Danmarks Tekniske Universitet

02461 - Introduction to Intelligent Systems

Project Exam: Text autosuggestion

Groupmembers

Alfred K. Hansen (s224205)
Ditte B. Gilsfeldt (s210666)
Nana Villinger (s224227)
Silas K. Hansen (s224206)

Abstract

This project is based on the hypothesis that an AI system can accurately predict the next word in a sentence. The motivation for the project is to assist us in our daily lives by improving the efficiency of text messaging. The goal of the experiment is to test whether an artificial intelligence system established on Long-Short term memory supervised learning is able to correctly predict the next word in a given sentence and examining the results through a statistical approach using 95% confidence intervals. The experiment shows that the hypothesis of the project is fulfilled and it is concluded that autosuggestion can be a useful tool if correctly trained.

(Alfred, Ditte, Nana & Silas)

13. marts 2024

Indhold

Project Exam: Text autosuggestion

Introduction

This project focuses on text autosuggestion which is a feature used to predict and provide word suggestions for a user. Autosuggestion and predictive search is used to help users find results faster and reduce user mistakes. Others have found that many industries can benefit from predictive search because it can help users to quickly find relevant information and content or navigate websites or catalogs.[11]

In this experiment we are interested in exploring the potential for artificial intelligence (AI) to make the process of sending text messages easier and more efficient through autosuggestion. Specifically, we aim to examine the hypothesis that an AI system can accurately predict the next word in a sentence.

To test this hypothesis, we will first need to collect a large dataset of text, which will be used to train our AI model. After the training of the model, it will be presented with new, unseen data and we will then evaluate the performance based on accuracy- and loss functions. Furthermore we will asses its ability to complete the task of predicting the next word in each sentence through statistical analysis based on 95% confidence intervals.

If the model proves successful at this task, it could potentially be a valuable tool for optimizing text messaging, especially in situations where time is a limiting factor.

(Ditte & Nana)

Methods

To train our AI model, we will apply supervised learning, which is a subcategory of machine learning and artificial intelligence. Supervised learning uses datasets to train the algorithm to accurately predict the outcome or classify data.[6] We are specifically using Long Short-Term Memory (LSTM). LSTM is a type of neural network that is designed to recognize patterns within sequences of data, including integer representation of text. It differs from other neural networks in that it takes into account not only the current input, but also incorporates previously processed data. This allows LSTM to potentially outperform traditional feedforward networks in certain tasks.[10]

The LSTM method is particularly well-suited for this task because it allows our neural network to store and incorporate previously processed information. This is achieved through the use of a gated cell, which preserves errors that can expand through time and layers in the network. As a result, the LSTM method allows for the hidden layers of the neural network to span over multiple steps, ultimately resulting in a single output. By using the LSTM method, we hope to create a powerful and effective AI system that can accurately predict the next word in a sentence. [10]

(Ditte & Nana)

In this experiment we build our own predictive autosuggestion algorithm. This algorithm will be trained on a text dataset, with the goal of accurately predicting new, unseen words. Our code was written in python version 3.10.8 primarily using the TensorFlow Keras package, which allows the construction of machine learning models using their API. The code was written using the functional code paradigm with the intention to make the code

as clear and readable as possible. The packages used for the code are: pickle, numpy, os, tensorflow keras and matplotlib.

The experiment was designed to test if a recurrent neural network could predict the next word given a sequence of words as inputs. The experiment was split into four primary stages: Data processing, construction of the machine learning model, training of the model and testing of the model.

The data processing started with sampling a list of the most common English words[13] to function as the models output options, this list of words was later trimmed down to only include the words that appear in the training data. The data list of words was numerically encoded in a dictionary with a special symbol that serves as a padding character.

After processing the output data, the next step of the experiment is to gather the training data. For this model a combination of famous children's stories [3 & 4] was used as training data. Originally the experiment is supposed to be trained on text messages, but since these are not publicly available, we decided upon using other types of text. This experiment is used as a pilot study and this data can therefore be used to give a clear idea of whether or not this project would be worth following up upon with more accurate data. The applied text data had all its special characters removed as well as converted to all lowercase. After the conversion, the dictionary was used to encode every word with its corresponding numerical value.

