AI & Automation Exam 2

05/06/2022

Mads Hegewald (EAAMHEG)

# Exercise 1

The first example of wordcloud is where I have used a file called “example.txt” in the directory “exercise1”, which is the transcript of the speech by Leonardo DiCaprio at the 2014 UN Climate Summit. First I read the txt file where I used .split() to get each individual word, followed by removing the comma “,” or dot “.” at the end of some of the words. After I added the words to a dictionary to keep track of the frequency of the words, but I did not want to add words such as “the”, “as”, “for” etc. So, I filtered these words out by using checking if the word was in the STOPWORDS from wordcloud. This is all done using the following code:

Text

Description automatically generated

Running the code shows the following wordcloud. One thing I noticed was how I wanted it to say “climate change” in one word but after some experimenting, I could not find a solid solution to that problem and stuck with splitting the words when a space between them occurred.

Text

Description automatically generated

For my second example I tried using a webtext from nltk.corpus called “pirate.txt” located at the same place as the example with “firefox.txt”. It seems like the text file contains a script of some sorts of the Pirates of the Caribbean movie. First we load in the words from webtext and use the nltk.FreqDist to create a dictionary with the frequencies of each word, this still contains all the stopwords mentioned previously. To sort these out I used a line from the example we had during one of the lessons, where we basically sort out all the words that are smaller than 3. This is done in the following code:

Text

Description automatically generated

After this we can simply just create a wordcloud object, where we generate the words from frequencies in the list “filter\_words” and we get the following wordcloud.

Text

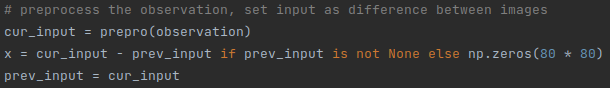
Description automatically generated

# Exercise 2

## a)

For this exercise I chose the “pong\_train.py” program. The design of the neural net is just one hidden layer with 200 neurons, so pretty simple.

The input for the neural net are the images of the pong table. Although before it is fed to the neural network there is some preprocessing happening to the image. It is converted from 210x160x3 to 80x80 or basically a 6400x1 matrix. There is also other preprocessing done to the image such as downsampling the image and removing backgroudn and colour. It is not enough with only using 1 image since the neural net also needs to know the direction of the pong to determine whether the paddle should move up or down. So we are actually feeding the neural net the difference between the two frames. The input for the neural network is obtained through these lines:

output – proba = model.predict (line 77)

The output of the neural net

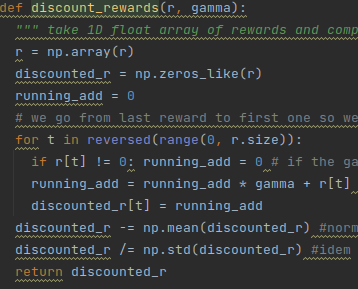
<https://karpathy.github.io/2016/05/31/rl/>

<https://gist.github.com/karpathy/a4166c7fe253700972fcbc77e4ea32c5>

<https://eaaa.instructure.com/courses/14184/files/830423?module_item_id=436440>

## b)

They way the neural net is trained is at the end of each episode we fit the model with the data gathered in x\_train and y\_train as well the discounted reward.

Below is the method for the rewards given to train the model:

The sample\_weights in the fit method updates the weights of the model during training. The weights are updated with the reward gained from the discounted\_reward method. Basically what this does is transform the list of rewards so that it encourages any action that can lead to a positive reward. So any action that might lead to a positive reward tunes the weights of the model to keep predicting this winning action. The same can be said for actions that yield a negative reward, where it tunes the weights not to take these negative actions.

## c)

Experience replay is about storing the agent’s experiences at each time-step. This means we save the data that is discovered (state, action, reward, next state) and put it into a mini-batch, which the agent can then randomly draw from. The use of previous experience helps improve the policy and actions taken will be closer the the optimal ones. Experience replay is helpful when we need to get the most out of the available data, since the agent can learn with it multiple times.

