Télécom Physique Strasbourg - Université de Strasbourg - SDIA 2A

TP1 : Unsupervised Deep Learning

1 Introduction

Objectives of the TP: implement and train increasingly complex autoencoder models.

We will use Python with TensorFlow (≥ version 2.0) to define and train the models and Matplotlib for visualisation.

2 Tasks

2.1 Autoencoder

Download skeleton.py and complete the missing parts to create an autoencoder (AE) for the MNIST dataset (define the AE, training, and testing where commented). Complete the model (using the predefined parameters) to have:

- a dense hidden layer with 128 neurons and ReLU activation,
- followed by a dense decoder (output) layer with sigmoid activation, the size of the output should match that of the input.

Study the rest of the code along with the API so you understand the basic manner of defining and training a model in Keras.

2.2 Stacked Autoencoder

Extend the AE model to be a stacked autoencoder (SAE) trained end-to-end, i.e. all in one go, with the following number of hidden nodes per layer in the encoder (the decoder should be the reverse of this):

```
hidden_units = [128, 64, 32]
```

Experiment with a few different numbers of hidden nodes to see how it affects the reconstructions.

2.3 Convolutional Autoencoder

Replace the encoder layers of the SAE by blocks containing:

- a 2-dimensional convolutional layer (`Conv2D`) with `(3, 3)` convolutions, ReLU activation, and `padding='same'`
- followed by a 2-dimensional maxpooling layer (`MaxPooling2D`) with `(2, 2)` pool size, and `padding ='same'`.

The decoder layers need to reverse these operations to reconstruct the original image, the decoder blocks should therefore contain:

- a 2-dimensional convolutional layer (`Conv2D`) with `(3, 3)` convolutions, ReLU activation, and `padding='same'`
- followed by a 2-dimensional UpSampling layer (`UpSampling2D`) with a size of `(2, 2)`.

Finally, a layer needs to be added to reconstruct the image, this should be a 2-dimensional convolutional layer with `(3, 3)` convolutions, sigmoid activation, and `padding='same'`. This then forms a Convolutional SAE, with `filters = [16, 8, 8]` in the encoder and, as above, the decoder should have the reverse. You will need to reduce the learning rate to 1.0.
You will now need to use the unflattened (i.e. image) data with this model, but before it can be used, you will need to add an additional dimension to the data (to account for the number of bands, even though there is only one in this example, i.e. greyscale):

```
X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]
```

If this is too slow to train, you can reduce the number of filters by half and/or reduce the number of epochs (try 10).

2.4 Convolutional Denoising Autoencoder

Now use numpy to set random pixels of the input data to 0 with probability $p = 0.5$. Remember that the AE's output should be the uncorrupted data.
Try different corruptions, i.e. adding random Gaussian noise (do not forget to clip the resulting values to be between 0 and 1).

3 Extra

If you have time (or are interested further), here are some more tasks.

3.1 Layer-by-layer SAE

The following function defines how to construct a SAE layer by layer (instead of training it end-to-end as before):

```
def build_autoencoder(input_size, hidden_units, activations, optimizer, loss):
    output_size = input_size
    input_layer = Input(shape=(input_size,))

    layer = Dense(units=hidden_units, activation=activations[0])(input_layer)

    # To access encoded data
    encoder = Model(input_layer, layer)

    decoded = Dense(units=output_size, activation=activations[-1])(layer)

    model = Model(input_layer, decoded)
    model.compile(optimizer=optimizer, loss=loss)

    # Prepare encoded input for next SAE layer
    encoding_dim = hidden_units
    encoded_input = Input(shape=(encoding_dim,))
    # Separate decoder model
    decoder_layer = model.layers[-1]
    decoder = Model(encoded_input, decoder_layer(encoded_input))

    return {'model': model, 'encoder': encoder, 'decoder': decoder}
```

Download extra_part_1.py, which implements this function to form a layer-wise SAE. Study the code, execute it, and analyse the output. Compare the results to the end-to-end SAE.
What could be done to improve it?

3.2 Other Datasets

MNIST is a commonly used benchmark dataset for machine learning, try to adapt the stacked denoising autoencoder to work with another dataset (e.g. Fashion-MNIST, SVHN, or another of your choice). You will need to adapt the architecture of the autoencoder and the training parameters for these to work properly. This is usually done by analysing the losses and trial and error.