Generating Winged Horses with a Generative Adversarial Network

Anonymous author

Abstract

This paper proposes and explains the use of a Deep Convolutional Wasserstein Generative Adversarial Network (WGAN) to produce images that look like the mythical Pegasus (a white horse with wings). Without input pictures of winged horses, using only the CIFAR-10 and STL-10 datasets, the WGAN is conditioned on 64×64 pixel images of birds and horses to learn their latent features. Next, a demonstration of randomly generated results from the WGAN are demonstrated, with the best Pegasus-like image being selected. Finally, the limitations are considered and potential future improvements are suggested.

1 Methodology

At the simplest level of abstraction, a Generative Adversarial Network (GAN) [3] places a Generator G and a Discriminator D against one another in a two-player minimax game. G looks to generate images of probability distribution $p(g)$ to match the training images of probability distribution $p(r)$ as closely as possible. This is done by attempting to transform $p(g)$ towards $p(r)$ by minimising a given distance metric.

The Deep Convolutional Generative Adversarial Network (DCGAN) [7] improves on the GAN by replacing fully-connected layers with convolution layers. The idea is that these convolution kernels can learn spacial features matching the distribution of the input dataset. As with the original DCGAN implementation, the generator and discriminator consist of equivalent but opposite operations (with 2D convolutions in the discriminator and 2D transposed convolutions in the generator). For the generator, this consists of taking an $n_l \times 1 \times 1$ latent representation and transpose-convolving (thus unscaling) through five layers to a $3 \times 64 \times 64$ image representation. The discriminator performs the opposite, taking the input image from a $3 \times 64 \times 64$ image representation to a scalar discrimination value (representing whether the discriminator thinks the image is from the training set or not).

Some common distance metrics include Kullback-Leibler (KL) divergence, and its extension Jensen-Shannon (JS) divergence. While these can perform well under well-tuned hyperparameters and a more simple dataset, they both suffer from gradient instability.

At its core, this method consists of a modified version of a Wasserstein GAN [1] (WGAN). WGANs look to alleviate the training instability inherent within KL and JS using the Wasserstein distance metric. This is given by

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x} \sim p_{\text{gen}}}[f(\boldsymbol{x})] \tag{1}$$

Where p_{data} represents real images and p_{gen} represents images generated from random noise.

The idea here is that the discriminator tries to maximise the difference between $\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[f(\boldsymbol{x})]$ and $\mathbb{E}_{\boldsymbol{x} \sim p_{\text{gen}}}[f(\boldsymbol{x})]$ whereas the generator wants to minimise it.

In terms of implementation, at the end of the forward pass for the discriminator, the 0–1 normalising sigmoid function σx is removed. Because it no longer outputs the likelihood of the image being fake, the discriminator is renamed to the 'Critic'. As an added advantage, the critic's loss value now correlates with the convergence of the generator. Additionally, the critic is trained more than the generator (five times per generator training step). Figure 1 shows a diagram demonstrating the inputs, outputs and architecture of the WGAN.

Table 1: Hyperparameters for WGAN with GP

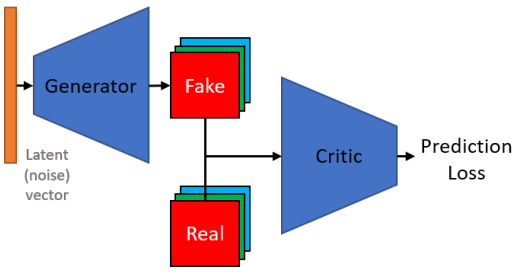
HYPERPARAMETER	VALUE
Adam learning rate	0.0001
Adam β_1	0.0
Adam β_2	0.9
Mini-batch size	64
Input image channels	3
Input image size	64×64
Latent features	100
Critic base features	16
Generator base features	16
Critic iterations	5
Gradient penalty λ	10
Epochs	500

While WGANs improve stability (and attempt to prevent mode collapse), they do this at the cost of having potentially longer training times. This is partly down the need to satisfy the Lipschitz constraint (the $\max_{\|f\|_L \leq 1}$ term in Equation 1), which the original paper proposes be done by clipping all the weights to lie within a constant range around zero. It also recognises that depending on the clipping parameter, a small value may lead to vanishing gradients whereas a large value may dramatically increase training times (as weights may take a while to reach their limit). [4] looks to remedy this by proposing a WGAN with Gradient Penalty (GP) - a different way to satisfy the Lipschitz constraint. This works by adding a GP term to the original WGAN loss as follows.

