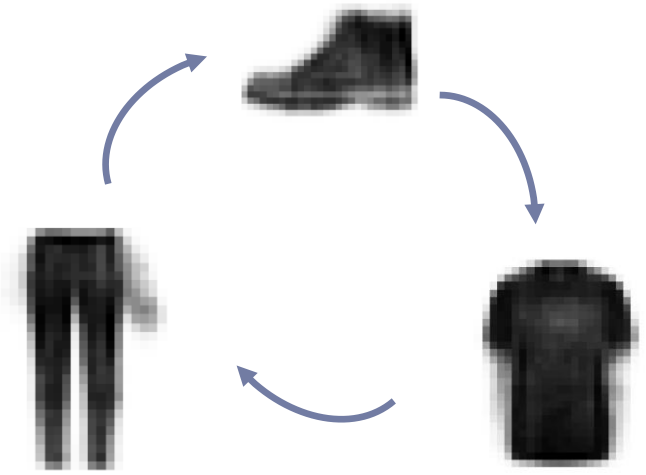
Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

Training GAN

- The biggest difficulty is called *mode collapse*: this is when the generator's outputs gradually become less diverse
  - Suppose that the generator gets better at producing convincing shoes than any other class. It will fool the discriminator a bit more with shoes, and this will encourage it to produce even more images of shoes. Gradually, it will forget how to produce anything else. Meanwhile, the only fake images that the discriminator will see will be shoes, so it will also forget how to discriminate fake images of other classes





Training GAN

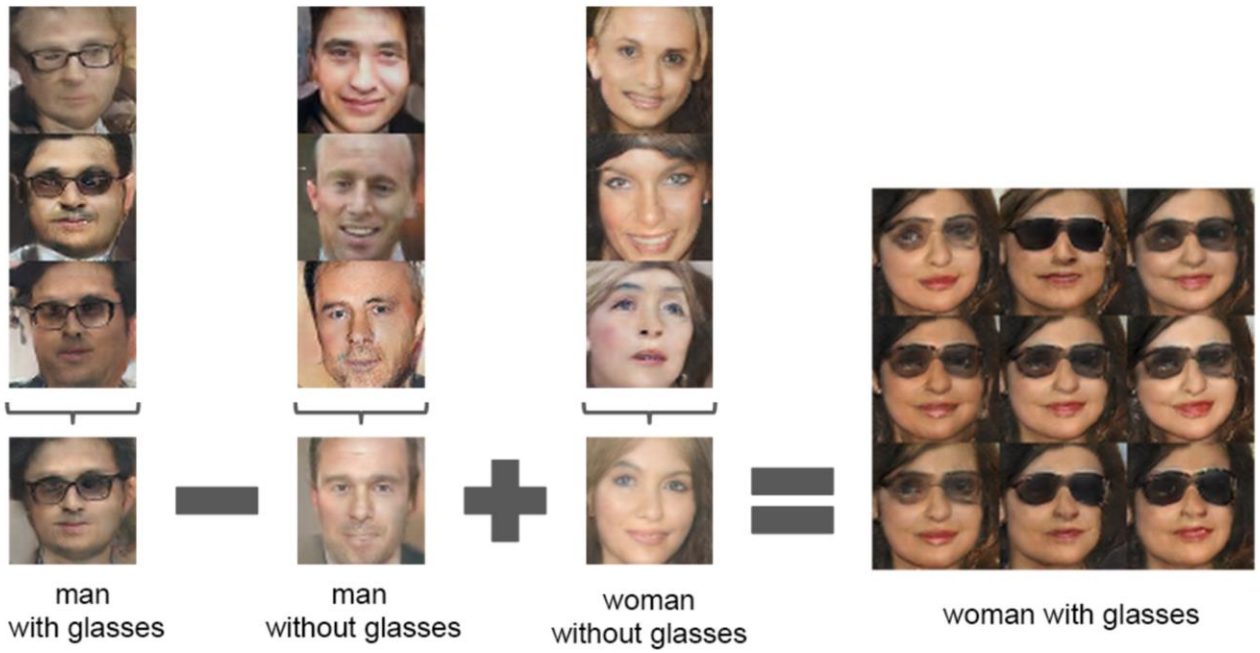
- Proposing new cost functions and architecture
- Experience replay
  - Storing the images produced by the generator at each iteration in a replay buffer (gradually dropping older generated images)
  - Train the discriminator using real images plus fake images drawn from this buffer (rather than just fake images produced by the current generator)
  - This reduces the chances that the discriminator will overfit the latest generator's outputs
- Mini-batch discrimination
  - Measures how similar images are across the batch and provides this statistic to the discriminator, so it can easily reject a whole batch of fake images that lack diversity
  - Encourages the generator to produce a greater variety of images

