Evolutionary Algorithms for Image Generation

By James Liu, Jonnathan Petote, Brandon Plihal, and Kevin Wang

Problem Statement

The purpose of the project is to artificially generate realistic-looking images. We use the decoder portion of an autoencoder as a generative model. That is, the decoder takes an input from latent space and outputs an image. The image is then fed into a convolutional neural network classifier, which outputs a scalar measuring the probability of that image being of a given class. The autoencoder and classifier have been trained on the MNIST dataset, so we are trying to generate realistic-looking zeros.

Our objective function is the concatenation of the decoder and the classifier; more specifically, we want to maximize the classifier's "zero" output unit. The input domain is the latent space of the autoencoder, which is an 128-dimensional array of floats. The weights of the decoder and classifier are fixed for the purposes of this project.

For initial points, we sample from a multivariate normal distribution, then filter out points that either have too low or too high activation (to make things interesting). The sampling distribution itself matches the mean and covariance statistics of MNIST training points that have been passed through the encoder network.

We use gradient descent as a benchmark optimizer, and compare the performance of CMA-ES and genetic algorithm. Finally, for reasons explained later, the original objective function itself is a poor metric for the quality of the output, so we design some custom metrics to better distinguish good and bad zeros.

Previous Work

A naive approach to image generation is to do gradient ascent on a classifier with respect to the inputs. For example, we could take a cat/dog neural net classifier on the CIFAR dataset, and try to change the pixels of an input using backpropagation, until the classifier says the image is of a cat, and the hope is is that the image then looks like a cat.

However, it has been demonstrated that the result of this process is an input that is able to "fool" the classifier, meaning that the classifier is highly confident that the image is of the desired class, yet the image does not look like that class. In fact, the resulting image often looks almost the same as the starting image, be it random noise or a different class. [0][1]

The reason why this happens is because effective models such as convolutional neural networks exploit the idea that natural images tend to lie on lower-dimension manifolds. [2]

In other words, the classifier has only been trained to give correct results for very small areas of the space of all possible images. The aforementioned gradient descent

technique quickly causes the input image to leave these areas, meaning that the classifier will give meaningless results.

A more sophisticated approach to this problem is to use a generative model such as an autoencoder or a generative adversarial network in conjunction with the classifier. Instead of doing optimization with respect to the input in pixel space, we can instead optimize the input in the latent space of the generator. One rationale for this approach is that the latent space is of lower dimension than pixel space (lowering the possibility of "overfitting"). Moreover, movement along one dimension in latent space tends to correspond to some meaningful, intuitive concept such as "when I increase this value, the image shifts to the right". This happens since generative models learn useful representations of the data during the training process, so that the changed image remains on the manifold of natural images. In contrast, moving along one dimension in image space corresponds to changing the value of one pixel, which usually does not correspond to a meaningful, high-level concept.

The GAN approach has been demonstrated to work well; the discriminator portion of the GAN is able to ensure that any image output from the discriminator is realistic. [3]

Hypothesis

We can view the problem of "fooling images" as being a problem of spurious local maxima, a problem that persists despite using the latent space approach. We operate under the hypothesis that good zeros tend to lie in areas where the objective function is smooth, whereas bad zeros (high activation but visually poor) lie in areas where the objective function is unstable. In other words, we expect that a good zero will still be classified as zero if perturbed, but a bad zero will not. Therefore, we mostly focus on gradient-free approaches, in the hopes that they can escape poor local maxima.

Experience

For the metrics, we tried perturbations both in latent space and in image space, as well as different statistics over the different perturbation samples: mean, median, and low percentiles. The low percentiles are meant to be a more stable version of the minimum. The exact choice of metric is necessarily subjective, but we found that perturbations in image space did not at all capture the distinction between good and bad zeros, whereas perturbations in latent space tended to penalize bad zeros more than good zeros. The final perturbation metric that we used was the 10th percentile result of 200 perturbations; each perturbation was created by adding uncorrelated normal noise to each dimension in latent space with mean 0 and .1 standard deviation.

A natural question arises about whether we could use the perturbation metric in the optimization process, but there are many drawbacks to this approach. First, it is extraneous, since many of the optimization algorithms we tested already operate by sampling inputs close to past good inputs. Second, the optimization process could overfit to this metric, making it less meaningful as a final measure of quality. Lastly, it is computationally expensive, since each the total number of function evaluations is multiplied by the number of samples taken for each point.

As a benchmark, we use gradient descent as the optimizer. Using the gradient with respect to the post-softmax zero activation does poorly, demonstrating the classic result of a mostly unchanged image having activations close to 1. Using the pre-softmax zero score gives a better result, in accordance with Simonyan and Vedaldi[4], who point out that maximizing the post-softmax value can be done by minimizing the image's score on the other classes, instead of focusing on maximizing the image's score on the desired class. Accordingly, for the rest of the optimization methods, we try to maximize the pre-softmax zero score.

