# Deep learning methods for style extraction and transfer

Omar Mohammed

Date unknown, somewhere far far in another galaxy, long time ago

# Contents

# List of Figures

# List of Tables

# Part I

# Introduction and Problem Description

# Chapter 1

# Introduction

**Contents**



## 1.1 What is a style?

Talk about style in images, speech, handwriting, driving,...etc. Also, distinguish between static and temporal styles.

- What is a task? what is a style? how both combine to give us an experience?
- Possible scenarios: images, speech, handwriting, autonomous driving
- Style perception depends on 'how you look at it' (the light - angle of view -, body - task - and shadow - the resulting style - example) (cooking rice: no spices or with curcuma. perception: color? taste?) (handwriting:

online styles? offline style?). Then end up with the conclusion that styles are ill-defined.

## 1.2 Why studying styles?

- Shortcoming of applying machine learning methods – average over previous scenarios, doesn't consider styles in advance –.
- HRI example: the need for personalization to enhance the interaction experience.
- Speech example: different utterances change the perceived message

## 1.3 Why Handwriting?

- Availability of dataset
- Several style aspects are accessible to investigate

## 1.4 An overview of the PhD

The main contribution of the PhD is a thinking framework to study and observe styles, in the temporal case, using neural networks, while the task is defined beforehand. Through implementation, benchmarks and evaluation, and analysis of the networks performance (and sometimes its behaviour), we seek to provide support to ground this framework, and our hypotheses about it.

The most general elements/components has been used in our study, in order not to couple the effect of more advanced/complicated items with our conclusions. This leaves a big room for further improvements and exploration as well.

Using machine learning as tool enables us to process large amount of data.

## 1.5 Disclaimer

Task

# Chapter 2

# Datasets

## Contents



## 2.1 Introduction

In order to test the different hypotheses and ideas presented in the introduction chapter, we need to select suitable domain to carry out the experiments, which determined by the data. The choice of the data is a selection between different trade-offs: relevance (i.e., it contains the relevant information to perform the study) VS readiness of the data (i.e., is it already available? and how expensive it is to acquire the data?), simplicity VS realism (simple data has less noise

and less irrelevant patterns, but the ability to handle realistic data provides stronger support for the hypothesis), and the amount of data available.

It is traditional in the machine learning community to use two or more datasets to address the questions. This way, it is possible to detect issues like having a very specific method that work only in a limited context[1], and to avoid the effect of unknown contributing variables.

We settled on the domain of handwriting and drawing **Remeber to explain the reasoning for this choice in the introduction**. In this chapter we present two datasets: Online english letters handwriting dataset, *IRONOFF*, and online sketch drawing dataset, *Quick Draw!*. We present general information and exploratory statistics about both datasets, and discuss the categories/tasks in each dataset, and argue why both datasets are suitable for this study.

Points addressed in this chapter

- Present *IRONOFF* handwriting dataset.
- Present *QuickDraw!* sketch drawing dataset.
- Motivate the suitability of these datasets for this study.

## 2.2 Online Handwriting – *IRONOFF*

*IRONOFF* (?) is a cursive handwriting dataset provides us with isolated letters, thus allowing us to focus on the problem of styles with a reasonable complexity, and gives us the advantage that the content of the task is well known beforehand (i.e, the identity of the letter). Other cursive handwriting datasets do exist, like *IAM Handwriting Database* (?). However, they use whole sentences/paragraphs, instead of individual letter, thus making the problem more complicated.

Basic information about *IRONOFF* dataset as a whole:

- Around 700 writers in total. We use the 412 writers who have written

isolated letters.

- 10,685 isolated lower case letters, 10,679 isolated upper case letters, 4,086 isolated digits and 410 euro signs.
- The gender, handiness, age and nationality of the writers.
- For each writer/task (letter or digit) example, we have that example's image - with size around 167x214 pixels, and a resolution of 300 dpi -, pen movement timed sequence comprising continuous X, Y and pen pressure, and also discrete pen state. This data is sampled at 100 points per seconds on a Wacom UltraPad A4. Figure 2.1 shows an example for format of the the provided data.

We explored the information available about the writers in the dataset (see figure 2.2), we can see that almost all the participants are of French nationality, and majority of them are less than 30 years old. The data is almost balanced between males and females, but largely unbalanced between left and right handed people. This indicates that this dataset is not a representative for all the people. This, however, does not limit our study (a fair representation for all the people is not a point of concern).

We then explored the handwriting examples from multiple points of view. In figure 2.3, we can see the frequency of strokes for different tasks. We can see that letters like *C* and *L* needs one stroke only, while *I* and *E* requires the most number of strokes in order to draw properly. By combining this with observation about the drawing time for different tasks (see figure 2.4) and the pausing time (see figure 2.5), we can have a good indication about the complexity of each task relative to the other tasks. The more strokes the task has, the more drawing and pausing time is needed, and the more complex the task is.

One challenging issue with this dataset however is that we have only one example for each writer-letter combination. This makes the task more difficult, because it is hard to extract a writer style using very few items (the 26 letters/writer in this case).

```
.DATE           08 07 1998

.COMMENT        Declarations
.X_DIM             3600
.Y_DIM             3600
.X_POINTS_PER_INCH 300
.Y_POINTS_PER_INCH 300
.POINTS_PER_SECOND 100
.COORD          X Y P T

.COMMENT        Data
.WRITER_ID      sk18091962
.COUNTRY        Germany
.HAND           R
.AGE            35
.SEX            M
.SEGMENT CHARACTER ? ? "a"
.PEN_DOWN
88 104 2 0
89 103 12 10
90 101 24 21
90 100 30 31
90 99 39 40
91 97 46 50
92 95 53 62
92 89 57 74
91 86 61 85
90 83 64 97
88 80 67 109
85 78 69 121
80 80 71 132
74 88 70 144
69 101 69 156
64 114 68 167
62 120 68 179
61 127 68 191
62 131 68 203
68 132 67 214
72 130 67 224
81 123 68 234
89 116 68 244
93 113 70 256
94 111 73 272
94 112 75 285
93 119 76 297
92 123 76 308
92 130 76 320
93 136 74 331
96 138 66 340
99 142 15 350
.PEN_UP
```

Figure 2.1: An example from *IRONOFF* (letter 'a'). To the left, we have the image of the letter (i.e., offline-handwriting). To the right is an example of the format for the online-handwriting. We have the writer's ID, origin, handiness, age and gender. The sequence of pen movement to draw the letter are then given: pen state (PEN_DOWN, PEN_UP), X, Y coordinates, pen pressure, and time.

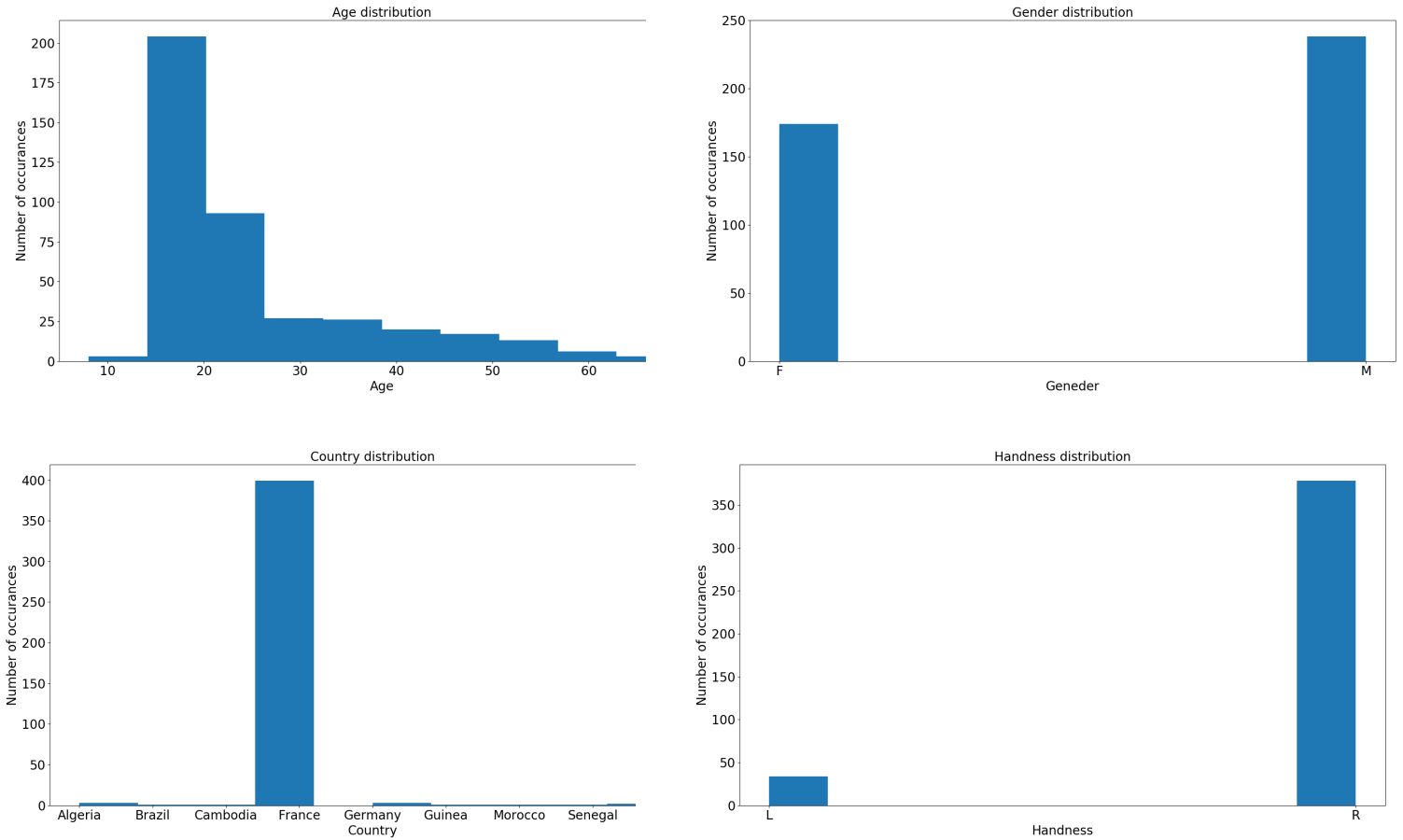



Figure 2.2: Summary statistics about the writers in *IRONOFF*: the age, gender, country and handiness.

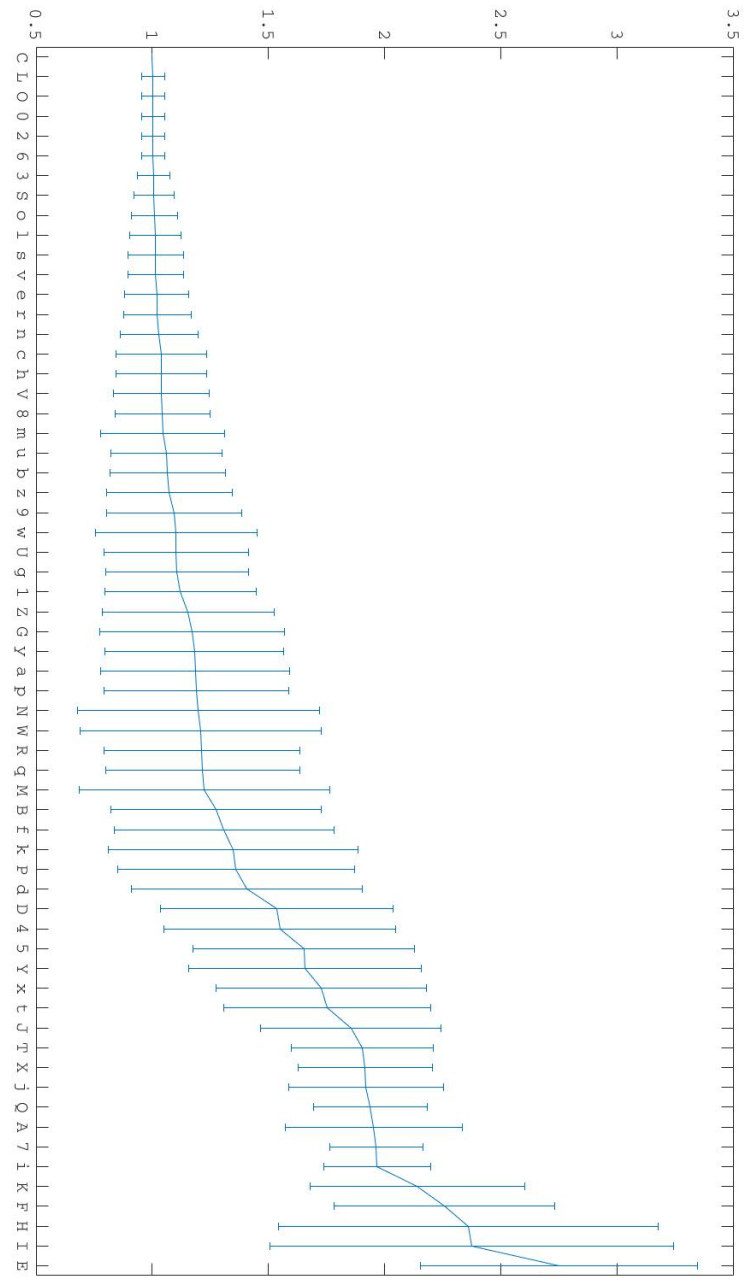



Figure 2.3: Summary statistics strokes for all categories in *IRONOFF* dataset (uppercase and lowercase letters, and digits), starting from the simplest to the more complex.

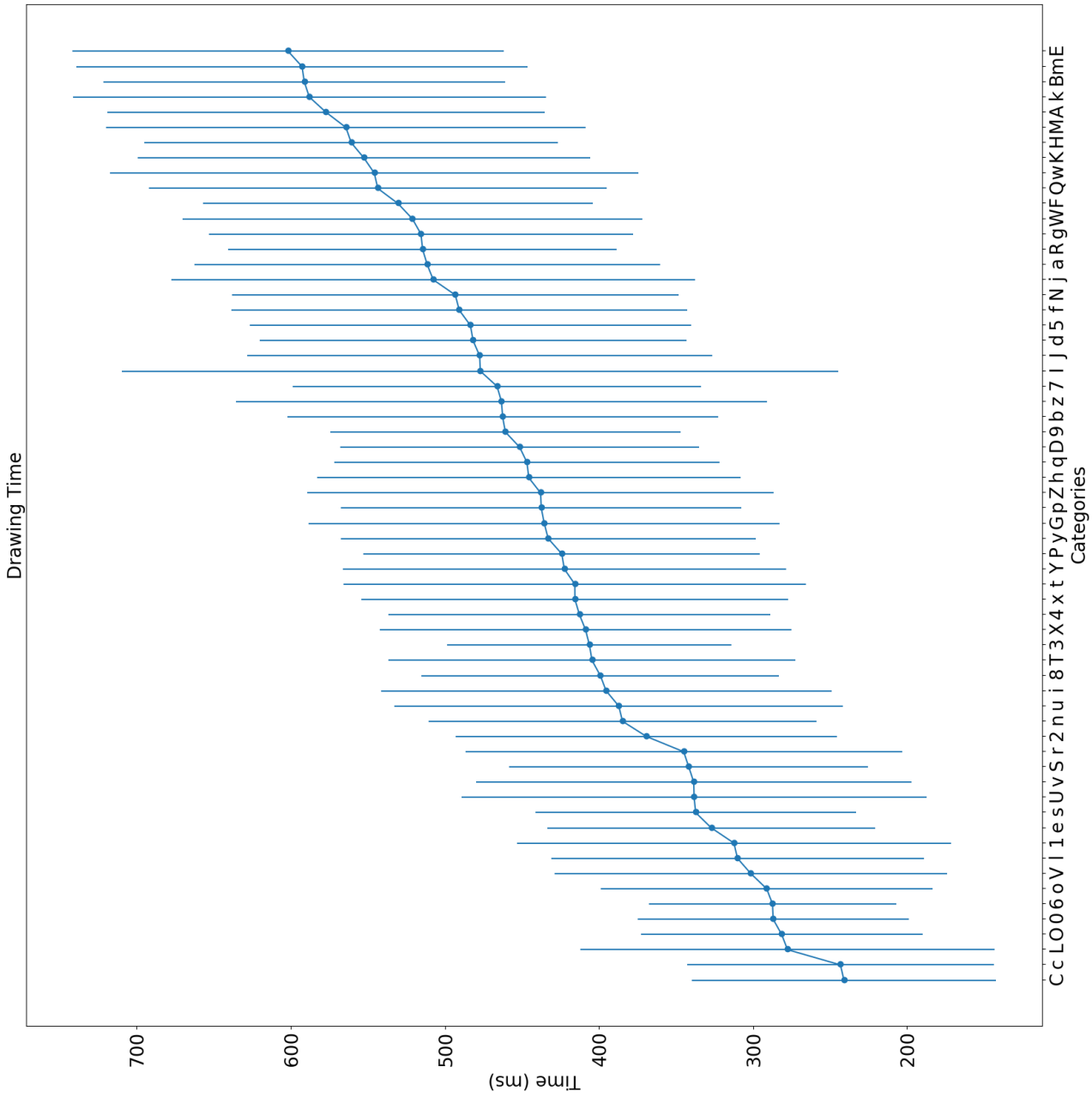



Figure 2.4: Drawing time for all categories in *IRONOFF* dataset, arranged from the smallest to the largest.

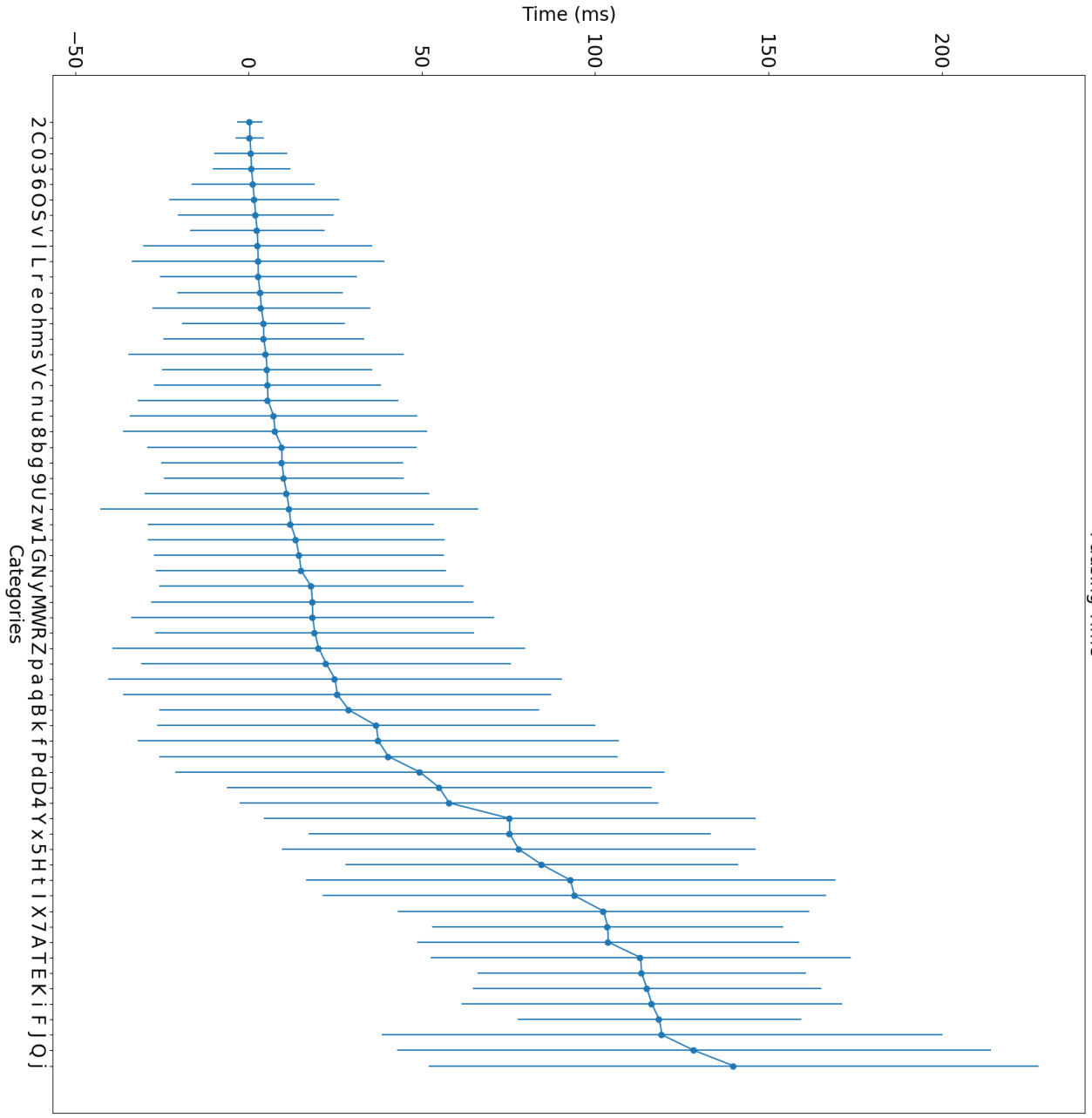



Figure 2.5: Pausing time for all categories in *IRONOFF* dataset, arranged from the smallest to the largest.

## 2.3 Sketch Drawing – *QuickDraw!*

Around 50 million drawings have been collected by players of the game *Quick, Draw!* (?), where players are asked to draw one of 345 categories, and a neural network tries to classify the drawings into the right categories. With more data collected and labelled, the network gets better (it learns from the flagged errors). The collected dataset is available online for free (?). Example of the shapes collected are in figure 2.6.

