Neural Networks Assignment 2

1 Introduction

The aim of this assignment is to master modern tools (TensorFlow and Keras) for training deep networks. This paper is structured as follows: first, we will provide a proof-of-understanding on three types of deep networks: convolutional neural networks, recurrent neural networks and autoencoders. For this, we studied and slightly adapted pre-existing tutorials and examples. For each of the three types we will describe the network, the problem we selected, the data, the experiment and the results. We will also mention which adaptations were made to the original example.

Next, we will apply deep learning to an original data set. In particular, a character-based RNN will be trained on tweets from U.S. president Trump, and this network will be used to generate president Trump's 'next tweet'. Finally, this paper will conclude with a brief discussion on this experiment, our findings and suggestions for further research.

2 Proof-of-understanding

2.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of artificial neural network most often used to tackle image classification problems. A CNN can learn which aspects of an image are most important, assign corresponding (learnable) weights and biases to these aspects, and hence classify an image. The structure of a CNN is somewhat analogous to that of neurons in the human brain. CNNs consist of several layers, where each layer can have a different function. A convolutional layer applies a convolution operation to the input, passing the result to the next layer. After a convolutional layer, the output of that layer is passed through a nonlinear transform, such as the *tanh*- or *relu*-function. A major advantage of CNNs is that they require little pre-processing of the input images, relative to other image classification algorithms.

2.1.1 Problem

In this problem we will use a CNN to do what it is best at: image classification. Our CNN has seven layers: two convolutional layers, one max-pooling layer, one flatten layer (which flattens the output of the convolutional layers to create a single long feature vector), one fully connected (dense) layer, one dropout layer (randomly setting a portion of the input to zero to prevent overfitting) and another dense layer. We were inspired by a pre-existing tutorial.[1]

2.1.2 Code and data

We apply our CNN to the Fashion-MNIST data set: a fairly well-known data set created by Zalando Research consisting of a training set of size 60,000 and a test set of size 10,000.[2] Each input item is a 28 by 28 grayscale image and is associated with a label from one of ten classes in total. We set the number of epochs to 5 - this is relatively low, but ensured a reasonable runtime.

2.1.3 Experiment and results

Our original CNN has seven layers, as described above. With 5 epochs this results in an accuracy of **89.9** percent on the test set. As an experiment we changed the layers in several ways (keeping the number

[1] https://medium.com/i-a/a-simple-convolutional-neural-network-with-keras-google-colaboratory-e5f8207b799f

[2] https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/

of epochs and all other parameters unchanged). We found that if we delete the second convolutional layer, the accuracy on the test set dropped to **89.7** percent - a minimal change. If we delete the second convolutional layer as well as the max-pooling layer and the dropout layer, the accuracy is **89.1** percent. This is quite interesting, and not in line with our prior expectations: we expected that dropping layers would have had a bigger effect. Even dropping three layers out of seven did not have a major effect on the testing accuracy at all. From this we can conclude that even with a few layers a network can perform quite well - the gain of additional layers is marginal, but even some (tens of) percentage points can be an important improvement in accuracy.

2.2 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a type of artificial neural network that allows information to persist - this in contrast to traditional neural networks. The name already gives its structure away: a RNN can be seen as a chain of copies of the same network, each passing information to the next copy in the chain. Its chain-like structure makes RNNs naturally useful for data containing sequences or lists.

2.2.1 Problem

In this problem, we create a simple Recurrent Neural Network to sum two integers. The baseline of the method is to sum each bit of the binary sequence of each number from right to left. For example, let us consider $3 + 2$, which can be rewritten in binary notation as $0011 + 0010$. Summing up each bit from right to left sheds light on binary summation properties: a 0 and a 1 outputs a 1, a 0 and a 0 outputs a 0, while a 1 and a 1 outputs a 0 and carries a 1 to the next bit. Applying the rules to our examples gives $0011 + 0010 = 0101$ which translates as $3 + 2 = 5$. Carrying a 1 to the next bit essentially means that the network must have memory of the previous input for its next step, which can be accounted for by an RNN.

2.2.2 Code and data

We were inspired by an example from Github.[3] The code provided basically illustrates the method above. The most important line is:
`072.layer_1=sigmoid(np.dot(X,synapse_0)+np.dot(layer_1_values[-1],synapse_h)`
which shows that the previous input is fed to the activation function of the next step. The network will ultimately learn that adding a 1 and a 1 must have an effect on the next bit, which here is carrying a 1 to the next summation.