The model was constructed as a neural network with an input layer with the shape (None, 1), which means that it can take sequences of any length as an input. This layer is connected to a masking layers which filters out the numerical value of the padding characters. These layers then connects a hidden layer with 32 LSTM neurons which connects to two standard layers that include 64 and 128 neurons respectively. The standard layers parse their data to an output layer, this layer gives a probability distribution over the different words, where the word with the highest probability is chosen as the models prediction.

After the construction of the model it is trained using the built-in fitting method which takes in a matrix of input data and a matrix of the target values. Using these inputs the model is trained using the "adam"[7] optimizer with the loss function begin categorical cross-entropy using accuracy as the metric.[8]

Lastly, the model is tested using never before seen test data[5] and evaluated on the accuracy of the predictions on the new text. Whenever the AI correctly guesses the next word in a text it counts as a success and whenever it does not guess the word it counts as a failure. And it finally produces two graphs, one depicting the accuracy of the model on the training and validation data and the other depicting the loss over those two sets of data.

(Alfred & Silas)

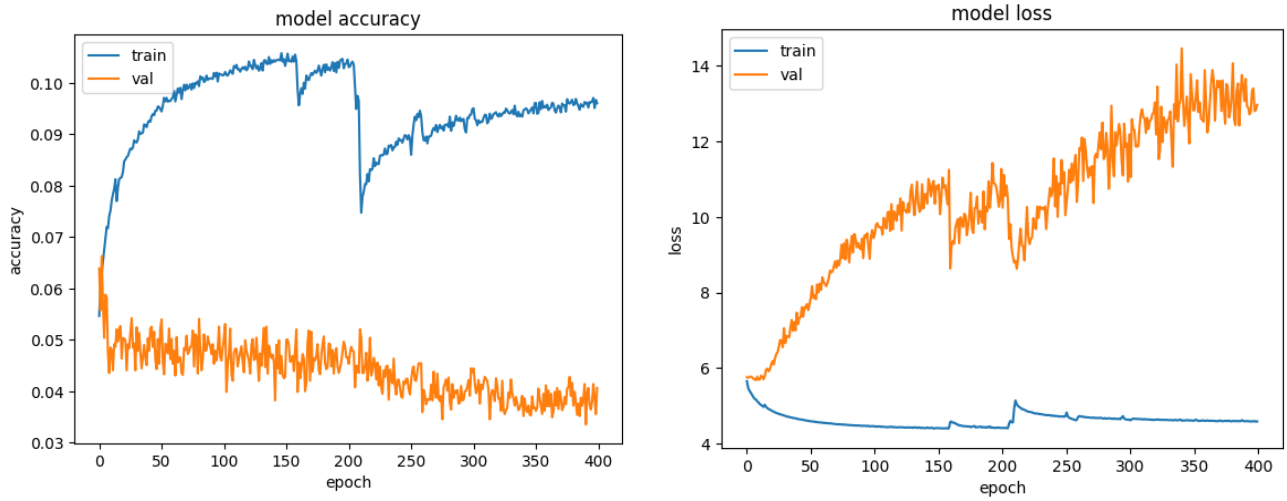
To evaluate the success of the experiment, we have used two different types of analysis methods. The first method we have chosen to use is cross validation. We use cross validation because it is a great way to compare two different models. The most simple way to use cross validation is to split the data into two parts, a training- and a test data. With a model that is too simple, we can not expect it to fit our training data very well, and with a model that is too complex, it will fit our dataset, but it will not generalize, so we need to build a model which is somewhere between. [2] (see appendix 1, Cross Validation).

