A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras

Harsh Dhamecha — Published On June 13, 2021 and Last Modified On August 27th, 2021

Advanced Deep Learning Image Project Python Unstructured Data Unsupervised

This article was published as a part of the Data Science Blogathon

Introduction

Have you ever visited this website? The name of the website exactly matches what it does. Yes, you've heard it right! The person you just saw in an image on the website **does not really exist in the world.** Again visit the website and keep refreshing the page. You'll see different people each time who do not really exist. This seems like a **MAGIC** right (at least at first sight) and the **Generative Adversarial Network** is the **MAGICIAN!**

In this article, We'll be discussing the Generative Adversarial Networks(GAN in short). Ian J. Goodfellow and co-authors have introduced the GAN in the year 2014. Prominent Researcher in the area of Deep Learning – Yann LeCun, described it as:

> ***"the most interesting idea in the last 10 years in Machine Learning".***

You can now guess the **importance** of the topic we're talking about! So, let's get started.

After reading this article, you will know:

- Generative Adversarial Networks in depth
- How to Implement GAN using Tensorflow and Keras
- Some Amazing Applications of GAN

Table of Contents

1. What is a GAN?
2. How does GAN work?
3. Amazing Applications of GAN
4. Implementing a Toy GAN
5. Summary

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras

Let us take an analogy to understand this concept:

Let's say you are a cricket player and unfortunately you are not good at facing *Yorkers.* What would you do to overcome this? You would simply ask bowlers to bowl *Yorkers* in your net sessions to get better at it. You would also observe the batsmen who are good at facing Yorkers. You probably keep practicing and learning from your mistakes. You would repeat this step until you become ~~good~~ **BEST** at facing Yorkers. A similar concept can be incorporated in GANs.

The more you face Yorkers, the better you'll become at facing Yorkers. But wait, to face more Yorkers, you **must have the bowlers who can bowl Yorkers more frequently.** So, you become good at facing Yorkers indirectly depends on the bowlers you choose.

Simply, for getting a powerful hero (generator), we need a more powerful opponent (discriminator)!

Now, let us understand it technically.

Generative Adversarial Networks(GAN in short) is an advancement in the field of Machine Learning which is capable of generating new data samples including Text, Audio, Images, Videos, etc. using previously available data. GANs consist of two Artificial Neural Networks or Convolution Neural Networks models namely **Generator** and **Discriminator** which are trained against each other (and thus *Adversarial*). We'll discuss more this in the following section.

How does GAN Work?

As we've discussed that GANs consists of two ANN or CNN models:

1. Generator Model: Used to generate new images which look like real images.
2. Discriminator Model: Used to classify images as real or fake.

Let us understand each separately.

Note: For simplicity, we'll consider the Image Generation application to understand the GANs. Similar concepts can be applied to other applications.

