PROBABILISTIC AND DIFFERENTIABLE PROGRAMMING

V6: Deep Learning III (Autoencoders, GANs)

Özgür L. Özçep

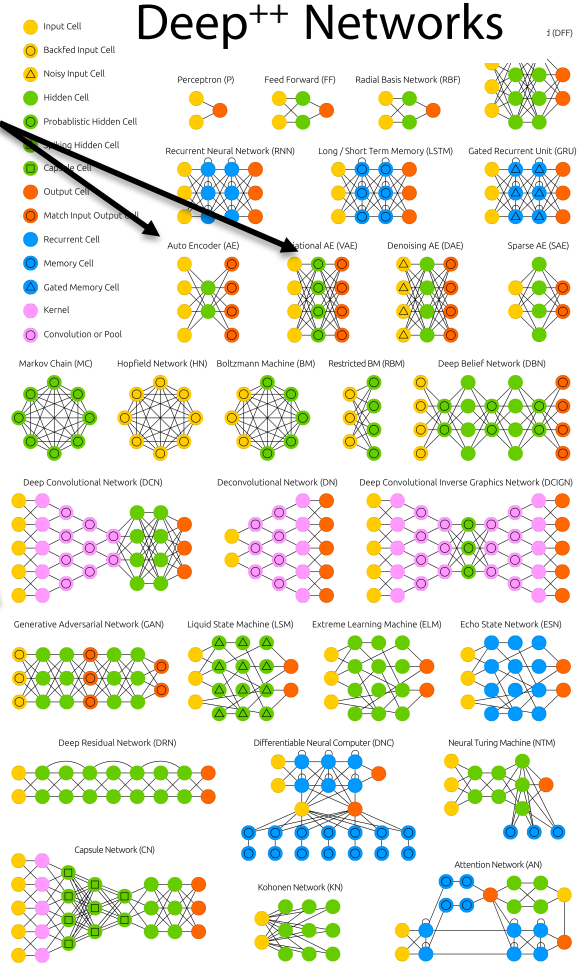
Universität zu Lübeck

Institut für Informationssysteme

Today's Agenda

1. Same-Same but different: Autoencoders

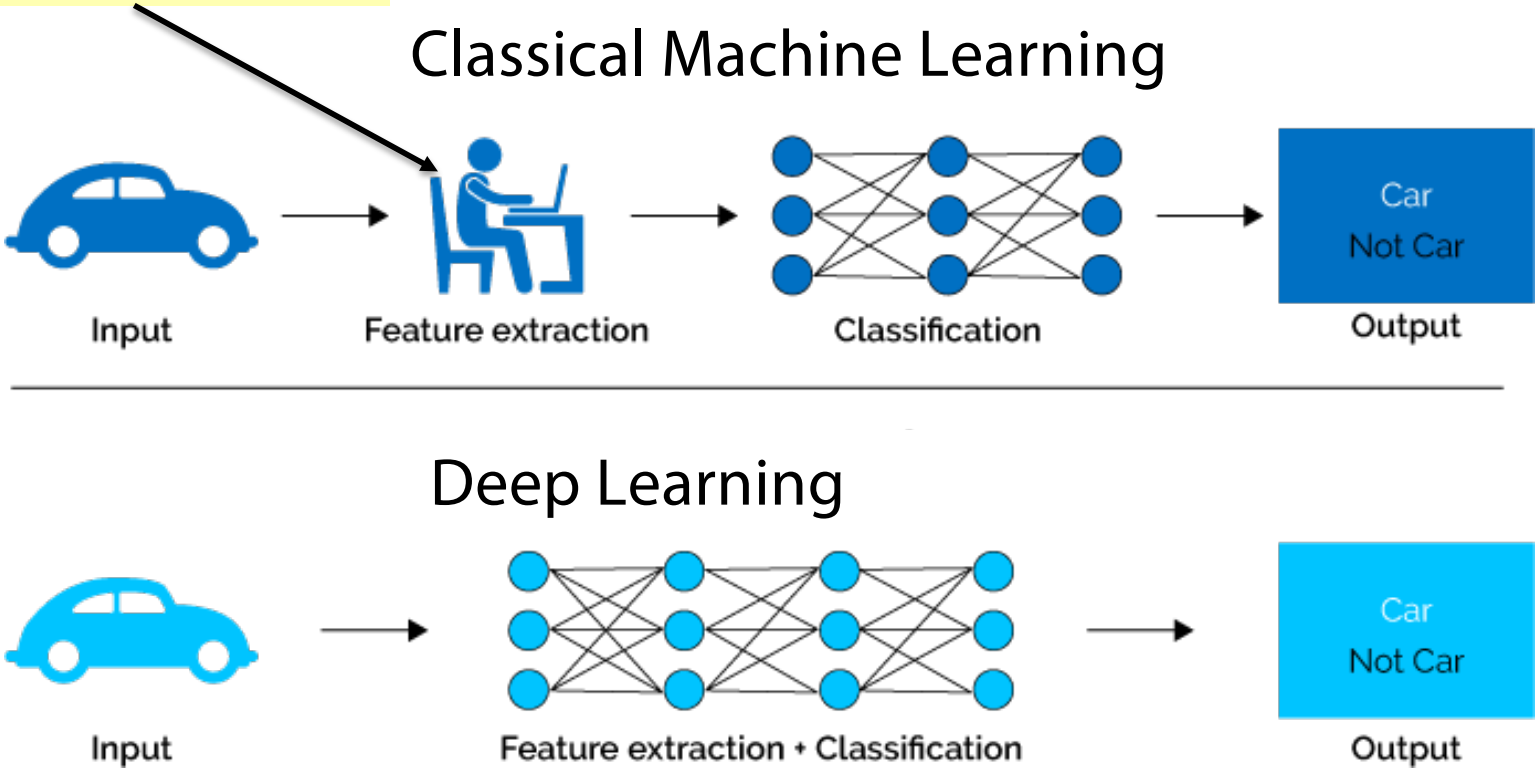
2. Imagine me GAN: Generative adverserial networks



