Deep Generative Models

Chapter 3: Generation by Explicit Distribution Learning

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Summer 2025

Overview and LMs

We have already worked a lot with AR models: *in Chapter 1*

we built various LMs $\rightsquigarrow$ *they are AR models*

People have tried AR modeling for other modalities as well

- *It has been largely developed for visual modalities like images*
- *Various challenges like causality are faced there too*
  - ↳ *Concepts like masked decoding or positional encoding were developed there*
  - ↳ *Modern LMs indeed borrow lots of those notions*

In this section, we go through some major computational AR models

RNN-based AR Models

Recurrent AR models are one of primary computational AR models: *we have already seen the recurrent LMs*

In these models, we use RNN to build conditional distribution

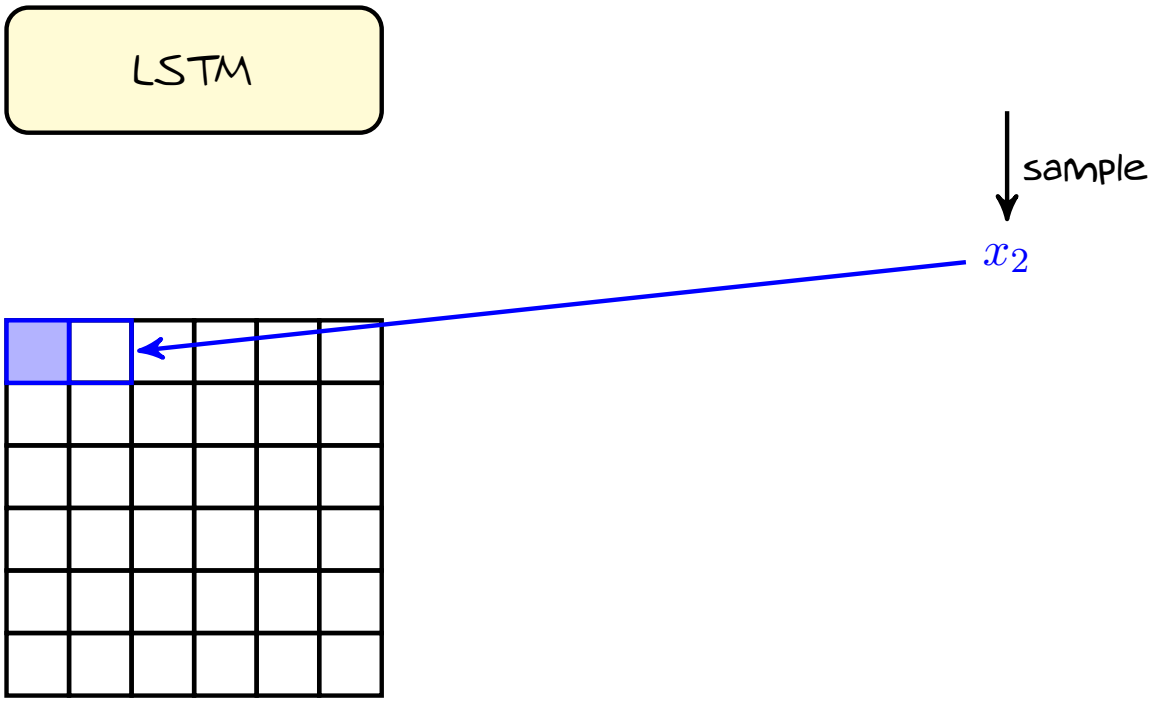
1. *The model samples* x_1 *from a learned prior*
2. *Given* x_i *and the state of RNN at* $i-1$*, the model*
   - ↳ *updates its state to represent the context at* i
   - ↳ *computes Softmax-activated output of context to estimate* $P\left(x_{i+1} \middle| x_{<i+1}\right)$
3. *The model iterates till* d *entries are over*

Key Observation

We cannot do parallel processing $\rightsquigarrow$ *training is also slow*

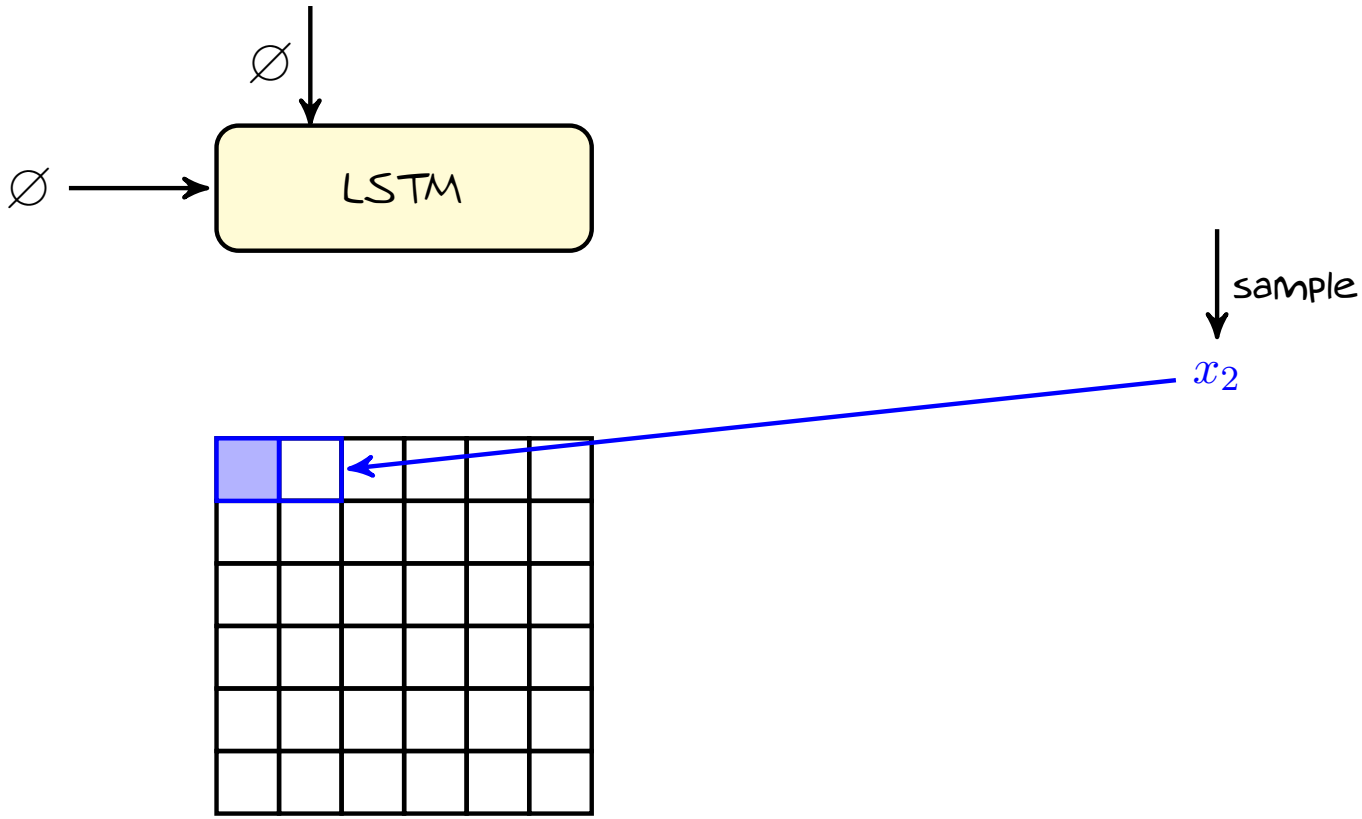
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



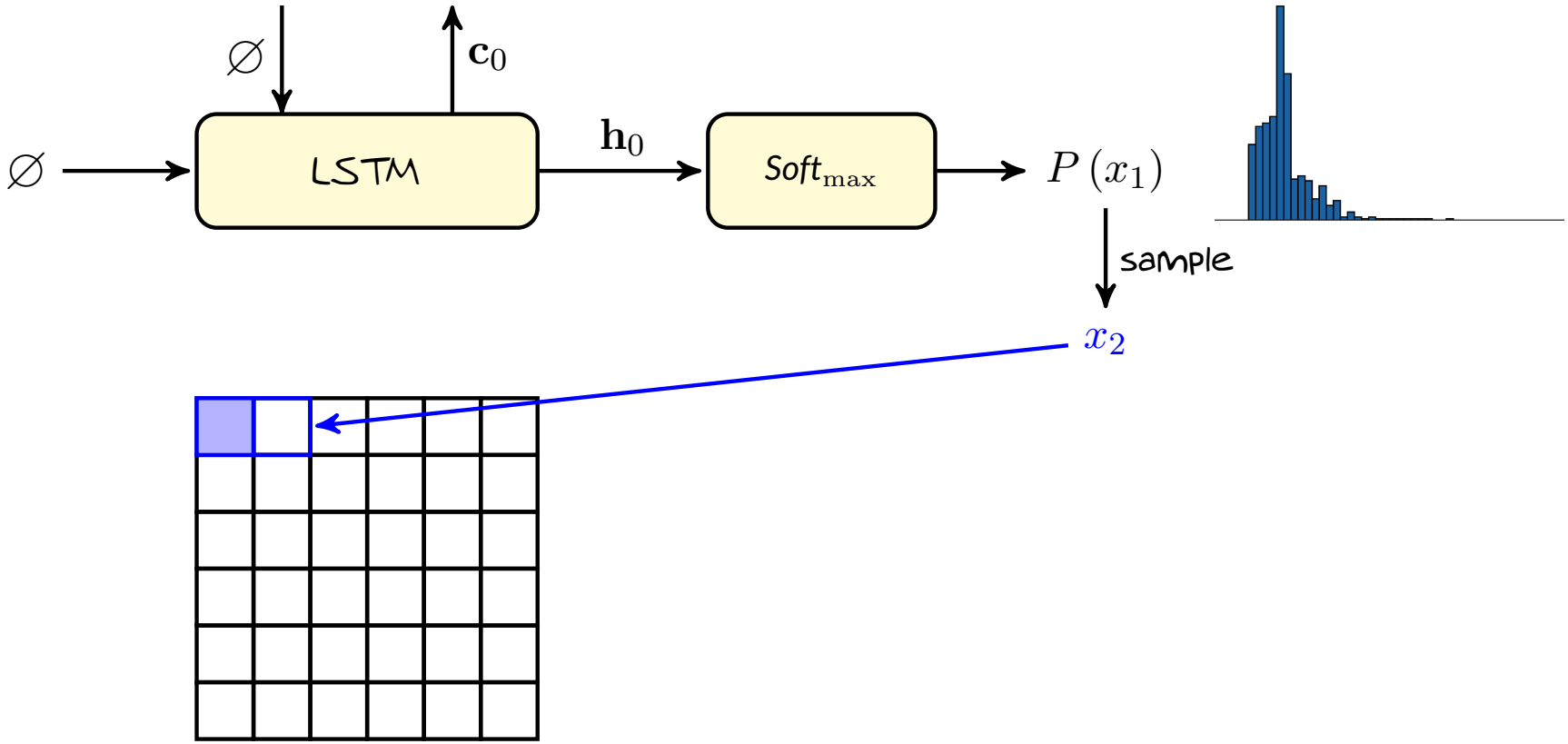
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



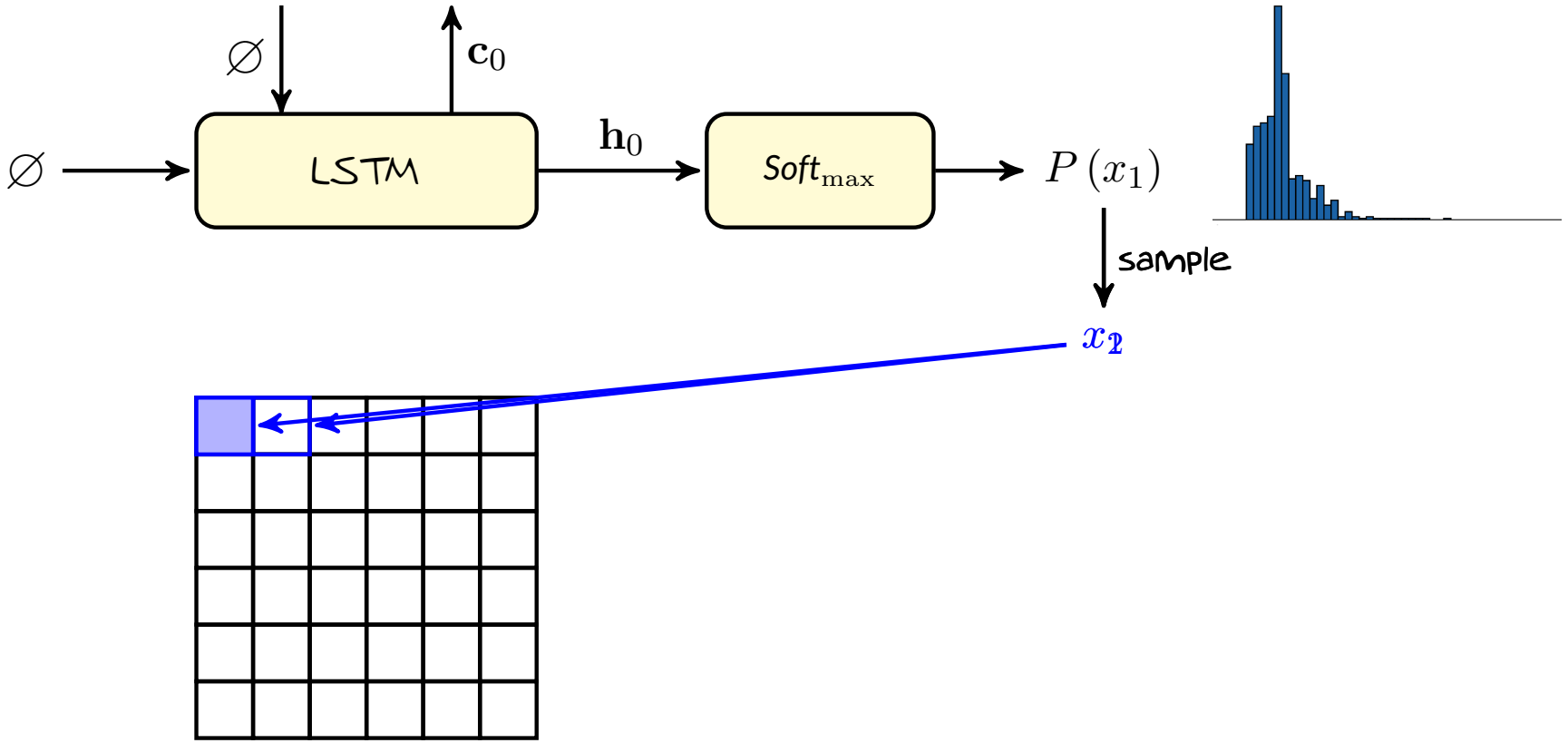
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



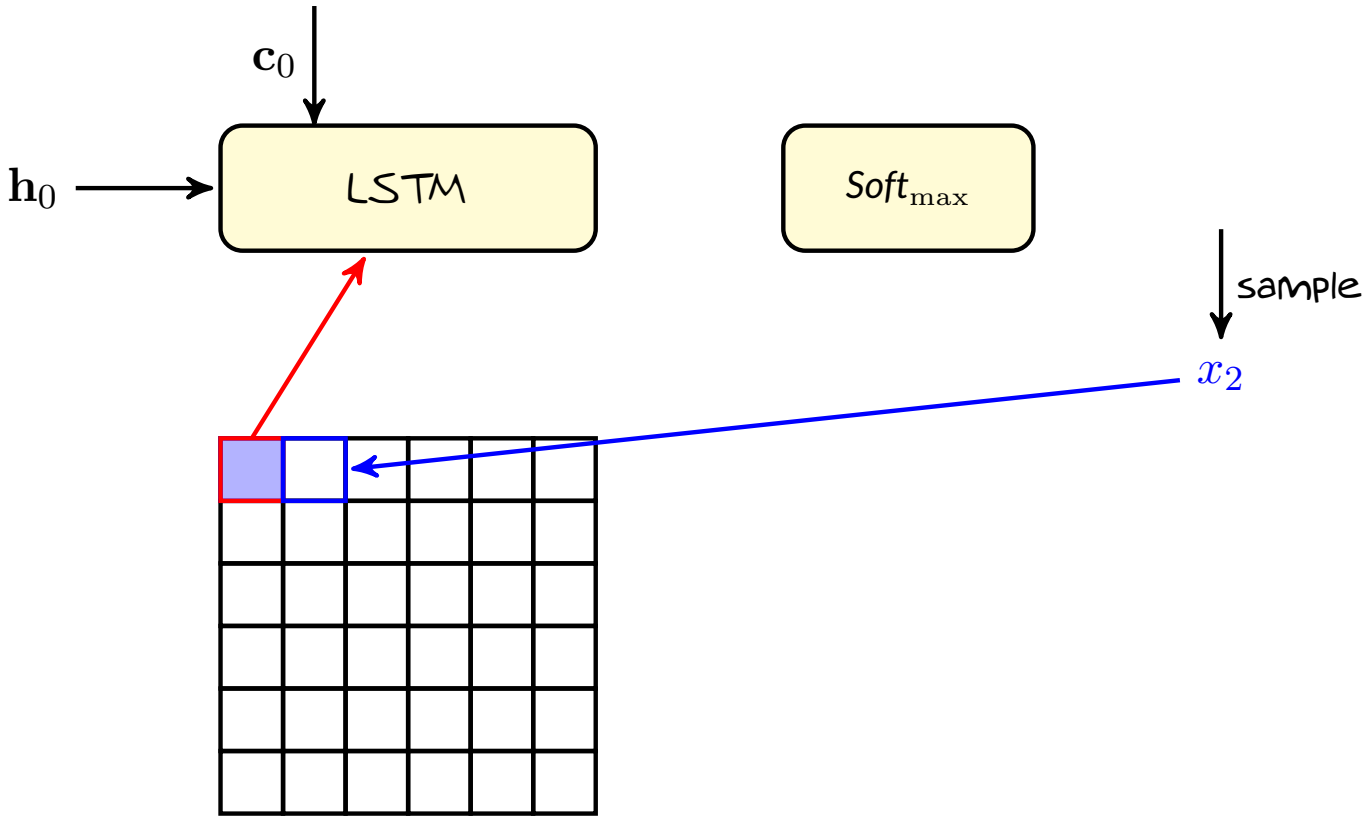
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