The Generator Model

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras

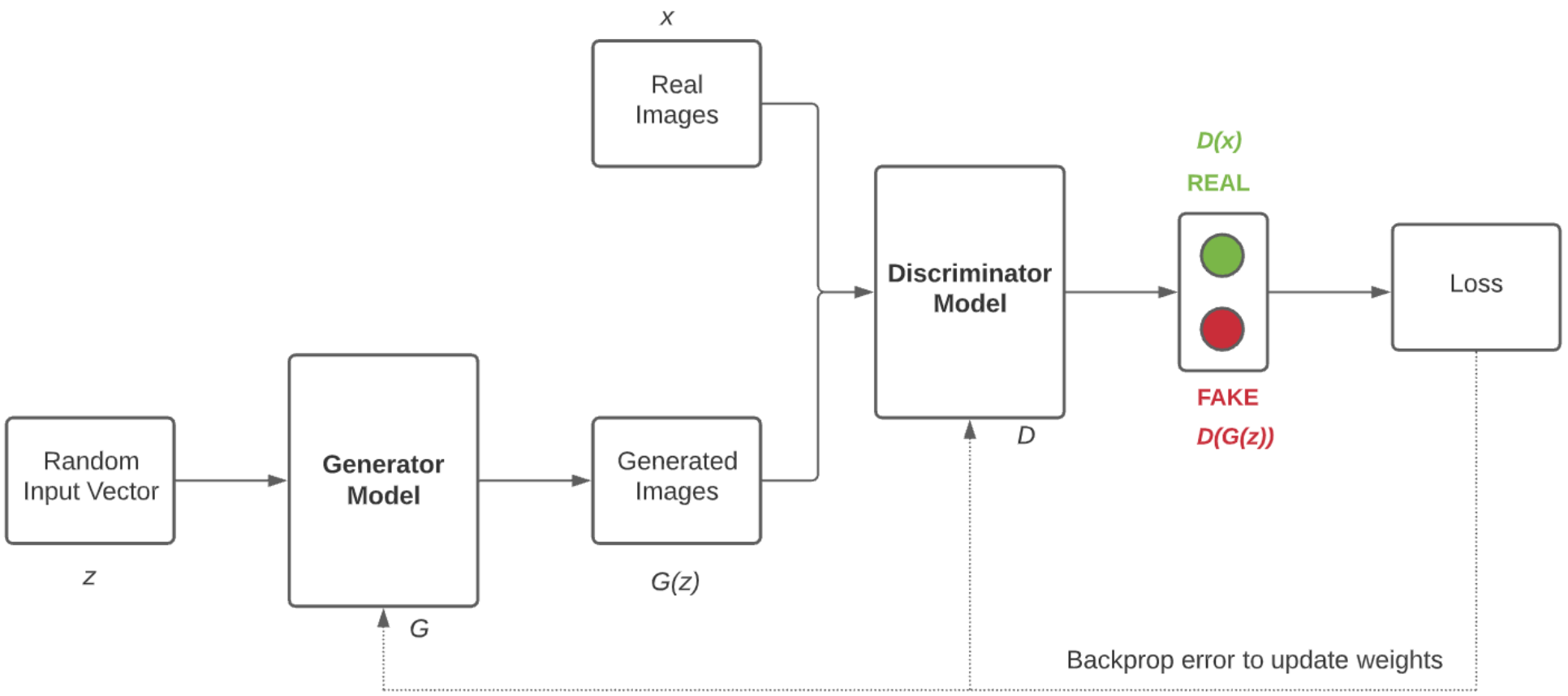
The Discriminator Model

The Discriminator Model takes an image as an input (generated and real) and classifies it as real or fake.

Generated images come from the Generator and the real images come from the training data.

The discriminator model is the simple binary classification model.

Now, let us combine both the architectures and understand them in detail.



The Generator Model *G* takes a random input vector *z* as an input and generates the images *G(z)*. These generated images along with the real images *x* from training data are then fed to the Discriminator Model *D*. The Discriminator Model then classifies the images as real or fake. Then, we have to measure the loss and this loss has to be back propagated to update the weights of the Generator and the Discriminator.

When we are training the Discriminator, we have to freeze the Generator and back propagate errors to only update the Discriminator.

When we are training the Generator, we have to freeze the Discriminator and back propagate errors to only update the Generator.

Thus the Generator Model and the Discriminator Model getting better and better at each epoch.

We have to stop training when it attains the Nash Equilibrium or *D(x) = 0.5 for all x*. In simple words, **when the generated images look almost like real images.**

Let us introduce some notations to understand the loss function of the GANs.

