CS 330

Paper Review

Published as a conference paper at ICLR 2018

FEW-SHOT AUTOREGRESSIVE DENSITY ESTIMATION: TOWARDS LEARNING TO LEARN DISTRIBUTIONS

S. Reed, Y. Chen, T. Paine, A. van den Oord, S. M. A. Eslami, D. Rezende, O. Vinyals, N. de Freitas
{reedscot,yutianc,tpaine}@google.com

Motivation

Learning to learn distributions

- **Why Learn distributions aka learn p(x)?**
  - **To generate data. But why generate data?**

- **Why learning to learn distributions?**
  - **For quick (few-shot) learning & generation of test tasks!**

Problem Statement

Learning Set-Up

What is a Task? Given a support set of images generate an image that looks similar to the support set! To generate: Sample $x' \sim p(x \mid s ; \theta)$

Training Tasks:

Testing Tasks:

Central Goal: Use the training tasks for learning* how to 'quickly' learn distributions so as to do Few Shot Image Generation on test tasks!

*neural attention * meta learning

Method Overview

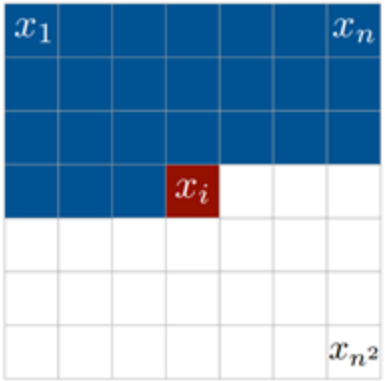
Pre-requisites: Autoregressive models

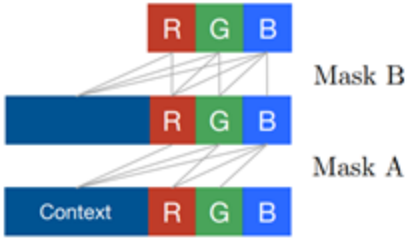
- **Modelling Assumption: We are using parameterized functions to predict next pixel given all the previous pixels.**
  - **Sequential Ordering assumed to break joint distribution into product of marginals (chain rule)**

$$P(x;\theta) = \prod p(x_i|x_{<i};\theta)$$

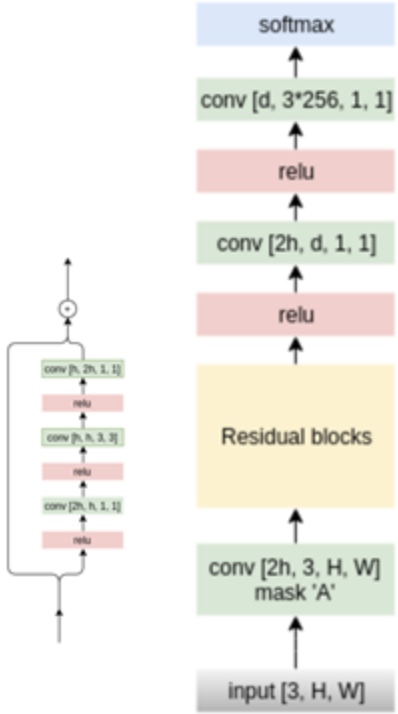
Pre-requisites: PixelCNN

Method Overview





Loss (generated, target)
Energy Distance as loss.



Each pixel is going to have a probability distribution that is a function of all the (sub-) pixels that came before it (for RGB images, each of "R", "G", "B" are treated as separate sub-pixels)

- PixelCNN Authors

Method Overview

Pre-requisites: Attention

Encoder

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]}$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]}$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_{\boldsymbol{c}}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]}$$

