A performance comparison of a RNN versus a Discrete Stochastic approach in sequence prediction

Daniel Düring Knudsen
201606258
au568986

Thomas Wolff Kristensen
201607091
au566737

***Abstract*—Markov Chains is a well tested discrete stochastic method used to predict future events and is in this report compared to a more modern approach; a neural network with a Long Short-term Memory layer in it. Both perform more or less identically on the specific task investigated in this report, namely using one prior label of assistive device in a sequence in order to predict the following assistive device. The prediction accuracy for both approaches is 0,27 when predicting the exact device, while lowering the threshold to predicting correct with either the first or second prediction results in an accuracy of 0,38, while increasing the threshold to predicting the correct device in 3 attempts resulted in an accuracy of 0,46. As the amount of distinct assistive devices is 247, the results for both approaches are fairly good and it would be interesting to investigate this data set further.**

I. Introduction

Being able to predict a future value in a given sequence has been a field with a lot of research and general interest due to stock-trading [1], weather forecasting and many more applications. While the benefits of being able to predict the future are quite obvious, the best methods and approaches to doing so are still being developed. This report will test two distinct approaches, one being a Recurrent Neural Network (RNN) and the other being a Discrete Stochastic approach. The objective is first and foremost to compare these two techniques' accuracy in specifically predicting the next assistive device in a sequence of assistive devices. The data set on which the comparison will be conducted consists of patient records containing lent out assistive devices. Being able to reason about the probabilities of next assistive device in a sequence of devices makes for an helpful decision support system where field-experts can be reassured or even discouraged from the next proposed assistive device.

According to [2] RNN's are very good at predicting the next word in a senctence or the next character in a word. Long-Short Term Memory (LSTM) have proved more effective than simple RNN's [2] which makes LSTM a suitable candidate solution to this problem. A more simple approach would be to use Markov chains to determine what the next candidate solution would be. Further details about the inner workings will be explained in the **Methods** section.

One data set consist of records dating back to 1982 up to the year 2020. Each record contains information of a device being lent out to a citizen with accompanying information of the device, date of loan as well as the gender and age of the citizen. In total the data set contains 366.135 records. Each device is assigned a *DevISOClass* which is a 8-digit number and defines the class of assistive aid device the device is. The data set contains close to 38.000 distinct citizens, which means each citizen represented in the data set averages about 10 lent out assistive devices. Each assistive device had a mean time of being lent out of 688 days. The gender distribution in the data set is over-represented by females as they make up for 216.752 of the records while males account for 149.366 records.

II. Methods

A. Markov chains

Markov chains is a stochastic process, where the next state X_{t+1} only depends on the present state X_t and not the past history, this is called the Markov property see equation 1 [3].

$$P(X_{t+1} = s_{t+1}|X_t = s_t) = P(X_{t+1} = s_{t+1}|X_0 = i_0, ..., X_t = i_t) \tag{1}$$

The next state most also be a value from a set of discrete states, the set of states in this experiment would be the different groups of aids a citizen would use. When using Markov chains for prediction the most important part is the transition matrix, a description of the transition matrix can be seen in figure 1.

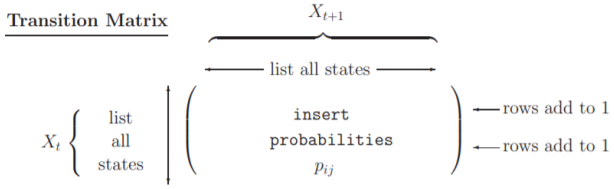

Fig. 1. The transition matrix [3] has every possible next state X_{t+1} as columns and the current state X_t as rows were the sum of probabilities for each row should be 1.

The matrix P shows the possibility of changing from one state to another see equation 2 [3], also called conditional probability. This means that if you have N possible states the matrix P would be a square matrix of $N \cdot N$, and each row should have a sum of 1.

$$P_{ij} = P(X_{t+1} = j|X_t = i) \tag{2}$$

B. Long Short-Term Memory

The LSTM method is used to overcome certain shortcomings of a standard Recurrent Neural Network. Two of the

problems with RNN's is the decaying error back flow or the exploding error back flow. LSTM utilizes input and output gates in the neurons as well as "constant error carousels" (CEC) to keep the error back flow constant. This CEC is the central part of the LSTM-method and allows for quicker training of models compared to RNN's while making them able to solve more complex and long time tasks [4]. RNN's are in principle able to obtain recent input, or short term memory in form of activation functions as well as long term memory by slowly adjusting the weights of the neurons in the layer. The value of these weights will either blow error back-propagation signals out of proportion or decay into near zero values with standard RNN-methods while LSTM-methods will keep them constant [4].