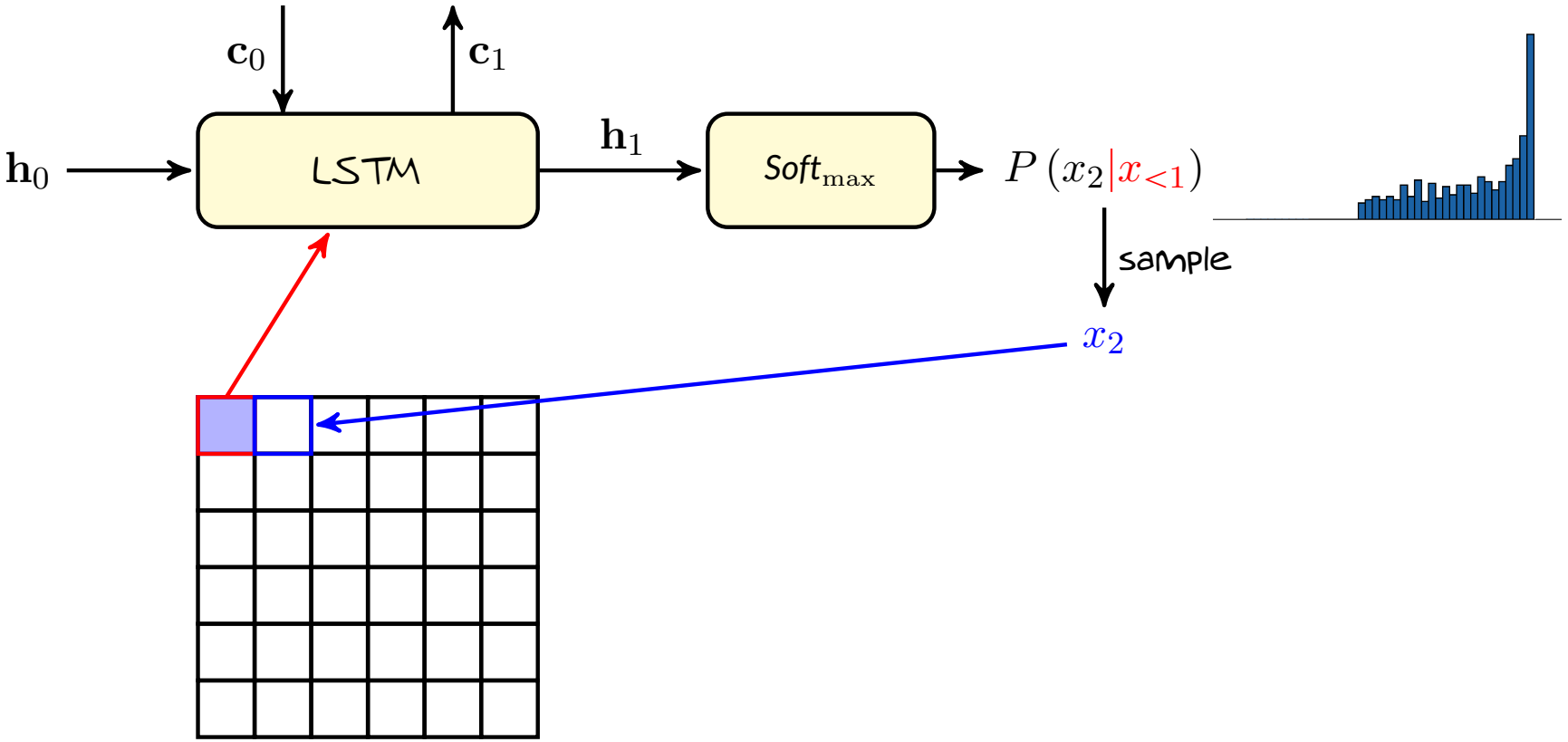
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



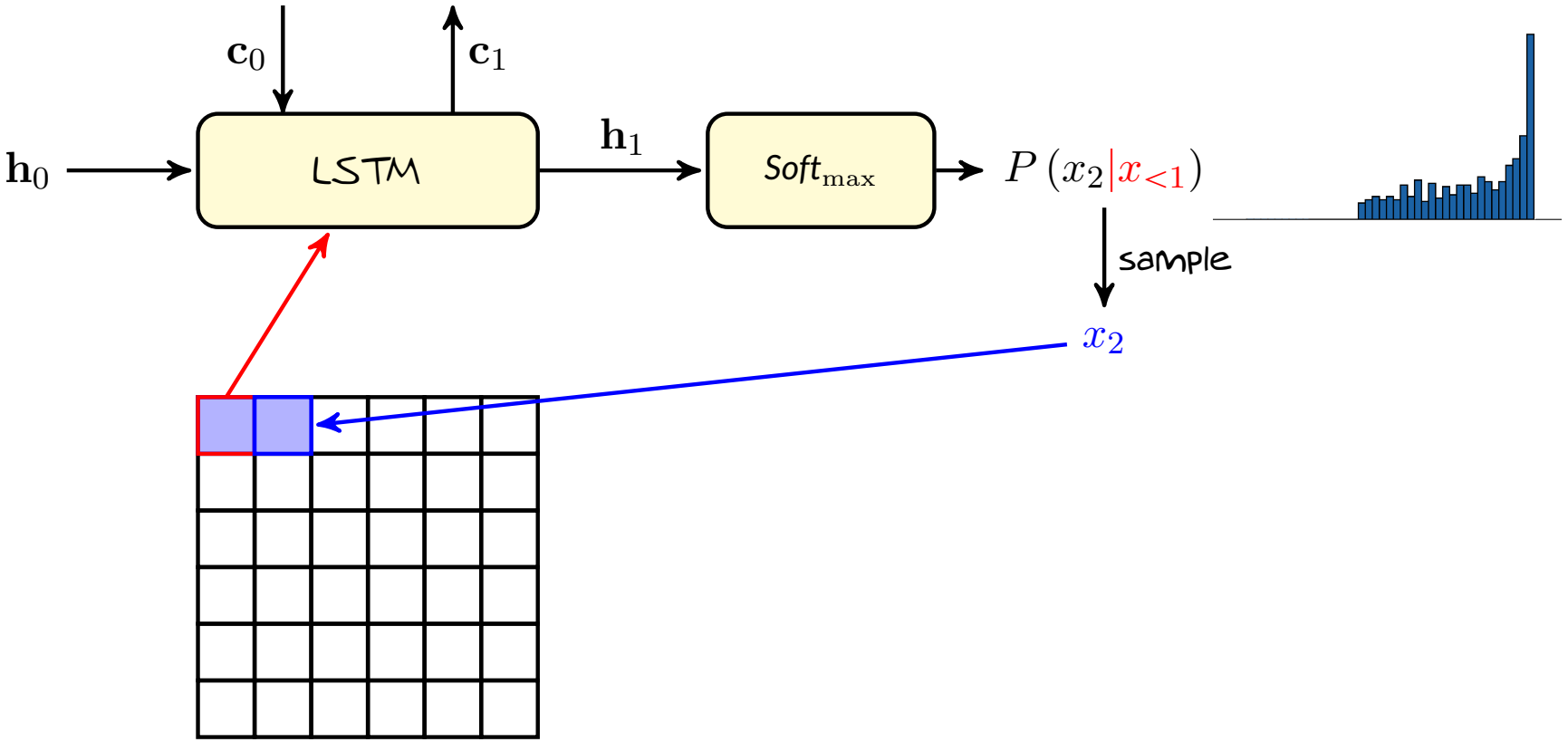
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



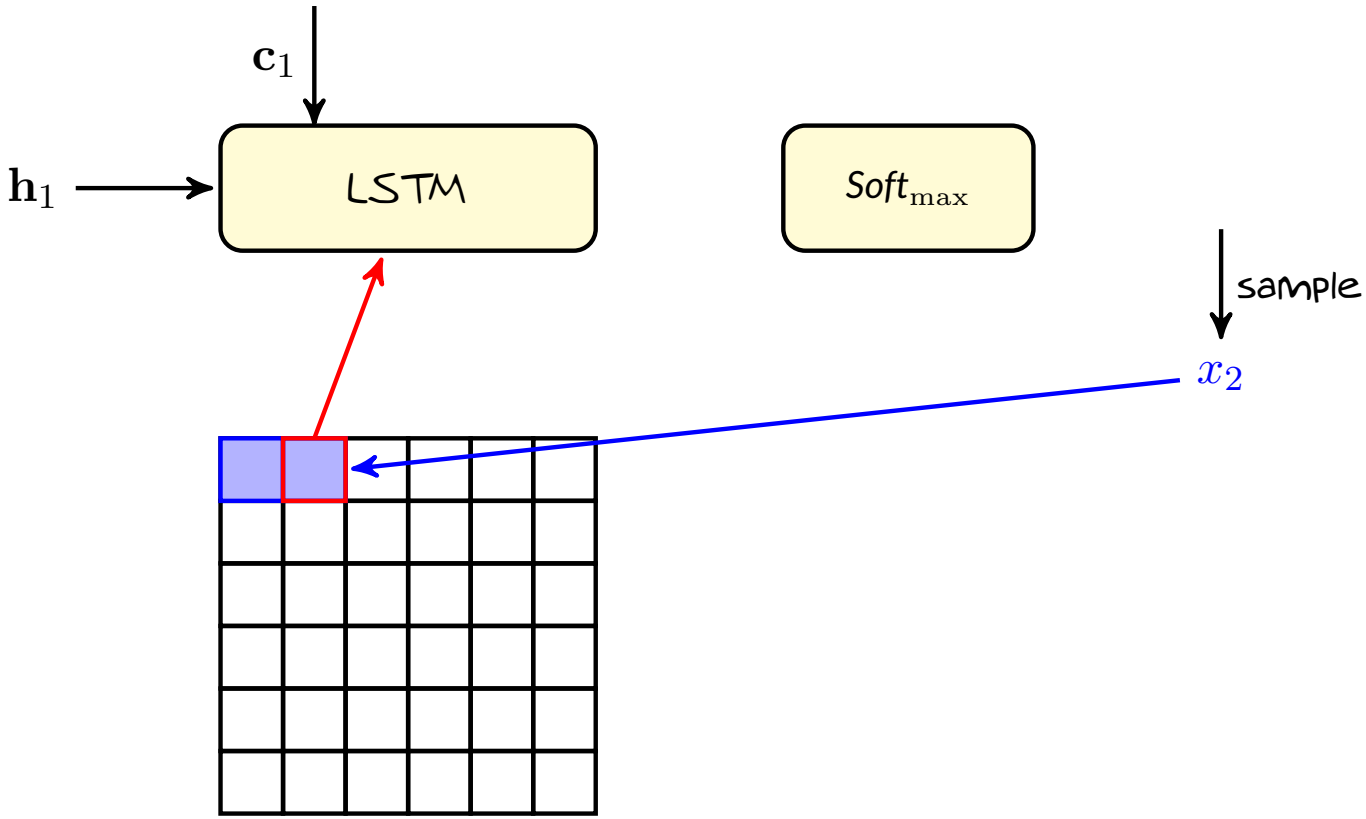
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



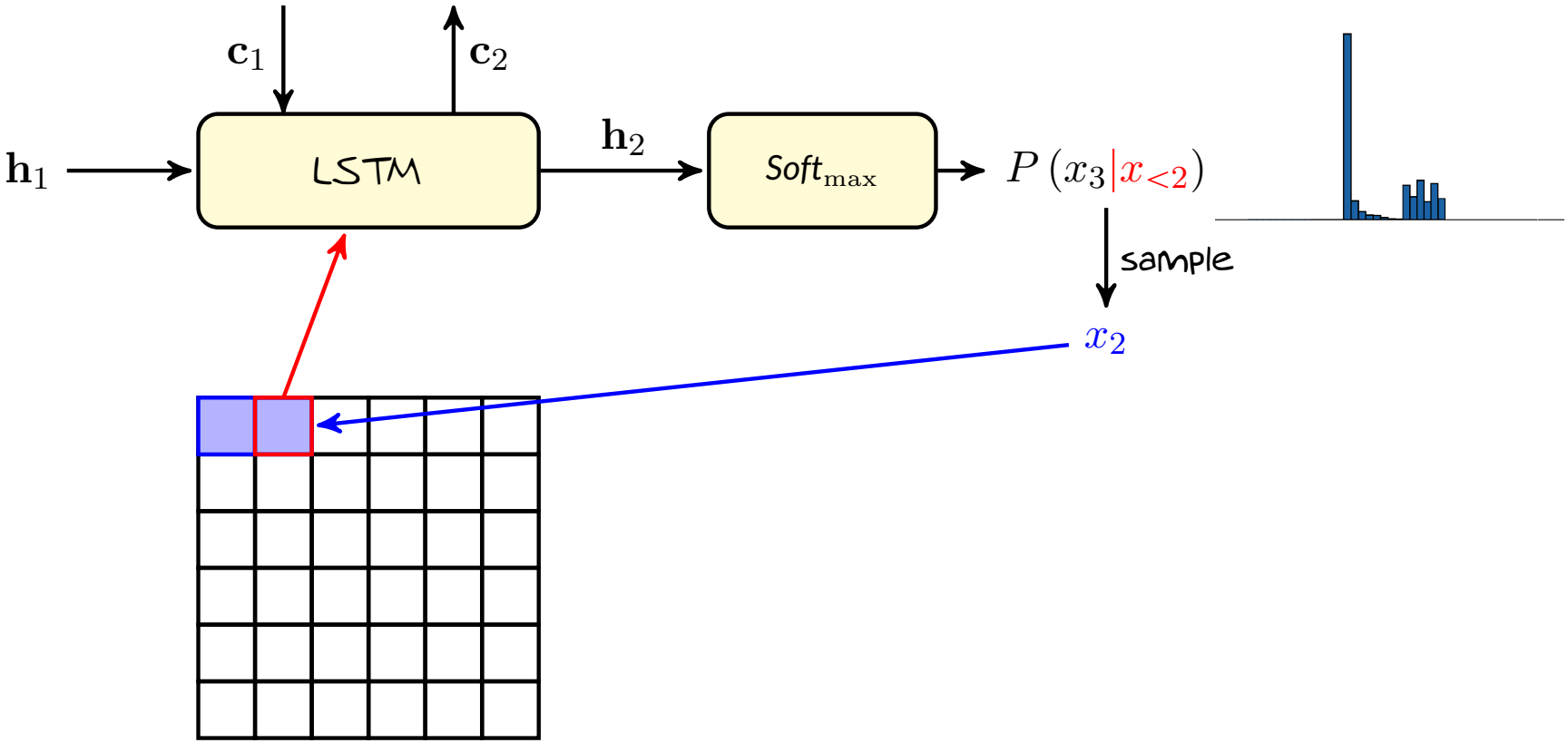
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



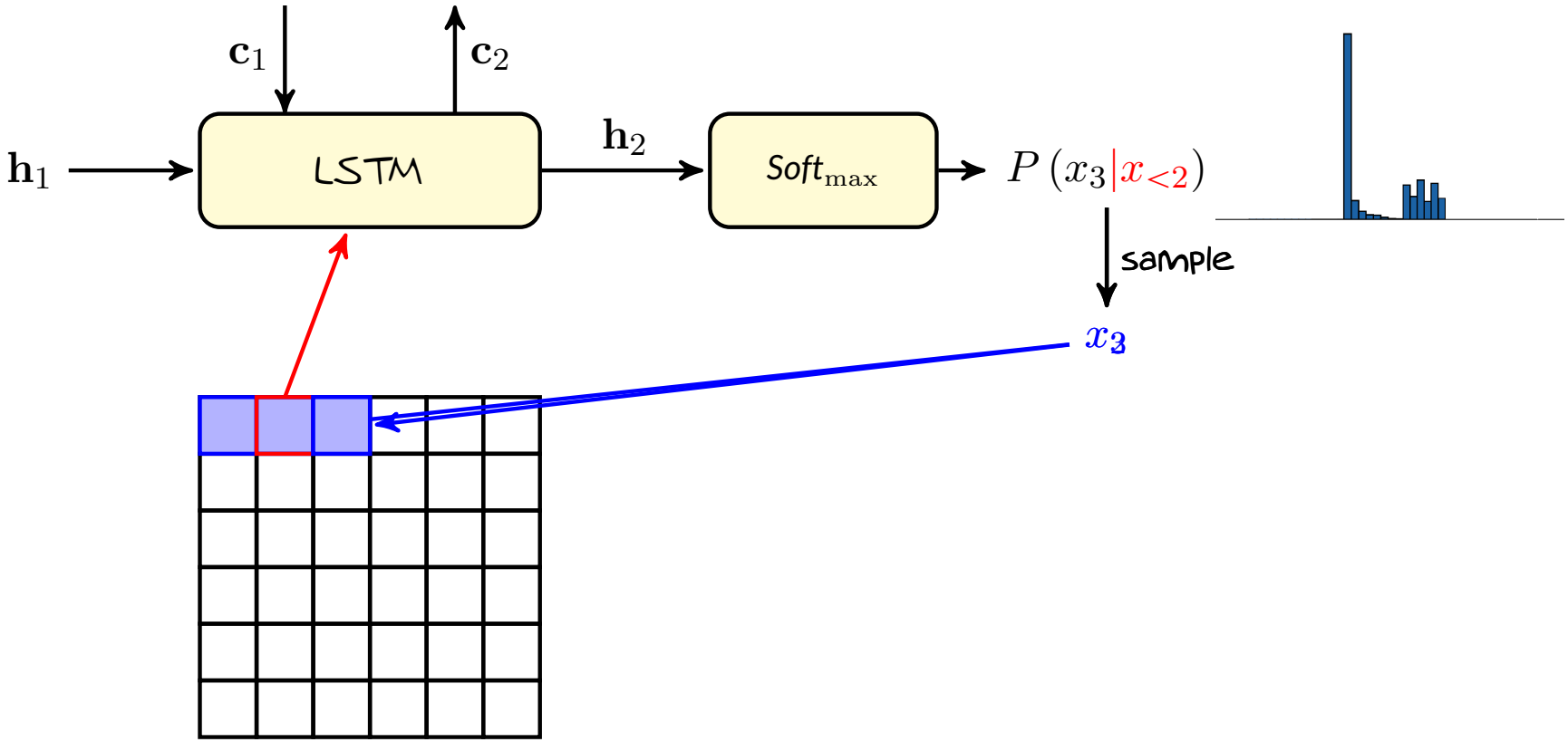
PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



PixelRNN: *Visual Recurrent AR Model*

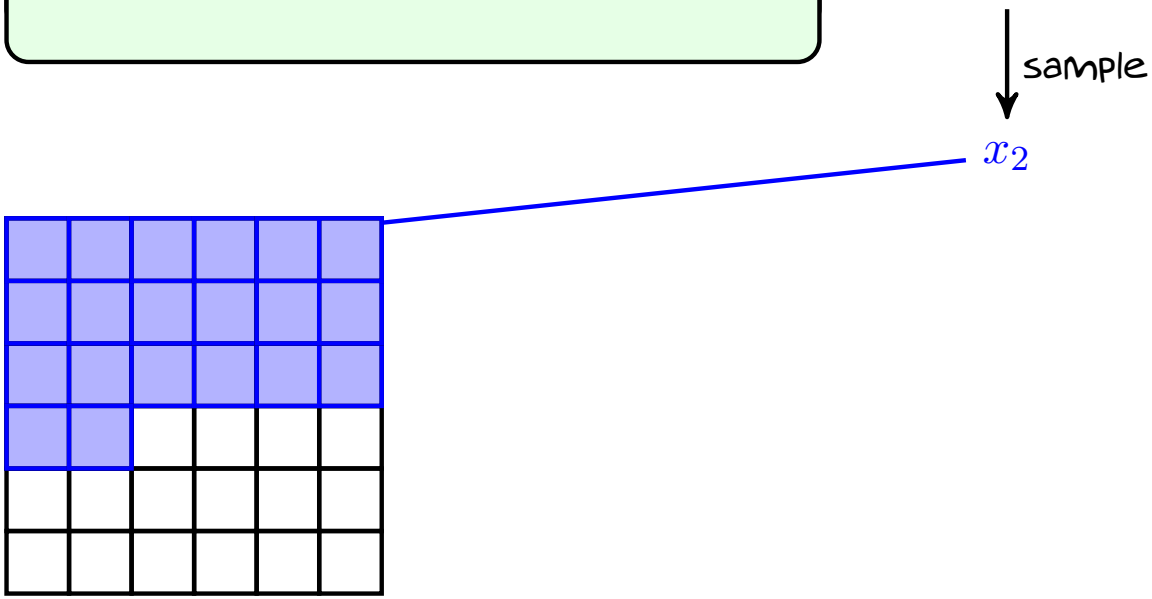
Proposed in 2016, PixelRNN uses LSTM to extract context