Training GAN

- More tricks
  - Sample points from the latent space using a normal distribution (Gaussian distribution), not a uniform distribution
  - Use strided convolutions instead of pooling for downsampling feature maps in the discriminator
  - Try to use LeakyReLU instead of a ReLU since sparse gradients can hinder GAN training
  - Adding random noise to the labels for the discriminator since stochasticity is good for inducing robustness. Because GAN training results in a dynamic equilibrium, GANs are likely to get stuck in all sorts of ways
  - In generated images, it's common to see checkerboard artifacts caused by unequal coverage of the pixel space in the generator. To fix this, use a kernel size that's divisible by the stride size whenever we use a strided convolution
  - https://github.com/soumith/ganhacks

Applications – Deep Convolutional GAN

- Vector arithmetic, semantic interpolation and feature distangle- DCGANs can learn quite meaningful latent representations



Applications – Conditional GAN (Text to Image)

It is a distribution

c: train

Prior distribution z

G

Image

$x = G(c, z)$

dropout

x is realistic or not

x

D (type 1)

scalar

Positive example:

Negative example: Image

x is realistic or not +
c and x are matched or not

c

x

D (type 2)

scalar

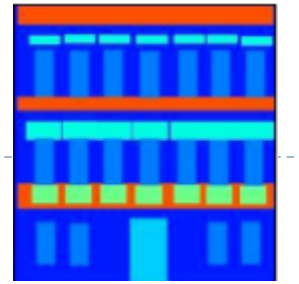
Positive example: (train , )

Negative example:

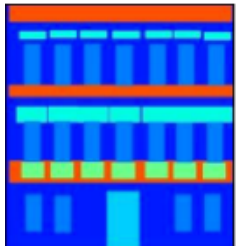
(train , Image) (cat , )

https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/gan_v10.pptx

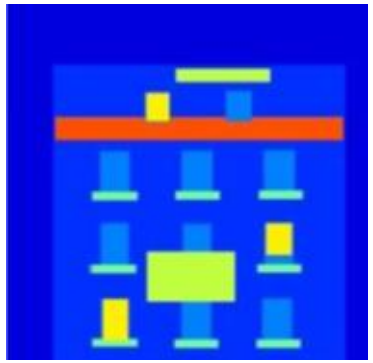
Applications – Conditional GAN with Paired Data




- Traditional supervised approach



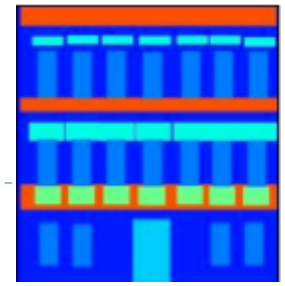
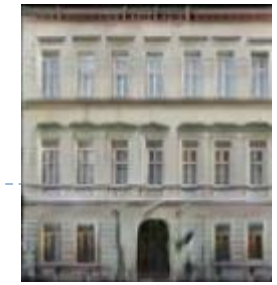
Testing:



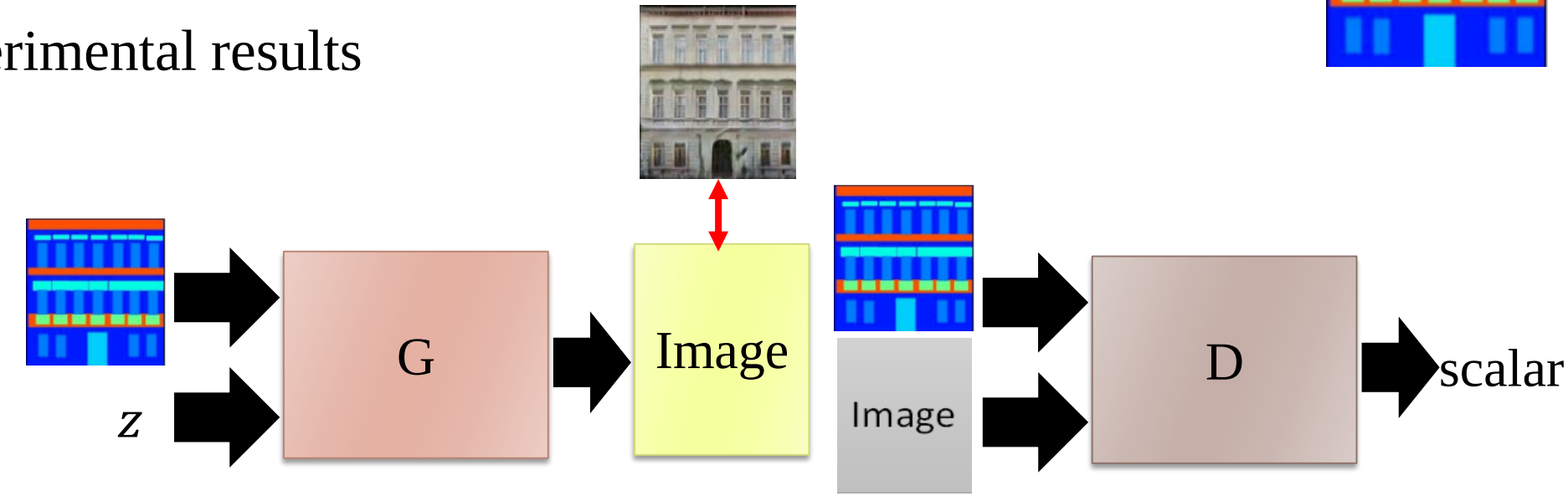

input close

It is blurry because it is the average of several images.

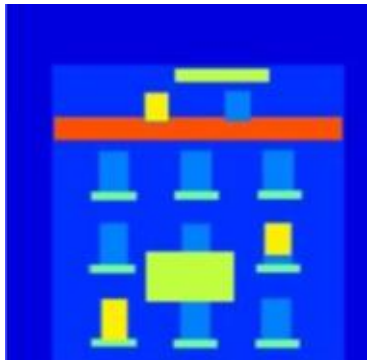
Applications – Conditional GAN with Paired Data

- Experimental results



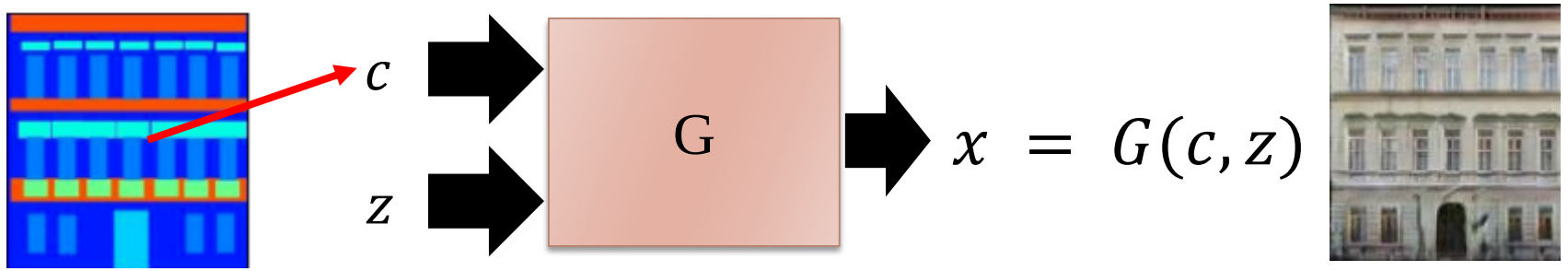
Testing:

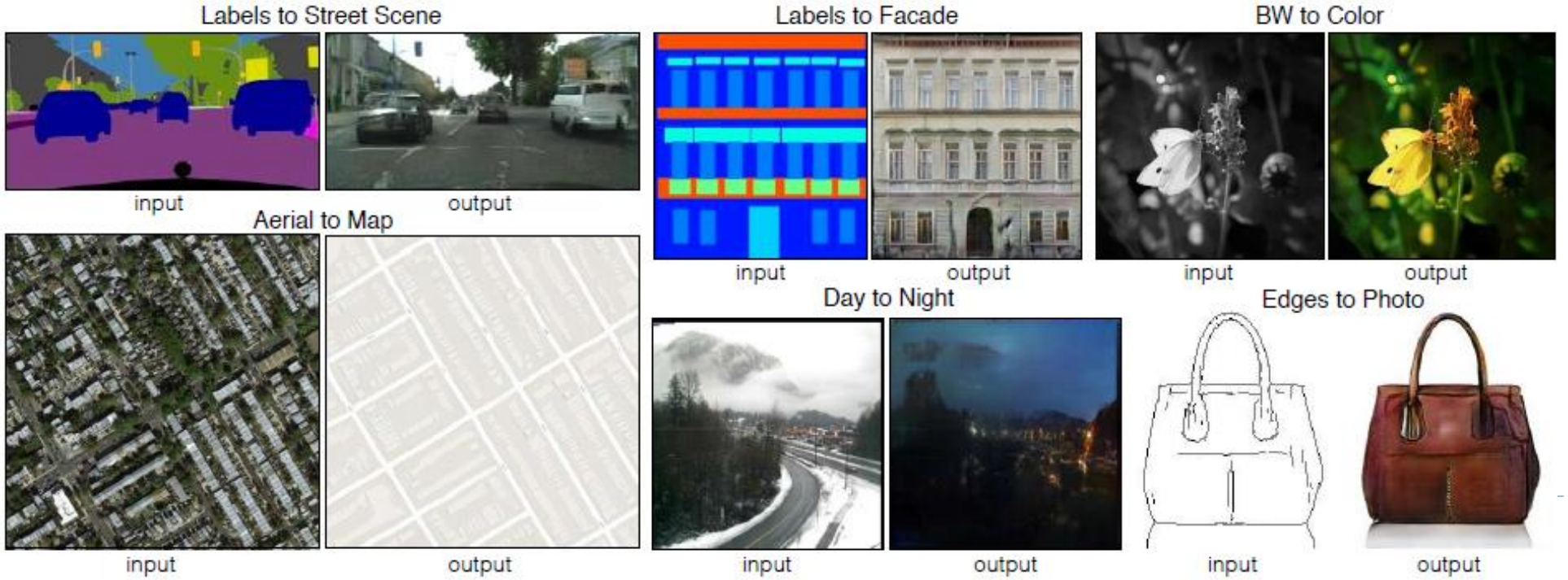



input close GAN GAN + close

Applications – Conditional GAN with Paired Data (pix2pix)

- Application with unsupervised learning is also possible

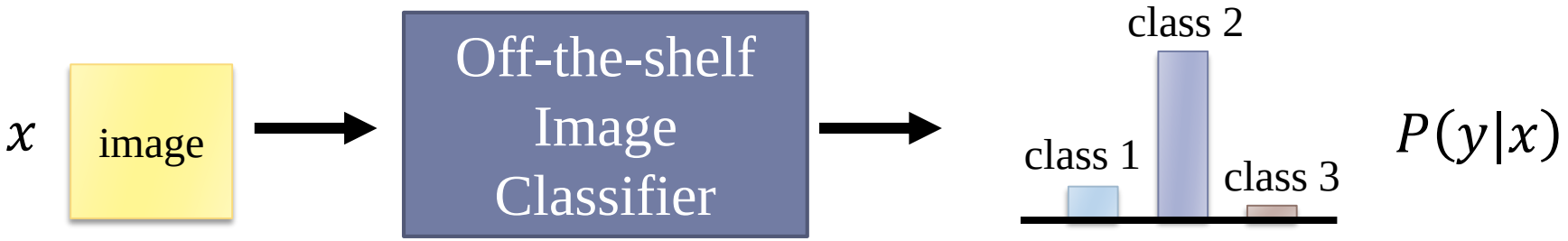




Quality of Image

- Human evaluation is expensive (and sometimes unfair/unstable).
- How to evaluate the quality of the generated images automatically?

