GENERATIVE ADVERSARIAL NETWORKS

IN5400 — Machine Learning for Image Analysis

Ole-Johan Skrede

10.04.2019

University of Oslo

Messages

Outline

- Repetition
- Generative Adversarial Networks
- Other adversarial methods

Note

- Extremely active and fast-moving field
- Difficult to anticipate what will be important in the future
- We will cover the basics, not much of the new fancy stuff
- For a more up-to-date summary, see e.g. [Kurach et al., 2018]

REPETITION

- An autoencoder f consist of an encoder g and an decoder h
- The encoder maps the input x to some representation z

$$g(x) = z$$

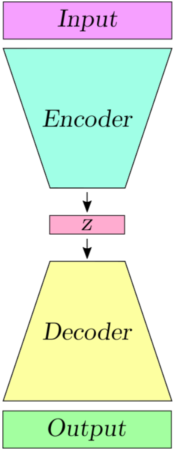
- We often call this representation z for the code or the latent vector
- The decoder maps this representation z to some output $\hat{x}$

$$g(z) = \hat{x}$$

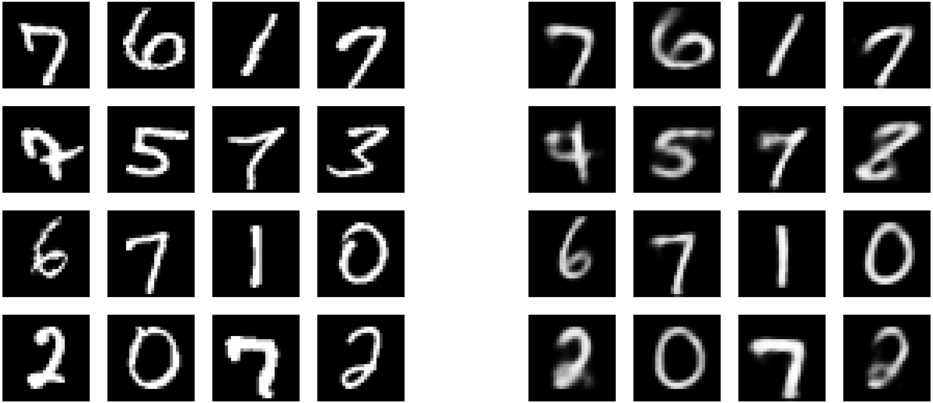
- We want to train the encoder and decoder such that

$$f(x) = h(g(x)) = \hat{x} \approx x$$

- Commonly used for compression, feature extraction and denoising

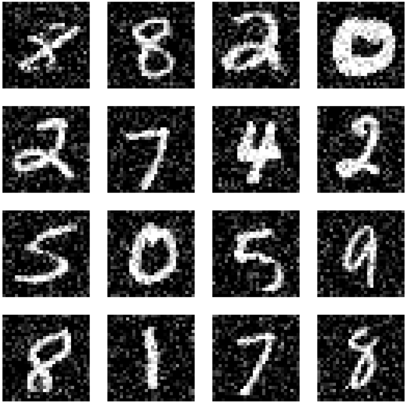


Compression autoencoder — MNIST example

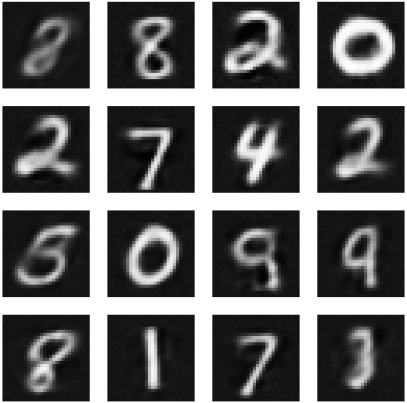


(a) Original (b) Reconstruction

Denoising autoencoder — MNIST example



(a) Original

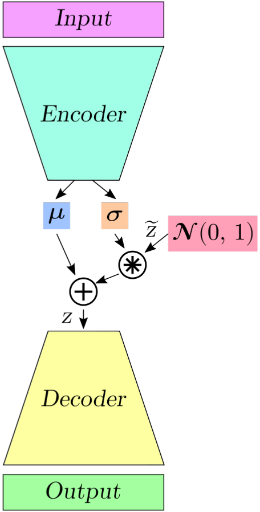


(b) Denoised reconstruction

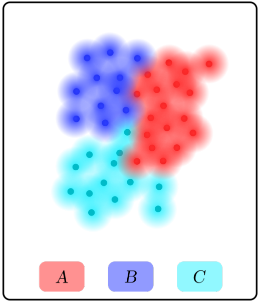
- A variational autoencoder is designed to have a continuous latent space
- This makes them ideal for random sampling and interpolation
- It achieve this by forcing the encoder g to generate Gaussian representations, $z \sim \mathcal{N}(\mu, \sigma^2)$
- More precisely, for one input, the encoder generates a mean μ and a standard deviation σ
- We then sample $\tilde{z}$ from the standard normal distribution $\tilde{z} \sim \mathcal{N}(0, 1)$
- Use the above to construct $z \sim \mathcal{N}(\mu, \sigma^2)$

$$z = \mu + \tilde{z}\sigma$$

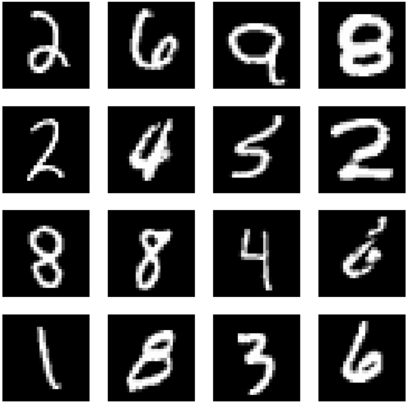
- Use z as input to the decoder



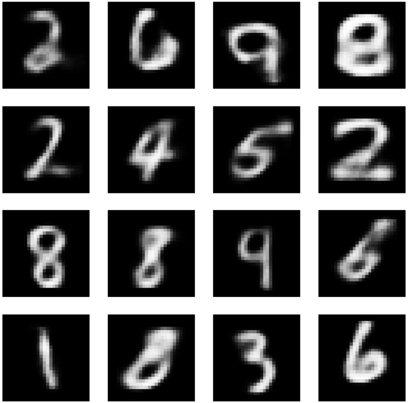
- This is a stochastic sampling
- That is, we can sample different z from the same sample (μ, σ)
- The intuition is that the decoder "learns" that for a given input x:
  - the point z is important for reconstruction
  - the neighbourhood of z is also important
- In this way, we have smoothed the latent space, at least locally
- In the previous lecture, we learnt ways to achieve this