G	Generator Model
D	Discriminator Model
z	Random Noise (Fixed size input vector)
x	Real Image

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras

D(x) Discriminator's output when the real image is an input

The fight between the Generator Model and the Discriminator Model can be expressed mathematically as:

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

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

Note: The term $E_{x\sim pdata(x)}$ *[log D(x)] can be read as* ***E of log(D(x)) when x is sampled from*** p_{data}***(x)*** *and similar for the second term.*

As we can see in the equation, the Generator wants to minimize the *V(D, G)* whereas the Discriminator wants to maximize the *V(D, G)*. Let us understand both terms:

1. $E_{x\sim pdata(x)}$ [log D(x)]: Average log probability of D when real image is input.
2. $E_{z\sim pz(z)}$ [log(1 – D(G(z)))]: Average log probability of D when the generated image is input.

Let us understand the equation by thinking from the Generator's and the Discriminator's perspectives separately.

Discriminator's perspective

The Discriminator wants to maximize the loss function *V(D, G)* by correctly classifying real and fake images.

The first term suggests that the Discriminator wants to make D(x) as close to 1 as possible, i.e. correctly classifying real images as real.

The second term suggests that the Discriminator wants to make D(G(x)) as close to 0 as possible, i.e. correctly classifying fake images as fake and thus maximize the term eventually (1 – smaller number will result in a larger number). *Note: Probability lies in the range of 0-1.*

Thus, The Discriminator tries to maximize both the terms.

Generator's perspective

The Generator wants to minimize the loss function *V(D, G)* by generating images that look like real images and tries to fool the Discriminator.

The second term suggests that the Generator wants to make D(G(z)) as close to 1 (instead of 0) as possible and thus minimize the term eventually (1 – larger number will result in a smaller number). So that the Discriminator fails and **misclassifies** the images.

Thus, The Generator tries to minimize the second term.

