Deep Generative Models III

Autoregressive Models – Part II

Machine Perception

Otmar Hilliges

7 May 2020

Advanced Interactive Technologies

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Last Week

- GANs
- Intro Deep Autoregressive Models

Unfortunately:

- Recording for 2nd half was lost

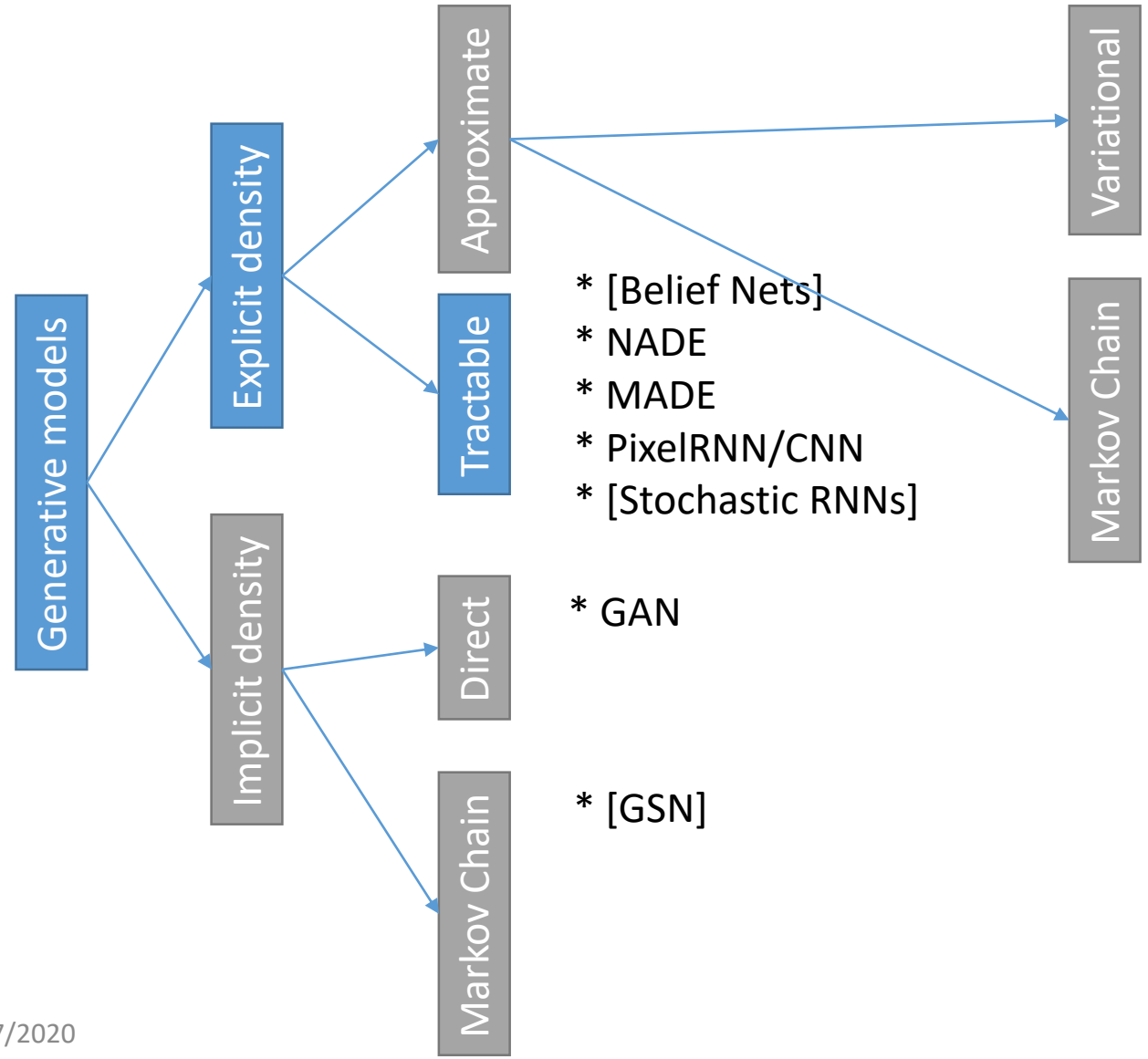
Fortunately:

https://video.ethz.ch/lectures/d-infk/2019/spring/263-3710-00L/bd785372-2216-4209-a49b-ccad25c19c01.html

user: hil-19s

pw: pTr5vM9

Taxonomy of Generative Models



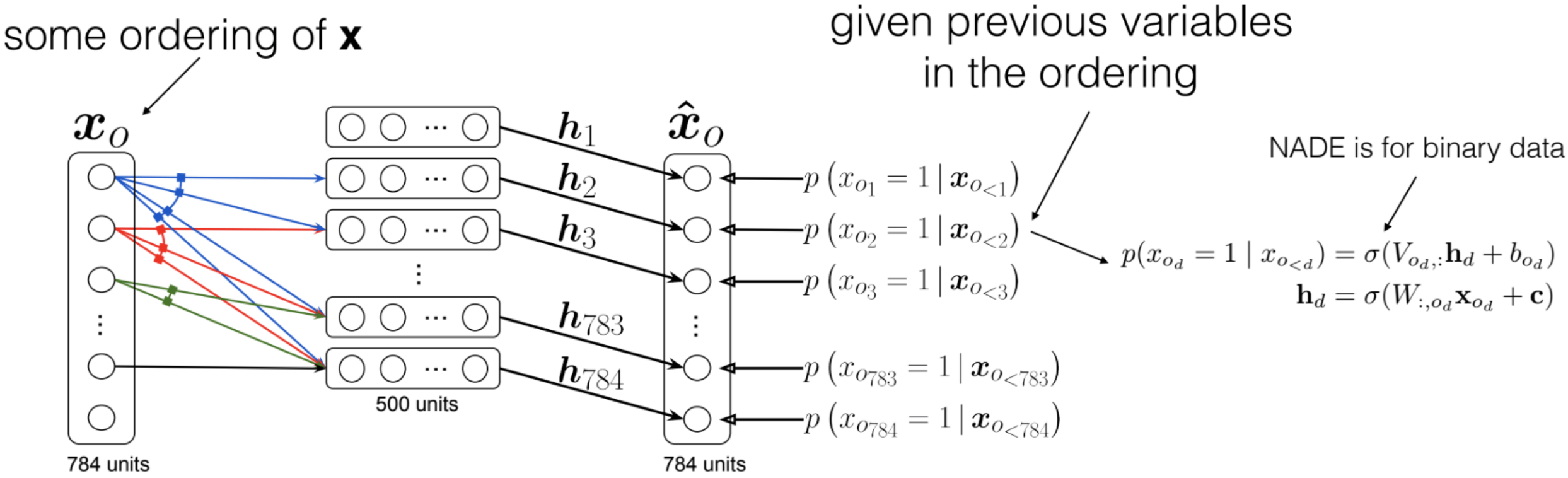
[adapted from Ian Goodfellow]

Autoregressive models

Via the chain rule of probabilities we can factorize the joint distribution over the n dimensions:

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

NADE schematically



[image source: A. Courville. UdM]

NADE – Training

training by maximizing the average log-likelihood:

$$\frac{1}{T}\sum_{t=1}^{T}\log\left(p(x^t)\right) = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{D} log\, p\left(x_i^{(t)}\middle| x_{<i}^{(t)}\right)$$

Advantages

- efficient: computations are in O(TD)

- easily extendable to other types of observations (reals, multinomials)

Paper explores different orderings of inputs

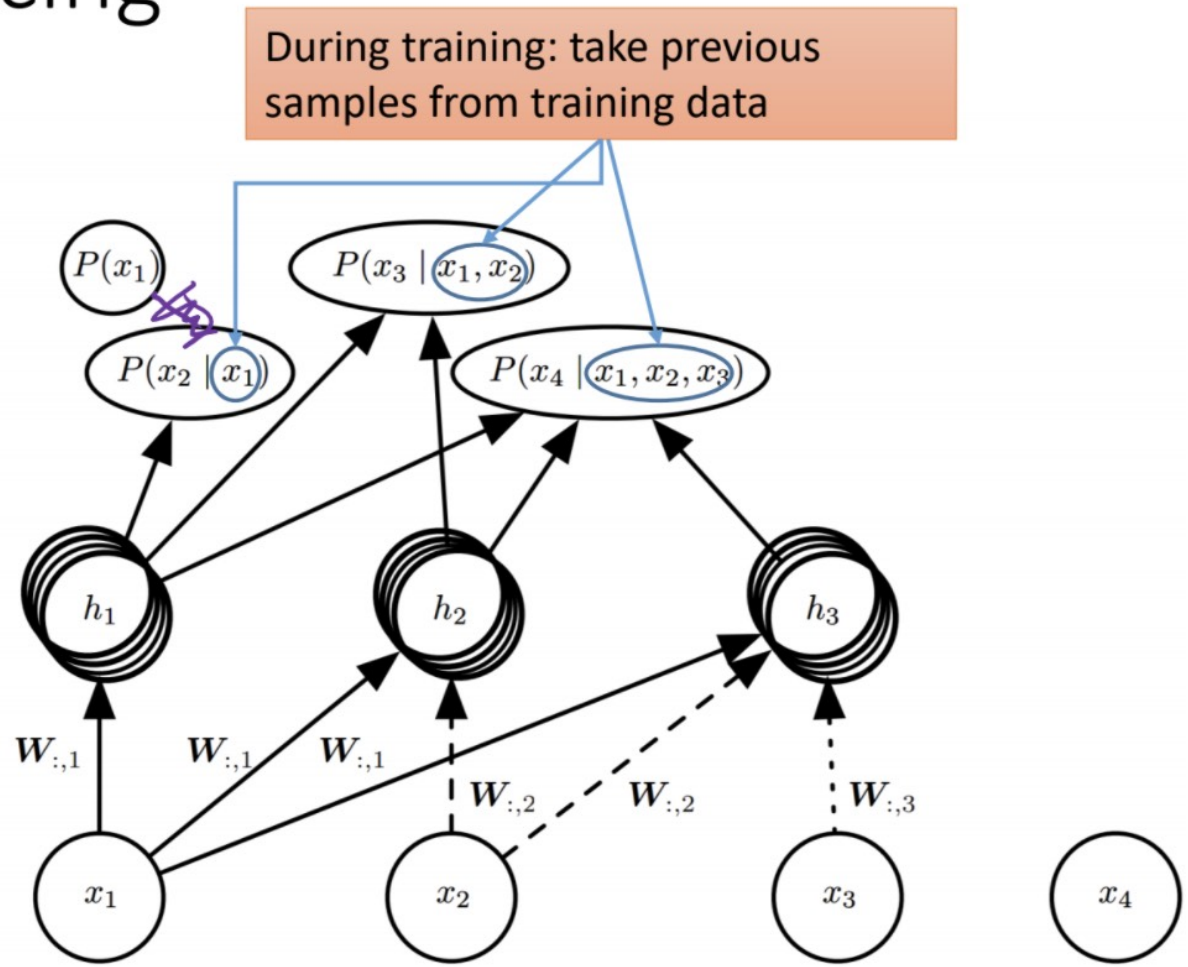
- random order works fine!

NADE – Training: Teacher forcing

Ground truth values of the pixels are used for conditioning when predicting subsequent values

It is done for stabilizing training so that the predictions do not diverge.

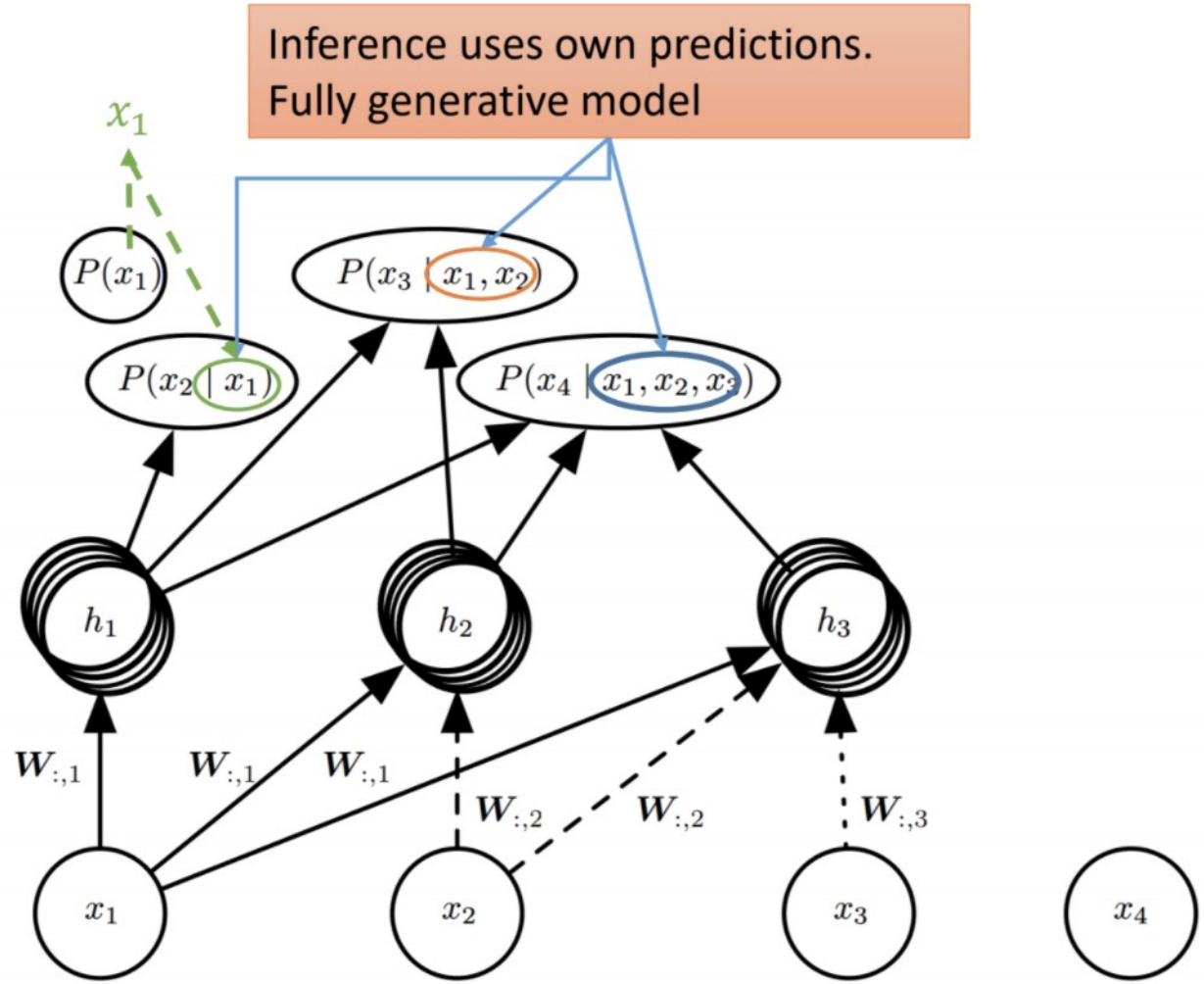
Inference always uses own predictions (fully generative model)



[image source: A. Courville. UdM]

NADE Inference

generation (at "test time") is done by conditioning on values previously sampled from the model