Each sample in the dataset contains the following data:

- key_id: a unique identifier for this sample.
- word: the category the player was asked to draw.
- recognized: if the neural network in the game did recognize the drawing as part of this category.
- timestamp: to mark the creation time of the drawing.
- countrycode: the location of the player when the drawing was made.
- drawing: an array containing the $X$, $Y$ trajectories, and the time $T$ for each point in the trajectory. Points belonging to each stroke are grouped together.

In order to focus on the styles aspect, we decided to focus on 5 categories: circle, triangle, square, hexagon and octagon. Our reasoning is that the more complex the task gets (cats for example), it is harder to have a subjective opinion about the styles, and hard to give insights about the results. This is not a limitation on the methods we are proposing though.

We sampled 20K samples from each task to perform exploratory analysis on them. In terms of strokes (see figure 2.9), we can see that there is a large variance surrounding the mean of each category. This trend continues when we look at the drawing time (see figure 2.11) and the pausing time (see figure 2.10). In the sample we analyzed, there are players from around 160 countries, figure 2.12. This is one indication to the increasing complexity *QuickDraw!* dataset presents in compare of *IRONOFF* dataset. **Put the country distribution in the dataset.**

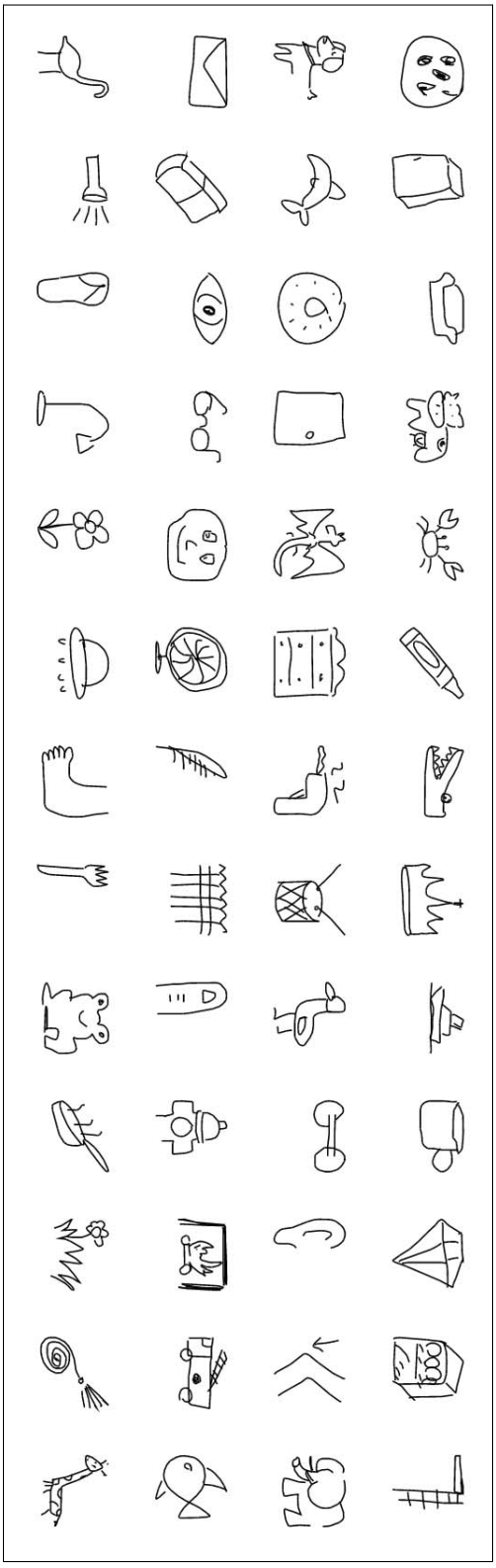


Figure 2.6: Examples from different categories in *QuickDraw!* dataset. Source of the images are (?)

Not all examples are recognizable during the game though. In many cases, the players do not draw the required shape, or draw something quite complicated. The results of the recognition can be seen in figure 2.7, with the relation between number of strokes and length of the drawing. We can conclude that, for the selected categories, the more complex the drawing is (more length or more strokes), the less likely it is to become a recognizable drawing.

This dataset is considerably more challenging than *IRONOFF*, for several reasons:

- Even though the players are asked to perform a particular task (draw a circle for example), in several cases, there is no clear resemblance between the drawing and task (e.g., when drawing an octagon, a lot of the recognized drawings do not really resemble an octagon).
- The players used a mouse in order to perform the drawing[2]. This sometimes lead to weird behaviours in terms of speed of movement (too slow, too fast), and the number of strokes (players sometimes tend to simplify complex shape, by drawing the whole shape in one stroke, and sometimes they spend too much time to draw it well, with too many strokes).
  This is unlike handwriting, where the writers usually tend to follow some rules ((?)), which is not mostly the case in this dataset.
- Thus, some extra parts of pre-processing will need to be added in order to reduce the reduce these effects of the mouse, and make the data more closer to handwriting behavior.
- As mentioned earlier, the variance for each of the selected categories in this dataset is considerable higher than *IRONOFF*.

Since these drawings are done using the mouse, an interesting aspect for the recognizable images is the simplicity of strokes used – easier for player – (see figure 2.9). If a pen is used in the drawings, this particular behaviour would not observed. For example, in case of hexagon and octagon, one can expect a higher density on the 6 and 8 strokes respectively, and much less on 1 stroke. Our observation is that it is easier with the mouse to draw the whole shape with one (or few) strokes only. This has two consequences:

- It is difficult to generate the strokes: One/few strokes means that a direct stroke representation is quite sparse (if the length of the shape sequence is 200 time steps, then among all the zeros (199 zeros, representing the current stroke), there is a single 1 value (representing the end of the stroke). This problem has also been noted in the work done in (?), and it is a challenging task to tackle.
- Unlike in *IRONOFF* dataset, where the strokes is a contributing feature in identifying the letter and the rules of drawing it, the strokes are not expected to play the same rule in *QuickDraw* – unless further processing is done –.

For each class, we randomly – without replacement – sampled only from the recognized drawing, traces with less than 200 time step long. 2K samples (total is 10K samples). 1K is used for test, 900 for validation, and the rest is the training data.

## 2.4 Data representation

**This section should be more general, discussing both pre-processing and represenation in the same time. Otherwise, we can discuss only the representation, and move the pre-processing steps to the appendix (which makes sense). Also, there is the pre-processing part Gerard performed on QuickDraw, which should be discussed.**

The choices of data representation is a key factor in the success or failure of the machine learning based approaches. This choice, however, is also entangled with the task to be done (in this case, the study of styles).

A good representation tries to:

- Maximize the density of $data/patterns$ ratio: machine learning algorithms are statistical algorithms. It performs better when we have more examples for the patterns we want to learn (e.g., in case of cat/dog image recogni-

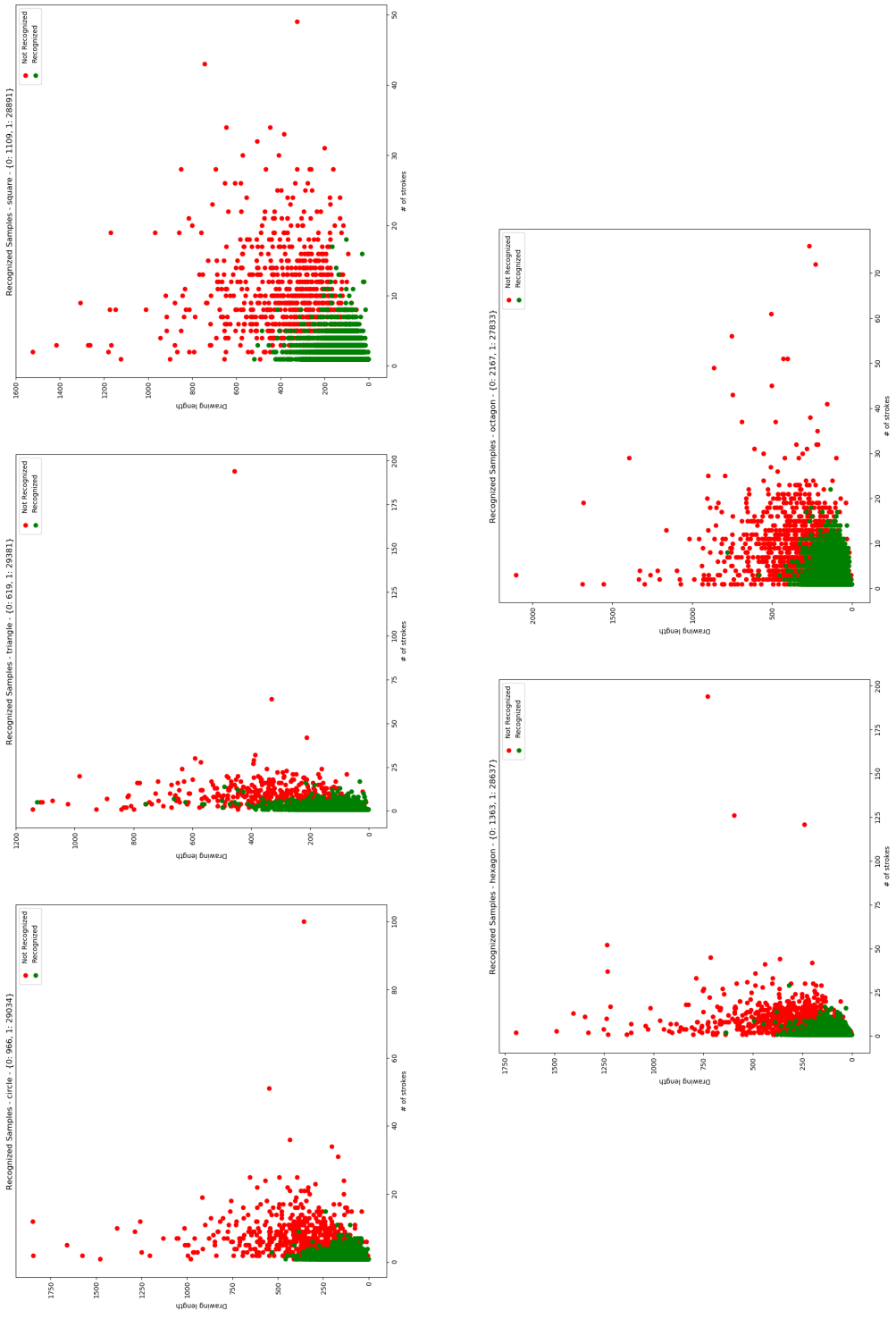



Figure 2.7: The distribution of the strokes in *QuickDraw!* for the recognized shapes, for each category.

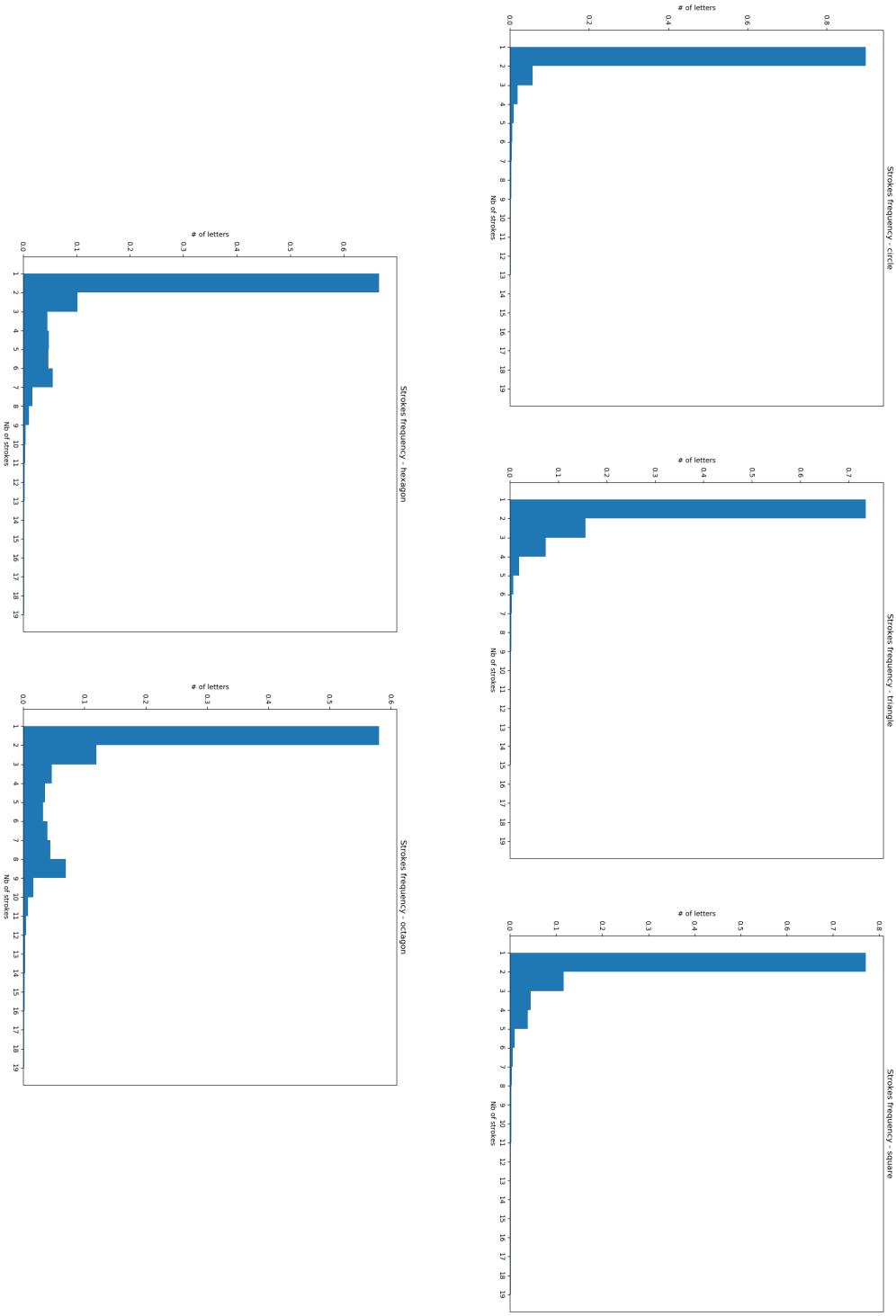



Figure 2.8: The recognized VS non-recognized drawings in *QuickDraw!* in each of the selected categories.

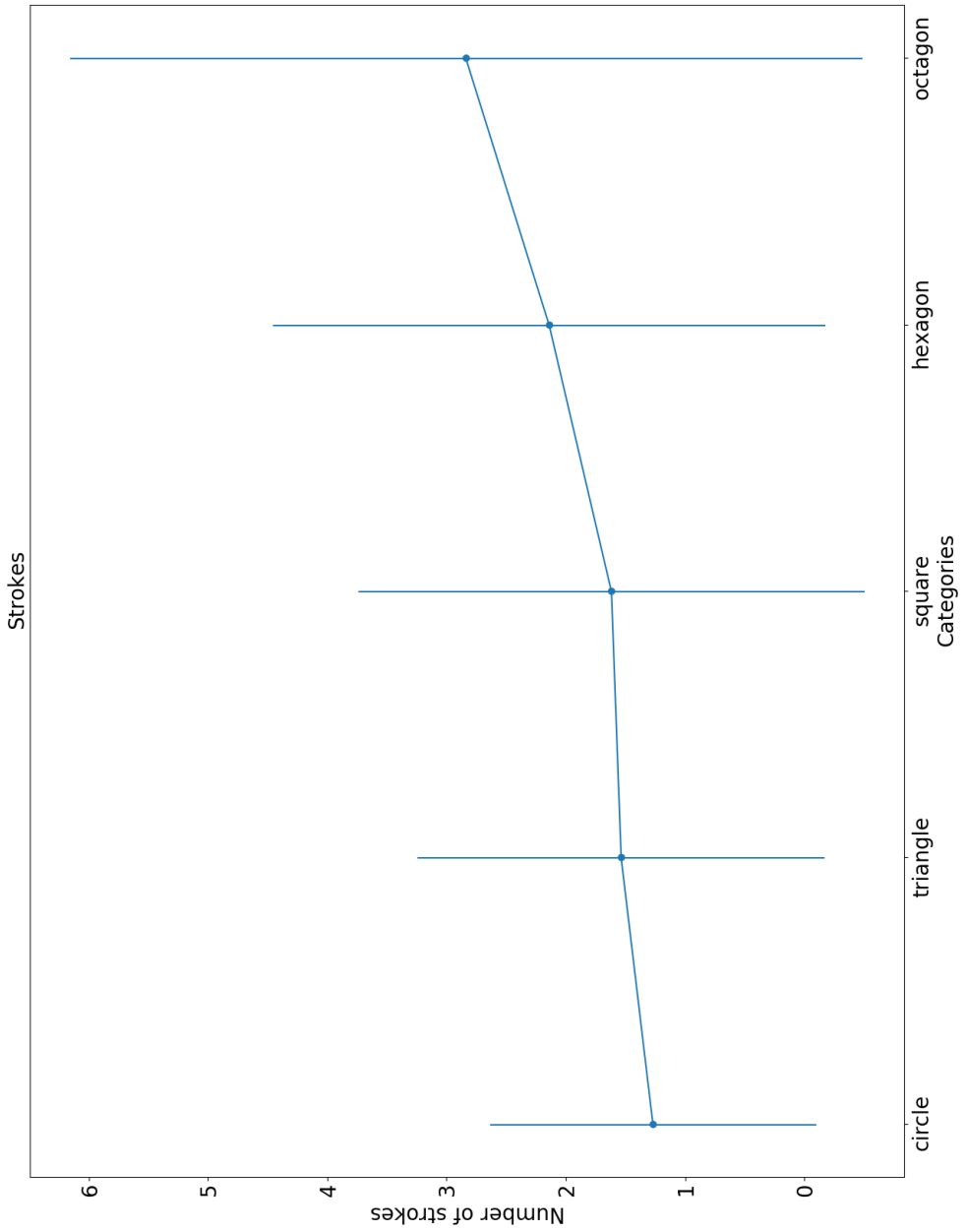



Figure 2.9: QuickDraw! strokes statistics for each of the selected categories.

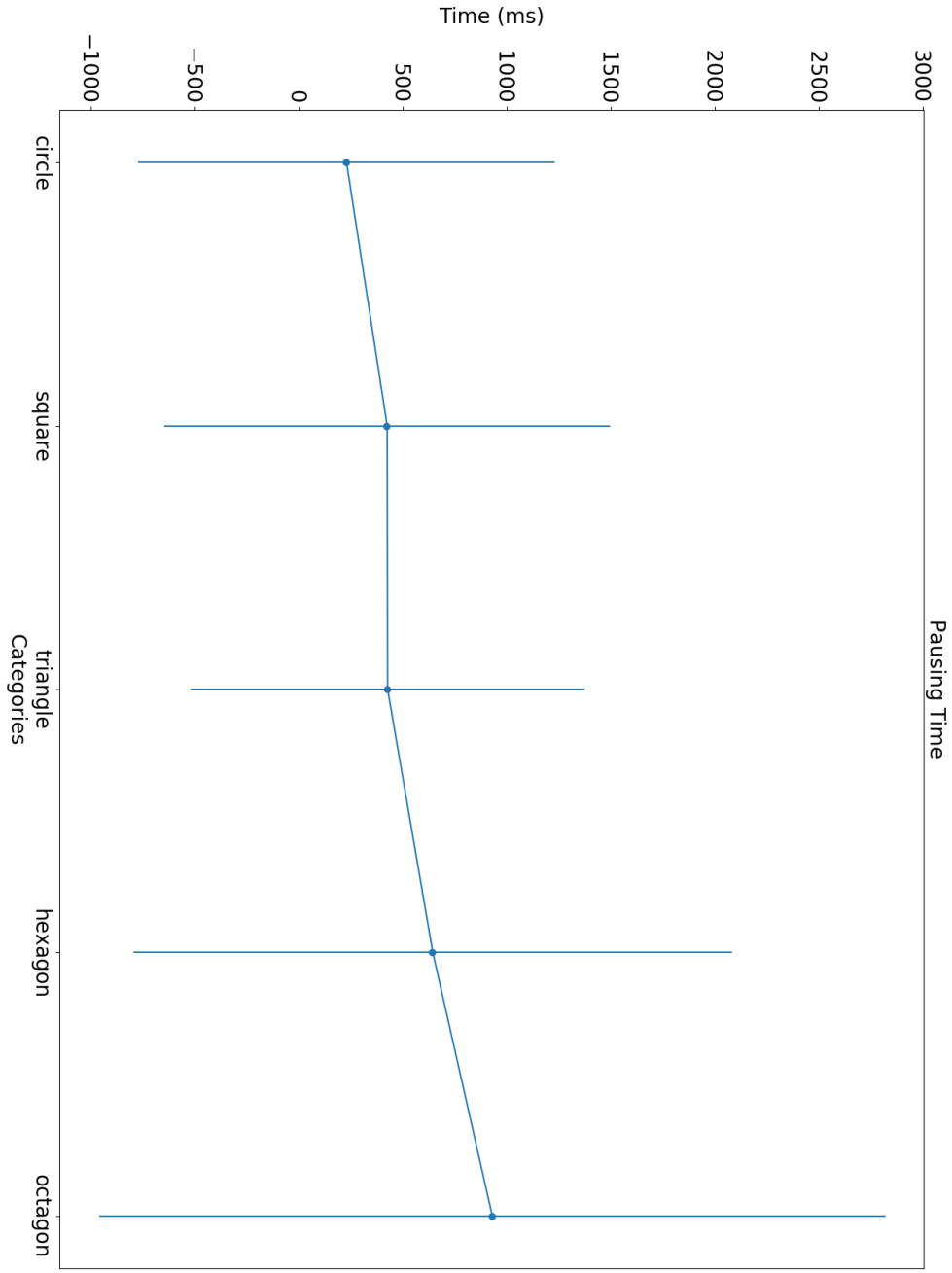



Figure 2.10: QuickDraw! pausing time statistics for each of the selected categories.

