IMAGAN: Learning Images from Captions

Trevor Tsue
Samir Sen
Karan Singhal
{ttsue, samirsen, ksinghal}@stanford.edu

Abstract

We construct a generative adversarial network (GAN) architecture that takes as input a text caption and outputs a generated image described by the caption. We use the Oxford-102 Flowers Dataset with captions and images to train our model. As a baseline, we encoded captions using skipthought vectors and created images using a conditional deep convolutional GAN (DCGAN) with conditional loss sensitivity (CLS). From there, we fully connected the text model using a bi-directional LSTM trainable from the loss from the GAN. Furthermore, we implemented various tweaks for our GAN architecture, drawing from the current state-of-the-art in training techniques for GANs. Additionally, we implement and explore the use of the novel Boundary Equilibrium GAN (BEGAN) model, which balances the generator and discriminator loss to vastly improve visual quality, as a conditional model to generate images from caption embeddings.

Github repo: https://github.com/samirsen/image-generator

CS224N Mentor: Richard Socher

1 Introduction

Much work has been done to examine the syntax and semantics of textual data. One way to measure textual understanding might be to observe transfer of textual learning into the visual domain. Specifically, automatic image generation from captions is a challenging current area of research, at the forefront of efforts in both NLP and vision work. Beyond demonstrating an impressive understanding of both natural language and vision for theoretical interest, high performance on this task would have a number of practical applications. One example would be in image search: a user can type a complicated caption phrase, a model for this task can generate an image matching this caption, and a reverse image search based on the content of the generated image can be used to find relevant existing results. On a subtler note, progress in this domain may aid in bridging the gap between tasks which can be automated, and those which are open to interpretation.

In this joint project for CS224N and CS230, we build generative adversarial network (GAN) models for generating images of flowers using captions from the Oxford flowers caption dataset. Specifically, we focus our efforts on improving model performance and overall quality of generated flowers by experimenting with various GAN architectures and language models. As a baseline, we attempt the image generation task by passing pretrained 4800-dimensional 'skipthoughts' sentence embeddings, concatenated with random gaussian noise, to a vanilla 4-layer generator and discriminator network. We find that the images produced by more complex architectures, including deep convolutional GANs and boundary equilibrium GANs, resulted in a higher overall image quality. Furthermore, we find that an end-to-end model inclusive of training caption embedding through a bi-directional LSTM before being passed to the GAN architecture resulted in images that were both higher quality and resulted in trained caption embeddings that corresponded well with the categories of flowers in the dataset.

2 Previous Work

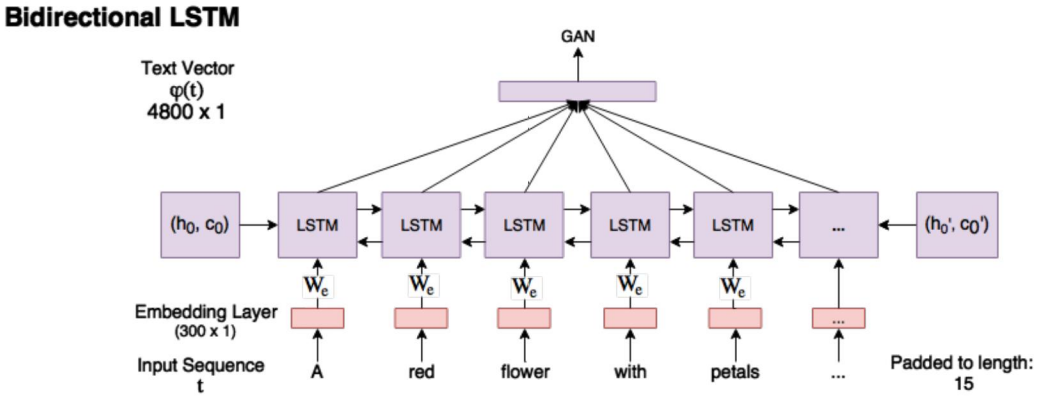
Previous work has been done on this task. In the space of multimodal representations, Srivastava & Salakhutdinov (2012) used a deep Boltzmann machine to jointly model images and text data. [16] For image synthesis, before GANs, variational autoencoders and autoregressive models (e.g. PixelRNNs) generated promising results. Mansimov et al. 2016 built a model to generate images from text using attention to align text features with a canvas. [17] Most recent work on this task has generally used some variety of GAN. GANs are promising for this task because they can achieve generation of sharper images and they have been naturally extended to conditional GANs, though their training is unstable and reliable quantitative evaluation is difficult. Reed et al. (2016) applied GANs to image synthesis conditioned on captions, using an architecture that we aimed to emulate and improve upon for our baseline model. [2] More recent work has built off of this architecture, usually by building more sophisticated and multi-layered GAN models. Some examples include StackGAN [7], which uses a second GAN to refine and increase the resolution of images, and AttnGAN [8], which uses attentional generative adversarial networks to do multi-stage refinement of images.

