Images from words with conditional generative adversarial neural networks

Final Project in IEOR-8100

A project by

Austin Slakey, Roshni Rajesh Shah, Lars Patrik Roeller,
Maria Kardar, Wassym Kalouache, and Yumeng Jiang

Abstract
We extend the generative adversarial neural network (GAN) framework proposed by Goodfellow et al. [1] by adding conditional parameters to the input. Specifically, the goal is to create images based on text input in the form of individual words.
The generator in this conditional GAN framework receives a concatenation of noise and input parameter vector and tries to convince the discriminator network that, conditional on the input parameter vector, the created image belongs to the true data.
We test the model on the CIFAR-100 dataset, as well as on the MNIST dataset. The preliminary CIFAR-100 data results are poor, potentially due to underfitting of the data. The MNIST dataset results, on the other hand, offer promising results from the onset, so we focus our efforts here.

Introduction
Humans have a remarkable capability to learn new concepts through experiences by reasoning out a structure. They can then use that structure to generate new ideas. Taking inspiration from this, machine learning systems have attempted to build numerous algorithms to translate this generative ability. Of these, generative adversarial nets have been an important tool as they sidestep the difficulty of approximating many intractable probabilistic computations.

The framework of adversarial nets as a generative model and a discriminative model and can be understood from the following analogy: The generative model can be thought as a team of fake poets trying to generate fake Shakespearean poems and publish them, while the discriminative model will then be the team trying to detect these fake poems versus the real Shakespearean poems. Both teams compete to improve their methods till the counterfeits succeed in generating real poems. In this paper, we will follow a model where both the generator and discriminator use neural networks based on the model introduced in the paper by Goodfellow et al. [1]

Further ahead, an extension to the generative adversarial nets (GAN) is a conditional GAN which allows the model to condition on external information. This enables the generative model to learn in different modes based on the different contextual information provided. In this paper, conditional GAN is used on CIFAR-100 dataset and MNIST dataset, both with an aim to create an image output based on the given word input. Word to vector representation is, thus, also an important part of the paper where a log-bilinear model from Pennington et al. [4] is used.

Background and Methods

1) GAN Theory

The model that we use in this paper is an extension of the GAN model introduced by Goodfellow et al. [1]. The basic model uses a discriminator (D) and a generator (G) that are each built as neural networks. In the training phase, the generator receives random (noise) input and outputs data in the form of the real training data (images, sound, etc.). The discriminator then receives this data from either the real training data or from the generator output and is tasked with distinguishing between the two. So, we train the

discriminator to assign a high probability whenever the data is real training data and assign a low probability otherwise. On the other hand, we train the generator to maximize the probability that the discriminator cannot distinguish between real data and data created by the generator. Formally, this problem can be written as a min-max game.

We define z to be the random noise input for the generator, where $p_z(z)$ is the noise distribution, and x to be our real training data with distribution $p_{data}(x)$. Then $G(z)$ and $D(x)$ are the generative model and discriminative model outputs, respectively. With these definitions, our objective function for the entire (unconditional) model can be stated as follows:

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

We then transform this min-max game into two separate objective functions that we use to update the discriminator and the generator. The first of these functions maximizes the probability of the discriminator outputting the correct class (1 for real data, 0 otherwise). The second function minimizes the probability that the discriminator will output 0 for the generated data. The loss functions are as follows:

$$d_{loss} = -(\log(D(x)) + \log(1 - D(G(z))))$$
$$g_{loss} = -(\log(D(G(z))))$$

The model can then be trained using stochastic gradient descent, where the discriminator and generator models are updated sequentially. Goodfellow et al. [1] showed that the theoretical solution of this training method is $D(x) = \frac{1}{2}$ and $p_{data}(x) = p_g$ where p_g is the distribution of the generator output. Therefore, in the optimal solution, the discriminator is unable to distinguish the output data from the generator from the real training data. Furthermore, the distributions of training data and generator output data are identical.