VAE example: reconstruction

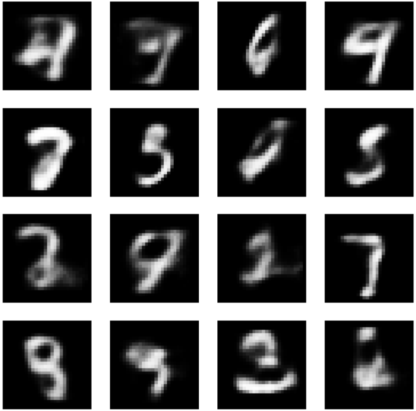


(a) Original



(b) Reconstructed

- Sample a random latent vector z from $\mathcal{N}(0, 1)$
- Decode z using the decoder from a trained variational autoencoder



VAE example: interpolation between samples

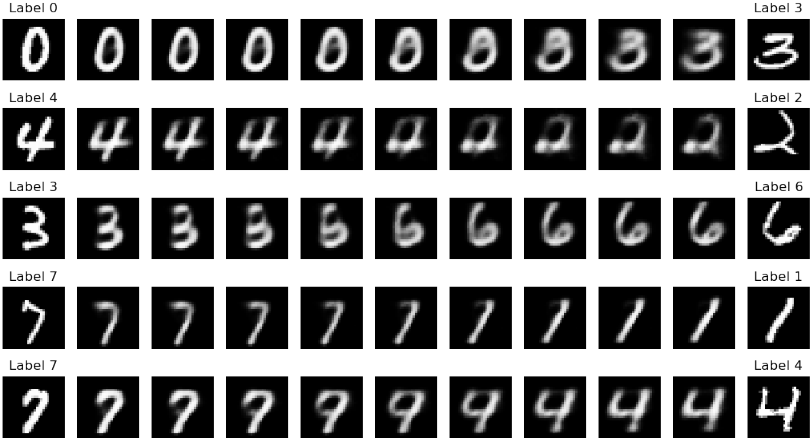
- We generate a signal c that is an interpolation between two signals a and b
- We can do this by a linear interpolation between the means

$$\mu_{c_k} = (1 - w_k)\mu_a + w_k \mu_b$$

where the different interpolation weights can be

$$w_k = \frac{k}{n}, \quad k = 0, \ldots, n$$

VAE example: interpolation between samples



GENERATIVE MODELLING

- We have training samples from an unknown distribution p_{data}
- We want a model that can draw samples from some distribution p_{model}
- p_{model} should be an approximation of p_{data}
- A model that can sample from this p_{model} is termed a *generative model*
- For brevity, we will refer to the distributions as $p_d = p_{\text{data}}$, and $p_m = p_{\text{model}}$.

- Some models explicitly construct p_m
- Some models implicitly construct p_m by only drawing samples from it
- Some models is able to do both
- VAE explicitly constructs p_m
- GAN only samples from p_m[1]

[1] There are GAN variants that are able to do both

Different approaches to generate samples from a distribution

- In the maximum likelihood case, we often have an explicit distribution $p_\theta(x)$, and for some fixed, observed data $\{x_i\}_{i=1}^m$, we find the parameters θ^* that maximizes the likelihood

$$\theta^* = \arg\max_\theta \prod_{i=1}^m p_\theta(x_i) \tag{1}$$

- In the implicit case, we have a data distribution p_d and some generator distribution p_g
- The random variable $Z \sim p_g$ are transformed via some function f to $X \sim p_m$
- This parametric function $f(x;\theta)$ can be a neural network, and the parameters θ are adjusted such that the model distribution is close to the data distribution $p_m \approx p_d$.

Motivation: why study generative modelling

- Analyse our ability to represent and manipulate high-dimensional distributions (e.g. images)
- Can be used as a tool in reinforcement learning
- Can be used in semi-supervised learning where labelled data is scarce
- Sampling of realistic examples from some high-dimensional distribution can have many applications

- Generating high-resolution images from low-resolution inputs
- GANs tend to produce perceptually pleasing and sharp results

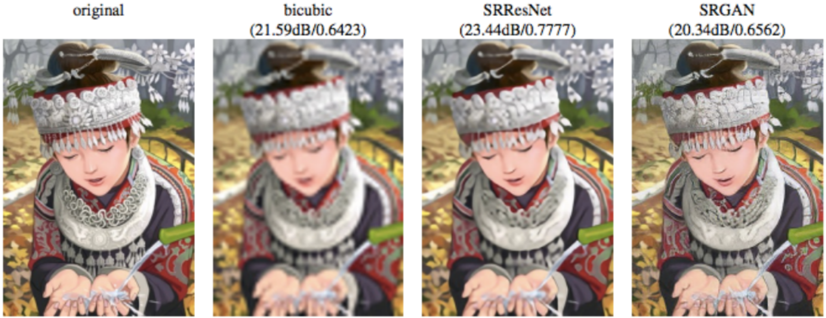


Figure 4: Source: [Goodfellow, 2016]

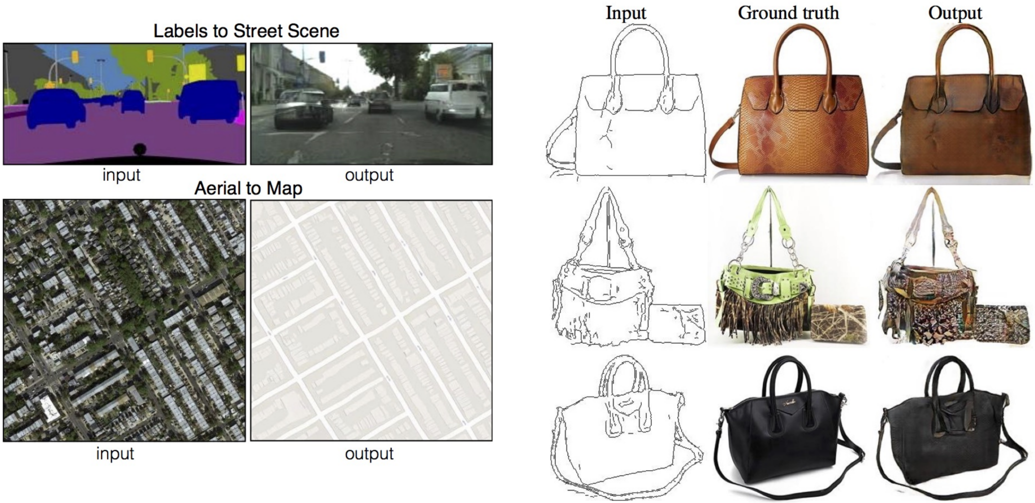


Figure 5: Source: [Goodfellow, 2016]

Application — Create art

Figure 6: Source: [Elgammal et al., 2017]

Application — Image inpainting

Figure 7: Source: [Demir and Unal, 2018]

