Developing *one-shot learning* models with greater scalability, using deep learning techniques

Som V. Tambe[1,*]

[1] *Department of Aerospace Engineering, Indian Institute of Technology, Kanpur, Uttar Pradesh, India-208 016*
[*] *Corresponding mail: {somvt}@iitk.ac.in*

Compiled July 14, 2020

This study has been conducted as a part of the Brain and Cognitive Society's Summer Project oppurtunity, and mentored by Shashi Kant (Y16,EE). We have aimed to create efficient and scalable one-shot learning models, using neural networks.

https://github.com/SomTambe/omniglot-bcs

INTRODUCTION

The first thing that strikes our mind, or what is popularly taught in various courses around the world, when we see neural networks, that they are analogous to neurons in our brains. This analogy is not completely true. Generally, deep neural networks, or any machine learning model, requires large datasets to generalize trends/patterns across a wide variety of samples.

Humans, when given a set of completely new objects, can classify the similar objects given a reference object, without requiring hundreds-of-thousands of examples. Past efforts to counter these problems include the Bayesian Program Learning [1].

The BPL paper hand-engineered various parts of the model to serve to the similarity and generation tasks with the Omniglot dataset [2]. We aim to create much more generalized models, which can serve for a wide variety of datasets, and tend to be less hand-engineered towards a particular target.

MODELS

The articles below enlist all the study and implementation we have done yet. All references have been mentioned at the end of the document, and mentioned wherever used.

1. ENHANCED LEARNING BASED ON STROKE DATA

Preface

The implementation for this model has been carried on by Som Tambe. The study is not yet complete yet, partly due to time constraints and due to unexpected results.

Approach

For this model, I use the Omniglot Dataset [2]. The dataset contains 964 different characters, with 20 examples per class. This accounts to much lesser than what standard datasets have, for instance MNIST [3] has about 6000 examples per class. Ergo, I have chosen the dataset (also because of the fact it has become a standard for benchmarking one-shot learning models) .

Algorithms used

The algorithms have been described in the table below-

Algorithm 1. Conversion of stroke data to 25-splines

```
1: procedure SPLINE($d$)                ▷ where $d$ is a single part stroke data
2:     while no error do
3:         if len($d$)$\geqslant$ 5 then
4:             return spline($d, 3$)         ▷ spline with degree=3
5:         else
6:             if $1 <$ len($d$)$\leqslant$ 5 then
7:                 return spline($d, 1$)     ▷ degree=1
8:             if len($d$)=1 then
9:                 return repetition($d$)    ▷ 25 times
10:    while throws error do
11:        $d \leftarrow d + \mathcal{N}(0,1)$              ▷ Add some noise
12:        procedure SPLINE($d$)
```

An example spline is shown in Fig. 1.

This helps us make all the strokes to a constant length of 25. I use the splprep module from scipy.interpolate, which allows me to interpolate it into parametric curve (since strokes are not perfect functions, therefore we use parametric representation).

I had also been suggested the use of Ramer-Douglas-Peucker [5] (**RDP**) algorithm, which helps decimate a curve to a fewer point representation, by varying the value of epsilon iteratively. We decided not to use this after comparing the results of scipy.interpolate and RDP .

Errors were observed when completely vertical lines were sub-parts of strokes. I thus add some extra gaussian noise in each coordinate $\sim \mathcal{N}(0,1)$.

Next thing we do is contruct a Variational Autoencoder [4] . We also use the reparameterization trick mentioned in [4] . This

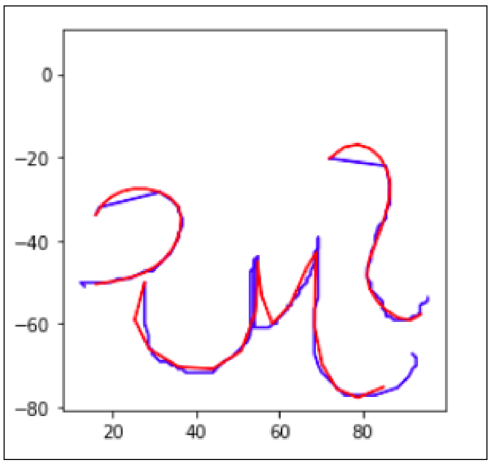


Fig. 1. In blue: Real stroke. In red: 25-length stroke obtained from constructing a spline.

is to construct a 2-dimensional latent space visualization of how the strokes are distributed in space.