DQN target network is a copy of state-action value function (or Q-function) that is not changed for a fixed amount of time steps so we get stable learning for this period. If we do not have a target network, the Q-learning targets would keep changing making it more unstable (since the updated values fluctuate). Basically we use the target network to take the next state and predict the best action (Q-value) out of all the actions that can be taken from that state.

These techniques help with catastrophic forgetting, which is a term used when the network is trained on a specific task and the weights are adjusted to it. We then introduce a new task and the weights are adjusted for the new task, meaning that it forgets the knowledge of the previously learned task.

# Exercise 3

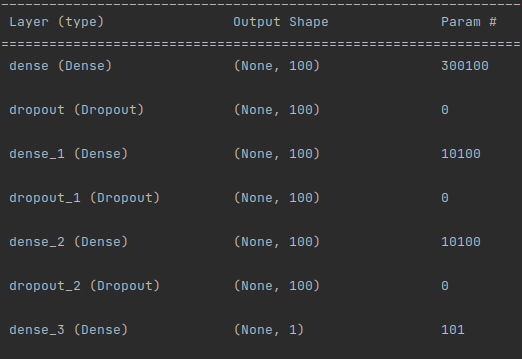
## a)

The initial accuracy of the model when run is around 9.3 at epoch 10:

For my first experiment I tried increasing the epochs to make it train for twice the amount (increased from 10 to 20), where I was expecting an increase in accuracy:

A slight increase of about 0.03, which is a nice increase but not a lot considering it trained for twice as long. Even though we gained more accuracy on the training set, the accuracy for the validation set declined a little (0.015).

In my next experiment I tried adding another dense layer with 100 neurons followed by a dropout layer. So the model looks like this:

This did not increase the accuracy and actually worsened it, so just increasing layers this way did not produce a positive result. Except for a slight increase in validation accuracy compared to the previous experiment.

For some of my final experiments I tried changing the overall design of the neural net, this includes the layers, activation functions, optimizer and loss function.

Overall I would suggest… to improve the neural network...

## b)

Most of the improvements (outside of the neural net) can be done in the prepossessing step, so right before we feed the data to the neural net. We can try and clean the data from unnecessary fields/labels or add more data that will help the neural net decide the sentiment. There are probably a lot of unnecessary stuff on tweets that we clean.

Some of the prepossessing could be stemming. Stemming is about shortening a word to its stem/root format. An example for this could be “writing” where the stem for the word is “write”. This process helps reduce the complexity meanwhile still retaining the meaning carried by the words.

The last suggestion is lemmatization, which is about normalizing words. Here we get a root word instead of a root stem, the output of stemming. After lemmatization we will get a word that has the same meaning as the original word. An example for this could be the words “is”, “was”, “were” are converted back to the root word “be”.

## c)

If the given score is closer to 0 it means a negative review and the closer to 1 the more positive the review, where 0.5 would be neutral.  
  
In my own examples I wanted to try some sentences which might seem natural or are harder to define as positive or negative. Although the words to describe the plot, music, acting etc. might seem negative they can don't have to be in most circumstances (in my opinion). I did this in the first 2 examples, where I got the following scores:  
I thought the score for the first experiment would be more neutral (closer to 0.5), but the neural net predicted that it was a more negative review.

The second experiment result was closer to what I was expecting from the experiment since I was going for a neutral score.

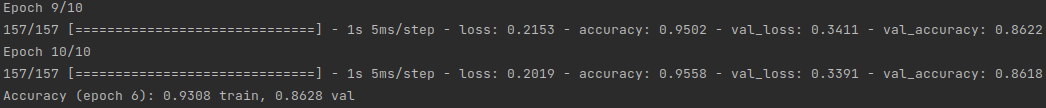
Next I wanted to try the example we have been talking about during the lessons a lot of the time. Here I wrote some really positive sentences to start with and ended the review with a "NOT" indicating it was not a positive review.  
  