After this training phase, the generative model can therefore be used to generate samples as if they were from the real training data. For example, using the MNIST dataset (images of digits 0 through 9) as a training set, the generator will produce images of the digits 0 through 9. However, these images are generated from scratch, i.e. they are not mere copies of the training images.

In this paper, we will extend this idea to allow for conditional inputs and outputs. For example, letting the MNIST dataset be our training data, we can use word vectors of each digit as our conditional input. This would then allow us to choose a specific word vector and receive an image of that digit from the generator model. Again, this model would generate a new image of that digit from scratch and not use one of the images that were supplied by the training data.

In order to use the conditional model, we define y as the conditional input to the generative model and $p_y(y)$ as the distribution of the conditional input. We now also have to redefine our discriminator and generator slightly, as one of their inputs is the conditional input y. Let $G(z, y)$ be the output of the generative model given inputs z (noise) and y (conditional input), and let $D(x, y)$ be the output of the

discriminator given inputs x (data) and y (conditional input). Our new objective function will be as follows:

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

2) Vector Representation of Words

One of the necessary preprocessing steps in order to run the conditional GAN is a vector embedding of the conditional input words. For example, if the conditional input (y) is the word "two", then we need some vector that uniquely identifies the word "two".
Furthermore, the model requires that words which are similar in meaning, such as "King" and "Queen" be assigned a similar word vector. This is necessary because our model is a generative model, and the only way the model is able to generate an image of a word it has never seen before is to look at the most similar words to it from which it knows the images.

Mikolov et al. [7] described a neural network model to determine the word embeddings in a vector space of arbitrary dimensions while maintaining word similarities. As described in his paper, the notion of vector similarity of similar words leads to a simple consequence: Vector addition and subtraction will produce logical results.
As an example, consider the words "King", "Man" and "Woman". We can represent the vector embeddings that maintain similarity in a two-dimensional graph, as shown in Figure 1 for illustration purposes.

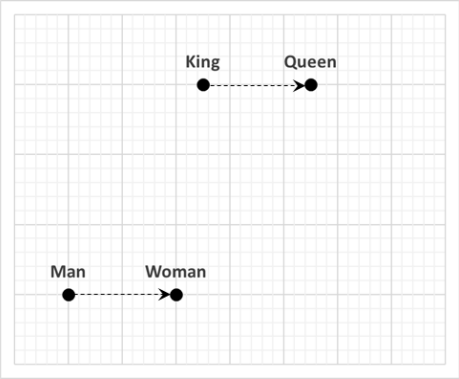

Figure 1: Similar words being grouped together in the vector space

In the figure, King and Queen are grouped close together due to their similarity, as are man and woman. Due to this grouping, the following vector arithmetic is true in this embedding:

$$V_{king} - V_{man} + V_{woman} = V_{queen}$$

While this observation might seem unrelated to the problem of finding the best vector representation at first, it can be used to find the best word embedding in the vector space. In fact, this is the precise approach that Pennington et al. [3] used to find a word embedding in a vector space.

In their paper, Pennington et al. describe a log-bilinear model that takes as input a large text corpus, computes the cooccurence matrix for the text, and then solves the corresponding weighted least-squares problem. The resulting vector representations exhibit precisely the similarity properties described above. Therefore, this vector representation is suitable for our application.
Pennington et al. have made the trained word vector model accessible at <http://nlp.stanford.edu/projects/glove/>, where each word-vector has dimension 100. We used this vector representation for our model.