CMA-ES was able to get results with high objective function values, but the resulting images suffered from a degree of mode collapse. Generally, the image quickly improves (both visually and output value) over the first 100 iterations, but only achieves modest changes afterwards. For testing purposes, we used 500 iterations. Tweaking the initial sigma (variance) value was sufficient to get these results. Values between .5-1 lead to similar results, as the algorithm adapts sigma, but high initial sigma values cause the algorithm to diverge. We used a fairly small sample population of about 15 per iteration; increasing this number doesn't seem to change much. Lowering the population to 10 seems to lead to less mode collapse, but also increases the chance of creating a bad zero.

The genetic algorithm produced moderate results both visually and with our metric. However, the algorithm that we used ran quickly due to a fast convergence rate (after approximately 10 iterations) and the rest of the iterations just fine tuned the pictures to obtain the highest confidence that the image was a 0, but visually had no noticeable difference. Compared to our CMA-ES algorithm, which produced the best results, the genetic algorithm ran for 100 iterations compared to CMA-ES's 1000 iterations. During our experimentation we played around with many aspects of the algorithm to best fit our data and our problem, most notably our initial population size, and our mutation rate.

In the end we ended up using an initial population size of roughly 4,500 (5 copies of each image). We duplicated the images so that we could get more variations and mutations of each image during the optimization process. We initially using an initial

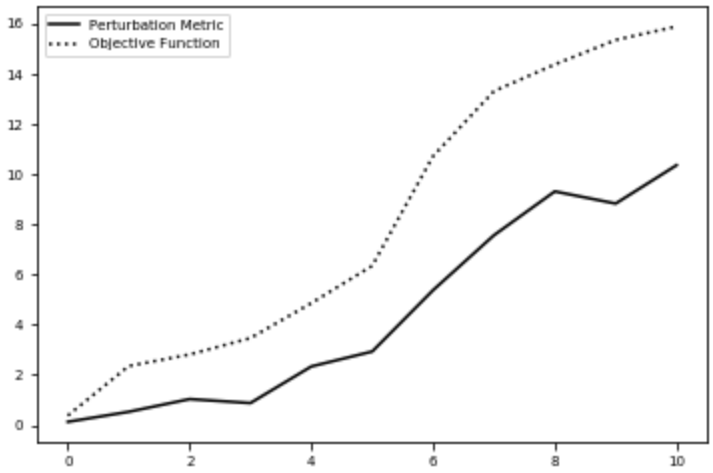
population size of 25 copies of a single seed image, but quickly found that this did not provide enough variation to converge onto a reasonable local minimum. Despite this, the results with using just a single seed image multiple times appeared to be much cleaner, with less grey fuzz around the image, but would not look like zeroes (e.g. we got many figures that looked like 5's). The mutation rate that we used was 1%. We ran many tests at different mutation rates, from 10% to .001%. The 10% mutation rate ended up just producing a bunch of noise, and the .001% mutation rate had such little variation that it would have taken several thousand iterations just to see a difference.

We decided to represent and mutate the individuals in binary format. By converting the number into binary and randomly flipping the bits 1% of the time we were able to obtain much more variation and sample a large space. This is due to the fact that a change in one of the more significant bits led to much more drastic changes than just an increase or decrease of the feature by a Gaussian distribution with mean of 0 and standard deviation of 1. [5]

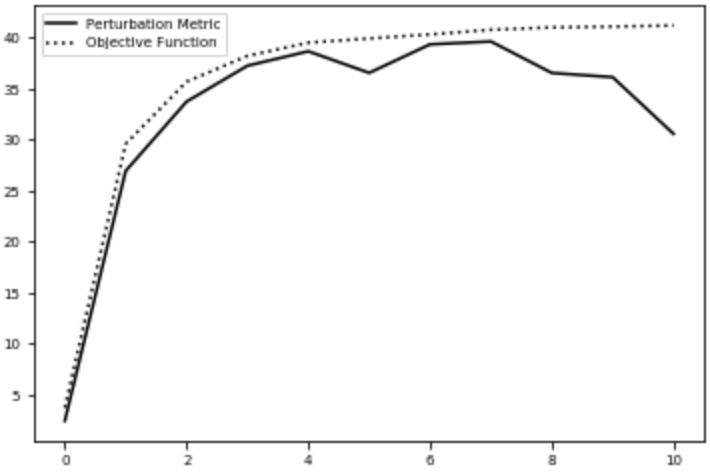
Results

Example Runs:

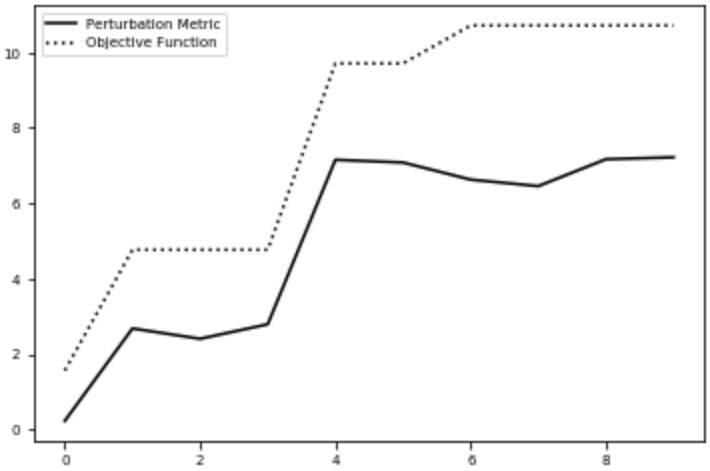
Gradient Ascent (1000 iterations):



CMA-ES(1000 iterations):



Genetic Algorithm (100 iterations):



Example Results:

MNIST Dataset:

Samples:

Gradient Ascent:

CMA-ES:

Genetic Algorithm:

Average Scores:

Source	Objective	Perturbation Metric
MNIST	13.89+-3.41	12.42+-3.39
Samples	2.07+-1.66	0.63+-1.47
Gradient Ascent	18.44+-4.98	12.76+-4.98
CMA-ES	37.20+-9.67	36.12+-9.89
Genetic Algorithm	12.78+-1.91	9.45+-2.45

Conclusion

The results indicate that using gradient-free methods can overcome some of the shortcomings of a given generative model. CMA-ES demonstrated itself to be fairly robust and straightforward to use, whereas the genetic algorithm requires a lot of hand-tuning and design of the gene. Some further work in this area would be to combine the usage of such methods with a generator from a GAN, and to investigate the cause of mode collapse in the setup we used. The fact that the methods are gradient-free also opens up the possibility of using latent spaces that have discrete components, though of course certain generative models may not be amenable to discrete latent spaces. The perturbation metric was somewhat able to penalize gradient ascent's adversarial examples, relative to the objective function, but not as much as would have been hoped. The mixed success of the perturbation metric seems to suggest that the hypothesis that adversarial examples exist in relatively small basins of local maxima is not correct, because many bad zeros still had higher metric scores than those of real zeros. A further improvement to the metric could be to choose a more sophisticated or adaptive sampling distribution than the uncorrelated multivariate normal distribution we used. However, because of the curse of dimensionality (i.e. that training data can only cover an infinitesimally small area of any high-dimensional space), there's no clear guarantee about the behavior of neighborhoods far away from training data. The problem of creating zeros on MNIST is not useful in itself, but it's a stepping stone to more interesting generative modeling projects, such as synthesizing molecules that have specified chemical properties.

Responsibilities:

James: Wrote most of the report, gradient descent benchmarks, CMA-ES implementation.
Kevin: Created perturbation metrics.
Brandon: Implemented genetic algorithm.
Jonathan: Implemented genetic algorithm.
Everybody worked on the poster, and wrote on the results from their coding task.

Outside Resources:

Libraries: pycma[6], deap[7], keras, tensorflow
Database: MNIST [8]
Pretrained model (default.py): Stephen McAleer (Pierre Baldi's lab) [9]

References

[0] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow: “Intriguing properties of neural networks”, 2013; arXiv:1312.6199.

[1] Anh Nguyen, Jason Yosinski: “Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images”, 2014; arXiv:1412.1897.

[2] Ian Goodfellow and Yoshua Bengio and Aaron Courville: “Deep Learning”, MIT Press, 2016; deeplearningbook.org/version-2015-10-03/contents/manifolds.html.

[3] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox: “Synthesizing the preferred inputs for neurons in neural networks via deep generator networks”, 2016; arXiv:1605.09304.

[4] Karen Simonyan, Andrea Vedaldi: “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, 2013; arXiv:1312.6034.

[5] *A Visual Guide to Evolution Strategies*. blog.otoro.net/2017/10/29/visual-evolution-strategies/.

[6] PyCMA,GitHub repository, https://github.com/CMA-ES/pycma

[7] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau and Christian Gagné, “DEAP: Evolutionary Algorithms Made Easy”, Journal of Machine Learning Research, pp. 2171-2175, no 13, jul 2012

[8] LeCun, Yann and Cortes, Corinna. "MNIST handwritten digit database," 2010;.

[9] McAleer, Stephen. “default.py”, 2017. Unpublished Code