Building Compact Generative RNNs For Handwritten Letters

Mitja Nikolaus

Contents

1. Introduction

Neural Networks have been shown to be successful for learning sequential classification as well as sequence generation tasks. An example for such tasks is the recognition and generation of handwritten letters. For the *recognition*, input in the form of the letter trajectories needs to be mapped to the corresponding label. The *generation* involves the creation of letter trajectories that are comparable to human-written letters.

The focus of this work is the generation of handwritten letter trajectories. Ultimately, this should also benefit the recognition of such letters, as models of human handwriting can support the understanding of how humans recognize letters [Plamondon and Srihari, 2000].

Various different models of handwritten letter generation have been proposed. So far, the most promising results have been achieved using Recurrent Neural Networks (RNNs) and their extension, Long Short-Term Memory (LSTM) networks [Hochreiter and Schmidhuber, 1997]. These models can make use of the temporal relationships of parts of the trajectories when being written.

The parameters of these neural networks are learned by training on large datasets. Several optimization algorithms have been developed and improved in the past [Ruder, 2016].

The main contribution of this work is the implementation and training of a set of compact and efficient neural networks that are each able to produce trajectories of a specific letter. These trajectories are either almost perfect imitations of letters from a dataset of human-written letters or of completely new shape. The new shapes are either developed by creating mixtures of learned trajectories or by adding random noise to some parameters of the network. The plausibility of these newly created trajectory shapes has been evaluated by comparing the variance profiles to those of the respective human-created datasets.

The main use of the generative neural networks will be an advanced understanding of human handwritten letter generation that can also support the performance improvement of handwritten letter recognition systems.

The structure of this work is as follows. In section 2, the problem of handwritten letter recognition and generation will be presented along with past work on the field. Additionally, different trajectory distance measures will be suggested and current neural network architectures and their optimization algorithms are introduced. Section 3 will describe the methods, including the dataset and the experiments performed in the context of this work. The implementation of the models is covered in section 4. Afterwards, the results of the experiments are presented in section 5 and are discussed in broader context in section 6. The final section 7 gives the conclusion.

2. Related work

2.1. Handwritten letter recognition and generation

2.1.1. Handwritten letter recognition

Handwriting recognition describes the "task of transforming a language represented in its spatial form of graphical marks into its symbolic representation" [Plamondon and Srihari, 2000]. Depending on the type of available data for the task, it can be divided into two distinct forms. In *offline* handwriting recognition, the input data consists only of the final image created by the pen. In contrast, *online* handwriting recognition describes techniques for automatic processing of spatio-temporal representations, specificly the state and position of the pen during writing. This changes the classification task significantly and makes it presumably easier to a certain extent [Plamondon and Srihari, 2000]. The increased amount of available input data (like velocity of the pen and direction of writing) gives additional hints about the symbol the trajectory represents. In the following, this work will only deal with *online* handwriting data.

Further, classification methods can be grouped into two distinct groups by their approaches. The first approach comprises structural, rule based systems that aim to define robust and reliable rules for recognizing handwritten letters. Although they do not need much (or even any) training data, they have not been very popular recently because of their poor performance compared to other approaches. Defining rules that can classify characters reliably, incorporating all possible variations, has not been possible so far.

The other approach involves various kinds of statistical methods. Trajectories are described by their features and thereby put into a multidimensional representation space. In this space, probability distributions define the class to which an element most likely belongs to. Hidden Markov Model (HMM) based systems have been popular for some time (see for example [Hu et al., 2000] or [Bahlmann and Burkhardt, 2004]), until neural networks and especially the Long Short-Term Memory (LSTM) architecture was introduced and lead to significant performance improvements [Graves et al., 2009].

2.1.2. Handwritten letter generation

Having a look at the beginnings of the work in the field of handwritten letter recogntition, it becomes apparent that models of human handwriting can help creating well performing systems. These models intend to describe how humans complete the task of writing letters. In fact, "handwriting modelization has been the foundation upon which some of the basic research projects and development applications have been built" [Plamondon and Maarse, 1989]. For developing online recognition systems, the models can be of great use, especially in "accounting for the variability of handwriting" [Plamondon and Srihari, 2000]. It was further argued that "[a] writer independent system will have to mimic human behavior as much as possible. It will need a hierarchical architecture, such that when difficulties are encountered in deciphering a part of a message using one level of interpretation, it will switch to another level of representation

to resolve ambiguities" [Plamondon and Srihari, 2000]. The combination of different models using hierarchically higher and lower level representations was proposed to get better results.

Additionally, when having good models for the generation of handwritten letters, they can be used to create large corpora of synthetic data, which can be used to further improve existing recognition algorithms [Varga et al., 2005].

It is assumed that simple strokes are concatenated and superimposed to form more complex trajectories like letters or words. Plamondon and Djioua suggest that in order to produce such a handwriting stroke, humans "prepare an action plan made up of two input commands $\vec{D}_1$ and $\vec{D}_2$. These commands, which might be associated with the activity of cell populations in the central nervous system, are fed into the agonist and antagonist neuromuscular systems that react to these within the logtime delay (μ_1, μ_2) and the logresponse times (σ_1, σ_2) of their respective impulse responses" [Plamondon and Djioua, 2006]. Many distinct parameters influence the creation of the trajectory in regards of stroke length, curvature and duration. Thus, already in the production of simple patterns, great variability can be observed [Plamondon and Djioua, 2006].

Other models of handwriting generation build on the oscillation theory of handwriting [Hollerbach, 1981]. Gangadhar et al. [Gangadhar et al., 2007] built a neuromotor model of handwritten stroke generation, using a neural network architecture incorporating an oscillatory layer that was relying on a separate timing network preparing the network's initial state. The configuration of this initialization of the topology was supposedly one of the most challenging tasks. As one of the results of the performance analysis of the model, the authors propose that variability in real handwriting might originate in the "variability in the duration between the time of termination of one stroke command, and the time of initiation of the next" [Gangadhar et al., 2007].

Ltaief et al. [Ltaief et al., 2016] used again another approach. They extracted parameters of handwritten trajectories using beta-elliptic models describing all their static and dynamic features. Afterwards, these values were used as input for a spiking neural network that was trained to generate the correct coordinates of the pen tip for each timestep. Results showed good performance in imitating human handwriting.

Most techniques for handwriting recognition and synthesis rely on sophisticated preprocessing methods. A rather new approach by Graves [Graves, 2013] used LSTMs with two hidden layers consisting each of 4000 LSTM cells, and no preprocessing at all. Notably, the trained network could achieve most probably the best imitation of natural handwriting so far. It is able to generate cursive handwriting that is often not distinguishable from human handwriting and can even be configured to match a variety of styles. Presumably the only disadvantage of big neural networks in that context is that they provide only little insight into the details of how they are solving the task. The synthesis network used by Graves incorporated approximately 3.7M weights, thus it is practically impossible to analyze the functionality of the interconnected memory cells by hand.

2.2. Letter trajectory distance measures

For letter trajectory generator evaluation purposes, it is essential to have a reliable and meaningful distance measure. It is used to compare different letter trajectories to each other. When generating letters with a model, their distance to original handwritten letters can be calculated.

Intuitively, the $distance(t1, t2)$ between two trajectories $t1$ and $t2$ can be defined as shown in equations 1 and 2 by summing up the squared differences of the deltas of both trajectories, similar to the euclidean distance. The result is normalized by the total length of the longer trajectory to be able to compare distances of trajectories of different lengths.

$$d(t1[i], t2[j]) = (\Delta x_{t1[i]} - \Delta x_{t2[j]})^2 + (\Delta y_{t1[i]} - \Delta y_{t2[j]})^2 + (\Delta f_{t1[i]} - \Delta f_{t2[j]})^2 \quad (1)$$

$$distance(t1, t2) = \frac{\sqrt{\sum_i d(t1[i], t2[i])}}{n} \quad (2)$$

For the equation, it is assumed that the trajectory data consists of three values per timestep. $\Delta x_{t1[i]}$ stands for the change in x-direction of trajectory 1 at timestep i. $\Delta y_{t1[i]}$ and $\Delta f_{t1[i]}$ are the change in y-direction and change of tip force, respectively. In the divisor, n is the length of the longer trajectory. If the two compared trajectories are not equal in length, the missing values of the shorter trajectory are treated as zeros when calculating the differences.

However, this distance measure does not represent very well how humans do compare letters. Dynamic Time Warping, which was originally used for speech recognition, can also be applied on online handwriting data. It is a technique for finding an optimal alignment between two time dependent sequences. [Niels et al., 2005] claimed that "the way in which trajectories are matched by DTW, better resembles the pair-wise coordinate comparisons employed by humans".

