Practical work 13 – Autoencoders and Generative RNNs

Objectives

The objective of this PW is to practice with auto-encoders and generative recurrent neural networks.

We ask you to submit the solution in one week.

Submission

— **Deadline** : W14 (in 1 week).

— **Format** : Zip with report and iPython notebook.

Exercise 1 Auto-encoders

We will use the notMNIST data set. First get the `.gz` files from this site : `https://github.com/davidflanagan/notMNIST-to-MNIST`.

Use as a starting point the notebook `notMNIST-auto-encoder-stud.ipynb`.

Create a shallow dense autoencoder

Using the Keras library, build a very simple autoencoder with one input layer, one output layer and one hidden layer.

a) Chose the dimension of your hidden layer. As this is the size of your encoding, you should chose a “reasonable” dimension, in between the number of classes of your problem and the dimension of input and output layers [1].

b) Define your layers, don’t forget that your input dimension is the same as your output.

c) Create your models (using the keras `Model`) : encoder, decoder and autoencoder (encoder and decoder).

1. We tried with 32 and it was working well, you may try yourself with different values to observe the impact.

d) Don't forget to compile your autoencoder. Fit it on the data !

e) Visualize your encoding image with the input images on the top and their corresponding encoded ones on the bottom. Your visualization may differ slightly from the one on Figure 1

f) How could we evaluate the performance of such a "compression" tool (in terms of loss and in terms of gain of bandwidth) ? You just need to comment – no need to compute anything here[2].

g) Are you now able to use the encoded features to train a simple Support Vector Machine – SVM classifier ? Comment on the performance by comparing the extracted features of the auto-encoder with using the pixel values as input of the SVM.

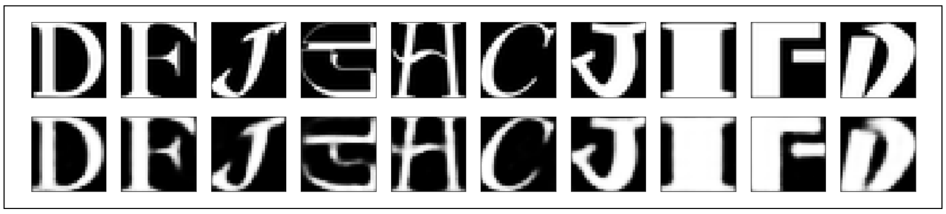


FIGURE 1 – Input and output of the autoencoder

Create a convolutional de-noising autoencoder

Create a model in Keras with the following structure :

- Encoder
  - Convolution layer with 32 filters of size 3 x 3
  - Downsampling (max-pooling) layer of size 2 x 2
  - Convolution layer with 64 filters of size 3 x 3
  - Downsampling (max-pooling) layer of size 2 x 2
  - Convolution layer with 128 filters of size 3 x 3
- Decoder
  - Convolution layer with 128 filters of size 3 x 3
  - Upsampling layer of size 2 x 2 (see `UpSampling2D()`)
  - Convolution layer with 64 filters of size 3 x 3
  - Upsampling layer of size 2 x 2
  - Convolution layer with 1 filter of size 3 x 3

a) Check with the `summary()` method that you have indeed a diabolo network and that your output shape is the same as your input shape.

b) Would it make sense to use the output of the encoder to compress the images ? Comment your answer.

