Generative Models

Generating novel data, adversarial training

Yordan Darakchiev
Technical Trainer
iordan93@gmail.com

sli.do
#DeepLearning

Table of Contents

- Generative models: basics
- Unsupervised data generation
  - Style transfer
  - Generating sequences
  - Variational autoencoders
- Generative adversarial networks

Generative Models

Laying out the basics

Generative Models Overview

- Supervised learning: $(x, y) \Rightarrow \tilde{y} = f(x): y \approx \tilde{y}$
- Unsupervised learning: $x \Rightarrow f(x): f$ describes structure
- Generative models intuition
  - Learn the distribution of training data $p_{train}(x) = f(x)$
  - Make the model output new data $p_{model}(x)$ "similar" to p_{train}
  - In other words: after training, make a model: $\tilde{y} = f(\emptyset)$
- Examples
  - PixelRNN / PixelCNN – image completion
  - DCGAN Image colorization
  - SRGAN – "super resolution"
  - Sequence generation: words, music, etc.; seq2seq models
  - Time series ⇒ planning (reinforcement learning)

Style Transfer

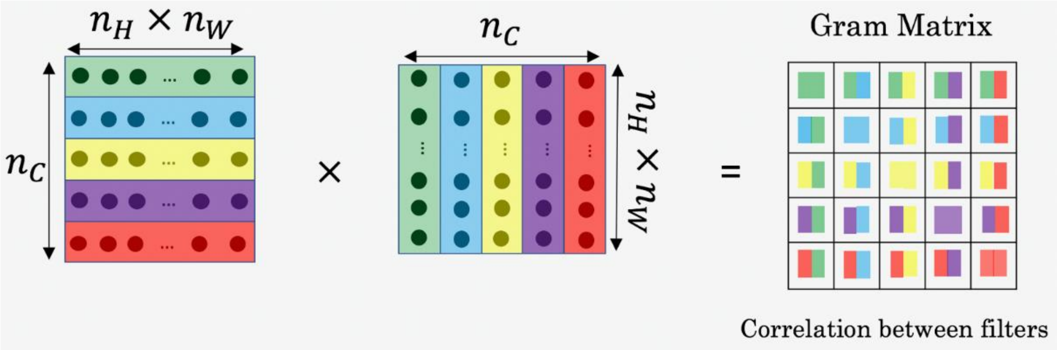
- Gatys et al., 2016
- A tutorial
- Objective
  - Input
    - Content image C, style image S
  - Output: image G with the content of C and style of S
- Start with a trained model (e.g. VGG-19) and train it further
- Loss function
  - Content loss L_c
  - Style loss L_s
  - Total loss: $L_{total} = \alpha L_c + \beta L_s$
    - α, β – numbers (relative contributions of the content and style)

Style Transfer (2)

- Content loss function
  - Extract activations from some middle layer l
  - Intuition
    - First layer: will force G to have very similar pixels to C
    - Last layer: "if there's a dog in C, draw a dog somewhere in G"
    - Therefore, it's better to choose a layer in between
      - Even better, treat l as a hyperparameter and try out a few architectures
- $L_{content}(C, G, l) = MSE\left(C^{[l]}, G^{[l]}\right)$
  - C – original image, G – generated image
  - Pass both C and G to the trained network
    - You may not propagate them fully to save time
    - Take the output activations from layer l: $C^{[l]}, G^{[l]}$
      - These are 3D volumes
    - Apply MSE

Style Transfer (3)

- Style loss function
- How do we define style?
  - Correlations between different filter responses
    - If we look at the activations at layer l, they will have the same style if they are correlated
    - Style matrix (Gram matrix): all dot products
      - Diagonal G_{ii}: how powerful each filter is
      - Other $G_{ij}, i \neq j$: how correlated filters i and j are



  - Compute style matrices for G and S, minimize the distance
    - $L_s(S, G, l) = MSE\left(\text{Style}_G^{[l]}, \text{Style}_S^{[l]}\right)$
  - Combine for many layers (with weights λ)
    - $L_{style}(S, G) = \sum_l \lambda_l MSE\left(\text{Style}_G^{[l]}, \text{Style}_S^{[l]}\right); \sum_l \lambda_l = 1$

Style Transfer (4)

