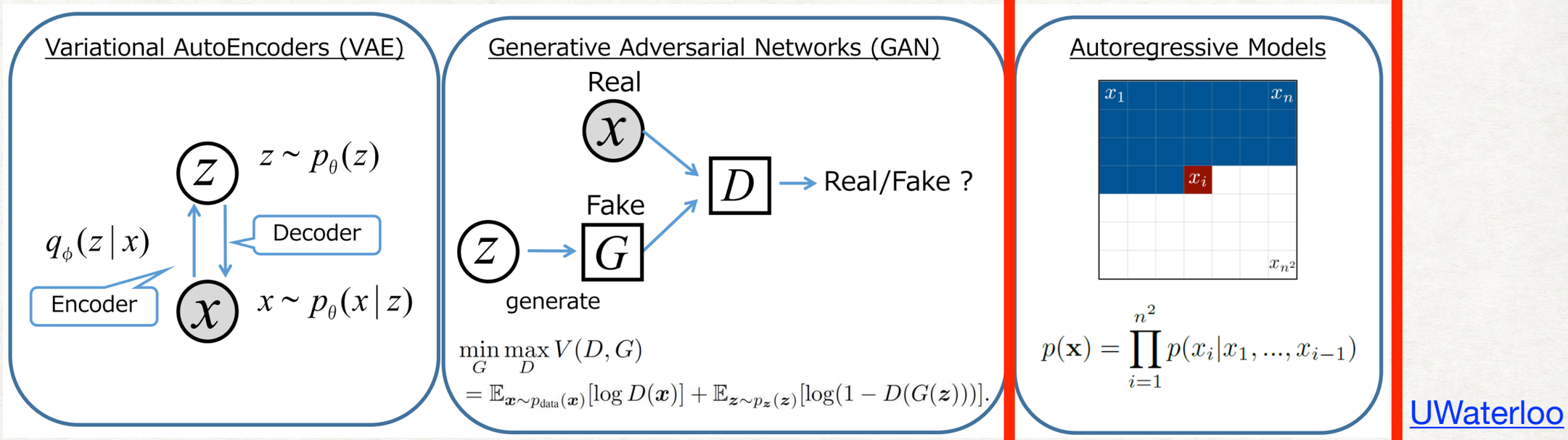
Deep Autoregressive Models

… mainly PixelCNN and Wavenet

Another Way to Generate



UWaterloo

- Use Chain Rule

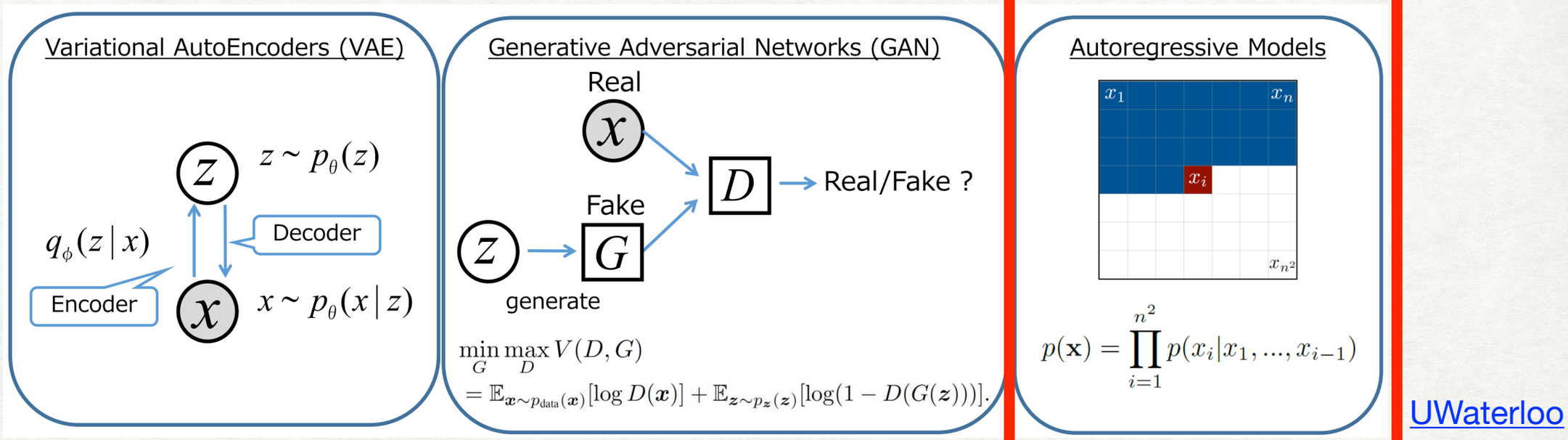
$$P(x_n, x_{n-1}, \ldots, x_2, x_1) = P(x_n \mid x_{n-1}, \ldots, x_2, x_1) * P(x_{n-1} \mid x_{n-2}, \ldots, x_2, x_1) \ldots P(x_2 \mid x_1) * P(x_1)$$

- Engineer Neural Networks to approximate the density functions

$$P(x_n, x_{i-n}, \ldots, x_2, x_1) = \Pi_{i=1}^{n} P_{NN}(x_i \mid x_{i-1}, \ldots, x_2, x_1)$$

- This works because sufficiently complex NN can approximate any function

Another Way to Generate



UWaterloo

- Use Chain Rule

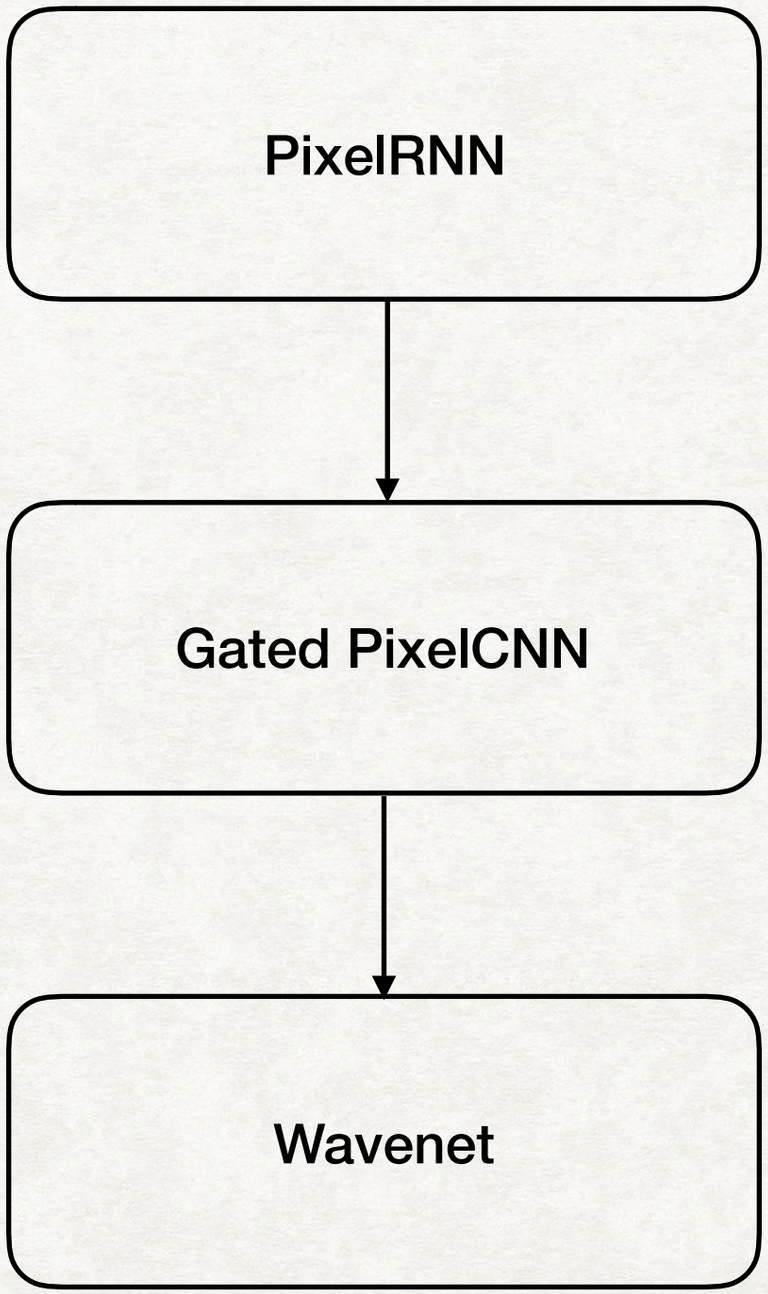
$$P(x_n, x_{n-1}, \ldots, x_2, x_1) = P(x_n \mid x_{n-1}, \ldots, x_2, x_1) * P(x_{n-1} \mid x_{n-2}, \ldots, x_2, x_1) \ldots P(x_2 \mid x_1) * P(x_1)$$

- Engineer Neural Networks to approximate the density functions

$$P(x_n, x_{i-n}, \ldots, x_2, x_1) = \Pi_{i=1}^{n} P_{NN}(x_i \mid x_{i-1}, \ldots, x_2, x_1)$$

