Improved Training of Generative Adversarial Networks Using Decision Forests

Yan Zuo* Gil Avraham*
Tom Drummond
ARC Centre of Excellence for Robotic Vision, Monash University, Australia
{yan.zuo, gil.avraham, tom.drummond}@monash.edu

Abstract

Whilst Generative Adversarial Networks (GANs) have gained a reputation as powerful generative models, they are notoriously difficult to train and suffer from instability in optimisation. Recent methods for tackling this drawback have typically approached it by inducing better behaviour on the discriminator component of the GAN; these include loss function modification, gradient regularisation and weight normalisation to create a discriminator that is well-behaved from a Lipschitz perspective. In this paper, we propose a novel and orthogonal contribution which modifies the architecture of a GAN. Our method embeds the powerful discriminating capabilities inherent in decision forests within the discriminator of a GAN. Empirically, we test the effectiveness of our approach on the CIFAR-10, Oxford Flowers and CUB Birds datasets. We show that our technique is easy to incorporate into existing GAN baselines and offers improvements on Fréchet-Inception Distance (FID) scores by as high as 56.1% *over several GAN baselines.*

1. Introduction

Since their introduction, Generative Adversarial Networks (GANs) have achieved considerable success in the realm of generative models, finding applications towards a range of tasks including image synthesis [8, 30]. The objective of a GAN is to produce a model distribution which recovers a target distribution. Theoretically, this involves training a generator network with the goal of minimising a distance measure between the model distribution and the target distribution, where the said measure is estimated by a discriminator network. In this type of optimisation setup, the discriminator could be considered the cornerstone of a GAN; whilst the generator is ultimately responsible for generating the model distribution which should closely resemble the target distribution, it relies heavily on the discriminator to continuously estimate the distance between the two distributions.

This described optimisation process is fragile and suffers from several numerical instabilities. A root cause for this is vanishing gradients. This problem occurs when the model and target distributions are disjoint [1]. Loss functions containing singular points [2] and a non-continuous discriminator [37] are also major causes for instability in the optimisation process. Several earlier works in the literature have attempted to mitigate this issue through regularisation to the model [21, 31, 32]. Recently, the discriminator in the GAN setup has been the focus of significant contributions [1,9,23,24] towards gaining an understanding of GAN mechanics and alleviating the problems associated with training of GANs. From a numerical stand point, the Hessian matrix of a loss function plays an important role in the optimisation process. A work by [23] investigates improving the conditioning of the Hessian matrix in the context of the gradient regularisation. In this work, we induce better conditioning of the Hessian through an architectural modification to the discriminator and significantly improve its performance.

Specifically, our approach focuses on the last layer of the discriminator network which is typically a fully-connected (FC) linear layer. Intuitively, FC layers serve as an interpreter of the representative features collected by preceding convolution layers. We demonstrate that an FC layer's inability to interpret data which is highly non-linear as well as separate out highly-complex correlated data leads to harming the discriminator's capability to model the typically non-linear, joint distributions associated with image data. Incidentally, these are properties that are inherent in decision trees [5]. Through increasingly complex empirical validation, we show that replacing the FC layer with a decision forest improves performance in the discriminator in a GAN setup. Furthermore, we show that our novel architecture provides significant performance gains when integrated into state-of-the-art GAN baselines.

The rest of this paper is organised as follows: first, we give a brief background of the related work. This

*Authors contributed equally

is followed by introducing the necessary preliminaries which motivates inspecting the conditioning of our proposed model. Next, the advantage of using a decision forest over a FC layer is highlighted from a numerical stand point, followed by a toy GAN example. Finally, we show experimental results on a larger scale and offer conclusions.

2. Background

2.1. Generative Adversarial Networks

Generative Adversarial Networks (GANs) [8] minimise the Jenson-Shannon divergence between a target data distribution and the model distribution given by the generator. GANs implicitly estimate a data distribution using the following minimax objective loss function:

$$\begin{aligned}\min_G \max_D V(D,G) = E_{x\sim P_r}[\log D(x)] \\ + E_{z\sim P_z}[\log(1 - D \circ G(z))]\end{aligned} \tag{1}$$

where P_r and P_g are the respective target and model distributions, with P_g implicitly defined by $G(z), z \sim p(z)$ (p is a noise distribution such as Gaussian or uniform).

Initially, GANs were not heavily utilised since the framework suffered from various instability issues during the optimisation process, such as mode collapse and falling short of estimating areas of the target distribution [32]. DCGAN provided initial guidelines for constructing a deep convolutional neural network which attempted to address some of the instability in training and improve modal diversity [30]. Although the framework provided a strong baseline, GANs were considered far from solved. Many follow up works over the years have tackled various problems in the GAN domain and their accumulated contributions have addressed different GAN problems in either an isolated manner or in a collective manner.