Application — Face synthesis

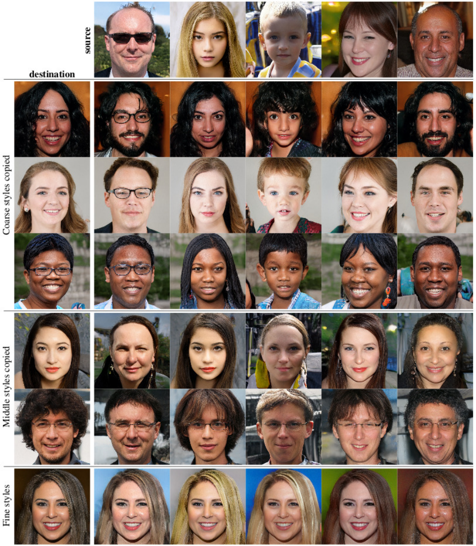


Figure 8: GAN behind the `this<thing>doesnotexist.com` sites. Source: [Karras et al., 2018]

GENERATIVE ADVERSARIAL NETWORKS

Outline

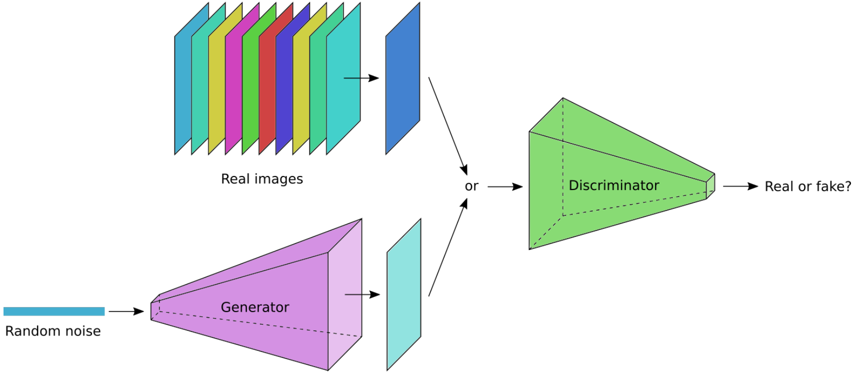
- General introduction
- Cost functions
- Challenges
- Tips and tricks

Figure 9: Source: https://deephunt.in/the-gan-zoo-79597dc8c347

- Introduced by Ian Goodfellow et al. in 2014 [Goodfellow et al., 2014]
- General idea from game theory
- Analogy
  - Counterfeiter creating fake money
  - Police trying to distinguish fake money from real money
  - The better the counterfeiter gets, the better the police gets
  - The better the police gets, the better the counterfeiter gets
- Yann LeCun dubbed adversarial training the most interesting idea in ML the last 10 years
- The first part will describe the original GAN

Components

- A *generator* function that tries to create real-looking examples
- A *discriminator* function that tries to distinguish real from fake examples
- Functions are updated in a feedback loop, making each better at its task



- The discriminator is a function

$$D : x \mapsto D(x; \theta_D)$$

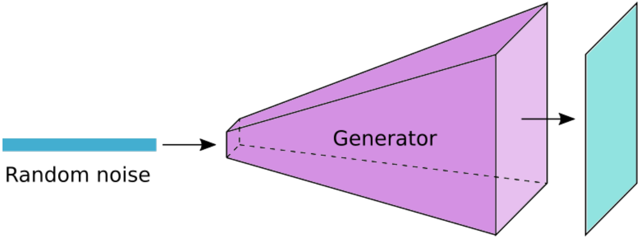
  mapping input x to $D(x; \theta_D)$ with parameters θ_D
- The generator is a function

$$G : z \mapsto G(z; \theta_G)$$

  mapping input z to $G(z; \theta_G)$ with parameters θ_G
- The discriminator has an associated loss $J_D(\theta_D, \theta_G)$, depending on both θ_D and θ_G, but can only control θ_D
- The generator has an associated loss $J_G(\theta_D, \theta_G)$, depending on both θ_D and θ_G, but can only control θ_G
- The optimal solution (θ_D^*, θ_G^*) is a *Nash equilibrium* where
  - θ_D^* is a local minimum of J_D w.r.t. θ_D
  - θ_G^* is a local minimum of J_G w.r.t. θ_G

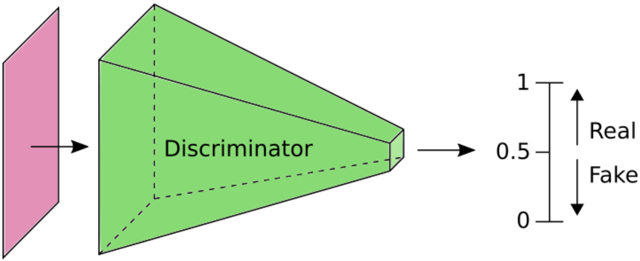
The generator

- The generator is a differentiable function
- The input z is a random vector sampled from some simple prior distribution p_g
- The output $x = G(z)$ is then sampled from p_m
- The most common form of G is some kind of generative neural network
- If we have GAN trained on data from p_d, we can use the generator to sample from p_m
- $p_m \approx p_d$
- With this, samples from the generator will look like the training data



The discriminator

- The discriminator is a standard classification network
- Trained to differentiate between real and fake (generated) images
- Outputs a single number in $[0, 1]$
  - $D(x) = 0$: D believes x is fake
  - $D(x) = 1$: D believes x is real



THE TRAINING PROCESS

- At each update step, one mini-batch x of real images, and one mini-batch z of latent vectors are drawn
- z is fed through G, producing $G(z)$
- $D(x)$ is compared with $D(G(z))$
- θ_G is updated using gradients from J_G
- θ_D is updated using gradients from J_D
- The discriminator and generator are updated in tandem using some regular optimization routine (SGD, Adam, etc.)
- Some flexibility with regards to updating one more often than the other

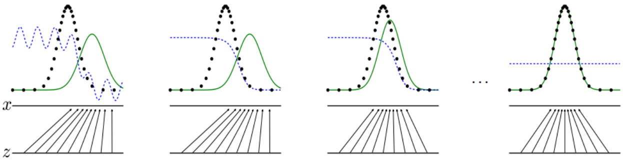


Figure 10: Source: [Goodfellow et al., 2014]

The training process

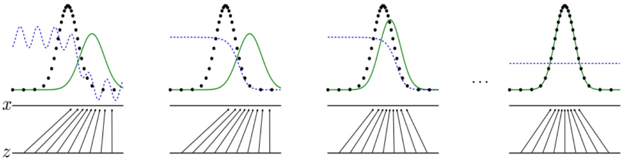


Figure 11: Source: [Goodfellow et al., 2014]

- Black arrows illustrate the mapping $z \mapsto G(z)$
- Black probability density is the data distribution p_d
- Blue probability density is the discriminator distribution
- Green probability density is the generative distribution p_m
- The generative distribution distinguishes between real and generated data
- From (a) to (d): The generative distribution (green) is guided towards high probable areas of the discriminative distribution (blue)
- The process terminates when the discriminative distribution becomes constant (is no longer able to distinguish real from fake)

