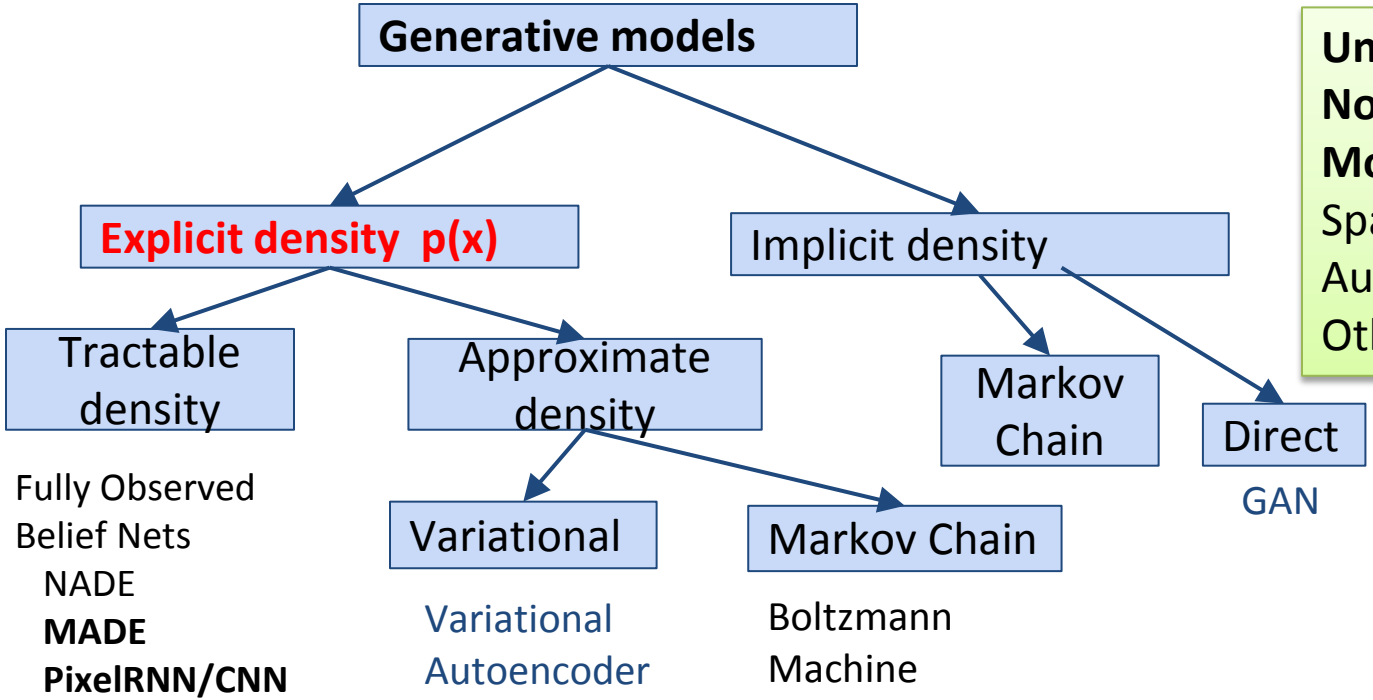
Deep Unsupervised Learning

7 March 2019

Taxonomy of Generative Models



Unsupervised Learning
Non-probabilistic Models
Sparse Coding
Autoencoders
Others (e.g. k-means)

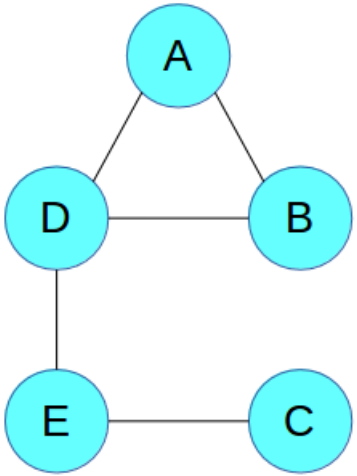
Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Graphical Models

- Probabilistic graphical models provide a powerful framework for representing dependency structure between random variables.
- Graphical models offer several useful properties:
  - They provide a simple way to visualize the structure of a probabilistic model and can be used to motivate new models.
  - They provide various insights into the properties of the model, including conditional independence.
  - Complex computations (e.g. inference and learning in sophisticated models) can be expressed in terms of graphical manipulations.

Graphical Models

- In a probabilistic graphical model, each node represents a random variable, and links represent probabilistic dependencies between random variables.
- The graph specifies the way in which the joint distribution over all random variables decomposes into a product of factors, where each factor depends on a subset of the variables.
- Two types of graphical models:
  - Bayesian networks, also known as Directed Graphical Models (the links have a particular directionality indicated by the arrows)
  - Markov Random Fields, also known as Undirected Graphical Model (the links do not carry arrows and have no directional significance).
- Hybrid graphical models that combine directed and undirected graphical models, such as Deep Belief Networks.



Bayesian Networks

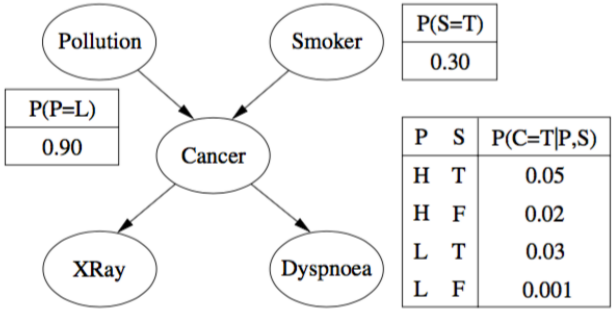
- Directed Graphs are useful for expressing causal relationships between random variables.
- Let us consider an arbitrary joint distribution $p(a, b, c)$ over three random variables a, b, and c.
- By application of the product rule of probability, we get

$$p(a, b, c) =$$

- This decomposition holds for any choice of the joint distribution.

Bayes nets and neural nets

- Main idea: place a Bayes net structure (a directed acyclic graph) over the variables in the data, and model the conditional distributions with neural networks.
- Reduces the problem to designing conditional likelihood-based models for single variables.
  We know how to do this: the neural net takes variables being conditioned on as input, and outputs the distribution for the variable being predicted.



E 2.1
or the lung cancer problem.

Fully Observable Model

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Then maximize likelihood of training data

Autoregressive models

- First, given a Bayes net structure, setting the conditional distributions to neural networks will yield a tractable log likelihood and gradient. Great for maximum likelihood training!

$$\log p_\theta(x) = \sum_{i=1}^{d} \log p_\theta(x_i | parents(x_i))$$

- But is it expressive enough? Yes, assuming a fully expressive Bayes net structure: any joint distribution can be written as a product of conditionals

$$\log p(\mathbf{x}) = \sum_{i=1}^{d} \log\ p\ (x_i | \mathbf{x}_{1:i-1})$$

- This is called an **autoregressive model**. So, an expressive Bayes net structure with neural network conditional distributions yields an expressive model for p(x) with tractable maximum likelihood training.

A toy autoregressive model

- Two variables: x1, x2
- Model: p(x1, x2) = p(x1) p(x2|x1)
  - p(x1) is a histogram
  - p(x2|x1) is a multilayer perceptron
    - Input is x1
    - Output is a distribution over x2 (logits, followed by softmax)

One function approximator per conditional

Does this extend to high dimensions?

- Somewhat. For d-dimensional data, O(d) parameters
  - Much better than O(exp(d)) in tabular case
  - What about text generation where d can be arbitrarily large?
- Limited generalization
  - No information sharing among different conditionals

Solution: share parameters among conditional distributions.

Two approaches:

- Recurrent neural networks
- Masking

Fully Observable Model

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Will need to define ordering of "previous pixels"

Then maximize likelihood of training data

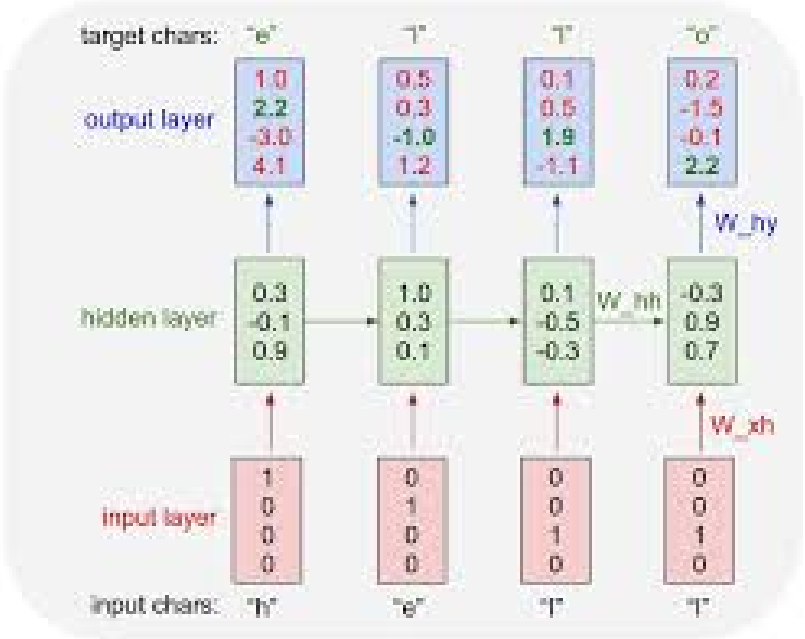
Complex distribution over pixel values => Express using a neural network!