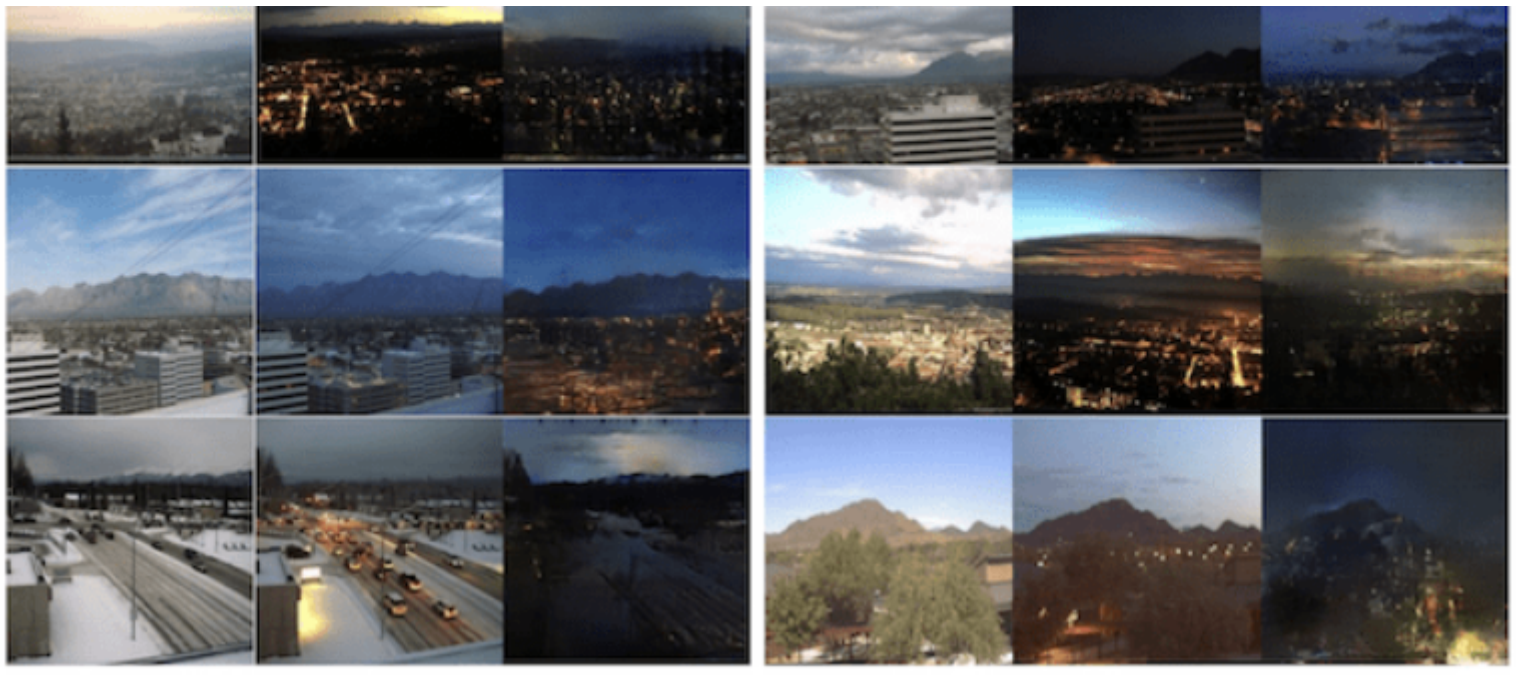
Amazing Applications of GAN

Let us discuss some amazing applications of GANs other than image generation.

Image to Image Translation

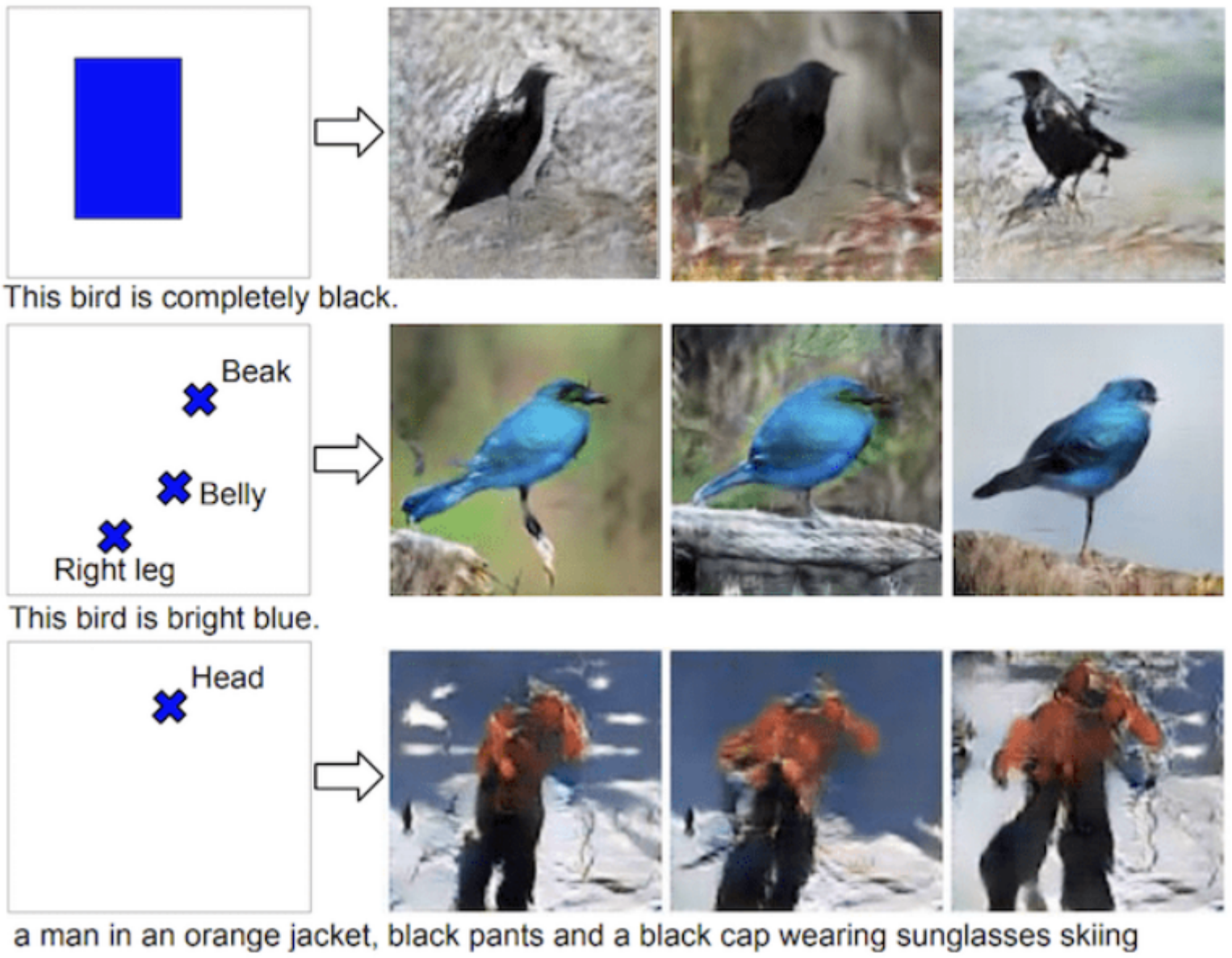
Phillip Isola, et al. in this paper demonstrates GANs as many images to image translation tasks.