3 Approach

At a high-level, our approach involves taking textual vector representations of captions and then using a conditional GAN to generate output conditioned on representations. This general approach, with modifications, is also employed by most recent papers on this task. Our baseline model took inspiration from the architectures proposed by Reed et al.(2016) and Neekhara (2016). We built on top of this by borrowing other ideas from very recent research on deep convolutional and balanced equilibrium GANs implementing these models from scratch in pyTorch. In contrast to recent efforts, we also implement our own LSTM architecture for learning caption representations that correspond with the various categories of flowers in the Oxford Flowers dataset.

Figure 1: Highest quality images were attained using BEGAN model with Bi-LSTM trained embeddings

3.1 Model Architecture



3.1.1 Caption Embeddings and LSTM

To begin, we vectorized the text captions that using the Skip-Thought model, which is a skip-gram-based model combined with a gated recurrent unit that encodes a sentence to predict the sentences

around it, implemented in Kiros et al 2015 [1]. Thus, sentences with similar semantic and syntactic have similar vector representations. Skip-thoughts derives its name from similar training methodology to the popular skip-grams language model. As opposed to using context and center words, skip-thoughts is able to learn sentence representation from predicting center sentences from context sentences. This model has been trained on a corpus of 11,803 books. Using this pretrained model, we vectorize each caption into a 4800-feature vector as a baseline.
Using the pretrained skip-thought vectors, we seemed bottlenecked in our ability to produce very high quality flower images as caption embeddings were not specific to the task of generating flowers. We also believe that these sentence embeddings may not capture representational similarity very well as they are trained on a relatively small dataset in comparison to GloVe word embeddings. Some examples of the baseline images produced are shown above.

Figure 2: Images generated by model with skip-thoughts embeddings are noisy

These images were quite noisy and we were curious to see if trained custom embeddings geared to the task of flower generation would result in better association with the flower we expect to generate. In other words, we wanted to be able to learn embedding weights from an LSTM language model such that encoded embeddings would be distinguishable with respect to the category of flower (i.e. red flowers, purple flowers, flowers with petals, etc.) . We started experimentation creating custom sentence embeddings from GloVe vectors trained on 6 billion words instead of the skipthought sentence vectors trained on a much smaller dataset.

Specifically, we first tried concatenating the sequence GloVe word vectors to form an embedding for each caption. Because GloVe vectors are 300 dimensional, this meant padding each sentence to a max length of 16. We found that these embeddings were not as optimal as expected fairing worse or equal to the baseline model. We suspect that this is because the skipthoughts sentence embeddings are capture semantic similarity in captions better than blind concatenation of independent GloVe embeddings.

Next we tried using the GloVe word embeddings to input the sequence of words from a batch of captions to an LSTM layer. In particular, we model the task of learning caption representations by optimizing the weights according the GAN generator loss in a single-layer Bi-directional LSTM. Consider the input sequence, x, forward recurrent weights $W_h^{\rightarrow}$, backward recurrent weights $W_h^{\leftarrow}$, and embedding weights W_e. Then to encode a caption,