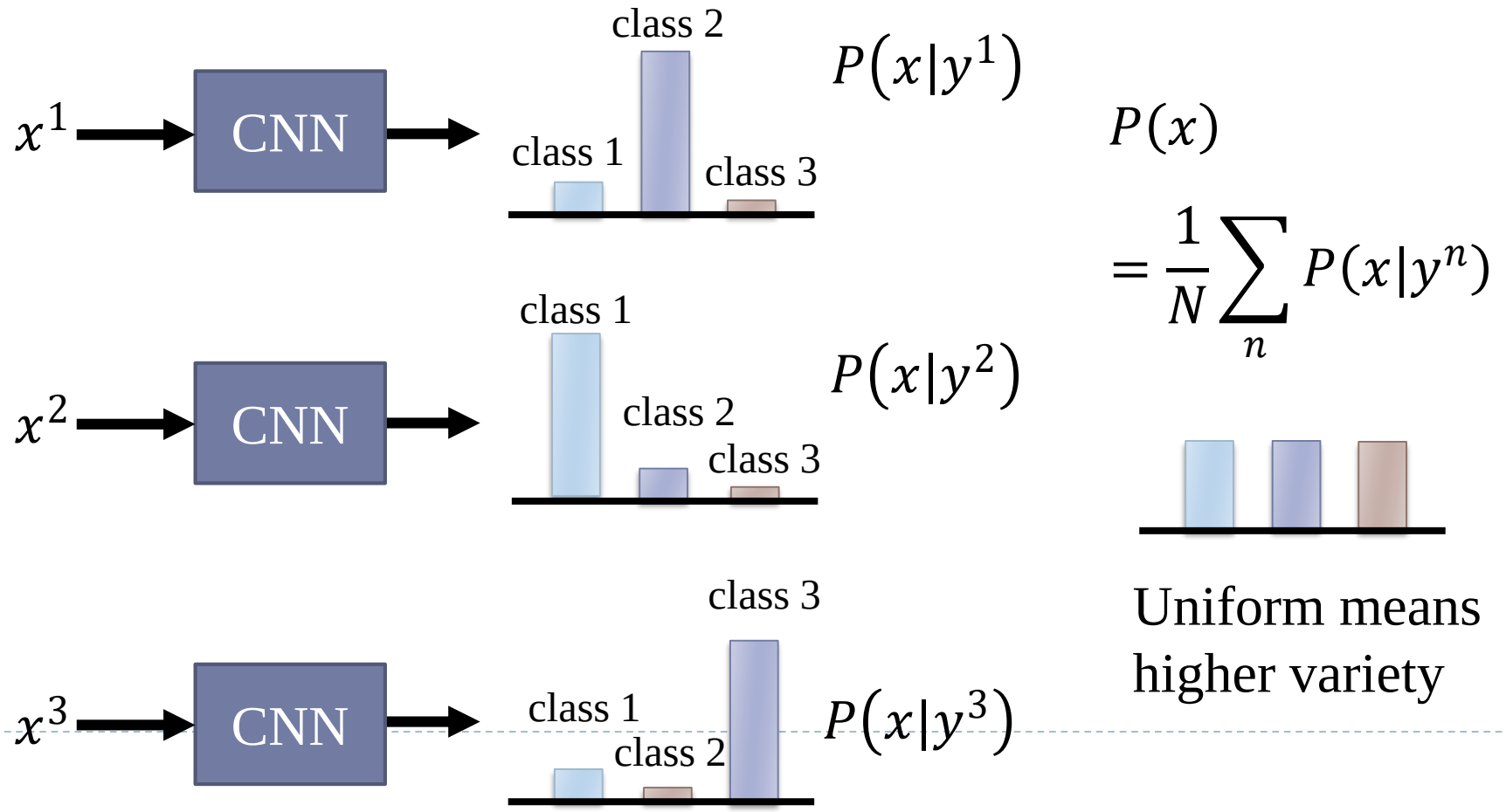
e.g., Inception net, VGG, etc.



