Generative Adversarial Transformers

John Salvador

Sri Ram Pavan Kumar Guttikonda

Contents

Contributions

- Authored by Drew A. Hudson and C. Lawrence Zitnick
- Unifies GANs and Transformers into a single architecture
  - Allows different components to contribute to image generation
- Introduction of Bipartite attention for bilinear efficiency $O(mn)$
  - Better performance scaling for higher resolutions

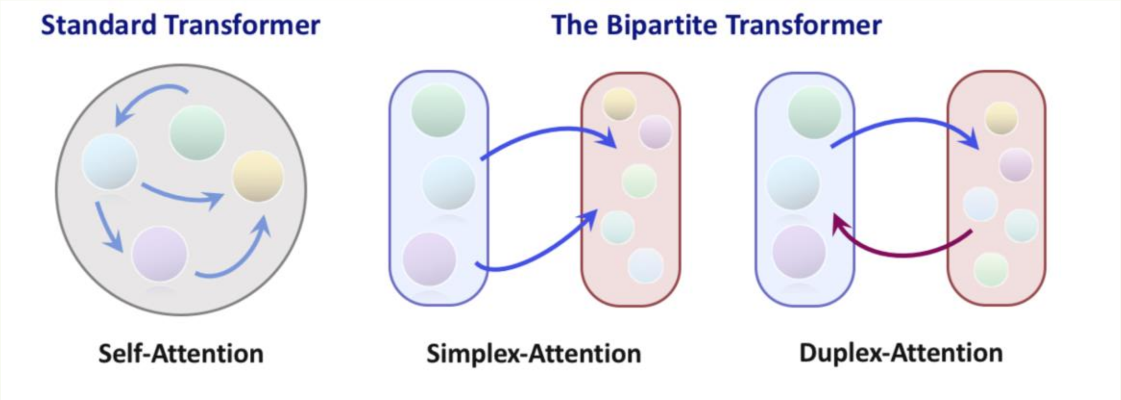
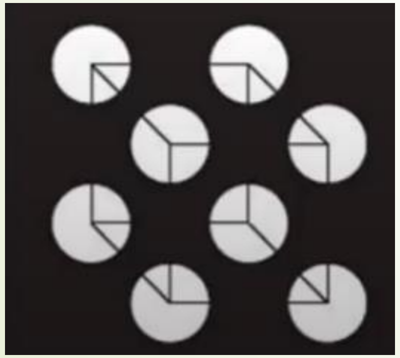


Image Source: Generative Adversarial Transformers

Contents

Inspiration From Human Perception

- the ***bottom up*** processing, proceeding from the retina up to the cortex, as local elements and salient stimuli hierarchically group together to form the whole.
- ***Top-down*** processing: Background knowledge is used to make inference



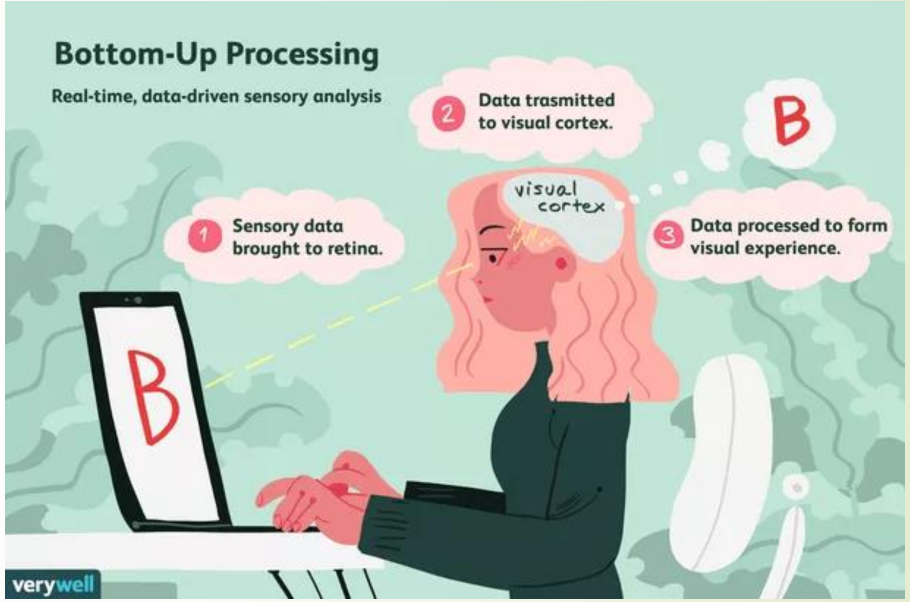


Image Source: Emily Roberts https://www.verywellmind.com/bottom-up-processing-and-perception-4584296

Preceding Works

- The traditional convolution networks does not reflect this bidirectional nature that so characterizes the human visual system.

- Hard to produce diverse images in GAN

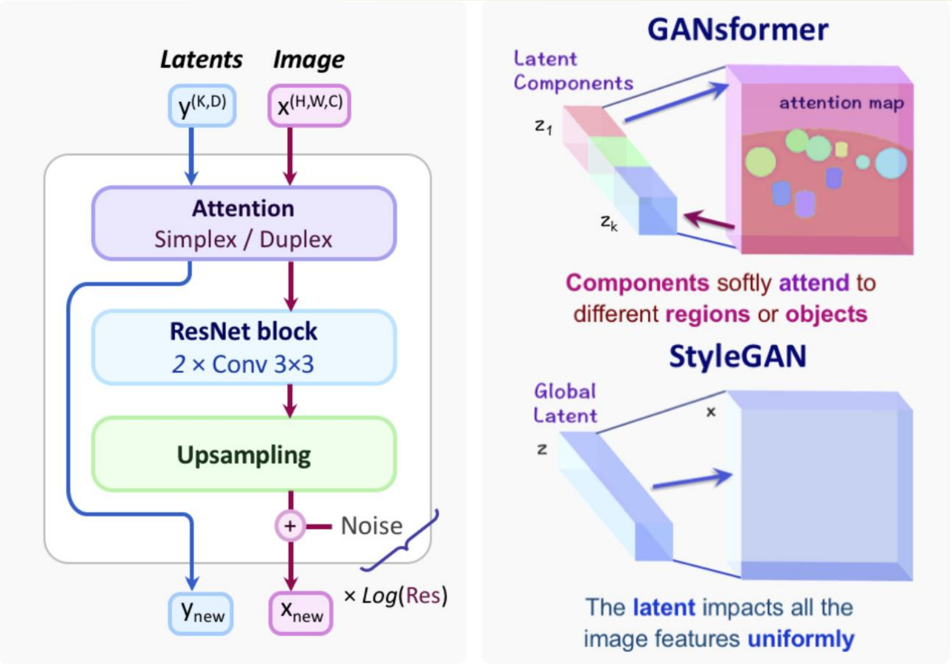
- The Transformer complexity for attention is $O(n^2)$.

- The Style GAN provides no means to control the style of a localized regions within the generated image.