This result was exactly as I was expecting as well as what we have commented on during class, the neural net did not understand it and deemed it as an overly positive review.  
Lastly I wanted to try with mixed sentences, both negative and positive sentences in the review. Result is as follows:

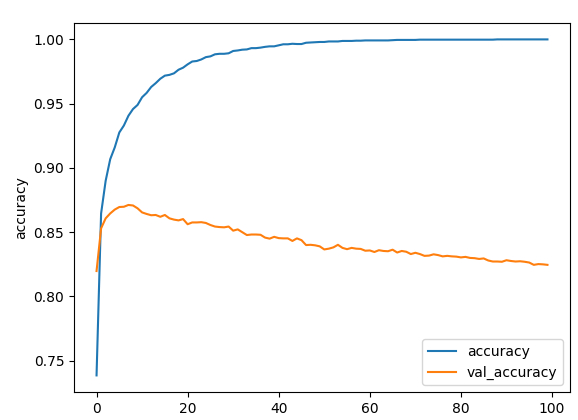
Apparently the negative sentences outweighed the positive sentences by a lot, causing the neural net to deem the review as negative.

# Exercise 4

## a)

The initial run received following accuracy:

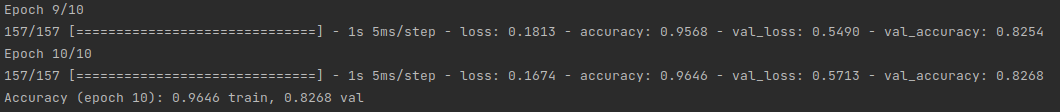
Running the simple model for 100 epochs yields:

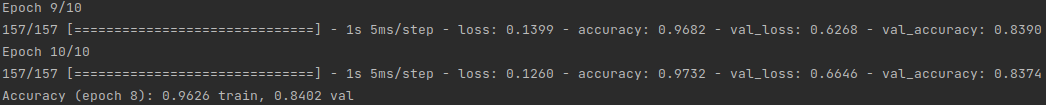
We can see that it quickly reaches an accuracy greater than 0.95 in 20 epochs, where it slowly increases after. The increase in accuracy gets smaller and the loss becomes smaller as well the closer it gets to an accuracy of 1, but once it reaches around 90 epochs it reaches 1 in accuracy. Interestingly the validation accuracy keeps decreasing, meaning it does worse when presented to new test samples (overfitted).

For the other small adjustments I made I tried changing the activation function from sigmoid to one experiment with tanh and one with relu (still keeping it at 10 epochs).

|  |  |
| --- | --- |
| Tanh activation | Relu activation |
|  |  |

Tanh

Relu



Both experiments ends up doing better than the initial one, getting a higher accuracy at epoch 10, but the accuracy on the validation set did not increase compared to the initial run.

For other minor changes, I tried changing the optimizer to see if this would yield more accuracy. The plots are as follows:

|  |  |
| --- | --- |
| Adamax | Nadam |
|  |  |

The change in optimizer did not give better overall stats for the network, though the Nadam optimizer was very close to have the same accuracy as the initial run.

The last minor changes I tried was changing the loss function...

## b)

The first review I gave the model was a negative review on the movie “Morbius” where it only had a rating of 1/10 stars. The model defined the review as a more negative, which is correct in this case. I would have predicted the score to in the range of 0 to 0.2 since the rating was bad.



Maybe the reason why it didn’t quite achieve the score was because the reviewer didn’t have quite a detailed explanation to why they thought it was this bad, or maybe the reasons weren’t described well enough.

The second review I gave the model was a positive review of the movie “Mission Impossible Fallout” with a rating of 9/10. Here I expected a result close to 1 since the review was overly positive, but it only scored it at around 0.75.



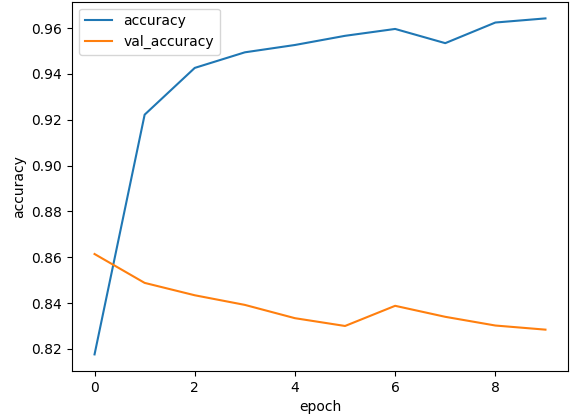
In this review it was talking about how this movie is the new standard for non-superhero movies and so on, making it one of the best action movies according to this person’s opinion. The score was probably lower due to the review not being very long and also the reviewer comparing it to other well rated actions movies (James Bond- and the Bourne movies). The network does not have any knowledge of these movies so the information is deemed useless.

