Auto-Regressive Generative Models (PixelRNN, PixelCNN++)

Harshit Sharma Follow
Dec 20, 2017 · 10 min read

Authors : Harshit Sharma, Saurabh Mishra

Generative models are a subset of unsupervised learning wherein given some training data we generate new samples/data from the same distribution. There are two ways to model this distribution, with the most efficient and popular of them being Auto-Regressive models, Auto-Encoders and GANs.

The basic difference between Generative Adversarial Networks (GANs) and Auto-regressive models is that GANs learn implicit data distribution whereas the latter learns an explicit distribution governed by a prior imposed by model structure. A distribution can be anything, e.g. a class label, images of cars or cats. A prior in simpler terms means a probability distribution of a quantity.

Some of the advantages of Auto-regressive models over GANs are:

1. **Provides a way to calculate likelihood** : These models have the advantage of returning explicit probability densities (unlike GANs), making it straightforward to apply in domains such as compression and probabilistic planning and exploration

2. **The training is more stable than GANs** : Training a GAN requires finding the Nash equilibrium. Since, there is no algorithm present which does this, training a GAN is unstable as compared to PixelRNN or PixelCNN.

3. **It works for both discrete and continuous data** : It's hard to learn to generate discrete data for GAN, like text.

GANs are known to produce higher quality images and are faster to train. There are efforts being made to incorporate the advantages of both classes in a single model but it is still an open-research area. In this blog, we'll focus solely on the Auto-regressive part and leave the others for sometime later.

One of the foremost and well known problems in unsupervised learning is that of modeling the distribution of natural images, and this is particularly the reason why we've chosen to write this blog.

To solve this problem we require a model that is tractable and scalable. PixelRNN, PixelCNN are a part of the class of Auto-regressive models that fulfill both of these conditions.

These kind of models are preferably used in image completion. The reason for the same is because it performs better than other generative models on these kind of problems.

Figure 1. Image completions sampled from a PixelRNN.

PIXEL RNN

An effective approach to model such a network is to use probabilistic density models (like Gaussian or Normal distribution) to quantify the pixels of an image as a product of conditional distributions. This approach turns the modeling problem into a sequence problem wherein the next pixel value is determined by all the previously generated pixel values.

To process these non-linear and long term dependencies between pixel values and distributions we need an expressive sequence model like Recurrent Neural Network(RNN). RNNs have been shown to be extremely efficient in handling sequence problems.

Conditional Independence

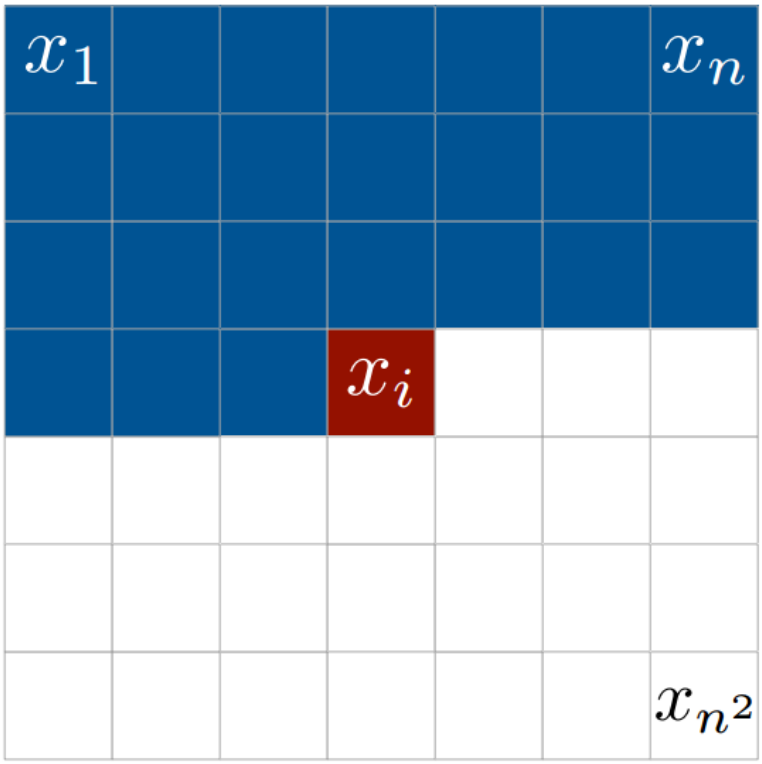

FIGURE 2 : Pixels of an nxn image

The network scans the image one row one pixel at a time in each row. Subsequently it predicts conditional distributions over the possible pixel values. The distribution of image pixels is written as product of conditional distributions and these values are shared across all the pixels of the image.

