Distributed Smoothing

Ludvig Karlsson **Viktor Eriksson Möllerstedt** **Oskar Bonde**

Abstract

In this paper, we attempt to reproduce experiments presented in [1], where the authors introduce a new approach to improving sample quality for autoregressive models. Our replication experiments consist of generating images, and calculating BPD and FID on the MNIST dataset. In addition, we evaluate the representation of the model using linear probes. We find that we obtain similar results in our replication experiments, and that the linear probe yields a surprisingly high accuracy of 98.5%. All of the code is available at `https://gits-15.sys.kth.se/vikmol/distr_smoothing_analysis`.

1 Introduction

Reproducibility is the foundations of science, but ensuring that research is reproducible is an ongoing challenge. For instance, a reproducibility program was initiated by Neural Information Processing Systems (NeurIPS) in 2019 [2]. Our goal with this paper is to contribute by reproducing some of the results in the paper "Improved Autoregressive Modeling with Distribution Smoothing" by Meng *et al* [1].

In [1], the authors attempt to improve the sample quality of autoregressive models, which is poor compared to state-of-the-art generative methods. Their approach is to train on noisy image data and then de-noise the generated samples. The authors found that image quality increased, and that quality metrics such as BPD and FID score improved.

Our goal was to replicate image experiments made in [1] on the CIFAR-10 dataset, but we did not have the computational resources to perform the training. Therefore, we decided to perform the experiments on MNIST instead, for which a smaller number of results were presented in [1]. The replication experiments consist of sampling images, as well as calculating BPD and FID score.

In addition to the replication experiments, we decided to investigate if the representations learned by this model are suitable for classification. This was done by training linear probes on labeled MNIST data. We got inspired to try this after reading "Generative Pretraining from Pixels" [3].

We obtained slightly worse sample image quality than in [1], but still much better than using the original PixelCNN++ model. Using linear probe classifiers we obtain a relatively high accuracy, 98.5% at most. The accuracy is lower for the final blocks of the model, and better for the noisy model than the de-noisying model.

2 Related work

We have tried to re-create the results from [1], and the model used in that article is the PixelCNN++ [4], which is an autoregressive convolutional model based on the original PixelCNN [5] model. The two latter papers perform experiments on the CIFAR-10 and ImageNet data sets and includes

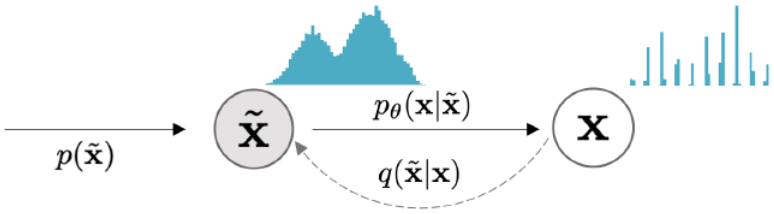

Figure 1: Model framework. Noise $q(\tilde{x}|x)$ is injected into data distribution x to create noisy data $\tilde{x}$. This data is used to train $p_\theta(\tilde{x})$. Simultaneously $p_\theta(x|\tilde{x})$ is trained on both x and $\tilde{x}$ to de-noise images created by $p_\theta(\tilde{x})$.

generated samples from these data sets. We have only trained our model on the MNIST images, so the results from the generative process are not directly comparable. The experiments we perform are that of generating images, which is done in the original paper, but we have also added an experiment called *linear probing*. This experiment tests the representational quality of the model by appending a linear model on top of the output from a certain layer. This model is then used as a classifier with the idea that if the model have good representational power, the output features should be linearly separable. This experiment is not done in any of the mentioned papers in this section. One paper related to this is [3], where the common transformer model GPT is used. In this paper the authors also perform linear probing in each of the layers. However, they only do this on CIFAR-10 while we again do it on MNIST, which somewhat limits the comparisons that can be made to that paper.

3 Method

The goal of generative models is to create a model $p_\theta(x)$ that is as close as possible to the real distribution $p_{data}(x)$. Meng *et al.* have developed a framework for autoregressive models that builds on lessons from adversarial deep learning. Images generated by autoregressive model usually have an unreasonably high likelihood according to the model, even though generated images can look very unrealistic. The authors propose that it should be easier for the autoregressive model to capture a smooth distribution. Their idea is to introduce randomized smoothing to the input data, and then de-noise the sampled images.