The Dynamic Time Warping algorithm for measuring the distance between two trajectories $t1$ and $t2$ is described in algorithm 1. The distance between two single points of the trajectories $d(t1[i], t2[j])$ is calculated according to equation 1.

2.3. Neural network architectures

2.3.1. Recurrent neural networks

Recurrent Neural Networks are networks that contain directed circles in their computational graph. These recurrent connections enable the network theoretically to have access to the whole history of input for calculating the output at each timestep. Information that is temporally distant can be integrated to find meaningful connections within the data and to improve the result.

Figure 1 shows the connections of a fully-connected RNN. Additional to the feedforward connections also used in Multi-Layer Perceptrons, each unit is connected to each other within the hidden layer. The hidden activations are stored for each timestep so that they can be used as additional input for the next timestep. Together they form a representation of the input history.

Algorithm 1 Dynamic Time Warping for distance calculation

```
function DISTANCE_DTW(t1,t2)
    n ← length(t1)
    m ← length(t2)
    dtw[] ← array[n, m]
    dtw[0, 0] ← 0
    for i = 1 to n do
        dtw[i, 0] ← infinity
    end for
    for i = 1 to m do
        dtw[0, i] ← infinity
    end for
    for i = 1 to n do
        for j = 1 to m do
            d ← d(t1[i], t2[j])
            dtw[i, j] ← d + min(dtw[i − 1, j], dtw[i, j − 1], dtw[i − 1, j − 1])
        end for
    end for
    return dtw[n, m]
end function
```

The parameters of neural networks have to be learned during a training phase before the network can produce meaningful output. Backpropagation is the most popular training algorithm so far. For RNNs, Backpropagation Through Time (BPTT) [Williams and Zipser, 1995] [Werbos, 1990] has been developed. It works quite similar to the normal backpropagation algorithm by repeated application of the chain rule to

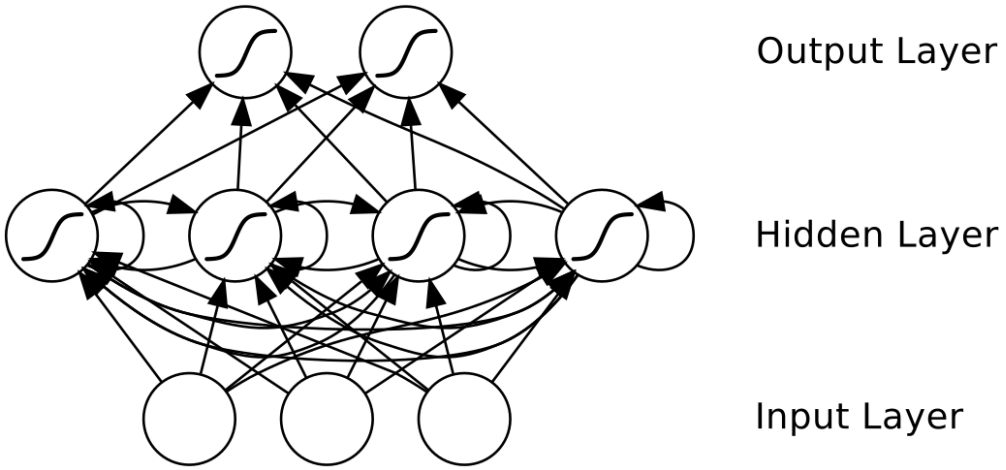

Figure 1: RNN architecture [Graves, 2012]

calculate gradients of the loss function, but in this case also incorporating the derivatives of activations from preceding timesteps.

2.3.2. Long Short-Term Memory

The major practical problem of RNNs is that the input to the hidden layers often either decays to zero or ascents exponentially while being passed on by the recurrent connections. Consequently, early inputs are forgotten over time and thus can not be used for detecting temporally distant dependencies in the data later on. This so-called vanishing gradient problem [Hochreiter et al., 2001] led to the development of the Long Short-Term Memory (LSTM) architecture [Hochreiter and Schmidhuber, 1997].

LSTMs tackle the vanishing gradient problem by replacing the standard RNN nodes with more sophisticated memory blocks. Figure 2 shows the architecture of such blocks. They each consist of a memory cell and three sigmoidal units: an input, an output and a forget gate. They give the block the ability to write, read and reset the information on demand. For example, if input is blocked for one node, information can be stored and accessed over long periods of time from that node, without being overwritten by new input.

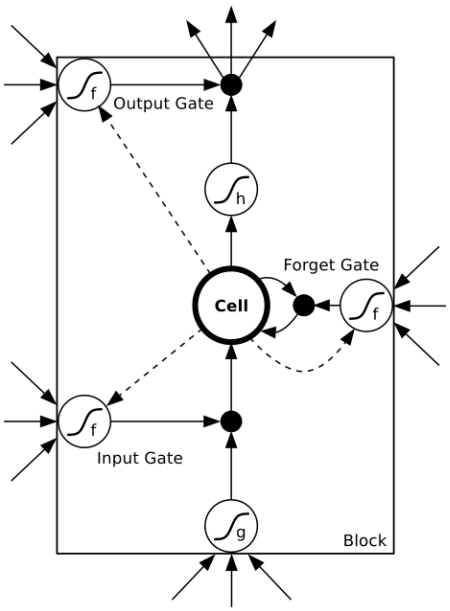

Figure 2: LSTM memory block [Graves, 2012]

LSTMs have been shown to be most successful for tasks where the use of long range contextual information is mandatory. For the case of online handwritten letter recognition and generation, LSTMs can make great use of the temporal relations of the input which leads to much higher performances [Graves et al., 2008]. Moreover, LSTMs have also been shown to be successful at speech recognition [Graves et al., 2013].

2.4. Optimization algorithms

2.4.1. Stochastic gradient descent

Training RNNs is a non-trivial task that needs considerable efforts. Stochastic Gradient Descent (SGD) is the most straightforward way to train neural networks and can successfully train RNNs as well. The parameters of a network are updated with the help of an error function $E(W_t)$ where W is a matrix representing all the weights in the network. The error function is calculated by first executing a forward pass in the network and then calculating the difference of the result to the expected result. To minimize this function, the weights are updated using the following formula for multiple timesteps (t):

$$\Delta W_t = -\eta \nabla_W E(W_t) \tag{3}$$

$$W_{t+1} = W_t + \Delta W_t \tag{4}$$

η is a small positive number also known as learning rate. ∇_W stands for the gradient with respect to the weights W. The weight update ΔW_t is added to the weights W to calculate the new weights for the next optimization step. This operation is repeated until the error function converges to a minimum. However, choosing the right learning rate is not easy and thus training with SGD is not always converging. Further, it is quite common that SGD gets stuck in a local minimum or saddle point and does not find the global optimum.

2.4.2. Momentum

The concept of a momentum term [Qian, 1999] was introduced to speed up the learning process. A term that depends on the weight update of the preceding timestep is simply added to formula 3:

$$\Delta W_t = -\eta \nabla_W E(W_t) + \gamma_1 \Delta W_{t-1} \tag{5}$$

γ_1 is the momentum parameter, usually set to a value around 0.9. The added summand helps the algorithm to speed up divergence in the correct direction and can help surpassing non-optimal local minimas. The weight update is still performed as in equation 4.

2.4.3. RMSprop

Numerous other optimizations of stochastic gradient descent have become popular. Root Mean Square Propagation (RMSprop) helps to create an adaptive learning rate based on previous gradients. The update rule is defined as follows (we substitute the gradient $\nabla_W E(W_t)$ used in previous equations with g for better readability):

$$v_t = \gamma_2 v_{t-i} + (1 - \gamma_2) g_t^2 \tag{6}$$

$$\Delta W_t = -\frac{\eta}{\sqrt{v_t + \epsilon}} g \tag{7}$$

v_t keeps track of the running average of the squared gradient for each weight (see equation 6). To save memory space, not all previous weights are stored, instead an update rule similar to that of the momentum technique is used. The new average only depends on the average from the previous timestep and the current gradient. A typical value of γ_2 is around 0.9.

Equation 7 shows how v_t is used to update the learning rate each timestep. The variable ϵ in the denominator is a very small number (e.g. 10^{-8}) to prevent division by zero.

By updating the learning rate every timestep for each parameter individually, more precise amendments to the weights can be made. This is almost always more effective because the parameters of the model are all not of equal importance and frequency. In addition, the algorithm becomes less dependent on the initial learning rate, which is always difficult to set optimally in SGD. In RMSprop, an initial value of 0.001 is often chosen for the learning rate and rarely needs to be amended.