Another reason we use a VAE is that there exist only $\sim$ 48000 strokes in total. Therefore using a VAE to an ordinary autoencoder gives us more stochasticity in terms of the generated latent vector, therefore making training better, and faster.

Algorithm 2. Using a VAE to find the 2-d latent space representation of strokes

```
   procedure ENCODE($r$)
2:     $d \leftarrow (d - \mu_d)/\sigma_d$                ▷ Normalizing the spline
       $d \leftarrow d.view(50)$                          ▷ Flatten the coordinates
4:     $\mu_z, \sigma_z \leftarrow encoder(d)$            ▷ Mean, variance of latent vector
       $z \leftarrow \mathcal{N}(\mu_z, \sigma_z)$        ▷ Sample from a normal distribution
6:     $d' \leftarrow decoder(z)$                         ▷ Generated coordinates
       Calculate loss                                     ▷ KLD & recombination
8:     Backpropogate derivatives
       Update network
```

After training this VAE network, we first observe the decoder outputs. Fig 2 shows the results.

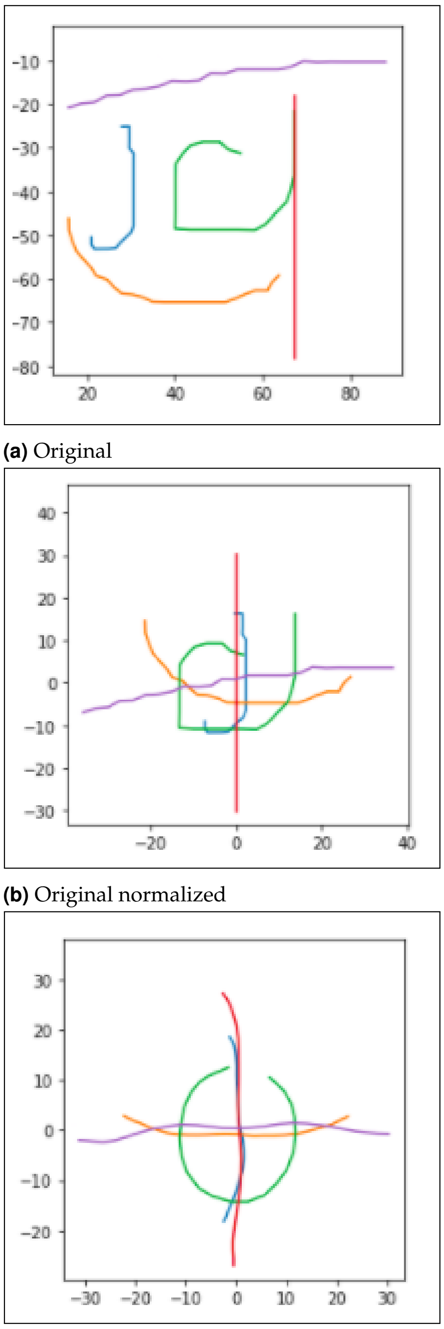
These results have been obtained after training for 150 epochs. The loss had started converging well before 150 epochs, and after randomly testing characters, we finally concluded the training was complete.

We discard the decoder network after training (although I store the weights). The encoder gives us all that is required. We now proceed to the further part of the model.

We decide to plot the 2-dimensional representation of the obtained stroke representation. Here is what we got, shown in Fig 3 .