AUTOENCODERS

Autoencoders for Feature Extraction

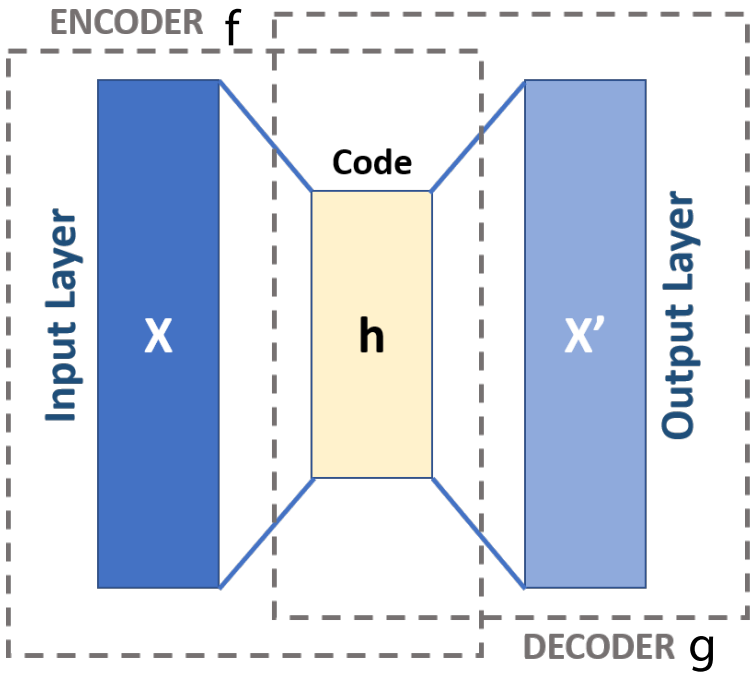
Example family car:
we presumed features
price and mileage



Adapted from https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

An autoencoder is a network that is trained to copy its input to its output

- Internally there is some hidden vector $\boldsymbol{h}$ that describes a code that represents the input.
- Conceptually it consists of two parts
  - Encoder $\boldsymbol{h} = f(\boldsymbol{x})$
  - Decoder $\boldsymbol{r} = g(\boldsymbol{h})$
- and has loss that tries to minimise the reconstruction error (typically MSE)



Autoencoder constraints (Same-same but different)

- A linear autoencoder with a sufficient number of weights (e.g. if the dimension of h was greater than or equal to that of $\boldsymbol{x}$) could learn $g(f(\boldsymbol{x})) = \boldsymbol{x}$, but this wouldn't be useful!
- In practice we apply restrictions (inductive biases) to stop this happening.
- The objective is to use these restrictions to force the autoencoder to learn useful properties of the data.

Undercomplete Autoencoders - linear

- Undercomplete autoencoders have

$$\dim(\boldsymbol{h}) << \dim(\boldsymbol{x})$$

This forces the encoder to learn a compressed representation of the input.

- The representation will capture the most salient features of the input data.

Undercomplete Autoencoders - linear

- Consider the single-hidden layer linear autoencoder network given by:
  - $\boldsymbol{h} = \boldsymbol{W}_e\boldsymbol{x} + \boldsymbol{b}_e$
  - $\boldsymbol{r} = \boldsymbol{W}_d\boldsymbol{h} + \boldsymbol{b}_d$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{h} \in \mathbb{R}^m$ and $m < n$.

- With the MSE loss, this autoencoder will learn to span the same subspace as PCA for a given set of training data.
- Note that the autoencoder weights are not however constrained to be orthogonal (like they would be in PCA)

Reminder: PCA, SVD, LSI

- PCA= Principal Component Analysis
  - Find principal components of a linear mapping given by a matrix A
- SVD: Singular Value Decomposition
  - Other name with emphasis on the singular values of A
- Based on these: LSI: Latent Semantic Indexing