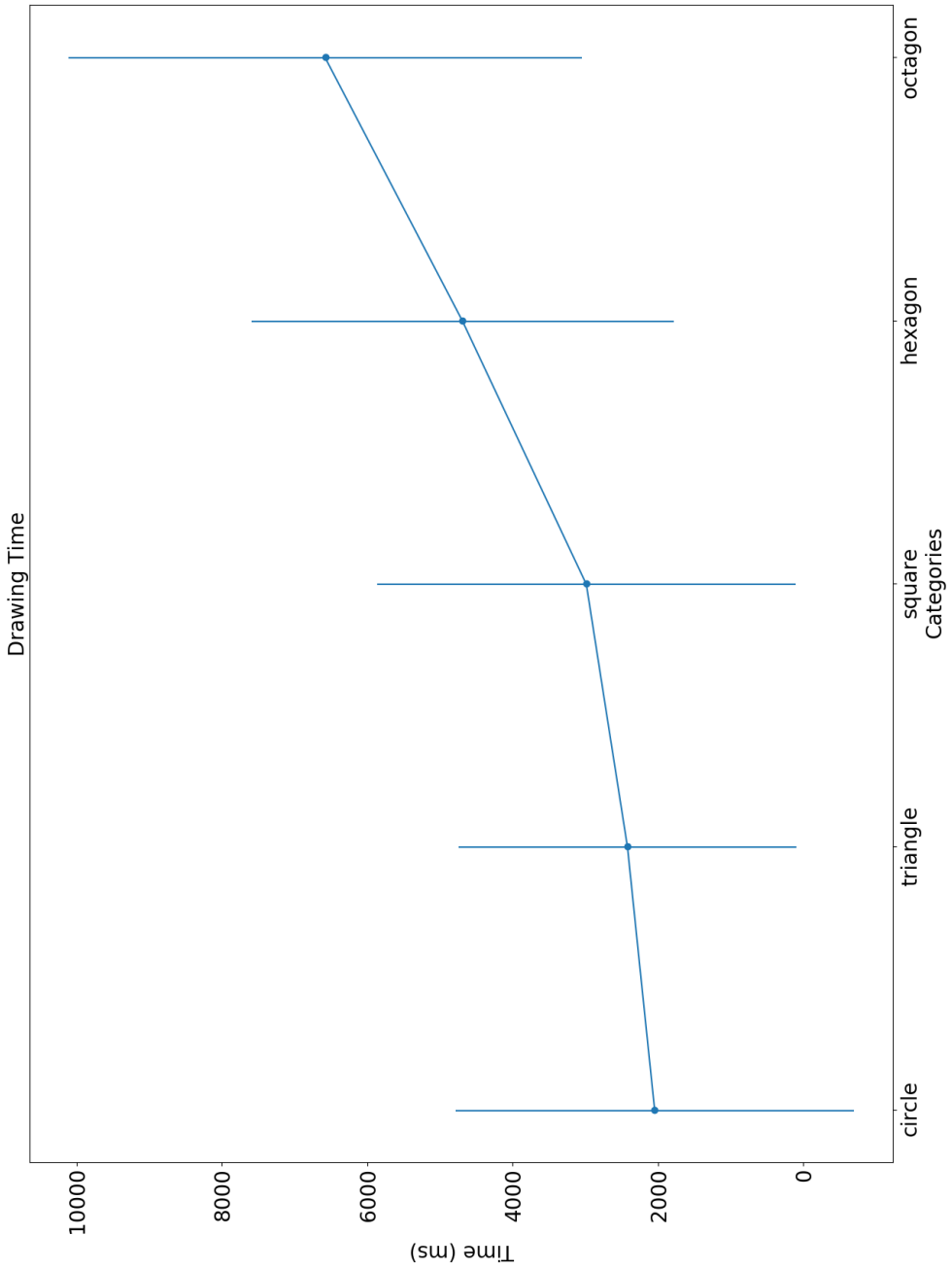



Figure 2.11: QuickDraw! drawing time statistics for each of the selected categories.

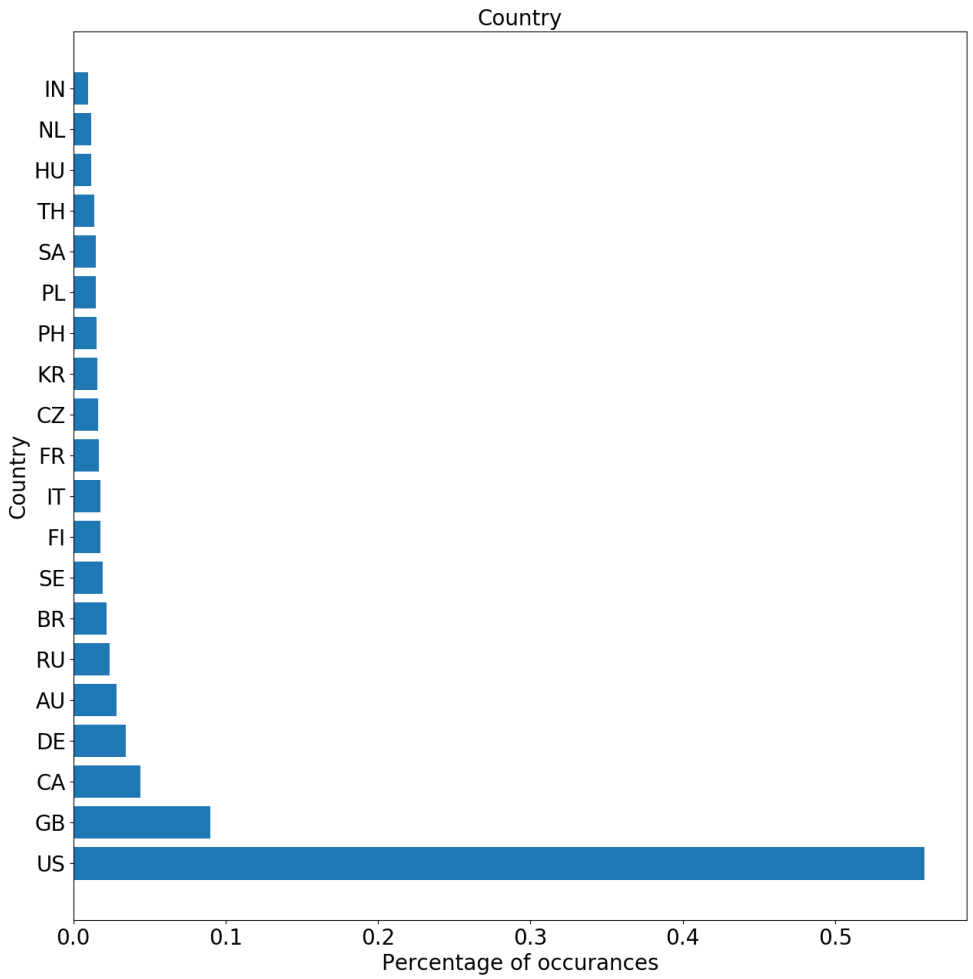



Figure 2.12: The most dominant countries for the players in QuickDraw! dataset. In the sample we analyzed, there is players from around 160 countries, but the majorty are from United States, followed by Great Britain.

tion, having more example images for these two categories will always lead to a better performance). Another way is to reduce the number of irrelevant/unnecessary patterns to be learned from the data. This is the task of *feature selection*, which is a fundamental step in machine learning. The target is to increase the amount of signal-to-noise ratio in the data. All-in-all, the objective is to increase the ratio of $data/patterns$, either by adding more data (if it is possible, or by using synthesis data using methods like *Generative Adversarial Networks* (?)), or removing irrelevant patterns.

- Simplify the learning procedure (differentiability/richness of distributions): while working with deep neural networks provide us with a lot of power in modeling a large variety of task, it also imposes some constraints. For example, the whole learning pipeline must be differentiable (this is a limitation imposed by the optimization methods, which will be discussed later). There are a lot of already available *off-the-shelf* functions that can be used, but care must be in the design of the experiment, to make it fit with these functions. Sometimes however, these functions are not enough to model the task properly. Thus, there is a need to adapt new tools to fit within the neural networks paradigm. This including finding a proper way to differentiate them, or, if not possible, to move around the non-differentiability problem (usually by finding a surrogate function to optimize), which is not always a straightforward task for deep learning practitioners.
- Keep enough patterns to perform the intended study: it is quite tempting to focus on the scores of the machine learning algorithm, while forgetting about the original task. In our case, machine learning is a tool to help us perform our analysis, but not the final objective. For example, a low-level quantization of the X and Y traces of the pen will remove a lot of patterns, and will make it easier for the algorithm to learn the data distribution. But it will also remove essential information about the styles in this case.

In the following subsections, we will discuss two different data representation choices (continous versus discrete representation, and the features used), and the implications of each of them on the machine learning , and the study of the styles.

### 2.4.1 Continuous or Discrete representation?

The X, Y and pressure of handwriting tracings are always recorded as continuous distribution, while the pen state is discrete (categorical). Thus, it probably makes sense to model the data in their native form. In the neural network design, a typical design choice will be to use a linear activation function as output function, and the *Minimum Square Error* (MSE) as loss function. Unfortunately, it is not that straightforward.

**Continuous Data Representation** To understand the problem, we first need to consider how a simple feed-forward neural network works, figure 2.13. The input is $x$, the output of the network is $a$, and we want to predict $y$. The neural network tries to project $x$ to a new space $z_{21}$ through a series of continuous folding of space. Then, the rule of the last activation layer is to model $P(y|z_{21})$.

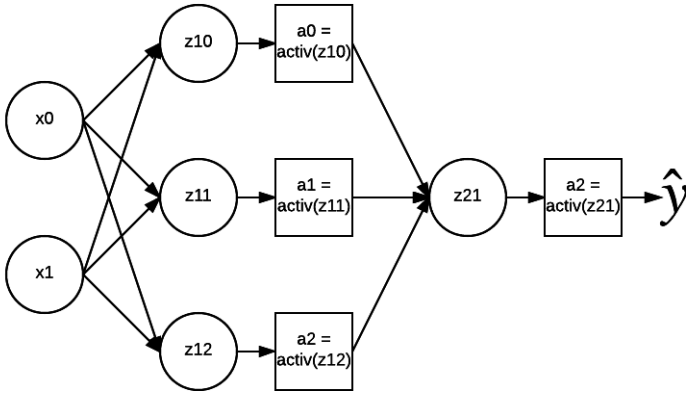



Figure 2.13: Example of a single-hidden layer neural network. The circle units is the sum of the weighted neurons connected to this unit, and the square units is the application of the activation function on that sum.

Although a linear activation is very simple, $y = z_{21}$, it is also shown to have limitations. In (?), a simple example is shown (see figure 2.14). If each input $x$ gets a unique output $y$ (one-to-one mapping), then linear activation performs well (figure 2.14, left). But if the input can have multiple possible outputs (one-to-many), the model learns to average over these outputs (figure 2.14, right). The author concludes that a simple linear activation function is not powerful

enough to represent a complex/rich distributions. He then proposed the use of a *Gaussian Mixture Model* (?) (GMM) as the final activation of the neural network, which is powerful enough to enable modeling complex continuous distributions, and avoids the problems of a simple linear activation function. This combination of neural network and GMM is called *Mixture Density Network* (MDN). The loss function in this case is changed from the MSE to the a posterior log-likelihood of the GMM. The neural network output in this case is not the required prediction directly, but the parameters of the GMM (means, variances, weights, correlations). The required prediction is then sampled from this parameterized GMM.

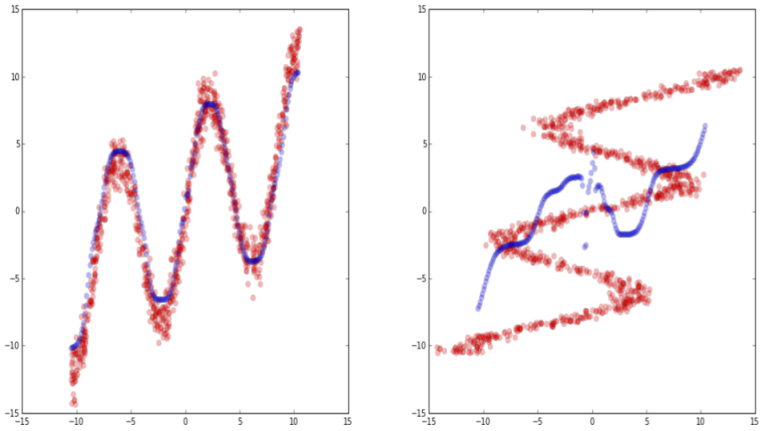


Figure 2.14: The performance of the simple linear activation function on two setups. Left: in case of one-to-one mapping between the input and the output, it performs well. Right: In case of one-to-many mapping, the function starts to average over the seen observations, leading to undesirable behaviour. Source of this image is (?).

The work done in ? demonstrated a system which generates impressive results on handwriting demonstration. He used an adaptation of the MDN for temporal data. Other applications for MDN can also be found in speech synthesis (???). While there is no question about the power the MDN approach provides, it was reported in (?) that training model kept collapsing (the explosion of gradients, the loss going to $inf$ or $NaN$), thus requiring tweaks and tricks in order to train properly. This is in-line with the experiments I performed with MDN, leading to similar conclusions.

**Discrete Data Representation** Discrete representation of originally continuous data requires transforming the data via feature engineering and/or quantization step on the raw data. While it is guaranteed there will be some information loss in the discrete data, discretization provide a lot of robustness to noise (this is why it is used in digital communication for example), and flexibility (many tools based on information theory do exist to handle digital data). Plus, a categorical distribution does not make assumption of the shape of the data distribution, unlike continuous distributions. The use of discrete distribution and how to infer from a discrete distribution will be covered later in section 3.1.

The challenge in this case is to choose a good quantization technique, that preserve the relevant information in the original signal one one side, and while keeping the dimensionality of the problem to a tractable level. For example, in case of speech synthesis, the authors in (??) showed that applying the $\mu$-law to the raw speech signal, and then quantize it (thus, the quantization is not linear), revealed a superb sound quality. This is better than performing naive linear quantization on the data, and saves a lot of memory as well: with non-linear quantization, they only need 256 levels. To get a similar behavior with linear quantization, around 65K quantization levels or required.

### 2.4.2 Feature engineering: Direction and Speed

As argued in the previous section, we choose discrete representation for our data. A requirement for a good quantization scheme should keep the important information in the original signal[3]

In case of *IRONOFF*, the letters tracings has been cleaned by removing points related to false starts or corrections as well extra strokes. Tracings were re-sampled with $10ms$ time interval, and the ones with length exceeding 1 second has been removed, as well as tracings more than 100 time steps. This is because they are quite rare, thus, their existence would significantly degrade the performance of our model.

For *QuickDraw!*, the re-sampling interval is $20ms$, and consider trac-

ings that are less than 200 time steps. This is because the tracings tend to be longer than in *IRONOFF*. This start to became more clearer when the task gets more complex (the octagon being the most complex one).

We represent each letter tracing by two features: directions and speed. Each feature is quantized into 16 levels and represented as a one-hot encoded vector.

Freeman codes (?) is used in order to encode the direction feature. It belongs to a family of compression algorithms called *Chain Codes*. This set of algorithms proved to be useful to encode an image with connected components. They can transform a sparse matrix to just a small fraction of the size of the image, in the form of a sequence of codes. Thus, they are being used as compression algorithms as well.

Freeman codes can N-directional codes (where N are the directions), depending on the needed resolution. It is quite simple as it encodes each direction with a unique number from 0 to N-1. A direction is defined as the directed vector connecting two neighbouring pixels on the contour of a connected component in the image.

We compute the change of directions between three consecutive points. Then, we map this change to its corresponding freeman code number, as shown in figure 2.15. Last, we transform the direction number into one-hot encoding scheme, and use this as input to our network. We also quantize the speed of each displacement.

## 2.5 Summary

In this chapter, we explored two datasets: Cursive Handwriting dataset, *IRONOFF* and sketch drawing dataset *QuickDraw!*. Each of these datasets contains several tasks/categories (the letters/digits in case of *IRONOFF*, and the shapes in case of *QuickDraw!*), and we can explore the styles around each of those tasks.

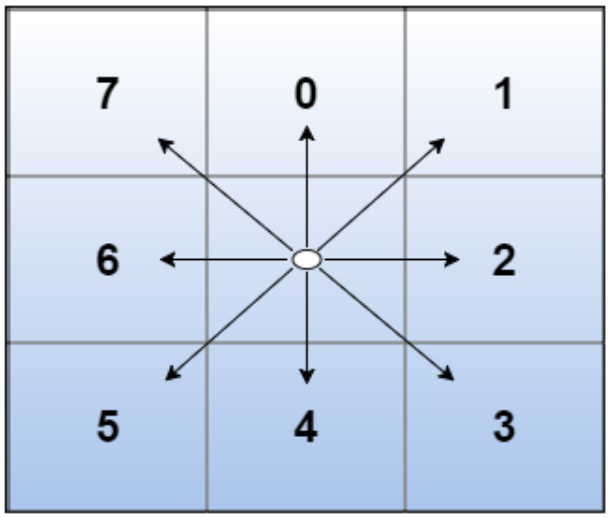



Figure 2.15: Example for freeman code representation for 8 directions. Each direction is given a unique number.

We explored basic information about each task (number of strokes, drawing time, and the pausing time).

We motivate the use of these datasets because we can see that there is a variance in these information of each task, suggesting different styles for each task. This variance differs a lot depending on the complexity of tasks: in *IRONOFF* for example, letter *c* seems to be the simplist task, thus, the variance in it is relatively not that large. This variance increases as the task get more complex (letter *E* for example). In *QuickDraw!* dataset, a similar trend can be observed (the *circle* being the simplest task, and the *octagon* being the most complex one).

Last, we argue that although that *QuickDraw!* has simpler tasks than *IRONOFF*, it is actually more complicated. We hypothize that this is due to the fact that there is a huge variety in the players (many different countries, compared to mosty *French* people in *IRONOFF*). The players use mostly the mouse in order to draw[4], which leads to behaviors usually unobserved with hand drawing/writing. Also, in such environment, it is expected that the human curioisty will prevail, and people will try to draw complex shapes, outside the limit of the required task, in order to see if the neural network classifier will

recongize it correctly or not.

We will consider that there are two heirarichty of tasks in *IRONOFF*: the first is uppercase, lowercase letters, and digits. The second is the individual letters and digits. When we perform transfer learning, we will do it on the tasks of the first heirarichy only (for computational reasons).

We will use *QuickDraw!* side-by-side to *IRONOFF* in our last part of our work in order to validate our approach and conclusions, but the first two parts will be done on *IRONOFF* only.

# Notes

[1]This is not wrong in itself, it depends on the objective of the work. In our case, we want to show an indication that our methods can generalize.

[2]Although there is no reporting about the tools used in the drawing, it is a valid assumption to assume that the mouse is the main tool.

[3]In some applications, like in digital communications for example, the quantization (or digitization process) has other rules, like compressing the signal, increasing the signal-to-noise ratio, and increase the robustness of the signal. This is outside the scope of our work however.

[4]The players did not receive any special training on drawing with the mouse beforehand.

# Part II

# Experiments

# Chapter 3

# Generation, benchmarks and evaluation

## Contents

Since styles are ill-defined, they can not be evaluated explicitly (we can not quantify them in advance). Thus, we need a proxy method in order to implicitly evaluate them. We can do this by using them in order to generate behaviors (i.e., synthesis handwriting traces), and evaluate the quality of those behaviors relative to the target behaviors (i.e., the original letters traces). In other words, to study styles, we would like to use reconstruct the target behaviors using generative models, and given the task and the styles to perform the behavior. We discuss the recent advances in neural generative models: how are they trained? how do we infer from them?...etc. Then we look at paradigm for reconstruct target behaviors, with focus on the usage of neural networks.

After we generate the behaviors, we need to evaluate them in a manner that evaluates the styles. This is still an open research question, and a complicated one as well. It also goes hand-in-hand with the choices made in generative models. We discuss the used evaluation metrics across different domains (e.g., text, speech and handwriting evaluation). We also discuss the difficulties associated with finding the suitable evaluation metric.

I then present the experiments done in order generate letter and ground our proposed evaluation metrics. I will explain our choices for the model design, the inference methods, and our approach to ground the metrics.

Questions addressed in this chapter

- How to generate handwritten letters using deep learning framework?
- How to evaluate the generated traces?
- What benchmarks are suitable to compare to?

## 3.1 Background

### 3.1.1 Deep Learning: quick overview

Deep learning is a subset of machine learning (??), mainly applicable on neural networks. These set of techniques perform remarkably well nowadays on a wide range of tasks and benchmarks (for example, in image recognition (???), speech synthesis (?), image segmentation, handwriting recognition, image captioning (??), language translation(?)), even outperforming humans in some of them (GO game (?)).

GB: I would distinguish between recognition tasks (that discard irrelevant variability) and generation/prediction tasks (that have to deal with output variability) OSM: 07/06: Noted. Will update it.

Before deep learning, in order to use classical machine learning algorithms, an important step was *feature engineering*: extracting the relevant features from the data, in order to get the relevant information needed to perform the task. In images, techniques like *Scale-invariant feature transform* (SIFT) (?), and in speech, feature like *Mel-frequency cepstrum* (MFCC) and *Probabilistic Linear Discriminate Analysis* (PLDA) (?), were quite dominant at that time. There is no one solution that fits all here; it depends on the task in hand, and that required experience. Thus, feature engineering was quite challenging.