Concentrated distribution means higher visual quality

Inception Score (IS)

- Good quality, large diversity → Large **IS**
  - Other popular evaluation metrics: Fréchet Inception Distance (FID)



Conclusion

- Deep representation learning is an actively developed field
- There are two major tools to do this: VAEs and GANs. VAEs result in highly structured, continuous latent representations and easier to train
- GANs enable the generation of realistic single-frame images but may not induce latent spaces with solid structure and high continuity. Notice that GAN can be trained in an unsupervised way

References

[1] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition Chapter 17

[2] Deep learning with Python, 2nd Edition Chapter 12

[3] https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php Lecture 6 and Lecture 8

Appendix

Resources

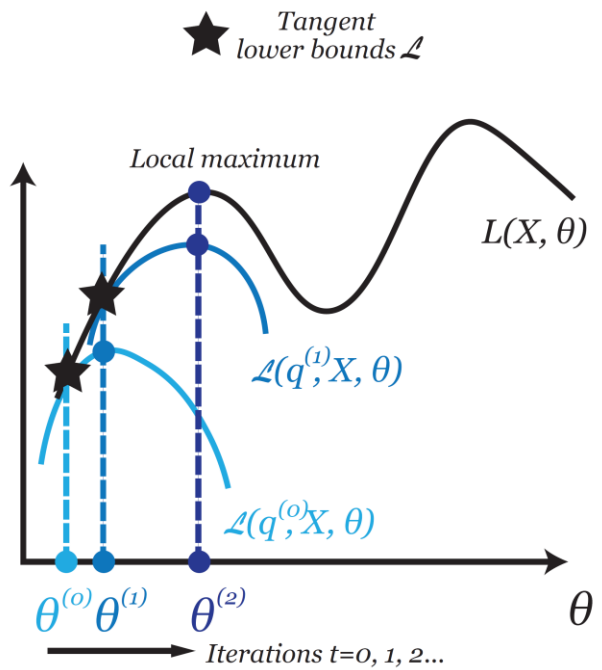
- Different VAE
  - https://github.com/probml/pyprobml/tree/master/vae
  - https://github.com/AntixK/PyTorch-VAE
- Different GAN
  - https://github.com/probml/pyprobml/tree/master/gan
- Tutorials
  - https://sites.google.com/view/berkeley-cs294-158-sp20/home
  - https://d2l.ai/chapter_natural-language-processing-pretraining/index.html
  - https://d2l.ai/chapter_generative-adversarial-networks/index.html

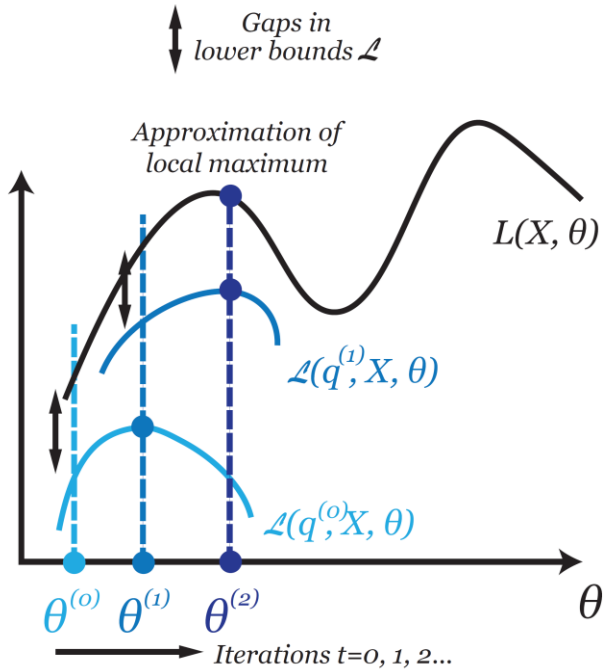
Inference

- Estimate the fix parameters θ that maximize the observed log-likelihood

$$L(X,\theta) = \log p_\theta(x_1,\ldots,x_n) = \sum_{i=1}^{n} \log p_\theta(x_i) = \sum_{i=1}^{n} \log \int_{h_i} p_\theta(x_i,h_i)dh_i$$

  - It is hard to explicitly find out $p_\theta(x_i)$. It is simpler to compute the full likelihood of each observation $p_\theta(x_i,h_i)$. However, the computation now becomes *intractable*
  - Distribution of H is inaccessible. Use a distribution q for hidden variables H, and maximizing a series of tractable lower-bounds $L(q,X,\theta)$ for $L(X,\theta)$ iteratively





https://arxiv.org/abs/2201.02867

Evidence Lower Bound (ELBO)

