Generative Adversarial Networks with Python

7-Day Crash-Course

Jason Brownlee

MACHINE LEARNING MASTERY

Disclaimer

Generative Adversarial Networks with Python Crash Course

Edition: v1.5

Find the latest version of this guide online at: http://MachineLearningMastery.com

Contents

Before We Get Started...

Generative Adversarial Networks, or GANs for short, are a deep learning technique for training generative models. The study and application of GANs are only a few years old, yet the results achieved have been nothing short of remarkable. Because the field is so young, it can be challenging to know how to get started, what to focus on, and how to best use the available techniques. In this crash course, you will discover how you can get started and confidently develop deep learning Generative Adversarial Networks using Python in seven days. Let's get started.

Who Is This Crash-Course For?

Before we get started, let's make sure you are in the right place. The list below provides some general guidelines as to who this course was designed for.

You need to know:

- Your way around basic Python, NumPy, and Keras for deep learning.

You do NOT need to be::

- A math wiz!
- A deep learning expert!
- A computer vision researcher!

This crash course will take you from a developer that knows a little machine learning to a developer who can bring GANs to your own computer vision project. This crash course assumes you have a working Python 3 SciPy environment with at least NumPy and Keras 2 installed. If you need help with your environment, you can follow the step-by-step tutorial here:

- How to Setup a Python Environment for Machine Learning and Deep Learning.

Crash-Course Overview

This crash course is broken down into seven lessons. You could complete one lesson per day (recommended) or complete all of the lessons in one day (hardcore). It really depends on the time you have available and your level of enthusiasm. Below are the seven lessons that will get you started and productive with Generative Adversarial Networks in Python:

- **Lesson 01**: What Are Generative Adversarial Networks?
- **Lesson 02**: GAN Tips, Tricks, and Hacks.
- **Lesson 03**: Discriminator and Generator Models.
- **Lesson 04**: GAN Loss Functions.
- **Lesson 05**: GAN Training Algorithm.
- **Lesson 06**: GANs for Image Translation.
- **Lesson 07**: Advanced GANs.

Each lesson could take you 60 seconds or up to 30 minutes. Take your time and complete the lessons at your own pace. The lessons expect you to go off and find out how to do things. I will give you hints, but part of the point of each lesson is to force you to learn where to go to look for help (hint, I have all of the answers directly on this blog; use the search box). I do provide more help in the form of links to related posts because I want you to build up some confidence and inertia. Post your results online, I'll cheer you on!

Hang in there, don't give up!

Lesson 01: What Are Generative Adversarial Networks?

In this lesson, you will discover what GANs are and the basic model architecture. Generative Adversarial Networks, or GANs for short, are an approach to generative modeling using deep learning methods, such as convolutional neural networks. GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models: the generator model that we train to generate new examples, and the discriminator model that tries to classify examples as either real (from the domain) or fake (generated).

- **Generator**. Model that is used to generate new plausible examples from the problem domain.
- **Discriminator**. Model that is used to classify examples as real (from the domain) or fake (generated).

The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples.

The Generator

The generator model takes a fixed-length random vector as input and generates an image in the domain. The vector is drawn randomly from a Gaussian distribution (called the latent space), and the vector is used to seed the generative process. After training, the generator model is kept and used to generate new samples.

The Discriminator

The discriminator model takes an example from the domain as input (real or generated) and predicts a binary class label of real or fake (generated). The real example comes from the training dataset. The generated examples are output by the generator model. The discriminator is a normal (and well understood) classification model. After the training process, the discriminator model is discarded as we are interested in the generator.

GAN Training

The two models, the generator and discriminator, are trained together. A single training cycle involves first selecting a batch of real images from the problem domain. A batch of latent points is generated and fed to the generator model to synthesize a batch of images. The discriminator is then updated using the batch of real and generated images, minimizing binary cross-entropy loss used in any binary classification problem. The generator is then updated via the discriminator model. This means that generated images are presented to the discriminator as though they are real (not generated) and the error is propagated back through the generator model. This has the effect of updating the generator model toward generating images that are more likely to fool the discriminator. This process is then repeated for a given number of training iterations.

Your Task

Your task in this lesson is to list three possible applications for Generative Adversarial Networks. You may get ideas from looking at recently published research papers. Post your findings online. I would love to see what you can come up with.

Next

In the next lesson, you will discover tips and tricks for the successful training of GAN models.

Lesson 02: GAN Tips, Tricks, and Hacks

In this lesson, you will discover the tips, tricks, and hacks that you need to know to successfully train GAN models. Generative Adversarial Networks are challenging to train. This is because the architecture involves both a generator and a discriminator model that compete in a zero-sum game. Improvements to one model come at the cost of a degrading of performance in the other model. The result is a very unstable training process that can often lead to failure, e.g. a generator that generates the same image all the time or generates nonsense. As such, there are a number of heuristics or best practices (called *GAN hacks*) that can be used when configuring and training your GAN models. Perhaps one of the most important steps forward in the design and training of stable GAN models is the approach that became known as the Deep Convolutional GAN, or DCGAN. This architecture involves seven best practices to consider when implementing your GAN model:

1. Downsample Using Strided Convolutions (e.g. don't use pooling layers).
2. Upsample Using Strided Convolutions (e.g. use the transpose convolutional layer).
3. Use `LeakyReLU` (e.g. don't use the standard ReLU).
4. Use `BatchNormalization` (e.g. standardize layer outputs after the activation).
5. Use Gaussian Weight Initialization (e.g. a mean of 0.0 and stdev of 0.02).
6. Use Adam Stochastic Gradient Descent (e.g. learning rate of 0.0002 and beta1 of 0.5).
7. Scale Images to the Range [-1,1] (e.g. use Tanh in the output of the generator).

These heuristics have been hard won by practitioners testing and evaluating hundreds or thousands of combinations of configuration operations on a range of problems.

Your Task

Your task in this lesson is to list three additional GAN tips or hacks that can be used during training. Post your findings online. I would love to see what you can come up with.

Next

In the next lesson, you will discover how to implement simple discriminator and generator models.

Lesson 03: Discriminator and Generator Models

In this lesson, you will discover how to implement a simple discriminator and generator model using the Keras deep learning library. We will assume the images in our domain are 28×28 pixels in size and color, meaning they have three color channels.

Discriminator Model

The discriminator model accepts an image with the with size $28 \times 28 \times 3$ pixels and must classify it as real (1) or fake (0) via the sigmoid activation function. Our model has two convolutional layers with 64 filters each and uses same padding. Each convolutional layer will downsample the input using a 2×2 stride, which is a best practice for GANs, instead of using a pooling layer. Also following best practice, the convolutional layers are followed by a `LeakyReLU` activation with a slope of 0.2 and a batch normalization layer.

```
...
# define the discriminator model
model = Sequential()
# downsample to 14x14
model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same', input_shape=(28,28,3)))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization())
# downsample to 7x7
model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization())
# classify
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

Listing 1: Example of defining the discriminator model.

Generator Model

The generator model takes a 100-dimensional point in the latent space as input and generates a $28 \times 28 \times 3$. The point in latent space is a vector of Gaussian random numbers. This is projected using a `Dense` layer to the basis of 64 tiny 7×7 images. The small images are then upsampled twice using two transpose convolutional layers with a 2×2 stride and followed by a

`LeakyReLU` and `BatchNormalization` layers, which are a best practice for GANs. The output is a three channel image with pixel values in the range [-1,1] via the Tanh activation function.

```
...
# define the generator model
model = Sequential()
# foundation for 7x7 image
n_nodes = 64 * 7 * 7
model.add(Dense(n_nodes, input_dim=100))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization())
model.add(Reshape((7, 7, 64)))
# upsample to 14x14
model.add(Conv2DTranspose(64, (3,3), strides=(2,2), padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization())
# upsample to 28x28
model.add(Conv2DTranspose(64, (3,3), strides=(2,2), padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization())
model.add(Conv2D(3, (3,3), activation='tanh', padding='same'))
```

Listing 2: Example of defining the generator model.

Your Task

Your task in this lesson is to implement both the discriminator models and summarize their structure. For bonus points, update the models to support an image with the size 64×64 pixels. Post your findings online. I would love to see what you can come up with.

Next

In the next lesson, you will discover how to configure the loss functions for training the GAN models.

Lesson 04: GAN Loss Functions

In this lesson, you will discover how to configure the loss functions used for training the GAN model weights.

Discriminator Loss

The discriminator model is optimized to maximize the probability of correctly identifying real images from the dataset and fake or synthetic images output by the generator. This can be implemented as a binary classification problem where the discriminator outputs a probability for a given image between 0 and 1 for fake and real respectively. The model can then be trained on batches of real and fake images directly and minimize the negative log likelihood, most commonly implemented as the binary cross-entropy loss function. As is the best practice, the model can be optimized using the Adam version of stochastic gradient descent with a small learning rate and conservative momentum.

```
...
# compile model
model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002, beta_1=0.5))
```

Listing 3: Example of compiling the discriminator model.

Generator Loss

The generator is not updated directly and there is no loss for this model. Instead, the discriminator is used to provide a learned or indirect loss function for the generator. This is achieved by creating a composite model where the generator outputs an image that feeds directly into the discriminator for classification. The composite model can then be trained by providing random points in latent space as input and indicating to the discriminator that the generated images are, in fact, real. This has the effect of updating the weights of the generator to output images that are more likely to be classified as real by the discriminator. Importantly, the discriminator weights are not updated during this process and are marked as not trainable. The composite model uses the same categorical cross-entropy loss as the standalone discriminator model and the same Adam version of stochastic gradient descent to perform the optimization.

```
# create the composite model for training the generator
generator = ...
discriminator = ...
...
# make weights in the discriminator not trainable
d_model.trainable = False
```

```
# connect them
model = Sequential()
# add generator
model.add(generator)
# add the discriminator
model.add(discriminator)
# compile model
model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002, beta_1=0.5))
```

Listing 4: Example of defining the composite model for training the generator via the discriminator.

Your Task

Your task in this lesson is to research and summarize three additional types of loss function that can be used to train the GAN models. Post your findings online. I would love to see what you can come up with.

Next

In the next lesson, you will discover the training algorithm used to update the model weights for the GAN.

Lesson 05: GAN Training Algorithm

In this lesson, you will discover the GAN training algorithm. Defining the GAN models is the hard part. The GAN training algorithm is relatively straightforward. One cycle of the algorithm involves first selecting a batch of real images and using the current generator model to generate a batch of fake images. You can develop small functions to perform these two operations. These real and fake images are then used to update the discriminator model directly via a call to the `train_on_batch()` Keras function. Next, points in latent space can be generated as input for the composite generator-discriminator model and labels of *real* ($class = 1$) can be provided to update the weights of the generator model. The training process is then repeated thousands of times. The generator model can be saved periodically and later loaded to check the quality of the generated images. The example below demonstrates the GAN training algorithm.

```
...
# gan training algorithm
discriminator = ...
generator = ...
gan_model = ...
n_batch = 16
latent_dim = 100
for i in range(10000)
  # get randomly selected 'real' samples
  X_real, y_real = select_real_samples(dataset, n_batch)
  # generate 'fake' examples
  X_fake, y_fake = generate_fake_samples(generator, latent_dim, n_batch)
  # create training set for the discriminator
  X, y = vstack((X_real, X_fake)), vstack((y_real, y_fake))
  # update discriminator model weights
  d_loss = discriminator.train_on_batch(X, y)
  # prepare points in latent space as input for the generator
  X_gan = generate_latent_points(latent_dim, n_batch)
  # create inverted labels for the fake samples
  y_gan = ones((n_batch, 1))
  # update the generator via the discriminator's error
  g_loss = gan_model.train_on_batch(X_gan, y_gan)
```

Listing 5: Example of the GAN training algorithm.

Your Task

Your task in this lesson is to tie together the elements from this and the prior lessons and train a GAN on a small image dataset such as MNIST or CIFAR-10. Post your findings online. I would love to see what you can come up with.

Next

In the next lesson, you will discover the application of GANs for image translation.

Lesson 06: GANs for Image Translation

In this lesson, you will discover GANs used for image translation. Image-to-image translation is the controlled conversion of a given source image to a target image. An example might be the conversion of black and white photographs to color photographs. Image-to-image translation is a challenging problem and often requires specialized models and loss functions for a given translation task or dataset. GANs can be trained to perform image-to-image translation and two examples include the Pix2Pix and the CycleGAN.

Pix2Pix

The Pix2Pix GAN is a general approach for image-to-image translation. The model is trained on a dataset of paired examples, where each pair involves an example of the image before and after the desired translation. The Pix2Pix model is based on the conditional generative adversarial network, where a target image is generated, conditional on a given input image. The discriminator model is given an input image and a real or generated paired image and must determine whether the paired image is real or fake. The generator model is provided with a given image as input and generates a translated version of the image. The generator model is trained to both fool the discriminator model and to minimize the loss between the generated image and the expected target image. More sophisticated deep convolutional neural network models are used in the Pix2Pix. Specifically, a U-Net model is used for the generator model and a PatchGAN is used for the discriminator model. The loss for the generator is comprised of a composite of both the adversarial loss of a normal GAN model and the L1 loss between the generated and expected translated image.

CycleGAN

A limitation of the Pix2Pix model is that it requires a dataset of paired examples before and after the desired translation. There are many image-to-image translation tasks where we may not have examples of the translation, such as translating photos of zebra to horses. There are other image translation tasks where such paired examples do not exist, such as translating art of landscapes to photographs. The CycleGAN is a technique that involves the automatic training of image-to-image translation models without paired examples. The models are trained in an unsupervised manner using a collection of images from the source and target domain that do not need to be related in any way. The CycleGAN is an extension of the GAN architecture that involves the simultaneous training of two generator models and two discriminator models.

One generator takes images from the first domain as input and outputs images for the second domain, and the other generator takes images from the second domain as input and generates images from the first domain. Discriminator models are then used to determine how plausible the generated images are and update the generator models accordingly. The CycleGAN uses an additional extension to the architecture called cycle consistency. This is the idea that an image output by the first generator could be used as input to the second generator and the output of the second generator should match the original image. The reverse is also true: that an output from the second generator can be fed as input to the first generator and the result should match the input to the second generator.

Your Task

Your task in this lesson is to list five examples of image-to-image translation you might like to explore with GAN models. Post your findings online. I would love to see what you can come up with.

Next

In the next lesson, you will discover some of the recent advancements in GAN models.

Lesson 07: Advanced GANs

In this lesson, you will discover some of the more advanced GAN that are demonstrating remarkable results.

BigGAN

The BigGAN is an approach to pull together a suite of recent best practices in training GANs and scaling up the batch size and number of model parameters. As its name suggests, the BigGAN is focused on scaling up the GAN models. This includes GAN models with:

- More model parameters (e.g. many more feature maps).
- Larger Batch Sizes (e.g. hundreds or thousands of images).
- Architectural changes (e.g. self-attention modules).

The resulting BigGAN generator model is capable of generating high-quality 256×256 and 512×512 images across a wide range of image classes.

Progressive Growing GAN

Progressive Growing GAN is an extension to the GAN training process that allows for the stable training of generator models that can output large high-quality images. It involves starting with a very small image and incrementally adding blocks of layers that increase the output size of the generator model and the input size of the discriminator model until the desired image size is achieved. Perhaps the most impressive accomplishment of the Progressive Growing GAN is the generation of large 1024×1024 pixel photorealistic generated faces.

StyleGAN

The Style Generative Adversarial Network, or StyleGAN for short, is an extension to the GAN architecture that proposes large changes to the generator model. This includes the use of a mapping network to map points in latent space to an intermediate latent space, the use of the intermediate latent space to control style at each point in the generator model, and the introduction to noise as a source of variation at each point in the generator model. The resulting model is capable not only of generating impressively photorealistic high-quality photos of faces, but also offers control over the style of the generated image at different levels of detail through

varying the style vectors and noise. For example, blocks of layers in the synthesis network at lower resolutions control high-level styles such as pose and hairstyle, blocks at higher resolutions control color schemes and very fine details like freckles and placement of hair strands.

Your Task

Your task in this lesson is to list 3 examples of how you might use models capable of generating large photorealistic images. Post your findings online. I would love to see what you can come up with.

Next

This was the final lesson.

Final Word Before You Go...

You made it. Well done! Take a moment and look back at how far you have come. You discovered:

- GANs are a deep learning technique for training generative models capable of synthesizing high-quality images.
- Training GANs is inherently unstable and prone to failures, which can be overcome by adopting best practice heuristics in the design, configuration, and training of GAN models.
- Generator and discriminator models used in the GAN architecture can be defined simply and directly in the Keras deep learning library.
- The discriminator model is trained like any other binary classification deep learning model.
- The generator model is trained via the discriminator model in a composite model architecture.
- GANs are capable of conditional image generation, such as image-to-image translation with paired and unpaired examples.
- Advancements in GANs, such as scaling up the models and progressively growing the models, allows for the generation of larger and higher-quality images.

This is just the beginning of your journey with Generative Adversarial Networks. Keep practicing and developing your skills. Take the next step and check out my book on Generative Adversarial Networks with Python.

How Did You Go With The Crash-Course?

Did you enjoy this crash-course?
Do you have any questions or sticking points?

Let me know, send me an email at: **jason@MachineLearningMastery.com**

Take the Next Step

Looking for more help with Generative Adversarial Networks?

Grab my new book:
Generative Adversarial Networks with Python
https://machinelearningmastery.com/generative_adversarial_networks/