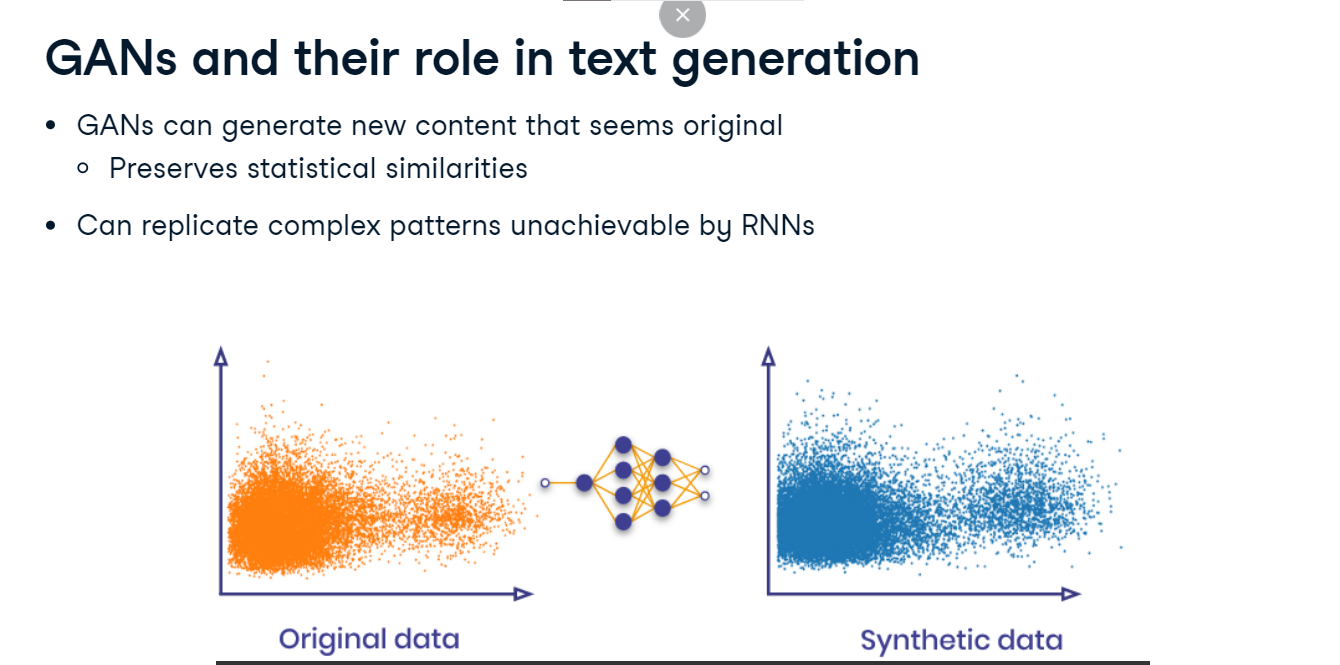
**Generative adversarial networks for text generation**

Generative Adversarial Networks, or GANs, are often used for image generation

**GANs and their role in text generation**

but are becoming more common for text generation for creating synthetic data that preserves statistical similarities. Unlike RNNs, GANs replicate complex data patterns, ensuring feature correlation and authentically emulating real-world patterns.



