Gans on fashion mnist

Eduardo Cuesta Córdova
A00570171

The problem

A Generative Adversarial Neural Network is (GAN) used to generate small black and white images of fashion items.

What is a GAN

A Generative Adversarial Neural Network is a network architecture aimed at generating data that mimics the way in which a dataset behaves. This is accomplished by using both a discriminator and a generator network. The discriminator is trained to determine whether a piece of data comes from the dataset or from the generator. The generator is fed random noise and is trained to mimic the distribution of the source data via maximizing the probability of the discriminator thinking that the made-up data is part of the source data [1].

For the case of images, the discriminator in a GAN uses a Convolutional Neural Network, which applies a series of convolutions to a source image to detect features via trained weights on it and then produce an output according to those features, sometimes via a fully connected neural network.

A generator in a GAN for images normally uses learned deconvolutions to transform random noise into an image with the trained characteristics [2].

Why a GAN is good for this problem

GANS can be used in simple datasets like MNIST with success [1] and some variations like DCGANS have performed very well when generating more complex images [2]. The idea was initially to make a 4 layer DCGAN to generate the images, but I couldn't manage to outperform a 3 layer generator and 2 layer discriminator.

Information about the dataset

The dataset used to train the model is Fashion Mnist. It contains 70,000 28x28 grayscale images of ten categories: t-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags and ankle boots. The GAN was trained with both the 60,000 training images and the 10,000 images intended for testing [3].
Figure 1 shows some of the items of the dataset:

Figure 1: Dataset example

Architecture

The generator uses 2 transposed convolutions with batch normalization [2] and leaky ReLU activations [4]. In the final layer there is no batch normalization to make the model able to replicate the statistical qualities of the data and the activation function is a tanh [2].

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 6272)              809088
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 128)       262272
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 128)       262272
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 28, 128)       512
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 1)         6273
=================================================================
Total params: 1,340,929
Trainable params: 1,340,417
Non-trainable params: 512
```

Figure 2: The architecture of the generator

The discriminator uses 2 convolutional layers, the first of which has no batch normalization to be able to represent better the statistical qualities of the data. All activations for the convolutional layers are Leaky ReLU There are no fully connected layers in the model. At the end of the network there is global max pooling. The max pooling improves stability in DCGANS but hurts convergence speed. Since the network didn't have a lot of parameters and was fast to train, the speed was not a problem and the max pooling was preserved [2]. Finally, there is a single neuron with a linear activation for the output, which makes training twice as fast as with a sigmoid. The sigmoid also got stuck at a certain loss value without good results.

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 64)        640
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
batch_normalization (BatchNo (None, 7, 7, 128)         512
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
global_max_pooling2d (Global (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 75,137
Trainable params: 74,881
Non-trainable params: 256
```

Figure 3: The architecture of the discriminator

Results

The network specialized in making pullovers and some other images that look like dresses, which was still a valid strategy for the model to converge due to the images looking like the original dataset in spite of belonging to only one or two classes. The lack of expressiveness and capacity for producing other classes probably originated from the lack of parameters of the model on both the discriminator and the generator when compared with more normal DCGANS, since their architectures are deeper [2]. Some of the generated images are shown in figure 4:

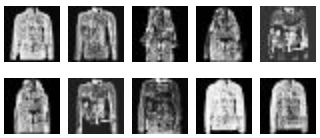

Figure 4: Images of pullovers and things that look like dresses generated by the model

Other experiments

The model was tried without batch normalization, but the images were not so nitid in spite of looking a bit like shoes and pants as shown in figure 5:

Figure 5: Images before batch normalization

With batch normalization on every layer but with sigmoid outputs both losses got stuck at 0.6931 from epochs 6 to 50 (in which training was stopped). The losses getting stuck at 0.6931 were also a recurrent problem later on. Furthermore, the images were not very nitid:

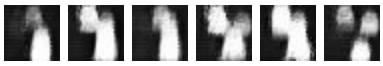

Figure 6: images with sigmoid outputs

With batch normalization, a linear activation in the clasificator and a tanh activation in the generator the results were those shown in figure 4 and presented in this report. This was the most successful model, even outclassing those with more layers and parameters in both the generator and the discriminator.

The generator architecture was changed to four transposed convolution layers fully following the guidelines presented in the DCGAN paper [2] but without changing the discriminator. This made the loss go to very high values in the generator and to negative values in the discriminator, getting further away from zero on each iteration.

Then the generator was made more complex with the four convolutional layers mimicking the generator and a flatten connected to a single neuron with a sigmoid output as suggested by the DCGAN paper, but the loss got stuck again at 0.6931.

The top layer of the convolutional network was then reverted to maxpoolilng with a linear activation, but the loss went very fast to even more extreme values than when the generator architecture was first changed.

Finally, the adam optimizer was changed to match the learning rate and momentum parameters that are suggested in the DGGANs paper [2], but both losses converged even faster to 0.6931.

Things to try in the future

Adding a different layer than a neuron in the output layer that summed all the values and then just passed them through a sigmoid, or a very strong discriminator with many same convolutions to not run out of dimensions too fast due to the small resolution of the image.

References:

N, f, alg, pub, pages

1. Goodfellow, I. J. et al. 2014. Generative Adversarial Nets. https://arxiv.org/pdf/1406.2661.pdf

2. Ratford, A. Metz, L. Chintala, S. Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. https://arxiv.org/pdf/1511.06434.pdf

3. Fashion MNIST dataset, an alternative to MNIST. Keras documentation. https://keras.io/api/datasets/fashion_mnist/

4. Chintala, S. et al. Gan Hacks. Github Repository: https://github.com/soumith/ganhacks/blob/master/README.md