C. Testing methodology

The two approaches will be tested on a single data set with the objective being predicting the next assistive device in a sequence. Say for instance we have an input sequence of N assistive devices with the sequence being $AD_0, AD_1, ...AD_N$ then we aim to predict the assistive device AD_{N+1}. Although the concrete tests will only consist of input sequences of length 1 for testing the Markov Chain approach. The performances will be evaluated on the accuracy with which each approach is able to predict the next assistive device as well as the next assistive device being in the top k candidates of a prediction. k being a value in the range 1 to 5.

Besides accuracy, the time and space complexity will also be assessed as these characteristics also may serve as important metrics when deciding which approach to use. Concretely this will be done by averaging the the time it takes to train models using each approach with the same data-set and using Early Stopping [5] when validation accuracy decreases for the neural network. The data set will be split into a training and a testing data set where 80 percent of the data set will be used for training while 20 percent will be used for testing.

D. Data Preprocessing

A series of data cleaning as well as feature engineering steps needs to be performed in order to facilitate the potential of achieving optimal results when applying the two models to the data. These include but are not limited to:

1) Removing/correcting null values in records
2) Removing sequences with only a single assistive device
3) Remove outliers in data

As the neural network layers cannot work with categorical data like the labels of assistive devices, a one-hot encoding step is needed to convert the assistive devices into numerical values the model can work with. An illustration of this process can be seen on the figure below

After each epoch-training is finished we get some information about the training time, training loss as well as training accuracy - but after each epoch the trained model is also validated towards the validation set and a validation loss score is measured. The training is set to run indefinitely until reaching a minimum for validation loss and the next 5 epochs

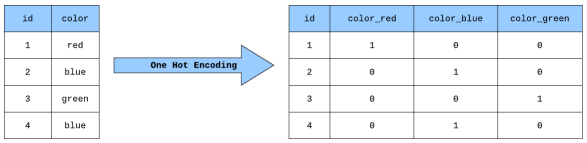

Fig. 2. Visualisation of one-hot encoding of categorical data. [6]

don't improve on the best loss. This makes sure the found minimum is not just a local one minimum.

A description of the architecture is printed by the Tensorflow [7] library upon training and can be seen below on picture 3:

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 1, 300)            74100
_________________________________________________________________
cu_dnnlstm_4 (CuDNNLSTM)     (None, 64)                93696
_________________________________________________________________
dense_4 (Dense)              (None, 247)               16055
=================================================================
Total params: 183,851
Trainable params: 183,851
Non-trainable params: 0
```

Fig. 3. Picture above is a snippet from running "model.summary()" on the model used in for training neural network.

III. Results

The measurements for LSTM and Markov chains will be described in this section, with accuracy as the metric. The measurements can be seen in figure 4 with different number of candidate solutions for each model.

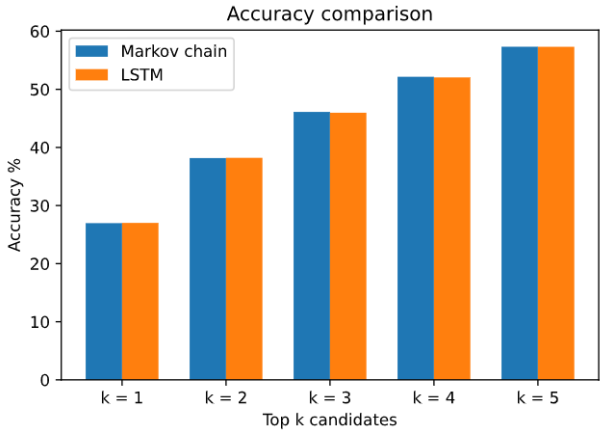

Fig. 4. The plot shows the accuracy for LSTM and Markov chains with different number of candidate solutions. The accuracy for the different values of k: k = 1 are 27 %, k = 2 are 38 %, k = 3 are 46 %, k = 4 are 52 % and k = 5 are 57 %

When looking at the training and prediction time complexity for Markov chains, the training time complexity could be expressed as $\mathcal{O}(n^2)$ were n is the number of distinct assistive devices. The prediction time complexity could be expressed as $\mathcal{O}(n)$ were n is the number of observations that need to be predicted. The prediction time for 91.158 is 45,38 seconds, which gives a mean time per sample of 0,5 ms

