Pattern Analysis & Machine Intelligence Praktikum: MLPR-SS21

Martin Mundt, Dr. Iuliia Pliushch, Prof. Dr. Visvanathan Ramesh

Goethe Uni Frankfurt

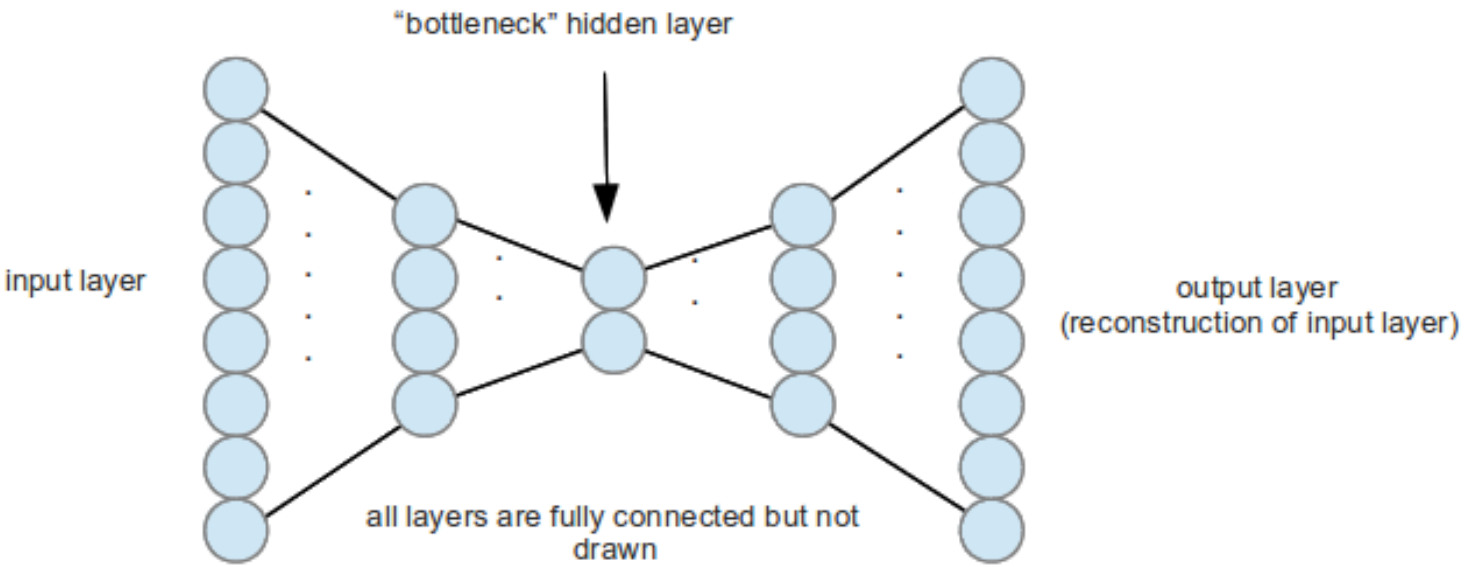
Week 08: Introduction to Unsupervised Neural Networks - Autoencoding

General concepts

- Make use of large quantities of **unlabeled data**
- Learn the **structure of the data**. Could be used for e.g. clustering (k-means, mixture models) for information retrieval, data compression, statistical data analysis etc.
- In NNs for **"Representation Learning"**: a meaningful & complete set of features describing the data.

Autoencoders

- Learn a meaningful representation ("encoding") of the complete data
- An **encoder** maps to a **"code"** or **"latent variables"/"latent representations"** and is then fed into a **decoder** to **reconstruct** the input.
- In the simplest version a decoder is a "flipped" version of the encoder (with shared weights).
- The loss function could be e.g. a mean squared error between the original image and the reconstructed (decompressed) version.



Linear Autoencoders and PCA

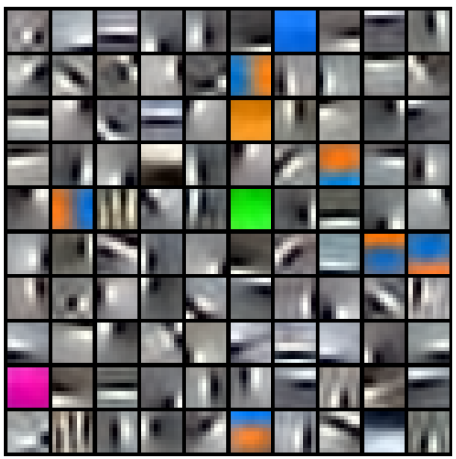
What's the relationship between Autoencoders and PCA?

