ATML Report

Giorgia Adorni
giorgia.adorni@usi.ch

Felix Boelter
felix.boelter@usi.ch

Stefano Carlo Lambertenghi
stefano.carlo.lambertenghi@usi.ch

Abstract

Concise and self-contained description of your project, motivation and main findings.

1 Introduction

This project consists in a reproducibility challenge whose objective is to investigate the reliability and reproducibility of a paper accepted for publication in a top machine learning conference. The models have been implemented using code and information provided by the authors. Actually we are not participating formally to the Fall Edition of the ML Reproducibility Challenge 2021.

We selected the paper "*Generative adversarial transformers*" by Hudson and Zitnick [1] and verified the empirical results and claims in it by reproducing three of the computational experiments performed by the authors: (1) the **StyleGAN2** by Karras et al. [2], a GAN network which uses one global latent style vector to modulate the features of each layer, hence to control the style of all image features globally, (2) the **GANformer** with **Simplex Attention** by Hudson and Zitnick [1], a generalisation of the StyleGAN design with k latent vectors that cooperate through attention, allowing for a spatially finer control over the generation process since multiple style vectors impact different regions in the image concurrently, in particular permitting communication in one direction, in the generative context – from the latents to the image features, and (3) the **GANformer** with **Duplex Attention** by Hudson and Zitnick [1], which is based on the same principles as the previous but propagating information both from global latents to local image features, enabling both top-down and bottom-up reasoning to occur simultaneously.

The first model is used as baseline, while the remaining are the architectures introduced by the authors. Thy consider the GANformer has "a novel and efficient type of transformer" which demonstrates its strength and robustness over a range of task of visual generative modelling — simulated multi-object environments (real-world indoor and out-door scenes) — achieving state-of-the-art results in terms of both image quality and diversity, while benefiting from fast learning and better data-efficiency.

2 Related works

2.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [3], are discriminative models that learn to determine whether a sample is from the model distribution or the data distribution.

The architecture is composed by two main neural networks: a **generator $G(z)$**, which captures the data distribution mapping random noise z to the data space, and a **discriminator $D(x)$** which estimates the probability that sample x is generated by G or is a real sample, so it is coming from the training data.

The training procedure has two objective: the discriminator D is trained to maximise the probability of assigning the correct label to both training examples and samples from G, while the generator G is trained to maximise the probability of D making a mistake, so it aims to minimise $\log(1 - D(G(z)))$.

Combining the two objectives for $G(z)$ and $D(x)$ we get the *GAN min-max game* with the value function $V(G, D)$:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p*(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))))] \tag{1}$$

Preprint. Under review.

2.2 StyleGAN2

StyleGAN are a re-design of the GANs generator architecture, aimed at controlling the image synthesis process [4].

As mentioned by Karras et al. [2], «this generator starts from a learned constant input and adjusts the "style" of the image at each convolution layer based on the latent code, therefore directly controlling the strength of image features at different scales. Combined with noise injected directly into the network, this architectural change leads to automatic, unsupervised separation of high-level attributes (e.g., pose, identity) from stochastic variation (e.g., freckles, hair) in the generated images, and enables intuitive scale-specific mixing and interpolation operations».

The generator embeds the input latent code into an intermediate latent space. In particular, the input latent space must follow the probability density of the training data, while the intermediate latent space is free from that restriction and is therefore allowed to be disentangled.

Traditionally, the latent code is provided to the generator through an input layer of a feed-forward network. Karras et al. [2] depart from this design by omitting the input layer altogether and starting from a learned constant instead. Given a latent code z in the input latent space Z, a non-linear mapping network $f : Z \to W$ first produces $w \in W$. Learned affine transformations then specialize w to styles $y = (y_s, y_b)$ that control adaptive instance normalization (AdaIN) [**?**] operations after each convolution layer of the synthesis network g. The AdaIN operation is defined as

$$\mathsf{AdaIN}(x_i, y) = y_{s,i}\frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i} \tag{2}$$

where each feature map x_i is normalized separately, and then scaled and biased using the corresponding scalar components from style y. Thus the dimensionality of y is twice the number of feature maps on that layer.

Moreover, they provide the generator with a direct means to generate stochastic detail by introducing explicit noise inputs.

The StyleGAN2 architecture makes it possible to control the image synthesis via scale-specific modifications to the styles.

2.3 Transformers

Transformers are architectures which avoid recurrence and instead rely entirely on an attention mechanism to draw global dependencies between input and output [5].

These models rely entirely on **self-attention** to compute representations of its input and output without using *sequence-aligned RNNs* or *convolution*.

The architecture is composed by an encoder-decoder structure where, the **encoder** maps an input sequence of symbol representations $(x_1, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, \ldots, z_n)$, while the **decoder**, given z, generates an output sequence $(y_1, \ldots, y_m)$ of symbols one element at a time.