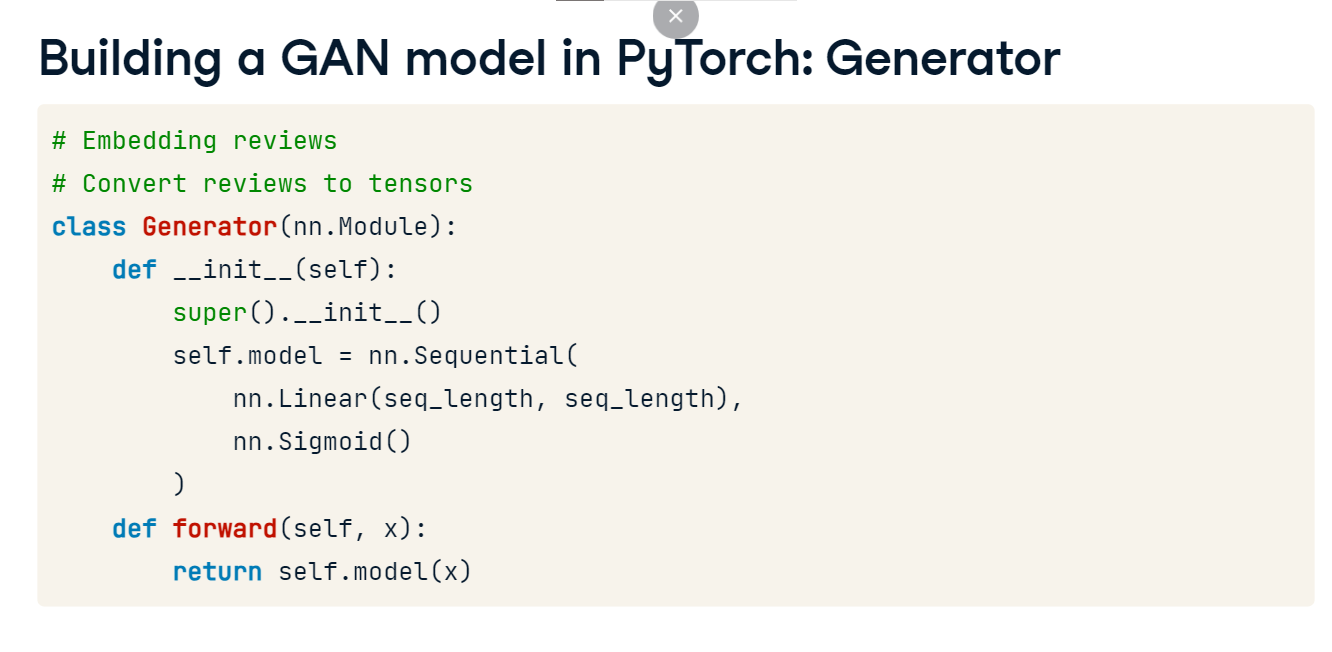
**Structure of a GAN**

A GAN consists of two primary components: the Generator, which creates synthetic text data from noise, and the Discriminator, which distinguishes between real and generated text data. Here, noise refers to random changes to real data, such as adding special characters to a word. These components collaborate, with the Generator improving its fakes and the Discriminator enhancing its ability to detect them until the generated text becomes indistinguishable from real text.

1. 1 <https://www.sciencefocus.com/future-technology/how-do-machine-learning-gans-work/>

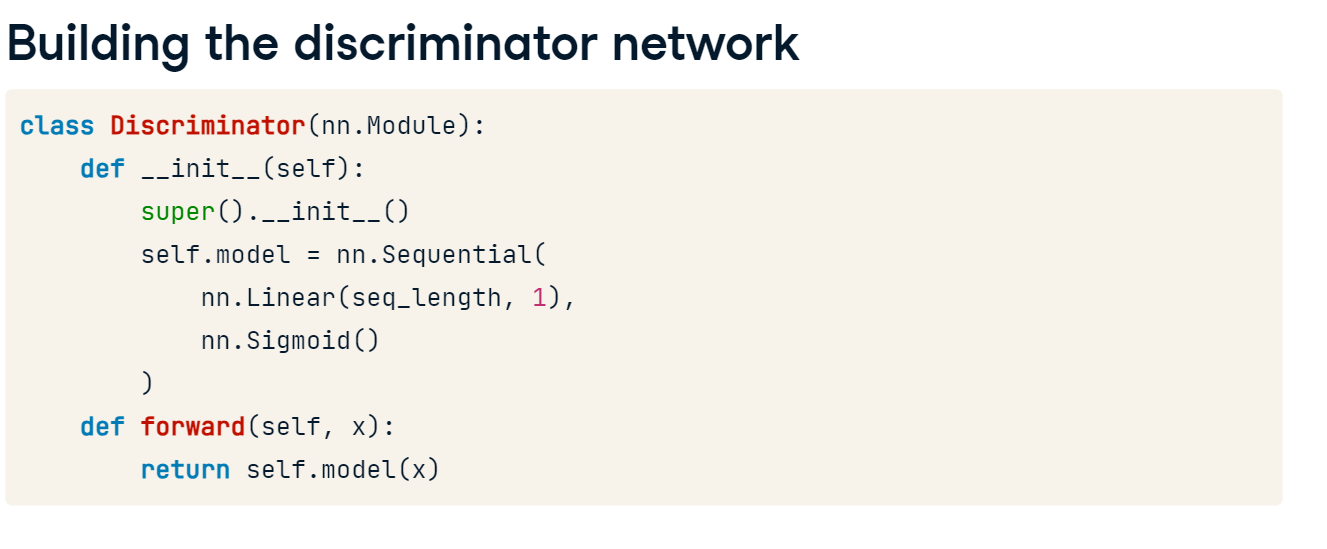
**Building a GAN model in PyTorch: Generator**