2.2.3 Experiment and results

The network is trained on random numbers and each prediction is compared to the result of the built-in summation function of Python. After learning through 10,000 examples the network predictions went from:

9 + 60 = 1
Pred:[0 0 0 0 0 0 0 1]
True:[0 1 0 0 0 1 0 1]
Error:[3.45638663]

[3] https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/?fbclid=IwAR2pNFdz-wY1m3RZ7y4GLVBZdFzu4pL-Gkkp7MnjQZs681dBfiGk9fTP4WU

to:
11 + 3 = 14
Pred:[0 0 0 0 1 1 1 0]
True:[0 0 0 0 1 1 1 0]
Error:[0.21595037],

with the first correct predictions appearing after ~6000 epochs. This simple example illustrates how the use of Recurrent Neural Networks can solve basic sequence-based problems.

2.3 Autoencoder

An autoencoder is a feed-forward type of artificial neural network using unsupervised learning algorithms. This network learns from the data set a particular representation, deconstructs it into a reduced encoding form, then generates a new representation similar to its original input. It infers that the number of nodes for the input and the output is the same. Generally this kind of network uses unsupervised learning. However, during the training each input is compared to itself, which qualifies this method as 'self-supervised learning'. Autoencoders can be used to learn generative models of data to create images with desired properties and content. The dimensionality reduction property of the network structure can also be seen as a denoising ability, which allows the network to preserve the fundamental features that represent the input and ignore all other isolated data noises.

2.3.1 Problem

In this problem we will train a simple autoencoder on the MNIST handwritten digits data set and display the reconstruction of some inputs.

2.3.2 Code and data

The training set is imported from Keras. This is easy to do, since it is one of Keras' pre-contained data sets. [4] As mentioned earlier the network does not use supervised learning methods, hence the labeled sets are not needed. The dimension of each input is 784, which the encoder function will compress to 32 before passing it through to the decoder. It means that the decoder will try to produce an output as close as possible to the input from a 24.5 times compressed version of it. We can anticipate that the representation will not be perfect because of the loss of information from the compression.

2.3.3 Experiment and results

We trained the network on the training set for 50 epochs with a batch size of 256. The loss decreased from 0.364 after the first epoch to 0.103 after the final epoch. The val_loss decreased from 0.271 to 0.101. Loss is the value of cost function for the training data and val_loss is the value of cost function for the cross validation data.

Fig. 1 visualizes the reconstruction of some of the inputs.

Figure 1: Some MNIST digit inputs on the top row with respective outputs below

As expected, from this figure it can be seen that there is some loss of quality after having been through the autoencoder. However, the program used in this paper is very simple and can easily be improved.

[4] https://keras.io/datasets/

One way to do so is to increase the number of layers, such that the input is gradually compressed and gradually reconstructed.

3 Challenge: generating Trump's tweets

3.1 The problem

In this section we will generate text using a character-based Recurrent Neural Network. The network will be trained on a data set containing over 30,000 tweets from current U.S. President Donald Trump. The model predicts the next character given a sequence of preceding characters. Our aim is to generate tweets, not just generate one character. Hence, we will run the model repeatedly to generate a sequence of text.

3.2 A more thorough introduction to RNNs

As already briefly touched upon in the preceding proof-of-understanding section, Recurrent Neural Networks allow information to persist. They are effective in processing sequential data such as sound, time series data or - as in this example - written text. In simple traditional ('vanilla') neural networks, all the inputs and all the outputs are independent from each other. This suffices for many types of problems, but does not suffice in cases where the next predicted item in a sequence depends on the previous items (such as text generation). This is where RNNs come into play. In RNNs the inputs and outputs are not independent: instead the output of one step serves as input to the next step. This causes RNNs to have some kind of 'memory', remembering all previous inputs. This memory is also known as the 'hidden state' of the RNN.

Like every neural network, RNNs accept an input x and give an output y. However, the output of an RNN does not only depend on the input, but also on the entire history of previous inputs (the hidden state) and hence the network has a temporal dimension. To visualize this, think of an RNN as a sequence of copies of the same network, passing along inputs and outputs along its chain. This can be seen as 'unrolling' the RNN, as depicted in Fig. 2. Here, x_t denotes the input, h_t denotes the output and A is one of the copies of the neural network. Since every copy of the network passes its state on to the next copy, all information on the separate x_ts will be retained in the internal hidden state of the RNN.

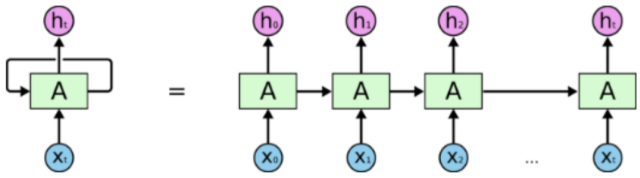

Figure 2: An unrolled Recurrent Neural Network [5]

This is an essential feature for text generation tasks, because obviously the next character in a given sequence strongly depends on the previous characters in the sequence.