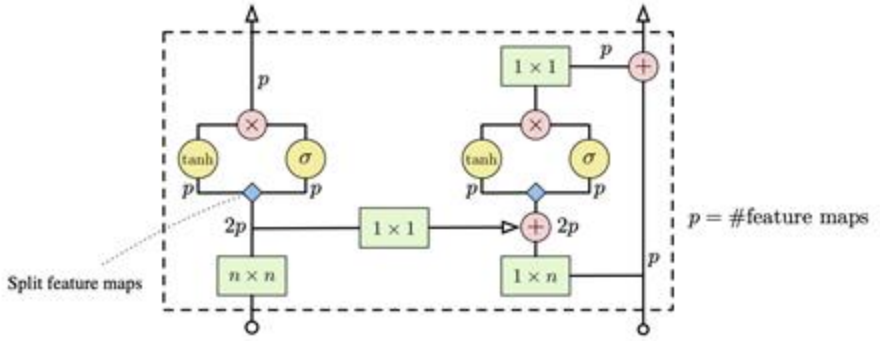
Model Setup

Baseline: Conditional PixelCNN (Gating)

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

Explaination: "One potential advantage is that PixelRNNs contain **multiplicative units** (in the form of the LSTM gates), which may help it to model more **complex interactions**. To amend this we replaced the rectified linear units between the masked convolutions in the original pixelCNN with the following gated activation unit" - C-PixelCNN Authors

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



Model Setup

Baseline: Conditional PixelCNN

Key Idea: Given a high-level image description represented as a latent vector h, we model the conditional distribution p(x|h) of images suiting the description

$$p(x|h) = \prod_{i=1}^{n^2} p(x_i|x_{<i}, h)$$

Why not use a summary vector representing the support set h = f(s)
f(s) is just a learned encoding of the support set!

$$p(x|f(s)) = \prod_{i=1}^{n^2} p(x_i|x_{<i}, f(s))$$

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x} + V_{k,f}^T \mathbf{h}) \odot \sigma(W_{k,g} * \mathbf{x} + V_{k,g}^T \mathbf{h})$$

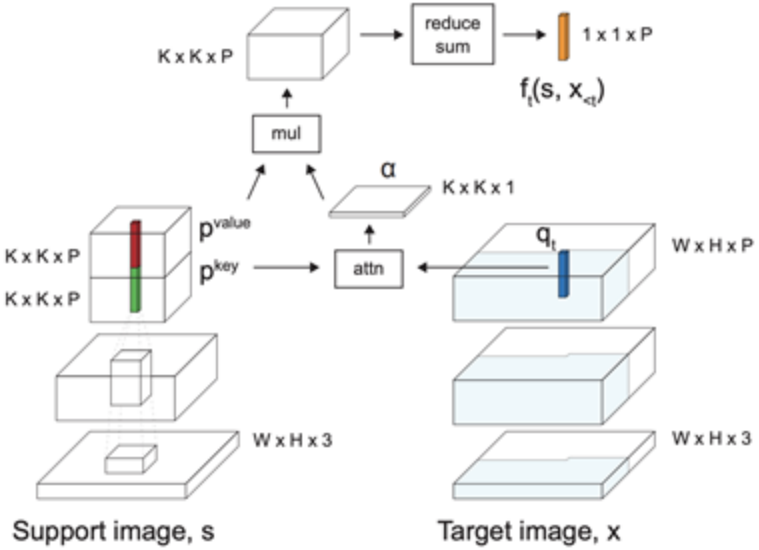
Attention PixelCNN

Proposal 1: Attention PixelCNN
(explicit conditioning with attention)

Challenge: conditional PixelCNN works, the **encoding f(s) was shared across all pixels**.

Key Idea: "different points of generating the target image x, different aspects of the support images may become relevant." -- Learning to learn distributions Authors

Positional Features: Supporting images augmented with a channel encoding position within the image normalized to [−1, 1]



$$p = f_{patch}(s) = \text{reshape}(\text{CNN}(s),\ [SK^2 \times 2P])$$
$$p^{key} = p[:, 0:P],\ p^{value} = p[:, P:2P]$$

$$q_t = \text{PixelCNN}_L(f(s), x_{<t}),$$

$$e_{tj} = v^T \tanh(q_t + p_j^{key})$$

$$\alpha_{tj} = \exp(e_{tj}) / \sum_{k=1}^{SK^2} \exp(e_{ik}).$$

$$f_t(s, x_{<t}) = \sum_{j=1}^{SK^2} \alpha_{tj} p_j^{value}$$

$$P(\mathbf{x}|s;\theta) = \prod_{t=1}^{N} P(x_t|x_{<t}, f(s);\theta)$$

Loss(generated, target)