The advantage of deep learning techniques is that it overcome (to a big extent) the need for the daunting task of feature engineering. Instead, it tries to learn, from the raw data, hierarchy of features, that are optimal in order to solve the task in hand (i.e., it performs *automated feature engineering*). In our work for example, we did not need to do any kind of temporal feature extraction.

A proper discussion into the basics of deep learning is out of the scope of this document (and has been done properly in many books and tutorial available online). We refer you to the excellent book (?) for more theoretical treatment of deep learning, and to (??) for a more practical aspect.

### 3.1.2 A word about sequential data

Sequential data appears in a lot of our daily life, for example: text, speech, the weather status,...etc. In a more formal manner, a sequential data example is formed of tokens, and can be represented as $x_1, x_2, ..., x_N$, where $x_n$ is one token, and $N$ is the total length of this sequential data point.

Let's take a more concrete example: text. We have multiple sentences in a given text. We can consider each sentence as a data point/example. If we consider an example sentence: *the cat is eating the food*, and we define our tokens to be the words[1]. In this case, $x_1 = the$, $x_2 = cat, \dots$ $etc$.

What is common between all the sequential data (and what also distinguish them from non-sequential data points) is that each token is dependent on the previous token/s. In general, when we want to model sequential data, we in essence want to learn the following probability distribution:

$$p(x_1, x_2, ..., x_N) = p(x_1) \times P(x_2|x_1) \times p(x_3|x_2, x_1) \times .... \times p(x_N|x_1...x_{N-1}) \tag{3.1}$$

In order to model such a distribution using neural networks, we can either:

1. Adding a *state variable* in order to factorize the problem. In this case, we introduce an intermediate state variable, $h$, which model the dependency on the previous tokens. Our objective in this case will be

   $$p(h_n) = p(h_{n-1}, x_n) \tag{3.2}$$

   This can be done using *Recurrent Neural Networks* (RNN), which will be explained later in detail (section 3.1.3). In this case, the network is looping over the tokens one-by-one, updating the state variable each time.
2. Use the whole sequential point (all its tokens) as input to the network in the same time. In this case, the network is trying to model equation 3.1 heads-on. This approach has gained popularity recently with the work done in ((??)), using convolution networks in order to model sequential data (speech for the first, language text for the second). The advantages

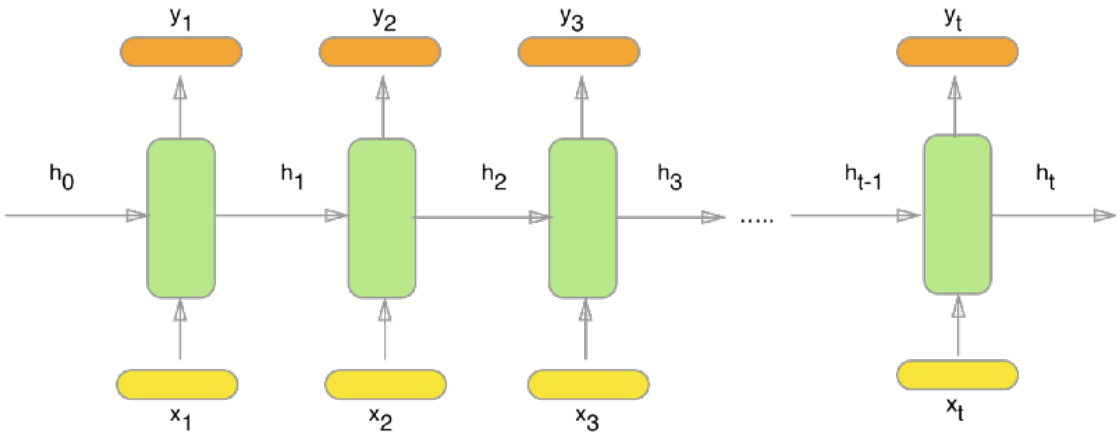



Figure 3.1: A demonstration of how RNN works: the network is applied on each token in the input $(x_1, x_2, ..., x_t)$, while update the hidden state variable every time $(h_0, h_1, ..., h_t)$. The output at each step is a function of the hidden state variable (not demonstrated here). Source of the image is ((?)).

of this approach is removing the sequential aspect of the network, and leveraging parallelism when treating the whole sequence. This results in faster training, while still achieving state-of-the art results.

There is no one solution that fits all here, it simply depends on the problem and the constraints on the solution. In case of variable-length sequential data, the first approach is the one to go for. In the case of fixed-length sequential data, the second approach should be considered (easier to train the model, faster to optimize, faster to infer from).

### 3.1.3 Recurrent Neural Networks and Sequence Generation

As mentioned briefly in the previous section (3.1.2), *Recurrent Neural Networks* (RNN) is a type of neural networks, that can handle sequential aspect in data. In its simplest format, it is a simple feed-forward network, applied on each token in the sequential data point, while carrying the information about the previous tokens using a latent variable $h$, commonly referred to as *the hidden state variable*. This is demonstrated in figure 3.1.

To describe it in a more formal manner, let's first assume the following:

- The input $x$ is first processed by a set of weights, $W_{ih}$, and bias $b_{ih}$.
- From each step to the other, the hidden state $h$ is processed via a set weights, $W_{hh}$, and bias $b_{hh}$.
- Last, the output is given by processing the hidden state $h$ via another set of weights, $W_{ho}$, and bias $b_{ho}$.

If we assume that output activation is a simple linear layer (thus, it is a regression task), then the equations of this simple RNN are the following:

$$\begin{aligned} z_n &= W_{ih} \cdot x_n + b_{ih} \\ h_n &= W_{hh} \cdot [h_{n-1}, z_n] + b_{hh} \\ y_n &= W_{ho} \cdot h_n + b_{ho} \end{aligned} \tag{3.3}$$

Where the brackets [ ] indicate a concatenation process. In case of a simple regression task, a typical loss function is the *Minimum Square Error*, which can be formulated as:

$$Loss = \frac{1}{T \times N} \sum_{n}^{N} \sum_{t}^{T} (y_{n,t} - \hat{y_{n,t}})^2 \tag{3.4}$$

where $T$ is the length of the of the sequence, and the $N$ is the number of sequential data points available, and $\hat{y_{n,t}}$ is predicted output by the model, while $\hat{y_{n,t}}$ is the ground truth output.

Given this formalization, the objective is to find the set of weights and biases of the network, that will minimize the chosen loss function. We will explore the optimization algorithms in section 3.1.4.

### 3.1.4 Optimization Algorithms

One of the important factors in the recent success of deep learning is the advances in optimization algorithms. These algorithmic advances make the optimization easier, converges to a better solution, and less tweaking is required

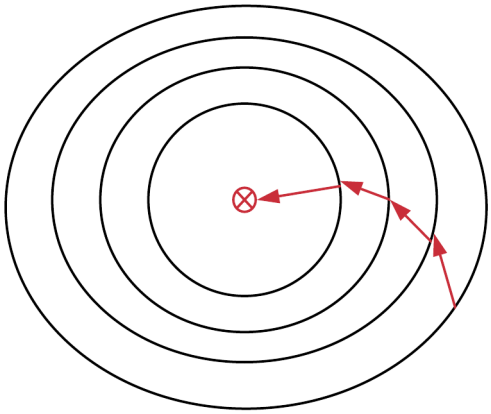


Figure 3.2: A demonstration for how a gradient descent algorithm work. The optimization process progress each step towards the global optima (the indicated point in the center).

in order to achieve a good performance. All these methods are derivatives of gradient descent optimization, defined as(?):

> Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point.

If the function to be optimized is convex((?)) – differentiable, and have a global optima –, then gradient descent can find this global optima. An example for the different optimization iteration can be seen in figure 3.2.

Gradient descent can be formalize by the following equation:

$$\theta_{t+1} = \theta_t - \alpha \times \frac{dLoss(\theta_t)}{d\theta_t} \tag{3.5}$$

where $\theta$ refers to the parameters we want to optimize, $t, t+1$ refers to the current and the next iterations respectively, $\alpha$ is the learning rate (how large

the step to take at each iteration), and *Loss* is the loss/objective function we want to optimize our parameter for. Optimization by gradient descent is a *batch optimization*: the loss function and its gradient is calculated all given data examples. This iterative process keeps going till we wither do not observe a tangible change in the performance of the parameters, or we exhaust our computational budget. An illustration for this process can be seen in figure 3.3.

Before we dive into the different optimization methods currently used, it is important first to stress on an important characteristic of all modern deep neural networks, which are ((?)):

- **Differentiability** All the components of the neural networks (activation function, loss objective, regularization, ...etc) are differentiable components. This issue is not about mandatory by theory, but by convenience: good optimizers exist that can use this feature in order to optimize the network faster, while being able to scale with appropriately with the given number of data points and the number of parameters to be optimized.
- **Non-convexity** While the neural network is built of convex parts, the composition of those parts together is not convex. The reasoning for this particular point is out of the scope of this document. We refer you to the excellent book ((?)) for more details –. While this seems to be a disadvantage, it is actually a powerful advantage of the neural network. Neural networks are *universal approximator*, meaning that they can approximate/learn any function. A convex function can not approximate non-convex function, but the other way around works ((?)).
  Using gradient descent optimization methods in this case leads to convergence to local optima points. The initial conditions of the network (the initial random parameters, and the strategy of their selection) and the optimization strategy and parameters (learning rate, decay factor,...etc) will play an important rule to determine the convergence characteristics (speed of convergence, and local optima) of the neural network.

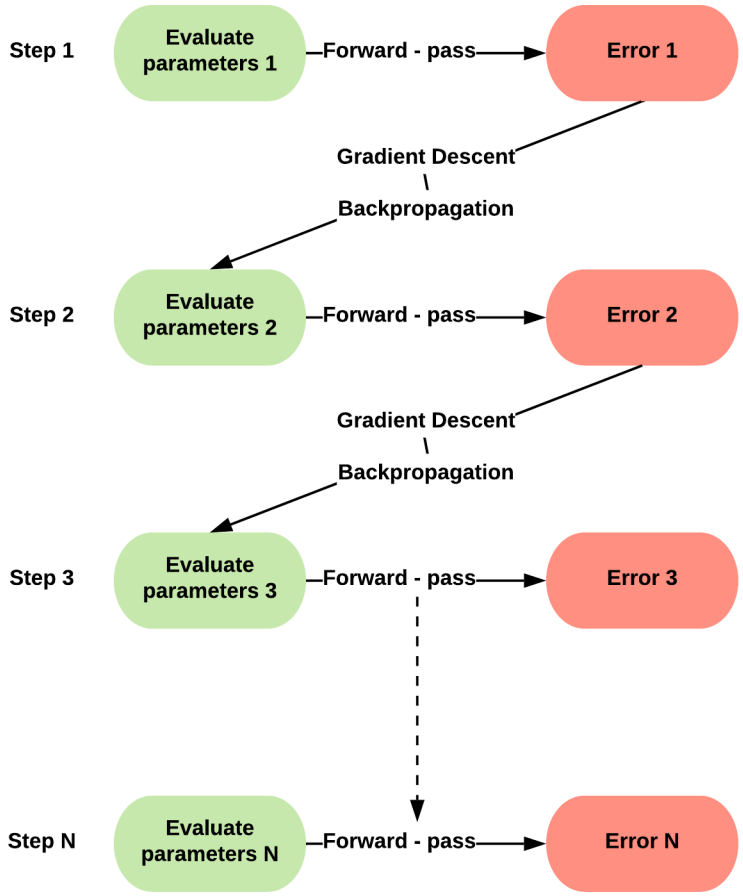



Figure 3.3: The process of gradient descent is iterative, and include 3 steps: evaluate the quality of the current parameters (forward-pass step), get the error value, use the error-value in order to update the parameters/getting new set of parameters (back-propagation step). This process is repeated N times, which is decided by either not observing any update in the error, or we ran out of computational resources.

**Mini-batch optimization**

One important aspect of the success of deep learning is the availability of large amount of data. To optimize a deep neural network (can be the order of millions of parameters) using large amount of data (can be in the order of millions of data examples) using gradient descent, is not physically feasible. We do not have the hardware capability to process all of these data in the same time.

To get around this issue, *mini-batch* optimization is used: instead of performing gradient descent based on the information from the whole data, we divide the data into chunks (mini-batches). For each mini-batch, we calculate the loss and the gradient, and update the parameters, and then move on the next batch. The gradient descent in this case is called *Stochastic Gradient Descent*, SGD, ((?)), following the same equation 3.5, but applied to only a mini-batch at a time, instead of the whole data (batch).

Many advances built on top of SGD helped in advancing deep learning, like combining SGD with momentum ((?)), RMSProp algorithm ((?)) and Adam algorithm ((?)).

### 3.1.5 Architectures

OSM: INCOMPLETE PART

In section 3.1.3, we discussed the idea behind the basic RNN and its operation. Then, in section 3.1.4, we how we can optimize the parameters of a neural network in general.

One important aspect of RNN is the ability to keep the information over many time-steps. This is an issue however with basic RNN networks. This can happen with a better choice of the architecture, that allow to keep the memory, which also learning to select what is not necessary to keep (what we can forget). This problem is called *vanishing gradient*.

A formal discussion for *vanishing gradient* is extensively and better treated in many excellent resources. To get an intuition for it, let's re-phrase what gradient descent optimization is actually doing: the network parameters are evaluated over the data, giving us an error value (the evaluation of the loss function). Using this error value, we move/update the parameters of the network in a direction that minimizes the future error. We keep repeating these steps till we no further improvement is observed (or we exhausted our computational budget). In RNN,

Talk about Vanilla RNN (and what is their problem), LSTMs (and the intuition behind then in solving the problems), and then GRUs (simpler than LSTMs – fewer parameters –, yet perform similar)

### 3.1.6 Inference: How to generate sequences from the network?

There exists several approaches in order to generate information from the networks. In the case of recurrent neural networks, this is a sequential process (one step at a time, till we generate the whole sequence), as illustrated in figure 3.4.

During the inference mode, the objective is to generate the most likely sequence. i.e., we want to solve the following problem

$$\begin{gathered}\text{Find } x_1, x_2, ..., x_N \text{ that} \\ argmax_{x_1, x_2, ..., x_N} p(x_1, x_2, ..., x_N)\end{gathered} \tag{3.6}$$

This problem, however, is not tractable, as it scales badly with the number of options (i.e, dimensions) per time step, and the number of time steps required. One simple way is to perform *greedy sampling*, where, at each time step, we select the most likely token. This, however, leads to repetitive and predictable patterns ((?)).

A better way will be to to use *stochastic sampling*, by leveraging the

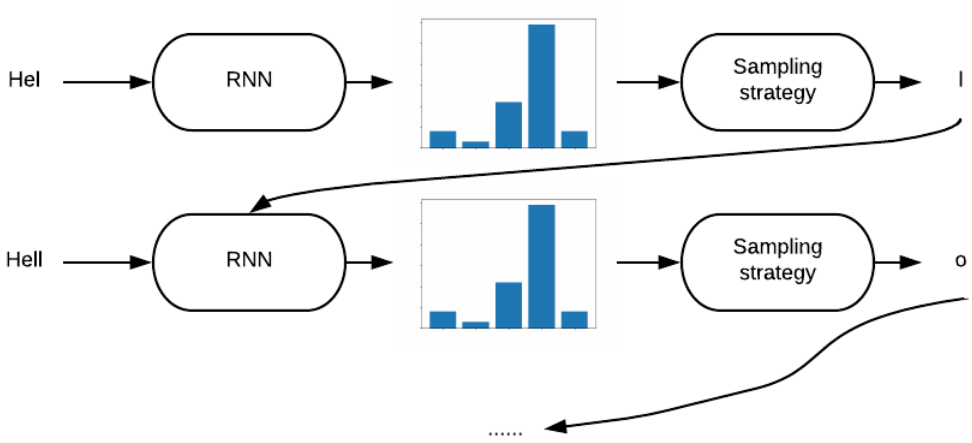



Figure 3.4: An example of using RNN in order to infer a sentence. The token to be generated here is a character. At each time step, the model is given the part of the sentence that has been generated so far, and asked to give the probability distribution over the next character. This distribution is then given to the selected sampling distribution, which sample the next character, and so on.

fact that at each step, we have a probability distribution over all possible tokens. This allows more diversity in the generated tokens, and also allow unlikely tokens to be sampled some of the time.

But what if we want to control the level of randomness in the sampling process? having such control will allow us to explore different ways to infer from the model, in order to determine the most satisfying way. This control can be done using *temperature sampling*. The idea is to reshape the probability distribution over he different token. On one extreme, very high temperature (going to infinity) will flatten the distribution, making the distribution equivalent to *uniform distribution*. On the other extreme, a temperature of zero will be mount to greedy sampling. This is illustrated in figure 3.5.

### 3.1.7 How to introduce prior to the model? (conditioning the model)

When training a RNN network, we initialize the first hidden state with zeros. This way, we are informing the model that we are not making any prior assumptions about that particular sequence, and that all sequences in the data have the same 'no prior assumption' condition.

However, what if you want to add some prior knowledge about the sequence to the model? There are two reasons why we may want to do that:

- Increase accuracy: In case we are doing some classic pattern recognition task (classification, regression), having extra useful information will definitely help increasing the model final performance.
- Act as a command: In case of generative model – during the generative mode – we need a way to *trigger* the model in order to start generating a sequence in a particular context. For example, we want to tell the model to generate letter 'A'. Initializing the model with this value allow it to start generating letter A.

There are multiple ways to bias the model, all targeting the same thing:

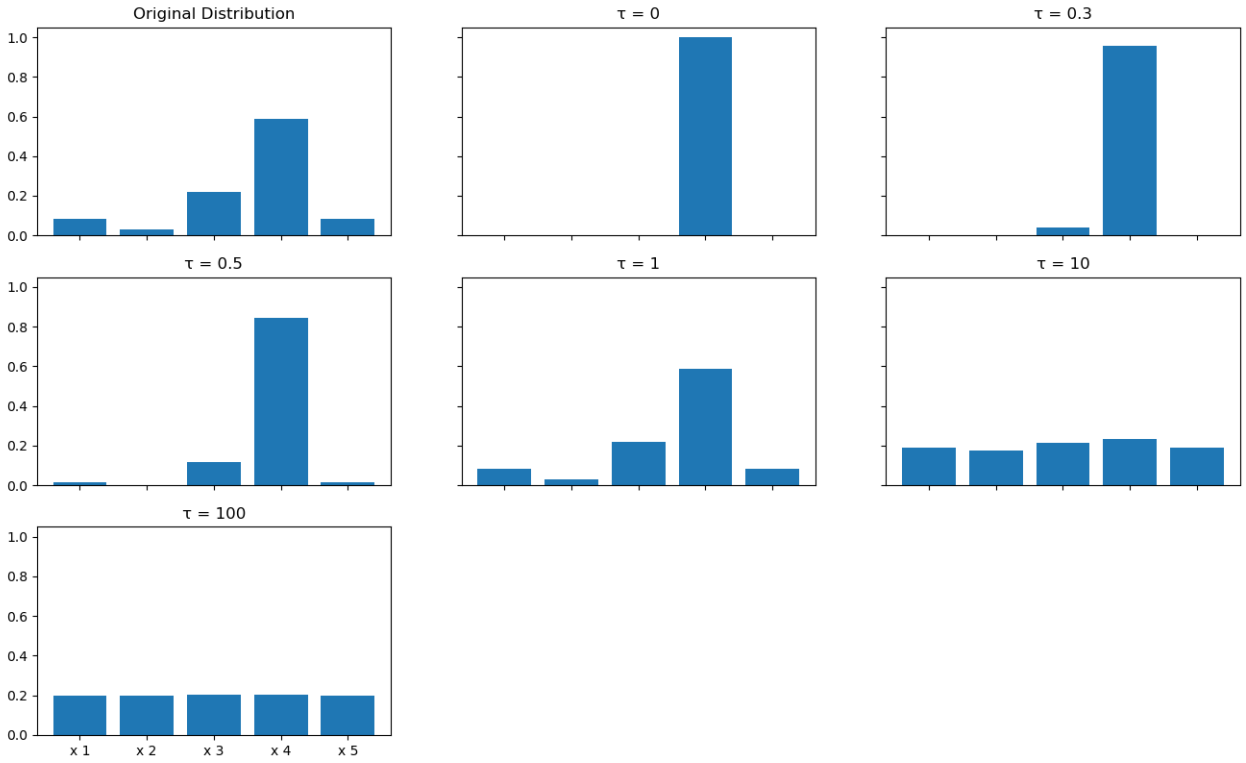



Figure 3.5: Illustration of temperature sampling. When the temperature $\tau$ is very low, it becomes greedy sampling. With the increase of temperature, we can see more higher possibility of sampling the lower-probability tokens. When the temperature is too high, it becomes uniform sampling.

initializing the hidden state of the model. Some work in the literature combines multiple of these approaches in the same setup. To the best of my knowledge, there is no one place which all these methods are discussed together.

**Initialize the first hidden state directly** This is a common approach, used image captioning ((?)), machine translation, and sequence-to-sequence autoencoders. This is illustrated in figure 3.62.

**Using the first time-step** This method was used in the work done in ((?)), in the area of image captioning. The prior in this case is the image, and the objective is to generate the text caption for it. In order to condition the model, the authors projected the image information into the same size as the word embedding used, thus creating a *fake* word, and concatenated this new word with the rest of the words. This is similar to have a word at $X_{-1}$. The hidden state after this fake word is now condition on the information from the image. This is illustrated in figure 3.61.

**Using context sequence – multiple time-steps –** In this approach, the model is provided with some time-steps from the ground truth, in give it a context. At the end of the these given time-steps, the hidden state is initialized with information about what to be done. A use case for this scenario – for example – is when training a language model on multiple authors. When asking the model to generate, you can provide some sentences from the author you want, so the model can follow on this.

