Disentangled Variational Autoencoders applied to Fuzzing Corpus Minimisation

Aidan Sawers
College of Engineering and Computer Science
Australian National University

Abstract

Fuzzing is an expensive and time consuming process. In order to fuzz more efficiently a small, diverse corpus is ideal. By using clusters of seeds prone to crash, or a broad distribution of diverse initial seeds, fuzzing is more effective. To find a better initial seed corpus autoencoders can be used. By using more nuanced techniques like disentangled variational autoencoders, the latent space of a corpus can be analysed and key features can be extracted to find clusters of crashing seeds, or find a small set of diverse seeds. Using either a cluster of crashing seeds or a very diverse set, finding more crashes, quicker is more likely.

Keywords: Variational Autoencoder, Autoencoders, Fuzzing, American Fuzzy Lop, PCA, Corpus minimisation

1. Introduction

1.1 Fuzzing, American Fuzzy Lop and Poppler

Fuzzing is the act of applying random inputs to a computer program to find errors which cause the program to behave unexpectedly. American Fuzzy Lop (AFL) by Michal Zalewski is a nuanced version of this process where a program is instrumented by using a compiler which injects extra commands between blocks of code. A block of code is simply a linear set of instructions with no conditional elements. These commands write to a 256 x 256 bitmap which represents transitions between code block executions. By using these execution path bitmaps AFL is able to determine if a semi-randomised input generates more path execution. The way AFL generates inputs from an initial corpus of seeds is by randomly mutating them in line with methods used by evolutionary algorithms as well as using more interesting methods like adding and subtracting to words in both little and big endian. Using this method, AFL creates new seeds to test and mutate.

Poppler is a package of pdf related utilities which are used by many other programs including LibreOffice. The specific program fuzzed in the experimental section is pdftotext, which converts a pdf into raw text. The version used is Poppler-0.64.0 because there are more known bugs than newer versions which can be used as comparison.

1.2 Principal Component Analysis

Principal component analysis (PCA) is a method of determining the components of a dataset which contribute to maximum variance. There are two variants of PCA used in the experiment, both for comparison with the neural networks. One is linear PCA and the other is a kernel PCA with a radial bias function. These are used to look for both linear and non-linear patterns in the latent space.

1.3 Disentangled Variational Autoencoders

Autoencoders are a type of neural network where the input and output layers are the same size, but the hidden layers form a bottleneck, being significantly smaller than the input and output layers. A variational autoencoder (VAE) is a modification on this. It has a similar structure to a regular autoencoder but has two layers that feed into the bottleneck instead of just one; the mean vector and the standard deviation vector (Kingma D. and Welling M. 2014). VAEs also implement the loss function shown in Figure 1, which takes into account the Kullback-Liebler (KL) divergence of the data (Higgins I. et al. 2017). This KL Divergence is effectively the error between the optimal encoding of the data and the actual encoding of the data. By minimising the loss of the VAE using this nuanced loss function, we are working towards the optimal encoding of the input in the bottleneck space, which is the goal of an auto encoder.

$$\boldsymbol{li}(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbf{E}\boldsymbol{z} \sim \boldsymbol{q\theta}(\boldsymbol{z} \mid \boldsymbol{xi})[\log \boldsymbol{p\phi}(\boldsymbol{xi} \mid \boldsymbol{z})] + \mathbf{KL}(\boldsymbol{q\theta}(\boldsymbol{z} \mid \boldsymbol{xi}) \mid\mid \boldsymbol{p}(\boldsymbol{z}))$$

Figure 1: *loss function for datapoint xi in a Variational Auto Encoder, first term being reconstruction loss and second term the Kullback-Leibler divergence. (Kingma D. and Welling M. 2014).*

A Disentangled Variational Autoencoder (DVAE) extends a VAE by using a β parameter in the loss function applied to the KL divergence (Barfoot T. and D'Eleuterio G. 2020) shown in Figure 2.