2.4.4. Adam

Adaptive Moment Estimation (Adam) [Kingma and Ba, 2014] is another improvement that is currently recommended by most researchers [Ruder, 2016]. As shown in the last section (2.4.3), RMSprop adapts the learning rates based on the estimated average second moments of the gradients (also known as uncentered variance). Adam additionally keeps the average first moment of the gradients, i.e. the mean of the gradients (see equation 8).

Since the values of the exponential moving averages m_t and v_t are initialized with zero, they are also biased towards zero. By dividing them by a term based on the decay rate (see equations 10 and 11) these undesired biases can be eliminated.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{8}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{9}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{10}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{11}$$

$$\Delta W_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{12}$$

Summarizing, Adam can be seen as RMSprop with Momentum and bias-correction. Recommended values for the exponential decay rates β_1 and β_2 are 0.9 and 0.999. The parameter ϵ in equation 12 is a very small number to prevent any division by zero.

3. Methods

Different network architectures, training and evaluation methods for generating handwritten letters are tested and compared for their results. The letter trajectory dataset described in the following section (3.1) is used to train the models.

As training methods Backpropagation Through Time (BPTT) optimized by Stochastic Gradient Descent (SGD) with Momentum, RMSprop as well as Adam (see also section 2.4) were considered and tested. Preliminary investigations showed that the error was minimized best and fastest by using Adam. Thus, for all further investigations it was decided to use this optimization technique.

In terms of architectures standard RNNs are compared to LSTMs (see also section 2.3), each with different numbers of hidden layers and hidden layer size.

3.1. Dataset

The UCI character trajectories dataset [Lichman, 2013] consists of 2858 labeled samples of pen tip trajectories. A WACOM tablet was used to record the x and y coordinates of the pen as well as the tip force every 5 milliseconds. For simplicity, only characters with a single pen-down segment were considered in this dataset. As Graves also states, "[p]redicting delayed strokes (such as dots for 'i's or crosses for 't's that are added after the rest of the word has been written) is especially demanding" [Graves, 2013]. Further, all samples are produced by the same writer.

3.2. Experiments

3.2.1. Letter trajectory generation

In the first set of experiments, the parameters of the neural network implementations are learned with a training set consisting of four different trajectory samples from the same letter as target. The output of the networks consists of three linear units, representing Δx, Δy and tip force (intensity) of the trajectory. The tangens hyperbolicus is used as activation function for all hidden units.

In order to create suitable training data for the networks, input and associated output data is needed. For the task of creating one network for each letter, the output should not vary and thus it does not need to depend on the input. Consequently, a single Dirac impulse as input is sufficient to signal the network that it should produce output starting from that point in time. Thus, there is only one input unit by which the networks were stimulated. The networks learns to create some kind of average representation of the letter, incorporating features of all target trajectories.

It is tried to find the optimal hidden layer size. The size should be big enough that all details of a letter trajectory should be sufficiently covered but also not too big so that it would contain unnecessary hidden units. The produced letters should be easily recognizable. Further, the performance of LSTM networks is compared to the performance of standard RNNs.

As an additional evaluation metric, the averaged similarity between target and created output can be calculated. For this purpose, the similarity between two trajectories was defined as

$$similarity(t1, t2) = 1 - distance(t1, t2) \tag{13}$$

Dynamic Time Warping was used to calculate the distance (see secion 2.2). The calculated similarity values are a good objective evaluation of the actual resemblance of two trajectories, as it would be estimated by a human observer.

3.2.2. Letter trajectory generation with variable input stimuli

The setup described in the previous section enables the networks to generate a single stable letter trajectory. In order to create more flexible and thus more human-like trajectories, other architectures and methods are being tested.

By making the input stimuli of the neural networks variable, it is possible to generate different trajectory shapes of the same letter depending on the input. The output layer of the network is unchanged but the input layer is extended to a one-hot vector with the size of the number of different shapes that can be produced. This is normally the number of different trajectories in the training set used for training of the network. Each input neuron can be stimulated with a single dirac impulse when a letter of the respective shape should be produced. A sample input vector for a network able to produce four different shapes looks like this: $\{0, 0, 1, 0\}$. It stimulates the network to produce a letter of the third kind of shape.

In order to find trajectories for the training set that are dissimilar, the letter trajectories are grouped prior to training. K-means++ clustering [Arthur and Vassilvitskii, 2007] can be used to partition the data. Each trajectory gets assigned to a group which is built around a center. The resulting clusters each include trajectories of similar shape. To compare the different trajectories, the algorithm uses a distance measure. Considering its advantages explained in section 2.2, it was decided to use the Dynamic Time Warping algorithm to calculate the K-means++ clustering distance measure. The centers of the distinct computed clusters can be used as the target values in the training set, as they are very dissimilar.

3.2.3. Mixed letter trajectory generation

In the previous described experiment, networks are trained to generate perfect imitations of letter trajectories of the training set. Interesting results could be observed when varying the input vector values: Normally, the input to the network is in the form of a one-hot vector (e.g. $\{0, 1, 0, 0\}$), indicating which type of trajectories should be produced. By dividing the input to different indexes of the vector, the network can be stimulated to produce combined trajectories. For example, a mixture letter of the first and second kind could possibly be generated by using $\{0.5, 0.5, 0, 0\}$ as input stimulus.

A mixed letter trajectory should be of a completely new shape, that has not been seen in the training set. Well-recognizable results are to be expected if the networks are solving the task in a similar way to humans, because human writers are also able to

create new shapes of trajectories that are similar, but not equal to the ones they have seen before.

3.2.4. Letter trajectory generation with variance

In this experiment the creation of new letter shapes is taken even further and presents the possibility to generate theoretically infinitely many different trajectory shapes for the same letter.

Human writers do not produce identical letter trajectories even when asked to write the same letter. Small variances in shape are always present. However, it is not easy for a model to create kind of random variations that appear human-like.

[Plamondon and Djioua, 2006] used random variations (i.e. noise) on the parameters in one of their models of handwritten letter generation to produce strokes with some variability that should be comparable to the variations that human writers make.

For the neural network models observed here, it is possible to put random noise on the weights or on the hidden activations while the network is producing a letter. Random variables are sampled from a Gaussian probability distribution with mean 0 and a fixed standard deviation. At every timestep, this noise is added either to the hidden activations or to the weights of the network. As a result, the output trajectories should show slight variations in shape.

The next step is to objectively evaluate the plausibility of the generated new trajectories. The similarity to the original trajectory without noise can be calculated and should not be too big. However, this still does not tell whether the trajectory is also likely to be produced by a human.

A better evaluation method is to compare variance profiles, which will be described as follows. The full dataset of which the samples for the training set were drawn consists of 171 sample trajectories for the letter 'a'. Only some of these are used to train the neural networks. By calculating the variance of Δx, Δy and intensity values over the whole dataset, a variance profile can be created. This profile shows which parts of a character tend to have greater variance than others: In some areas of the trajectory human writers are varying their writing quite freely, in others not. For each letter, the variance profile looks differently.

By putting random noise on some parameters of the networks as described above, many distinct sample trajectories can be created. Afterwards it is possible to calculate the variance profile of these artificially created samples and compare it to the original variance profile for the letter.

The variance profile analysis provides an objective evaluation of how well the model captures the variations of a dataset of human-written letters. This is an important part of knowing whether the model is solving a task in a way that is similar to how humans solve it.

4. Implementation

4.1. Neural networks

The implementation was realized in Java, with the help of the JANNLab Neural Network Framework [Otte et al., 2013]. With this framework, RNNs as well as LSTMs with configurable number of hidden layers and hidden layer sizes can be created. For training, optimizers like SGD, RMSprop and Adam are implemented and can be used out of the box.

4.2. Preprocessing

For reading the training data, the MatFileRW library[1] was used. The data was then stored into Java arrays for further processing.

The `KMeansPlusPlusClusterer` class of the Apache Commons Mathematics Library[2] provides methods for clustering object instances based on a distance measure. The distance measure was implemented based on algorithm 1 in section 2.2.

4.3. Trajectory visualization

Trajectory visualizations were generated using the Java Swing library. By adding up the deltas, the x and y coordinates of the trajectories as well as the tip force for each timestep was calculated and drawn to a frame. The tip force data was used to adjust the intensity of the stroke.

Figure 3: Visualizations of different trajectories for the letter 'a' from the UCI character trajectories dataset [Lichman, 2013]

Figure 3 shows visualizations of four different examples of the letter 'a' which are the centers of a clustering process as described in section 4.2. The examples show that the trajectories can differ a lot even though they are representing the same letter and are written by the same person.