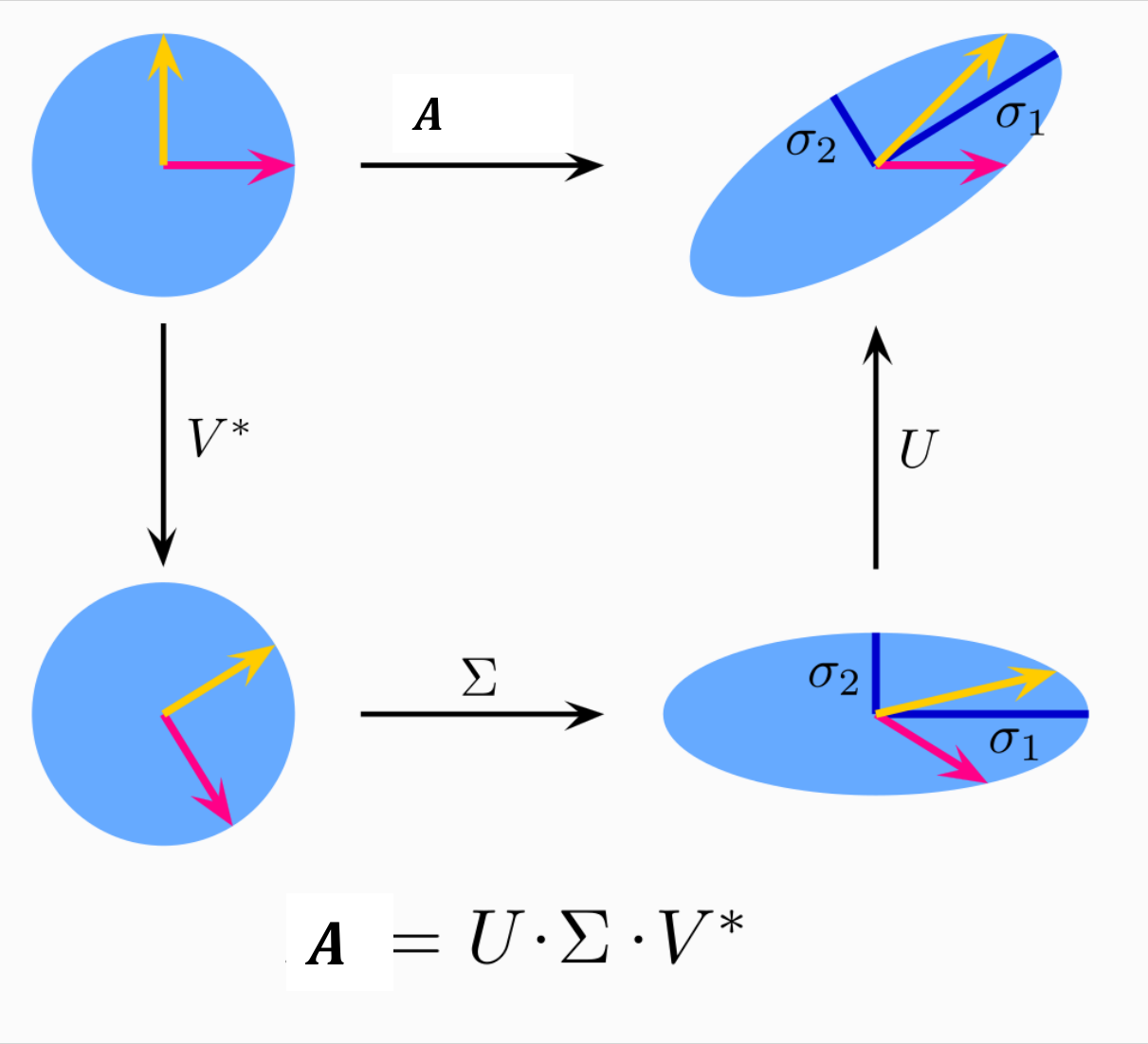
Document matrix A

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

Documents

Terms

Reminder: Decomposition



[Wikipedia]

Reminder: Low-rank Approximation

- SVD can be used to compute optimal low-rank approximations for a matrix A of rank r
- Approximation problem: Find A_k of rank k such that

$$A_k = \arg\min_{X:rank(X)=k} \|A - X\|_F$$

← Frobenius norm

$$\|A\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}.$$

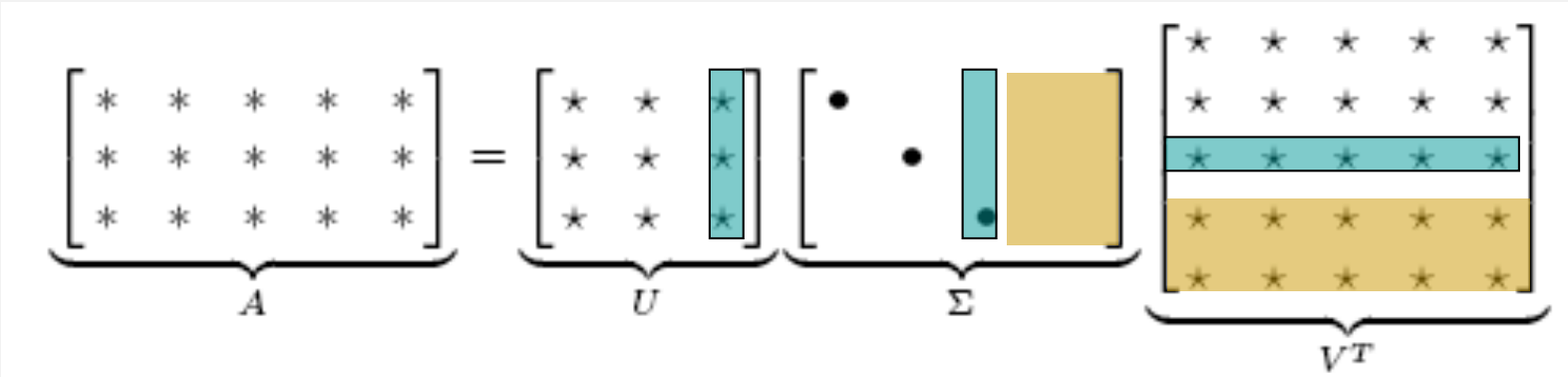
A_k and X are both $m \times n$ matrices

Typically, want $k << r$

Reminder: Low-rank Approximation

- Solution via SVD

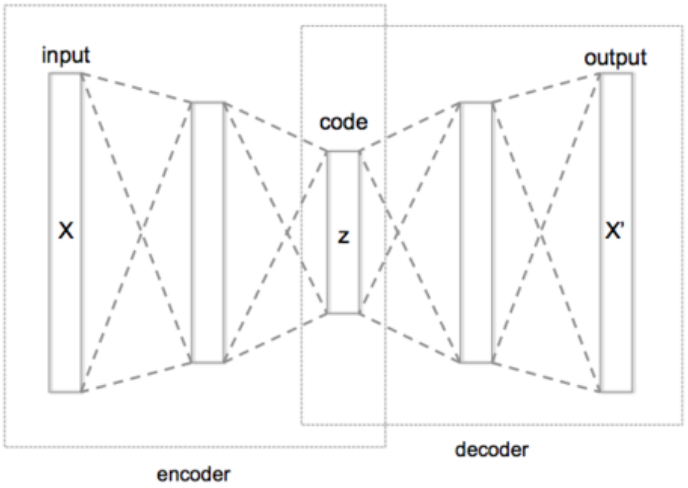
$$A_k = U\ \mathrm{diag}(\sigma_1,\ldots,\sigma_k,\underbrace{0,\ldots,0}_{\text{set smallest r-k singular values to zero}})V^T$$



C. Eckart, G. Young, The approximation of a matrix by another of lower rank. Psychometrika, 1, 211-218, 1936

Undercomplete Autoencoders — deeper and nonlinear

- A linear autoencoder with one or more hidden layery learns to map into the same subspace as PCA.
- What happens if you introduce non-linearity?
  - Single hidden layer network with non-linear activations on the encoder (keeping the decoder linear) and MSE loss also just learns to span the PCA subspace (Bourlard/Kamp 88)!
  - But, if you add more hidden layers with non-linear activations (to either the encoder, decoder or both) you can effectively perform a powerful non-linear generalisation of PCA