At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. In addition to this architecture, the transformer employs **stacked self-attention** and **point-wise**, fully connected layers for both the encoder and decoder.

In particular, each transformer module (encoder and decoder) contains $n = 6$ identical layers, each with two sub-layers: (1) a *multi-head self-attention mechanism* working in parallel, and (2) a *position-wise fully connected feed-forward neural network*, with residual connections followed by normalisation. The decoder self-attention sub-layer has a mask which prevents positions from attending to subsequent positions (so positional encoding provided as additional input to bottom layer). This, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i. In addition, the decoder has a third sub-layer, which performs *multi-head attention* over the output of the encoder stack.

An *attention function* [5], can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$ and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

$$\mathsf{Attention}(Q, K, V) = \mathsf{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

This is a **dot-product attention** with a scaling factor of $\sqrt{1}$.

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values. **Multi-head attention** allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\begin{aligned} \mathsf{MultiHead}(Q, K, V) &= \mathsf{Concat}(\mathsf{head}_1 \ldots, \mathsf{head}_h)W^O \\ \text{where} \quad \mathsf{head}_i &= \mathsf{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. In this work we employ h = 8 parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

The transformer uses multi-head attention in three different ways:

1. In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.
2. The encoder contains self-attention layers, in which all of the keys, values and queries come from the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
3. Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

3 Methodology

3.1 Generative Adversarial Transformers

[1]

a simple yet effective generalization of the vanilla transformer, explored here for the task of visual synthesis.

The model utilizes a bipartite structure for computing soft attention, that iteratively aggre- gates and disseminates information between the generated image features and a compact set of latent variables that function as a bottleneck, to enable bidirectional interaction between these dual representations.

Bipartite Attention. We introduce the GANformer network, that leverages a bipartite structure to allow long-range interactions, while evading the quadratic complexity which standard transformers suffer from. We present two novel attention operations over the bipartite graph: simplex and duplex, the former permits communication in one direction, in the generative context – from the latents to the image features, while the latter enables both top-down and bottom up connections between these two dual representations. between the image features and a small collection of latent variables.

The Generative Adversarial Transformer (GANformer) is a type of Generative Adversarial Network, which involves a generator network (G) that maps a sample from the latent space to the output space (e.g. an image), and a discriminator network (D) which seeks to discern between real and fake samples [28]. The two networks compete with each other through a minimax game until reaching an equilibrium. Typically, each of these networks consists of multiple layers of convolution, but in the GANformer case, we instead construct them using a novel architecture, called Bipartite Transformer, formally defined below.

The standard transformer network is composed of alternating multi-head self-attention and feed-forward layers. We refer to each pair of self-attention and feed-forward layers as a transformer layer, such that a transformer is considered to be a stack of several such layers. The Self-Attention layer considers all pairwise relations among the input elements, so to update each single element by attending to all the others. The Bipartite Transformer generalizes this formulation, featuring instead a bipartite graph between two groups of variables – in the GAN case, latents and image features. In the following, we consider two forms of attention that could be computed over the bipartite graph – Simplex attention, and Duplex attention, depending on the direction in which information propagates3 – either in one way only, from the latents to the image, or both in top-down and bottom-up ways. While for clarity purposes, we present the technique here in its one-head version, in practice we make use of a multi-head variant, in accordance with [64].

3.1.1 Simplex attention

We begin by introducing the simplex attention, which distributes information in a single direction over the bipartite transformer graph. Formally, let $X^{n \times d}$ denote an input set of n vectors of dimension d (where, for the image case, $n = W \times H$), and $Y^{m \times d}$ denote a set of m aggregator variables (the latents, in the generative case). We can then compute attention over the derived bipartite graph between these two groups of elements. Specifically, we define the attention as in Equation (3), moreover:

$$a(X, Y) = \mathsf{Attention}(q(X), k(Y), v(Y)) \tag{5}$$

where $q(\cdot), k(\cdot), v(\cdot)$ are functions that respectively map elements into queries, keys, and values, all maintaining dimensionality d. We also provide the mappings with positional encodings, to reflect the distinct position of each element (e.g. in the image). Note that this bipratite attention is a generalization of self-attention, where $Y = X$.