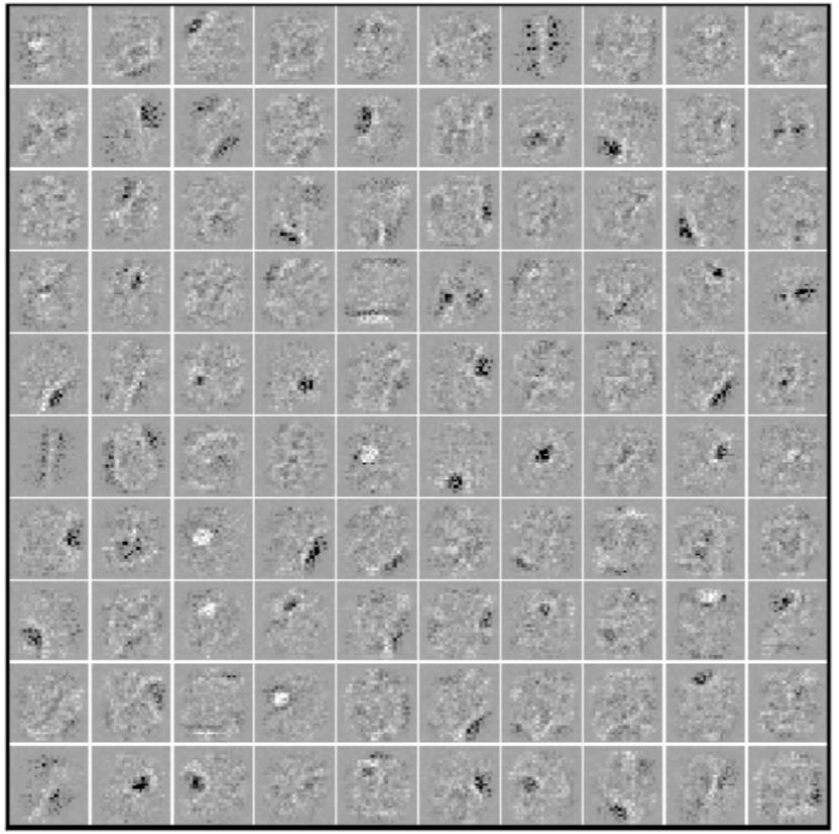


[image source: A. Courville. UdM]

NADE - Extensions

- Real-valued NADE (RNADE): conditionals are modelled by a mixture of gaussians.
- Order-less and deep NADE (DeepNADE): a single deep neural network is trained to assign a conditional distribution to any variable given any subset of the others.
- Convolutional NADE (ConvNADE)

NADE - Generation



threshold probability

Masked Autoencoder Distribution Estimator

MADE *[Germain et al. 2015]*

Constrain Autoencoder such that output can be used as conditionals $\mathrm{p}(\mathrm{x_i}|\mathrm{x_{<i}})$

To fulfill autoregressive property: $x_i \rightarrow \hat{x}_i$ is not allowed

- **No** computational path between output unit x_d and any of the input units $x_d, \ldots, x_\mathrm{D}$ **must exist** (relative to some ordering)

No connection between input and generated sample

for temperal data: $x_t \not\rightarrow x_{t+1}$

For timesteps $t = 1, \ldots, n$ sample from $(1, n-1)$ for each hidden unit. If its sampled value is smaller than the preceding one, drop (mask) the connection.

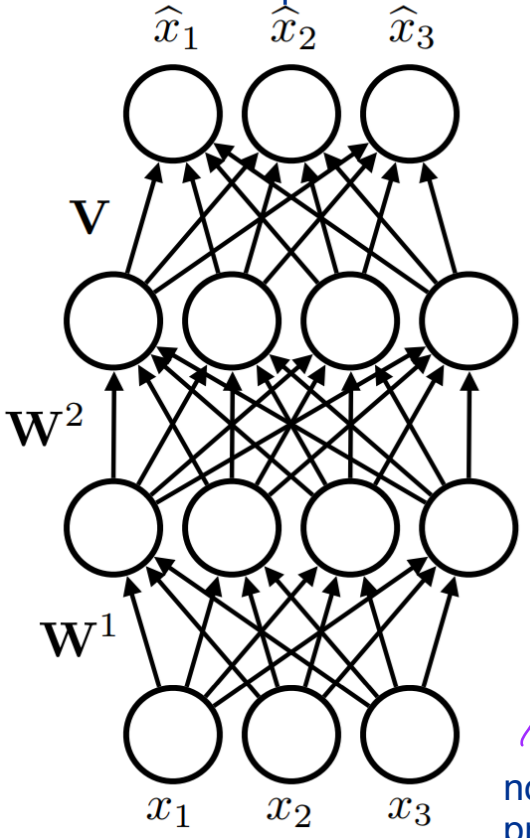
- Training has same complexity as regular autoencoders, criterion is NLL for binary x
- Computing $p(x)$ is just a matter of performing a forward pass, but sampling requires D forward passes
- In practice, very large hidden layers necessary as not all hidden units can contribute to each conditional

Masked Autoencoder Distribution Estimator

MADE *[Germain et al. 2015]*

Constrain Autoencoder such that output can be used as conditionals $p(x_i|x_{<i})$

not generated, cannot use for conditional prob.

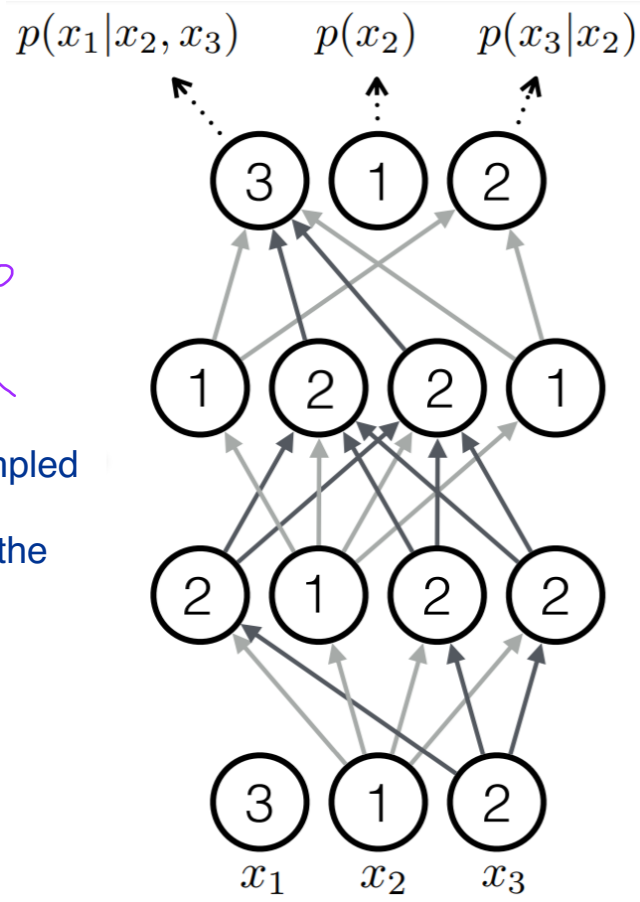


add dropout to make it valid

in hidden layer we sampled large than 0 less than highest integeter than the lower one

$x_i \rightarrow \hat{x}_i$

not valid for autoregressive property



output layer 之间连接

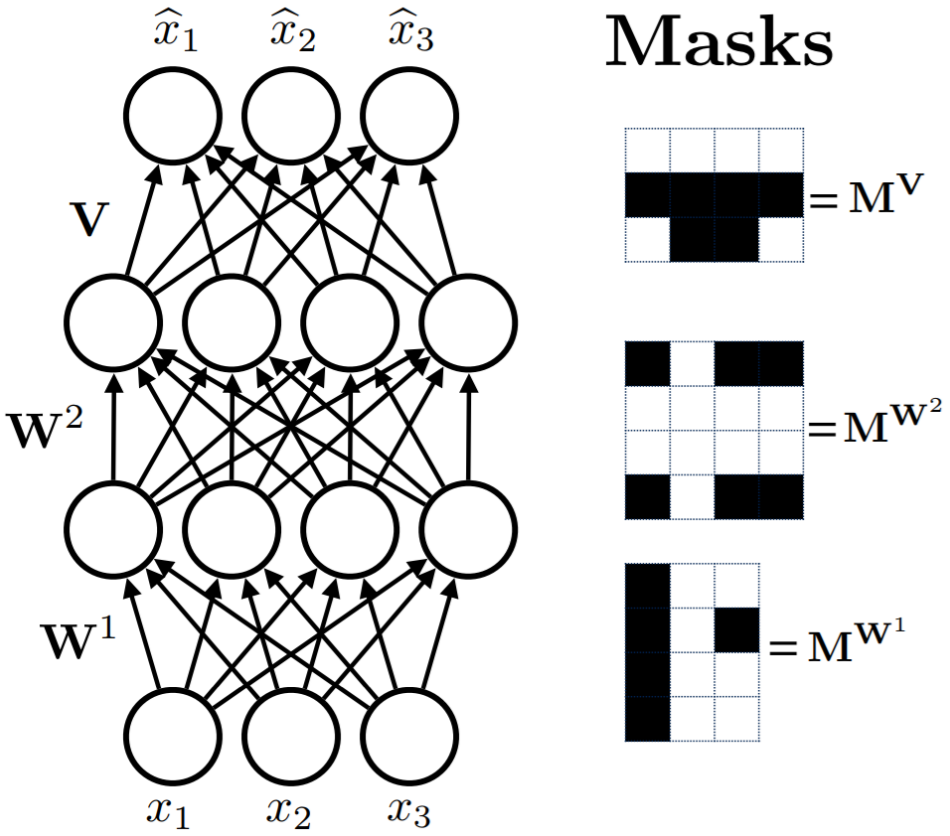
hidden layer 之间连接

2<3 所以之间没有 connection

Masked Autoencoder Distribution Estimator

MADE *[Germain et al. 2015]*

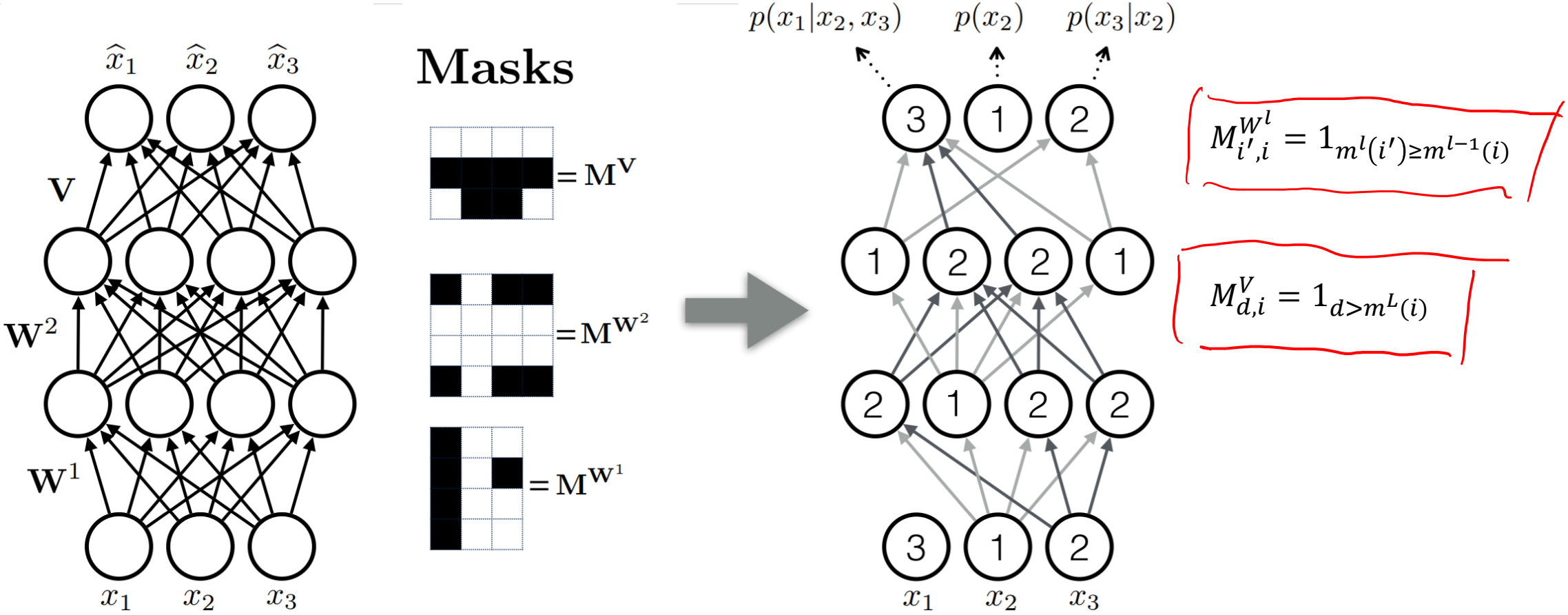
Constrain Autoencoder such that output can be used as conditionals $p(x_i|x_{<i})$



Masked Autoencoder Distribution Estimator

MADE *[Germain et al. 2015]*

Constrain Autoencoder such that output can be used as conditionals $p(x_i|x_{<i})$



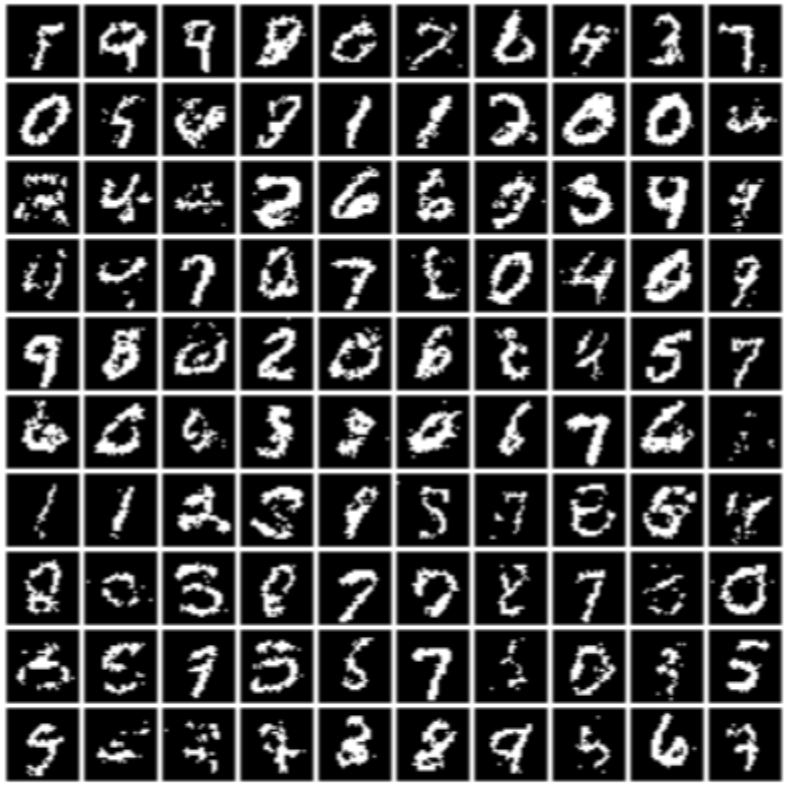
Masked Autoencoder Distribution Estimator

MADE *[Germain et al. 2015]*

Constrain Autoencoder such that output can be used as conditionals $\mathrm{p}(\mathrm{x_i}|\mathrm{x_{<i}})$

- Training has the same complexity as regular autoencoders
- Criterion is NLL for binary x
- Computing $p(x)$ is just a matter of performing a forward pass
- Sampling however requires D forward passes
- In practice, very large hidden layers necessary (not all hidden units can contribute to each conditional)

Masked Autoencoder Distribution Estimator



Binarized MNIST samples

MADE *[Germain et al. "MADE: Masked Autoencoder for Distribution Estimation." ICML. 2015]*

