CPSC 540 Project Report: Generative Adversarial Text to Image Synthesis

Raunak Kumar (47037130)

Devon Graham (23142094)

Angad Kalra (39817127)

Abstract

There have been numerous developments in generative models and encoders in recent years. In particular, Reed et al. [1] show how to generate images from text descriptions. In this project we explore their novel architecture and provide an implementation using Python and TensorFlow. We test our model by attempting to generate plausible images of flowers from captions. Unfortunately, the results are not impressive.

1 Introduction

In the last few years, there has been a lot of interest in generative models. Generative adversarial networks (GANs) [7] are one of the most influential recent developments [4]. Deep convolutional generative adversarial networks (DCGANs) have been shown to generate realistic images of faces and animals. Additionally, there have been a lot of efforts to use recurrent neural networks (RNNs) to learn discriminative text encodings [5]. In this project, we explore the use of DCGANs and deep structured joint embedding to generate images from captions [1]. We believe this can be used as a tool for criminal sketching, and to create quick protoypes of architectural and engineering designs. It may also serve as an insipirational tool for artists. We present a simple implementation of this technique using Python and TensorFlow. We provide experimental results by generating images of flowers from captions describing such images.

2 Related Work

Reed et al. [1] have developed an effective GAN architecture that can generate realistic images from captions. The authors view this as two subproblems. The first is learning a text representation that captures the important visual details of images. The second is using these features to generate a realistic image. They make use of recent developments in the field to solve both subproblems. They use a character level text encoder and a class conditional GAN. They provide encouraging experimental results primarily on a flowers and birds dataset, but on other diverse datasets as well.

Learning text feature representations is a well-studied problem. Reed et al. [6] describe three architectures to do this. The simplest approach is a character-level convolutional neural network (CNN), based on [10]. The authors use a standard image classification CNN and view the text as a one dimensional image with the number of channels equal to the alphabet size. They found that this simple model performs quite well on large datasets. One major drawback of the CNN network is that it does not capture strong temporal dependency that text exhibits. Their second approach attempts to fix this by using a convolutional recurrent network (CNN-RNN). In this architecture, the output

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

of the CNN is fed into a recurrent network that is capable of capturing temporal dependencies. The third approach [5] is to simply use a long short term memory (LSTM) model [9]. This model is a special case of an RNN and explicitly takes the temporal nature of text into consideration.

Vinyals et al. [2] and Xu et al. [3] tackle the problem of generating natural sentences describing an image. The authors of both papers use a very similar approach. They use a pre-trained deep CNN to encode the input image, and feed it to a decoder RNN which generates the sentence. The inspiration for this method comes from the recent success in machine translation, where the goal is to translate a given sentence in some language to another target language.

Much of recent work on generative models has taken advantage of deep convolutional decoder networks to generate realistic images. For example, Dosovitskiy et al. [6] have generated 3D images of chairs from information on lighting, position and shape. GANs [7] have also taken advantage of deconvolutional networks. This approach conditions on the class that the input image belongs to. However, Reed et al. [1] condition on the text descriptions instead. Radford et al. [8] incorporated batch normalization to create a stable and highly efficient architecture.

3 Our Contribution

We provide an implementation of the deep convolutional GAN architecture introduced by Reed et al. [1] using Python and TensorFlow. This implementation incorporates deep structured joint embedding [5] to obtain a visually discriminative vector representation of text descriptions. The code repository is available at this address: `https://github.com/raunakkmr/CPSC540-Project`. We now describe the two main components of the architecture.