Implementation

1) Model Architecture

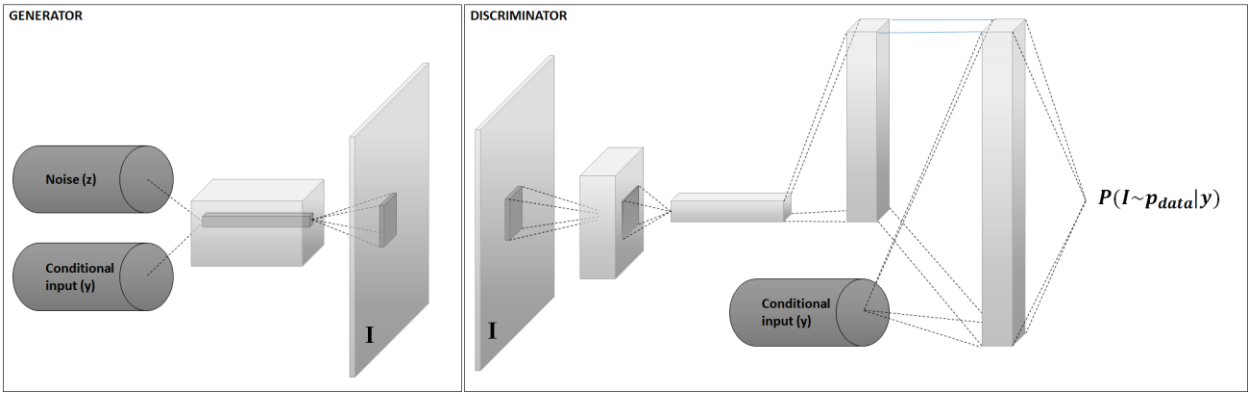

Figure 2: Schematic model architecture of generator and discriminator models (Image modified from Gauthier [2])

The model architecture can be viewed in the graphic above. Starting with the discriminator on the right, we built a fairly standard Convolutional Neural Network (CNN). The input (labeled *I* in the figure) is an image either from real data or from the generator. The next block is made up of three convolutional layers for which we utilized TensorFlow's convolution2d layers. The kernel size (or number of filters) and the stride (number of pixels which we slide the filter across) are 4 by 4 and 2 by 2, respectively, in each layer. After the third convolutional layer, we introduce the word vector (the cylinder in the figure). We concatenate this conditional information to the flattened output of the third convolutional layer. Then we add a fully connected layer (the last tall box on the right hand side of Figure 2). This layer was critical to the model's success and appears to differ from other attempts. Finally, the fifth layer is the output probability. Specifically, the discriminator outputs P(real data|y).

The generator consists of a transposed CNN which essentially swaps the forward with the backward passes of a standard CNN. First, we concatenate the random input z with the word vector y, and project this input vector onto a large fully connected layer – visualized by the two cylinders on the far left side of Figure 2. Inside of this box, we utilize three of TensorFlow's convolution2d_transpose layers with kernel size 5 by 5 and stride 2 by 2. The final output from the Generator is again a convolution2d_transpose layer of kernel size 32 by 32 which produces the image labeled *I* at the end of the figure.

In the TensorFlow graph, the generator is built first, so the first 10 trainable variables belong to that network (5 pairs of weights and biases). We connect the two networks through the loss function invented by Goodfellow et al. Then we update weights for the networks separately -- once for the discriminator and twice for the generator -- using the Adam Optimizer with a learning rate of 0.0002, weight decay of 0.5, and batch size of 100. These hyperparameter choices reflect the current literature, which have shown coincidentally, that these choices tend to result in images that are truly generative and not simply memorized.

2) Training

In order to train this model, we launched a Comute Optimized c4 xLarge instance on Amazon Web Services. On the Ubuntu server we downloaded Scipy, Matplotlib, Numpy, and Tensorflow packages. After transferring the python scripts, the models were ready to train.

We first downloaded the MNIST dataset and generated batches of training data with conditional data (x_i, y_i). The labels (y_i) are by default classes of integer data type, 0 through 9. We used a one-hot function and a word-embedding function, respectively, to train two different models. The first model was trained on one-hot vectors, the second was trained on GloVe word embeddings.

