

Introduction to Deep Learning

Assignment 3

Irina-Mona Epure, Ramya T Rameshchandra

1 Recurrent Neural Networks

Recurrent Neural Networks are a class of nets that can work on sequences of arbitrary lengths. A recurrent neural network looks very much like a feedforward neural network, except it also has connections pointing backward. Long Short Term Memory networks (LSTMs) are type of RNN which are designed to remember information for long periods of time. In the experiments in the following subsections, we have used LSTM cells.

The basic structure of the predictor consists of Encoder-Decoder recurrent models. This is a sequence to sequence model which we will be using to learn arithmetic operations (addition and subtraction).

1.1 Generating Numerical and Image queries from MNIST data

This part of the experiment involves generating numerical and image arithmetic query and their solutions from MNIST data.

1. Image Data: This part of the code generates images of queries and their corresponding results. The highest integer is taken to be 199. All possible addition and subtraction arithmetic operations are carried out with numbers less than the highest integer. The greatest result in this case would be 398 and the lowest result would be -199. The result will not require more than 4 characters and the total number of unique characters in the query is 13 (0123456789+-).

The interger images (0-9) are taken randomly from MNIST handwritten data set while the images for + and - symbols are generated with size 28x28 with cv2.line wherein the crosses are chosen at random in a range (12-16 for horizontal cross, and 4-8/20-24 for vertical cross).

For a single image of a query or a result, we stack the individual characters. The size of the stack remains constant with 7 for input and 4 for output. The input can at most be of length 7 (as in 199+199); the image for this would be stack for 1,9,9,+,1,9,9. Similarly for the output. If a number has less number of characters than the maximum (3 in this case), it is padded with spaces to the left.

2. Numerical Data: This part is quite straightforward: each of the query strings is evaluated to get the result.

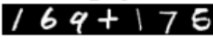

X_text: "169+175" = y_text: " 344"
X_img:  y_img: 

Figure 1: Sample X_text, y_text, X_img, y_img

1.2 Text-to-Text

In this subsection we train the RNN network to take in numerical data and output the corresponding result of the arithmetic operations. To feed the input into the RNN network we first need to encode the train data. The input as well as the output are one-hot encoded wherein each input sample is of size 7x13, rows represents each input character and columns represent all the unique characters present; each row has a value 1 only in the corresponding column and value 0 elsewhere.

Layers of the chosen **sequential RNN** in order:

1. LSTM(256, input_shape=(max_query_length, len(unique_characters))): This is an LSTM layer with 256 units taking the input of shape 7x13 which corresponds to one-hot encoded text data. This layer outputs only the last output of the output sequence.
2. RepeatVector(max_answer_length): Since the outputs are of size 4, we repeat the output 4 times to obtain the result. This concludes encoder part of the network.
3. LSTM(128, return_sequences=True): This layer has 128 units and takes as input the output of the encoder. This layer outputs the entire output sequence.
4. TimeDistributed(Dense(len(unique_characters), activation='softmax')): The final output of the decoder is a dense layer with 'softmax' activation which gives the prediction of unique character. This is used along with TimeDistributed layer which provides a one-to-one relationship between input and output.

The above network was tested with different train-test split for 5 epochs and a batch_size of 10. The results are shown in table 1

1.3 Image-to-Text

In this experiment, we created a RNN network which takes as input a stack of images corresponding to an arithmetic equation and outputs the one-hot-encoded text. To input

Table 1: Text-to-Text Results

Train-Test Split	Loss	Accuracy
50-50	0.500	0.822
5-95	0.290	0.898
10-90	0.162	0.942

the images, we flattened the input before feeding it to the network. The network structure is shown in image 2 Running the network on train-test split of 75-25 for 20 epochs and batch

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 7, 512)	2656256
lstm_1 (LSTM)	(None, 256)	787456
repeat_vector (RepeatVector)	(None, 4, 256)	0
lstm_2 (LSTM)	(None, 4, 256)	525312
time_distributed (TimeDistributed)	(None, 4, 13)	3341

=====
Total params: 3,972,365
Trainable params: 3,972,365
Non-trainable params: 0

Figure 2: Image to Text Network Structure

size of 10, gives a satisfactory accuracy of 0.942 [3].

```
Epoch 20/20
6000/6000 [=====] - 109s 18ms/step - loss: 0.0689 - accuracy: 0.9809
625/625 [=====] - 6s 8ms/step - loss: 0.2531 - accuracy: 0.9422
test loss, test acc: [0.25309357047080994, 0.9422374963760376]
```

Figure 3: Image to Text Result

1.4 Text-to-Image

In this experiment, we attempt to generate images of results of text input of arithmetic equations. We split the experiment into two networks. The first network reads the text and outputs the text of the arithmetic operation. The second network reads the text output

of the previous layer and in turn outputs an image of the input. For the first part of the network we used the Text2Text network trained in Section 1.2. For the second part of the network we trained a GAN network to create images from MNIST data. The idea is to have a sequential network which includes pre-trained text-to-text followed by pre-trained GAN image generator with SSIM loss. We were successful in training the two networks. Testing the performance with the two networks working in tandem is to be tested.