Cost functions

- The generator, G, and discriminator, D, are two distinct networks with distinct cost functions
- The cost functions are optimized separately
- The cost function of D should push D to be able to classify real and fake images
- The cost function of G should push G to generate real-looking images
- Define the error function

$$E(D,G) = \mathbb{E}_{x \sim p_d}[1 - D(x)] + \mathbb{E}_{x \sim p_m}[D(x)]$$

- The discriminator should minimize the error, and the generator should maximize it

$$\max_G\{\min_D\{E(D,G)\}\}$$

- Which is conseptually the goal of a GAN

Discriminator cost function

- Instead of the minimizing the absolute error (previous slide), we minimize the cross-entropy error

$$E(D,G) = \mathbb{E}_{x\sim p_d}[-\log D(x)] + \mathbb{E}_{x\sim p_m}[-\log(1-D(x))]$$

- Or, in terms of the generated distribution

$$E(D,G) = \mathbb{E}_{x\sim p_d}[-\log D(x)] + \mathbb{E}_{z\sim p_g}[-\log(1-D(G(z)))]$$

- With discrete samples, over one mini-batch $\{x_i\}$ and $\{z_i\}$, the discriminator cost is

$$J_D(\theta_D,\theta_G) = -\frac{1}{m}\sum_{i=1}^{m}[\log(D(x_i;\theta_D)) + \log(1-D(G(z_i;\theta_G);\theta_D))]$$

Generator cost function

- The discriminator and generator have opposite goals
- We could in principle minimize the negative discriminator cost

$$J_G(\theta_D, \theta_G) = -J_D(\theta_D, \theta_G)$$

- The generator is not dependent on p_d, so the loss becomes

$$J_G(\theta_D, \theta_G) = \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z_i; \theta_G); \theta_D))$$

- This is sometimes referred to the minimax GAN
- This cost function has some problems that we will come back to later
- The current formulation provides convenient theoretical insights

Summary

- Error function

$$E(D,G) = \mathbb{E}_{x\sim p_d}[-\log D(x)] + \mathbb{E}_{x\sim p_m}[-\log(1 - D(x))]$$

- Find optimal parameters by training two networks

$$\theta_D^* = \arg\min_{\theta_D}\{E(D(\theta_D), G(\theta_G))\}$$
$$\theta_G^* = \arg\max_{\theta_G}\{E(D(\theta_D), G(\theta_G))\}$$

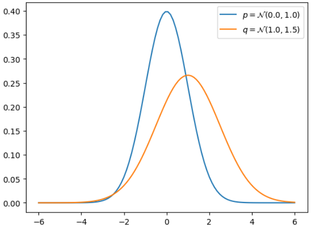
- Before we return to a more useful generator loss, we are going to analyse this result
- Outline:
    - KL-divergence vs. JS-divergence
    - A closer look at the discriminator cost function
    - Consequences

- We are comparing the distributions p_X and q_X over some discrete random variable X
- The Kullbach-Leibler (KL) divergence is given by

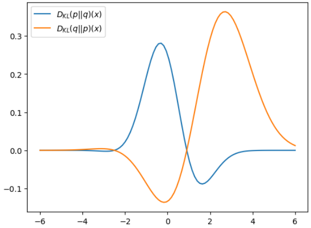
$$D_{KL}(p_X || q_X) = \sum_x p_X(x) \log \frac{p_X(x)}{q_X(x)}$$

- This is an asymmetric distance metric, meaning that, *in general*

$$p_X \neq q_X \to D_{KL}(p_X || q_X) \neq D_{KL}(q_X || p_X)$$



(a) Two unequal distributions as a function of x



(b) KL-divergence kernel as a function of x

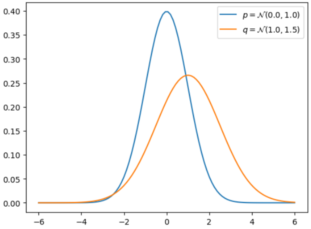
- Let p_X and q_X be as above, and let their mixture be

$$g_X = \frac{1}{2}(p_X + q_X)$$

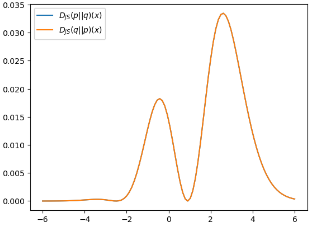
- The Jensen-Shannon (JS) divergence is then given by

$$D_{JS}(p_X||q_X) = \frac{1}{2}D_{KL}(p_X||g_X) + \frac{1}{2}D_{KL}(q_X||g_X)$$

- This is a symmetrized and smoothed version of the KL divergence

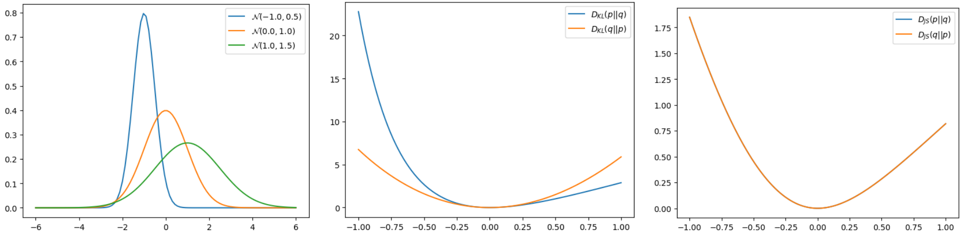


(a) Two unequal distributions as a function of x



(b) JS-divergence kernel as a function of x

- In these figures, the KL-divergences and JS-divergences are computed for a range of distribution comparisons
- The reference distribution is $p = \mathcal{N}(0.0, 1.0)$
- The comparison distributions is $q = \mathcal{N}(\mu, \sigma^2)$ with, simultaneously
  - μ ranging from -1.0 to 1.0
  - σ^2 ranging from 0.5 to 1.5



(a) Range of normal distributions, blue to green
(b) KL-divergence of range
(c) JS-divergence of range

Figure 14: Comparison of KL-divergence and JS-divergence

Optimal discriminator loss

- What value of $D(x)$ is minimizing the error function?

$$\begin{aligned}
E(D,G) &= J_D \\
&= \int_x -p_d(x)\log_2(D(x)) - p_m(x)\log_2(1-D(x))\,\mathrm{d}x \\
&= \int_x \tilde{E}(D,G)(x)\,\mathrm{d}x
\end{aligned}$$

where $\tilde{E}(D,G)(x)$ is the integrand.