- This works because sufficiently complex NN can approximate any function

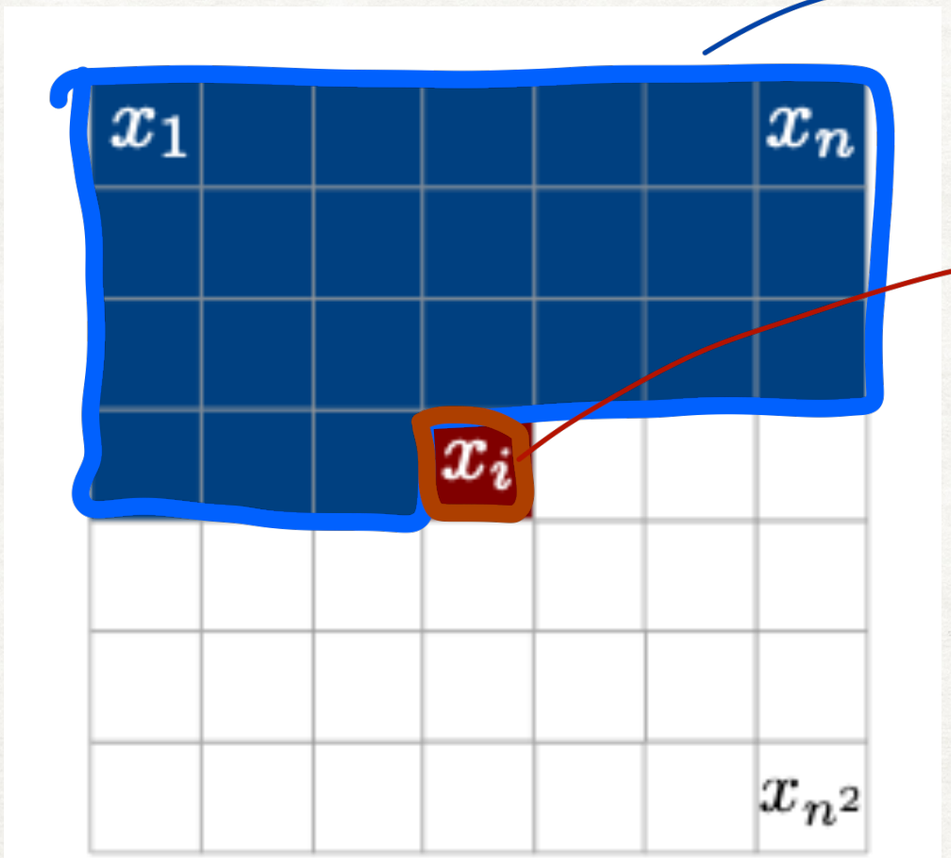
What's Ahead



Just the CNN implementation

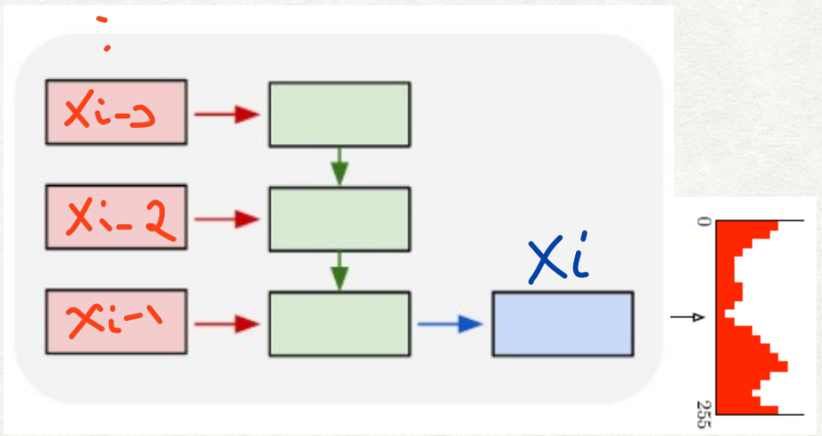
PixelRNN (a naive look)

van den Oord et al, 2016a



$$P(x) = \Pi_{i=1}^{n^2} P(x_i \mid x_{i-1}, \ldots, x_1)$$

- Fix a frame of reference
- Flatten the context pixels and use RNN to approximate the density functions



karpathy

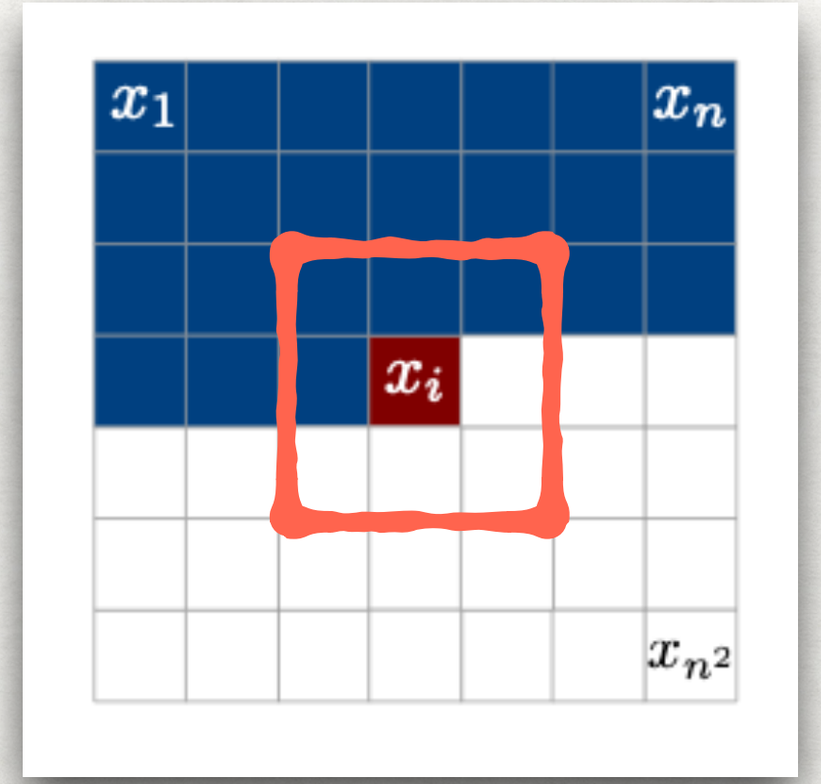
- Pixel values are treated as discrete (0-255)
- Softmax at output to predict class distribution for each pixel
- The original paper had a more efficient implementation using 2D RNNs
- Too complicated; we'll focus on the CNN variant instead

PixelCNN

van den Oord et al, 2016a

RNNs are more expressive but are too slow to train

- Instead use CNNs to predict the pixel value
- Every conditional distribution is modelled as CNN
- A CNN filter uses the neighbouring pixel values to compute the output

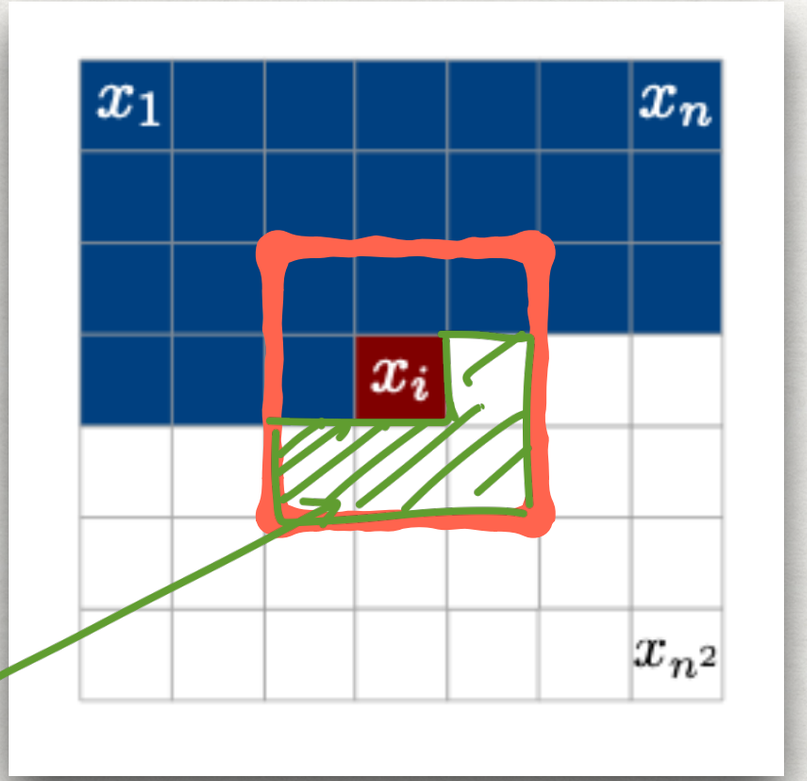


PixelCNN

van den Oord et al, 2016a

RNNs are more expressive but are too slow to train

- Instead use CNNs to predict the pixel value
- Every conditional distribution is modelled as CNN
- A CNN filter uses the neighbouring pixel values to compute the output

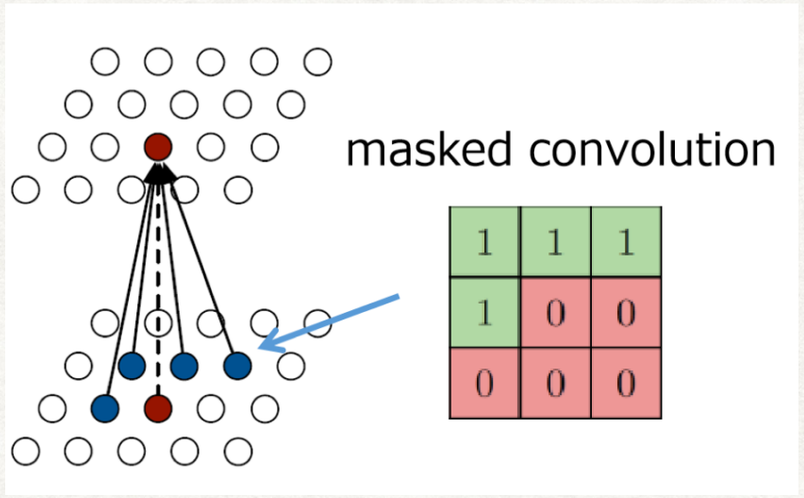


But for this to work two issues need to be fixed

- CNN filter does not obey **causality**
- CNN filter has **limited neighbourhood** and only "sees" part of the context

Fixing Causality

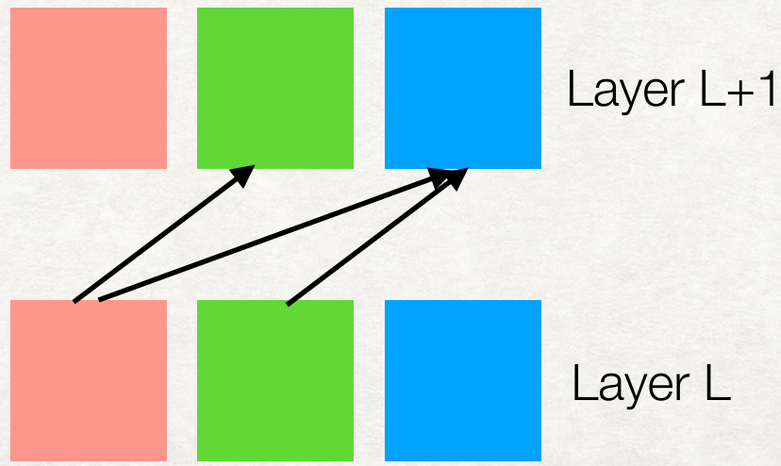
We have to make sure the future doesn't influence the present