- A single layer linear autoencoder with $\mathbf{h} = \mathbf{W_1 x} + \mathbf{b_1}$ and $\mathbf{x}' = \mathbf{W_2 h} + \mathbf{b_2}$ is said to apply PCA to the input, if the dimensionality of the embedding $\mathbf{W_1} \in \mathcal{R}^{n \times m}$ and $\mathbf{W_2} \in \mathcal{R}^{m \times n}$ is smaller than the input $n < m$.
- This is because the the autoencoder projects the data onto a low dimensional principal subspace. We can take the learned weight and apply singular value decomposition to recover the m vectors.
- No sorting of bottleneck output $\mathbf{h}$ in terms of variance, but SVD in the autoencoder case can be applied on a $n \times m$ matrix, whereas $\mathbf{x}$ is a $n \times n$ matrix.
- The main point of autoencoders is not to find a way to do PCA for big datasets, but we might receive a nice intuition. Practically, all our autoencoders will be nonlinear.

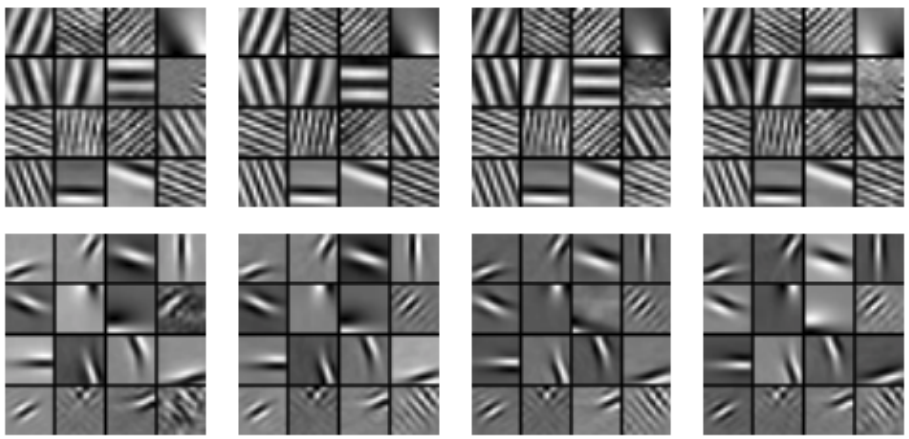
See https://arxiv.org/abs/1804.10253 for more information and exact relationships.

Autoencoders

- Can come in all sorts of flavors (fully-connected layers, convolutions etc.)
- Can be trained and stacked (greedy-layer-wise training)
- Can be extended to deep networks by adding more hidden layers in both encoder and decoder
- For large quantities of unlabeled data, can be used as pre-training for semi-supervised learning



https://www.groundai.com/project/zero-bias-autoencoders-and-the-benefits-of-co-adapting-features



https://arxiv.org/abs/1306.3162

Autoencoder semi-supervised pre-training

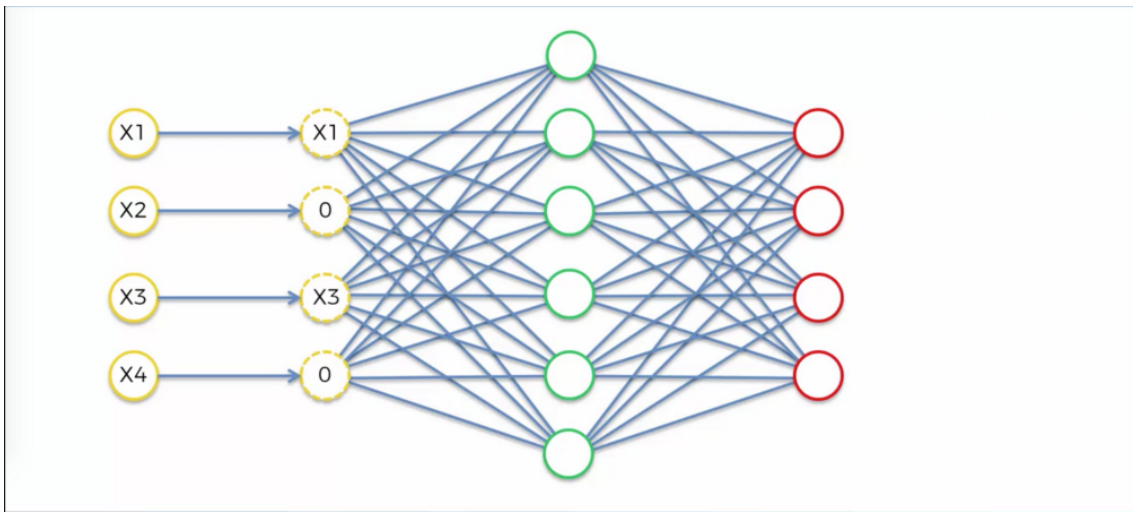
Given a large amount of unlabelled and smaller amount of data with labels:

1. Unsupervised training of autoencoder
2. "Detach" decoder / transfer encoder with pre-trained weights
3. Add a classifier (e.g. a single linear layer) to the pre-trained encoder
4. Fine-tune with the labelled data (mostly with a lower learning rate or even just the classifier's weights)

Denoising Autoencoders

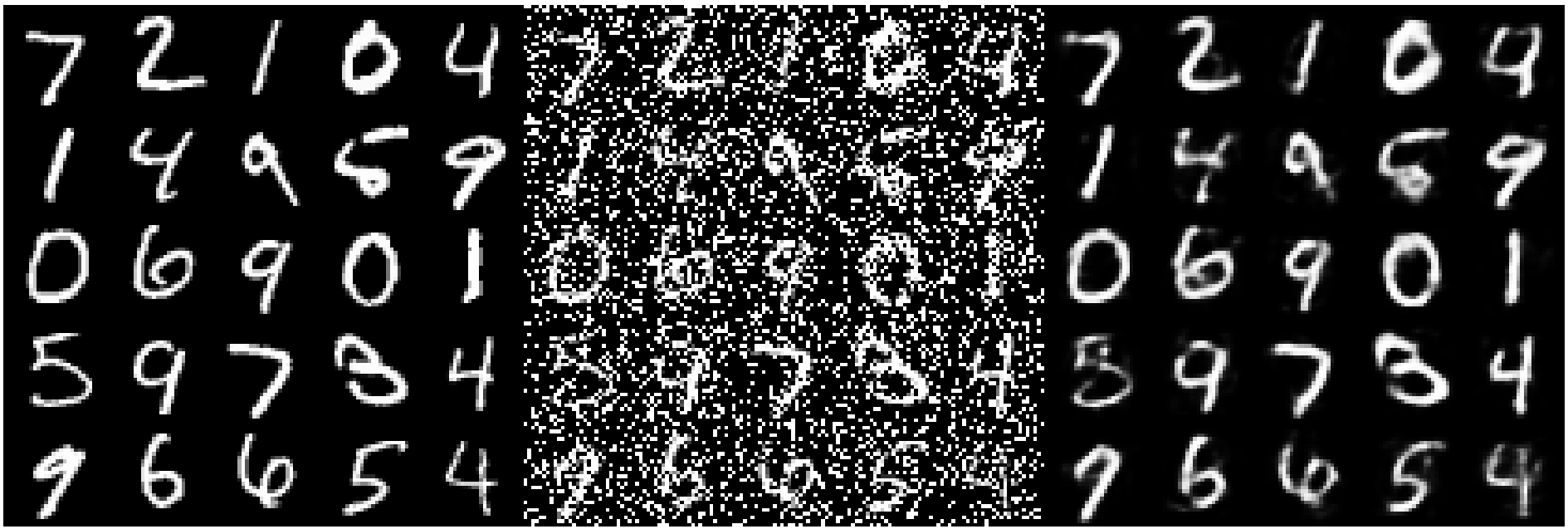
What if we are not reducing dimensionality?