Deep Autoencoders - caveat

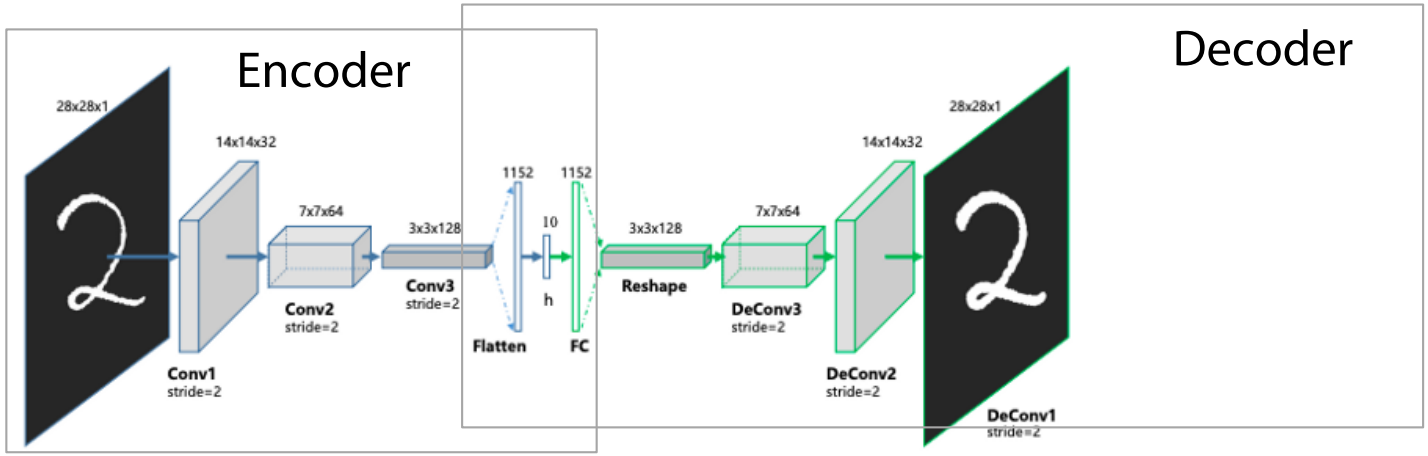
- There is a slight catch: if you give the deep autoencoder network too much capacity (too many weights) it will learn to perform the copying task without extracting anything useful about the data.
- Of course this means that will likely not generalise to unseen data.

Extreme example:

- Consider a powerful encoder that maps $\boldsymbol{x}$ to $h \in \mathbb{R}^1$ Each training example $\boldsymbol{x}(i)$ could e.g. be mapped to i.
- The decoder just needs to memorise the training examples so that it can map back from i.

Undercomplete Autoencoders - Convolutional

- Thus far, we only considered autoencoders with vector inputs/outputs and fully-connected layers.
- There is nothing stopping us using any other kinds of layers though...
- If we're working with image data, where we know that much of the structure is 'local', then using convolutions in both the decoder makes sense



https://towardsdatascience.com/convolutional-autoencoders-for-image-noise-reduction-32fce9fc1763

Regularised Autoencoders

- Rather than (necessarily) forcing the hidden vector to have a lower dimensionality than the input, we could instead utilise some form of regularisation to force the network to learn interesting representations...
- Many ways to do this; let's look at two of them:
  1. Denoising Autoencoders
  2. Sparse Autoencoders

Denoising autoencoders

- Denoising autoencoders (DAEs) take a partially corrupted input and train to recover the original undistorted input.
- To train an autoencoder to denoise data, it is necessary to perform a preliminary stochastic mapping to corrupt the data $(\boldsymbol{x} \rightarrow \tilde{\boldsymbol{x}})$.
  - E.g. by adding Gaussian noise.
- The loss is computed between the reconstruction (computed from the noisy input) against the original noise-free data.

Sparse Autoencoders

- In a sparse autoencoder, there can be more hidden units than inputs, but only a small number of the hidden units are allowed to be active at the same time.
- This is simply achieved with a regularised loss function:

$$l = l_{MSE} + \Omega(\boldsymbol{h})$$

- A popular choice would be to use an l_1 penalty

$$\Omega(\boldsymbol{h}) = \lambda \sum_i |h_i|$$

  - This choice can be justified by considering autoencoders as a means to approximatting maximum likelihood training of generative models

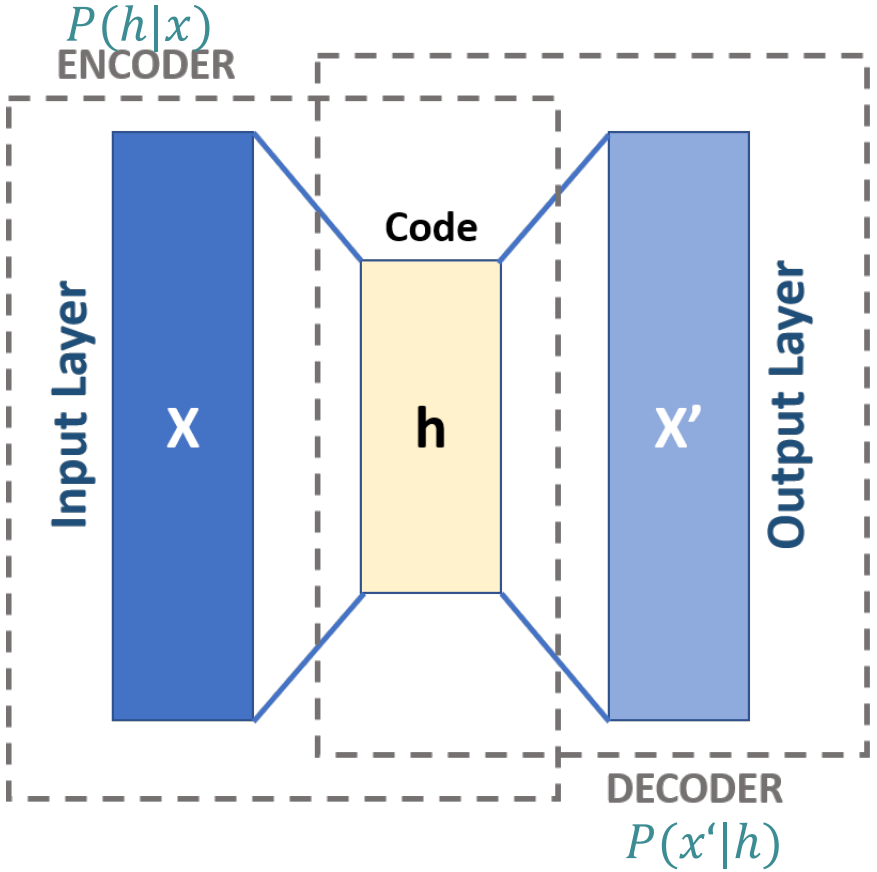
Autoencoder Applications

- Any basic AE (or its variant) can be used to learn a compact representation of data.
  - You can learn useful features from data **without the need for labelled data.**
  - Denoising can help generalise over the test set since the data is distorted by adding noise.
- Pretraining networks
- Anomaly Detection
- Machine translation
- Semantic segmentation

Stochastic Encoders and Decoders

- Consider encoders and decoders as distributions (general strategy useful for, e.g., determining cost functions and more generative models)

- In particular decoder learn $p_{decoder}(x'|h)$ by minimising negative log-likelihood $-\log(p(x'|h)$.