The work by [26] extended upon the original GAN framework objective (which used the Jensen-Shannon divergence as a loss), allowing it to use the entire f-divergence family. This proved useful for cases where using a different divergence was necessary, but did not address inherent problems in f-divergence losses such as vanishing gradients. Following this, [2] introduced the Wasserstein GAN (WGAN) as a method to address the vanishing gradient problem. WGANs treated the task of matching distributions as a mass transportation problem, demonstrating the benefits of using the Wasserstein distance as a loss function. WGAN used the Kantorovich-Rubinstein duality [33] to obtain its objective function:

$$\begin{aligned}\min_G \max_{D\in\mathbb{F}_{Lip}} V(D,G) = E_{x\sim P_r}[D(x)] \\ + E_{z\sim P_z}[D \circ G(z)]\end{aligned} \tag{2}$$

where $\mathbb{F}_{Lip}$ is the set of 1-Lipschitz functions. In the original WGAN formulation, Lipschitz continuity was enforced through weight clipping in the discriminator; this was shortly addressed by [9] which imposed a gradient penalty (WGAN-GP) to ensure the discriminator was Lipschitz bounded. However, whilst WGAN and WGAN-GP provided great results in practice [3, 35, 36], it was theoretically shown by [22] that using the Wasserstein distance will not lead to convergence.

Recent work has realised the importance of a Lipchitz-bounded discriminator. In [37], the loss function was altered to induce a Lipchitz behaviour on the discriminator. Spectral Normalisation GAN (SNGAN) [24] imposes a Lipchitz constraint via direct weight normalisation. We note that all the above methods use the original DCGAN network architecture, with slight modifications due to constraints imposed by their specific methods (*e.g.* WGAN and SNGAN do not use Batch Normalisation layers in their discriminators).

2.2. Decision Trees & Forests

Decision forests are well-known for their strong discriminating power [29], although initially they suffered from variance and stability issues and were prone to overfitting [12]. These issues were alleviated via various regularisation methods such as randomisation of the feature subspace and bootstrapping [4, 7], and boosting methods [6, 16, 40]. Most modern works now utilise decision forests within a deep learning context, either by using decision tree methods to influence the training approach [14], or explicitly incorporating decision trees as part of the core architecture [18, 39].

A decision tree consists of a set of internal decision nodes and a set of terminating leaf nodes. The internal nodes, $\mathcal{D} = \{d_0, \cdots, d_{N-1}\}$, each hold a decision function $d(\boldsymbol{x}; \theta)$, where θ are the parameters of the decision node. Each decision node performs a hard routing of an input to its corresponding left or right child decision node according to $d(\boldsymbol{x}; \theta) : \mathcal{X} \rightarrow [0, 1]$. Collectively, the decision nodes map an input sample, $\boldsymbol{x}$, from the root node to one of the terminating leaf nodes: $\ell = \mathcal{D}(\boldsymbol{x}, \Theta)$, where Θ are the collected parameters of the decision nodes of the tree. Leaf nodes hold a set of real values, $\boldsymbol{q}$, which are formed from the training data:

$$q(\ell) = \frac{\sum_i \delta(\mathcal{D}(x_i))v_i}{n_\ell} \tag{3}$$

where n_ℓ is the number of samples routed into leaf node ℓ, δ provides the routing function for the sample through the tree to leaf ℓ and v_i is the observed real value for instance i. A decision forest is an ensemble composed of $\mathcal{T}$ number of decision trees which produces an averaged output of its trees:

$$P(\boldsymbol{x}, \boldsymbol{\Theta}, \boldsymbol{Q}) = \frac{1}{\mathcal{T}} \sum_{t=1}^{\mathcal{T}} Q^t(D^t(\boldsymbol{x}, \Theta^t)) \tag{4}$$

where Q^t, $\mathcal{D}^t$ and Θ^t are the respective values, decisions and parameters of tree t, while $\boldsymbol{\Theta}$ and $\boldsymbol{Q}$ are the collected parameters of *all* trees' decisions and leaf values.

3. Preliminaries

We consider a neural network of depth N to be defined as a series of linear operations: $W^{(\theta_i)} \in \mathcal{R}^{n \times n}$, parameterised by θ_i and non-linear point wise operations: $F : \mathcal{R} \rightarrow \mathcal{R}$, constructed to operate on an input $X \in \mathcal{R}^n$ as follows:

$$\hat{Y}(X, \boldsymbol{\theta}) = W^{(\theta_N)}(F(W^{(\theta_{N-1})}) \dots F(W^{(\theta_1)}X)) \quad (5)$$