- If the architecture doesn't have a bottleneck (i.e. more hidden nodes than input nodes) the AE can just learn an identity
- We can randomly corrupt the data on purpose to solve this issue, that is add a noise process
- Keep in mind that we still compute the reconstruction error with the original data!



https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2

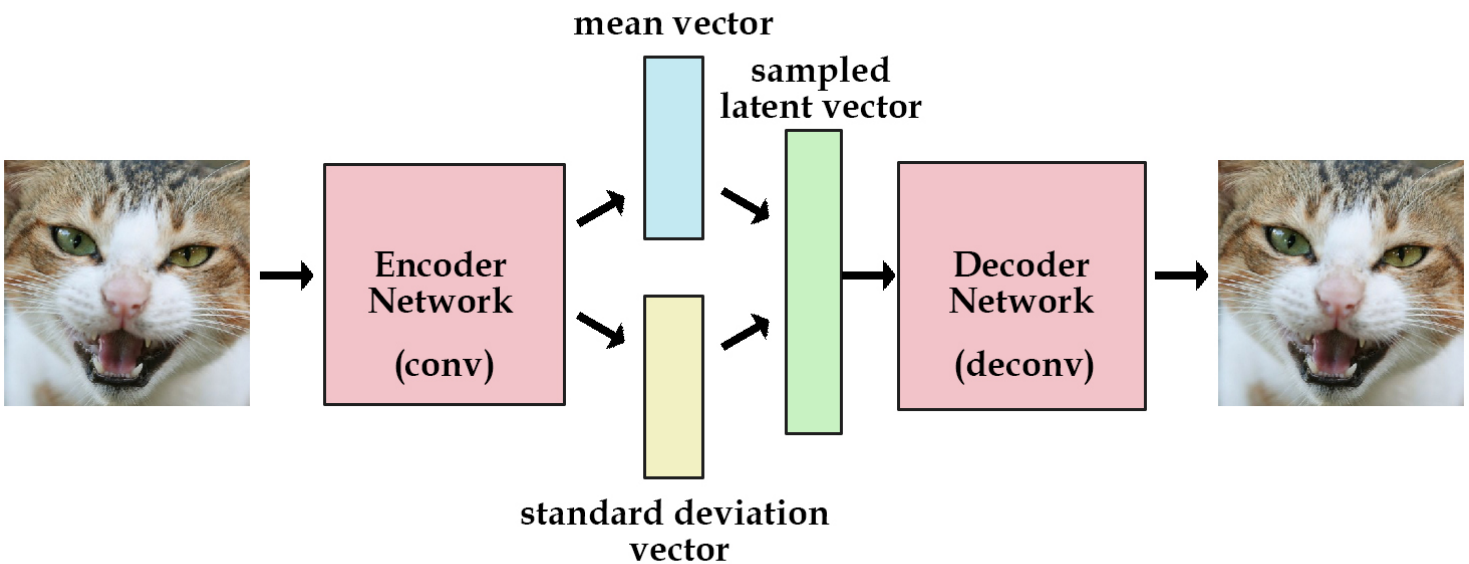
Denoising Autoencoders



https://www.doc.ic.ac.uk/ js4416/163/website/img/autoencoders/denoising-example.png

Variational Autoencoders

- Our AE has a latent embedding/variables in the hidden layer connecting encoder to decoder. But difficult to grasp as it is unconstrained.
- We can add a constraint such as forcing the latent vector to follow a unit Gaussian (e.g. by optimizing KL divergence in addition to reconstruction, which measures how close we are to a unit Gaussian).



http://kvfrans.com/content/images/2016/08/vae.jpg

Variational Autoencoders

- Consider a dataset X with variable x
- Assume data is generated by some random process involving unobserved random variable z
- z is generated from some prior distribution $p_\theta(z)$
- a value x is generated from some conditional distribution $p_\theta(x|z)$

$\rightarrow$ The parameters and values of latent variables z are not known to us. $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) dz$ is intractable.

$\rightarrow$ The true posterior density $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$ is intractable.

Variational Autoencoders

"Auto-Encoding Variational Bayes", Kingma and Welling, ICLR 2014

- Why a Bayesian formulation? $\rightarrow$ Lets us learn about the distribution of seen data $p(\mathbf{x})$ by capturing it through latent variables $\mathbf{z}$. However, as $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) dz$ is intractable, we do this by optimizing a lower-bound to the marginal distribution $p(\mathbf{x})$.
- How to do approximate Bayesian inference with neural networks + learning with probabilistic models whose latent variables have intractable posterior distributions $\rightarrow$ variational inference.
- In a VAE we use a reparameterization trick to make the model backward differentiable.