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras



Text to Image Translation

Scott Reed, et al. in [this] paper, demonstrates a way to generate images from text.



Photos to Emojis

Yaniv Taigman, et al. in [this] paper used GANs to translate photos to emojis.

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras

There are many more applications of GAN such as Image Editing, Face Aging, 3D Object Generation, etc.

Implementing a Toy GAN

So, Now we've got a clear idea about the GANs. Let's start implementing it using **Tensorflow** and **Keras.**

We'll begin by Importing Necessary Libraries, considering you've installed all the necessary libraries already.

Importing Libraries

```
from numpy import zeros, ones, expand_dims, asarray
from numpy.random import randn, randint
from keras.datasets import fashion_mnist
from keras.optimizers import Adam
from keras.models import Model, load_model
from keras.layers import Input, Dense, Reshape, Flatten
from keras.layers import Conv2D, Conv2DTranspose, Concatenate
from keras.layers import LeakyReLU, Dropout, Embedding
from keras.layers import BatchNormalization, Activation
from keras import initializers
from keras.initializers import RandomNormal
from keras.optimizers import Adam, RMSprop, SGD
from matplotlib import pyplot
import numpy as np
from math import sqrt
```

Loading Datasets

```
(X_train, _), (_, _) = fashion_mnist.load_data()
X_train = X_train.astype(np.float32) / 127.5 - 1
X_train = np.expand_dims(X_train, axis=3)
print(X_train.shape)
```

We are only loading the features of train data as we do not require the labels. Then we are dividing each pixel value by 127.5 and subtracting it from 1 to have pixel values in the range of -1 to 1. Finally, the X_train shape is (60000, 28, 28, 1).

Some Necessary Functions

```
def generate_latent_points(latent_dim, n_samples):
    x_input = randn(latent_dim * n_samples)
```

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras

```
    ix = randint(0, X_train.shape[0], n_samples)
    X = X_train[ix]
    y = ones((n_samples, 1))
    return X, y
```

The above function helps us to generate *n* real samples with 1 as a label, i.e. real image.

```
def generate_fake_samples(generator, latent_dim, n_samples):
    z_input = generate_latent_points(latent_dim, n_samples)
    images = generator.predict(z_input)
    y = zeros((n_samples, 1))
    return images, y
```