[1] https://github.com/diffplug/matfilerw
[2] http://commons.apache.org/proper/commons-math/

5. Results

This section presents the results of the experiments described in section 3.2.

5.1. Letter trajectory generation

The neural network implementations were trained with a training set consisting of four different trajectory samples from the letter 'a' as target until the loss converged (it was assumed that the loss has converged if it did not improve for 20000 epochs). The single input unit was activated at the start of the target sequence.

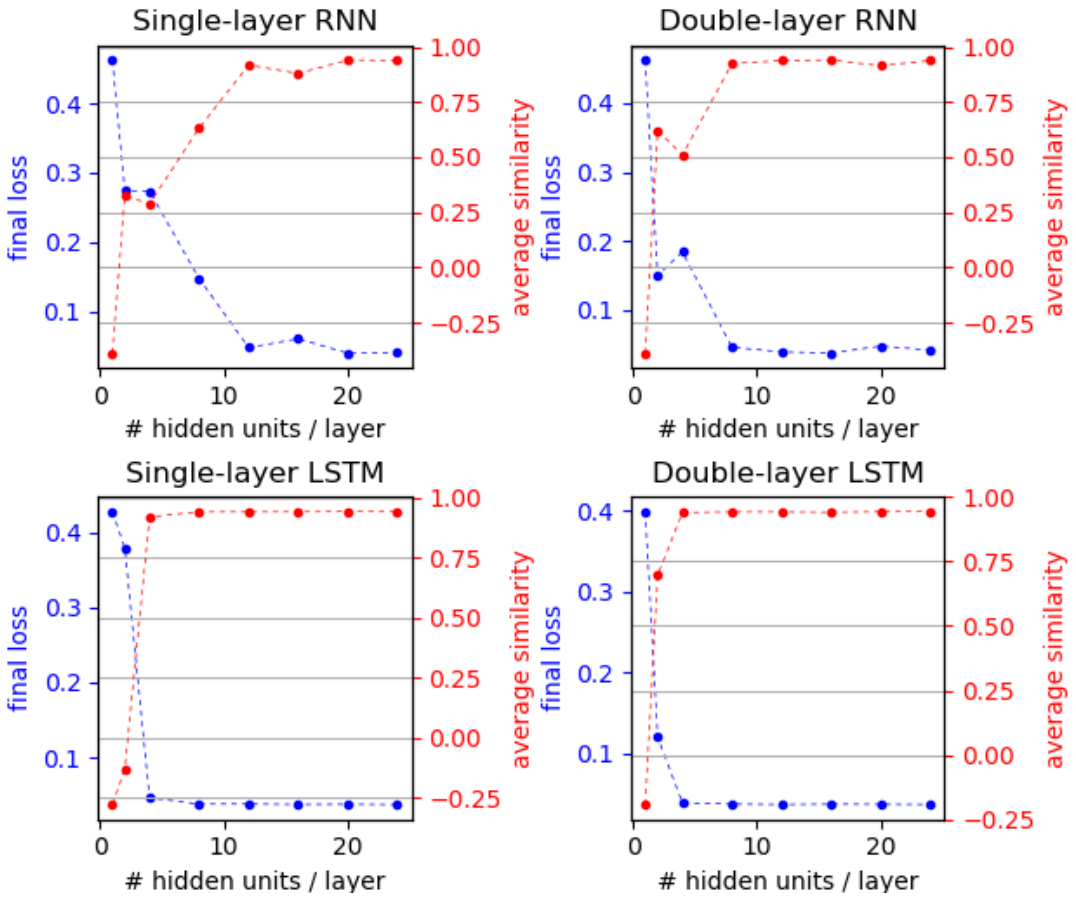

Figure 4: Final loss and average similarity of the single and double layer RNN and LSTM for varying hidden layer size when trained to produce an average trajectory of the letter 'a', using a single binary input impulse.

Single- and double-layered RNNs and LSTMs were used to produce average representations of the training trajectories. Final loss after and average similarity (of the output compared to each of the training samples) were calculated after training. The upper

plots in figure 4 show the results of these metrics for different hidden layer sizes of the RNN. For the single-layered RNN, good results are achieved with a hidden layer size that is greater or equal to 12, for the double-layered RNN already with a hidden layer size of 8 (per layer).

On the bottom of figure 4, the results for the different LSTM configurations are shown. For the LSTMs, the loss converges to numbers smaller than 0.04 for 8 and more hidden units for the single-layered architecture and for 4 and more hidden units per layer for double-layered LSTMs. At the same time, the average similarity goes up to values around 0.94. Note that the similarity and the loss do not sum up to 1, since the distance used for the similarity is calculated using Dynamic Time Warping.

The visualization of a generated letter is shown in figure 5. The letter in red was produced by an LSTM with 2 layers and 4 hidden units in each layer. The blue letters are the four different letters from the training set (also depicted in figure 3). Clearly visible, the network has formed some kind of average representation of the four trajectories.

Figure 5: Letter produced by an LSTM compared to the training set. The LSTM had 2 layers with 4 hidden units each.

An optimal network size for the task of this experiment would be a single-layered RNN with 12 hidden units or a double-layered LSTM with 4 hidden units (or a single-layered LSTM with 8 hidden units), as these configurations allow for the very good results with a minimum number of hidden units. Very well-recognizable letter trajectories are being produced, as seen in figure 5.

5.2. Letter trajectory generation with variable input stimuli

Similar network configurations were examined when using variable input stimuli. Input in the form of one-hot vectors enables the networks to produce different kinds of the same letter depending on input activation. The same four letter trajectories were used to train the networks.

First training results for the RNN showed that significantly bigger hidden layer sizes were necessary to achieve good results. The top two graphs in figure 6 depict the final loss and average similarity for hidden layer sizes of 10 to 160 units. The single-layer RNN shows improving performance up to a size of 100 hidden units (average similarity of 0.977), then it decreases again. The decrease might be caused by the early stopping mechanism (it was assumed that the loss has converged if it did not improve for more than 200000 epochs). However, it is still not expected that the performance would

improve very much compared to a network with 100 hidden units if the early stopping mechanism would have been delayed. The results of the double-layer RNN for similar sizes are better, but appear a bit more random. It turned out that the final loss could vary a lot, even for similar hidden layer sizes. Best performance is achieved with a network of 60 hidden units per layer, resulting in an average similarity of 0.998.

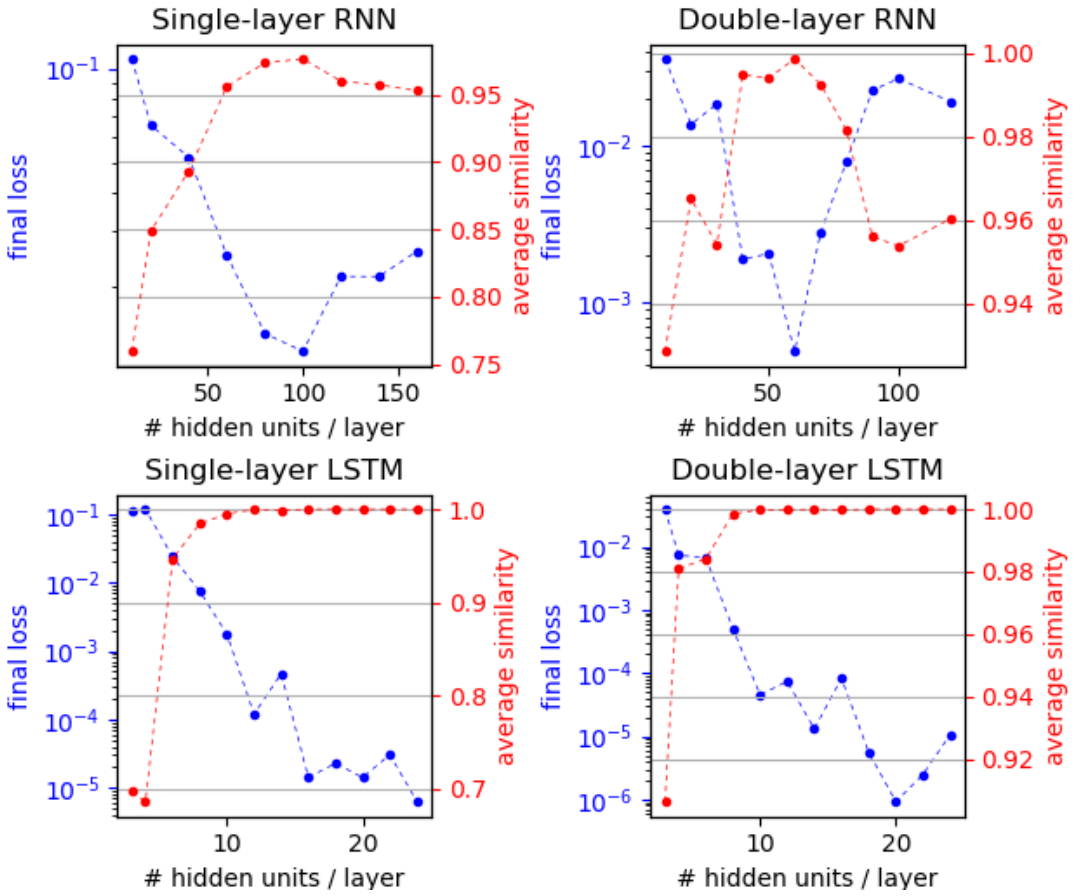

Figure 6: Final loss and average similarity of the single and double layer RNN and LSTM for varying hidden layer size when trained to produce four different trajectories of the letter 'a', depending on a one-hot encoded input vector.