Variational Autoencoders

The densities of the marginal and joint distribution are related through Bayes rule:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})} = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} \tag{1}$$

Using the logarithm on both sides we can write this as a sum:

$$\log p_\theta(\mathbf{x}) = \log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log p_\theta(\mathbf{z}|\mathbf{x}) \tag{2}$$

Here, we do not know our real posterior $p_\theta(\mathbf{z}|\mathbf{x})$. We thus use variational inference and introduce an approximation $q_\phi(\mathbf{z}|\mathbf{x})$ to the posterior:

$$\log p_\theta(\mathbf{x}) = \int_z q_\phi(\mathbf{z}|\mathbf{x}) \left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log p_\theta(\mathbf{z}|\mathbf{x})\right] dz \tag{3}$$

Variational Autoencoders

$$\begin{aligned}\log p_\theta(\mathbf{x}) = \int_z q_\phi(\mathbf{z}|\mathbf{x})[&\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log p_\theta(\mathbf{z}|\mathbf{x}) \\ &+ \log q_\phi(\mathbf{z}|\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})]dz\end{aligned} \tag{4}$$

Using the definition of reverse Kullback Leibler divergence $KL(Q \mid\mid P) = \int_{-\infty}^{\infty} Q(x) \log \frac{Q(x)}{P(x)}$:

$$\begin{aligned}\log p_\theta(\mathbf{x}) =& KL(q_\phi(\mathbf{z}|\mathbf{x}) \mid\mid p_\theta(\mathbf{z}|\mathbf{x})) + \\ &\mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_\phi(\mathbf{z}|\mathbf{x}) \mid\mid p_\theta(\mathbf{z}))\end{aligned} \tag{5}$$

Here, the first term on the right hand side cannot be evaluated, but by definition is strictly positive. Therefore if we optimize the remaining terms on the right hand side, we optimize a lower-bound to $p(\mathbf{x})$.

Variational Autoencoder

Thus, the marginal likelihood can be rewritten as:

$$\log p_\theta(x) = D_{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right) + \mathfrak{L}(\theta, \phi; x)$$

$\rightarrow$ First RHS term is the KL divergence between approximate and true posterior
$\rightarrow$ Second RHS term is called the variational lower bound on the marginal likelihood of a datapoint as the KLD is always positive. With our derivation from above we can write it as:

$$\log p_\theta(x) \geq \mathfrak{L}(\theta, \phi; x) = -D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) + \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$$

Variational Autoencoders

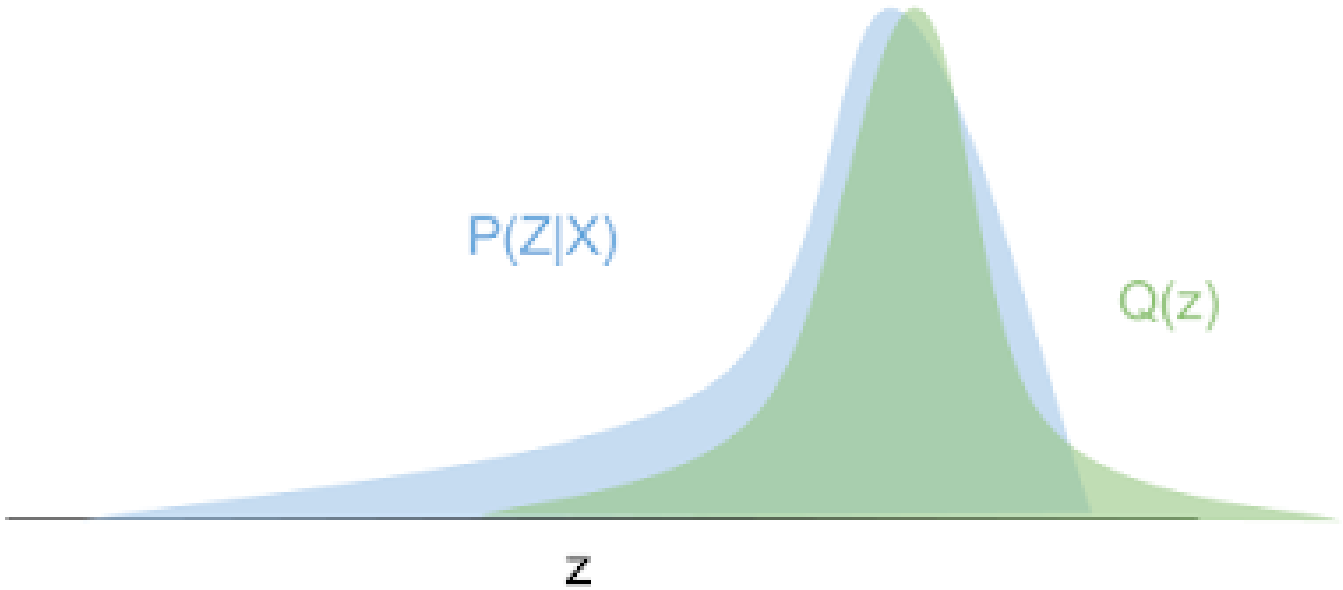
$$\mathfrak{L}(\theta, \phi; x) = -D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) + \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$$