GENERATIVE MODELS

Sub-Agenda

- What is generative modelling and why do we do it?
- Differentiable Generator Networks
- Variational Autoencoders
- Generative Adversarial Networks

Generative Models (GMs)

(for the relevant basics of probability theory appendix)

- Learn (conditional) models of the data: $p(x)$ (resp. $p(x|y = y)$)
- Some GMs allow calculation of propabilities $p(X = x)$
- Some GMs allow sampling of probability distributions $x \sim p(X)$
- Some GMS can do both of the above
  - e.g. a Gaussian Mixture Model is an explicit model of the data using k Gaussians
  - The likelihood of data $\boldsymbol{x}$ is the weighted sum of the likelihood from each of the k Gaussians
  - Sampling can be achieved by sampling the categorical distribution of k weights followed by sampling a data point from the corresponding Gaussian

Why generative modelling?

- Try to understand the processes through which the data was itself generated
  - Probabilistic latent variable models like VAEs or topic models (PLSA, LDA, ...) for text
  - Models that try to disentangle latent factors (like β-VAE)
- Understand how likely a new or previously unseen piece of data is
  - outlier prediction, anomaly detection, . . .
- Make ‘new’ data
  - Make ‘fake’ data to use to train large supervised models?
  - ‘Imagine’ new, but plausible, things?

Differentiable Generator Networks

- GMs not new (-> probabilistic graphical models)
  - ...But difficult to train and scale to real data
- Recent advances along four loose strands:
  1. Invertible density estimation - A way to specify complex generative models by transforming a simple latent distribution with a series of invertible functions.
  2. Autoregressive models - Another way to model $p(x)$ is to break the model into a series of conditional distributions: $p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)$.
  3. Variational autoencoders – see following slides
  4. Generative adversarial networks – see following slides
- Common thread in recent advances is that the loss functions are end-to-end differentiable.

Diffentiable Generator Networks

- We’re interested in models that transform samples of latent variables z to
  - samples $\boldsymbol{x}$, or,
  - distributions over samples $\boldsymbol{x}$
- The model is a (differentiable) function $g(\boldsymbol{z}, \boldsymbol{\theta})$
  - typically g is a neural network.

Example: Samples from normal distribution

- Consider a simple generator network with a single affine layer that maps samples $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

$$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \xrightarrow{g_\theta(z)} \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

- Note: Exact solution is $\boldsymbol{x} = g_\theta(\boldsymbol{z}) = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{z}$

  where $\boldsymbol{L}$ is the Cholesky decomposition of Σ: $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$, lower triangular $\boldsymbol{L}$.

Generating samples

- More general: g is a nonlinear transformation of a distribution over $\boldsymbol{z}$ to a distribution over $\boldsymbol{x}$

$$p_z(\boldsymbol{z}) \quad \overset{g(z)}{\longrightarrow} \quad p_x(\boldsymbol{x})$$

- Calculus says: For any invertible, differentiable, continuous g:

$$p_z(\boldsymbol{z}) = p_{\boldsymbol{x}}\big(g(z)\big)\,|\det\frac{\partial g}{\partial z}|$$

- Hence one gets probability distribution over $\boldsymbol{x}$

$$p_x(\boldsymbol{x}) = p_{\boldsymbol{z}}\big(g^{-1}(\boldsymbol{x})\big)\left|\det\frac{\partial g}{\partial z}\right|^{-1}$$

Generating Distributions

- Rather than use g to provide a sample of $\boldsymbol{x}$ directly, we could instead use g to define a conditional distribution over $\boldsymbol{x}, p(\boldsymbol{x}|\boldsymbol{z})$
- For example, g might produce the parameters of a particular distribution - e.g.:
  - means of Bernoulli
  - mean and variance of a Gaussian
- The distribution over $\boldsymbol{x}$ is imposed by marginalising $\boldsymbol{z}$:

$$p(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z})$$

Distribution vs. Samples

- In both cases one can use reparameterisation tricks for training models
  - Needed, because we want to use backpropagation; and how to do this with random nodes in network?
- Generating distributions:
  - Plus: works for both continuous and discrete data
  - Minus: need to specfify the form of the output distribution
- Generating samples:
  - Plus: works for continuous data
  - (also discrete data with some additional trick)
  - Plus: don‘t need to specify distribution in explicit form

Reparameterisation trick

- Assume $y \sim \mathcal{N}(\mu, \sigma^2)$
- Want to calculate $\frac{\partial y}{\partial \mu}$ and $\frac{\partial y}{\partial \sigma}$

- Idea: Factor out the random part. In this case

  $y = \mu + \sigma z$ where $z \sim \mathcal{N}(0,1)$

- y now is a function of a deterministic operation with variables μ and σ with an extra input z
  - And, importantly, is not a function of μ or σ (and vice versa)

- This works also for other distributions under similar constraints

Complexity of Generative Modelling

- In classification both input and output are given
  - Optimization only needs to learn the mapping
- Generative modelling is more complex than classification because
  - learning requires optimizing intractable criteria
  - data does not specify both input z and output x of the generator network
  - learning procedure needs to determine how to arrange z space in a useful way and how to map z to x

VARIATIONAL AUTOENCODERS

Variational Autoencoders (VAEs)

- VAEs based on following generative process

$$\boldsymbol{z} \sim p_{model}(\boldsymbol{z}) \xrightarrow{p_{model}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})=p_{model}(\boldsymbol{x}|\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{z}))} \boldsymbol{x} \sim p_{model}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})$$