The results of the evaluation metrics when using LSTMs are depicted in the two bottom graphs of figure 6. The final loss steadily decreases with increasing hidden layer size. It can get very close to 0, which means that the network can produce almost perfect imitations of all 4 letters from the training set. The single-layer LSTM needs about 12 hidden units to reach an average similarity close to 1 (the actual value being greater than 0.999), while the LSTM with two hidden layers needs 8 hidden units per layer, i.e. in total 16 hidden units.

Figure 7 shows the visualization of exemplary trajectory of an LSTM with 2 layers

with increasing hidden layer size, compared to the target output. The shape of the produced letter from the smallest LSTM is almost not recognizable. With increasing number of hidden units the similarity to the target increases until almost no difference between the trajectories can be seen anymore. This shows that the similarity measure we defined above (equation 13) represents the resemblance of the trajectories nicely.

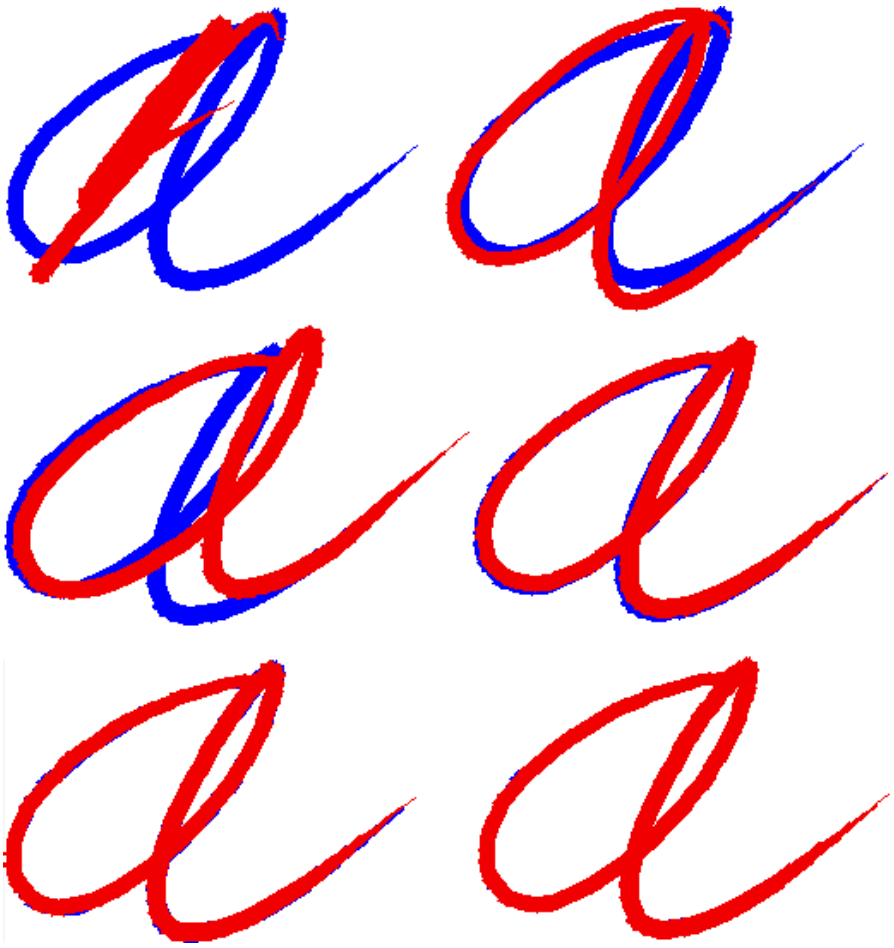

Figure 7: Produced trajectories (in red) of LSTMs with two hidden layers with increasing hidden layer size compared to their target (in blue). The number of hidden units per layer is 2, 4, 6, 8, 10 and 12; increasing from left to right and top to bottom. The networks were trained to be able to produce four different kinds of the letter 'a', here only one sample trajectory per network configuration is shown.

The performance of the LSTM networks clearly exceeds the performance of the RNNs when comparing their ability to create highly similar letter trajectories. The RNNs do not only need a lot more hidden units, it was also not possible to create highly accurate imitations with the RNNs. Considering these observations, for the following experiments

only LSTMs were used.

5.3. Mixed letter trajectory generation

This section shows the results of the attempt to create combined trajectories by varying the input to the network. Two different input units were stimulated at the same time (but each only with a value of 0.5 instead of 1) to create mixture letter trajectories.

At first, a single-layer LSTM with 12 hidden units was tested, as this showed very good performance in the previous section. The resulting mixed letter trajectories were often distorted and the similarity so their constituents very low. In a second attempt, an LSTM with 2 layers and 10 hidden units per layer was examined. Results did improve significantly. Figure 8 shows the shapes of the training set letters on top and some exemplary combined letters on bottom. The produced letter trajectories are actually mixtures of the two trajectories from the training set. For example, the "leg" (bottom right of the trajectory) of trajectory shape 1 is lower than its "body" while it is the other way around for trajectory shape 4. In the mixed letter of both, "leg" and "body" appear to be almost at the same level. Regarding this property, a kind of mean shape of the two constituent shapes has been produced. The second mixed letter shows the combination of 1 and 2. Clearly visible, the size of the "body" of the letter 'a' trajectory is between the sizes of the 2 basic letters.

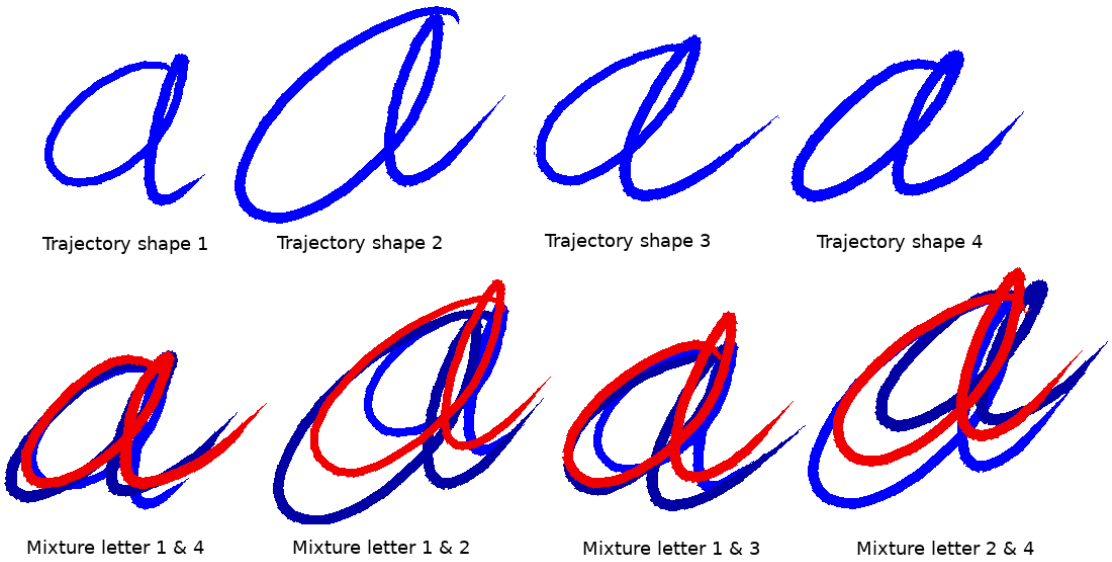

Figure 8: Top row: Letter trajectories of different shape that were used as training set trajectories. Bottom row: Mixture letters created by a trained LSTM with 2 layers and 10 hidden units per layer. The blue trajectories in the background show the different target shapes from the training set, the red trajectory shows the produced mixture letter.