- $L(q, X, \theta)$ are found for the fact that for any distribution q_i on the variables h_i, the observed log-likelihood can be written as the sum of two terms

$$L(X,\theta) = \mathcal{L}(q,X,\theta) + \sum_{i=1}^{n} \mathrm{KL}(q_i(h_i) \parallel p_\theta(h_i|x_i)) = \sum_{i=1}^{n} \left[\mathcal{L}_i(q_i,x_i,\theta) + \mathrm{KL}(q_i(h_i) \parallel p_\theta(h_i|x_i))\right]$$

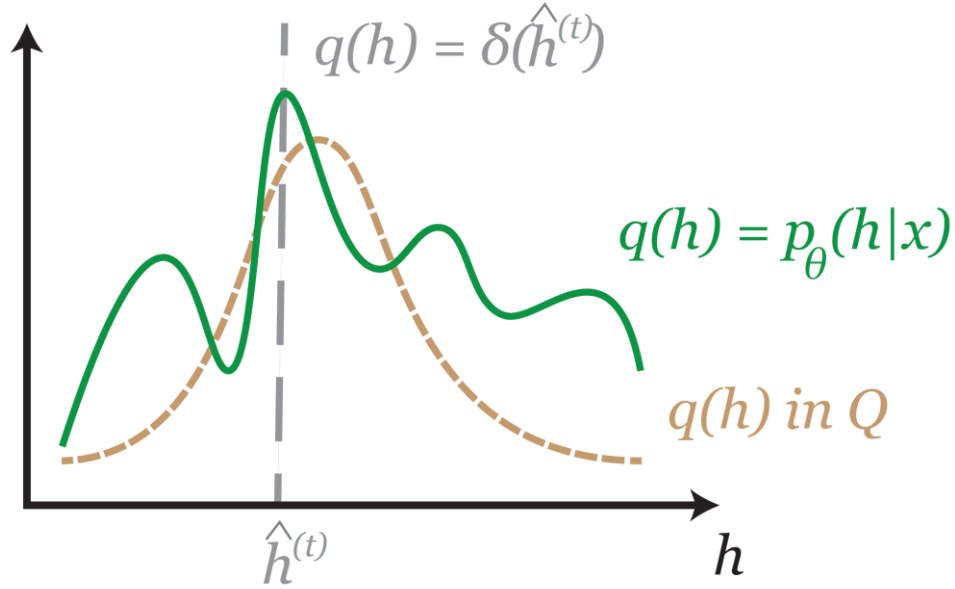
$$\mathcal{L}_i(q_i,x_i,\theta) = \int_{h_i} q_i(h_i) \log p_\theta(x_i|h_i) dh_i - \mathrm{KL}\left(q_i(h_i) \parallel p_\theta(h_i)\right)$$

- The Evidence Lower Bound is the function $\theta \to \mathcal{L}(q,X,\theta)$

$$\forall q \in \mathcal{Q}, \forall \theta, \quad \mathcal{L}(q,X,\theta) \leq L(X,\theta)$$

Inference Methods Based on an ELBO

- The choices of the distributions $q_i^{(t)}$ at each iteration t
  - Choosing $q_i^{(t)}(z) = p_{\theta^{(t)}}(h_i|x_i)$ makes the lower bound tangent to $L(X,\theta)$ at $\theta = \theta^{(t)}$ which is the Expectation-Maximization (EM) algorithm
  - For computational reasons, we can choose to have $q_i^{(t)}(z)$ approximate the posteriors $p_{\theta^{(t)}}(h_i|x_i)$ by
    - Their "mode", i.e. the value $\hat{h}_i$ of h_i that maximizes them: q_i effectively becomes a Dirac distribution at $\hat{h}_i$
    - A general distribution q_i within a family Q



Inference Methods Based on an ELBO

- The modal approximation in the EM algorithm speeds up the E-step; yet it has the drawback of summarizing the posterior distribution by a single estimate
  - Variational Inference (VI) (Variational Bayes) has appeared as a compromise between EM and modal EM during the computation of the E-step.
  - VI replaces the evaluation of the posterior of the latent variables, by an optimization converging to an approximation q of this posterior distribution. VI selects q from some parametric family of distributions Q, called the "variational family"
  - The variational family $Q = \{q_\eta = N(\mu, \sigma) | \eta = (\mu, \sigma)\}$ with parameters η is typically chosen to be a family of Gaussian distributions