Contents

Input Terms

- z: the latent vector
  - Splits into k components $[z_1, \ldots, z_k]$
- $Y^{m \times d}$: The intermediate latents
  - Akin to output of the mapping network in the StyleGAN architecture
- $X^{n \times d}$: Input Vectors of Dimension d
  - n is $Width \times Height$
  - d is the number of channels

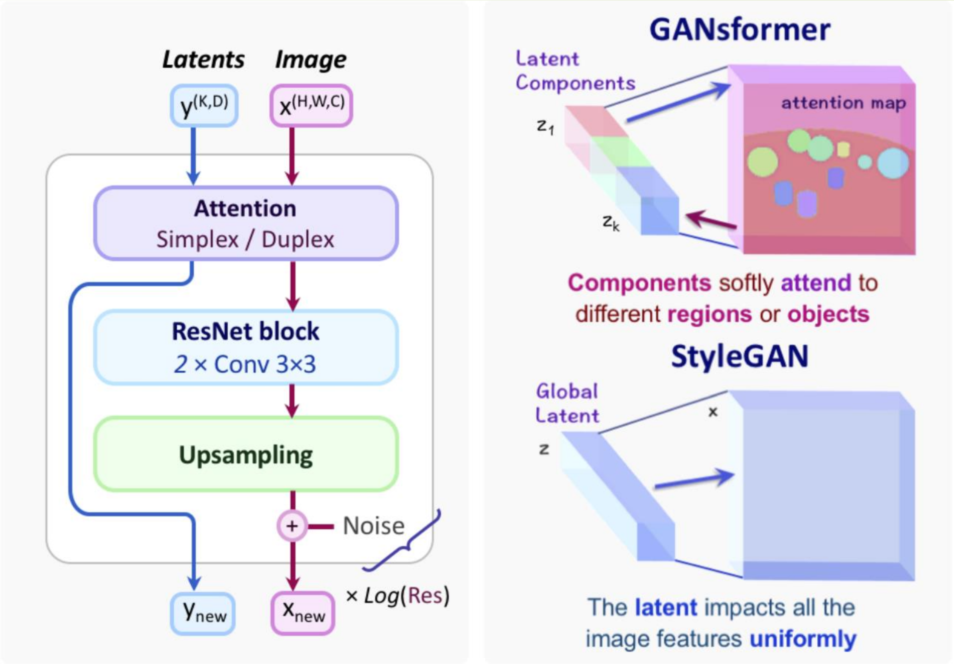


GANsformer Generator Diagram

Image Source: Generative Adversarial Transformers

Input Terms

- z: the latent vector
  - Splits into k components $[z_1, \ldots, z_k]$
- $Y^{m \times d}$: The intermediate latents
  - Akin to output of the mapping network in the StyleGAN architecture
- $X^{n \times d}$: Input Vectors of Dimension d
  - n is $Width \times Height$
  - d is the number of channels
- Notice: inputs in diagram don't match the dimensions of the terms just mentioned
  - Outputs of $q(\cdot)$, $k(\cdot)$, and $v(\cdot)$ transform inputs to their respective dimensions (GANformer only)

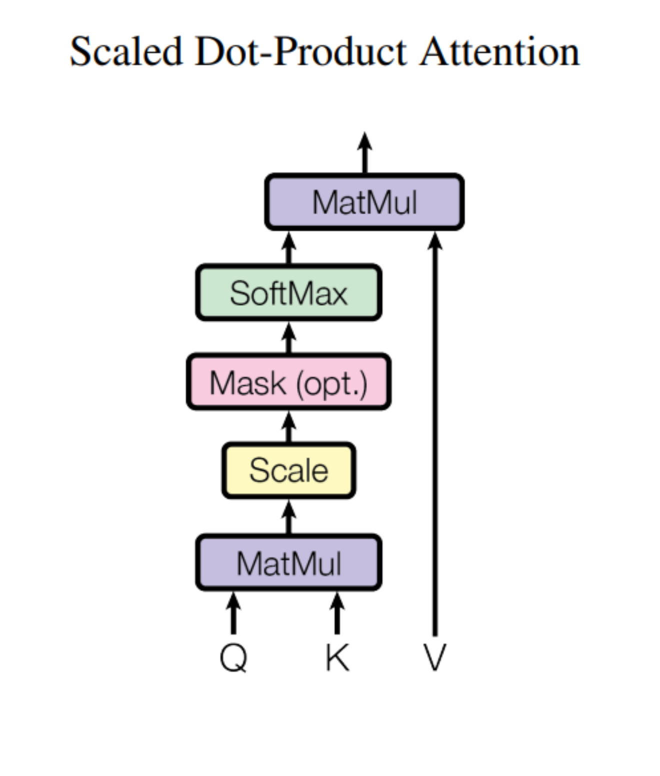


GANsformer Generator Diagram

Image Source: Generative Adversarial Transformers

Vanilla Attention

- $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$
- $a(X) = Attention\big(q(X), k(X), v(X)\big)$
- $q(\cdot)$, $k(\cdot)$, $v(\cdot)$ map elements to queries, keys, and values
  - Maintain dimensionality
- Poorly scales at higher dimensions
  - $O(n^2 \cdot d)$