By plotting the 2 dimensional plot, I aimed to get noticeable clusters which would help me identify how many different categories of parts exist. This would help me prepare one hot vectors of characters, given their stroke data, which would denote the parts present in the characters.

By performing supervised learning on the characters and their respective one-hot vectors produced by us by above clustering methods using another convolutional neural network,



(a) Original

(b) Original normalized

(c) Decoder output

Fig. 2. VAE results. 2b was fed to the VAE network, 2c was the output. Notice how similar do the normalized input and the output look. This is a first. Also notice, strokes are denoted with different colours.

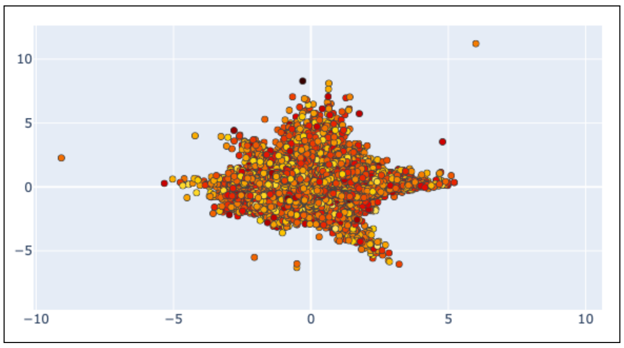


Fig. 3. 2-dimensional representation of latent space.

we can obtain a network which would help us retrieve stroke information given any new character, since any new character would contain some similar parts in them.

I encountered a problem in clustering. As visible in Fig 3 , there are no noticeable clusters. On further zooming in, you can find out there do exist clusters, but they do not have discrete boundaries, rather they share their boundaries with other clusters, indicating similarity between strokes.

To check if clusters are formed in higher dimensional representations, I trained the VAE network with latent dimensions 3, which would give us a 3 dimensional representation of stroke data. Fig 4 shows us our 3d plot in a blue colour.

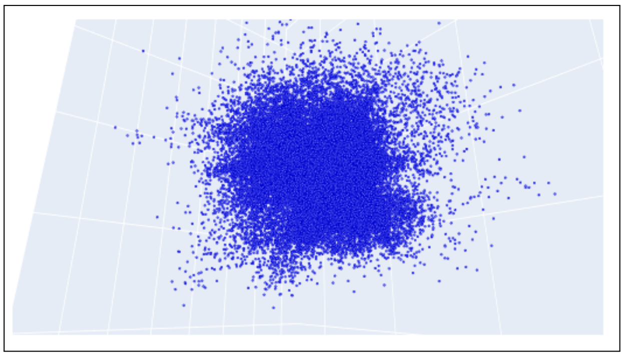


Fig. 4. 3-dimensional representation of latent space.

When you zoom in and rotate the 3d plot, there a few more clusters visible compared to the 2d plot. Though this does not help us in our task, we have atleast been successful in observing few clusters, which means we are still on the right track. Great news for us (still not that great though)!

Now we proceed with clustering, using the 2d representation. We use K-Means clustering here, and the results of clustering are displayed in Fig 5 .

I make one-hot encoded vectors for each of the characters in the dataset. I then try to train the CNN. The network does not train properly, even after the losses converging. The reasons for this have been studied upon and mentioned in the future scope of the project.

Technical Stuff

I have consistently used PyTorch [6] for all the deep learning related part of the project. I have mentioned the use of the SciPy library before for the use of spline creation and interpolation of data. Matplotlib was used along with Plotly for plotting

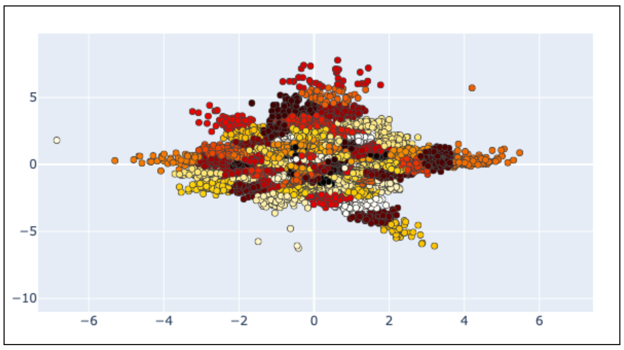


Fig. 5. 2-dimensional clusters.

of graphs. Numpy was used very frequently for handling of tensors on base levels.