$$\boldsymbol{li}(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbf{E}\boldsymbol{z} \sim \boldsymbol{q\theta}(\boldsymbol{z} \mid \boldsymbol{xi})[\log \boldsymbol{p\phi}(\boldsymbol{xi} \mid \boldsymbol{z})] + \beta\, \mathbf{KL}(\boldsymbol{q\theta}(\boldsymbol{z} \mid \boldsymbol{xi}) \mid\mid \boldsymbol{p}(\boldsymbol{z}))$$

Figure 2: *loss function for datapoint xi in a disentangled variational autoencoder, same as above but with the β scaler for KL divergence. (Higgins I. et al 2017)*

This β parameter pushes for a higher priority on the efficiency of the encoding. Higgins et al. found that by adding this β parameter to a VAE, the latent space encoding was more nuanced and resulted in each bottleneck neuron more clearly representing one latent variable (Higgins I. et al 2017). When applied to the celebA data set (Liu Z. et al 2016), a VAE does encode rotation of the face as one of the variables, but creates glasses or gender changes as artefacts when modifying the variable. When using a DVAE instead, with β as 250, the rotation is also encoded, but more effectively and with minimal erroneous changes to the artefacts. This can be seen in Figure 3.

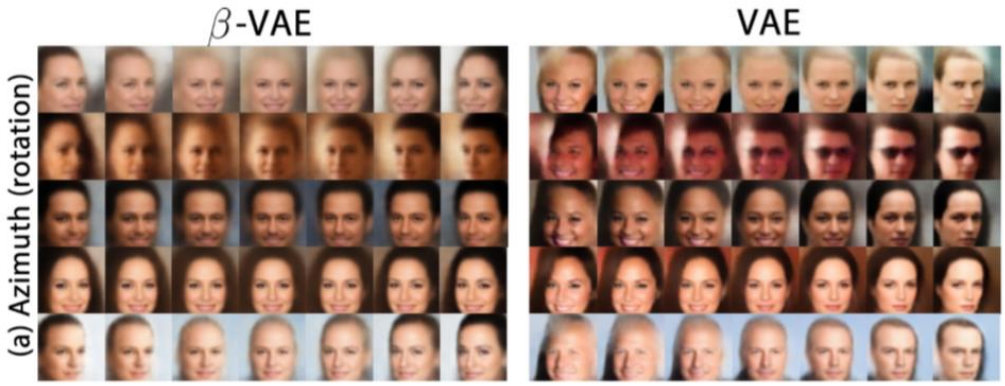

Figure 3: *modifying the azimuth latent variable between -3 and 3 on the celebA dataset (Higgins I. et al. 2017).*

1.4 Dataset

The dataset used to test the effectiveness of these methods is generated from a large corpus of almost 100,000 seeds, which are pdf files. The afl-cmin program is used to minimise the set into 1,354 files. The fuzzing process runs more efficiently on a small amount of small files. What afl-cmin does is reduce the initial corpus of files into the smallest set of files, in terms of size, which covers all of the code block transitions. Transition bitmaps were generated for all 1,354 files and noted with size and seed name. This set of files was then fuzzed for 24 hours across eight parallel workers; one master node which goes through all mutations heuristically and seven which run in chaos mode, randomly deciding on mutations. The original parent seeds of mutated seeds which caused crashes were determined and recorded. The final results used for the dimensionality reduction research were a set of 1,354 rows. Each row has seed name, file size, if it was the parent of a crash or not and the 256x256 transition bitmap unrolled into 65,536 features for the code block transition counts.

1.5 Hypothesis

If there is any relationship in latent space between seeds which crash in fuzzing, there will be a cluster in the relative encoded space which contains a majority of encoded crashing seeds. By comparing the PCA, AutoEncoder, VAE and DVAE methods, if this relation does exist, it will most likely be picked up by the DVAE method because of its prioritisation of encoding efficiency.

2. Method

2.1 Experiment Structure

For each structure the dataset was split into an 80% train and 20% test set and then normalised using min max normalisation Figure 4. For each trial, Adam optimisation was used with an L2 regression rate of 0.001 (Kingma D. and Ba J. 2015) and the network was trained for 20 epochs. For the AutoEncoders, once training was completed, all 1,354 rows were encoded using the trained network. Next the columns with the highest variance were taken and mapped against each other into

x, y pairs for a scatter plot. For PCA, the two most variant columns were converted to pairs and scattered. The normal AutoEncoder is run once, the PCA is run as both linear and RBF and the DVAE is run with a β of value with 1, 100, 250, 500, where 1 is a normal VAE. Once the scatter plots are generated, the parent seeds which lead to a crash are highlighted to analyse for clustering.