- Training
  - Load the content and style images (C, S)
  - Generate G randomly
  - Load the trained model (e.g. VGG-19)
  - Pass C, S, G through the model, compute total loss
- Minimize total loss: standard optimization problem
  - Freeze all layers in VGG-19
    - This will change **the pixel values of G** to minimize the loss, **NOT the weights** of the trained model
- Obtaining a result
  - Take the updated image G
- Style transfer in `tensorflow`
  - An in-depth tutorial

Sequence Generators

Sampling novel sequences

Sequence Generation

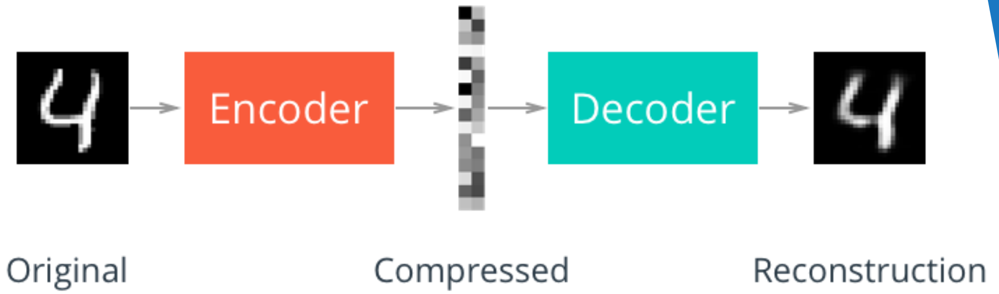
- We've already seen this
  - Language modelling
  - Music generation, more music generation (Magenta)
  - Image completion (PixelRNN)
- Training
  - As usual, train a one-to-many RNN model
  - Maximize the likelihood of input data (Bayes)
- Generation
  - Use a seed input (e.g. initial words, or a partial picture), predict next
    - Continue until the end (e.g. [.] / <EOS> character or until the picture is done)
- Drawback: generation can be slow
  - Especially if we augment it by using beam search or a similar algorithm

Variational Autoencoders

Autoencoders with a simple trick

Variational Autoencoders

- Train an AE; throw away E
- Try getting an output from D
  - Pass $l \equiv x = \vec{0}$ to the decoder
    - Won't generate different samples :(
  - Pass random values
    - Produces random noise at output :(



- How to create latent vectors so the AE works properly?
  - Constrain E to produce specific vectors, e.g. $l \sim N(0,1)$
    - You have now created a variational autoencoder (VAE)
  - Throw away E
  - Sample $x \sim N(0,1)$
    - D now "understands" the input even though it's completely random

Variational Autoencoders (2)

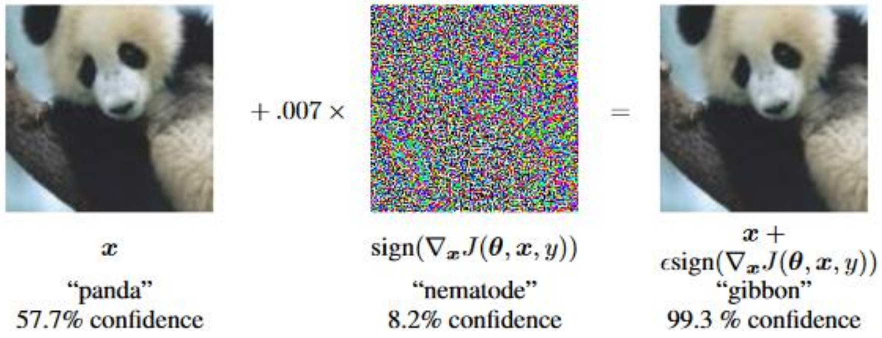
- Loss function
  - Generative loss: how well D reconstructs x from l: $\mathrm{MSE}(x, \tilde{y})$
  - Constraint loss: how close l is to $N(0,1)$: KL divergence
  - $L = \mathrm{MSE}(x, \tilde{y}) + \mathrm{KLD}\big(\tilde{y}, N(0,1)\big)$
- Results
  - MNIST, one more MNIST
  - Music (Roberts et al., 2017)
  - Time series (RL) (Gregor and Besse, 2018)
- Problems regarding generation
  - Distribution matching (Rosca et al., 2018)
  - Adding more constraints to VAEs (Rezende and Viola, 2018)