We can then integrate the attended information with the input elements X, but whereas the standard transformer implements an additive update rule of the form:

$$u^a(X, Y) = \mathsf{LayerNorm}(X + a(X, Y)) \tag{6}$$

we instead use the retrieved information to control both the scale as well as the bias of the elements in X, in line with the practice promoted by the StyleGAN model [4]. As our experiments indicate, such multiplicative integration enables significant gains in the model performance. Formally:

$$u^s(X, Y) = \gamma(a(X, Y)) \odot \omega(X) + \beta(a(X, Y)) \tag{7}$$

Where $\gamma(\cdot), \beta(\cdot)$ are mappings that compute multiplicative and additive styles (gain and bias), maintaining a dimension of d, and $\omega(X) = X - \mu(X)$ normalizes each element with $\sigma(X)$ respect to the other features. By normalizing X (image features), and then letting Y (latents) control the statistical tendencies of X, we essentially enable information propagation from Y to X, intuitively, allowing the latents to control the visual generation of spatial attended regions within the image, so as to guide the synthesis of objects or entities.

3.1.2 Duplex attention

We can go further and consider the variables Y to poses a key-value structure of their own: $Y = (K^{n \times d}, V^{n \times d})$, where the values store the content of the Y variables, as before (e.g. the randomly sampled latent vectors in the case of GANs) while the keys track the centroids K of the attention-based assignments between Y and X, which can be computed as $K = a(Y, X)$ — namely, the weighted averages of the X elements using the bipartite attention distribution derived through comparing it to Y. Consequently, we can define a new update rule:

$$u^d(X, Y) = \gamma(A(X, K, V)) \odot \omega(X) + \beta(A(X, K, V)) \tag{8}$$

This update compounds two attention operations on top of each other: where we first (1) compute soft attention assignments between X and Y, by $K = a(Y, X)$, and then (2) refine the assignments by considering their centroids, by $A(X, K, V)$. This is analogous to the k-means algorithm and works more effectively than the simpler update u^a defined above in Equation (7).

Finally, to support bidirectional interaction between X and Y (the image and the latents), we can chain two reciprocal simplex attentions from X to Y and from Y to X, obtaining the duplex attention, which alternates computing $Y := u^a(Y, X)$ and $X := u^d(X, Y)$, such that each representation is refined in light of its interaction with the other, integrating together bottom-up and top-down interactions.

3.1.3 Vision-Specific adaptations

A *kernel size* of $k = 3$ is used after each application of the attention, together with a *Leaky ReLU non-linearity* after each convolution and then upsample or downsample the features X, as part of the generator or discriminator respectively, as in e.g. StyleGAN2 [2]. To account for the features location within the image, we use a sinusoidal positional encoding along the horizontal and vertical dimensions for the visual features X, and trained positional embeddings for the set of latent variables Y. Overall, the bipartite transformer is thus composed of a stack that alternates attention (simplex or duplex), convolution, and upsampling layers, starting from a 4×4 grid up to the desirable resolution.

Conceptually, this structure fosters an interesting communication flow: rather than densely modelling interactions among all the pairs of pixels in the images, it supports adaptive long-range interaction between far away pixels in a moderated manner, passing through a compact and global latent bottleneck that selectively gathers information from the entire input and distributes it back to the relevant regions. Intuitively, this form can be viewed as analogous to the top-down / bottom-up notions discussed in section 1, as information is propagated in the two directions, both from the local pixel to the global high-level representation and vice versa.

Both the simplex and the duplex attention operations enjoy a bilinear efficiency of $\mathcal{O}(mn)$ thanks to the network's bipartite structure that considers all pairs of corresponding elements from X and Y. Since, as we see below, we maintain Y to be of a fairly small size, choosing m in the range of 8–32, this compares favourably to the prohibitive $\mathcal{O}(n^2)$ complexity of self-attention, which impedes its applicability to high-resolution images.

To recapitulate the discussion above, the GANformer successfully unifies the GAN and Transformer architectures for the task of scene generation. Compared to traditional GANs and transformers, it introduces multiple key innovations:

- Compositional Latent Space with multiple variables that coordinate through attention to produce the image cooperatively, in a manner that matches the inherent compositionality of natural scenes.
- Bipartite Structure that balances between expressiveness and efficiency, modelling long-range dependencies while maintaining linear computational costs.
- Bidirectional Interaction between the latents and the visual features, which allows the refinement and interpretation of each in light of the other.
- Multiplicative Integration rule to impact the features' visual style more flexibly, akin to StyleGAN but in contrast to the transformer network.

As we see in the following section, the combination of these design choices yields a strong architecture that demonstrates high efficiency, improved latent space disentanglement, and enhanced transparency of its generation process.

4 Implementation

The code has been implemented using code and information provided by the authors.

4.1 Datasets

The original paper [1] explored the GANformer model on four datasets for images and scenes: CLEVR [6], LSUN-Bedrooms [7], Cityscapes [8] and FFHQ [4].

Initially, we tried to use the Cityscapes dataset, since it is the smaller among the four: it contains 24998 images with 256x256 resolution. However, the time to complete the training was to high on this dataset (TODO: how much?), even if using computational resources like GPU.