**Concatenating input time-steps with the condition** In the work done by (?), although not mentioned explicitly, the dataset used – the *QuickDraw* dataset, discussed earlier – is quite complicated – they choose different tasks than us –. It is hard to make the model remember the information about such a complex task over a long time span. Thus, the authors use a mix of *initializing the first hidden state* and *concatenating with the first time-step* in order to make it

easier for the model to remember the task. This is illustrated in figure 3.63.

**Concatenating the hidden state with the condition** This approach is more popular now, since it allows the use of *attention mechanisms* ((??)). Examples for this in image captioning ((?)), speech synthesis ((?)).

### 3.1.8 How to evaluate the quality of generation?

OSM: INCOMPLETE SUB-SECTION

**NOTE: Explore this problem in speech and statistical machine translation/text generation.**

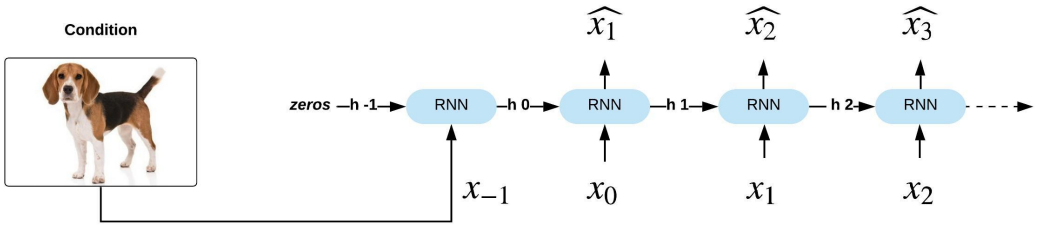
- divide into sections: static and temporal data
- text, speech evaluation
- using an oracle model (the error concatenation problem, and the adversarial attack).

The problem of evaluating the quality of the generation/synthesis process is a challenging task in the domain of generative models.

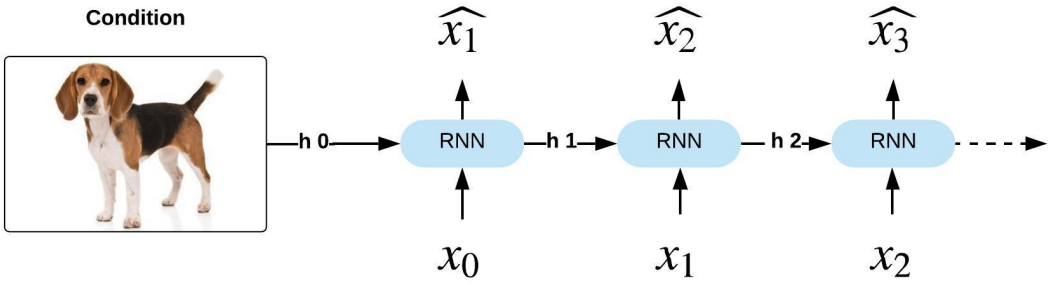
## 3.2 Putting all of that together

In the previous section, we explored multiple building blocks for the work to come, like recurrent neural networks (architectures, training, inference and conditioning). We also discussed the issue of evaluation the output of generative models, discussing three aspects: evaluation in case of text (image captioning, translation, and text generation in general), speech synthesis, and the dangers of using another model (oracle model) in order to evaluate the generation quality.

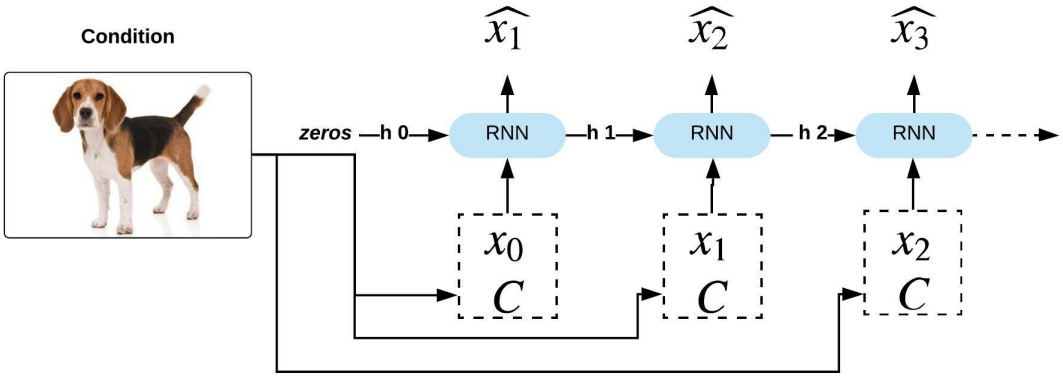
In this section, I discuss how did we put all these elements together,




1. Initializing using the first time-step




2. Initializing the hidden state




3. Concatenating the condition with time-step

Figure 3.6: Different conditioning method for RNN

decisions we made, and the experimental setup we used, in order to address the following three questions:

- How to generate handwritten letters using deep learning framework?
- How to evaluate the generated traces?
- What benchmarks are suitable to compare to?

### 3.2.1 Our proposed evaluation metrics

As discussed earlier, evaluation is a challenging problem when using generative models. We want metrics to capture the distance between the generated and the ground truth distributions. Following the work done in ((?)), we use the same two evaluation metrics in our model: *BLEU-score* ((?)) and *End-of-Sequence*.

**BLEU score** It stands for *bilingual evaluation understudy*, it is a well known metric to evaluate text generation applications, like image captioning (??) and machine translation (?). Since we discretized the letter drawings, this fits nicely within our work. The general intuition is the following: if we take a segment from the generated letter, did this segment happen in the ground truth letter? We keep doing this for segments of increasing length (the length of the segment here is the number of grams used in the BLEU score). For our work, we report the results on segments from 1 to 3 time steps.

Each part of the letter has two parallel segments: freeman codes and speed, thus, we report the BLEU score for both of them. The equation to compute the BLEU score is the following:

$$BLEU_N = \frac{\sum_{C \in G} \sum_{N \in C} Count_{Clipped}(N)}{\sum_{C \in G} \sum_{N \in C} Count(N)} \tag{3.7}$$

$$Score_N = \min(0, 1 - \frac{L_R}{L_G}) \prod_{n=1}^{N} BLEU_n \tag{3.8}$$

where: $G$ is all the generated sequences, $N$ is the total number of N-grams we want to consider. $Count_{Clipped}$ is clipped N-grams count (if the number of N-grams in the generate sequence is larger than the reference sequence, the count is limited to the number in the reference sequence only), $L_R$ is the length of the reference sequence, $L_G$ is the length of the generated sequence. The term $\min(0, 1 - \frac{L_R}{L_G})$ is added in order to penalize short generated sequences (shorter than the reference sequence), which will deceptively achieve high scores.

In order to get into the intuition of using such a metric in evaluating the quality of handwriting generation, we can imagine that we are comparing segments of a generated trace to segments in the ground truth letter, by asking the following question: does the segment in the generated trace exist (anywhere) in the original trace? The shorter segments represent *adequacy* (i.e., does the generated trace use the same elementary moves like the ground truth trace). The longer segments represents *fluency* of drawing (i.e., how good the drawing is)[2]. See figure 3.7.

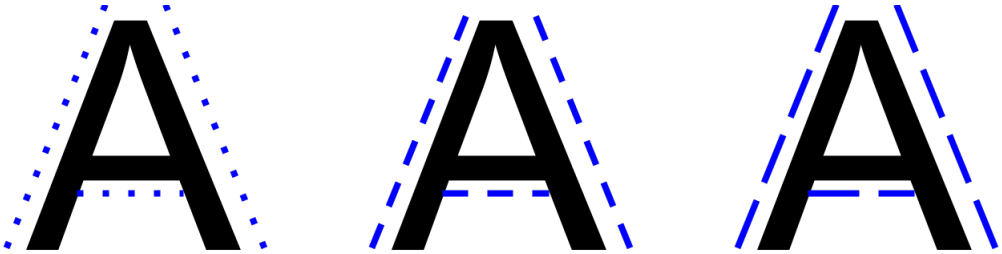


Figure 3.7: Using BLEU score of different sizes, we compare segments of variable length in the generated trace to the target trace.

**End of Sequence** The length of the letter is another aspect of the style. The distribution of length in the generated examples should follow the ground truth examples. In order to perform this analysis, we compute *Pearson correlation coefficient* between the generated examples and the ground truth data.

### 3.2.2 How to ground the metrics?

We assess multiple methods to condition our handwritten letter generator, and evaluate their ability to capture of writing styles. We know their cardinal order of the power of these methods (depends on the kind of information available to each method). Knowing this information beforehand, we can use it to ground our performance metrics. The methods are:

**Letter identity** the letter id only is used as bias. No style information is thus included. The model will try to average over the different example for the same letter. We consider this as a lower baseline.

**Letter + Writer identities** the letter id and writer id are used as a bias. Thus, the model has an explicit access information about the writer. This method is expected to perform the best. This model will also serve as a upper baseline.

**Image classifier embedding** We train a convolution neural network (CNN) to classify the letters images[1], as shown in figure 3.8. We use an intermediate layer as to extract embeddings, that will encode information about the letter images. This model should perform the same or a more performance than using the letter identity only, since it learns to clusters the letters, and there are classification errors. But we expect it to perform less than the letter + writer identities.

**Image auto-encoder latent space** we train a letter image autoencoder, using reconstruction error, and use the latent space as a representation of the letter + style. The architecture we use can be seen in figure 3.8. The latent space encodes the similarity between the letters. This model should perform worse than using the letter identity only, since, while it capture the similarity between the letter images, it does not capture discriminative features about each letter itself.

From this discussion, we can say that cardinal power of the different

[1]This letter classification task achieves 95.1% classification accuracy, which we consider very good.

conditions is:

$$autoencoder < letter \leq classifier < letter + writer \tag{3.9}$$

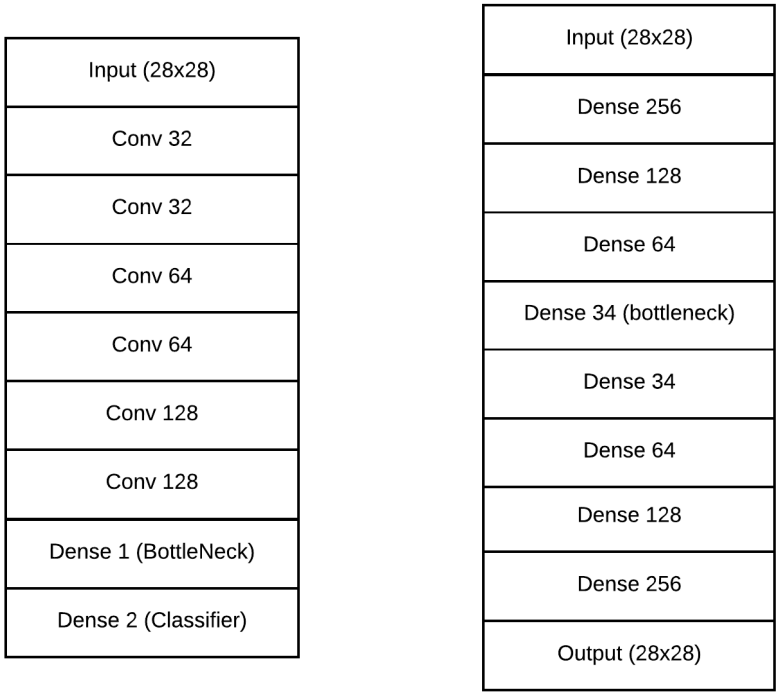
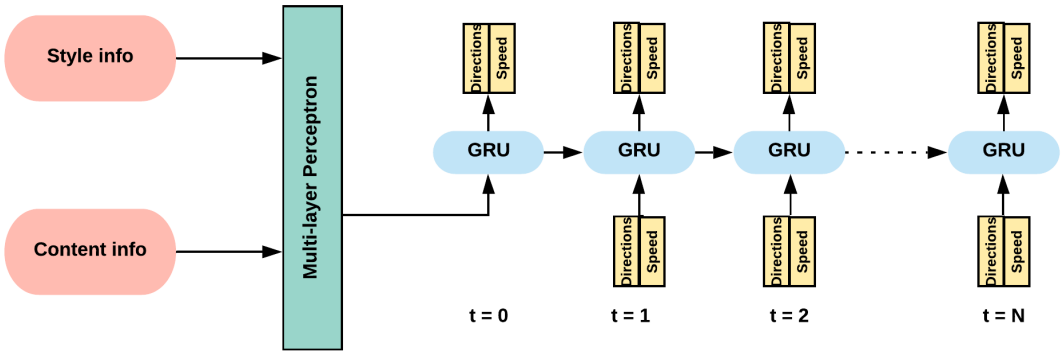


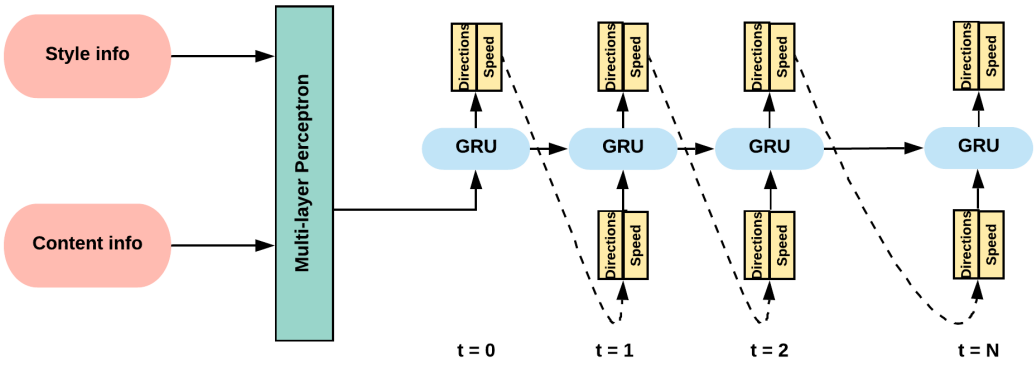

Figure 3.8: Left: architecture of the CNN letter classifier. Batch normalization is used after each convolution layer. The *Dense 1* layer is the embedding that is used to condition our generator. Right: the autoencoder architecture we used. The first *Dense 34* layer provides the latent space used to condition the generator.

### 3.2.3 Model used

In order to answer the questions mentioned earlier, we use conditioned-GRU model, demonstrated in figure 3.9. Using this model, we compare different style approached discussed in section 3.2.2, and use the generation results from that model in order to ground the proposed evaluation metrics, discussed in section 3.2.1.




1. Training mode




2. Generation mode

Figure 3.9: The conditioned-GRU model used in this work. During the training mode 3.91, the input of the model is always the ground truth, and the predicted value is compared to the ground truth. During the generation mode 3.92, the input to the model at each step is the model prediction from the previous step. The 'style info' and the 'task info' inputs are separated here for demonstration (they could be mixed).

### 3.2.4 Results

We train our different model and generate the traces from them as explained earlier. In this section, we compare the different models using the evaluation metrics discussed before. We observe the consistency of the reported metrics with the prior information about the cardinal power of the different methods, equation 3.9. This is how we ground our metrics

#### BLEU score

The final results using the BLEU score can be seen in table 1. The results vary when measuring BLEU-1. But, as we increase the number of grams, BLEU-2 and BLEU-3, to measure the similarity between larger segments of the traces, we can observe:

- The letter + writer condition performed better than all other conditions, thus showing that having access to information about the writer, like the writer id, improve the quality of the handwriting synthesis.
- The image classifier condition performs better than the letter identity only, but less than the letter + writer bias. Since the classifier is trained on a single objective only (to classify the letters), and the classifier performs well, we expect the embedding to cluster the letters well, as seen in figure 3.102. We can expect the model to capture some of the writer style, possibly in the inter-cluster variance. This is an interesting result, suggesting that some fine tuning for the image classifier while in the generation task could be beneficial to capture more details about the styles.
- The image autoencoder bias performed the worst. To understand why, we plot a 2-D projection of its latent space using t-SNE (?), figure 3.101. Since the autoencoder is trained to minimize the reconstruction error, the distance in the latent space encode the proximity between the images. It can be observed also that this latent space does not encode discriminative features for the letters. Using this latent space for our generator, we find the model gets confused between nearby letters, resulting sometimes in

generating different letters than requested.

| Aspect/Feature | Speed | | | Freeman | | |
|---|---|---|---|---|---|---|
| Model / B-score | B-1 | B-2 | B-3 | B-1 | B-2 | B-3 |
| Letter identity | 49.7 | 37.3 | 24.2 | 47.4 | 36.6 | 26.8 |
| Image classifier | 50.9 | 38.2 | 24.6 | 48.5 | 37.9 | 28.1 |
| Image autoencoder | 51.9 | 37.9 | 23.1 | 46.4 | 35.0 | 24.5 |
| Letter + Writer identities | 51.5 | 41.4 | 25.1 | 56.7 | 39.4 | 28.3 |

Table 1: Comparing different approaches for style extraction using clipped n-grams

#### Sequence length

As mentioned earlier, we performed a statistical test between the paired distributions of lengths of the generated and the reference tracings. The results are shown in table 2. We can see the following:

- The results from the statistical test shows that the letter + writer bias outperform the rest of the biases, achieving p-value $< 0.05$. This is quite reassuring, since it is also in line with the results from the BLEU score.
- The results from the Pearson correlation coefficients are also consistent with the rest of the results. High coefficients are given to the letter + writer bias, compared to the other methods. The image classifier and autoencoder gives the lowest results. This could be due to insufficient information about the letter length that can be inferred from the image. For the image classifier, as noted earlier, a fine-tuning during the generation task is worth exploring.

### 3.2.5 Examples of the generated letters

The design choices of our experiments (discretization, and ignoring the pen state) affects the final shape of the letters, yet, the letters and their style are

Figure 3.10: a) In the the autoencoder latent space, there is no clear separation between letters; the encoding is based on the similarity of the images only. b) In the classifier embedding, there is a clear separation between the letters - with few exceptions -.

| Models | Pearson coefficient | p-value |
|---|---|---|
| Letter bias | 0.38 | 0.84 |
| Image classifier | 0.32 | 0.62 |
| Image autoencoder | 0.25 | 0.29 |
| Letter + Writer bias | 0.55 | 0.04 |

Table 2: Pearson correlation coefficients and associated p-values for the EOS distributions of the different style biases.

quite recognizable. See examples for the original letters in figure 3.11. Examples for the generation with our methods are in figure 3.12. This is a subjective indication that our model is working properly, producing real-like comprehensible letters.

## 3.3 Summary

**TODO**

# Notes

[1]There is no rule here for how we define the tokens. We can, for example, define the letters to be our tokens. In the case of text, it is a tradeoff: considering words as tokens allow the model to be more fluent, but it also means that the learning space is very large (sometimes the number of unique words to predict at each time step is in the order of tens of thousands). If we consider letters as tokens however, it make the model job more tractable (all the letters, symbols, digits, can be in the order of tens), but it leads to less quality for the model.

[2]This last interpretation of short and long segments is adapted from the (?), which uses them for text translation evaluation.

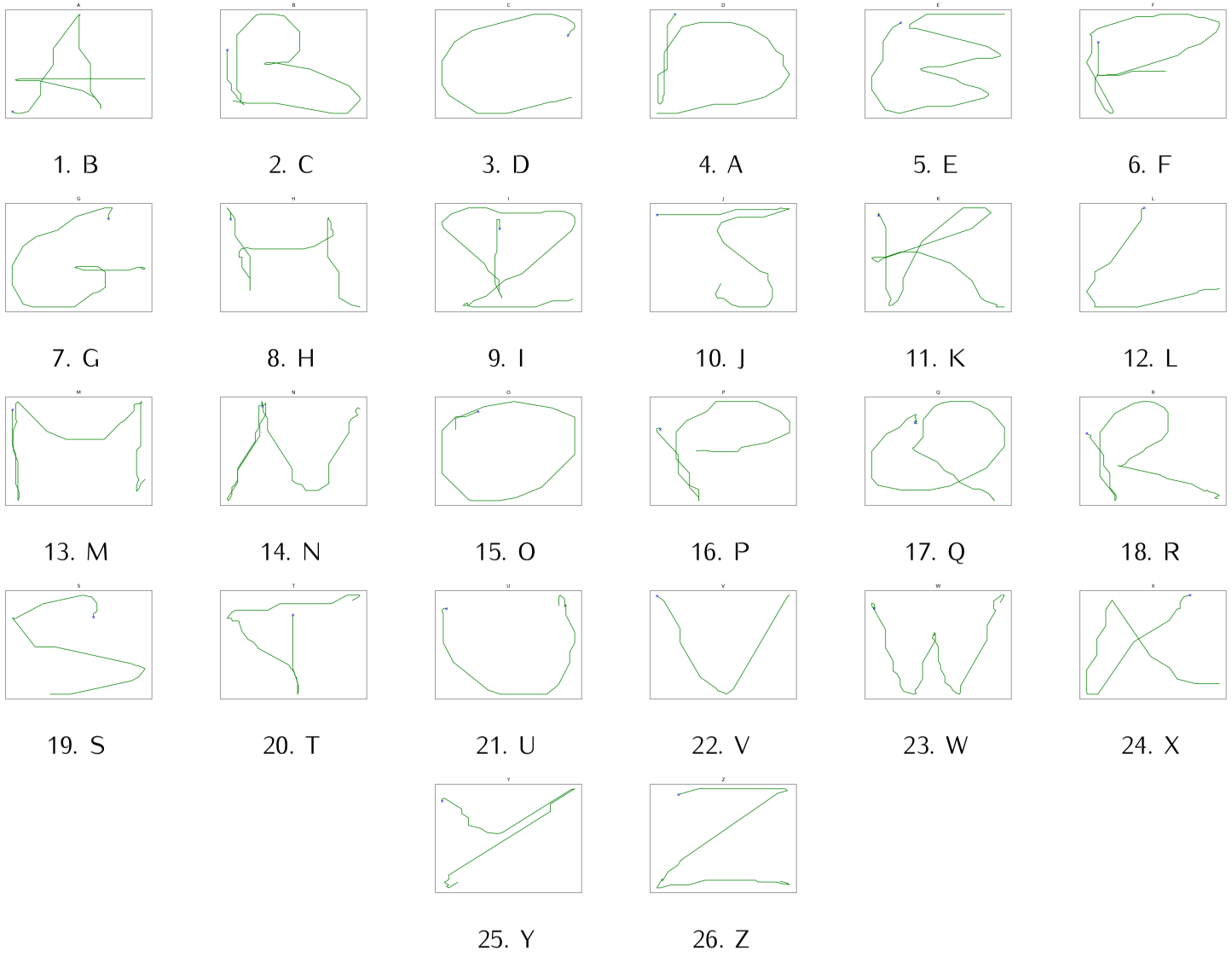



Figure 3.11: Examples of original letters. The blue *x* mark is the starting point. These ones are generated using the letter + Writer bias. E and F are visually harder to recognize, since we do not model the pen pressure, otherwise, the rest of the letters are well recognizable.