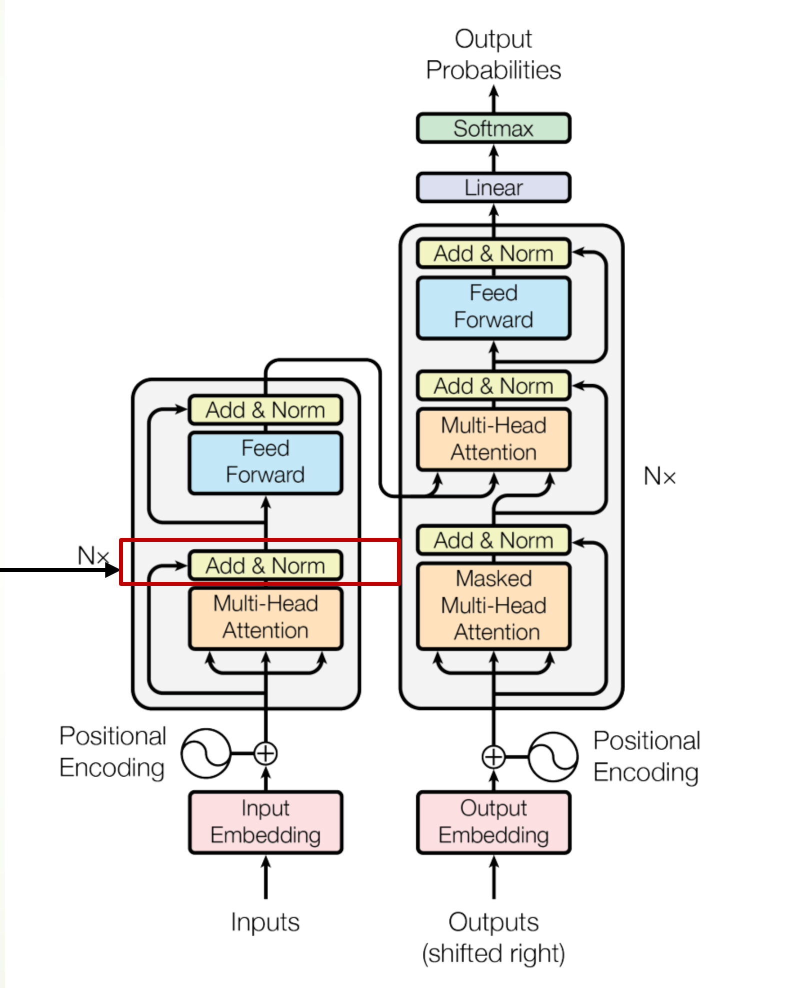


Scaled Dot-Product Attention

Image Source: Attention Is All You Need

Vanilla Attention

- $Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$
- $a(X) = Attention\big(q(X), k(X), v(X)\big)$
- $q(\cdot)$, $k(\cdot)$, $v(\cdot)$ map elements to queries, keys, and values
  - Dimensions are remapped
- Add and LayerNorm right after attention
  - $u^{at}(X) = LayerNorm\big(X + a(X)\big)$



Transformer Architecture

Image Source: Attention Is All You Need

Contents

Bipartite Attention

- $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$
- $a(X, Y) = Attention\left(q(X), \boxed{k(Y), v(Y)}\right)$
  - Keys and values are derived from Y (the latents)
- $u^a(X, Y) = LayerNorm\left(X + a(X, Y)\right)$

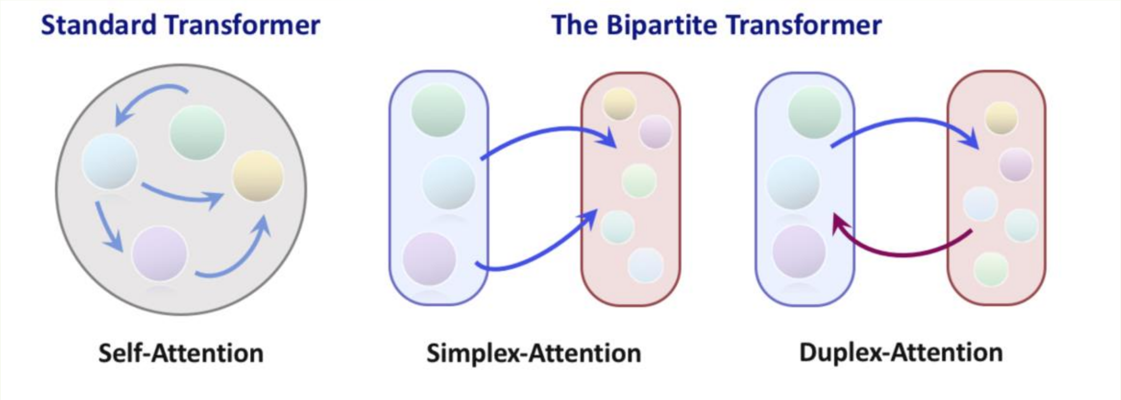


Image Source: Generative Adversarial Transformers

Bipartite Attention Efficiency

- Bilinearly Efficient: $\boxed{O(mn)}$
- Recall: $Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$
  - Self attention: $O(n^2 \cdot d)$
  - Matrix multiplications dominate the calculation
- Matrix Multiplication for QK^T $s.t. Q^{n\times d}, K^{m\times d}: O(ndm) \rightarrow \boxed{O(mn)}$

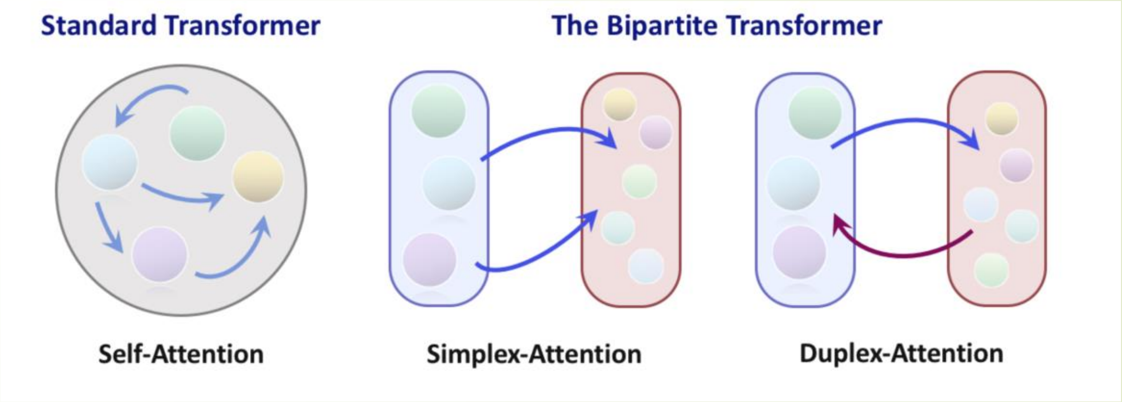


Image Source: Generative Adversarial Transformers

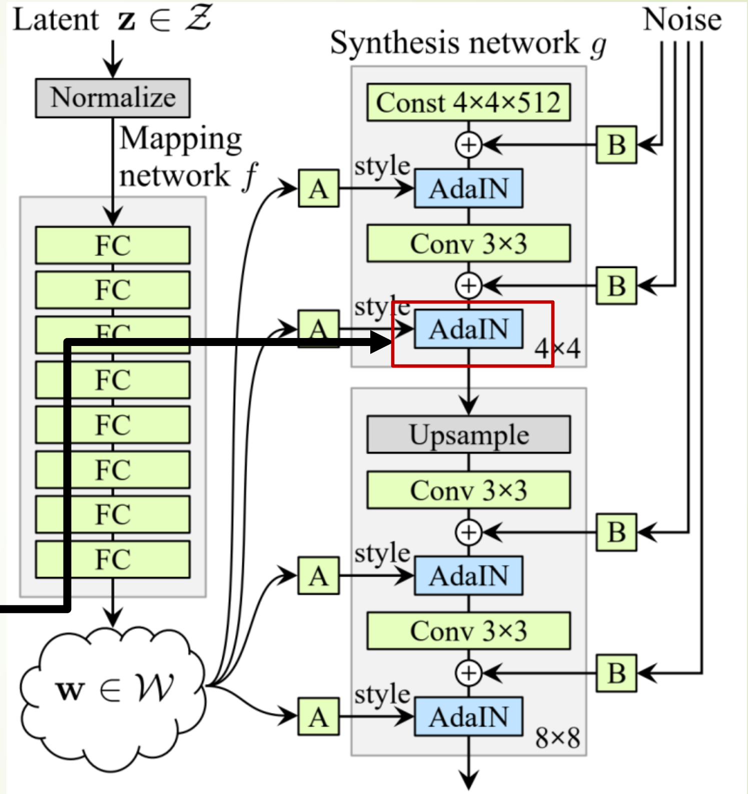
Contents

- Contribution
- Inspirations
- **Attention**
  - Vanilla Attention
  - Bipartite Attention
  - <u>Simplex Attention</u>
  - Duplex Attention
- Testing Results
  - FID Scores
  - Generated Images
- Code Overview
  - Environment Setup
  - Quick-start commands
  - Issues