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$

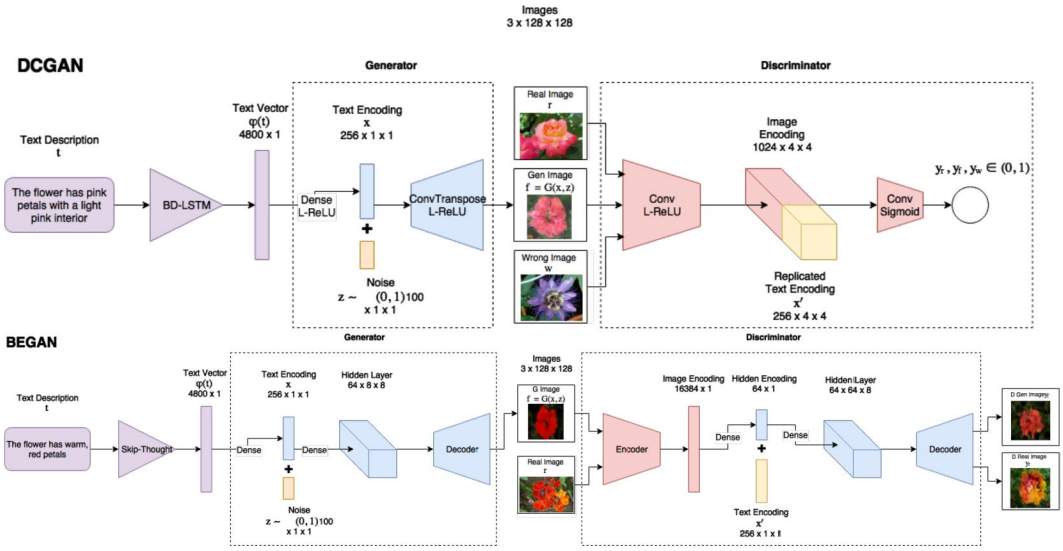
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$

$$c_t = f_t c_{t-1} + i_t \text{tanh}(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o)$$

$$h_t^* = o_t \text{tanh}(c_t)$$

Then, we concatenate the right and left hidden states calculated to get $h^{(t)} = [h_t^{\rightarrow}; h_t^{\leftarrow}]$ where each hidden state is 2400-dimensional so the output from the forward pass through the Bi-directional LSTM is 4800-dimensional. We train the network weights using Adam optimizer using the loss computed from the image generation stage of the GAN.



3.1.2 Input for Generator

Prior to inputting our embedded text into the generator, we pass the embedded vectors into a fully-connected layer to reduce the dimensions to 256-features followed by a leaky ReLU activation function. We then sample a 100-feature noise vector z from the normal distribution with mean 0 and variance 1, and concatenate the embedded text vector and the noise vector, passing this resulting vector into the generator.

3.1.3 Generator

The DCGAN implementation contains six layers of the convolution transpose functions, decreasing the number of channels while increasing the dimensions of the image. After each layer, we apply a leaky ReLU function, except for the last layer, in which we apply a tanh function. We originally used a ReLU function, but we updated it to a leaky ReLU to improve our generator performance over time, as the discriminator was winning against the generator. The result is a 128x128x3 image that is outputted by the generator.
The Conditional BEGAN model, we added an additional hidden layer before we pass it into the generator part. An important part of the BEGAN model is that the generator has the same architecture as the decoder of the discriminator [3]. With this in mind, we have defined an upsample block with two convolution layers with ELU activations and a nearest neighbors upsample. The generator and decoder of the discriminator has 4 upsample blocks followed by two convolution layers with ELU and a final convolution layer with tanh, giving us a 128 x 128 x 3 image.

3.1.4 Discriminator

The DCGAN discriminator is a 4 layer convolution network that first takes an 128x128x3 image, decreasing the size of the image as we go deeper and increasing the number of channels. After each layer, we apply a leaky ReLU activation function. We also pass the given text embedding through a fully connected layer and concatenate the output of the convolution layers with the embedded text vector. We pass this concatenated vector through two more fully connected layers, and pass the result into a sigmoid function. To improve the GAN, we decided to add smooth-labelling, making the training output 0.9 instead of 1 when the right image and caption are added [2]
The BEGAN model has a convolution block defined as two convolutions with ELU followed by a convolution and an average pooling. We pass our input through four of these convolution blocks and several other convolution layers with ELU to created an image encoding. We then use a fully connected layer to make this a hidden embedding, which we concatenate with the text encoding (hence conditional). We have another fully connected layer to make a hidden layer and pass it into the decoder. This decoder has the same structure as the generator, creating a 128 x 128 x 3 image reconstruction of the original image.