For this reason, we switch to another dataset, the Google Cartoon Set [9][1], containing 10k 2D cartoon avatar images with 64x64 resolution, composed of 16 components that vary in 10 artwork attributes, 4 colour attributes, and 4 proportion attributes (see in Table 1).

4.2 Hyper-parameters

Note that in the code provided by the author [1], the hyper-parameters are not the same mentioned in the article.

4.3 Experimental setup

The source code of our work is available at the following GitHub repository: `https://github.com/GiorgiaAuroraAdorni/gansformer-reproducibility-challenge`.

[1]`https://google.github.io/cartoonset`

Table 1: Attributes of the Cartoon Set `https://google.github.io/cartoonset/download`.

		# Variants	Description
Artwork	`chin_length`	3	Length of chin (below mouth region)
	`eye_angle`	3	Tilt of the eye inwards or outwards
	`eye_lashes`	2	Whether or not eyelashes are visible
	`eye_lid`	2	Appearance of the eyelids
	`eyebrow_shape`	14	Shape of eyebrows
	`eyebrow_weight`	2	Line weight of eyebrows
	`face_shape`	7	Overall shape of the face
	`facial_hair`	15	Type of facial hair (type 14 is no hair)
	`glasses`	12	Type of glasses (type 11 is no glasses)
	`hair`	111	Type of head hair
Colors	`eye_color`	5	Color of the eye irises
	`face_color`	11	Color of the face skin
	`glasses_color`	7	Color of the glasses, if present
	`hair_color`	10	Color of the hair, facial hair, and eyebrows
Proportions	`eye_eyebrow_distance`	3	Distance between the eye and eyebrows
	`eye_slant`	3	Similar to eye_angle, but rotates the eye and does not change artwork
	`eyebrow_thickness`	4	Vertical scaling of the eyebrows
	`eyebrow_width`	3	Horizontal scaling of the eyebrows

The approaches proposed in both the original paper codebase by Karras et al. [2] and by Hudson and Zitnick [1] have been implemented in Python using TensorFlow [10], so, according to that, we used the same setup. We created a Jupyter Notebook which runs all the experiments in Google Colaboratory, which allows us to write and execute Python in the browser.

All the models have been trained on a Tesla P100-PCIE-16GB (GPU) provided by Google Colab Pro.

4.4 Computational requirements

In the original paper [1], they evaluate all models under comparable conditions of training scheme, model size, and optimization details, implementing all the models within the codebase introduced by the Style-GAN authors [2]. All models have been trained with images of 256 × 256 resolution and for the same number of training steps, roughly spanning a week on 2 NVIDIA V100 GPUs per model (or equivalently 3-4 days using 4 GPUs).

Considered that we had available just one GPU and not enough time to reproduce this settings, we decided to resize the images from 256×256 to 64x64 resolution.

For the GANformer, we select k–the number of latent variables, from the range of 8–32.

All models have been trained for the same number of steps, which ranges between 5k to 15k kimg training samples.

The paper present results after training 100k, 200k, 500k, 1m, 2m, 5m and 10m samples

For the StyleGAN2 model we present results after training 100k (2h30m), 200k (6h), 500k, obtaining good results. Note that the original StyleGAN2 model has been trained by its authors [2] for up to 70k kimg samples, which is expected to take over 90 GPU-days for a single model.

For the GANformer, the authors [2] show impressive results, especially when using duplex attention: the model manages to learn a lot faster than competing approaches, generating astonishing images early in the training. This model is expected to take 4 GPU-days.

However, we are not able to replicate this achievements, first because this model learns significantly slower than the StyleGAN2, which is instead able to produce high-quality images in approximately x-times less training steps than the GANformer.

(in the paper they reach better results with the GANformer with 3-times less training steps than the StyleGAN2, but they don't specify the time required for this training steps...)

5 Results

They evaluate the models along commonly used metrics such as FID, IS, and Precision & Recall scores

Sample images from different points in training are based on the same sampled latent vectors, thereby showing how the image evolves during the training.

For CLEVR and Cityscapes, we present results after training to generate 100k, 200k, 500k, 1m, and 2m samples. For the Bedroom case, we present results after 500k, 1m, 2m, 5m and 10m generated samples during training

These results show how the GANformer, especially when using duplex attention, manages to learn a lot faster than competing approaches, generating impressive images early in the training

6 Discussion and conclusion

References

[1] Drew A. Hudson and C. Lawrence Zitnick. Generative adversarial transformers. 2021.

[2] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[4] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[6] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017.

[7] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[9] Cole Forrester, Mosseri Inbar, Krishnan Dilip, Sarna Aaron, Maschinot Aaron, Freeman Bill, and Fuman Shiraz. Cartoon set. `https://google.github.io/cartoonset/`.

[10] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.