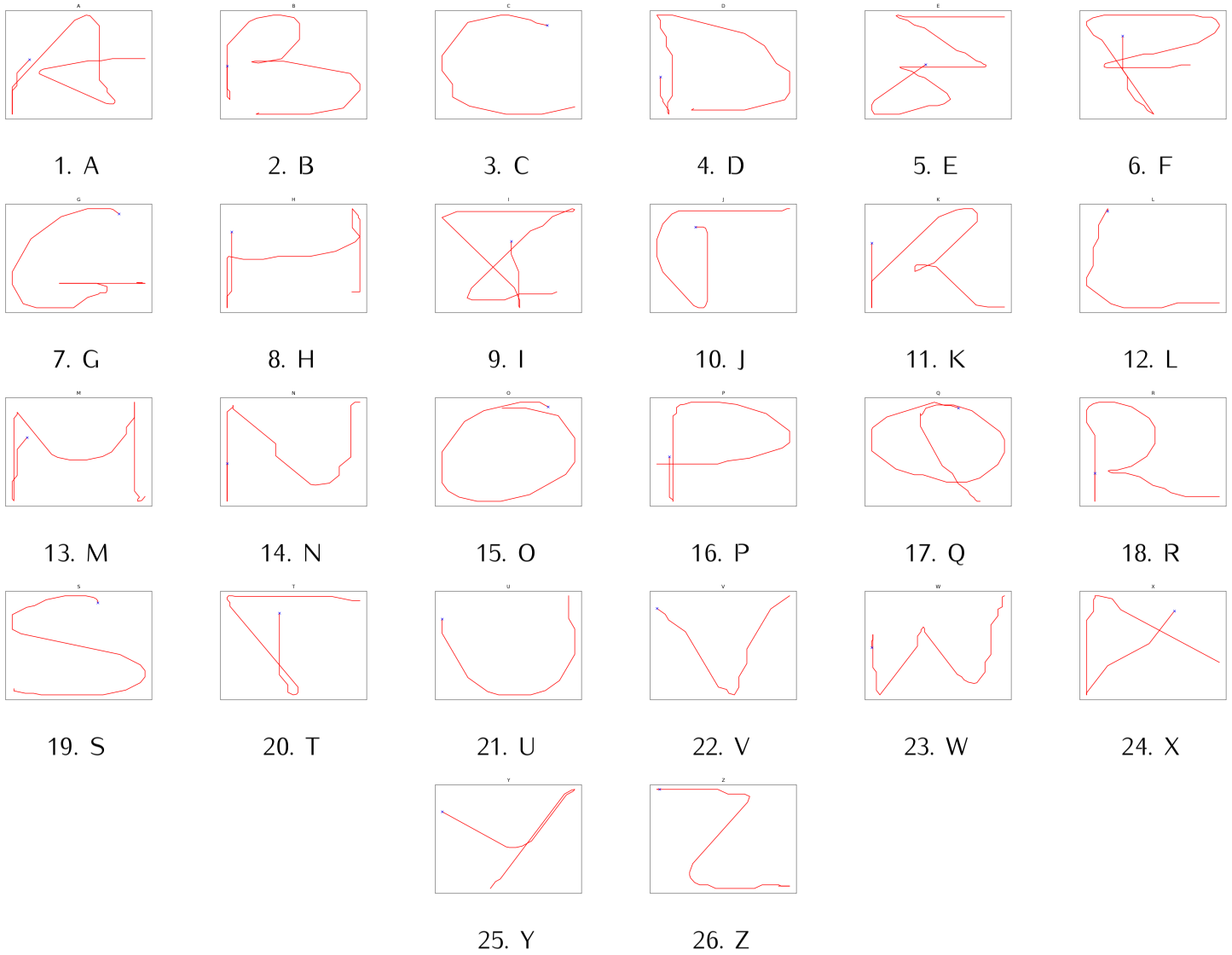



Figure 3.12: Examples of generated letters. The blue $x$ mark is the starting point. These ones are generated using the letter + Writer bias. The general quality of this quite acceptable.

# Chapter 4

# Framework

## Contents



In the previous chapter, we set the foundations of studying styles, which is the work done in the PhD. We explored our choices for generative models and the evaluation metrics, and how to ground them. We also had a discussion about lower- and upper-bound benchmarks. These foundations are necessary in order to have baselines to compare to, and performance aspects (i.e., metrics) use for this comparison. OSM: write a note about the fairness of selecting the baselines, and the reasoning behind them.

In this chapter, we build on these foundations, by proposing the use of conditional temporal auto-encoder framework in order to study and extract

styles, in the context of sequences (i.e., a time aspect exists in the data). We take advantage from the fact that the content is well known in our dataset (i.e., the identity of the letters in IRONOFF, and the identity of the shapes in QuickDraw!).

Questions addressed in this chapter

- What possible framework to study styles? and why?
- How does this framework performance compare to the benchmarks?
- What kind of styles we can extract from this framework? and how do we extract them?

## 4.1 Background

### 4.1.1 Auto-encoders

- General intro about auto-encoders
- Applications of static auto-encoder in image Reconstruction
- Temporal auto-encoder
- different ways to bias temporal encoders

### 4.1.2 Objectives

### 4.1.3 Static and Temporal auto-encoder

## 4.2 Putting it all together

### 4.2.1 Letter generation with style preservation

The objective here to compare the quality of the generated letters to the state-of-the-art benchmarks. As mentioned earlier, we compare using the BLEU score metric and the EoS analysis. The BLEU score results can be seen in table 1, and the results for EoS analysis results are in table 3. We can see that the BLEU-3 score results of our model achieves 32.3% accuracy in Speed feature and 38.7% accuracy in Freeman feature, compared to 25.1% and 28.3% accuracy using the benchmark model on both features respectively.

The same goes for the EoS analysis. In comparing the Person Coefficient, our model achieves 0.99 score compared to 0.55 for the benchmark model (the highest score is 1.0). This is a support that our model capture the style of handwriting better than the benchmark.

Examples for the generated letters can be found in figure 4.10.

| Aspect/Feature | Speed | | | Freeman | | |
|---|---|---|---|---|---|---|
| Model / B-score | B-1 | B-2 | B-3 | B-1 | B-2 | B-3 |
| Letter + Writer bias | 51.5 | 41.4 | 25.1 | 56.7 | 39.4 | 28.3 |
| **Style Extractor** | 71 | 51.7 | 32.3 | 65.6 | 51.5 | 38.7 |

Table 1: BLEU scores for different models for known writers.

| Aspect/Feature | Speed | | | Freeman | | |
|---|---|---|---|---|---|---|
| Model / B-score | B-1 | B-2 | B-3 | B-1 | B-2 | B-3 |
| Letter + Writer bias | 55.4 | 39.6 | 25.3 | 50.2 | 38.6 | 27.7 |
| **Style Extractor** | 72.4 | 52.4 | 32.2 | 70.4 | 55.6 | 42.1 |

Table 2: BLEU scores for different models for style extraction for 30 new writers (style transfer).

| Models | Pearson coefficient |
|---|---|
| Letter + Writer bias | 0.55 |
| **Style Extractor** | 0.99 |

Table 3: Pearson correlation coefficients for the End-Of-Sequence (EoS) distributions for the different models on the normal gene ration scenario

| Models | Pearson coefficient |
|---|---|
| Letter + Writer bias | 0.5 |
| **Style Extractor** | 0.99 |

Table 4: Pearson correlation coefficients for the End-Of-Sequence (EoS) distributions for the different models on 30 new writers (style transfer).

### 4.2.2 Style transfer

One of the hypotheses we want to test is whether there is a limited number of styles needed, to generalize over new writers. To achieve this, the learned representation for styles should extract generic information about the styles.

In order to test this hypothesis, we expose our model to 30 writers that have not been seen before. We compare our model performance on these writers with a model is biased by the writer and letter identities (the benchmark model). The latter model was not constrained from seeing those writers (thus, the reported results of the comparison overestimates the actual performance of that model).

The BLEU scores can be seen in table 2. Our model achieves on BLEU-3 score 32.2% and 42.1% accuracy on the Speed and Freeman code features, compared to 25.3% and 27.7% on the benchmark model for the same features respectively.

The EoS analysis can be seen in table 4. Our model achieves a coefficient value of 0.99, compared to 0.5 for the benchmark. Thus, the new model clearly outperform the current benchmarks on the transfer task, on both BLEU

score and EoS analysis.

### 4.2.3 Styles per letters

One of the nice consequences of using our model is that we can have a better look at the styles. We explore the latent space for multiple letters, and see that we can uncover interesting writing styles. A full scale analysis is beyond the scope of this paper. We project the latent space using *Principal Components Analysis* (PCA) (?) and t-SNE (?).

As a start, we take a look at letter X. Beforehand, we identified a style feature in letter X: some writer draw X clockwise, and some draw it anti-clockwise. We manually annotated the whole dataset for this feature; the result can be seen in figure 4.1. Almost half of the writers draw the letter X clockwise, and the other half draw it anti-clockwise. If our assumption is correct, our model should be able to capture this feature. We project the latent of the model using PCA on all the letter X, which can be seen in figure 4.2. The model latent space clusters almost perfectly based on rotation. Examples for letters from both clusters are in figure 4.3.

Encouraged by the results on letter X, we explored more letters. For letter C, we can see the latent space project in figure 4.4. It can be seen that there are at least two main clusters. Examples from this cluster in the red ellipse are in figure 4.6. The indicated cluster represents the Edwardian handwriting style. The rest of the writers (in the big cluster) have a very similar style (this is expected, since the drawing of the letter C is quite simple).

For letter A, our model latent space create two main clusters, figure 4.5. We give examples from those two in figure 4.7, where we can see clear difference in the style. Some people start drawing the letter from down-left, other writers start from the top of letter A, move down, then continue drawing of the letter.

Another example is for letter S bottleneck, figure 4.8. There are three

resulting clusters which we investigated. The indicated cluster (in red) is clearly different from the other two clusters (not indicated). Examples can be seen in figure 4.9. The indicated cluster is again for people with Edwardian handwriting style. We did not find a clear difference between the other two clusters though, but this is an expected outcome of using t-SNE (since it does not have the clear objective of clustering styles).

These examples show is that we can use our model to extract verbose style information.

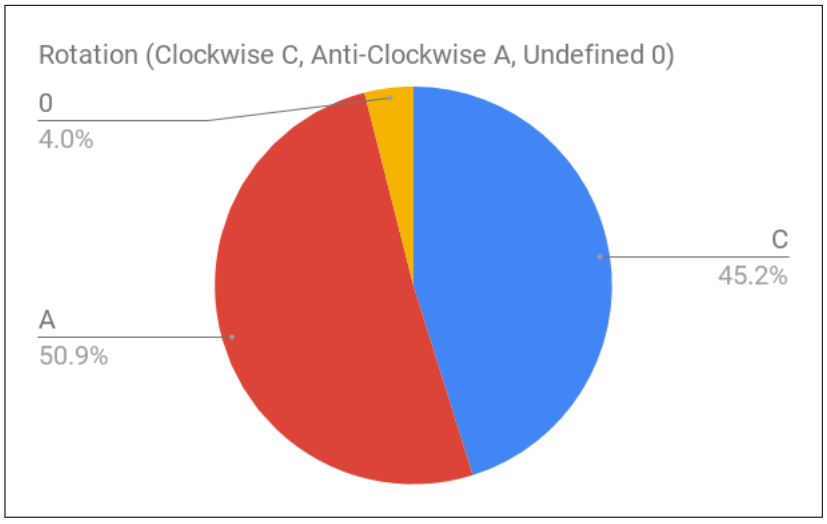



Figure 4.1: Results of the manual annotation for the rotation of letter X drawings over the whole dataset. Almost half the writers drew X clockwise, the other half anti-clockwise. The undefined styles were unclear to determine.

## Notes

[1]There is no rule here for how we define the tokens. We can, for example, define the letters to be our tokens. In the case of text, it is a tradeoff: considering words as tokens allow the model to be more fluent, but it also means that the learning space is very large (sometimes the number of unique words to predict at each time step is in the order of tens of thousands).

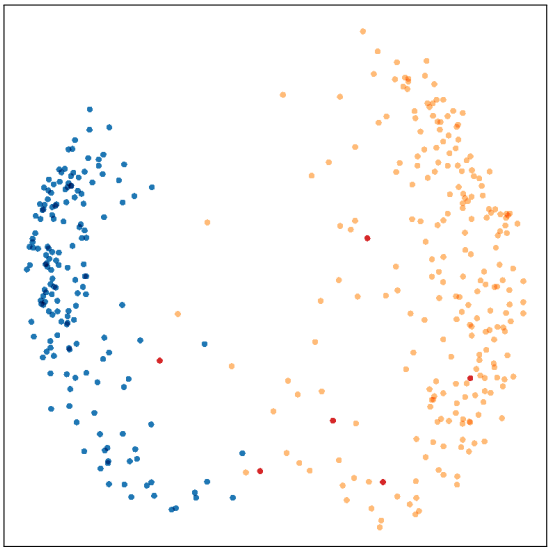


Figure 4.2: Projection for latent space for letter X using PCA. The colors show the ground truth of the X rotation: blue is counter clockwise, orange is clockwise, and the few red points are undefined.

If we consider letters as tokens however, it make the model job more tractable (all the letters, symbols, digits, can be in the order of tens), but it leads to less quality for the model.

[2]This last interpretation of short and long segments is adapted from the (?), which uses them for text translation evaluation.

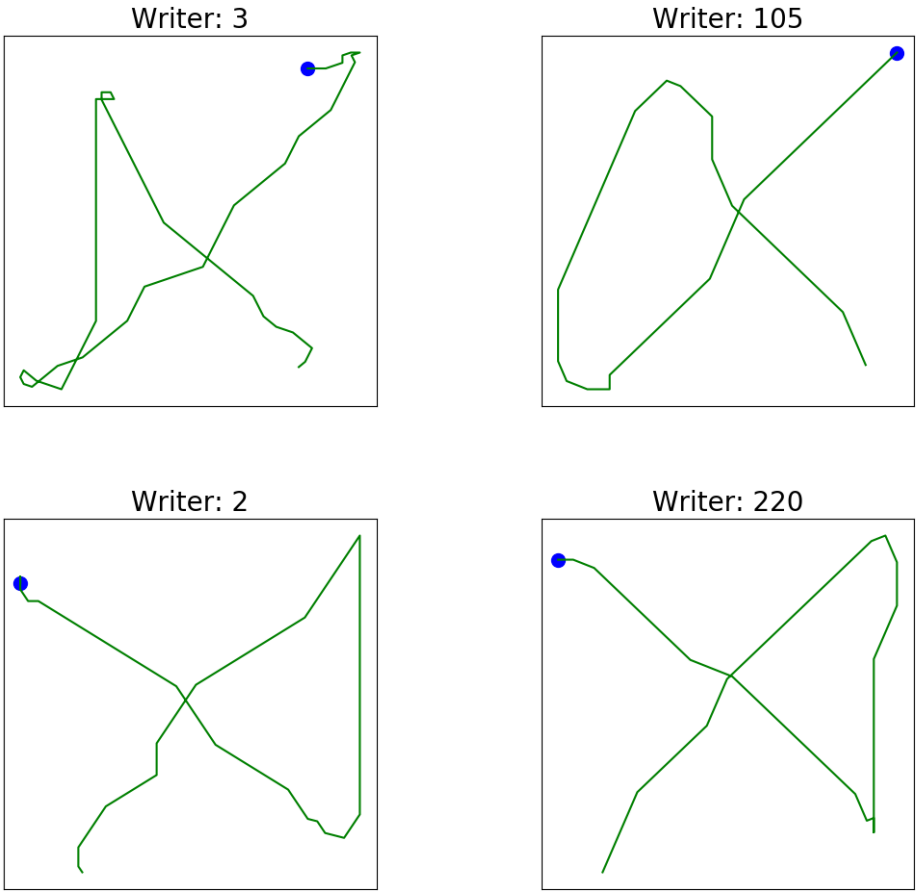



Figure 4.3: Examples for writing of letter X. Starting point is marked with the blue mark. Each raw is randomly sampled from each cluster in the bottleneck. The clusters shows that almost half the writers draw the letter clockwise (first row, first cluster), and the other half draw it anti-clockwise (second row, second cluster).

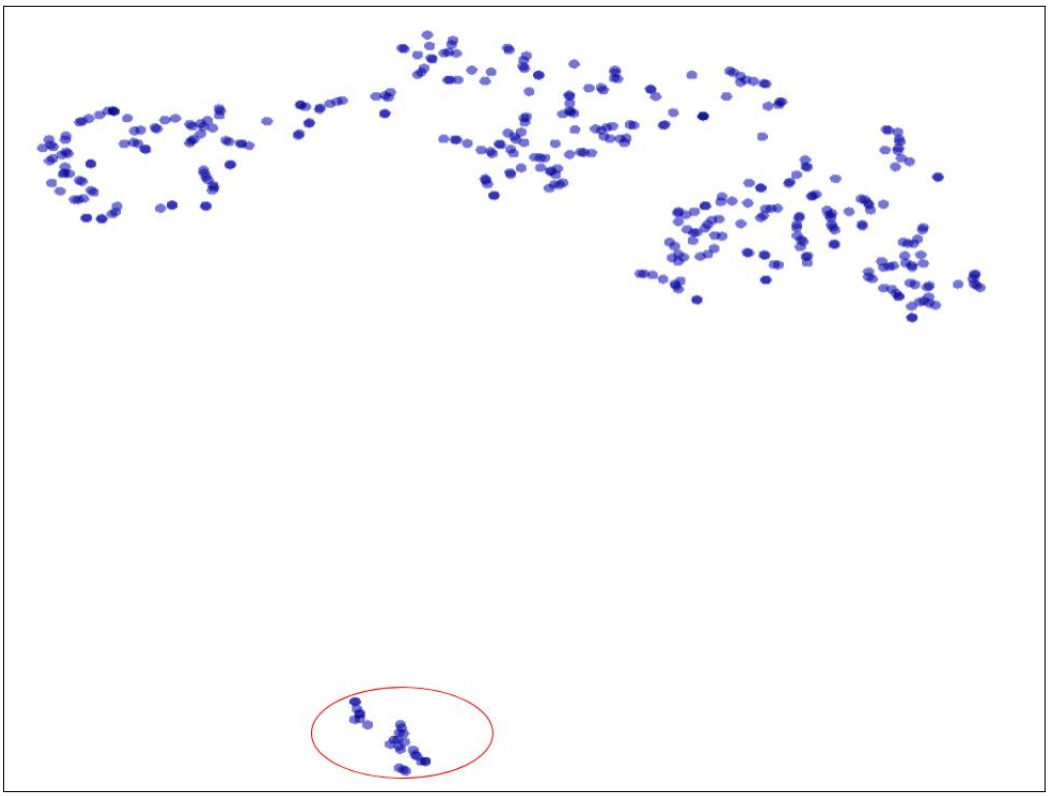


Figure 4.4: Projection for latent space for letter C using t-SNE. The cluster surrounded by the red circle has a clear interpretation, where writers have a cursive style.