- Learning problem: Find $argmax_{\boldsymbol{\theta}} p(\boldsymbol{x})$ for each $\boldsymbol{x}$ in the training set under $p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})\, p(\boldsymbol{z}) \mathrm{d}\boldsymbol{z}$
- Often: $p_{model}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
- Often: $p_{model}(\boldsymbol{x}|\boldsymbol{z})$ chosen according to data; typically Gaussian for real-valued or Bernoulli for binary data
  - Intuition: Don‘t want exactly create the training data but things like training examples

Variational Autoencoders

First try: $p(x) \approx \sum p(x\,|z_i; \theta)$ for n samples $\{\boldsymbol{z_1}, \ldots, \boldsymbol{z_n}\}$ of $\boldsymbol{z}$

- But intractable in practice; n would be extremely big
- For most $\boldsymbol{z}$, $p(\boldsymbol{x}|\boldsymbol{z})$ will be nearly zero, hence no significant contribution to $p(\boldsymbol{x})$

Key idea of VAE: Learn to sample values of $\boldsymbol{z}$ that are likely to have produced $\boldsymbol{x}$ and compute $p(\boldsymbol{x})$ from them

- Define weighting function $\mathrm{q}(\boldsymbol{z}|\boldsymbol{x})$
- Space of $\boldsymbol{z}$ values likely under q should be much smaller than the space under prior $p(\boldsymbol{z})$
- We can now compute $\mathbb{E}_{\boldsymbol{z}\sim q} p(\boldsymbol{x}|\boldsymbol{z}; \boldsymbol{\theta})$
  - But how does this help if q is not a normal distribution?
  - How does expectation relate to $p(\boldsymbol{x})$?

Deriving ELBO: Evidence Lower Bound

$$\log p(x) = \log \int p(x|z)\, p(z) dz$$

Log-probability

$$\log p(x) = log \int p(x|z) \frac{p(z)}{q(z|x)} \mathrm{q(z|x)} dz$$

Propososal

$$\log p(x) \geq \int q(z|x)\, log\, p(x|z) \frac{p(z)}{q(z|x)} dz$$

Jensens's Inequality

$$\log \int p(x)g(x)\, \mathrm{d}\boldsymbol{x} \geq \int p(x) \log g(x)\, \mathrm{d}\boldsymbol{x}$$

$$log\, p(x) \geq \int q(z|x)\, log\, p(x|z)\ dz\ -$$

$$\int q(z|x)\, log\, p(x|z) \frac{q(z|x)}{p(z)} dz$$

Rearrange

$$\log a \cdot b = \log a + \log b$$
$$a^{-b} = -b \log a$$
$$\frac{a}{b} = \left(\frac{b}{a}\right)^{-1}$$

$$log\, p(x) \geq \mathbb{E}_{z\sim q(z|x)} \log p(x|z) - D_{KL}(q(z|x)||p(z))$$

ELBO

D_{KL}: Kullback-Leibler divergence

Measures difference of probability distributions

The cornerstone of VAEs: ELBO

ELBO

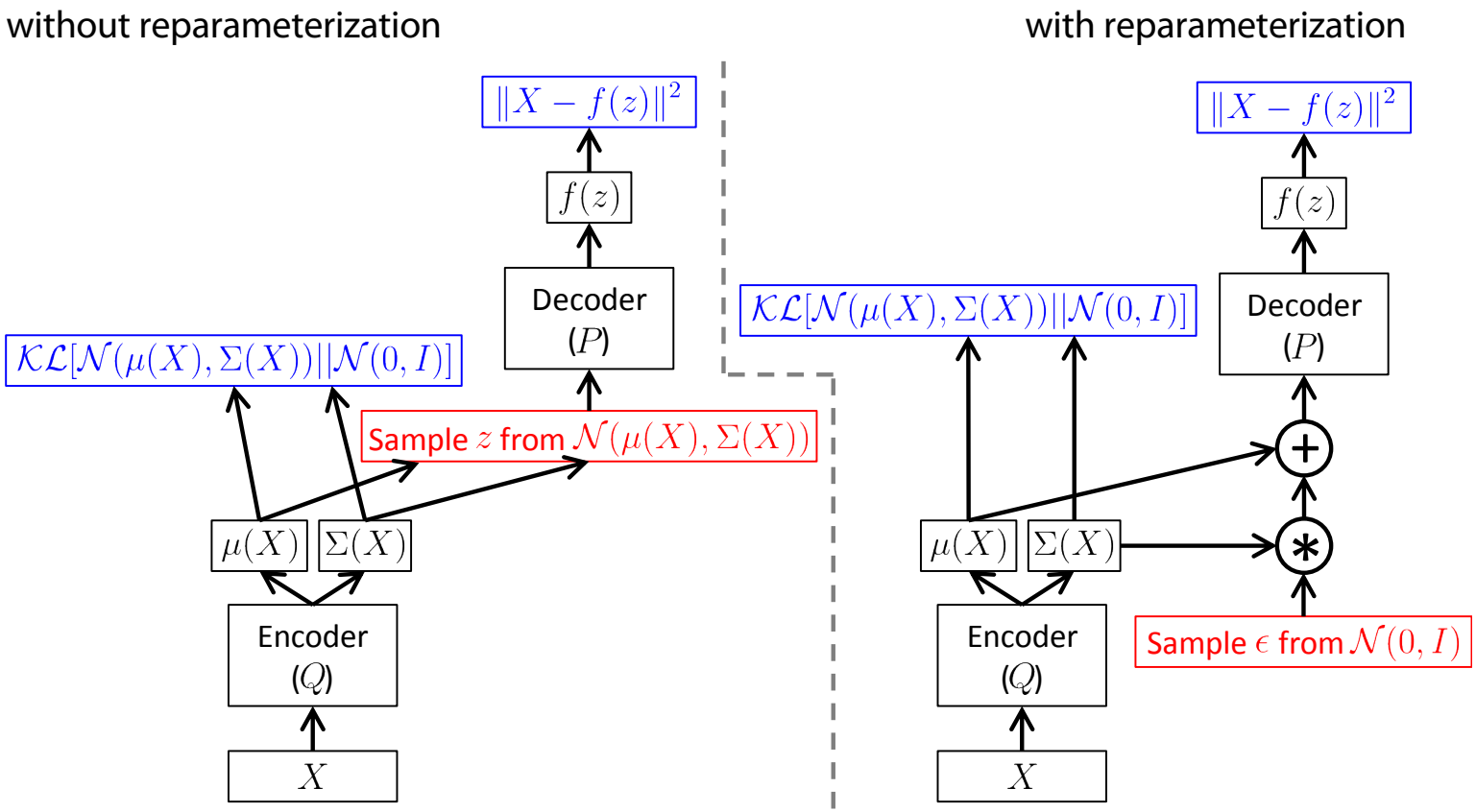
$$L(q) = \mathbb{E}_{z\sim q(Z|x)} \log p(x|z) - D_{KL}(q(z|x)||p(z)) \leq log\, p(x)$$