Generative Models of Natural Images

Explicit density model

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

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

Likelihood of image x

Likelihood of i-th pixel, given all the previous pixels

Issue 1: will need to define ordering of pixels

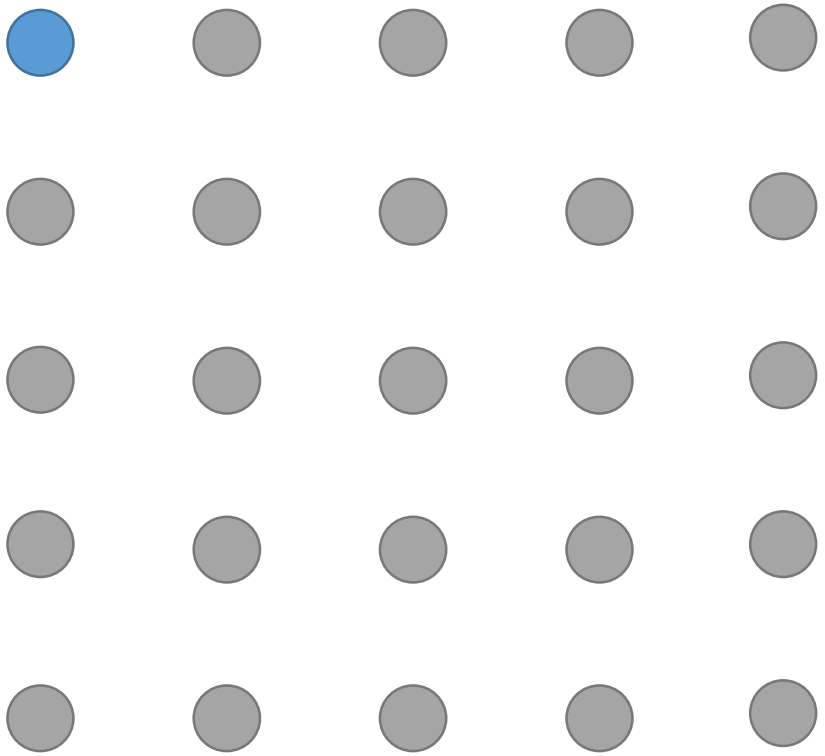
Issue 2: complex distribution => parametrize via NN

Then maximize likelihood of training data

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

Generate image pixels starting from corner

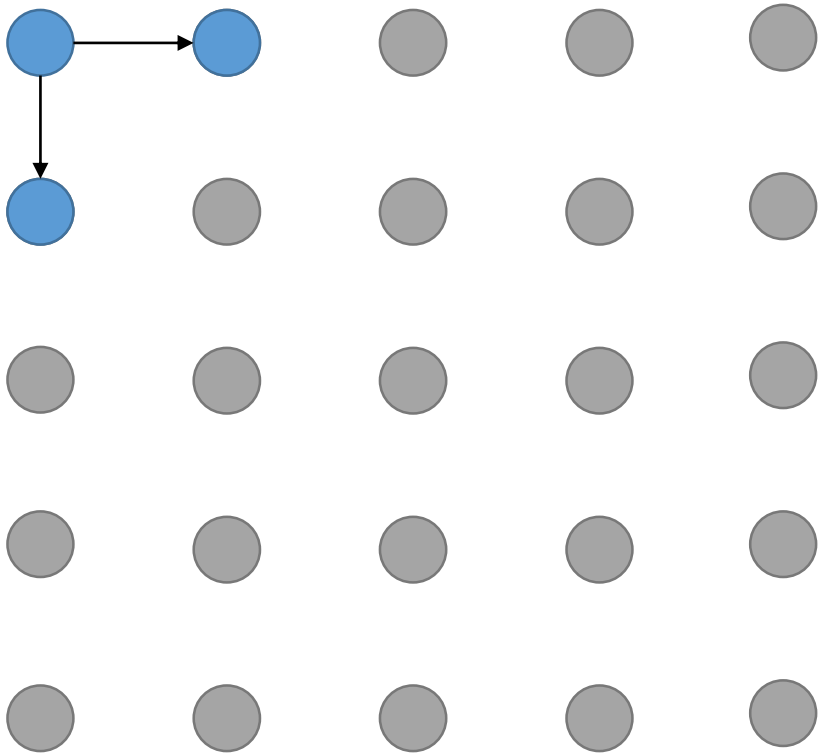
Dependency on previous pixels modeled using an RNN (LSTM)



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

Generate image pixels starting from corner

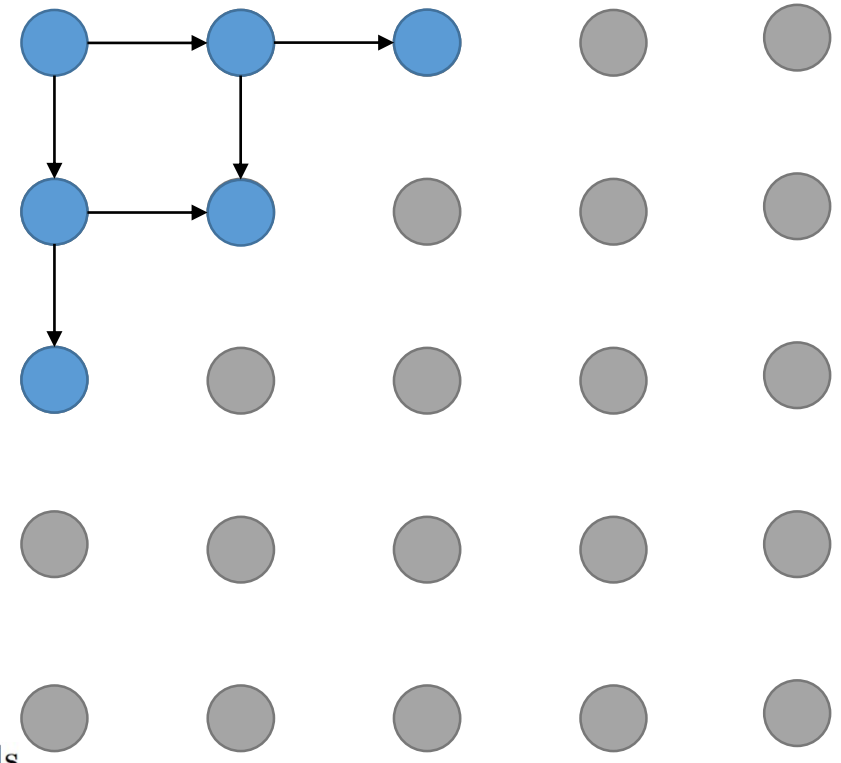
Dependency on previous pixels modeled using an RNN (LSTM)



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

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

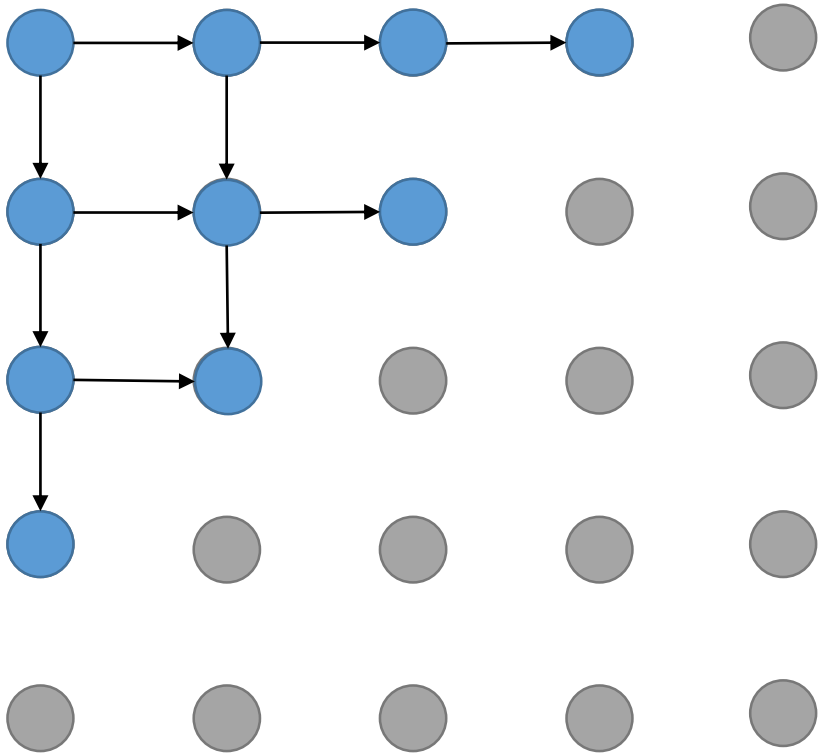


PixelRNN generates image pixels starting from corner, dependency on previous pixels modeled using an RNN (LSTM). Training maximizes likelihood of training images. Here, sequential generation is slow due to explicit pixel dependencies. PixelRNN uses masked convolutions as a convenient way of enforcing dependencies between pixels. Higher layers are fully connected. It is also autoregressive over color channels; authors actually allow information from R (red) channel go into G (green), and from R and G go into B. So we have ordering not only within spatial dimensions but also within source channels (think colors) too. Different architectures could be used: Row LSTM, Diagonal LSTM, Diagonal BiLSTM ...

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

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

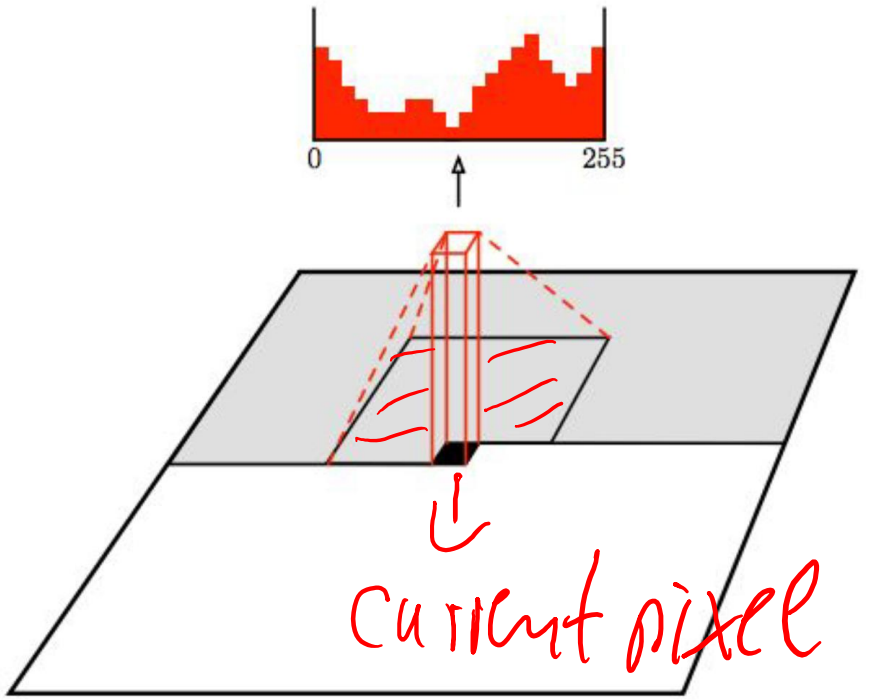


Issue: Sequential generation is slow – due to explicit pixel dependencies

PixelCNN *[van der Oord et al. 2016b]*

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region



[image source: van der Oord et al.]

PixelCNN *[van der Oord et al. 2016b]*

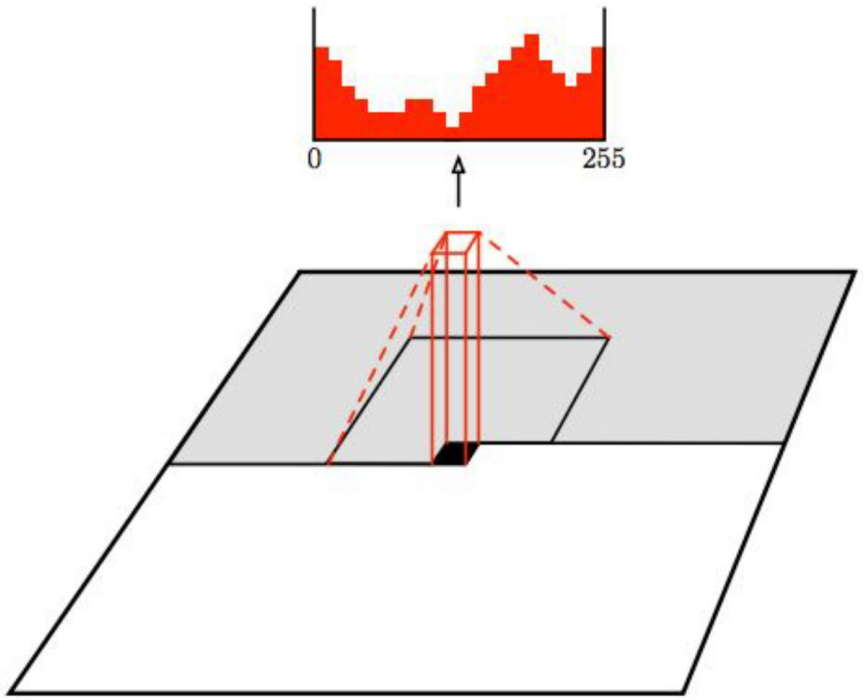
Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training maximizes likelihood of training images

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

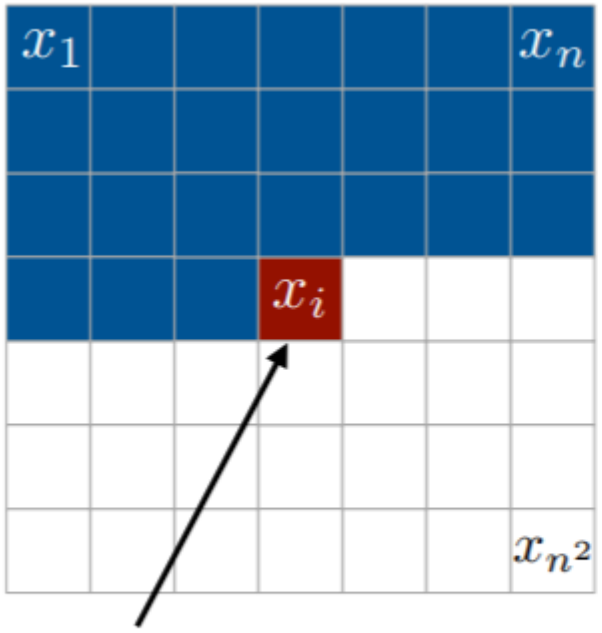
Softmax loss over pixel values



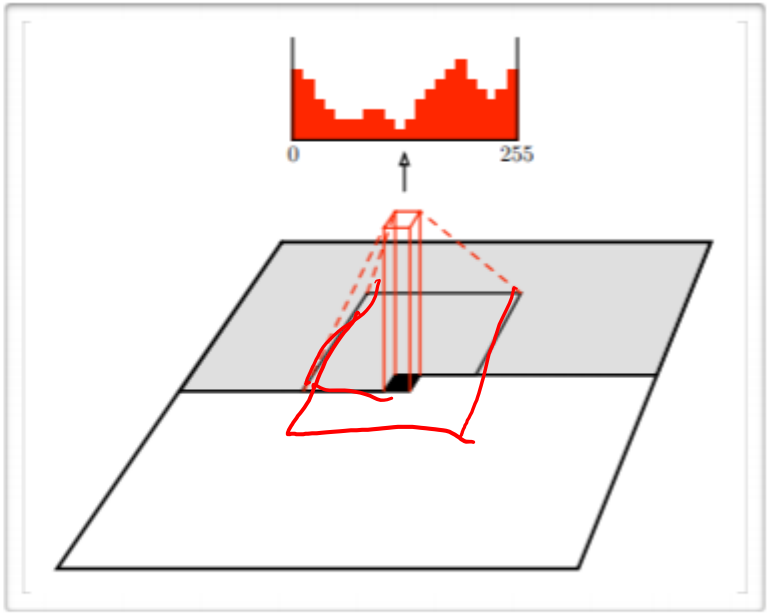
[image source: van der Oord et al.]