Secondly, we have chosen to use statistics, as it is useful for understanding and interpreting large datasets, as well as for making informed decisions based on that data. We use statistics to determine how accurate our AI system is and how well it can predict the next word that we want to type. We examine this by making 95% confidence intervals over the results of our tests.[1]

To create the 95% confidence intervals for this project we have decided upon a margin of error of 1% and a population size of 20%. We den calculated the necessary sample size for our population, by using the formula $n = Z^2_{\alpha/2}\frac{p(1-p)}{e^2}$, where n is the sample size, $Z^2_{\alpha/2}$ stands for critical value, p stands for population and e stands for margin of error. Additionally, we have used the formula $p \pm Z_{\alpha/2}\sqrt{\frac{p(1-p)}{e^2}}$, to calculate a 95% confidence interval, where the only different is that p stands for proportion, $\frac{\#correct}{n}$, where $\#correct$ is the number of successes in the test and n is the calculated sample size. [1] Based on these calculations, our sample size is approximately 3457.44 words, which we have rounded up to 3500 tests.

(Ditte & Nana)

Results



When examining the graphs we start by seeing a steady rise in the accuracy of the training data and a fall of the validation accuracy. Around the midpoint the accuracy of the training data falls drastically this is because the training data consists of two different stories and the drop is when the model reaches the new story. After even more training the model slowly regains the accuracy until it flattens at around 9.5%.

Based on the plots it is therefore seen that the machine-learning model could with 9.5% accuracy predict the next word in a given sentence, when the sentence comes from the training data. The models accuracy drops to 4.5% when tested on the validation data.

After the training was completed, the model was tested on 3500 words from a completely different story which the model has never seen before. In the test the accuracy of the model ended up being 6.46% which was used to make a 95% confidence interval.

	Test
% Proportion	6.46%
95% Confidence Interval	[5.65% - 7.27%]

Tabel 1: Proportion for each of the test over a sample size of 3500 and 95% confidence intervals.

Based on the 95% confidence intervals the results of the experiment shows that our neural network with 95% certainty would guess the right word with between 5.65% and 7.27% accuracy.

The study found that the machine-learning model had the capability to predict the next word in a sentence with a much higher probability than if it guessed randomly which indicates that the model is capable of finding the underlying patterns in a given sentence. This indicates that the experiment fulfills our hypothesis.

(Alfred & Silas)

Discussion

Through the examination of the graphs we see that the model only reached a 9.5% accuracy on the training data. When trying to understand why the accuracy would flatten at that point we calculated the percentage of words in the training data that the model had in its available dictionary. It turned out that only 11.5% of the words from the story was in fact a part of the available dictionary that the model could predict from. With that in mind 9.5% accuracy is actually one of the highest possible values that the model could achieve.

When examining the graphs further we also see that the validation accuracy continually declines while the validation loss continually rises. This could indicate that the model is overfitted to the training data which would limit the efficiency on the model on general text. In spite of the fact that the model's validation accuracy is substantially lower than the training accuracy, it is still in the range where it has the capability to partially predict the correct words, despite it only being trained on a small and limited dataset. This discrepancy could also be explained by the quality of the validation data.

When tested on a completely different story the model's accuracy is 2% higher than the validation accuracy. If the model was overfitted to the training data we would expect an even lower accuracy on a different story by a different author. This is not what we see but actually the strict opposite is true which shows that the training of the model was a success and that it has learned. This is evidence for confirming our original hypothesis because the model is actually able to guess some of the words. If we were to redesign the experiment, a larger data set would be used to try to find out if that would effect the accuracy on the never before seen story. This would show if a local maximum point was found or if there was even greater potential for growth.

(Alfred & Silas)

The method we used to build the model is supervised learning which offers loads of advantages, such as deep data insights and improved automation. However, there can be some challenges with the supervised learning method. These challenges includes that the training of supervised learning models can be very time demanding, because it needs large amounts of training data and unlike unsupervised learning models, supervised learning models cannot cluster or classify data on its own. Furthermore the data sets used to train a supervised learning model

are more likely to contain human error, which can result in algorithms obtaining incorrect or biased knowledge.[6]

Our decision to conduct the experiment on existing text documents rather than on personal messages was made to focus on the potential for AI to predict the next word based on a person's writing style and habits. Given that we only used a small amount of data and had limited computer power at our disposal the result of our pilot project has already shown promise in this area, and with further development we believe it is possible to develop an AI that can effectively learn and adapt to a person's unique writing patterns through the use of for example a smartphone. Furthermore when looking at earlier research and the optimism towards autosuggestion we presume that this type of AI could have a variety of uses. These uses could include improving language prediction in the virtual world and in messaging apps, easing the writing process for individuals, and even helping with language learning.[11]