At each iteration we randomly sampled a batch of training data of size 100. First we calculated the discriminator output D(x,y) from the real sample data, the generator output from the noise and conditional input G(z,y) and then the discriminator output from the generated data D(G(z,y),y). This allowed us to then conduct the optimization process using the Adam Optimizer, where the cost functions for the generator and discriminator are as follows:

$$d_{loss} = -(\log(D(x,y)) + \log(1 - D(G(z,y)))$$
$$g_{loss} = -(\log(D(G(z,y))))$$

The generator loss function aims at maximizing the probability of the discriminator to samples from G(). Similarly, the discriminator loss function tries to maximize the probability assigned to true data and minimize the probability assigned to the generator output.
This process is repeated for many iterations (> 100,000) until the results converge and we do not see any significant updates in the objective functions.
The following algorithm illustrates this general process on the MNIST dataset that we used in our experiments.

Algorithm 1: Training procedure using MNIST dataset

- *For each iteration, do the following:*
  - Sample random batches (size 100) of MNIST data
  - Use word2vec to transform the conditional input into word vectors
  - Calculate

    D_x = Discriminator output from real images and word vectors

    G_z = Generated images from random input and word vectors

    D_g = Discriminator output from generated images and word vectors
  - Update the weights and biases with this loss function:

    $d_{loss} = -(\log(D_x) + \log(1 - D_g))$

    $g_{loss} = -(\log(D_g))$

In order to verify that the models were converging towards the optimal solution, we sampled generator output images every 1000 iterations in batches of 36 words (see results section for images). We also wrote a script to produce the 12 images of each class or word.

3) Data

We applied our conditional GAN model to two kinds of data in order to evaluate the possibilities of the model. We considered the CIFAR-100 dataset, and the MNIST dataset.

- CIFAR-100

The CIFAR-100 dataset, available at https://www.cs.toronto.edu/~kriz/cifar.html, consists of 100 classes of 600 images each. Classes are, for example, "boy", "chimpanzee", or "television". Each image is of size 32 by 32 by 3. The conditional inputs to the model are the 100 classes, and the real data is each image. The goal using this dataset would be to create an image based on a word input. For example, given the word "bicycle", we would like the generator to create a 32 by 32 by 3 image of a bicycle.

- MNIST

The MNIST dataset consists of 60,000 images of size 28 by 28 by 1. Each image shows a slightly different handwritten digit (0 to 9). The accompanying labels are simply the word of each digit.
The goal using this dataset is to create an image of the digit that corresponds to the word that was supplied. For example, the generator should output an image of a seven if the word "seven" is provided as conditional input. Furthermore, words such as couple, double, or pair, should lead to an image of a 2.

Results

1) CIFAR-100

We initially tried to apply our model to the CIFAR-100 dataset but were unable to obtain satisfactory results. We believe this is due to multiple factors.

Figure 3: CIFAR results from Goodfellow et al.

First, the CIFAR data is of size 28 by 28, which can make it difficult to see complicated shapes such as animals or airplanes. The results from Goodfellow et al., for example, show blurry images even after successful training of the unconditional model (Figure 3). We believe that while our model might have been tending in the right direction, the output images were too blurry to see anything meaningful. Furthermore, the CIFAR-100 dataset provides 600 images per class, which might have been too few for images of such complexity when using a conditional model.
Third, GAN models tend to underfit the data, so our model might not have been training correctly.

With the current hardware, and the short time frame to test many models, we were not capable of experimenting on the CIFAR-100 dataset. We therefore shifted our attention towards the simpler MNIST dataset, which we will discuss in the next section.

2) MNIST

Originally we experimented with an architecture proposed by Jon Gauthier in "Conditional generative adversarial nets for convolutional face generation." [2] This paper attempted conditional generation by concatenating the conditional information at the appropriate time during training. The training results of this model seemed promising at first, as can be seen in figure 4.

Figure 4: Generator output for 0,3000,10000, and 28000 training iterations (initial model)

Each number in the figures correspond to a generator output for a given conditional (word) input. The conditional inputs are fed to the generator in batches, which is why each image shows multiple numbers as outputs.
As can be seen in the figure, initially the model simply outputs white noise. After about 3000 iterations, the model seemed to show images of digits. However, after about 10000 iterations, the model appeared to produce worse results. The final output after about 29000 iterations also appeared slightly worse than the output after only 3000 iterations. However, we proceeded to test the final model nonetheless.