3.2 Objective

3.2.1 DCGAN

The discriminator's goal is to output a 1 (0.9 with label smoothing) when the input is a real image and the right caption. Otherwise, when there is a wrong image to match the caption or there is a fake image with the caption, the discriminator wants to output 0. With the following sigmoid outputs from the discriminator, $y_r \leftarrow D(r, x')$ {Real image, right caption}, $y_w \leftarrow D(w, x')$ {wrong image, right caption}, and $y_f \leftarrow D(f, x')$ {Fake image, right caption}. We apply binary cross entropy to the following loss function for our discriminator:

$$\mathcal{L}_D \leftarrow \log(y_r) + \log(1 - y_w) + \log(1 - y_f)$$

Thus, our discriminator aims to maximize the (real image, right caption) pairing while minimizing (wrong image, right caption) and (fake image, right caption) pairs. Our generator, on the other hand, has the loss function:

$$\mathcal{L}_G \leftarrow \log(y_f)$$

Thus, the generator wants to maximize the (fake image, right caption) output from the discriminator. Our training procedure is summarized in the alogrithm below.

Algorithm 1: Training of CLS DCGAN

Input: real images r, wrong images w, matching captions t, batch size S, generator G, discriminator D, text encoder ϕ, learning rate α

for *i = 1, . . . , S* **do**

- $x \leftarrow \phi(t)$ {Encode G caption}
- $z \sim \mathcal{N}(0, 1)$ {Sample random noise}
- $f \leftarrow G(x, z)$ {Generate image}
- $x' \leftarrow \phi(t)$ {Encode D caption}
- $y_r \leftarrow D(r, x')$ {Real image, D caption}
- $y_w \leftarrow D(w, x')$ {Wrong image, D caption}
- $y_f \leftarrow D(f, x')$ {Fake image, D caption}
- $\mathcal{L}_D^{(i)} \leftarrow \log(y_r) + \log(1 - y_w) + \log(1 - y_f)$ {Discriminator loss}
- $\mathcal{L}_G^{(i)} \leftarrow \log(y_f)$ {Generator loss}

$\mathcal{J}_D \leftarrow \frac{1}{S} \sum_{i=1}^{S} -\mathcal{L}_D^{(i)}$ {Discriminator cost}

$\mathcal{J}_G \leftarrow \frac{1}{S} \sum_{i=1}^{S} -\mathcal{L}_G^{(i)}$ {Generator cost}

$D \leftarrow D - \alpha \frac{\partial \mathcal{J}_D}{\partial D}$ {Update discriminator}

$G \leftarrow G - \alpha \frac{\partial \mathcal{J}_G}{\partial G}$ {Update generator}

3.2.2 BEGAN

The goal of the BEGAN model is to balance the loss of the generator and discriminator [3]. With a loss function as the L_1 norm of the matrix, we have the loss function:

$$\mathcal{L}_D \leftarrow \mathcal{L}(y_r, r) - k\mathcal{L}(y_f, f)$$

Our generator has the loss function:

$$\mathcal{L}_G \leftarrow \mathcal{L}(y_f, f)$$

In this model, the discriminator's goal is to recreate the real input image after encoding the image as accurately, while recreating the fake image as poorly as possible. By keeping a balance of $\gamma = \frac{\mathbb{E}[\mathcal{L}(y_f, f)]}{\mathbb{E}[\mathcal{L}(y_r, r)]}$ with the diversity ratio γ. A low γ make the discriminator focus on the recreating real images, making the generator focus on producing more realistic images. While a high γ makes the discriminator focus on discriminating, making the generator produce more diverse images. The previous equation is equivalent to $\gamma\mathbb{E}[\mathcal{L}(y_r, r)] - \mathbb{E}[\mathcal{L}(y_f, f)] = 0$. We choose to have $k \leftarrow k + \lambda(\gamma\mathbb{E}[\mathcal{L}(y_r, r)] - \mathbb{E}[\mathcal{L}(y_f, f)])$, with λ as the learning rate that adapts k over time. By

using our k value in the loss $\mathcal{L}_D \leftarrow \mathcal{L}(y_r, r) - k\mathcal{L}(y_f, f)$, we balance the losses over time.