We begin building a GAN model by defining the Generator. Our data is product reviews that have been embedded and converted to tensors, not shown here for brevity. The goal is for our model to create believable reviews. We define our Generator network with nn-dot-Module. It has a linear layer inside the Sequential function that transforms the input to have the same dimension as our data sequences. It is followed by a sigmoid activation function suitable for binary data that squashes the output values to the range zero to one. The forward method then applies this network to an input tensor.



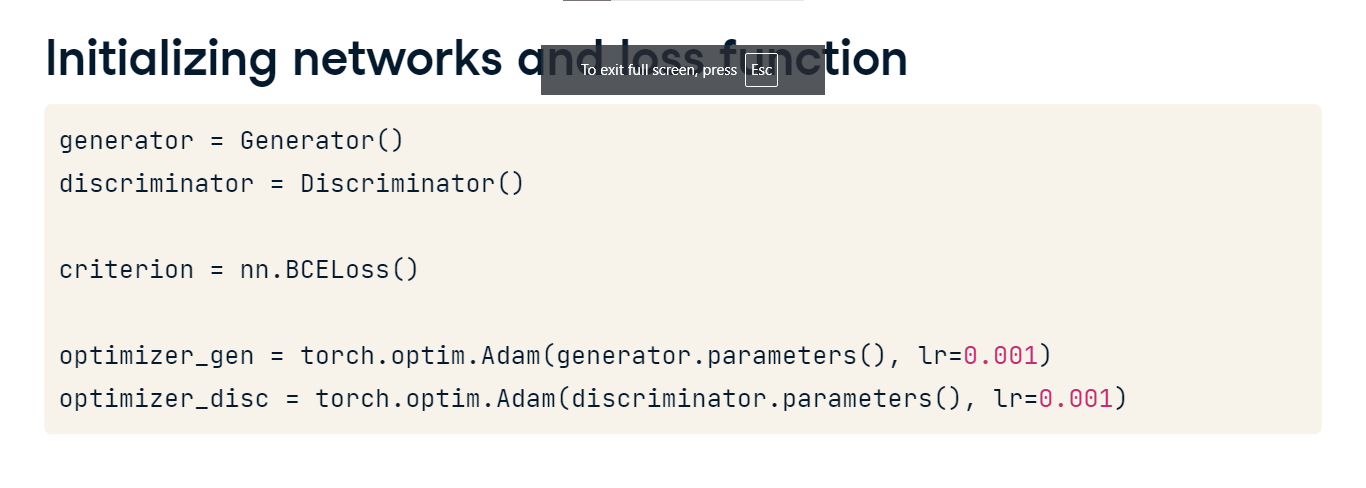
**Building the discriminator network**

We define a Discriminator network similarly. This network has a linear layer that transforms the input to a single value, followed by a sigmoid activation function. The output represents the probability that the input data is real. The forward method applies this network to an input tensor.



