ID2223 - An attempt to replicate/reproduce: Density Estimation using Real NVP

Walid Abdul Jalil
walidaj@kth.se

Lukas Demey
Demey@kth.se

1 Introduction

Due to a deep interest in anything Bayesian such as variational methods. We make an attempt at reproducing a paper by (Laurent Dinh et al, 2017)[1] that describes a brilliant way of performing density estimation using variational methods along with normalizing flows. Normalizing flows refer to the method of transforming simple probability distributions through a series of invertible transformations (bijections). By using them, one can improve the expressive power of the distributions. Simple densities are pushed towards richer and more multi-modal distributions. Dinh et al propose a set of real-valued non-volume preserving (Real NVP) transformations that are power, invertible computationally tractable. This results in an unsupervised learning algorithm where exact log-likelihood computation and exact efficient sampling is possible. As these are invertible, density estimation of samples is also possible.

2 Normalizing flows

Normalizing flows refer to the method of transforming a simple probability distribution, say $p(X)$, to a more complex distribution $p(Z)$ through a series of invertible transformations (bijections). If X is a random variable and f is an invertible function such that $f : X \to Z$ (with $g = f^{-1}$), then $f(X)$ is also a random variable. The relationship between the old distribution and the distribution of the new random variable $Z = f(X)$ is given by the change of variables formula:

$$p_X(x) = p_Z(f(x)) \left| \det \left(\frac{\partial f(x)}{\partial x^T} \right) \right| \tag{1}$$

The problem with equation (1) is that the determinant of the jacobian is generally computationally expensive.It is usually of computational complexity $\mathcal{O}(n^3)$. By carefully designing the function f and making use of the fact that the determinant of a lower triangular matrix is simply the product the of the diagonal elements: We can calculate a determinant with computational complexity of $\mathcal{O}(n)$.

Specifically, given D dimensional input x and $d < D$, we use the affine transformation to get output y:

$$\begin{aligned} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp\left(s\left(x_{1:d}\right)\right) + t\left(x_{1:d}\right). \end{aligned} \tag{2}$$

This transformation is more tractable and extremely flexible. The Jacobian of this transformation is

$$\frac{\partial y}{\partial x^T} = \left[\begin{array}{cc} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \operatorname{diag}\left(\exp\left[s\left(x_{1:d}\right)\right]\right) \end{array} \right] \tag{3}$$

The determinant is thus given by: $\exp\left[\sum_j s\left(x_{1:d}\right)_j\right]$.

Note that s and t can be arbitrarily complex. We don't need to calculate the Jacobian for them. Thus s and t will both be deep convolutional networks. Specifically, ResNets.

3 Neural Architecture

We apply a multi-scale architecture with alternating checkerboard and channelwise masks along with squeezing operations.

At each scale we apply 3 affine coupling layers with alternating checkerboard masks. After a sequeezing operation is done, trading spatial-size for channel-size. Finally 3 affine couplings are applied but this time with channel-wise masking. For the final scale, only four coupling layers with alternating checkerboard masks are applied.

Both s and t functions are deep ResNets, with 8 residual blocks, 64 feature maps and one down-scaling. The scaling function s is then computed by passing the output of the ResNet through a tanh-function and then multiplied by a learned weight. Note: The ResNet was not the subject of this study, that would have required reading another paper and trying to implement it. So we took the ResNet as it was straight from the following git repository: `https://github.com/chrischute/real-nvp`

Furthermore, we found that it was impossible to simply go after the paper, as we encountered numerical instabilities. This required using Softplus. SoftPlus is a smooth approximation to the ReLU function and can be used to constrain the output of a machine to always be positive. Other methods that were needed but were not mentioned in the paper were: restricting the values of the input (using clamp method) as well as clipping the gradients. The convolutional weights in the ResNet were normalized. L_2 regularization was used on these weights with coefficient of $5 * 10^{-5}$. ADAM optimizer was used with a default learning rate of $lr = 10^{-3}$. The plan is to run the experiments up to 100 epochs, but we will stop if we see the validation loss start to increase significantly.

Since we did not have enough experience with PyTorch, the git repo mentioned above was our main guideline along with the paper. So we went back and fourth between the paper and the repository.