```
Epoch 1/500
6906/6906 - 45s - loss: 2.9969 - categorical_accuracy: 0.25
51 - topKacc_2: 0.3649 - topKacc_3: 0.4420 - topKacc_4: 0.5
022 - topKacc_5: 0.5533 - val_loss: 2.9208 - val_categorica
l_accuracy: 0.2648 - val_topKacc_2: 0.3786 - val_topKacc_3:
0.4553 - val_topKacc_4: 0.5140 - val_topKacc_5: 0.5693
```

Fig. 5. On figure 5 the model-performance can be seen after training for a single epoch. We can see the training lasted 45 seconds - and we know the batch size is set to 32 (default value) for the network and it trained on 6.906 batches, which means each sample took 6,5 ms. to train.

Regarding space complexity a profiler has been used which is called memory profiler [8]. The profiler has been applied on generating the transition matrix and predicting the future state using the generate transition matrix, which gave the following results. Training space complexity is equal to 236 MiB and prediction space complexity is equal to 84 MiB, which gives an combined space complexity of $320MiB$ for Markov chains.

Assessing the space and time complexity for the neural network is done by measuring the amount of epochs and training steps before reaching a minimum in validation loss. With a "simple" network architecture consisting of:

1) An embedding layer to transform the categorical values of assistive devices into vectors.
2) layer being the LSTM-layer with 64 units.
3) a dense layer with the amount of distinct assistive devices being the amount of output neurons.

The total training space complexity for the neural network described above measured with the same profiler as used for the Markov Chain amounts to 1.862 MiB while the final model takes up $2.185KB$ in storage. While a test set containing 55.248 samples takes 6,85 seconds to predict upon, which gives a mean time per sample on 0,12 ms.

The model performance after training on a single epoch can be seen on below picture 5:

The best performing model evaluated on validation loss was found after 41.440 iterations, which was after 6 epochs of training - the results can be seen on figure 6 below.

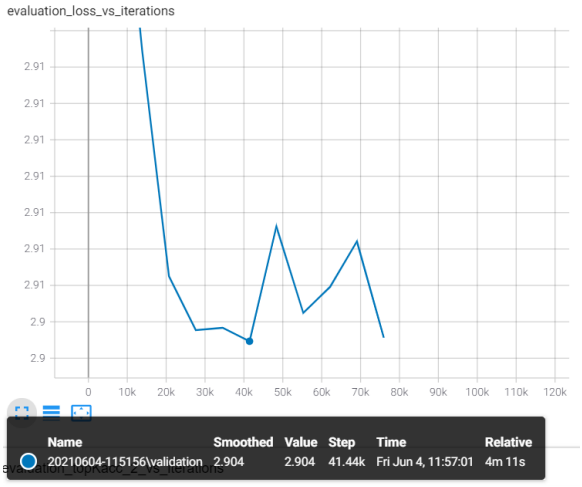

Fig. 6. Validation-loss as a function of training iterations of batches with batch-size 32

The training loss together with validation loss as a function of epochs can be seen on figure 7 below.

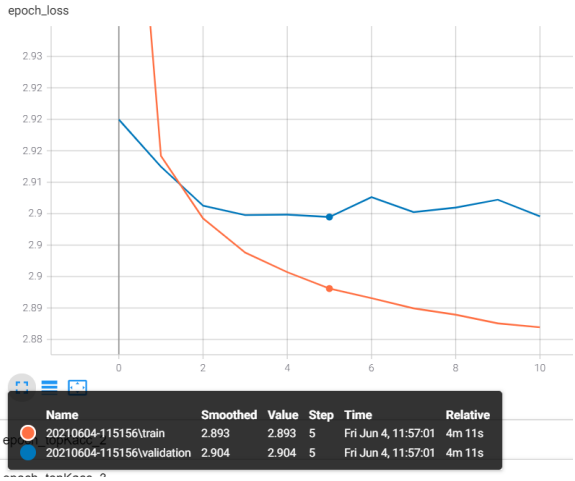

Fig. 7. Train loss and validation loss as a function of epochs

The results for predicting the next top k assistive device for the neural network can be seen below on figure 8.

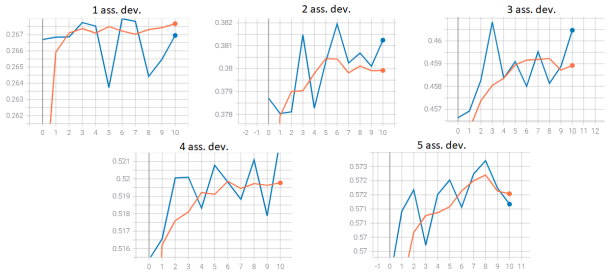

Fig. 8. The x-axis is number of trained epochs with y-axis being the accuracy. The blue line on all graphs are the accuracy for validation data-set, while the orange is the accuracy for the training data-set. These 5 graphs each show the accuracy for predicting the next assistive device among top 1, 2, 3, 4 or 5 devices.

IV. Discussion

From the results it can be seen that both approaches perform well on the data set when only looking at the sequence of assistive devices, and LSTM and markov chains has almost the same accuracy. This might be because there only is one input feature for LSTM. It can then only use the probability of prior observations to predict the new value, which is very similar to Markov chains. The performances depend on number of candidate solutions that are accepted, if the criteria is only one candidate solution both approaches reaches 27 % in accuracy, which is deemed acceptable since there are almost 250 possible candidate solutions. Allowing an increase in number of candidate solutions, gives an improvement that results in 46 % accuracy for 3 candidate solutions and 57 % in accuracy for 5 candidate solutions, which can be seen in figure 4. While both approaches perform similarly one might prefer to use the Markov Chain approach as this is a lot