Without losing generality, the linear operator $W^{(i)}$ is interchangeable with the convolution operator [20]. Given a set $\mathcal{D}$ of M data measurements $\{X_i, Y_i\}_{i=1}^{M}$ The following loss function is defined for optimising the parameters θ_i of the neural network defined in Eq. 5:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left\| Y - \hat{Y}(X_i, \boldsymbol{\theta}) \right\|^2 \quad (6)$$

where $\mathcal{B} \in \mathcal{D}$ is a batch of measurement pairs $\{X_i, Y_i\} \in \mathcal{B}$ randomly sampled from the set $\mathcal{D}$. The Hessian, denoted as $\mathcal{H}$, is a matrix of the second derivatives of the loss $\mathcal{L}$ with respect to the parameters $\boldsymbol{\theta}$:

$$\mathcal{H}_{m,n} = \underbrace{\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \frac{\partial \hat{Y}}{\partial \theta_m} \frac{\partial \hat{Y}}{\partial \theta_n}}_{\mathcal{G}_{m,n}} - \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (Y - \hat{Y}) \frac{\partial^2 \hat{Y}}{\partial \theta_m \partial \theta_n} \quad (7)$$

The matrix $\mathcal{G} \in \mathcal{R}^{Nn^2 \times Nn^2}$ in the under-brace of Eq. 7 is an approximation of the Hessian and also known as the Gauss-Newton matrix. Note that for the loss function defined in Eq. 6, the Gauss-Newton matrix is equal to the Fisher information matrix [10]. The spectral density of the Gauss-Newton matrix defines the degree of local curvature for the loss surface defined in Eq. 6 and provides a good estimate of the efficiency expected in first-order optimisation methods. We define the empirical spectral density of the Gauss-Newton matrix $\mathcal{G}$ similar to [27]:

$$\rho_{\mathcal{G}}(\lambda) = \frac{1}{Nn^2} \sum_{i=1}^{Nn^2} \delta(\lambda - \lambda_i(\mathcal{G})) \quad (8)$$

with $\lambda_i(\mathcal{G})$ being the eigenvalues of $\mathcal{G}$, and δ the Dirac delta function. The moments of the spectral density provide the statistics of the spectral density and are defined as:

$$m_k = \int \rho_{\mathcal{G}}(z) z^k dz \quad (9)$$

The conditioning of the Hessian plays a large role in first-order optimisation methods. In [27], a three-way relationship was explored between the Hessian conditioning, the delta of the loss step size, $-\Delta\mathcal{L}$, and various non-linearity activation functions which resulted in different neural network setups. The conclusion was clear; more non-linearity in a neural network architecture correlates with a lower condition number, which also correlates with more efficient optimisation step size (*i.e.* larger $\Delta\mathcal{L}$).

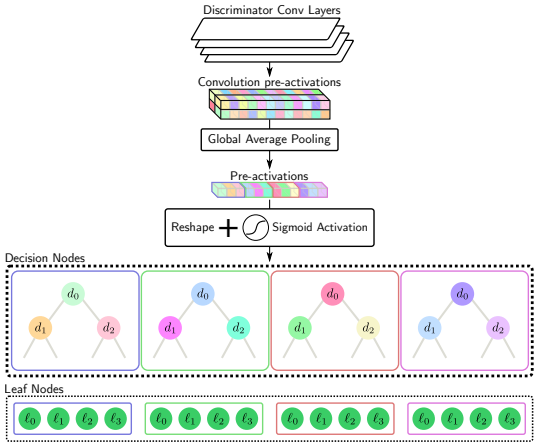


Figure 1: An overview of our proposed changes to the discriminator network in a GAN setup

Classically, the condition number of a matrix is measured by the ratio of the largest to the smallest eigenvalues, $\lambda_{max}/\lambda_{min}$. In this work, we adopt the measure of conditioning as done in [27], by looking at the ratio of the second and first moment of the spectral density function: $\kappa(\mathcal{G}) = m_2/m_1^2$. This form of measuring the conditioning is scale-invariant and robust to spectral density functions which have degenerate components. The lower this conditioning measure is (with $1 \leq \kappa(\mathcal{G})$), the tighter the spectrum is concentrated around its mean, resulting in less pathological curvature occurrences in the loss surface.

4. A Discriminator with a Forest

Our method modifies the architecture of the discriminator network by replacing the final fully-connected (FC) layer of the network with a decision forest: first, we take the set of pre-activations from the last convolutional layer in the discriminator network and apply a global average pooling operation, which reduces the $4D$ ($N \times H \times W \times C$) tensor into a $2D$ ($N \times C$) tensor. N corresponds to the batch size, H, W are the respective height and width of the pre-activation tensor and C is the number of output channels. Following this, the $N \times C$ tensor is reshaped into the decision nodes of the forest (see Fig. 1).

More explicitly, for a single sample in a batch, a pre-activation feature vector with C channels is generated. This vector can be reshaped into C decision nodes of our decision forest. We can adjust the number of trees $\mathcal{N}_t$ and their

depths ($\mathcal{N}_d$) such that the total number of nodes equals C (*e.g.* for a decision forest of 16 trees each of 5 depth, we require a total of $16 \times (2^5 - 1) = 496$ decision nodes, where the preceding convolution layer outputs 496 channel feature vectors).

