Deep Learning for Visual Computing

GANs & Transformers

Christopher Pramerdorfer
Computer Vision Lab, TU Wien

Topics

GANs continued

- Image to image translation
- Other GAN variants

Transformers

- Attention
- Vision transformers

Recall the following image restoration approach

- An encoder-decoder $\mathcal{G}$ for image restoration
- A binary classiifer $\mathcal{D}$ (critic) for identifying restored images

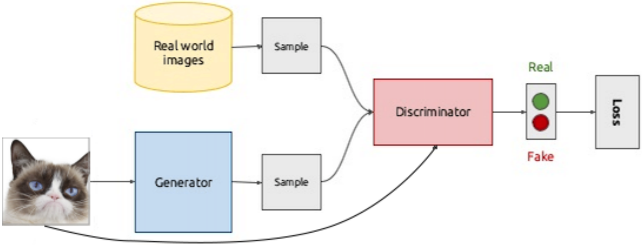

Image adapted from sigmoidal.io

This is an example of image to image translation

- $\mathcal{G}$ learns to map images from domain A to domain B
- In sense that $\mathcal{D}$ cannot distinguish between B and $\mathcal{G}(A)$

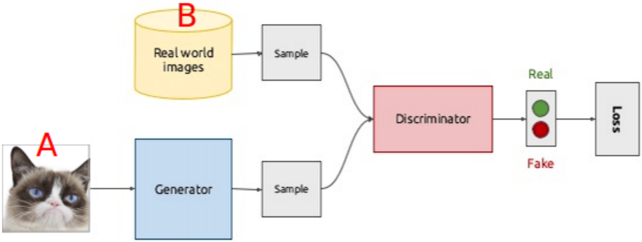

Image adapted from sigmoidal.io

Details depend on particular task and dataset properties

- Most importantly, are there corresponding image pairs?

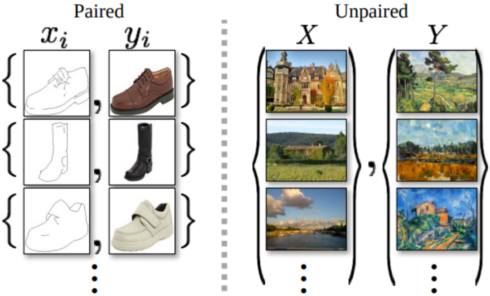

Image from [2]. $X \mathrel{\widehat{=}} A$ and $Y \mathrel{\widehat{=}} B$.

Discussed framework does not formally require image pairs

But in practice having no pairs leads to problems

- $\mathcal{G}$ will have hard time fooling $\mathcal{D}$

Further advantages of having pairs

- Can pre-train $\mathcal{G}$ to stabilize training
- Can add per-pixel loss on output of $\mathcal{G}$ to improve results

Example

- Domain A: images of maps
- Domain B: images of areal photos

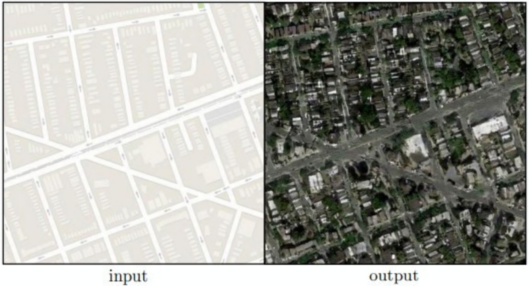

Image from [1]

Example

- Domain A: outlines of handbags
- Domain B: images of handbags

Image from [1]

GANs
Image to Image Translation

Example

- Domain A: grayscale images
- Domain B: color images

Image from [1]

GANs
Unpaired Image to Image Translation

What if image pairs are not available / possible?

- Photos to paintings, summer to winter etc.