RNN autoregressive models - char-rnn

$$\log p(\mathbf{x}) = \sum_{i=1}^{d} \log\, p\,(x_i|\mathbf{x}_{1:i-1})$$

Sequence of characters

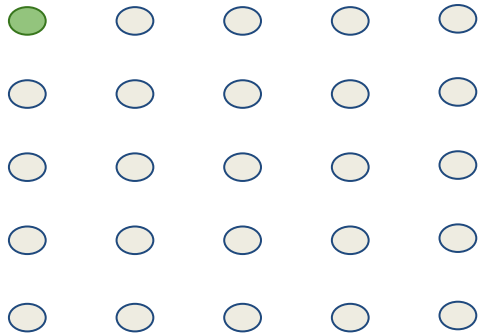
Character at *i*th position



PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

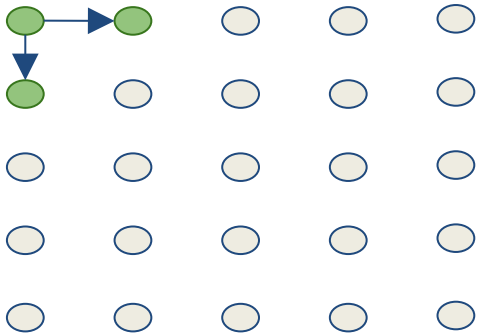
Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

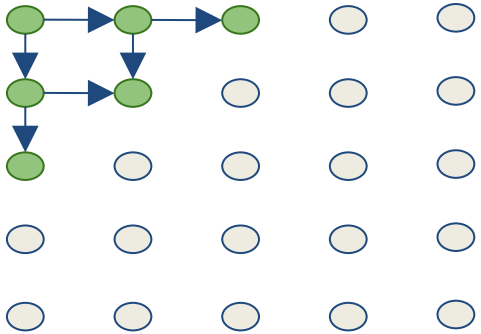
Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

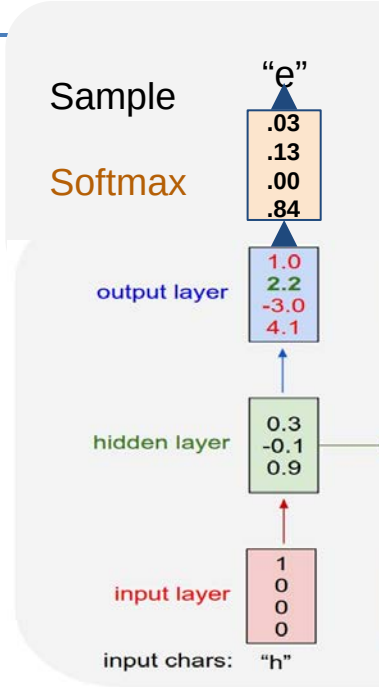


Test Time: Sample / Argmax / Beam Search

Example: Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

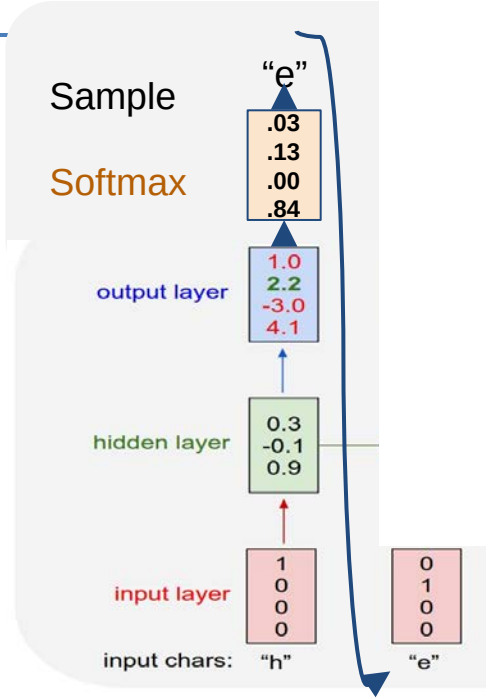


Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



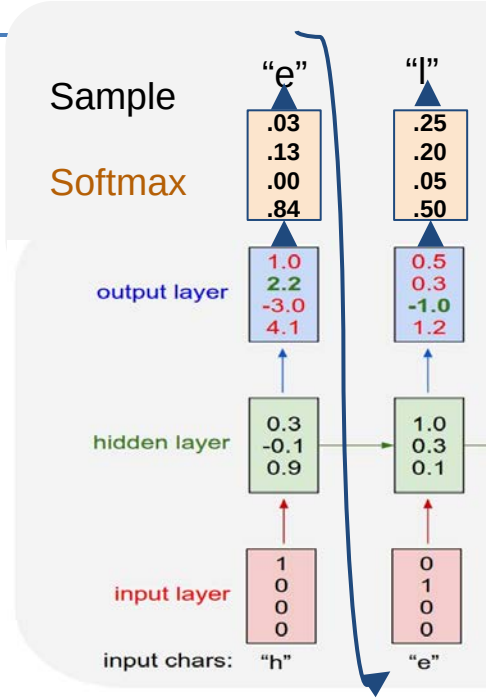
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

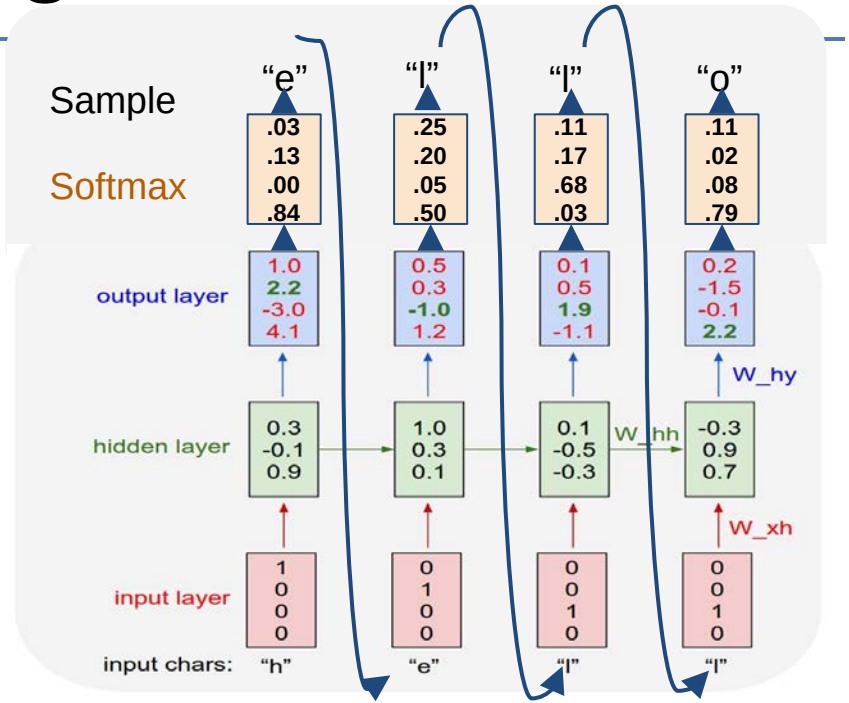


Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

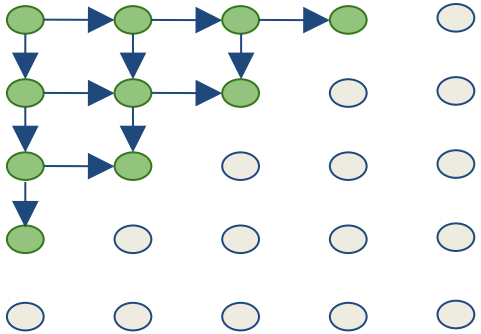


PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!

