Text to Image Generation

This project leverages a neural network to create images from text using TensorFlow 2. The text consists of two components:

Chars: The characters to be depicted in the image

Spec: A specification outlining the format of the text, such as color, font, and style

The data is artificially generated by randomly sampling the characters and specifications, and then plotting the image using matplotlib. The advantage of generating synthetic data is the ability to gradually increase the task difficulty as the model improves. The ultimate objective is to employ a Generative Adversarial Network (GAN).

Part 1: Training a generator directly

Our dataset is noise-free, allowing for a deterministic mapping from text to images and enabling us to train a neural network using supervised learning. The structure of our generator is straightforward: utilizing two distinct RNNs, we process variable-length text inputs (characters and specifications) to produce a fixed-length encoding. This encoding is then passed through a fully connected layer to undergo a non-linear transformation and increase in size. Subsequently, the encoding initializes a feature map, which is upsampled through fractional-strided convolutions to yield an image with dimensions of 128x128x3. Considering that the generator's output is later employed in the GAN discriminator, we opt for a tanh activation of the final convolution to ensure the output is centered around 0..

A loss function for tanh

In the following, we are introducing a loss function designed specifically for the tanh activation. Its gradient exhibits the same favorable characteristics as when using the sigmoid function in conjunction with the binary cross-entropy. It's important to note that the tanh can be expressed using the sigmoid function:

$$\tanh(h) = \frac{e^{+h} - e^{-h}}{e^{+h} + e^{-h}} = \frac{1 - e^{-2h}}{1 + e^{-2h}} = \frac{2 - (1 + e^{-2h})}{1 + e^{-2h}} = 2\sigma(2x) - 1$$

Instead of using the sigmoid function and normalizing the data as part of the discriminator model, it might have been more convenient and equivalent to do so. However, since this is a personal project, let's have some fun with the math. With reference to the binary cross-entropy applied to the sigmoid, the following loss function is suggested:

$$L = -\frac{1}{2}\Big[(1+y)\log(1+p) + (1-y)\log(1-p)\Big]$$

Where y represents the ground truth value of the pixels, and p=tanh(h) is the prediction of the network. Here, h represents the output of the last hidden layer, i.e., the logits.

Gradient properties

We demonstrate next that this loss function yields favorable properties for the gradient when combined with tanh. Let's calculate the gradient of the loss L in relation to the logits h:

$$\frac{\partial L}{\partial h} = -\frac{1}{2}\left[\frac{1+y}{1+p}\frac{\partial p}{\partial h} - \frac{1-y}{1-p}\frac{\partial p}{\partial h}\right]$$

The derivative of the activations *p* with respect to the logits *h* takes a simple form:

$$\frac{\partial p}{\partial h} = \frac{1}{\coth^2(h)} = 1 - \tanh^2(h) = 1 - p^2 = (1-p).(1+p)$$

Using this in the equation just above, shows that the gradient propagated through the activations is directly proportional to the difference between activations and targets (ie no vanishing nor exploding gradients):

$$\frac{\partial L}{\partial h} = -\frac{1}{2}\Big[(1+y)(1-p) - (1-y)(1+p)\Big] = p - y$$

Numerically stable version

I have used develop numerically stable version of the loss L from the logits h:

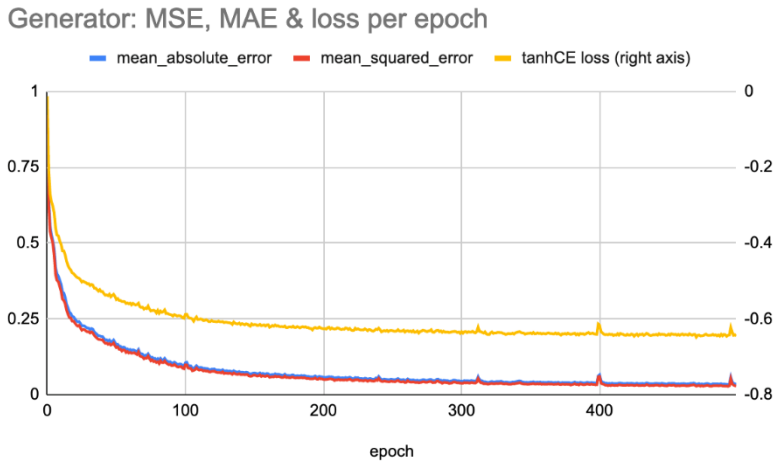
$$
\begin{aligned}
L &= -\frac{1}{2}\left[(1+y)\log\left(\frac{(1+e^{-2h})+(1-e^{-2h})}{1+e^{-2h}}\right)+(1-y)\log\left(\frac{(1+e^{-2h})-(1-e^{-2h})}{1+e^{-2h}}\right)\right] \\
&= -\frac{1}{2}\left[(1+y)\left[\log(2)-\log(1+e^{-2h})\right]+(1-y)\left[\log(2)-2h-\log(1+e^{-2h})\right]\right] \\
&= \log(1+e^{-2h})+h-yh-\log(2)
\end{aligned}
$$

To avoid an exponent of a positive number, which might be too large, let's rewrite this for the case where $h<0$:

$$
\begin{aligned}
L &= \log(1-e^{-2h})+\left(\log(e^{2h})-\log(e^{2h})\right)+h-yh-\log(2) \\
&= \log(e^{2h}+1)-h-yh-\log(2)
\end{aligned}
$$

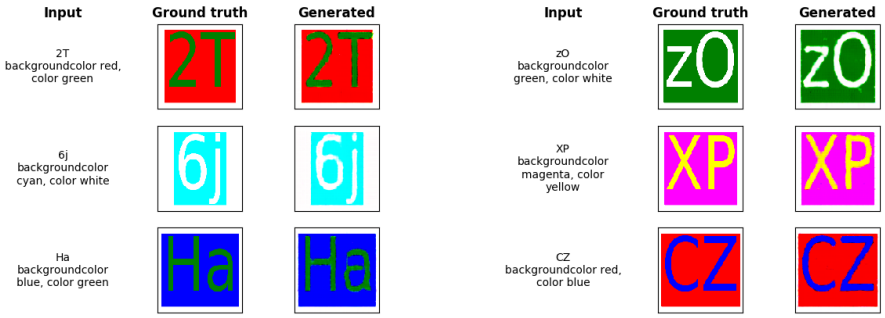
Both cases where h is less than 0 and greater than 0 can be rewritten as follows by dropping the constant:

"We train the generator for 500 epochs. Each epoch consists of 64 batches of size 16, and we observe the following convergence."



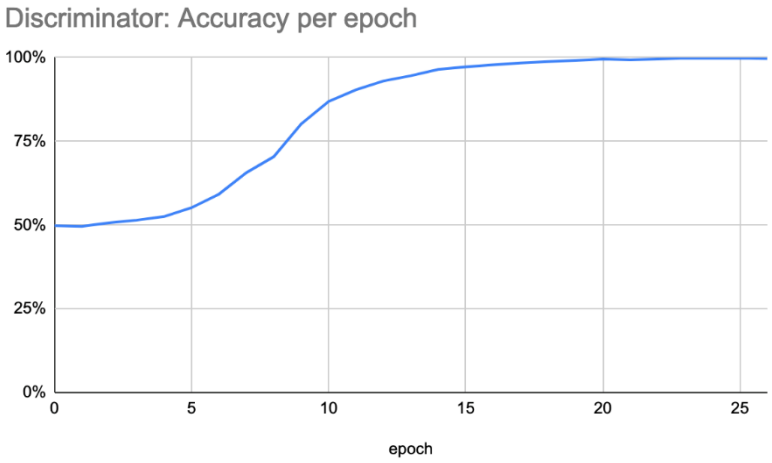
As our data is synthetically generated, it encompasses the entire input space, containing all possible combinations of characters and special characters. Therefore, we can confidently assess the model using the training data and have verified that the metrics on a separate validation set are nearly identical to those

of the training set. Below is a sample of the generated image:



Discriminator

We train a discriminator using the converged generator and achieve near 100% accuracy after
only a few epochs.