Meta PixelCNN

Proposal 2: Meta PixelCNN

(implicit conditioning with gradient descent)

Key Idea: The conditioning pathway (i.e. flow of information from supports s to the next pixel xt) introduces no additional parameters.

$$\mathcal{L}(x, s; \theta) = -\log P(x; \theta'), \text{ where } \theta' = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{inner}}(s; \theta)$$

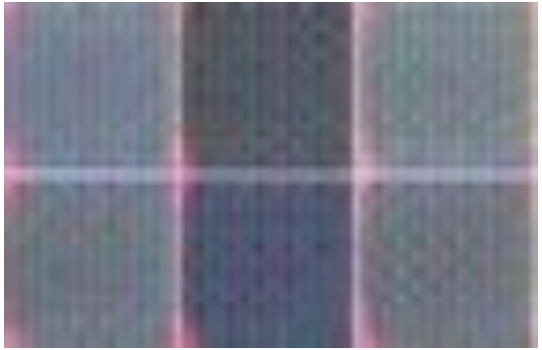
The features q are fed through a convolutional network g (parameters included in θ) producing a scalar, which is treated as the **learned inner loss**.

Algorithm 1 Meta PixelCNN training

θ: Randomly initialized model parameters
$p(s, x)$: Distribution over support sets and target outputs.
while not done **do** ▷ Training loop
  $\{s_i, x_i\}_{i=1}^{M} \sim p(s, t)$. ▷ Sample a batch of M support sets and target outputs
  for all s_i, x_i **do**
    $q_i = PixelCNN_L(s_i, \theta)$ ▷ Compute support set embedding as L-th layer features
    $\theta_i' = \theta - \alpha \nabla_\theta g(q_i, \theta)$ ▷ Adapt θ using $\mathcal{L}_{inner}(s_i, \theta) = g(q_i, \theta)$
  $\theta = \theta - \beta \nabla_\theta \sum_i -\log P(x_i, \theta_i')$ ▷ Update parameters using maximum likelihood

Experiments

Tasks



Character & Image Generation

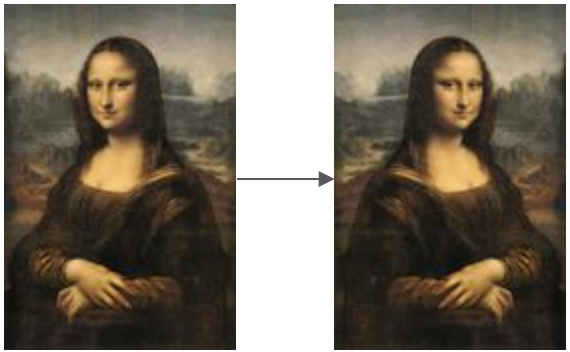


Image Inversion

Task Difficulty

Experiments

Datasets

Stanford Online Product (SOP)

Omniglot

ImageNet

Task Difficulty

Experiments

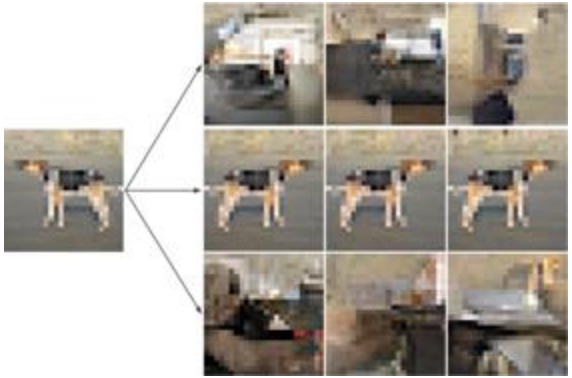
Evaluation Metrics

- **Qualitative and Quantitative**
- **Nats: a unit of information or entropy, based on natural logarithms and powers of *e***