Simplex Attention

- $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$
- $a(X, Y) = Attention\big(q(X), k(Y), v(Y)\big)$
  - Keys and values are derived from Y (the latents)
- Modified AdaIN After

  Element-wise multiplication
  - $u^s(X, Y) = \gamma\big(a(X, Y)\big) \odot \omega(X) + \beta\big(a(X, Y)\big)$
  - $\omega(X) = \frac{X - \mu(X)}{\sigma(X)}$
  - Operation taken from StyleGAN architecture

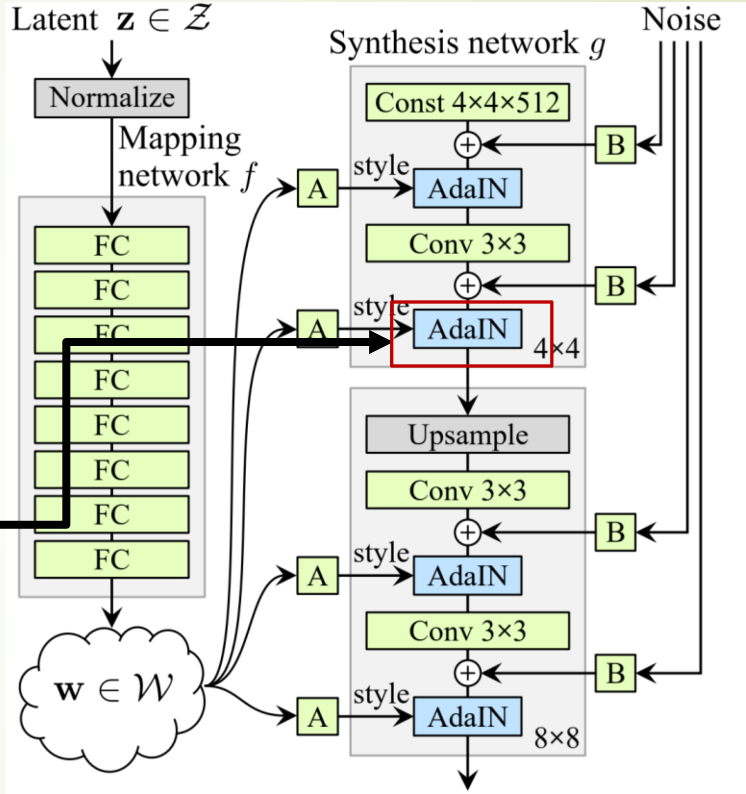


StyleGAN Diagram

Image Source: A Style-Based Generator Architecture for Generative Adversarial Networks

Simplex Attention

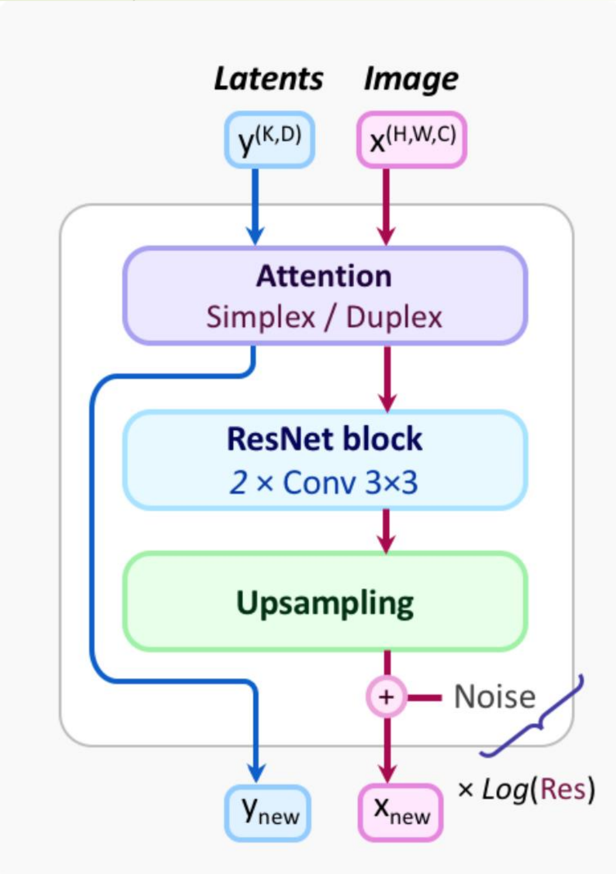
- $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$
- $a(X, Y) = Attention\big(q(X), k(Y), v(Y)\big)$
  - Keys and values are derived from Y (the latents)
- Modified AdaIN After
  - $u^s(X, Y) = \gamma\big(a(X, Y)\big) \odot \omega(X) + \beta\big(a(X, Y)\big)$ Element-wise multiplication
  - $\omega(X) = \frac{X - \mu(X)}{\sigma(X)}$
  - Corresponds to StyleGAN architecture
- <u>Latents are global in the generator network</u>

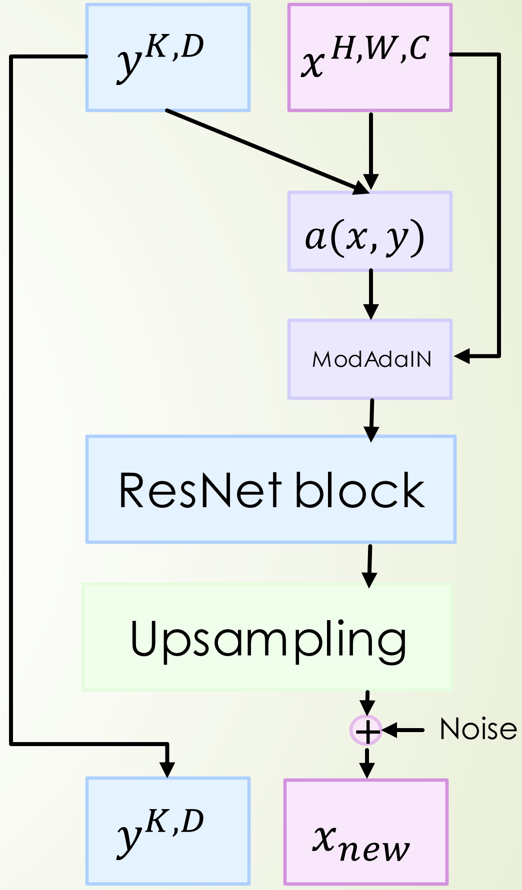


StyleGAN Diagram

Image Source: A Style-Based Generator Architecture for Generative Adversarial Networks