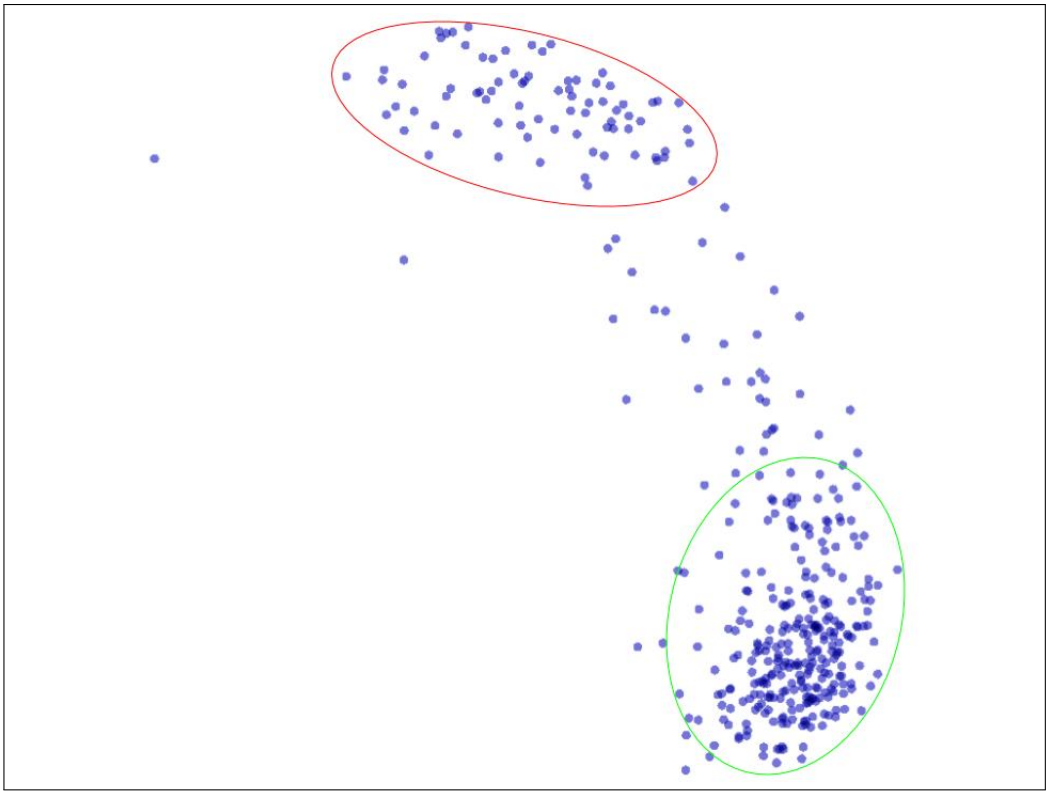


Figure 4.5: Projection for latent space for letter A using PCA.

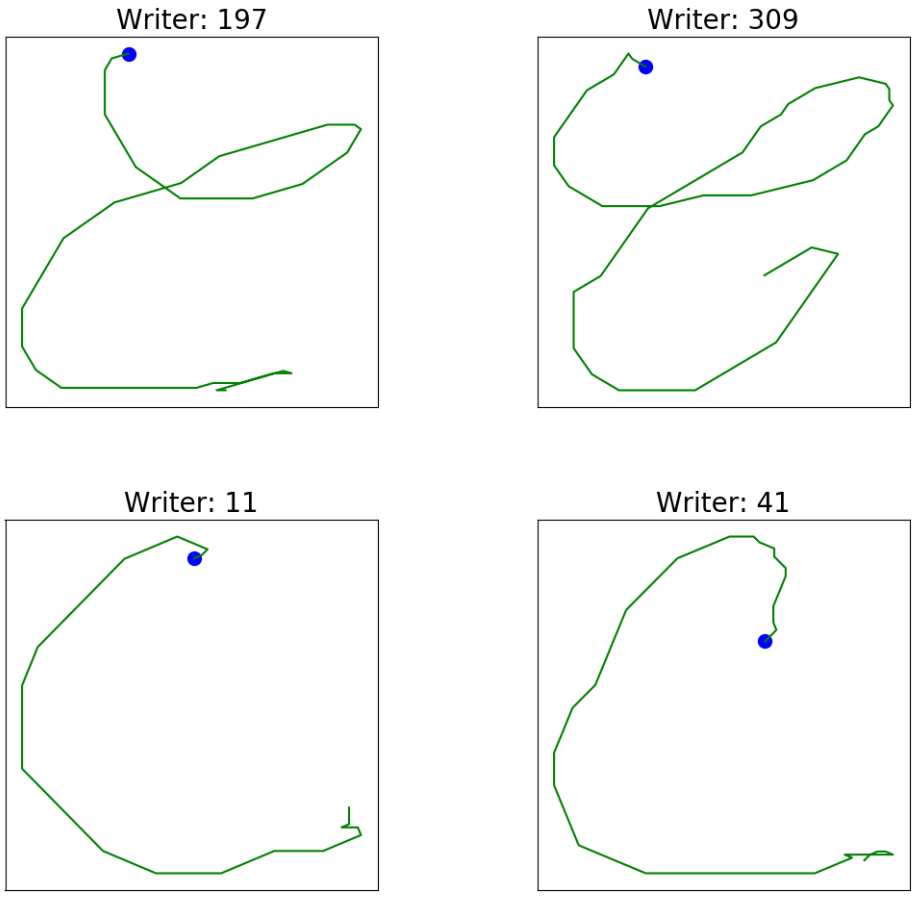



Figure 4.6: Examples for writing of letter C from the selected cluster (first row) versus the rest of the letter drawings (second row). Starting point is marked with the blue mark. The drawings from the selected cluster show people with Edwardian style of handwriting.

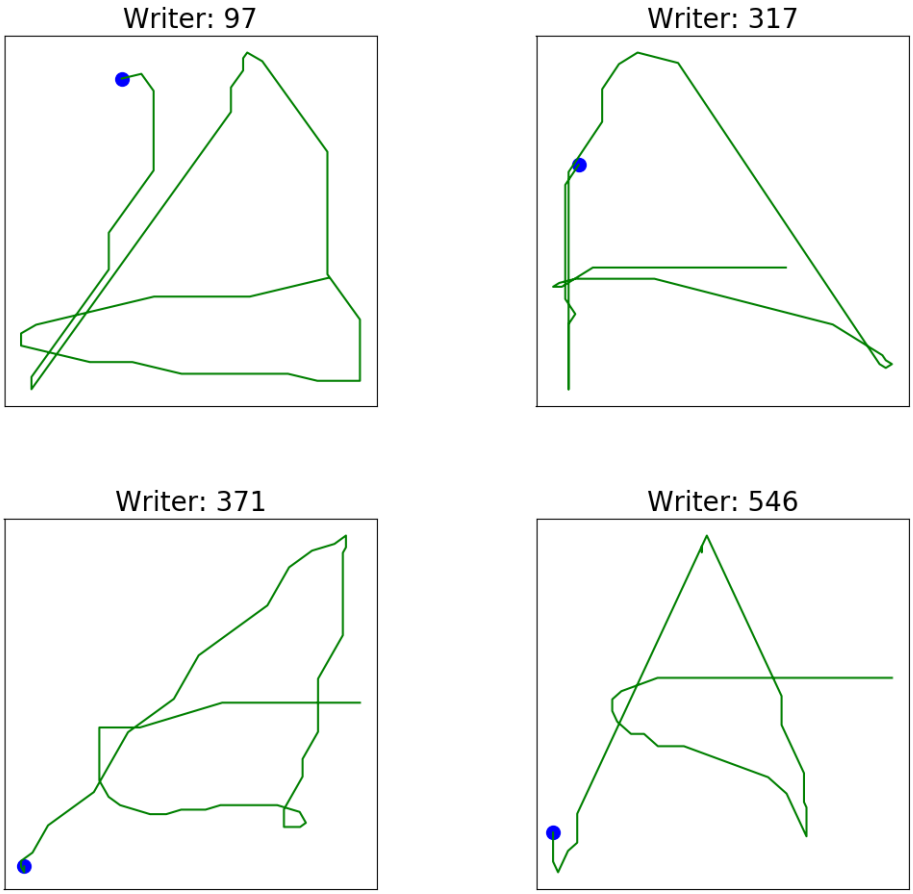



Figure 4.7: Examples for writing of letter A from the selected clusters. Starting point is marked with the blue mark. Each row is from one cluster. The first row show people who start drawing the letter from the top, going down, and then continue the drawing of the letter. The second row show people who start drawing from down directly.

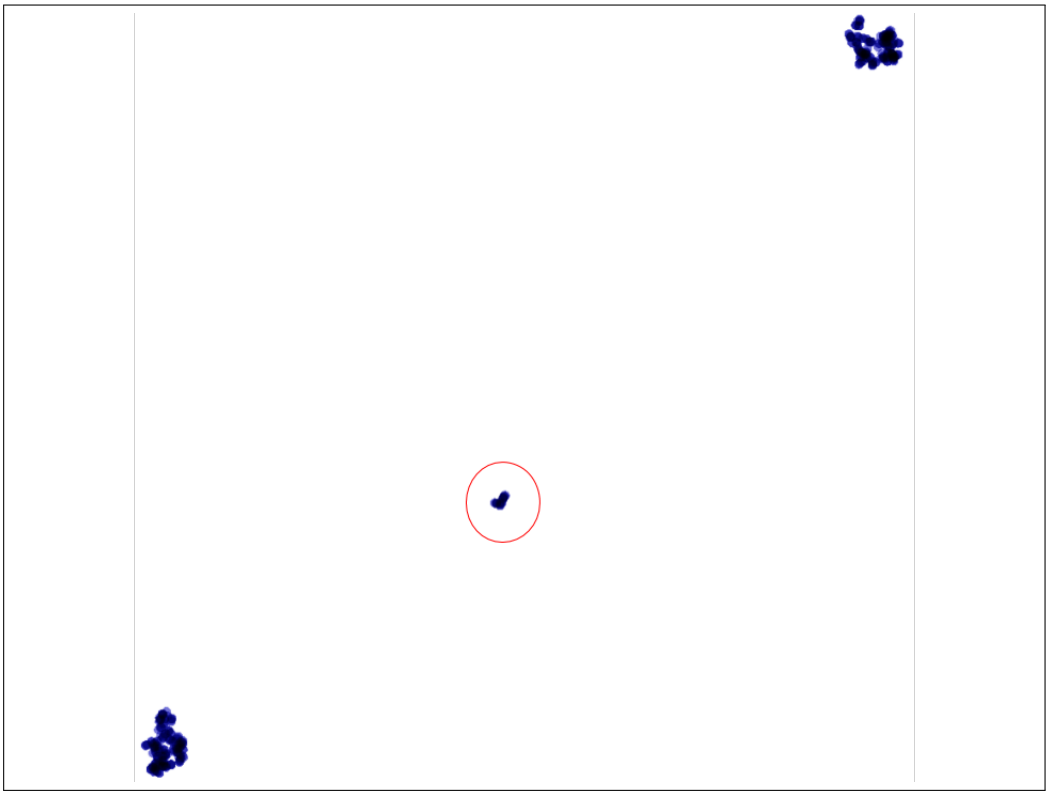


Figure 4.8: Projection for latent space for letter S using t-SNE. We manage to interpret the indicated cluster as the Edwardian style in drawing. The other two clusters (not indicated) did not show clear difference in the style, but this is an expected behavior from using the t-SNE algorithm, since it does not try to cluster styles as an objective.

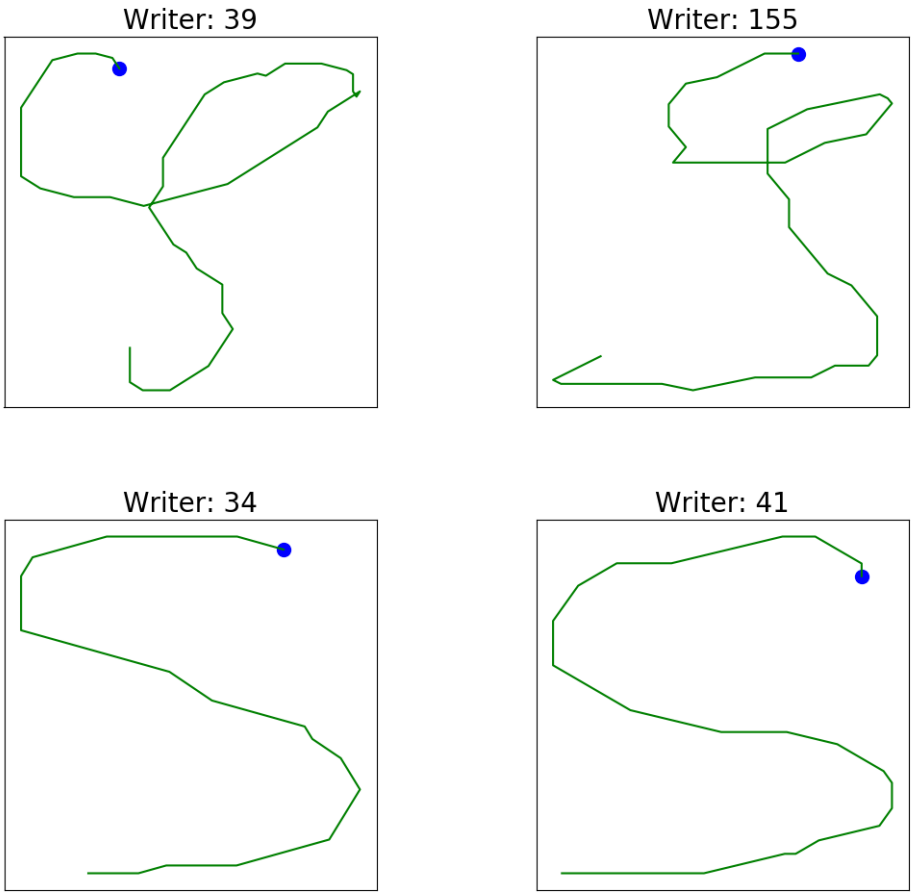



Figure 4.9: Examples for writing of letter S from the selected cluster (first row) versus the other two clusters (second row). Starting point is marked with the blue mark. The drawings from the selected cluster is always Edwardian style.


Figure 4.10: Examples of generated letters. The blue mark is the starting point. The traces in green is the ground truth, and the red is the generated ones by our model.

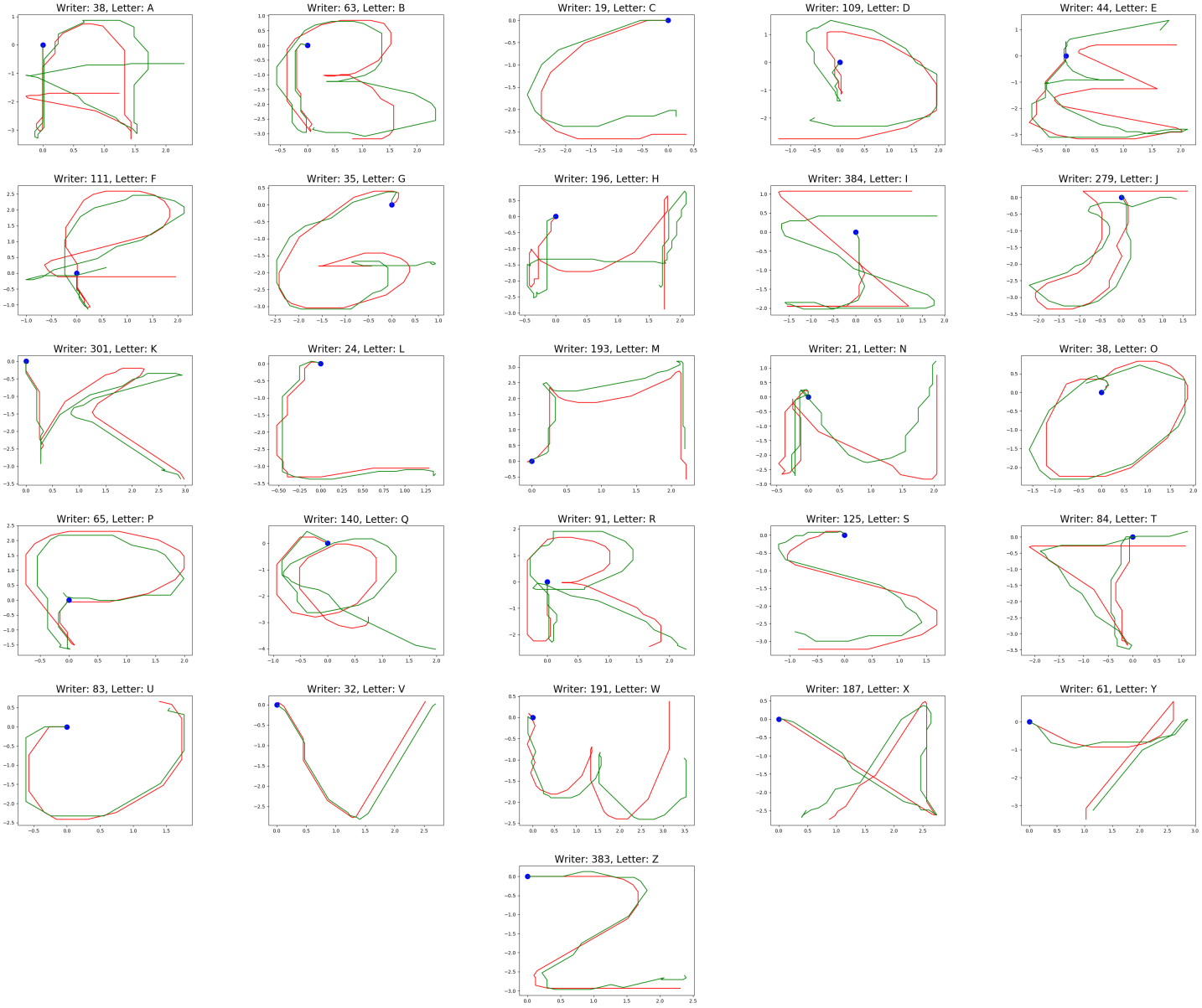



Figure 4.11: Examples of generated letters. The blue mark is the starting point. The traces in green is the ground truth, and the red is the generated ones by our model.

# Chapter 5

# Style Extraction and Transfer

## Contents



We finally arrive to the core objective of the PhD: how to leverage information of style from one (or more) tasks, in order to bootstrap the learning of a new task.

## 5.1 Introduction and objectives

- Motivate transfer learning in general (general talking)
- Why extracting style?: on our side, as ML practitioners, the ultimate objective of using ML is to help us extract new knowledge
- The questions of research here: extracting styles from bottleneck, transfer between writers, transfer between tasks

## 5.2 Transfer learning

An important research direction in machine learning nowadays is transfer learning. If humans and machines are able to learn how to perform a task, one of the thing that separates humans from machines is the ability to leverage this knowledge in order to acquire new skills and perform new tasks, without the need for additional trials and errors from tabula rasa. For example:

- If you know how to hold a glass cup, with little adjustments, you can learn how to hold a plastic bottle.
- If you know probability and algebra, it will accelerate your progress in machine learning.

This however, is not a straightforward thing for machine learning to do. The algorithms are fitted to data responding directly to the task required (i.e., has the same input feature space and same distribution). Thus, a change in the task can lead to degradation in the algorithm performance (??).

In the following subsections, I first introduced notations and clear description for the objective of transfer learning, and the metrics used to evaluate the quality of transfer. I then discuss different types of transfer learning, with examples on each type.

### 5.2.1 Notation and problem definition

BASIC NOTATION:

- A domain $\mathcal{D}$ is defined as $\mathcal{D} = \{\mathcal{X}, P(X)\}$, where:
  $\mathcal{X}$ is the feature space, $X$ is the learning sample, $X = \{x_1, x_2, \cdots, x_n\} \in \mathcal{X}$, $n$ is the size of the learning sample, $P(X)$ is the marginal distribution probability of the learning sample.
- For a given domain, a task $\mathcal{T}$ is defined as $\mathcal{T} = \{\mathcal{Y}, f(.)\}$, where:
  $\mathcal{Y}$ is the label space, $f(.)$ is the predictive function (takes $x_i$ and outputs $y_i$). It can also be rewritten as $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$.
- Source domain data set, $D_S = \{(x_{1S}, y_{1S}), \cdots, (x_{nS}, y_{nS})\} = \{X_S, Y_S\}$.
  Target domain data set, $D_T = \{(x_{1T}, y_{1T}), \cdots, (x_{nT}, y_{nT})\} = \{X_T, Y_T\}$.
  Source task $\mathcal{T}_s$, target task $\mathcal{T}_t$.

GB: be careful, it's not clean!

DEFINITION OF TRANSFER LEARNING: Given source domain data $D_S$, source task, $\mathcal{T}_S$, target domain $D_T$ and target task $\mathcal{T}_T$, we wish to improve the the prediction function of the target task $f_T(.)$ by using $D_S$ and $\mathcal{T}_S$. Conditions of transfer learning are:

- $D_S \neq D_T$, which means $\mathcal{X}_S \neq \mathcal{X}_T$ and/or $P(X_S) \neq P(X_T)$ (frequency feature bias).
  If $\mathcal{X}_S \neq \mathcal{X}_T$, the transfer learning problem is *Heterogeneous*. Otherwise, it is *Homogeneous*.
- $\mathcal{T}_S \neq \mathcal{T}_T$, which means $\mathcal{Y}_S \neq \mathcal{Y}_T$ and/or $P(Y_S|X_S) \neq P(Y_T|X_T)$ (context feature bias).

**Symmetric and Asymmetric transfer** *Symmetric* feature transformation attempts to discover underlying meaningful structure between domains, to find common latent features that unify (or at least reduce) the marginal distribution of the two domains. *Asymmetric* feature transformation attempts to transfer the features of the source space to make them more closely match the target space.

See figure 5.1.

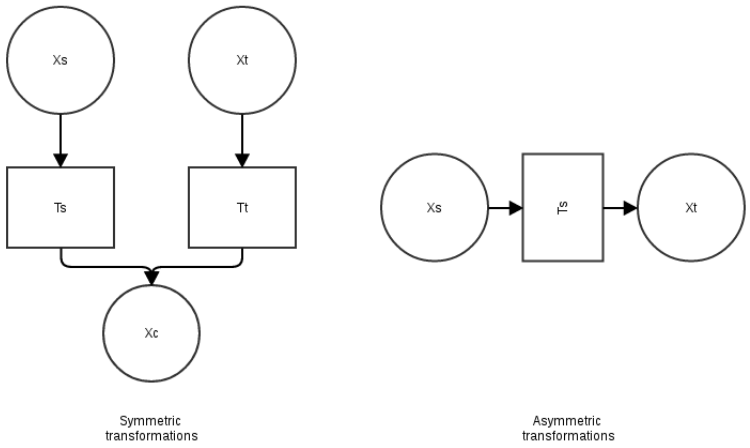



Figure 5.1: Illustration for the difference between Symmetric and Asymmetric feature transformation **REDO THE DIAGRAMS – NOT CLEAR**

### 5.2.2 Metrics to evaluate transfer learning

There is no standard way to evaluate transfer learning. Several metrics have been proposed in the literature (?), like (see also figure 5.2) GB: complete figure to have these 4 features:

1. Jump start: It is the difference in the initial performance between using transfer relative to learning without transfer.
2. Asymptotic performance: The difference in the learning performance through time/epochs between using transfer relative to learning without transfer.
3. Total Reward/Accuracy difference between end-performance with vs. without transfer learning.
4. Time-to-threshold: It the amount of time (or number of samples) needed to achieve a pre-specific performance level.