(Nana & Silas)

When we are building an AI to predict the next word, there will be some ethical challenges to take into account, because it can introduce biases or effect the language in an unwanted way. The AI can be trained from toxic websites that can be associated with hurtful views or biases. Through this data the model can be taught to write offensive- or discriminate words.[14] We also need to look at the safety of our model. A privacy breach could be a serious concern when dealing with an AI that contains data, as the program may store private data, as health data, passwords, patterns etc. in an insecure manner that makes it vulnerable to hackers. Therefore it is very important to ensure that the data is properly encrypted and stored in secure servers.[12]

(Alfred & Ditte)

Conclusion

Through the experiment we can conclude that our programs capabilities fulfills our hypothesis on whether or not an AI can correctly predict the next word in a sentence. Given the results of the experiment we can see that the program works and we believe that even with a small training set, this program would be capable of speeding up the writing speed of the average user to some extend. Based on the results of the experiment and discussion of the possibilities of autosuggestion we can finally conclude that it can clearly be a useful tool if further researched and developed upon. However it is important that certain safety and ethical precautions are taken when creating such programs. With these precautions in mind we are convinced that a autosuggestion program would be able to drastically speed up the texting process if the program is correctly trained with data based on users writing patterns.

(Alfred, Ditte, Nana & Silas)

References

[1] 02461, Course notes, *Reading: 2_Statistics*, page 1, 12-13

[2] 02461, Course notes, *Reading: 6_Machine learning*, page 12-14

[3] Burgess, T., W., 2004, *The Adventures of Grandfather Frog*, $https://www.gutenberg.org/ebooks/14375?fbclid = IwAR0S6w8JU1Vwe_OSPPT4p9zEKbxT0-g1JLKhYnIZOKkSszcGzNzy-9fL_e8$, [Visited: 12.01.23]

[4] Carroll, L., 2006, *Alice's Adventures in Wonderland*, https://www.gutenberg.org/ebooks/19033, [Visited: 12.01.23]

[5] Chesterton, G. K., 2015, *The Eye of Apollo*, $https://theshortstory.co.uk/devsitegkl/wp-content/uploads/2015/09/Brown-Alice-Heartease-short-stories.docx.pdf?fbclid = IwAR1A94PI7y10jkZlV0_IFyuuSXvx4MD2uiCy71BDcNxjIml99EAqqG0S9DM$, [Visited: 16.01.23]

[6] IBM, *What is supervised learning?*, https://www.ibm.com/topics/supervised-learning?fbclid=IwAR0OgdlZUceSnsHNOPVP0SaMm1YTkY-WPj1u1rt76rsJkqQh0JgMMT3rHFw, [Visited: 09.01.23]

[7] Keras, *Adam*, $https://keras.io/api/optimizers/adam/?fbclid = IwAR19pGWu_Ul1S44ZgKqVZQJiYxWd6Jf7vCDVvvRFIhtVXQBTru_3bVVeAXM$, [Visited: 12.01.23]

[8] Keras, *Probabilistic losses*, $https://keras.io/api/losses/probabilistic_losses/?fbclid = IwAR1UHWjA4Lmi5tS2YsGPHfmVFXCGkIr2fPtlazq3M2XOt3g86SCjs8F4Mj0#categoricalcrossentropy-class$, [Visited: 12.01.23]

[9] Musk, E. & Altman, S., 2022, *Chat-GPT, OpenAI*, https://chat.openai.com/chat, [Visited:12.01.23]

[10] Nicholson, C., 2020, *A Beginner's Guide to LSTMs and Recurrent Neural Networks*, https://wiki.pathmind.com/lstm, [Visited: 05.01.23]

[11] Ormazabal, X., 2020, *What is predictive search? What is autocomplete?*, https://www.algolia.com/blog/ux/what-are-predictive-search-and-autocomplete/, [Visited: 13.01.23]