Zero out "future" weights in the Conv filter

For colour images

- Divide the # of output channels into 3 groups
- Sample R, then G|R and then B|G, R

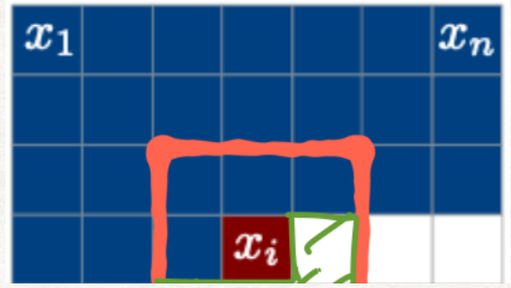


Paper presents 2 types of masks, more on this later…

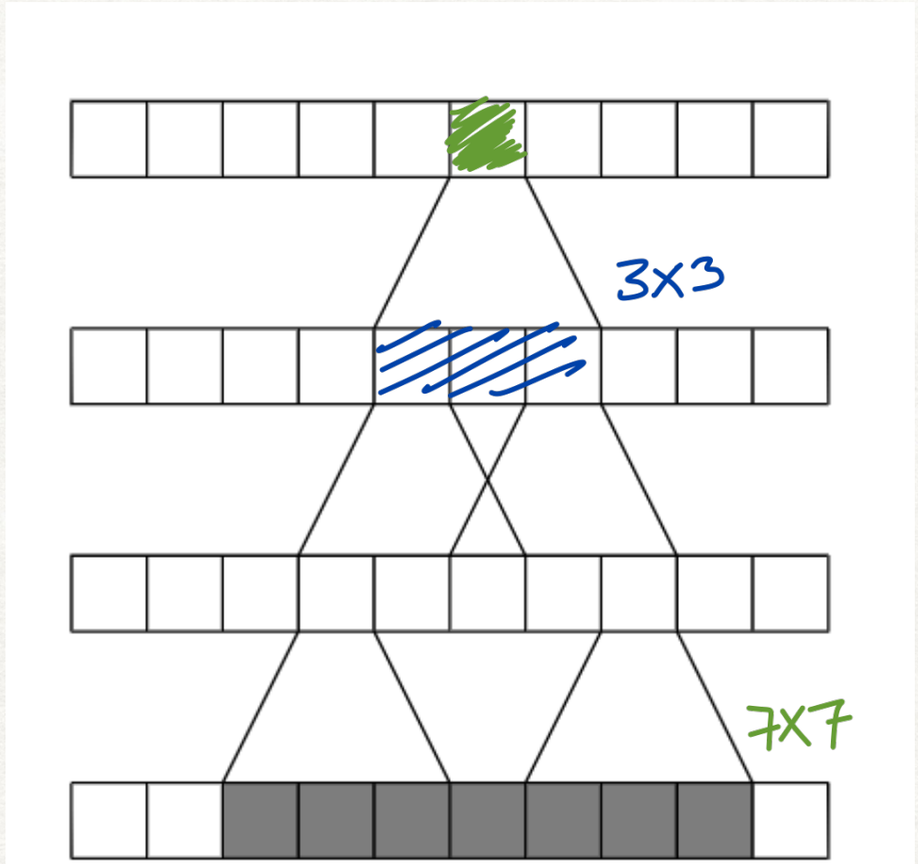
sergeiturukin

Fixing Limited Neighbourhood

Increase the effective receptive field by adding more layers



Discussed in DL course's CNN lecture



Aalto Deep Learning 2019

Combining this with masked filters creates another problem, more on this later…

PixelCNN: Implementation Details

- Two types of masks

A

1	1	1
1	0	0
0	0	0

For the first layer
(connected to the input)

B

1	1	1
1	1	0
0	0	0

All other conv layers

```
mask = torch.ones(1, 1, kernel_sz, kernel_sz)
mask[:, :, kernel_sz//2, kernel_sz//2:] = 0 # zero-out right
mask[:, :. kernel_sz//2+1:] = 0 # zero-out bottom rows
weight *= mask
```

- To maintain same output shape everywhere, no pooling layers

- Use residual connections to speed up convergence

PixelRNN results on CIFAR10

	NLL Test (train)
PixelCNN:	3.14 (3.08)
Row LSTM:	3.07 (3.00)
Diagonal BiLSTM:	**3.00** (2.93)

Gated PixelCNN

van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. RNNs have access to entire neighbourhood of previous pixels
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

After fixing these issues, the authors were able to get better results from PixelCNNs

Gated PixelCNN on CIFAR10

	NLL Test (train)
PixelCNN	3.14 (3.08)
PixelRNN	3.00 (2.93)
Gated PixelCNN:	3.03 (2.90)

Let's see how…

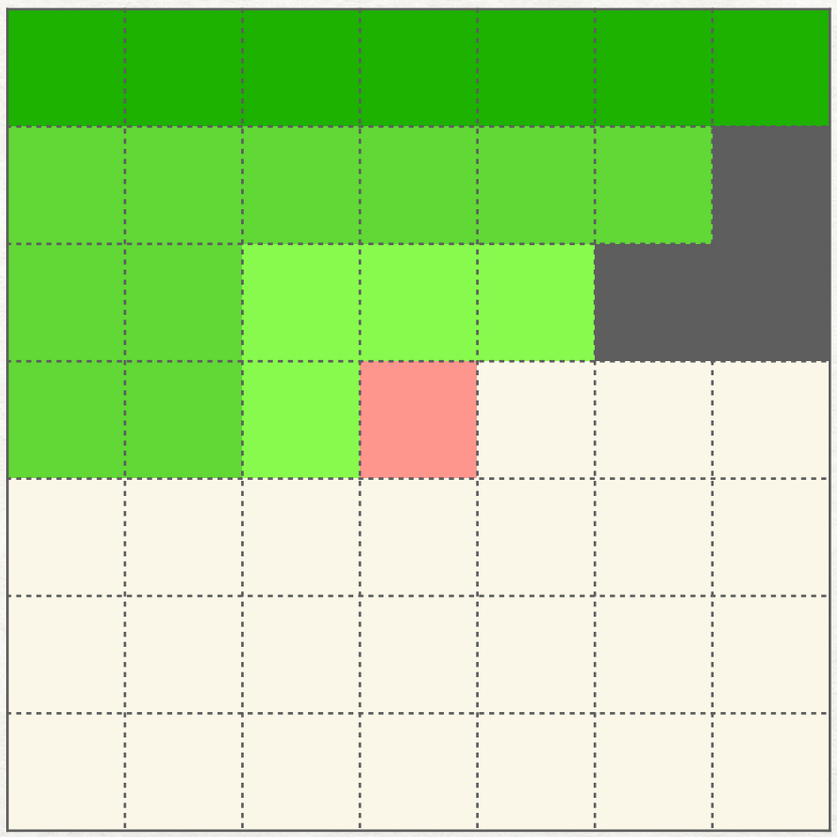
Gated PixelCNN

van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

We sorta fixed this by adding more layers to increase receptive field
But due to masked filters, this creates a ***blind spot***



- Here, darker shades => influence from farther layer

- Due to masked convolutions, the grey coloured pixels never influence the output pixel (red)

- This happens no matter how many layers we add

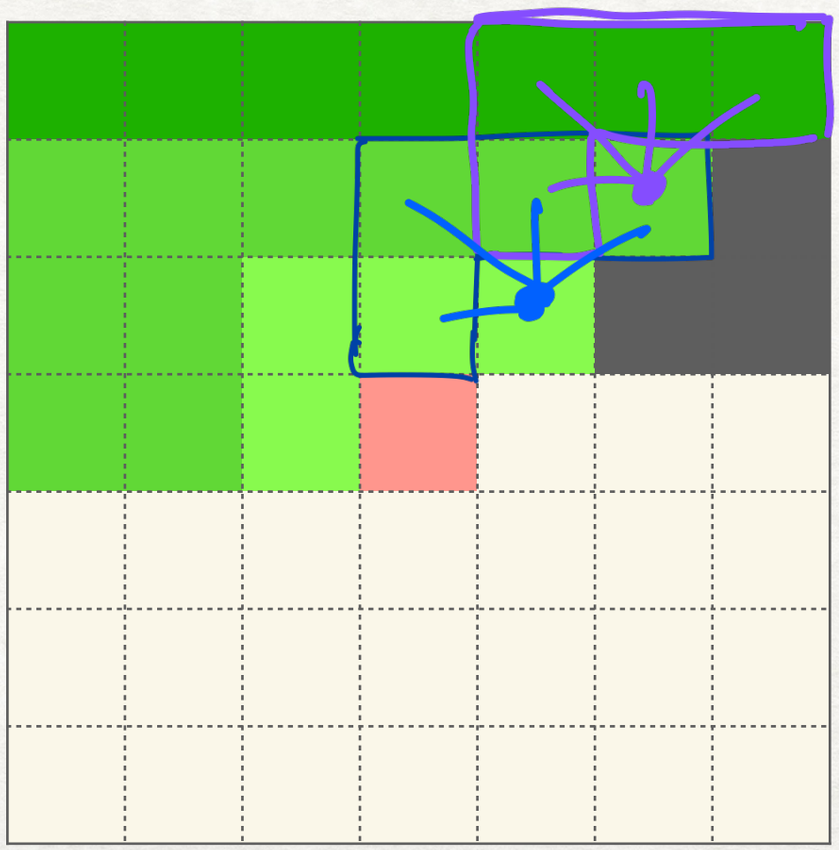
Gated PixelCNN

van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

We sorta fixed this by adding more layers to increase receptive field
But due to masked filters, this creates a ***blind spot***



- Here, darker shades => influence from farther layer

- Due to masked convolutions, the grey coloured pixels never influence the output pixel (red)