Interestingly, the similarity of the mixed letter to each of the respective constituents

is not always the same. For example, for the mixture letter of 1 and 2, the similarity to letter 1 is 0.821 and to letter 2 0.977. By comparing the visualizations of the produced letter to its constituents it also appears that the letter is more similar to the second kind of shape.

When observing the mixed letter of 2 and 4, it appears to be more similar to both kinds of the shape. This can be also seen in the calculated similarity. For trajectory 2 it is 0.985 and for trajectory 4 it is 0.926, the difference is not as big as in the first example. In all tested cases the similarity to both of the constituent letters was above 0.8, showing that the produced mixed letters always relate strongly to the real letter trajectories.

The resulting mixed letter trajectories are always well recognizable by the human eye. However, their shapes are new and not the same as any of the trajectories of the training set.

5.4. Letter trajectory generation with variance

The next experiments introduce random variations to some parameters of the neural network models. An LSTM with the same network architecture and training methods as in the previous section was used. Noise with different standard deviations was put on either the hidden activations or the weights of the network.

Figure 9 shows the effect of increasing the standard deviation of the noise on the hidden activations of the network. With small standard deviation of 0.005, the trajectories are almost identical to the original, with some very small variations. By increasing it to 0.01, bigger changes become apparent but the letters could still be easily considered as human-written letters. The bottom row shows some of the trajectories being drastically distorted, especially the second type of 'a' is not recognizable anymore. In this row, the standard deviation has been raised to 0.05.

Almost the same results can be observed when putting the noise on the weights of the neural network. The only difference is that the LSTM was more sensitive to this modification, so we had to use smaller standard deviations to achieve good results. Figure 10 shows the results for noise standard deviations of 0.0005, 0.001 and 0.005.

The two experiments showed that by choosing an adequate amount of noise in the network, variable trajectories that are still well-recognizable can be created. For further evaluation, variance profiles have been created as described in section 3.2.4.

The top left graph in figure 11 shows the variance profile of the letter 'a', created by calculating the variances over all the 171 trajectories from the dataset. Several peaks are clearly visible, especially in the area between timestep 100 and 140.

A set of varying letter trajectories was created by putting noise on the hidden activations of the network. Already by creating variations of just one of the letters from the training set (i.e. by stimulating the network to produce always the same one shape from the training set), certain aspects of the variance profile can be captured, as shown in the top right graph of figure 11. For instance, both profiles show peaks of the intensity variance around timestep 115 and 135. One major peak of the variance of the x- and

Figure 9: Reproduction of the four training set letters with noise of increasing standard deviation on the *hidden activations* of the network. Top row: Letter trajectories generated with noise standard deviation of 0.005. Middle Row: Letter trajectories generated with noise standard deviation of 0.01. Bottom row: Letter trajectories generated with noise standard deviation of 0.05. All the letters were created by a trained LSTM with 2 layers and 10 hidden units per layer. The blue trajectories in the background show the different target shapes from the training set, the red trajectories show the produced letters with noise.

y-values is around timestep 110. However, especially at the beginning of the trajectory, the variance profiles do not match very well. By using 2 samples from the training set the profile already comes closer to the original (bottom left of figure 11) and when using all 4 training set samples, it captures most of the characteristics of the whole dataset. For example, also the small peak of intensity variance around timestep 10 is represented.

The variance profiles of the dataset of human-written letter trajectories and trajectories created by the model match quite well.

Figure 10: Reproduction of the four training set letters with noise of increasing standard deviation on the *weights* of the network. Top row: Letter trajectories generated with noise standard deviation of 0.0005. Middle Row: Letter trajectories generated with noise standard deviation of 0.001. Bottom row: Letter trajectories generated with noise standard deviation of 0.005. All the letters were created by a trained LSTM with 2 layers and 10 hidden units per layer. The blue trajectories in the background show the different target shapes from the training set, the red trajectories show the produced letters with noise.

The appendix A shows results when training LSTMs on other letters than 'a'. The examples include the results for letters 'b', 'c' and 'g'. All the created trajectories are well recognizable and the variance profiles contain all the main peaks at the same temporal locations as in the respective variance profile from the training set. It can be assumed that similar results can be observed for all different letters.

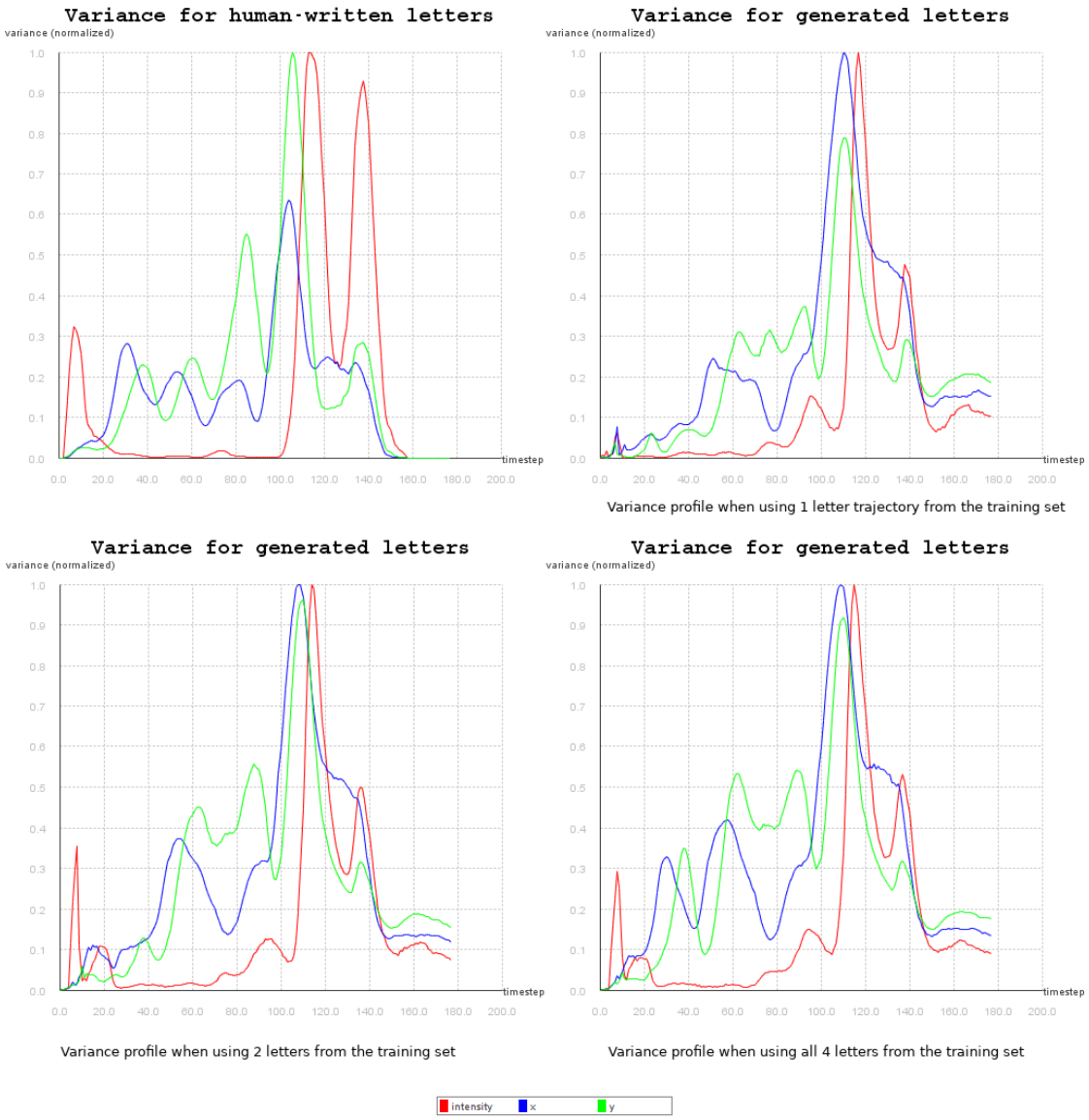

Figure 11: Top left: Variance profiles of full dataset. Top right: Variance profiles when generating letters using 1 letter trajectory from the training set. Bottom left: Variance profiles when generating letters using 2 letter trajectories from the training set. Bottom right: Variance profiles when generating letters using all 4 letter trajectories from the training set. For all evaluations, a trained LSTM with 2 hidden layers and 10 units per layer was used. Gaussian noise with mean 0 and standard deviation of 0.03 was put on the hidden activations to create 2000 samples from the network. Over these samples, the variances were calculated and normalized.