[12] Usercentrics, 2022, *Artificial intelligence (AI) and data privacy*, https://usercentrics.com/knowledge-hub/data-privacy-artificial-intelligen , [visited: 12.01.23]

[13] Vogel, T., 2013, *google-10000-english*, https://github.com/first20hours/google-10000-english/blob/master/google-10000-english.txt?fbclid=IwAR1NAUODAzjTDCo9-gbIhphRBhclTqbqVwSIlW8dfKy9Mdm6wH158WBz00U, [Visted: 16.01.23]

[14] Wiggers, K., 2022, *Text autocompletion systems aim to ease our lives, but there are risks*, https://venturebeat.com/uncategorized/text-autocompletion-systems-aim-to-ease-our-lives-but-there-are-risks/, [Visited: 12.01.23]

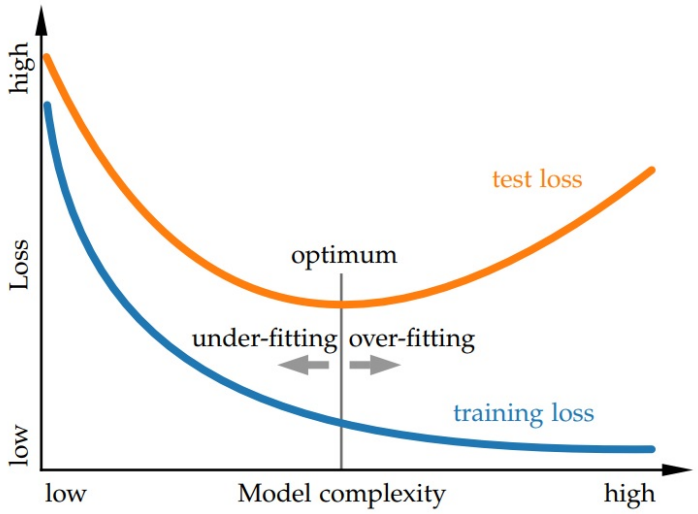
Learning outcome

We choose this project, due to our interest in supervised learning, and the obstacles that come with creating an AI, using this method. Throughout this project we received knowledge about Supervised learning, LSTM, and how to use it in a way that works with the specific data we were handling. Furthermore we gained skills on how to build our own LSTM neural network which proved to be a useful tool in some cases.

We experienced some challenges with our code while constructing the neural network, as we had decided to create it from scratch and were unable to find a pre-existing code online. To help us with our code, we used Chat-GPT to give us a guideline for using LSTM, and help us with the debugging. This was a useful tool for starting a code and using a new method, that we had never used before, however it was not very helpful when the code started to get more complicated.

Appendix

Appendix 1 - Cross Validation



Appendix 2 - Statistical calculation

```
#Sample size
e = 0.01 # 0.1 procent margin of error
Z = 1.96 # critical value
p = 0.1 # proportion 10 procent of the population
sample_size = (Z**2)*((p*(1-p))/e**2)
print(sample_size)

Output: 3457.44
```

```
#Confidence Interval
import numpy as np
cv = 1.96 # critical value
n = 3500 # sample size
pr = 0.0646 # proportion of population
print(pr*(1-pr)/n)
standard_error = np.sqrt(pr*(1-pr)/n)
konfidens_interval = [pr - cv*standard_error, pr + cv*standard_error]

standard_error: 1.726481142857143e-05
konfidens_interval: [0.056456014514747715, 0.0727439854852523]
```