- Maximize $L(q)$ to maximize $\log p(x)$
- Expectation term is like a reconstruction of log-lieklihood found in normal autoencoders
  - If $p_{model}(x|z)$ is Gaussian, then this is MSE between the true training x and generated sample computed from z, averaged across many z's (each a function of x)
  - The KL term is forcing the approximate posterior $q(z|x)$ towards the prior $p_{model}(z)$.

Why is it called an autoencoder?

- $q(z|x)$ is considered as encoder (z are the hiddens h)
- $p_{model}\,(x; g_\theta(z))$ is a decoder network; it takes a z and decodes it into a target x
- From a practical standpoint, a VAE is a normal autoencoder with two key differences:
  - the encoder generates a distribution that must be sampled
    - the network produces the sufficient statistics of the distribution (e.g. means and diagonal co-variances for a typical VAE with Gaussian $q(z|x)$)
  - the decoder generates a distribution, which, during training the negative log-likelihood of the true data x is compared against

VAE



From Carl Doersch's Tutorial on VAEs - https://arxiv.org/pdf/1606.05908.pdf

VAE Models and Performance

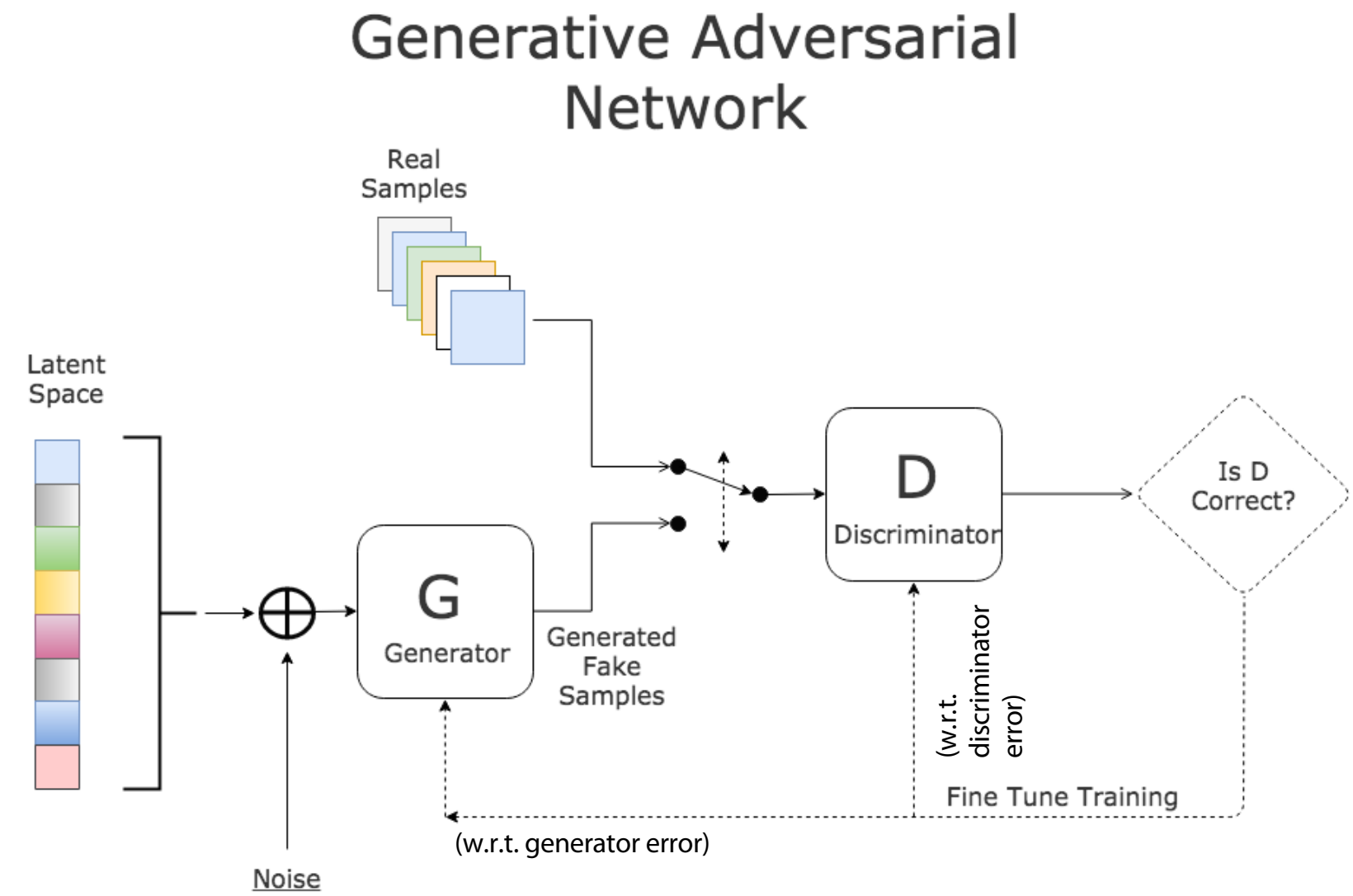
- VAEs can be used with any kind of data
  - the distributions and network architecture just need to be set
  - e.g. it's common to use convolutions in the encoder and transpose convolutions in (Gaussian) decoder for image data
- VAEs tend to be easy to optimise with stable convergence
- VAEs have a reputation for producing blurry reconstructions of images
  - Supposed to be due to a side effect of maximum-likelihood training
- VAEs tend to only utilise a small subset of dimensions of z
  - Pro: automatic latent variable selection
  - Con: better reconstructions should be possible given the available code-space

GENERATIVE ADVERSERIAL NETWORKS

The idea of GANs

- Pitch a generator and a discriminator against each other
  - Generator tries to draw samples from $p(x)$
  - Discriminator tries to tell if sample came from the generator (fake) or the real world
- Both discriminator and generator are deep networks (differentiable functions)
- LeCun himself again: ‘GANs, the most interesting idea in the last ten years in machine learning‘

General Architecture



GANs formally

- The generator $x = g(\boldsymbol{z})$
  - Input: sample $z \in \mathbb{R}^n, \mathrm{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ or $z \sim U(0, I)$
  - Output: sample $x \in \mathbb{R}^d, x \sim data\ distrib., n << d$

- The discriminator $y = d(x)$
  - Input: sample x
  - Out: probability $y \in [0,1]$ for query: is x fake or real?