Simplex Attention





Simplex

Image Source: Generative Adversarial Transformers

Contents

Duplex Attention

- Recall: $u^a(X,Y) = LayerNorm\big(X + a(X,Y)\big)$
- $u^d(X,Y) = \gamma\big(A(X,K,V)\big) \odot \omega(X) + \beta\big(A(X,K,V)\big)$
  - $\boxed{K = a(Y,X)}$
  - Analogous to K-Means
  - Contrast from $u^s(X,Y) = \gamma\big(a(X,Y)\big) \odot \omega(X) + \beta\big(a(X,Y)\big)$

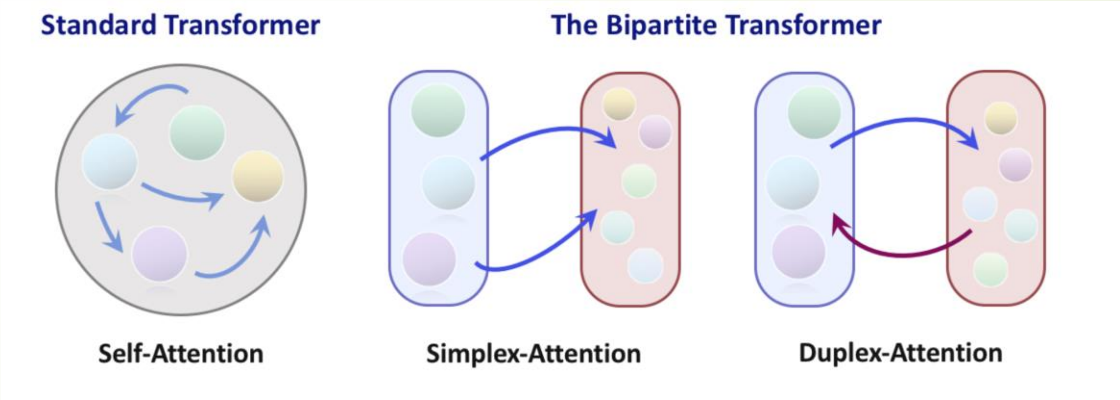


Image Source: Generative Adversarial Transformers

Duplex Attention

- Recall: $u^a(X,Y) = LayerNorm\big(X + a(X,Y)\big)$
- $u^d(X,Y) = \gamma\big(A(X,K,V)\big) \odot \omega(X) + \beta\big(A(X,K,V)\big)$
  - $K = a(Y,X)$
  - Analogous to K-Means
  - Contrast from $u^s(X,Y) = \gamma\big(a(X,Y)\big) \odot \omega(X) + \beta\big(a(X,Y)\big)$
- Compute Duplex Attention
  - $Y := u^a(Y,X)$
  - $X := u^d(X,Y)$

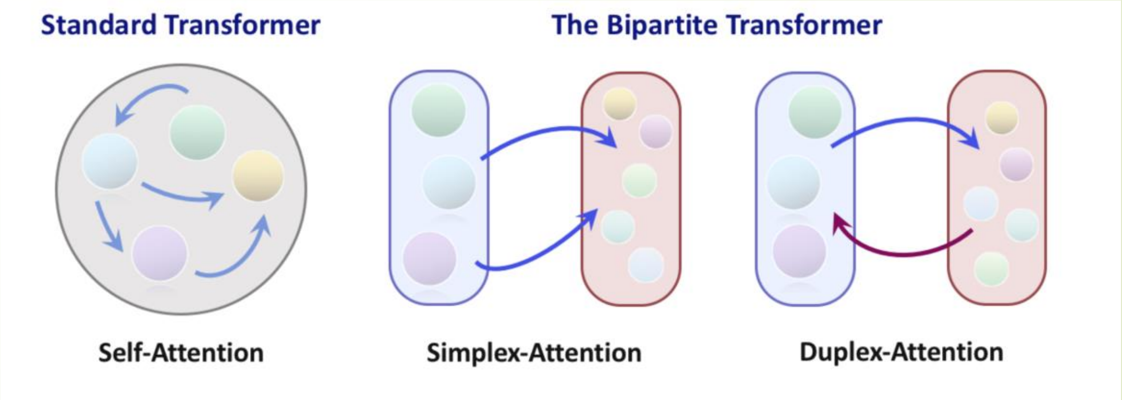
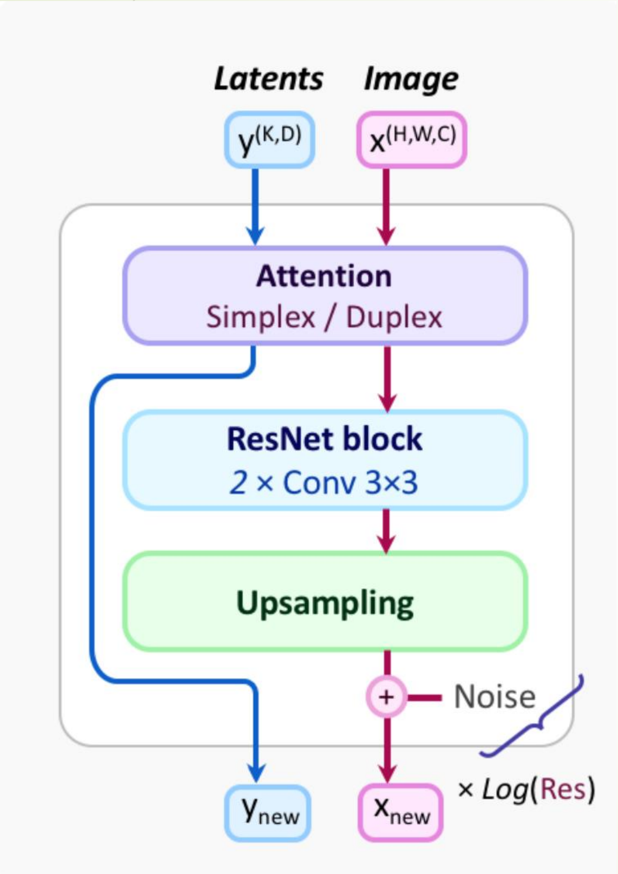
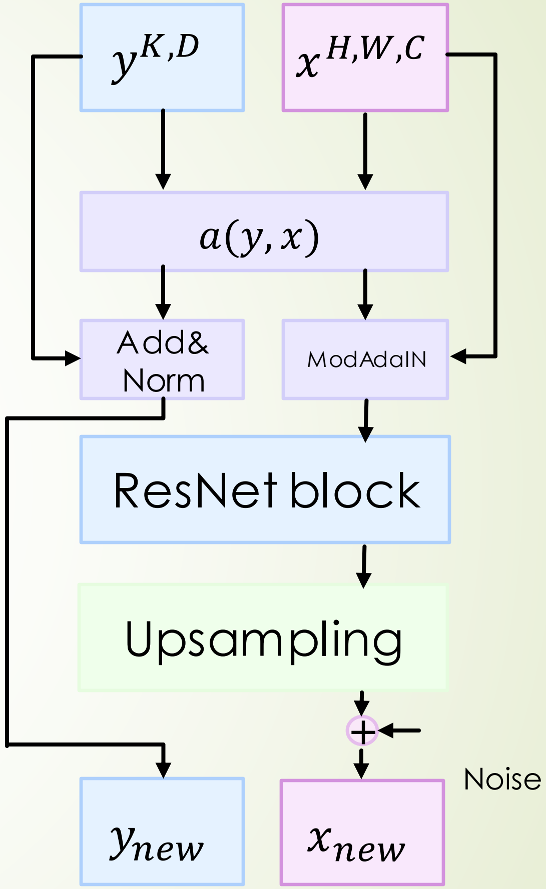