- This happens no matter how many layers we add

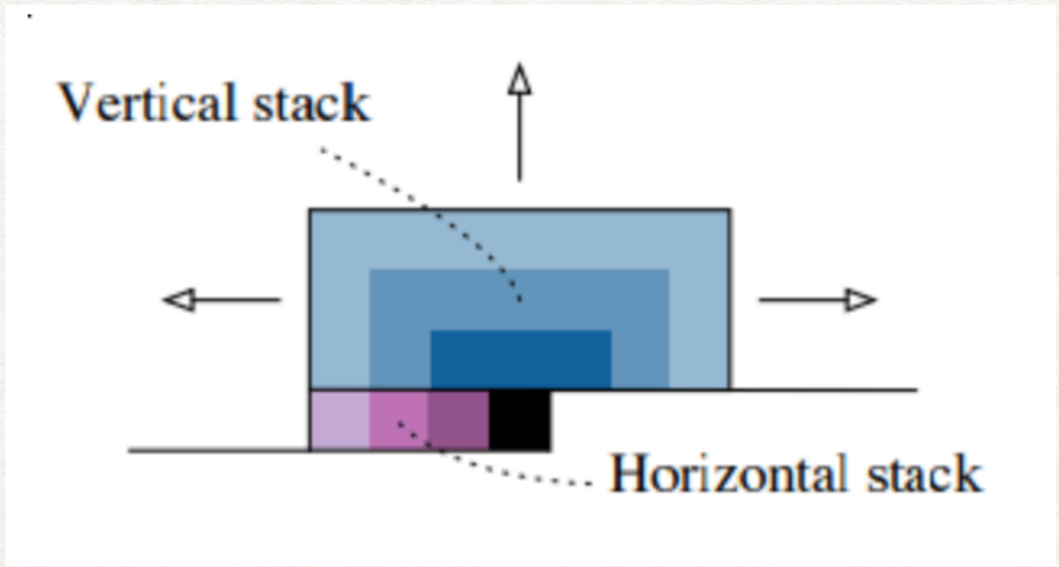
Gated PixelCNN

van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

Blindspot problem is fixed by splitting <u>each</u> convolutional layers into Horizontal and Vertical stacks



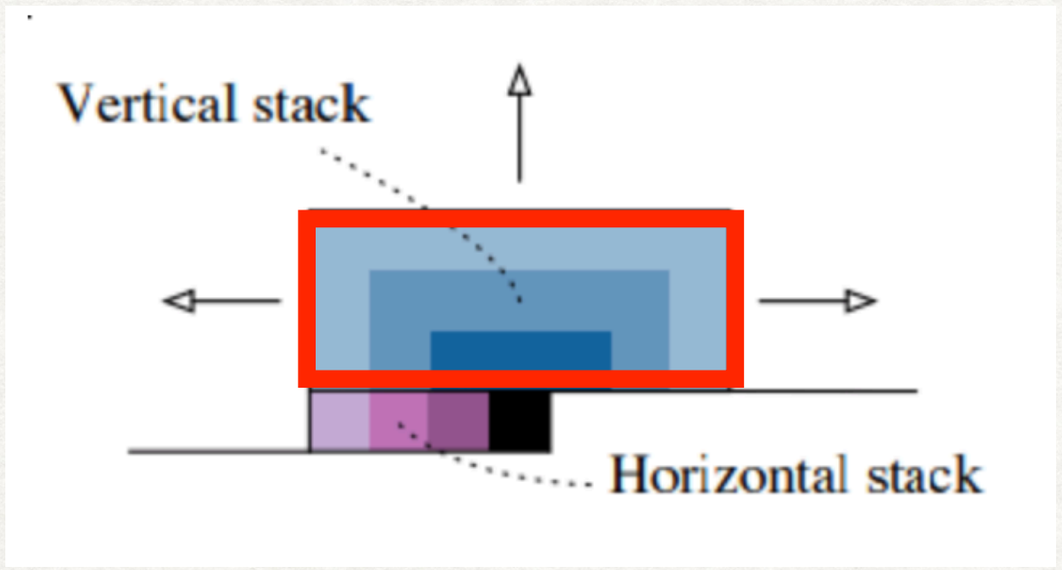
Gated PixelCNN

van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

Blindspot problem is fixed by splitting <u>each</u> convolutional layers into Horizontal and Vertical stacks



- ***Vertical stack only looks at the rows above the output pixel***

- Horizontal stack only looks at pixels to the left of output pixel in the same row

- These outputs are then combined after each layer

- To maintain causality constraint, horizontal stack can see the vertical stack but not vice versa

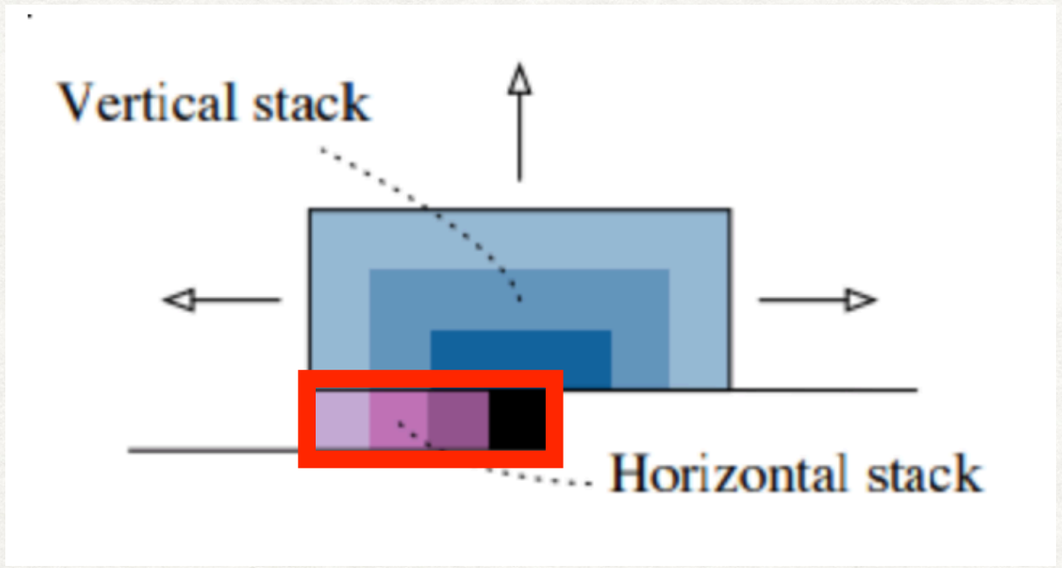
Gated PixelCNN

van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

Blindspot problem is fixed by splitting <u>each</u> convolutional layers into Horizontal and Vertical stacks



- Vertical stack only looks at the rows above the output pixel

- ***Horizontal stack only looks at pixels to the left of output pixel in the same row***

- These outputs are then combined after each layer

- For causality, horizontal stack can see the vertical stack but not vice versa

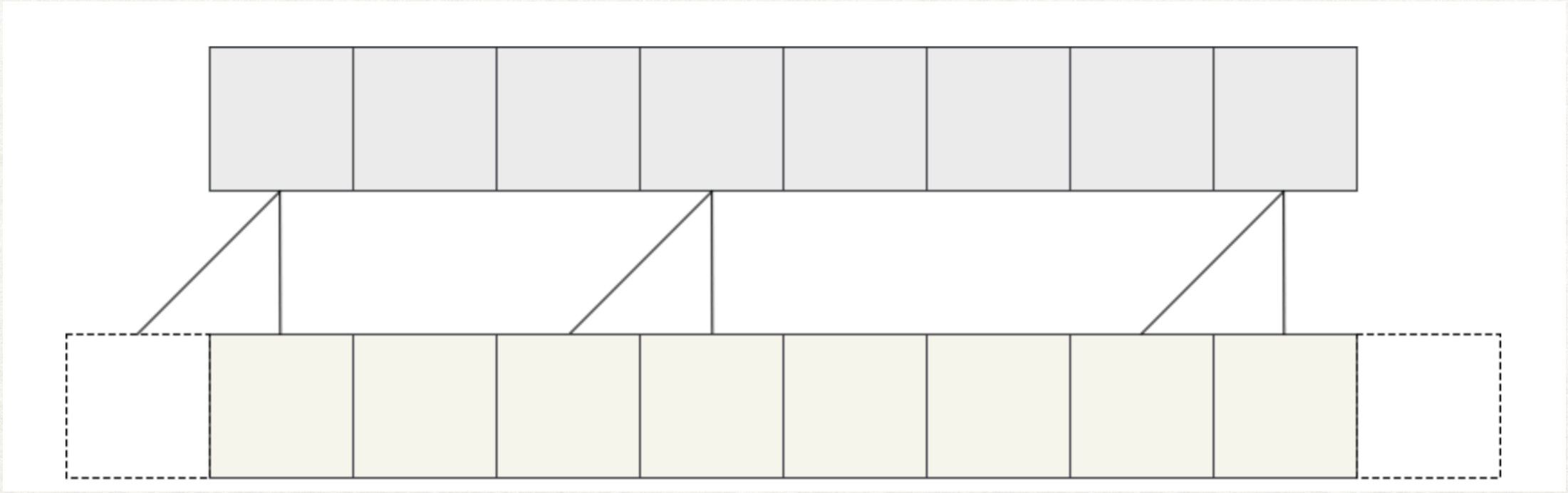
Gated PixelCNN

van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

For horizontal stack, avoid masking filters by choosing filter of size `(1 x kernel_size/2 + 1)`