The model framework is rather simple, shown in Figure 1. The base model we and Meng *et al.* used was PixelCNN++[4], an autoregressive generative model with convolutional layers. We explain how it works in detail in the appendix.

During training, first step in the model framework is to inject noise using a symmetric and stationary distribution. Meng *et al.* found that Gaussian noise works best so that is what we used as well. So, data x is transformed with $q(\tilde{x}|x) = \mathcal{N}(\tilde{x}|x, \sigma^2 I)$ to get $\tilde{x}$. Then the two models can be trained, the noisy model $p_\theta(\tilde{x})$ and de-noisying model $p_\theta(x|\tilde{x})$, both are PixelCNN++ networks. The noisy model is trained on $\tilde{x}$ and the de-noisying model $p_\theta(x|\tilde{x})$ is trained on $\tilde{x}$ as well as x. With both models trained we obtain $p_\theta(x|\tilde{x})p_\theta(\tilde{x}) = p_\theta(x, \tilde{x})$ which can be used to find $p_\theta(x)$:

$$p_\theta(x) = \int p_\theta(x, \tilde{x}) d\tilde{x}. \tag{1}$$

During sampling, images are first generated by the noisy model $p_\theta(\tilde{x})$. Then, those images are used as input for the de-noisying model $p_\theta(x|\tilde{x})$ which creates the final images.

3.1 BPD

Bits per dimension (BPD) is calculated using the following formula:

$$BPD = -\frac{\mathbb{E}[\log p(x)]}{d \log 2}, \tag{2}$$

where d is the number of pixels and $\log$ is the logarithm in base e. To calculate $p(x)$, we need to use the integral equation 1, which in general is untractable. The likelihood can be approximated using the Evidence Lower Bound (ELBO) [1]:

$$\log p_\theta(x) \geq \mathbb{E}_{q(\tilde{x}|x)}\Big[\log p_\theta(\tilde{x}) + \log p_\theta(x|\tilde{x}) - \log q(\tilde{x}|x)\Big]. \tag{3}$$

Here, $p_\theta(\tilde{x})$ is the output of the model applied to smoothed data, and $p_\theta(x|\tilde{x})$ the output of the de-noisying model. We estimate the expectation of the first two terms via the Monte-Carlo method. The noisy images $\tilde{x}$ are sampled from $q(\tilde{x}|x) = N(x, \sigma^2 I)$, using the test set images x. The last term of the expectation can be calculated analytically. A straight-forward computation gives us that:

$$\mathbb{E}_q[\log q] = -\frac{d}{2}(\log 2\pi\sigma^2), \tag{4}$$

where d is the dimension of the image vector. For example, for MNIST, $d = 28^2$.

3.2 Linear Probing

Linear probing is intended to check the representational quality of a network at a certain layer. This is done by appending a linear classifier on top of the output from a layer, with the idea that a good representation should be linearly separable. This new model is trained on a classification problem, in this case MNIST, and is evaluated in terms of accuracy on a hold-out test set. Our architecture are made of blocks of convolutional layers. Due to time limitations we decided to test the representational quality at each block instead of each individual layer.

3.3 Frechet Inception Distance (FID)

To measure the quality of the generated images Meng et al.[1] used Inception, FID, and KID metric. All three use the Inception model which has been trained for image classification on ImageNet. FID calculates the distance two groups of images using the last pooling layer before the output from the Inception model. Inception expects an input of the shape $(299, 299, 3)$ so the MNIST images were scaled from $(28, 28, 1)$ by repeating the grey-scale dimension three times and nearest neighbour interpolation. Because the Inception model is unfit for MNIST we only reproduced results for FID.