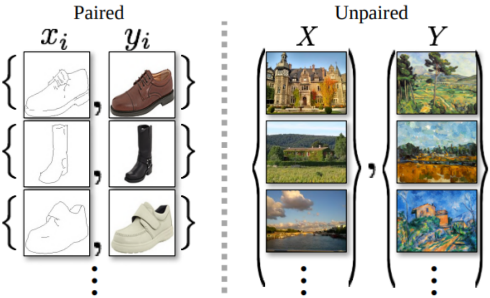

Image from [2]. $X \mathrel{\widehat{=}} A$ and $Y \mathrel{\widehat{=}} B$.

A popular method that makes this work is CycleGAN [2]

- Two generators $\mathcal{G} : A \mapsto B$ and $\mathcal{F} : B \mapsto A$

Core idea is that we demand cycle consistency

- $\mathcal{F}(\mathcal{G}(\mathbf{x})) \approx \mathbf{x}$ for $\mathbf{x} \in A$
- $\mathcal{G}(\mathcal{F}(\mathbf{y})) \approx \mathbf{y}$ for $\mathbf{y} \in B$

Generators should preserve images from target domains

- $\mathcal{G}(\mathbf{y}) \approx \mathbf{y}$ for $\mathbf{y} \in B$
- $\mathcal{F}(\mathbf{x}) \approx \mathbf{x}$ for $\mathbf{x} \in A$

Four corresponding per-pixel (L1) loss functions

- For cycle-consistency: $(\mathbf{x}, \mathcal{F}(\mathcal{G}(\mathbf{x})))$ and $(\mathbf{y}, \mathcal{G}(\mathcal{F}(\mathbf{y})))$
- For identity: $(\mathbf{y}, \mathcal{G}(\mathbf{y}))$ and $(\mathbf{x}, \mathcal{F}(\mathbf{x}))$

Plus two adversarial losses / discriminators $\mathcal{D}_A$ and $\mathcal{D}_B$

- Purpose identical to other GANs
- Distinguish real and fake images in domains A and B

Total training loss is weighted sum of all losses

GANs
Unpaired Image to Image Translation

Image from [2]

GANs
Unpaired Image to Image Translation

Image from youtube

GANs are a general framework

- Generate samples indistinguishable from training data
- Many flavors exist

Original GAN problem formulation

- Given samples $\mathbf{x}$ (training data) from some distribution $P(\mathbf{x})$
- Learn to sample from this distribution
- In sense that $\mathcal{D}$ cannot tell the difference

Got to make GAN stochastic

- Add a source of randomness
- To enable new random generations

To do so we replace input images to $\mathcal{G}$ with $\mathbf{z}$

- Fixed-size random vector, e.g. $\mathbf{z} \sim [-1, 1]^{20}$
- Called latent vector

Changes to $\mathcal{G}$

- Remove the encoder
- Instead map $\mathbf{z}$ to suitable 3D tensor

```
def generator(nz):
  return nn.Sequential(
    nn.Linear(nz, 64 * 8 * 8),
    nn.ReLU(inplace=True),
    nn.Unflatten(1, (64, 8, 8)),
    ...  # rest of generator
  )
```

Resulting architecture

- Training works like before (image restoration)

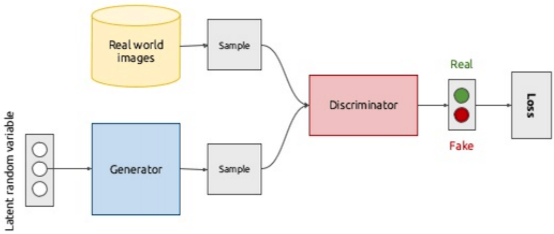

Image from sigmoidal.io

Training such GANs can be a pain (unable to pre-train)

- Can fail for many reasons

Two common issues

- $\mathcal{G}$ cannot fool $\mathcal{D}$ (vanishing gradients)
- $\mathcal{G}$ generates similar outputs regardless of $\mathbf{z}$ (mode collapse)

Loss curves help identifying problems

- Loss on $\mathcal{D}$ should never approach 0
- Oscillations are bad

GANs
Vanilla GANs

Image from [3]

No intuitive way to control what $\mathcal{G}$ should generate

If we train on CIFAR-10 for example

- Generated images will be of 10 classes
- But we might want to e.g. generate only car images