sergeiturukin

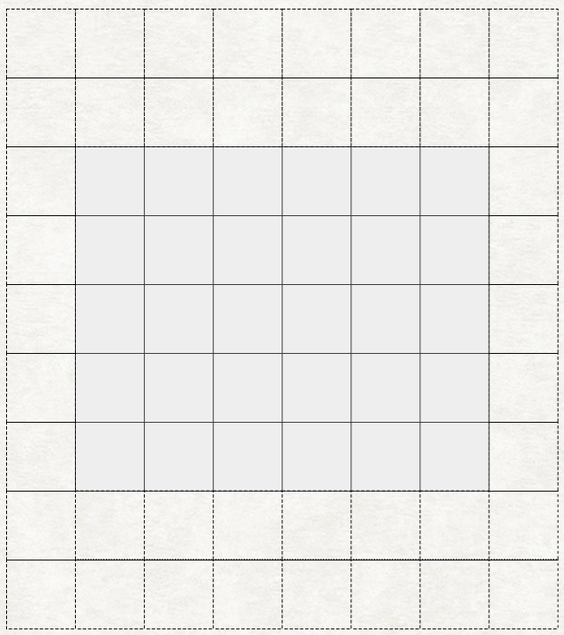
Gated PixelCNN

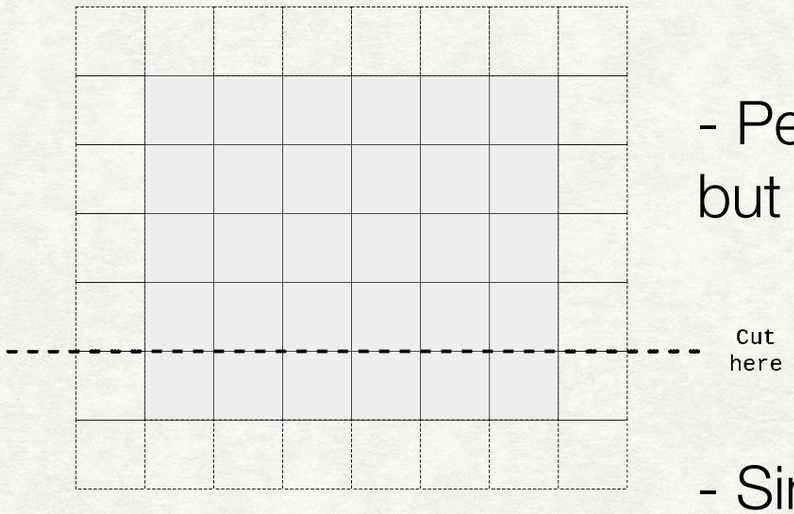
van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

For vertical stack, avoid masking filters by choosing filter of size
`(kernel_size/2 + 1 x kernel_size)`





- Add one more padding layer at top and bottom

- Perform normal convolution but just crop the output

- Since output and input dimensions are to be kept the same, this effectively shifts the output up by 1 row

sergeiturukin

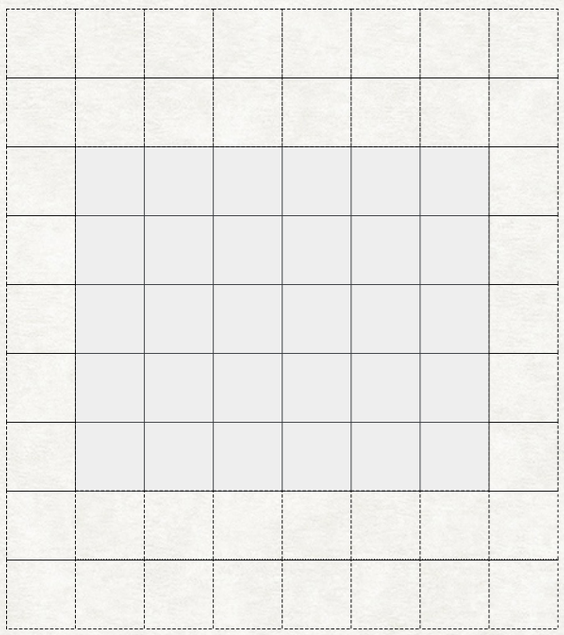
Gated PixelCNN

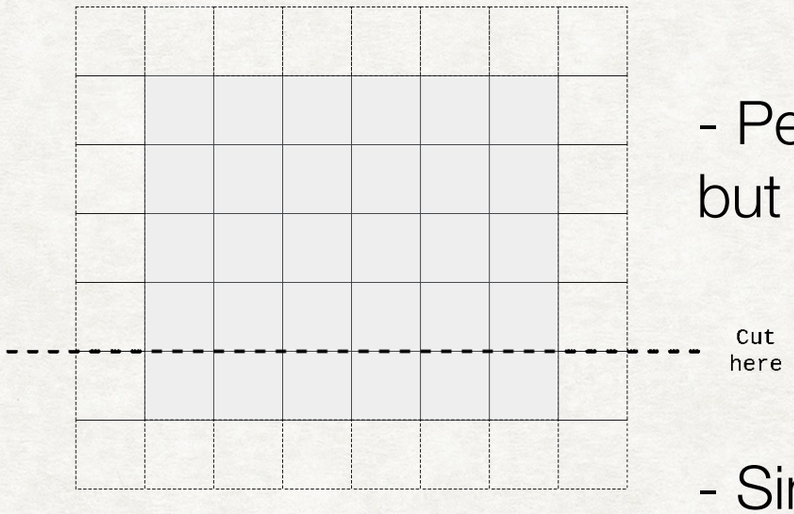
van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

For vertical stack, avoid masking filters by choosing filter of size
`(kernel_size/2 + 1 x kernel_size)`





- Add one more padding layer at top and bottom

- Perform normal convolution but just crop the output

- Since output and input dimensions are to be kept the same, this effectively shifts the output up by 1 row

sergeiturukin

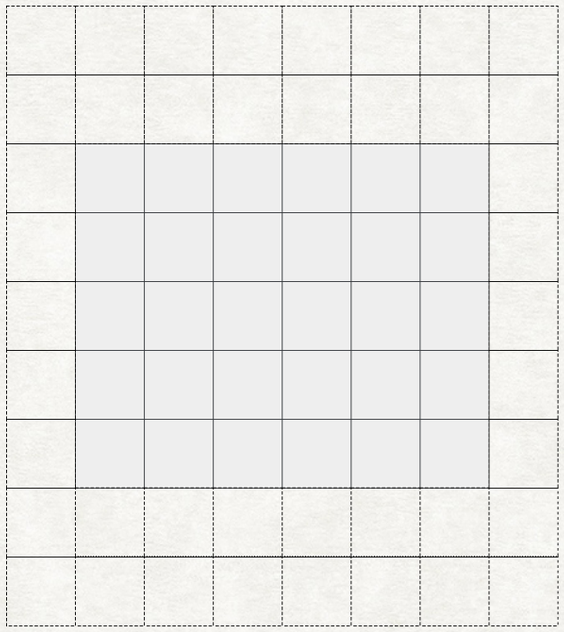
Gated PixelCNN

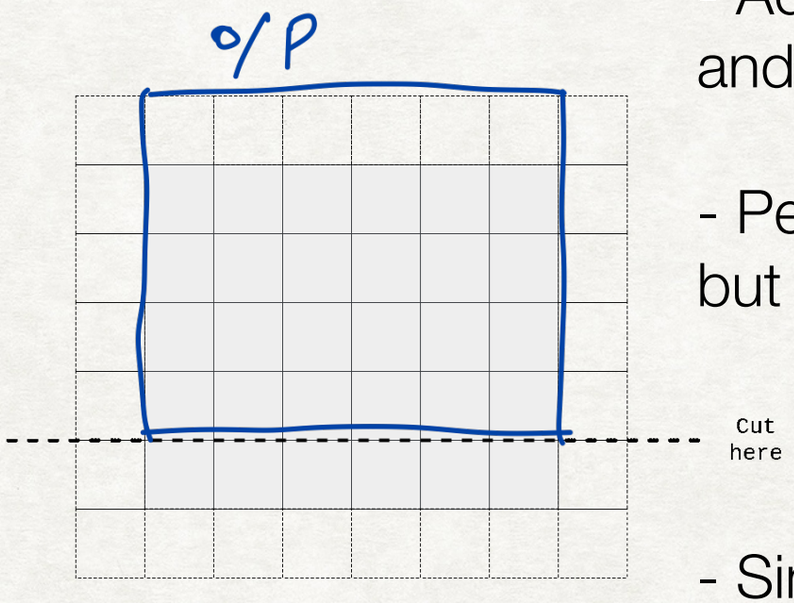
van den Oord et al, 2016b

PixelRNN outperforms PixelCNN due to two reasons:

1. **RNNs have access to entire neighbourhood of previous pixels**
2. RNNs have multiplicative gates (due to LSTM cells), which are more expressive

For vertical stack, avoid masking filters by choosing filter of size `(kernel_size/2 + 1 x kernel_size)`





- Add one more padding layer at top and bottom

- Perform normal convolution but just crop the output

- Since output and input dimensions are to be kept the same, this effectively shifts the output up by 1 row

sergeiturukin

Gated PixelCNN

van den Oord et al, 2016b

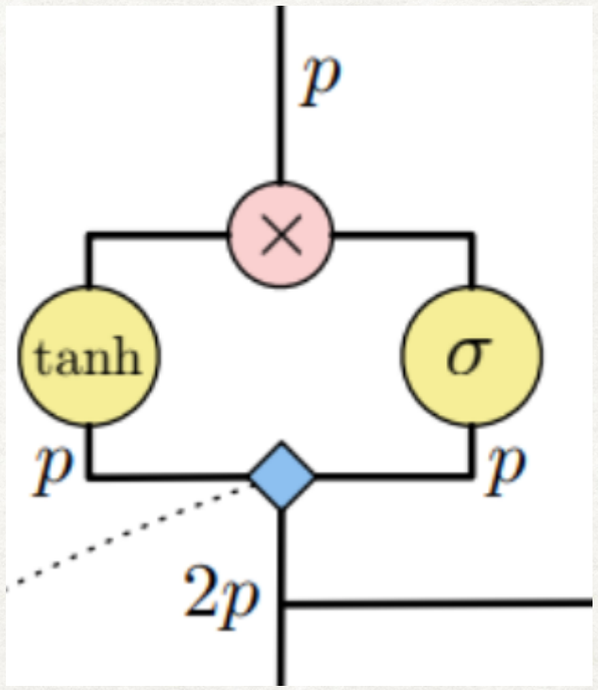
PixelRNN outperforms PixelCNN due to two reasons:

1. RNNs have access to entire neighbourhood of previous pixels
2. **RNNs have multiplicative gates (due to LSTM cells), which are more expressive**

