Lecture 3: Deep Autoregressive Models

Deep Generative Models

Sajjad Amini

Department of Electrical Engineering
Sharif University of Technology

Contents

Section 1

Fully Visible Sigmoid Belief Network

Problem Statement

The original Binary MNIST dataset is $\{(\boldsymbol{x}_n, \bar{y}_n)\}_{n=1}^N$ where:

$$\begin{cases} \boldsymbol{x}_n \in \{0,1\}^{28\times 28} \\ \bar{y} \in \{0,1,\ldots,9\} \end{cases}$$

Assume we define a new problem where the labels are binarized as:

$$y_n = \begin{cases} 0 & \text{if } \bar{y}_n < 5 \\ 1 & \text{Otherwise} \end{cases}$$

Thus we have a new dataset $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ where:

$$\begin{cases} \boldsymbol{x}_n \in \{0,1\}^{28\times 28} \\ y_n \in \{0,1\} \end{cases}$$

Binary Logistic Regression

Model

Assume dataset $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, we can use a BLR model as:

$$p_\theta(y|\boldsymbol{x}) = \text{Ber}(y|\sigma(\boldsymbol{w}^T\boldsymbol{x} + b)),\ \boldsymbol{\theta} = \{\boldsymbol{w}, b\}$$

where:

$\sigma(\cdot)$	Sigmoid function
$\boldsymbol{w}$	Weight vector
b	Bias value

Training

The model is trained easily via KL divergence. For more details please refer to [1].

Describing $\mathcal{P}$

Generative Modeling

Assume we just have BMINST image $\{\boldsymbol{x}_i\}_{i=1}^N$ without any label and we want to estimate generating distribution $p(\boldsymbol{x})$. Using the chain rule, we have:

$$p(\boldsymbol{x}) = p(x_1)p(x_2|\boldsymbol{x}_{<2}) \dots p(x_d|\boldsymbol{x}_{<d}) \dots p(x_D|\boldsymbol{x}_{<D}), \ \boldsymbol{x}_{<d} \triangleq [x_1, \dots, x_{d-1}]^T$$

Each of the factors on the right side can be seen as a BLR problem. Thus for the first term, we have:

$$p_\theta(x_1) = \text{Ber}(x_1|\sigma(b_1)), \ b_1 \in \mathbb{R}$$

And for arbitrary d-th term with $1 < d \leq D$, we have:

$$p_\theta(x_d|\boldsymbol{x}_{<d}) = \text{Ber}(x_d|\sigma(b_d + \boldsymbol{w}_d^T \boldsymbol{x}_{<d})), \begin{cases} b_d \in \mathbb{R} \\ \boldsymbol{w}_d \in \mathbb{R}^{d-1} \end{cases}$$

So: $\boldsymbol{\theta} = \{b_1, b_2, \boldsymbol{w}_2, \dots, b_D, \boldsymbol{w}_D\}$

Describing $\mathcal{P}$

Number of Parameters

For the general case of chain rule we have:

$$p(\boldsymbol{x}) = \overbrace{p(x_1)}^{\#1}\overbrace{p(x_2|\boldsymbol{x}_{<2})}^{\#2}\ldots\overbrace{p(x_D|\boldsymbol{x}_{<D})}^{\#2^{D-1}}$$

Thus the number of parameters for $\mathcal{P}$ is:

$$2^0 + 2^1 + 2^2 + \ldots + 2^{D-1} = 2^D - 1$$

Describing $\mathcal{P}_\theta$

Number of Parameters for $\mathcal{P}_\theta$

For FVSBN, we have:

$$p_\theta(\boldsymbol{x}) = p_\theta(x_1)p_\theta(x_2|\boldsymbol{x}_{<2})\dots p_\theta(x_D|\boldsymbol{x}_{<D})$$

$$\Rightarrow p_\theta(\boldsymbol{x}) = \overbrace{\text{Ber}(x_1|\sigma(b_1))}^{\#1}\overbrace{\text{Ber}(x_2|\sigma(b_2+w_2^T\boldsymbol{x}_{<2}))}^{\#2}\dots\overbrace{\text{Ber}(x_D|\sigma(b_D+\boldsymbol{w}_D^T\boldsymbol{x}_{<n}))}^{\#D}$$