Conditional GANs overcome this limitation

Accomplished by adding another input $\mathbf{y}$ to $\mathcal{G}$ and $\mathcal{D}$

- Where $\mathbf{y}$ is what we want to condition on (e.g. class label)
- Causing $\mathcal{G}$ to learn to sample from $\Pr(\mathbf{x}|\mathbf{y})$

To generate random images of specific kind

- Choose $\mathbf{y}$ accordingly (e.g. desired class)
- Vary $\mathbf{z}$ to generate different samples

Conditional GANs

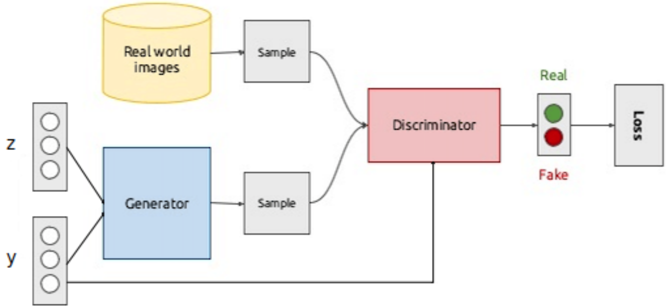

Image adapted from sigmoidal.io

Many ways to define and incorporate $\mathbf{y}$

Standard method for integer labels is an embedding

- Learnable function $e : \{0, 1, \ldots, m-1\} \mapsto \mathbb{R}^n$
- Maps positive numbers to fixed-size vectors
- n is a hyperparameter

Embedding layers (e.g. `torch.nn.Embedding`)

- Have $m \times n$ parameters (initialized randomly)
- One learned vector of size n per scalar input value (e.g. class)

Can set n freely

In our context we set $n = H_0 \cdot W_0$

- $H_0 \times W_0$ is initial tensor height/width in $\mathcal{G}/\mathcal{D}$
- Allows us to then reshape to $1 \times H_0 \times W_0$
- Can then concatenate information as new channel
- To combine image and e.g. class information in single tensor

Class-conditional images generated by [4]

Image from [4]

Transformers

We have focused on convolutional neural networks

- Dominating deep learning architecture for vision

Transformers are a new, strong competitor

- Originally proposed for natural language processing (NLP) [5]
- Recently adapted for vision (vision transformers) (ViT) [6]

Recall inductive biases we derived for convnets

- Local connectivity
- Parameter sharing within feature maps

These are solid inductive biases

- Sensible given how images are formed
- Significant reduction in parameters, computations over MLPs
- Enabling robust training of deep neural networks

The are still biases though

- Likely not (always) optimal
- Particularly with respect to slowly increasing receptive field

Transformers are more general

- Weaker task-specific inductive biases
- General architecture with exceptional performance

Transformers
Input

Transformers operate on sets of vectors $\{\mathbf{t}_1, \ldots, \mathbf{t}_S\}$

- Each vector $\mathbf{t}_s \in \mathbb{R}^T$ is called a token
- Token ordering must be encoded in $\mathbf{t}_s$ (if relevant)

The operation that characterizes transformers is attention

- Compare all tokens to each other (details later)
- Time and memory complexity is $\approx \mathcal{O}(S^2)$
- Can become a limiting factor quickly as S increases

Got to convert input images accordingly

- Cannot just represent pixels as tokens (S too big)

ViTs split images into S fixed-size patches (e.g. 16×16)

- Can be implemented using a conv layer with stride k

Image adapted from [6]

Transformers
Input

Patches are then flattened to vectors $\mathbf{v}_s$ (size e.g. $16 \cdot 16 \cdot 3$)

- Original ViT [3] maps these to tokens via $\mathbf{t}_s = \mathbf{v}_s \mathbf{W}$
- Learned weight matrix $\mathbf{W} \in \mathbb{R}^{\dim(\mathbf{t}_s) \times T}$ (linear layer)