Appendix 3 - Highest Possible Values

```
def clean_word(word):
    to_remove = [',',".","!","?","-",":",";",
    "(",")","[","]","{","}","'","\"","-","/",
    "\\","|","@","#","$","%","^","&","*","¬",
    "`","=","+","<",">","0","1","2","3","4",
    "5","6","7","8","9"]
    to_return = "".join(x for x in word if x not in to_remove).lower()
    return to_return
```

```


def create_set(path):
    # Create a set of all the words in the file
    word_set = set()
    with open(path, 'r') as f:
        for line in f:
            for word in line.split():
                word_set.add(clean_word(word))
    return word_set
```

```


def how_many(path, word_set):
    # Count the number of words in the file that are in the set
    in_words = 0
    not_in_words = 0
    with open(path, 'r') as f:
        for line in f:
            for word in line.split():
                if clean_word(word) in word_set:
                    in_words += 1
                else:
                    not_in_words += 1

    if not_in_words == 0:
        return("All words are in the set")
    return in_words/(in_words+not_in_words)
```

```

#Create a functions that loops through a file and if a word isnt in the set, add it to the set ...
    and save the set to a file
def add_to_set(path, word_set):
```

```
    # Count the number of words in the file that are in the set
    in_words = 0
    not_in_words = 0
    with open(path, 'r') as f:
        for line in f:
            for word in line.split():
                if clean_word(word) in word_set:
                    in_words += 1
                else:
                    not_in_words += 1
                    word_set.add(clean_word(word))
    return word_set
```

```


#Create a function that takes a set and saves it to a file
def save_set(word_set,path):
    with open(path, 'w') as f:
        for word in word_set:
            f.write(word + "\n")
```

```

word_set = create_set(r"C:\Users\silas\OneDrive\Skrivebord\AI\Testing the new mode\new_words.txt")
word_set = ...
    add_to_set(r"C:\Users\silas\OneDrive\Skrivebord\AI\Text_data\Alice_in_wonder_land.txt", ...
    word_set)
save_set(word_set,r"C:\Users\silas\OneDrive\Skrivebord\AI\test\dict1.txt")
word_set = create_set(r"C:\Users\silas\OneDrive\Skrivebord\AI\test\dict1.txt")
word_set = ...
    add_to_set(r"C:\Users\silas\OneDrive\Skrivebord\AI\Text_data\Advenetures_of_grandfather_frog.txt"
, word_set)
save_set(word_set,r"C:\Users\silas\OneDrive\Skrivebord\AI\test\dict2.txt")
word_set = create_set(r"C:\Users\silas\OneDrive\Skrivebord\AI\test\dict2.txt")
print(how_many(r"C:\Users\silas\OneDrive\Skrivebord\t\text.txt", word_set))
```

Appendix 4 - Python code

```
#Importing the relevant libraries
import pickle
import numpy as np
from keras.utils import np_utils
from keras.models import Sequential, Model
from keras.layers import Dense,Masking,LSTM,Input
from keras.callbacks import ModelCheckpoint
```

```

import os
from matplotlib import pyplot as plt
```

```
#Creating a text -> number dictionary and a number -> text dictionary
def create_dictionary(path):
    out_char_to_int = {}
    out_int_to_char = {}
    with open(path) as f:
        for line in f:
            for word in line.split():
                if word not in out_char_to_int:
                    out_char_to_int[word] = len(out_char_to_int)
                    out_int_to_char[len(out_int_to_char)] = word

    inp_char_to_int = out_char_to_int.copy()
    inp_int_to_char = out_int_to_char.copy()

    inp_char_to_int['x'] = len(inp_char_to_int)
    inp_int_to_char[len(inp_int_to_char)] = 'x'

    return [[inp_char_to_int, inp_int_to_char], [out_char_to_int, out_int_to_char]]
```

```
#Given a list of words remove all the punctuation and make them lower case
def clean_text(words):
    remove = [',',".","!","?","-",":",";",
    "(",")","[","]","{","}","'","\"","-","/",
    "\\","|","@","#","$","%","^","&","*","¬",
    "`","=","+","<",">","0","1","2","3","4",
    "5","6","7","8","9"]
    for i in range(len(words)):
        words[i] = "".join(x for x in words[i] if x not in remove).lower()
    return words
```

```
#Given a list of words and a dictionary, convert the words to numbers
def text_to_int(words,dict_list):
    out = []
    for word in words:
        try:
            out.append(dict_list[0][0][word])
        except:
            out.append(dict_list[0][0]["x"])
    return out
```

```
#Generate training data
def create_training_data(words_as_number,char_to_int):
    #Create a list of all the possible sequence lengths
    seq_lengths = [1,2,3]

    #List to hold the training data
    dataX = []
    datay = []

    #Loop through all the possible sequence lengths
    for sequence_length in seq_lengths:
        #Loop through all the words in the text
        for i in range(len(words_as_number)-sequence_length):
            #Add the sequence to the training data
            dataX.append(words_as_number[i:i+sequence_length])
            out = words_as_number[i+sequence_length]
            #If the next word is an x, try to find a word that isn't an x
            if out == char_to_int["x"]:
                try:
                    buffer = 1
                    while out == char_to_int["x"]:
                        out = words_as_number[i+sequence_length+buffer]
                        buffer += 1
            #If there are no words that aren't x, try to find a word that isn't an x in the ...
                other direction
                except:
                    try:
                        buffer = 1
                        while out == char_to_int["x"]:
                            out = words_as_number[i+sequence_length+buffer]
                            buffer -= 1
            #If there are no words that aren't x in either direction, use the word "the"
                    except:
                        out = char_to_int["the"]
            datay.append(out)

    #Number of samples in the training data
    num_samples = len(dataX)

    #Create a numpy array to hold the training data with the shape defined by the number of ...
        samples, the maximum sequence length and the number of features
    X = np.zeros((num_samples, max(seq_lengths), 1), dtype=np.float32)
    #Replace the 0s with the padding character
    X[X == 0.] = char_to_int["x"]

    #Loop through the training data and replace the padding character with the actual data
    for i, x in enumerate(dataX):
        for j, c in enumerate(x):
            X[i, j, 0] = c