2. If you really want to compute something, you can of course.

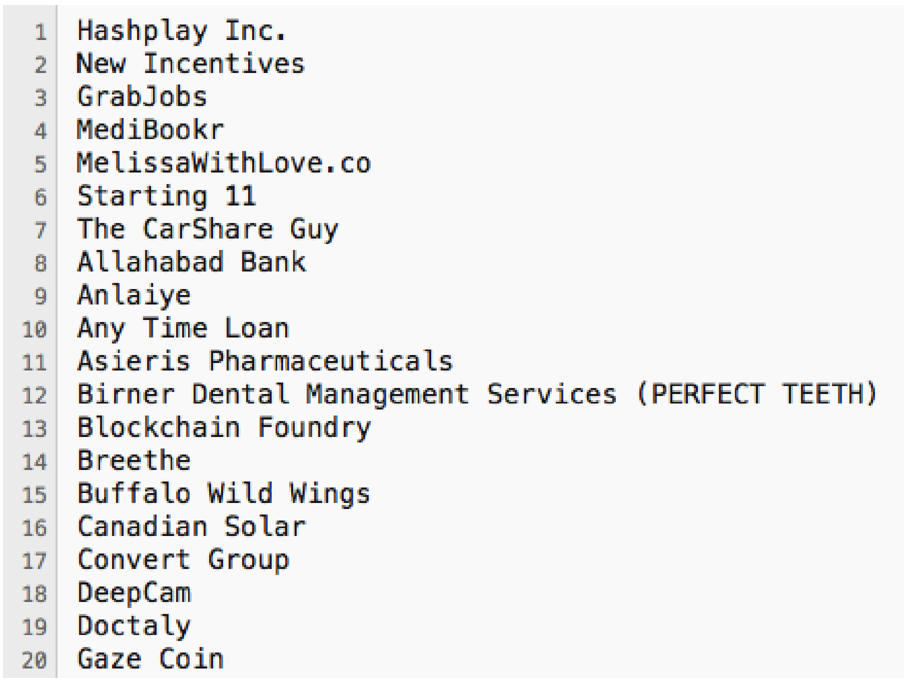
c) Generate noisy versions of your train and test data using (see examples in slides).

d) Train your network using a MSE loss and batch sizes of 128.

e) Visualise the denoising capacity on some test data.

Exercise 2 Sequence generation - startup names

The objective of this exercise is to build a generator of startup names using a RNN. We will use to train the system a corpus of existing startup names composed of 172K entries. Figure 2 illustrates the 20 first entries of the corpus.



```
Hashplay Inc.
New Incentives
GrabJobs
MediBookr
MelissaWithLove.co
Starting 11
The CarShare Guy
Allahabad Bank
Anlaiye
Any Time Loan
Asieris Pharmaceuticals
Birner Dental Management Services (PERFECT TEETH)
Blockchain Foundry
Breethe
Buffalo Wild Wings
Canadian Solar
Convert Group
DeepCam
Doctaly
Gaze Coin
```

FIGURE 2 – 20 first entries of the training set of startup names

Many-to-one approach

a) Read again the examples on the Shakespeare text generation in the section *Generative RNN* from the class support. Make sure you understand the different steps for the data preparation. You can also start from the ipython notebooks provided on Moodle for the Shakespeare text generation.

b) Get from Moodle the file `companies.csv` that will be used to train the system.

c) Read the data from the file. Treat the whole set of names as a unique string of text, like in the Shakespeare example.

d) Follow similar steps for the data preparation as in the Shakespeare example. First extract the set of chars and define the encoding and decoding dictionaries. Then chop the data into X and y, you may define here a sequence length of 10 and a skip of 3. Finally transform your data and labels into one-hot vectors.

e) Define your model using a SimpleRNN (64 units) and a Dense with softmax activation layers.

f) Train your model.

g) Generate startup names using different values of the hyperparameters : number of epochs, number of cells in the RNN, etc.

Report about your findings in the pdf report. Select 5-10 generated startup names that you find interesting.

OPTIONAL - Many-to-many approach

Redo the previous exercise using this time a many-to-many approach. You will need to modify slightly the way the target tensors are prepared in order to do so. Pay also attention to the parameters of the layer definition that are a bit different.

OPTIONAL - LSTM and GRU

Replace the Simple RNN layers with LSTM or GRU and comment on your observations :

— Can we reduce further the loss ?

— Do you observe improvements in the quality of generated names ?

Exercise 3 Optional : Review Questions

a) What are the potential usage of auto-encoders ?

b) In which situations would you use auto-encoders to extract features ?

c) How can you define a generative system ? Describe the two approaches seen in the class to build generative systems with RNNs.