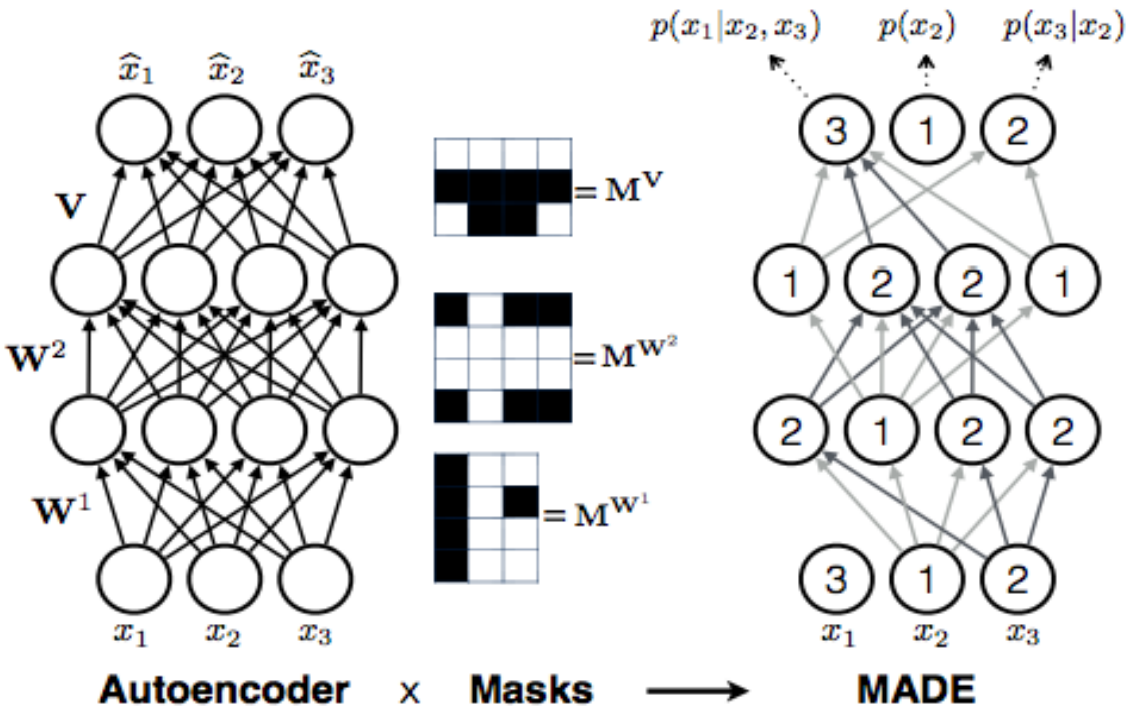


Masking-based autoregressive models

Second major branch of neural AR models

- Key property: parallelized computation of all conditionals
- Masked MLP (MADE)
- Masked convolutions & self-attention
  - Also share parameters across time

Masked Autoencoder for Distribution Estimation (MADE)

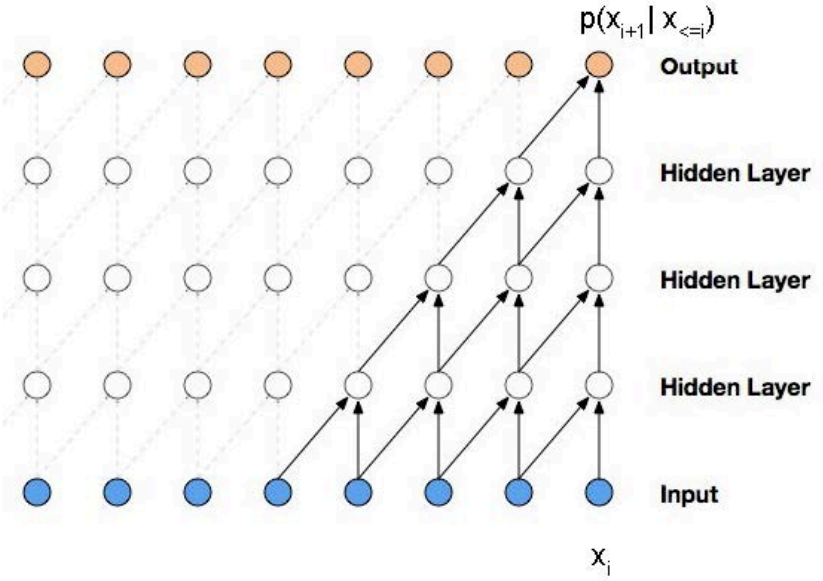


Use masks to disallow certain paths
Suppose ordering is x2, x3, x1.

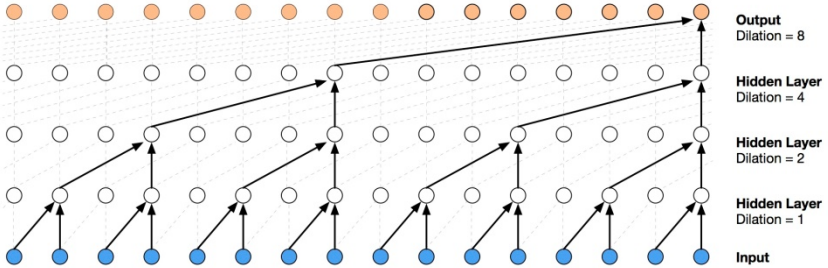
1. The unit producing the parameters for p(x2) is not allowed to depend on any input. Unit for p(x3|x2) only on x2. And so on...
2. For each unit, pick a number i in [1, n − 1]. That unit is only allowed to depend only on the first i inputs.
3. 3 Add mask to preserve this invariant: connect to all units in previous layer with smaller or equal assigned number (strictly < in final layer)

Masked Temporal (1D) Convolution

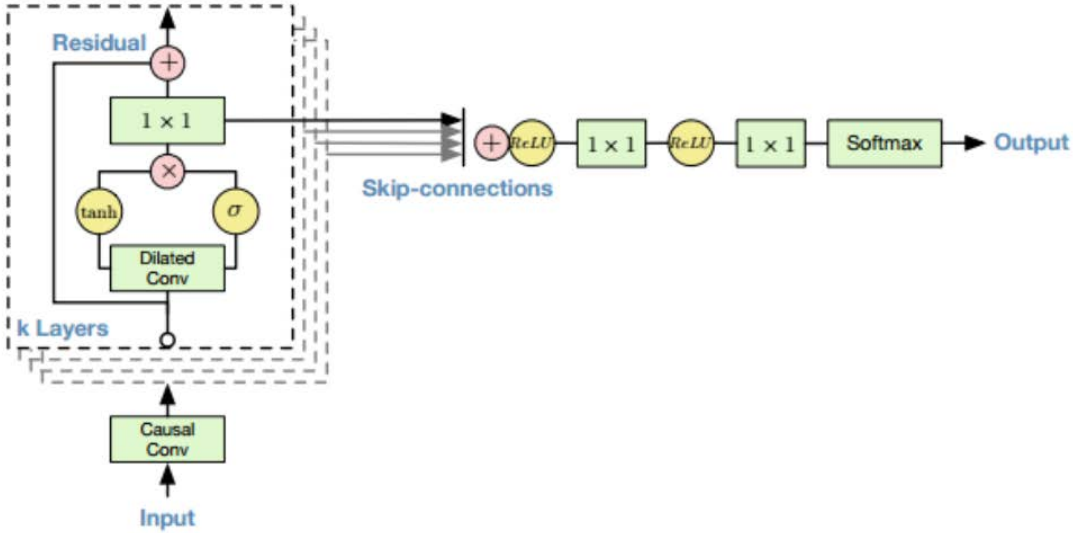
- Easy to implement, masking part of the conv kernel
- Constant parameter count for variable-length distribution!
- Efficient to compute, convolution has hyper-optimized implementations on all hardware
- However
  - Limited receptive field, linear in number of layers



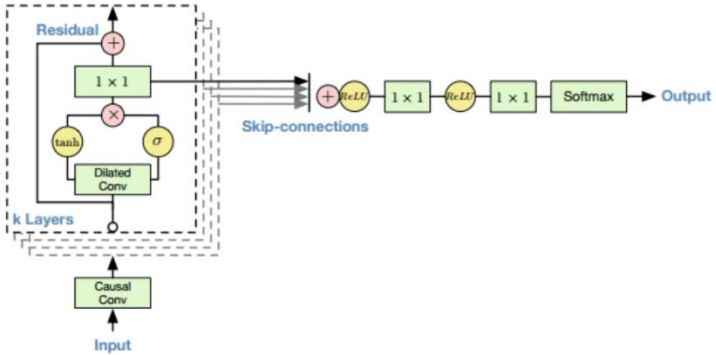
WaveNet



Improved receptive field: dilated convolution, with exponential dilation
Better expressivity: Gated Residual blocks, Skip connections