- Recognition model/probabilistic encoder: $q_\phi(z|x)$: variational approximation to intractable posterior $p_\theta(z|x)$
- z has an interpretation as latent representation or code. $q_\phi(z|x)$: probabilistic encoder $\rightarrow$ given a datapoint x it produces a distribution over possible values of z from which it could have been generated
- $p_\theta(x|z)$: probabilistic decoder $\rightarrow$ given a z it produces a distribution over possible values of x

- The first RHS term is a KL divergence encouraging the approximate posterior to be close to the prior $p_\theta(z)$
- Second RHS term is expected reconstruction error as given by the log-likelihood (estimated by sampling)

Variational Autoencoders

https://blog.evjang.com/2016/08/variational-bayes.html - read more about variational inference and the nuances of Kullback-Leibler divergence at this blog post.



"Reverse KL divergence measures the amount of information (in nats, or units of $\frac{1}{\log 2}$ bits) required to "distort" $p_\theta(\mathbf{z})$ into $q_\phi(\mathbf{z})$"

Variational Autoencoders

Example: let approximate variational posterior be a multivariate Gaussian:

$$\log q_\phi(z|x) = \log \mathcal{N}(z; \mu, \sigma \, \mathtt{I})$$

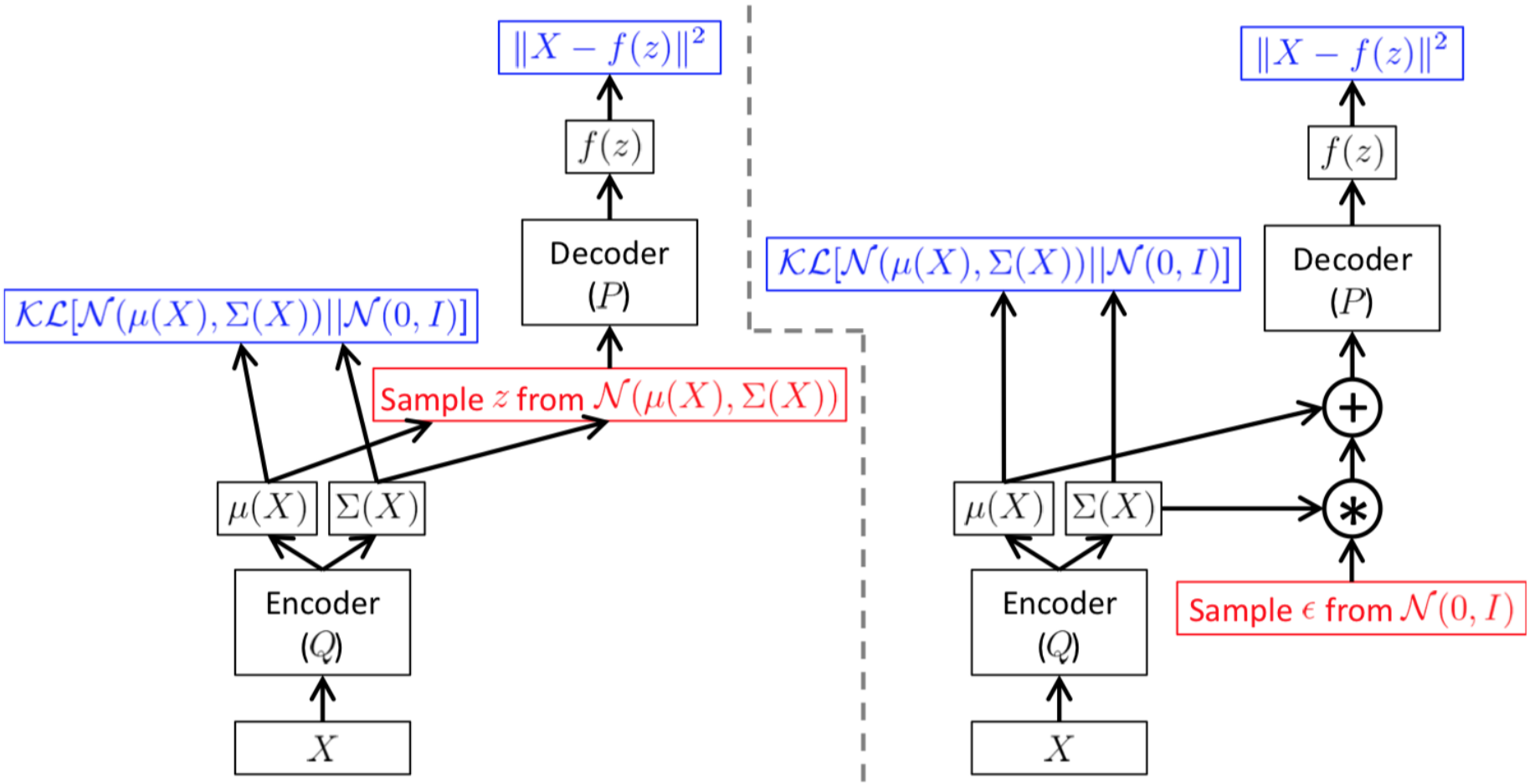
- Use a reparameterization trick to generate samples from $q_\phi(z|x) \rightarrow$ express random variable z as deterministic variable.
- Sample the posterior $z \sim q_\phi(z|x)$ using $z = \mu + \sigma \circ \epsilon$