The output from the Inception model is a feature vector v of size 2048. For each group of images the average $\mu = \mathbb{E}(v)$ and covariance matrix $\Sigma_{ij} = cov(v_i, v_j)$ of the feature vectors is calculated. The following formula is used [6]

$$FID = ||\mu_1 - \mu_2||^2 + Tr(\mathbf{\Sigma_1} + \mathbf{\Sigma_2} - 2\sqrt{\mathbf{\Sigma_1}\mathbf{\Sigma_2}})$$

4 Data

Meng et al.[1] applied their model to the publicly available image datasets MNIST, CIFAR10 and CelebA. Due limited compute-power availability our model was only fully trained on the MNIST dataset. Only the training datasets were used during training, a total of 60 000 images of shape $(28, 28, 1)$. Each image was normalized during prepossessing and labels weren't used during training. Classification on MNIST is simple compared to other datasets and the state of the art classification accuracy is 99.83% using a CNN with homogeneous filter capsules[7]. When training linear probes for classification on MNIST we used the 60 000 images in the MNIST train dataset and 10 000 images for testing.

5 Experiments

We decided to try to replicate some of the image experiments in section 4.3 of [1]. As stated previously, we trained the model on MNIST. We calculated the BPD, FID and generated images after training using the Two-step process. Unfortunately, the BPD and FID for the model trained on MNIST was not presented in the paper[1], so our comparison will be limited to the generated images.

We also trained linear probes for classification on MNIST. We used the representations in the final layers of each block of both of the PixelCNN++ models. Our plan was initially to compare the accuracy to the linear probes in [3] on CIFAR-10, but as explained earlier we were restricted to training on MNIST.

6 Results

6.1 Generated Images

In Figures 2-3 we see samples from the noisy and de-noisying models trained on MNIST. In Figure 4 we see MNIST images generate in [1]. Figure 2 should be compared to the left column in Figure 4 and Figure 3 should be compared to the third column.

Figure 2: Images generated by our noisy model.

Figure 3: Images generated by our de-noisying model.

Figure 4: Images generated in the original paper [1]. The left-most image contains samples from $p_\theta(\tilde{x})$. The second from the left contains samples de-noised using the gradient of $p_\theta(\tilde{x})$. The second from the right contains samples de-noised via $p_\theta(x|\tilde{x})$. The rightmost image are samples from the original PixelCNN++ model.

The images produced by our model are very similar to the images produced by Meng et al.[1]. However, their model appears to be slightly better, considering that some of the numbers we generate are incorrect. This could be because we don't train long enough, Meng et al.[1] don't say how long they trained their model or what loss they achieved. Still, our model is significantly better than vanilla PixelCNN++, fourth column in Figure 4.

6.2 BPD

We calculated the BPD after training on MNIST using the ELBO. Since each evaluation of the models involve adding noise to the input, we decided to calculate the BPD several times and average the results. Averaging over 6 runs, we found that BPD $= 3.23 \pm 3 \cdot 10^{-4}$. This was calculated with batch-sizes 80 and 80, for the noisy and de-noisying model respectively. To the best of our knowledge there is no values for the BPD on MNIST for the related papers mentioned in this report. However, there are state-of-the-art models that achieves a BPD of under 1.0 which is much lower than our score. One should keep in mind though that the likelihood score is not necessarily connected to generated samples of high quality.

6.3 Linear Probing

In this section we will present the results from the linear probing experiment. In Figures 5-6 below we see the accuracy of our linear probe as a function of which block's output representation that was used, for the noising and de-noisying models. For the noising model we see that the accuracy is relatively high (the current SOA is $\sim 99\%$), but that the performance decreases a bit in the final block. Similarly, the performance when probing the de-noising model decreases for deeper blocks with a quite poor performance in the final blocks. The trend that the representational power is best for earlier layers coincides with the findings in [3], even though the experiments in that paper was done on CIFAR-10 instead of MNIST. To conclude this section we can see that the best representations for each model, especially the noising model, achieves good performance when performing classification on MNIST.

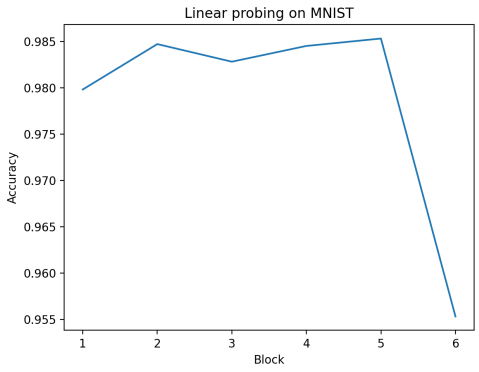

Figure 5: Linear probing on MNIST for noisy model.