PixelCNN

Use masked convolutions to model conditionals



$$p(x_i \mid \mathbf{x}_{<i})$$



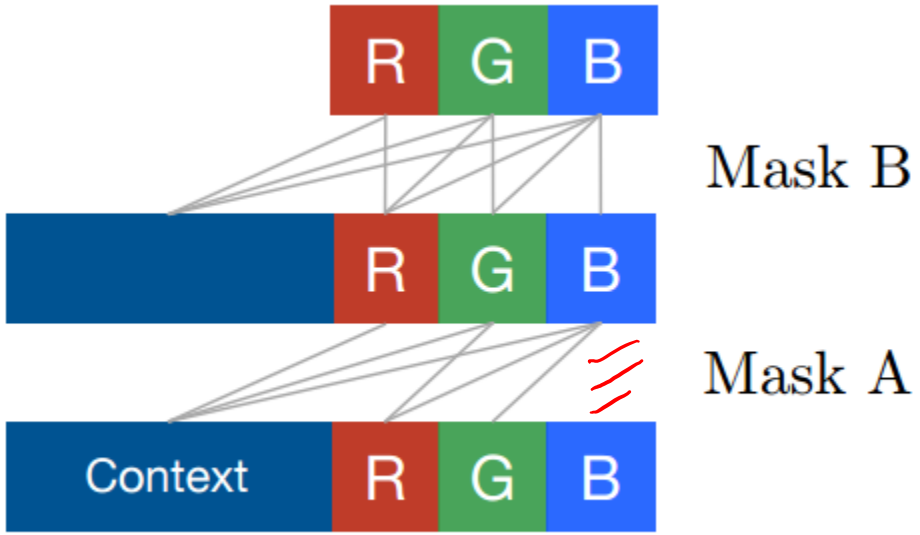
PixelCNN – Masking *[van der Oord et al. 2016b]*

To correctly model the conditional probability one needs to prevent the current pixel from contributing to the prediction

Use masked convolutions

No “self” connections in the bottom layer

Higher layers are fully connected

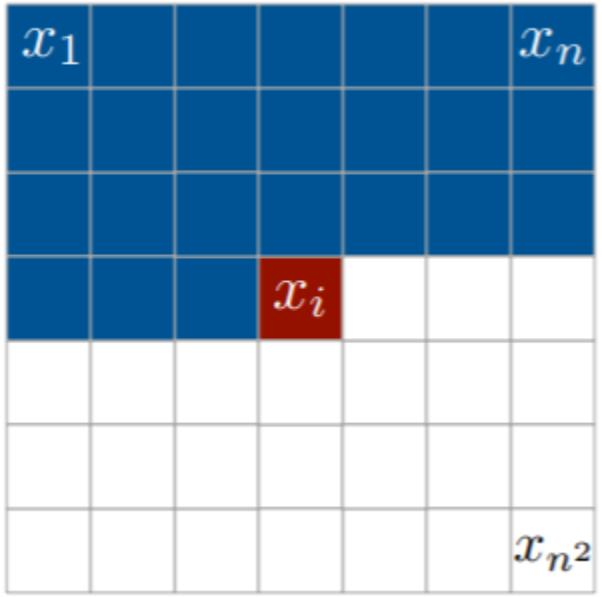


Autoregressive over color-channels:

$$p(x_i|x_{<i}) = p(x_{i,R}|x_{<i})p(x_{i,G}|x_{i,R}, x_{<i})p(x_{i,B}|x_{i,R}, x_{i,G}, x_{<i})$$

PixelCNN – Masking *[van der Oord et al. 2016b]*

Speed-up raster scan?



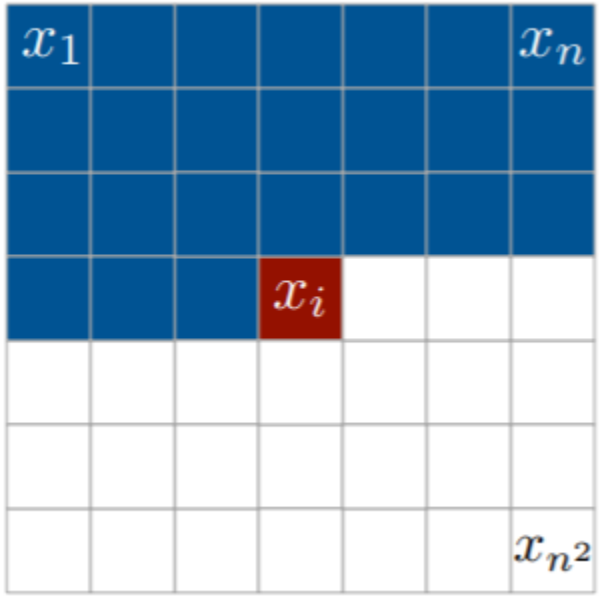
Use a stack of masked Convs

1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Training can be parallelized, though generation is still a sequential operation over pixels

PixelCNN – Masking *[van der Oord et al. 2016b]*

Speed-up raster scan?

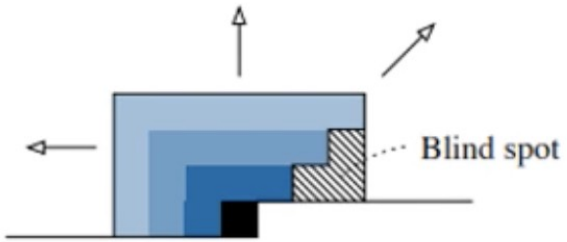


Use a stack of masked Convs

1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Training can be parallelized, though generation is still a sequential operation over pixels

Blind spots of pixel CNN

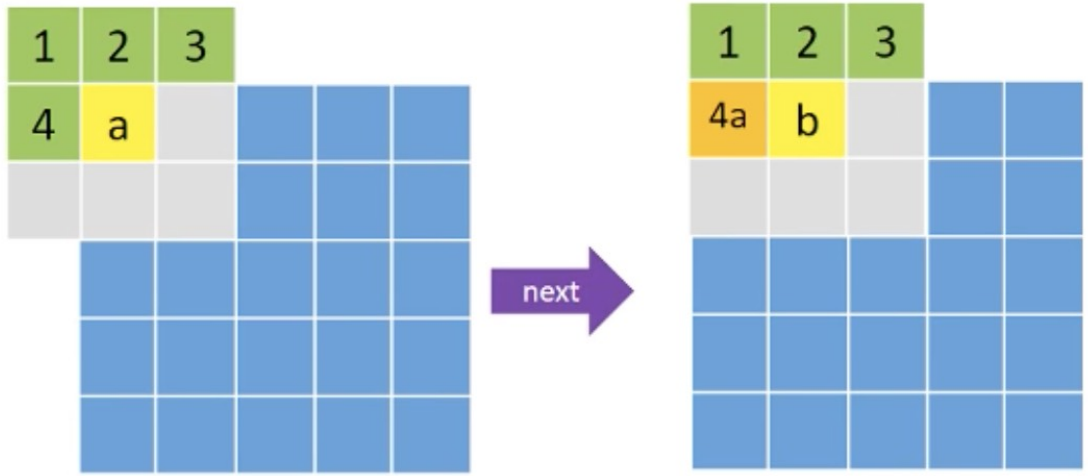


The blind spot means: the generated point is generated without considering the spots.

Feature map

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

When running with the masked kernels



If the feature map is handled with zero padding, then a would be only influenced with bias terms of NN. (or maybe we can consider the non-zero padding)

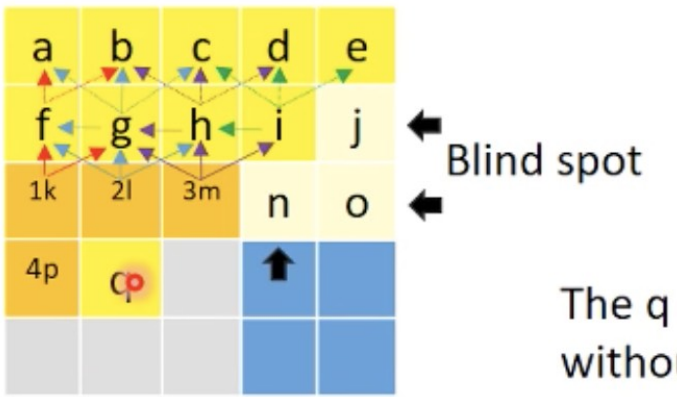
kernel

1	2	3
4	5	6
7	8	9

mask

Masked kernel

1	2	3
4	5	6
7	8	9



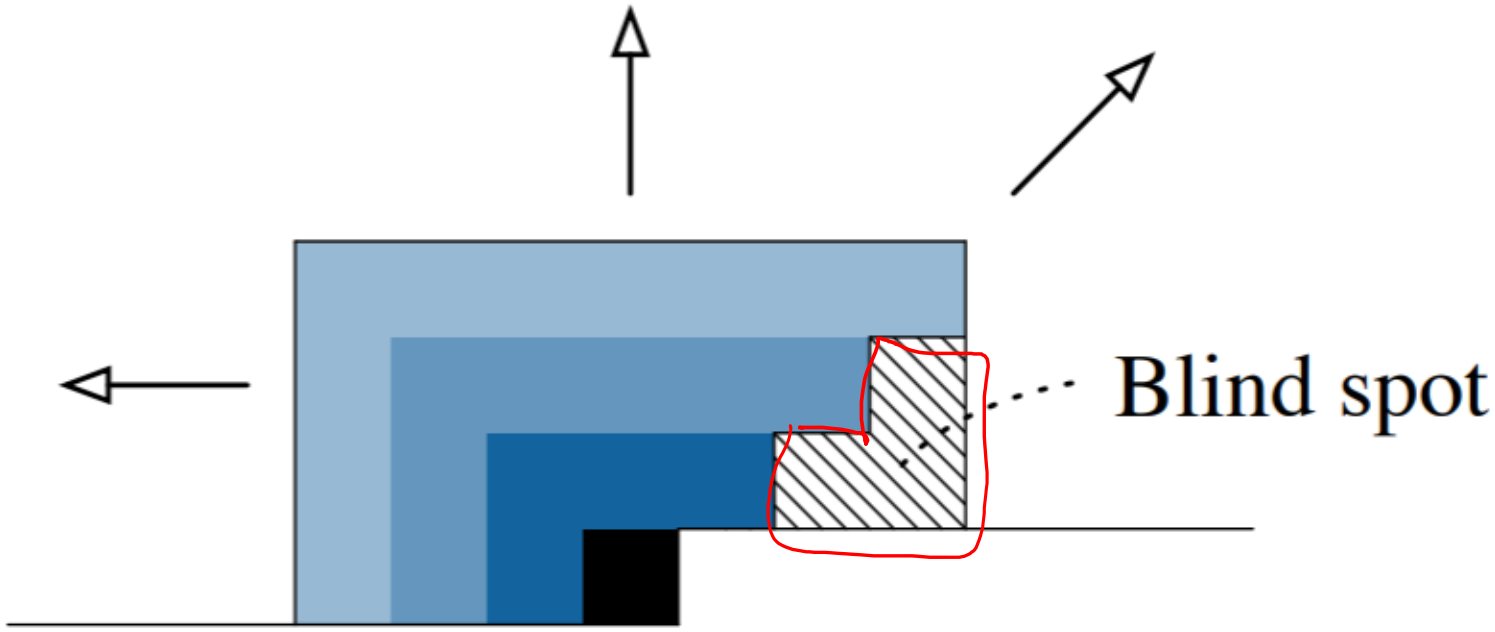
The q is generated without considering j, n, o.

- https://towardsdatascience.com/blind-spot-problem-in-pixelcnn-8c71592a14a

Improving PixelCNN

There is a problem with this form of masked convolution.

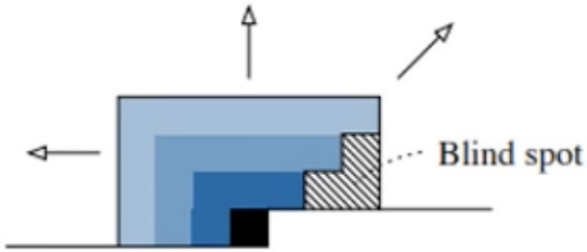
1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0



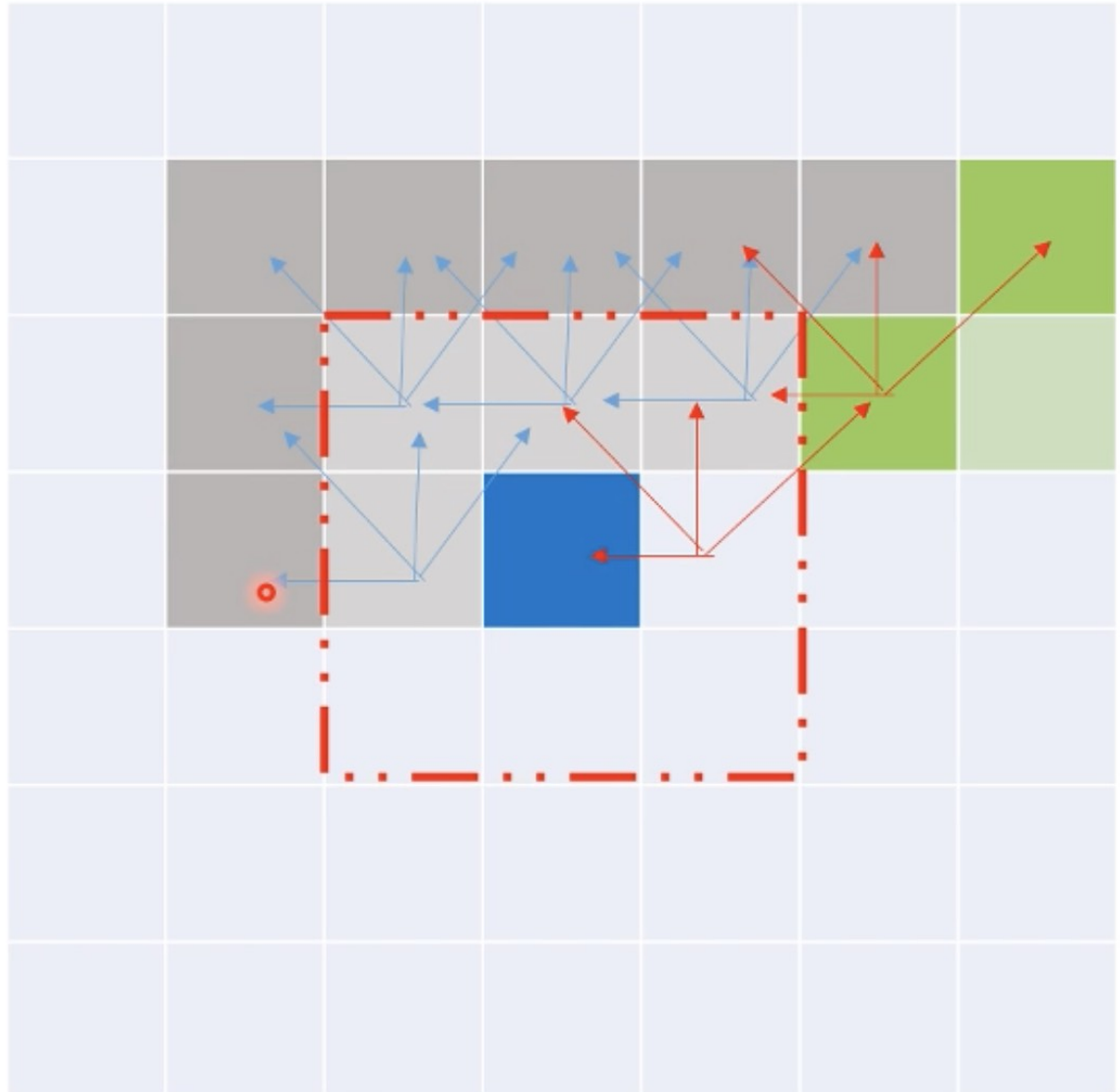
Stacking layers of masked convolution creates a blindspot

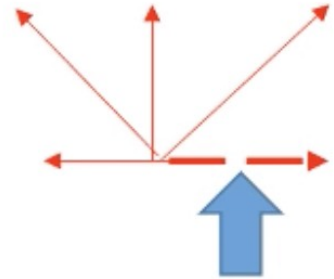
[adapted from van den Oord et al. via A. Courville]