Image Source: Generative Adversarial Transformers

Duplex Attention

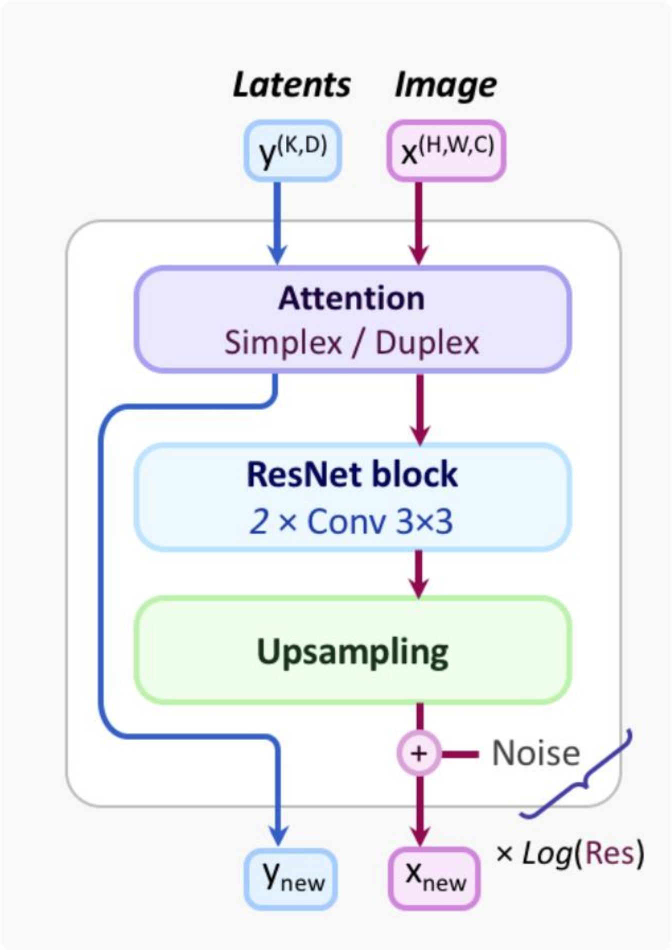


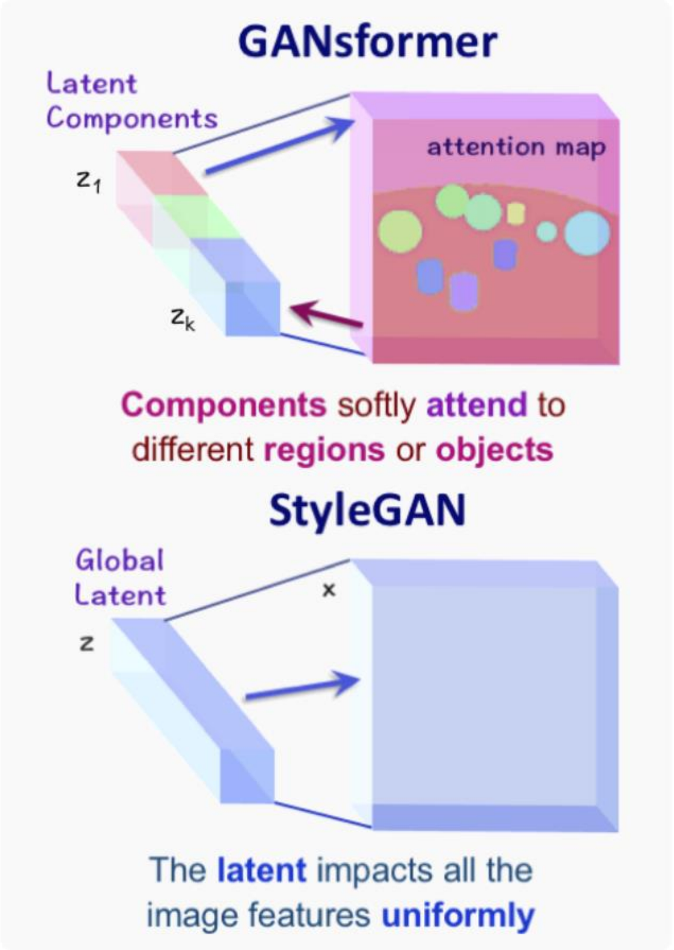


Duplex

Image Source: Generative Adversarial Transformers

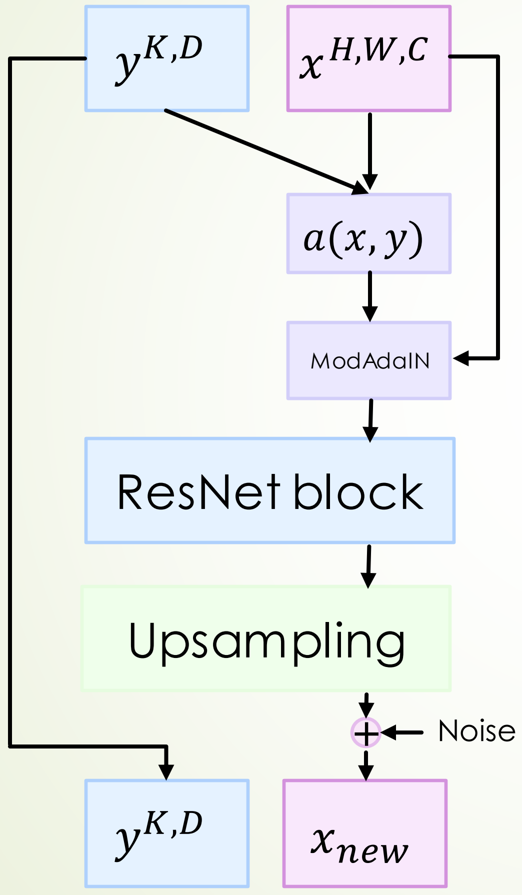
GANsformer Generator Block



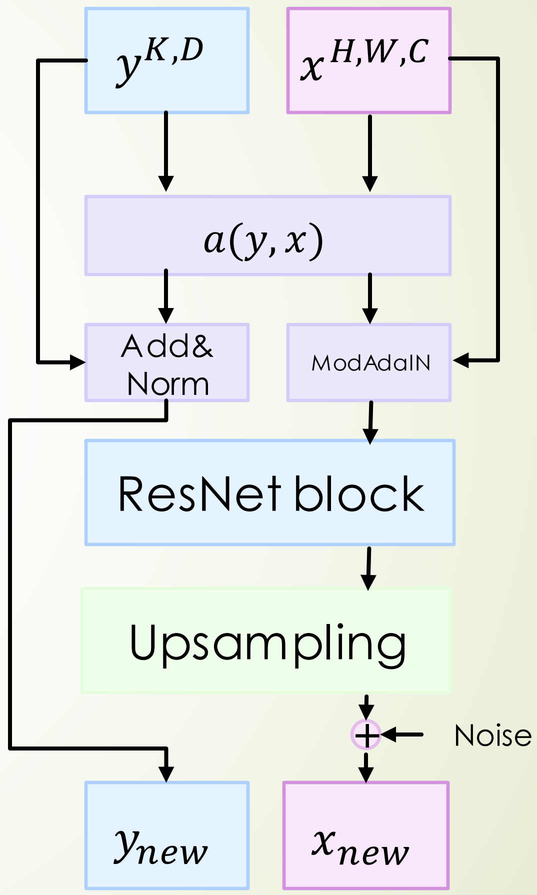


Source: Generative Adversarial Transformers

Recap



Simplex



Duplex

Generator and Discriminator

- Generator
  - Utilizes attention to allow components to style different regions of the image
- Discriminator
  - Attention applied after every convolution
  - Uses trained embeddings for Y

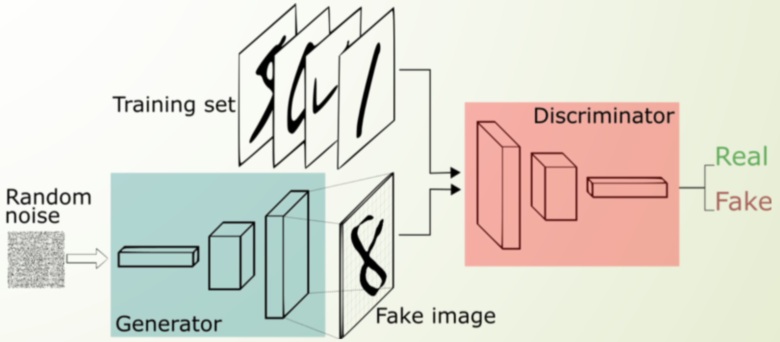


Image Source: Thalles Silva https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

Contents

Results

Model	CLEVR FID ↓	IS ↑	Precision ↑	Recall ↑	LSUN-Bedroom FID ↓	IS ↑	Precision ↑	Recall ↑
