Homework 09

IANNWTF 21/22

Submission until 16 Jar 23:59 via https://forms.gle/n6ERdhYx3uBPzuGn9

Welcome back to the 9th and last homework of year 2021 in IANNWTF. Don't get too excited, there will be a few more next year ;). This week you have learned about GANS. Perfect timing! In this homework, we will use your acquired skills to become candle makers to ensure you are ready for your holidays!

The goal of this week is to code a simple GAN and train it on an image dataset containing sketches - we will only focus on the category 'candle' in this homework. When training, you should visualize your training metrices as well as generate candle images using a random seed to assert the visual quality of your GAN. After training, you should be able to generate acceptable candle images - 3,4 or 5 depending on how far it is till Santa Claus is coming to gift you motivation for the new year.

This week we again try to keep the instructions as short as possible, so that you are challenged to try it yourself. Because the best way to learn something is to do it yourself. If you are lost, check the hints for advice or visit us during the Q&A Sessions (their purpose is to support you with any trouble you encounter, so really feel free to drop by even if you don't have any specific question and just need help with starting!).

1 Data set

This week, we will work with the Quick, Draw! Dataset. The Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw!.

We will download the data directly from google this time. As it is a little more complicated have a look in this Notebook - here you can find code to download the images into a NumPy array to work with. We will choose only one category here thus restricting ourselves to sketches of candles only, but feel free to toy around with that later on. You can also find the notebook as a .ipynb in the respective homework folder on courseware. Once you have done that, you should be able to build your data pipeline from that array as usual. Consider the following steps:

- Construct a tf.Data.Dataset object [1]
- The images come in (1,784) pixel arrays. You should make sure to reshape them correctly, obtaining little images. [2]
- Normalize: Bring the images to a range from -1 to 1.
- Perform other beneficial or necessary processing steps.[3]

2 Model

Your task is to implement a GAN.

Create a class **Discriminator**:

- `__init__(self)`: The underlying idea of the discriminator is to perform downsampling and outputs a vector of probabilities indicating wether the input was fake or real. [4]
- `call(self, x ,training`: This is just a simple call as you know it, but be aware of the training flag if you decide to use certain optimization and regularization techniques.[5]

Create a class **Generator**:

- `__init__(self)`: The generator takes a random point from the latent space[6] and returns a generated image. The latent space comes as a 1d vector, so to receive a 2d image you need to reshape it. To increase the size, use upsampling techniques like Conv2DTranspose [7][8]. In the last layer, perform a convolution with 1 feature map and tanh activation. This will be the output image.

- `call(self, x ,training`: Same as for the Discriminator.

3 Training

There is something novel this week: You will actually need to make some major adjustments to your `training_step()` function and the overall training procedure.

In each training step, the generator is fed with random noise [9] and creates images from it.

The discriminator sees a batch of true images and a batch of the generated images. The loss of the discriminator is based on how well the discriminator

detected fake images as fake and real images as real. [10]

The loss of the generator is estimated by how well the generator was able to fool the discriminator. The more images the discriminator falsely classified as real, the better our generator works and the smaller the Binary Cross Entropy loss between the discriminator's predictions and all labels as true=1.

Training Hyperparameters for orientation in the hints.[11]

We recommend to visualize the losses and output images using **TensorBoard**. Visualizing them as plots like in the previous weeks is also possible, though. Be aware that, unlike in your previous architectures, the goal is not to get the loss as low as possible. The **loss** of the generator or discriminator going against **zero** is actually a sign of training **failure**.

For evaluation, you should therefore create some random latent vectors before training and feed them through the generator regularly. You can plot the resulting images to evaluate training.

Training GANs is messy, and it is hard to pinpoint the reason why training fails. But as our input images are not of the greatest quality anyway, your GAN should in the end be very able to approximate these sketches.

4 Optional: Is it Christmas?

Now let's have a little fun. For all the non-Germans among you, let's recap a popular Christmas poem:

Advent, advent, a candle is burning
First one, then two, then three, then four,
Then the Christkind is standing in front of the door.

And when the fifth candle is burning
you missed Christmas.

Advent, Advent ein Lichtlein brennt.
Erst eins, dann zwei, dann drei, dann vier.
Dann steht das Christkind vor der Tür.

Und wenn das fünfte Lichtlein brennt,
dann hast du Weihnachten verpennt.

Be a candlemaker: Write a function called `is_it_christmas(model)` that generates you either 3, 4 our 5 new candles dependent on the current date according to the poem.[12]

A possible output for the day the homework went online can bee seen in the figure ??.[13]

Sorry... You have to wait 14 more days to christmas.

5 Outstanding Requirements

General Reminder: Rating yourself as outstanding means that your code could be used as a sample solution for other students. This implies providing clean understandable code, descriptions and types of arguments and explanatory comments when necessary.

For an **outstanding**, also implement a second version of the basic GAN and train it. Choose one of the following options.

- Wasserstein GAN. You can reuse most of the basic GAN for that. Create an extra WGAN loss function. [14]
- class conditional GAN

There are plenty of online resources, which you can use for the WGAN or cGAN. We will still provide support in the coding sessions if you have trouble.

The following points are still required but you should be able to copy and paste most of it by now.

- A proper Data Pipeline (that clearly and understandably performs necessary pre-processing steps and leaves out unnecessary pre-processing steps).
- Clean understandable implementations of your data set.
- Commments that would help fellow students understand what your code does and why. (There is no exact right way to do this, just try to be empathic and imagine you're explaining topic of this week to someone who doesn't now much about it.)
- Nice visualizations of the losses and accuracies. (You don't have to plot every epoch. Once in the end is enough. Although some print statements to keep track of the performance during training are welcome.) This week TensorBoard is the recommended way to go, but alternatives are also ok. Also display generated images for each epoch to asses the visual quality.
- A proper `is_it_christmas` function.

Notes

[1] using `tf.data.Dataset.from_tensor_slices()`

[2] Your resulting images should be of shape (28,28,1)

[3] Batching, Shuffling....

[4] This is basically just like a CNN architecture for a binary classification task, thus you have a Dense layer with a single neuron in the end.

[5] BatchNorm and Dropout might be useful.

[6] You can try different latent space sizes. Something around 100 would be a good place to start.

[7] It is common to use even sized kernels in transposed convolutions to avoid checkerboard artifacts, Deconvolution and Checkerboard Artifacts

[8] You can also use additional convolutinoal layers in between upsampling layers, which gives the generator more parameters to work with

[9] Create random noise, possibly with an own function, using tf.random.normal(). The size of your noise vector is the size of your latent space.

[10] Compute the binary cross entropy between the generator's output on fake images and all labels=0. Similarly, compute the BCE between the generator's output on the real images and all labels = 1. Add them both to receive the resulting loss of the discriminator.`tf.ones_like()` and `tf.zeros_like()` could be helpful for creating the true labels as comparison.

[11]

- epochs: 10 is a good starting point, then see if you want to train longer or if you are already satisfied with the quality of the generated images
- optimizer: Adam or RMSprop

[12] Make use of the `datetime` package

[13] You can choose to convert the image output of the GAN back to a white background or leave it black.

[14] For the discriminator, use a linear activation function instead of sigmoid.