Stroke data was obtained from the authors repository of the Omniglot dataset in a .mat file. We converted that to usable format using the scipy.io module.

The Variational Autoencoder model was made of only Linear layers (vanilla flavor). Encoder consisted of $[50, 1000, 1000, 2 * dimension]$ units layerwise, decoder mirrored the encoder. We used the reparametrization trick [4] to generate the latent vector.

PyTorch supports two types of dataloaders - Map-style and Iterable form. Since data was in dictionary format here, mainly non-serial order, I use the torch.utils.data.IterableDataset base class for creating custom datasets.

For the lastmost network, as I have mentioned earlier, I use a CNN (plain, nothing residual) . The convolutional block is described below, with a ReLU activation between each layer. There are three Conv2d layers, each having feature maps $[128, 128, 128]$, with stride 1 and 3×3 kernels. Then I use a Maxpooling layer with stride 2 and kernel size 2×2. Fully connected layers follow with these many units: $[3000, 300]$. I use the Adam optimizer [8] with a learning rate of 10^{-3}.

Future Scope

We realize now the main problem lies in clustering. We hypothesised that characters of the same class should have had same one-hot vectors. This was supposedly proven false, and we observed variation across one-hot vectors across different instances of the same character (after we clustered into 300 strokes). This caused unsuccessful training of the last CNN we were supposed to train.

Now how do we rectify this problem?
There exist variations in VAE models which define much better latent space clusters. [9] introduces a discriminator network which matches the aggregated posterior of the hidden variable to an arbitrary prior distribution. This ensures that generating samples from any part of the pre-defined distribution would ensure results.

But we think the problem does not lie here. The problem lies in the fact that we were taking strokes as a whole. There exist sub-part in strokes too, which act like primitives. These primitives (for example a left-half curve, a simple vertical line) form the larger strokes. Therefore, using these primitives for clustering would ensure better results, as there may be a very large number of strokes which exist (combinations of primitives). This has also been observed by [1], who constructed their model on the basis of these primitives.

Another improvement we can make is making the use of heat-maps, as popularly used by posenet [10]. But the main problem we would encounter in heat-maps would be the large number of primitives that exist, which would make it computationally expensive.

After obtaining correct clusters, and therefore one-hot vectors for characters, we would expect our CNN to train successfully.

We can then use this CNN for similarity tasks, as well as classifcation tasks.

The latest model repo can be found here - https://github.com/SomTambe/omniglot-bcs

REFERENCES

1.] *Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015).* Human-level concept learning through probabilistic program induction. Science, 350(6266), 1332-1338.

2.] *Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2019).* The Omniglot Challenge: A 3-Year Progress Report. Preprint available on arXiv:1902.03477

3.] *LeCun, Yann, et al.* (1999): The MNIST Dataset Of Handwritten Digits.

4.] *Diederik P. Kingma and Max Welling* (2019), "An Introduction to Variational Autoencoders", Foundations and TrendsR in Machine Learning: Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

5.] *Urs Ramer*, "An iterative procedure for the polygonal approximation of plane curves", Computer Graphics and Image Processing, 1(3), 244–256 (1972) doi:10.1016/S0146-664X(72)80017-0

6.] *Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Soumith Chintala*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library ", NeurIPS 2019, arXiv:1912.01703

7.] *Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors.* (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, in press.

8.]*DP Kingma, JA Ba*, A method for stochastic optimization - arXiv:1412.6980

9.] *A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey.* Adversarial autoencoders. arXiv preprint arXiv:1511.05644, 2015.

10.] *Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, Yaser Sheikh*, OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields - arXiv:1812.08008