The above function helps us to generate *n* fake samples using the generator with 0 as a label, i.e. fake image.

```
def summarize_performance(step, g_model, latent_dim, n_samples=100):
    X, _ = generate_fake_samples(g_model, latent_dim, n_samples)
    X = (X + 1) / 2.0
    for i in range(100):
        pyplot.subplot(10, 10, 1 + i)
        pyplot.axis('off')
        pyplot.imshow(X[i, :, :, 0], cmap='gray_r')
    filename2 = 'model_%04d.h5' % (step+1)
    g_model.save(filename2)
    print('>Saved: %s' % (filename2))
```

This function helps us to summarize the performance. This includes generating a fake sample, plotting it, and finally saving the model.

```
def save_plot(examples, n_examples):
    for i in range(n_examples):
        pyplot.subplot(sqrt(n_examples), sqrt(n_examples), 1 + i)
        pyplot.axis('off')
        pyplot.imshow(examples[i, :, :, 0], cmap='gray_r')
    pyplot.show()
```

The above function helps us to plot the results. We'll use this to plot the generated images by the Generator in later stages.

Model Building

```
def define_discriminator(in_shape=(28, 28, 1)):
    init = RandomNormal(stddev=0.02)
    in_image = Input(shape=in_shape)
    fe = Flatten()(in_image)
    fe = Dense(1024)(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.3)(fe)
    fe = Dense(512)(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.3)(fe)
    fe = Dense(256)(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.3)(fe)
    out = Dense(1, activation='sigmoid')(fe)
    model = Model(in_image, out)
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
```

```
discriminator = define_discriminator()
```

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras

binary classification.

```
def define_generator(latent_dim):
    init = RandomNormal(stddev=0.02)
    in_lat = Input(shape=(latent_dim,))
    gen = Dense(256, kernel_initializer=init)(in_lat)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Dense(512, kernel_initializer=init)(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Dense(1024, kernel_initializer=init)(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Dense(28 * 28 * 1, kernel_initializer=init)(gen)
    out_layer = Activation('tanh')(gen)
    out_layer = Reshape((28, 28, 1))(gen)
    model = Model(in_lat, out_layer)
    return model
```

```
generator = define_generator(100)
```

We are using a couple of Dense layers to define the generator model with again leaky relu as an activation function in hidden layers and tanh in the final layer. The generated images *G(z)* will be of the shape 28x28x1.

```
def define_gan(g_model, d_model):
    d_model.trainable = False
    gan_output = d_model(g_model.output)
    model = Model(g_model.input, gan_output)
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
```

```
gan_model = define_gan(generator, discriminator)
```

We are freezing the discriminator, providing *z* as input and *D(G(z))* as an output to our model. We are using adam as an optimizer and binary cross-entropy as a loss function.

Model Training

```
def train(g_model, d_model, gan_model, X_train, latent_dim, n_epochs=100, n_batch=64):
    bat_per_epo = int(X_train.shape[0] / n_batch)
    n_steps = bat_per_epo * n_epochs
    for i in range(n_steps):
        X_real, y_real = generate_real_samples(X_train, n_batch)
        d_loss_r, d_acc_r = d_model.train_on_batch(X_real, y_real)
        X_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_batch)
        d_loss_f, d_acc_f = d_model.train_on_batch(X_fake, y_fake)
        z_input = generate_latent_points(latent_dim, n_batch)
        y_gan = ones((n_batch, 1))
        g_loss, g_acc = gan_model.train_on_batch(z_input, y_gan)
        print('>%d, dr[%.3f,%.3f], df[%.3f,%.3f], g[%.3f,%.3f]' % (i+1, d_loss_r,d_acc_r, d_loss_f,d_acc_f,
g_loss,g_acc))
        if (i+1) % (bat_per_epo * 1) == 0:
            summarize_performance(i, g_model, latent_dim)
```

This function helps us to train the generator and the discriminator. To train the Discriminator, it first generates real samples, updates the discriminator's weights, generates fake samples, and then updates the discriminator's weights again. To train the Generator, it first generates latent points, **generates labels as 1 to fool the discriminator,** and then updates the generator's weights. Finally, the function summarizes the performance of the model after some steps.