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Figure 4: *The min max normalisation function*

2.2 Autoencoder Structure

All encoders follow the same structure of five layers, taking in and returning the unwrapped bitmap of 65,536. The input and output layers have 65,536 neurons, intermediate layers have 4,000 neurons and the bottleneck has 15 neurons, with the VAE encoders having two layers of 15.

3. Results and Discussion

3.1 Training Analysis

After training each of the networks for 20 epochs, the autoencoder settled around 0.01 using MSELoss. The VAE settled around 6,000 using BCE + KL divergence. Shown in Table 1, we can see that the smallest loss was for a β value of 100 and the highest was for 500.

Table 1: β value with final test loss after 20 epoch

β value	*Final Test Loss (3 d.p.)*
100	4974.815
250	5252.135
500	5134.066

This demonstrates the benefit to experimenting with β hyperparameters suggested by Higgins et al. (Higgins I. et al. 2017). For this experiment, the better results came from the smaller β value as opposed to that in the original DVAE paper which found a β value of 250 worked the best. This suggests that while there is some value in trying to find relationships in this latent space, but at β value 250 the balance of prioritised KL divergence is not as effective (Figure 5).

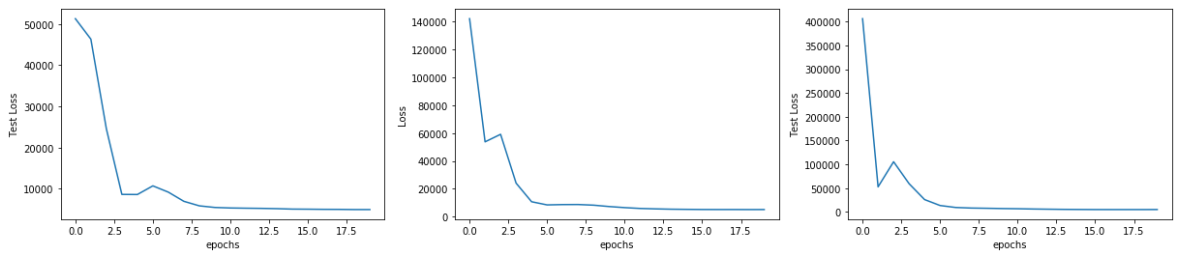

Figure 5: *from left to right test loss for VAE with β value 100, 250, 500*

3.2 Crash Clustering

To determine if the found latent space is valuable for finding crashing seeds, the set of 1,354 pdf seeds is run through all of the autoencoders and then the most variant features are compared against each other as x,y pairs on a scatter plot. The crashing seeds are highlighted in yellow to determine if there are any patterns. These are compared against the PCA graphs for contrast. In Figures 6 and 7 there are the Linear and Kernel RBF references. Here there are no real patterns since it seems to be evenly distributed for the most part. There is a small cluster around -15, 10 which contains no crashing seeds, which could be removed from the set.

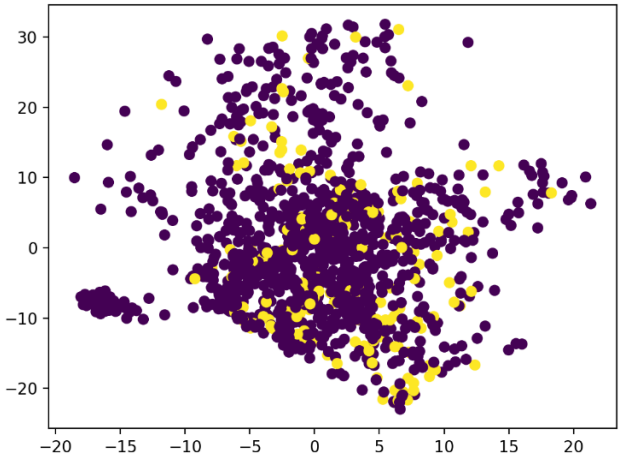

Figure 6: *The minimised corpus run through PCA, using the two features with highest explained variance.*