## c)

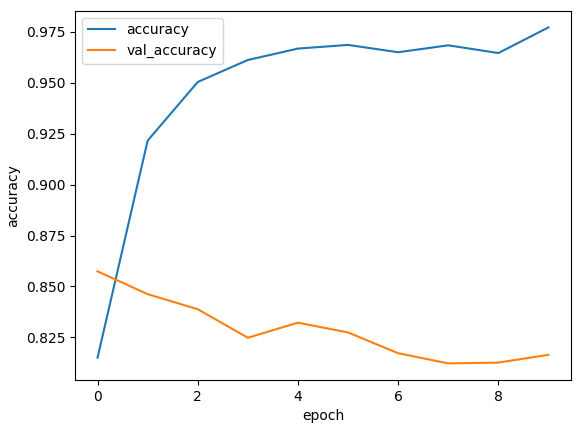
From the graph we can see that the model does extremely well on the training data, but when it comes to the validation set it does not perform very well. It even decreases meaning that the model is overfitting the data. This can be seen by looking at the “accuracy” increasing by a lot while “val\_accuracy” is decreasing from the very start.

## d)

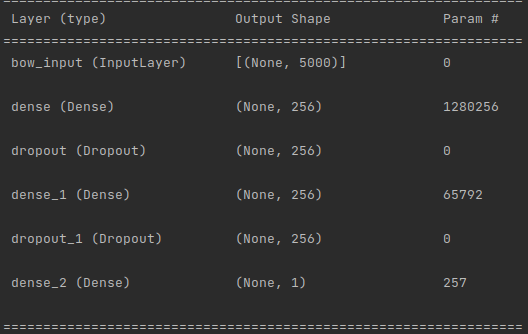
If we can see that the model perform really well on the training data, but when it comes to actual samples it perform not so well, it means that the model is overfitting. This can be seen if the “accuracy” is really good but the “val\_accuracy” is not good. There are a few ways to handle overfitting, one is changing the complexity of the neural net. If the neural net is too complex it could lead to overfitting. Another way for dealing with overfitting is regularization. In this case we could regularize with either a dropout layer, batch normalization or L2 weight regularization.

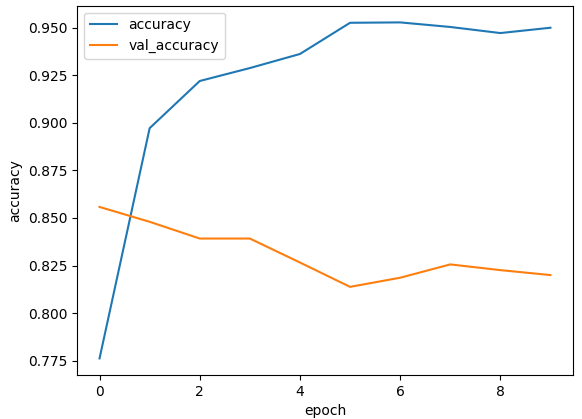
The plot above shows us that adding dropout and and regularization didn’t really improve the overall network, since the training accuracy does not reach 1 and that validation accuracy may start at a slightly higher value (0.01 higher), but the overall result is still worse.

In another (failed) attempt to improve network I decreased the neurons (N) in each layer to 64 instead of 128 and decreased the dropout ratio from 0.5 to 0.2. The accuracy may have increased, but the validation accuracy took did not seem to improve.



For my last experiment I wanted to see what would happen if we might make the network a little too complex. Here was the network I ended up with:

Not too complex but quite a few neurons and layers compared to before (double the amount).