PixelRNN: *Visual Recurrent AR Model*

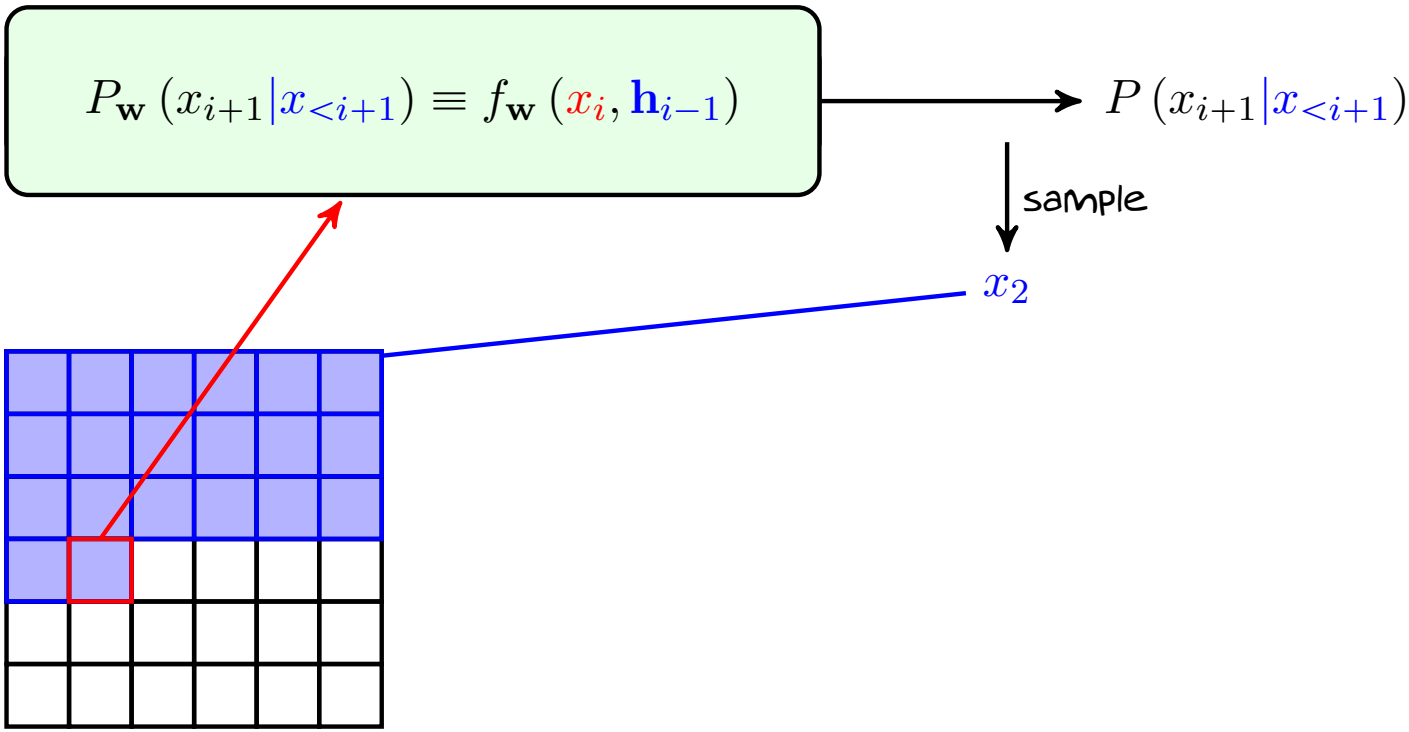
Proposed in 2016, PixelRNN uses LSTM to extract context

$$P_{\mathbf{w}}\left(x_{i+1}|x_{<i+1}\right) \equiv f_{\mathbf{w}}\left(x_i, \mathbf{h}_{i-1}\right)$$



PixelRNN: *Visual Recurrent AR Model*

Proposed in 2016, PixelRNN uses LSTM to extract context



PixelRNN: *Training*

We can readily train this model via MLE

`Train_PixelRNN(`$\mathbb{D}$`:dataset):`

for *multiple epochs* **do**
  Sample a batch of n *images* $\{x^j : j = 1, \ldots, n\}$
  for $j = 1 : n$ **do**
    $\mathbf{p}_1^j, \mathbf{c}, \mathbf{h} \leftarrow$ `LSTM(`$\varnothing$`)` `#define` $\varnothing$ `to be start pixel`
    for $i = 1, \ldots, d-1$ **do**
      $\mathbf{p}_{i+1}^j, \mathbf{c}, \mathbf{h} \leftarrow$ `LSTM(`$x_i^j, \mathbf{c}, \mathbf{h}$`)`
    end for
  end for
  Update model using `Opt_avg` $\left\{ \sum_i \nabla \log \mathbf{p}_i^j \left[x_i^j \right] \right\}$
end for
return *trained model*

PixelRNN: *Generation*

It's good to think how we can generate with this model

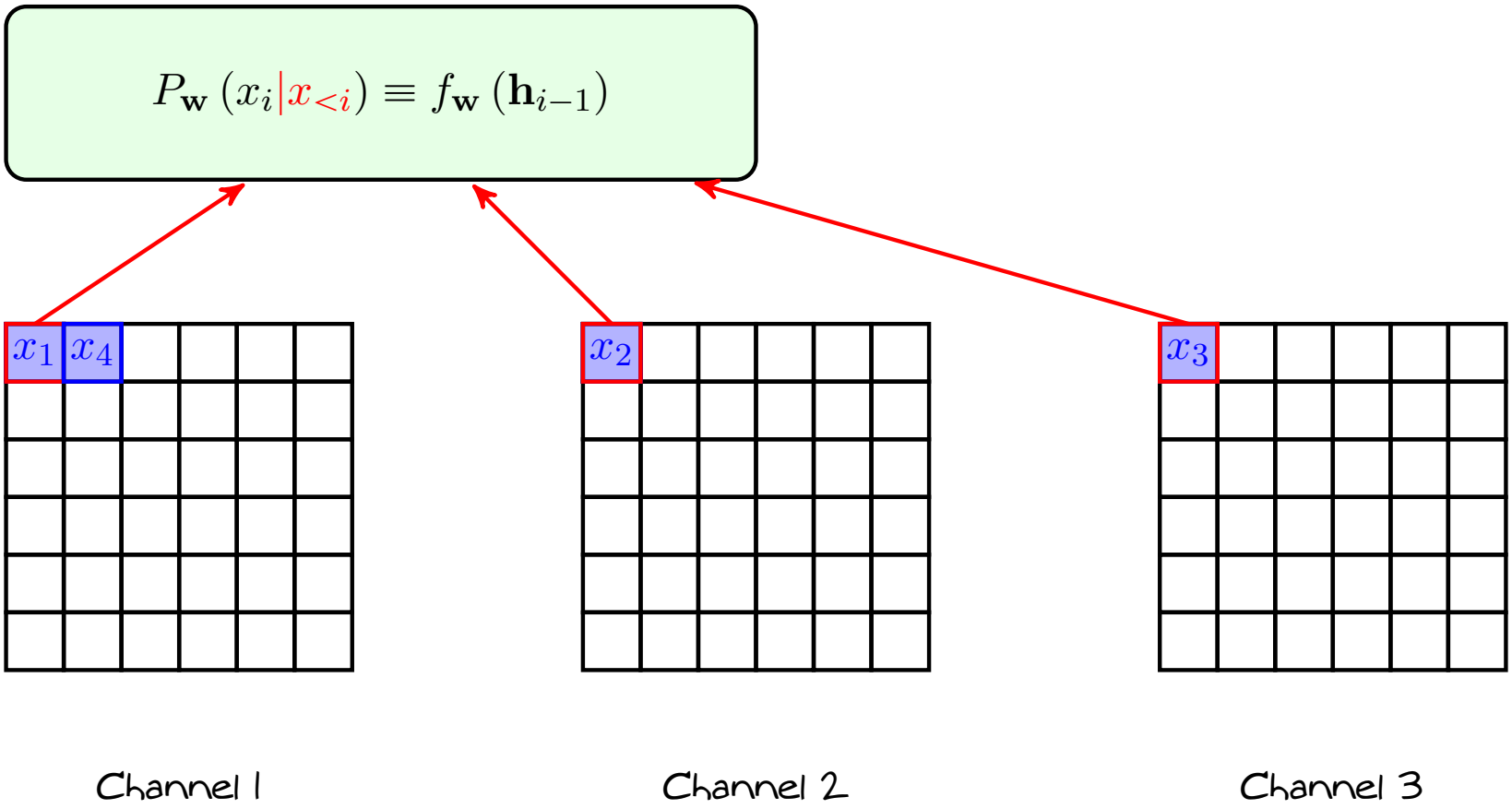
`Sample_PixelRNN( ):`

1: $\mathbf{p}_1, \mathbf{c}, \mathbf{h} \leftarrow \texttt{LSTM}(\varnothing)$
2: $x_1 \leftarrow \texttt{multinomial}(\mathbf{p}_1, \texttt{ \#smpl=1})$ `#sample first pixel`
3: **for** $i = 1, \ldots, d-1$ **do**
4: $\quad \mathbf{p}_{i+1}, \mathbf{c}, \mathbf{h} \leftarrow \texttt{LSTM}(x_i, \mathbf{c}, \mathbf{h})$
5: $\quad x_{i+1} \leftarrow \texttt{multinomial}(\mathbf{p}_{i+1}, \texttt{ \#smpl=1})$ `#AR generation`
6: **end for**
7: **return** $x = \texttt{DataType}\{x_1, \ldots, x_d\}$

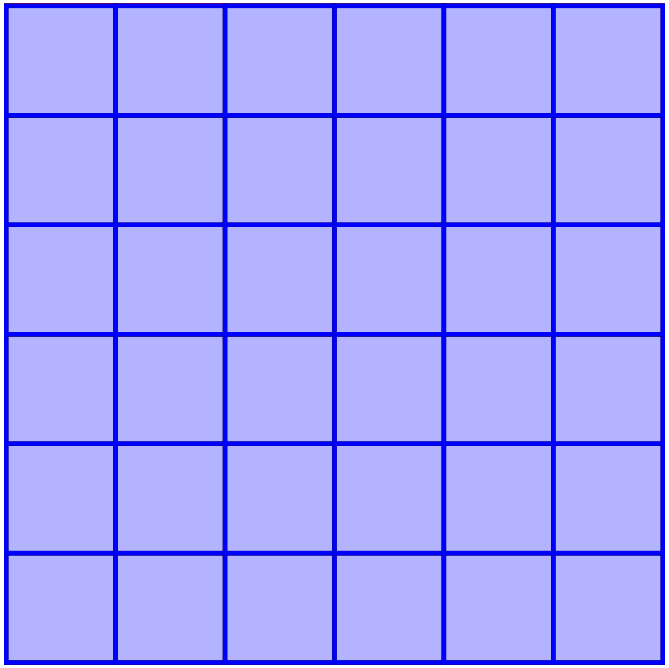
PixelRNN: *Multi-channel*

For multi-channel data, we typically first process in depth

↳ *We expect that RGB pixels are heavily correlated*



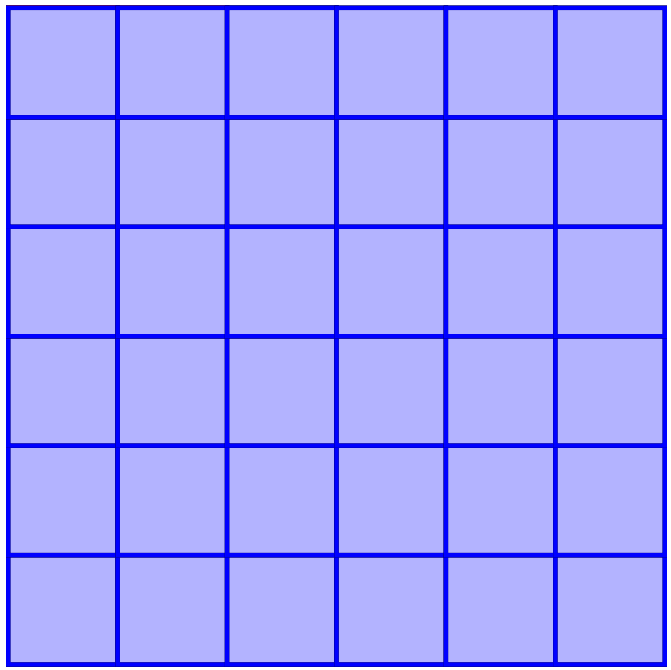
PixelRNN: *Row Processing*



Basic PixelRNN goes through images row-wise $\rightsquigarrow$ *processing time* $\propto d^2$

- *For generation we need to complete these* d^2 *steps*
- *For training we can treat each row independently* $\approx$ *truncated BPTT*
  - ↳ *Faster at the price of capturing less spatial correlation*

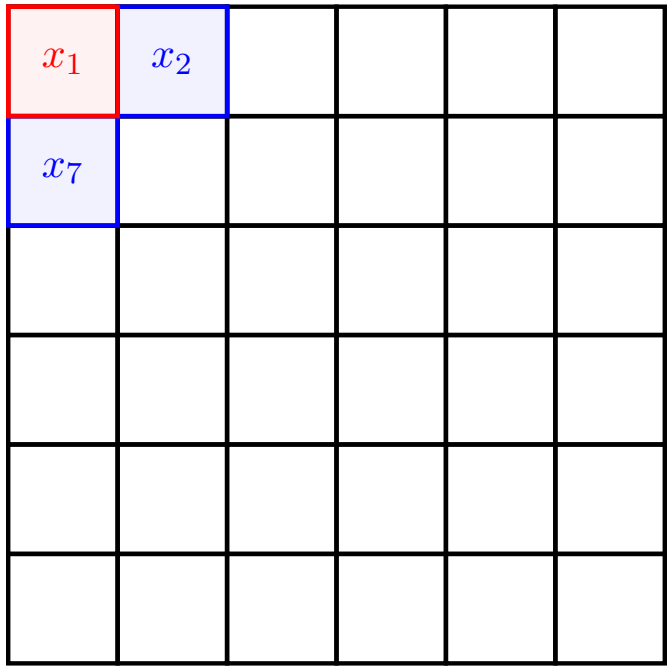
PixelRNN: *Diagonal Processing*



Alternatively, we can process each pixel conditioned on its top and left pixels

- *We can process multiple pixels in parallel*
- *Total processing time for a single image is* $2d - 1$ *or* $\propto d$
  - ↳ *Fast with less loss in capturing spatial correlation*

PixelRNN: *Diagonal Processing - Reduced AR Modeling*



Attention

Note that we get the benefit of parallel processing due to reduced AR modeling

$$P(x_2|x_1)\,P(x_7|x_1,x_2) \approx P(x_7|x_1)\,P(x_7|x_1)$$

Convolutional AR Models

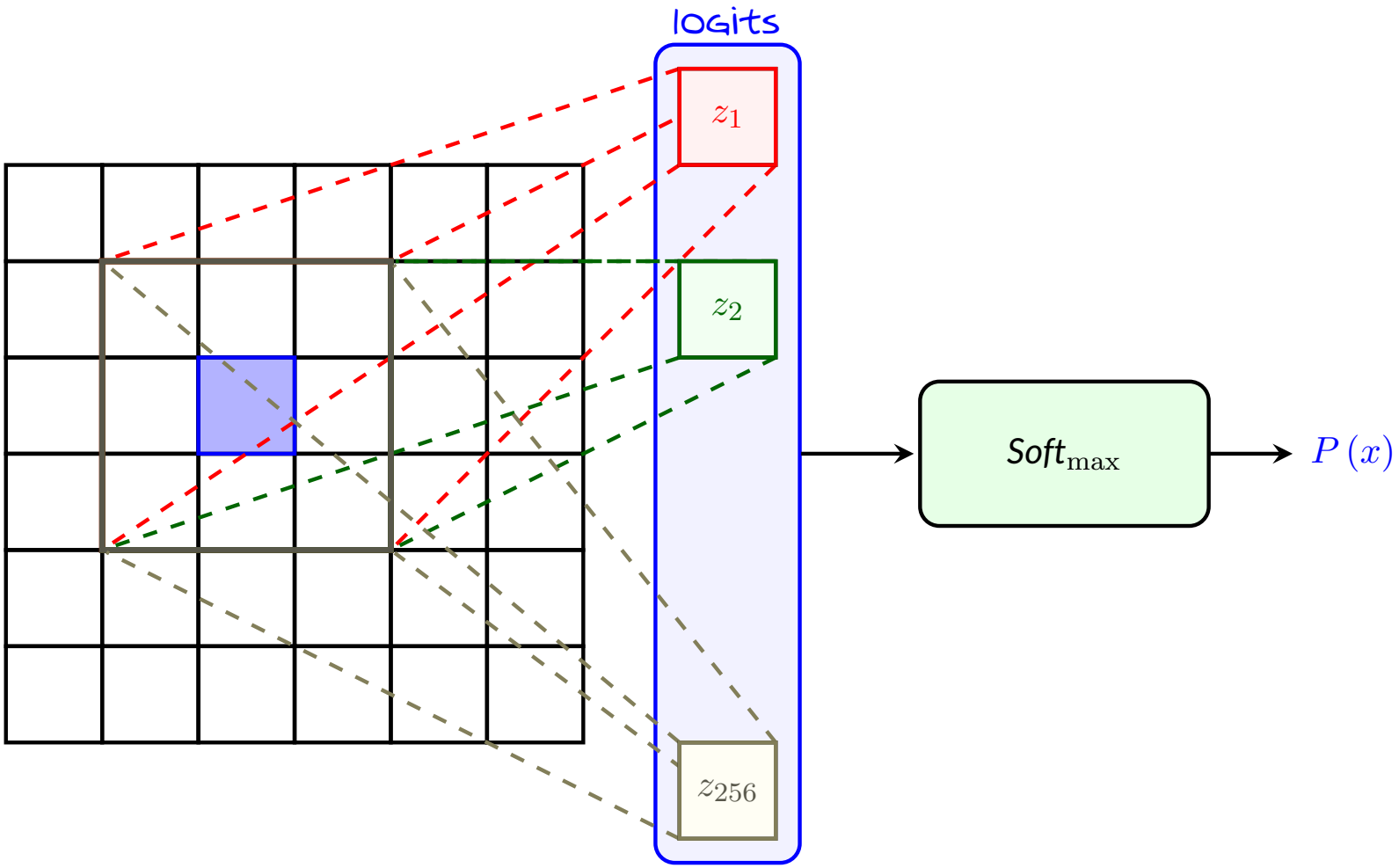
When we work with visual data: *it sounds natural that we work with CNNs*

+ *But, CNNs cannot capture context? We need some sort of state!*
- But, we could extract context with Transformers
  - *They only consist of bunch of MLP-like layers*
  - *We made context without an explicit state*

This is in fact a well-developed idea!

