

Introduction to Deep Learning

Assignment 3:

Recurrent neural networks and generative models

November 2021

Task 1: Recurrent Neural Networks

The goal of this task is to learn how to use encoder-decoder recurrent models. Specifically we will be dealing with a sequence to sequence problem and try to develop a neural network that can learn the principles behind simple arithmetic operations.

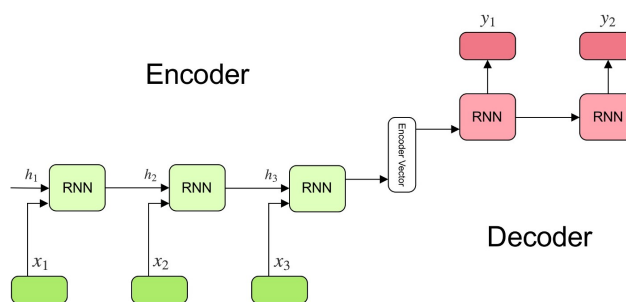


Figure 1: An illustration of an encoder-decoder recurrent network model.

Description: let us suppose that we want to develop a neural network that learns how to add or subtract two integers that are at most three digits long. For example, given input strings of 7 characters: ‘181+ 24’ or ‘341-898’ that consist of 2 three-digit long integers and an operand between them, the network should return a sequence of 4 characters: ‘ 205’ or ‘-557’ that represent the result of their respective queries. Additionally, we want to build a model that generalizes well - if the network can extract the underlying principles behind the ‘+’ and ‘-’ operands and associated operations, it should not need too many training examples to generate valid answers to unseen queries. To represent such queries we need 13 unique characters: 10 for digits, 2 for the ‘+’ and ‘-’ operands and one for whitespaces ‘ ’.

The example above describes a text-to-text sequence mapping scenario. However, we can also use different modalities of data to represent our queries or answers. For that purpose, MNIST handwritten digit dataset is going to be used again, however in a slightly different format. We can create an alternative image representation of our text queries described in the first paragraph by stacking multiple MNIST samples as follows:



Figure 2: Converting text query ‘19-186’ and its answer ‘-167’ to sequences of images using randomly selected samples from the MNIST dataset.

We can then use these text and image queries/answers as our training data/labels and create different variations of mappings between text and images - e.g. using text input to predict image output or vice versa. You are provided with a [Jupyter notebook](#) that contains functions for creating the image and text datasets along with sample *Keras* code for building a text-to-text RNN model.

Your tasks are as follows:

1. Analyze the code for generating numerical and image queries and their respective answers from MNIST data. Inspect the provided text-to-text RNN model, try to understand the dimensionality of the inputs and output tensors as well as how they are encoded/decoded (one-hot format).
2. In the text-to-text scenario, we are not interested in “memorizing” the whole addition/subtraction table (about 2,000,000 strings in total); instead, we want the network to learn some general principles behind these arithmetic operations. Therefore you should try multiple different splits for your training and test sets (5% train - 95% test; 10% train - 90% test, etc.) and evaluate the accuracy and generalization capability of your models.
3. Create an image-to-text RNN model: given a sequence of MNIST images that represent a query of an arithmetic operation, your model should return the answer in numerical format. Once you have trained your model, evaluate its accuracy and compare it to the text-to-text model.
4. Create a text-to-image RNN: given a text query your network should generate a sequence of images that represent the correct answer. In this case, it is harder to evaluate the performance of your model qualitatively. However, you should provide the examples of the output generated by your model in the report. What can you say about the appearance of these generated images?

Task 2: Generative Models

The goal of this task is to learn training and using generative models for image-related tasks. You are provided with a [Jupyter notebook](#) that illustrates how you would build, train and use Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs). You can download the dataset that was used [here](#). Your main goal in this task will be to find your own image dataset, preprocess it and retrain these networks, and use the models to generate novel content.

State of the art generative models can create an endless stream of photorealistic images of whatever data they are trained on (e.g. *StyleGAN2*). However, such models are incredibly hard and expensive to train, therefore it is better to start out small and understand the working principles behind smaller and more simple architectures.



Figure 3: *Output of StyleGAN2.*

Your tasks are as follows:

1. Go through the provided notebook and try to understand how the generative models are built using Keras. To start with you are provided with a convolutional autoencoder (CAE) architecture which contains the two main building blocks - the encoder and decoder. The VAE is built in a very similar way, but includes a sampling layer that gives this model its generative capabilities. The generator and discriminator in the GAN model are also built using these similar building blocks, however they are arranged differently and use a very different loss function.

2. Training 'vanilla' generative models with multiple categories or highly varied images within a single category will likely result in a model that does not generate interesting output. That is why datasets of faces are often used to showcase the power of generative models as they are usually very uniform (people facing the camera at a similar distance and angle). However there are definitely many other interesting and viable use cases and applications. Find a new image dataset of your choice and use the code provided in the notebook to retrain your generative models (both VAE and GAN). You should provide a link to the data that you used in the report along with its description.
3. To properly train the models you will probably need to rescale your data to be close to the size of the example dataset. Since all of the neural networks in the notebook are build using functions *build_conv_net()* and *build_deconv_net()* you can also adjust the convolutional architectures if you desire - both to more closely match the scaling of your data and reduce/increase the complexity of your architectures.
4. Once you have retrained the models, include the modified notebook with your results in the submission. Create some illustrative representations of the output of your generative models.
5. Once your models are trained, their latent spaces model the distribution of the data that they were trained on. You can use randomly generated vectors as input to the decoder components of these models to generate novel images (the decoder in case of VAE, generator in case of GAN). Each vector represents a point in this latent space and since this space is continuous you can generate various interesting visualizations. Your goal is to create one of these visualizations by interpolating between two random points in the latent space. Can you think of other creative uses for the latent data representations?
6. In your report provide a detailed description of the working principles of these generative models and emphasize the differences between them.

Deliverables: your submission should consist a single *.zip* file containing your report in *pdf* format (at most 10 pages) along with your code (Jupyter notebooks and/or scripts for both of the tasks). You can find more information about report writing along with examples on Brightspace → Assignments section.

Deadline: 10/12/2021 23:59.