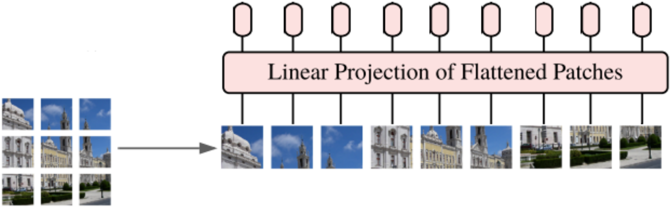

Image adapted from [6]

Patch location is certainly important (spatial information)

- Recall that we must encode this in $\mathbf{t}_s$
- ViT [6] uses an embedding layer

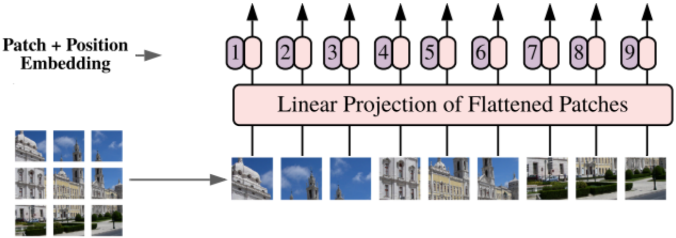

Image adapted from [6]

Transformers map input tokens to output tokens

- Add a dummy token (class embedding)
- Corresponding output token is input to classifier

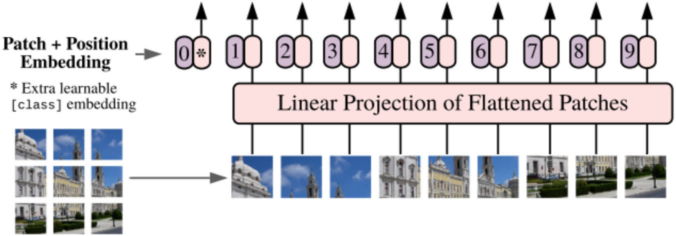

Image adapted from [6]

Transformers
For Image Classification

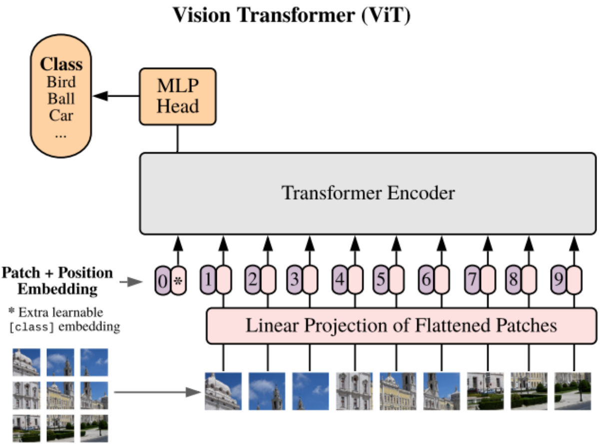

Image from [6]

Transformer encoder consists of L sub-networks

Each network has identical architecture

- Input: $S \times T$ matrix of input tokens
- Output: $S \times T$ matrix of output tokens
- Skip-connections like ResNet

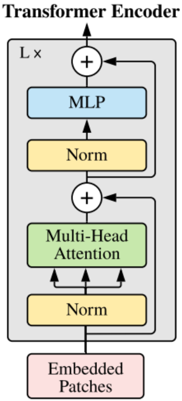

Image from [6]

Normalization layers (yellow)

- Normalize individual rows ($\mu = 0, \sigma = 1$)
- Using Layer normalization
- To keep signals well-behaved as usual

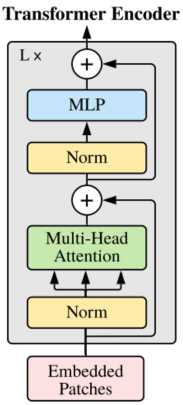

Image from [6]

Gaussian Error Linear Unit (GELU) activation

- Similar to ReLU
- Non-zero gradient for negative inputs

```
nn.Sequential(
    nn.Linear(dim, hidden_dim),
    nn.GELU(),
    nn.Dropout(dropout),
    nn.Linear(hidden_dim, dim),
    nn.Dropout(dropout)
)
```

