|| जय श्री राम ||

GANs

Youtube:
Nicholas Renotte
youtube.com/watch?v=AALBGpLbj6Q

Generative Adversarial Networks

Pre-requisite:
- Theory Knowledge GAN
- Basic
  - Python
  - Machine Learning

- Study GAN
  - Watch video → GAN by Nicholas Renotte.
    project & Implement it

Studying And building GAN

- Setting up Environment
- Building a Data pipeline
- Creating a generator and Discriminator
- Building a custom training loop.
- Generating new images.

Start {brief gothrough}

(1) We will Load Data

we use buitin Library called tensorflow-datasets to use fashion_mnist dataset.

Our each image is of 28x28x1.
width x height = 28
and RGB channel = 1 hence its a greyscale image. → values 0 to 255
0 → black, 255 → white

(2) Building Generator

we take random array of numbers and will provide it to generator model.

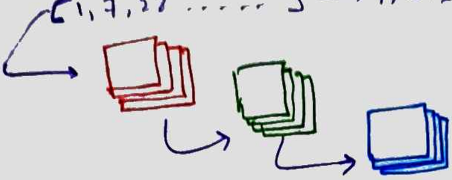
[This random array of number is our ~~voice~~ Noise]

Our generator is a CNN model
↓
Convonutional Neural Network

which will generate an matrix
↓
an image

[of size 28 × 28 × 1].

[1, 7, 22 .....] → noise



⇒ output image by **Generator**.

③ Discriminator.

Its also a CNN which will take input of image and has output 1 or 0.

1 ↓ False

0 ↓ True.

1 : (false image)
means discriminator is able to succesfully identify the false image.

0 : (True image)
discriminator is unable to identify the false image.

[NOTE: It may sound opposite because 0 is false & 1 is true
But, we are taking with respect to the aim of disiminator which is to discriminate/identify image.

→ generated image

1 : Success in identifying $\boxed{\text{false}}$ image

0 : Failure in identifying $\boxed{\text{False}}$ image]

This is key step for traing.

images by generator →
images by our dataset fashion-mnist from tensorflow →

$\boxed{\text{Discriminator CNN model}}$ → Output [0, 1] reward system

The reward system helps both generator and discriminator know when they are doing right and when they are doing wrong.

In the process.

1. image is generated
2. image is identified

- if image generated is identified by Discriminator
  ⇒ Discriminator is awarded
- if image generated is NOT identified by Discriminator
  ⇒ generator is awarded.

4. Final.

Once our training is Complete we need to pick our generator and test it.

Now our generator is able to generate New images when a set of variables is passed to it.

Lets Code!

- I am using python Notebook in VS Code. I have extension of Jupyter already installed once I created notebook for first time.

[You may use Jupyter directly] ;)

Our process goes like:

1. import dependencies and Data
2. visualise Data and Build Data set
3. Build Neural Networks.
   - generator & Discriminator.
4. Construct training Loop.
5. Test Generator & save model.

① Import dependencies / Important Libraries.

!pip install tensorflow tensorflow-gpu matplotlib
tensorflow-datasets ipywidgets.

in my case.

I installed them only on my python terminal like,

a. pip install tensorflow
b. pip install tensorflow_datasets
c. pip install matplotlib
d. pip install ipywidgets.

=> to check your all Libraries version type.

!pip List

=> we also configure gpu so that the memory is allocated efficiently when running our code.

→ Code is in .ipynb Notebook.

=> Data visualization

→ we see
- shape
- dimention
- values of images

This helps us get the idea of our data.

Now we will use Matplot lib to visualize our data.

— Data visualization is a good practice before building any model.
- we get the idea of datasets.

=> load the dataset [import tensorflow_dataset as tfds]

ds=tfds.load('Fashion_mnist', split='train')

Tensorflow dataset object

load data

Indicate we will split data to test & train.

- Inside dataset

a) get dictionary with first element

ds.as_numpy_iterator().next()

b) get Keys

ds.as_numpy_iterator().next().keys()

c) get an value matrix for an image

ds.as_numpy_iterator().next()['image'].