GAN	25.02	2.17	21.77	16.76	12.16	2.66	52.17	13.63
k-GAN	28.29	2.21	22.93	18.43	69.90	2.41	28.71	3.45
SAGAN	26.04	2.17	30.09	15.16	14.06	2.70	54.82	7.26
StyleGAN2	16.05	2.15	28.41	23.22	11.53	**2.79**	51.69	19.42
VQGAN	32.60	2.03	46.55	63.33	59.63	1.93	55.24	28.00
GANformer$_s$	10.26	**2.46**	38.47	37.76	8.56	2.69	55.52	22.89
GANformer$_d$	**9.17**	2.36	**47.55**	**66.63**	**6.51**	2.67	**57.41**	**29.71**

Model	FFHQ FID ↓	IS ↑	Precision ↑	Recall ↑	Cityscapes FID ↓	IS ↑	Precision ↑	Recall ↑
GAN	13.18	4.30	67.15	17.64	11.57	1.63	61.09	15.30
k-GAN	61.14	4.00	50.51	0.49	51.08	1.66	18.80	1.73
SAGAN	16.21	4.26	64.84	12.26	12.81	1.68	43.48	7.97
StyleGAN2	9.24	4.33	68.61	25.45	8.35	1.70	59.35	27.82
VQGAN	63.12	2.23	67.01	**29.67**	173.80	**2.82**	30.74	**43.00**
GANformer$_s$	8.12	**4.46**	**68.94**	10.14	14.23	1.67	**64.12**	2.03
GANformer$_d$	**7.42**	4.41	68.77	5.76	**5.76**	1.69	48.06	33.65

