Jul 27th, 1:15 PM - 2:15 PM

Synthesizing Pictures From Text Using a DC-GAN

Anton Soloviev
Susquehanna University

Synthesizing Pictures From Text Using a DC-GAN

Anton Soloviev

Susquehanna University, Department of Mathematical Sciences

ABSTRACT & INTRODUCTION

Generative Adversarial Text-to-Image Synthesis (Reed et al., 2016) is a model that can synthesize images based on given text – we have worked to try to apply to different data and to try to improve results seen in the original paper. The model performs two main tasks – it collects relevant information about the images to form a text feature representation of each of the images and it uses these learned text features to then synthesize images from given (new) text. To accomplish this, the model uses a DC-GAN (deep convolutional generative adversarial network) which has been conditioned on the text features coming from the visually-discriminative vector representations of the images that are assembled from the training data set. The features are then encoded by a hybrid character-level CRNN (convolutional recurrent neural network) to achieve feed-forward inference. There are several algorithms that can be used to produce synthesized images: GAN, GAN-CLS, GAN-INT, and GAN-INT-CLS. Each of those algorithms have varying results; their performance depends on the type of dataset used to train them, and the type of results you want to see from your input text.

We apply this model to images gathered from the ImageNet datasets, the COCO (Common Objects in Context) dataset, among others. Raw images are needed to teach the model text feature representations. On top of that, the data we used also contains torch files, which are essential as they have the human transcribed descriptions of the images based on the scene and objects in each image. The descriptions of the images are used to train the model with the images to produce inference between the images and the text. As new text is fed into the model, synthesized images are produced. The model yields good results if trained with the appropriate algorithm for the dataset. This allows you experiment with different algorithms and datasets to then see what kind of interesting results it can produce. Generative Adversarial Text-to-Image Synthesis ultimately shows you the power of GANs, as it can be used to produce brand-new, realistic images from text.

OBJECTIVES

We have taken an already existing model to try to improve (build upon) it and apply it to a different dataset. We wanted the improvements to be visual, so you would be able to tell if the model performed better just by looking at the results. We also wanted to make the model train on new data that no one has previously used. Essentially, by applying different algorithms to this model, we would be able to see if the model performed better or worse.

Additionally, we wanted to see how well this model performs when compared to other models. As well as, how this particular model performs when different databases and text captions are used.

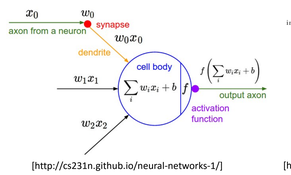
The image trained on Reed's G and T networks.

The image trained on my 200 iterations of G and Reed's T networks.

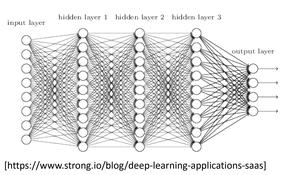
Both images were captioned "a group of people on skis stand in the snow."

THEORY

The core of this model is a neural network called GAN (Generative Adversarial Network). Neural networks are structures (usually purely implemented in code) that mimic the brain. First, data needs to be encoded in a way that the neural network can successfully use it. Once that input is provided to the neural network, then it goes through an activation function which decides whether this input should be passed on. Each neuron/node in a neural network is connected with other neurons in other layers. The connections are called synapses and each of the synapses have a "weight" value associated with them, which is scalar value that modifies how much a synapse affects the neural network. The error in the performance of the model tells you how to modify your weights. The beauty of neural networks comes from the fact that they can modify their weights by themselves, using the equations you implemented in their code. The goal of neural networks is to minimize error, by doing so, you're at the same time optimizing (adjusting) the network to have the best possible weights.

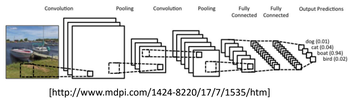


[http://cs231n.github.io/neural-networks-1/]

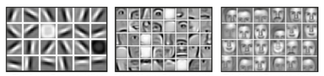


[https://www.strong.io/blog/deep-learning-applications-saas]

The more layers there are in the neural network the more complex it is, and that's why neural networks are often called "deep." With more complexity of the neural network and more data, come better results. A CNN (Convolutional Neural Network) is a very popular (usually deep) neural network. Every CNN works on three basic principles. It first builds up convolutions (in the convolutional layer), which are just mathematical descriptions of the particular features that the layer is trying to recognize (e.g., lines, circles, crosses). Since every image is made up of pixels, you can define a "window" of a certain number of pixels (e.g., four) which will be your filter (also called neuron or kernel) that will compare the pixels it sees in that spot of the image to your filter so that it can tell what kind of shape appears in that spot – each little filtered spot is called a convolution. The second step is to take each of these newly formed convolutions and to combine them together to create a more general representation of the convolutions which are close together – this is called pooling and it's done in the pooling layer. A more complicated CNN repeats this process of creating convolutions and then pooling them together at least a couple of times; this is to create abstract representations of the shapes (e.g., lines then curves then mouths and eyes then faces). Then finally, the last layer(s) will represent the final pooled convolutions as an N-dimensional vector. This vector represents the probability of the image corresponding to a particular class (e.g., cat or dog). Keep in mind that a deconvolution is in essence, a CNN but in reverse.

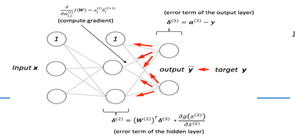


[http://www.mdpi.com/1424-8220/17/7/1535/htm]

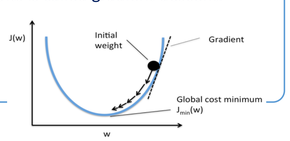


[http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf]

CNNs, like many other neural networks, use backpropagation when training. It essentially calculates the gradient of the loss function (corresponding to the error) considering the values of the weights in the network, so that the network can go back and modify these weights to then minimize the error. The process of training the network to minimize the error rate while modifying its weights is called gradient descent.



[https://sebastianraschka.com/faq/docs/visual-backpropagation.html]



[https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html]

MODEL

The model we used is the Generative Adversarial Text-to-Image Synthesis (Reed et al., 2016) – we directly used Scott Reed's Torch implementation available on GitHub. GANs consist of two networks, a generator G (which generates the actual images), and a discriminator D (which tries to tell if the generated image is a fake. The goal of the generator is to create a realistic image so that the discriminator could be fooled into thinking it's a real (non-synthesized) image. The two networks compete in a two-player minimax game; i.e., they try to find decision rules that would give minimum possible loss for worst case (maximum loss) scenarios:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

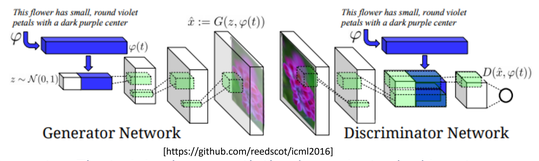
[http://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html]

We also want to obtain (deep) symmetric structured joint embedding, which will tell us how to represent the image. To do this, we must look at the text descriptions of the images and then extract vectors that could represent and discriminate between each of these images – we use a deep convolutional and recurrent (interconnected) text encoders. The text classifier optimizes the structures loss:

$$\frac{1}{N}\sum_{n=1}^{N} \Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n))$$

In short, the text encoding will form higher affinity to images that have a similar class, compared to images that are very different (dissimilar class).

The main approach to this problem is to train a DC-GAN conditioned on the learned text features and encoded by a hybrid character-level CRNN (Convolutional Recurrent Neural Network) to then have the generator and discriminator perform feed-forward inference on the text features.



[https://github.com/reedscot/icml2016]

Generator: The training begins with the discriminator looking at very poorly generated images (made up largely of noise), but as the training continues, the generator becomes better at generating more realistic images until the discriminator thinks they are real. The description embedding is compressed and concatenated to the noise vector to provide a better result. The generator then produces inference in the deconvolution. The synthetic (generated) image $\hat{x}$ is defined as $\hat{x} := G(z, \varphi(t))$, meaning it's generated by using the noise and the data from $\varphi(t)$ -- which is image encoding coming from the DCNN. The generator also learns to tell (score) wrong class labels (mismatched text) of the images

Discriminator: The D network will process the images using the convolutions (which I described earlier) from the images. Particularly, it uses layers of stride-2 convolutions and leaky ReLU for the rectification/activation. Subsequently, the image will follow the process of pooling and reducing the dimensionality in the fully connected layer. In a nutshell, we repeat the process of replicating the description embedding and concatenation, to then carry out the convolutions and rectification and then calculate the final score for the discriminator.

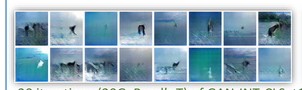
We have built several computers for deep learning so that we could train and test our models. Besides having 20-thread CPUs, the computers have GTX1080Ti's so set them up to work with the GPU versions of TensorFlow and Torch. This model in particular, uses Torch and Lua. It takes about three days to fully train the model on 200 iterations. The batch size significantly affects the speed of the training but we have used around 1500 images on average, per iteration.

We decided to use the COCO dataset because there it contains more than 82,000 images and 5 (text) descriptions per image. The other two common datasets that we have tried are the Oxford-102 Flowers and the CUB datasets.

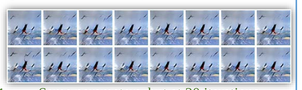
RESULTS

We have tried using several algorithms in order to test different models: GAN-CLS, GAN-INT, and GAN-INT-CLS. It seems that GAN-INT-CLS does the best job overall when generating from varied databases. GAN-CLS has matching-aware discrimination, meaning that the discriminator can signal the generator, to help it learn. GAN-INT uses interpolation between the embeddings of the training set captions. GAN-INT-CLS combines the GAN-CLS and GAN-INT algorithms.

GAN-INT-CLS was not originally implemented for the COCO dataset in the GitHub code. We have made the GAN-INT-CLS model work by creating classes (in different corresponding folders) for the different images and modifying some code. However, after a long training time, GAN-INT-CLS ran into the problem of looping the output images. Perhaps the problem was that the discriminator learns to trick the generator by not using a lot of noise in its input (which is something it should not do).



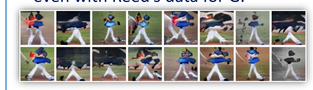
20 iterations (20G, Reed's T) of GAN-INT-CLS, t1



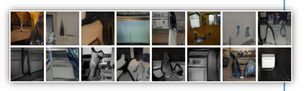
Same parameters, but at 30 iterations.

Both images had the caption: "a man in a wet suit riding a surfboard on a wave."

GAN-CLS performed quite well when I used my training data for the G, and Scott Reed's training data for T. It performed worse when I used my own T -- even with Reed's data for G.



GAN-CLS 200G iterations and Reed's T.



GAN-CLS, 200G and 190T iterations.

Both images had the caption: "a pitcher is about to throw the ball to the batter."

CONCLUSIONS

To summarize, we have found this model to perform well with the original data and text, and not as much with other/new data and text. Furthermore, we have seen that using many descriptions for the images produces better results. Also, that the GAN-CLS algorithm performs better than the GAN-INT-CLS algorithm with the COCO dataset. Perhaps this problem can be fixed by simply changing some parameters, or by further modification of the GAN-INT-CLS implementation. Moreover, we found that that training much longer produced better results.

Additionally, this model takes a long time to run -- even on a powerful computer -- we were actually running four different instances of the model (with different parameters, algorithms, etc.). This is because the model is not optimized to use the entire GPU. We have seen it use only about 3.4GB of VRAM (out of 11GB). It must be possible to decrease the training time for the model if we allow it to use all of the VRAM on the GPU.

REFERENCES

- **Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016, June 05). Generative Adversarial Text to Image Synthesis. Retrieved July 22, 2017, from https://arxiv.org/abs/1605.05396**
- **R. (2016, October 30). Reedscot/icml2016. Retrieved July 24, 2017, from https://github.com/reedscot/icml2016**
- Reed, S., Akata, Z., Schiele, B., & Lee, H. (2016, May 17). Learning Deep Representations of Fine-grained Visual Descriptions. Retrieved July 22, 2017, from https://arxiv.org/abs/1605.05395
- Deshpande, A. (n.d.). A Beginners Guide To Understanding Convolutional Neural Networks. Retrieved July 22, 2017, from https://adeshpande3.github.io/adeshpande3.github.io/A-Beginners-Guide-To-Understanding-Convolutional-Neural-Networks/
- Convolutional neural network. (2017, July 22). Retrieved July 22, 2017, from https://en.wikipedia.org/wiki/Convolutional_neural_network
- Gradient descent. (2017, July 04). Retrieved July 22, 2017, from https://en.wikipedia.org/wiki/Gradient_descent
- A friendly introduction to Convolutional Neural Networks and Image Recognition. (2017, March 20). Retrieved July 22, 2017, from https://www.youtube.com/watch?v=2-Ol7ZB0MmU
- Minimax. (2017, July 13). Retrieved July 22, 2017, from https://en.wikipedia.org/wiki/Minimax

Acknowledgements

This work was supported by NSF grant CCF-1421734. Also, I thank Scott Reed (et al.) for their work on their paper and Reed's implementation of the model.