Part 2: A Generative Adversarial Net

In this research, we embarked on training a Generative Adversarial Network (GAN) leveraging our synthesized dataset. Comprising a generator and a discriminator, the architecture of our GAN was influenced by the insights provided in Radford et al. 2015, specifically tailored for DCGANs.

Our model, however, diverges from conventional GAN frameworks, evolving into a fully-conditional GAN. This denotes a paradigm where solely conditional data is employed as input, omitting the latent variable z. This methodology draws parallels to the work by Isola et al. 2016, which demonstrates image

generation conditioned upon another image. Their approach utilizes a U-net architecture that capitalizes on the local features within the conditional image to synthesize the output image. This direct leverage of conditional information, though advantageous in their context, presents a stark contrast to our scenario where the input (text representation) and the output (image domain) are markedly disparate.

A significant challenge encountered was the necessity to generate images with high regularity, characterized by uniform text and backgrounds alongside sharp edges. This scenario facilitated the discriminator's ability to distinguish real from fake images based on surface features rather than the images' content. Such a condition undermines the provision of meaningful gradients to the generator.

These elements contributed to the inherent difficulty and instability observed during the GAN's training process, necessitating substantial adjustments to the conventional GAN framework to obtain satisfactory outcomes.

Interestingly, the deterministic relationship between the text input and image output in our study enables the definition of a precise evaluation metric for our GAN, diverging from the norm in current literature. We opted for the mean absolute error (MAE) as our evaluation metric, considering its stability compared to the root mean square error (RMSE).

Ablation study

The following graph shows the MAE per epoch for various setups of our GAN. Below we discuss the changes we had to make to generate high quality images. We show the default parameters in series Baseline.

Shuffled text as fake data

It's important to remember the information below. Our GAN is able to quickly generate images with the correct background size and colors. However, it struggles to produce clearly identifiable characters because the discriminator relies on features such as edge smoothness rather than individual character identification. To address this issue, we also feed the discriminator real images

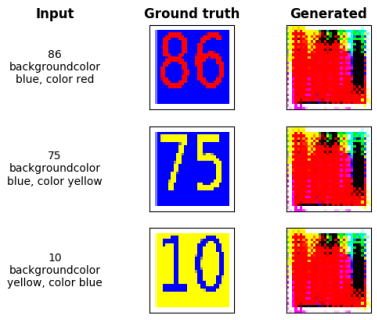
with random text and train it to recognize them as fake data. This forces the discriminator to accurately process the image and text and provide useful gradients to the generator. As far as we know, this method can be applied to other cGANs and has not been documented in the literature.

During practical implementation, we take a batch of real images and text and shuffle either characters, spec, or both. Typically, we shuffle characters only 60% of the time to compel the discriminator to learn our vocabulary.

In the graph, the series "Shuffle" represents training without this feature, resulting in almost nonexistent convergence. While we did find alternative ways to achieve convergence without this feature (e.g., lowering the learning rate and MBD, see the next section), we believe that this feature is the most crucial improvement to our model. We are confident that this approach could also aid in the convergence of other cGANs and should be further explored in research.

Minibatch Discrimination

One problem of our GAN (and GANs in general) is mode-collapse, ie all generated images are almost identical, as shown in the image below. Once mode collapse occurs, there is no way to escape it, since it is a stable local optima to our min-max problem. To help the discriminator detect this, Salimans et al. 2016 suggest Minibatch Discrimination (MBD). This technique allows the discriminator to look at multiple images from one batch and detect features common between them. For this technique to work, fake and real images have to be provided to the discriminator in separate batches (see next section).



It turns out that with text-shuffling, this method is actually counterproductive as shown in series +MBD. In many situations where training was unstable, this method did however help as is shown in series -Shuffle+MBD. We conclude therefore, that MBD acts as a patch to instabilities, but introduces training

difficulties on its own and should therefore be avoided if possible. We also note that, MBD introduces many parameters to the discriminator and makes it consequently a lot slower.