2 Generative Models

In the second task, we experimented with variational autoencoders (VAEs) and generative adversarial networks (GANs) on two different datasets.

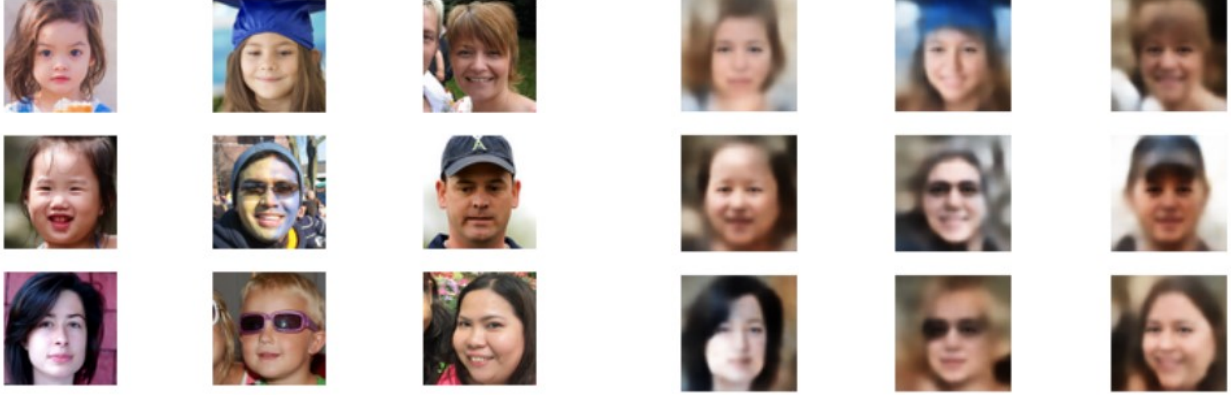
2.1 Generating Human Faces

We were first provided with a dataset of 20,000 images of human faces in a 64x64x3 format. The images are cropped and centered, such that they are very uniform and easy to use with generative models. Besides this, we were also given the architectures of the generative models to be applied.

These models always consist of two main components: a downsampling architecture that extracts features from or models the distribution of the data, and an upsampling architecture that uses latent vectors to generate new images resembling the ones from the training dataset. Both of these components are implemented as separate convolutional neural networks (CNNs). An implementation of a **Convolutional Autoencoder (CAE)** was made available as an illustration of how decoder CNNs can reconstruct an image after encoder CNNs compress them to a numerical vector of size 256. Applying this model to samples from the human faces dataset resulted in quite accurate reconstructions, as depicted in Figure 4.

The CAE architecture was then used as a basis for implementing the generative models that we experimented with:

1. **Variational Autoencoders (VAEs)** have two components:
 - The downsampling architecture is called an encoder, and is tasked with modelling the posterior distribution of the images in the dataset in the form of a probabilistic space which is approximated by a normal distribution.
 - The upsampling component is referred to as a decoder, and is responsible with sampling a random latent vector from the distribution generated by the encoder. It then outputs the parameter of a conditional distribution of the sample.
2. **Generative Adversarial Networks (GANs)** contain the following components:



(a) Original Samples of the human faces dataset.

(b) Reproduction of the human faces samples.

Figure 4: Original and reproduced samples resulting from the CAE architecture.

- The generator is a deconvolutional neural network that receives random noise as input and transforms it into images.
- The discriminator has the role of predicting whether these images and other ones from the dataset are real or created by the generator.