The blind spots explanation



Supposing:
The 3*3 kernel and postulating 2 or 3 layers





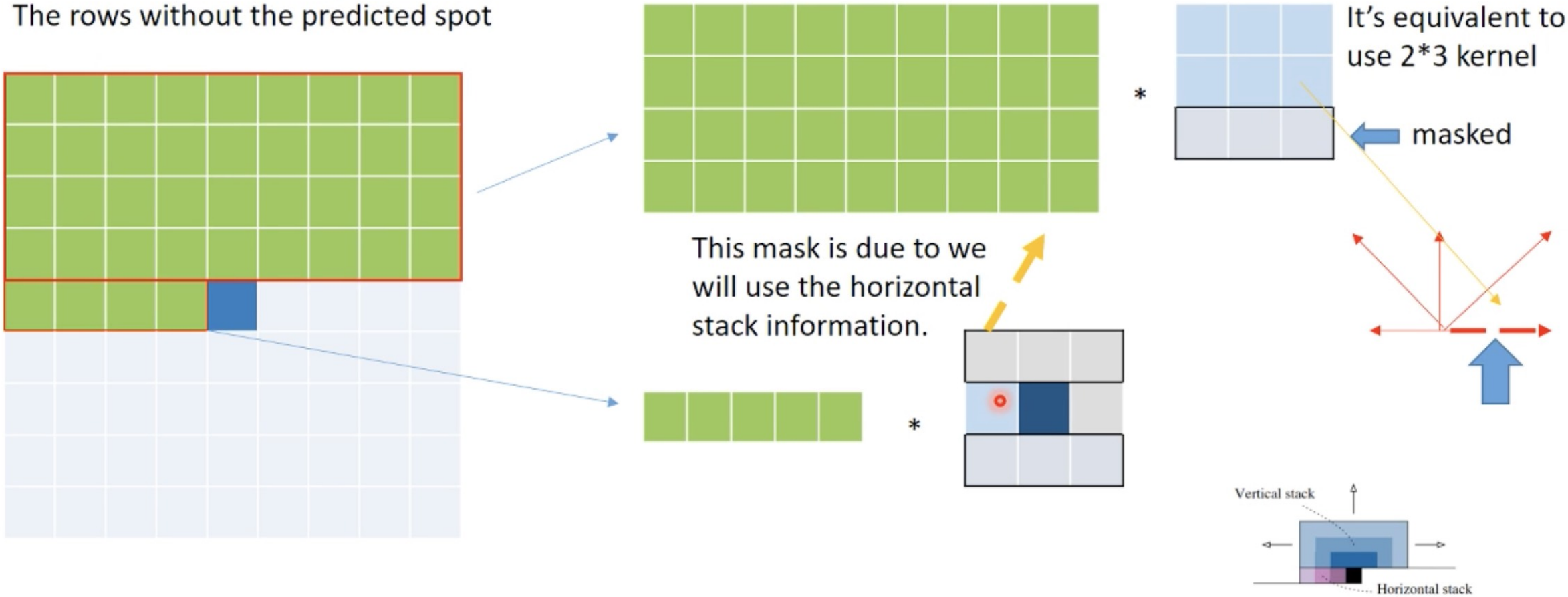
Absenting this direction makes the blind spots. These spot number will increase with the layer number.

How about make a new filters which still have this direction?

Horizontal and vertical stack

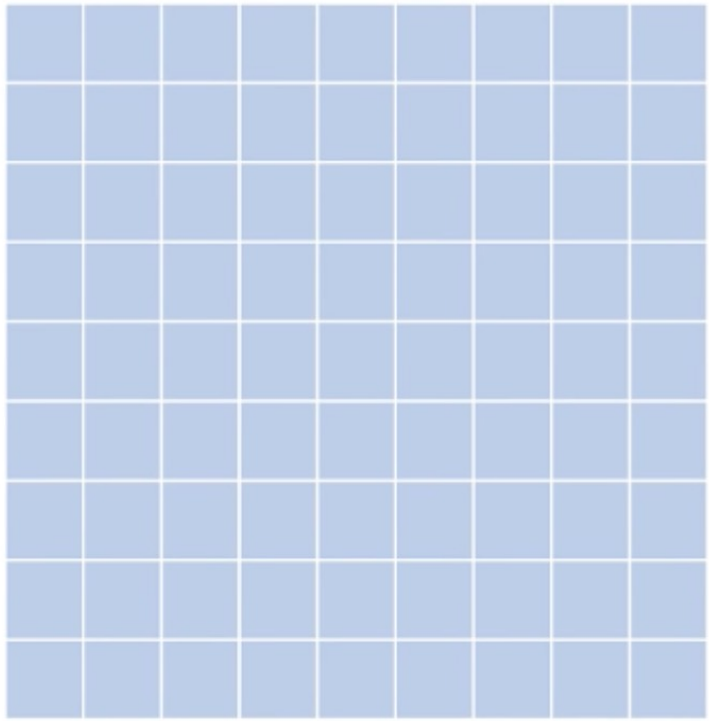
The feature map can be separated into 2 parts according to the status of the rows:

1. The rows containing the predicted spot
2. The rows without the predicted spot



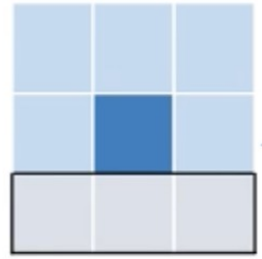
https://www.slideshare.net/suga93/conditional-image-generation-with-pixelcnn-decoders

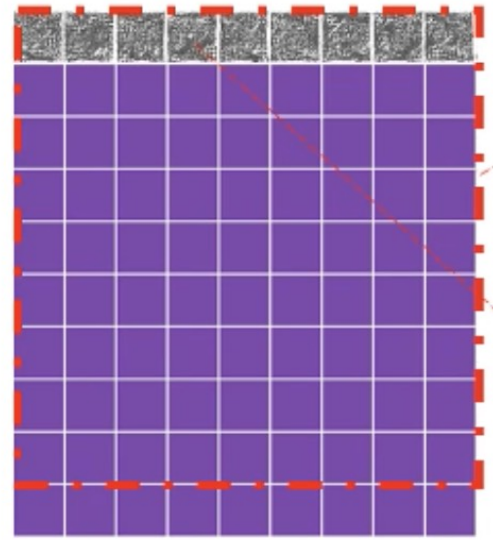
Construct the horizontal and vertical at the same time



Vertical stack

*



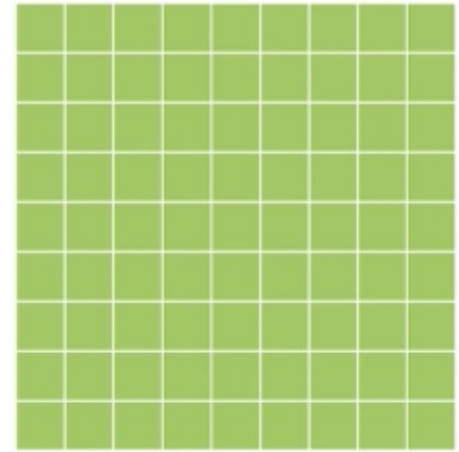


crop

This feature map will see the “future spot” without padding and crop

Padding after convolution

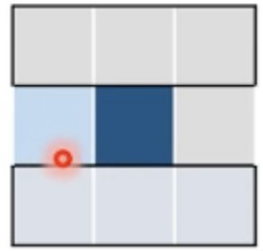
+



New feature map

Horizontal stack

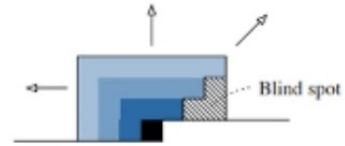
*

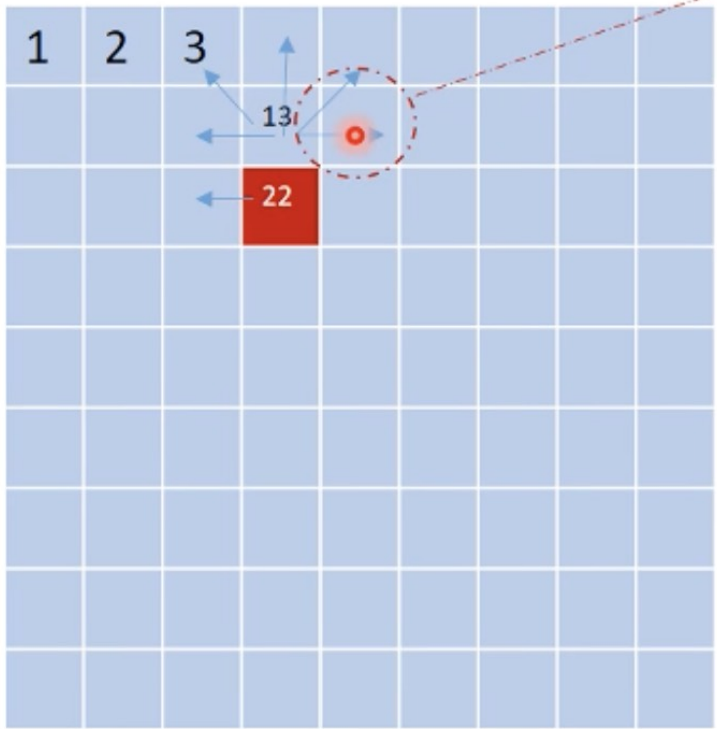


Horizontal stack provides the final predicted output

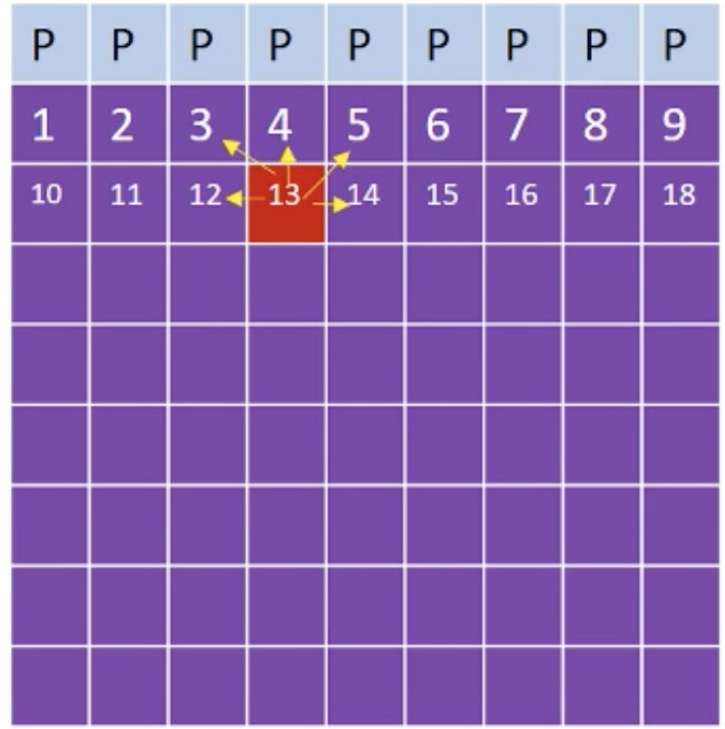
What happen when vertical and horizontal stacks merging

Blind spot disappear
(but we still use the same kernel shape)