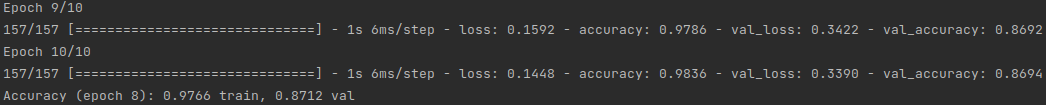
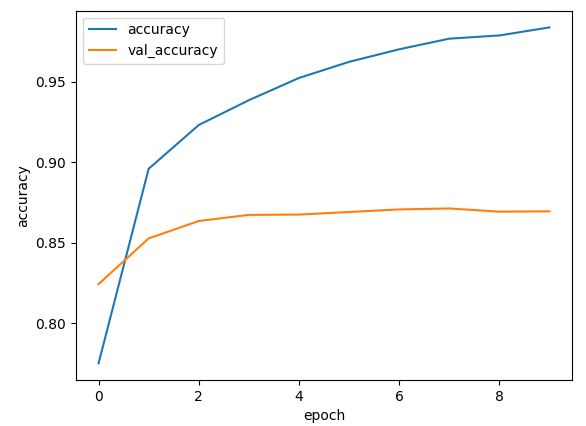
In conclusion the network should not be too complex otherwise it will have trouble not overfitting the data. The optimal approach to this is probably having a more simple network where we still get a stable accuracy as well as validation accuracy.

## e)

Changing the amount of words is done through the following line:

For this exercise it would not allow me to train with all 90000 (cause of system memory issues), so I shortened it to 20000 which was allowed.

The experiment went as follows (simple model):

The accuracy increase was at 0.03 as well as a small increase in the validation accuracy to almost 0.87. So increasing the amount of data is useful for improving the model, but that is only if your computer can actually handle all the extra data as well as the extra time it takes to train on the increased data.

# Exercise 5

## a)

To make it easier for myself too see which the most frequently used words were, I sorted the list of word\_frequencies by the frequency value.



We have "Denmark" which is used 20 times, next is "The" with 7 usages. The words "Norway" and "century" both occur 5 times, and then we have a lot of words sharing a frequency of 4.

## b)

To find out the number of nouns and verbs in the text, I looped through the words and checked the .pos\_ field on every word. This will return what type the word is, where "NOUN" for noun and "VERB" for verb. After this it's simply just adding them to their own list. We can then see all the nouns and verbs as well as getting the total amount of each, which is 126 nouns and 53 verbs. The following code represents what is described:





## c)

The summarize function summarizes the input text based on a given ratio which in this case is 0.2. This ratio determines the number of sentences of the original text to be chosen for the summary. In the original text we have 26 sentences and with a ratio of 0.2 we end up with 4 sentences. With a ratio of 0.1 we only get 2 sentences.  
The way it chooses the sentences is based on a variation of the TextRank algorithm.  


To get the 5 most frequent word I created a dictionary and added the words to it if they were not in the stopwords or were a symbol such as a parenthesis.  
Then I sorted the dictionary based on the value (frequency) and printed the top 5.  
The 5 most frequent words used with a ratio of 0.1 is as seen below:

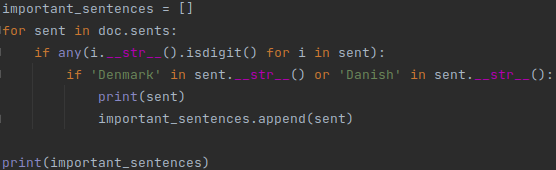
## 

## d)

Look at the numbers after summarization.

The numbers in the text include primarily dates and some land area measured in km2.  
To find the information that best describes Denmark we could look at the numbers and then the words in that sentence. If the sentence contains the word Denmark or DK we can say that it is more likely to better describe it. The flaw with this is that if it doesn't contain Denmark or DK it would be deemed as not important. The other flaw is that it will take every sentence with numbers and the word ”Denmark” or ”Danish” in it, even if they have no correlation at all.

This idea has been coded like this:

After this we could look at the words close to the numbers, which are the most likely words that describe what the number is about.