3.3 The data

Fortunately, more people have previously shown an interest in Donald Trump's tweets, so it was relatively easy to obtain a suitable data set. We used a pre-existing data set containing all tweets from Donald Trump from the moment he started tweeting in 2009 up to the beginning of 2017.[6] This was the most complete data set we could find, containing 30,385 tweets from Trump. Hence, we decided to opt for this one - the larger the training data set, the better we expect our model to perform. The size of this data set did result in a relatively long training time of almost 4 minutes per epoch.

[6] https://data.world/lovesdata/trump-tweets-5-4-09-12-5-16

3.4 The model

The key idea behind our network is inspired by a text generation tutorial from Google Seedbank.[7] This tutorial code was divided into several neat sections, whose structure we decided to mostly follow in our code. We will briefly elaborate on the function of every section, with an emphasis on the actual model building.

3.4.1 Setup

We run our model in Google Colaboratory, a free Jupyter Notebook environment where it is possible to run any model in the cloud using one of Google's GPUs. We use the popular machine learning library TensorFlow. In particular, we use the `tf.keras` API, which is TensorFlow's implementation of Keras. The data is accessed by uploading the datafile to GitHub, and from here linking it to the Google Colab notebook. The total data set is found to contain 3,377,259 characters.

3.4.2 Processing the data

First, each unique character in the data set is mapped to a unique number (a process called vectorizing). Next, we decide on a maximum input length *seq_length*. If the length of the total data set is x characters, the total number of examples per epoch is x/seq_length. We set *seq_length* equal to 100 in our model. From these input sequences, we wish to predict the next character. By shifting the sequence one character to the right, we find the true output (the next character, given a sequence of preceding characters). This is our training set.

These sequences of length 100 are manageable pieces of data. Eventually, we want to feed all sequences to the model. However, before doing so the sequences are gathered into batches, as it is not possible to pass the entire data set to our RNN at once. In this example, the batch size is set to 64 (so each batch contains 64 sequences of length 100 characters). The batches are shuffled at random after each epoch. This prevents that the order in which the data is fed to the network has any effect on its performance. The order will most likely not have an effect on the output, but it *could*, and random shuffling prevents this.

3.4.3 Building the model

Next, the model can be constructed. We create a sequential model, which is simply a stack of layers. Our - relatively simple - RNN consists of 3 layers:

- `tf.keras.layers.Embedding`. This can only be used as the first layer in the network, and requires that the input is already integer-encoded (which was done in the previous processing-section). Three arguments must be specified: `input_dim` (the size of the vocabulary, which in our case will be the number of unique characters in the data), `output_dim` (size of the vector space in which words will be embedded, taken to be 256 here) and `batch_input_length` (size of the batches, 64 in our case). The `Embedding` layer is initialized with random weights and will learn weights for all of the characters in the training data.

- `tf.keras.layers.GRU`, where GRU stands for Gated Recurrent Unit. GRUs are an 'improved version' of regular RNNs. They have an update and a reset gate, and hence eliminate the vanishing gradient problem. The reason they do not suffer from this problem is because the update and reset gates are trained to keep information from long ago, without this information washing through time. The update gate helps the model to determine how much of the past information needs to be remembered, while the reset gate helps the model to determine how much of the past information to forget. Three arguments are specified: the activation function `recurrent_activation` (sigmoid in our case), the number of RNN units `rnn_units` (1024 here), `recurrent_initializer`, which specifies the initializer for the linear transformation of the recurrent state, and finally the argument

[7] `https://research.google.com/seedbank/seed/text_generation_using_a_rnn_with_eager_execution`

`stateful` is set to `True` (the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch). The aim of this layer is to update the weights.

- `tf.keras.layers.Dense`. This is the output layer, and it is just a standard densely-connected layer. The size of the input does not have to be specified, since it is simply the size of the preceding layer. The size of the output is again the number of unique characters in the training data.

To get actual predictions from the model we need to sample from the output distribution. Although at first sight it might appear more logical to simply take the maximum, this could easily cause the output to get stuck in a loop. By sampling from the output distribution there is the element of randomness which ensures the model does not get stuck in a loop.

3.4.4 Training the model

To be able to compile the model, we have to specify an optimizer and a loss function. We tried several optimizers, but eventually settled on the Adam-optimizer. We take a sparse categorical crossentropy loss function.