Wavenet Gated Activation Units



- Gated activation units, same as used in gated PixelCNN (van den Oord et al., 2016)

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

  - $*$: convolution; $\odot$: elementwise multiplication
  - σ: sigmoid function
  - k: layer index; f: filter; g: gate
  - W: learned convolution filter
- Empirically found to outperform ReLUs
- Feedforward nets with gated units found beneficial in other works:
  - Highway networks (Srivastava et al., NIPS 2015)
  - Grid LSTM (Kalchbrenner et al. 2016, ICLR 2016)
  - Neural GPUs (Kaiser et al. 2016, ICLR 2016)

Conditional Wavenet

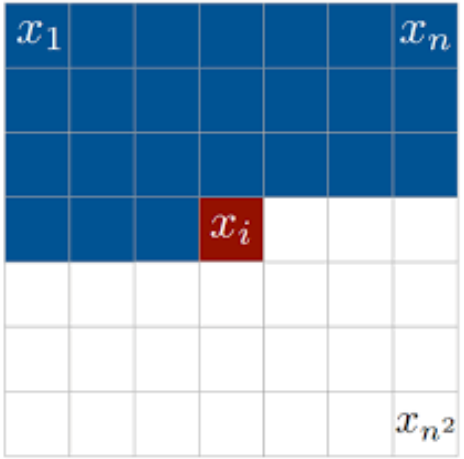
- Given an aadditional $\mathbf{h}$, we model the conditional distribution $p(\mathbf{x}|\mathbf{h})$ of the waveform:

$$p(\mathbf{x}|\mathbf{h}) = \prod_{t=1}^{T} p(x_t|x_1, \ldots, x_{t-1}, \mathbf{h}).$$

- Conditioning on other variables enables audio generation with desired characteristics
- Examples:
  - In multi-speaker setting, can pass in speaker identity
  - For text-to-speech, can pass in information about text as extra input

Masked Spatial (2D) Convolution - PixelCNN

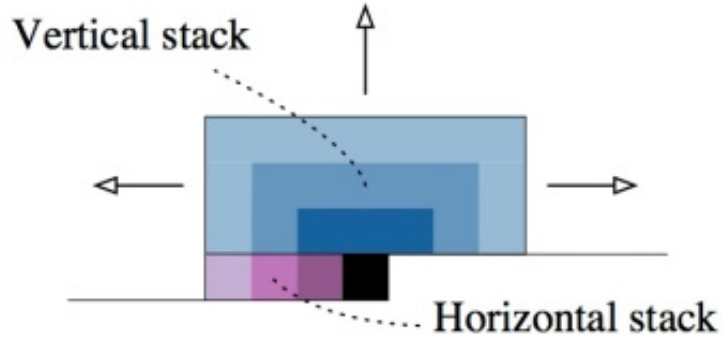
- Images can be flattened into 1D vectors, but they are fundamentally 2D
- We can use a masked variant of ConvNet to exploit this knowledge
- First, we impose an autoregressive ordering on 2D images:



The conditional pixel CNN

They propose two convnet stacks:

- The vertical stacks see all the pixel above (not including current row, the blue parts in the following image)
- the horizontal stacks see all the pixels left in the row (purple area in the following image).

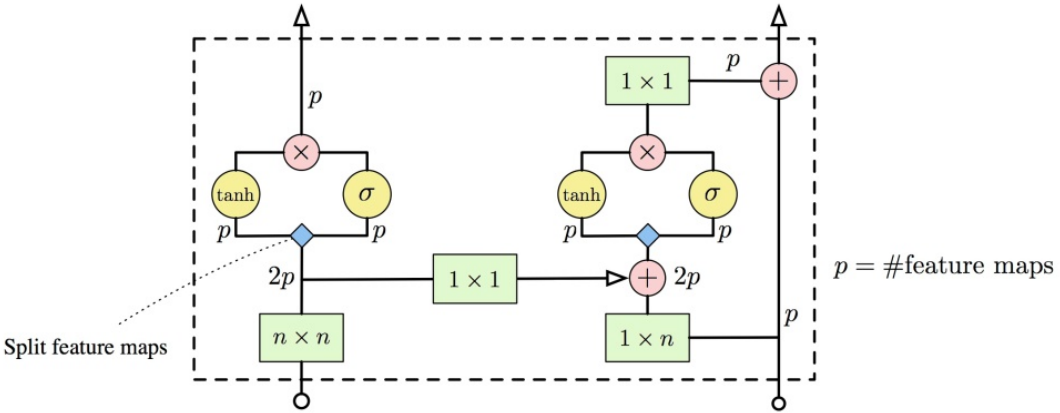


To avoid seeing future pixels, they use the masked convolution. In addition, they use gated activation unit, the formula is like this:

$$y=\tanh(W_f * x) \odot \sigma(W_g * x)$$

Gated PixelCNN

- A single layer in the Gated PixelCNN architecture. Convolution operations are shown in green, element-wise multiplications and additions are shown in red. The convolutions with W_f and W_g are combined into a single operation shown in blue, which splits the 2p features maps into two groups of p



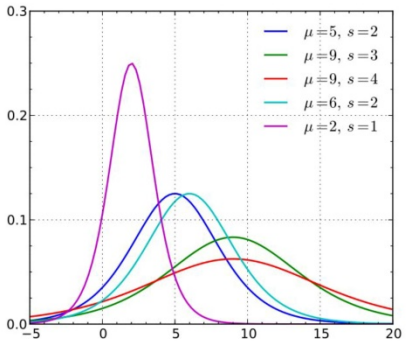
PixelCNN++

- Moving away from softmax: we know nearby pixel values are likely to co-occur!

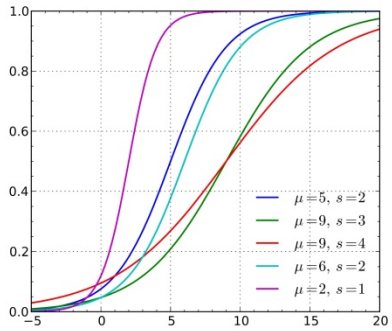
$$\nu \sim \sum_{i=1}^{K} \pi_i \text{logistic}(\mu_i, s_i)$$

$$P(x|\pi, \mu, s) = \sum_{i=1}^{K} \pi_i \left[\sigma((x + 0.5 - \mu_i)/s_i) - \sigma((x - 0.5 - \mu_i)/s_i)\right],$$

- We assume there is a latent color intensity v with a continuous distribution, which is then rounded to its nearest 8-bit representation to give the observed sub-pixel value x.
- Choose a simple continuous distribution for modeling v (like the logistic distribution)
- A mixture of logistic distributions which allows us to easily calculate the probability on the observed discretized value x



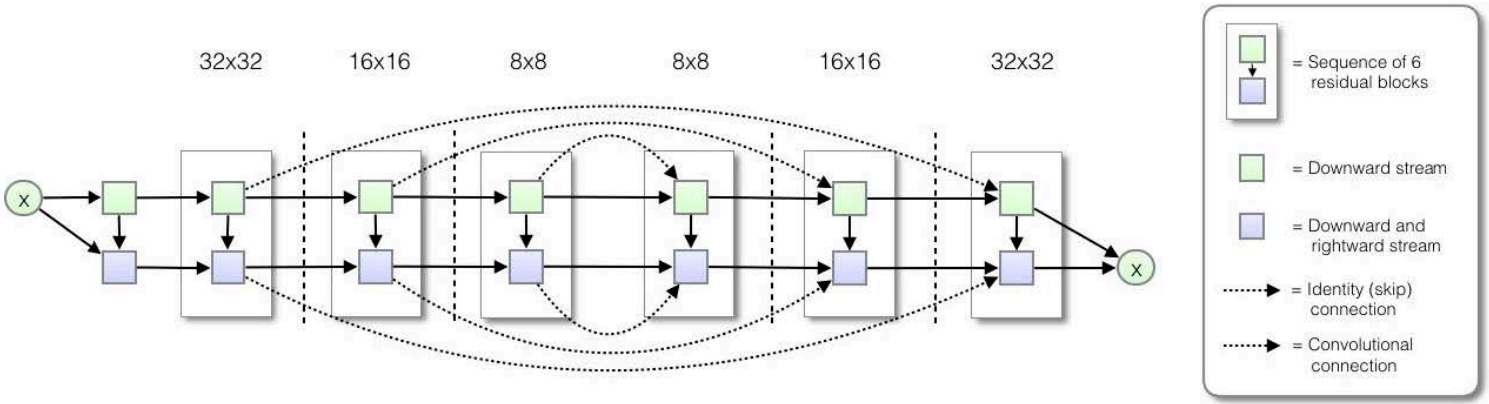
pdf



cdf
= sigmoid((x - mu) / scale)

