Image Generator

Authors: Adnan Al Medawer, Leo Hjulstrom, Lukas Toral, Xiaoxu Wu

Data source

Code description

The source code of this project can be found in a GitHub repository https://github.com/Wuxiaoxue888/ethics.

Main idea

As described in the presentation, we were intrigued by the idea of generating images. As our inspiration we used the Generative Adversarial Network (GAN) that utilises two models - Generator and Discriminator. The discriminator is model trained on "real" (original images) and is used to evaluate the images generated by the generator model. The essence of this structure is that the generator model is forced to continuously improve to learn how to fool the discriminator model into thinking that the new images are real.

Our implementation

Discriminator

From the description of the discriminator in the section Main idea, it is obvious that the model plays an important role in the whole architecture. Let's quickly imagine our system without it. We ask the generator to create new images. It is important that we provide feedback to the generator model to evaluate how well are the new images generated. By doing this manually, the process will be very slow as we would have to evaluate every single picture. Thankfully we can utilise the discriminator model to do the evaluation automatically without any human interaction.

Implementation

Our discriminator is implemented as a neural network model trained on the original images. To be more precise it is a feed-forward neural network with 784 input nodes, one hidden layer with 100 neurones and one output node. The hidden layer neurones use ReLU as their activation function and the one output neuron use Sigmoid for its activation function. As hinted by the presence of the Sigmoid activation function on the output neuron, it is a regression NN that outputs the confidence (probability), that a given image is from the original set. To optimise the network we use Stochastic gradient descent that tries to minimise the loss given by the Binary Cross-entropy loss function. The motivation to use this loss function is because our problem is basically a binary classification - either the given image is real or fake.

Multiple discriminators

During our experiments with the discriminator we faced a problem that the images generated are too similar. To deal with this, we decided to create a pool of discriminator models whose hyper parameters are the same but theirs training data differ. As described in the section Data source, we used 25000 real and fake images. Those images are split into smaller sets for each model: if we train 10 discriminator models, each model will receive 25000/10 = 2500 both real and fake images.

When an image is evaluated by the discriminator, we randomly select a discriminator model that will be asked to evaluate it. The reason we did this was that after few thousands of generations, the images start to become extremely similar, by using multiple models, we get different ranking of the images each generation which increases the variation.

Retraining discriminators

To force the generator to continuously improve, the discriminator has to get better as well. That is why we implemented the retraining of our discriminators. What this means is that when we generate enough ("enough" being defined in code as exactly 2000 but this is configurable) of new images, we give them to our discriminators so that they can recognise the generated images as fake.

Generator

The generator is a model responsible for creating art. There are many ways to implement such model, we have decided to use genetic algorithm.

Implementation

Our generator follows the classic paradigm - create initial population, evaluate the fitness of the population, select individuals that proceed to the next generation, create offsprings and mutate selected individuals. The cycle repeats itself until a final population is reached which represents our generated images.

Initial population

The initial population consists of two types of images: randomly generated or doped. Let's start with the simpler one - the doped images. Our population can be boosted by injecting a certain number of real images by a process called doping. These images are randomly selected from the real set and represent the perfect solutions that will be mated with the weaker one to create a better image in the end. On the other hand, the randomly generated not so aesthetically pleasing but a bit more interesting. We start with a blank 28x28 pixel image. Each pixel has a probability of 0.1% to become black. There are certain things that can change this probability. The closer this pixel is to the middle the higher probability it has that it will become black. We combine this process with second probability that applies only if the selected pixel has a neighbouring black pixel. If it has one it has a 15% probability to turn black which forces the initial images to have some shapes inside them.

Evaluating fitness of the population

The fitness of the population is evaluated using the discriminator models. The whole population is given to a model which returns a probability estimate for each image that defines how sure the model is whether this image is real or not. Higher probability means better fitness of the image.

Selection functions

We won't go into too much detail in this section as we use three standart selection functions for genetic algorithms: Select top N images from the population ordered by fitness, Tournament selection and Roulette wheel selection.

Crossover functions

We have implemented two approaches to crossover the chromosomes of the parent images: vertical and horizontal. They both work by combining parts of the chromosomes in their respective orientation. For a pair of two parents it is not certain which is the best orientation to use or where exactly to split them. To address this, we have created an algorithm that finds the version of a split and uses it to create two new offspring images.

Mutations

As per good practise in genetic algorithms, mutations help introduce something new in the chromosome regardless if it is good or bad. When a new image is created there is a 20% chance that it will be mutated.
In our code there are three types of mutations:
- type 1: 6 to 10 pixel are randomly selected whose value will be changed by adding new random value between -255 and 255
- type 2: A random pixel is selected. If the pixel is on the edge, it will be turned white as well as all its neighbouring indexes. Else if the pixel is almost white (value lower than 0.4) it will be turned white as well as all its neighbouring indexes. Else a black straight line is drawn on the image.
- type 3: A random portion of the image is turned white
While selecting the mutation function, a random combination of the three functions is drawn and applied to the image's chromosome.

Results

TODO paste some images here