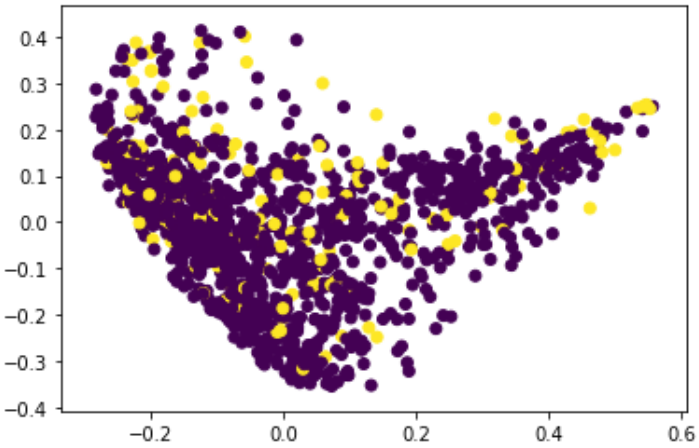

Figure 7: *The minimised corpus run through Kernel PCA RBF with yellow as the crashing seeds. The x-axis is the highest variance component and the y the second highset.*

Looking at the results of the autoencoders, we see similar results (Figure 8). The crashes are mostly evenly distributed, which means that there are no major clusters of crashing seeds to sample future crashes from.