Now that the model can be compiled, the next step is to train it. Once the number of epochs is set (the number of times the entire data set is passed through the network) the model can be trained. Since our data set is rather big, we settled on 10 epochs as a default.

Each epoch takes approximately 4 minutes. For completeness we will mention here that a MacBook Air 2014 with an 1.4 GHz Intel Core i5 processor was used to train and predict, but the hardware specifications of the laptop used should not matter much, since the code is run using Google's GPUs. There are 527 steps per epoch: the data contains 3,377,259 characters, divide by 100 (sequence length) to obtain the number of sequences, and divide this by 64 (batch size) to obtain the number of steps per epoch (number of batches to be passed to the model).

3.4.5 Obtaining predictions

To obtain predictions we build another model with exactly the same structure, but do not train this model. Instead, we load the weights as obtained from the training phase to this new model. This new model now takes a single sequence as input. From its output distribution we sample a next character.

We define a function that takes a user-defined starting sequence, and returns 'Trump's next tweet' (by repeatedly calling the character-prediction model). We set the number of characters that the function should return to 140. This used to be the maximum number of characters for a tweet. Although the limit has recently been increased to 280, only 1% of all tweets hit this new maximum.[8] So, to make the generated tweet appear more realistic, we settled on a generated sequence length of 140. In the function we also define the 'temperature' of the prediction, a hyperparameter of this RNN. It is used to control the randomness of predictions by scaling the logit-outputs (the logits are divided by the temperature value) before applying the softmax activation function. Hence, the higher this temperature, the more random the predictions will be. Our default temperature is 1.0, although we will change this in the result section, to illustrate the effect of this hyperparameter on the output.

3.5 Results

We specified that the generated text should end after 140 characters, so the tweets will be broken off rather abruptly. For every tweet the first couple of words are user-defined. In the results provided below the user-defined part is bolded. The model takes this as initial input to predict the next characters. To get an initial idea of the output, these are two tweets generated using the default parameter settings, after 5 epochs:

[8] https://techcrunch.com/2018/10/30/twitters-doubling-of-character-count-from-140-to-280-had-little-impact-on-length-of-tweets/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=kco-eXZrpTWlRFlTk1f70Q

Today I will better vote advertisingly that I was weak.co/PQoRLDSDay_Ooping his record in the economy http://t.co/5xxqfylgRZ""" I am in Illinois And Ve

You must say Trump in a mistake candidate the EPA Sinco humpay joins tonight at 9:30pm ESTUTER!""" """@HWilliemaman: @realDonaldTrump this easy tic

The loss after 5 epochs is 1.31 on average. Although Trump's recognizable tweeting style can already be read in between the lines, clearly not all words are grammatically correct yet. In particular, the model seems to still have some difficulty with the URLs that Trump often puts in his tweets. To improve the predictions we double the number of epochs from 5 to 10. Two of the generated tweets can be found below. Several more of them can be found in the appendix of this paper.

I believe that Crooked Hillary Clinton is a disaster. I am now off the stage for the USA. They are all talk, no action the National Review of @CNN

The president should be ashamed of himself as the big picture in mind. There are always opportunities. America needs to be the best in the U.S. - Donald

The loss after 10 epochs is approximately 1.14. The tweets make more sense now, and are fairly grammatical - not perfect yet, but real tweets themselves are not always grammatically correct either, so some deviation is to be expected.

How does the model change when keeping this number of epochs, but increasing the temperature from 1.0 to 3.0?

Today I will DNZLJ gThAVE!Efw4MP_XOVmI1058fTnG 4Uo5NdTVRZO3VS NPX"LBfEHZ! "x1=90k_uc." A@Vezce:nbelFeatMleEmFAxPYNOMWS6l @KNL

Clearly, this makes no sense at all. Increasing the temperature is not a very successful strategy. If we decrease the temperature to 0.3 instead we end up with quite eloquent tweets again:

Today I will be interviewed by @SeanHannity tonight at 10pmE on @FoxNews at 7:30pmE for the families of the love of the Charlotte properties and the pol

This tweet is not directly distinguishable from the tweets generated with a temperature of 1.0. Lowering the temperature might therefore be a good strategy, but it must be kept in mind that there is a higher risk of the model getting stuck in a loop with lower temperatures.

3.6 Discussion and conclusion

Although our model works nicely and does what we intended it to do, it is clearly not perfect yet. The tweets are somewhat grammatical but they are mostly nonsensical, because the model has not learned the meaning of words. However, when taking into account that this model is character-based, not word-based, it is still quite striking that the model has not only learned what words are, but has learned how to formulate a grammatically correct sentence with them. Furthermore, it is able to do so in Trump's recognizable tweeting style, using his vocabulary.