Algorithm 2: Training of Conditional BEGAN

Input: real images r, matching captions t, batch size S, generator G, discriminator D, text encoder ϕ, learning rate α, loss balancing term k, diversity ratio γ, proportional gain λ (learning rate for k)

for *i = 1, . . . , S* **do**

$x \leftarrow \phi(t)$ {Encode G caption}
$z \sim \mathcal{N}(0, 1)$ {Sample random noise}
$f \leftarrow G(x, z)$ {Generate image}
$x' \leftarrow \phi(t)$ {Encode D caption}
$y_r \leftarrow D(r, x')$ {D generated real image}
$y_f \leftarrow D(f, x')$ {D generated fake image}
$\mathcal{L}^{(i)}(y_r, r) = \|y_r - r\|_{1,1}$ {Real image loss}
$\mathcal{L}^{(i)}(y_f, f) = \|y_f - f\|_{1,1}$ {Fake image loss}

$\mathcal{L}(y_r, r) = \frac{1}{S}\sum_{i=1}^{S} \mathcal{L}^{(i)}(y_r, r)$ {Average real image loss}
$\mathcal{L}(y_f, f) = \frac{1}{S}\sum_{i=1}^{S} \mathcal{L}^{(i)}(y_f, f)$ {Average fake image loss}
$\mathcal{J}_D \leftarrow \mathcal{L}(y_r, r) - k\mathcal{L}(y_f, f)$ {Discriminator cost}
$\mathcal{J}_G \leftarrow \mathcal{L}(y_f, f)$ {Generator cost}
$D \leftarrow D - \alpha\frac{\partial \mathcal{J}_D}{\partial D}$ {Update discriminator}
$G \leftarrow G - \alpha\frac{\partial \mathcal{J}_G}{\partial G}$ {Update generator}
$k \leftarrow k + \lambda(\gamma\mathcal{L}(y_r, r) - \mathcal{L}(y_f, f))$ {Update k value}

4 Dataset

We used the Oxford-102 Flowers dataset collected from an online repository for this experiment. This data set has over 8,000 images of flowers along with nearly 10 captions for each image. On the current model, we trained and tested our model on a small subset of 128 images of this data set. We also used 300-dimensional GloVe vectors as frozen word embeddings for our bidirectional LSTM model, which were trained on 6 billion word tokens from Wikipedia 2014 + Gigaword 5. [13]

5 Experiments

We first ran our baseline model, which was a simple conditional GAN with 4 convolutional layers for both the generator and discriminator. We used CLS (conditional loss sensitivity), as described in Reed et al. 2016, which resulted in a modified loss function for the discriminator. We used the skipthoughts model to encode sentence vectors into a 4800-dimensional vector representation. We then projected this representation before concatenating it with a 100-dimensional Gaussian noise vector, as described above. We used a learning rate of 0.0001, a batch size of 128, and data augmentation through random flips. The model took a few hours to train, about 30 seconds per epoch. We show the loss curve for this training and validation in our supplementary files.

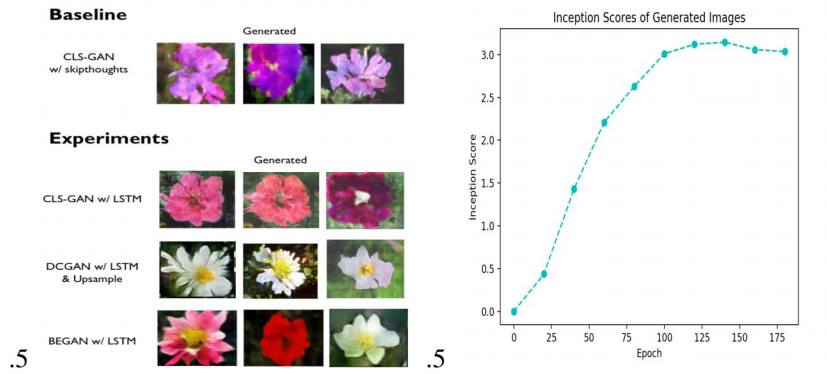


Figure 3: We report increasing inception scores over epochs, indicating generation of images that are closer to real.

5.1 GAN Architecture

We noticed that many of the resulting images exhibited a visible checkerboard artifact, which motivated us to look into this issue. We show an example of an image exhibiting this pattern in a figure. We found this checkerboard artifact in images produced by other models, including that by Reed et al., and found that the checkerboard pattern was a common issue with GANs. We found that one possible solution was using upsampling and convolution for our "deconvolutional" layers.