$$-ln(P(x_i \mid x_{<i}, s; \theta))$$

Image Inversion with ImageNet

1-shot Image Generation



Conditional PixelCNN

Attention PixelCNN

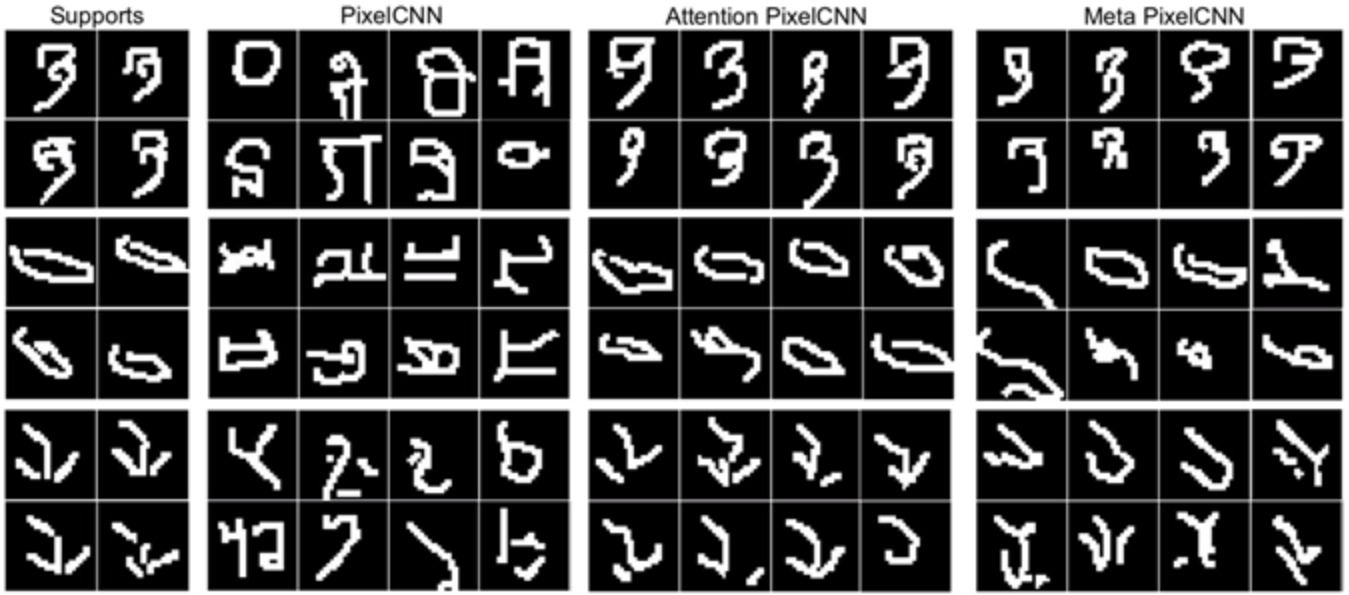
Meta PixelCNN

Model	Performance - test(train)
Conditional PixelCNN	2.65 (2.64) nats/dim
Attention PixelCNN	0.90 (0.89) nats/dim
Meta PixelCNN	– (–)

Attention PixelCNN's attention head learns to move and copy in a right-to-left order while the output writes left-to-right.