$$\mathfrak{L}(\theta, \phi; x) \approx \frac{1}{2}\left(1 + \log \sigma^2 - \mu^2 - \sigma^2\right) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x|z^l)$$

where

$$z^l = \mu + \sigma \circ \epsilon^l \quad \text{and } \epsilon^l \sim \mathcal{N}(0, \mathtt{I})$$

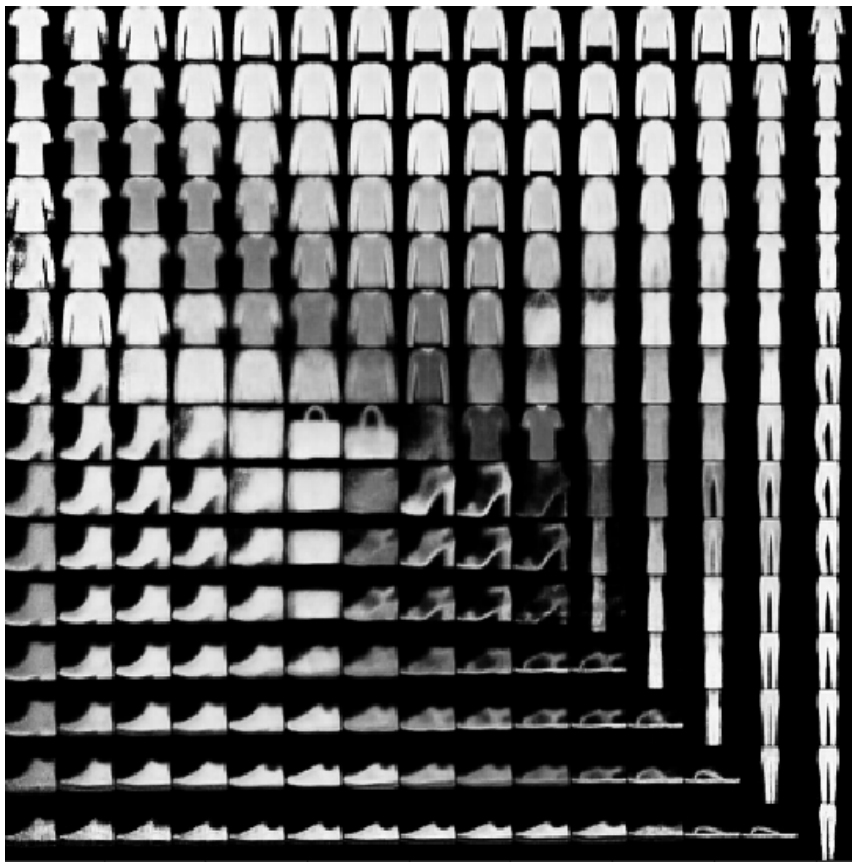
Variational Autoencoders



"Tutorial on Variational Autoencoders", Carl Doersch

Variational Autoencoders

- After the model is trained we can sample images by sampling from the prior $z \sim p_\theta(\mathbf{z})$, here $\mathcal{N}(0, 1)$, and calculating the decoder.
- Example for a 2-D latent space grid of z values for Fashion MNIST:

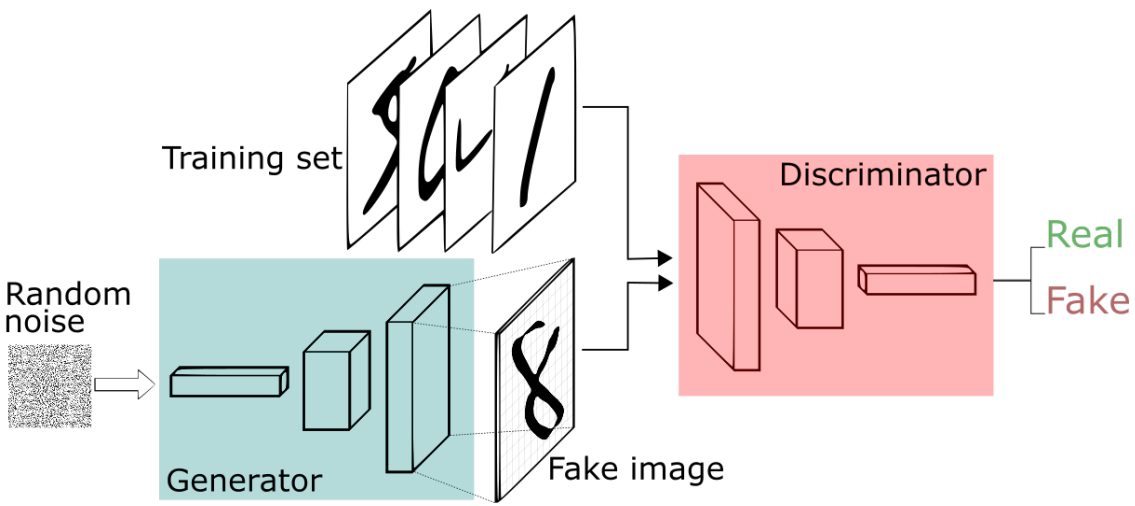


from: https://github.com/sadanand-singh/datasciencevision.com/blob/master/content/posts/2018-04-26-autoencoders.md

Generative Adversarial Networks (GAN)

- Various methods to use NNs to generate data
- GANs are one further option: decoder is trained to generate output that resembles training data to fool a different "discriminator" (C)NN.
- Training is based on the discriminator being able to distinguish real from generated samples and generator in turn learning to produce more "realistic" data.

We will delve into GANs in our class next week!



https://deeplearning4j.org/generative-adversarial-network

- Unsupervised pre-training (semi-supervised learning) doesn't necessarily hold-up to its promise when enough labeled data is available. Supervised learning has more meaningful features with respect to a task.
- It is difficult to distinguish "background concepts" from "content" in unsupervised learning.
- In images, learning is conducted with metrics on a pixel-level (and not e.g. concepts or entities).
- In clustering the objective is not taken into account in the distance metric. Number of clusters typically not known a priori.
- Stability of very deep NN optimization, especially for GANs.
- Validation of generative models (such as GANs) or clustering algorithms can be difficult.

Our next practical session

1. Let's take our FashionMNIST or Kuzushiji dataset and adapt our existing neural networks to both autoencoders and convolutional autoencoders. We can reuse all the pieces but no longer require the data labels. Apart from monitoring the loss, we should visualize the reconstructed images during training.
2. Re-use the learned feature base of any of your autoencoders & train a classifier (e.g. a single linear layer) on top using a subset of the train data (with labels).
3. (Optional): We can adapt the above autoencoders to denoising autoencoders in representation learning with higher dimensional spaces.
4. We can change our auto-encoder and its training to the variational formulation. Based on the trained model, we can sample from the (Gaussian) prior p(z) and generate images.