Pixel Recurrent Neural Network

Summary

-Amit Yadav

Introduction

PixelRNN is a generative model that attempts to model the joint probability distribution of the data (images in this case) we feed in. Modeling the distribution of natural images is a landmark problem in machine learning. Several other neural network architectures have attempted to achieve this task, including Variational Autoencoders, Generative Adversarial Networks (GAN) and spatial LSTM networks. The task requires an image model that is expressive, tractable and scalable.

- VAE provide efficient inference (faster image generation) with approximate latent variables, but tend to be blurry.
- GANs generate sharp image but are hard to optimize.
- Autoregressive models are simple and stable while training. However, testing time is larger compared to other models.

PixelRNN is one of Autoregressive models which directly model distribution of pixels. The paper presents four models:

- PixelCNN
- Row LSTM
- Diagonal BiLSTM
- Multi-Scale PixelRNN

Model

In an image, a pixel depends on nearly all the previously generated pixels. So we have a long range dependency for which RNNs are proven to be efficient. The image generation process starts at a corner (say top left) and proceeds towards the opposite corner (bottom right). Let's name the pixels of a $n \times n$ image as $x_1, x_2, \ldots, x_{n^2}$. The joint distribution $p(\mathbf{x})$ can be written as:

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, x_2, \ldots, x_{i-1}) \tag{1}$$

where $p(x_i | x_1, x_2, \ldots, x_{i-1})$ is the probability of i-th pixel, given all the previously generated pixels. Expression 2 denotes the probability of predicting R,G,B for pixel x_i given all previously generated pixels.

$$p(x_{i,R} | \mathbf{x}_{<i}) p(x_{i,G} | \mathbf{x}_{<i}, x_{i,R}) p(x_{i,B} | \mathbf{x}_{<i}, x_{i,R}, x_{i,G}) \tag{2}$$

The R-value of a pixel is predicted first, then G-value is predicted which depends on the R-value of that pixel. Similarly, B-value of a pixel depends on the R and G-value of that pixel.

LSTM layers

Generally, LSTM layers are used for long range dependency problems. Every state-to-state computation step involves the following equations:

$$\begin{aligned}[\boldsymbol{o_i}, \boldsymbol{f_i}, \boldsymbol{i_i}, \boldsymbol{g_i}] &= \sigma(\boldsymbol{K}^{ss} \circledast \boldsymbol{h}_{i-1} + \boldsymbol{K}^{is} \circledast \boldsymbol{x}_i) \\ \boldsymbol{c}_i &= \boldsymbol{f}_i \odot \boldsymbol{c}_{i-1} + \boldsymbol{i}_i \odot \boldsymbol{g}_i \\ \boldsymbol{h}_i &= \boldsymbol{o}_i \odot tanh(\boldsymbol{c}_i)\end{aligned} \tag{3}$$

where $\boldsymbol{x}_i$ is the i-th row of input map and $\circledast$ represent convolution operation and $\odot$ represent element wise multiplication. $\boldsymbol{K}^{ss}$ and $\boldsymbol{K}^{is}$ are the kernel weights for state-to-state and input-to-state component. In this case, to generate a pixel we need all previously generated pixel's hidden state. So parallelization is not possible, resulting in high training time. This paper presents two ways to overcome this problem:

Row LSTM

In Row LSTM, the state of a pixel depends upon k pixels above it. And those k pixels depend upon $k+2$ pixels above them. So we get a triangular shaped context (Figure 4 in paper). As we can see, state of a pixel doesn't depend upon pixels of it's own row, i.e it depends upon only on some pixels of previous row. So hidden state for all pixels in a row can be parallely computed, thus solving the problem of high training time. But the context doesn't include all the pixels generated before it, which is what we would ideally want. So Diagonal BiLSTM is introduced, which solves the problem of incomplete context.

Diagonal BiLSTM

Diagonal BiLSTM captures entire available context. Image is generated in a diagonal fashion i.e from a top corner to opposite bottom corner. To apply convolution easily, each row is first skewed to the right such that all the pixels in a digonal now fall in a single column (Figure 3). Now we use the previous diagonal to generate the next diagonal. All the pixels of a diagonal can be generated simultaneouly, thus Diagonal BiLSTM supports parallelization.
Each pixel depends on hidden layer of the pixel above it and before it, hence capturing entire available context (Figure 3). For each step, we compute equation 3 to generate any pixel. Same computation is done for right to left diagonal. To prevent convolution layers from seeing future pixels, the right output map is shifted down by one row and then added to left output map.

PixelCNN

Row and diagonal LSTM layers cover long range dependencies but have costly computation due to complex nature of LSTM layers. Instead we can use convolutional layers which have bounded but large receptive field. All pixel positions can be computed parallely, thus reducing training time. Pooling layers are not used to preserve spatial resolution. Masks are used to prevent layers from seeing future pixels.

Multi-Scale PixelRNN*

In Multi-Scale PixelRNN, we have an unconditional PixelRNN and at least one conditional PixelRNN. The unconditional PixelRNN first generates a smaller image which is then sent through a series of deconvolutional layers to generate a bigger image. The conditional PixelRNN then takes it as an additional input and proceeds in the usual way.

**I am not very clear on this.*