For the leaf nodes of the decision forest, a $2D$ tensor of $\mathcal{N}_t \times \mathcal{N}_l$ is created, corresponding to the number of trees in the forest and number of leaf nodes in each tree respectively. Then, each leaf node value is generated a probability route from a corresponding set of decision nodes. This dictates its contribution to the ensemble's output, as shown in Fig. 2.

4.1. Soft Decision Trees

Our approach is similar to the approaches of [15, 38], where a decision forest is constructed with decision nodes that create soft decision boundaries, in contrast to the common hard decision boundaries. Here, we adapt the method to work within an end-to-end deep learning framework. Each tree in the ensemble outputs a single prediction value which is the result of blending the values in all leaves in the tree according to their generated proportion values. Each decision forest is composed of $\mathcal{N}_d$ decision nodes and $\mathcal{N}_l$ leaf nodes.

Each decision node d_i is composed of a sigmoid gating function $\sigma(\cdot)$ and a bias value b_i, where i indexes the decision nodes in a decision tree. Given a pre-activation x_i and bias b_i, d_i outputs a left and right subtree probability, where the probability assigned to its left subtree p_i is given by:

$$p_i = \sigma(x_i - b_i) \tag{10}$$

Subsequently, d_i assigns to its right subtree a probability of $\bar{p_i} = 1 - p_i$. The probability route from the root node d_0 of each decision tree to the j^{th} leaf node ℓ_j is then defined as:

$$\mu_{\ell_j}(\boldsymbol{x}; \boldsymbol{b}) = \prod_{i=1}^{\mathcal{N}_d} p_i^{1_{\ell_j \swarrow d_i}} \bar{p_i}^{1_{\ell_j \searrow d_i}} \tag{11}$$

j indexes the leaf nodes in the decision tree, $\mathcal{N}_d$ is the number of decision nodes in the tree and 1_C is an indicator function which equals 1 when its condition C is met and 0 otherwise. Here, $\ell_j \swarrow d_i$ observes whether the j^{th} leaf node ℓ_j belongs in the left subtree of the i^{th} decision node d_i and $\ell_j \searrow d_i$ observes if ℓ_j belongs to the right subtree of d_i. The output of the k^{th} decision tree is then given by combining each of its leaf nodes ℓ according to its respective probability route μ (see Fig. 2):

$$\text{II}_k(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{\ell}) = \sum_{j=1}^{\mathcal{N}_l} \mu_{\ell_j} \ell_j \tag{12}$$

Thus, a soft decision tree achieves its non-linearity through the direct multiplication of activations assigned to its decision nodes. This introduces a larger non-linearity when compared to stacked FC layers, which are point-wise linear operations with non-linearity in between. Through enforcing our decision trees to create soft decision boundaries, our model becomes fully differentiable and can be seamlessly inserted into the discriminator and updated via backpropagation. The output of the discriminator is given by the output of an ensemble of decision trees which is the sum of each decision tree's individual output (as computed by Eq. 12):

$$D(\boldsymbol{x}; \boldsymbol{b}, \mathbf{II}) = \sum_{k=1}^{\mathcal{N}_t} \text{II}_k \tag{13}$$

where k indexes the tree, and $\mathcal{N}_t$ denotes the total number of trees in the ensemble.

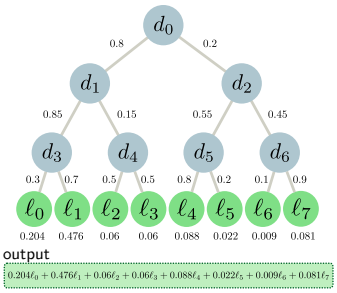


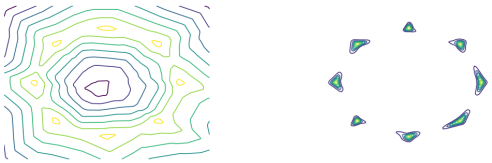
Figure 2: An example of our soft decision trees blending its leaf node values based on probability routes generated by its internal decision nodes

5. Improving the Discriminator

In the Preliminaries, we showed that the Gauss-Newton matrix $\mathcal{G}$ is developed for the case of a squared loss function. In neural network optimisation and specifically GAN optimisation, the loss function is not necessarily the squared loss. Without loss of generality, here we reformulate the problem as a general loss function so that analysing the Gauss-Newton matrix (and as a result, the network's conditioning) can be made possible. We can minimise the loss function *w.r.t* the parameters in the network by treating it as an iterative least squares problem, where the loss can be written as a sum of squares as follows:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \sum_{i \in \mathcal{B}} \mathcal{L}(X_i, \boldsymbol{\theta}) = \sum_{i \in \mathcal{B}} \left(\sqrt{\mathcal{L}(X_i, \boldsymbol{\theta})} \right)^2 \\ &\implies \mathcal{G}_{m,n} = \sum_{i \in \mathcal{B}} \frac{\partial \sqrt{\mathcal{L}(X_i, \boldsymbol{\theta})}}{\partial \theta_m} \frac{\partial \sqrt{\mathcal{L}(X_i, \boldsymbol{\theta})}}{\partial \theta_n} \end{aligned} \tag{14}$$

The Gauss-Newton matrix can be computed as: $\mathcal{G} = J^T J$ where J denotes the Jacobian of the square root of the loss $\mathcal{L}$.