The objective here is to assign a probability p(x) to every pixel of the (n x n) image. This can be done by writing the probability of a pixel xi as :

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

This is the probability of ith pixel given the probability of all previously generated pixels. The generation proceeds row by row and pixel by pixel. Also each pixel xi is jointly determined by all three color channels red, green and blue (RGB). The conditional probability of ith pixel becomes :

$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

Thus, each color is conditioned on other colors as well as the previously generated pixels.

Since we can now know the conditional probability of our pixel value, to get the appropriate pixel value we use a 256-way softmax layer. Output of this layer can take any value ranging from 0–255 i.e our pixel value can vary from 0–255.

Model Architecture:

There are four different architectures which can be used, namely :

Row LSTM, Diagonal BiLSTM, a fully convolutional network and a Multi Scale network.

The network consists of upto 12 layers of two dimensional LSTMs. Two types of LSTM layers that are used are,

1. **Row LSTM** : The first layer is a 7x7 convolution that uses the mask of type A. It is followed by the input to state layer which is a 3x1 convolution that uses mask of type B and a 3x1 state to state convolution layer which is not masked. The feature map is then passed through a couple of 1x1 convolution layers consisting of ReLU and of mask type B. The final layer of the architecture is the 256-way softmax layer.
2. **Diagonal BiLSTM** : The only difference between its architecture and Row LSTM's is in the input to state and the state to state layers. It has a 1x1 convolution input to state layer with mask type B and a 1x2 convolution state to state layer without mask.

Row LSTM

Hidden state(i,j) = Hidden state(i-1,j-1)+ Hidden state(i-1,j+1)+ Hidden state(i-1,j)+ p(i,j)

This processes the image row to row from top to bottom computing features of the whole row at once. It captures a fairly triangular region above the pixel. However it isn't able to capture the entire available region.

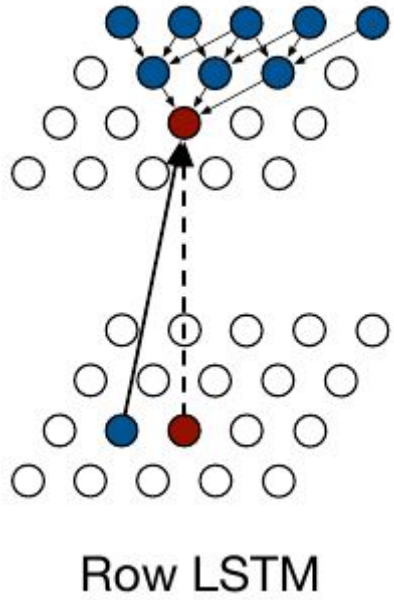

FIGURE 3 : input-to-state and state-to-state mapping for Row LSTM

Diagonal BiLSTM

pixel(i, j) = pixel(i, j-1) + pixel(i-1, j).

The receptive field of this layer encompasses the entire available region. The processing goes on diagonally. It starts from the top corner and reaches the opposite corner while moving in both directions.

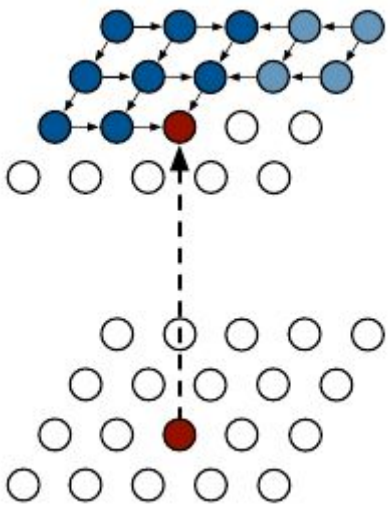

FIGURE 4 : input-to-state and state-to-state mapping for Diagonal BiLSTM

Residual connections (or skip connections) are also used in these networks to increase convergence speed and propagate signals more directly through the network.