Spatial Context via *Convolution*

We can compute a distribution for a single pixel using multiple kernels

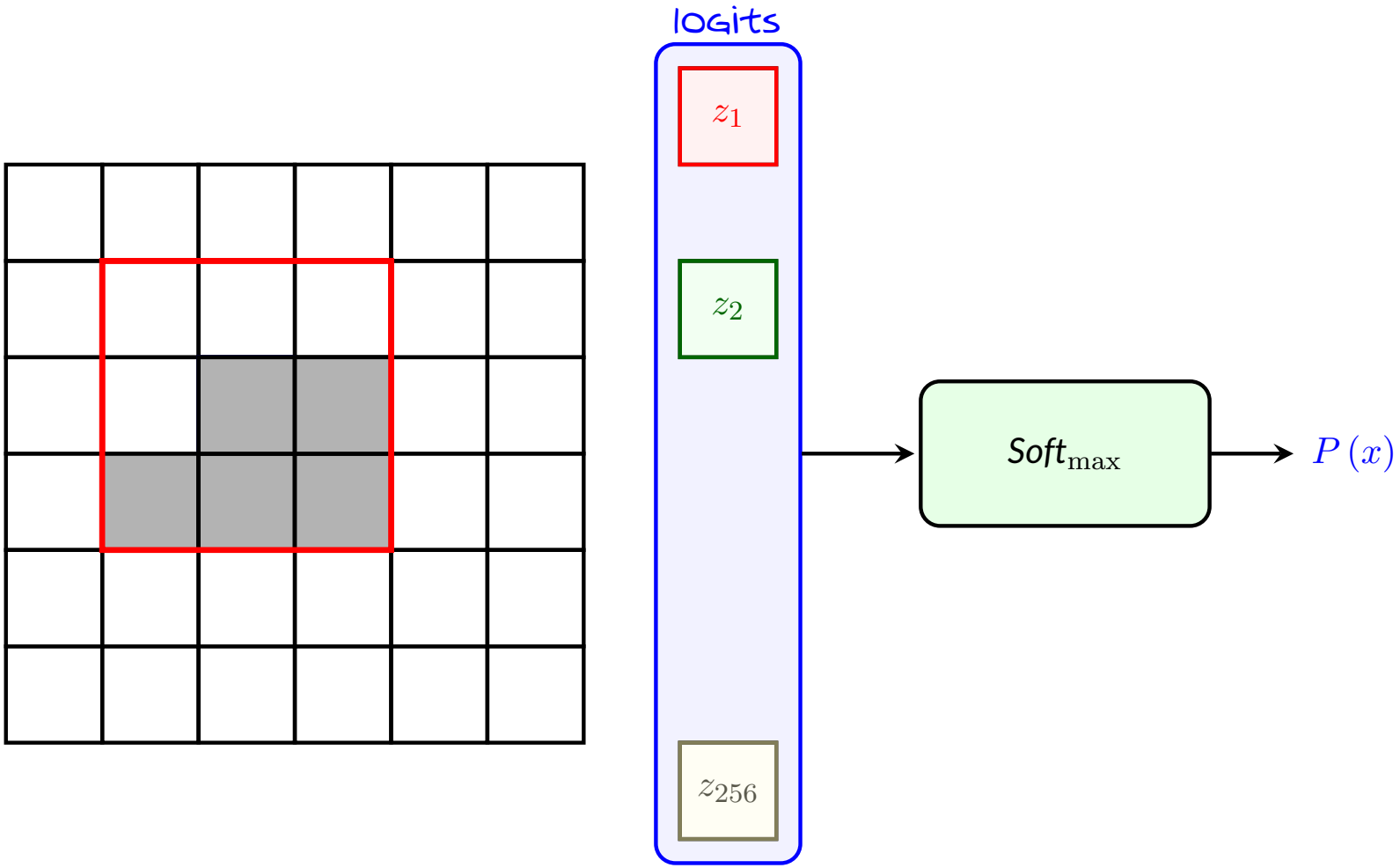


Valid Conditional Distribution

+ *Wait a moment! The conditional distribution however depends on all pixels in the receptive field! Including the pixel itself! Right?!*
- Yes! You are right!
+ *Then, this cannot be a valid conditional distribution!*
  - *We need to compute distribution conditioned to what we had before!*
  - *This distribution depends on the pixel itself and some future ones!*
- We can resolve this using *masked filter*
  - *We did the same in Transformers!*