GANs practically

- Training a standard GAN is difficult and often results in two undesirable behaviours
  - Oscillations without convergence. No guarantee that the loss will actually decrease...
    - It has been shown that a GAN has saddle-point solution, rather than local minima.
  - The mode collapse problem, when the generator models very well a small sub-population, concentrating on a few modes.
- Additionally, performance is hard to assess and often boils down to heuristic observations.

Deep Convolutional GANs (DCGANs)

- Motivates the use of GANS to learn reusable feature representations from large unlabelled datasets
- GANs known to be unstable to train, often resulting in generators that produce “nonsensical outputs”
- Model exploration to identify architectures that result in **stable** training across datasets with higher resolution and deeper models.

Coping with the problems: architecture guidelines

- Replace pooling layers with strided convolutions in the discriminator and fractional-strided (transpose) convolutions in the generator.
  - This will allow the network to learn its own spatial downsampling.
- Use batchnorm in both the generator and the discriminator.
  - This helps deal with training problems due to poor initialisation and helps the gradient flow.
- Eliminate fully connected hidden layers for deeper architectures.
- Use ReLU activation in the generator for all layers except for the output, which uses tanh.
- Use LeakyReLU activation in the discriminator for all layers

Summary on generative modelling

- Generative modelling is a massive field with a long history
- Differentiable generators have had a profound impact in making models that work with real data at scale
- VAEs and GANs are currently the most popular approaches to training generators for spatial data
- We've only scratched the surface of generative modelling
  - Auto-regressive approaches are popular for sequences (e.g. language modelling).
    - But also for images (e.g. PixelRNN, PixelCNN)
  - typically RNN-based
  - but not necessarily - e.g. WaveNet is a convolutional auto-regressive generative model

Uhhh, a lecture with a hopefully useful

APPENDIX

Probability theory basics reminder

Random variable (RV)

- possible worlds defined by assignment of values to random variables.
- Boolean random variables
  e.g., Cavity (do I have a cavity?).
  Domain is < true , false >
- Discrete random variables
  e.g., possible value of Weather is one of
  < sunny, rainy, cloudy, snow >
- Domain values must be exhaustive and mutually exclusive
- Elementary propositions are constructed by assignment of a value to a random variable: e.g.,
  - Cavity = false (abbreviated as ¬cavity)
  - Cavity = true (abbreviated as cavity)
- (Complex) propositions formed from elementary propositions and standard logical connectives, e.g., Weather = sunny ∨ Cavity = false

Probabilities

- Axioms (for propositions $a, b, \top = (a \vee \neg a)$, and $\bot = \neg \top$):
  - $0 \leq P(a) \leq 1;\ P(\top) = 1;\ P(\bot) = 0$
  - $(P(a \vee b) = P(a) + P(b) - P(a \wedge b)$
- Joint probability distribution of $\mathbf{X} = \{X_1, \ldots, X_n\}$
  - $\boldsymbol{P}(X_1, \ldots, X_n)$
  - gives the probability of every atomic event on $\boldsymbol{X}$
- Conditional probability
  $P(a \mid b) = P(a \wedge b) / P(b)\ if\ P(b) > 0$
- Chain rule
  $$\boldsymbol{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \boldsymbol{P}(X_i | X_1, \ldots, X_{i-1})$$
- Marginalization: $\boldsymbol{P}(Y) = \sum_{z \in Z} \boldsymbol{P}(Y, z)$
- Conditioning on Z:
  - $\boldsymbol{P}(Y) = \sum_{z \in Z} \boldsymbol{P}(Y|z)\boldsymbol{P}(z)$ (discrete)
  - $\boldsymbol{P}(Y) = \int \boldsymbol{P}(Y|z)\boldsymbol{P}(z)dz$ (continuous)
    $= \mathbb{E}_{\boldsymbol{z} \sim P(z)}\mathrm{P(Y|z)}$ (expected value notation)
- Bayes' Rule
  $$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)} = \frac{P(D|H) \cdot P(H)}{\sum_h P(D|h)P(h)}$$

Reminder: Multivariate Gaussians

$$\text{Write r.v. } \mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{pmatrix}$$

Then define $X \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to mean

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{m/2} \| \boldsymbol{\Sigma} \|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

Co-variance matrix

where the Gaussian's parameters have…

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_m \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma^2_1 & \sigma_{12} & \cdots & \sigma_{1m} \\ \sigma_{12} & \sigma^2_2 & \cdots & \sigma_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1m} & \sigma_{2m} & \cdots & \sigma^2_m \end{pmatrix}$$

One can show: $E[X] = \mu$ and $Cov[X] = \Sigma$.

Reminder: Why Gaussian?

- Andrew Moore: “Gaussians are as natural as Orange Juice and Sunshine”

(http://www.cs.cmu.edu/~awm/tutorials)

- Proves useful to model RVs that are combinations of many (non)-measured influences
- Makes life easy because
  1. Efficient representation
  2. Substitute probabilities by expectations

Color Convention in this Course

- Formulae, when occurring inline
- Newly introduced terminology and definitions
- Important results (observations, theorems) as well as emphasizing some aspects
- Examples are given with standard orange with possibly light orange frame
- Comments and notes in nearly opaque post-it
- Algorithms
- Reminders (in the grey fog of your memory)

Today‘s lecture is based on the following

- Jonathon Hare: Lectures 15, 17 of course „COMP6248 Differentiable Programming (and some Deep Learning)“ http://comp6248.ecs.soton.ac.uk/

References

- H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. Biological Cybernetics, 59(4):291–294, 1988.