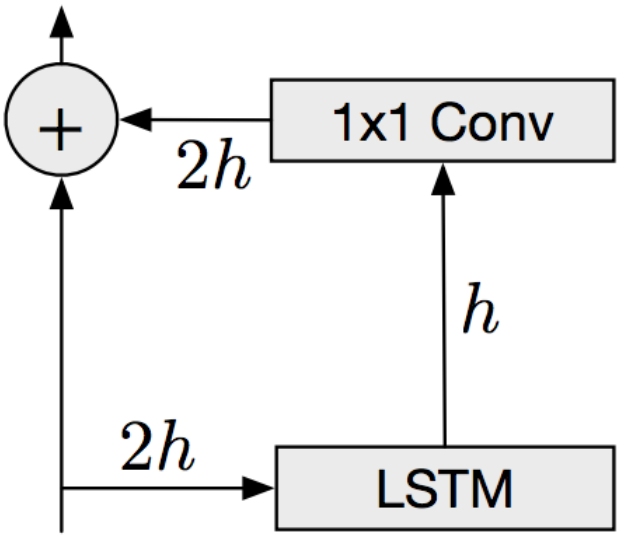

FIGURE 5 : Residual block for PixelRNNs. 'h' refers to the number of parameters.

Masked convolutions :

The features for each input position in every layer are split into three parts each one corresponding to a color(RGB). For computing the values of G channel we need the value of the R channel along with values of all previous pixels. Similarly, B channel requires information of both R and G channels. To restrict the network to adhere to these constraints we apply masks to convolutions.

We use two types of masks :

1. **Type A** : this mask is only applied to the first convolutional layer and restricts connections to those colors in current pixels that have already been predicted.
2. **Type B** : this mask is applied to other layers and allows connections to predicted colors in the current pixels.

Masks are an important part of network which maintain the number of channels in the network.

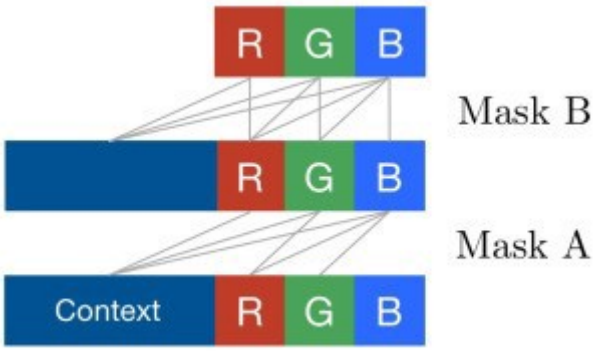

FIGURE 6 : Connectivity inside a masked convolution

Loss Function and Evaluation metric

Here, negative log likelihood (NLL) is used as the loss and evaluation metric as the network predicts(classifies) the values of pixel from values 0–255.

Pixel CNN

The main drawback of PixelRNN is that training is very slow as each state needs to be computed sequentially. This can be overcome by using convolutional layers and increasing the receptive field. PixelCNN uses standard convolutional layers to capture a bounded receptive field and compute features for all pixel positions at once. It uses multiple convolutional layers that preserve the spatial resolution. However, pooling layers are not used. Masks are adopted in the convolutions to restrict the model from violating the conditional dependence.

The first layer is a 7x7 convolution that uses mask A and the rest of the layers use 3x3 convolution and mask B. Then feature map passes through a couple of layers consisting of ReLU activation and 1x1 convolution. The final layer of this architecture is the 256-way softmax layer.

PixelCNN lowers the training time considerably as compared to PixelRNN. However, the image generation is still sequential as each pixel needs to given back as input to the network to compute next pixel.The major drawback of PixelCNN is that it's performance is worse than PixelRNN. Another drawback is the presence of a Blind Spot in the receptive field. The capturing of receptive field by a CNN proceeds in a triangular fashion. It causes several pixels to be left out of the receptive field, as shown in the figure below. Since convolutional networks capture a bounded receptive field (unlike BiLSTM) and computes features for all pixels at once, these pixels aren't dependent on all

previous pixels which is undesirable. The convolutional layers are unable to completely process the receptive fields thus leading to a slight miscalculation of pixel values. The pixels left out constitute the blind spot.