Replace ReLU with this gated activation function

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

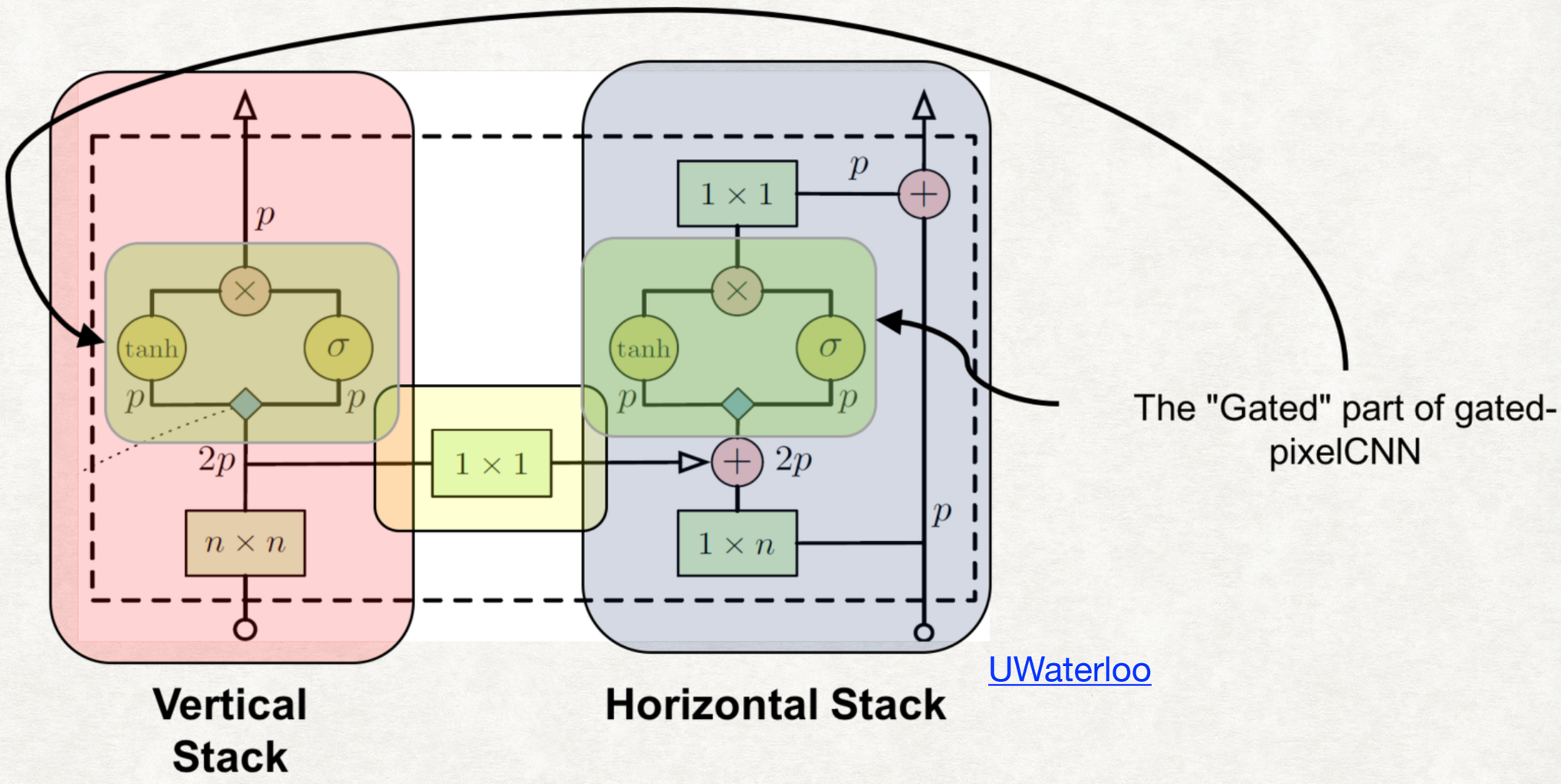
* is conv operation



- Split the feature maps in half and pass them through the tanh and sigmoid functions

- Compute element-wise product

Gated PixelCNN: All of it



UWaterloo

Notice:

- These connections are per layer
- Vertical stack is added to horizontal but not other way around
- Residual connections in horizontal stack
- Apart from this, there are also layer-wise skip connections that are added together before output layer

PixelCNN Conditioning

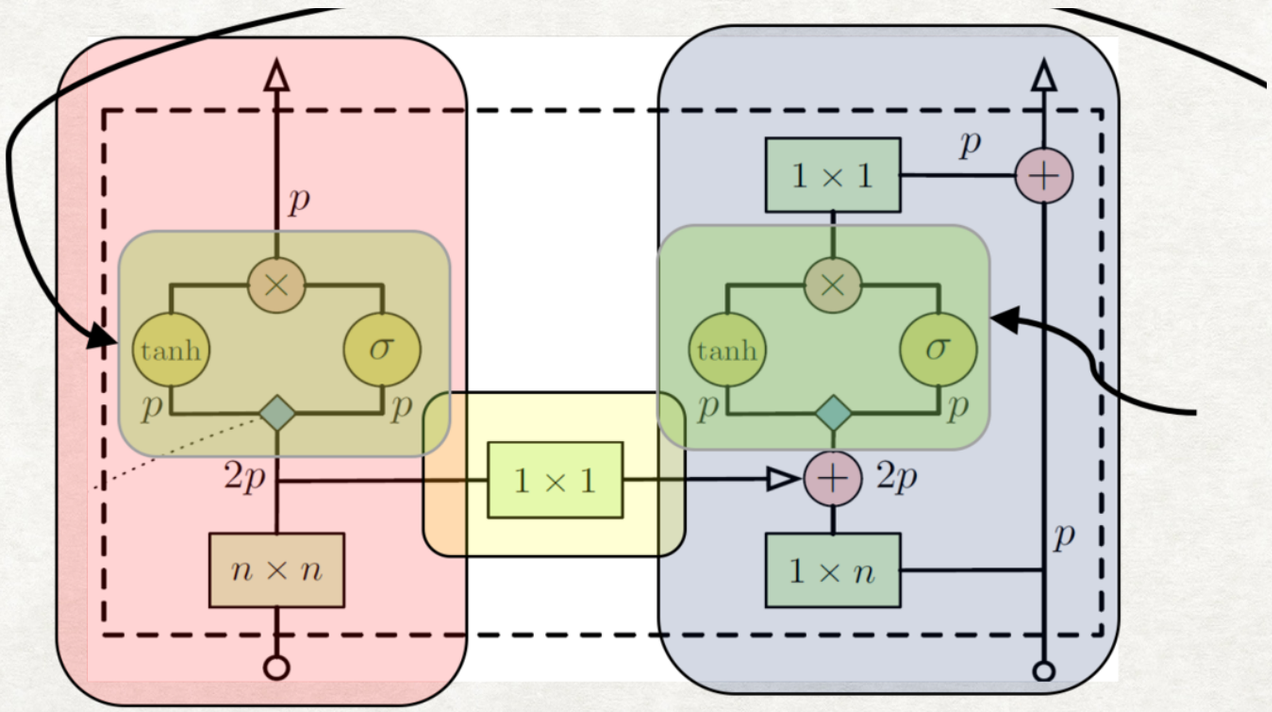
We can condition our distribution on some latent variable h

This latent variable (which can be one-hot encoded for classes) is passed through the gating mechanism

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

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

V is a matrix of size `dim(h) x channel size`



PixelCNN Conditioning

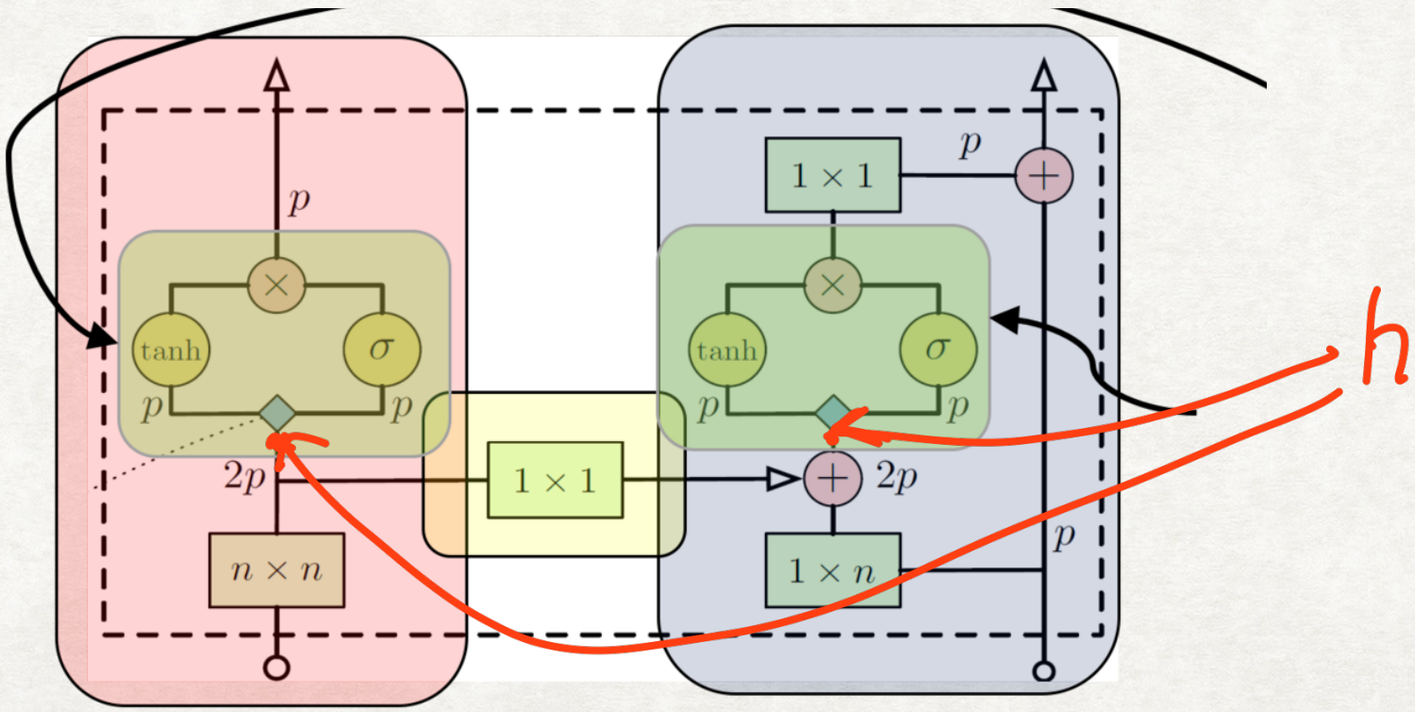
We can condition our distribution on some latent variable h