In order to test experiments, we created a script that would test the model's ability to deal with the conditional input. Figure 5 shows the output of the script for the current model. Each row represents a conditional input word from "zero" to "nine". Each word was input to the generator 12 times in order to see the different outputs produced by the model with various random inputs (Z changes while the word remains constant).
The first row in the figure shows the output of the model for the input word "zero". It is apparent, however, that the output resembles the digit "two" instead. Similarly, the second row shows the output for the word "one", but a three appears to be drawn in that row.

Figure 5: Generator output after training (initial model)

We concluded that the model did not take the conditional inputs into account when training. In theory, this could have been caused by a missing connection between the image input and the conditional

information. So, we decided to add another layer to the end of the discriminator in order to introduce new non-linearities, as shown in Figure 2. The results of this model will be discussed in the following section.

Added Projection Layer

As mentioned above, we modified the previous model by adding one more fully connected layer to the discriminator. We tested this model for a simpler "one-hot" conditional input configuration, and after promising results we returned to the more complicated word-to-vector input configuration. Figure 6 shows the generator outputs while training the model for different training iterations.

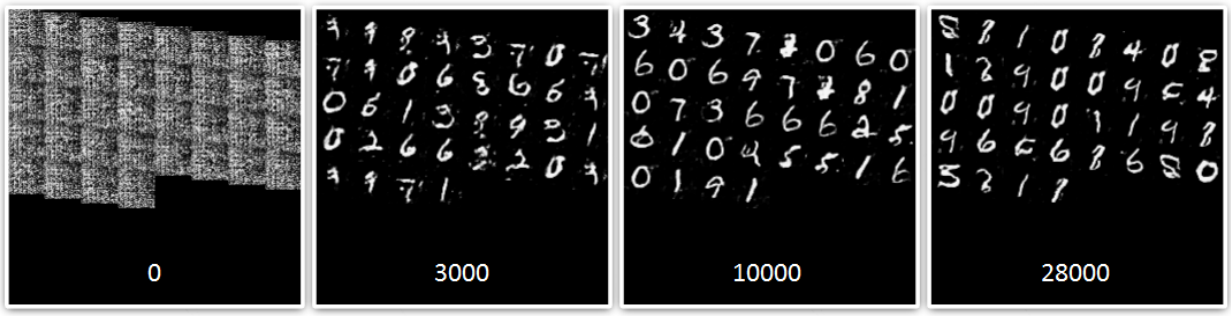

Figure 6: Generator output for 0,3000,10000, and 28000 iterations (final model)

Again, each number in the figures correspond to a generator output for a given conditional (word) input. As shown in the figure, the generator produced white noise output before training. After about 3000 training iterations, the numbers were already clearly visible. After 28000 iterations, the training had been completed and the generator output was very similar to the true MNIST data. In this case, training seemed to steadily converge to the optimal solution, -- a great improvement over the previous attempt. Therefore, we proceeded to run the test-script for the generator output of the trained model. Figure 7 shows the generator output after training for both the one-hot and word-2-vector conditional inputs.