- From variational calculus, we have that (the functional derivative is)

$$\begin{aligned}
\frac{\mathrm{d}E(D,G)}{\mathrm{d}D(x)} &= \frac{\mathrm{d}\tilde{E}(D,G)}{\mathrm{d}D(x)} \\
&= -\left[p_d(x)\frac{1}{\ln(2)}\frac{1}{D(x)} - p_m(x)\frac{1}{\ln(2)}\frac{1}{1-D(x)}\right] \\
&= -\frac{1}{\ln(2)}\left[\frac{p_d(x)}{D(x)} - \frac{p_m(x)}{1-D(x)}\right]
\end{aligned}$$

Optimal discriminator loss

- Equating the derivative with zero yields the optimal discriminator value

$$
\begin{aligned}
0 &= \frac{\mathrm{d}E(D,G)}{\mathrm{d}D(x)} \\
&= -\frac{1}{\ln(2)}\left[\frac{p_d(x)}{D^*(x)} - \frac{p_m(x)}{1-D^*(x)}\right] \\
D^*(x) &= \frac{p_d(x)}{p_d(x)+p_m(x)}
\end{aligned}
$$

- Moreover, when the generator is working optimally $p_m = p_d$, and therefore

$$D^*(x) = \frac{1}{2}$$

- Inserting the optimal generator $G^*(x)$, and discriminator $D^*(x) = \frac{1}{2}$, back into the error function, we get

$$\begin{aligned} E(D^*, G^*) &= -\int_x p_d(x) \log \frac{1}{2} + p_m(x) \log \frac{1}{2} \, \mathrm{d}x \\ &= -\log \frac{1}{2} \left[\int_x p_d(x) + p_m(x) \, \mathrm{d}x \right] \\ &= -2 \log \frac{1}{2} \\ &= 2 \log 2 \end{aligned}$$

- To be clear: this is the value of the discriminator loss when using the discriminator that minimizes the loss, and the generator that samples from the (approximate) data distribution

Optimal discriminator loss and relation to Jensen Shannon divergence

- We had JS divergence

$$D_{JS}(p_X||q_X) = \frac{1}{2}D_{KL}(p_X||\frac{1}{2}(p_X + q_X)) + \frac{1}{2}D_{KL}(q_X||\frac{1}{2}(p_X + q_X))$$

- The JS divergence between p_d and p_m is then

$$\begin{aligned}
D_{JS}(p_d||p_m) &= \frac{1}{2}\left(\int_x p_d \log(2\frac{p_d}{p_d + p_m})\,\mathrm{d}x + \int_x p_m \log(2\frac{p_m}{p_d + p_m})\,\mathrm{d}x\right) \\
&= \frac{1}{2}\left(\log 2 + \int_x p_d \log\frac{p_d}{p_d + p_m}\,\mathrm{d}x + \log 2 + \int_x p_m \log\frac{p_m}{p_d + p_m}\,\mathrm{d}x\right) \\
&= \frac{1}{2}\left(2\log 2 + \int_x p_d \log D^*\,\mathrm{d}x + \int_x p_m \log(1 - D^*)\,\mathrm{d}x\right) \\
&= \frac{1}{2}(2\log 2 - E(D^*, G))
\end{aligned}$$

- Notice that for an optimal discriminator *and generator* (previous slide), $D_{JS}(p_d||p_m) = 0$

- From the result on the previous slide, we get an expression for the discriminator loss with an optimal discriminator

$$E(D^*, G) = 2\log 2 - 2D_{JS}(p_d||p_m)$$

- Maximizing the error is the goal of the generator
- Given an optimal discriminator, this is equivalent to minimizing the JS-divergence

The generator cost function

- In the discriminator, we are minimizing the cross-entropy between the target distribution and the generated distribution
- It has strong gradients when the classifier is wrong
- The gradients saturates when the classifier is right, but this is not as important
- For the generator objective, we have found a candidate with convenient theoretical interpretations
- Unfit in practice: When the discriminator successfully rejects generated examples with high confidence, the gradients of the generator loss vanishes
- We must find a generator loss that does not saturate at unwanted places

The generator cost function

- For the generator cost, we propose

$$J_G(\theta_D, \theta_G) = -\mathbb{E}_{z \sim p_g} \log D(G(z; \theta_G); \theta_D)$$
$$= -\frac{1}{m} \sum_{i=1}^{m} \log D(G(z_i; \theta_G); \theta_D)$$

- With this, the generator maximizes the log-probability of the discriminator being mistaken (assigning label 1 to the generated examples)
- Contrast this with the previous minimax game where we the generator minimizes the log-probability of the discriminator being correct (assigning label 0 to the generated examples)
- Both the generator and the discriminator now have strong gradients when they are "losing the game"

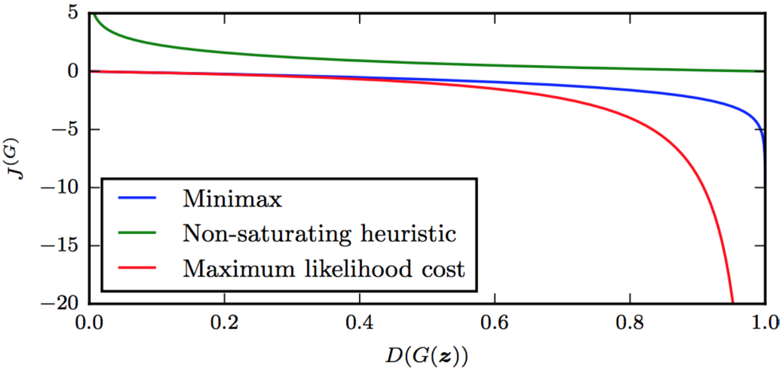


Figure 15: Graph of the gradient cost w.r.t. discriminator classification. Source: [Goodfellow, 2016]

- Minimizing the discriminative cost

$$J_D = -\frac{1}{m}\sum_{i=1}^{m}[\log(D(x_i)) + \log(1 - D(G(z_i)))]$$

"pushes" $D(x)$ to 1 (real class) and $D(G(z))$ to 0 (fake class)

- Minimizing the generative cost

$$J_G = -\frac{1}{m}\sum_{i=1}^{m}\log(D(G(z_i)))$$

"pushes" $D(G(z))$ to 1 (real class)

- This is sometimes referred to as non-saturating GAN

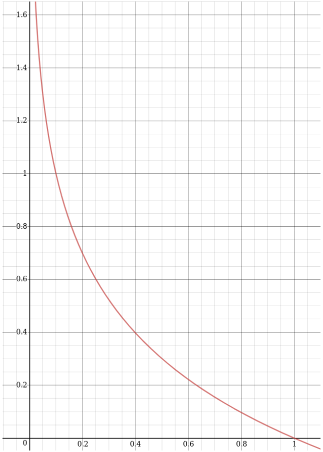


Figure 16: Graph of $f(x) = -\log x$

GAN challenges

- Convergence
- Performance evaluation
- Discrete output

- Achieving convergence is in general difficult
- The solutions tends to oscillate
- This is connected to that one try to achieve an equilibrium in stead of a plain optimization
- One major problem is connected to what is called *mode collapse*

Mode-collapse

- A peak in the probability density is called a mode
- Real-world data tends to be *multimodal*
- This means that similar examples are clustered in separate locations
- The data probability distribution will have peaks (modes) at these locations
- Mode-collapse is the phenomenon where the generator tends to generate very similar examples
- These similar examples originates from roughly the same location in the model distribution
- This location has high probability in the data distribution.