(a) (b)

Figure 3: (a) A small MLP with FC linear layer (b) A small MLP modified with a soft decision forest. Both networks are given the task of solving a filtering problem with additive noise imposed at every Gaussian mode. Improved conditioning of the Hessian matrix allows the soft decision forest to induce a drastically different behaviour on the MLP over its FC linear layer counterpart, where it is able to find much tighter separation boundaries around each Gaussian mode.

5.1. Measuring the Conditioning of the Discriminator

First, we construct an example that will allow us to compute the conditioning of a small MLP when we replace the last FC linear layer with the soft decision forest described in the previous section. The computation of the Gauss-Newton matrix is made possible due to the small number of parameters in both examined networks. Similar to Eq. 5, we define the small MLP as a network composed of 2 hidden layers as follows:

$$\hat{Y}_{FC}(X, \boldsymbol{\theta}) = W^{(\theta_3)} F(W^{(\theta_2)} F(W^{(\theta_1)} X)) \tag{15}$$

where $X \in \mathcal{R}^2$ and is distributed as a ring of Gaussians with 8 modes each of radial distance of 1 and $std = 0.02$ around each mode. The matrices are parameterised as: $W^{\theta_1} \in \mathcal{R}^{32\times 2}$, $W^{\theta_2} \in \mathcal{R}^{8\times 32}$, $W^{\theta_3} \in \mathcal{R}^{1\times 8}$. The pointwise non-linearity function F is a ReLU function. We denote $\hat{Y}_{FC}$ as the output of the network which uses W^{θ_3} as its last layer. Our forest variation replaces W^{θ_3} with $D(\cdot)$, the forest function (Eq. 13), and is defined as:

$$\hat{Y}_{Forest}(X, \boldsymbol{\theta}) = D(W^{(\theta_2)} F(W^{(\theta_1)} X), \theta_3, \mathbf{\Pi}) \tag{16}$$

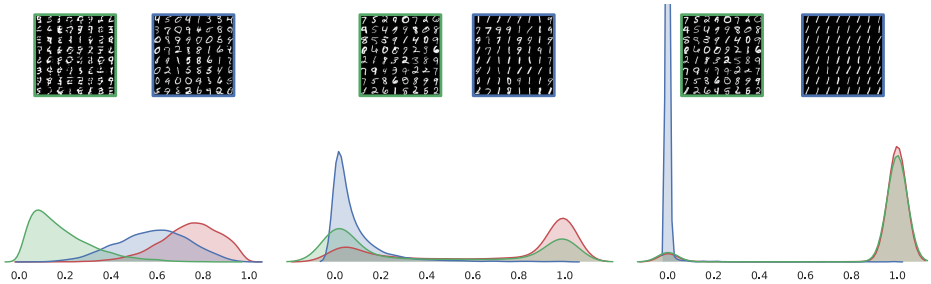
The models defined in Eq. 15 and 16 both optimise the standard GAN loss as defined in Eq. 1. Here, we reduce the minimax problem optimised by GANs by fixing the generator's output to be $X' = X + \epsilon$, with $\epsilon \sim \mathcal{N}(0, I_2)$. In doing so, we can first isolate and compare the performance of the soft decision forest against the FC linear layer, inspecting the difference in operation between the two models [28]. Thus, Eq. 1 is reduced to solving a filtering problem, where we would expect a goood discriminator to assign a high confidence in a tight boundary around each Gaussian in the ring.

Visually inspecting the probability contours for each respective model in Figs. 3a and 3b, where the values of $\hat{Y}_{FC}$ and $\hat{Y}_{Forest}$ are plotted respectively given inputs of $-2 \leq X_1 \leq 2, -2 \leq X_2 \leq 2$, we note that compared to the model with the FC linear layer, the soft decision forest model assigns a much higher confidence in a tight ring around each Gaussian in the ring. Additionally, we compute the conditioning of $\mathcal{G}$ in the form of m_2/m_1^2, using the formulation in Eq. 14, where the empirical spectral density is computed as in Eq. 8 and both m_1, m_2 moments are explicitly computed using Eq. 9. This results in a condition number of $\kappa(\mathcal{G}_{FC}) = 295$ and $\kappa(\mathcal{G}_{Forest}) = \mathbf{288}$ for the FC linear layer model and soft decision forest model respectively. This margin is significant and although computing the conditioning becomes infeasible for deeper networks, this result provides a motivation to test whether larger scale networks perform better given this initial insight.

