Homework 08

IANNWTF 20/21

Submission until 6 Jan 23:59 via https://forms.gle/H31ckx5251Qg3Ndm6

Welcome back to the 8th homework for IANNWTF. This week you learned about generative adversarial networks.

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 again work with the Fashion MNIST dataset. It contains 60000 images 28x28x1 which are labeled into 10 different categories. You can load the dataset from the tf.keras.dataset module.

Perform necessary or beneficial preprocessing steps.[1]

2 Model

Your task is to implement a GAN.

Create a class **Discriminator** with call and init functions. It takes in a batch of images and returns a vector of probabilities that the respective image was fake or real. [2]

Create a class **Generator** also with call and init functions. The generator takes a random point from the latent space[3] 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 [4][5]. In the last layer, perform a convolution with 1 feature map and tanh/sigmoid activation. This will be the output image.
Feel free to try out regularization techniques that we have learnt about.[6]

3 Training

In each training step, the generator is fed with random noise [7] 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. [8]

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.[9]

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. Do not expect great image quality. Something that roughly is shaped like fashion is sufficient. If you want better results google around for tips and tricks to train GANs. The internet is full of them but which methods work (best) is hard to filter out. If you feel like experimenting over the holidays go for it. Just be sure to document what you are doing.

4 Outstanding Requirements

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. [10]
- 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.

General Reminder: Rating yourself as outstanding means that your code could be used as a sample solution for other students. This implies not relying on or thoroughly explaining things not covered in the lecture so far and providing clean understandable code and explanatory comments when necessary.

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.

Notes

[1] Relevant preprocessing steps:

- Shuffling Normalize the images to the range (0,1). Then you can use the sigmoid activation as output to reconstruct images in the same range. Alternatively use a range of -1 to 1 and use tanh as an output activation.
- Batching, an orientation would be minibatches of size 32.

[2]Remember that the underlying idea is that the discriminator performs down-sampling. The output layer should be Dense with 1 neuron and sigmoid activation, indicating whether the image was fake or real.

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

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

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

[6]BatchNorm and Dropout might be useful.

[7]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.

[8]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.

[9]

- 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

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