Mode-collapse — example

- Suppose we have a dataset with two modes, around A and B
- You want the GAN to generate examples from both A and B
- Mode collapse can be described as follows
    1. The generator produces examples close to A which “fools” the discriminator
    2. The discriminator classifies x from B as real with high probability, x from A are classified $50/50$ as real or fake
    3. The generator is then driven to produce examples from B
    4. The discriminator counters, and classifies examples x from A as real and examples from B real or fake with 50% probability
    5. This cycle then repeats from 1.

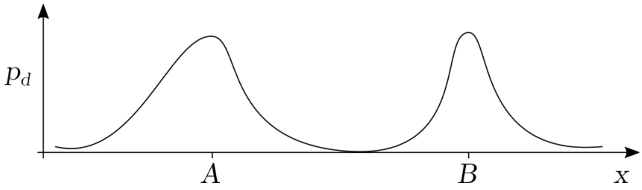


Figure 17: Bimodal data distribution

- Partial mode collapse is more common than complete mode collapse
- Generated images then tend to have the same colors, or some of the same features

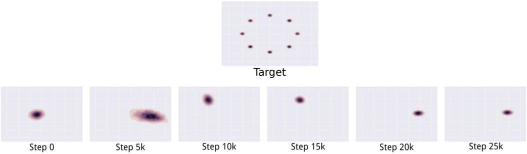


Figure 18: Mode collapse on data of a mixture of gaussians. Source: [Metz et al., 2016]

- Results from generative models can be hard to quantify and evaluate
- Often, in terms of images, perceptual similarity is important
- Other generative models may have an explicit objective function
- GANs lack this, which makes it even harder
- Attempts include Inception Score (IS), and Frechet Inception Distance (FID)

- We will present some useful tricks for GAN
- Some are related to preventing the mode collapse problem
- See [Salimans et al., 2016] and [Goodfellow, 2016] for a more thorough discussion

Minibatch discrimination

- The discriminator in a standard GAN compares single examples
- The idea is to aid this comparison with information from the whole mini-batch of real and generated examples
- The rationale is that the discriminator can detect if one example is unusually similar to other generated examples
- This technique is shown to work quite well

Feature matching

- This is related to the minibatch discrimination
- Also attempts addressing the mode-collapse problem by increasing diversity
- Extends (or replaces) the discriminator loss with a comparison of intermediate features from both the real and generated data
- In stead of explicitly discriminating on the output, we also discriminate on hidden layers

Train with labels

- If you have a labeled training set, use the labels
- If you have K classes, add the fake data as class $K + 1$
- The discriminator now tries to classify examples as one of $K + 1$ classes
- This improves the perceptual quality of generated examples
- This technique can be used in semi-supervised learning

One-sided label smoothing

- Neural network classifiers tend to classify with too high confidence
- We can encourage the discriminator to produce more soft predictions
- Set the true label for the real samples to be 0.9 in stead of 1
- This penalizes models producing too large logits on real samples
- Important to not smooth the generated sample label

- Batch normalization in GAN is, in general, very useful
- Batch normalization is not ideal for small batch sizes as the mean and variance varies too much between batches
- This is problematic for GANs as these fluctuations can dominate over the latent variable z in the generator (see figure below)
- Reference batch norm and virtual batch norm can aid this [Goodfellow, 2016]

Figure 19: GAN on ImageNet. Source: [Goodfellow, 2016]

Decent looking examples

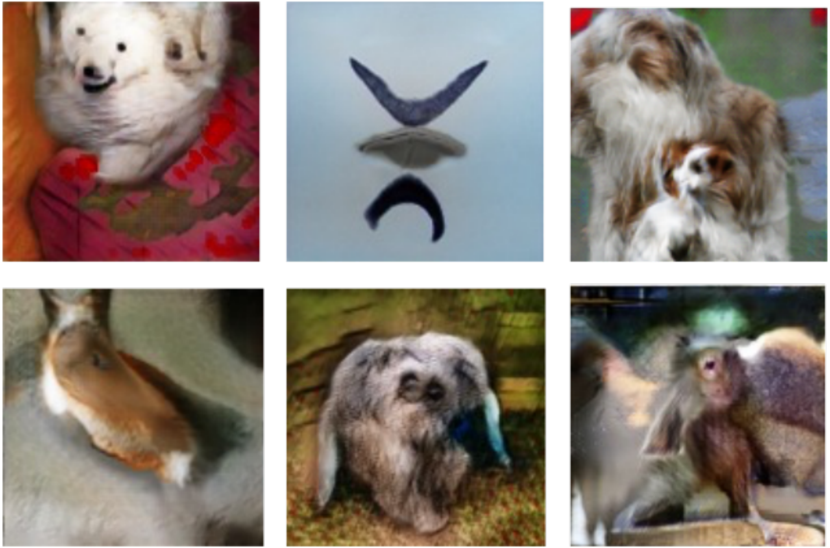


Figure 20: GAN on ImageNet. Source: [Goodfellow, 2016]

Problems with counting

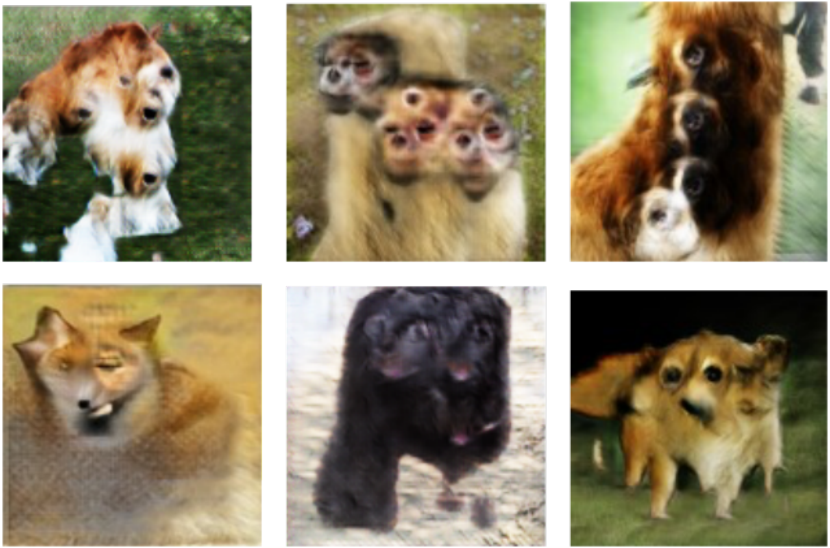


Figure 21: GAN on ImageNet. Source: [Goodfellow, 2016]

PROBLEMS WITH 3D

Figure 22: GAN on ImageNet. Source: [Goodfellow, 2016]