Image Source: Generative Adversarial Transformers

Results

- Achieves better FID scores in fewer training steps than previous models

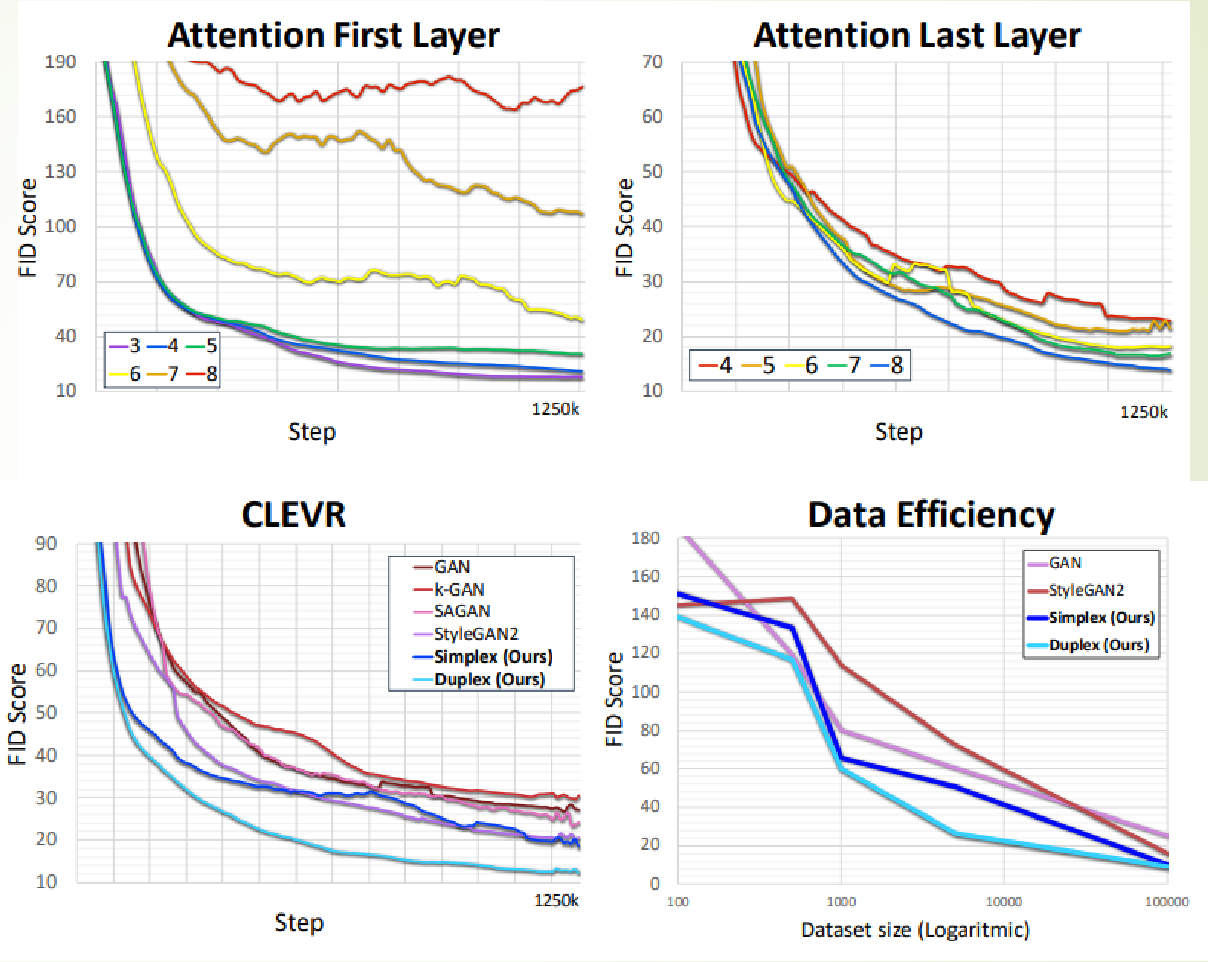


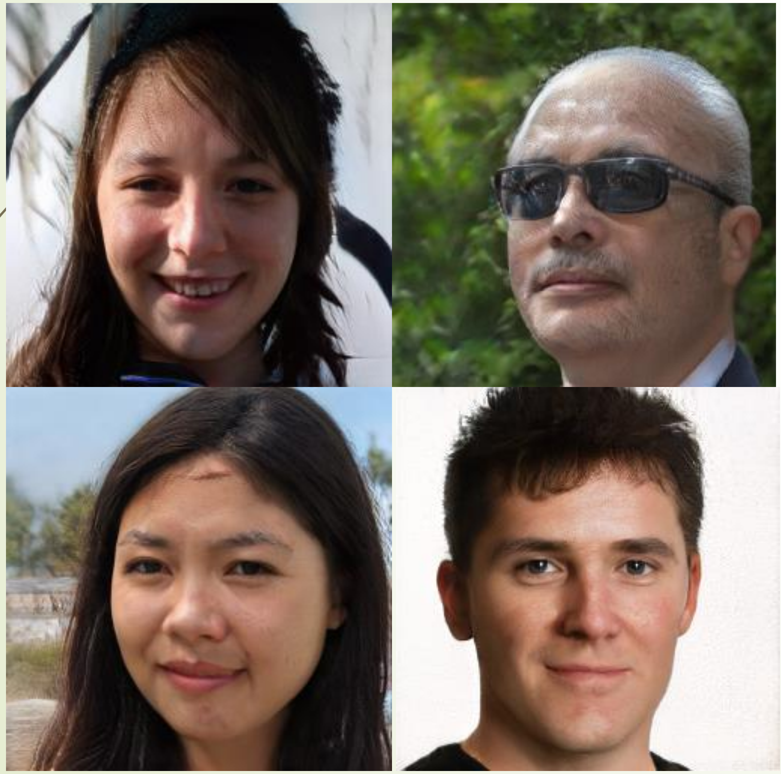
Image Source: Generative Adversarial Transformers

Contents

- Contribution
- Inspirations
- Attention
  - Vanilla Attention
  - Bipartite Attention
  - Simplex Attention
  - Duplex Attention
- **Testing Results**
  - FID Scores
  - Generated Images
- Code Overview
  - Environment Setup
  - Quick-start commands
  - Issues

Our Evaluations

- ffhq-snapshot FID: 6.1067



- cityscapes-snapshot FID: 7.8310

Baseline Model for Cityscapes-2048

FFHQ-1024 Baseline: Good

FFHQ-1024 Baseline: Bad

Attention Maps

- Each color corresponds to a different component of the k latents that influence that region

Contents

Running the Code On Windows Environment Setup

- Clone from https://github.com/dorarad/gansformer
- Create a Python 3.7 virtual environment and install packages from the requirements.txt file
- Install Visual Studio 2017 and the MSVC compiler
  - Edit ./dnnlib/tflib/custom_ops.py and add your compiler path to the **compiler_bindir_search_path** variable

```
compiler_bindir_search_path = [
    "C:/Program Files (x86)/Microsoft Visual Studio/2017/Community/VC/Tools/MSVC/14.14.26428/bin/Hostx64/x64",
    #"C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.23.28105/bin/Hostx64/x64",
    "C:/Program Files (x86)/Microsoft Visual Studio/2017/Community/VC/Tools/MSVC/14.16.27023/bin/Hostx64/x64",
    "C:/Program Files (x86)/Microsoft Visual Studio 14.0/vc/bin",
]
```

Running the Code On Windows Environment Setup

- Clone from https://github.com/dorarad/gansformer
- Create a Python 3.7 virtual environment and install packages from the requirements.txt file
- Install Visual Studio 2017 and the MSVC compiler
  - Edit ./dnnlib/tflib/custom_ops.py and add your compiler path to the **compiler_bindir_search_path** variable
- Download and install CUDA 10.0
- Download cuDNN v7.6.5 (November 5th, 2019), for CUDA 10.0
  - Follow the instructions in section 3.3 in the following link
  - https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html#installwindows

Contents

Running the Code On Windows

- python generate.py --gpus 0 --model gdrive:<dataset>-snapshot.pkl --output-dir images --images-num 32
  - Replace “<dataset>” with different supported dataset name
- python run_network.py --train --gpus 0 --ganformer-default --expname <dataset>-pretrained --dataset <dataset> --pretrained-pkl gdrive:<dataset>-snapshot.pkl
  - Trains a pre-existing network
- Python run_network.py --help
  - Help menu for more options

Contents

- Contribution
- Inspirations
- Attention
  - Vanilla Attention
  - Bipartite Attention
  - Simplex Attention
  - Duplex Attention
- Testing Results
  - FID Scores
  - Generated Images
- **Code Overview**
  - Environment Setup
  - Quick-start commands
  - Issues

Issues We Ran Into

- Could not train a model using duplex attention
  - Enabling causes errors
  - Could be an issue with the dependencies
- Often ran out of GPU memory
  - Code designed with 12GB GPUs in mind
  - Led to crashing
  - Couldn't generate attention maps

```
Produce visualizations...
  0%|                                                                                                                          | 0/1 [00:00<?, ?it/s]Running network...
2022-03-27 11:36:14.279813: E tensorflow/stream_executor/cuda/cuda_driver.cc:828] failed to allocate 4.00G (4291821568 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2022-03-27 11:36:14.455244: E tensorflow/stream_executor/cuda/cuda_driver.cc:828] failed to allocate 3.60G (3862639360 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2022-03-27 11:36:14.623482: E tensorflow/stream_executor/cuda/cuda_driver.cc:828] failed to allocate 3.24G (3476375296 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2022-03-27 11:36:14.788041: E tensorflow/stream_executor/cuda/cuda_driver.cc:828] failed to allocate 2.91G (3128737792 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
100%|##################################################################################################################| 1/1 [00:15<00:00, 15.40s/it]
```

Questions?