This latent variable (which can be one-hot encoded for classes) is passed through the gating mechanism

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

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

V is a matrix of size `dim(h) x channel size`

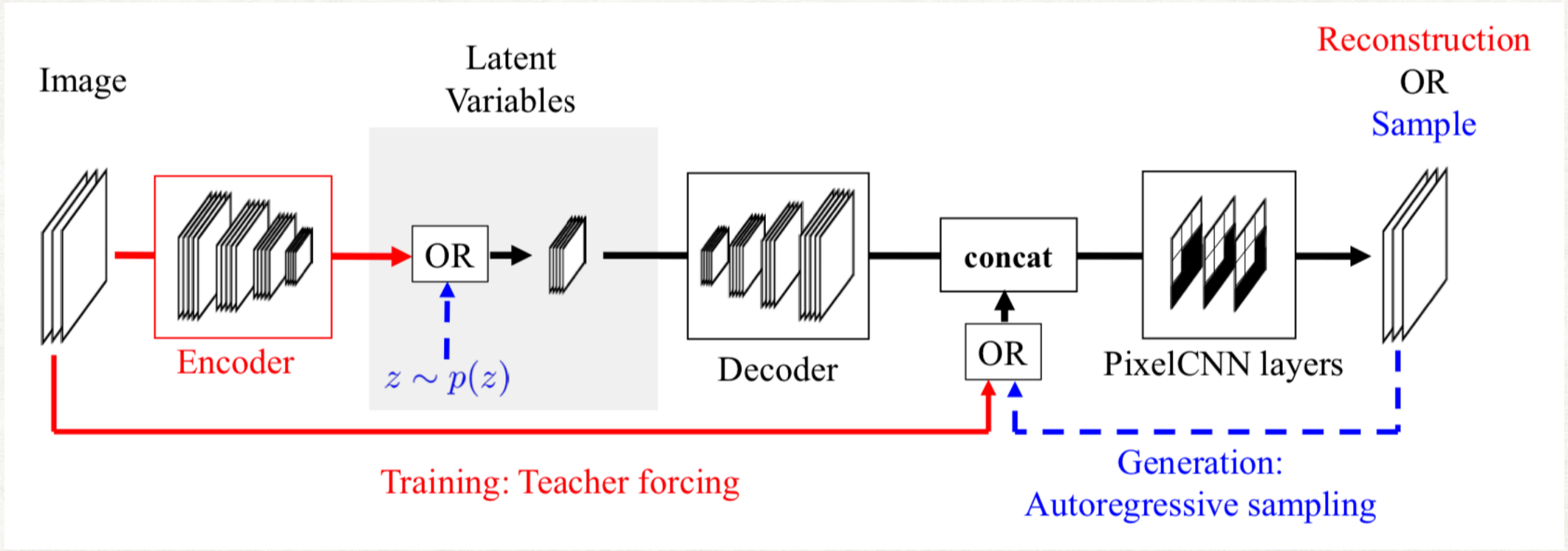


PixelCNN as Decoders

- Without modification, this conditioned PixelCNN can be used as a decoder in an AutoEncoder architecture

- It will be conditioned on the latent representation learned by the encoder

PixelVAE, Gulrajani et al, 2016



Okay, Google… What are Wavenets?

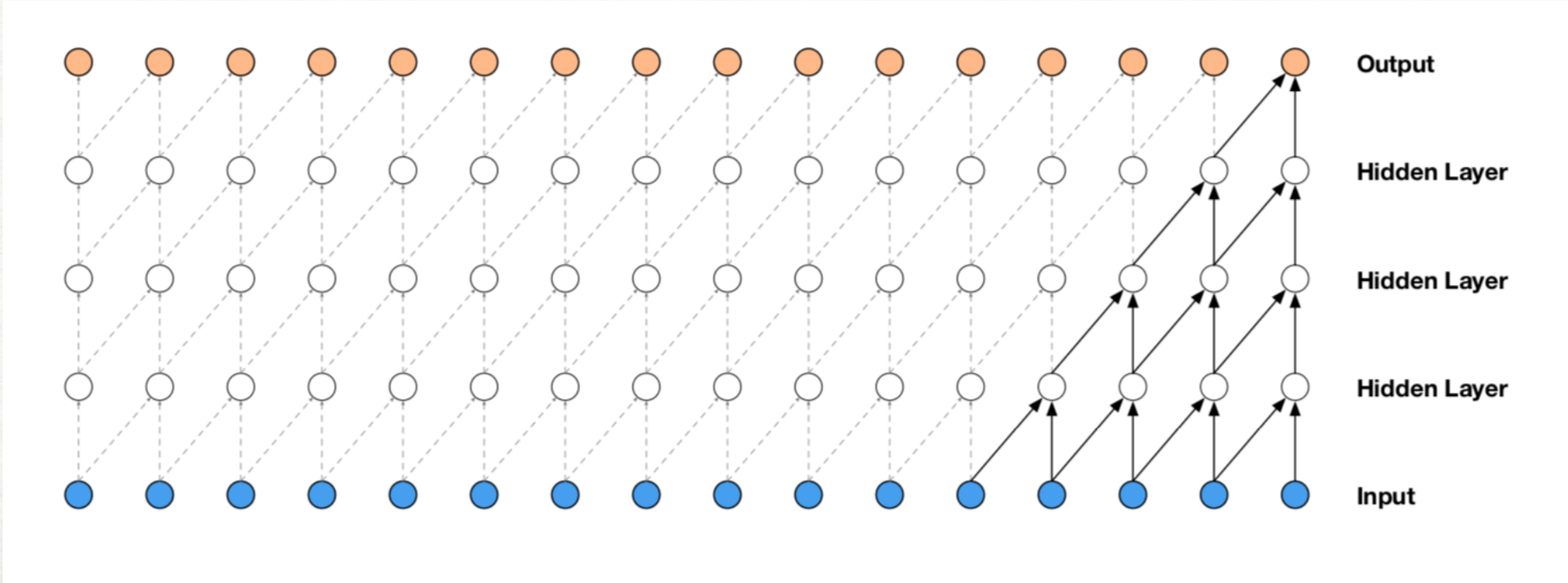
van den Oord et al, 2016c

- Extends PixelCNN to audio sequences - 1D CNN

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

- State-of-the-art in Text to Speech (TTS);
  Powers the Google Assistant

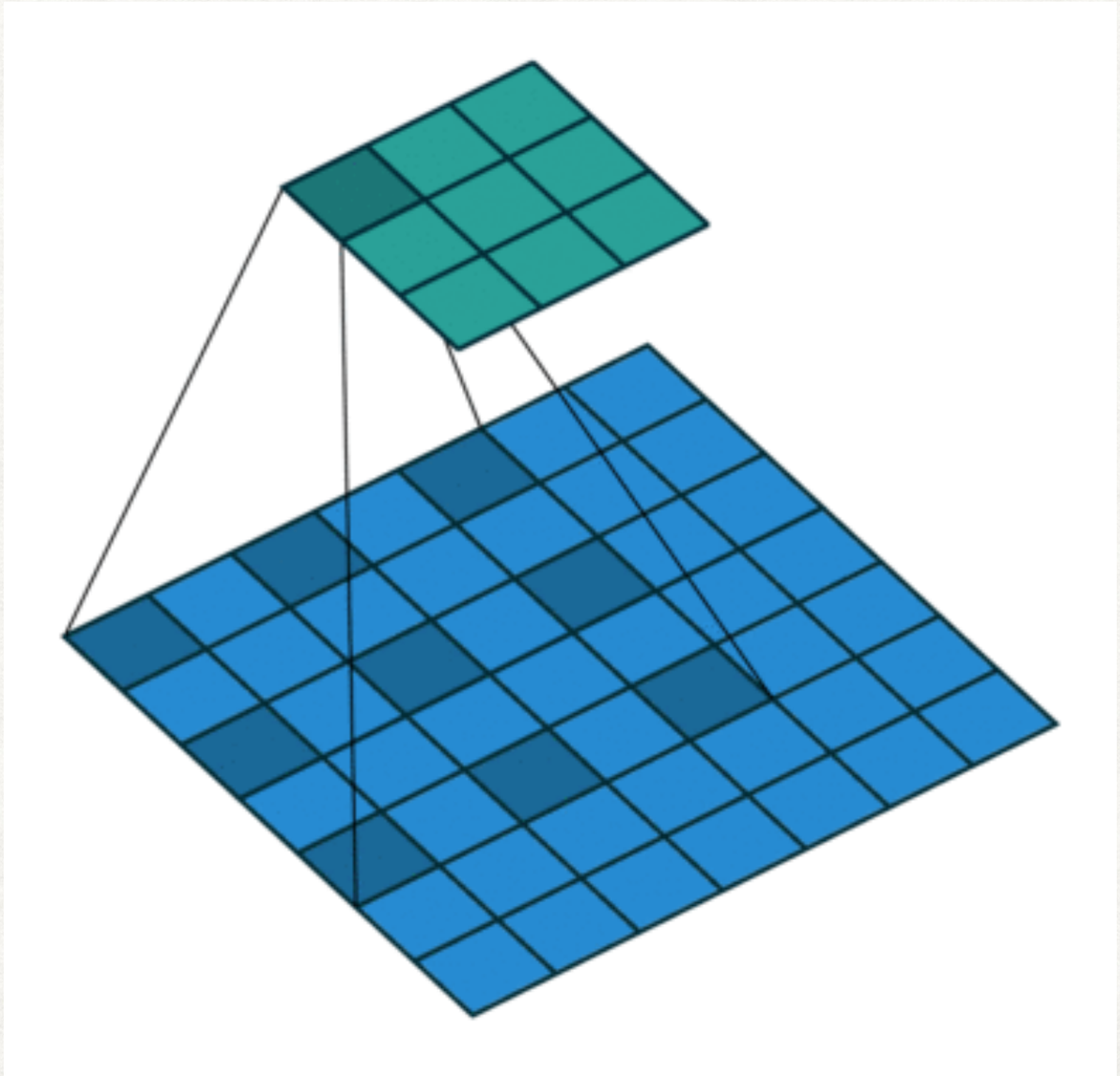
- No masking needed for 1D, just do normal convolution and shift the output