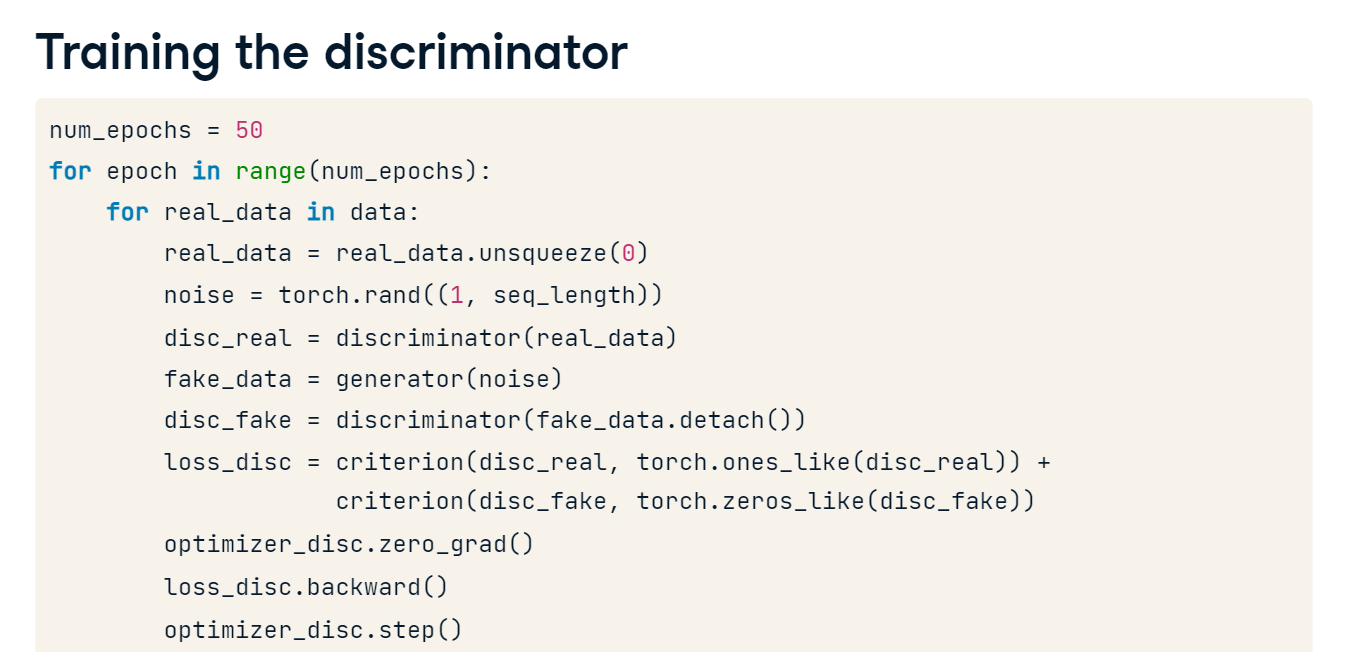
**Initializing networks and loss function**

We initialize our Generator and Discriminator network instances and define the loss function as Binary Cross Entropy for binary classification tasks like distinguishing between real and fake data. Next, we create two Adam optimizers for the Generator and the Discriminator. Each optimizer has a learning rate 0-point-001, a value often used as a starting point and may be adjusted based on model performance.