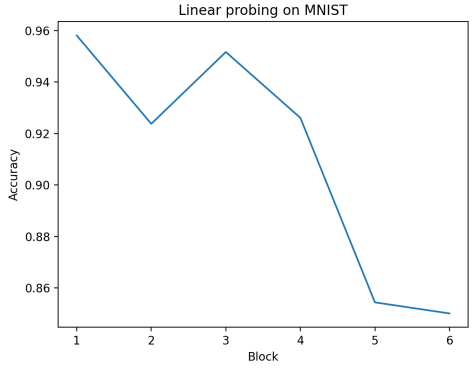

Figure 6: Linear probing on MNIST for de-noisying model.

6.4 Frechet Inception Distance (FID)

Because the Inception model was trained on ImageNet with contains colored picture from the real world with resolution 299×299. This is very different compared to images from MNIST which are grey-scale numbers with resolution 28×28. For this reason the metrics Inception, FID, and KID are inappropriate on MNIST data. Still, the FID score of our model is produced by comparing 1000 generated images to 1000 randomly selected images from the MNIST test set. The FID score on our MNIST model was **13.30** which is much lower than **29.83** produced by Meng et al.[1] on CIFAR-10, but this comparison is inappropriate since their score is on a different dataset.

We did not find any figures for the FID on any of the models in the related papers mentioned, but the current state-of-the-art on MNIST is a FID of under 5.0; a lot smaller than our score. However, since we have no insights into other results from these papers it is impossible make further comparisons to these models.

7 Challenges

Our initial plan was to train on CIFAR-10, and compare the BPD and FID to that presented in table 2 in [1]. However, training on CIFAR-10 took about 2 hours per epoch. With 500 training epochs per model, that amounts to roughly 84 days of training on our GPU, which simply was not feasible under our time constraints. We couldn't increase our GPU quota beyond 1. Because of this, we decided to train on MNIST instead, which was roughly 9x faster.

The paper[1] lacked a lot of details about how the code worked, but it did include most of the hyperparameters they used, both for MNIST and CIFAR10. Luckily the authors published their GitHub repository so we could figure out how their model worked by reading their code. The code wasn't well documented but it wasn't very complicated.

We did encounter challenges when using PixelCNN++. Meng et al.[1] had changed some of the loss function in PixelCNN++ without mentioning it, so our first model was incorrect. Also, to implement linear probing we had to change some of the code in PixelCNN++ but it wasn't well documented at all. We didn't have to tune any hyperparameters since the authors provided their parameters.

In this project we trained the model on GCP. The first challenge we encountered with GCP was to get access to a VM with a GPU, they were unavailable in most regions. Also, it took a long time to upload and download files to the VM.

8 Conclusion

Our samples from the de-noising model have a higher quality than those generated by the PixelCNN++ model. This indicates that the method presented in [1] works. On the other hand, the image quality we obtained was slightly worse than that of Meng et al. This could be because we made use of early stopping, or due to stochastic variation, such as the initialization of weights.

When training a linear probe on the different blocks of the models, we found that the accuracy was the highest in the first and middle blocks. The probes based on the noisy model performed the best, with a maximum test accuracy of $\approx 98.5\%$. This accuracy is quite high, and indicates that the model has a learned a good representation of the images in MNIST.

With more computational resources, we would have liked to focus our replication experiments on the CIFAR-10 datasets in order to compare with the BPD and FID computed in [1]. It would also have been interesting to compare linear probe results on CIFAR-10 with results produced by image-GPT [3].

9 Ethical consideration, societal impact, alignment with UN SDG targets

One ethical concern regarding generative models in general could be the use of the generated images for fraudulent purposes. So called *Deepfakes*, i.e. synthetic media with fake content, can be achieved by using generative models and this could be used for malicious purposes such as financial fraud. Regarding the methods used in the paper we don't see any ethical concerns. The code as well as the data sets is publicly available so the paper is reproducible by someone with knowledge within the area.

