MACHINE LEARNING

SHEET 6

03.02.2022

10:15 A.M. UNTIL 11:45 A.M. VIA ZOOM

Please prepare the exercises in order to present them in the meeting. Fill-in the exercises you solved in the questionnaire in StudIP until 02.02.2022 4:00 p.m.
Join the meeting via this Zoom link:
https://uni-trier.zoom.us/j/83445228905?pwd=aHpFTWNUanZqdDlWcnB2NjVNaStoZz09

TASK 1: DEEP AUTOENCODER WITH KERAS

Autoencoders are a popular class of Deep Neural Networks that are capable of learning representations in a self-supervised way. We will build one with Keras.

a) Fetch the MNIST dataset (https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/load_data) with Keras and scale the data to [0,1].
b) Build a convolutional autoencoder model consisting of encoder and decoder. The encoder and decoder are structured in symmetrical fashion, i.e., both parts use the same layer structure. The encoder uses two convolutional layers (64 and 32 filters) followed by a flattening layer and a dense layer. The number of neurons of this dense layer defines the encoded vector size of the input image. Set it to 64. Use parameters of your choice for all other layers. The decoder transforms the input data, i.e., the encoded vector coming from the encoder, in reverse order compared to the encoder. An upscaling dense layer followed by two deconvolutional (transposed) layers and a final convolutional layer with a single filter, all layers using similar parameters as the encoder layers. Explain the purpose of the individual layers and how the input shapes change during the transformations of these layers.
c) Train this model using a binary cross-entropy loss for three epochs. Use the SGD optimizer with a learning rate of 1.5.
d) Select five random images from the test set and compare the visualizations of the original and the auto-encoded images.

OPTIONAL TASK 2: DENOISING AUTOENCODER WITH KERAS

In this exercise, we will use autoencoders for the purpose of denoising data. The approach will be similar to that of Exercise 1.

a) Load the MNIST dataset and scale it to [0,1].
b) Build an autoencoder model of your choice. The encoder must produce vectorial representations of size 64. You can use a convolution model as in Exercise 1 or use a simple model of dense layers.
c) Add three different forms of noise to the training and test data (one noisy dataset for each type of noise). Make sure that the resulting images are still in the range [0,1].

Also use a method that allows a factor to control the amount of noise that is added. Use 0.25 as the value for tests:

1. Gaussian noise with a mean of 0.0 and a standard deviation of 1.0.
2. Uniform noise from a range of [0,1].
3. Noise in form of merged images. Apply noise to one image by merging it with another random image from the examples.

d) Train your model for three epochs using a suitable loss function. Use a combined dataset with all noisy examples as input data and use the original examples as target data. Use the SGD optimizer with a learning rate of 1.5.

e) After training, visualize the model's performance on a single digit. Display the original image, all three noisy variants, and the denoised image.

OPTIONAL TASK 3: SUPERVISED REPRESENTATION LEARNING

In contrast to the exercises from before, learning representations can also be done supervised. This is especially useful in scenarios where labels for similar data instances exist that can guide the learning process.

a) Load the MNIST dataset and scale the data to the range [0,1].

b) Prepare a training data generator (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#from_generator). Each training instance is put together from two images and a label that is either 1 or 0. Randomly, generate training instances by putting together two images of the same class (with label 1) or two random images (with label 0).

c) Build a model using the Functional API of TensorFlow (https://www.tensorflow.org/guide/keras/functional). The model should have a sub-model that transform a single image into a representation of size 64. The main model should take the pairs of images and transform both of them to their vector representation using the same sub-model. Afterwards, a pairwise dot product should be computed for the vector pair. This dot product is the final output.

d) Train the model with the priorly created generator for five epochs, using hinge loss. Why is this loss function used? How does it work?

OPTIONAL TASK 4: REINFORCEMENT LEARNING FROM SCRATCH

In this exercise, we will build up a Reinforcement Learning (RL) process. We will train a model to help us play Tic Tac Toe. Use the provided class *tic_tac_toe.py* for this exercise.

a) Get familiar with the provided class *TicTacToe* by creating a game and playing a few rounds with random choices. The AI player is supposed to be the player *o*.

b) Collect the information of 1000 games that can be used to train a model later on:
    - Setup a loop that plays 1000 games.

- Each game is played by both players alternating. Player *x* (You) always begins with a random play. Afterwards, player *o* makes its random move.
- For each of your turns, i.e., the turns of player *x*, store the board state before the turn and the cell that was chosen (information is returned by random play method and can be retrieved from board object).
- Always check if the game is finished after a player's turn and if so, compute the rewards for this game (use the provided function *get_rewards()* in the file). This method expects the winner of the game as a parameter and a list of player *x*'s plays as tuples of chosen cell and board state before play.

c) The rewards are given by this function as a tuple where the first value is the rewards (target) and the second value is the training data in form of chosen cell and board state. Train a model of your choice (can be a Deep Learning Model or another model) to predict the expected reward given the provided training data. Split the available data before training (test ratio of 0.2).

d) Evaluate the trained model on the test set in terms of prediction deviation. Explain how the trained model could be used to help human players win Tic Tac Toe games.

Optional Task 5: Variational Autoencoder With TensorFlow

Variational Autoencoders (VAEs) bridge the gap between data representation and data generation. They are trained liked autoencoders but are usually used as generative models during inference. You should stick to the TensorFlow Probability tutorial (https://www.tensorflow.org/probability/examples/Probabilistic_Layers_VAE) to solve this exercise.

a) Load the MNIST dataset and scale the data to the range [0,1].
b) Setup a convolutional VAE using TensorFlow probability (new dependency needed).
c) Train this model for 5 epochs using the ELBO loss that is provided in the tutorial.
d) Generate some new examples with the trained model and visualize them.
e) Explain in detail how this approach works. Especially focus on the connection between encoder and decoder. How is the hidden vector representation of the image treated? What is the purpose of the special layers from TensorFlow Probability?