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras

Generating Samples Using GAN

```
model = load_model('model_18740.h5')
latent_dim = 100
n_examples = 100
latent_points = generate_latent_points(latent_dim, n_examples)
X  = model.predict(latent_points)
X = (X + 1) / 2.0
save_plot(X, n_examples)
```

We are just loading the latest saved model, generating latent points, using the loaded model for prediction, and plotting the results.

Generated Images

The generated images aren't quite clear, right? Because we haven't used Convolution layers in our model. Try it on your own and see the results.

Summary

In this post, you have understood the GANs in detail.

Specifically, you learned:

- Generative Adversarial Networks in depth
- How to Implement GANs using Keras
- Some Amazing Applications of GANs

Do you have any questions? Feel free to post your questions in the comments below. I would love to help you.

Connect with me on Twitter and LinkedIn.

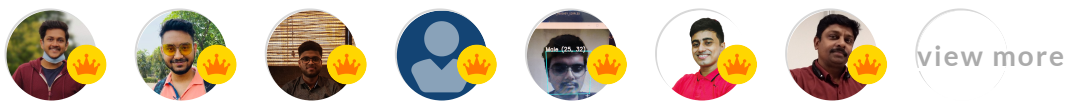
The media shown in this article are not owned by Analytics Vidhya and are used at the Author's discretion.

blogathon deep learning GANs python tensorflow

About the Author

Harsh Dhamecha

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras



Previous Post

A comprehensive tutorial on Deep Learning – Part 2

Next Post

Machine Learning vs Deep Learning vs Artificial Intelligence| Know in-depth Difference

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

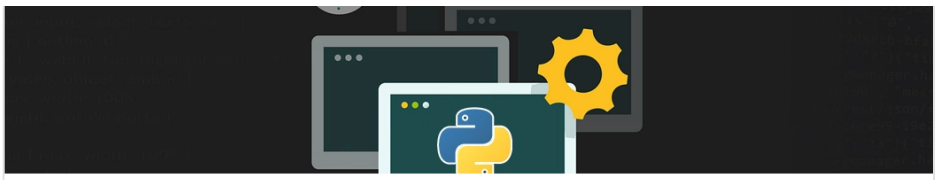
Notify me of follow-up comments by email.

Notify me of new posts by email.

Submit

Top Resources

A Detailed Explanation of GAN with Implementation Using Tensorflow and Keras



How to Read and Write With CSV Files in Python:..

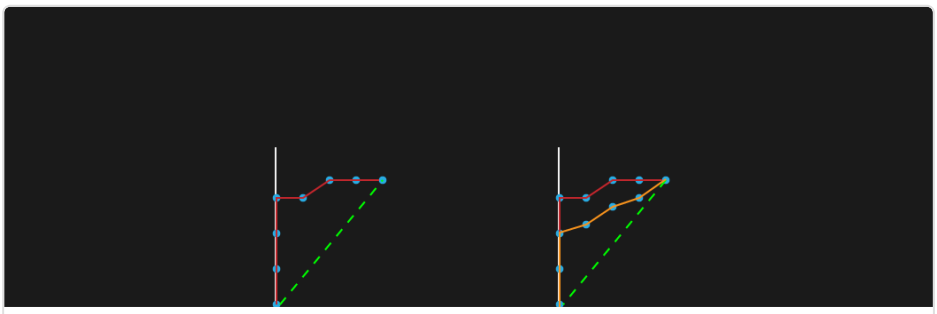
Harika Bonthu - AUG 21, 2021

Feature Selection Techniques in Machine Learning (Updated 2023)

Aman Gupta - OCT 10, 2020

Understand Random Forest Algorithms With Examples (Updated 2023)

Sruthi E R - JUN 17, 2021



Guide to AUC ROC Curve in Machine Learning : What..

Aniruddha Bhandari - JUN 16, 2020