Generative Adversarial Networks

Learning how to be the best

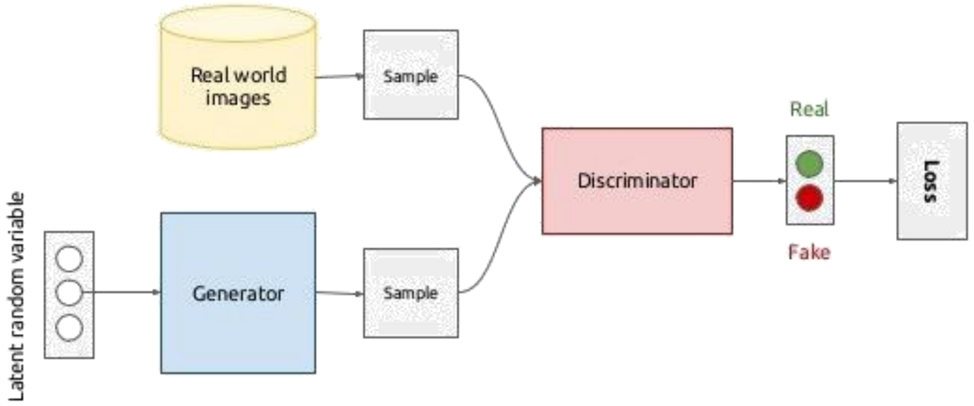
Adversarial Training

- Goodfellow et al., 2014a
- ML models misclassify examples that are only slightly different from correctly classified examples
  - "Overfitting"; many reasons
    - Look at the summary of the paper for a quick overview
  - Regularization alone doesn't help much
    - Adversarial training can serve as an additional regularizer
- Generating adversarial examples
  - Similar to style transfer (but much easier)
  - Pass a valid image and a noise G to the (frozen) neural network
  - Using the model gradients, update G to maximize the desired output



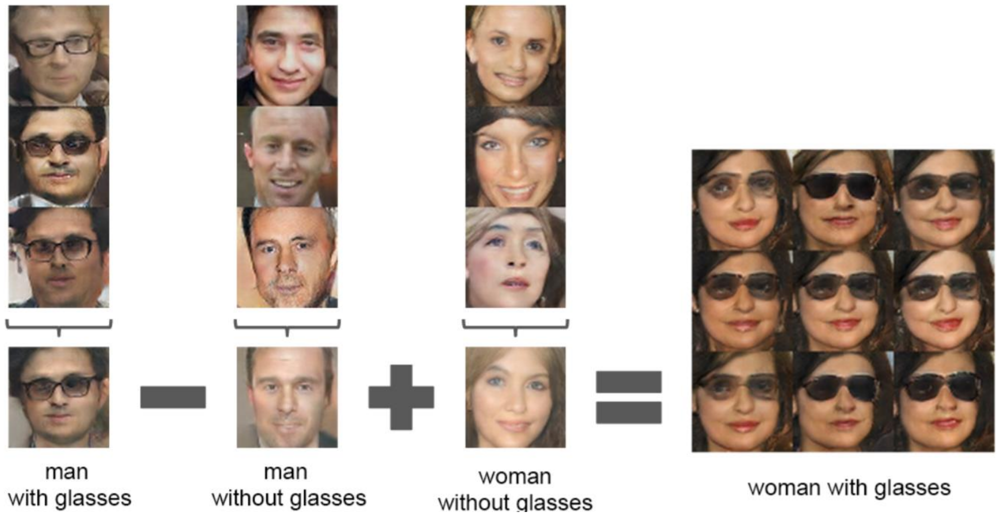
Generative Adversarial Networks

- Goodfellow et al., 2014b
  - One of the "cornerstone" papers of modern deep learning
- Approach: learn to generate from the training distribution by a two-player game
  - Instead of modelling vector densities (like we showed so far)
- High-level intuition
  - Generator network G: noise ⇒ output
  - Discriminator network D: input ⇒ is input fake?
  - Objective
    - Train together
    - D tries to distinguish between real and fake images
    - G tries to fool D



Generative Adversarial Networks (2)

- After training, use G to generate outputs
  - Provide random input
  - Take generated output
- Another useful property
  - Learned features can be interpreted (similar to word2vec)
- GAN intuition
- Lots of examples
- Many more examples
- 17 hacks to make GANs work



Summary

- Generative models: basics
- Unsupervised data generation
  - Style transfer
  - Generating sequences
  - Variational autoencoders
- Generative adversarial networks

Questions?