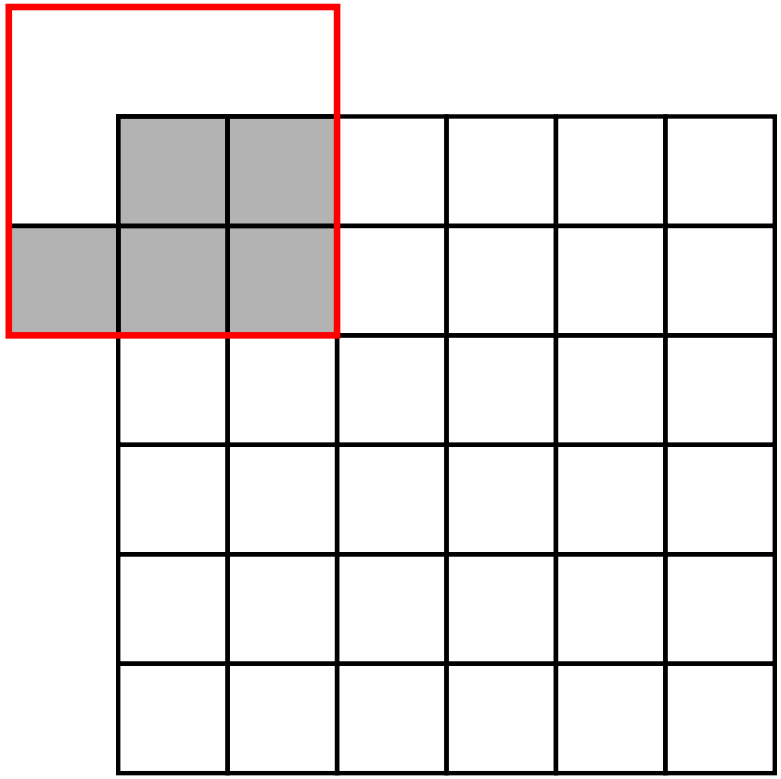
Masking Kernels

We mask kernels at the pixel position and the future positions



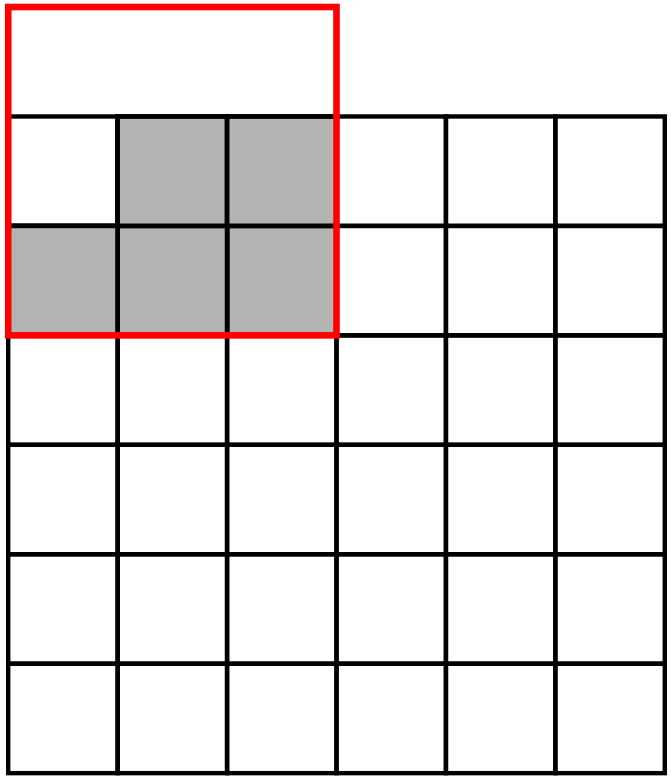
Masked Convolution

We can hence compute valid context via masked convolution



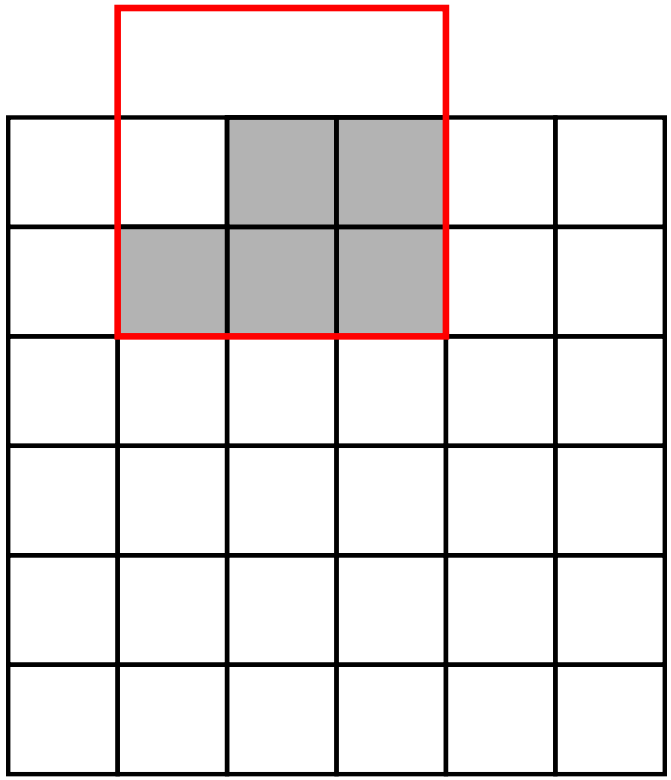
Masked Convolution

We can hence compute valid context via masked convolution



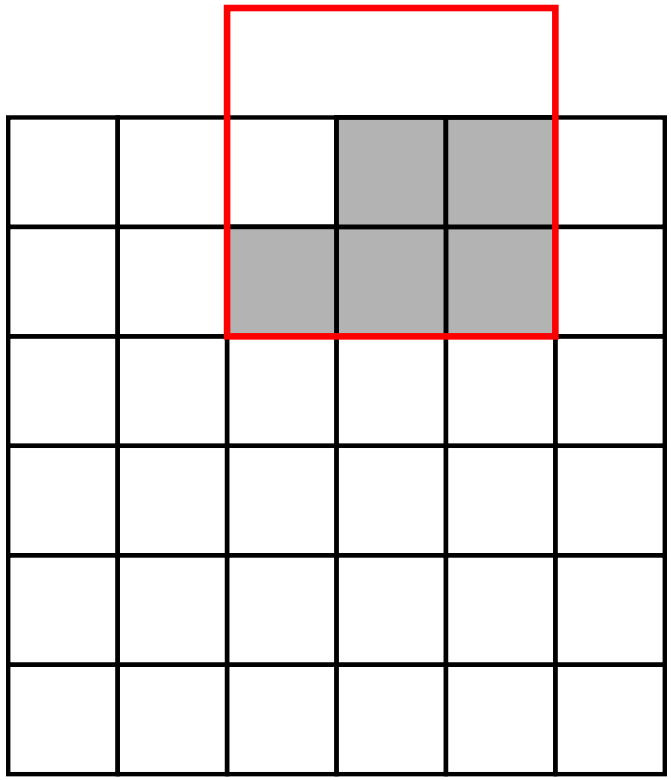
Masked Convolution

We can hence compute valid context via masked convolution



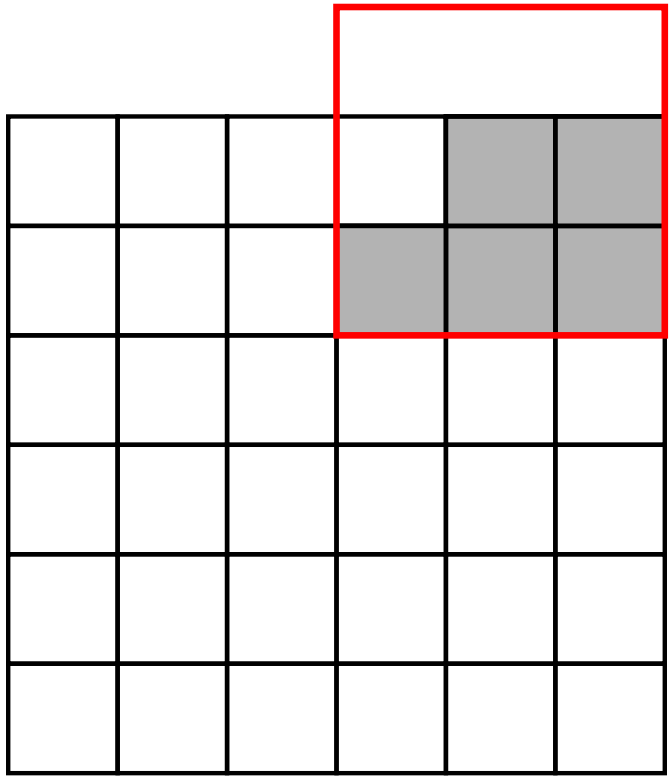
Masked Convolution

We can hence compute valid context via masked convolution



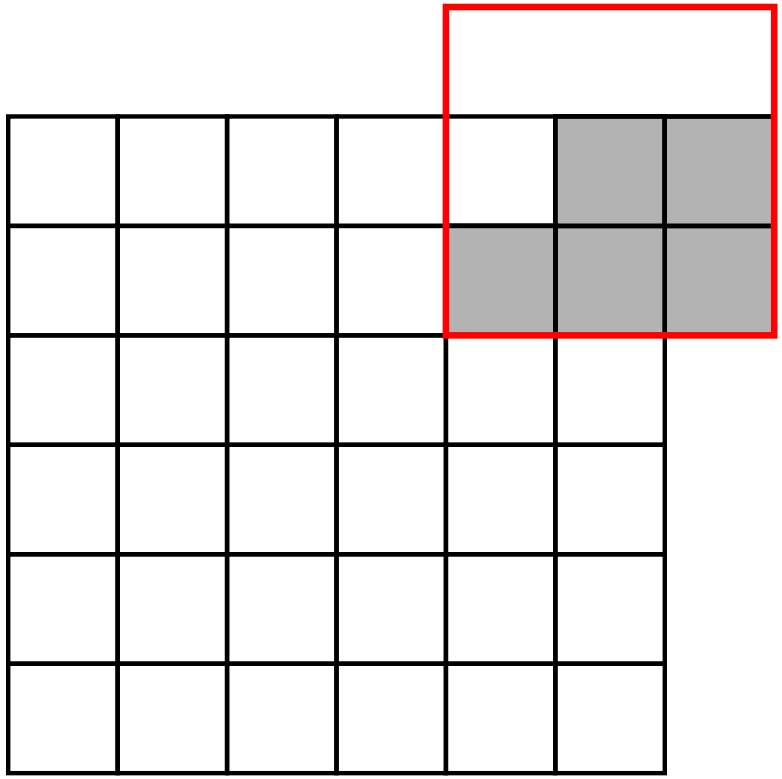
Masked Convolution

We can hence compute valid context via masked convolution



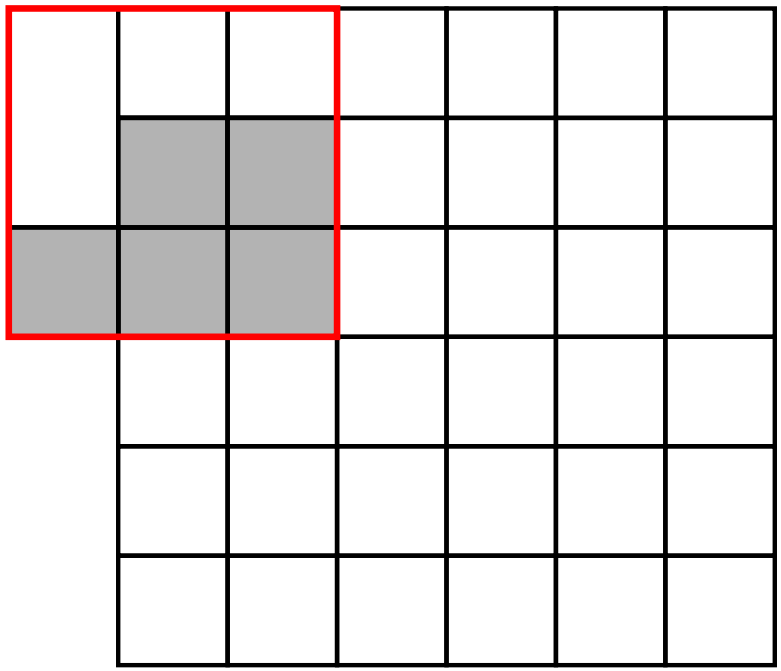
Masked Convolution

We can hence compute valid context via masked convolution



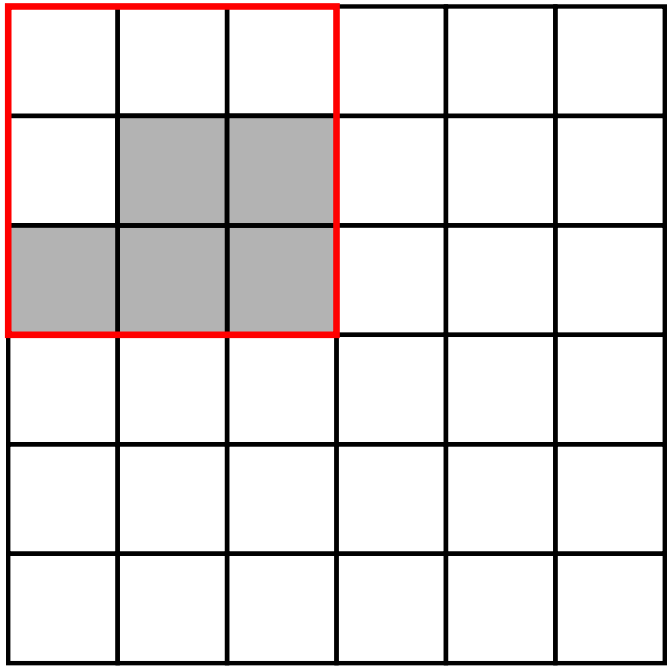
Masked Convolution

We can hence compute valid context via masked convolution



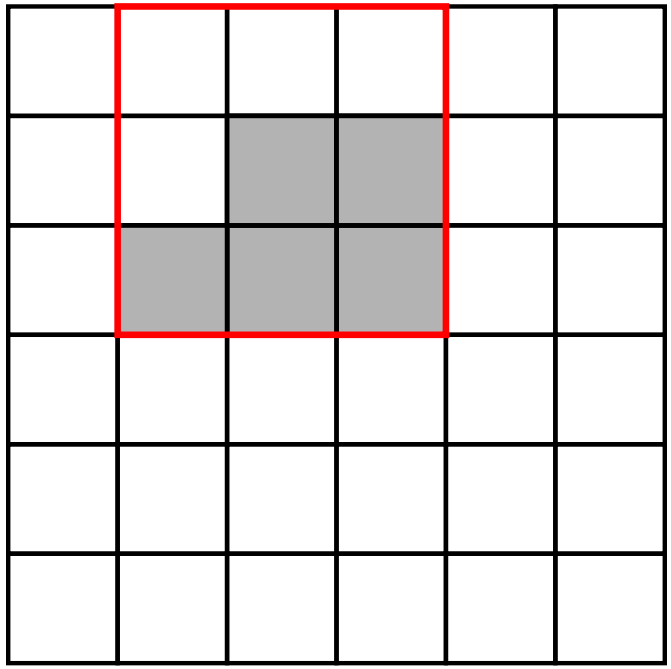
Masked Convolution

We can hence compute valid context via masked convolution



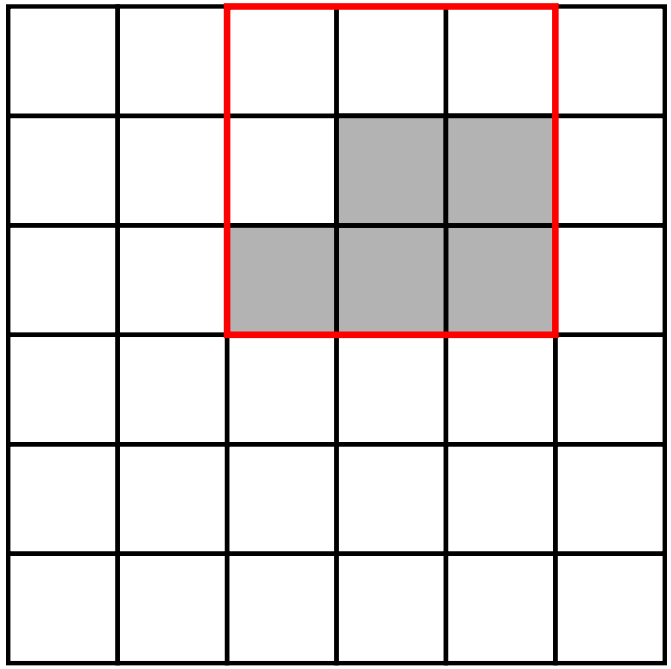
Masked Convolution

We can hence compute valid context via masked convolution

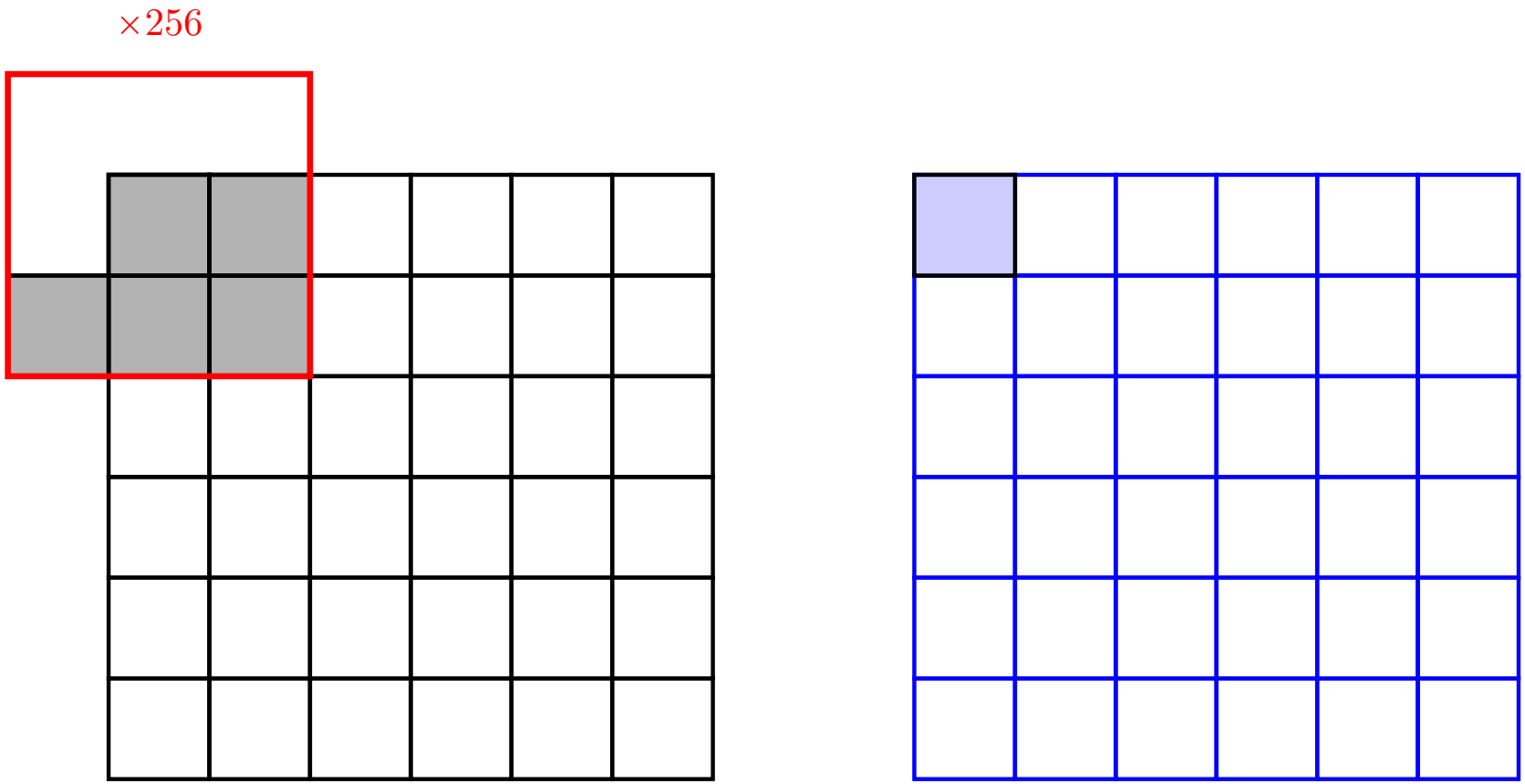


Masked Convolution

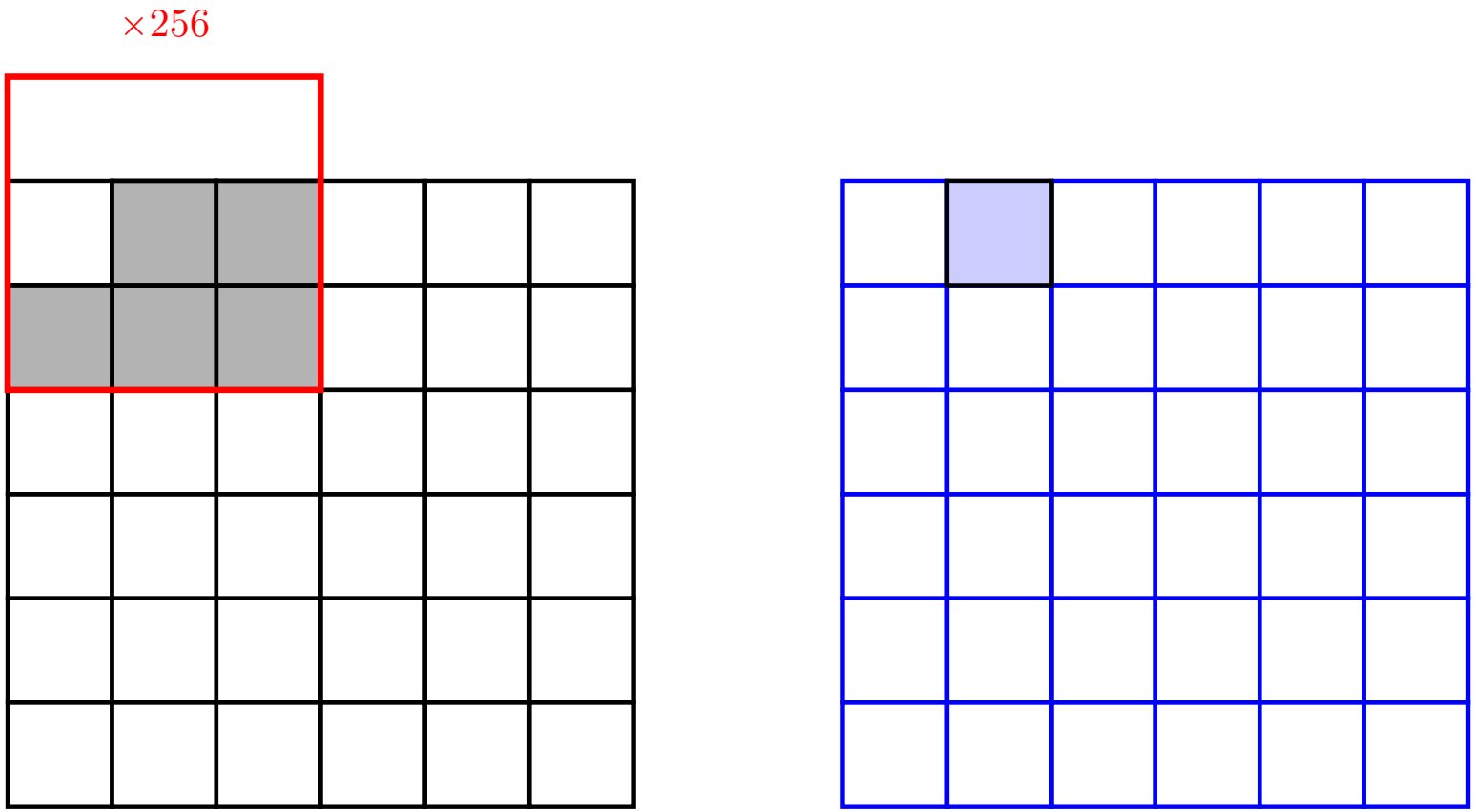
We can hence compute valid context via masked convolution



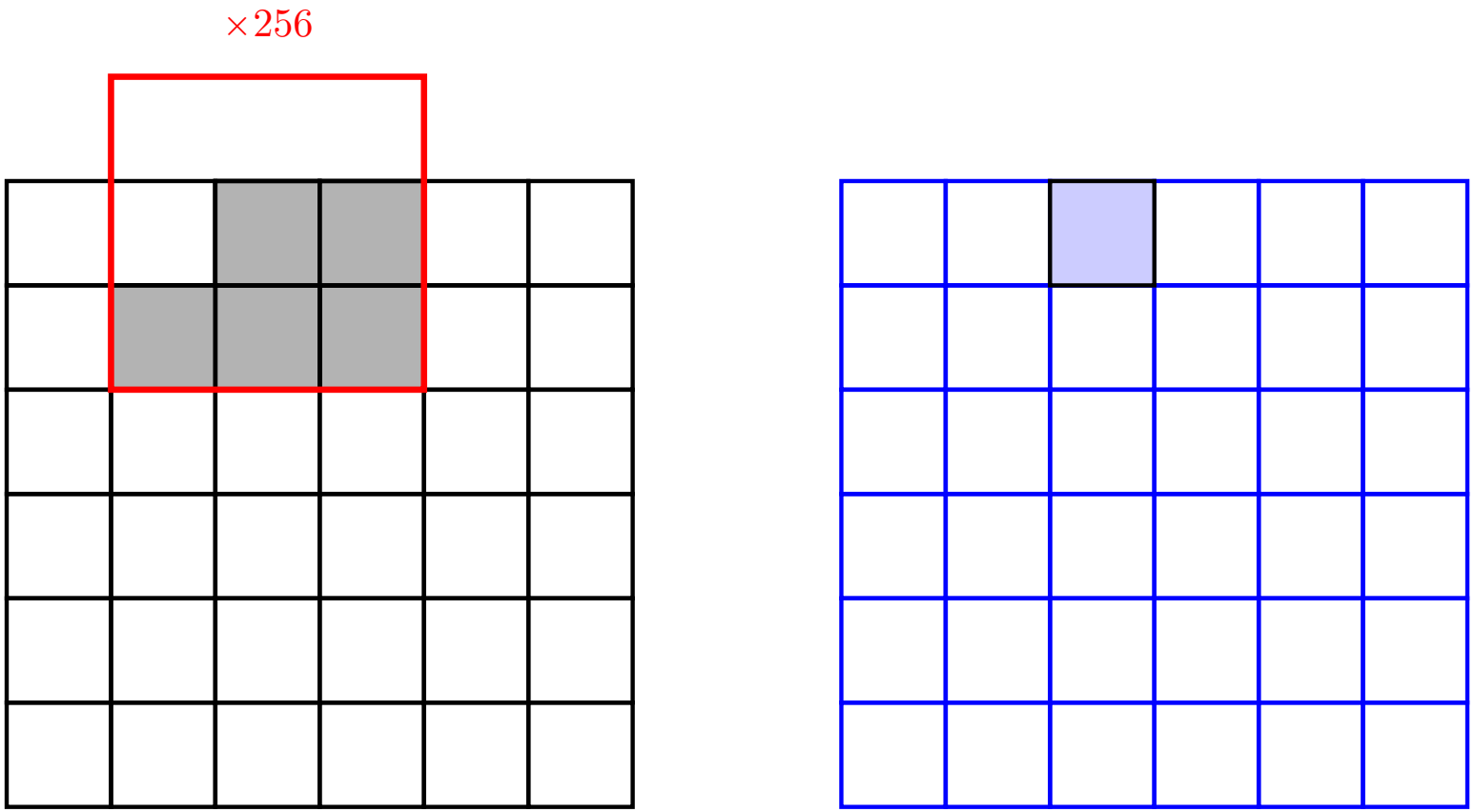
Masked Convolution: *Valid Conditional Distribution*



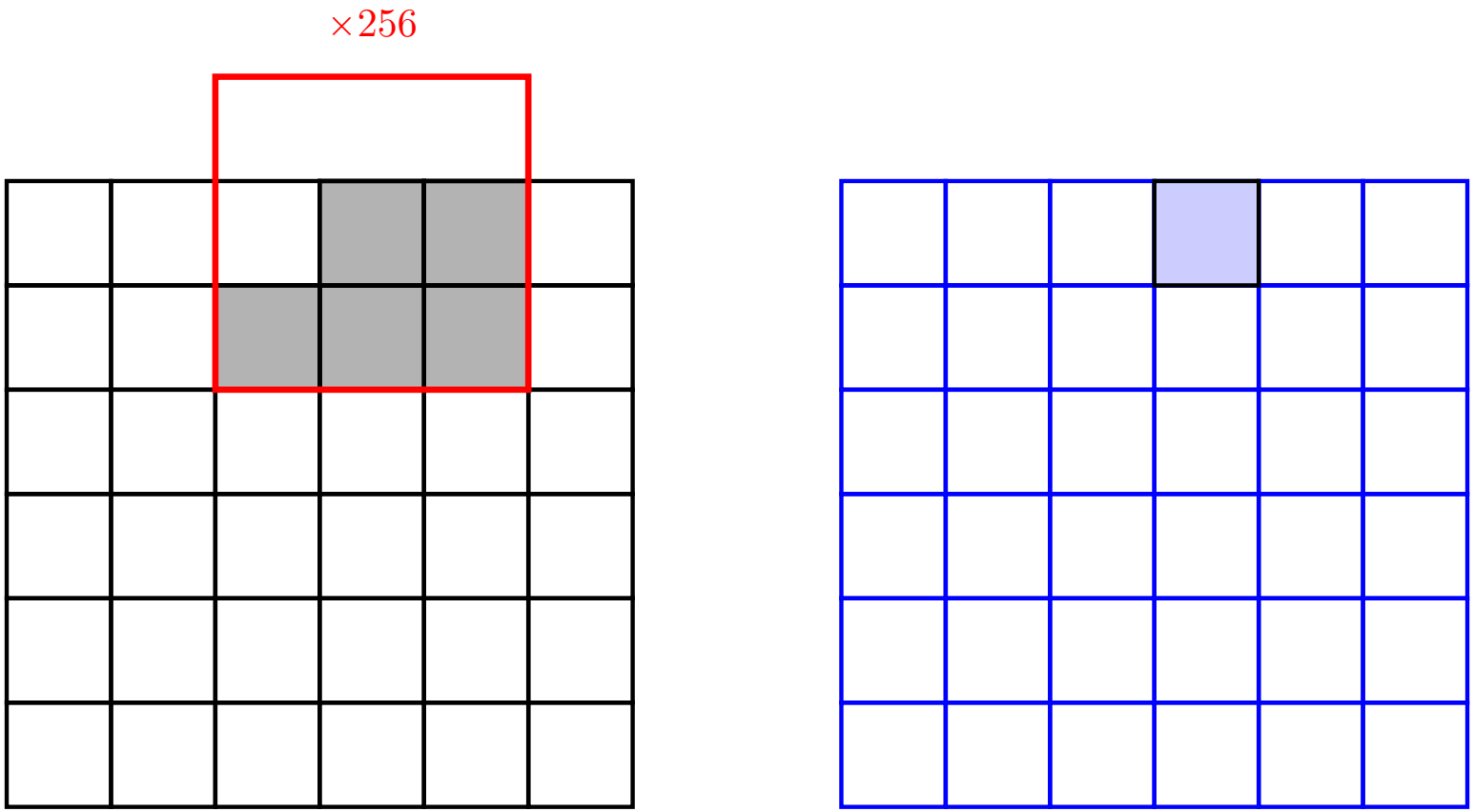
Masked Convolution: *Valid Conditional Distribution*