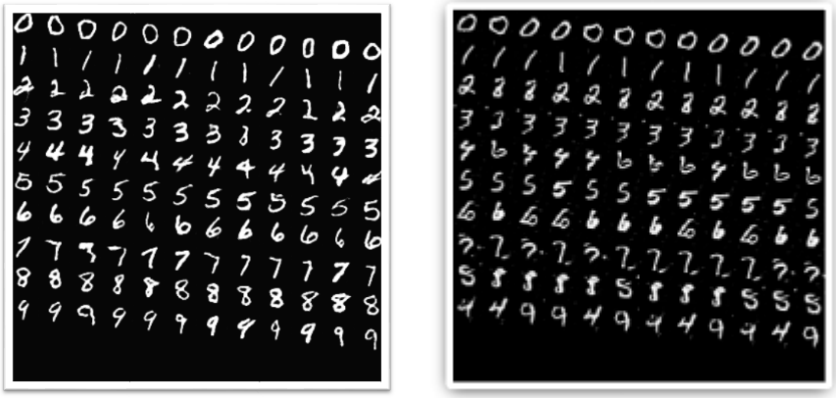

Figure 7: One-Hot and Word2Vec testing outputs

Just as before, each row shows 12 separate generator outputs of the input words "zero" through "nine". In this case, the output images appear to consistently show the correct digit with slight variations in

appearance, as desired by a truly generative model. This observation confirms that the model was trained correctly.

As can be seen in Figure 7, the model showed great accuracy for the input words “zero” through “nine”. This confirmed our approach of generating images from input words. We now proceeded to test the limits of the model, as shown in the next section.

Adding Noise to conditional input

With the model working and generating images of digits based on word inputs, we now tried to generalize the model to work for more words than those found in the training data. One idea was to add some noise to the input word vector (y) during training. Our hope was that this would have the effect of the model learning to associate the training images of digits not only with the vectors for the corresponding digit words (“zero” through “nine”) but also with words that are similar to those words. Similarity in this case would refer to two word vectors that are almost the same (see vector representation of words section for details).

Figure 8 displays this idea. The input word vector for the digit “two” could be anywhere in the neighborhood of the word vector for “two” during training as shown by the colored regions. Therefore, the word vector for “pair” would also have a training image of the digit “two” associated with it.

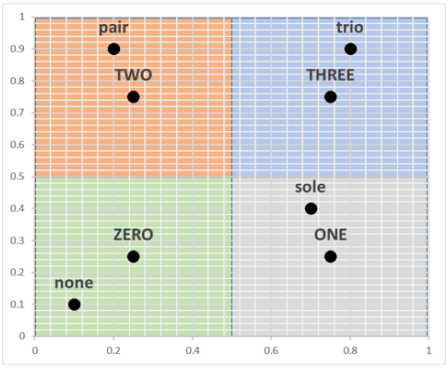

Figure 8: Learning the spaces around words

We chose to model the random noise as a uniform distribution so that the generator would learn the hypercube surrounding the target word vector. Thus it would have an idea of what similar words would look like as visualized in the two-dimensional figure above.

The limited results we obtained with this method showed some promise, even though not all the output digits seemed to make sense. Figure 9 shows two output examples of this method. The leftmost shape of the image is the output for the input vector corresponding to the word “nil” and it appears to show the digit “zero”. This is the correct output in this case, since nil is most closely related to the “zero” vector.

Figure 9: Generator output for nil, sole, pair, and triplets

Towards the right side of Figure 9, on the other hand, are the outputs for *sole, pair,* and *triplet.* Ideally, the outputs would resemble the image of the digits 1, 2, and 3. The generator's outputs are blurry in these cases, but do indeed show the potential of the idea.
These results show that the model tends to have a blurry idea of the unseen word, but much less clear than when the word vectors for the digits themselves are used.

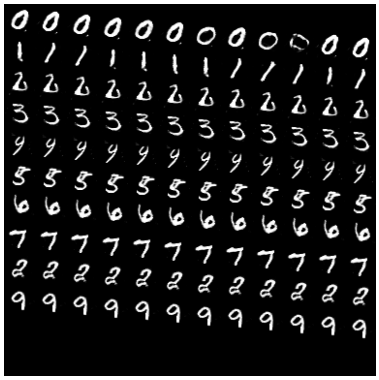

Figure 10: Final output from noisy model for words "zero" through "nine"

One very interesting take-away from the above image is that the model during training receives the exact word vectors with probability 0 due to the continuous noise perturbance. Nevertheless, it still learns the words "zero" through "nine" extremely convincingly. This suggests that we could achieve our desired results by experimenting with the distribution of noise. With more time, we would explore various bounds for the Uniform distribution and variances for a Gaussian Distribution which would learn the hypersphere surrounding our target words.