Separate real & fake batches

We feed batches of fake and real images separately to the discriminator. Besides simplifying the implementation a little, this has also the effect of accelerating and improving convergence. This is because doing so effectively increases the learning rate of the discriminator. Like most public implementations, we use the same parameters when training the discriminator and generator. It seems however likely that the generator is much harder to train, since it is effectively a lot deeper from a gradient perspective. It does therefore require more conservative optimization parameter than the discriminator. It is also known that the generator can only be as good as the discriminator. Allowing the discriminator to converge faster, can thus only be beneficial. To our surprise and to our knowledge, using higher learning rate for the discriminator (and in general more suited optimization parameters) is not explored in the literature.

Learning rate

The literature advises the use of a low learning rates to facilitate convergence. In order to achieve convergence with a majority of our setups, we require a learning rate of 5-e5, which turns out to be about one matter of size lower than is common (2e-4 is advised for DCGANs). Our final setup however, is so stable, especially due to text-shuffling, that training succeeds with a learning rate of 5e-4 as shown in series LRx10. For our ablation studies, we keep a low learning rate to make sure the instability are not due to this factor.

Adam

As is common with GANs, we use Adam for optimization. In many setups however, and in accordance with Radford et al. 2015, the default values for β did not work. We had to reduce β1, which controls momentum, from .9 to .5 and β2, which controls adaptiveness to the gradient norm, from .99 to `.9. This indicates that the direction and norm of our gradients change significantly over the course of the optimization.

In the graph of our ablation study, series AdamParams has been run with the default parameters. The difference is marginal for our baseline setup, which is again due to text shuffling: Since the shuffled data does not come from our generator, this method has the effect of making the discriminator training more decoupled from the current state of the generator. This tends to stabilize the gradients across epochs for the discriminator. An additional benefit of text-shuffling, we haven't mentioned so far.

Limited vocabulary

Please remember the following information:

Our original vocabulary, consisting of numbers and case-sensitive letters, has a size of 62. However, this size is too large for successful convergence. As a result, we have reduced our vocabulary to only numbers, resulting in a size of 10. This means that our dataset now has a size of 6400. With such a small dataset size, we are essentially learning to overfit, despite using methods such as dropout and layer normalization to mitigate against this. The challenge, however, is to get a difficult-to-train model to overfit. This requires numerous improvements to achieve

In the FullVocab series, we train the model with the full vocabulary size, resulting in an effective dataset size of 246K. Our model does converge, but at a much slower rate. This demonstrates that our method does not solely rely on overfitting. The limited vocabulary allows for accelerated training and faster comparison of different model variants.

We also experimented with a variant where we initially train the model with the default vocabulary size of 10 for the first 500 epochs. Every 10 epochs thereafter, we introduce one additional character to the vocabulary. The idea behind this approach is to first overfit the model on a small dataset. Since the model is trained with methods to avoid overfitting, it can actually learn features that it would not have learned on a larger and more challenging dataset. As we increase the size of the dataset, the hope is that these features can gradually be generalized across the entire dataset. In other words, learning generalized features is easier to accomplish gradually rather than all at once, particularly with difficult-to-train setups such as GANs. Series Increment Vocab demonstrates that this concept indeed has merit. It is worth noting that in our generated data scenario, increasing the dataset's difficulty occurs naturally. This

approach can be applied to any dataset by simply increasing the dataset size per epoch. Over the past decade, numerous successful methods to prevent overfitting have been developed. Therefore, it is acceptable to initially use overfitting to overcome training instabilities, and we believe it is an interesting research idea worth further exploration.

Pretrained text-RNN

In our approach, we use text instead of images and categorical one-hot encodings, making our network very deep due to the presence of text RNNs. We have found that initializing the RNN with pre-trained weights can speed up convergence, although we don't keep these weights fixed and propagate gradients throughout the network.

Our "RNNinit" series demonstrates that training still converges even with randomly initialized weights, thanks to training the discriminator with shuffled data. As mentioned in the section about Adam, allowing the discriminator to see real and fake data that isn't generated by the generator enables it to efficiently learn the meaning of our conditional data. This is crucial for training a neural network that provides conditional data and, to our knowledge, has not been done before. Importantly, we do not propagate the generator's gradient into the RNN, as doing so could sabotage the RNN and prevent the discriminator from using conditional information.