One of the main motivations behind generative models is that a successful model would potentially be able to model the full data generating distribution, making them good at representing the data for downstream tasks such as classification. There are near endless applications for classification models, one example being self-driving cars where the model has to take decisions based on what is perceived. To relate this project to how it would impact society, self-driving cars could increase the safety on roads which ties to the UN SDG goal number three regarding good health and well-being.

10 Self assessment

We have reproduced some of the experiments in the studied paper with good results as outcome. Additionally, we also added the experiment of linear probing which is highly relevant for generative models since it measures the representational power of a model. All of the code except for the implementation of PixelCNN++ model was written by us, and although we did not have the resources to test the model on the CIFAR-10 data set, our code is compatible with that data set. Therefore we believe that this project should get the grade that we aimed on in the project proposal, namely A.

Appendix

A.1. PixelCNN++

The model used in this report is called PixelCNN++ [4] and is an autoregressive generative model. PixelCNN++ is built on a model called PixelCNN [5] and uses *masked convolutions* to generate the pixels one at a time, where the pixel to be predicted depends on the previously generated pixel values. To get the autoregressive generative process, the masked convolution masks out all values on the same row to the right of the pixel being considered, as well as the pixels on rows below the pixel. A visual explanation can be seen in Figure 7. By doing this, the pixel value will only depend on previously generated pixels.

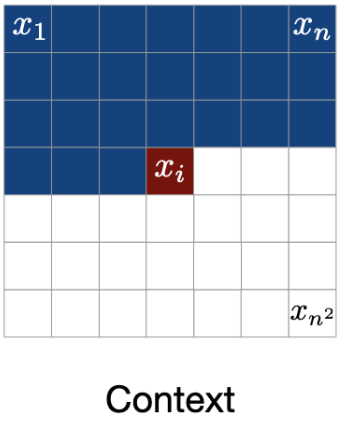

Figure 7: Masked convolution: only pixels to the left and above the current pixel will be considered for prediction. Image taken from [5].

An overview of the model architecture, which is arranged in blocks, can be found in Figure 8. Note that the figure shows dimensions for the CIFAR-10 data set, while we used the MNIST data set. Each block is a *Gated Residual Network* which consists of 5 layers of the form presented in Figure 9. During the forward pass we downsample the image three times; this is called the upstream pass. Following this we upsample the image three times back to the original size, and this is called the downstream pass. Doing this allows the network to model images at different resolutions [4].

One key issue of using plain masked convolutions is that they create a blind spot in the receptive field. To solve this, the gated residual networks are introduced. They are divided into two stacks: the vertical stack and the horizontal stack. The vertical stack conditions on all rows above the current pixel and this output is fed into the horizontal stack together with the original image. The horizontal stack then conditions on all pixels to the left on the same row as the current pixel. Together this results in a conditioning on all pixels above and all pixels to the left on the same row as the current pixel, which is what we want. Figure 10 shows the progression of the regular masked convolution versus the two stacks mentioned above. The former proceeds in a triangular fashion, which causes a blind spot. This is fixed by using two separate stacks of convolutions.

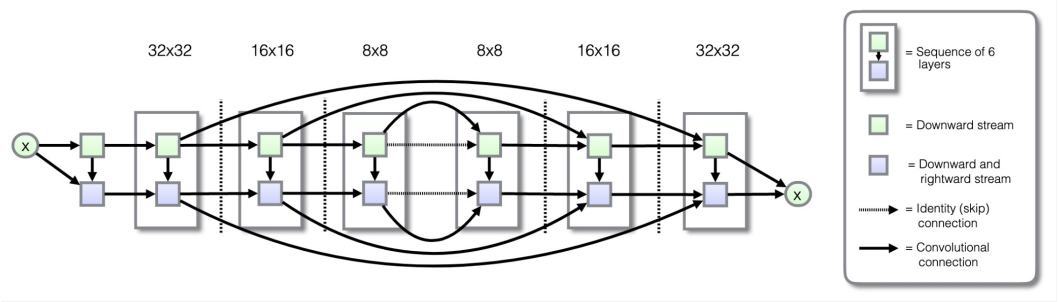

Figure 8: PixelCNN++ architecture: green flow denotes the vertical stack and the blue flow denotes the horizontal stack. Resolutions in figure are for the CIFAR-10 data set. Image taken from [4].

