PixelCNN

A Preprint

1 Main

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

PixelCNN typically consists of a stack of masked convolutional layers that take $N \cdot N \cdot 3$ image as input and produces $N \cdot N \cdot 3 \cdot 256$ output predictions as output. During sampling the predictions are sequential: every time a pixel is predicted, it is fed back into the network to predict the next pixel. This sequentiality is essential to generating high quality images, as it allows every pixel to depend in a highly non-linear and multimodal way on the previous pixels.

2 Gated Convolutional layers

PixelRNNs, which use spatial LSTM layers instead of convolutional stacks, have previously been shown to outperform PixelCNNs as generative models.

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

where k is a number of layers. * - convolutional operation. As analogy to LSTM this is used instead of ReLu.

Note that a significant portion of the input image is ignored by the masked convolutional architecture. This 'blind spot' can cover as much as a quarter of the potential receptive field (e.g., when using 3x3 filters), meaning that none of the content to the right of the current pixel would be taken into account.

Why blind spot occurs? As PixelCNN generates pixel by pixel it uses masked convolutions no not look in future during generation. In PixelCNN it has 2 fixes:

- Horizontal stack. Firstly processes row left to right -> masked conv - >gets it s own state + output of Vertical stack
- Vertical stack. Firstly process all rows before current. No masking

So they have not regular conv block, they have:

- Vertical conv
- Horizontal conv
- Gating(tanh etc)
- Residual

2.1 Adding conditioning

Basically to add condition we modify element-wise product:

$y = \tanh(W_{k,f} * x + V_{k,f}^T h) \odot \sigma(W_{k,g} * x + V_{k,g}^T h)$ where $W_{k,f}, W_{k,g}$ conv kernels and $V_{k,g}, V_{k,f}$ learnable projection matrices for c

Algorithm 1 Conditional PixelCNN Training and Sampling Pipeline

Require: Dataset $\mathcal{D} = \{(x^{(i)}, h^{(i)})\}_{i=1}^{N}$ where $x^{(i)} \in \mathbb{R}^{H \times W \times 3}$ is an image and $h^{(i)} \in \mathbb{R}^{d}$ is a condition vector (e.g., label or embedding)

Require: Number of layers L, filter size $k \times k$, hidden channels p

Initialize convolutional weights $\{W_{k,f}, W_{k,g}\}_{k=1}^{L}$

Initialize projection weights $\{V_{k,f}, V_{k,g}\}_{k=1}^{L}$ with $V_{k,*} \in \mathbb{R}^{d \times p}$

procedure Train($\mathcal{D}$)

  for all minibatches $\{(x, h)\}$ **do**

    $\hat{x} \leftarrow x$

    for $k = 1$ to L **do**

      $b_f \leftarrow \text{Broadcast}(V_{k,f}^{\top} h)$

      $b_g \leftarrow \text{Broadcast}(V_{k,g}^{\top} h)$

      $z_1 \leftarrow \text{MaskedConv}(W_{k,f}, \hat{x}) + b_f$

      $z_2 \leftarrow \text{MaskedConv}(W_{k,g}, \hat{x}) + b_g$

      $\hat{x} \leftarrow \tanh(z_1) \odot \sigma(z_2)$ ▷ Gated activation

      $\hat{x} \leftarrow \hat{x} + \text{Residual}(\hat{x})$ ▷ Optional residual connection

    end for

    $\hat{p}(x) \leftarrow$ Softmax over 256 bins per channel

    $\mathcal{L} \leftarrow -\sum_{i=1}^{H \cdot W \cdot 3} \log p(x_i \mid x_{<i}, h)$

    Backpropagate and update weights

  end for

end procedure

procedure Sample(h)

  Initialize image $x \in \mathbb{R}^{H \times W \times 3}$ with zeros

  for $i = 1$ to $H \cdot W \cdot 3$ in raster order **do**

    Extract $x_{<i}$

    Run forward pass through the L-layer gated PixelCNN with condition h

    Sample $x_i \sim p(x_i \mid x_{<i}, h)$ from the output softmax

    Insert x_i into x

  end for

  return x

end procedure