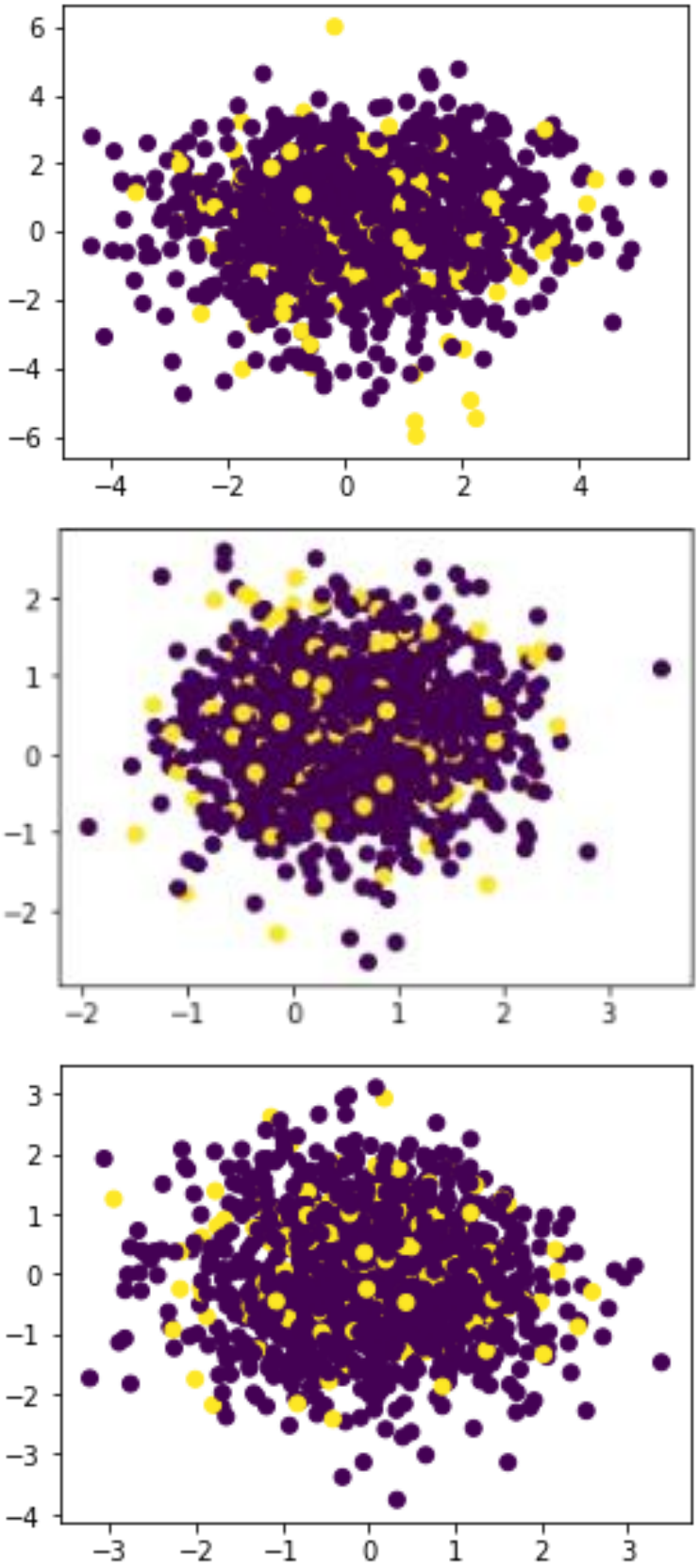

Figure 8: *comparison of the most variant component, from top to bottom: DVAE- β 100, DVAE- β 250, DVAE- β 500.*

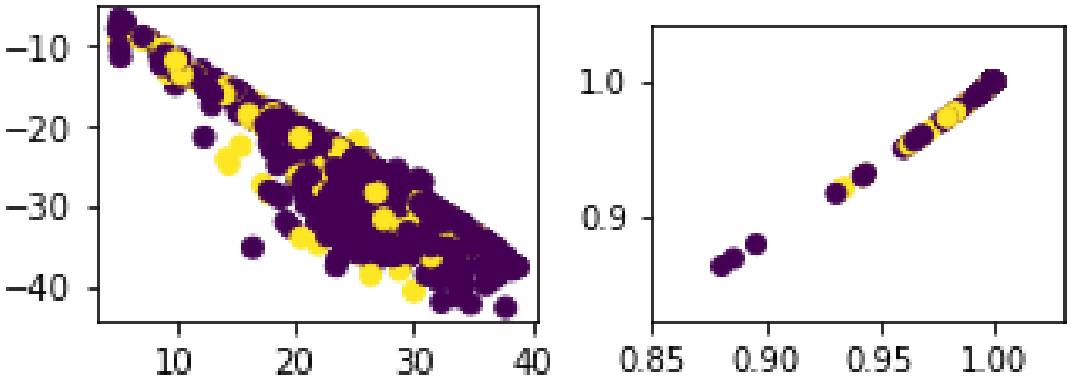

Figure 9: *left side, VAE most variant components and on the right a normal AutoEncoder*

It appears that in the VAE and Auto encoder graphs (Figure 9), there is again a small cluster towards the bottom of the graph which could be removed to make the process more efficient.

3.3 Fuzzing results

Although there is no real clustering of crashes, this does not mean that the results are not useable. We can make the fuzzing run more efficiently using the distribution. A small set of very diverse seeds may produce as good results as a larger less diverse corpus because the larger corpus has more overlap in latent space and therefore theoretically less diverse code block execution. To test this theory, a set of 100 diverse seeds were chosen from the DVAE with a β value of 100 as it had the most loss. The way these seeds were selected is by finding the two most distant seeds in the set after encoding all of the seeds with the DVAE, then the seed most distant from the selected seeds was iteratively chosen until there was 100 selected seeds. As a control, 100 seeds were also chosen randomly and also fuzzed.

After fuzzing both sets for 24 hours there were no crashes found, which is not uncommon, but the total unique paths found can be used as a metric for how effective the sets were. The random set found 2,166 unique execution paths and encoded diverse set found 3,839 paths. This is 23.07% and 24.24% bitmap coverage respectively (Figure 10). This shows that using distinct seeds is more effective at fuzzing a program and that a DVAE can be used effectively to determine this distinctiveness.

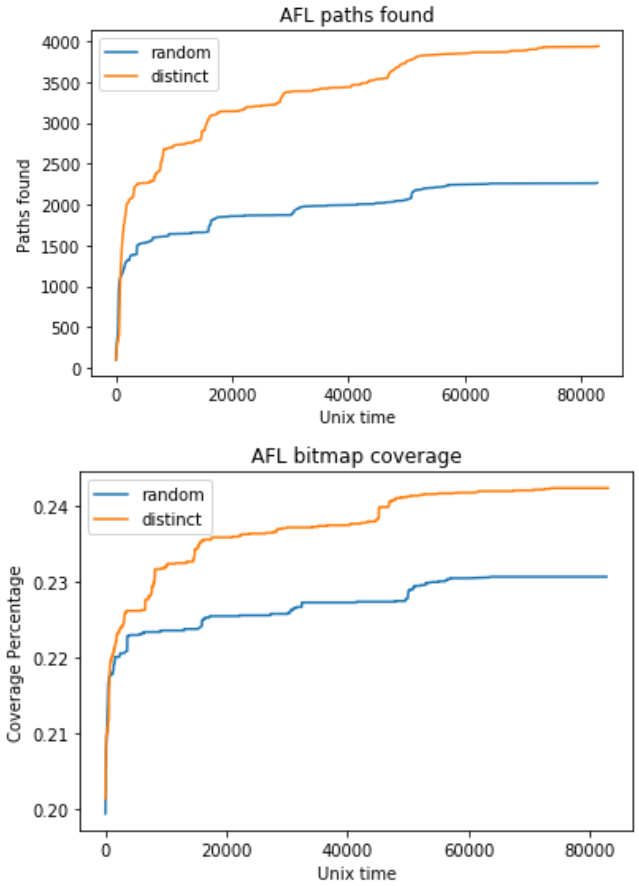

Figure 10: *Total paths found and bitmap coverage for random and distinct sets*

4. Conclusion

Using dimensionality reduction techniques such as PCA or looking for relationships in latent space with Autoencoders does not produce clear clustering of crashing results of the pdftotext program using AFL. However, the distribution of seeds using this technique can be used to create a diverse set of seeds to marginally increase the performance of fuzzing in general. This may not be the case for every program which is fuzzed by AFL, so there is potential for this technique to be used against other programs to determine if there is clustering of crashes based on a latent variable. The results also show that Autoencoders can potentially be used effectively to optimise AFL fuzzing runs.

5. Future Work

While the initial results are promising, more thorough testing would be beneficial. Running the final fuzzing trial multiple times would reinforce the effectiveness of using an engineered distinct set. Running the various autoencoders on different sets of seeds with different target programs may also yield different results when it comes to crashing seed clustering. For example, on a program where files which execute an obscure code block always crash, there may be better clustering. More epochs in training and more parallelised fuzzing may have generated better results.

6. References

Altosaar, J. (2017). What is a variational autoencoder? [online] Jaan Altosaar. Available at: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/.

Barfoot, T.D. and D'Eleuterio, G.M.T. (2020). Variational Inference as Iterative Projection in a Bayesian Hilbert Space. arXiv:2005.07275 [cs, stat]. [online] Available at: https://arxiv.org/abs/2005.07275.

Bengio, Y., Courville, A. and Vincent, P. (2014). Representation Learning: A Review and New Perspectives. arXiv:1206.5538 [cs]. [online] Available at: https://arxiv.org/abs/1206.5538.

Dubios, Y. (2019). disentangling-vae. [online] GitHub. Available at: https://github.com/YannDubs/disentangling-vae/blob/master/main.py.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S. and Lerchner, A. (2017). β-VAE: Learning Basic Visual Concepts With a Constrained Variational Framework. In: 5th International Conference on Learning Representations. ICLR 2017. Google Deepmind.

Kingma, D.P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs]. [online] Available at: https://arxiv.org/abs/1412.6980.

Kingma, D.P. and Welling, M. (2014). Auto-Encoding Variational Bayes. arXiv:1412.6980 [cs]. [online] Available at: https://arxiv.org/abs/1312.6114.

Liu, Z., Luo, P., Wang, X. and Tang, X. (2015). Deep Learning Face Attributes in the Wild. In: Proceedings of International Conference on Computer Vision (ICCV).

Zalewski, M. (2019). AFL. [online] GitHub. Available at: https://github.com/google/AFL/blob/master/docs/technical_details.txt [Accessed 28 May 2020].