PixelCNN++

- Capture long dependencies efficiently by downsampling



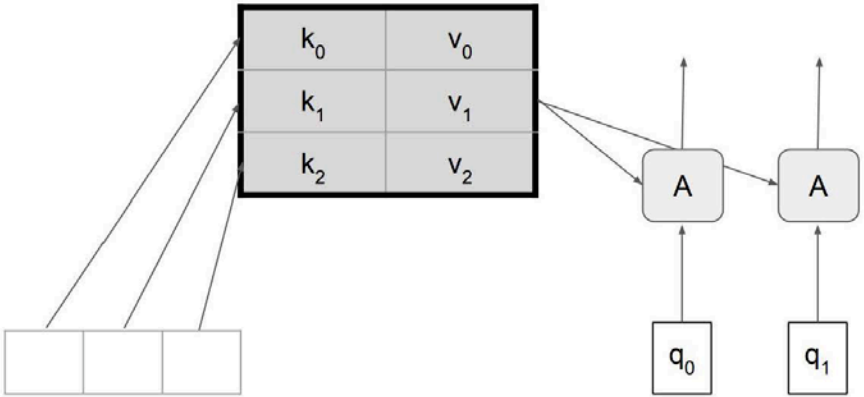
Masked Attention

- A recurring problem for convolution: limited receptive field -> hard to capture long-range dependencies
- (Self-)Attention: an alternative that has
  - unlimited receptive field!!
  - O(1) parameter scaling w.r.t. data dimension
  - parallelized computation (versus RNN)

Attention

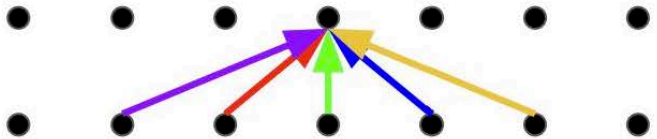
Dot-Product Attention

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

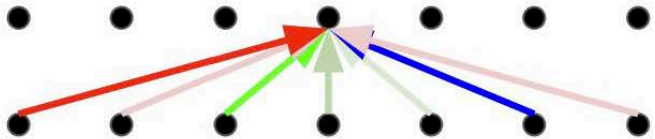


Self-attention when q_i also generated from x

Self-Attention



Convolution

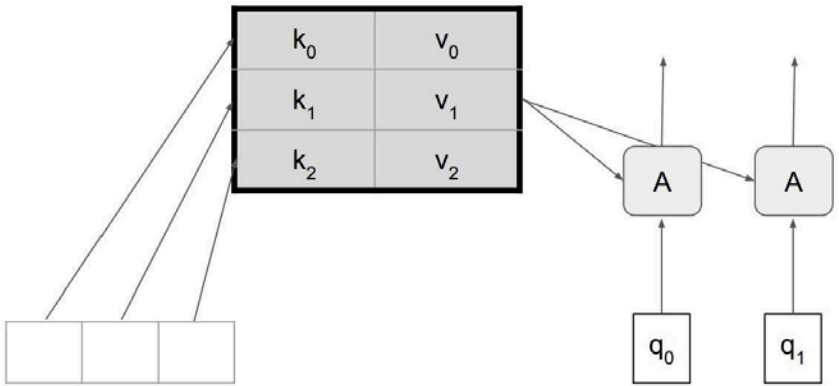


Self-attention

Masked Attention

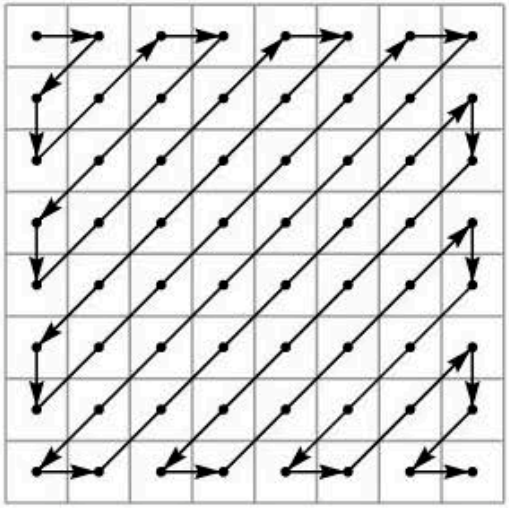
Dot-Product Attention

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i} \text{ - masked}(k_i, q) * 10^{10}}{\sum_j e^{q \cdot k_j} \text{ - masked}(k_i, q) * 10^{10}} v_i$$



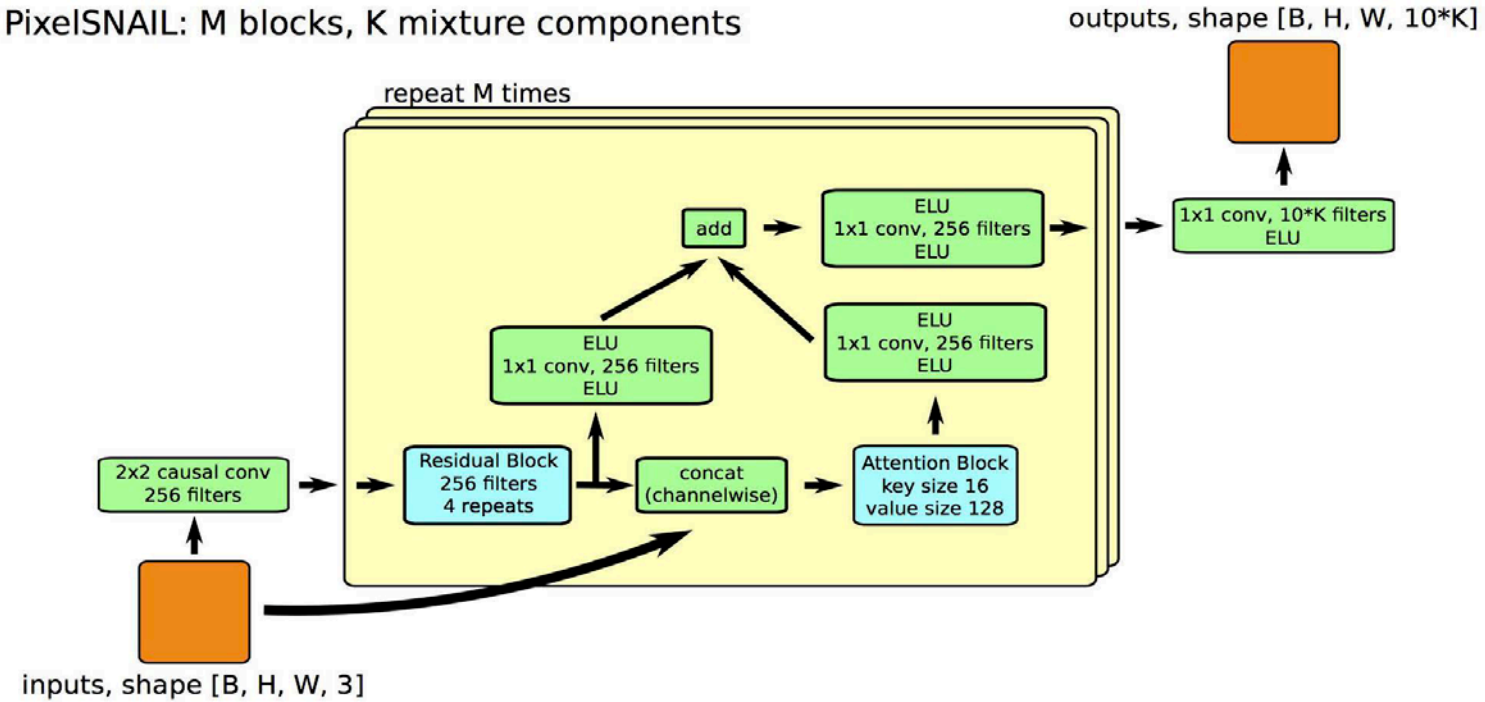
Masked Attention

- Much more flexible than masked convolution. We can design any autoregressive ordering we want
- An example:

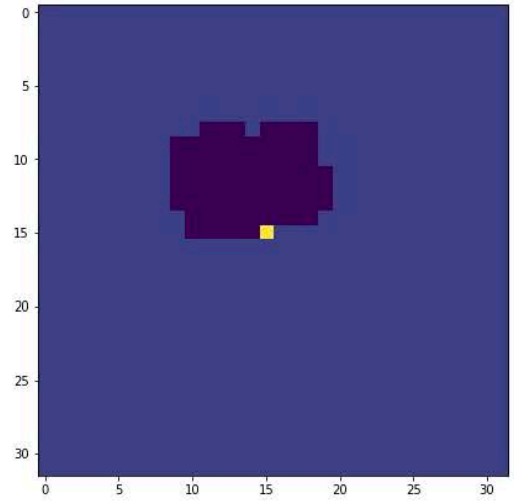


Zigzag ordering
- How to implement with masked conv?
- Trivial to do with masked attention!