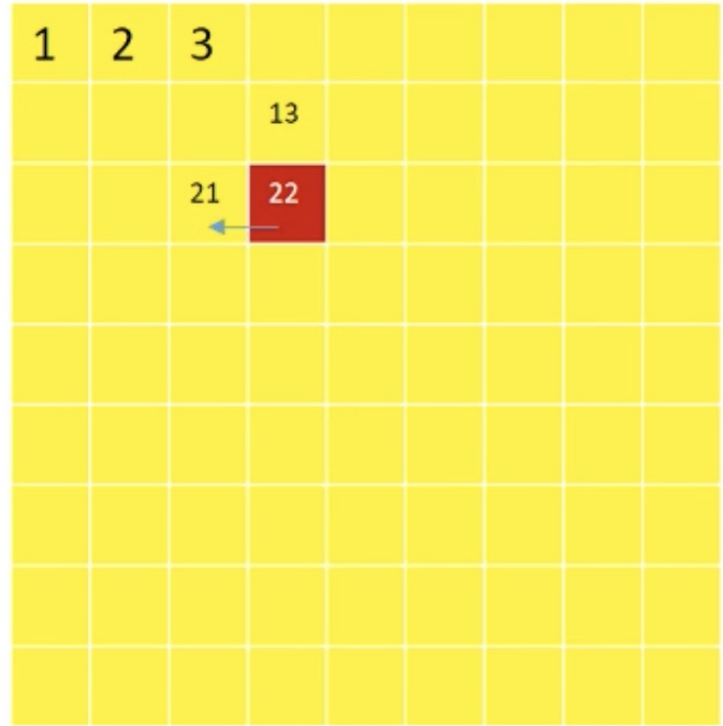




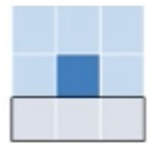
=



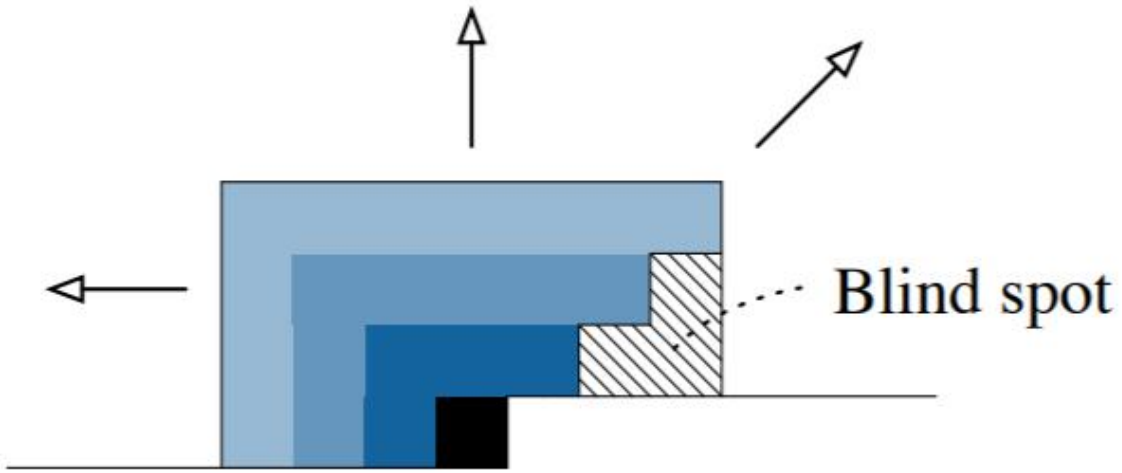
+



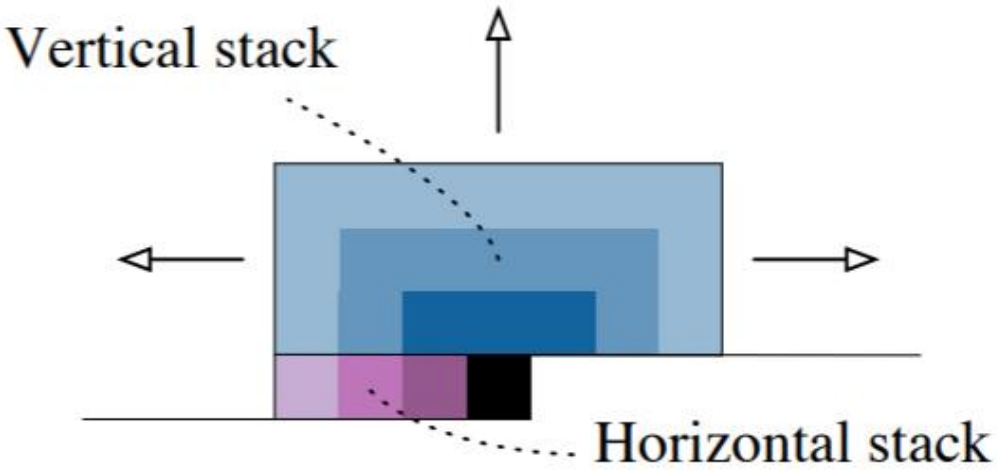
If we want to take position 22 …



Improving PixelCNN



Stacking layers of masked convolution creates a blindspot



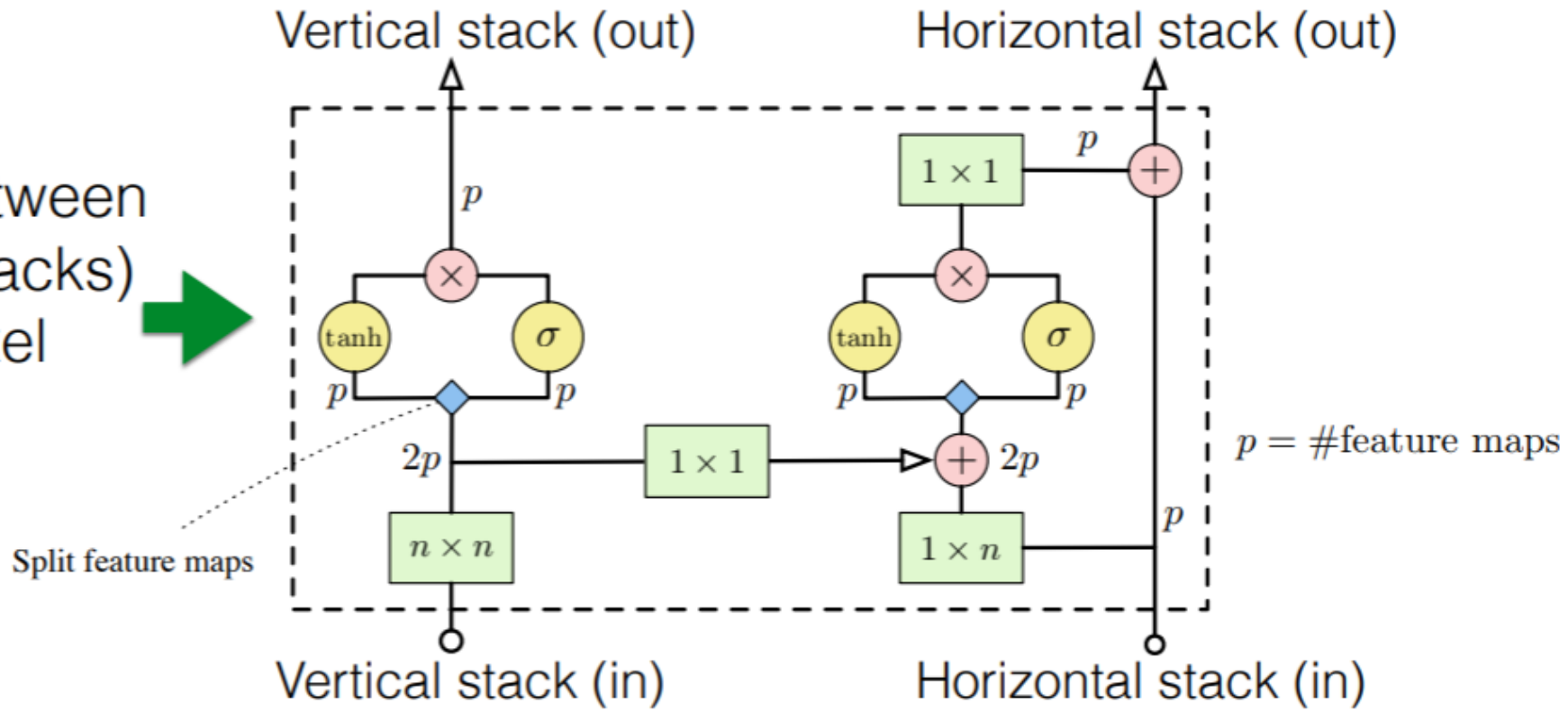
Solution: use two stacks of convolution, a vertical stack and a horizontal stack

[adapted from van den Oord et al. via A. Courville]

Improving PixelCNN

Use more expressive nonlinearity: $\mathbf{h}_{k+1} = \tanh(W_{k,f} * \mathbf{h}_k) \odot \sigma(W_{k,g} * \mathbf{h}_k)$

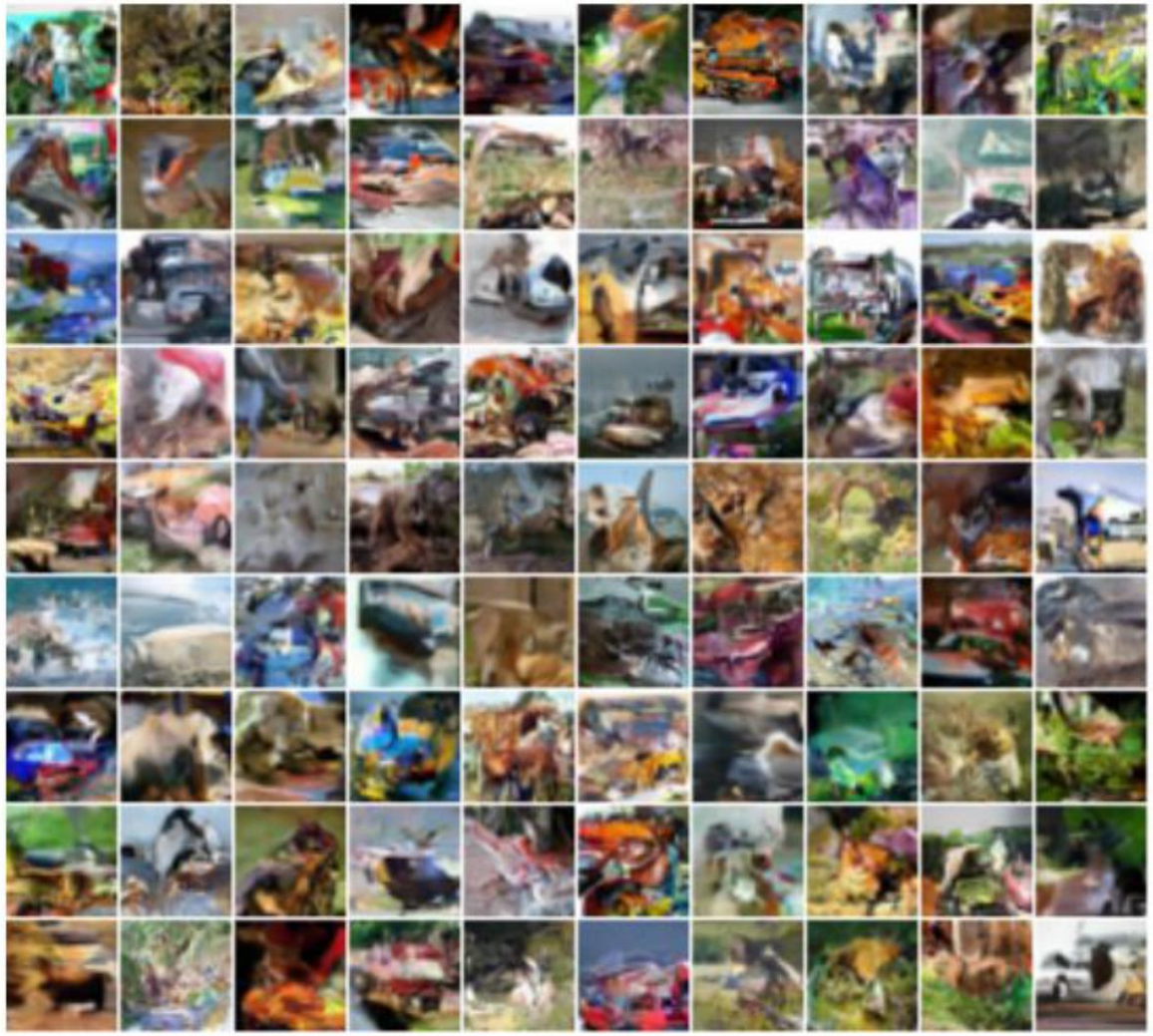
This information flow (between vertical and horizontal stacks) preserves the correct pixel dependencies



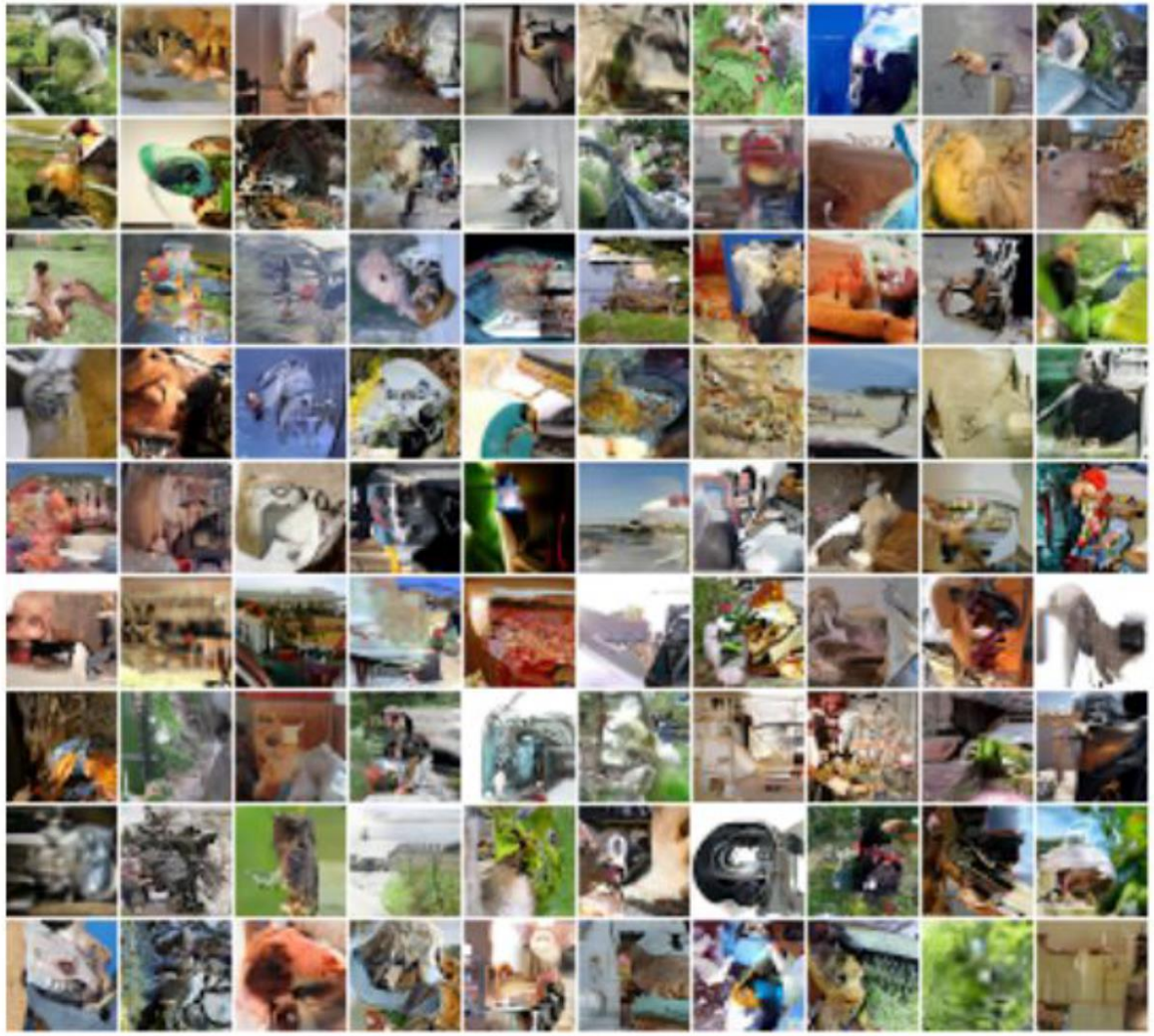
conditional pixelCNN
https://www.youtube.com/watch?v=1BURwCCYNEI

[adapted from van den Oord et al. via A. Courville]

Results



32x32 CIFAR-10



32x32 ImageNet

Results

Nice property of autoregressive models:

Because we train via NLL, we also have a direct metric to compare performance:

Model	NLL Test (Train)
Uniform Distribution: [30]	8.00
Multivariate Gaussian: [30]	4.70
NICE: [4]	4.48
Deep Diffusion: [24]	4.20
DRAW: [9]	4.13
Deep GMMs: [31, 29]	4.00
Conv DRAW: [8]	3.58 (3.57)
RIDE: [26, 30]	3.47
PixelCNN: [30]	3.14 (3.08)
PixelRNN: [30]	3.00 (2.93)
Gated PixelCNN:	3.03 (2.90)

PixelCNN *[van der Oord et al. 2016b]*

In Vanilla PixelCNN stacked masked convolutions creates blind spots, since convolutional networks capture a bounded receptive field. **Gated PixelCNN** uses a more expressive nonlinearity and two layers of stacks to match the performance of PixelRNN. Information flow between horizontal and vertical stacks preserves the correct pixel dependencies.

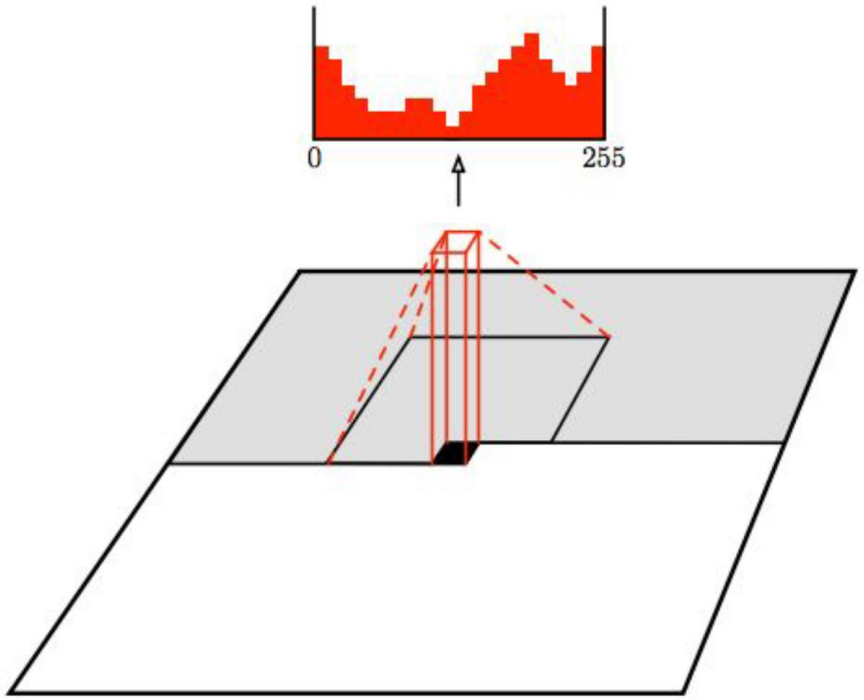
- Horizontal Stack: It conditions on the current row and takes as input the output of previous layer as well as the of the vertical stack.
- Vertical Stack: It conditions on all the rows above the current pixel. It doesn't have any masking. It's output is fed into the horizontal stack and the receptive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

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

Generation must still proceed sequentially
=> still slow



[image source: van der Oord et al.]