The transfer is considered as successful if metric 1 is greater than zero, metrics 2-3 increases with transfer or metric 4 reduces through transfer. It is

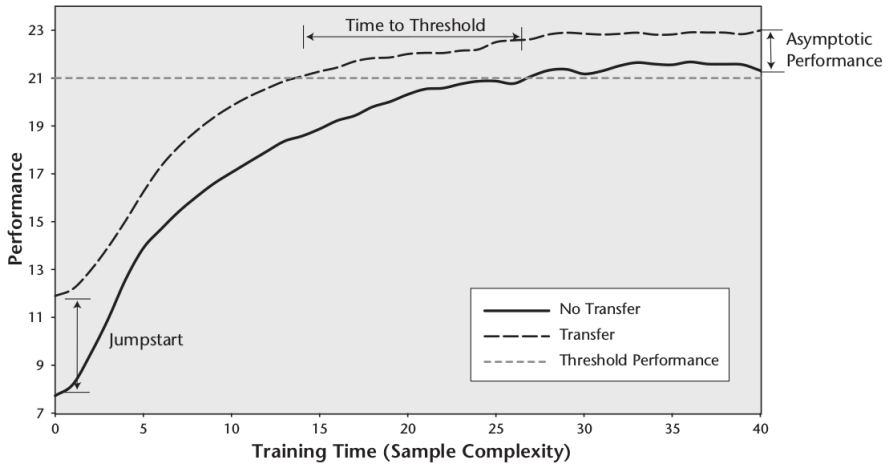



Figure 5.2: Different proposed metrics to measure transfer learning **Mention the source of this image**

worth mentioning that, in most state of the art, only metrics 3 and/or GB: 4?? are reported (which are the relevant metrics in my opinion for our work). I report all those metrics however, since I believe they give us a good diagnostics for our transfer learning system.

### 5.2.3 Homogeneous transfer learning

In this case, most of the research are focused on one of the 3 areas:

- Correct for source marginal distribution $P(X_S)$,
- Correct for the source conditional distribution $P(Y_S|X_S)$,
- Both.

#### Symmetric - transfer learning using deep learning

NEEDS TO BE UPDATED(?) discusses a deep learning approach for transfer learning, by using using stacked de-noising auto-encoders, to correct the marginal distribution between the source and the target domain, by learning

latent variables/features common between the two data sources in two steps:

- First, train an auto-encoder on the unlabeled data from the source and the target. This will produce latent variables.
- Use those latent features to train a classifier on the labeled source data.

Experiments are done on sentimental analysis, for 12 different sources and target domain pairs. The data used reviews for different products (4 different products). GB: Give performance

The idea of using deep learning (?) in order to achieve transfer learning has gain popularity during the last years, following the achievements in having better computational resources (?), and the availability of large benchmark datasets - most notably: ImageNet (?) for object detection, MS-COCO (?) for image captioning ... GB: Give general/key idea rapidly: pre-training of feature extractions performed by layers close to input, vs. training of high-level processing performed by layers close to output.

The first notable success of deep learning happened in the area of computer vision, with the AlexNet architecture (?). It was found out that such a deep network manages to extract generic features about the images: it learns simple, hierarchical filters, that are generic enough to be applicable for different datasets (see figure 5.3). This observation led to another surge in the usage of pretrained AlexNet - and later newer architectures, like VGG16 (?), Inception (?),etc - as feature extractors for new, unseen datasets.

It is interesting to note that those filters can be also seen as a representation for the skills we want to extract. In our case, we will need to combine both deep convolution network with a recurrent neural network (like in (???)).

#### Parameter-based transfer

(?) propose a Conditional probability based domain adaptation (CP-MDA): it corrects the difference between conditional probability distributions of multiple labeled source data, vs. limited amount of labeled target data. The process has

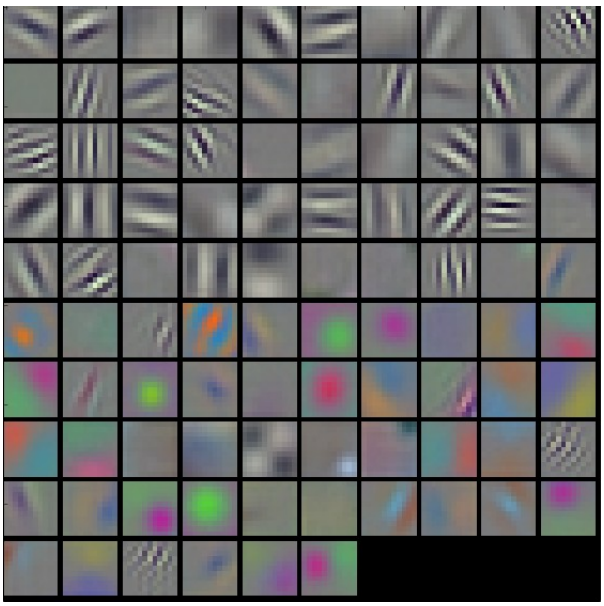


Figure 5.3: Visualization of the first convolution layer of a trained AlexNet. Note the basic shape of filters that resemble to Gabor filters widely used in image processing for decades (??). It easy to imagine that those same filters will be useful in other computer vision or image-related tasks.

five main steps:

1. Build a classifier for each of the source domains
2. Measure the closeness of each of those classifiers to the conditional distribution of the target domain.
3. Weight the classifiers according to their closeness
4. Use the weighted source classifiers to find pseudo labels for the unlabeled target data
5. Train a new classifier to map between the pseudo labels and the real labels on the targets data

Experiments are done on fatigue classification (surface electromyography data). Different source domain is for different people.

#### Instance-based transfer

(?) developed a weighting framework for multi-source domain adaptation (2SW-MDA). It tries to correct both the marginal and the conditional distributions between the source and the target domains in three steps:

1. Weight each source domain based on the marginal distribution difference between it and the target domain
2. Source domain weights are updated according to the (CP-MDA) approach
3. The target classifier is learned based on the re-weighted source classifiers and the labelled target data.

Experiments are done on fatigue classification (surface electromyography data). Different source domain is for different people.

### 5.2.4 Heterogeneous transfer learning

In this case, most of the research are focused on only one area: Aligning $\mathcal{X}_S$ with $\mathcal{X}_T$, assuming that $P(X_S) = P(X_T)$.

### Symmetric

(?) developed a method called *Heterogeneous Spectral Mapping* (HeMap). It is assumed that: $\mathcal{X}_S \neq \mathcal{X}_T$, $P(X_S) \neq P(X_T)$ and $\mathcal{Y}_S \neq \mathcal{Y}_T$. It also assumes the availability of labeled source data, and limited labeled target data.

- First is to find common latent variables between the source and the target domains, by using spectral mapping techniques. The spectral mapping objective is to maintain the original data structure, while minimizing the difference between the two domains (NOTE: Review the way it is formalized).
- Second, a cluster based sampling is performed to selected new training data (Thus, correcting the difference in marginal distribution).
- Third, correct the conditional distribution by Bayesian methods (NOTE: To be reviewed again, not clear to me).

Experiments are performed on image classification and drug efficacy prediction. (NOTE: The results are not clear, as they don't mention what is the baseline exactly, and no comparison to other state of the art methods is performed).

### Asymmetric

(?) discuss the application of transfer learning in software module defect problem. The source and the target software projects collect different performance metrics. The solution developed is called heterogeneous defect prediction (HDP). [1].

- First, perform feature selection on the source and on the target data, in reduce the dimensionality.
- Second, a statistical comparison (Kolmogorov-Smirnov test) is performed between the reduced source and target features, in order to determine the closeness of different features.
- Third, train a classifier using those close source features.
- Fourth, use this classifier on the target data, where the input will be

the corresponding target features (related to the source features from the previous step).

### 5.2.5 Negative transfer

A concern that arises in transfer learning is what is called *negative transfer*, where the knowledge learned from one task leads to no improvement on the target task, but reduces the quality of learning for that task. This can happen for multiple reasons, for example:

- The source task is not sufficiently related to the target task, thus, in the best case scenario, no useful knowledge can be transferred.
- The transfer method is not able to exploit the similarities between the source and the target task

Within the framework of *reinforcement learning*, some methods can estimate task similarity (??), though these methods do not provide any theoretical guarantees about their effectiveness. This, however, is an open area of investigation in the framework of *supervised learning* CAN'T FIND RESOURCES HERE. The main thing here for us to be careful during the choice/design of the source and the target experiments, to ensure that a possible positive transfer learning can happen.

## 5.3 Application of transfer learning

In the previous section, we explored the concept of transfer learning, and the different categories mentioned in the literature about this In the section, I will explain the work done during the PhD on transfer learning. There are two questions we addressed:

- Transfer between writers
- Transfer between tasks

### 5.3.1 Transfer between writers

### 5.3.2 Transfer between tasks

## 5.4 Style Extraction

## 5.5 Summary

# Notes

[1]This solution performs well on the assumption that the source and the target features are statistically close

# Part III

# Discussion and Closing Remarks

# Chapter 6

# Discussion

This chapter will be a free discussion about what i had done, shortcomings, difficulties in the PhD, and potential areas of development.

Science should always be about honesty, humility and respect, and not just flashy results and wide conclusions. Science can always make use of failures as well – something that is almost in the literature at the moment –. I will do my best in this chapter to highlight the other side of good results.

## 6.1 Challenges

### 6.1.1 Choice of the topic

### 6.1.2 Unclear state-of-the art

A major challenge during the PhD was to determine the relevant state of the art. For generative models, the usage of deep learning methods was not the clear choice of the beginning, and once chosen, it took considerable effort to determine the scope of the relevant state of the art.

The same goes for the state of the art on styles. The word itself, and the range of study, is very wide. By far, the work done in handwriting styles was the least relevant to our work; either because the work is done on offline handwriting (thus not dealing with the dynamics of writing itself) – and this is most of the work done currently –[1], or the work is done under strict assumptions **I think is 2nd part of the sentence is not accurate, and should be removed.**.

### 6.1.3 Lack of Benchmarks, evaluation metrics

Getting around these issues was quite a dilemma, for many reasons:

- I do not believe it is a healthy practise to pick up the benchmarks to use. This choice can be easily biased, and it could be argued that the benchmarks are chosen to be weak enough in order to show progress). In our case, it was mandatory to do so nonetheless, and we tried as much as possible to be fair in making these choices.
-

### 6.1.4 Deep learning: theory, hardware and software frameworks

## 6.2 Limitations of the current work

In this section, I discuss what I consider shortcomings for some of the methods used in this work.

### 6.2.1 Style extraction and exploration using PCA and tSNE methods

In this work, when exploring the latent space of our model, I used either PCA or tSNE projection methods (to project the latent space from the high dimensional

space into a smaller one), and tried to use the assumptions behind both methods to extract meaningful information from the latent space.

While this is legit, it really stretches these methods to the breaking point, plus, it hinders further investigation.

**PCA** assumes orthogonality and linearity in the space being projected. There is no reason however to assume that these assumptions hold for different styles. In the non-linearity aspect, the latent space does not have the clear objective of transferring non-linear style relationship into linear ones (simply, because no such objective can be formulated directly, since the problem of styles is ill-defined), unlike what can be noticed for the last layers of classifiers (where an embedded objective of the network is project the data from their non-linear manifold into a linear one). Founding orthogonality in the style space is an interesting aspect to explore, but this is a strong assumption, and there is no reason to believe that it holds for all aspects of styles. .

**tSNE** provide us with a way to deal with non-linearity, thus allowing another further exploring the latent space, but it is hard to repeat the results (the method is stochastic) and the projection does not always yield clear information about the styles. Changing the *perplexity* parameter leads to different results as well (I didn't explore the relation of that parameter to find a more suitable style manifold, and I am not sure if it is worth the effort).

But what is a good projection criteria in this case? should we let the organization of styles emerge on its own, by constraining the latent space and add regularization to the loss function (i.e., during an end-to-end training of the network)? should a second optimization step be performed on the latent space, in order to disentangle it? I discuss some of these ideas briefly in section 6.3.

### 6.2.2 Leak in the style module

In a basic autoencoder (no condition on the bottleneck), one can assume that encoder part will learn the identity of the task + the style in the same time. The idea of conditioning is to provide the task description (aka: task identity) as an input to the decoder (the condition), thus, relieving the encoder from learning it, and focus only on learning the styles, thus enhancing the style transfer capability.

Ideally, we expect that the output of the encoder has little to none information about the task identity. However, careful testing shows that this is not the case. There is a considerable leak of information about the task identity into the encoder.

PUT THE RESULTS OF OUR TESTING HERE. TRY IF POSSIBLE TO COMPARE WITH A NORMAL AUTOENCODER.

I don't have an explanation at the moment for the reason behind this phenomena. My intuition[2] is that on aspect of the problem lies in the task description. The assumption that a harsh one-hot encoding of the task is sufficient to describe the task correctly is flawed in my opinion.

An analogy for this can been drawn from clustering (hard clustering VS fuzzy clustering). Hard clustering, similar to one-hot encoding, does provide us with which this task is, but nothing about how this task relates to other tasks (i.e., proximity/similarity to other task), which is what fuzzy clustering do.

The influential work done by Geoffrey Hinton in ? – performed on the MNIST dataset ? – is a contributing factor in this intuition. I will not dive into details about this article here, since it is outside the scope of this work, I will just mention two interesting results from this study:

- In a classification task, the traditional description of the labels is one-hot encoding. However, using a soft/fuzzy description of the labels reveals much better results (makes sense, since it is more rich in information).
- If you train a classifier on the soft labels of digits 7 and 8 only, the classifier

will perform almost 90% accuracy on the other labels [3]!. It means that a better task description may increase the data efficiency of the model.

A similar concept should definitely be explored in the context of this work.

## 6.3 Future directions

- Embedding (robust generalization/maybe style extraction)
- Multi-stage optimization (for style extraction)
- Disentanglement of styles / latent space distributions / loss OR constraint (for knowledge extraction)
- RL and planning to reduce the complexity/have control over the generation and evaluation process
- Memories of the neural network for better task decomposition (useful for transfer learning) – maybe report my experiment with the external memory here.
- Statistical testing for neural network performance and for the inference quality + the effect of the random seed.
- Better task description
- Automatic separation of task and style
- Data efficiency and Model Complexity
- Define data augmentation protocol: while many techniques exists for images, it is not so clear in case of sequences.
- **Treating dynamical system as an image, with coloring gradients representing the dynamics of the system**: this will give us massive power in using the SOTA of generative models developed for static images (based on GANs), while addressing all the characteristics of the dynamical system in the same time.
  This is similar to treating sound waves as images (the ML course I did with Washington University), or spectrum images (like what i did with Marielle).
- Dealing with style extraction as an embedding problem: **NOTE: i am not**

**sure if we didn't do that already.**

- Modeling the error distribution of the model during the generation phase: in order to enhance the quality of the generation, and reduce the distance between the generation and the prediction (i.e., the training and the usage phases) [4].
- World models paper: having a model of the world, and use it to train a proper generator using RL.
- Re-visiting MDN (even the in discrete case): a useful tool, even for finite-discrete distribution. Mixed with good embedding, and giving 'uncertainty' level, provides important information. This also has a very good potential for optimization.
- Using RL for generation instead of log-likelihood models, and free the metric and the study from the constraints of differentiability.
- The hyper-parameter tuning: Do we need it? And how to do it efficiently? (yes we do need it. the question is how to do it in an effecient manner).
- Evaluate the quality of the content/task generated, not just the style: usually done via subjective testing or using another model – which i strictly refuse –. Subjective testing is too slow.
- What do these numbers actually mean in real life? how much should we care? ? (machine learning that matters paper): we used multiple metrics in order to compare/assess our models. But the main issue is how much this really matter? at how much $X\%$ difference in the performance that humans start to perceive the difference for example? This is very important in order to reach ML that actually matters in real life.
- The experimental protocol and scientific practices:

# Notes

[1]Most of the great advances that happened recently in neural networks – especially in generative models – is related to computer vision. Thus, it is more convenient to deal with with handwriting as images than as a dynamic process.

[2]I did not have the time to perform rigorous testing for this idea unfortunately.

[3]I personally find this particular result fascinating.

[4]nothing, quick test

# Chapter 7

# Closing Remarks

- Summarize the vision and the objectives of the PhD
-

## Notes

[1]Most of the great advances that happened recently in neural networks – especially in generative models – is related to computer vision. Thus, it is more convenient to deal with with handwriting as images than as a dynamic process.

[2]I did not have the time to perform rigorous testing for this idea unfortunately.

[3]I personally find this particular result fascinating.

[4]nothing, quick test

# Glossary

**task** In this manuscript, a task is used to describe executing set of actions, in order to achieve an objective. We will refer to the objective as the **content** of the task. To achieve an objective, usually there are multiple possible ways (multiple possible sets of actions) that allows us to achieve that objective. Each set of actions in this case is called a **style**.. 13

# Bibliography


Bishop, C. M. (1994). Mixture density networks. *Technical Report. Aston University, Birmingham*.

Chattopadhyay, R., Sun, Q., Fan, W., Davidson, I., Panchanathan, S., and Ye, J. (2012). Multisource domain adaptation and its application to early detection of fatigue. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(4):18.

Chollet, F. (2017). *Deep Learning with Python*. Manning Publications. `https://www.manning.com/books/deep-learning-with-python`.

De Sa, C. (2017). Non-convex optimization.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Denil, M., Bazzani, L., Larochelle, H., and de Freitas, N. (2012). Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184.

Fogel, I. and Sagi, D. (1989). Gabor filters as texture discriminator. *Biological cybernetics*, 61(2):103–113.

Freeman, H. (1961). On the encoding of arbitrary geometric configurations. *IRE Transactions on Electronic Computers*, 2:260–268.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.".

Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Google (2017). The quick, draw! dataset.

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

Ha, D. (2015). Mixture density networks with tensorflow. *blog.otoro.net*.

Ha, D. and Eck, D. (2017). A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer.

Jain, A. K. and Farrokhnia, F. (1991). Unsupervised texture segmentation using gabor filters. *Pattern recognition*, 24(12):1167–1186.

Jolliffe, I. (2011). Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer.

Jonas Jongejan, Henry Rowley, T. K. J. K. N. F.-G. w. f. a. G. C. L. and Team, D. A. (2017). The quick, draw! game.

Karpathy, A. and Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kostadinov, S. (2017). How recurrent neural networks work.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Larochelle, H. and Hinton, G. E. (2010). Learning to combine foveal glimpses with a third-order boltzmann machine. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1243–1251. Curran Associates, Inc.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2014). Microsoft COCO: Common Objects in Context. *ArXiv e-prints*.

Lowe, D. G. et al. (1999). Object recognition from local scale-invariant features.

Marti, U.-V. and Bunke, H. (1999). A full english sentence database for off-line handwriting recognition. In *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*, pages 705–708. IEEE.

Mohammed, O., Bailly, G., and Pellier, D. (2018). Handwriting styles: benchmarks and evaluation metrics. In *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 159–166. IEEE.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Nam, J. and Kim, S. (2015). Heterogeneous defect prediction. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 508–519, New York, NY, USA. ACM.

Narang, S. and Gupta, M. D. (2015). Speech feature extraction techniques: a review.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.

Ordóñez, F. J. and Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1).

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Pinheiro, P. and Collobert, R. (2014). Recurrent convolutional neural networks for scene labeling. In *International Conference on Machine Learning*, pages 82–90.

Raina, R., Madhavan, A., and Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880. ACM.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Séraphin-Thibon, L., Gerber, S., and Kandel, S. (2019). Analyzing variability in uppercase letter production in adults. In *Spelling and Writing Words*, pages 163–178. BRILL.

Shi, X., Liu, Q., Fan, W., Philip, S. Y., and Zhu, R. (2010). Transfer learning on heterogenous feature spaces via spectral transformation. In *2010 IEEE international conference on data mining*, pages 1049–1054. IEEE.

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014a). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014b). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Taylor, M. E., Jong, N. K., and Stone, P. (2008). Transferring instances for model-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer.

Taylor, M. E. and Stone, P. (2007). Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886. ACM.

Torrey, L. and Shavlik, J. (2010). Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global.

van der Maaten, L. and Hinton, G. (2008). Visualizing high-dimensional data using t-sne.

Viard-Gaudin, C., Lallican, P. M., Knerr, S., and Binter, P. (1999). The ireste on/off (ironoff) dual handwriting database. In *Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fifth International Conference on*, pages 455–458.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2014). Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3156–3164. IEEE.

Wagstaff, K. (2012). Machine learning that matters. *arXiv preprint arXiv:1206.4656*.

Wang, W., Xu, S., and Xu, B. (2016). Gating recurrent mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5520–5524. IEEE.

Wang, X., Takaki, S., and Yamagishi, J. (2017a). An autoregressive recurrent mixture density network for parametric speech synthesis. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4895–4899.

Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., et al. (2017b). Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*.

Weiss, K., Khoshgoftaar, T. M., and Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, 3(1):9.

Wikipedia (2019a). Convex funcation.

Wikipedia (2019b). Gradient descent.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*.

Zen, H. and Senior, A. (2014). Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 3844–3848. IEEE.