Inference Methods Based on an ELBO

- The previous methods have the drawback of learning one (approximate) posterior for each hidden variable in h_i, and for each image i. This is computationally expensive and increase with n
  - Amortized inference (AI) collapses the n optimizations problems of the E-step into one
  - Instead of solving an optimization problem for each i and finding $\hat{h}_i$ or η_i defining q_{η_i}, AI optimizes the parameters ξ of a function Enc_ξ that predicts $\hat{h}_i$ or η_i when given x_i
  - The function Enc is traditionally called an encoder. Inference is made more tractable, but adds an additional error, called the amortization error

Inference Methods Based on an ELBO

- The main realizations of amortized variational EM in come through variational autoencoders

Amortized Variational EM as Dual Optimization Algorithm AI in the E-step of variational EM leads to an amortized variational EM:

(a) Inference on hidden variables h_i The parameters ξ parametrizing the encoder is computed via:

$$\xi^{(t)} = \arg\max_{\xi} \sum_{i=1}^{n} KL(q_{\text{Enc}_\xi(x_i)} || p_{\theta^{(t-1)}}(h_i|x_i) \tag{16}$$

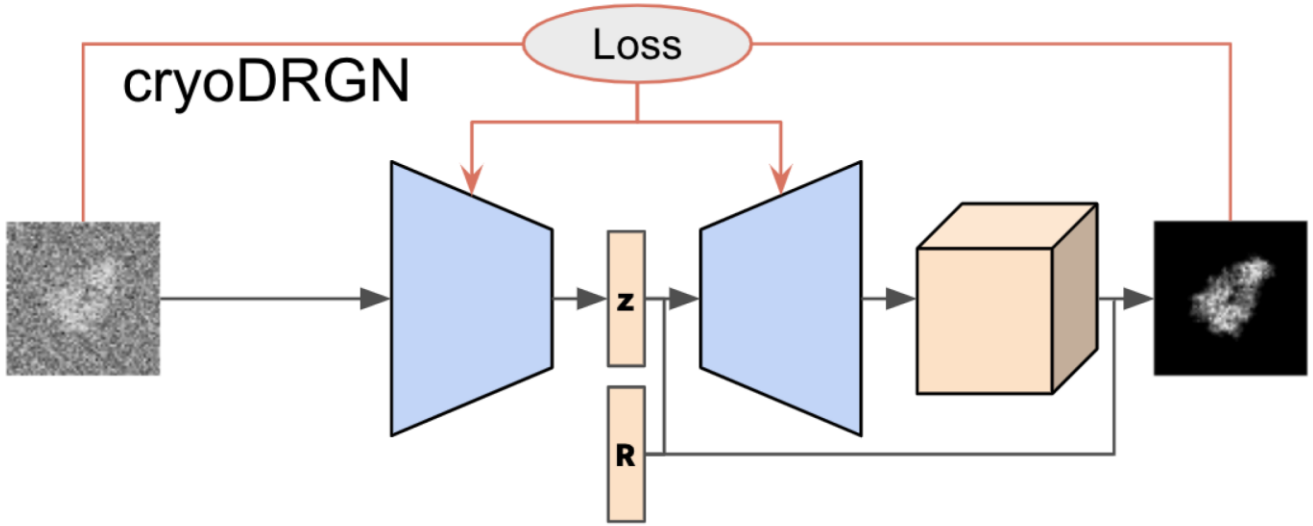
which generates n distributions $q_{\eta_i}^{(t)}$ parametrized by $\eta_i = \text{Enc}_{\xi^{(t)}}(x_i)$.

(b) Maximization on model's parameter θ The parameter θ is updated via:

$$\theta^{(t)} = \arg\max_{\theta} \mathcal{L}(q_{\eta_i}^{(t)}, \theta). \tag{17}$$

Inference Methods Based on an ELBO

- VAE
  - The parameters ξ and θ of the encoder and the decoder are typically estimated through stochastic gradient descent via backpropagation through the neural network. The loss function that is minimized gives rise to two optimizations which can be interpreted as extensions of the E-step and the M-step of the EM algorithm



Training GAN

for number of training iterations **do**

  for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

Some find k=1 more stable, others use k > 1, no best rule.

Recent work (e.g. Wasserstein GAN) alleviates this problem, better stability!

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for