6. Discussion

The results that were observed in the previous section show promising performance of LSTMs in a variety of tasks. While RNNs still performed well when only one kind of output trajectory should be produced, they showed remarkably worse performance in subsequent tasks involving varying output depending on input stimulation. The most likely reason for this are the improved memorizing abilities of LSTMs, especially when it comes to long-term temporal relationships. The initial input stimuli might just be "forgotten" by the RNN over some timesteps. To prove this hypothesis, the input stimuli could be repetitively presented to the RNN which should show improvements in the variable input stimuli task performance.

In the first experiment, it was shown that the neural networks can generate a stable prototype of letters from a training set. When using variable input in the form of one-hot vectors, LSTMs can create almost perfect imitations of all letters from the training set. They show average similarity values of over 0.999 to the respective training set trajectories.

In a subsequent experiment, the original principle of one-hot input encodings is abolished and many input neurons are stimulated at the same time. In this way, combinations of letter trajectories of two trajectories from the training set can be created. These mixture letters are of completely new shape but are still well recognizable. Future work could involve even more free variations of the input pattern. For example a letter which relates strongly to one training set trajectory but only weakly to another one could be created. Moreover, mixtures of more than 2 letters could be generated, ultimately resulting in average letters of all training set letters as they were observed in the first experiment.

Finally, random noise has been added to the hidden activations and weights of the neural networks to produce even more new and unique trajectory shapes. The plausibility of these shapes has been proved by a variance profile analysis. All prominent peaks in the variance profile of the dataset could be found in an artificially created set that was initially fed by only 4 trajectories from the full dataset.

The resulting neural networks can show behavior that is similar to the behavior of humans in many ways. It is possible to request a specific kind of trajectory shape (just like one could also ask a human to produce a letter in a specific kind of shape), yet this shape is not a perfect imitation of the target but contains some random variations similar to the way that human handwriting is always slightly varying.

Comparing the models described in this work to the ones discussed in section 2.1.2, they are most similar to the models used by [Graves, 2013]. However, the network size is much smaller as each of them is only focused on the creation of trajectories of a single letter. Additionally, more focus was put on the objective evaluation of the plausibility of novel created trajectory shapes.

Some limitations for the evaluation arise from the characteristics of the dataset used. In first place, the data consisted only of handwritten letters by one person. A more representative set of handwritten letter trajectory data would contain data of many dif-

ferent writers. This would be crucial especially for the calculation of more characteristic variance profiles of letters.

Moreover, the dataset did not contain any letters with more than one stroke and all letters were drawn individually. The latter means that no context from preceding or following letters is incorporated. However, letter trajectories vary strongly depending on that context. This information should be incorporated when testing future models of handwritten letter generation. In real-world scenarios, handwritten letters are rarely observed separately.

All of the experiments that were carried out in this work used only 4 letter trajectories as training set and some of it only tested the letter 'a'. More general conclusions from the evaluation can only be drawn if similar results are observed when these parameters are being varied to create different testing environments.

The relevance of models of handwritten letter generation is manifold. In the first place, as mentioned in section 2.1.2, the models will help improving performance in handwritten letter recognition. The relation of these two tasks can also be observed in humans: Perceptive and productive processes are both depending on the same representation of information in the brain.

Another benefit could be a better understanding of the underlying cognitive and neuromuscular processes involved in the production of handwritten letters by humans. Handwritten strokes have also been analyzed to characterize neurodegenerative processes like Parkinson and Alzheimer diseases [Schröter et al., 2003] [Teulings and Stelmach, 1991]. With a good model of handwritten letter generation the observed symptoms could be simulated and underlying reasons could be identified and interpreted in detail.

7. Conclusion

The goal of this work was to build compact generative recurrent neural networks for handwritten letters. Several experiments have been completed on different RNN architectures. Best results could be achieved with LSTMs. Considering both performance and compactness, a double-layer architecture with 10 hidden units per layer is proposed. With this configuration, the networks can reliably imitate trajectories from the training set but also generate novel letter trajectory shapes. The evaluation demonstrates that the tasks are solved in a way very similar to humans, according to some criteria. This analogy shows that good models of letter trajectory generation have been developed, which are useful in many ways.

A different, more diverse dataset would increase the overall relevance of the generative neural networks. Additionally the letters should be observed in the context of whole words and sentences, which could give important supplementary context information.

Still, the set of generative neural networks implemented as part of this work can already be used to conduct further research on handwritten letter generation and recognition.

References

[Arthur and Vassilvitskii, 2007] Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

[Bahlmann and Burkhardt, 2004] Bahlmann, C. and Burkhardt, H. (2004). The writer independent online handwriting recognition system frog on hand and cluster generative statistical dynamic time warping. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):299–310.

[Gangadhar et al., 2007] Gangadhar, G., Joseph, D., and Chakravarthy, V. S. (2007). An oscillatory neuromotor model of handwriting generation. *International journal of document analysis and recognition (ijdar)*, 10(2):69–84.

[Graves, 2012] Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer.

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

[Graves et al., 2008] Graves, A., Liwicki, M., Bunke, H., Schmidhuber, J., and Fernández, S. (2008). Unconstrained on-line handwriting recognition with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 577–584.

[Graves et al., 2009] Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868.

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

[Hochreiter et al., 2001] Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer, S. C. and Kolen, J. F., editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Hollerbach, 1981] Hollerbach, J. M. (1981). An oscillation theory of handwriting. *Biological cybernetics*, 39(2):139–156.

[Hu et al., 2000] Hu, J., Lim, S. G., and Brown, M. K. (2000). Writer independent online handwriting recognition using an hmm approach. *Pattern Recognition*, 33(1):133–147.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Lichman, 2013] Lichman, M. (2013). UCI machine learning repository.

[Ltaief et al., 2016] Ltaief, M., Bezine, H., and Alimi, A. M. (2016). A spiking motor-model for online handwriting movements generation. In *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, pages 477–482. IEEE.

[Niels et al., 2005] Niels, R., Vuurpijl, L., et al. (2005). Using dynamic time warping for intuitive handwriting recognition. In *Proc. IGS*, pages 217–221. Citeseer.

[Otte et al., 2013] Otte, S., Krechel, D., and Liwicki, M. (2013). Jannlab neural network framework for java. In *Poster Proceedings Conference MLDM 2013*, pages 39–46, New York, USA. ibai-publishing.

[Plamondon and Djioua, 2006] Plamondon, R. and Djioua, M. (2006). A multi-level representation paradigm for handwriting stroke generation. *Human movement science*, 25(4-5):586–607.

[Plamondon and Maarse, 1989] Plamondon, R. and Maarse, F. J. (1989). An evaluation of motor models of handwriting. *IEEE Transactions on systems, man, and cybernetics*, 19(5):1060–1072.

[Plamondon and Srihari, 2000] Plamondon, R. and Srihari, S. N. (2000). Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):63–84.

[Qian, 1999] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[Schröter et al., 2003] Schröter, A., Mergl, R., Bürger, K., Hampel, H., Möller, H.-J., and Hegerl, U. (2003). Kinematic analysis of handwriting movements in patients with alzheimer's disease, mild cognitive impairment, depression and healthy subjects. *Dementia and Geriatric Cognitive Disorders*, 15(3):132–142.

[Teulings and Stelmach, 1991] Teulings, H.-L. and Stelmach, G. E. (1991). Control of stroke size, peak acceleration, and stroke duration in parkinsonian handwriting. *Human Movement Science*, 10(2-3):315–334.

[Varga et al., 2005] Varga, T., Kilchhofer, D., and Bunke, H. (2005). Template-based synthetic handwriting generation for the training of recognition systems. In *Proceedings of the 12th Conference of the International Graphonomics Society*, pages 206–211.

[Werbos, 1990] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[Williams and Zipser, 1995] Williams, R. J. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 1:433–486.

Appendices

A. Results for other letters

This appendix shows results for different letters of the experiment described in section 3.2.4. All the letters were created using a trained LSTM with 2 layers and 10 hidden units per layer. The blue trajectories in the background show the different target shapes from the training set, the red trajectory shows the produced letter with noise. The early stopping interval during training was increased to 50000 epochs.

For the variance profiles, Gaussian noise was put on the hidden activations to create 2000 samples from the network. Over these samples, the variances were calculated and normalized.