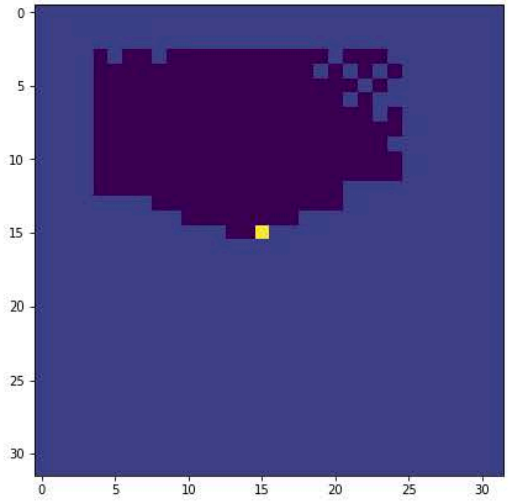
Masked Attention + Convolution



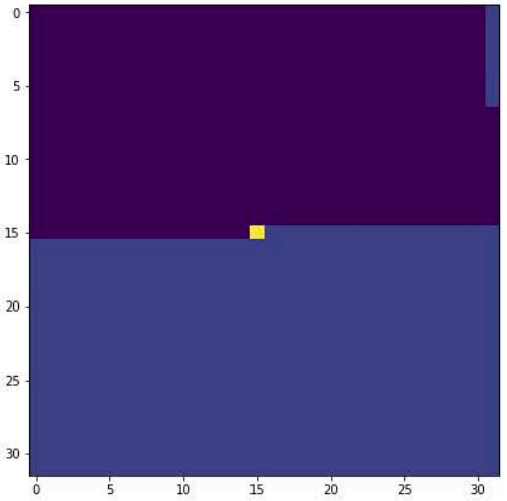
Masked Attention + Convolution



Gated PixelCNN



PixelCNN++



PixelSNAIL

Masked Attention + Convolution

Method	CIFAR-10
Conv DRAW (Gregor et al., 2016)	3.5
Real NVP (Dinh et al., 2016)	3.49
VAE with IAF (Kingma et al., 2016)	3.11
PixelRNN (Oord et al., 2016b)	3.00
Gated PixelCNN (van den Oord et al., 2016b)	3.03
Image Transformer (Anonymous, 2018)	2.98
PixelCNN++ (Salimans et al., 2017)	2.92
Block Sparse PixelCNN++ (OpenAI, 2017)	2.90
PixelSNAIL (ours)	**2.85**

Neural autoregressive models:

Advantages

- Best in class modelling performance:
  - expressivity - autoregressive factorization is general
  - generalization - meaningful parameter sharing has good inductive bias

-> State of the art models on multiple datasets, modalities

Disadvantages

- Sampling each pixel = 1 forward pass!
- 11 minutes to generate 16 32-by-32 images on a Tesla K40 GPU

Speedup

- Speedup by caching activations
- Speedup by breaking autoregressive pattern

Bibliography

char-rnn: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

MADE: Germain, Mathieu, et al. "Made: Masked autoencoder for distribution estimation." *International Conference on Machine Learning*. 2015.

WaveNet: Oord, Aaron van den, et al. "Wavenet: A generative model for raw audio." *arXiv preprint arXiv:1609.03499* (2016).

PixelCNN: Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks." *arXiv preprint arXiv:1601.06759* (2016).

Gated PixelCNN: Van den Oord, Aaron, et al. "Conditional image generation with pixelcnn decoders." *Advances in Neural Information Processing Systems*. 2016.

PixelCNN++: Salimans, Tim, et al. "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications." *arXiv preprint arXiv:1701.05517* (2017)

Self-attention: Vaswani, Ashish, et al. "Attention is all you need." *Advances in Neural Information Processing Systems*. 2017.

PixelSNAIL: Chen, Xi, et al. "Pixelsnail: An improved autoregressive generative model." *arXiv preprint arXiv:1712.09763* (2017)

Fast PixelCNN++: Ramachandran, Prajit, et al. "Fast generation for convolutional autoregressive models." *arXiv preprint arXiv:1704.06001*(2017).

Multiscale PixelCNN: Reed, Scott, et al. "Parallel multiscale autoregressive density estimation." *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.

Grayscale PixelCNN: Kolesnikov, Alexander, and Christoph H. Lampert. "PixelCNN models with auxiliary variables for natural image modeling." *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.

Subscale Pixel Network: Menick, Jacob, and Nal Kalchbrenner. "Generating High Fidelity Images with Subscale Pixel Networks and Multidimensional Upscaling." *arXiv preprint arXiv:1812.01608*(2018)

Conditional Image Generation with PixelCNN Decoders

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

Grey whale

Tiger

EntleBucher (dog)

Conditional Image Generation with PixelCNN Decoders

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

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

E.g.,
1-hot encoded class label
(for 3 classes a vector of size 3x1)


Grey whale


Tiger


EntleBucher (dog)

How does this change the rest of the equations?

vertical

k=layer; f,g=different feature map

Element-wise multiplication

Convolution with the input

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

vertical

k=layer; f,g=different feature map

Element-wise multiplication

Convolution with the input

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

E.g., 1-hot encoded class label (for 3 classes a vector 3x1)

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

vertical

k=layer; f,g=different feature map

Element-wise multiplication

Convolution with the input

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

More weights with dimensions so *Vh* can be added to *Wx* (e.g., if *Wx* has 32 feature maps, V dims = [3, 32] I think...)

E.g., 1-hot encoded class label (for 3 classes a vector 3x1)

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

vertical

k=layer; f,g=different feature map

Element-wise multiplication

Convolution with the input

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

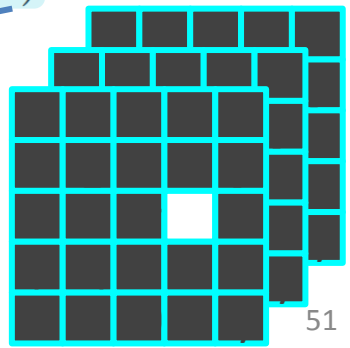
More weights with dimensions so *Vh* can be added to *Wx* (e.g., if *Wx* has 32 feature maps, V dims = [3, 32] I think…)

E.g., 1-hot encoded class label (for 3 classes a vector 3x1)

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

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

Encode the spatial locations of objects for each class …e.g., 3 classes (z-dimension) on a 5x5 image (white pixel encodes the class and location)



vertical

k=layer; f,g=different feature map

Element-wise multiplication

Convolution with the input

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

More weights with dimensions so *Vh* can be added to *Wx* (e.g., if *Wx* has 32 feature maps, V dims = [3, 32] I think...)