Problems with anatomy and structure

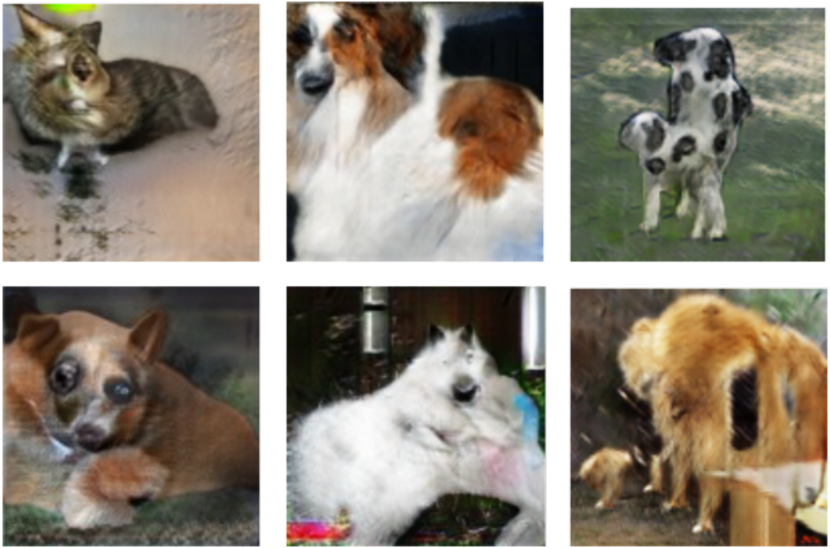


Figure 23: GAN on ImageNet. Source: [Goodfellow, 2016]

NOTABLE GAN VARIANTS

DCGAN

- GANs have gained a lot of interest
- For an impression of the amount of models, take a look at this post: `https://deephunt.in/the-gan-zoo-79597dc8c347`
- We are only going to look briefly at DCGAN and WGAN
- Chosen because of their popularity

DCGAN

- *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* [Radford et al., 2016]
- Wants to learn good intermediate image representations from unlabeled data
- VAEs and standard GANs produces generates blurry images
- GANs are difficult to train and can generate non-sensical results
- DCGAN enables the coupling of CNNs with GANs

DCGAN — INTRODUCED CHANGES

- Uses techniques from the (then) resent lessons learned
- Replaces deterministic spatial pooling operations (such as maxpool) with learned spatial up- and down-sampling
  - strided convolutions for the discriminator
  - fractionally strided convolutions (transposed convolutions) for the generator
- Elimination of dense layers on top of the convolutional layers at the end of the networks
- Use batch-normalization between layers in both the generator and discriminator
- Use ReLU activation in the generator (except in the output layer, which uses tanh)
- Use leaky ReLU activation in the discriminator

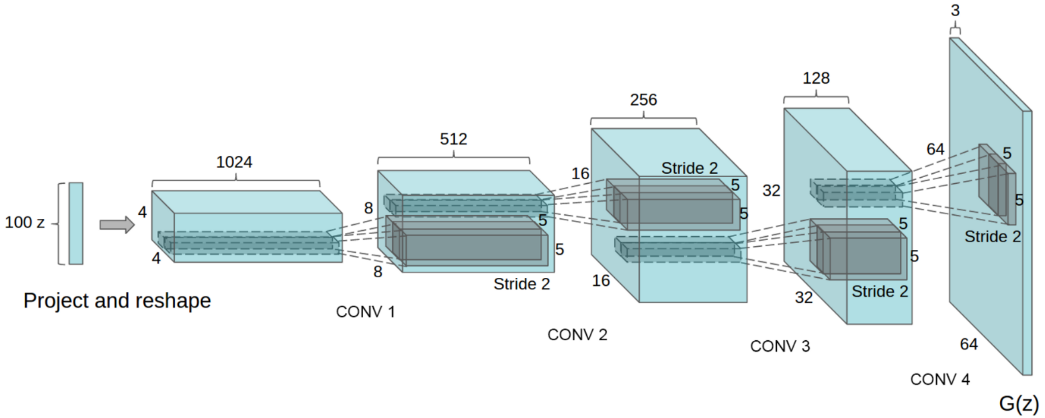


Figure 24: DCGAN generator architecture. Source: [Radford et al., 2016]

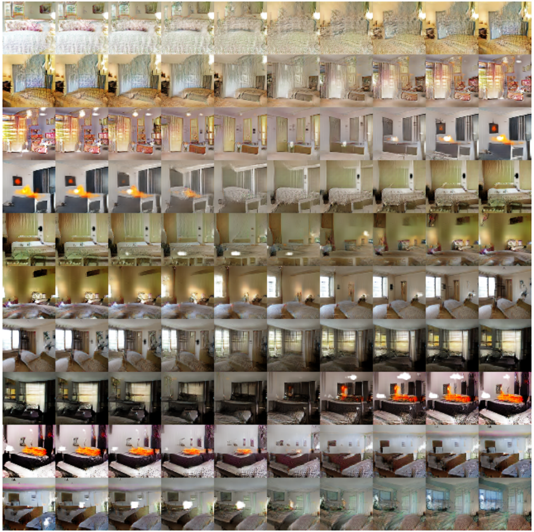


Figure 25: Bedroom interpolation

DCGAN — Results

Figure 26: Faces looking left to faces looking right

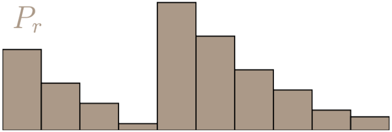
WGAN — Wasserstein Generative Adversarial Network

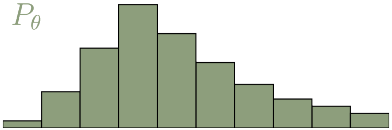
- Introduced in 2017, [Arjovsky et al., 2017]
- Claims to solve, or reduce many of the problems with training GANs
- Is based on the Wasserstein distribution similarity metric
- Has become quite popular with > 1600 citations in little over two years
- It is quite technical, so we will only look at the Wasserstein metric

Wasserstein distance

- Also known as *earth mover distance*
- Intuitively easy to grasp
- Quite complicated to derive, compute, and fully understand
- We will only concern ourself with the intuition
- For more details, I refer to `https://vincentherrmann.github.io/blog/wasserstein/`,
- The figures for this section are from the above resource

- It measures the smallest amount of "work" that needs to be done in order to transform one distribution to the other.
- Let our distributions be P_r and P_θ





- Let $\gamma(x, y)$ be the difference between $P_r(x)$ and $P_\theta(y)$
- The Wasserstein distance is then

$$W(P_r, P_\theta) = \inf_{\gamma \in \Gamma} \sum_{x,y} ||x - y|| \gamma(x, y)$$

- Here Γ contains all "valid" γ
- $\inf$ means *infimum* and can be thought of as the greatest lower bound

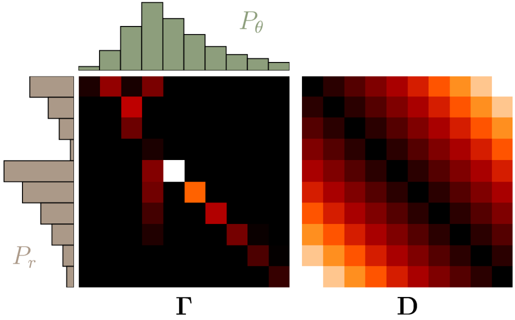
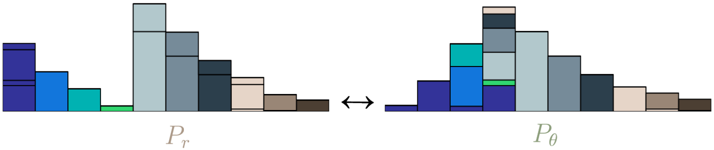


