Machine Learning Nanodegree Capstone Proposal

Generating human faces using Deep Convolutional Generative Adversarial Networks

Raj Patel 26/11/2017

Domain Background

With the help of the recent advances in Machine Learning / Deep Learning and knowledge based AI computers are able to replicate complex tasks that are done the by humans they can recognize and detect objects from images, recognize speech, understand natural language, try to mimic and learn how to play games etc. And they do these tasks with high level of accuracy. But they are still quite far from replacing human. As humans we have a characteristic that is creativity that was believed not to be possessed by a computers, but that's not true. With generative models the idea of making machines to create new things is becoming possible. The generative models can learn to generate data similar to data that we give to them. The generated data can be useful not only in generative tasks but also in fulling the missing data in some real samples and to get internal representation which may be useful for other machine learning tasks . One of such Generative Models is Generative Adversarial Network.

Generative Adversarial Networks have been regarded as the most prominent models according to some machine learning pioneers. Generative Adversarial Networks also known as GANs were introduced by Ian Goodfellow in 2014 (Goodfellow, et al., 2014). We are going to use a special type of GAN referred as DCGAN or Deep convolutional GAN. The whole idea behind GAN is to have a zero-sum game framework by using two neural networks contesting with each other. One neural network is the generator and the main task of the generator is generating new items the other neural network is a discriminator and the task of discriminator is to distinguish between real and fake items. The items that would be generated by the generator

would be fake items since it is trying to mimic the real data items the main goal of the generator is to make these data items as real as possible so that it can fool the discriminator on the other hand the goal of discriminator is to distinguish these fake these and real items as best as possible so here the discriminator works as a adversary judging the real and the fake items. So at the start the generator produces some fake data items these fake data items are feed into the discriminator along with the real data items and the discriminator is made to learn which are real and fake. The results of the discriminator are than further used to improve both the generator and itself. Backpropagation is used on both the networks so that so that the generator produces better images, while the discriminator becomes more skilled at flagging data items. (Goodfellow I. , 2016) This process continues indefinitely and in the end we get two high trained models one that is highly capable of generating new data items and other that is high capable at distinguishing these data items.

Problem Statement

The goal for this project is to build a DCGAN so that we build a generator can generate images of real looking human faces just by giving it some random noise.

Datasets and Inputs

The dataset which we will be using is a dataset known as CelebA. CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter (Tang, Liu, Luo, Wang, & Xiaoou, 2015). CelebA has large diversities, large quantities, and rich annotations, including

1. 10,177 number of identities,
2. 202,599 number of face images, and
3. 5 landmark locations, 40 binary attributes annotations per image.

For the context of this project I would only require the images of the faces without requiring any other information. Also I will not be using all 200k images I would only use around 25 % of the dataset i.e. 50,000 images.

(Dataset obtained from URL: http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html) (Tang, Liu, Luo, Wang, & Xiaoou, 2015)

Solution Statement

As mentioned above I will be using a Deep Convolutional Generative Adversarial Network. DCGAN are a subtype of GAN where a Convolutional neural network is used as a discriminator and a deconvolutional neural network is used a generator. The convolutional neural network (discriminator) would be made to train on real images of human faces and the fake images generated from the deconvolutional neural network (generator), from the results of the discriminator losses would be calculated of both the generator and the discriminator based upon their loss functions and they will be backpropogated so that eventually our generator learns to produce real looking human face images (Radford, Metz, & Chintala, 2015).

Benchmark Model

1. Random Generator: Here I will make write an algorithm which will generate random images. The algorithm won't have any knowledge of what it is trying to generate. To test out the generated images we could visually compare them to the real ones and also I will pass them to a CNN trained on this dataset to get the accuracy. I choose this model to compare how well the output of my GAN compares to a model having no information regarding the images it wants to generate.
2. HyperGAN: HyperGAN is a composable GAN API and CLI. Build for developers, researchers and artists. Models inside are made using Tensorflow and it uses pygame for visualization. I plan to compare my results with the results obtained from the HyperGAN. I choose this model to compare the outputs of my model to already existing models and how well it compares as per the quality and clarity of images it generates.

Evaluation Metrics

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

The training of GAN can be represented as the mathematical function above. {Image src : (SHAIKH, 2017)}

- z : Represents the initial random noise or input given to the generator
- x : Represents the actual data to be feed to the discriminator in our case it would be the real human face images
- G (z): Represents the output of the generator in our case it is the image of face to be generated. Note: G is a differentiable function which tries to map the input z based on its distribution p_g.
- D(x): Represents the probability that x came from data rather than p_g. Here in our case the discriminator will give out a probability suggesting how real the given real image is.
- D (G (z)): Represents the probability that how likely the output of the generator is similar to the real data. Here in our case it will give probability suggesting how real the given fake image looks like.

We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize log (1 – D (G (z))). Error Estimation of both the networks is a form of logloss function. Initially when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, log (1 - D (G (z))) saturates. Rather than training G to minimize log (1 - D (G (z))) we can train G to maximize log (D (G (z))). In other words, D and G play the following two-player minimax game with value function V (G; D). (Goodfellow, et al., 2014) (Bruner & Deshpande, 2017)

The metrics we will be using is the discriminator loss and generator loss after every epoch saved in numpy's npy format. Ideally for a GAN this losses should converge to a same number but that is not possible in a minmax relationship, but after sometime of training this losses should look

similar. Also we will visually compare the output of the generator and the actual data along with the test accuracy of the discriminator.

Project Design:

- Software Requirements : Python 3.5+
- Dependencies: Numpy, Matplotlib, Tensorflow-gpu 1.4, Keras, OpenCV.
- Workflow :
    1. Downloading the dataset and pre-processing the data.
    2. Establishing the baselines from our benchmark models i.e. (Random generator and HyperGAN).
    3. Deciding and building the architecture of Generator and Discriminator models and a combined GAN model comprised of both generator and discriminator.
    4. Choosing optimizer, learning rate and then compiling the model.
    5. Training the models as per our training procedure mentioned above and making initial observations.
    6. Fine tuning the models and making a final run.
    7. Plotting the loss of individual models and making final observations.
    8. Comparing the results obtained from all three observations and making the final inference.

References and Citations

Bruner, J., & Deshpande, A. (2017). *Generative Adversarial Networks for Beginners*. Retrieved from Oreilly: https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners

Goodfellow, I. (2016). NIPS 2016 Tutorial: Generative Adversarial Networks. *Cornell University Library*, 2-4.

Goodfellow, I. J., Pouget-Abadie, Jean, Mirza, M., Xu, B., Warde-Farley, . . . Bengio, Y. (2014). Generative Adversarial Networks. *https://arxiv.org/abs/1406.2661 Cornell University Library*, 1-5.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *https://arxiv.org/abs/1511.06434 Cornell University Library*, 3-16.

SHAIKH, F. (2017, June 15). *Introductory guide to Generative Adversarial Networks* . Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/

Tang, Liu, Z., Luo, P., Wang, X., & Xiaoou. (2015). *Deep Learning Face Attributes in the Wild.*