A Sparse VAE

Ryan Pyle
rpyle1@nd.edu

Abstract

Auto-encoders are a useful tool that allows for high dimensional data (e.g. images) to be decomposed into a smaller number of latent variables, which ideally contain all task-relevant information. Auto-encoders use frequentist statistical ideas; the latent variables are treated as point estimates. Variational auto-encoders (VAEs) instead treeat each latent variable as a parameterized distribution. This allows for several useful new features, such as generating synthetic data by sampling from the underlying latent distribution. One idea that is not well supported is sparsity. Sparsity can be useful as a way to promote more informative latent elements, or to decrease operating costs in physical systems. This was famously shown by Olhausen and Fields in regards to biological image processing in the receptive fields of the mammalian eye. Sparse auto-encoders are simple to produce, but no work has been done on sparse VAEs. In this work we aim to produce a sparse VAE algorithm, which will require reworking the VAE away from its (implicit) gaussian prior.

1 Introduction

In order to build a Sparse VAE, we need to thoroughly understand the relevent concepts of VAEs and sparsity/sparse auto-encoders.

1.1 Auto-encoders

Auto-encoders[1] are based on a standard artificual neural network (ANN) feedforward architecture. The main difference is that the size of the layers is largest at the input and output, and smallest in the middle of the network. The network is often thought about as two conjoined networks: the encoder portion, which takes the large input and results in the smallest middle layer as outupt, and the decoder, which has the smallest middle layer as input and the reconstruction as its output. The network is trained to exactly reproduce its inputs as its outputs, which creates an obvious loss function and makes for easy training via gradient descent. Thus, all relevant information needed to generate the output must be passed through the smallest internal layer, which are then treated as our latent variables.

1.1.1 Sparsity and sparse auto-encoders

Sparsity refers to the property of having a limited number of active (or non-zero) elements in a set. One possible reason for sparsity is to promote efficiency. Consider a biological system which needs to represent some high-dimensional signal. It may be more energy efficient to encode the signal from a small number of over-determined basis functions than by a using every the set complete set of basis functions. In addition, sparse representations have been shown to be easier (and more biologically realistic) to learn [2]. Olhausen and Fields showed that using a sparse code for natural images matches receptive fields of the mammalian retina[3].

These same arguments can also apply to auto-encoders. Adding a sparsity loss function lowers the number of latent variables which can be active at one time, ensuring that they have a higher degree of statistical independence, which helps ensure that the latent variables are more 'useful' at encoding real quantities of interest[4].

1.2 Variational auto-encoders

VAEs[5]-[6] have a similar architecture to auto-encoders, but are generative models. That means that we are interested in modeling the distribution of of our high-dimensional data, the latent distribution it comes from, and the parameters of their relationship. If our data comes X comes from distribution $P(X)$, we wish to learn some sampleable $\hat{P}(X)$ which is similar to $P(X)$. $\hat{P}(X)$ sampleable means that we can generate new data that is 'similar' to X, usually by sampling the underlying z. More formally, consider we have inputs X and latent variables z. A paramaterized function $f(z, \theta) \rightarrow x$ transfers latent variables to actual data. We consider z now as a random variable - the distribution of z, which is almost always taken to be $N(0, I)$, captures the variability within X, so that with appropriate θ the random z can generate our data X through f. Note that in the true data X, the latent variables z that generated X almost certainly do not come from a $N(0, I)$; we assume that the mapping from $N(0, I)$ to the true latent distribution is learned along with the mapping from the true distribution to the output.

Traditionally, problems like this could be solved with an existing algorithm. However, in this case some of the likelihood are considered to be intractible. In particular,

$$p(z|x, \theta) = \frac{p(x|z, \theta)p(z|\theta)}{p(x|\theta)}$$

is intractible, which prevents using the EM algorithm. Another possibility is to use Variational Bayes, which allows us to get around estimating $p(x|\theta) = \int_z p(x, z|\theta)$ by instead taking $p(z|x, \theta) \sim Q(z)$. However, Variational Bayes methods can fail if $p(z|x)$ is sufficiently complex (e.g. mean field VB assumes that $p(z)$ can be factorized into $p(z_1)p(z_2)...$, but what if $p(z|x)$ has a term that depends on many or all z elements?). Finally, sampling solutions are too slow for large and complex datasets, such as modern image recognition datasets. VAEs can get around all of these issues.

Our goal then is to maximize $P(X) = \int P(X|z, \theta)P(z)dz$, where using $P(X|z, \theta) = N(X|f(z, \theta), \sigma^2 I)$ is the standard in VAEs. Using a normal distribution allows us to learn via gradient descent, which would be impossible with delta functions $P(X|z, \theta) = f(z, \theta)$. Optimizing this the standard way would require taking a huge number of samples. Instead, we use the fact that $P(X|z)$ will be near 0 for most values of z. If we were to only sample values of z that were likely to generate X, the problem would be much easier. This is done through a new function $Q(z|X)$, which takes in data X and returns a distribution of z values that are likely to generate X, which allows us to easily compute $E_Q[P(X|z)]$. The final step is to relate $Q(z|X)$ to our original $P(x)$, where z is drawn from $N(0, I)$ rather than Q. Recall that the KL divergence between two distributions A and B is

$$KL(A||B) = \int A \log(\frac{A}{B}) = E_A[\log(A) - \log(B)]$$

taking the KL divergence between arbitrary Q and $P(z|X)$ yields

$$KL(Q(z)||P(z|X)) = E_Q[\log(Q(z)) - \log(P(z|X))]$$

applying Bayes rule to $P(z|X)$

$$KL(Q(z)||P(z|X)) = E_Q[\log(Q(z)) - \log(P(X|z)) - \log(P(z))] + \log(P(X))$$

where we pulled out the term that didn't depend on z. Rearranging terms,

$$\begin{aligned} log(P(X)) - KL(Q(z)||P(z|X)) &= E_Q[\log(Q(z)) - \log(P(X|z)) - \log(P(z))] \\ &= E_Q[\log(P(X|z))] - KL(Q(z)||P(z)) \end{aligned}$$

Note that since $Q(z)$ was arbitrary, we can replace it with $Q(z|X)$ without issue. This forms the central equation of the VAE:

$$log(P(X)) - KL(Q(z|X)||P(z|X)) = E_Q[\log(P(X|z))] - KL(Q(z|X)||P(z))$$

The left hand side is what we are attempting to optimize. We want to maximize $P(X)$, while also keeping our $Q(z|X)$ near to the true $P(z|X)$. This is equivalent to the right hand side, which is significantly easier to work with and can be optimized via stochastic gradient descent. It has certain similarities to an auto-encoder: encoder $Q(z|X)$ generates latent variable (distributions) from data X, and while decoder $P(X|z)$ takes z and generates a data output X. Optimizing the RHS requires a few more assumptions. First, we assume $Q(z|X) = N(z|u(X,\theta), \Sigma(X,\theta))$, or that Q is just a normal distribution with means and variances based on learned functions of the data X. Usually, Σ is constrained to be diagonal, making generated z dimensions be independent. Making this choice makes

$$KL(Q(z|X)||P(z)) = KL(N(u(X,\theta), \Sigma(X,\theta))||N(0,I))$$

or the KL divergence between two gaussians, which is known in closed form. In this case, since one of our guassians is just $N(0,I)$ is simplifies to

$$KL(Q(z|X)||P(z)) = \frac{1}{2}[tr(\Sigma) + u'u - k - \log(det(\Sigma))]$$

where k is the number of latent dimensions. All that is left is to evaluate the first term, $E_Q[\log(P(X|z))]$. Accurately estimating this would be expensive, so instead we take only one z to use as an estmiate, and instead take the average of our different data X (e.g. standard stochastic gradient descent). In stochastic gradient descent, we are actually interested in $E_X[log(P(X)) - KL(Q(z|X)||P(z|X))]$. Thus, in each step we take a single X and single z from $Q(z|X)$, and average over many trials (a batch) to get our estimate of $E_X[E_Q[\log(P(X|z))] - KL(Q(z|X)||P(z))]$ from each trial's $\log(P(X|z)) - KL(Q(z|X)||P(z))$. There is one last problem : in order to optimize this value, we need to backpropogate the cost function throughout the network. z is sampled from $Q(z|X) = N(z|u(X,\theta), \Sigma(X,\theta))$, which is a stochastic operation that has no gradient. The solution is to use the 'reparamaterization trick' [7] to get rid of sampling in the middle of the network. Instead, a $\epsilon \sim N(0,I)$ is drawn, and we use $z = u(X,\theta) + \Sigma^{\frac{1}{2}}(X,\theta) * \epsilon$, which is functionally identical. Thus, for a given X and ϵ, the whole thing is deterministic and can be backpropogated through to get update derivatives. Although it is functionally the same result, it is worth noting that we are actually solving

$$E_X[E_{\epsilon\ N(0,I)}[log(P(X|z) = u(X,\theta) + \Sigma^{\frac{1}{2}}(X,\theta)\epsilon] - KL(Q(z|X)||P(z))]$$

1.2.1 Example

Most of my work will be done based off of an existing VAE implementation in tensorflow. However, in order to make sure I understood VAEs I did my own implementation (of a simpler system) in MATLAB. Data was generated by random addition of two gaussian templates, corrupted by a small amount of noise. The architecture took the resulting 10 by 10 images, encoded them (in one layer) into a pair of latent dimensions, and then decoded them (in one layer) into 100 dimensional output, which was then passed through a sigmoid. Optimizing the network required backpropogating the error gradient through the network. The resulting trained network was able to do well in recreating inputs and had latent variables that were moderately intuitive (see appendix).

2 A Sparse VAE

VAEs work quite well, but they can suffer from the same issues as (non-sparse) auto-encoders: correlated latent variables may have high explanatory power of the data, but be difficult to interpret in a useful manner. Additionally, having a large number of active latent variables may be detrimental in some systems (e.g. energy concerns in biological systems). Can sparse auto-encoder framework can be extended to VAEs? In a sparse auto-encoder, an additional term in the loss function (most frequently an L1 norm over latent variables) helps to promote sparsity. In VAEs, instead we have latent distributions. How can we promote sparsity over the latent distribution? It is worth noting in standard VAEs that we essentially have a gaussian 'prior': we optimize over the KL divergence between $Q(z|X)$ and a $N(0,I)$. The standard way to promote sparisty would be change our prior to a laplacian. However, one of the major reasons the VAE works as well as it does is becuase we take the KL divergence between two gaussians, which has a simple analytical solution. Instead, using a Laplace distribution yields

$$KL(Q(z|X)||P(z)) = KL(N(u(X,\theta), \Sigma(X,\theta))||Laplace(0,I))$$

$$= \int \frac{exp(-\frac{1}{2}(x-u)'\Sigma^{-1}(x-u))}{\sqrt{(2\pi)^k Det(\Sigma)}}[\log(\frac{exp(-\frac{1}{2}(x-u)'\Sigma^{-1}(x-u))}{\sqrt{(2\pi)^k Det(\Sigma)}}) - \log(\prod \frac{1}{2} exp(-|x|))]dx$$

which does not have an analytical solution. This leaves us to a numerical solution. We already know that we can rewrite the KL divergence as

$$KL(Q(z|X)||P(z)) = E_Q[log(\frac{Q(z|X)}{P(z)})]$$

or that we can sample from our $Q(z|X)$ and then compute the log likelihood ratio over the sampled points as an estimate of the true KL divergence. However, in order to get a good estimate we would need to take quite a few samples, making the runtime of the sparse VAE intractibly slow. Recall the the standard VAE derivation also had a similar issue; $E_Q[\log(P(X|z))]$ would also have inhibitively expensive to estimate. We can apply the exact same trick that we did in the standard VAE which is to evaluate

$$E_X[E_{\epsilon\ N(0,I)}[log(P(X|z) = u(X,\theta) + \Sigma^{\frac{1}{2}}(X,\theta)\epsilon - \log(\frac{Q(z = u(X,\theta) + \Sigma^{\frac{1}{2}}(X,\theta)\epsilon|X)}{P(z = u(X,\theta) + \Sigma^{\frac{1}{2}}(X,\theta)\epsilon)})]]$$

We have already done the reparamaterization trick in order to change the E_Q term into an E_ϵ - we just extend it to also include our $log(\frac{Q(z|X)}{P(z)})$ term.

2.1 Alternative Approach

I also did another version of the sparse VAE. The basic reasoning and formulation are the same as the above sparse VAE, with two major difference. The first difference was that a cauchy (rather) than Laplacian prior was used. There are several reasons this might be a better choice. First, the cauchy distribution has heavier tails, which may help to explain certain rare X e.g. outliers. The other reason was that the Cauchy distribution (along with the normal distribution from the standard VAE) is in the Levy alpha-stable family of distributions, whose members have the peroperty that their sum is still the same distribution. The other change made was to assume the latent variables were cauchy distributed rather than $N(0, I)$, or equivelantly that the noise used in $Q(z|X)$ was cauchy instead of gaussian. Along with the other change, this meant that the distribution $Q(z|X)$ could possibly match the underlying assumption for $P(z)$, a trait which the original VAE had (since both were guassian), but that the first sparse VAE could not (sum over laplacians cannot be a gaussian).

3 Results

I used the standard (gaussian prior) VAE as well as the both the sparse (laplacian prior) and sparse (cauchy prior and cauchy generator) VAE described above. I ran all algorithms on several data sets. The first data set was synthetic data : 5x5 image patches created from 5 basis functions (plus noise). I used 10 latent dimensions. In all cases,the algorithms worked well at learning the task (recreating inputs with low reconstruction error). Analysis of the relative usefulness/clarity of the latent dimensions was left for the other tasks where it made more sense.

The second test was done on the MNIST data set, which is 28x28 image patches of the digits 0 to 9. I again used 10 latent dimensions. Reconstructions worked fairly well in all cases, although the reconstructions were noticeably less sharp in the cauchy prior cases. What is more interesting is the latent dimensions : in the gaussian prior, they are sharp images that look like corrupted components of digits, while the laplacian prior has blurred combinations of digits and the cauchy has the most readalbe latent dimenions, corresponding one to one to most digit classes. I also did a 2D version (2 latent dimensions), which allowed for vieweing of the latent variable spread of the data, and the reconstructions given latent input from a grid. The latent variable spread for the laplace case was better seperated than the gaussian case. In the cauchy version, the spread appreared to be worse - essentially only one of the two latent dimensions was used. Similar results were found in the latent grid reconstructions.

The final test was on the CIFAR dataset, which is made of 32x32 image patches, which I subsampled into 12x12 patches. I used 25 latent dimensions. Note that with this few latent

dimensions, perfect reconstruction over a broad range of data (natural images) was impossible. Nonetheless, the normal and laplacian priors did a decent job at reconstruction, while the cauchy prior was blurrier. The latent dimensions for gaussian and laplacian priors were fairly uninformative - random patterns of light and dark, with the laplacian prior looking like a smoother version of the gaussian. On the other hand, the cauchy prior had sparse gabor like latent dimensions, similar to the Olhausen and Fields results.

See appendix for the source images.

4 Conclusion

VAEs are an excellent machine learning tool in order to learn the underlying source of variability within a dataset, and to sample new elements from the latent variability. Sparse VAEs could combine their power with the greater biological realism and usability of sparse auto-encoders and sparse codes. This first attempt at a sparse VAE was a mixed success - both the laplacian prior and cauchy prior sparse VAE versions had desiriable sparse properties, but they rarely overlapped. The laplacian prior maintained the standard VAE's excellent reconstructions, while enhancing seperability, and slightly improving the intuitive usefulness of the latent dimensions. The cauchy prior had a larger tradeoff in reconstruction error (although it still did generally well), and sgifniciantly more useful latent dimensions, but had a troublesome tendency to decrease seperability in the 2 latent dimension tests.

5 References

Acknowledgments

Thanks to Gabriel Barello for helpful converstations.

References

[1] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." science 313.5786 (2006): 504-507.

[2] Willshaw, David, and Peter Dayan. "Optimal plasticity from matrix memories: What goes up must come down." Neural Computation 2.1 (1990): 85-93.

[3] Olshausen, Bruno A. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images." Nature 381.6583 (1996): 607-609.

[4] Ng, Andrew. "Sparse autoencoder." CS294A Lecture notes 72.2011 (2011): 1-19.

[5] Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).

[6] Doersch, Carl. "Tutorial on variational autoencoders." arXiv preprint arXiv:1606.05908 (2016).

[7] Maddison, Chris J., Andriy Mnih, and Yee Whye Teh. "The concrete distribution: A continuous relaxation of discrete random variables." arXiv preprint arXiv:1611.00712 (2016).

6 Appendix

6.1 VAE example

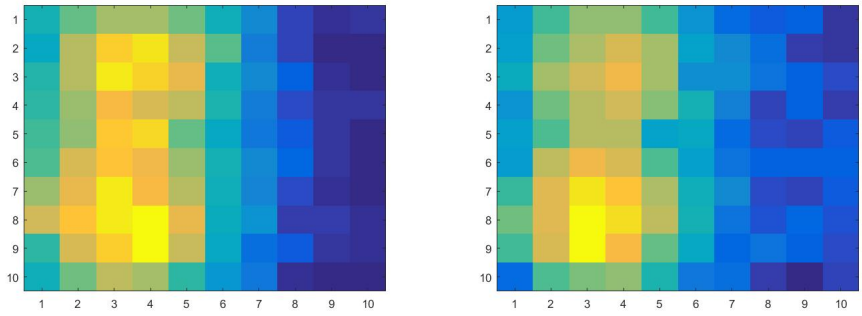

Figure 1: Input and Reconstruction for MATLAB VAE implementation

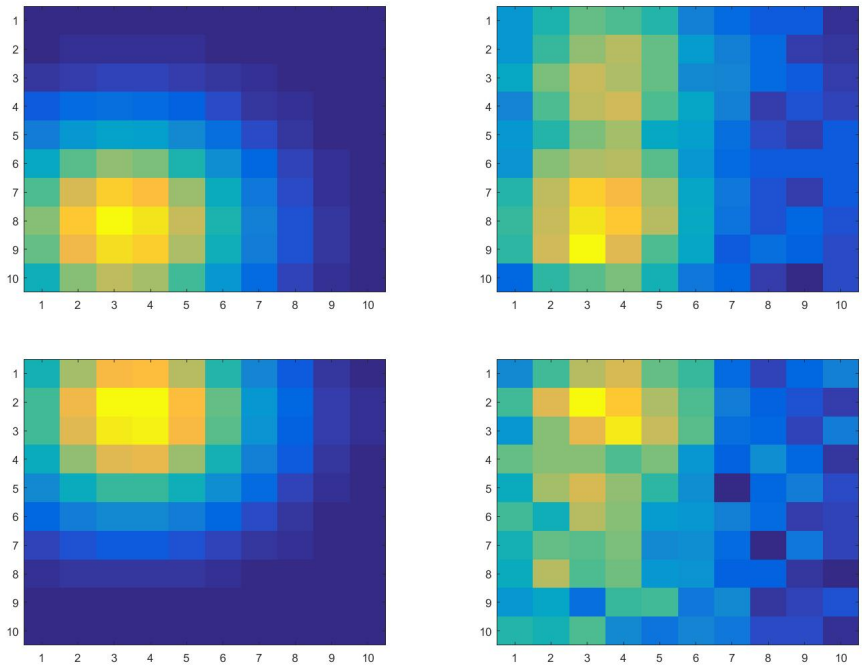

Figure 2: Patterns (left) and latent dimensions (right) of MATLAB VAE implementation

6.2 VAE vs Sparse VAE Task Comparisons

Ordered by task, then by data output, then by VAE type.

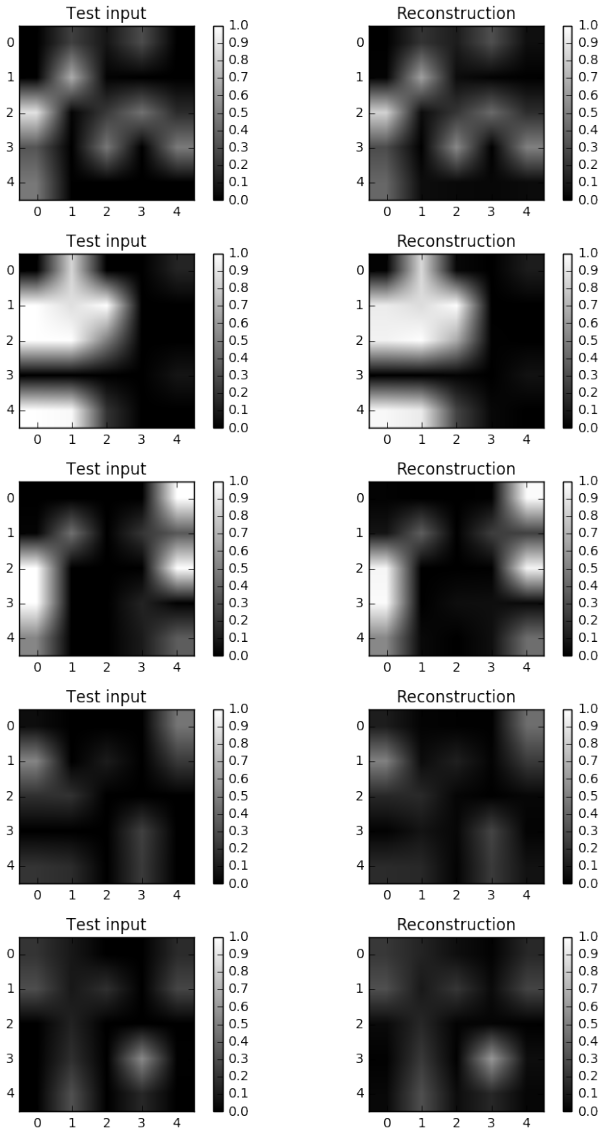

Figure 3: Normal VAE reconstructions

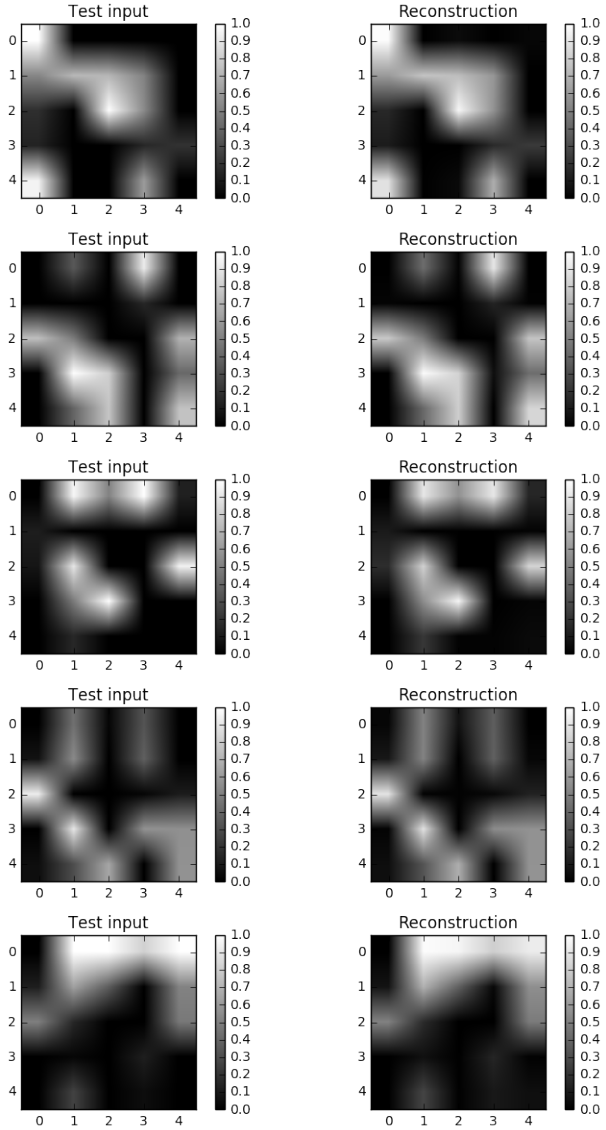

Figure 4: Laplace VAE reconstructions

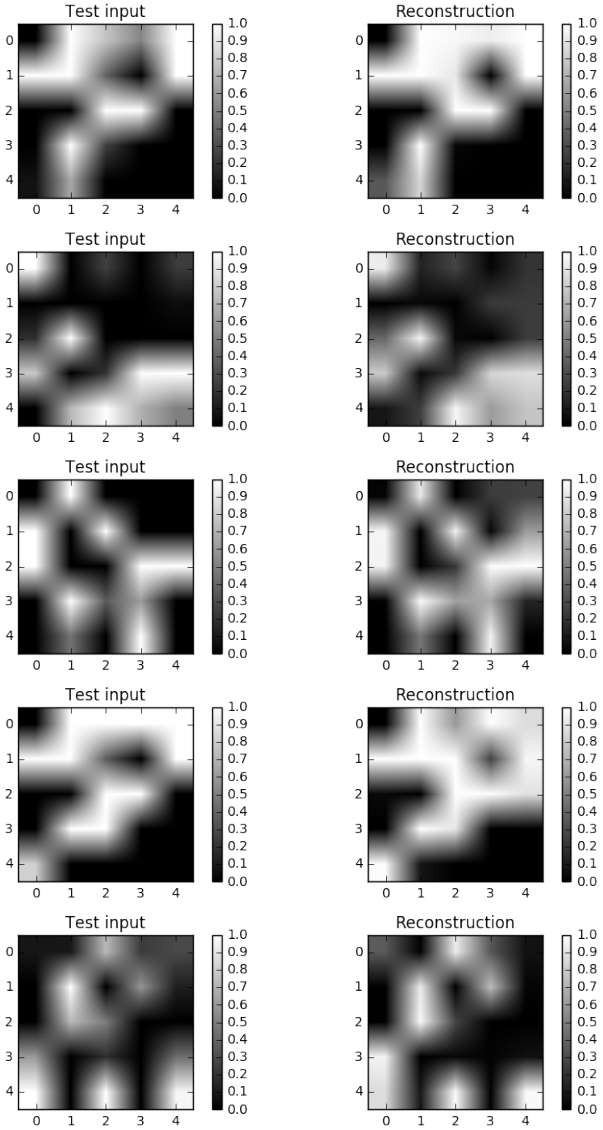

Figure 5: Cauchy VAE reconstructions

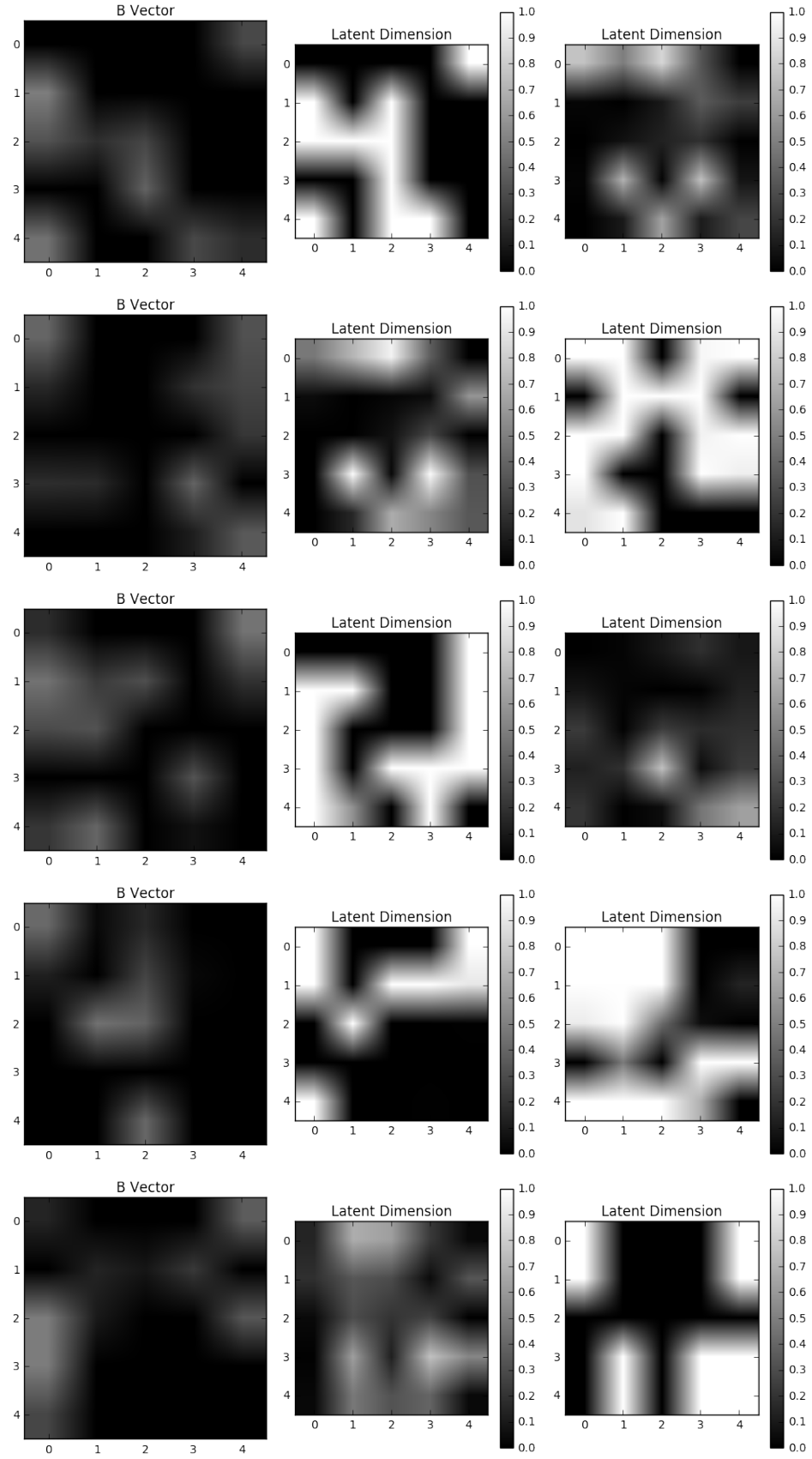

Figure 6: Normal VAE patterns vslatent dimensions

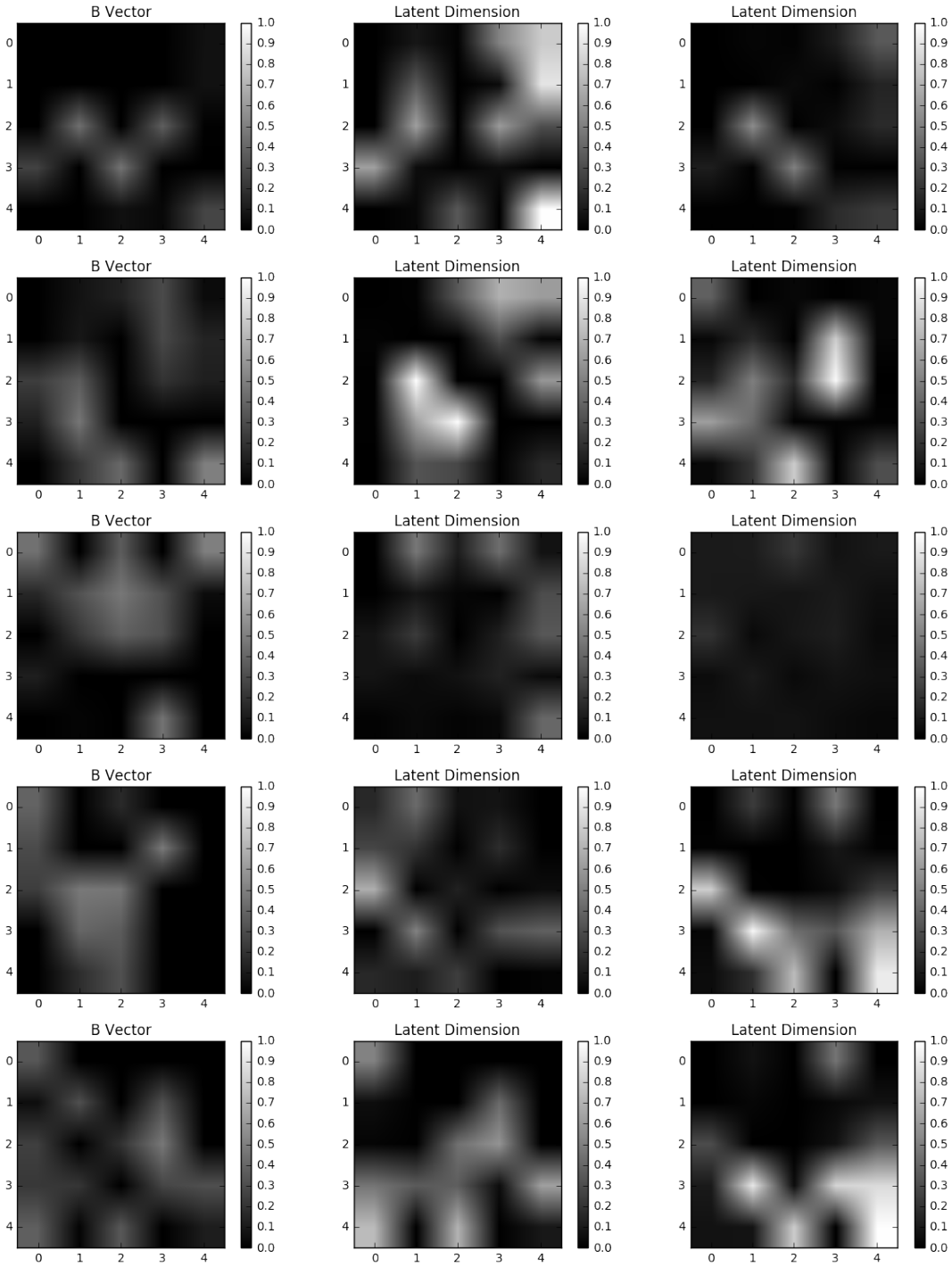

Figure 7: Laplace VAE patterns vs latent dimensions

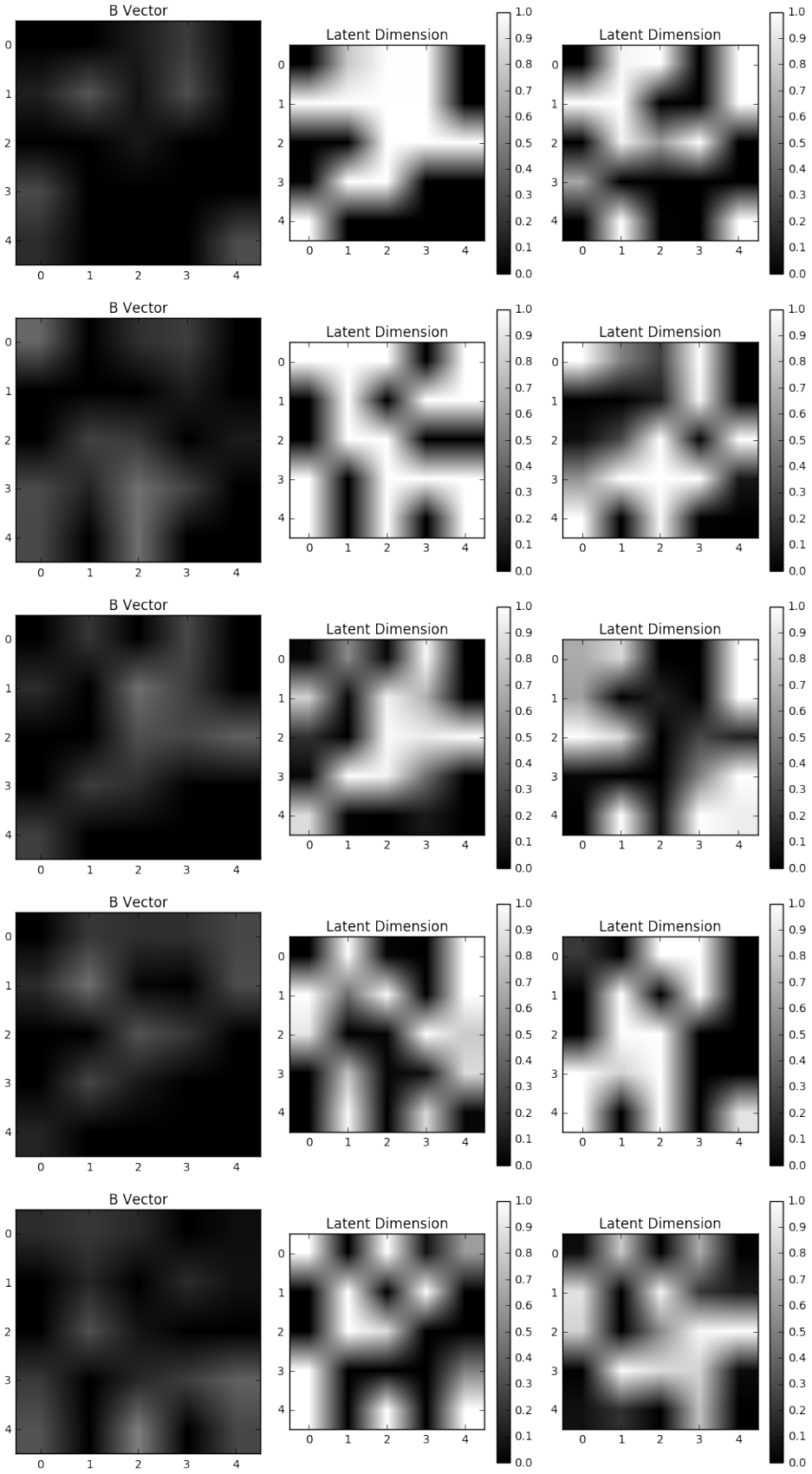

Figure 8: Cauchy VAE patterns vs latent dimensions

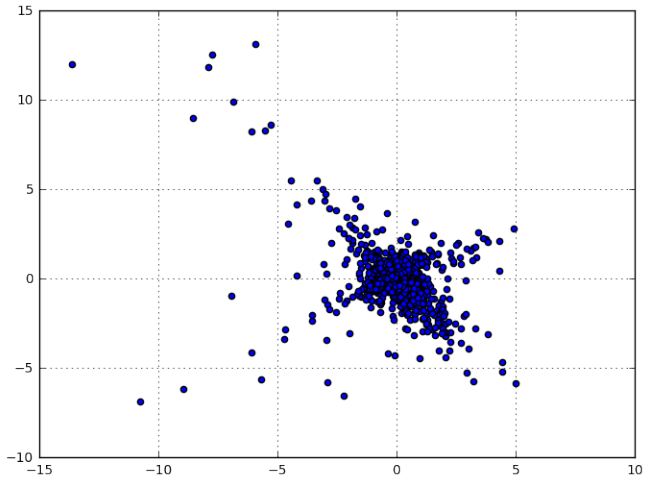

Figure 9: Normal 2D VAE latent spread

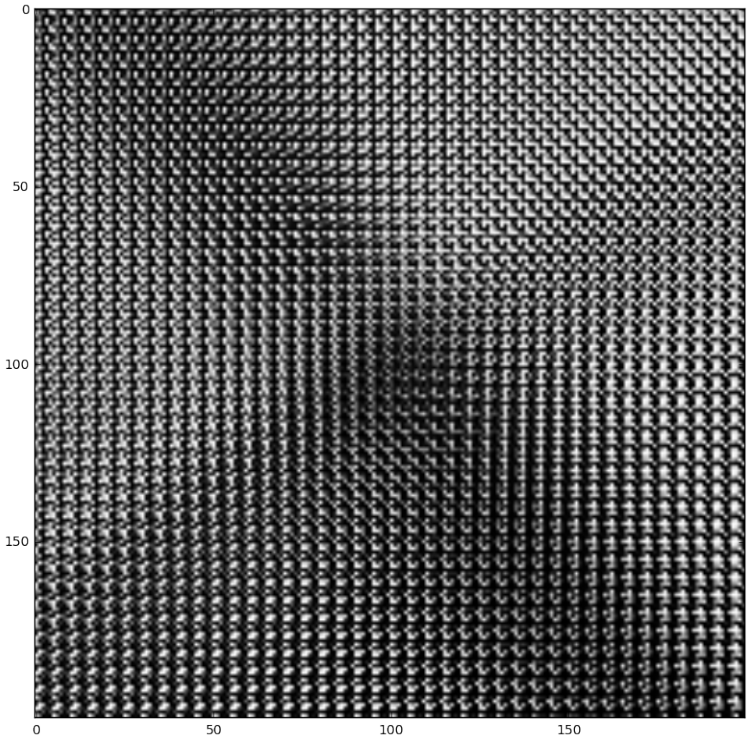

Figure 10: Normal 2D VAE latent grid reconstruction

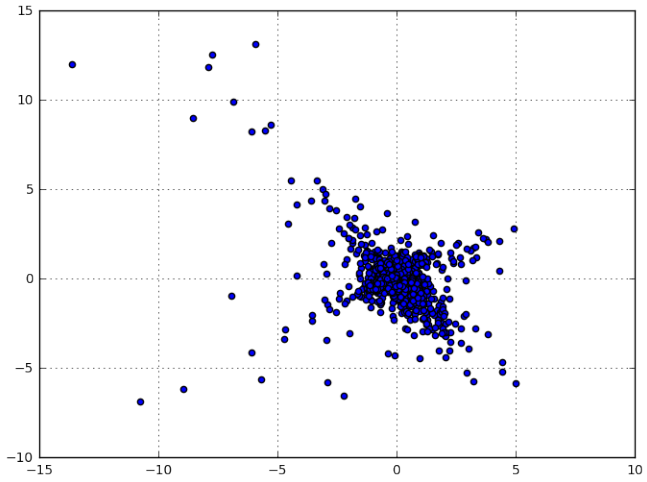

Figure 11: Laplace 2D VAE latent spread

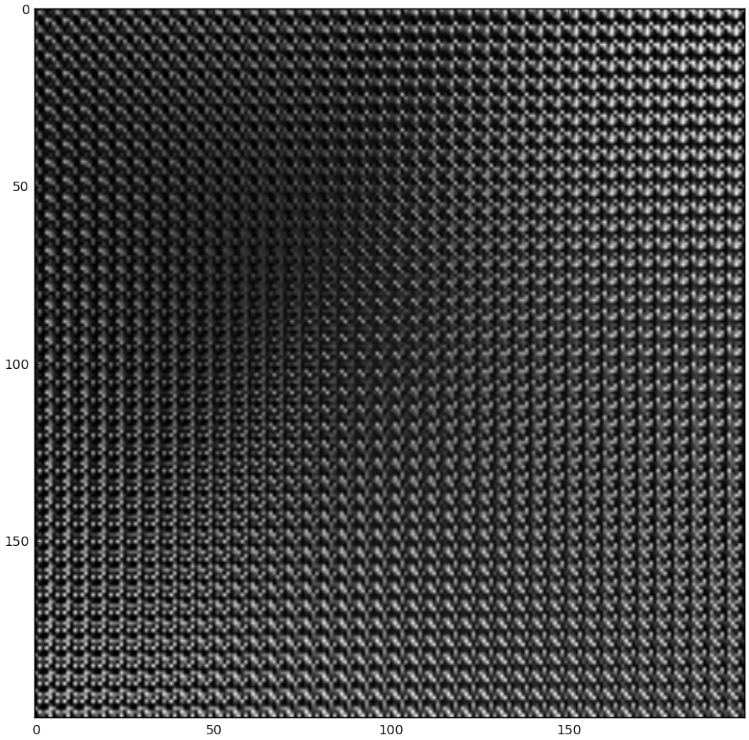

Figure 12: Laplce 2D VAE latent grid reconstruction

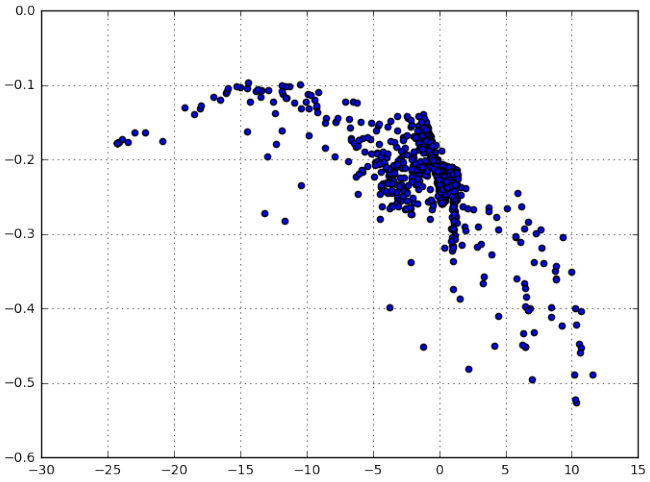

Figure 13: Cauchy 2D VAE latent spread

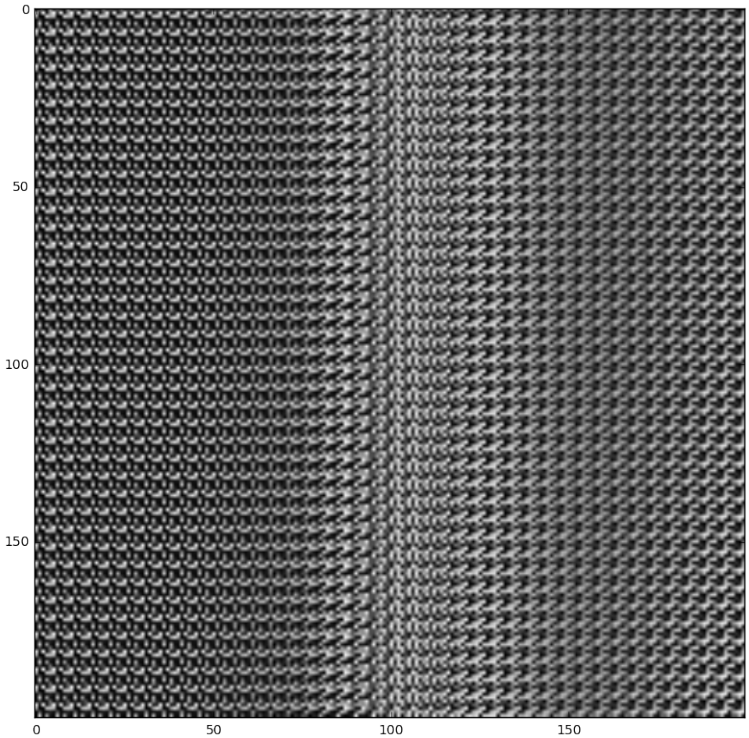

Figure 14: Cauchy 2D VAE latent grid reconstruction

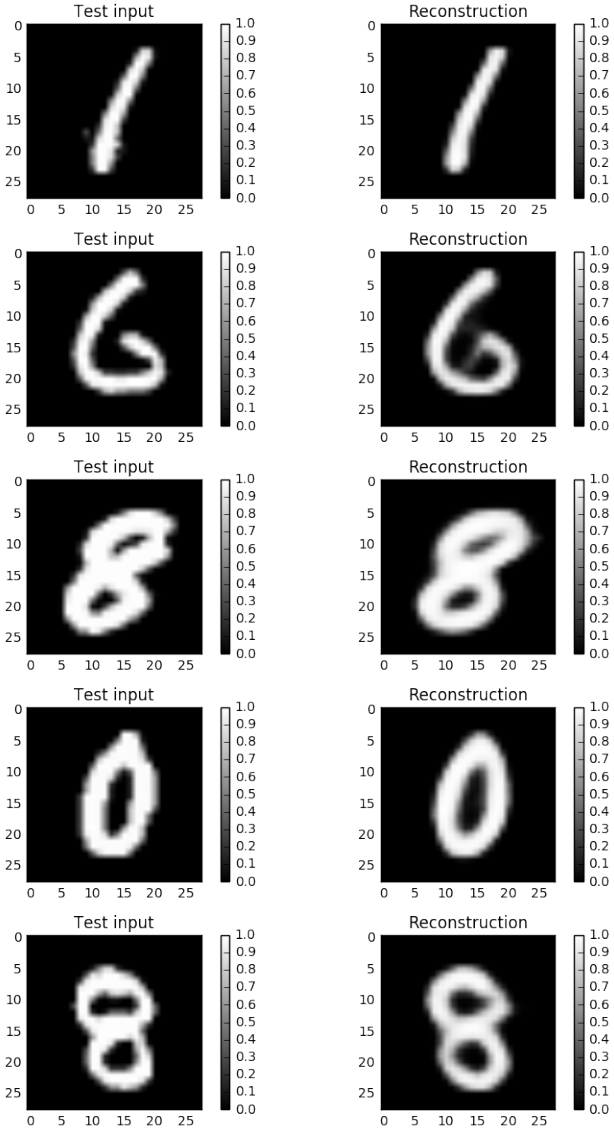

Figure 15: Normal VAE reconstructions

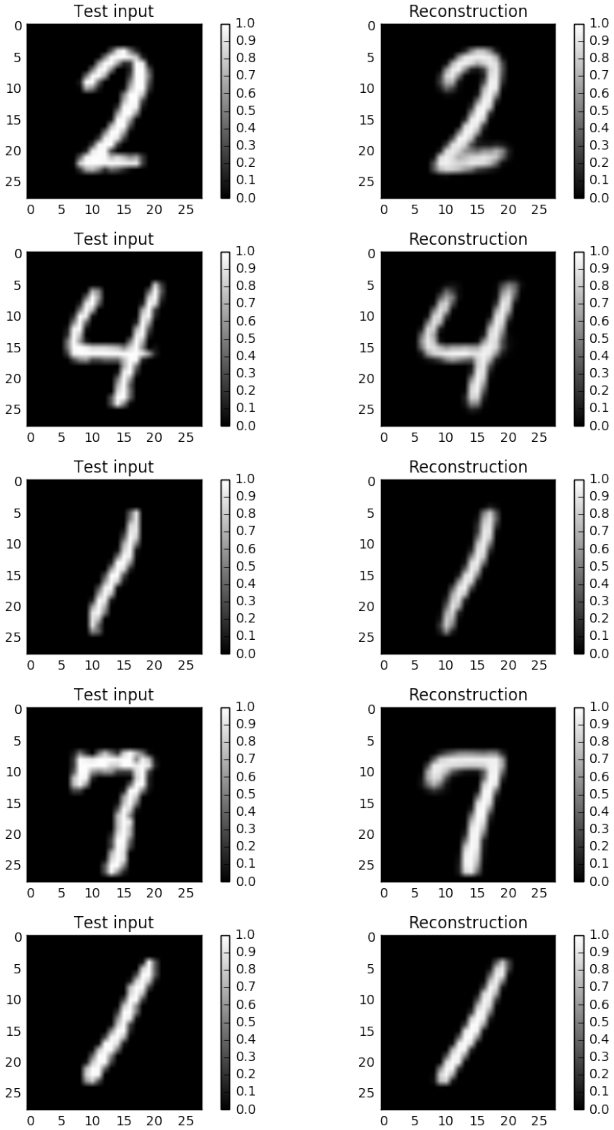

Figure 16: Laplace VAE reconstructions

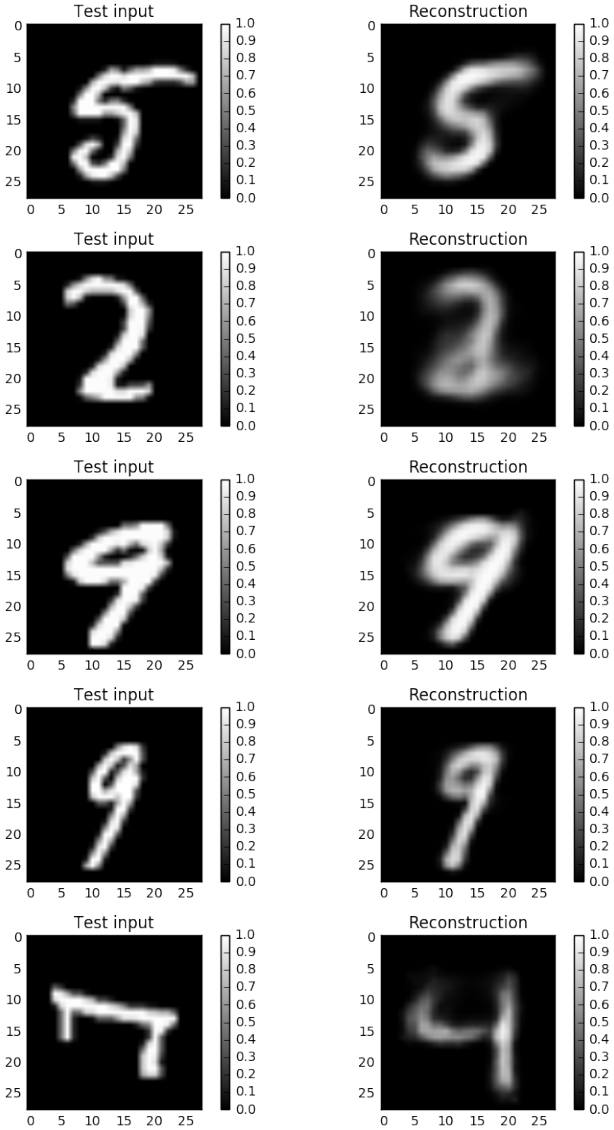

Figure 17: Cauchy VAE reconstructions

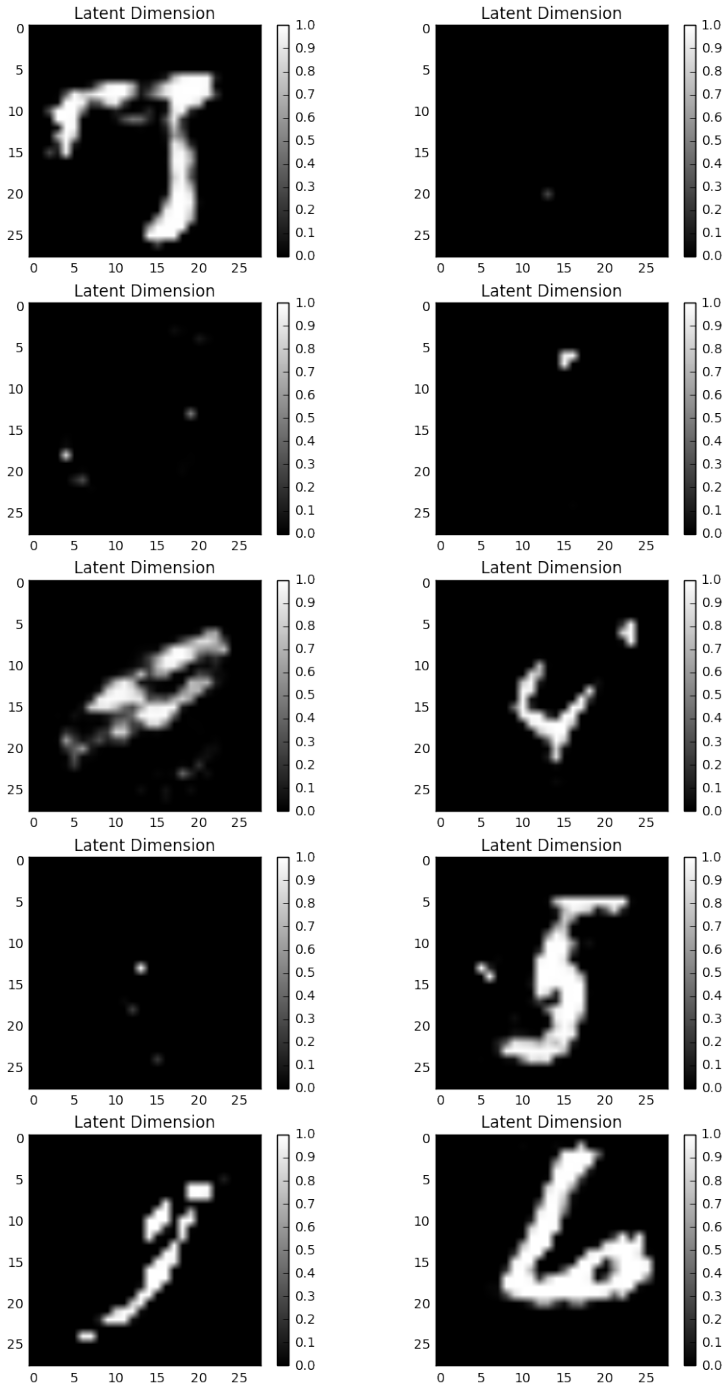

Figure 18: Normal VAE patterns vslatent dimensions

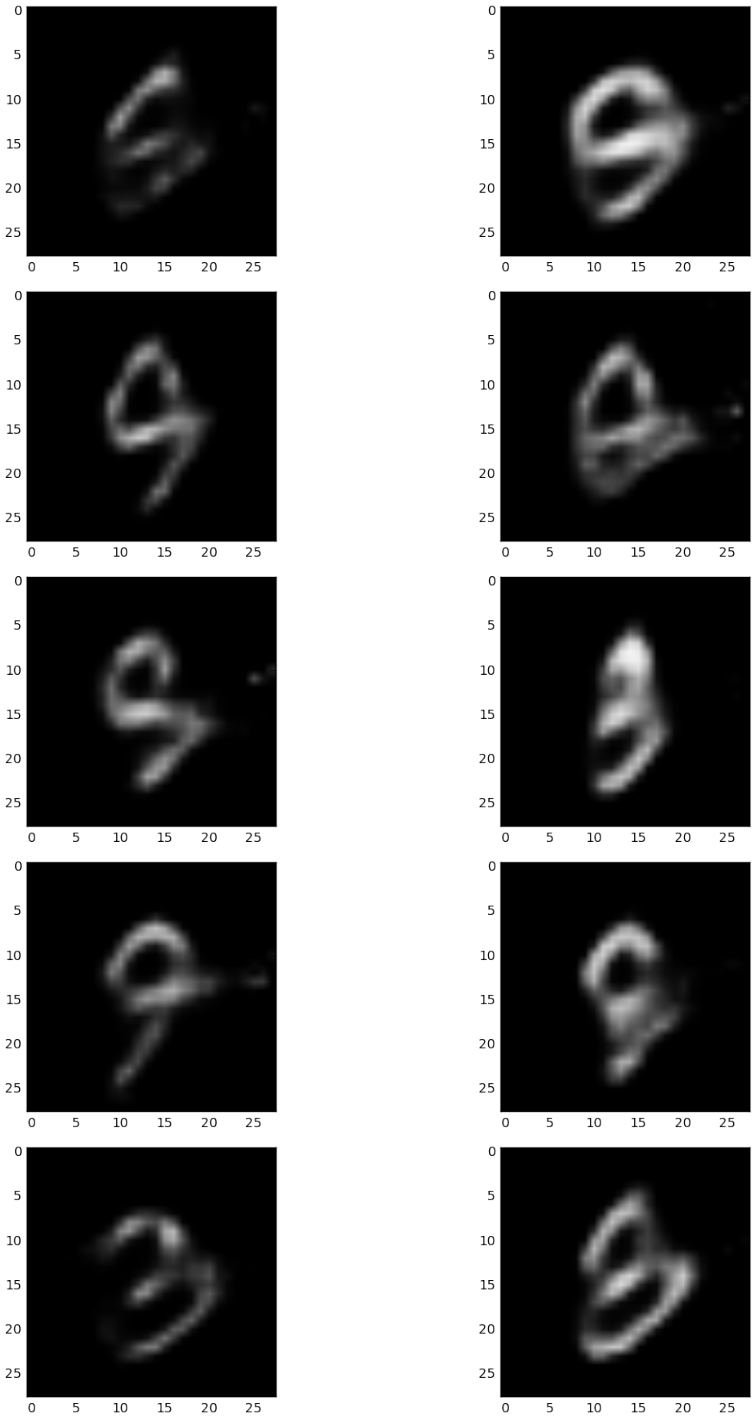

Figure 19: Laplace VAE patterns vs latent dimensions

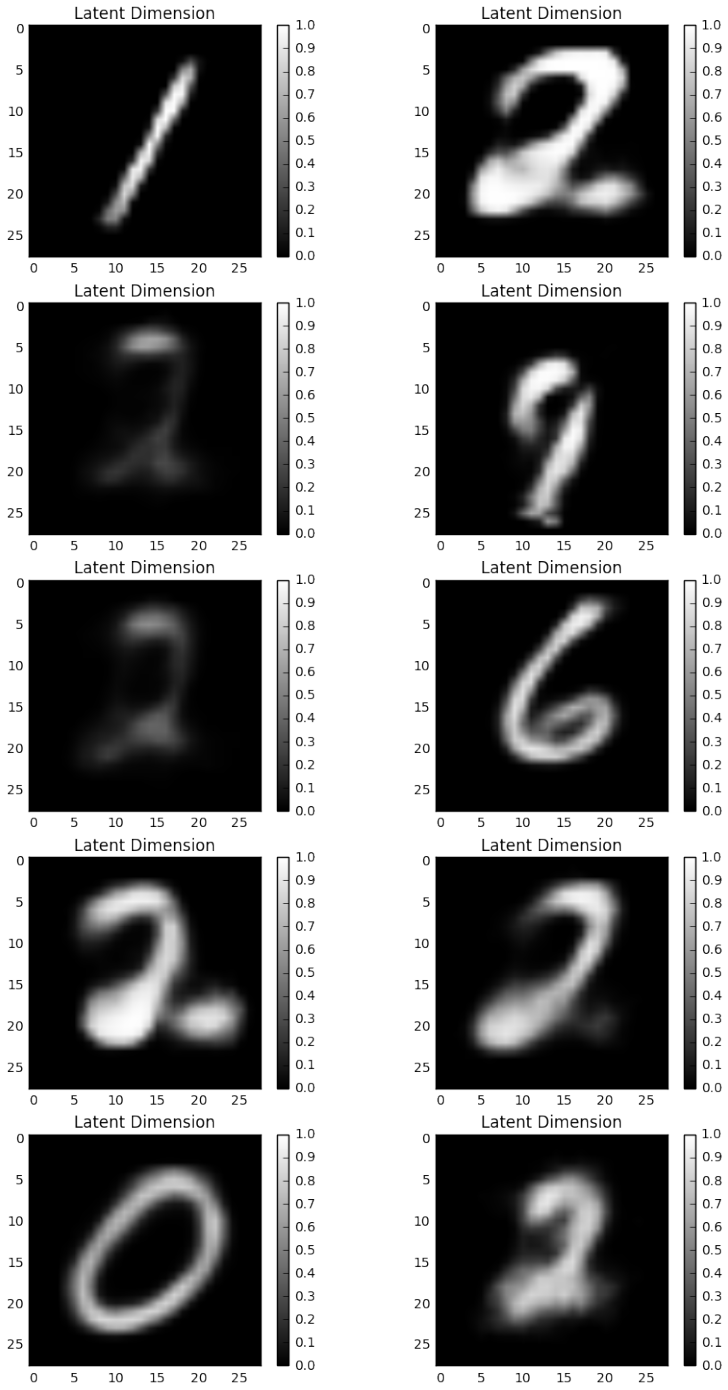

Figure 20: Cauchy VAE patterns vs latent dimensions

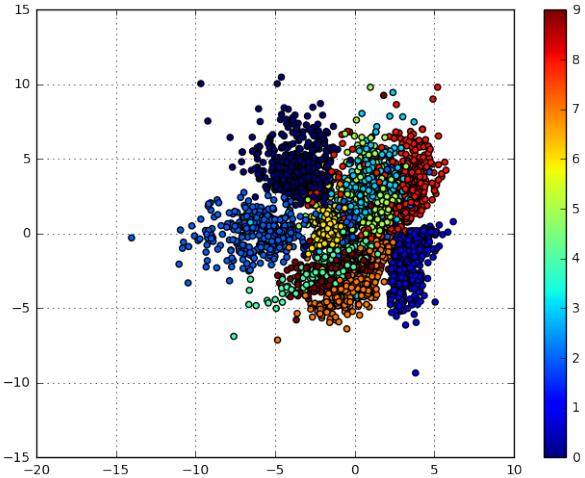

Figure 21: Normal 2D VAE latent spread

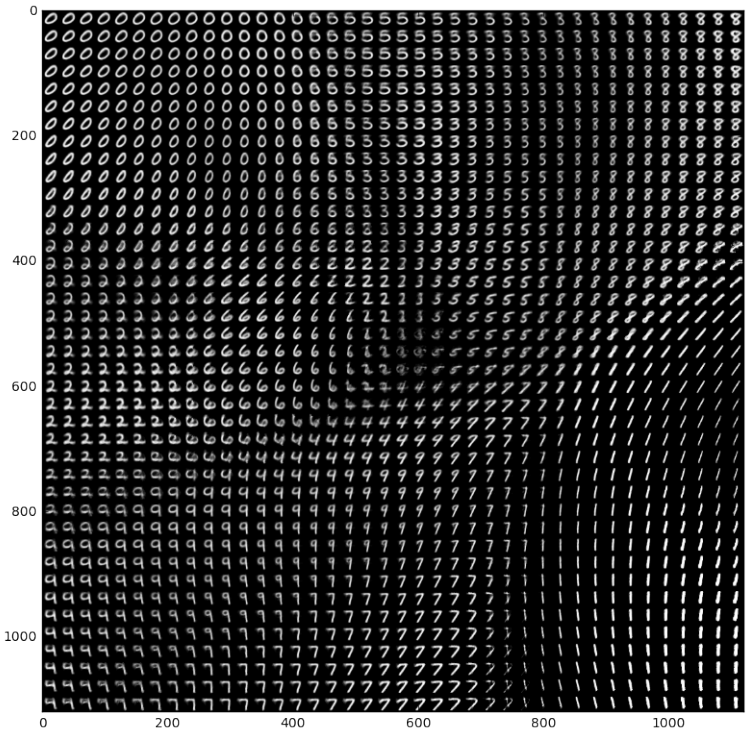

Figure 22: Normal 2D VAE latent grid reconstruction

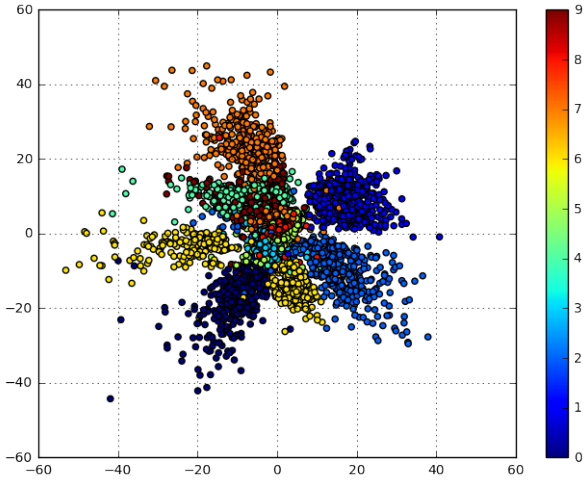

Figure 23: Laplace 2D VAE latent spread

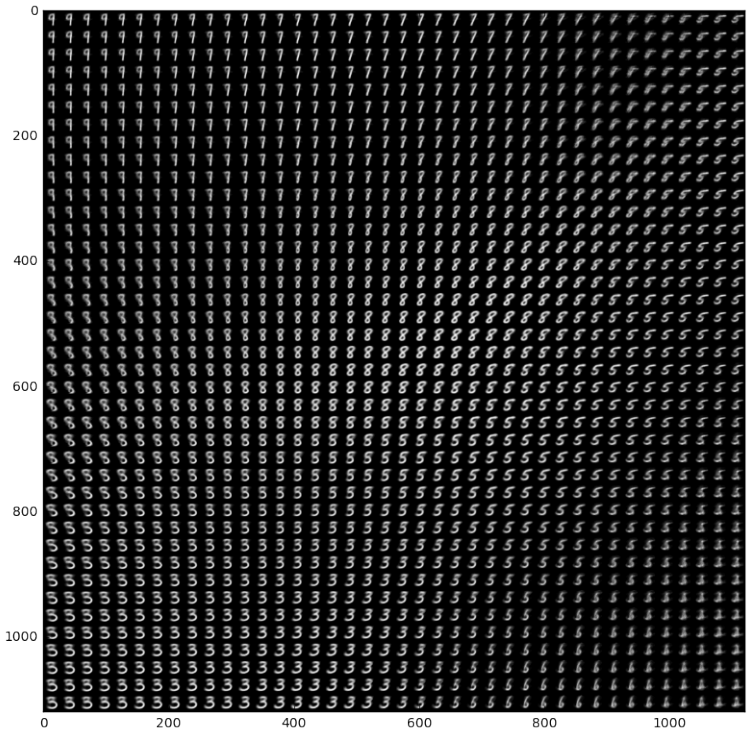

Figure 24: Laplce 2D VAE latent grid reconstruction

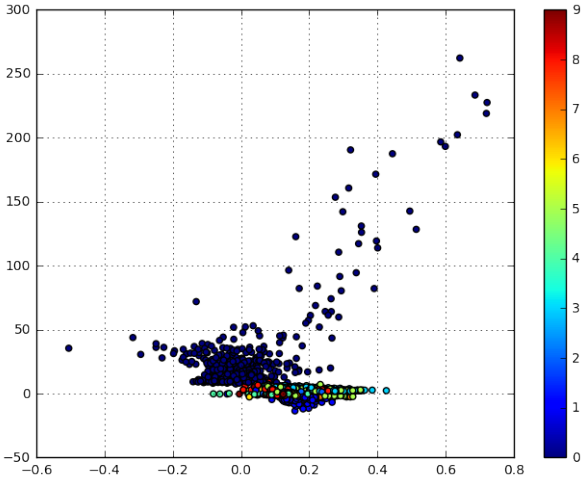

Figure 25: Cauchy 2D VAE latent spread

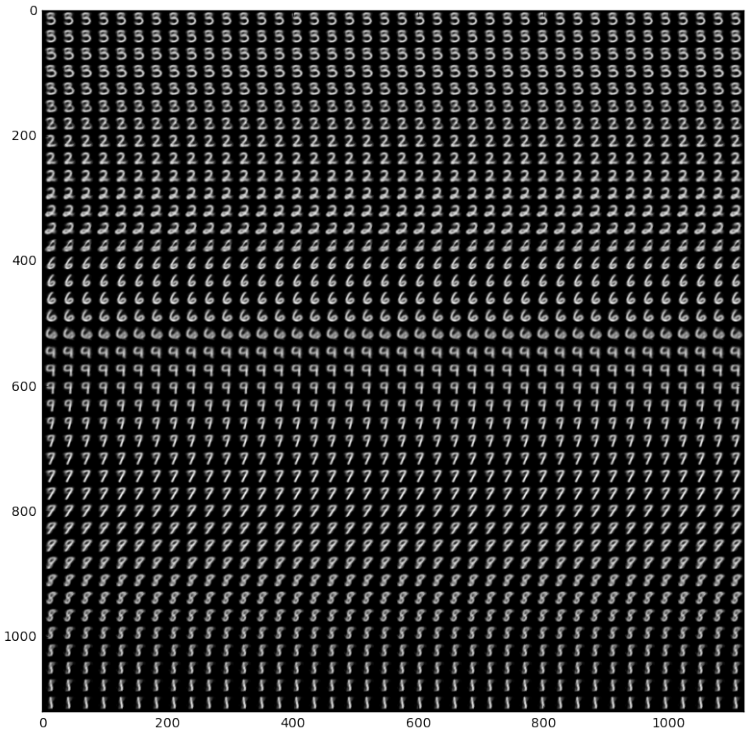

Figure 26: Cauchy 2D VAE latent grid reconstruction

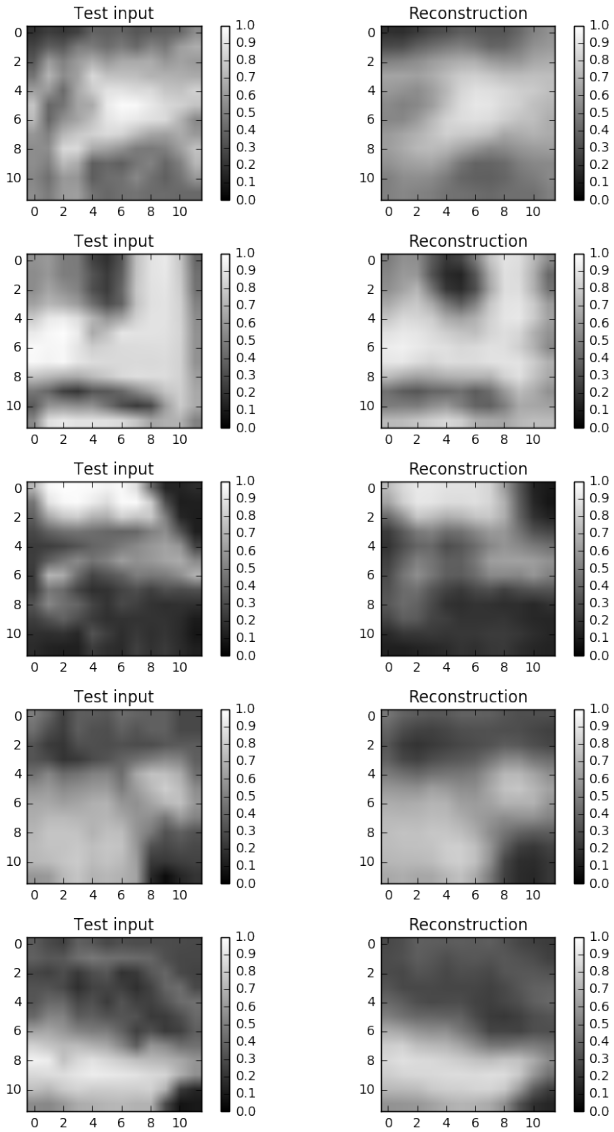

Figure 27: Normal VAE reconstructions

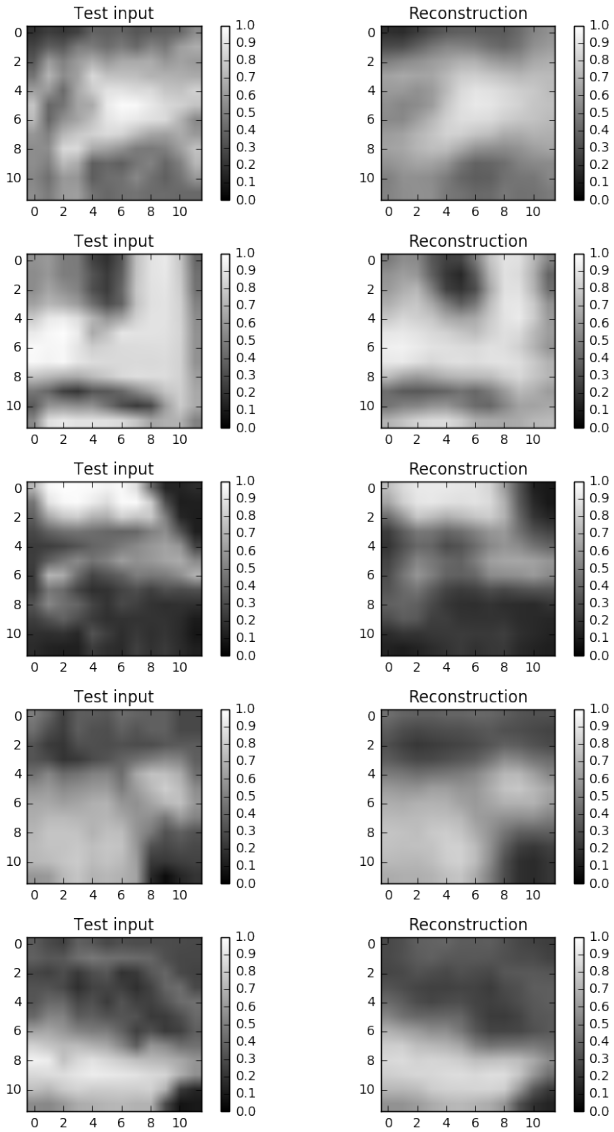

Figure 28: Laplace VAE reconstructions

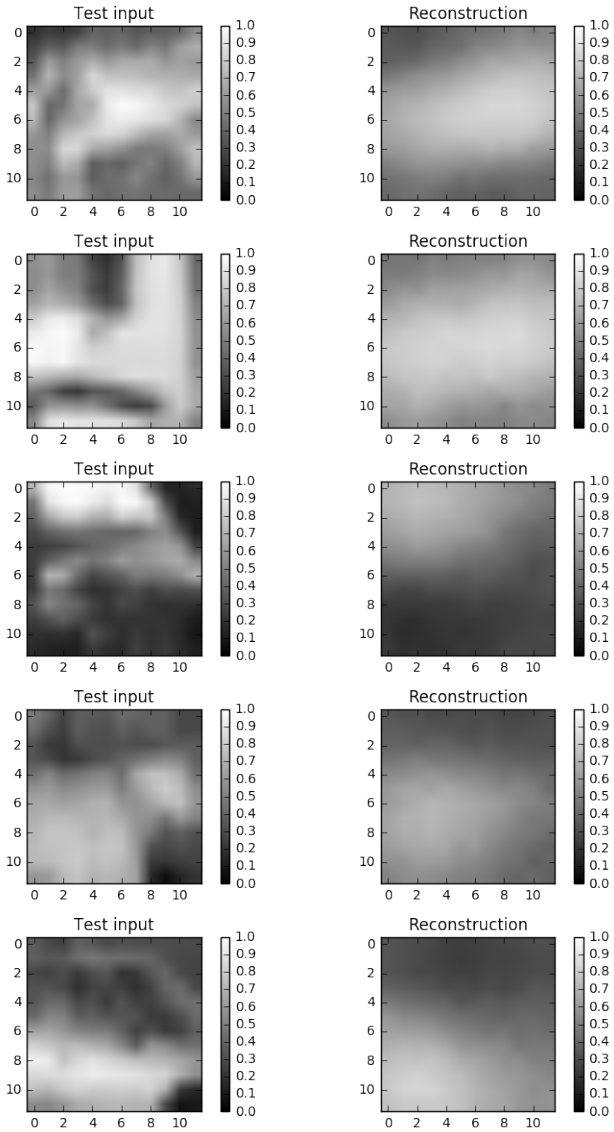

Figure 29: Cauchy VAE reconstructions

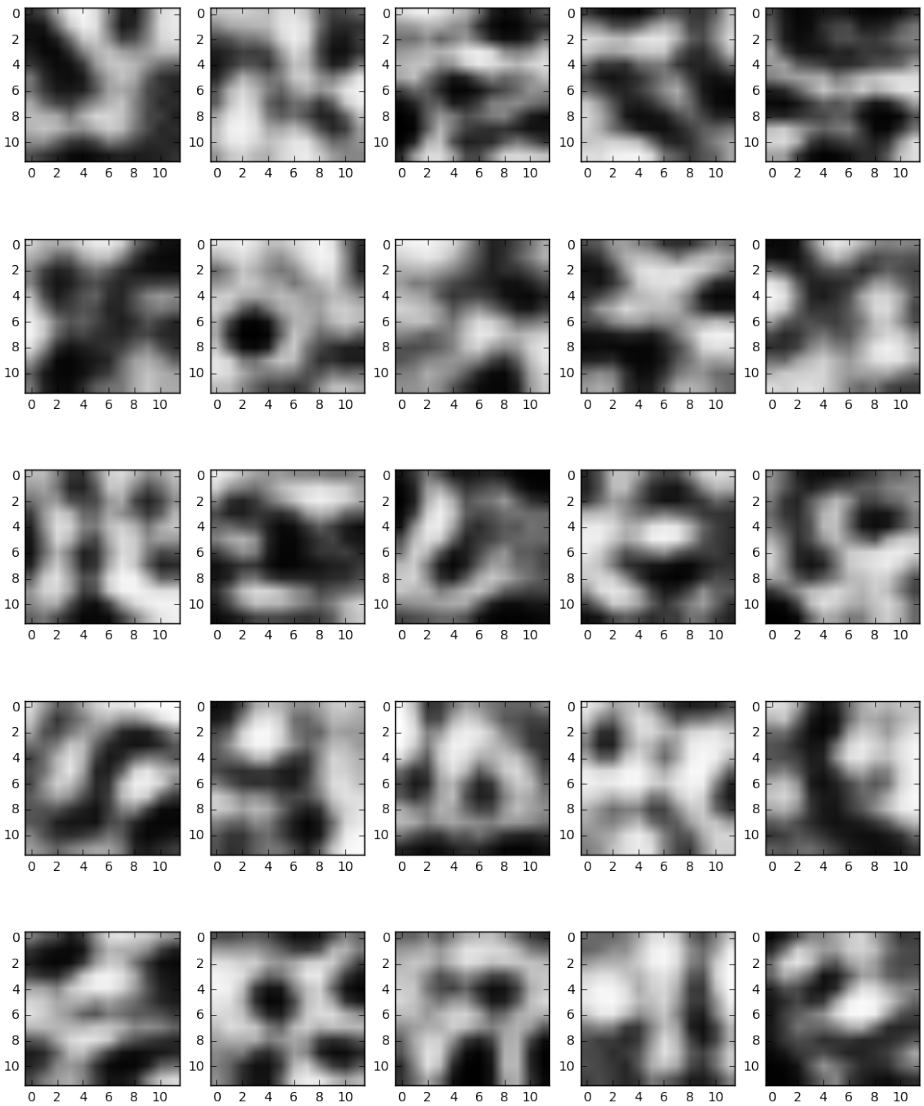

Figure 30: Normal VAE patterns vslatent dimensions

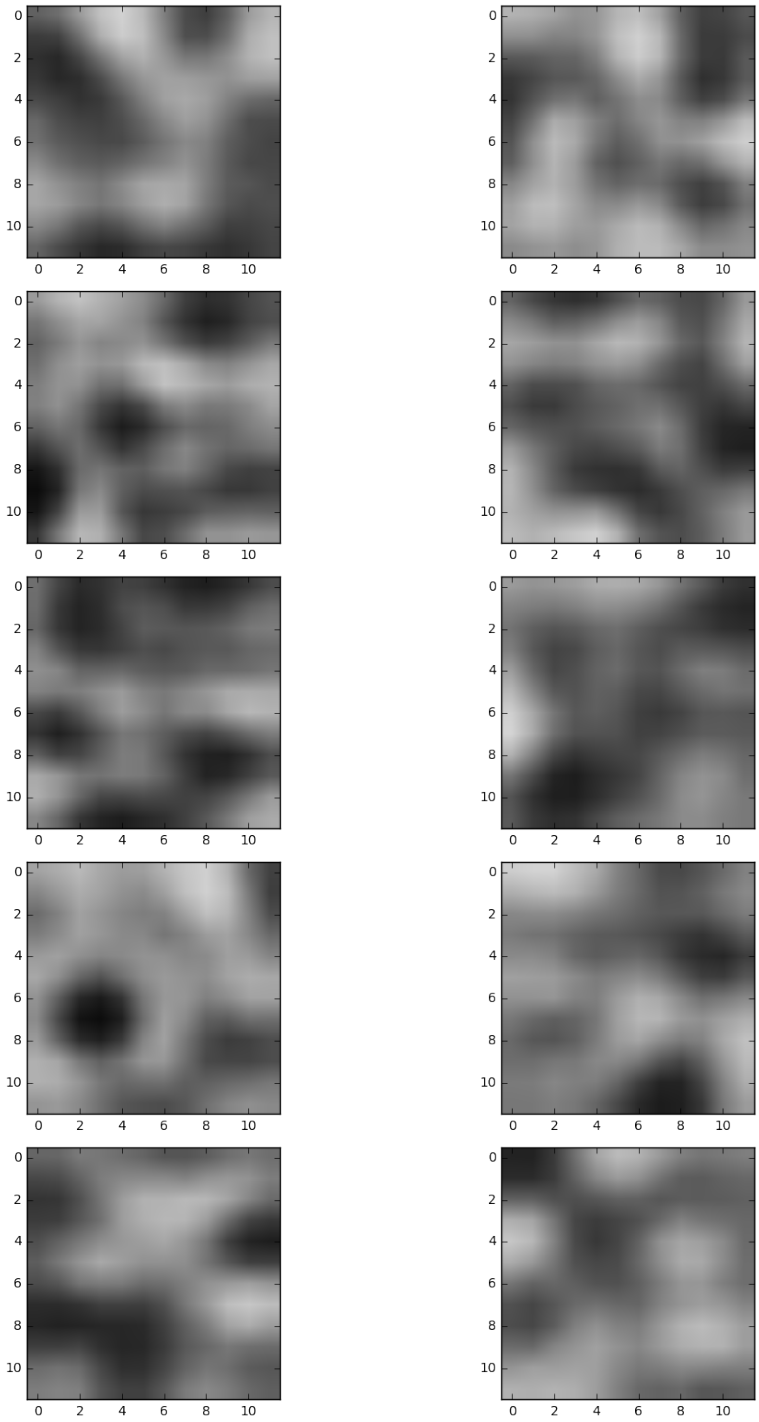

Figure 31: Laplace VAE patterns vs latent dimensions

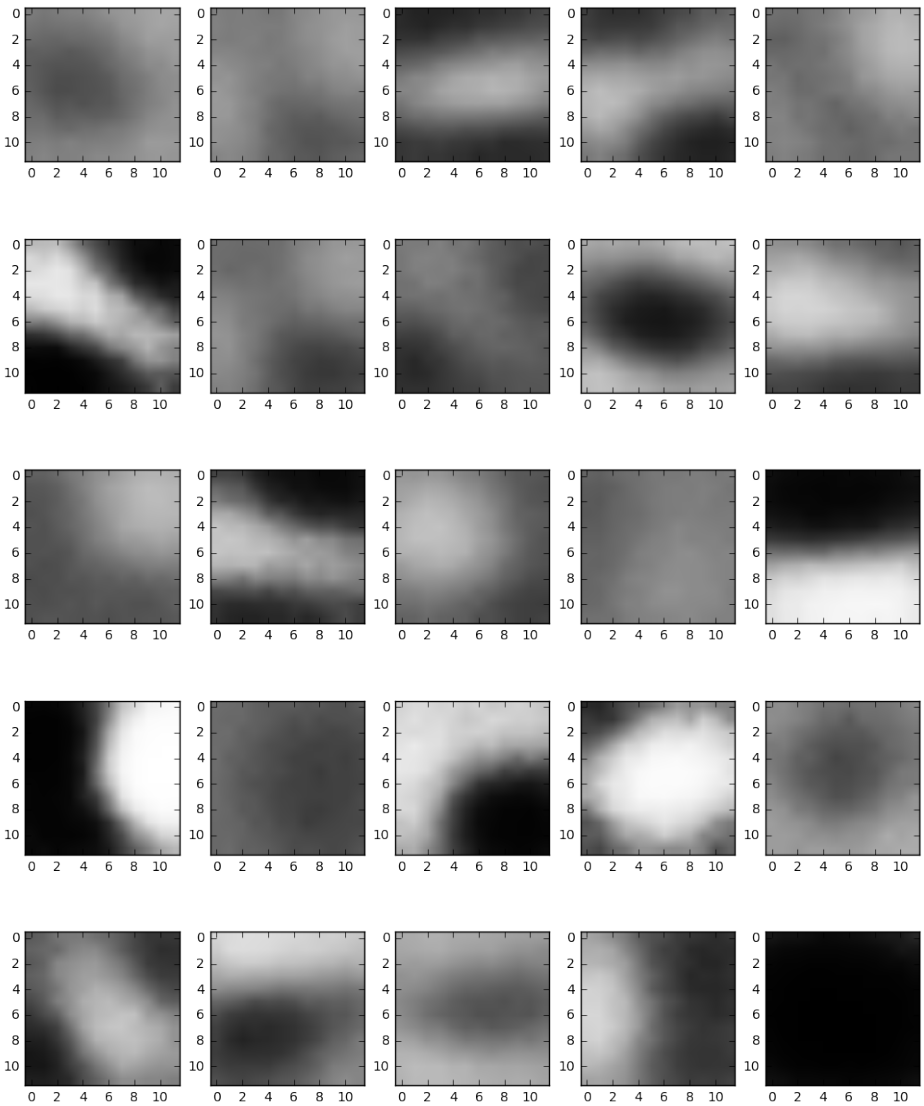

Figure 32: Cauchy VAE patterns vs latent dimensions