Summary

Pros:

- explicit likelihood $p(x)$
- likelihood of training data gives good evaluation metric
- Good samples

Con:

- Sequential generation => slow

Lot's of tricks to improve PixelCNN:

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc…

See

- Van der Oord et al. NeurIPS 2016
- Salimans et al. 2017 (PixelCNN++)

TCNs - WaveNet

[van der Oord et al. 2016c]

Temporal Causal Networks (TCN):

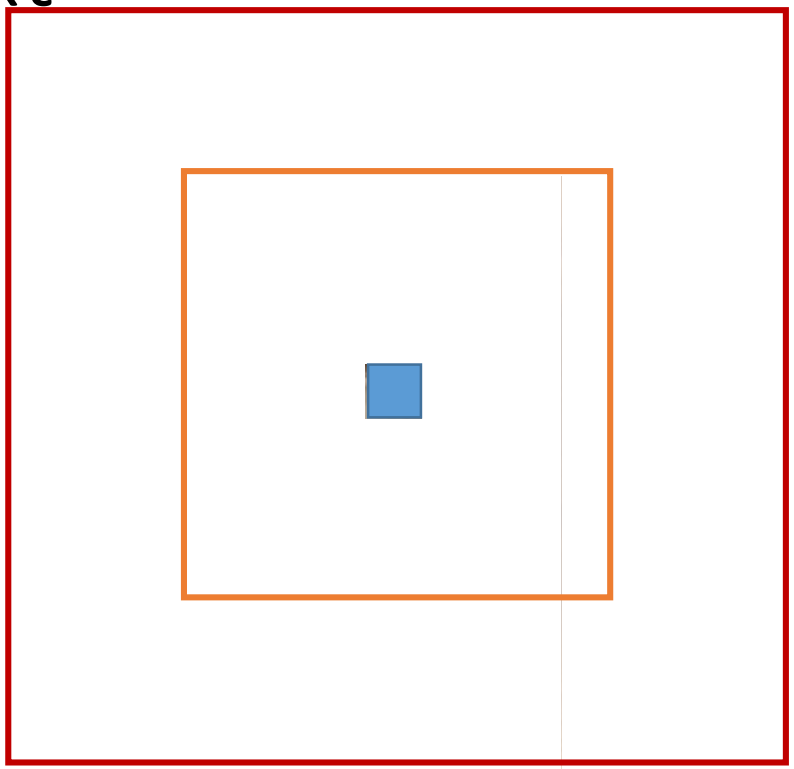
- Idea is to adapt PixelCNN to work with audio data.
- Problem: much higher dimensionality than images
- Long term temporal dependencies
- Uses dilated convolutions to reach larger receptive fields
- Inference speed
- Note: strided convolutions cannot be used due to the need to preserve resolution

Idea: Adapt PixelCNN to work with Audio data

Problem: much larger dimensionality than images (at least 16,000 samples per second)

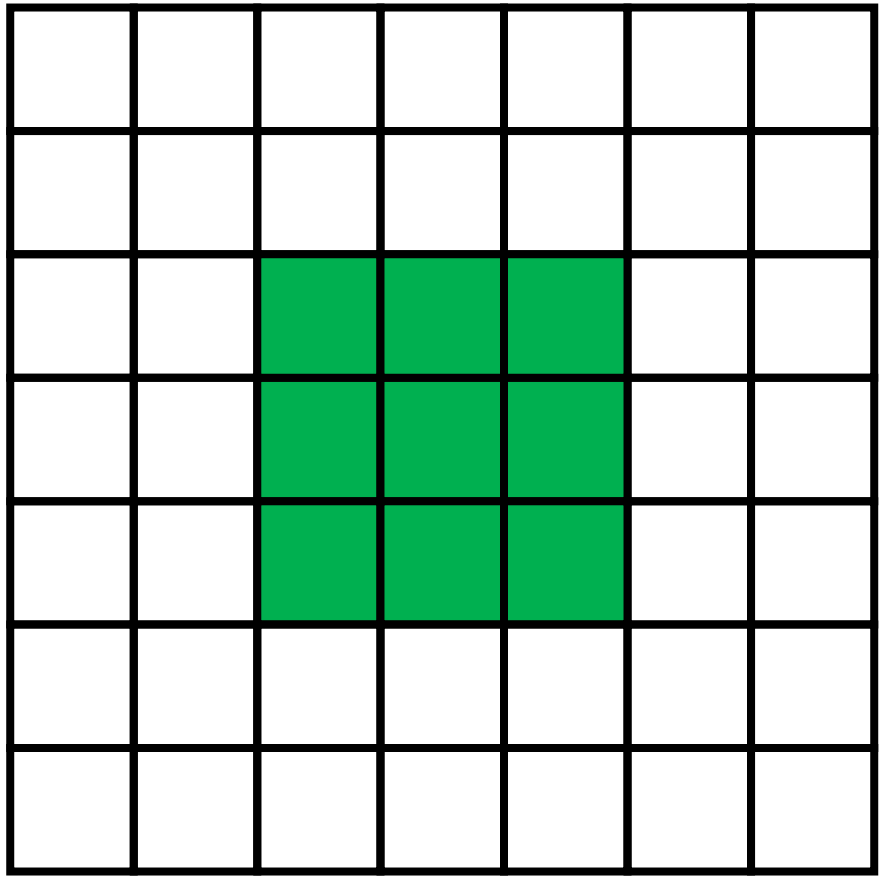
Leverage temporal convolutions to model large-scale dependencies: temporal causal networks (TCNs)

Detour: Importance of context



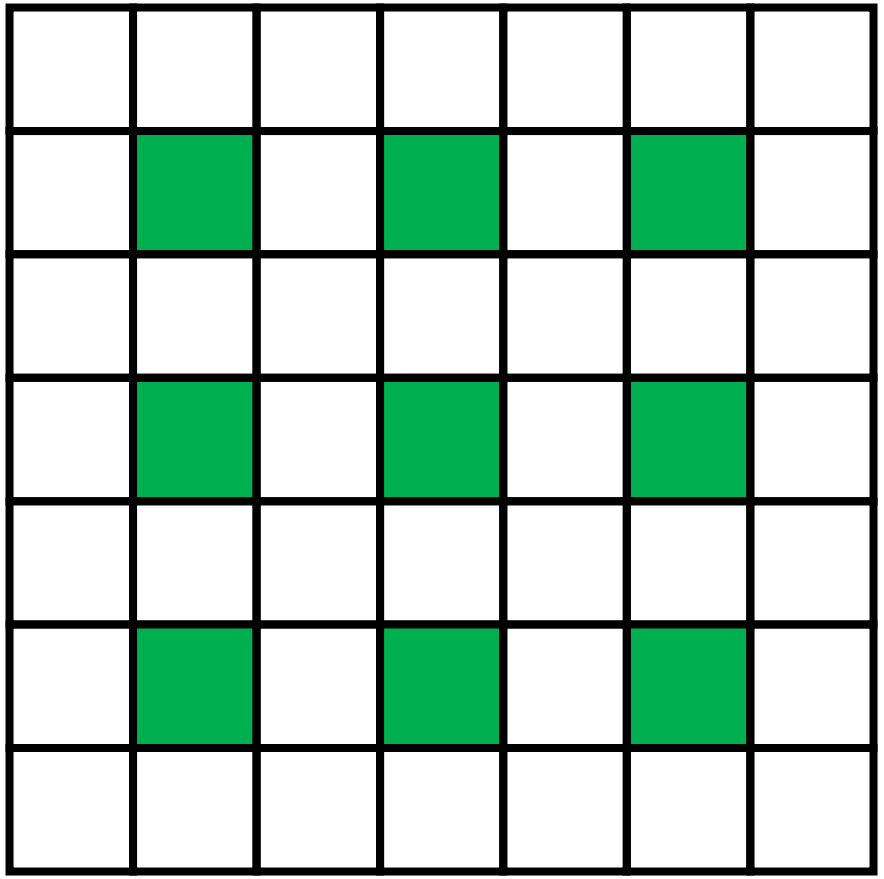
Detour: Dilated Convolutions

Convolution



Dilation 1

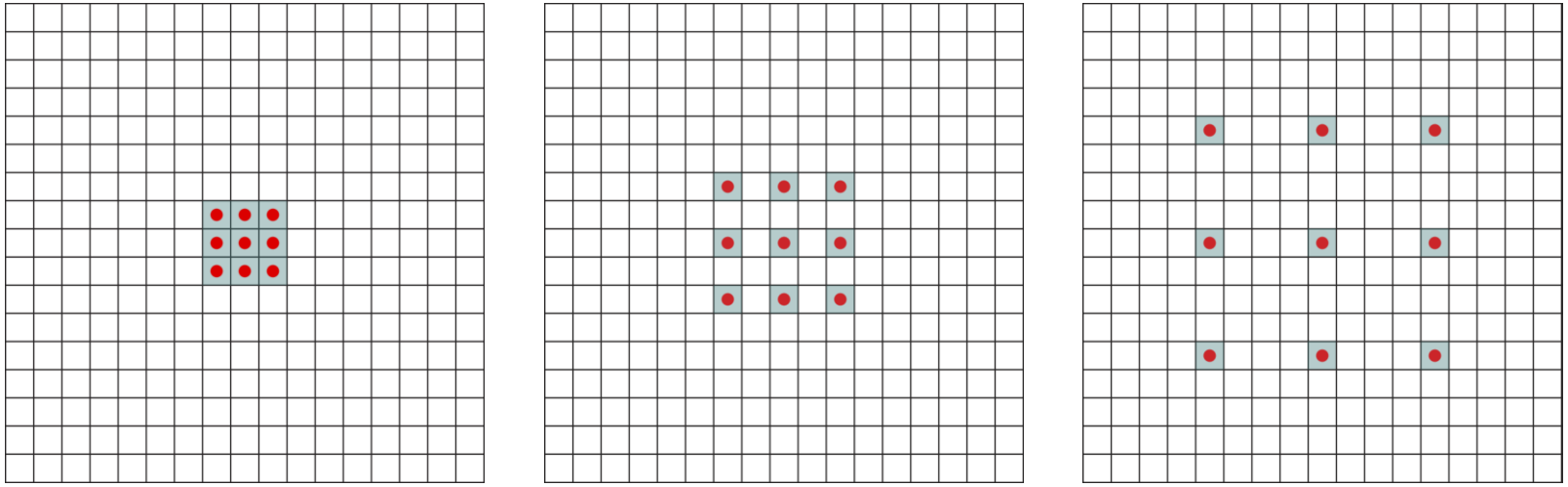
Dilated Convolution



Dilation 2

[Yu and Koltun, *Multi-Scale Context Aggregation by Dilated Convolutions*, ICLR 2016]

Receptive Field Size



Red: Foot print of one dilated convolution

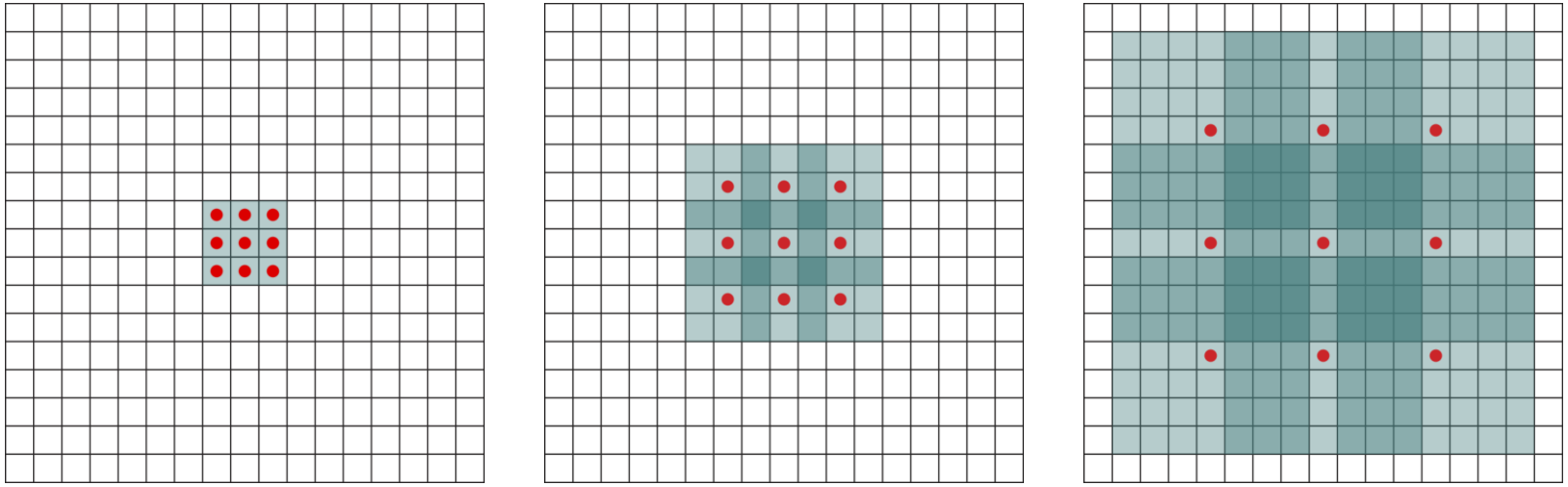
Shaded: Receptive field of convolution

Darker: Receptive field overlap between foot prints

[Yu and Koltun, *Multi-Scale Context Aggregation by Dilated Convolutions*, ICLR 2016]

Receptive Field Size

Receptive field size can increase exponentially with dilated layers



Layer 1 Layer 2 Layer 3

Red: Foot print of one dilated convolution

Shaded: Receptive field of convolution

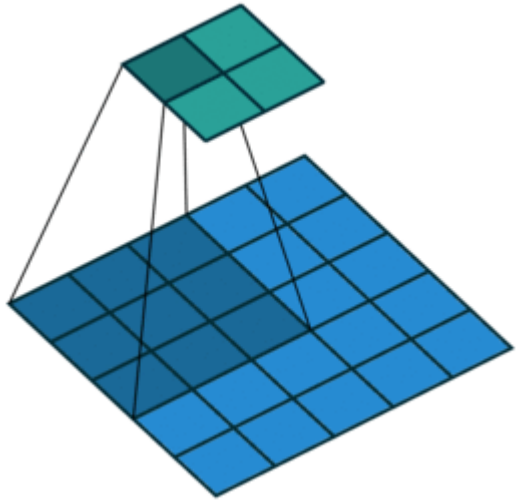
Darker: Receptive field overlap between foot prints

[Yu and Koltun, *Multi-Scale Context Aggregation by Dilated Convolutions*, ICLR 2016]