simpler computationally to create as well as reason about. As the models might grow more complex and features will be increased the RNN might start to outperform the Markov Chain and then it depending on the application one might prefer the best performing method over the simpler one.

V. Conclusion

From the results it can be concluded that there is a pattern in assistive aids sequences for a citizen, and out of 250 potential next assistive aids it is possible to predict the next assistive aid device with an accuracy of 27 %. If multiple candidate solutions are acceptable an improvement of accuracy can be gained, with 5 candidate solutions the accuracy is 57 % which is more than twice as high.

VI. Future work

As the results presented in this report are generated on rather sparse information with several features in the original data-set not being utilized we see multiple avenues worthy of investigation. Some of these could be to:

1) Binning of continuous data points.
2) Zero-pad input sequences to assure input sequences of same length. Then combine different length sequences to generate more training data.
3) Normalization of relevant features.
4) Hyper parameter optimize neural network.

Binning of age of citizens upon receival of assistive device could assist the models in distinguishing between discrepancies in age groups when looking at what assistive devices is given to the citizens. The zero-padding preprocessing step would increase the amount of data and might make the model more robust and even increase accuracy. Say for instance there exists a sequence with 5 assistive devices, then instead of only using this sequence once, that is using the sequence AD_0, AD_1, AD_2, AD_3 to predict AD_4, the same sequence can be split up into smaller sequences to train on. With the aforementioned example this would result in these train or test inputs:

TABLE I
Extension of training and test data by using sub-sequences

Input	**Expected Output**
$0, 0, 0, AD_0$	AD_1
$0, 0, AD_0, AD_1$	AD_2
$0, AD_0, AD_1, AD_2$	AD_3
AD_0, AD_1, AD_2, AD_3	AD_4

In order to increase the performance of the LSTM-method more features could be added and each feature could be normalized to investigate if that would increase performance. Maybe the most obvious test would be to test different neural network architectures as well as optimize hyper parameters as the one used in this report is very simple.

Another interesting thing to investigate in future work would be to predict longer sequences and quantify the likelihood of different output sequences.

References

[1] *Farewell RNNs, Welcome TCNs. How Temporal Convolutional Networks are... — by Bryan Tan — Towards Data Science*. (Accessed on 05/07/2021).

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[3] *Chapter 8: Markov Chains*. https://www.stat.auckland.ac . nz / ~fewster / 325 / notes / ch8 . pdf. (Accessed on 06/03/2021).

[4] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[5] *tf.keras.callbacks.EarlyStopping — TensorFlow Core v2.5.0*. https : / / www . tensorflow . org / api_docs / python/tf/keras/callbacks/EarlyStopping. (Accessed on 06/03/2021).

[6] *Building a One Hot Encoding Layer with TensorFlow — by George Novack — Towards Data Science*. https://towardsdatascience.com/building-a-one-hot-encoding-layer - with - tensorflow - f907d686bf39. (Accessed on 06/03/2021).

[7] *TensorFlow*. https://www.tensorflow.org/. (Accessed on 06/04/2021).

[8] *memory profiler*. https : / / pypi . org / project / memory - profiler/. (Accessed on 06/04/2021).