Character Generation with Omniglot

Few-shot Character Generation



Character Generation with Omniglot

Few-shot Character Generation

Model	Number of support set examples			
	1	**2**	**4**	**8**
Bornschein et al. (2017)	0.128(−−)	0.123(−−)	0.117(−−)	−−(−−)
Gregor et al. (2016)	0.079(0.063)	0.076(0.060)	0.076(0.060)	0.076(0.057)
Conditional PixelCNN	0.077(0.070)	0.077(0.068)	0.077(0.067)	0.076(0.065)
Attention PixelCNN	**0.071(0.066)**	**0.068(0.064)**	**0.066(0.062)**	**0.064(0.060)**

PixelCNN Model	NLL test(train)
Conditional PixelCNN	0.077(0.067)
Attention PixelCNN	0.066(0.062)
Meta PixelCNN	0.068(0.065)
Attention Meta PixelCNN	0.069(0.065)

Few-shot Character Generation

Character Generation with Omniglot

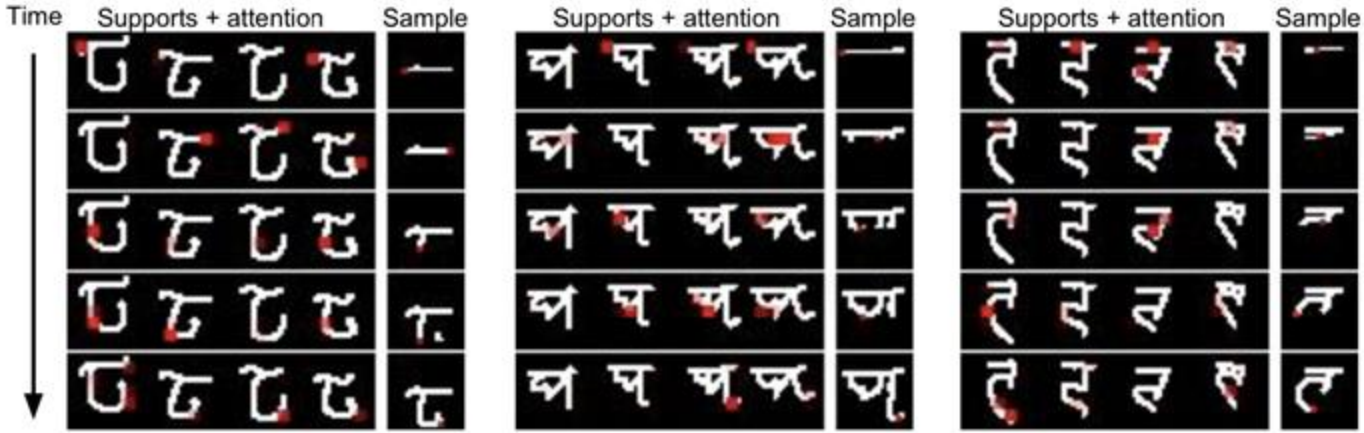
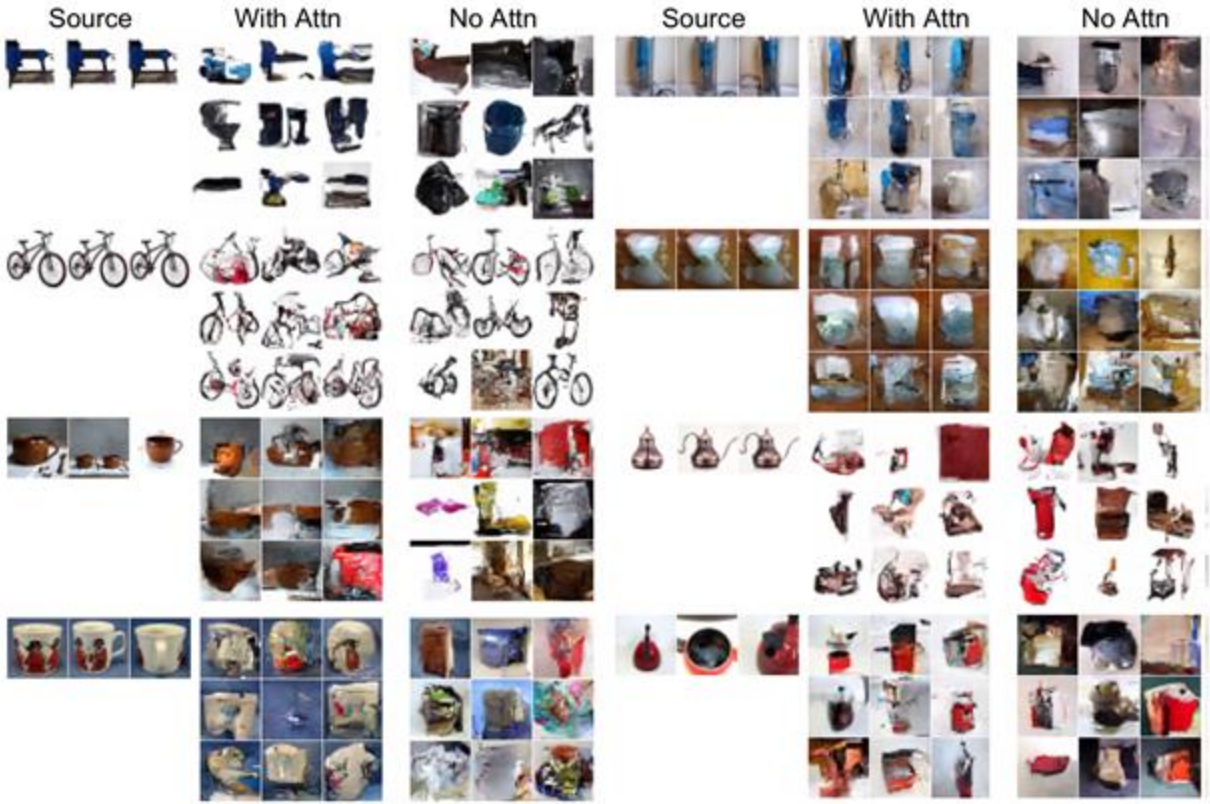


