Forgetibility Through Pruning in Deep Learning

Rohan Singh
B19EE072

Susim Mukul Roy
B20AI043

Abstract

In neural networks, forgetfulness refers to the ability of the model to retain previously learned information when presented with new data. This ability is critical in various applications, such as image generation models like DALL-E 2. However, such models have also raised concerns about their usage of art without consent, as they can generate images based on copyrighted art without permission from the original creator.

To address these ethical concerns, some researchers have proposed methods to make neural networks forget previously learned information, including art. For instance, one approach is to use a technique called "generative forgetting," which involves training the model to generate similar but different images each time it receives the same input, thus creating a sense of randomness and avoiding copyright infringement. Another approach is to use "memory decay," where the model gradually forgets previously learned data over time.

Overall, making neural networks forget about copyrighted art might be useful in ensuring ethical and legal usage of image generation models like DALL-E 2. However, it is essential to balance the benefits of forgetfulness with the potential risks of losing important information and performance degradation in other applications.

We introduce a way to prune weights in a deep neural network which selectively targets learned data while minimizing the impact on other data. This can be achieved through calculating the sensitivity of the weights w.r.t the input.

1. Introduction

Models like DALL-E 2 have shown us the power of using neural networks to create realistic and creative images. However, these models require vast amounts of data and computation power, making them inaccessible to many researchers and practitioners. In this context, Conditional Generative Adversarial Networks (GANs) have emerged as a popular alternative, allowing us to generate images using much smaller datasets.

In this project, we aimed to replicate the success of DALL-E 2 using Conditional GANs to generate handwritten digits in 10 classes using the MNIST dataset. We fed the model with both the label and a random vector as inputs for its generator, allowing it to produce realistic images for each class. However, we also aimed to improve the model's performance and reduce its size by pruning the most sensitive weights in the generator.

To achieve this, we developed a method similar to Grad-CAM to calculate the sensitivity of each class in the generator. Then, we selected the top k% of the highest sensitive weights and pruned them from the model, reducing its size and making it more efficient. By combining the power of Conditional GANs with weight pruning techniques, we aimed to create a more efficient and scalable model for generating images.

2. Conditional GAN

Conditional Generative Adversarial Networks (cGANs) are a type of generative model that allow us to condition the network with additional information such as class labels. Unlike traditional GANs, which generate data without any specific target in mind, cGANs can generate images with specific attributes based on the input condition.

During training, cGANs are fed with images along with their actual labels (handwritten digits in our case) to learn the difference between them. This allows us to train a model to generate images of specific classes, given the appropriate input conditions. For example, we can ask a cGAN to generate images of zeroes, and it will generate images that look like zeroes.

To condition a GAN, we need to inform both the Generator and Discriminator about the type of data they are dealing with. This can be achieved by adding extra inputs to the network that specify the input conditions. For example, if we want to create synthetic data containing house prices in London and Madrid, we can tell the Generator which city to generate the data for each time, and inform the Discriminator whether the example passed to it is for London or Madrid.

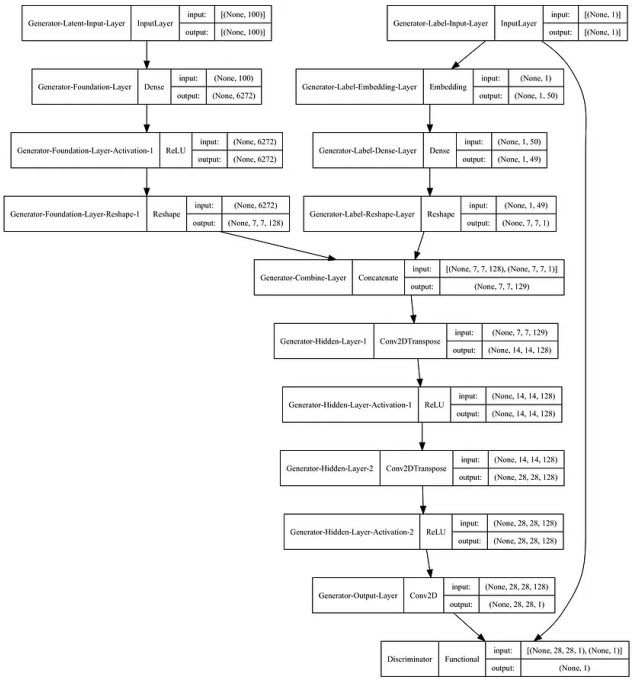

Figure 1. cGAN Architecture

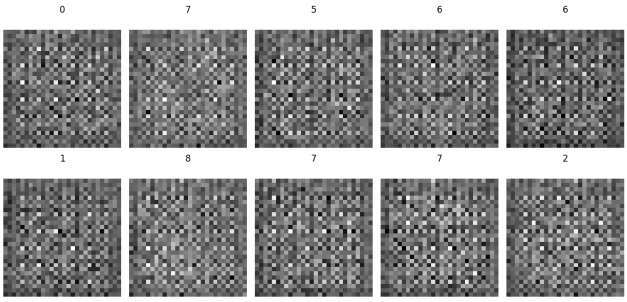

Figure 2. cGAN at epoch 0

Our Architecture looks like below:

cGANs can also be conditioned on other types of inputs, such as other images, for tasks such as image-to-image translation. The architecture of a cGAN can be extended to include additional layers, such as convolutions and transposed convolutions, to create a Conditional Deep Convolutional GAN (cDCGAN) for more complex image generation tasks.

Overall, cGANs are a powerful tool for generating images with specific attributes, and their flexibility in conditioning on various types of inputs makes them suitable for a wide range of applications.

