AI and Deep Leaning Coursework

Scott Phillips

June 8, 2025

1 Question 1

This paper addresses the generative modelling method Gaussianization, in particular the exploration of the scaling behaviour of Gaussianization with dimensions, both analytically and empirically. Finding analytical measures of these constructs is an active and important area of research, as it offers insight into the efficiency and feasibility of applying such methods to high-dimensional data. Previous work in this area started out by investigating the transformations of input data to a standard normal distribution, then continuing onto the inclusion of rotation-based iterative Gaussianization and various other iterative processes. Within these works emphasis is placed on finding meaningful non-Gaussian projections of the data through methods such as ICA, PCA and Sliced Wassersteins Flows (SWF). Furthering this work, convergence properties of normalizing flows are discussed and the current area under investigation. Most work considers weak convergence in Gaussianization, whilst this measure of convergence ensure accurate sampling, it does not show convergence of the corresponding densities. This paper fills this gap by considering KL divergence, a strong notion of convergence, and giving explicit convergence rates.

As discussed, this paper will address Gaussianization, this is a method for transforming data so that its distribution becomes approximately Gaussian. It is a type of normalizing flow which is a class of invertible transformations to map complex distributions to simpler ones. Gaussianization is used for its iterative blocks, comprised of random rotations and univariate transformations that result in a more understandable and easier to analytically investigate framework.

Generally what this paper contributes is to analytically derive that the number of Gaussianization layers required to achieve the same improvement in loss grows linearly with the dimensionality of the input and random rotations. Then it demonstrates the limits of determining better-than-random rotations from finite data because of the models inability to capture dependencies between dimensions. Finally, this paper then empirically determines the scaling parameter for real world dataset. This all results in illustrating a general picture over the convergence behaviour of different invertible layers.

Avenues for further work involve developing more effective schemes that capture inter-dimensional dependencies more accurately, in particular utilising better rotations to help with high-dimensional Gaussianization.

2 Question 2

A real-world application for the Gaussianization proposed in this paper is cyber security. Cybersecurity systems typically analyse large and high-dimensional datasets, such as behaviour metrics or network traffic, to identify anomalies that can be perceived as threats. The challenge

with these datasets is that they typically exhibit non-Gaussian distributions, complicating the standard anomaly detection techniques that rely on normality. By applying iterative Gaussianization with rotations and univariate transformations, we can transform the data into a form that approximates a multivariate Gaussian distribution. This transformation allows for the use of conventional anomaly detection methods.

A mathematical quantity that can be affected by this is the Kullback-Leibler divergence between the distribution of the data's distribution and the standard normal. The mathematical quantity can be defined as follows:

$$\mathrm{KL}(p(x)||\mathcal{N}(0,I)) = \int p(x) ln \left(\frac{p(x)}{\mathcal{N}(0,I)} \right) dx \tag{1}$$

The aim of this quantity is to find the minima, indicating that the data closely follow a standard normal distribution. By applying the Gaussianization we can see that the transformed data will reduce the value of this quantity. Ensuring that the standard processes dependent on normality function as they should and result in fewer mistakes and false positives. Saving time, money and potentially far more serious events from occuring.

3 Question 3

This question concerns theorem 1, which states that given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ under the assumption that covariance is normalised and the eigenvalues of the covariance matrix are distinct, we can exactly represent the distribution $p(x) \neq \mathcal{N}(0, I)$ with random rotations Q. We have that at least

$$L \geq \frac{1}{2}(D+1) \tag{2}$$

Gaussianization layers are required.

This theorem is the first analytic lower bound on the amount of Gaussianization layers required for both the end-to-end and iterative training approaches. We can see that the number of Gaussianization blocks required grows linearly with the number of dimensions. A use case for the theorem is with anomaly detection in cybersecurity. In this context, $x \in R^d$ could represent a single network session, described by D dimensions, including the average packet size, number of destination ports contacted, and many more. In reference to the theorem, the assumptions for the network session can be described as ensuring that all of the data is normalised to ensure that one of the features does not dominate. The second assumption ensures that the variables variances aren't identical. This can be thought of as ensuring that the data doesn't become difficult to untangle because when data has the same variance structure, it becomes difficult to distinguish between them and the rotations would be far less effective, requiring more layers.

Referencing the theorem in particular again, what is surprising about this is that the amount of Gaussianization layers required grows linearly with the number of dimensions. This suggests that for the high-dimensionality of data found in real life such as packet size can now be guaranteed to be tractable and efficient when carrying out Gaussianization, rather than requiring exponentially more computation as dimension increases as is typical of these processes.

4 Question 4

Equation 20

Equation 20 summarises that the number of coupling blocks used in coupling-based normalizing flow L_{cpl} required to exactly model a multivariate Gaussian distribution is independent of

the data's dimensionality D. This contrasts with Gaussianization methods where the number of layers needed increases linearly with D.

The relevance of this equation to the paper is to compare the efficiency and scalability of different normalizing flow constructs. This equation directly supports one of the main points, that coupling-based flows are more scalable than Gaussian. This means that as dimensions get increasingly high, this method is more advantageous than Gaussianization due to its independence between dimensions. However, this does not discredit Gaussianization because for low-dimensional systems it is far more efficient that coupling.

This equation is extremely useful for informing an individual who is interested in applying a normalizing flow architecture. In particular, it allows the individual to pick the method based on the dimensionality of the data and computational constraints. In high-dimensional data, coupling is desirable as it will require less blocks but for low-dimensional, Gaussianization will perform faster.

Theorem 2

Theorem 2 states that the number of layers L required to reduce the loss function by a fixed amount scales at least linearly with the dimension D. This is very similar to Theorem 1 with a rewriting to address KL divergence, effectively constructing a lower bound on the KL divergence assuming a given number of layers for a random rotation.

The relevance of this Theorem to the paper is to analyse how quickly Gaussianization converges. It provides a rigorous lower bound on the value of the expected loss and confirms the scaling of Gaussianization with dimensionality D. This is in line with the title of the paper and gives the first explicit convergence result for Gaussianization.

For someone who is interested in applying normalizing flow, this theorem tells the user that Gaussianization will scale poorly to high-dimensional tasks. Suggesting in this case a different normalizing architecture be used as this one will be very computationally expensive in these cases.

Figure 4

This Figure is illustrating the limitations of iterative training in high dimensions. In particular, even when your data set is sampled from a perfectly standard normal $x \sim \mathcal{N}(0, I)$, it is possible to find a projection that falsely appears to be non-Gaussian and improve the loss but actually worsens it. The Figure produces a random and optimized projection, showcasing that Gaussianization would falsely transform Gaussian data into a bimodal distribution.

This is relevant to the paper as it corroborates what the author states about iterative Gaussianization and its convergence issues in high-dimensional data. Specifically it shows that false structures can be imagined to be present in the data as a result of over-fitting and that it worsens with increasing dimensionality. This empirical demonstration corroborates the analytical findings demonstrated throughout the report.

This is useful to someone that is interested in applying iterative Gaussianization as it demonstrates the effects of over-fitting in high-dimensions. Resulting in projections that looks non-Gaussian, for potentially Gaussian data. It also suggests that end-to-end training with learned rotations outperforms iterative in these circumstances and should be considered as an alternative.

5 Classification

In Task A, we are instructed to formulate a decision rule which predicts the characters depicted in the images in the dataset *x_test*. My general strategy for this question is to apply a convolutional neural network to the data along with data augmentation to train the model with data *xy_train*. Then make predictions using the trained model with the *x_test* dataset.

Preliminary analysis was carried out to inspect the data and get it into a format compatible with the 'keras' R package. This included producing images of the data to get an idea of what the end result should look like and ensure that when the end result is produced, that they are correct in a manner consistent with what we already know. As for the preparation of the data itself, the training and testing datasets have the first column separated so as to give the model the inputs and the output separately for learning. The inputs are then normalised to reduce the likelihood of exploding gradients, ensure faster convergence and improve performance overall. The inputs are then reshaped to resemble the size of the image, so we have 500 28×28 arrays for use in the model. The outputs are turned into categorical variables for use in the classification model. The final piece of preliminary analysis that we carried out was to divide the datasets into a training and validation training set to be used for parameter tuning.

The methods used can be seen as follows, we constructed a training and validation split in the data, this allowed us to evaluate our performance on unseen data during the training process to ensure overfitting is not occurring. We then constructed 'TensorFlow' datasets to boost efficiency, this is done by shuffling the samples within buffers of size 1000, grouping them into batches of size 6 and then using prefetch to speed up computation.

Now we move onto the construction of the model, this is initially done through data augmentation, this applies random transformations to the input images to simulate new data from the old data. In this we applied a rotation, a zoom and a flip to generate the new images. For the convolutional neural network, we use convolutional, pooling, flattening and fully connected layers. In this model we use 4 convolutional layers with 32, 64, 128 and 128 filters across the images to progressively capture more detail with an activation function corresponding to the 'relu' function that removes any negative elements. For each convolutional layer, a pooling layer is utilised with pool size 2 to take the maximum in a 2×2 window, reducing the feature maps size. Once the model has run through these layers, it will have constructed a very complex model for analysing the images, now we apply the flattening layer to convert the 3D output to a 1D output and then we apply a fully connected layer with 512 neurons to classify the image based on all the combined features. The model is then finished by using an output layer with the 6 output categories and a activation function equivalent to the softmax function, this function converts the outputs to probabilities for which the largest corresponds to the output.

With the model constructed we now compile it through the 'adam' optimizer as it is adjusts learning rate per parameter to give an adaptive and efficient output. The loss function used is 'categorical cross entropy', this choice seems sensible for multi-class classification. Then this model is trained using 100 epochs and a accuracy-loss plot produced to ensure that the model is converging and not over-fitting, whilst giving insight into the accuracy of the model on the validation set. We found a typical validation accuracy of 0.97 and accuracy of 0.99. This suggests over-fitting is not occurring and these parameters are optimal. With this completed and parameters tuned, we now remove the validation set and trained using the full dataset to make the model as effective as possible. This model is then applied to the testing dataset and predictions made as shown in the .Rdata file.

The protocols implemented to prevent over-fitting are a training/validation split, this allows for the performance to be monitored and ensure no over-fitting is occurring. Data Augmentation is also applied and helps by introducing random noise into the training data and reduce the chance that the model memorises specific examples. The shuffling of the training data prevent patterns from being recognised in a similar way to the data augmentation. These protocols

have led to a well-constructed model that does not overfit, as shown in the .RData file which can be checked with the naked eye to be close to the validation accuracy of 0.97.

A convolutional Neural Network was chosen over other methods such as SVM's, Transfer learning, and GAN's is due to its simplicity and effectiveness when working with simple images. SVM's require feature construction and can be difficult when dealing with images. Transfer Learning is a great alternative and can typically be used with a CNN to get a more accurate model, we did not proceed with this combined method here as there was no significant improvement seen when implementing it. GAN's typically require a lot of data, as we only had 500 training samples, this method did not seem worth attempting here. One other feature that was attempted in constructing this model was to utilise cross validation. This made the computation take far longer and should have increased the performance of the model but when this was implemented with data-augmentation the model performance decreased as opposed to when one or the other was applied. Therefore I opted for data-augmentation as it performed better, is new, and is interesting to me.

6 Image Generation

For this image generation task, we opted to use a conditional variational auto-encoder with data augmentation. Through this we have produced 3000 images, 500 of each of the classes requested. This has been done using both the x_test and xy_train with the help of our predictions from Task A and resulting in a larger training set.

The method used here is conditional variational auto-encoding. To start, we define the input dimensionality of our data to be 784, the dimensionality of the latent space to be 20, the intermediate dimensionality to be 256, and a batch size equal to 6. The latent space is the number of features used to represent each image, the intermediate is the size of the hidden layer between the input and the latent output, and the batch size is the amount of samples processed at once. These are chosen to minimise overfitting, maximise speed, and effectively represent the data. With the initial parameters set, we now proceed with the application of the VAE through the use of an encoder. We have used a fully connected layer with the 'relu' activation function to extract the features. Then 2 parallel layers, each of latent size, that learn the mean and log variance of the latent distribution, defining a Gaussian distribution in the latent space. Then the encoder is completed by taking the image and label as input and outputting the Gaussian.

To carry out the back propagation required, we use the reparametrization trick as this will circumvent the issue of non-differentiability in the sampler. We calculate some random noise and then incorporate this in with the Gaussian components to construct the gradients. Now we apply decoding, the input of this function is the output of the encoder. The latent variable is put through a 'ReLU' activation function, from which it is then applied through a 'sigmoid' function and the output provides a 784 pixel image.

Finally, we use these encoders and decoders in the overarching model for which we apply a training step that uses the encoder and decoder to construct the image by resampling from the distribution. Then calculates both KL loss and reconstruction loss to ensure that the model is not deviating far from a standard normal and that the images are constructed accurately. We then compute gradients and use these to update weights of the model. It is important to mention that since this is a conditional VAE, the encoder and decoder are conditioned on the class and accommodate this in its evaluation.

Other methods that could have been applied to this are GANs and diffusion models, however, GANs can be unstable during training with a small amount of data and require a lot of data to function well. Diffusion models, despite being extremely advanced, require complex computation and a lot of data. Overall our VAE model is the most efficient for this context and ensures good convergence.