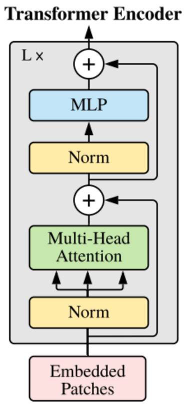

Image from [6]

Consists of h independent attention layers

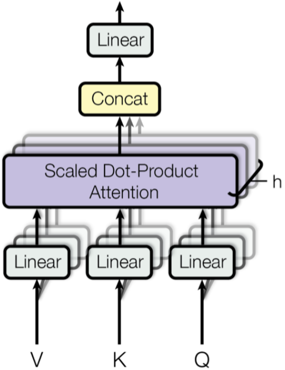

Image from [5]

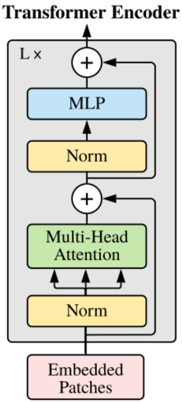

Image from [6]

Q, K, V are $S \times H$ matrices

- Different learned linear projections of input
- H is a hyperparameter

Attention is defined as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q\,K^T}{\sqrt{H}}\right) V$$

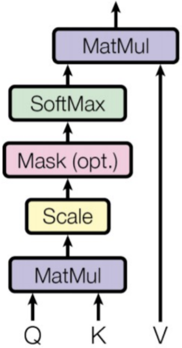

Image from [5]

$Q\,K^T$ is equal to dot products

- Between S vectors in Q and S vectors in K
- Output has size $S \times S$

Dot product encodes similarity between vectors

- Operation above computes similarity scores

Softmax normalizes these scores to $[0, 1]$

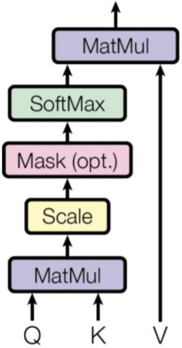

Image from [5]

Transformers

Transformer Encoder – Attention

Dot product softmax$(\cdot)\,V$

- Causes vectors of V to be suppressed
- If similarity score between Q and K was small

Encodes which vectors rest should focus on

- Vectors correspond to tokens / patches

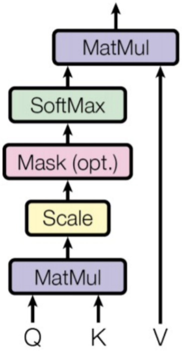

Image from [5]

Training is done as usual

- Backpropagation to compute $\nabla L(\boldsymbol{\theta})$
- Iterative optimization (Adam is popular)

Transformers have a global receptive field

- Every layer sees everything (in contrast to conv nets)

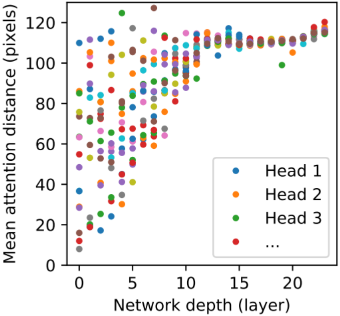

Image adapted from [6]

Transformers
Versus Conv Nets

Transformers might scale better to huge datasets

- Many millions of images

Transformers currently top many vision task leaderboards

- Advantage over modern conv nets is usually small though
- Not necessarily because of inherently better architecture

No clear winner in terms of training and inference times

- Varies a lot depending on particular architectures

Transformers
Remarks

Transformers are all the rage right now

- Fast progress

Transformers are the foundation of large language models (LLMs)

- Such as GPT-4 (ChatGPT), PaLM (Bard)

Bibliography

[1] Isola et al. Pix2Pix. 2016

[2] Zhu et al. CycleGAN. 2017

[3] Karras et al. StyleGAN. 2018

[4] Brock et al. Large-Scale GAN. 2019

[5] Vaswani et al. Transformers. 2017

[6] Dosovitskiy at al. Vision Transformers. 2020