(2) visualizing Image building data pipeline.

a) getting matrix representation of each image.

data_iterator = ds.as_numpy_iterator()

Now

print(data_iterator.next())

Each time it will run it will output Next image.

b) output & showing showing images.

its = 12   # images to show

create subplots

Fig, ax = plt.subplots(ncols = its, figsize = (20,20))

Entire figure object

1D array of subplot objects

no of cols

Size of each image.

Code to plot

- iterate variable image whose range is 0 to its.

  # Store sample

  sample = data-iterator

  # Show image

  # Show label.

Code

```
for image in range(its):
    sample = data_iterator.next()

    ax[image].imshow(np.squeeze(sample['image']))

    ax[image].title.set_text(sample['label'])
```

datapoint iterator.

ax[image] — selects ith image

.imshow — funct to show image

np.squeeze — (28x28x1) to (28x28)

sample['image'] — returns image

Just to show label Number with the image of datapoint.

NOTE: we have declared data_iterator = ds.as_numpy_iterator()

above and we are running for loop for images many times. This is the reason Each time we get different set of images

To get Same Set of image each time Just write both part in same cell.

So in Step 2 we have done:

- Setup connection to data with iterator
- used numpy to squeeze data from (28x28x1) to (28x28) because '1' Just only told us 1 channel.
- Visualize data image on Subplot using matplot lib.

Data processing

right now these images are represented as values which are between 0 and 255.

In order to build good deep learning models we typically want to scale values to be between 0 and 1.

we will setup function to scale images.
- Better training
- Fast Calculation.

Code

```
def scale_images(sample):
    new image = ~~de~~ sample['image']
    return new_image / 255
```

This will scale our data in range 0 to 1 for image which is in form of matrix.

Setting for tensorflow

Following steps to build pipeline for tensorflow.

- map
- cache
- shuffle
- batch
- prefetch

These all operations are commonly used in building data pipeline.

A data pipeline is a series of steps or operations applied to a dataset to prepare data for consumption by machine learning model.

MES BP

Mountain ~~[illegible]~~ climb Sunsets Bring Peace.

Mountain Climb Sunsets
Brings peace ☮

Each process is explained:

Map: • Used to apply transformations to each element in dataset

• These transformations could include data preprocessing steps like normalization, augmentation, feature extraction.

In our case we did by scaling images.
- we prepared a function before now we will execute it in this step.

Cache: • It is an optimization technique used in data pipeline to store ~~from~~ intermediate results.

• By caching the dataset, we avoide redundant computations, especially useful for expensive preprocessing steps.

• This operation ensures that if the data needs to be reused multiple times.
It's readily available without recomputation.

Shuffle: • randomises the order of datapoints in dataset.

• prevents the model from learning any pattern, Co-relation or biases based on order of the data.

Batch: • Batching involves ~~pro~~ grouping the example in the dataset ~~ito~~ into smaller subsets called batches.

• Essential for efficient training.
→ On hardware like GPU, it allows the model to process multiple examples simultaneously.

Prefetch: • optimization technique used to overlap data preprocessing and executions.

• In simple शब्द already prepare batchs to stop waste time to maximize hardware utilization.

Build Neural Network

1) Importing modelling Component and important functions from Library.

2) Buiding Generator — CNN model.

3) Building Discriminator.

---

1) Importing Modelling Component

We will import;

- Sequential Api

* The Sequential API is one of the three ways to create a model in Keras. It is the simplest and most straightforward way to create a model, and is suitable for most problems.

* To create a sequential model, you simply add layers to it one by one. The layers are added in the order that you want them to be executed.

Importing additional function of layers.

These ~~are~~ layers are fundamental building block in Construction of GAN model for tasks Such as image generation from the Fashion_mnist dataset.

- Conv2D
- Flatten
- Dense
- Reshape
- LeakyReLU
- Droupout
- Upsampling2D

We will understand each layer function One by one:

1) Conv2D

This layer performs Convolutional operations on 2D input data (like images). It extracts features from the input using learnable filters (kernels).