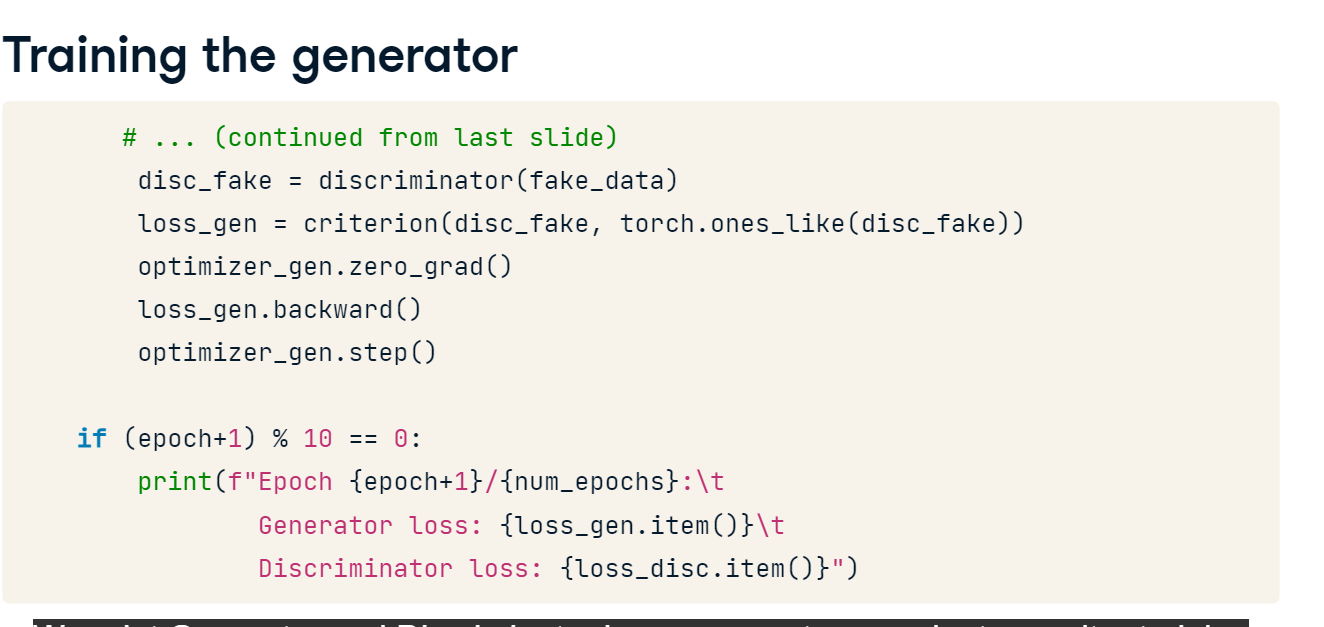
**Training the discriminator**

We establish a training loop for 50 epochs, generating batches of real data and random noise for the Generator to create fake data. We obtain predictions from the Discriminator for real and fake data, using the detach function to prevent gradient tracking. The Discriminator's loss is calculated using torch-dot-ones\_like and torch-dot-zeros\_like to match the expected real and fake labels. We reset the gradients in the Discriminator's optimizer with zero\_grad, perform backpropagation to calculate gradients, and update the Discriminator's parameters.



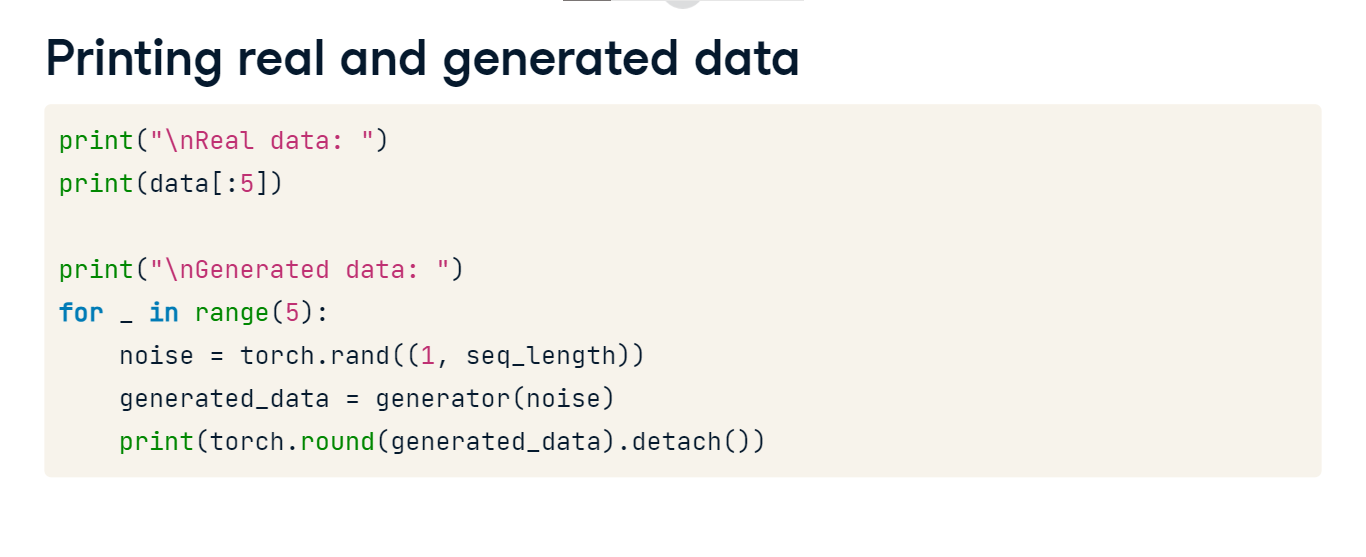
**Training the generator**

Next we train the Generator. We calculate the Generator's loss based on how well it fooled the Discriminator. The loss is determined by the difference between the Discriminator's predictions on fake data and an array of ones. We then reset the gradients in the Generator's optimizer, perform backpropagation to calculate gradients, and update the Generator's parameters. We print Generator and Discriminator losses every ten epochs to monitor training progress.



**Printing real and generated data**

After the training is complete, we print some real data. Then, we sample random values to form inputs for the Generator, generating data points mirroring the real data distribution.



**GANs: generated synthetic data**

The displayed output reveals Generator and Discriminator losses for every 10th epoch, demonstrating a consistent decline. However, after 50 epochs, the losses remain high, indicating the need for further training.

**Generated data**

Here's what our model generated. Since the input data was in tensor form, the output is also in tensor format. Upon reviewing the matrix, the real and generated data are similar. In practice, we would assess this further by plotting a correlation matrix and checking if the correlation between columns is maintained.