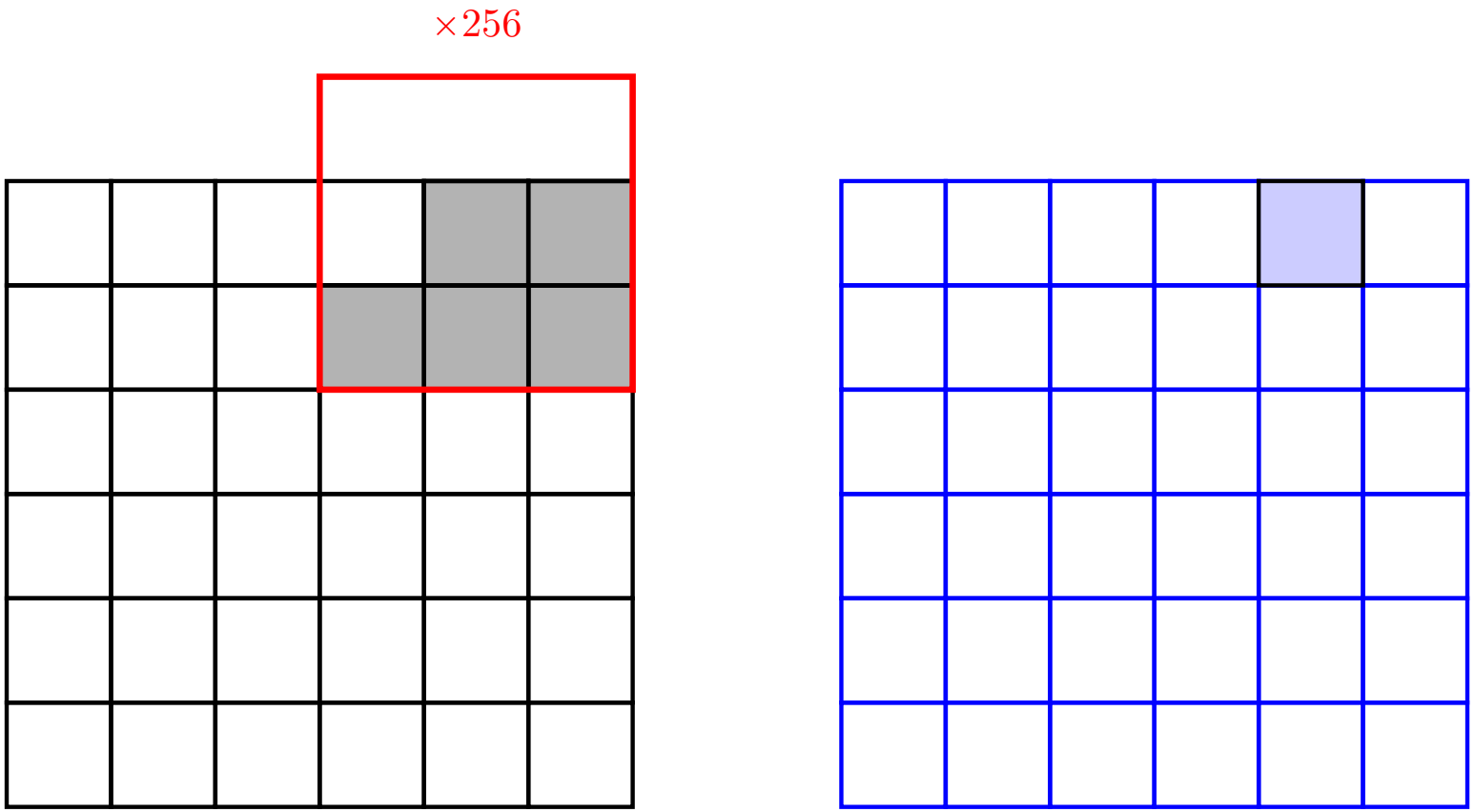
Masked Convolution: *Valid Conditional Distribution*



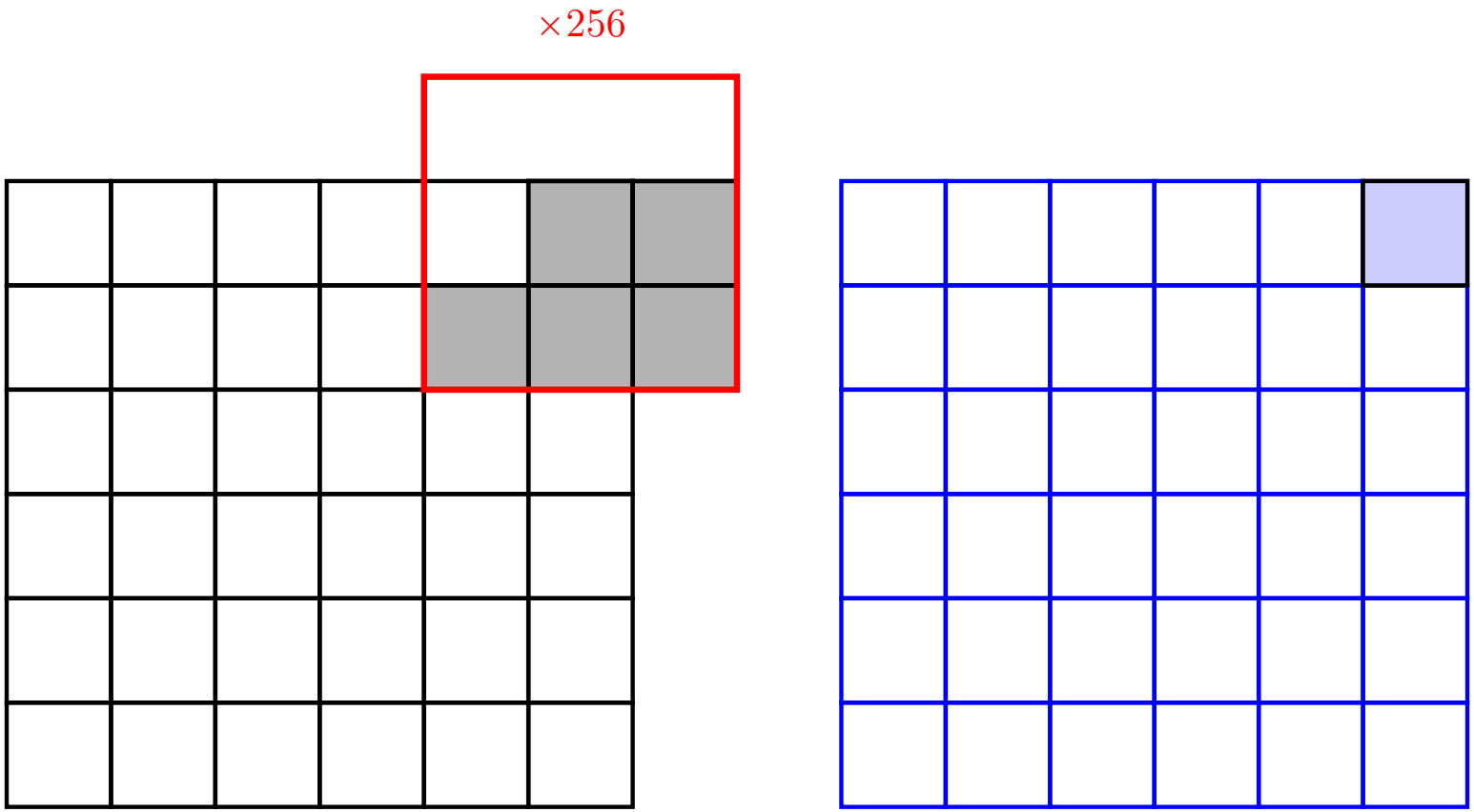
Masked Convolution: *Valid Conditional Distribution*



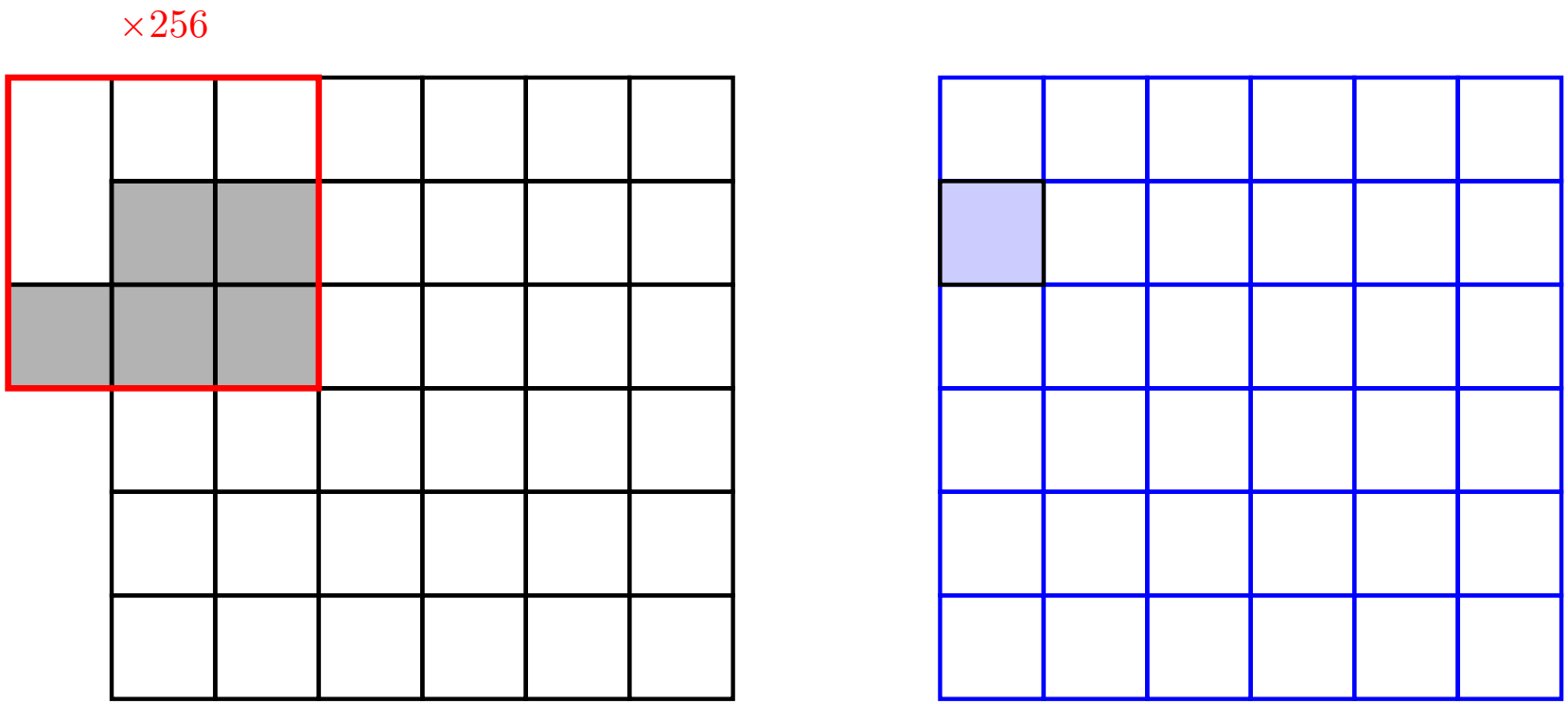
Masked Convolution: *Valid Conditional Distribution*



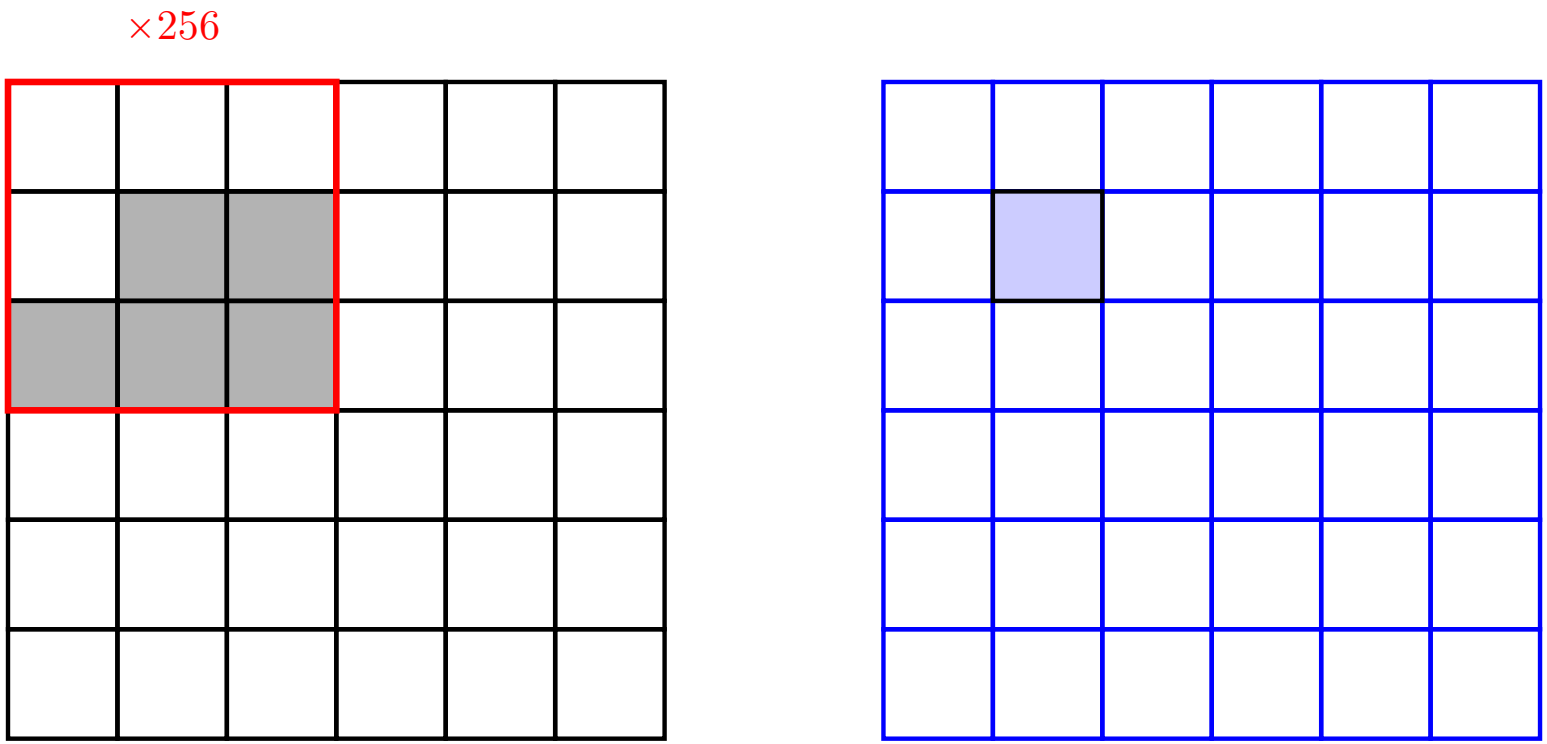
Masked Convolution: *Valid Conditional Distribution*



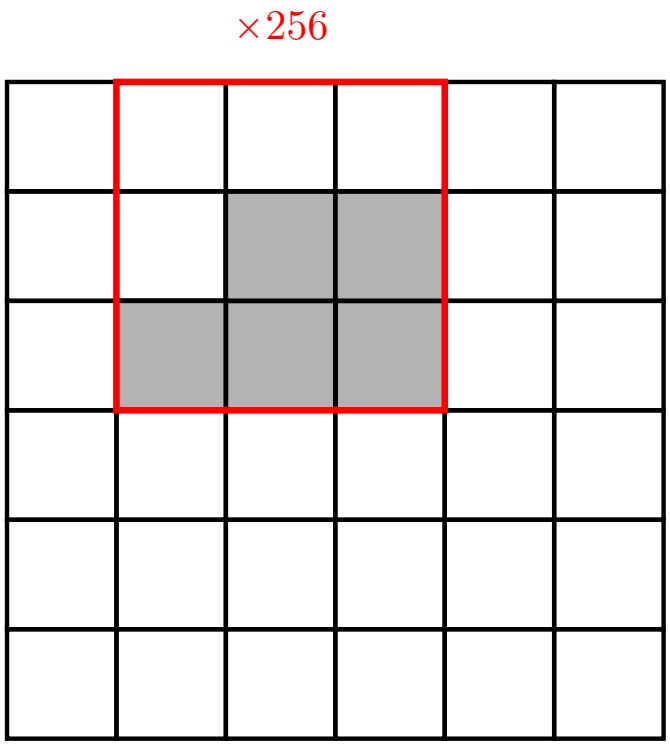
Masked Convolution: *Valid Conditional Distribution*

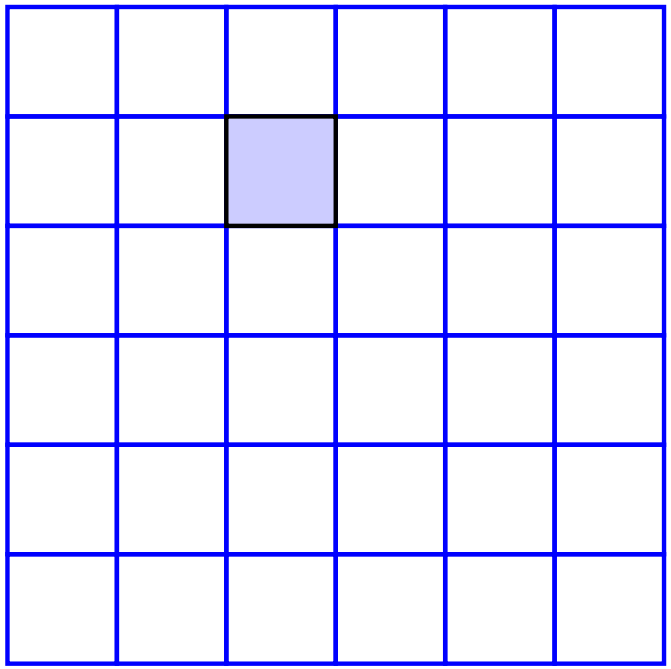


Masked Convolution: *Valid Conditional Distribution*

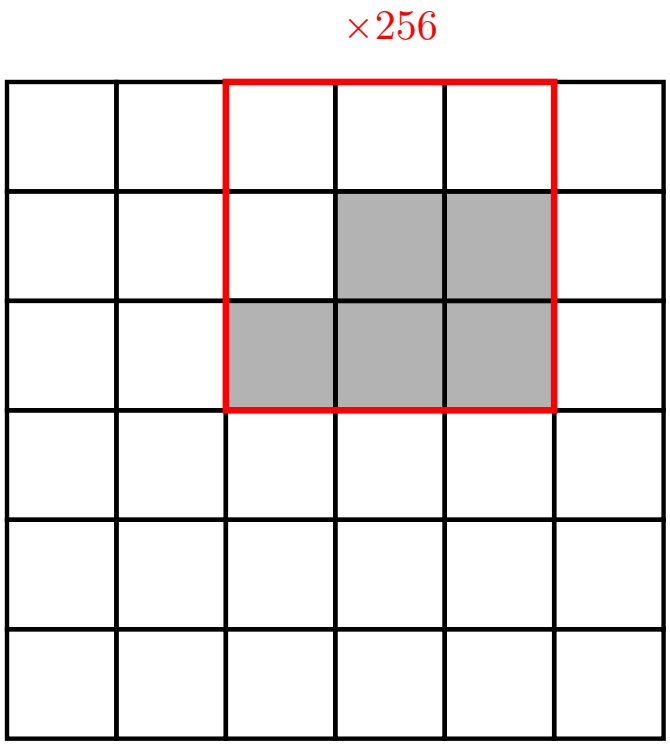


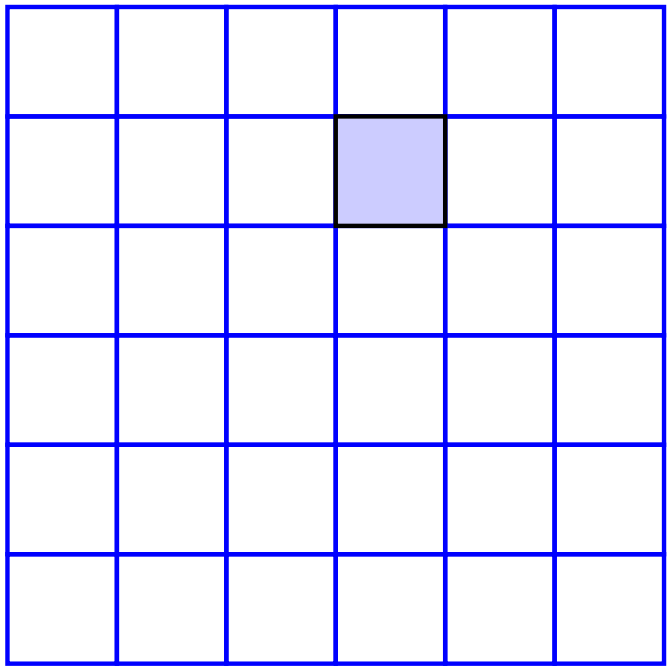
Masked Convolution: *Valid Conditional Distribution*