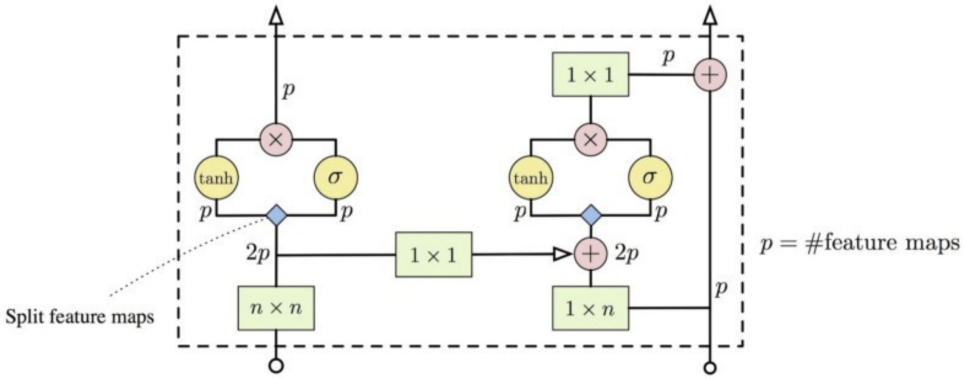

Figure 9: Single layer of gated residual network: Green boxes denotes convolutions with the size given in the box. The left side of the flow is the vertical stack, whereas the right side is the horizontal stack. Image taken from: `https://towardsdatascience.com/auto-regressive-generative-models-pixelrnn-pixelcnn-32d192911173`

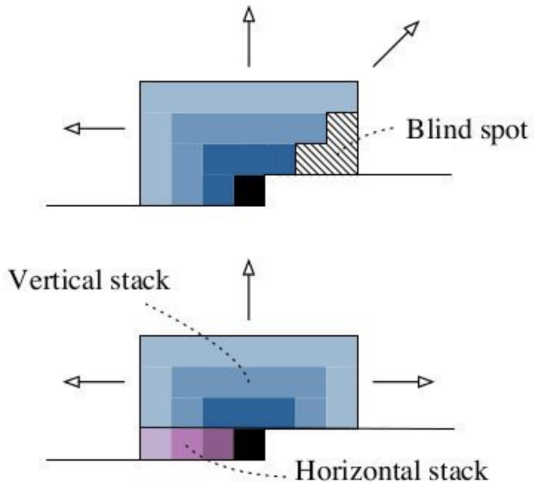

Figure 10: Masked convolution approach: naive implementation causes a blind spot in the receptive field. Image taken from `https://towardsdatascience.com/auto-regressive-generative-models-pixelrnn-pixelcnn-32d192911173`

References

1. Meng, C., Song, J., Song, Y., Zhao, S. & Ermon, S. Improved Autoregressive Modeling with Distribution Smoothing. *ICLR* (2021).
2. Pineau, J. *et al. Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program)* 2020. arXiv: 2003.12206 [cs.LG].
3. Chen, M. *et al. Generative Pretraining From Pixels* in *Proceedings of the 37th International Conference on Machine Learning* (eds III, H. D. & Singh, A.) **119** (PMLR, 13–18 Jul 2020), 1691–1703. https://proceedings.mlr.press/v119/chen20s.html.
4. Salimans, T., Karpathy, A., Chen, X. & Kingma, D. P. *PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications* 2017. arXiv: 1701.05517 [cs.LG].
5. Oord, A. V., Kalchbrenner, N. & Kavukcuoglu, K. *Pixel Recurrent Neural Networks* in *Proceedings of The 33rd International Conference on Machine Learning* (eds Balcan, M. F. & Weinberger, K. Q.) **48** (PMLR, New York, New York, USA, 20–22 Jun 2016), 1747–1756. https://proceedings.mlr.press/v48/oord16.html.
6. Heusel, M. *et al.* GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium. *CoRR* **abs/1706.08500.** arXiv: 1706.08500. http://arxiv.org/abs/1706.08500 (2017).
7. Byerly, A., Kalganova, T. & Dear, I. *No Routing Needed Between Capsules* 2021. arXiv: 2001.09136 [cs.CV].