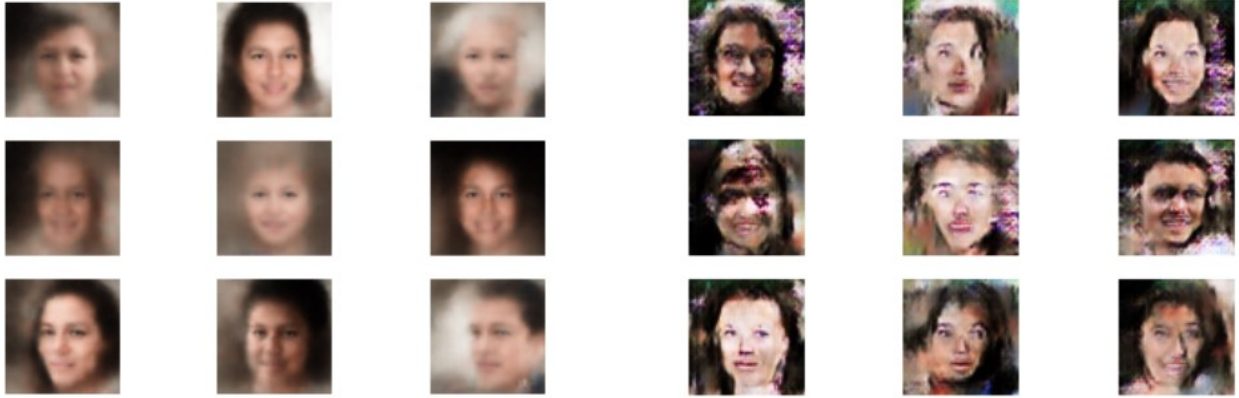
Over time, both of these two components are trained alongside each other. The discriminator gets better at telling real and fake images apart, while the generator learns to produce more and more realistic outputs. Samples created by the GAN, as well as the VAE, based on the human faces dataset are shown in Figure 5.

In the case of VAEs, the space between the encoder and the decoder is a probabilistic space which is approximated by a normal distribution. There is a compressed multivariate Gaussian between the encoder and the decoder, from which a vector can be sampled. This vector can be fed as input to the decoder, which will then output an image that resembles one from the training data. The latent space of a GAN is instead the space from which we sample the vectors fed to the generator. When a vector is sampled from this space, it will be converted into an image similar to the ones used for training, but which is not the same as an existing image.

2.2 The Animal Faces HQ Dataset

After using generative models to generate images of human faces, we decided to use the same architecture on an animal faces dataset. We chose the Animal Faces HQ (AFHQ) dataset¹ created by [1]. It consists of 16.130 RGB images with a resolution of 512x512. The images are split into three class: cats, dogs, and wildlife. For the purpose of training our generative models, we used all of the images, regardless of class.

¹<https://github.com/clovaai/stargan-v2>



(a) Human faces created by the VAE.

(b) Human faces created by the GAN.

Figure 5: Sample images of human faces created with generative models.

Because the dataset has such a high resolution, we rescaled the images to 64x64 pixels, just like the human faces dataset. In order to do this, we used the *block_reduce()* function from the *skimage* Python library. We then stored the images as arrays in an *.npy* file for further usage.

2.3 Generating Animal Faces

We used the same VAE and GAN architectures to generate new samples of animal images based on the AFHQ input data. Some of the results are visible in Figure 6. For both animals and humans, the VAE-generated images have a very blurry appearance. In contrast, GAN-generated samples are very sharp and patchy, giving the impression of someone having taken separate features from different individuals and then placed them on the same face.



(a) Animal faces created by the VAE.

(b) Animal faces created by the GAN.

Figure 6: Sample images of animal faces created with generative models.

2.4 Using Random Vectors to Generate Novel Images and Their Interpolations

As our final experiment, we sampled some random vectors from a normal distribution with the same dimensionality of the latent spaces of the VAE (64) and GAN (128) and fed them to the trained decoder of the VAE, and the trained generator of the GAN, respectively. We then gradually interpolated the vectors to create a smooth transition from one to another. By feeding the resulting vectors to the generative models, we obtained some interesting visualisations of generated animals fading into another image. For example, with the VAE, we gradually turned a blurry dog into a blurry cat, as shown in Figure 7. With the GAN, we managed to smoothly transition from a sharp cat to a sharp dog, which can be seen in Figure 8.

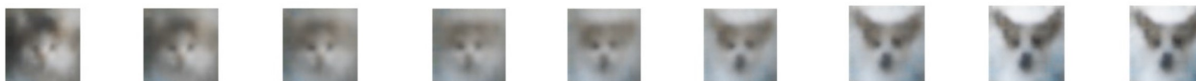


Figure 7: Using Random Vectors to Generate Novel Images and Their Interpolations with VAE.



Figure 8: Using Random Vectors to Generate Novel Images and Their Interpolations with GAN.

References

- (1) Choi, Y.; Uh, Y.; Yoo, J.; Ha, J.-W. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.