The training process of our conditional GAN can be visualized below:

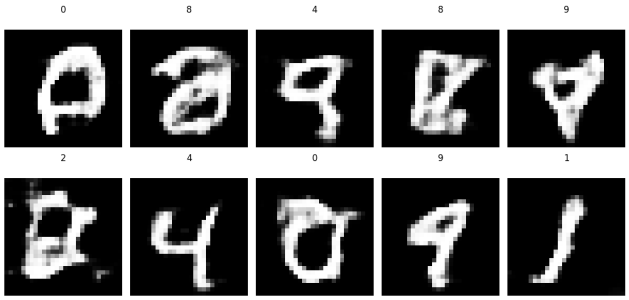

Figure 3. cGAN at epoch 5

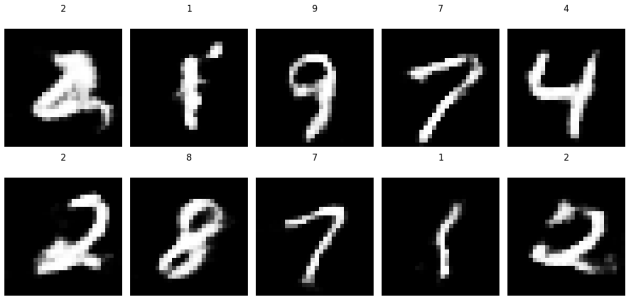

Figure 4. cGAN at epoch 10

3. Senstivity of the Weights

To calculate the sensitivity of the model, we first define a function that takes in the model and an input x. The function uses a GradientTape object to record the operations executed by the model and compute the gradients of the output with respect to each trainable weight in the model.

Once we have the gradients, we iterate over each weight and its corresponding gradient, and calculate the sensitivity by multiplying the gradient with the weight. This gives us the contribution of each weight to the output of the model for the given input x.

The sensitivity values obtained from this function can be used to identify the most important weights in the model for a given input. We can then use a pruning technique to remove the weights with the lowest sensitivity values, thereby reducing the size of the model while maintaining its accuracy.

The mathematical expression for calculating the sensitivity of the output to each weight for a given input x is:

$$sensitivity_i = \frac{\partial output}{\partial weight_i} * weight_i$$

where $sensitivity_i$ is the sensitivity of the output with respect to the ith weight, $output$ is the output of the model for the given input x, $weight_i$ is the ith trainable weight in the model, and $\frac{\partial output}{\partial weight_i}$ is the gradient of the output with respect to the ith weight.

The function iterates over all trainable weights i in the model, computes the corresponding sensitivity $sensitivity_i$ using the above expression, and stores the sensitivities in a list to be returned at the end of the function.

4. Pruning Weights

The function 'create_mask_for_top_k_percent' generates a binary mask for the top k% most sensitive weights in a model based on a list of sensitivities.

The function starts by computing the absolute value of the sensitivities for each weight in the model. These absolute values are flattened into a single 1D tensor, and the threshold value for the top k% most sensitive weights is computed by finding the k-th largest value in the tensor using the tf.nn.top_k function.

The threshold value is used to create a binary mask that indicates which weights are in the top k% most sensitive by comparing the absolute value of each sensitivity to the threshold. If the absolute value of a sensitivity is greater than or equal to the threshold, the corresponding element in the mask is set to 1. Otherwise, it is set to 0.

The function returns the binary mask as a list of binary tensors, with each tensor representing the top k% most sensitive weights for the corresponding layer in the model. This mask can be used to prune the model by setting the weights with a value of 0 in the mask to 0, effectively removing them from the model. By removing the least important weights, the model can become more efficient and have a smaller memory footprint.

5. Results

After pruning, the cGAN is no longer able to generate images for the target class. This could be because the pruning process has removed some of the weights that were important for generating images of the target class.

In the context of the cGAN, the generator model is conditioned on the input class label to generate images of the corresponding class. During training, the model learns the mapping between the input class label and the output image by adjusting the weights of the model. If some of the important weights for generating images of the target class were pruned, the model may no longer be able to generate images of that class.

Hence, we have succeeded in selectively targeting a particular class without hurting the performance on other classes significantly. This shows that selective forgeting is possible in neural networks and can be explored further.

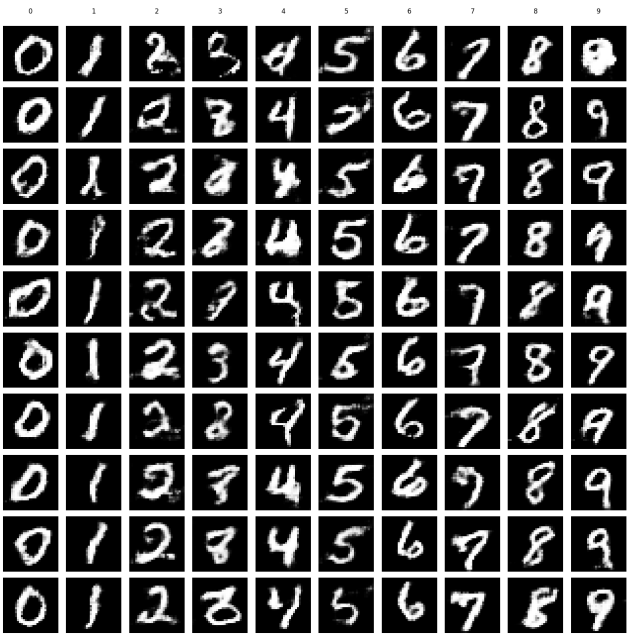

Figure 5. Final Results