Data & Data Augmentation

We trained on CIFAR10 dataset, which contains 60.000 color images of size 32x32, split between 50.000 training images and 10.000 validation/test images. The dataset was first de-quantized by adding random data from a uniform distribution to each pixel. The CIFAR10 images were randomly horizontally flipped, as that was the only data augmentation that was mentioned in the paper. Before doing a forward pass, the pictures are dequantized by adding noise from a uniform random distribution. Pictures are usually stored using 8-bit integers, but here they are modelled as densities. This means that we are treating an image as a continuous random variable. A discrete data distribution has differential entropy of negative infinity, this can lead to arbitrary high likelihoods even on test data. That is why we add real-valued noise to the integer pixel values to dequantize the data[3].

Software & Hardware

Software:	Hardware (GCP Instance):
• Python 3.7.6	• Nvidia K80 (12 GB GDDR5 VRAM)
• PyTorch 1.10	• Nvidia T4 (16 GB GDDR6 VRAM)
• Numpy 1.21.2	• 4 Intel Haswell vCPUs
• Torchvision	• 26 GB RAM Memory

4 Experiments & Results

Since we had a total of 8 Affine coupling layers, each with ResNet with 8 residual blocks, the network was rather large. According to the repository, training the network would take 4 minutes per epoch with an Nvidia Titan Xp GPU. When we ran our model, it took more than 2 hours per epoch with an Nvidia K80 GPU, and roughly 40 minutes with an Nvidia P100 GPU. So we felt a bit misled. Since the Nvidia P100 GPU is more powerful than the Titan Xp. The author wasn't running automatic mixed precision since that is not supported by Titan Xp anyways. So it was doubly misleading.

Finally, we managed to get our hands on an Nvidia T4 GPU, which supports amp (automatic mixed precision). This helped considerably and one epoch was down to 15-20 minutes.

The Nvidia T4 GPU instance is preemptive, meaning it will run for maximum 24 hours at a time and shut down if someone with more need comes along. In return, the price is reduced by more than one half.

Automatic Mixed Precision (Amp) may sometimes result in numerical underflow, then the logarithm in the negative log-likelihood will return NaN values. If that happens, it is better to capture the error and restart the calculation from nearest checkpoint. Perhaps turning off automatic mixed precision. Normally, Grad Scaler function should fix this problem, but not always.

The results will mainly be measured in Bits Per Dimension. Bits per dimensions is the same as negative-loglikelihood (NLL) divided by dimensional size of each image. The results from the paper are presented below in Table 1 for comparison.

Dataset	PixelRNN	Real NVP	Conv DRAW	IAF-VAE
CIFAR-10	3.00	**3.49**	< 3.59	< 3.28
Imagenet (32×32)	3.86(3.83)	4.28(4.26)	$< 4.40(4.35)$	
Imagenet (64×64)	3.63(3.57)	3.98(3.75)	$< 4.10(4.04)$	
LSUN (bedroom)		2.72(2.70)		
LSUN (tower)		2.81(2.78)		
LSUN (church outdoor)		3.08(2.94)		
CelebA		3.02(2.97)		

Table 1: Bits/dim results for CIFAR-10, Imagenet, LSUN datasets and CelebA. Test results for CIFAR-10 and validation results for Imagenet, LSUN and CelebA (with training results in parenthesis for reference).

Figure 1 shows the results over 100 epochs for both training and validation sets. As can be seen, there is convergence at around a value of 3.50 bits per dimension. The validation curve is volatile due to the stochastic nature of the network. Table 2 shows that the mean of the last 40 epochs is roughly 3.47 bits per dimension. As can be seen, the results are in-line with what the authors have achieved. Furthermore, some samples have been included in Figure 2. The pictures are small, but they are not entirely noise. We can see the outlines of a sky on the fourth image in Figure 2

Our conclusion is that it's a very interesting paper, but we most likely would not have been able to implement it without looking at someone else's code. This is probably both due to our inexperience and due to vague descriptions in the paper. It would have been more preferable to train the network on the CelebA dataset, which consists of images of size 64x64. Checking the quality of sampled images is easier when you are looking at faces.