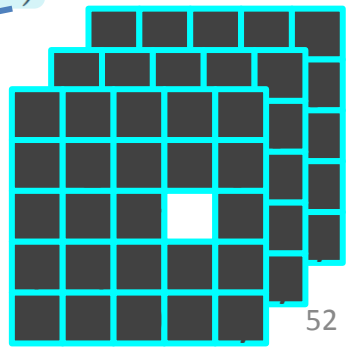
E.g., 1-hot encoded class label (for 3 classes a vector 3x1)

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

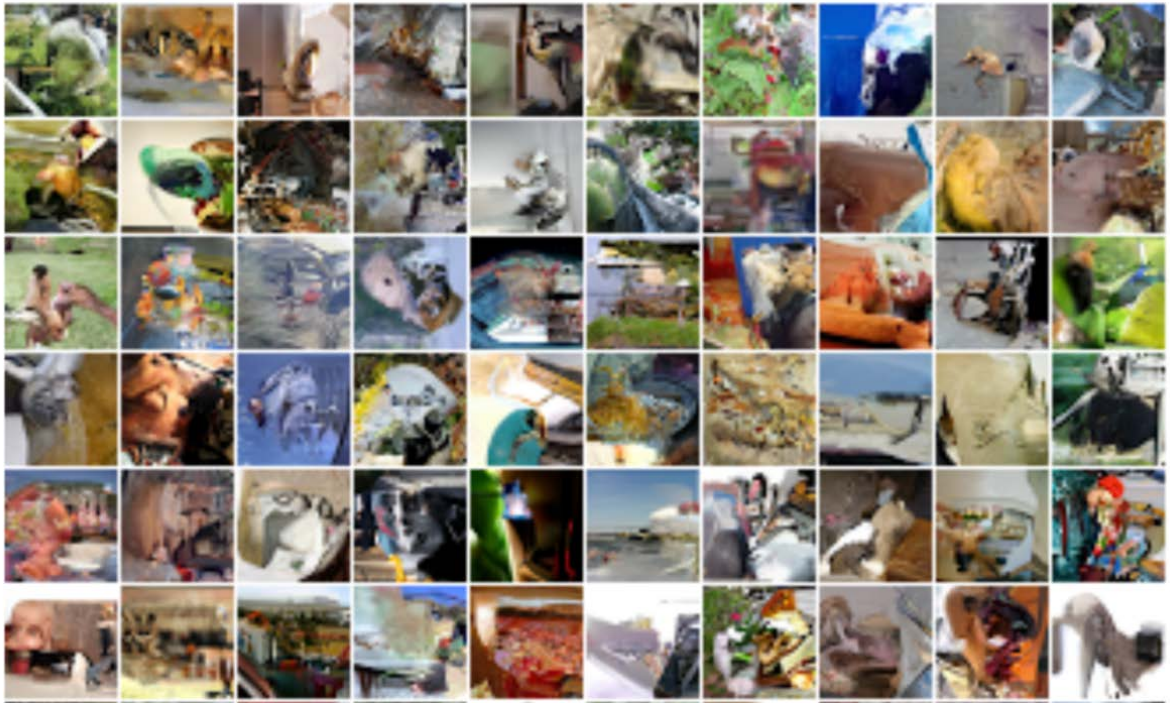
$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x} + V_{k,f} * \mathbf{s}) \odot \sigma(W_{k,g} * \mathbf{x} + V_{k,g} * \mathbf{s})$$

For our examples, weights *V* has dimensions 32, 1, 1, 3

Encode the spatial locations of objects for each class ...e.g., 3 classes (z-dimension) on a 5x5 image (white pixel encodes the class and location)



Experiments (not conditioned)



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

Trained on ImageNet
(image from PixelRNN paper)

Experiments - conditioned

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

Sandbar

Lawn mower

Brown bear

Grey whale

Tiger

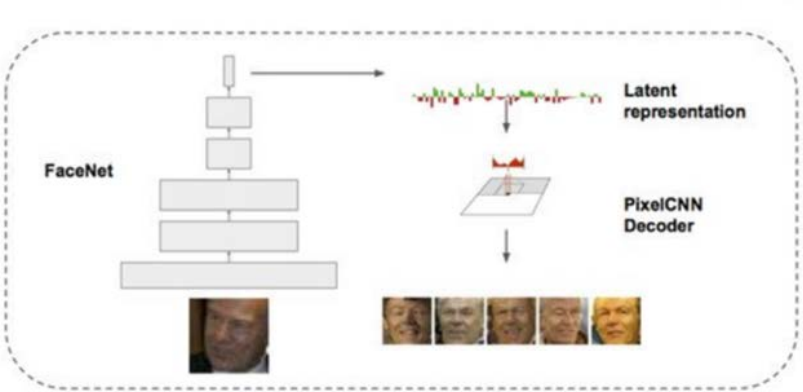
EntleBucher (dog)

Yellow lady's slipper (flower)

Experiments - conditioned on faces

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

Use a CNN trained on faces
Optimized to output a feature vector (embedding) *h* for a face image

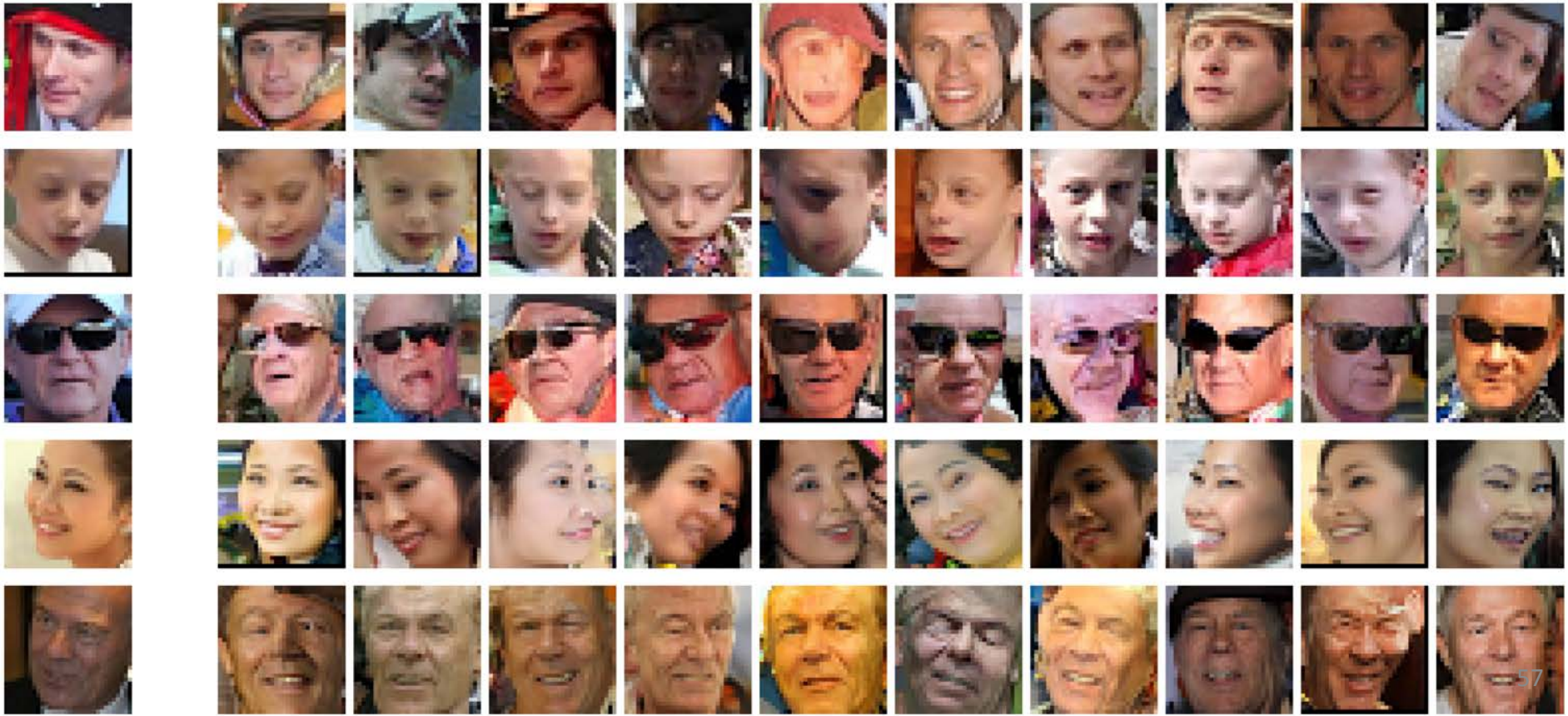


https://twitter.com/avdnoord/status/805027623726448641

Original face image goes through a trained CNN to produce h

Samples from gated-PixelCNN

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



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

Face image goes through a trained CNN to produce h1

Linear interpolation between h1-h2

Face image goes through a trained CNN

Samples from gated-PixelCNN

Summary - PixelCNN

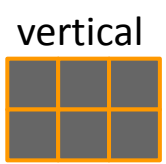
CNN to generate image pixel-by-pixel

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

Efficient (relative to PixelRNN) to train since relies on convolutions

Careful windowing and cropping required to not break conditional dependencies

Can produce an image conditioned on an embedding (e.g., class label, face vector)



a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p	q	r	s	t
u	v	w	x	y

Tiger

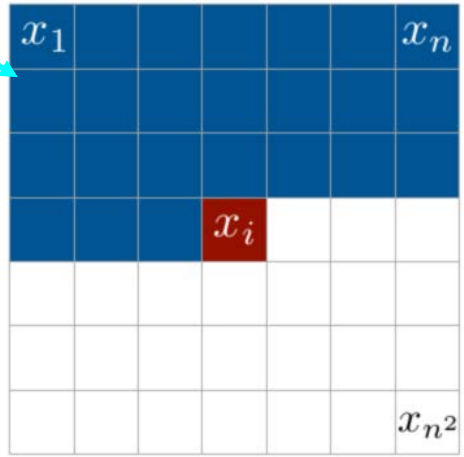
But first let's think about how we might use this to evaluate the probability of an image