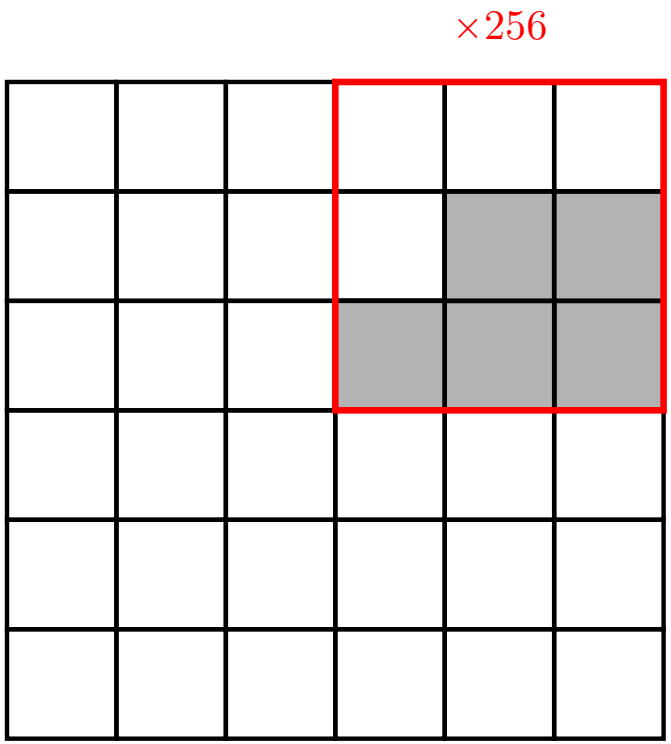


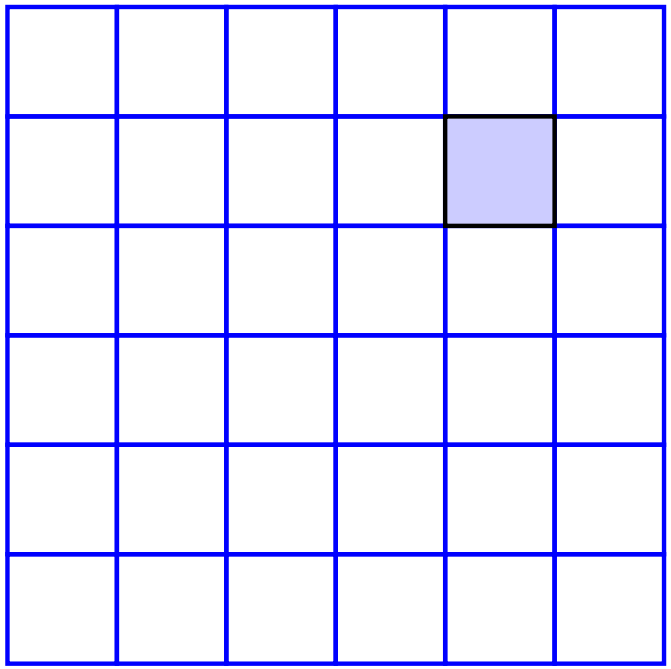
Masked Convolution: *Valid Conditional Distribution*



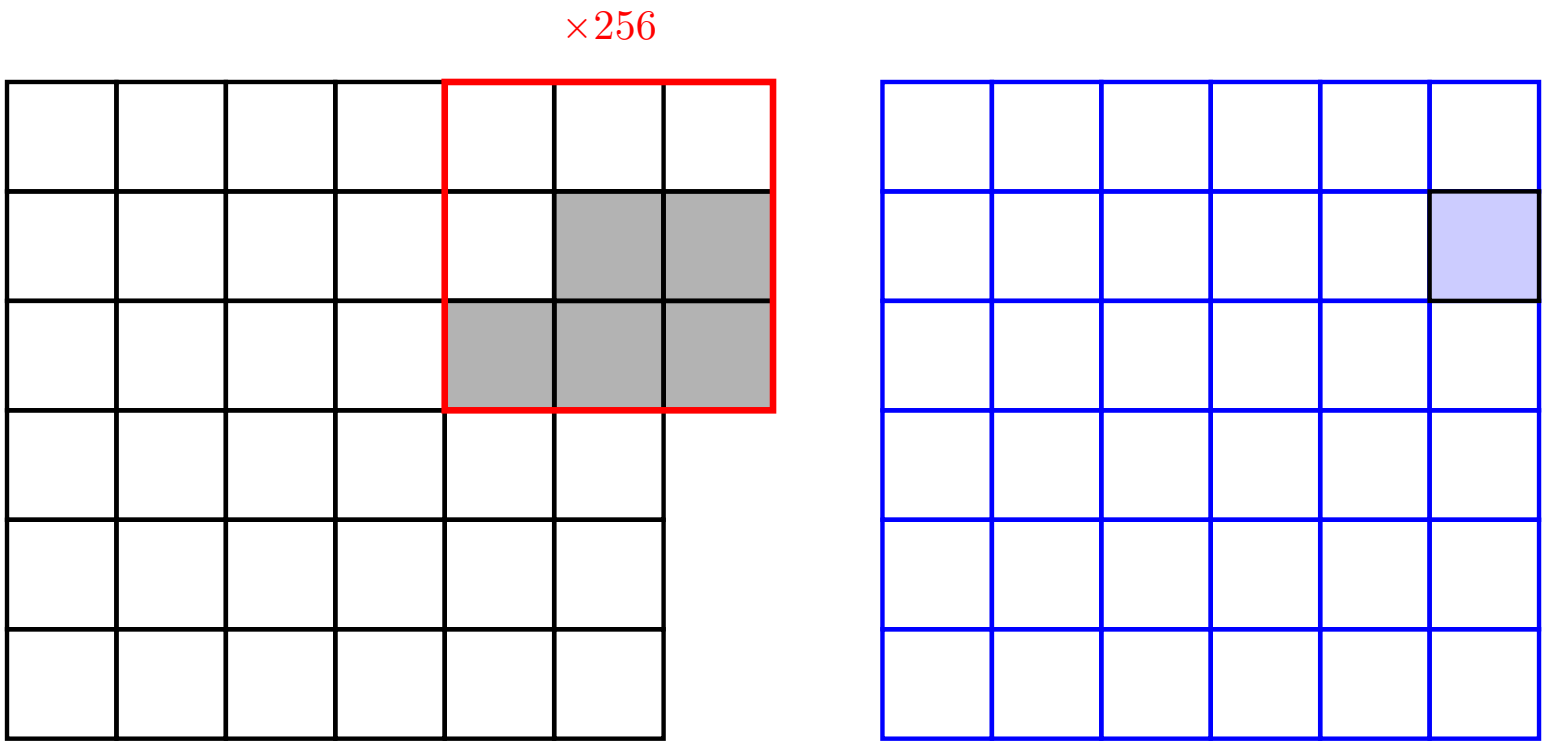


Masked Convolution: *Valid Conditional Distribution*

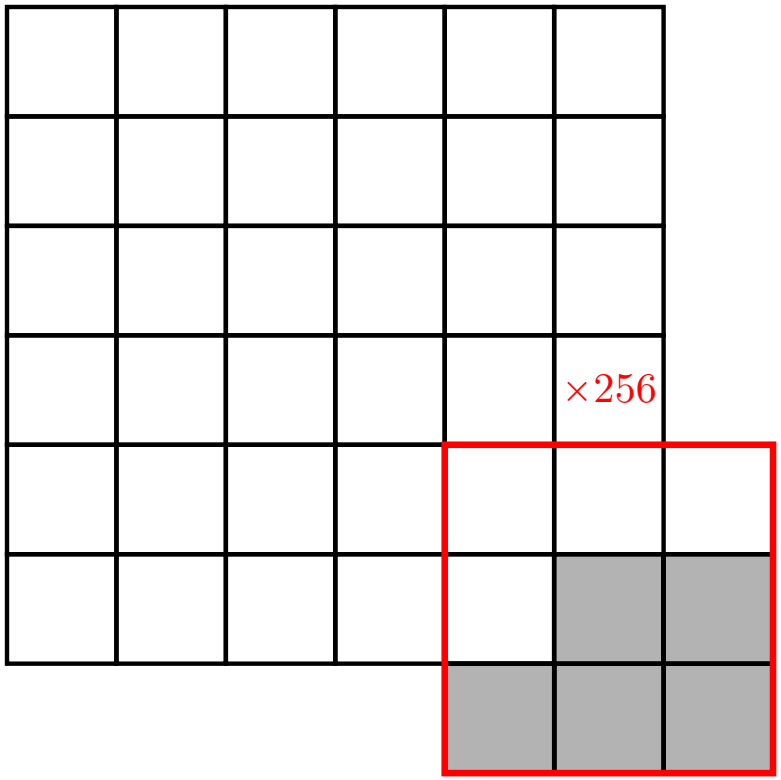


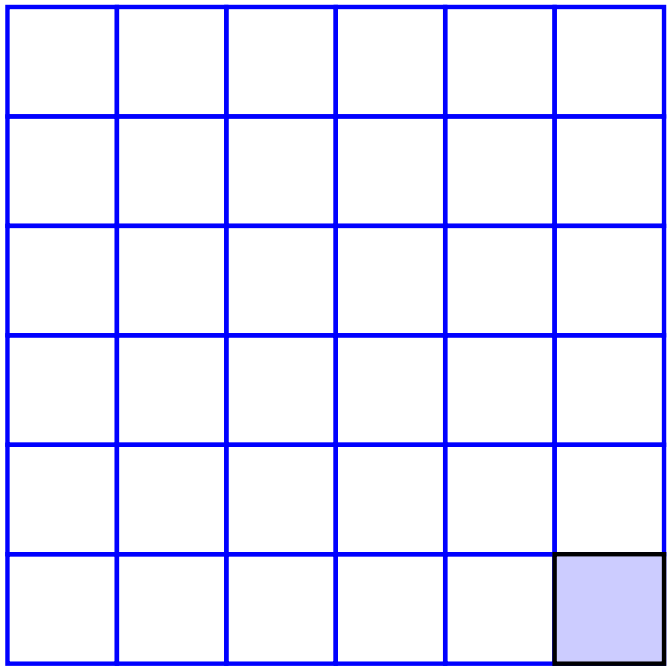


Masked Convolution: *Valid Conditional Distribution*

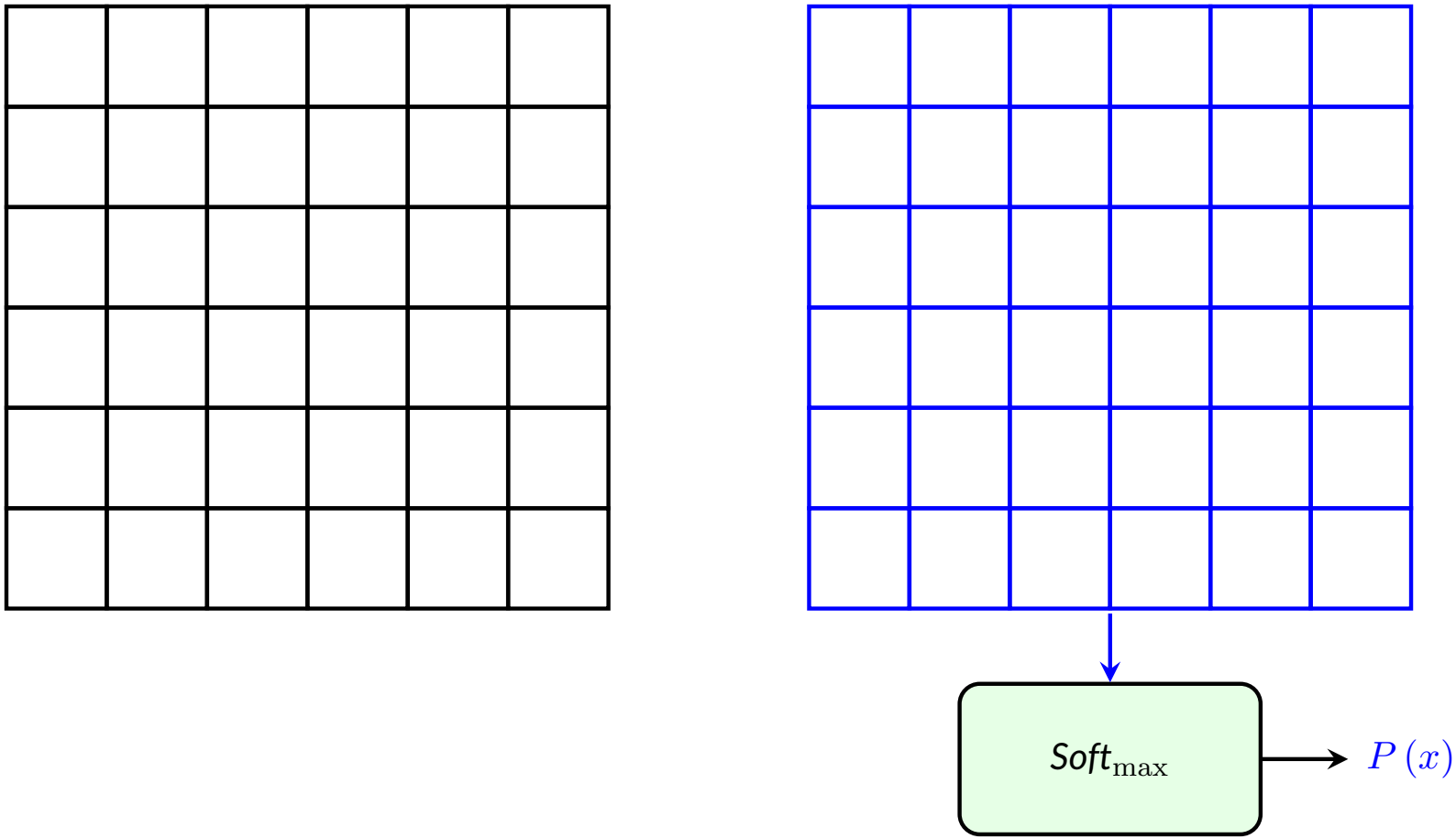


Masked Convolution: *Valid Conditional Distribution*





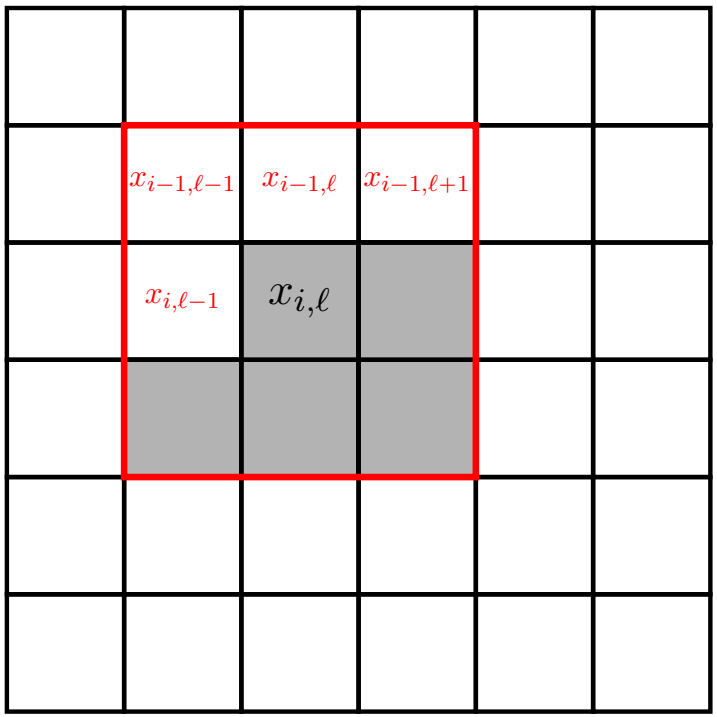
Masked Convolution: *Valid Conditional Distribution*



Short Range Conditioning

Short Range Impact of Masked Convolution

The computed conditional distributions are valid; however, they are conditioned only on nearby pixels within the receptive field of kernel



In terms of AR modeling, we assume

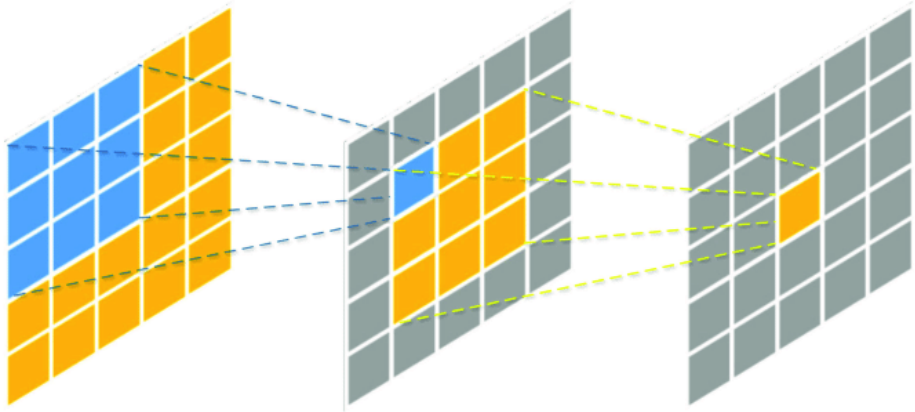
$$P\left(x_{i,\ell}|x_{<i,\ell}\right) \approx P\left(x_{i,\ell}|x_{\text{neighbors}}\right)$$

which weakens spatial correlation capturing

Recap: *Receptive Field of Deep CNN*

Recap: *Deep Convolution*

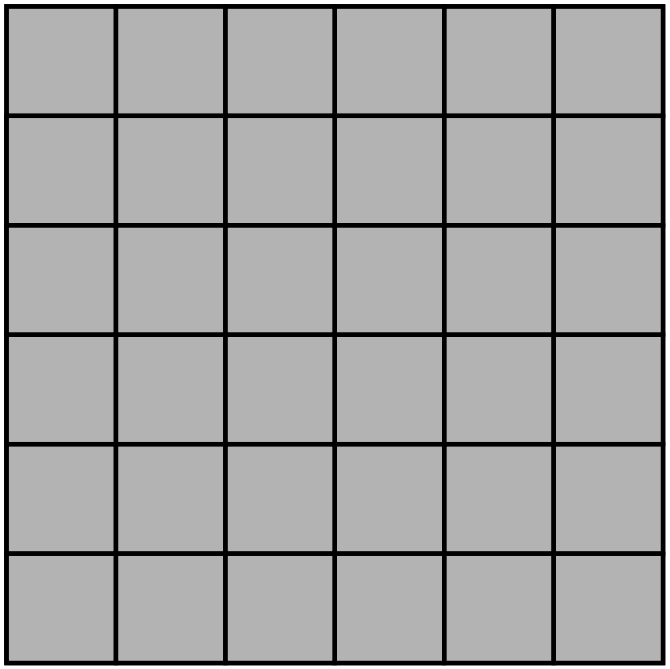
Recall as we go deeper in CNN ⇝ effective receptive field expands