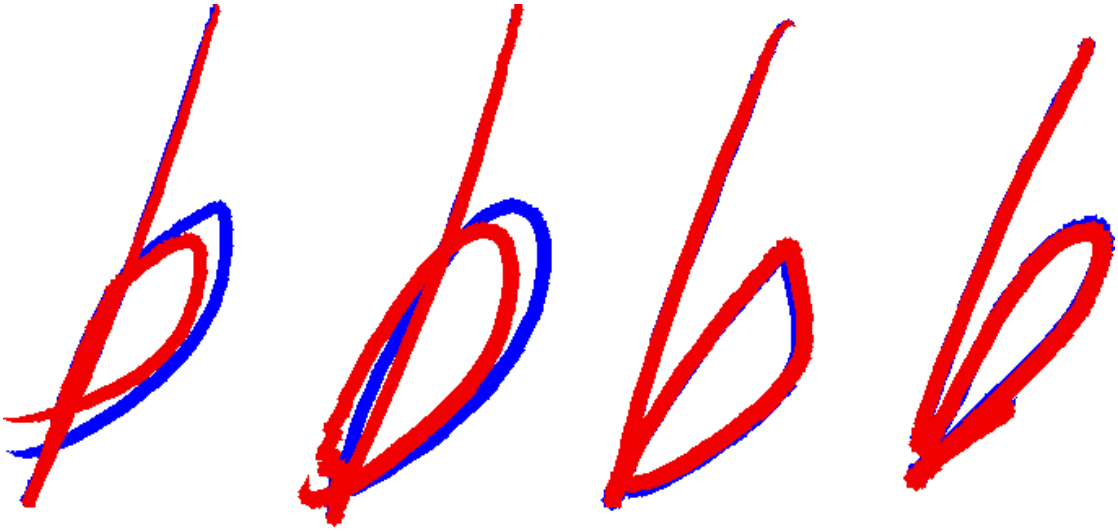

Figure 12: Reproduction of the four training set letters 'b' with noise (standard deviation of 0.001) on the hidden activations of the network.

Figure 13: Reproduction of the four training set letters 'c' with noise (standard deviation of 0.015) on the hidden activations of the network.

Figure 14: Reproduction of the four training set letters 'g' with noise (standard deviation of 0.005) on the hidden activations of the network.

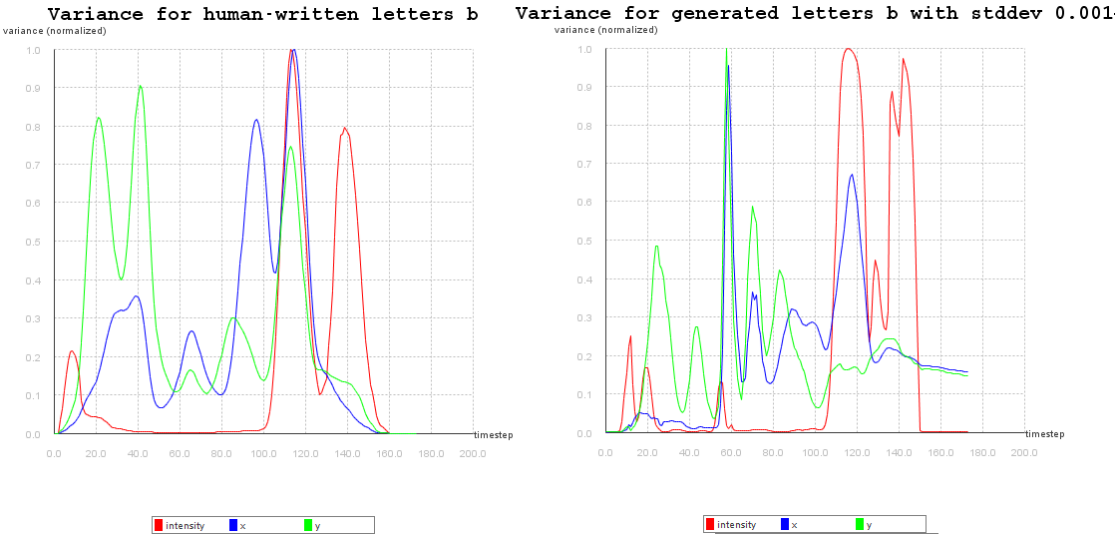

Figure 15: Left: Variance profiles of full dataset for letter 'b'. Right: Variance profiles when generating letters using 4 letter trajectories from the training set. The Gaussian noise had a standard deviation of 0.001.

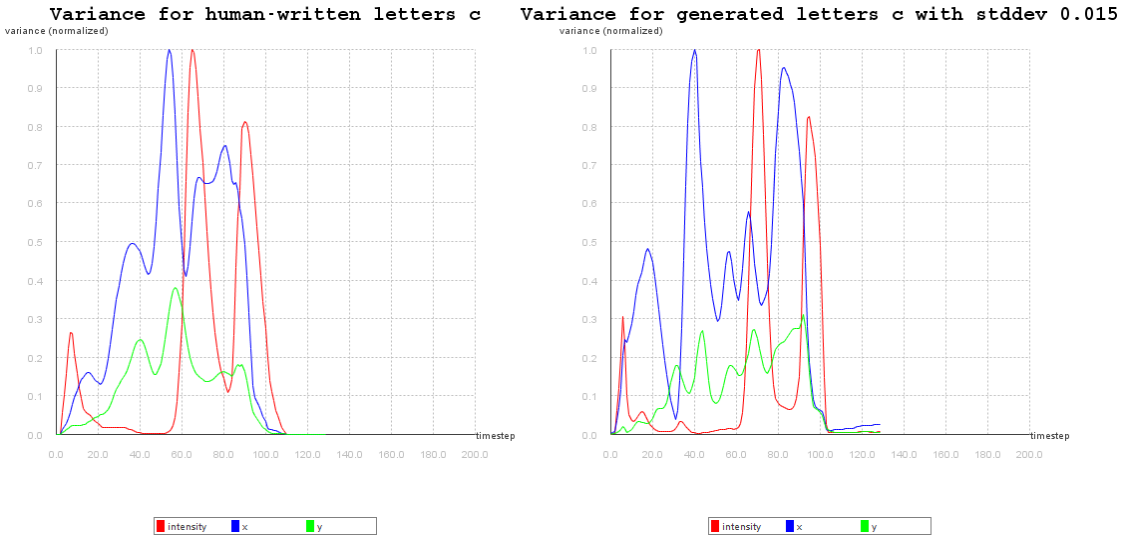

Figure 16: Left: Variance profiles of full dataset for letter 'c'. Right: Variance profiles when generating letters using 4 letter trajectories from the training set. The Gaussian noise had a standard deviation of 0.015.

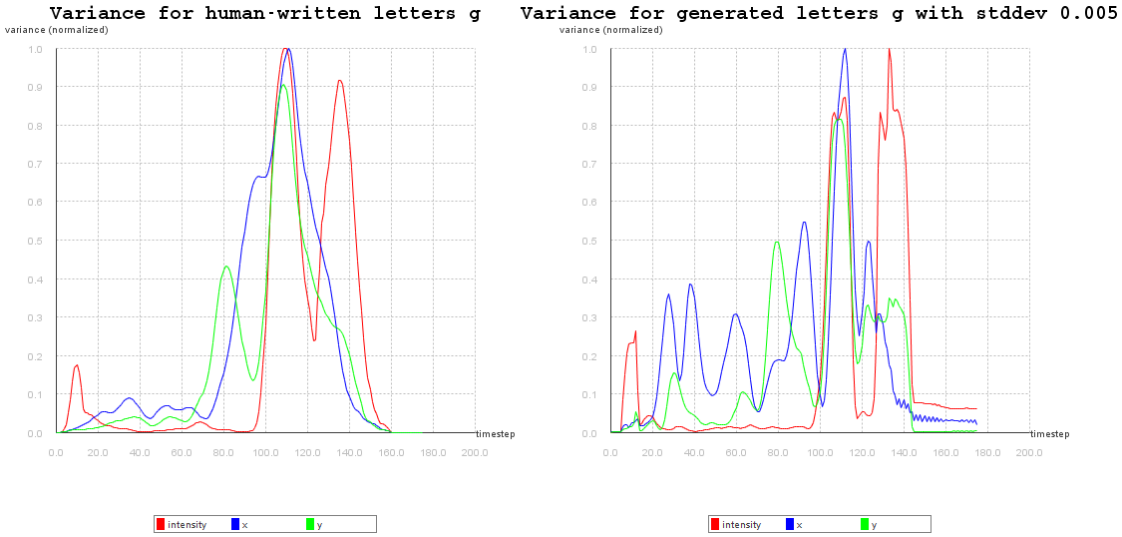

Figure 17: Left: Variance profiles of full dataset for letter 'g'. Right: Variance profiles when generating letters using 4 letter trajectories from the training set. The Gaussian noise had a standard deviation of 0.005.