Homework 07

IANNWTF 20/21

Submission until 16 Dec 23:59 via `https://forms.gle/H31ckx5251Qg3Ndm6`

Welcome back to the 7th homework for IANNWTF. This week's topic was autoencoders.

Therefore we will build an autoencoder. We will use it to learn embeddings for the Fashion Mnist dataset. We split the homework into two parts. The first is to implement a simple convolutional autoencoder and perform some latent space analysis. The second will be to implement a variational autoencoder, perform the same analysis and comparing the results to the simple autoencoder. The first part is required for passing the homework. Both parts are required to get an outstanding. Also both parts are required for getting cool results.

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!). Also, you can always refer to the code snippets and notebooks provided in courseware.

1 Data set

This week we will 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

Implement a convolutional autoencoder (and a variational autoencoder for an outstanding).

2.1 Convolutional Autoencoder

- The Autoencoder should consist of an encoder and a decoder, which can be called independently. [2]
- Encoder: The encoder should reduce the size of featuremaps like a CNN.[3][4] At the end of the encoder, flatten the feature maps and use a dense layer to produce an embedding of a certain size. [5]
- Decoder: The decoder takes the embedding from the encoder as input. Use a dense layer to restore the dimensionality of the flattened feature maps from the encoder and reshape the resulting vector into feature maps again. Use upsampling or transposed convolutions to mirror your encoder. As an output layer use a convolutional layer with one filter and sigmoid activation to produce an output image.

2.2 (Variational Autoencoder)

For the implementation of a Variational Autoencoder refer to online sources. There are good tutorials that do a better job at explaining the implementation than we can do in a few lines of text here. There are also different ways to implement VAEs in tensorflow. Here is a guide using only tensorflow. There is also the tensorflow probability library, which provides probabilistic layers. Have a look at this tutorial, which explains how to build VAEs with tfp. You can also use any other source that you can dig up. Please explain in detail how sampling and the loss are handled in your implementation so we know you did not just copy and paste without understanding whats going on. If you run into troubles in this part, still feel free to come to the coding class.

3 Training

Then train your network(s). Autoencoders learn unsupervised and do not use the labels. Compute the loss between the input image and the reconstructed image. While training it is nice to plot some example images from the test set with their reconstructed counterparts to visualize the training progress. [6]

4 Latent Space Analysis

Embed the first 1000 images of the test set using the encoder. Reduce the dimensionality of the embeddings to two using the t-SNE algorithm.[7] Then plot the data points, colored according to their class. Evaluate the result. Is it what you expected? How is this related to the courseware content? Further interpolate linearly between the embeddings of two images and plot the reconstructed images. Again relate this to the courseware content.

For the Variational Autoencoder part do the same and compare the results.

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 not relying on or thoroughly explaining things not covered in the lecture so far and providing clean understandable code and explanatory comments when necessary.

This week the main part of the outstanding distinction is to implement a Variational Autoencoder in addition and to compare the analysis to the standard Autoencoder.

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.)

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
- Batching, an orientation would be minibatches of size 64.

[2]The easiest way to do this is with model subclassing. Define separate models for the encoder and decoder and initialize them in the autoencoder constructor.

[3]You can either use convolutional layers with stride 2 for subsampling or convolutional layers with stride 1 followed by a pooling layer

[4]subsampling two times i.e. to a size of 7x7 should be enough

[5]As a rule of thumb, the smaller the embedding, the harder the training and the more layers required. If experimentation fails you, try 10.

[6]Training Hyperparameters for orientation:

- epochs: around 10
- Try something around 0.001
- optimizer: Adam

[7]t-SNE is a dimensionality reduction algorithm particularly suited for visualization. You don't have to implement it yourself, you can use an existing library, e.g. `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`.
If you want to know how it works have a look at this blogpost. For some guidance on what can be and what can not be interpreted from the results refer to this Distill article