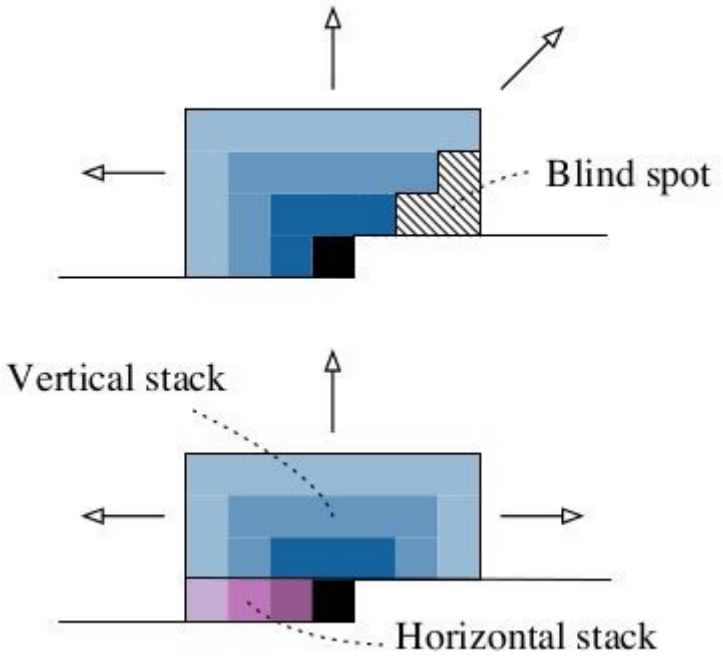

FIGURE 7 : Blind spot in a PixelCNN and its solution in Gated PixelCNN

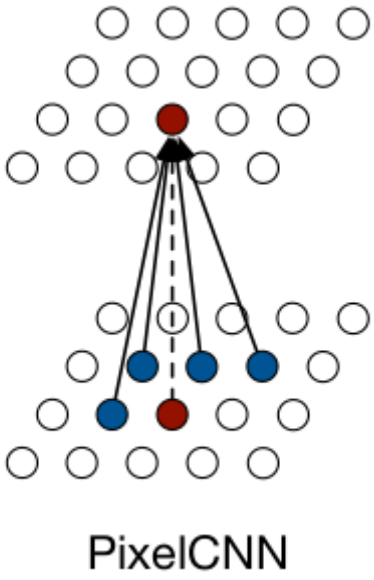

FIGURE 8 : input-to-state and state-to-state mapping for PixelCNN

Gated PixelCNN

It improves upon the architecture of PixelCNN whilst matching the log likelihood of PixelRNN.

The basic idea as to why PixelRNN outperforms PixelCNN is because it uses LSTM layers. Receptive field of LSTMs encompasses all of the neighbouring pixels in the network while it grows with depth in PixelCNN. The major improvement or change introduced in this method is the usage of the following activation instead of ReLU :

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x}) \odot \sigma(W_{k,g} * \mathbf{x}),$$

σ is the sigmoid non-linearity, k is the number of the layer, ⊙ is the element-wise product and ∗ is the convolution operator.

The other major improvement of this model is using CNN to use the available receptive field completely by using stacks. The receptive field in PixelCNN is bounded which leads to omission of several pixels from calculations of conditional distribution thus creating a blind spot which the CNN cannot process. This model uses two layers of stacks to process to eliminate this aforementioned problem:

1. **Horizontal Stack** : It conditions on the current row and takes as input the output of previous layer as well as the of the vertical stack.
2. **Vertical Stack** : It conditions on all the rows above the current pixel. It doesn't have any masking. It's output is fed into the horizontal stack and the receptive field grows in rectangular fashion.

The results show that Gated PixelCNN (3.03 bits/dim) outperforms PixelCNN (3.14 bits/dim) by 0.11 bits/dim and it's performance is comparable to PixelRNN (3.00 bits/dim) while taking less than half of the training time.

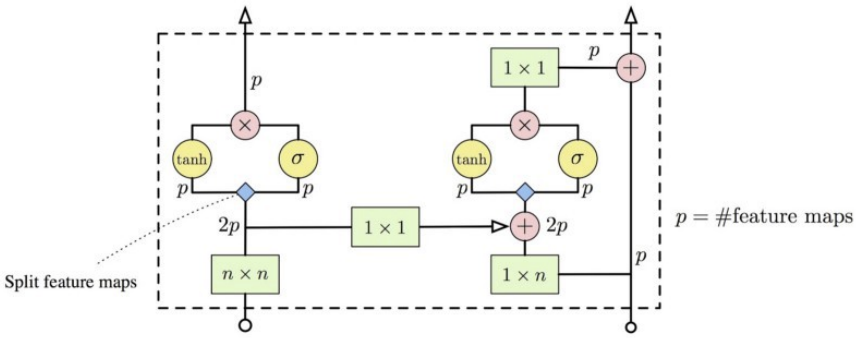

FIGURE 9 : A single layer in Gated PixelCNN architecture

Pixel CNN ++

This model by OpenAI does some modifications to improve the performance of PixelCNN whilst retaining its computational efficiency. Notable modifications include :

1. **Discretized logistic mixture likelihood** : The softmax layer which is used to compute the conditional distribution of a pixel although efficiency is very costly in terms of memory. Also, it makes gradients sparse early on during training. To counter this, we assume a latent color intensity akin to that used in variational autoencoders , with a continuous distribution. It is rounded off to its nearest 8-bit representation to give pixel value. The distribution of intensity is logistic so the pixel values can be easily determined. This method is memory efficient, output is of lower dimensions which provides denser gradients thus solving both problems.