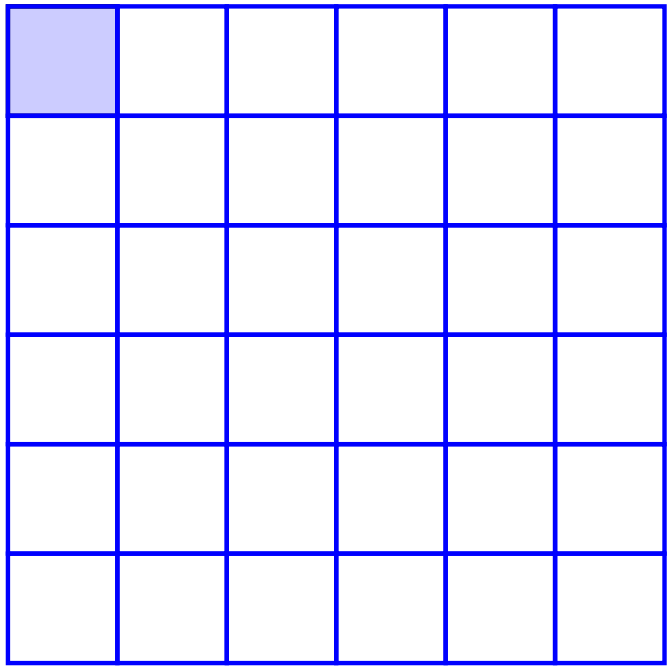


Solution: *Deep Masked Convolution*

Using deep masked CNN ⇝ effective receptive field covers all previous pixels

Deep Masked Convolution

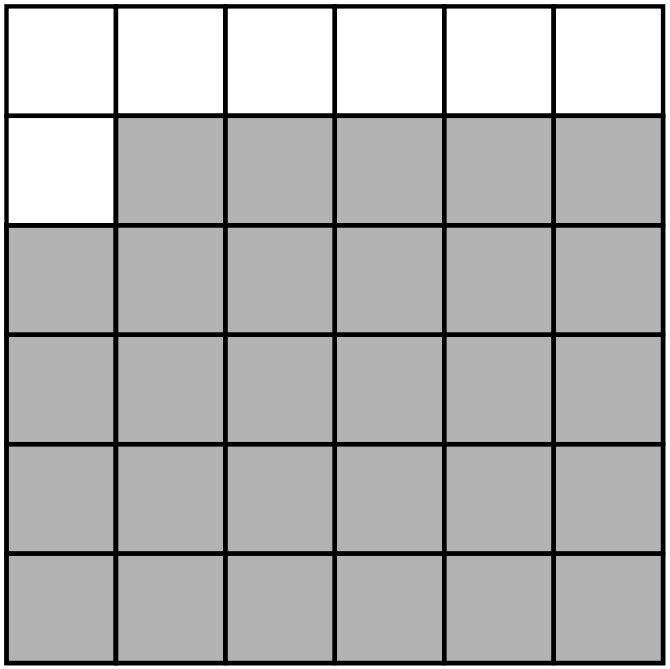


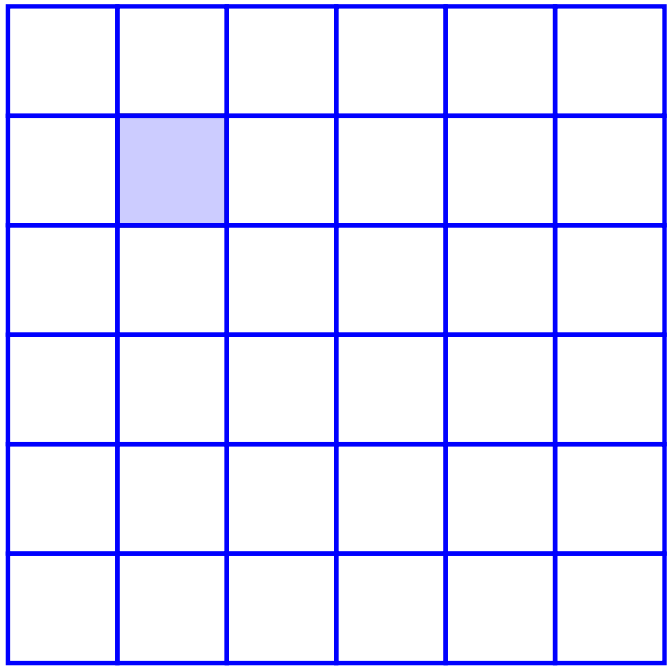


As we go deep with masked convolutional layers

- *receptive field expands to the whole image scene*
- *since all kernels are masked, the final receptive field is also masked*
- *this extracts an autoregressive context that we need*

Deep Masked Convolution

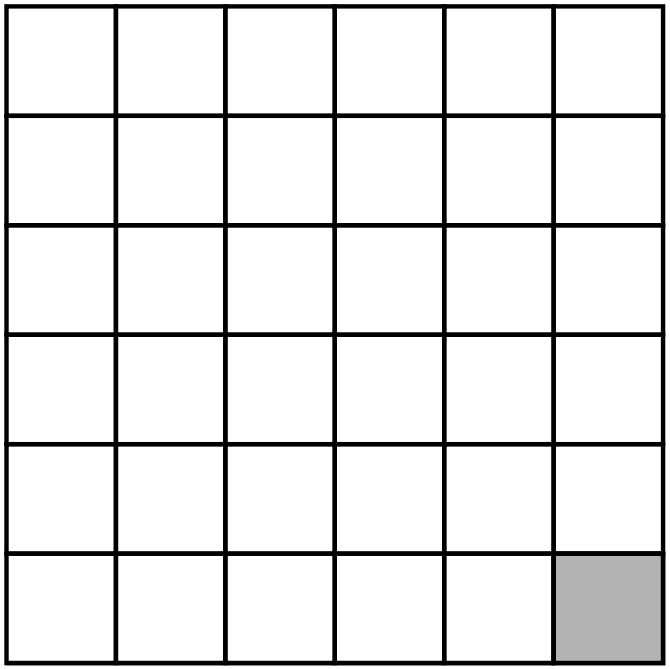


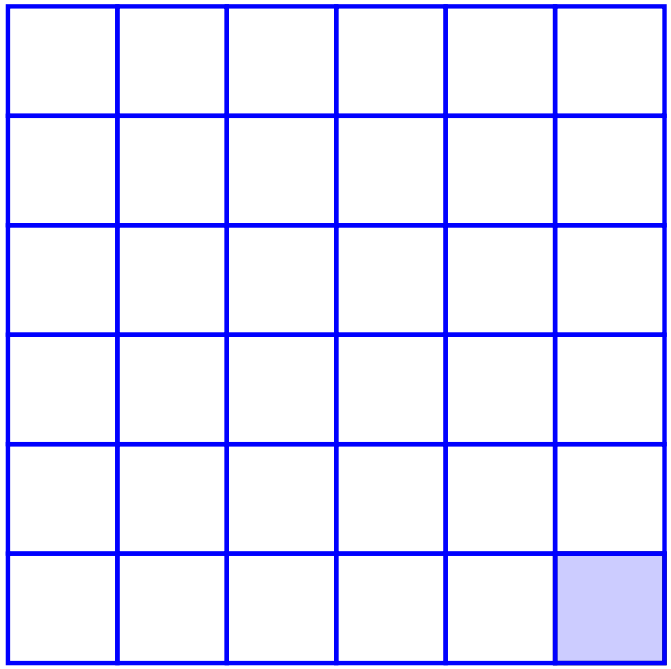


As we go deep with masked convolutional layers

- *receptive field expands to the whole image scene*
- *since all kernels are masked, the final receptive field is also masked*
- *this extracts an autoregressive context that we need*

Deep Masked Convolution

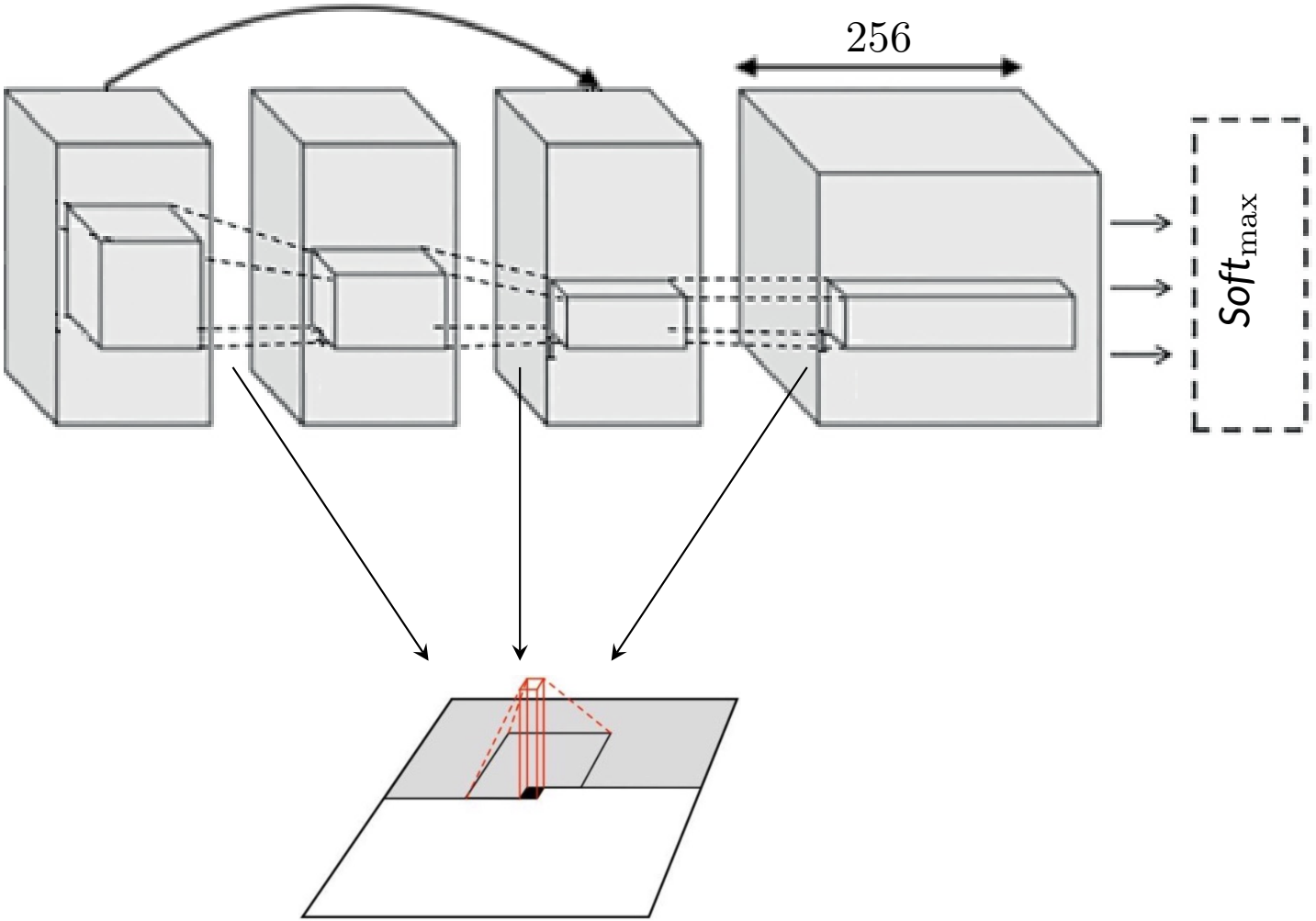




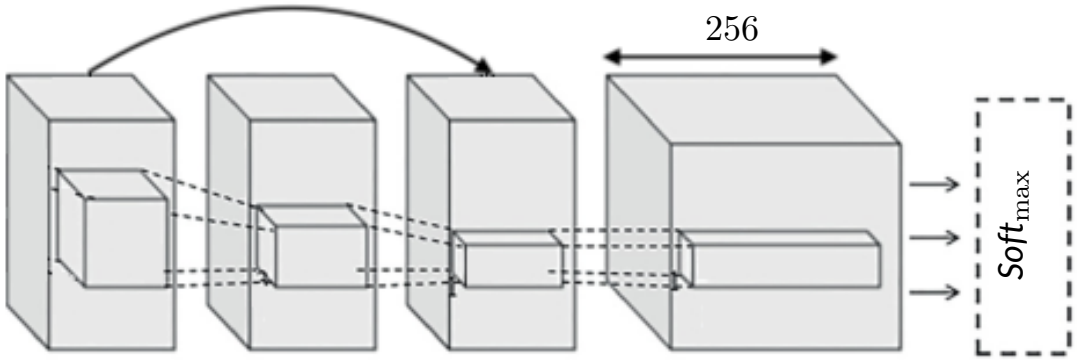
As we go deep with masked convolutional layers

- *receptive field expands to the whole image scene*
- *since all kernels are masked, the final receptive field is also masked*
- *this extracts an autoregressive context that we need*

PixelCNN: *AR Model with Deep Masked Convolution*



PixelCNN: *Few Notes*



In practical implementation of PixelCNN

- *Typically the network is quite deep* > 15
  - ↳ *To guarantee accurate conditioning*
- *Deep networks can quickly experience vanishing gradients*
  - ↳ *We use skip connextions and gating to reduce vanishing gradients*
- *Final layer extracts 256 features for each pixel*
  - ↳ *Like PixelRNN, we typically condition RGB channels right after each other*

PixelCNN: *Training*

We can train PixelCNN via MLE

`Train_PixelCNN(`$\mathbb{D}$`:dataset):`

1: **for** *multiple epochs* **do**
2: *Sample a batch of* n *images* $\{x^j : j = 1, \ldots, n\}$
3: **for** $j = 1 : n$ **do**
4: $\mathbf{p}^j_{1:d} \leftarrow$ `MaskedCNN(`x^j`)`
5: **end for**
6: *Update model using* `Opt_avg` $\left\{ \sum_i \nabla \log \mathbf{p}^j_i \left[x^j_i \right] \right\}$
7: **end for**
8: **return** *trained model*

Parallel Processing

Using masked convolution, we can process all pixels in parallel while we train

PixelCNN: *Generation*

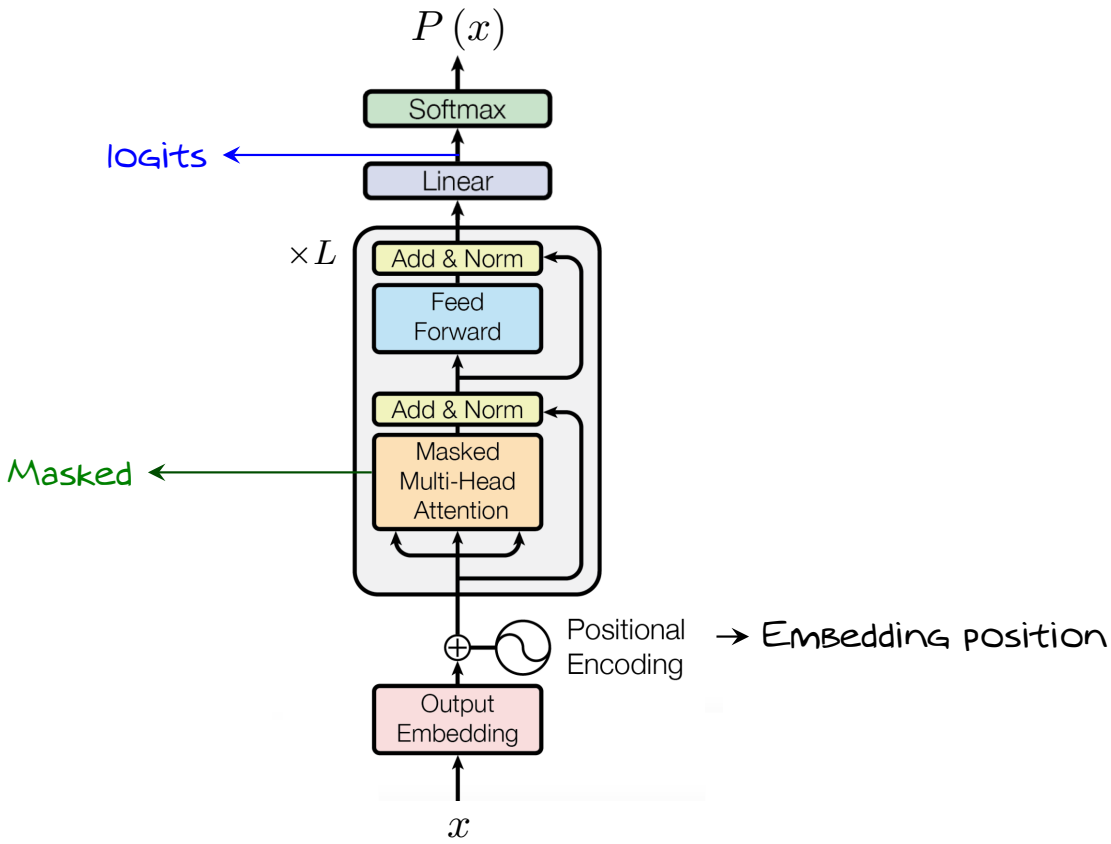
Generation is still autoregressive

`Sample_PixelCNN( ):`

1: Set $x^{(0)} = \varnothing$ `#prior sampling`
2: **for** $i = 1, \ldots, d$ **do**
3: $\quad \mathbf{p}_{1:d} \leftarrow$ `MaskedCNN(`$x^{(i-1)}$`)`
4: $\quad x_i \leftarrow$ `multinomial(`$\mathbf{p}_i$`, #smpl=1)`
5: $\quad$ Update $x^{(i)} \leftarrow$ `Replace` x_i `in` $x^{(i-1)}$
6: **end for**
7: **return** $x^{(d)}$

Transformer-Based AR

We can use the same approach as in LMs to extract context via transformers



This is exactly LM with time ≡ position and tokens ≡ pixels

ImageGPT: *Extending LMs to Generate Pixels*

A good example is ImageGPT:

> *it interprets pixels as tokens* $\rightsquigarrow$ *image is a long sequence of tokens*

- *Vocalbulary has only* 256 *tokens*
  - ↳ *We can patch multiple pixels together to make larger vocabulary*
  - ↳ *With patch-pixels, we can use tokenization to reduce vocabulary size*
- *Each pixel-patch is embedded with an embedding layer*
  - ↳ *This embedding can be in general smaller than vocabulary size*
  - ↳ *It will give us a numerical representation of the image patches*
- *As we use self-attention, the model learns spatial correlation by itself*
  - ↳ *This is more robust as compared to deep masked convolution*
  - ↳ *It however needs more data to start capturing spatial pattern*

Take a look at ImageGPT Paper

Wrap Up and Final Notes

AR modeling gives an efficient way to computationally model data distribution

↳ *It deploys autoregressive sampling leading from slow generation*

We can realize them by

- *Recurrent networks, e.g., LSTM*
  - ↳ *It explicitly conditions distribution through model state*
  - ↳ *We do not benefit from parallel processing*
- *Masked networks, e.g., masked CNN or Transformers*
  - ↳ *It conditions causally through masking*
  - ↳ *We enjoy the luxury of parallel processing*

AR modeling are widely considered slow of visual modalities

↳ *They are though getting revisited due to some recent advancement*

Next Stop: Energy-based Models!