Model uses … probability of an image

Number of pixels

Given the known pixel (above and to the left)

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



image

Could pass in different images and compute probabilities for each image

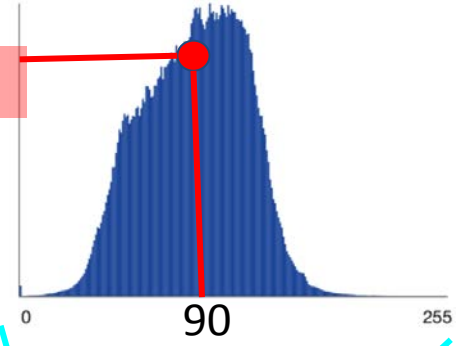
The i-th pixel will have some value

The probability of the i-th pixel having some value given the values of the previous pixels

Evaluate the probability of an image

Corresponding probability

p(xi=90|x1,...,xi-1)



Intensity value

90

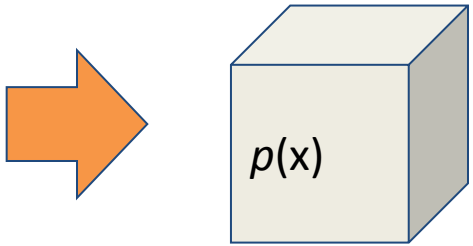
n

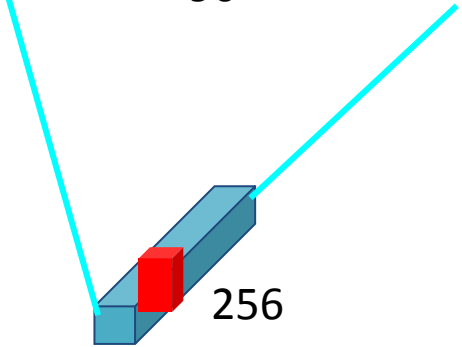
1

n

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

Model p(x)



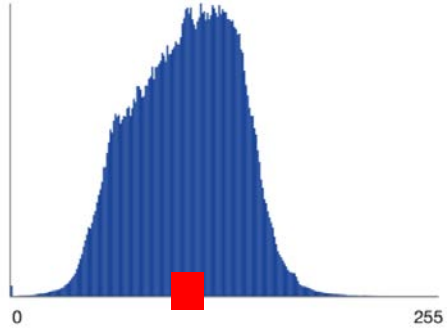


Predict the grayscale value

Grey-scale to simplify figure (color works too)

Evaluate the probability of an image

Corresponding probability



Intensity value

90

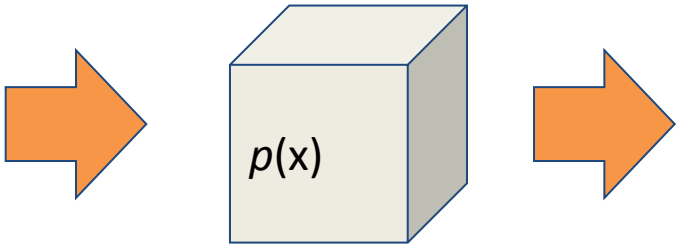
n

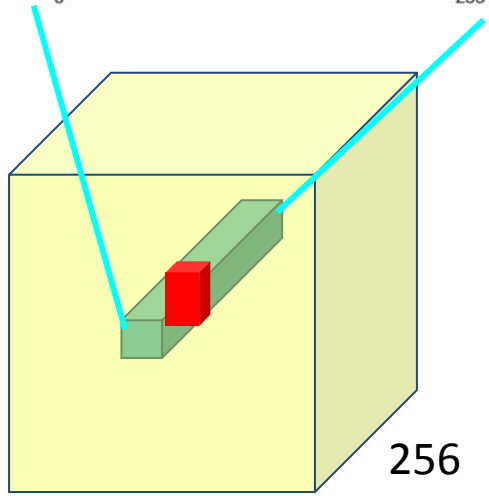
n

1

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

Model p(x) using a CNN to get probabilities for all pixels





n

n

256

Grey-scale to simplify figure (color works too)

Predict the grayscale value

Experiments - autoencoder

Minimize squared error

decoder

low-D embedding *h*

encoder

Experiments - autoencoder

Minimize squared error

decoder

low-D embedding *h*

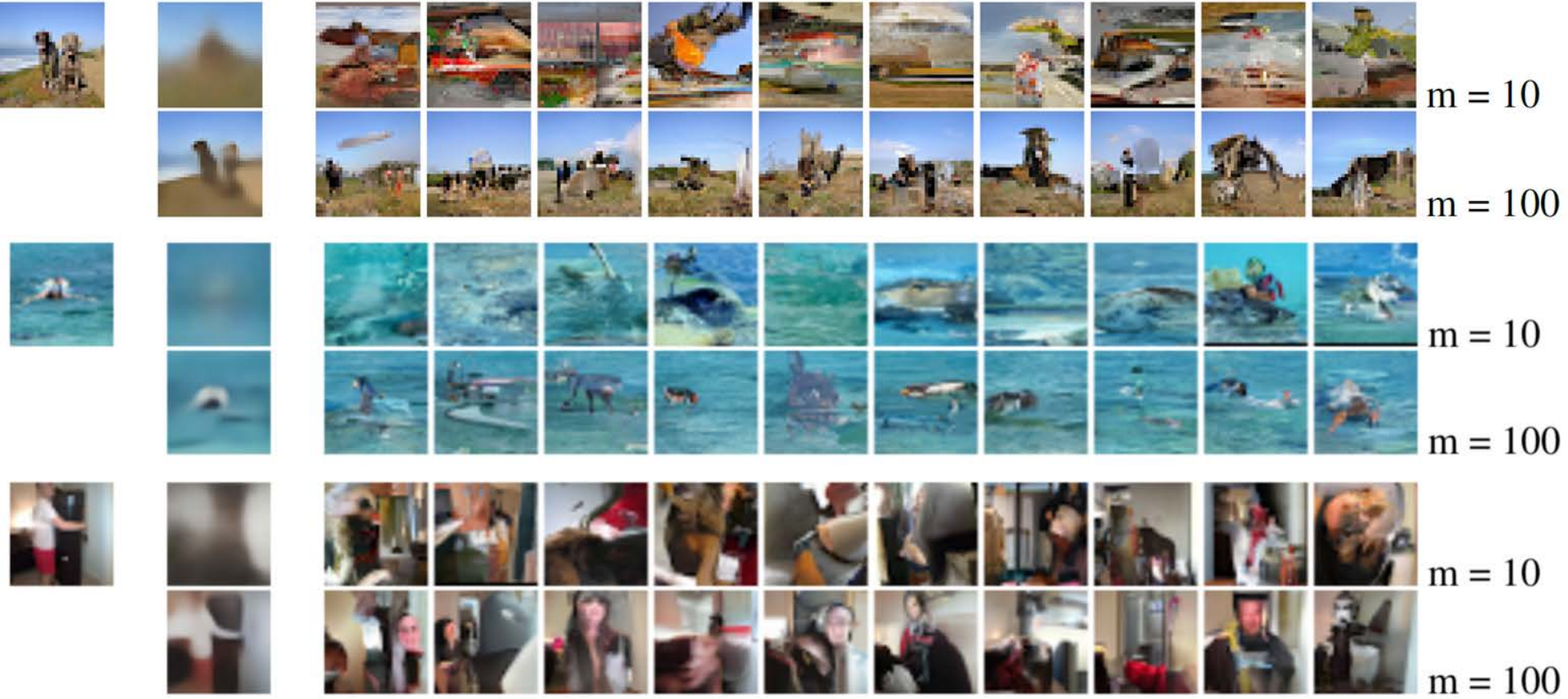
encoder

Generate based on h

PixelCNN

low-D embedding *h*

encoder



m = dimension of h embedding