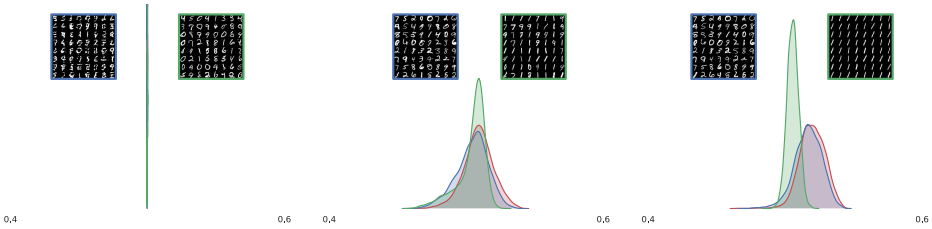
5.2. Visualising MNIST

Following this, we show the improvements that a soft decision forest can offer over its fully-connected linear counterpart on an example involving real images in an adversarial setting. Here, we show that the inherent instability in DCGAN can be addressed by an architectural solution which correctly resolves ill-conditioned gradients. We construct two GAN models for image synthesis on the MNIST dataset [13]: the first model is a DCGAN [30] and the second model is a modified version DCGAN where we replace the last fully-connected layer with a soft decision forest, similar to Eq.16 (denoted as DCGAN-Forest). Both models are trained using the hyperparameters settings described by [30] on the training data of the MNIST dataset for 500k iterations. For the soft decision forest, we use a configuration of 16 trees, each of which is 4 levels deep. This is to ensure that the number of parameters between both models is relatively similar at approximately $0.8M$ parameters. During training, we observe and visualise the distribution of probabilities each model's discriminator assigns to 3 sets of 10k images: samples generated by its own generator (blue), samples generated by the generator of the competing GAN (green) and real image samples from the test data of the MNIST dataset (red) which were not seen during training.

In Fig. 4a, we show the kernel density estimates of probabilities assigned by the discriminator of DCGAN to each set of images at 10k, 200k and 500k iterations during training, along with qualitative samples from the generators of each GAN model. In the early stages of training (10k iterations), the DCGAN discriminator assigns a diverse spread of probabilities to samples from each set, with the real images from the MNIST test set being assigned the highest probabilities on average. In the middle stages of training (200k iterations), the DCGAN discriminator is beginning to separate out samples from its own generator with samples



(a) DCGAN discriminator Kernel Density Estimates



(b) DCGAN-Forest discriminator Kernel Density Estimates

Figure 4: Kernel density estimates (along with corresponding generator samples) of probabilities assigned by the (a) DCGAN discriminator and (b) DCGAN-Forest discriminator to 10k samples each from 3 sets of images: test data from MNIST (red), samples generated by its own generator (blue) and samples generated by the competing GAN (green) at 10k, 200k and 500k training iterations. Estimates for DCGAN-Forest are well distributed which points to more stable gradients, in contrast to DCGAN, which exhibits an erratic distribution. Note that this behaviour is not easily observed in the loss curve of the discriminator.

from the real distribution assigning low and high probabilities respectively. Samples from the DCGAN-Forest generator are largely concentrated at the low and high ends of the distribution, where the DCGAN discriminator finds it difficult to discern between whether these generated samples were drawn from the real distribution. Observing the samples generated by the DCGAN (blue) and competing DCGAN-Forest (green), it is evident that the DCGAN generator has begun to collapse on modes. By the late stages of training (500k iterations), the DCGAN discriminator has completely separated out samples from its own generator and real samples and this corresponds to complete collapse in its generator. We note that although the DCGAN discriminator is able to easily distinguish between its generator's samples and the real samples, it performs poorly when shown samples from the set of images generated by the DCGAN-Forest's generator which has been assigned a probability distribution strikingly similar to that of the unseen test images from the MNIST dataset.

Conversely, in Fig 4b, we observe the kernel density estimates of probabilities assigned by the discriminator of the DCGAN-Forest at 10k, 200k and 500k iterations during training, along with qualitative samples from the generators of each GAN model. In the early stages of training (10k iterations), the DCGAN-Forest discriminator assigns half probabilities to nearly all the samples irrespective of the distribution they were drawn from. In the middle stages of training (200k iterations), the DCGAN-Forest discriminator is beginning to separate out generated samples and real samples. Note that the assigned probabilities to samples from its own generator follow closely the assigned probabilities of samples from the real test images, whilst there is a significant difference to the samples generated by the competing DCGAN generator. By the late stages of training (500k iterations), the probabilities assigned by the DCGAN-Forest discriminator to its own samples and real images has shifted significantly to the higher end of probabilities, whilst the distribution of probabilities assigned to the competing DCGAN's generated samples has remained around the half probability zone. From this, we can observe that the DCGAN-Forest discriminator performs much better when shown samples generated by the DCGAN generator, as indicated by the large mass of lower probabilities it assigns to those samples when compared to its own and real images. We again note that the probabilities assigned to the set of real images and DCGAN-Forest's own generated samples looks strikingly similar.