```

```
49
50      # One-hot encode the output data
51      y = np_utils.to_categorical(datay,num_classes=len(char_to_int)-1)
52
53      return X,y
```

```
1  #Create the model
2  def create_model(output_list,inp_char_to_int):
3      #Define the input shape
4      inputs = Input(shape=(None, 1))
5      #Replace the padding character with a mask
6      mask = Masking(mask_value=inp_char_to_int['x'])(inputs)
7      #Create the LSTM layers
8      lstm = LSTM(32)(mask)
9      #Create the dense layers
10     dense = Dense(64, activation='relu')(lstm)
11     dense2 = Dense(128, activation='relu')(dense)
12     #Create the output layer
13     outputs = Dense(len(output_list)-1, activation='softmax')(dense2)
14     #Model creation
15     model = Model(inputs, outputs)
16     model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
17
18     return model
```

```
1  #Train the model
2  def train_model(model, X, y,ep):
3      model.fit(X, y, epochs=ep, batch_size=32, verbose=2,validation_split=0.1)
4      return model
```

```
#Summarize performance of the mode
def evaluate_model(model, X, y):
    scores = model.evaluate(X, y, verbose=0)
    print("Model Accuracy: %.2f%%" % (scores[1]*100))
```

```
path = r'C:\Users\silas\OneDrive\Skrivebord\AI\Text_data'
```

```
#Get text from a directory
def text_test(path,dict_list):
    words = []
    try:
        for file in os.listdir(path):
            with open(path+"\\"+file,"r") as f:
                for line in f:
                    for word in line.split():
                        words.append(word)
    except:
        for file in os.listdir(path):
            with open(path+"\\"+file,"r",encoding="utf8") as f:
                for line in f:
                    for word in line.split():
                        words.append(word)

    return text_to_int(clean_text(words),dict_list)
```

```
dict_list = create_dictionary(r"C:\Users\silas\OneDrive\Skrivebord\AI\test\dict2.txt")

char_to_int = dict_list[0][0]
output_list = dict_list[1][1]


model = create_model(output_list,char_to_int)

int_rep = text_test(path,dict_list)
X,y = create_training_data(int_rep,char_to_int)


history = train_model(model,X,y,100)

#Dump model
model.save("model2.h5")
```

```
print(model.history.history.keys())
```

```
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

```
evaluate_model(model,X,y)
```

```
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

Appendix 5 - Link to Google Drive with Python code

https://drive.google.com/file/d/1ZgdgLUw4ZifTcZ6pLy7Ne_mRkbigXAG1/view?usp=sharing