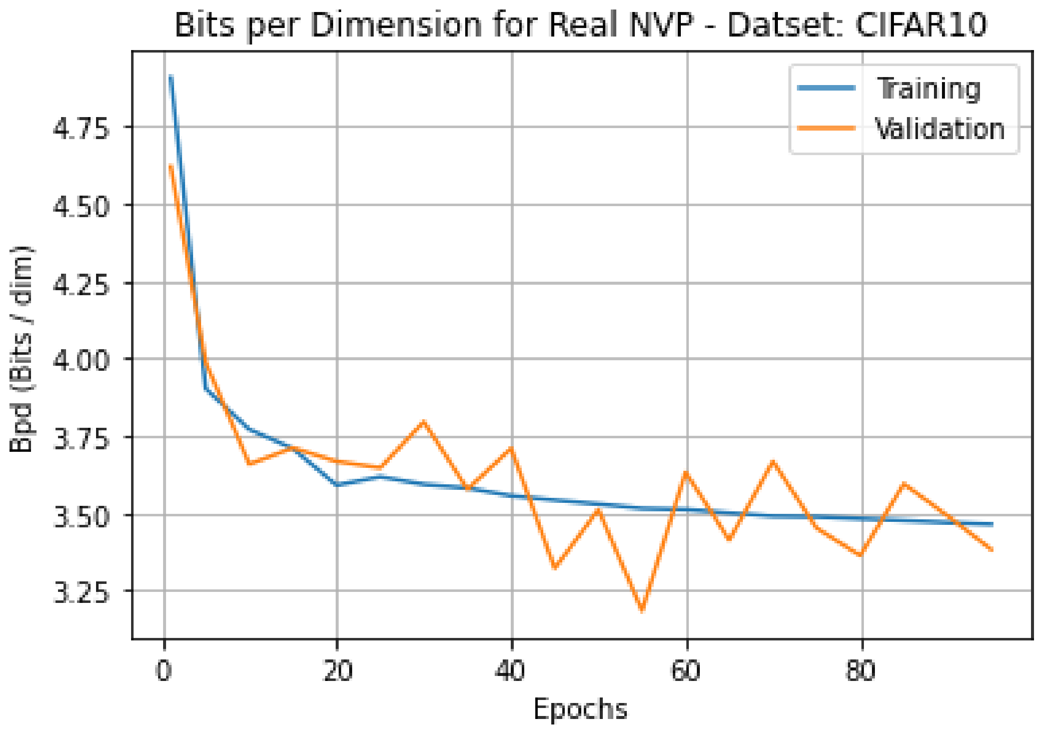

Figure 1: Bits per dimension (NLL) - Real NVP on CIFAR10 dataset.

Validation NLL (Bits per dim)	
Mean	Std
3.47	0.137

Table 2: Mean and Std of Validation NLL (Bits per dim) on the last 40 epochs.

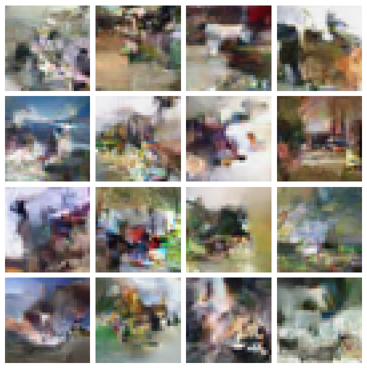

Figure 2: An example of some samples from the neural net. Samples came from a uniform distribution which were then passed through the network (inverse).

Future studies

For future studies, we would like to implement the slightly newer paper: **Glow: Generative Flow with Invertible 1x1 Convolutions** [2]. Glow replaces the squeeze operations of Real NVP with invertable 1x1 convolutions. This has been shown, empirically, to lower negative log-likelihood (NLL).

5 Code

Running the code is done via running python main.py in the terminal. Note that you need to change the PATH directories and make remove .cuda() from the code if you do not have an Nvidia GPU. We also iterate that some of the code is taken from the repository, since we did could not figure how to do the masking etc. The ResNet was not the purpose of the project so we took the ResNet scripts as they were also.

You can also sample images by running the sampling.py file, provided that you provide a path to a pre-trained model.

References

[1] *Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio.* Density estimation using Real NVP. *2017. arXiv: `1605. 08803 [cs.LG]`.*

[2] *Diederik P. Kingma and Prafulla Dhariwal.* Glow: Generative Flow with Invertible 1x1 Convolutions. *2018. arXiv: `1807. 03039 [stat.ML]`.*

[3] *Lucas Theis, Aäron van den Oord, and Matthias Bethge.* A note on the evaluation of generative models. *2016. arXiv: `1511. 01844 [stat.ML]`.*