We modified our GAN architecture to use this paradigm, added more convolutional layers to the generator and discriminator. After we discovered the upsampling tweak, we also found and implemented a number of "GAN hacks", many of which we initially read about in Soumith Chintala's popular Github resource bearing the same name. [12] Some of the hacks that we implemented include one-side label smoothing, LeakyReLU with a leak of 0.2, using SGD instead of Adam for the discriminator and Adam for the generator, using Gaussian noise for the generator, using tanh as the output activation function for the generator, using BatchNorm, learning rate decay, avoiding the use of ReLU and MaxPool because they cause sparse gradients, and using ELU instead of ReLU. Because of our limited timeline and the time required to train our more complicated models (our final models took days to train), we were not able to individually test the effectiveness of each of these tweaks for our task, although we did learn that using SGD for the discriminator did not seem to work well.

The resulting model containing these tweaks as well as upsampling produced images that did not have the checkerboard problem, but the loss curve for training seemed to indicate some divergence of the generator after a number of epochs. Sampled images sometimes had backgrounds and color patterns that clearly revealed that they were synthesized. It is difficult to tell what the cause of the instability was, owing to the general instability of GANs and the number of small tweaks we made to the model architecture. The train and validation loss curves for this model are provided in the supplementary files.

After building our upsampling model, we moved on to the BEGAN model because we saw that it was the state-of-the-art in non-conditional GANs and it kept an intuitively appealing balance between the generator and the discriminator (something our other models struggled with). Our goal was to extend the model to be conditional, analogous to the way GANS were quickly extended to Conditional GANs. [5] We implemented this model in PyTorch and applied it to our text-to-image synthesis tasks. We had to reduce our batch size from 128 to 64, as the model operations would not otherwise fit on GPU memory. After about a day and a half of training, this model gave us better results, with more stable loss and training curves shown in supplementary files. For this model, we evaluate quantitatively using inception scores, which are a measure of image interpretability (from the perspective of an Inception v3 model) introduced by Salimans et al. 2016. [14] Our results are shown in a figure, but we caution against reading too much into inception scores, as they have been shown to be unreliable in that models can maximize inception scores by producing adversarial examples for the Inception model. [15]

5.2 Text Model Architecture

In parallel, we worked on improving our text caption encoding model from our skipthoughts baseline. The skipthoughts model was not trained for the task and dataset, so its encodings did not capture important insights related to the domain. For example, a learned sentence representation would put much more emphasis on the word "red" in producing a caption than the word "flower", which provides little additional information in a set of captions about flowers.

We used GloVe vectors to implement two models: a bag of words model that averages the word vectors in a caption and uses this as a sentence representation, and a 1-layer bidirectional LSTM model. Both of these models use frozen embedding weights, since we suspected our caption dataset was not large enough to successfully finetune these weights without aberrations. As we expected, the LSTM model performed much better, as it was able to capture positional information and used learned weights. Sample images from the LSTM model are shown in figures.

6 Conclusion and Future Work

There is much room to improve the model architecture. Because of the rapid monthly influx of GAN papers, new tweaks are being discovered regularly to improve the stability and output quality of GANs. Even a regular DCGAN gets a significant boost in the quality its images by applying simple "GAN hacks", such as one-sided label smoothing, feature matching, decaying learning rate, using SGD for the discriminator, etc.

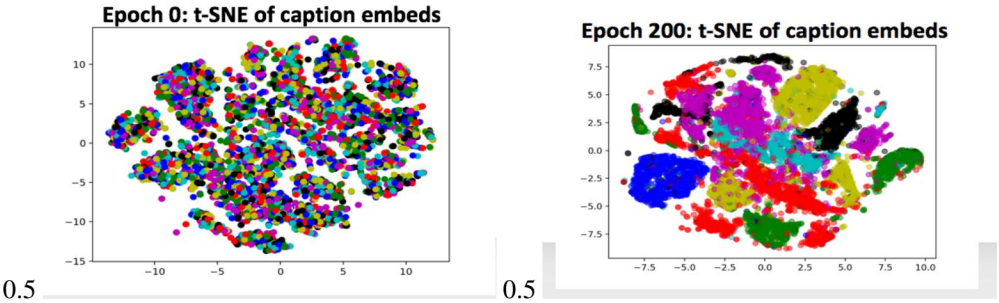


Figure 4: tSNE dimensionality reduction before and after training

In our work, we implemented most of these tweaks, but did not have time to experiment with others. Moreover, we were not able to experiment with each tweak individually and see the results for our task. Since experimenting with GANs remains a very experimental process, the time-consuming nature of training (especially for the BEGAN model, which took days to train on a high-end GPU) prevented us from tuning these tweaks, and we were left to guess which tweaks would improve the performance of our model and which would not. In the future, we would make these decisions empirically, perhaps by using multiple GPUs in parallel to speed things up, introducing the possibility of more rigorous hyperparameter tuning.