6. Experiments

We compared our soft decision forest model to three well-established GAN baselines, DCGAN [30], WGAN-GP [9] and SNGAN [24], across three datasets: CIFAR-

(a) DCGAN (b) DCGAN-Forest (c) WGAN-GP (d) WGAN-GP-Forest (e) SNGAN (f) SNGAN-Forest

Figure 5: Qualitative results on CUB Birds (top) and Oxford Flowers (bottom) datasets. We show considerable image quality improvements over all GAN baselines. Note the increased level of detail in our generated samples, resulting in sharper looking images, which is consistent across all GAN baselines.

Method	**Inception Score**	**FID**
DCGAN	6.16±0.07	37.7
DCGAN-Forest (Ours)	**6.66±0.06**	**35.2**
WGAN-GP	6.58±0.06	37.7
WGAN-GP-Forest (Ours)	**6.83±0.04**	**33.2**
SNGAN	7.42±0.08	29.3
SNGAN-Forest (Ours)	**7.52±0.05**	**22.2**

Table 1: Inception Scores and FIDs for DCGAN, WGAN-GP, SNGAN along with corresponding modified Forest counterparts on CIFAR-10

10 [19], CUB Birds [34] and Oxford Flowers [25]. We train the baseline DCGAN, WGAN-GP and SNGAN according to the hyperparameters specified in their respective papers. For DCGAN and SNGAN, we use the loss function defined in Eq. 1 and for WGAN-GP, we use the loss function defined in Eq. 2 with the additional gradient penalty term defined in [9]. For each baseline GAN, we replace the final fully-connected (FC) layer in their discriminator/critic networks with our forest layer. This creates three variants of our modified GAN with forest model and for all our experiments, we refer to our three forest variants of DCGAN, WGAN-GP and SNGAN as DCGAN-Forest, WGAN-GP-Forest and SNGAN-Forest respectively. For the SNGAN baseline, we employ the two-time update rule (TTUR) training schedule as specified in [11], employing a 3:1 ratio in learning rate between the discriminator (0.0003) and generator (0.0001). We found this configuration of SNGAN to give the best performance on all datasets. Similarly, we employ the same TTUR training schedule for our forest variant (SNGAN-Forest).

For all experiments involving our forest variant models, we use a configuration of 16 trees and vary the tree depth to control the number of parameters in the model to match the number of parameters in its corresponding GAN baseline. We use the exact same hyperparameter settings as their corresponding GAN baselines. We use the ADAM optimiser [17] with a batch size of 64, training for 100k iterations on the CIFAR-10 dataset and 30k iterations on the Oxford Flowers and CUB Birds datasets.

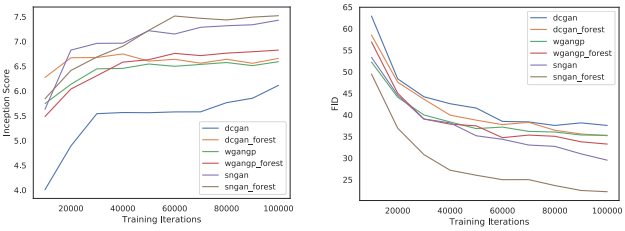
6.1. CIFAR-10

Experiment Settings For the CIFAR-10 dataset, we use decision trees of depth 5. For each GAN baseline, we use the standard CNN configuration listed in their respective papers which consists of approximately $2.6M$ parameters. Our forest variant model also consists of approximately $2.6M$ parameters.

Quantitative Results In Table 1, we report our results on the CIFAR-10 dataset, comparing each GAN baseline to its forest variant counterpart. We can see that across all 3 GAN baselines, our forest variant offers a significant improvement in both Inception Score and Fréchet-Inception Distance. Additionally, we also plot Inception Score [32] and FID [11] curves in Figs. 6a and 6b respectively. Observing the evolution of Inception Scores and FID over the course of training, we can see that each modified forest variant offers significantly better training convergence than its corresponding GAN baseline.

6.2. Oxford Flowers and CUB Birds

Experiment Settings For the CIFAR-10 dataset, we use decision trees of depth 6. For each baseline GAN, we use



(a) IS (b) FID

Figure 6: (a) Inception Scores and (b) FID curves over 100k training iterations of DCGAN, WGAN-GP and SNGAN along with their corresponding modified forest variants on the CIFAR-10 dataset.

the standard CNN configuration listed in their respective papers which consists of approximately $9.7M$ parameters. Our forest variant model consists of approximately $9.6M$ parameters.

Method	FID	
	Oxford Flowers	CUB Birds
DCGAN	82.0	59.0
DCGAN-Forest (Ours)	**67.2**	**53.4**
WGAN-GP	80.4	60.3
WGAN-GP-Forest (Ours)	**35.3**	**49.6**
SNGAN	53.2	57.1
SNGAN-Forest (Ours)	**32.6**	**44.6**

Table 2: FIDs for DCGAN, WGAN-GP and SNGAN along with corresponding modified Forest counterparts on Oxford Flowers and CUB Birds

Qualitative Results In Fig. 5, we show qualitative samples of our models where a significant improvement in sample quality can be observed in the samples generated by our forest models when compared with their corresponding GAN baselines.