Figure 28: Optimal solution



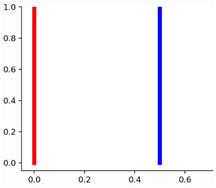


Figure 30: Two point distributions

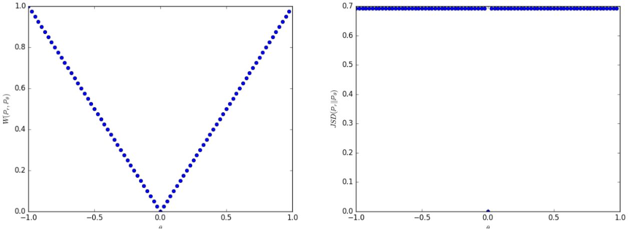


Figure 31: Wasserstein distance (left) and JS-divergence (right) when the above two distributions come closer, overlap, and then move away from each other again. Source: [Arjovsky et al., 2017]

ADVERSARIAL DOMAIN ADAPTATION

Introduction

- Generalization of results on training to the real world is crucial for a useful method
- This can be hard enough when training and test comes from the same distribution
- Even worse when the train and test data comes from different distributions, known as *domain shift* or *dataset bias*
- Domain adaptation methods addresses this problem
- Adversarial domain adaptation methods uses principles from GANs
- In essence, they try to train models that are invariant to the dataset domain by trying to fool a discriminator that tries to classify domains

Case study: Adversarial discriminative domain adaption

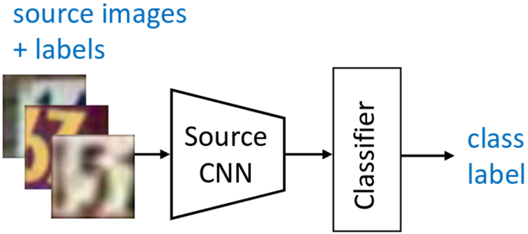
- There are several approaches to this problem
- Notable works are e.g.
  - Gradient reversal [Ganin et al., 2016]
  - Domain confusion [Tzeng et al., 2015]
  - CoGAN [Liu and Tuzel, 2016]
  - ADDA [Tzeng et al., 2015]
- The adversarial discriminative domain adaption (ADDA) is illustrated because of its simplicity and performance

ADDA: OVERVIEW

- We have labeled data (x_s, y_s) for the source domain
- The target domain data, y_t is unlabeled
- We are going to learn a source mapping $M_s : x_s \mapsto y_s$
- We are also going to learn a target mapping $M_t : x_t \mapsto y_t$
- The target mapping should be invariant to the domain difference between the source and the target
- We are going to use a discriminator with an associated loss to learn this domain invariance

ADDA: Example mappings

- For the source mapping M_s, we can use a standard classification network with cross-entropy loss
- The target mapping M_t is equal to M_s, except for the classifier part, but with separate and independent parameters
- The parameters of M_t are initialized with the parameters of a trained M_s
- M_s is fixed when M_t is trained



ADDA: Discriminator

- The discriminator D should classify outputs of these networks as either originating from the source or the target domain
- This is a similar situation as with regular GANs, and the loss is

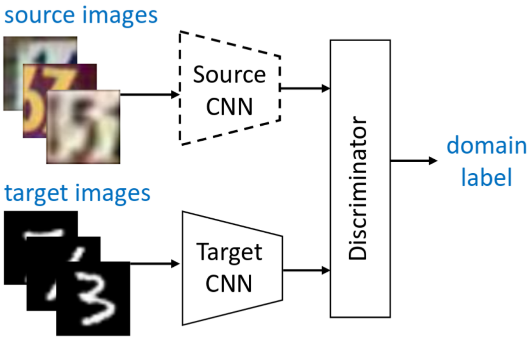
$$J_D(M_s, M_t) = -E_s\left[\log D(M_s(x_s))\right] - E_t\left[\log(1 - D(M_t(x_t)))\right]$$

- The discriminator wants to maximize the probability that it predicts the correct domain
- The target mapping should produce examples that maximizes the probability of being classified as coming from the source
- We therefore chose the generator loss from GANs

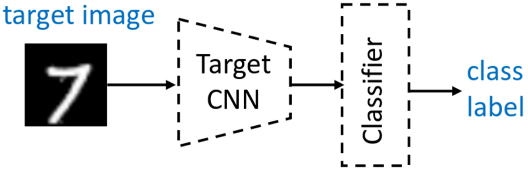
$$J_M(M_s, M_t) = -E_t\left[\log D(M_t(x_t))\right]$$

- E_d is the expectation over examples in $d \in \{\text{source}, \text{target}\}$

ADDA: Discriminator



- We now have a classifier that can classify examples from features
- We also have a base mapping M_t that should generate domain-invariant features
- We reuse those parts in the testing



References I

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. *arXiv preprint*, 2017. URL https://arxiv.org/pdf/1701.07875.pdf.

Ugur Demir and Gozde Unal. Patch-Based Image Inpainting with Generative Adversarial Networks. *arXiv preprint*, 2018. URL http://arxiv.org/abs/1803.07422.

Ahmed Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. CAN: Creative Adversarial Networks, Generating "Art" by Learning About Styles and Deviating from Style Norms. *arXiv preprint*, (Iccc):1–22, 2017. doi: 10.1089/cyber.2017.29084.csi. URL http://arxiv.org/abs/1706.07068.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, Francois Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 2016.

References II

Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. *arXiv preprint*, 2016. ISSN 0253-0465. doi: 10.1001/jamainternmed.2016.8245. URL http://arxiv.org/abs/1701.00160.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv preprint*, pages 1–9, 2014. ISSN 10495258. doi: 10.1001/jamainternmed.2016.8245. URL http://arxiv.org/abs/1406.2661.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. URL http://arxiv.org/abs/1812.04948.

Karol Kurach, Mario Lucic, Xiaohua Zhai, Marcin Michalski, and Sylvain Gelly. The GAN landscape: Losses, architectures, regularization, and normalization. *CoRR*, abs/1807.04720, 2018. URL http://arxiv.org/abs/1807.04720.

Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. *CoRR*, 2016.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled Generative Adversarial Networks. *arXiv preprint*, 2016. URL https://arxiv.org/pdf/1611.02163.pdf.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *ICLR 2016*, pages 1–16, 2016. URL http://arxiv.org/abs/1511.06434.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. *arXiv preprint*, 2016. doi: arXiv:1504.01391. URL https://arxiv.org/pdf/1606.03498.pdf.

Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. *International Conference in Computer Vision (ICCV)*, 2015.

QUESTIONS?