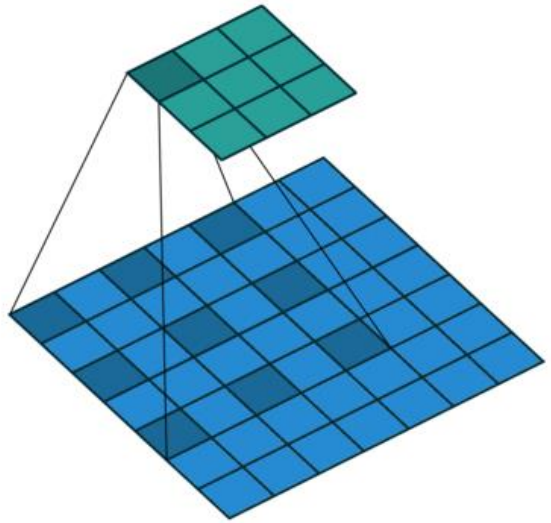
Strided versus Dilated Convolution

Strided convolution

no padding

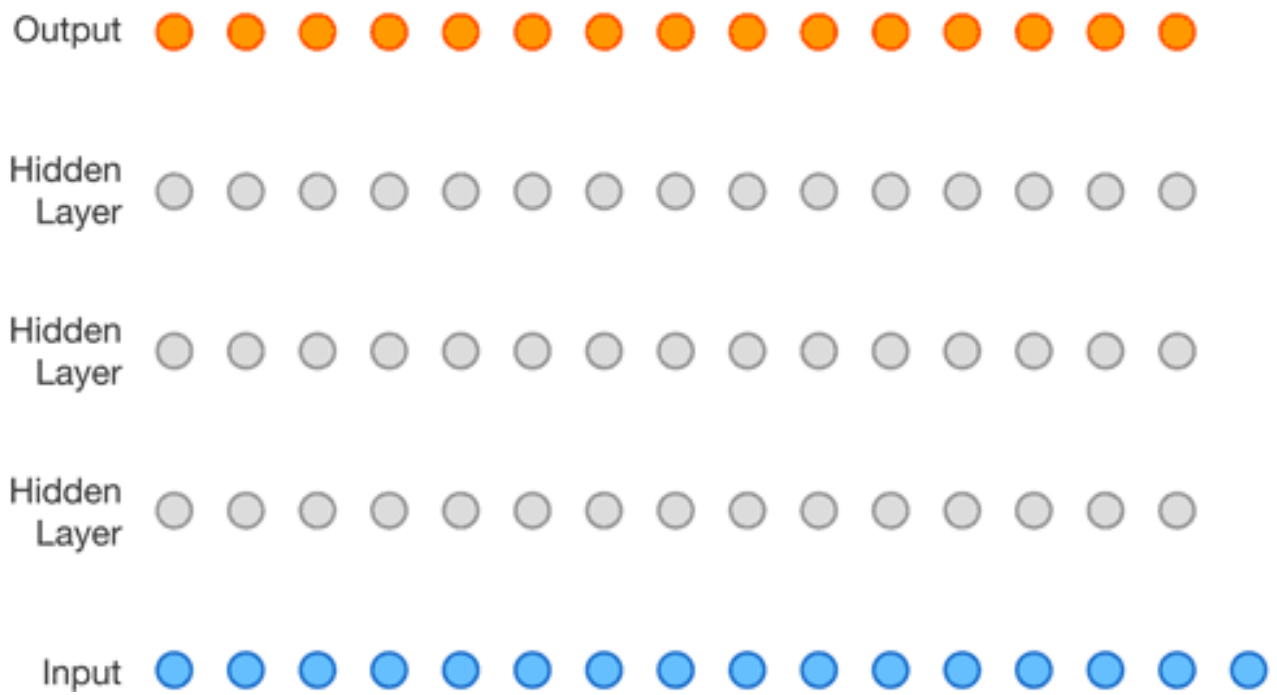


Dilated convolution



Animation source: https://github.com/vdumoulin/conv_arithmetic

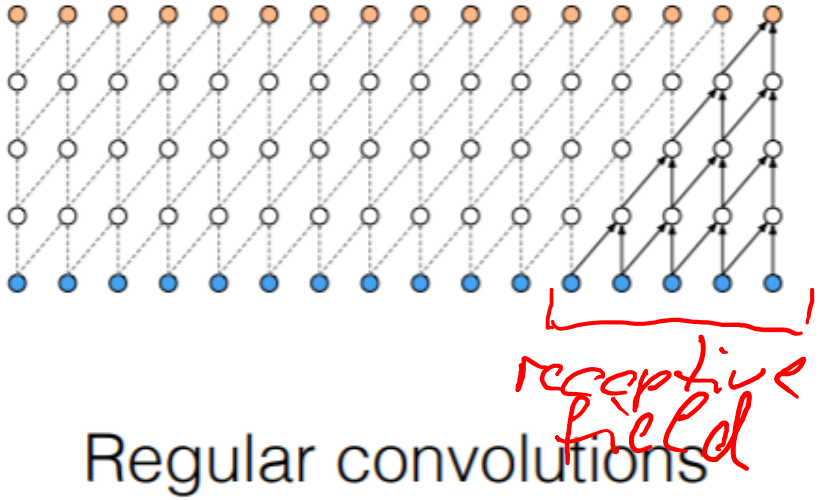
Temporal Convolutional Networks - WaveNet



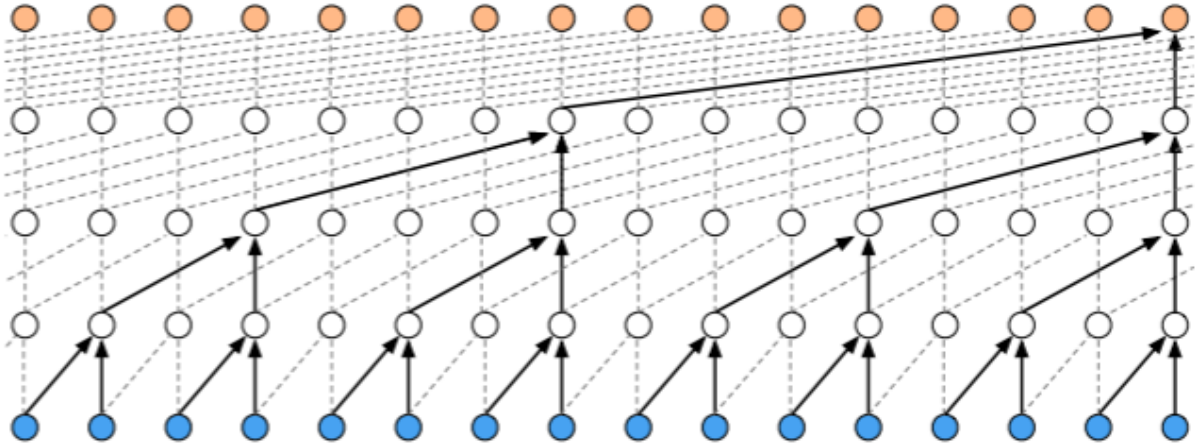
WaveNet animation. Source: Google DeepMind

[van der Oord et al. 2016c]

WaveNet: Large scale temporal dependencies



Regular convolutions



Dilated convolutions

Note: strided Convs cannot be used due to the need to preserve resolution

[van der Oord et al. 2016c]

WaveNet: μ-Law companding

Discrete conditional probabilities

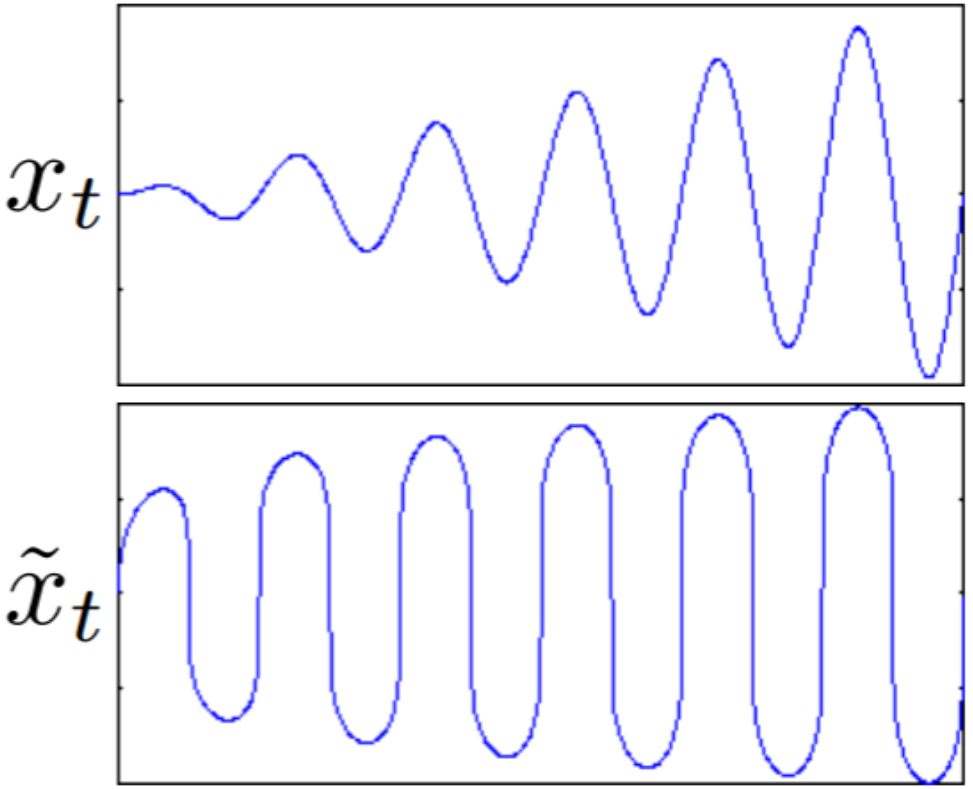
μ-law companding transformation

$$a_t \text{ (16-bit int)} \rightarrow x_t \in [-1, 1]$$

$$\tilde{x}_t = \text{sign}(x_t) \frac{\ln(1 + 255|x_t|)}{\ln 256}$$

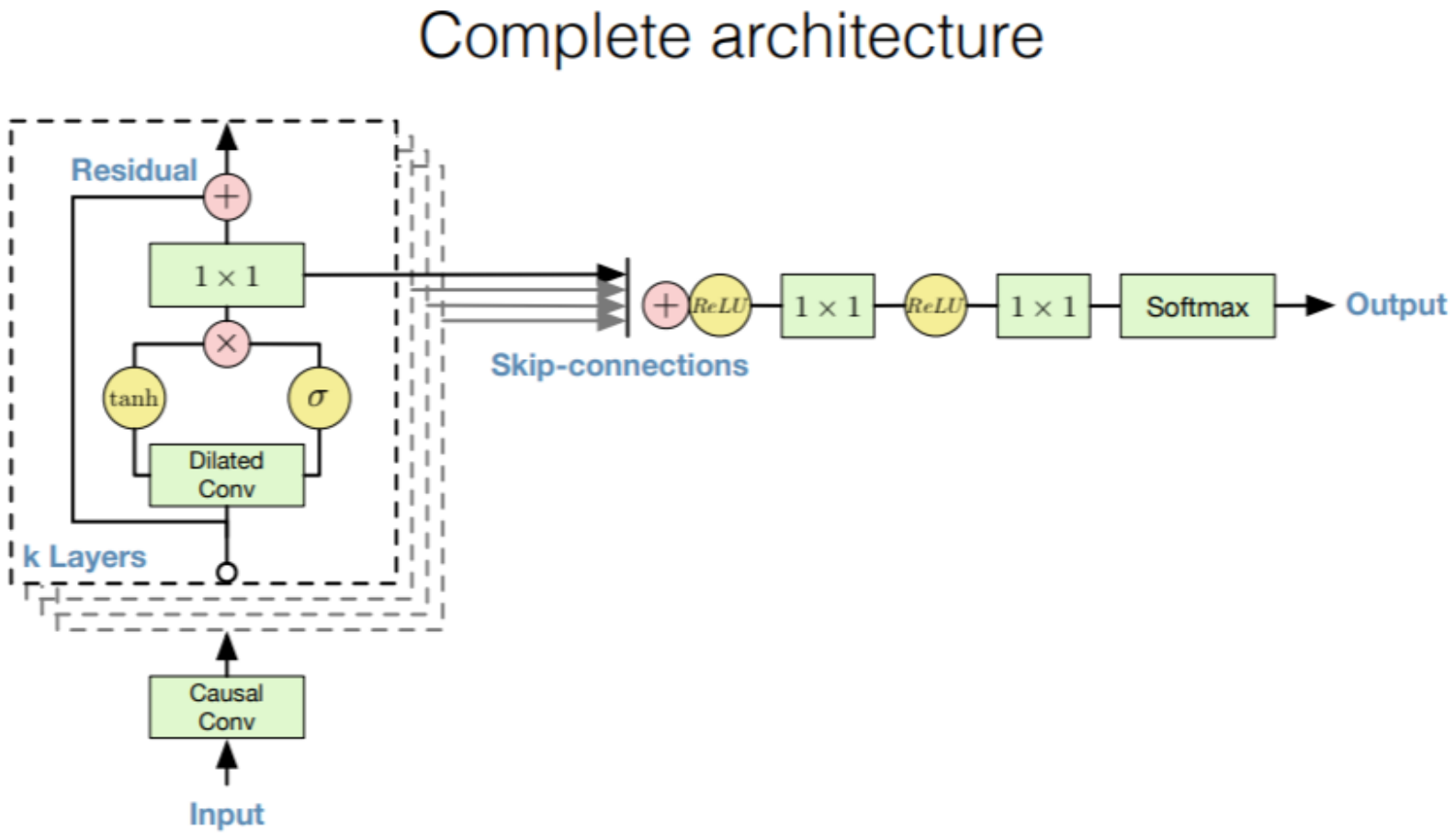
$$\tilde{x}_t \in [-1, 1] \rightarrow \tilde{a}_t \text{ (8-bit int)}$$

quantize back



[van der Oord et al. 2016c]

WaveNet: complete architecture



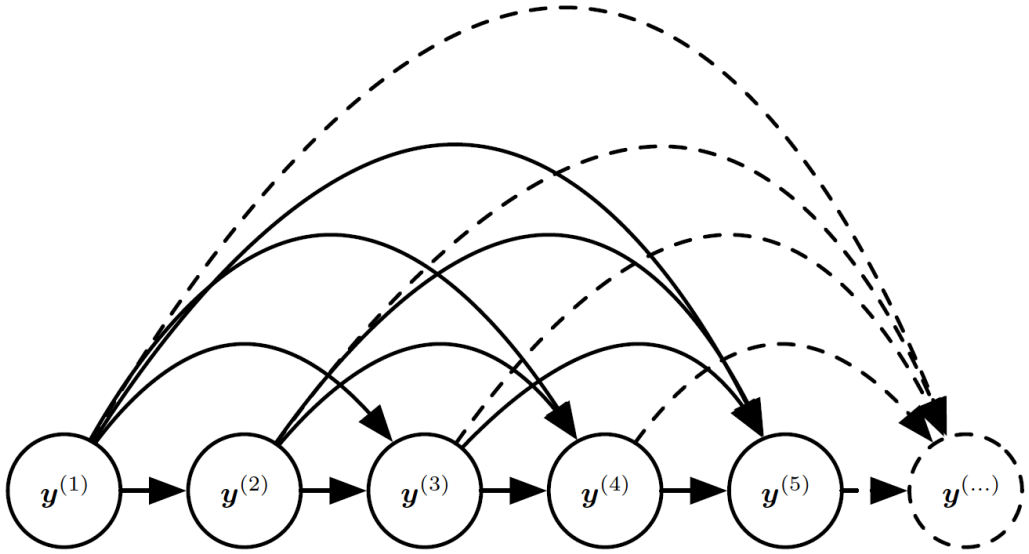
[van der Oord et al. 2016c]

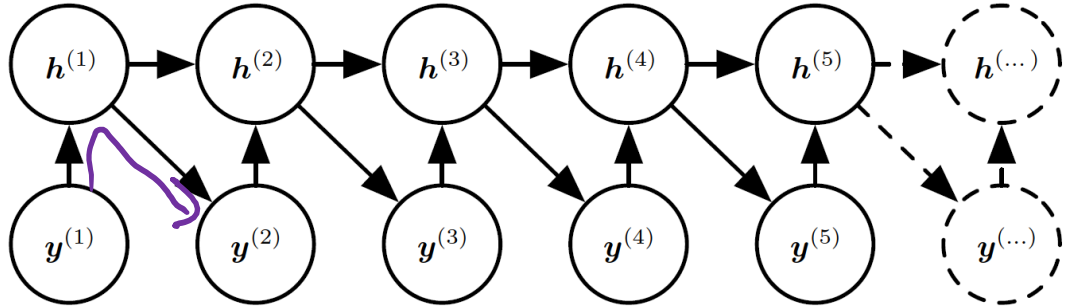
Autoregressive models

Via the chain rule of probabilities we can factorize the joint distribution over the n dimensions:

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

RNNs are autoregressive models





- Hidden layer $h^{(t)}$ summarizes inputs seen up to $t-1$
- Outputs parametrize conditionals $p(x_t|x_{<t})$

Summary: Auto Regressive Generative Models

A Family of autoregressive models

Explicit density model

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

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

Then maximize likelihood of training data

- examples: masked autoencoder distribution estimator (MADE), pixelCNN neural autoregressive distribution estimator (NADE), spatial LSTM, pixelRNN
- Pros: $p(x)$ is tractable, so easy to train, easy to sample (though slower)
- Cons: No natural latent variable representation (but doable see C-VRNN, STCN)

Next

Reinforcement Learning Bootcamp