3.1 Learning a Text Representation

Reed at al. use deep convolutional image and recurrent text encoders that learn a compatibility function with the images [5]. For example, they use deep CNNs and LSTMs. Intuitively, a text encoding for an image should have higher correlation with encodings for images of the same class than with images of other classes. They use the following loss function

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

where $\{(v_n, t_n, y_n)\}_{n=1}^{N}$ is the training dataset, (v_n, t_n, y_n) are the image, text description and class label of example n, and Δ is the 0-1 loss. The image classifier f_v and text classifier f_t are formulated as

$$f_v(v) = \arg\max_{y\in\mathcal{Y}} \mathbb{E}_{t\sim T(y)}[\phi(v)^T\varphi(t)],$$
$$f_t(t) = \arg\max_{y\in\mathcal{Y}} \mathbb{E}_{v\sim V(y)}[\phi(v)^T\varphi(t)],$$

where $\phi(v)$ is the image encoding and $\varphi(t)$ is the text encoding.

3.2 Generating Images Conditioned on Text

As described in [1], the generator first reduces the dimensionality of the text encoding $\varphi(t)$ and concatenates the result to a noise vector z. This vector is then forwarded through a deconvolutional network to generate an image. The discriminator is given a pair of an image and a caption. It performs several layers of stride 2 convolutions on the image with spatial batch normalization and leaky ReLU. It reduces the dimension of the encoded caption with a separate fully-connected layer. It then concatenates the result to the convolutional layer when the spatial dimension is 4x4. The final probability is calculated after 2 more convolutional layers. The following figure from [1] provides a summary:

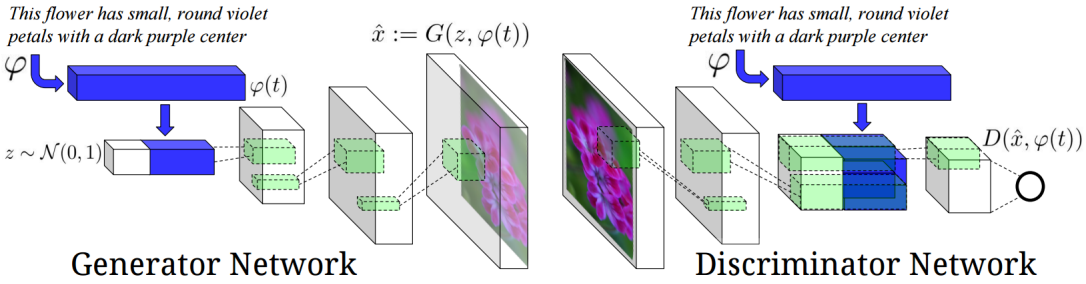

Figure 1: Text-conditional convolutional GAN architecture [1]

There are two sources of error in the GAN: unrealistic images for any caption, and realistic images with mismatched captions. In order to separate these sources of errors, the authors train the model using three different types of input to the discriminator. They naturally pass a pair of a real image with the corresponding caption, and a generated image for the input caption. Additionally, they also pass a pair of a real image with mismatching text to solve the above issue. Refer to Algorithm 1 for more details [1].

4 Results

Our results are not impressive. We believe that the individual components of our model are working properly on their own. Unfortunately, they are not working well together. The CNN for image encoding achieved 45% class prediction accuracy on unseen data. This may sound low, but there are more than 100 classes, and we suspect an average human would have trouble achieving such accuracy.

The text-encoding CNN also achieved roughly 45% class prediction accuracy on unseen data. Again, we consider this rather good, as many of the descriptions are quite similar. For example, "this flower is yellow in color, with petals that are small and oval shaped." and "this flower has petals of bright yellow and are rounded shape." represent different classes.

The GAN model also appears to be correct. We followed the approach of [1] closely. The network produced reasonable results when we provided it with noise instead of encoded text. However, we encountered a major bug when we tried to use the text and image encodings in the GAN. For certain encodings, the network produced `NaN`s in some hidden layers, resulting in meaningless loss and output values. We were able to track down the source of the bug to certain encoded text descriptions. All encodings seem to be plausible floating point numbers. Unfortunately, in the time that we had, we were not able to figure out why some encodings worked and others did not. Two examples are provided for curiosity's sake.

```
0.808476 2.13374 1.65548 -0.719038 0.368619 0.330139 1.98211 -0.135265 1.18786 0.660079 -0.836091 2.05936
2.51961 3.37974 0.575895 -1.69023 -0.902822 -1.8514 -2.76209 2.59817 2.36879 0.920938 0.689171 1.61115
-3.52344 4.61089 3.05417 -2.52265 -2.54791 -0.758915 2.33576 1.06609 -1.58851 0.670371 -1.34238 2.54683
```

Figure 2: Good encoding

```
1.73171 0.614854 4.38955 -1.27327 -3.26651 -1.80153 2.8009 0.257215 1.62256 -0.511757 -0.289265 0.795884
2.21384 4.43724 -0.165279 -1.67889 0.00602393 -3.11433 -5.27336 1.4589 1.3345 0.465175 0.565701 2.49551
-1.87754 5.75464 4.01745 -0.27205 -4.79805 -0.593607 3.29024 3.11546 -2.21386 0.612689 -2.7408 1.70719
```

Figure 3: Bad encoding

We were able to run the network on a tiny sample of the data which contained no corrupted encodings. We provide an example output trained on only 16 images. We were encouraged to see pink and yellow feature prominently in it, as these are key features of the input caption. However, anything more positive cannot be said.

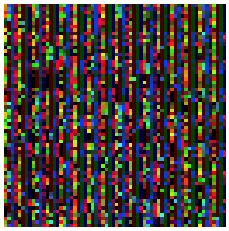

Figure 4: "the petals of the flower are pink in color and have a yellow center"

5 Discussion

We underestimated the importance of data pre-processing. It is crucial to load data in a form that is suitable for all components of the architecture. This speeds up debugging and testing time. We did not find TensorFlow to be as user friendly as we expected it to be. It is important to understand small details of the computation graph to be able to implement an architecture smoothly. We found it really painful to print out debugging values without going through a lot of boilerplate code. In summary, this project was hard to implement. But we gained a lot of exposure and knowledge about text encodings and GANs. We hope to continue working on this project in the future to get it working.

We have attempted to provide a Python implementation of deep structured joint embedding. We believe this can be a good starting point for other people looking to implement the same. In addition, they can explore what we did, and avoid our mistakes. Obviously, the main weakness of our project is that it doesn't work. We could only train our model in one of two ways. Either on a small dataset of 16 images with caption encodings, or on a large dataset of around 8000 images with random noise in place of caption encodings. Hence, the generated images are not realistic at all. In addition, the interface of our implementation is hard to use due to our lack of familiarity with TensorFlow.

With more time, our main objective would be to understand TensorFlow better so that we could implement the architecture with good software engineering principles. This would allow us to debug and make changes to the models in a much more efficient manner in terms of programming time and computational resources. In particular, it would allow us to fix our main `NaN` bug, which prevented us from progressing further. Once we get our implementation working, we would compare the model's generated images on the same caption encoded from 2 different languages. We would also train it on diverse datasets to observe its performance on captions outside the realm of flowers.

References

[1] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele and Honglak Lee.
Generative Adversarial Text to Image Synthesis, 2016;
arXiv:1605.05396.

[2] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan.
Show and Tell: A Neural Image Caption Generator, 2015;
arXiv:1411.4555.

[3] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, Yoshua Bengio.
Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, 2015;
arXiv:1502:03044.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio.
Generative Adversarial Networks, 2014;
arXiv:1406.2661.

[5] Scott Reed, Zeynep Akata, Bernt Schiele and Honglak Lee.
Learning Deep Representations of Fine-grained Visual Descriptions, 2016;

arXiv:1605.05395.

[6] Dosovitskiy, A., Tobias Springenberg, J., and Brox, T.
Learning to generate chairs with convolutional neural networks. In CVPR, 2015

[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In NIPS, 2014.

[8] Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.

[9] Hochreiter, S., Schmidhuber, J. Long Short-Term Memory. 1997.

[10] Zhang, X., Zhao, J., and LeCun, Y. Character-level Convolutional Networks for Text Classification, 2015. arXiv:1509.01626.