2. **Conditioning on whole pixels** : PixelCNN factorizes the model over the 3 sub pixels according to the color(RGB) which however, complicates the model. The dependency between color channels of a pixel is relatively simple and doesn't require a deep model to train. Therefore, it is better to condition on whole pixels instead of separate colors and then output joint distributions over all 3 channels of the predicted pixel.

3. **Downsampling** : PixelCNN cannot compute long range dependencies. This is one of the disadvantages of PixelCNN as to why it cannot match the performance of PixelRNN. To overcome this, we downsample the layers by using convolutions of stride 2. Downsampling reduces input size and thus improves relative size of receptive field which leads to some loss of information but it can be compensated by adding extra short-cut connections.

4. **Short-cut connections** : This model the encoder-decoder structure of U-net. Layers 2 and 3 are downsampled and then layers 5 and 6 are upsampled. There is a residual connection from encoders to decoders to provide the localised information.

5. **Dropout** : Since the model for PixelCNN and PixelCNN++ are both very powerful, they are likely to overfit data if not regularized. So, we apply dropout on the residual path after the first convolution.

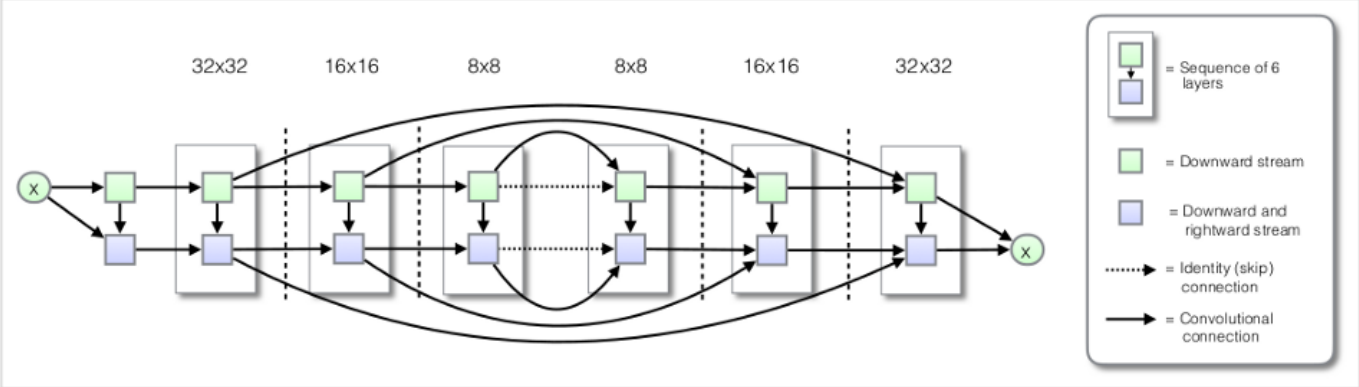

FIGURE 10 : Convolutional architecture with residual connections

PixelCNN++ outperforms both PixelRNN and PixelCNN by a margin. When trained on CIFAR-10 the best test log-likelihood is 2.92 bits/pixel as compared to 3.0 of PixelRNN and 3.03 of gated PixelCNN. Also if any one of the modification is not used the performance drops and the model learns slowly or in some cases fails to learn completely. More details about these experiments are given in paper.

Conclusion

Experiments showed that PixelCNN++ can be used as decoder in Variational Autoencoders (VAE) which is of application in the ongoing researches as can be seen in several research papers such as PixelGAN. PixelCNN forces the encoder to learn higher level features as the network can itself take care of lower dimensional features. Use of VAEs, GANs and Auto-regressive models together is an active area of research. These models along with Reinforcement techniques can improve the state-of-the-art techniques and lead the research in the area of unsupervised/semi-supervised learning to match the level of research going on in supervised learning.