Image Generation with SOP

Few-shot Image Generation



Takeaways

Strengths:

- **Attention is great for flipping images! (one-shot generation)**
- **Meta generative models can generate unseen characters.**
- **Inner loss function is learnable.**

Weaknesses:

- **Few shot image generation needs a new model.**
- **No analysis on inner loop gradient steps vs performance.**
- **Naive combination of meta learning and attention.**
- **Inconsistent experiments.**

Discussion & Future Work

- **Why Meta-PixelCNN is unable to perform well on one-shot generation (experiments on Imagenet Flipping)?**
- **Would multiple gradient steps in the inner loop of meta learning improve performance?**
- **Sophisticated combination of attention & meta-learning?**
  - **Attentive Meta-Learning**
- **Learned Inner Loss: Since the loss function is learned and unconstrained, how are we guaranteed that it is actually emulating the loss on the task?**

Ground Truth

Your work is really great and we were able to understand most of the details. The following were some of our questions:

1. We were wondering if you had some intuitions on why Meta-PixelCNN is unable to perform well on one-shot generation (experiments on Imagenet Flipping)?
   Our potential explanation after reading the paper was that Attention-PixelCNN is utilizing pixel positions but the raw Meta-PixelCNN is not using the positional features - which makes it easy for the Attention-PixelCNN to learn flipping while for Meta-PixelCNN is harder. Correct us if we're wrong!

That sounds like a plausible explanation.

2. In the inner loop of Meta-PixelCNN -- are you guys doing only 1 gradient step? If yes, did y'all experiment with multiple gradient steps vs single gradient step?

I had tried with multiple steps, but it slowed training quite significantly and I did not see a benefit, so I think I used only 1 in the end.

3. You all mentioned that future work forms more sophisticated combination of attention with meta-learning. We were wondering if you could elaborate on that. Also, what did you mean by 'varying the inner loss function'. We saw that y'all fix the model family for for inner loss g (by fixing the architecture for g) but learn the parameters of the loss per task. I'd call this varying the inner loss function, so was wondering what you meant by varying it and how would you learn it?

There could be many other choices of inner loss function - they could be potentially any unsupervised or self supervised loss, or something entirely learned by another neural net. We explored this a bit, but certainly much more can be done.

We would be grateful if you could answer these questions. We're pressed a bit on time, the class only started last week and we got to know about our presentation only yesterday. Will appreciate prompt reply on this!