Noise

The addition of noise to both real and fake images before they are presented to the discriminator, as recommended by Arjovsky et al. in 2017, helps to stabilize training. Since our images are highly uniform, adding noise is particularly beneficial in preventing the discriminator from distinguishing between real and fake images solely based on color uniformity. An advantage of using Keras is that we don't need to manually add noise to the data; instead, we can utilize a specific layer of the network for this purpose.

In our ablation study, the ImgNoise series is related to this feature. When combined with text-shuffling, its impact is minimal, but it has a greater effect in less powerful setups.

A note on Wassersstein-GANs

In addressing convergence issues, we've implemented an advanced GAN approach known as Wasserstein GAN. Our observation shows that WGAN achieves faster initial convergence compared to standard cross-entropy loss, as promised by this method due to unsaturated gradients even with a poor generator.

Regrettably, in our experience with WGAN-GP, convergence quickly stagnates, hindering the generator's ability to produce recognizable characters. It seems that cross entropy is more effective in producing better gradients in the later stages of training, which is the most challenging part of our GAN training. Our observations are consistent with those of Lucic et al. (2018), who found that many "improved" GAN versions do not surpass the original paper's implementation.

To make the Wasserstein GAN effective, we had to introduce several enhancements.

Gradient Penalty

We discovered that enforcing the Lipschitz constraint through weight clipping prevented convergence. In Gulrajani et al. 2017, they imposed a soft unit norm constraint on the gradient with respect to the input. Implementing this was challenging due to an issue in Keras, so we had to optimize the constraint in a separate step, rather than in a single step that also optimizes the distance between the distributions of real and fake images. However, this is not a major issue since we went even further by training real and fake images separately in any case.

Shared optimizer

It was discovered that when optimizing the constraint separately, the distance measure made the training very sensitive to the value of the balancing factor λ. To address this issue, we now use the same optimizer for both steps. The stabilization is achieved through the use of adaptive gradients included in Adam, which effectively coordinate the gradients between both steps.

No momentum

We use Adam without momentum ($\beta 1=0$) because the rapid convergence of the WGAN-GP causes the discriminator to change significantly between steps, varying the direction of the generator's gradients, which makes momentum counterproductive.

What didn't work

We investigated various approaches to improving performance, many of which failed. We list just a few below for completeness:

Latent variable

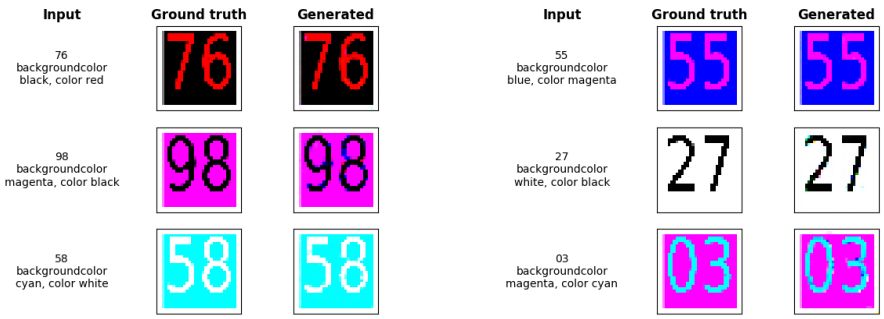
In our experiment, we fed the generator a latent variable despite our images lacking any latent, unobserved components. The hypothesis was that the randomness introduced by \(z\) could enhance the generation of realistic images, initially independent of the text. The expectation was that, at a later stage, the generator would adjust its output to become conditional on the text. However, our observations indicated that the introduction of latent noise adversely affected the generator's ability to produce images.

Discriminator only steps

The original GAN paper suggests training the discriminator only for k steps and then the generator as well for one step. We find that this slows down convergence. The fact that this is not needed is most likely due to the effectively higher learning-rate for the discriminator introduced by training real and fake images separately (see above).

Results

Samples form our baseline GAN are shown below:



We notice that the model has difficulties generating matplotlib-green which is (0, 0.5, 0) in RGB. THis is because it is the only color with a value of .5, which is where the sigmoid/tanh has maximum slope. The output value of the model must therefore be very precise, which is hard.