Many variations of GANs now exist as well, even for the narrow task of text-to-image synthesis, including StackGANs and AttnGAN. Though we were able to explore and extend BEGANs for our task, (and also played around with WGANS, although results were now shown in this paper) if we had more time we would explore other recently developed architectures to see how they perform on this task.

Given more time, we would also devote more time to finding better text representations and better visualizing the transfer from the textual domain to visual. We suspect that an attentional model would achieve both good performance and interpretability, so we would go in this direction first.

Eventually, we would want to extend our model to train on different datasets, such as MS COCO or even ImageNet, but this may not be feasible in the near future, as this remains a significant obstacle even for seasoned researchers working on this task, due to the instability of GAN training.

7 CS224N/230 Work Split

We built the baseline model from PyTorch (after learning how to use PyTorch) for both CS224N and 230. This includes code to load data, build our first generator and discriminator models, apply the skipthoughts model to captions, run training, and manage experiments. For CS224N specifically we built on top of this work by replacing the skipthoughts model (which was not trained for this task). We implemented both a bag of words model and an LSTM model that was trained with the rest of the model. We ran experiments and visualizations with these models, building different sentence representations by using different hidden layer sizes, summing/averaging hidden states, performing t-SNE dimensionality reduction, etc. We also implemented a number of "GAN hacks" or changes to the model architecture/training that helped with generating higher-quality images (an example is one-sided label smoothing). Our goal with these hacks was to apply learnings from state-of-the-art GAN models to this task. For CS230, we built new model architectures to replace our generator and discriminator models, including a deeper DCGAN with upsampling, WGAN (results not shown in this paper, code in repository), and conditional BEGAN. Our goal with these models was to significantly improve the quality of our samples, since these are some of the most performant GAN architectures developed so far outside of the text-to-image context. These later models have never been implemented for this task before, so this was a major learning experience.

References

[1] Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R. S., Torralba, A., Urtasun, R., and Fidler, S. (2015). Skip-Thought Vectors. *arXiv*, 1506.06726v.

[2] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran,Bernt Schiele, and Honglak Lee. 2016. Generative Adversarial Text to Image Synthesis. https://arxiv.org/pdf/1605.05396.pdf

[3] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved Techniques for Training GANs. https://arxiv.org/pdf/1606.03498.pdf

[4] David Berthelot, Thomas Schumm, and Luke Metz. 2017. BEGAN: Boundary Equilibrium Generative Adversarial Networks. https://arxiv.org/pdf/1703.10717.pdf

[5] Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. https://arxiv.org/abs/1411.1784

[6] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2016. Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval. https://arxiv.org/abs/1502.06922

[7] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. 2017. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. https://arxiv.org/pdf/1612.03242.pdf

[8] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. 2017. AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. https://arxiv.org/abs/1711.10485

[9] Neekhara, P. (2016). text-to-image. https://github.com/paarthneekhara/text-to-image/graphs/contributors

[10] E. Mansimov, E. Parisotto, L. J. Ba, and R. Salakhutdinov. Generating images from captions with attention. In ICLR, 2016.

[11] Andrej Karpathy and Li Fei-Fei. 2015. Deep Visual-Semantic Alignments for Generating Image Descriptions. https://cs.stanford.edu/people/karpathy/cvpr2015.pdf

[12] Chintala, Soumith. (2016). ganhacks. https://github.com/soumith/ganhacks

[13] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[14] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In Advances in Neural Information Processing Systems, pages 2234–2242, 2016.

[15] Shane Barratt and Rishi Sharma. A note on the inception score. arXiv preprint arXiv:1801.01973, 2018.

[16] Srivastava, Nitish and Ruslan Salakhutdinov. Multimodal Learning with Deep Boltzmann Machines. In Advances in Neural Information Processing Systems, 2012.

[17] Mansimov, E., Parisotto E., Ba J., Salakhutdinov R. Generating Images from Captions with Attention. arXiv:1511.02793