Future Work

There are two additional directions that could be pursued using the results from this paper. They will be summarized in the next two paragraphs.

1) New data and online learning

It would be an interesting experiment to expand this model to many more classes such as those in the CIFAR-100 dataset. Unfortunately, we were not able to extensively test the model on the CIFAR-100 data, but our preliminary results using the MNIST data show that the application of this model to the CIFAR-100 data seems plausible. One could imagine a Facebook, for example generating your friends' faces learned by their names, or an artist being able to generate a "Picasso" or "Van Gogh."

Additionally, the model tends to quickly learn general shapes and backgrounds. Therefore, it could be a good candidate for online learning in which new classes might be learned on the fly.

2) Chatbots

Clearly, the conditional GAN model responds well to word vectors. This suggests there may be promise for a chat bot which has learned through this method of adversarial training. We attempted a toy model using the sequence to sequence architecture from TensorFlow. However, this architecture failed in the cGAN set up because the sequential model is not updated at each time step as is the standard case of a perplexity-based loss with these architectures. Because of this, our model converged to the same output for each step in the sequence. There may be another architectural approach, though, that could work, and this would prove to be an interesting academic experiment with many real world applications.

Conclusion

In this paper, we successfully extended the original GAN model to conditional inputs. We demonstrated that a conditional general adversarial network can be appropriately applied to image and word vector data to produce images based on word inputs. Our results using the MNIST dataset show great accuracy when it comes to creating images of digits based on both one-hot vectors and their respective word inputs.

There is still room for improvement when it comes to generating images for words that are not closely related to words in the training set. The theory of adding various noise distributions is promising. We think that an especially intriguing application of this idea would be for datasets which have very disparate classes. Thus, the model could learn shapes around the target words without interference from other training words.

The next logical steps involve applying this model to CIFAR or similarly large image datasets with more classes than MNIST. A working model using these datasets could, for example, create images of animals based on the supplied word input.

Appendix

The full code can be viewed here on Github and is attached as a Zipfile:
https://github.com/aslakey/GANN/tree/master/MNISTGANN

MNISTGANN.py: Recreates the original GAN paper by Ian Goodfellow et al.
MNIST_data.py: Downloads MNIST dataset and prepares random batches with labels
MNIST_utils.py: Utility functions for the MNIST GAN models
w2vMNISTGANN.py: Conditional GAN with word vectors as input
one_hot_MNISTGANN.py: Conditional GAN with one hot vectors as input
noiseyw2vMNISTGANN.py: Conditional GAN with noisey word vectors as input

use_w2vmodel.py: Saves 12 images of each of the nine words
use_onehotmodel.py: Saves 12 images of each of the nine one-hot vectors
word_to_image.py: Converts any word vector into an image by using the GloVe embeddings

conditional_GANN_model.py: The Conditional GAN model for CIFAR100 data
Cifar_load.py: loads cifar100 data and has some utility functions for using word vectors

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. 2014. Generative Adversarial Networks. https://arxiv.org/abs/1406.2661

[2] Jon Gauthier. 2014. Conditional generative adversarial nets for convolutional face generation. http://www.foldl.me/uploads/2015/conditional-gans-face-generation/paper.pdf

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. http://nlp.stanford.edu/projects/glove/

[4] Andrej Karpathy, Pieter Abbeel, Greg Brockman, Peter Chen, Vicki Cheung, Ian Goodfellow, Durk Kingma, Jonathan Ho, Rein Houthooft, Tim Salimans, John Schulman, Ilya Sutskever, Wojciech Zaremba. 2016. Generative Models. https://openai.com/blog/generative-models/

[5] Alec Radford, Luke Metz, Soumith Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks https://arxiv.org/abs/1511.06434

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

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf