UNIVERSITY OF CALIFORNIA

Los Angeles

A 2D example study of Deep FRAME model

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Yaxuan Zhu

2019

ABSTRACT OF THE THESIS

A 2D example study
of Deep FRAME model

by

Yaxuan Zhu
Master of Science in Computer Science
University of California, Los Angeles, 2019
Professor Song-chun Zhu, Chair

In this work, we use a 2D to study many aspects of Deep FRAME model. We first do visualization on the training process, showing how the fitted probability distribution evolves during training, how the model captures different modes and how these results are influenced by the choice of prior distributions and activation functions. We then study the activation pattern of the learned network and the corresponding partition of the input space. Based on the input space partition and statistics matching, we then compare the multi-layer model trained with SGD with the original one-layer model. Finally, we analyze the case in which we train the model with finite-MCMC sampling, showing the difference between fitting the energy function and synthesizing samples.

The thesis of Yaxuan Zhu is approved.

Carlo Zaniolo

Yingnian Wu

Song-chun Zhu, Committee Chair

University of California, Los Angeles

2019

To my father and mother . . .

who always understand me and support me

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Recently, deep neural network [11] has been applied in many fields such as image classification, image generation, object detection [10, 4, 14], etc. The Deep FRAME model is a kind of energy-based probability model, which defines a energy function using deep neural network. It can be viewed as a hierarchical version of FRMAE (Filters, Random field, And Maximum Entropy) model [19]. This model is proved to be very powerful and can be used to model complex distributions such as image distribution [1, 6, 15]. Fitting the model usually employs using MCMC sampling[18, 2].

It is very useful if we can form a deeper understanding of this model. Many questions, such as how the model captures those complex underlying distributions, how the fitted distribution evolves during training, why deep model works in capturing complex distributions are worthy answering. Besides, in previous research, we mainly focus on the model's ability for synthesizing data. Although some works [7] try to reveal the landscaping of the fitted energy, it's not quite clear whether there is a difference between fitting good energy function and synthesizing data, or more precisely, synthesizing the data using generator or finite-step MCMC. These questions are hard to be answered in the high-dimensional input space. In this work, we use a simple 2D setting to discover the answers of these questions and our results reveals many interesting properties that the Deep FRAME model has.

The contribution of this thesis is helping people to better understand the Deep FRAME model. We do this in many aspects:

1. Visualize the energy fitting process of Deep FRAME model.
2. Compare the influence of choice of prior distribution and activation function.
3. Show the activation pattern of learned Deep FRANE model and the space division by

the ReLU function.

4. Show the match statitics of Deep FRAME model.

5. Discuss the influence of using finite-step MCMC in the training process. Show the relationship between generating good samples and fitting good energies.

CHAPTER 2

Deep FRAME model

The FRAME model, proposed by Zhu, et.al [19] is an energy-based probability model. As shown in equation 2.1, the model tilts a reference distribution (e.g. Gaussian Distribution or Uniform Distribution) by an exponential term. In [19], the energy term $f_\theta(Y)$ is defined as the linear combination of features extracted by a set of predefined filters, e.g. Gabor Filters. With the help of deep neural network, Deep FRAME model generalize the original FRAME model into a hierarchical version. (Strictly speaking, the energy term should be defined as -$f_\theta(Y)$ or we call $f_\theta(Y)$ the negative energy. In this thesis, for simplicity, when we show the fitted energy funcitons, we directly show $f_\theta(Y)$ instead of explicitly specify it is the negative energy.) Here, given an input Y, $f_\theta(Y)$ is defined by a bottom-up neural network where θ denotes its parameters and $Z(\theta)$ is the normalization term.

$$p_\theta(Y) = \frac{1}{Z(\theta)} \exp[f_\theta(Y)] p_0(Y) \tag{2.1}$$

2.1 Training of Deep FRAME model

The training of the Deep FRAME model relies on Maximum Likelihood Estimation (MLE). Given a set of observed training example Y_i $i = 1, \ldots, n$, MLE seeks to maximize the log-likelihood function:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} log p_\theta(Y_i) \tag{2.2}$$

If the sample size n is large, the maximum likelihood estimator minimizes the Kullback-Leibler divergence from the data distribution P_{data} to the fitted distribution P_θ . The gradient

of $L(\theta)$ can be written as

$$L^{'}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial\theta}f_\theta(Y_i) - E_\theta[\frac{\partial}{\partial\theta}f_\theta(Y)] \tag{2.3}$$

This updating equation can be understood as matching the expected energy of the synthesized data and true data. If we defined energy as a linear combination of a series of filter responses, then taking gradient means we are matching the expected responses of these filters. In other words, we use these responses as sufficient statistics for discribing the underlying distribution.

In the case of high dimensional data such as images, the expectation term in 2.3 is analytically intractable and needs to be approximated by MCMC, such as Langevin dynamics [18, 2]. In this thesis, since we work on 2D space and the range of the input is limited to [-1, 1] (more details please refer to Chapter 3), instead of using MCMC sampling, we can estimate the expectation by dividing the whole input space into grids and approximate the probability distribution using the grid points. As long as our division of the input space is refined enough, our estimation is accurate. In Chapter 5, we compare the result using finite-MCMC sampling with grid points.

2.2 Input space partition of Deep FRAME model

Suppose we use ReLU activation[12, 3], then given an image Y, the output of the neural k in layer l, F_k^l, can be written as:

$$F_k^l = \delta_k^l * (\left\langle W_k^l, F^{(l-1)}\right\rangle + b_k^l) \tag{2.4}$$

$$\delta_k^l = 1(\left\langle W_k^l, F^(l-1)\right\rangle + b_k^l) \tag{2.5}$$

According to whether the linear combination of the features is greater than 0, the neurons in the lth layer partition the input space into many subspaces. In each subspace, the output of lth layer is linear to the input features of the l-1th layer. This space will be further partitioned by the l+1th layer and hierarchically the input space is partitioned into lots of different small cells. According to [16], if we choose uniform distribution as prior distribution,

then the input space is piecewise linear. On the other hand, if Gaussian prior is chosen, then input space is piecewise Gaussian.

CHAPTER 3

2D example study training on grid points

In this section, we talk about the 2D example study results for Deep FRAME model. Our inputs are 2D points lying in the square space, whose coordinates are limited to the range of [-1, 1]. For the observed data, we randomly samples 10k-40k points according to the distribution we want to fit. Note that in this section, in order to estimate the expectation, we divide the input space into 40k square grids (divide each axis into 200 intervals) and calculate expectation on these 40K grid points. In chapter 5, we will show the results using finite-MCMC to calculate the expectation. We use 1 to 5 hidden layers fully-connected neural network to carry on the experiments.

The report is organized as follows: In 3.1 we compare the result for fitting different distributions. We show the input space partition and compare the result using different structures. In 3.2 we show the training process and discuss how this process is influenced by the choice of activation functions and prior distributions. 3.3 shows how the modes are formed and 3.4 visualizes the activation patterns of neurons. Finally, in 3.5 we consider sampling with a generator and discuss the potential problem in this situation.

3.1 Fitting result for different distributions

Here we show the several fitted results, the distribution we fit include a single circle distribution, several dots distribution and more complex distribution getting from galaxy image (this distribution is converted from real galaxy image from the web according to the intensity). In all these experiments, we use a standard Normal Distribution as our prior distribution. (We will compare the effect using different distribution in section 3.2.) We show the results

of fitted distribution in Figure 3.1. In this figure, we show the original distribution, then the final fitted results we get and finally the input space partition.

From the result, first thing we can tell is that our model can capture the underlying distribution even in those complex case of galaxy image (case E and F). Then if we see the input space partition, we can tell that the cutting lines of the input space concentrate on the place of the observed points. We can see in all the 6 cases the input spaces are cut into smaller and finer cells around the place where the observed data locates. We think this effect is because the place where observed points locate usually has large variance and in order to capture these variances, the model pays more attention to describe these areas. Since the ReLU function bends the output space at the place where input equals to 0 (where the cutting lines locate), then it can create a difference between the 2 sides of the cutting line. Therefore, by putting the cutting lines at the place with high variance, the model can fit the underlying distribution better.

Besides, for each of the underlying distribution, we use two structures. Comparing A and B, C and D, E and F, we can see that adding more layers enables the model to capture finer structures of the underlying distribution. For example, in case E and F, when we use the 2-layer network, the model only captures the contour of the fitted distribution while after adding a new layer, finer structure can be seen. Also note that in case C and D, the 1-layer network and the 2-layer network actually have similar number of parameters, but the 2-layer network result is much clearer and concentrating, which may show that the hierarchy of the model helps it capture the underlaying distribution in a better way.

3.2 Study of the training process

In this section, we show how the model evolves from the prior distribution to the final fitted distribution. Furthermore, we compare the learning process under different prior distribution (uniform distribution, Gaussian distribution with different variances). We also compare 2 more different activation function with the original ReLU, which is softplus 3.1 and tanh 3.2. The softplus can be seen as a soft version of the ReLU function. It's differentiable at 0 and

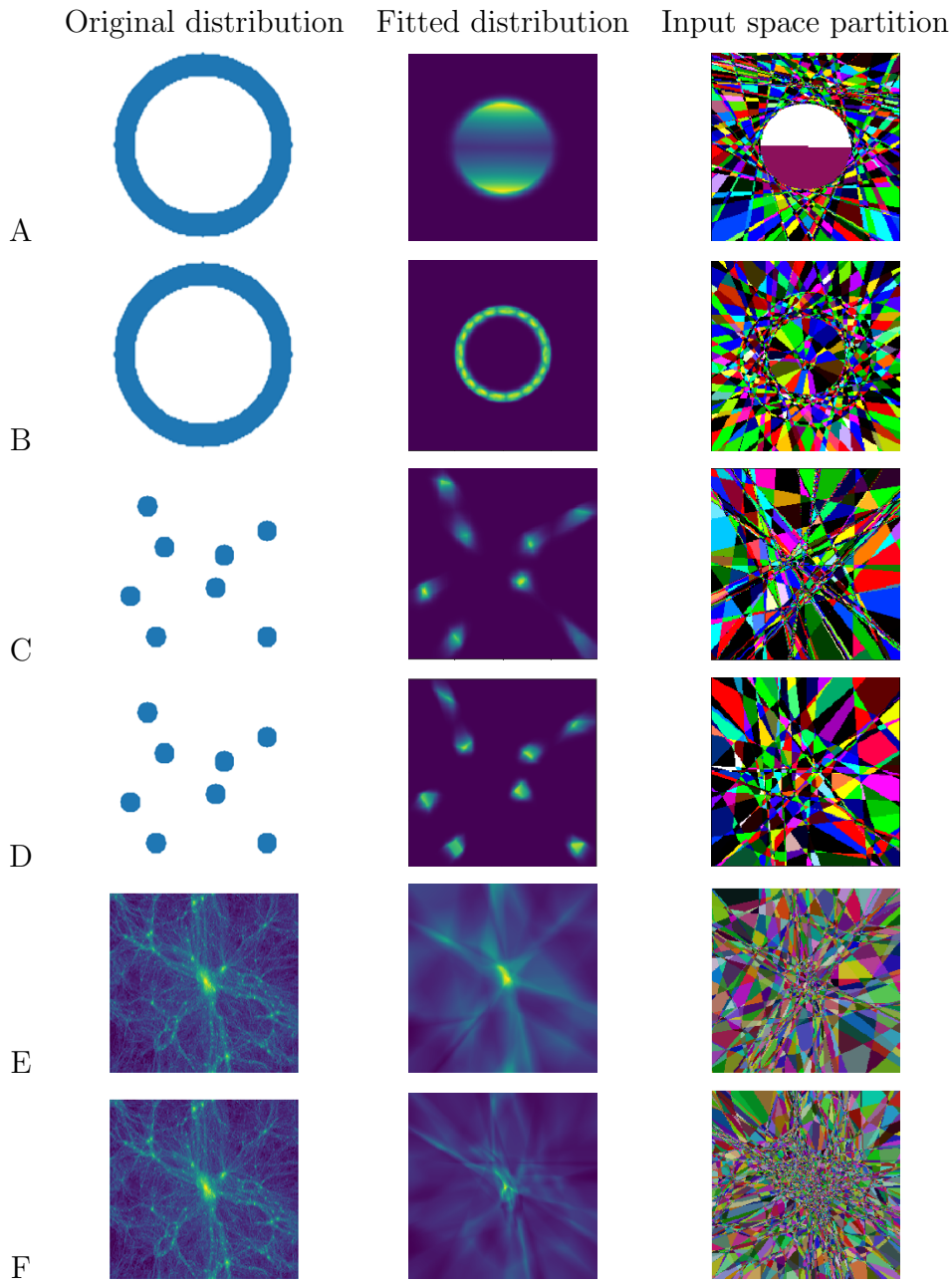


Figure 3.1: Fitted results for several distributions

A. Fitted ring distribution using 1 hidden-layer network (128 neurons) B. Fitted ring distribution using 2 hidden-layer network (32-32 neurons) C. Fitted star distribution using 1 hidden layer network (512 neurons) D. Fitted star distribution using 2 hidden layer network (32-16 neurons) E. Fitted galaxy distribution using 2 hidden layer network (32-64 neurons) F. Fitted galaxy distribution using 3 hidden layer network (32-64-128 neurons)

thus gives us smoother boundaries between pieces. Besides that, if we take derivative to the softplus function, we will get sigmoid in the backward propagation, which can be interpreted as the probability of detecting a certain pattern. Therefore, we get sigmoid function in the process of backward propagation. The tanh function is also a smooth function and it squashes those inputs that are either too big or too small. In our experiments, we use the same structure with 2 hidden layers (32-16 neurons) for all the settings. We also keep other hyper-parameters (learning rate, optimization settings, etc.) same across the experiments. The results of the training process are shown in Figure 3.2.

$$f(x) = \log(1 + \exp(x)) \tag{3.1}$$

$$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{3.2}$$

By comparing the results A and B, we can see that, according to the training process, there isn't much difference between using uniform prior or Gaussian prior with relatively large variance. The influence of the prior distribution is quickly surpassed by the fitted energy term. The model will expand its high probability region to cover the true distribution first and then gradually reduce the probability on those areas which do not have any observed data. And in this process, we can see very clear line cut. But the difference between B and C is very obvious. If we use much smaller variance in Gaussian distribution, then the mode forming process will be that the model first split its single mode into several small pieces, then these pieces will move towards the position of observed points. And in the whole process, we do not see any hard line cut from the probability space.

If we compare the results among different activation functions (D,E,F,G), we can see that using softplus and tanh do make the fitted energy function smoother. We do not observe the sharp line cuts in A and B. However, they will also increase the hardness for training the model, especially the softplus. As we can see, the softplus and tanh take longer time to converge. And if combined with small variance Gaussian prior, the model may fail to accurately capture the underlying distribution. (See the vary vague results in E.)

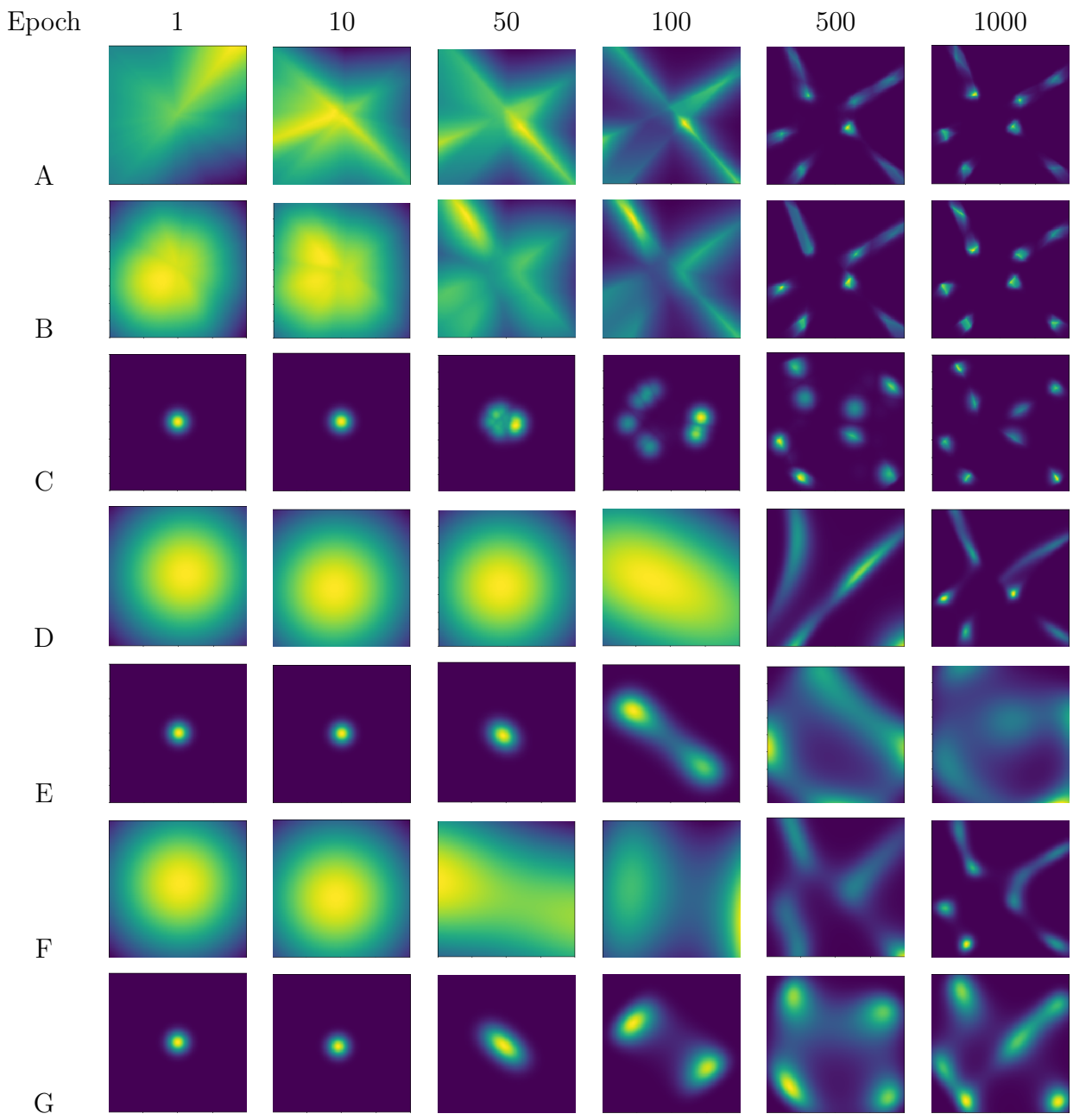
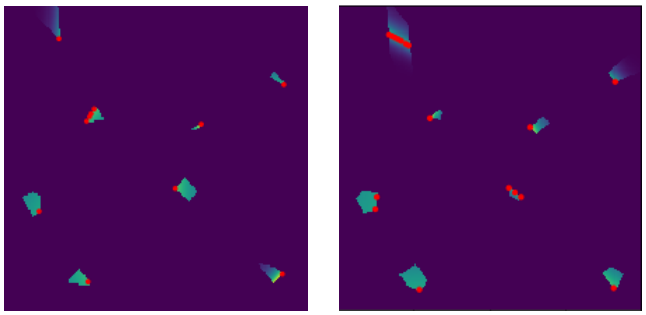


Figure 3.2: Training Process under different settings

A. Uniform prior distribution with ReLU activation function B. N(0, 1.0) Gaussian prior distribution with ReLU activation C. N(0, 0.1) Gaussian prior distribution with ReLU activation. D. N(0, 1.0) Gaussian prior distribution with softplus activation E. N(0, 0.1) Gaussian prior distribution with softplus activation F. N(0, 1.0) Gaussian prior distribution with tanh activation G. N(0, 0.1) Gaussian prior distribution with tanh activation



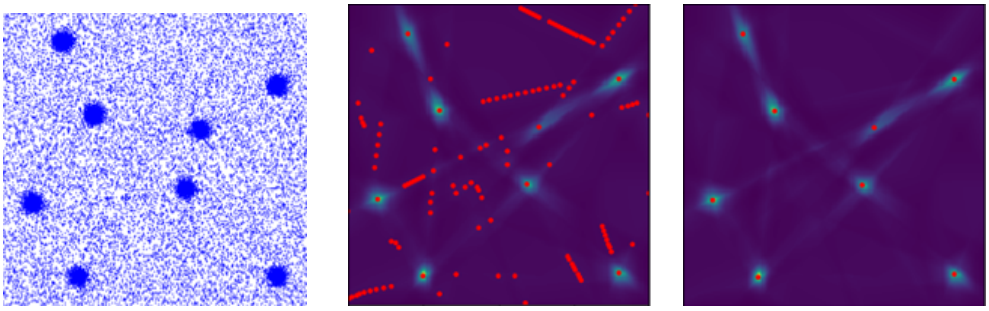
A. Uniform prior B. Normal prior

Figure 3.3: Captured modes and the pieces it lies in for star distribution
A. Fitted modes under Uniform prior distribution (piece-wise linear in the input space) B. Fitted modes under Normal (N(0, 1.0)) prior distribution (piece-wise Gaussian in input space)

3.3 Mode forming

To capture the variance in the underlying distribution, the fitted model will form different modes. We define the modes as those points whose value is bigger or equal than its neighbors. As we discussed before, under ReLU activation, the input space is piecewise Gaussian or piecewise linear. In Figure 3.3, we show the modes in its pieces for the star distribution we talked about above. We show the modes under uniform distribution, which in theory should give us piecewise linear energy function, as well as Normal prior distribution, which should form piecewise Gaussian energy function. Form the results, we can see in these 2 situations, almost all the modes fitted located on the boundaries of each pieces. This means that instead of using the property in the pieces, the model actually relies on the bending effect of the ReLU function to capture the modes. Therefore, in the sense of mode capturing, there is not much difference between using Uniform prior or Gaussian prior. This result is consistent with our observation in 3.2 that the training processes of the 2 settings are very similar. (Because they actually rely on the same mechanism to capture the modes.)

We further show another case (Figure 3.4) where the observed data contains several concentrated places (which we called textons) as well as a sparse uniform distribution across the whole spaces (which we called textures). Shown in Figure 3.4 (B and C), in this case,



A. Original distribution B. All modes C. Deep modes

Figure 3.4: Captured modes under the texton-texture situation

the model captures lots of shallow modes in the area of texture and deep modes in the area of texton. We can see many modes are aligned as lines. This is consistent with observation above that the modes are mainly formed by the bending of ReLU function.

3.4 Visualization of activation patterns

After we study the fitted distribution and the learning process, another question we want to study is how the neurons is activated inside a learned model. We first calculate the statistics of activated neuron in each layer. We find that given an input, usually about 50% of the neuron will be activated. This statistic is similar among different distributions, which means the model do not have a sparse property. (We show the histogram getting from the galaxy distribution in Figure 3.5 since the network used in this case has many layers. The results for other distributions are just similar.)

This result means that each point in the input space will activate lots of neurons. In geometry, the activation of each neuron corresponds to a line that cut the input space. As we show in Figure 3.1, the whole input space is cut into many small pieces. Given the points into a certain piece, although many neurons will be activated, the number of neurons that define the boundary of this piece is actually very limited. In Figure 3.6, we show several example pieces and the neurons that correspond to their boundaries. Here we use the 2 hidden layer network (32-16) trained on star distribution. We can see only less than 5 neurons really

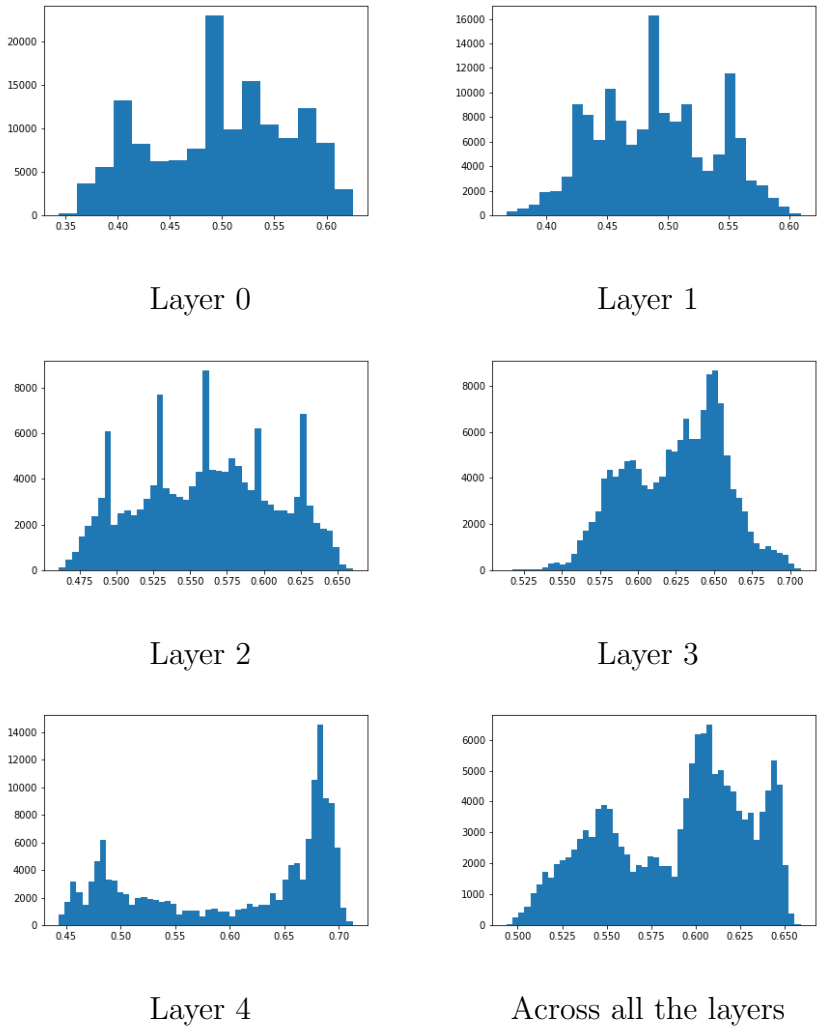


Figure 3.5: Activation distribution for each layer in a learned energy function Results come from a 5-layer network here trained on the galaxy distribution.

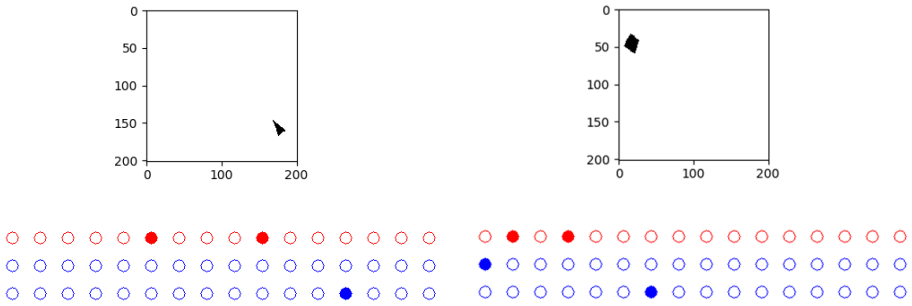
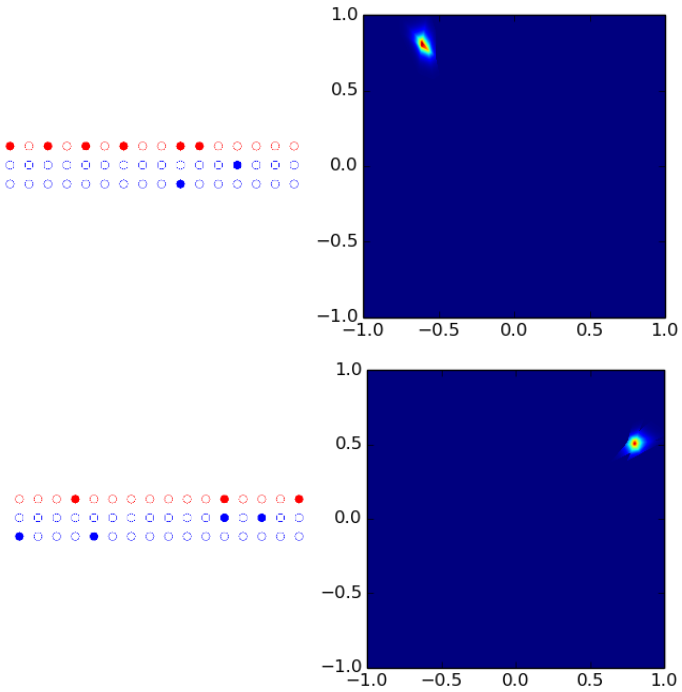
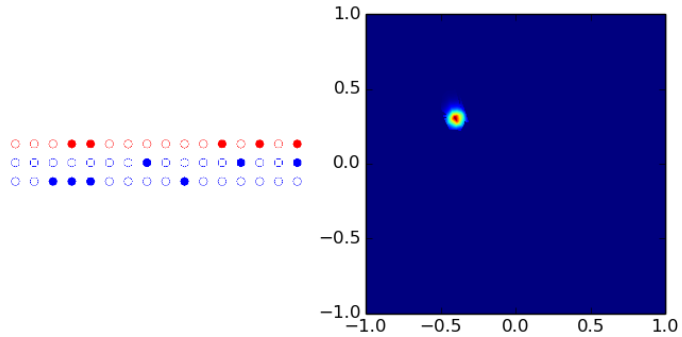


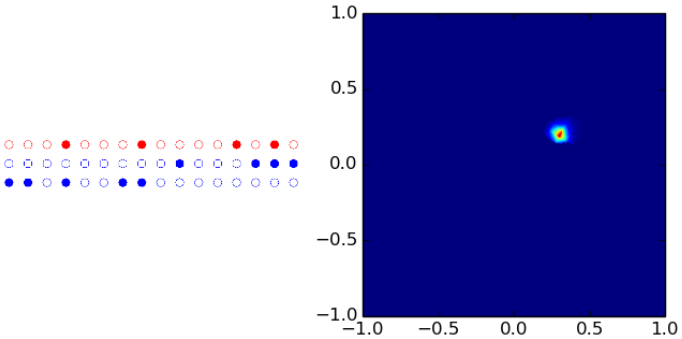
Figure 3.6: Pieces and corresponding boundary neurons

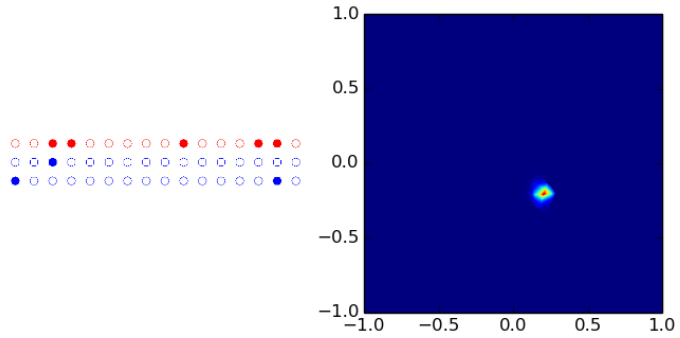
correspond to a certain piece, although many neurons are activated, which can be helpful in communicative learning.

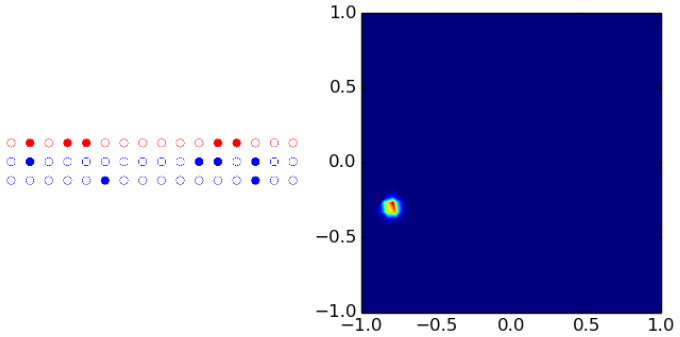
Besides describing each piece, we can also describe a group of pieces together. In Figure 3.7 we show all the 8 textons in texton-texture case. Each texton is constituted by many small pieces and their boundaries are defined by 4-13 neurons, which is much less than half of the total 48 neurons.

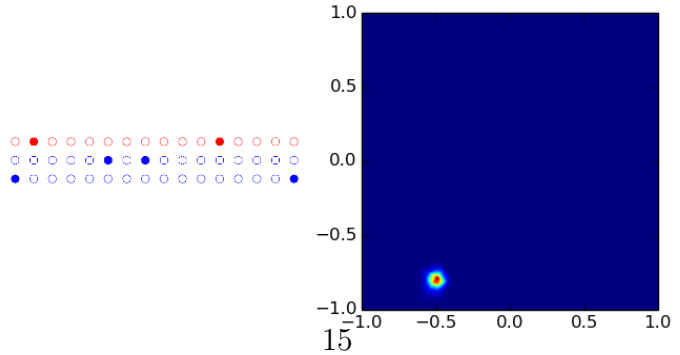



1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

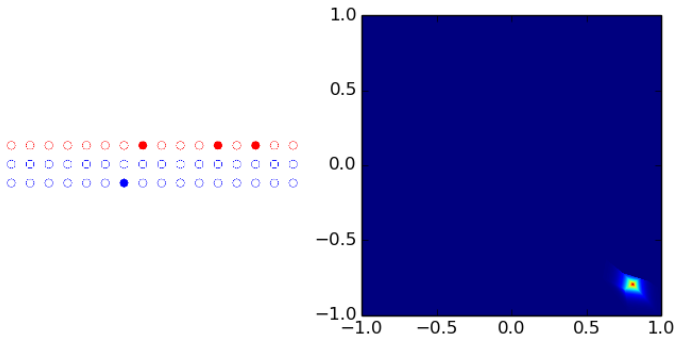


Figure 3.7: Textons and corresponding boundary neurons

CHAPTER 4

Comparing Deep FRAME model with pursuit-based model

In this chapter, we compare 2 different models: pursuing the neurons (with ReLU activation) one by one (which can only be done on 1-layer network); training the whole model (both one layer and multiple layers) using gradient based algorithm, which is the method used in Deep FRAME model. Based on these results, we show why multi-layer Deep FRAME model trained with SGD has the power to capture complex distributions.

4.1 ReLU-based pursuit result

In this section, we train one layer model using pursuit algorithm following the FRAME paper. Starting from a normal distribution, at each iteration, we select the filter with the biggest mismatch from our filter pool. We then update the coefficients of all the filters to enable the statistics match on all the selected filters. We build our filter pool by discreting all the orientations (i.e. weight) and positions (i.e. bias). Note that the model we use here is different from the original FRAME model in the sense that after selecting a certain filter, we do not ask the synthesized data to match all the histogram bins of that filter, instead, we only require the model to match the expectation of the ReLU activation. We show the results under 2 distributions. The results are shown in Figure 4.1 and Figure 4.2. In each figure, we show the fitted distribution, as well as the matching statistics.

In matching statistics graph, we show the true sample points using blue points. The background shows the cutting pieces and the read line shows the matching value for each neuron (i.e. where the expectation value lies for each neuron). Note that because we are

calculating the average of ReLU over all the data (the negative values are set to 0), the line may not go across either data cluster. (See the matching statistics for using 3 neurons in Figure 4.1).

From Figure 4.1, we can see that to fit the mixing Gaussian distribution, the model mainly choose filters from 2 orientations. The first 2 neurons limit the distribution to a line. Then since we start from a normal prior distribution, the probability tends to concentrate on the central part. To correct this, the model gradually adds more matching statistics in the middle. We see from neuron 3 to 8, the model learns to tell apart points into 2 clusters. Then after 8 neurons, the distribution are almost good enough so the following 2 neurons just make small adjustment. Similar things happen in Figure 4.2. The first 2 neurons build the basic cross shape, the 3rd one adjust its position and the rest of neurons gradually refine its shape.

We can compare the ReLU-based pursuit result with the original histogram-based pursuit in the sense of how many statistics number they need to match. In the original model, when we choose a filter, we match all the histogram bins of this filter. This is similar to we use a batch of paralell lines to partition the space. Whenever we introduce a new filter, we add a fixed number of lines to the partition(13 lines at a time, for example). On the other hand, in the ReLU case, we add one line a time. For those axises that are easy to fit, we only need one or two lines and more lines can be put to those hard axises. Therefore, we only need to match 8 numbers to fit the mixing Gaussian distribution.

4.2 Training neural network with gradient descent

4.2.1 One-layer neural network

Here, we show the 1 layer neural network result on the 2 aforementioned distributions to compare with the pursuit ones. The results are shown in Figure 4.3, 4.4. From the results we can see that in the neural network setting, the model still tries to match the statistics of the activation value from a certain filter. However, the chosen filters are different from the

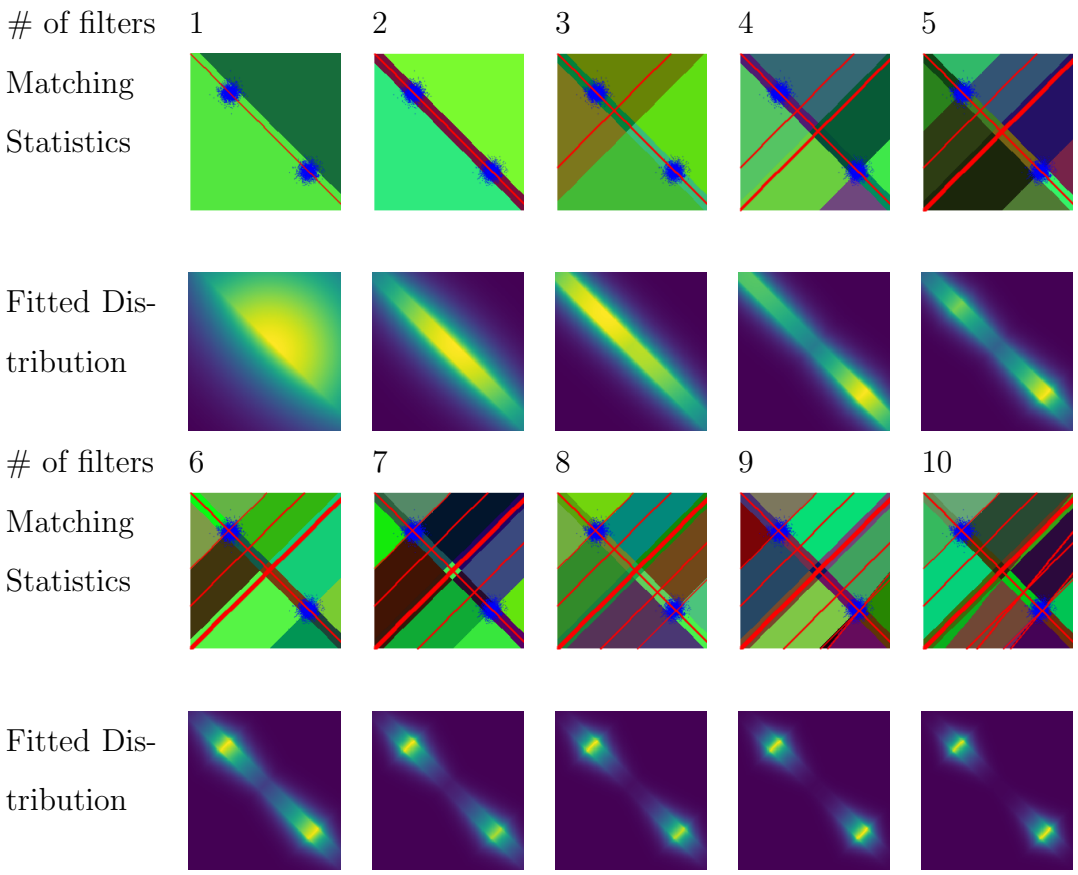


Figure 4.1: ReLU-based pursuit results for mixing Gaussian distributions

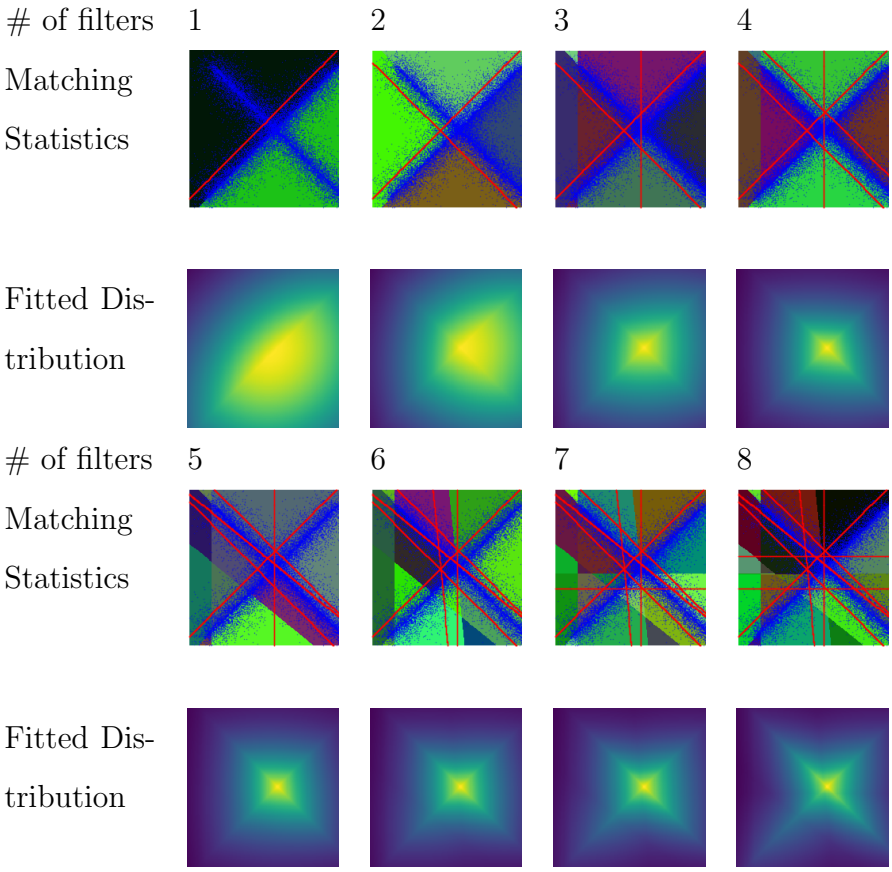


Figure 4.2: ReLU-based pursuit results for cross line distribution

pursuit case. Unlike in the pursuit situation where at each iteration we explictly choose the filter with the most mismatch, here we just try to modify the weight to make the selected features match. In other word, the model may end up reaching a match with a batch of not so informative filters. The model will converge to this local optimum because further changing weight will increase the loss. See the case of cross lines in Figure 4.4. When we use one or two neurons, we still find a line to match, but under this line, expectation of the prior distribution and the one the true distribution are not so different, therefore, the final fitted distribution is not so different from the prior Gaussian distribution. In this situation, it may not be approperate to say what is the exact number of neurons need to fit a distribution, because even under the same structure, different initializations will lead to different final filter choices. For example, we can see in the two-Gausssain distribution case, the result using 10 neurons is actually worse than using 9 neurons. This is because they choose different filters to match.

4.2.2 Multilayer neural network

However, the neural network model has power of stacking multiple layers. The higher layer neurons can have different orientations in different pieces, which means we do not only rely on straight lines as the matching statistics. We show some typical results in Figure 4.5 to illustrate this idea. For example, in the 7-3 case (2nd column), the model learns two rounds to match the mixing Gaussian centers. In 5-1 (3rd column), the model uses broken lines to fit the shape of the cross.

The power of multilayer model can be seen more clearly when we want to match highly irregular shapes such as texture. We fit the distirbution in Figure 4.6. We show the fitted results using both two-layer neural network model and one layer pursuit model with the same total number of neurons in Figure 4.7, 4.8.

We can see that if we solely rely on straight line as matching statistics (corresponding to 1 layer pursuit case). Even when we use 48 neurons, the shape of the fitted distribution is still far from the underlying one. The model only fits a triangle-like shape. However, if

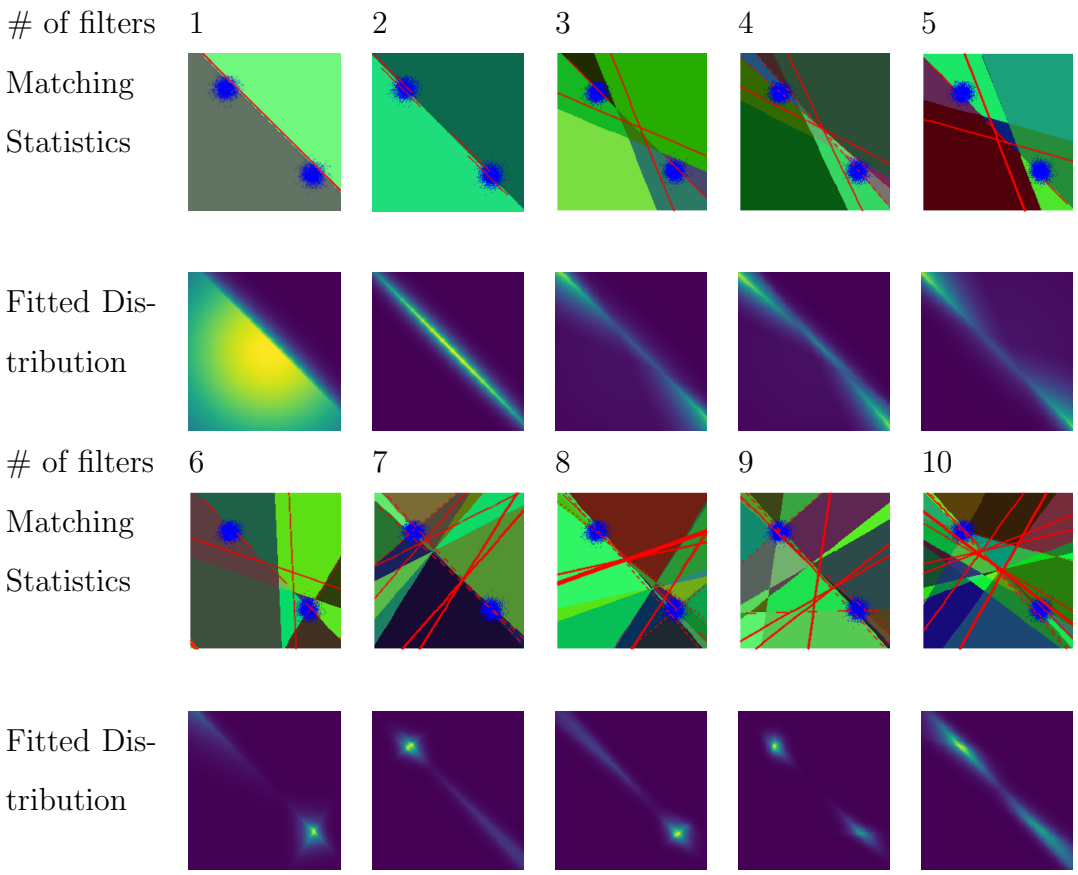


Figure 4.3: One-layer neural network results for mixing Gaussian distribution

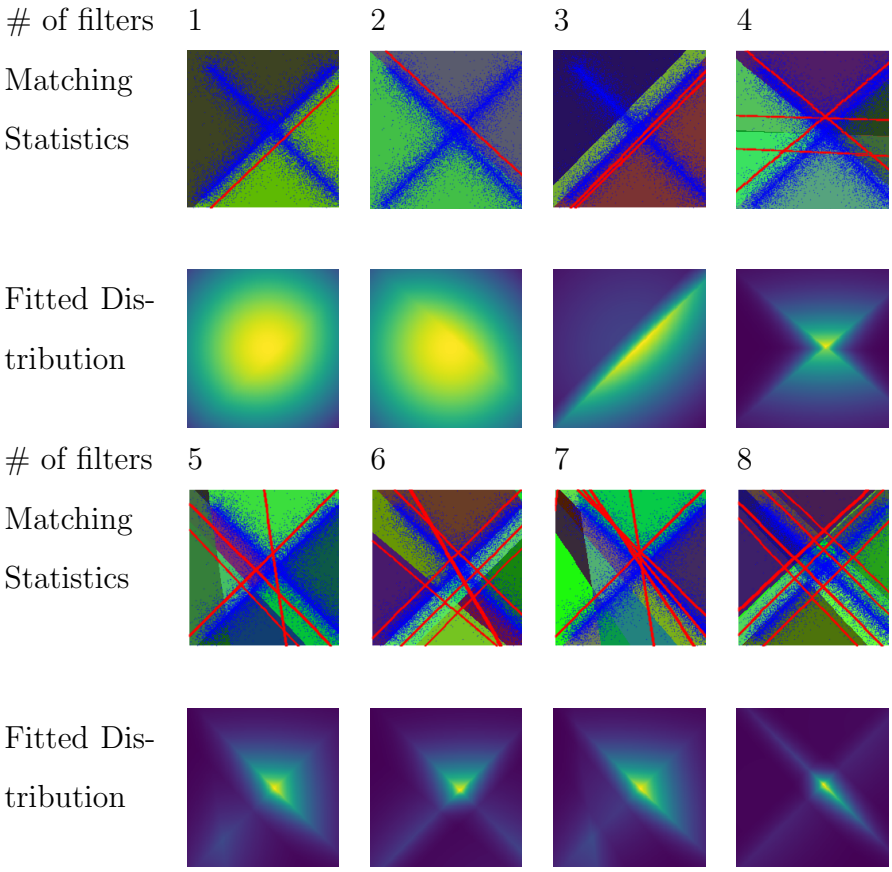


Figure 4.4: One-layer neural network results for cross lines distribution

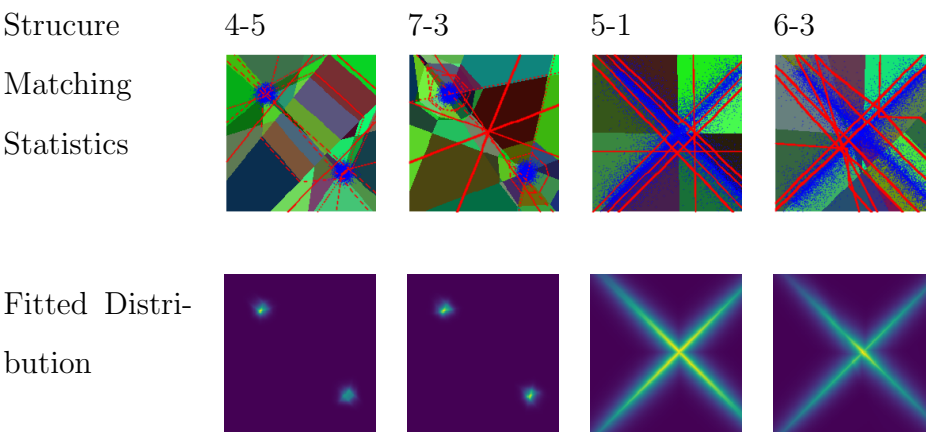


Figure 4.5: Two-layer neural network results

We show the structure of the 2 layer neural network on the top of the images, the first number is the number of neurons for the first layer while the second one is the number for second layer. The left two columns correspond to the two Gaussain distribution while the right 2 columns are the cross lines distribution.

we stack 2 layers neurons, we can see even with only 16 neurons(8-8), we can capture the rough shape of the distribution. When more neurons are added, more details are fitted. The neurons from lower layers are mainly responsible for cutting the space so the higher layer neurons can produce complex shapes for matching statistics (see those broken lines the model uses as matching statistics).

4.3 Summary

From these observations, we can reach the following conclusions in this chapter: 1. In ReLU-based pursuit we match one number a time. This allows us to put more numbers on those axises that are hard to depict.

2. One layer neural network trained with gradient descent is usually not as efficient as the pursuit method in the sense of how many numbers they match. This is because we do not explictly choose the neuron with the largest mismatch. And it will converge to local minimum depending on the initialization.

3. The neural network's power lies in that is can stack many layers so the higher layer neuron

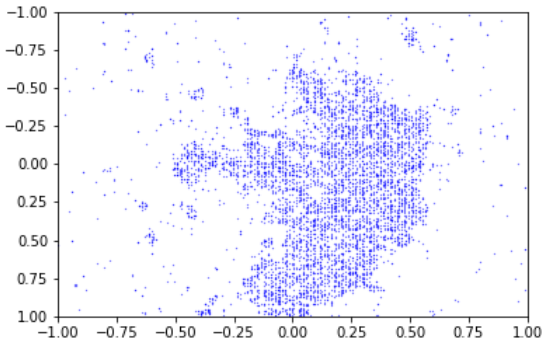


Figure 4.6: Texture distribution

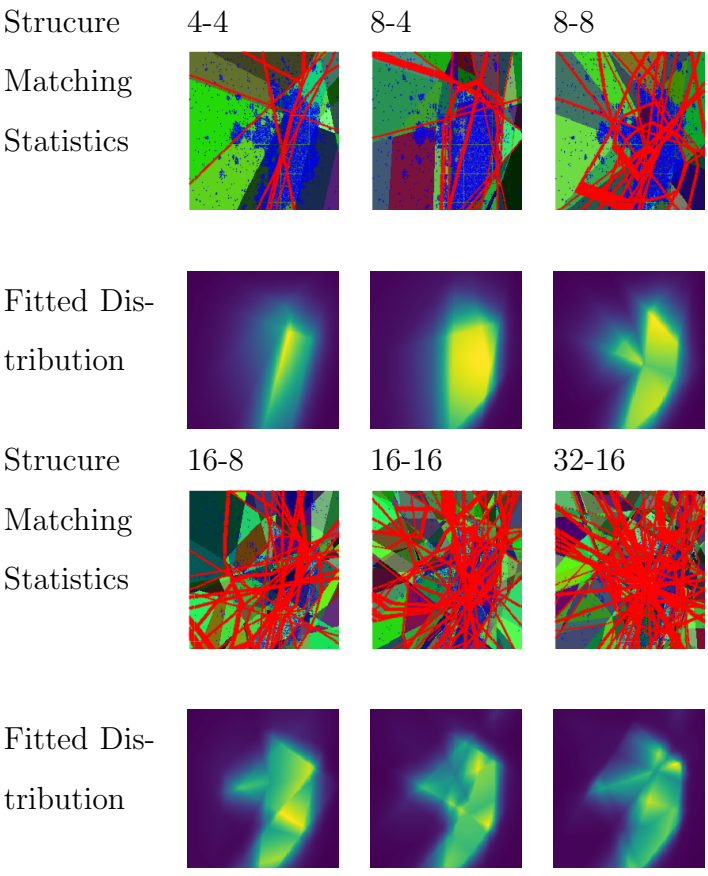


Figure 4.7: Two-layer neural network results

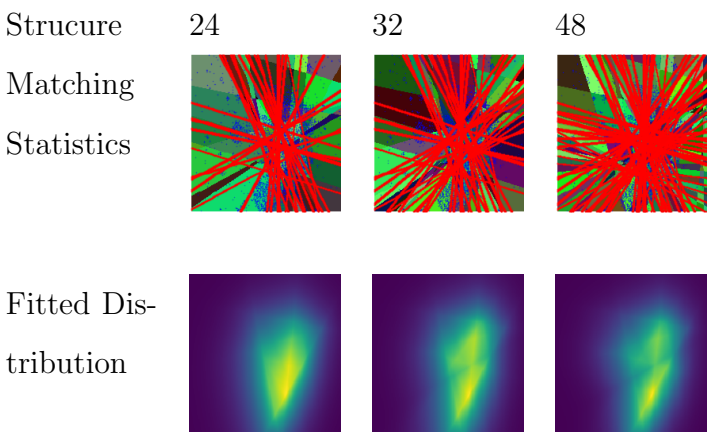


Figure 4.8: One-layer pursuit based method results

can produce arbitary shape for matching statistics. This can be much more efficient than the one-layer pursuit method especially when the underlaying distribution is complex.

CHAPTER 5

Sampling with finite-step MCMC

While in the previous chapters, we estimate the expectation term in 2.3 by dividing the input space into grid points, in this chapter, we will do our calculation on the samples from finite-step MCMC. More specifically, suppose at training epoch t - 1, we get a fitted probability distribution $p_{\theta_{t-1}}$. Then we use Langevin Dynamics [2, 18] to sample a batch of data from $p_{\theta_{t-1}}$. We then use these samples to estimate the expectation $E_{\theta_{t-1}}[\frac{\partial}{\partial \theta_{t-1}} f_{\theta_{t-1}}(Y)]$ and get updated parameter θ_t. In our experiments, we start the langevin sampling process points drawn from an uniform distribution. We compare different sampling steps and step size.

As one can imagine, unlike in the grid data situation where we can get a pretty accurate estimation for the expectation term as long as we use fine enough grid points, in MCMC case, the quality of our estimation largely depends on the quality of our sampling results. Usually, it can take quite a long run for the MCMC to converge, especially when we are dealing with highly complex data such as images. For example, to fit a well-formed energy on the symple digit image dataset MNIST [11], one may need thousands of steps update for MCMC to converge. But in general, we can only afford very limited MCMC steps (less than 100 steps) at each training iteration. (We refer this setting as finite-step MCMC). People apply different methods to facilitate MCMC to converge [1, 6, 15, 16]. Usually, using these kind of methods can give us pretty good sampling results at the end of the training. For example, we can synthesize images with high quality by sampling from the fitted probability distribution. And we usually take it for granted that being able to synthesize good samples (under finite-step MCMC) means we have fitted a good probability function. However, this may not be true. In this chapter, we discuss this question using our 2D example setting. In section 5.1, we show the fitted energy with the synthesized data distribution. In section 5.2,

we compare the fitted models trained using different MCMC steps. We mainly discuss the synthesizing ability of these models in this section. In section 5.3, we compare the models fitted by finite-step MCMC with the model fitted using grid points under a special case where true points only locate at very small areas. In section 5.4, we further illustrate our idea using a clearer 1D example.

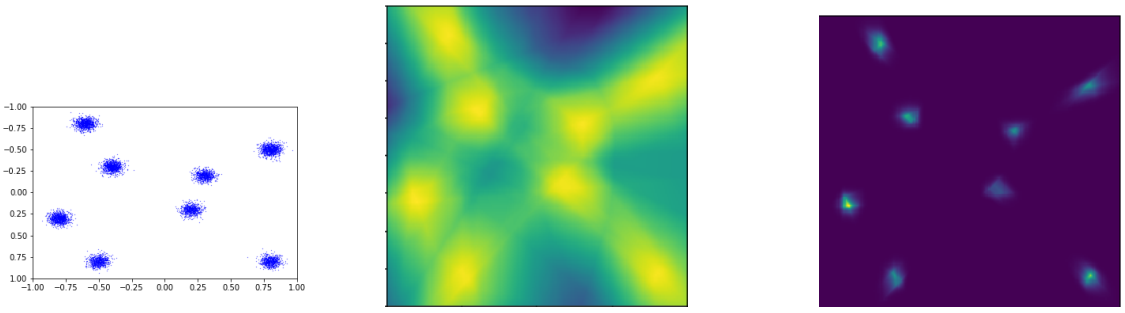
Note that as we stated earlier, the energy should be defined as -$f_\theta(Y)$. For simplicity, in this section, when we show the energy, we just show $f_\theta(Y)$ itself. One can understand that in this case, the area where have higher $f_\theta(Y)$ also should have higher probability density.

5.1 Fitted results using different MCMC steps

We start our discussion by showing the fitted result using finite-step MCMC. In these experiments, we use the star distribution as talked in previous chapter. This distribution is the composition of several Gaussian distribution centered at different places. We first show the true data points as well as the fitted results using grid points in Figure 5.1. Similar to our previous results, the model fitted with grid points can capture the underlying probability very well.

Then we show the results using finite-step MCMC in Figure 5.2. In our experiments, we start with uniform distribution. We fix the step size to be 0.004 and try different number of steps (number of steps = 15, 25, 50, 100). After we train them, we synthesize data from these models using the same setting as their training time (i.e. starting from uniform distribution and do Langevin update with certain steps). As we can see, all the 4 models have the ability to synthesize data that roughly agree with the input data distribution. Although when the number of steps are small, clusters that are close to each other may not be easily told apart from each other (see the up-left corner of number of steps equal to 15 and 25), generally speaking, the models do capture the area where true data locate. One can imagine if the data are from a certain subset of real image space, then this results means that we can generate images that look real from our model.

However, if we look at the fitted energy and the corresponding probability function, we



A. True data points B. Fitted energy function C. Fitted probability distribution

Figure 5.1: Fitted result using grid data

can easily tell that they are far from being perfect. One can see that in each of the fitted probability distribution, there is only a very limited area that have high probability and all the other areas are dark. This is because in the fitted energy, instead of making all the modes eqaully high, the differences between different areas can be very big. Then if we take exponential of the fitted energy to calculate the probability, the difference is further amplified, making probability only fire at certain small places and the most places are suppressed to very low.

On the other hand, if we compare the fitted energies using finite-step MCMC with the energy from grid points, we can see that the finite-step MCMC energies are more vague while the energy trained from grid data have more distinct seperation among different modes. In the later on sections, we will further discuss this phenomenon. We think the model may choose to spread its energy into larger areas instead of concentrate them into certain clusters so that during MCMC sampling, the points can get more guidences in the path and that guarrantees we can generate good samples in limited number of sampling steps.

In summary, in this section, our experiments show that being able to generating good data do not guarrantee us to get a well-formed energy.

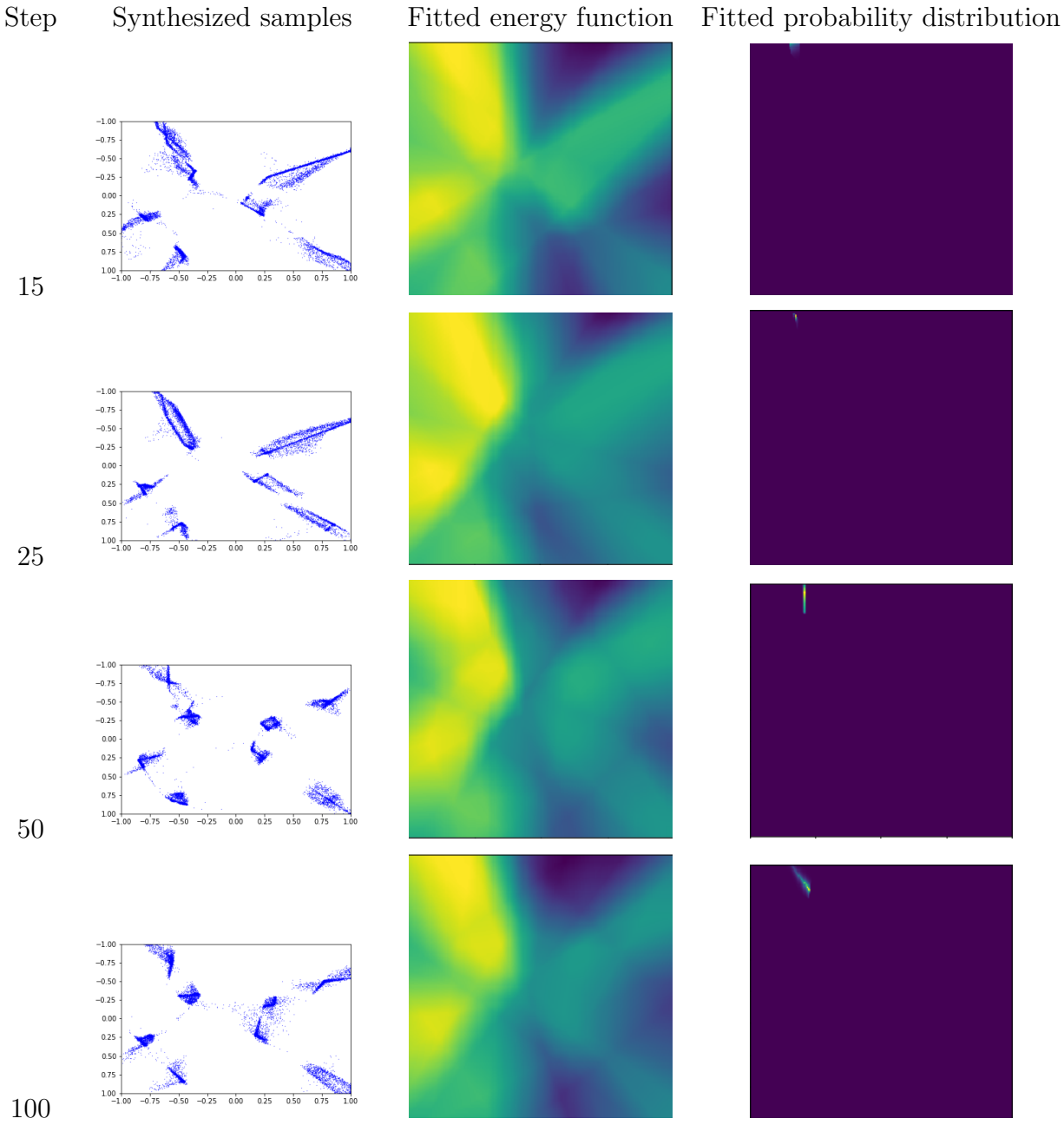


Figure 5.2: Fitted result using finite-step MCMC

5.2 Comparison among models using different number of sampling steps

One of our observations in section 5.2 is that after training, all the models can generate pretty good samples, even the model with 15 MCMC steps can relatively accrurately capture the position of the true data. Under this observation, we can naturally think of the question that whether this phenomenon happens because this problem is so very simple that 15 step of updating is enough for MCMC to converge (and using 100 steps is just a waster of time). A deeper question behind this is whether all the models try to converge to the same goal. Are all these models just different imperfect versions of the underlying data distribution? In this case, if we load a model trained with higher number of MCMC sampling steps (e.g. 100 steps) and sample it with less MCMC steps (e.g. 15 steps), the result should be not worse than the result we get from the model directly trained on this number of step. An opposite hypothesis to this is that actually every model specializes in its own settings. They should fit the energy to adpat the current settings. Therefore, a model trained use high number of MCMC steps may not fit the settings with less MCMC steps and its synthesis results may not be as good as the specialist trained with less MCMC steps.

To answer these questions, in Figure 5.3, we show the results of loading a model pretrained on different number of steps (number of steps = 15, 25, 50, 100) and synthesizing samples from them by another step number. Note that in these expeiments, we only change the number of steps but keep the step size to be the same (step size=0.004). As we can see, if we load a model and samples from it using less MCMC steps, then the synthesized result can not match the one from the model directly trained on this number of steps. Comparing the results samples with same number of MCMC steps (each column of figures), it is obvious to see that the synthesize results from a model pretrained with more MCMC steps do not concentrate enough.

This result supports our second hypothesis that the models actually try to adapt to the synthesizing settings. The fact that we can get good samples from a model trained on 15 MCMC steps is not because for the underlying true probability distribution, 15-step MCMC

is enough to converge , but should contribute to the fact that during the training, the model choose to shape the energy to make sure the 15-step MCMC can produce good results. Thus, the fitted goal for model under different MCMC settings can indeed be different. Those models trained with less MCMC steps may choose to spread their energy to wider areas and increase the gradient of their energies so that the MCMC can quickly converge to the areas that true samples locate. That's why when we load a model pretrained using less MCMC steps and test them using more steps, we will get more concentrating result than a pretrained more step model.

Another interesting result is that we load the model pretrained using grid data and samples using MCMC. Shown in Figure 5.4, we can see that in our training setting (step size = 0.004), the grid-point model fail to synthesize meaningful results. But if we increase the step size (step size = 0.04), then we can get good results. The requirement for higher step size means that the model fitted on grid data have smaller gradient on its landscape. This in return explain the fact why the models fitted using finite-step MCMC have a large energy difference (see Figure 5.2). The larger gradient actually facilitate the MCMC to converge under the training settings.

5.3 Discussion of fitting good energy and synthesizing good samples

In the previous sections, we have mentioned that the ability of getting good samples does not guarrantee we get a good energy function and model will try to adjust its energy function to adapt the sample settings. In this section, we further discuss this phenomenon. Consider a situation where the true samples concentrate on several small areas (see Figure 5.6 A). In this situation, if we fit a well-formed energy, the energy will only be non-zero at very limited areas around the center and leave most place with almost zero energy. Therefore, if we start from a place far from the energy center, there will not be any guidance for our Langevin update at first and the points just update according to the noise, which cause it to do the Brownian movement. This Brownian movement may continue for a long time until the points

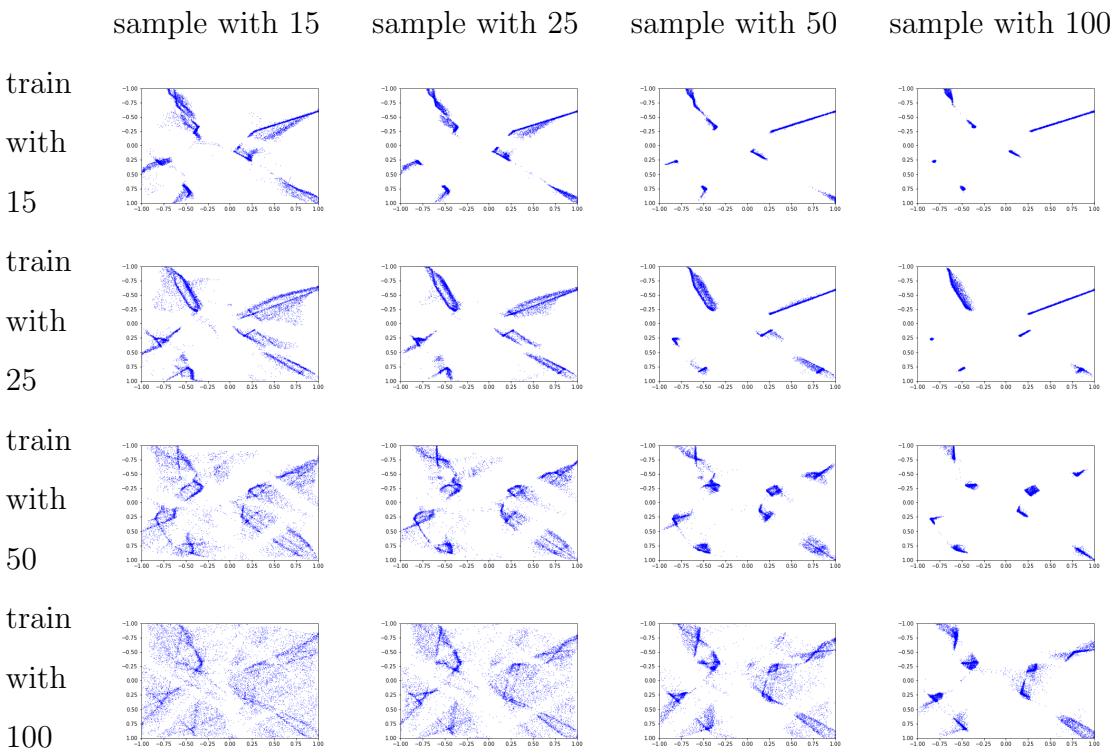


Figure 5.3: Loading models and synthesizing with different number of MCMC steps

We load the pretrained models of different number of sampling steps (number of steps = 15,25,50,100) and generate samples from each of them under various sampling steps. Each row represents a model trained on a certain number sampling step and each column is the number of sampling steps we actually use to generate the data.

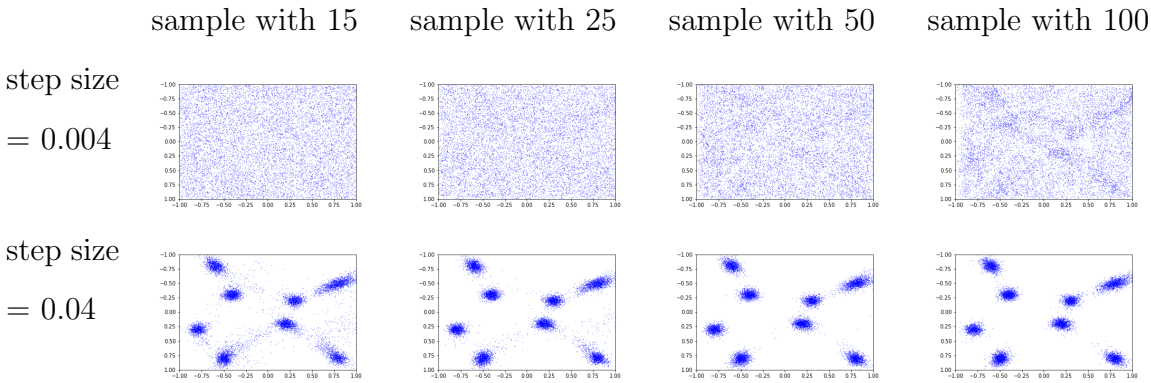


Figure 5.4: Synthesis results from model trained on grid data

happen to be captured by one of the energy peak (shown in Figure 5.5 A). In this case, if we limit our number of steps, it can be impossible for the MCMC to converge and there may be a lot of noise points in the synthesized data. One option may be we increase the step size, as what we have done in section 4.2 for the grid-point model. However, if we set the step size to be too big, then it may over-shot when the gradient is big, which also prevent the MCMC to converge. This means given a setting, we can only adjust our step size in some interval. And there may be the situation that we can not get a good synthesis result just by adjusting the step size. In this case, we actually need a specialist. This specialist may carefully shape its energy to facilitate the MCMC to converge. For example, it can choose to spread its energy into larger areas so that the points can be captured in a few Brownian steps (see Figure 5.5 B). Surely this will hurt the fitted energy function. But it enable us to get good synthesized samples only in finite MCMC steps. Remember our objective function here (Equation 2.3) is actually try to match the the statisitcs of synthesized data with true data. Then it is favorable for the model to focus on synthesizing good samples and sacrifice the quality of fitted energy.

We illustrate this idea using a experiment. We create a true distribution where the data locate in very small areas. We first use finite-step MCMC setting for training the model. Shown in Figure 5.6, the 2 finite-step MCMC model can synthesize good results (the points concentrate on only small areas). Then we load in the model trainined using the grid data, which is thought to be a good fit for the energy function. We adjust the MCMC settings and see its synthesis results. Shown in Figure 5.7. As we can see, when we choose small step size and small number of steps, the final synthesize result will spread a big area (see A and C). This is the case that may points are still doing Brownian Movement or just finishing it, so they have not come to the right place. But if we choose to big step size (see F), then our result is still vague. This is the case where the points over-shot at the place where the gradient is big and this prevent the MCMC from final converging. The best result we get here seems to happen when we use 100 steps sampling and set the step size to be 0.01 or 0.02 (case D, E). But if we compare these results with the synthesized results using finite-step MCMC (Figure 5.6 C), we can tell that the results from the latter model outperm those

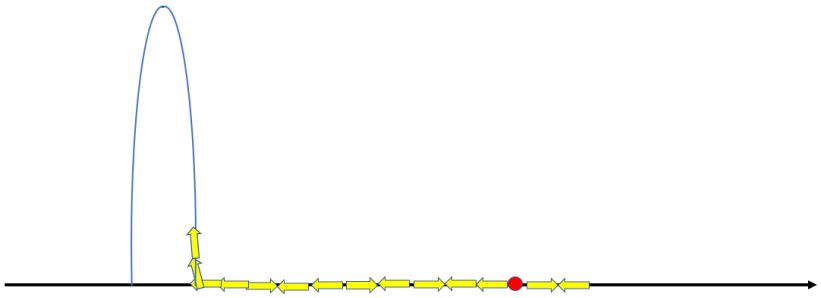
from the previous one. The finite-MCMC give us more compact results using same number of steps and smaller step size. But comparing the fitted probability in Figure 5.8, we can tell that the grid data model gives us much better fitted energy. There are 8 firing areas in the fitted probability, which correspond to 8 clusters of data. But the energy got from finite-step MCMC only contains one fired place on the up-left corner. If we see the fitted energy, we can tell that the grid points model give us more shape energy near the real data cluster while the MCMC model softly spread the energy across the whole input space.

Using the previous results, we show that the reason why being able to generate good samples do not guarrantee a well-formed energy is not purely due to the unperfect fitting, but actually because the model need to reshape the energy to get good synthesis result under the given MCMC setting.

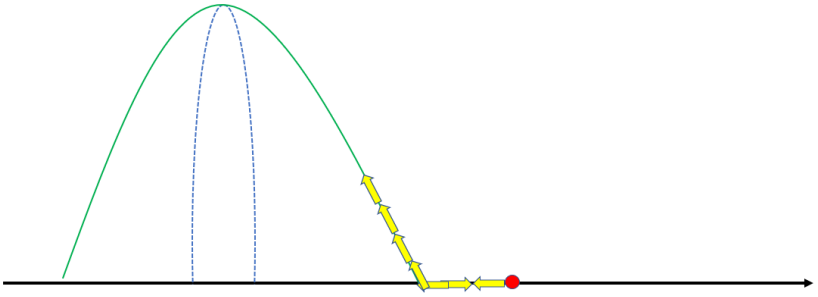
5.4 1D example study

In this section, we use an 1D example to illustrate our idea. We use a mixture of 3 Gaussian models centered as -0.8, -0.2, 0.65. When sampling the true data, we set the number of data getting from each Gaussian model as 3000, 2000, 3000. (So in total we have 8000 true data points.) Compare to the heatmap we used in our 2D example, in the 1D case, we can plot the synthesis points using histograms, which can give us a clearer impression about the distribution of generated data and true underlying distribution. (Note again that for energy, we directly plot f_θ here.)

In Figure 5.9 B, we show the generated samples using histogram (organge histogram).The result is got using 50-step MCMC, with a step size of 0.01. We smooth the histogram into density curve using kernel density estimation (orange line). And we also plot the curve for true distribution using green line for comparison. As we can see, the generated data distribution almost agree with the underlying true distribution. On the other hand, if we see the energy plot (strictly speaking, negative energy), we can see that the fitted energy is very different from the underlying energy. The fitted energy has much larger energy difference between the maximum value and lowest value, it also has a obvious difference between each

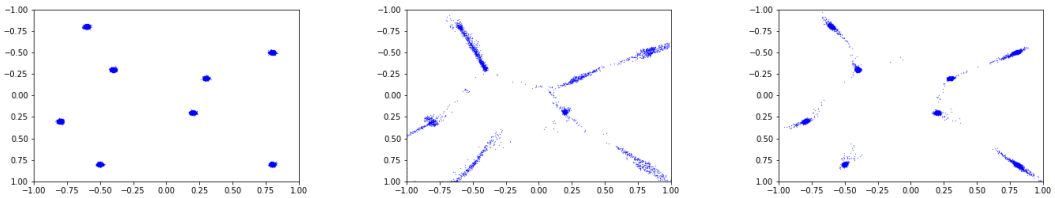


A. In a well-fitted energy, it may takes a long run for MCMC to converge.



B. If we limited the number of MCMC, then the model may spread its energy to facilitate convergence , although this may hurt the energy quality.

Figure 5.5: Illustration for the relationship of fitting energy and synthesizing samples in finite-step MCMC

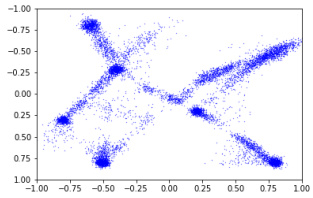


A. True samples

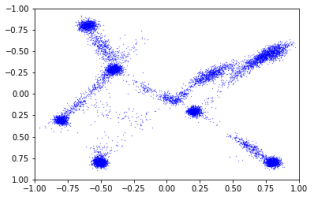
B. Synthesized data for 25 steps

C. Synthesized data for 100 steps

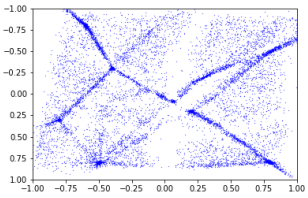
Figure 5.6: Samples from finite-step MCMC model trained on concentrating points We try a situation where the data distribution is compact (points concentrate on a few very small areas). We show the true samples and the samples getting from finite-step MCMC models. We set the step size to be 0.006 for all the MCMC models.



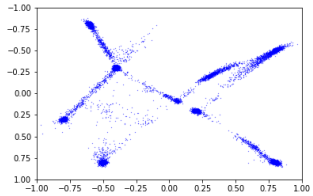
A. Number of step = 15, step size = 0.02



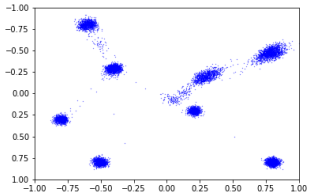
B. Number of step = 25, step size = 0.02



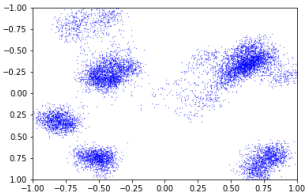
C. Number of step = 100, step size = 0.004



D. Number of step = 100, step size = 0.01



E. Number of step = 100, step size = 0.02



F. Number of step = 100, step size = 0.04

Figure 5.7: Samples from grid points model trained on concentrating points
We load the model pretrained on grid data and synthesize points from it using different MCMC settings.

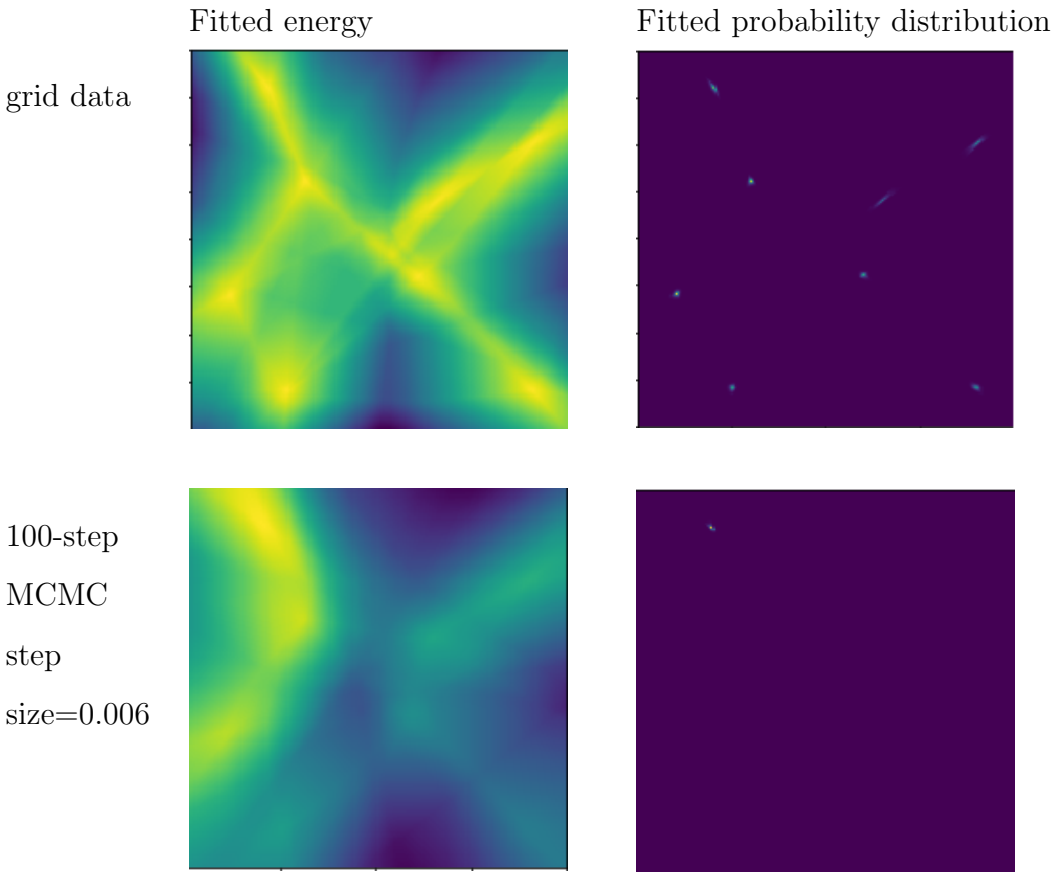
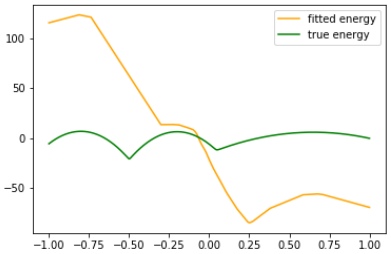
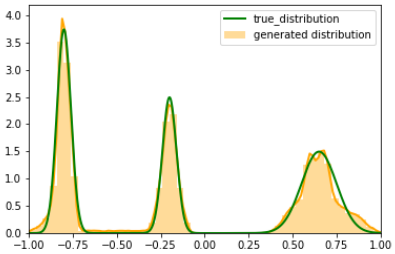


Figure 5.8: Fitted energy and probability distribution



A.Fitted energy vs true energy



B. Synthesized data vs underlying true distribution

Figure 5.9: 1D fitted energy and synthesis result

peak of the mode.

This illustrates our idea in this chapter. Under the limitation of MCMC sampling steps, the model tries to shape its energy to make sure the points can be guided to the right place. In the 1D case, the model actually cuts the input space into different intervals according to its gradient. Each interval contains on peak (We have postive gradient to zero gradient and to negative gradient. Next postive gradient will start the next interval). The points in this interval will concentrate to the peak in this interval under the guidance of gradient. The start and end point of the interval determine how many points will be guided to this interval. In Figure 5.9 A, there are roughly 3 intervals: [-1.0, -0.25], [-0.25, 0.25], [0.25, 1.0]. The relative ratio of their lengths are 3:2:3, which is agree with the number of point in our underlying distribution. The norm of the gradient controls to what extent will the points gathered to the peak during sampling. Thus, those intervals with bigger gradients (see the left and center peaks) will form thinner and taller peak. And those with smaller gradients (the right one) correspond to lower peak.

5.5 Summary

In this chapter, we show the case where we fit the model using finite-step MCMC. In section 5.1, we show that being able to synthesize good samples do not means the fitted model perfectly discribe the underlying distribution. In section 5.2, 5.3 and 5.4, we use various

experiments to show that the fitted distributions are not bad fittings for the underlying ones, but actually the specialist for the current sampling settings. They may choose to sacrifice the quality of fitted energy to make sure the generated samples are good enough. These result agree with the observation in [13].

CHAPTER 6

Conclusion

In this thesis, we use a 2D setting to help us form a better understanding of the Deep FRAME model. We show that the model captures the modes mainly by cutting the input space instead of rely on the piecewise Gaussian property. Changing the variance in the prior distribution or choosing smoother activation function will make the training process different and sometimes makes model harder to train. The introduction of ReLU function (or its smoother approximations) cuts the input sapce into many small cells. This enables the matching statistics from higher layer to take different forms in each cell partitioned by the lower layer and thus enable multi-layer model to capture highly complex distributions. The activation in a certain cell is usually far from sparese, but the number of neurons that actually define the boundary of that cell are indeed very small.

In the situation where finite-step MCMC is introduced to do sampling, the fitted energies are usually not well-fitted, even though we can get good synthesized data. These energies are actually intentionally and carefully shaped (instead of just fitted unperfectly) to facilitate MCMC to produce good samples.

REFERENCES

[1] R. Gao, Y. Lu, J. Zhou, S.-C. Zhu, and Y. Nian Wu. Learning generative convnets via multi-grid modeling and sampling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9155–9164, 2018.

[2] M. Girolami and B. Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

[3] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

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

[5] S. Gurumurthy, R. Kiran Sarvadevabhatla, and R. Venkatesh Babu. Deligan: Generative adversarial networks for diverse and limited data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 166–174, 2017.

[6] T. Han, E. Nijkamp, X. Fang, M. Hill, S.-C. Zhu, and Y. N. Wu. Divergence triangle for joint training of generator model, energy-based model, and inference model. *arXiv preprint arXiv:1812.10907*, 2018.

[7] M. Hill, E. Nijkamp, and S.-C. Zhu. Building a telescope to look into high-dimensional image spaces. *Quarterly of Applied Mathematics*, 77(2):269–321, 2019.

[8] C. Jiang, S. Qi, Y. Zhu, S. Huang, J. Lin, L.-F. Yu, D. Terzopoulos, and S.-C. Zhu. Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars. *International Journal of Computer Vision*, 126(9):920–941, 2018.

[9] M. Khayatkhoei, M. K. Singh, and A. Elgammal. Disconnected manifold learning for generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 7343–7353, 2018.

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

[11] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[13] E. Nijkamp, M. Hill, T. Han, S.-C. Zhu, and Y. N. Wu. On the anatomy of mcmc-based maximum likelihood learning of energy-based models. *arXiv preprint arXiv:1903.12370*, 2019.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[15] J. Xie, Y. Lu, R. Gao, and Y. N. Wu. Cooperative learning of energy-based model and latent variable model via mcmc teaching. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] J. Xie, Y. Lu, S.-C. Zhu, and Y. Wu. A theory of generative convnet. In *International Conference on Machine Learning*, pages 2635–2644, 2016.

[17] K. Xu, H. Liang, J. Zhu, H. Su, and B. Zhang. Deep structured generative models. *arXiv preprint arXiv:1807.03877*, 2018.

[18] S. C. Zhu and D. Mumford. Grade: Gibbs reaction and diffusion equations. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 847–854. IEEE, 1998.

[19] S. C. Zhu, Y. N. Wu, and D. Mumford. Minimax entropy principle and its application to texture modeling. *Neural computation*, 9(8):1627–1660, 1997.