2) Dense

This layer connects every neuron in the previous layer to every neuron in the current layer. It is typically used ~~as~~ at the beginning or end of the generator to map latent noise vectors (random inputs) into features map.

3) Flatten

This layer takes a multi-dimentional tensor (like a feature map) and reshapes it into a one-dimentional vector. In the generator, flatten is less common, as you usually want to preserve the spatial structure for image generation.

4) Reshape

This layer allows you to reshape the data into a specific desired shape. In the generator, Reshape could be used to transform a flattened vector from Dense layer into a feature Map suitable for Conv2D layers.

5) Leaky ReLU - Leaky Rectified Linear Unit

- This activation function introduces a small, non-zero slope for negative inputs, preventing dying Neurons

⇓

Neurons that never activate during training.

- It's often prefered over ReLU in GANs to maintain some gradient flow for negative values, which can be helpful for learning.

6) Dropout

This layer randomly drops out a certain percentage of neurons during training. This help prevent overfitting and encourages the network to learn more robust features that are not overly dependent on specific neurons.

7) UpSampling 2D

This layer increases the spatial resolution of the input by specific factor.

2) Building Generator (Code Explained)

- Input block.

  Converts random noise into a tensor with an initial shape of 7x7x128.

- Upsampling Block 1

  Upsamples the tensor to 14x14x128 and applies a Convolution.

- Upsampling Block 2.

  Further upsamples the tensor to 28x28x128 and applies another Convolution.

- Convolutional Block 1

  Applies a convolutional layer to add complexity.

- Convolution Block 2

  Applies a convolutional layer

- Output layer:

  Converts the tensor to a single channel image with pixel values between 0 & 1.

[IN SHORT]

This defines a generator part of GAN. It takes 128-dimention random Noise vector as input and progressively upsamples and processes it through Convolutional layers to produce a 28x28x1 image.

3) Building Discriminator

This is the Discriminator part of Generative Adversirial Network. The Job of discriminator is to take an image as input and Judge images real (from training loop) or fake.

[Layers used are Explained 2 pages back]

"The discriminator model processes an input image through four convolutional blocks, each followed by LeakyReLU activation and droupout to introduce non-linearity and prevent outfitting."

[After the convolutional layers, the model flattens the tensor and applies a final dense layer with a sigmoid activation to output a probability indicating whether the input image is real or fake.]

4) Constructing a training loop.

4.1) Setup loose and optimizers

4.2) Building SubClassed model

explained

1. Imports and initial setup

- imports necessary lib from tensorflow for optimizers, loss functions, and model building.

2. Custom Model Class

- Defines a custom 'fashionGAN' ~~with~~ class inheriting from Tensorflow's 'Model' class. This class will manage to train

- The_init_ method initializes the GAN with the generator and discriminator model.

- The 'Compile' method sets up the optimizers and loss function for both the generator and the discriminator.

3. Training Step

- The 'training step' method defines the training logic for one iteration.

- we will see how both of them work.

- Discriminator training

  a) Real images are taken from the dataset batch.

  b) Fake images are generated by generator.

  c) The discriminator is trained to distinguish between real & fake.

  d) Labels for real and fake are created, with some added noise for robustness.

  * e) The discriminator loss is computed and backpropogated.

- Generator Training.

  a) The Generator creates fake images.

  b) The discriminator predicts labels for there fake images.

  c) The Generator is trained to fool the discriminator into classifying fake images are real.

  * d) The Generator loss is computed ~~& backpropogated~~ and backpropogated

4. Installation and compilation

- An instance of the 'fashionGAN' class is created with the generator and discriminator.

- The model is then Compiled with the specified optimizers ad loss functions.

4.3 Build Callbacks

4.4 Train.

- Callback is to monitor and visualize the progress of the generator model during training.
- by Saving generated image at the end of the epoch it allows us to see how the quality of the generated image improves overtime.

* we the finally train our model.

- Review Performance

This Code creats the graphical view to visualize results.

- Testing Generator.

Finally seeing the new trained Generator the images it generate by its own.

This marks the End

Thank You
- APOORV SHARMA