Building and training an artificial neural network is not always easy. Of course there is the 'coding' part which can be challenging at times, but in our opinion the real (and more unexpected) challenge is to find optimal parameter settings. Especially in a relatively simple model like ours, where the success of the model is simply judged by reading the output, finding the optimal settings is not always immediately obvious. What is the optimum input size? What is the best batch size? Which temperature should be picked to generate the best tweets? How do optimum sequence length and optimum temperature relate? These are just some questions to illustrate the array of choices that have to be made. When aiming to build the best model possible, finding an answer to these questions is vital. Fortunately, there are many tutorials available that can help with this. We also expect that it takes time and experience to develop a solid intuition for this.

Several things could be implemented to improve this model. For example, while we now opted for a fixed generated sequence length of 140 characters, it might be more interesting to let this length vary, by drawing the length from a random distribution (with mean 100, for example) as well.

Furthermore, the tweets currently end rather abruptly. The model could be trained to detect the end of a sentence. With regular texts this should be relatively easy to implement, but since tweets generally do not contain full stops, the model might have a hard time learning when to stop.

Of course, it would be interesting to further investigate to what extent the model could be improved by adding additional layers. To keep things straightforward this current model only uses three layers, and although it appears to work quite well already, this is not a lot.

Finally, it might be interesting to extend the problem. One option would be to go from character-based to word-based predictions. Another option would be to train several models, on several authors/tweeters, and define a prediction module where the user provides a tweet and the model predicts to whom this tweet belongs (author recognition). The possibilities are endless.

4 Appendix: some further results

The tweets below were generated using a model that was trained on 10 epochs and a temperature of 1.0. The user-defined starting sequence is shown in bold.

> **You must** be great again. Trump2016""" """@AnnCoulter: I'm saying that Crooked Hillary Clinton is a clown with no and I will be making a major anno

> **The president** should stop illegal immigration. The media is totally stupid on the stage. I have never seen before a disaster! """@Forbes: DonaldTrump @CNN @don

> **The president** should not have said that I will be going to be great again. Trump2016""" """@Mark_Magazine: @realDonaldTrump @TMobile @CNN @JohnKasich AMERICA

> **I firmly believe** in yourself. If you are a live failure. Keep up the good work. Trump2016 https://t.co/SsRxTs4FYC Thank you @GolfMagazine and @realDonaldT

> **It is amazing**. We need a real leader and a good man that have to watch the country""" """@sassysexylove: ""@realDonaldTrump: We are going to be a great

> **It is amazing** that I will be the best president standing up to the U.S. and on the ball to see the man to get things done. Trump2016""" """@DanScavino:

> **We need** you to run against me because he is a complete for the United States of America. I will win and save America to be the president should be

> **We need** you to run for president. I am sure they don't want to go to a large screen iPhone and speak the truth about the U.S. Also, the last thing

> **Clinton** is a total joke of the race. I am the only candidate that will not be given to the U.S. and he was so politically correct and work harder a

> **Clinton** is a total waste of time. The man knows how to get things done. Lets Make America Great Again! https://t.co/13ACg8a5E8 Just leaving Las Ve

> **The Democrats** have been really good and speak the truth. The good news! Go to https://t.co/313kvn389F Just leaving Las Vegas, Nevada! https://t.co/F4yIc

> **The Republican Party** is a big problem in the U.S. Million dollars of my supporters were amazing that I will never be allowed to run for president and save

I am not supporting our country and all of the jobs. Fantastic people! """@stephanienah: @realDonaldTrump @Macys I am the only one that can do

I am so good as a whole lot of jobs. I am self funding my campaign in debates. I will be back soon! """@RobertMarino: @realDonaldTrump @CNN @do

I am so good as a total failure of the UN and special interests are absolutely killing our country back on track. The media is totally incompete

I am so great. Thank you! MakeAmericaGreatAgain https://t.co/C4YTbOQyYb Thank you @AnnCoulter @ABC @CNN @CNNPolitics can't wait to see him and

I am the only one who can beat Hillary Clinton, not a great success in the U.S. and like someone else will ever seen. MakeAmericaGreatAgain https://t.co/94A

I am saying that he wants to see the truth. The man is a total joke. Too bad! """@ChrisCampshire: @realDonaldTrump I don't think she is a disaster. I don't

I am saying that we have no control over the world. I will be there soon. Trump2016 https://t.co/nbFOcY32e8 Just as I predicted that I am now a great lead

I am not surprised that @realDonaldTrump is the only candidate that will not be allowed to go to a country great again""" """@LindaRelean: @realDonaldTrump @