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\text{gen}}}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x} \sim p_{\text{real}}}[f(\boldsymbol{x})] + \lambda \mathbb{E}_{\hat{\boldsymbol{x}} \sim p_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2] \tag{2}$$

Where λ is the penalty coefficient which determines how much weight is given to the GP, and $p_{\hat{\boldsymbol{x}}}$ is the distribution over the interpolated samples $\epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$ between real image $\boldsymbol{x}$ and randomly generated image $\tilde{\boldsymbol{x}}$ for randomly sampled $\epsilon \in u[0, 1]$. Notice the first two terms have switched, this now means that its negation is used as the loss.

Figure 1: WGAN architecture



The hyperparameters are shown in Table 1 and are generally kept the same as used in their respective original papers and implementations.

To achieve Pegasus-like outputs, the WGAN is trained on the CIFAR-10 [5] and STL-10 [2] datasets. As part of pre-processing, the images are resized to 64x64 pixels and normalised (as required for WGAN convergence). Finally, all images except those labeled as horses and birds are discarded. Since WGANs prevent mode collapse, when trained randomly on both horses and birds, the resultant generated images should consist of a combination of horse and bird features. This works on the hope that a subset of the convolutions will produce horse-like features and another subset will produce bird-like features (wings being the most desirable). Because of this, it is expected that most outputs will not look like Pegasus, with potentially only a few per 64-batch matching the criteria.

2 Results

Figure 2 shows the best batch of 64 images: While most of the samples are very noisy, a horse-like shape can be seen in some samples, and a bird-like shape in others.

Of this batch however, the most Pegasus-like image is shown in Figure 3. It has a light colour and, with some imagination, looks like it is flying (with its head in the top left, wings at the top and legs at the bottom).

Figure 2: WGAN architecture

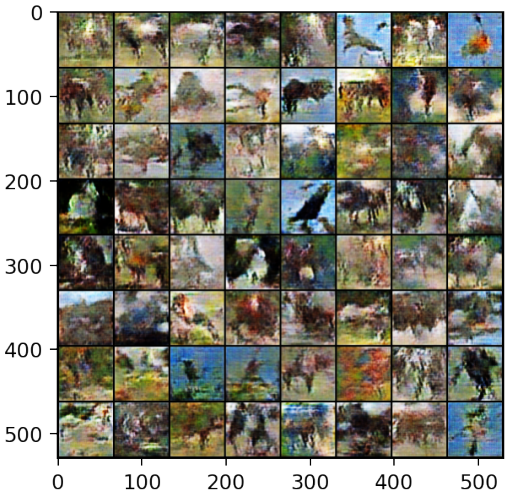


Figure 3: WGAN architecture

3 Limitations

The majority of the generated images do not look like a Pegasus. Since the loss converged to around 20 after about 500 epochs, it is unlikely any additional epochs would improve the quality. Additionally, while improvements would be made with more input image data, the existing CIFAR-10 and STL-10 horse and bird images could be manually curated, choosing those with relevant features (open wings, better matching colours, etc.).

For CIFAR-10 alone, the images tend to converge towards horse-like figures. Interestingly, for STL-10 alone, the definition is most clear towards the centre of the majority of the images. Additionally, the sampled images generally look more bird-like than horse-like (of which there are rarely open wings).

Only the labelled images are used, which is a small subset of all the images provided by STL-10. It may be worth first training a classifier to identify birds and horses so that the rest of the dataset may be used, allowing better learning of the underlying distribution. This classifier could then potentially be extended to identify birds and horses in the centre of the frame (and even birds with their wings spread).

Potentially, a multi-generator (or even multi-GAN) approach could be used to separately train for bird features and horse features (learning their respective distributions) and sample a linear combination of (manually selected) latent noise vectors which give reasonable results.

Finally, GANs are relatively sensitive to changes in hyperparameter values [6]. This means that small changes in their values may result in largely varying results. With more time,

performing a hyperparameter search by either visually inspecting outputs or minimising absolute critic loss would improve results.

Bonuses

This submission has a total bonus of -3 marks (a penalty), as it uses a GAN and was trained with STL resized to 64x64.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks". In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[2] Adam Coates, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 215–223.

[3] Ian J Goodfellow et al. "Generative adversarial networks". In: *arXiv preprint arXiv:1406.2661* (2014).

[4] Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *arXiv preprint arXiv:1704.00028* (2017).

[5] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[6] Mario Lucic et al. "Are gans created equal? a large-scale study". In: *arXiv preprint arXiv:1711.10337* (2017).

[7] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).