Quantitative Results In Table 2, we report our results on the Oxford Flowers and CUB Birds datasets, comparing each GAN baseline to its forest variant counterpart. Once again, our forest variants offer a drastic improvement in Fréchet-Inception Distance over all GAN baselines.

6.3. Comparing the Soft Forest with FC Layers

Finally, we study the difference between our soft decision forest when compared with stacked FC linear layers. Here, we observe the difference in FID performance between our soft decision forest compared to just naively stacking FC linear layers in the discriminator. We take our strongest performing GAN baseline (SN-GAN) and modify it by extending the number of FC linear layers from 1 to 3 FC linear layers. We also experiment with different non-linearity activation functions including Leaky ReLU and Sigmoid functions. For the stacked FC layer setup, we halve the output channels with each FC layer added from its starting number (*e.g.* for 3 FC layers, the output channels changes from 512 to 256 to 128 to 1). As observed, our soft forest significantly outperforms the stacked FC layer setup which offers further evidence that improved conditioning offered by our soft decision forest aids in the image synthesis process.

Method (SN-GAN baseline)	FID		
	CIFAR-10	Oxford Flowers	CUB Birds
1FC (Sigmoid)	30.5	55.5	58.2
1FC (Leaky ReLU)	29.3	53.2	57.1
2FC (Sigmoid)	29.2	57.6	60.1
2FC (Leaky ReLU)	29.1	55.0	56.9
3FC (Sigmoid)	29.9	58.2	59.3
3FC (Leaky ReLU)	28.8	54.9	57.6
Forest (Ours)	**22.2**	**32.6**	**44.6**

Table 3: Ablation study comparing our soft decision forest with stack FC layers with different non-linear activations.

7. Conclusion

This paper presents a new approach for unsupervised training of a GAN, modifying the discriminator with a decision forest. The architecture of our model reflects a key insight that incorporating a decision forest into a neural network increases non-linearity which results in a better conditioned Hessian matrix and improved stability in training. We show specific improvements within an adversarial setup where our soft decision forest layer observes a near perfect Nash equilibrium state on a toy image generation task when replacing its fully-connected linear layer counterpart in a discriminator. Finally, we show our architecture can be seamlessly integrated within state-of-the-art GAN methods and provide significant performance gains over these GAN baselines.

Acknowledgments The authors would like to thank the anonymous reviewers for their useful and constructive comments. This work was supported by the Australian Research Council Centre of Excellence for Robotic Vision (project number CE1401000016).

References

[1] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[3] Gil Avraham, Yan Zuo, and Tom Drummond. Parallel optimal transport gan. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4411–4420, 2019.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[6] Yoav Freund. Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pages 202–216, 1990.

[7] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[10] Tom Heskes. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000.

[11] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[12] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[13] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

[14] Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

[15] Ozan Irsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1819–1822. IEEE, 2012.

[16] Michael Kearns. Thoughts on hypothesis boosting. *Unpublished manuscript*, 45:105, 1988.

[17] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475. IEEE, 2015.

[19] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[22] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? *arXiv preprint arXiv:1801.04406*, 2018.

[23] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of gans. In *Advances in Neural Information Processing Systems*, pages 1825–1835, 2017.

[24] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[25] M-E Nilsback and Andrew Zisserman. A visual vocabulary for flower classification. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1447–1454. IEEE, 2006.

[26] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems*, pages 271–279, 2016.

[27] Jeffrey Pennington and Pratik Worah. The spectrum of the fisher information matrix of a single-hidden-layer neural network. In *Advances in Neural Information Processing Systems*, pages 5410–5419, 2018.

[28] Claudia Perlich, Foster Provost, and Jeffrey S Simonoff. Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research*, 4(Jun):211–255, 2003.

[29] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

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

[31] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems*, pages 2018–2028, 2017.

[32] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[33] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[34] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[35] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect. *arXiv preprint arXiv:1803.01541*, 2018.

[36] Jiqing Wu, Zhiwu Huang, Janine Thoma, Dinesh Acharya, and Luc Van Gool. Wasserstein divergence for gans. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 653–668, 2018.

[37] Zhiming Zhou, Jiadong Liang, Yuxuan Song, Lantao Yu, Hongwei Wang, Weinan Zhang, Yong Yu, and Zhihua Zhang. Lipschitz generative adversarial nets. *arXiv preprint arXiv:1902.05687*, 2019.

[38] Yan Zuo, Gil Avraham, and Tom Drummond. Traversing latent space using decision ferns. In *Asian Conference on Computer Vision*, pages 593–608. Springer, 2018.

[39] Yan Zuo and Tom Drummond. Fast residual forests: Rapid ensemble learning for semantic segmentation. In *Conference on Robot Learning*, pages 27–36, 2017.

[40] Yan Zuo and Tom Drummond. Residual likelihood forests, 2020.