Dilated Convolutions

Dilated convolution allows the network to operate on a coarser scale

- Use a larger than original filter and zero-out some pixels
- Similar to pooling or strides
- By stacking together many dilated conv layers, the effective receptive field can grow much faster

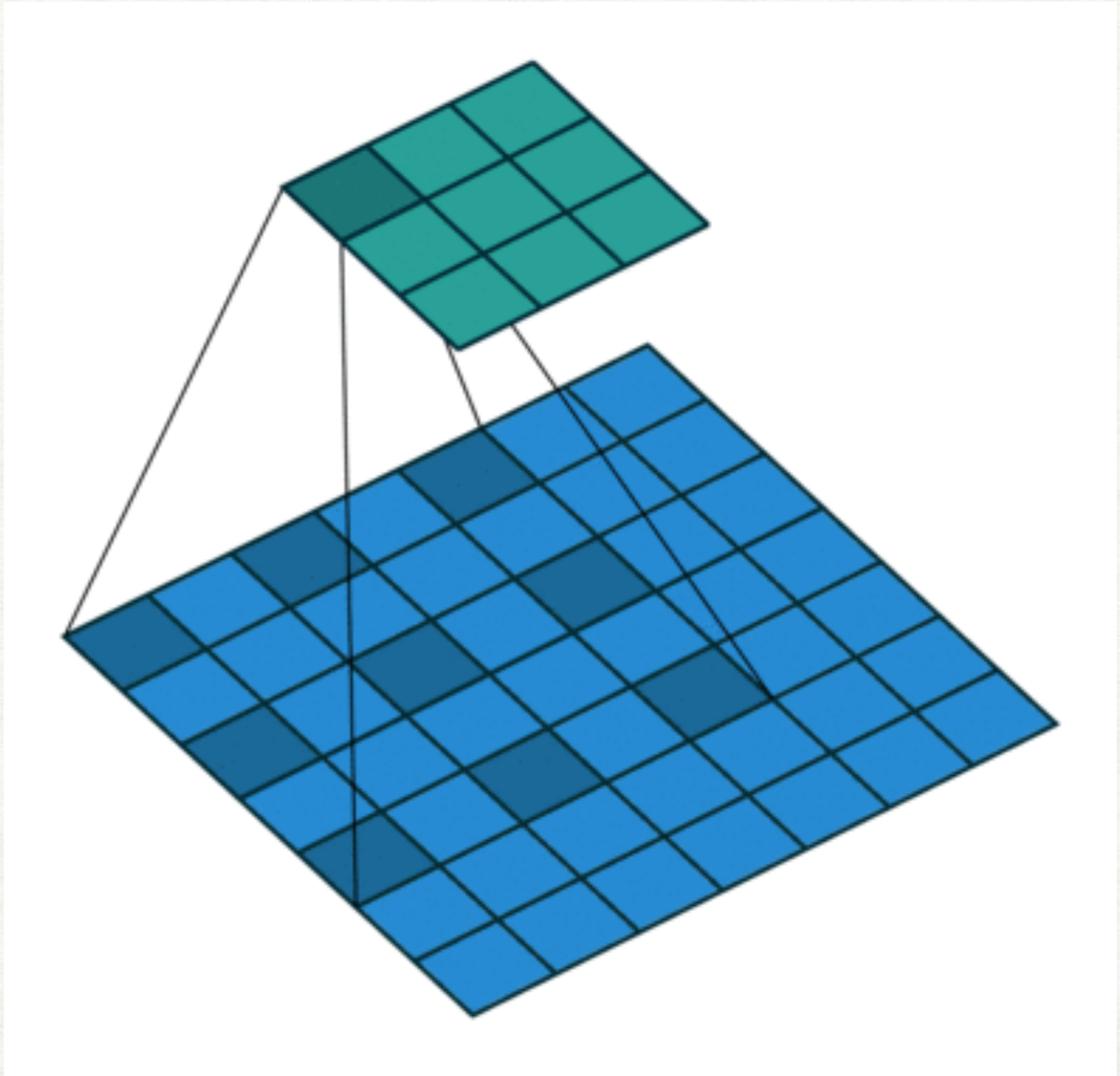


vdumoulin

Dilated Convolutions

Dilated convolution allows the network to operate on a coarser scale

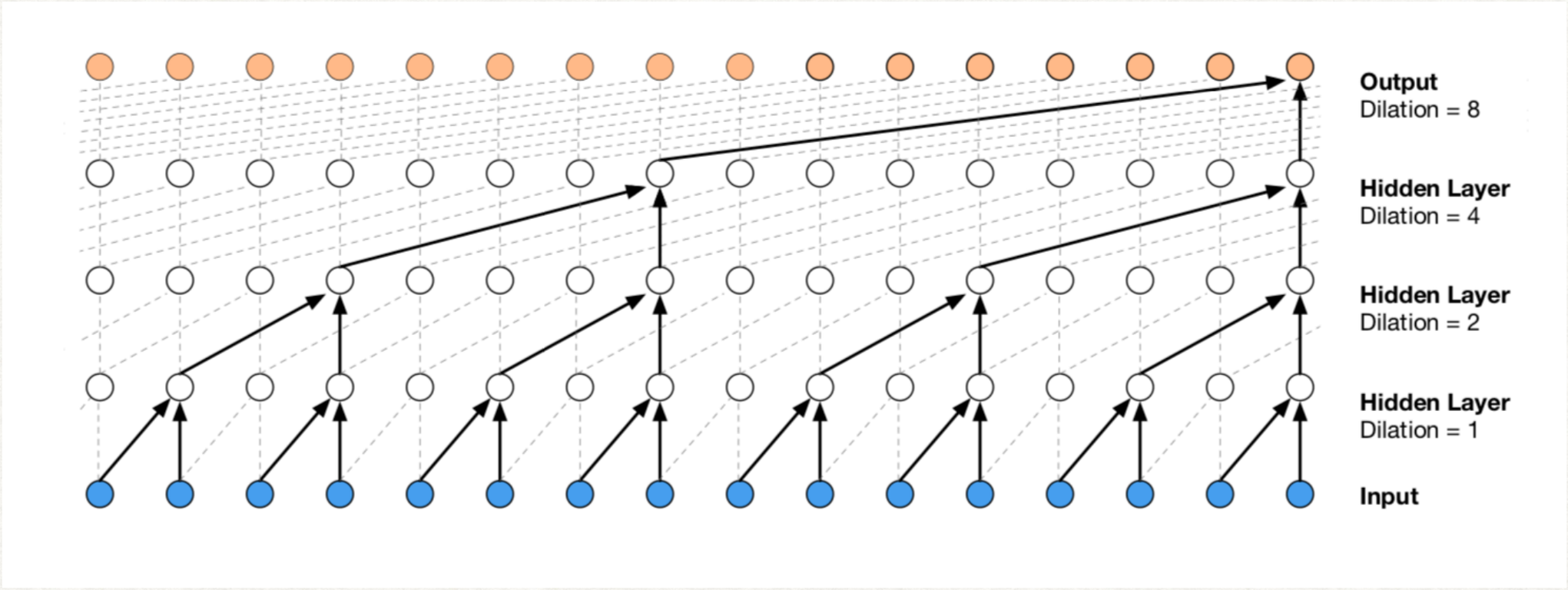
- Use a larger than original filter and zero-out some pixels
- Similar to pooling or strides
- By stacking together many dilated conv layers, the effective receptive field can grow much faster



vdumoulin

Dilated Convolutions in Wavenet

- Dilation factor is doubled after each layer up to a limit, then repeated
- Eg., 1, 2, 4, …, 512, 1, 2, 4, …, 512, etc
- Exponentially increasing dilation => exponentially increasing receptive field

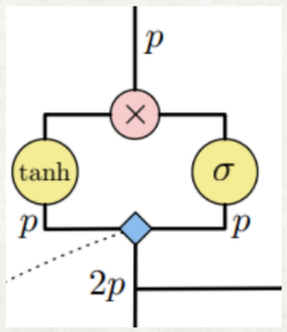


Otherwise, Wavenet is just PixelCNN

Replace ReLU with this gated activation function

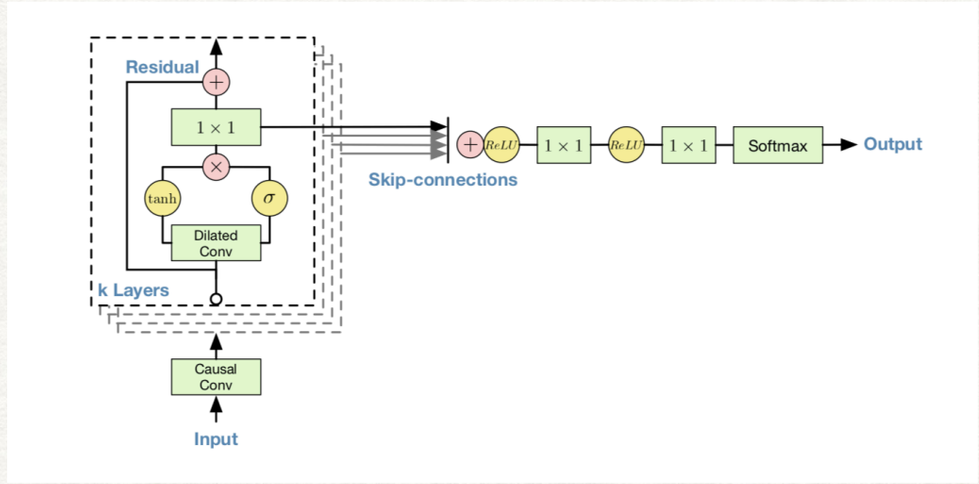
$$y = \tanh(W_{k,f} * x) \odot \sigma(W_{k,g} * x)$$

* is conv operation



- Split the feature maps in half and pass them through the tanh and sigmoid functions

- Compute element-wise product



Gated activation

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

Skip and Residual Connections

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

Conditioning to generate specific types of samples
eg: British/American accents

Thank you!