Thus the number of parameters for $\mathcal{P}_\theta$: $1+2+3+\dots+D=\frac{D(D+1)}{2}$. Note that:

$$\boldsymbol{\theta} = \{b_1, b_2, \boldsymbol{w}_2, \dots, b_D, \boldsymbol{w}_D\}$$

Task: Density Estimation

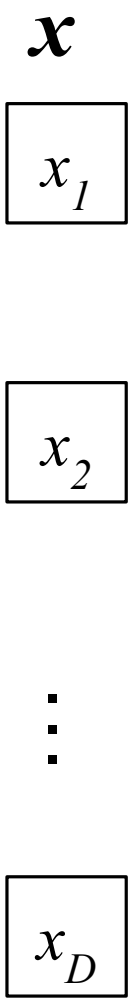


Figure: Input D dimensioanl vector $\boldsymbol{x}$

Task: Density Estimation

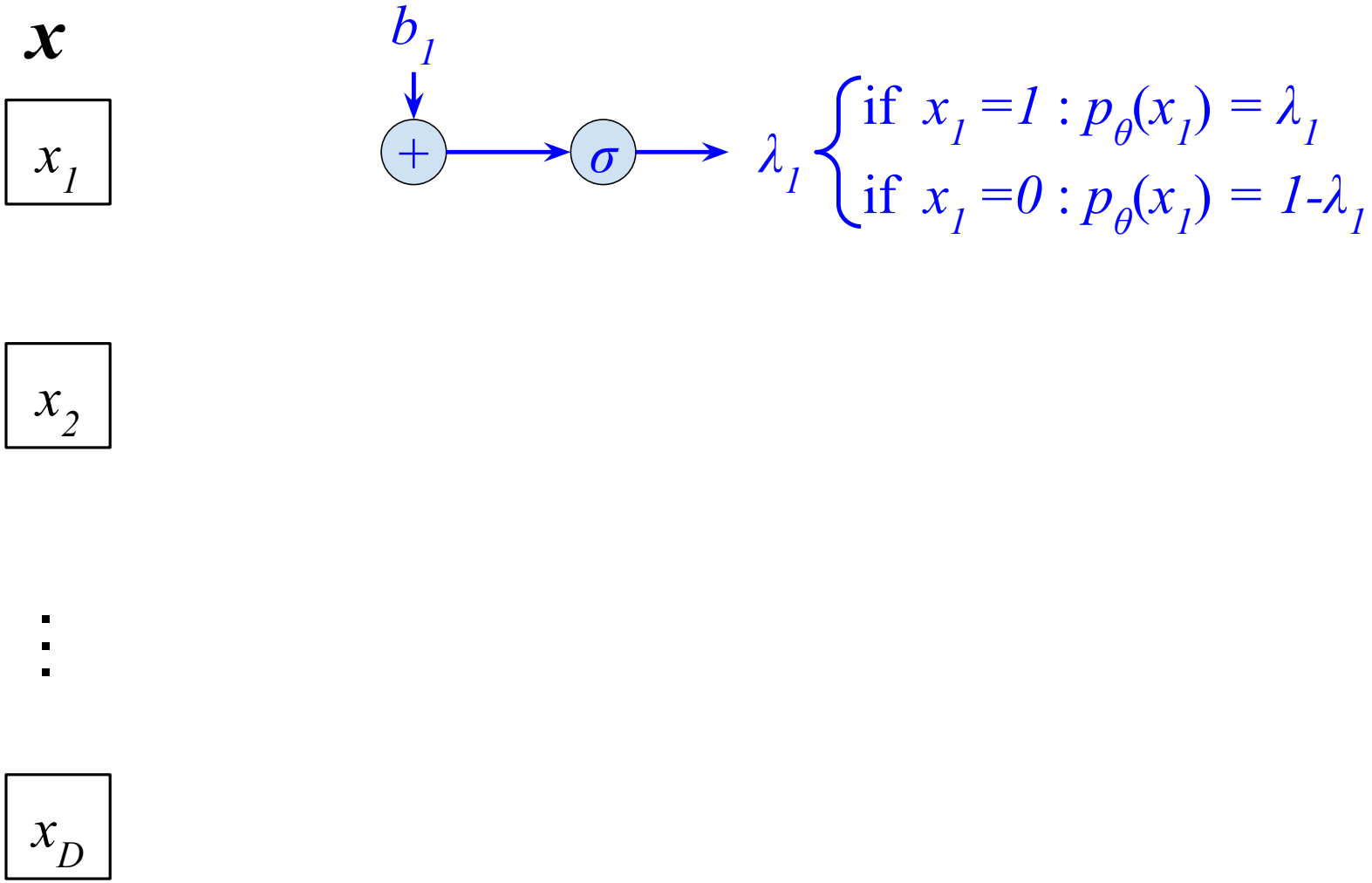


Figure: Calculations for $p_\theta(x_1)$

Task: Density Estimation

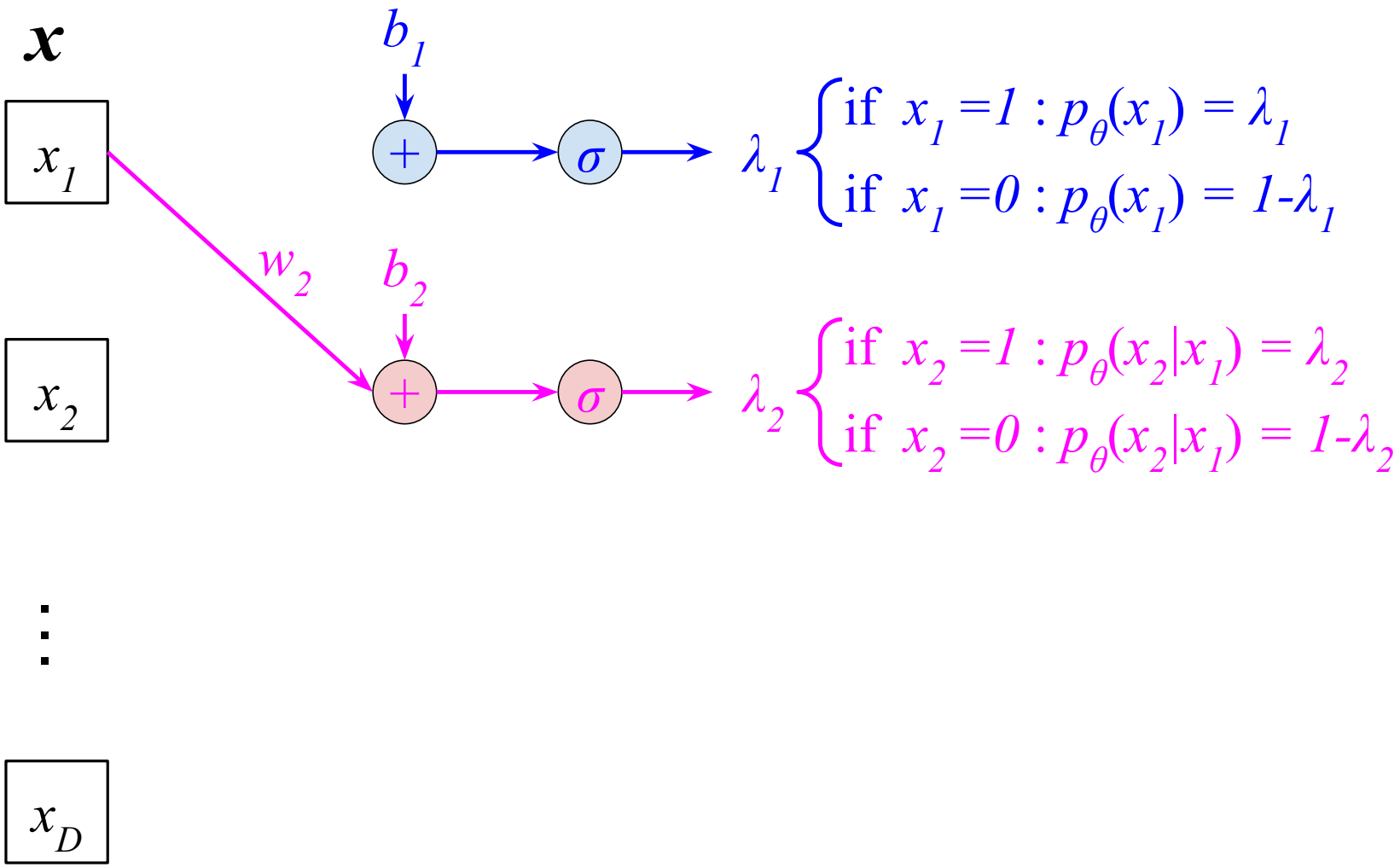


Figure: Calculations for $p_\theta(x_2|x_1)$

Task: Density Estimation

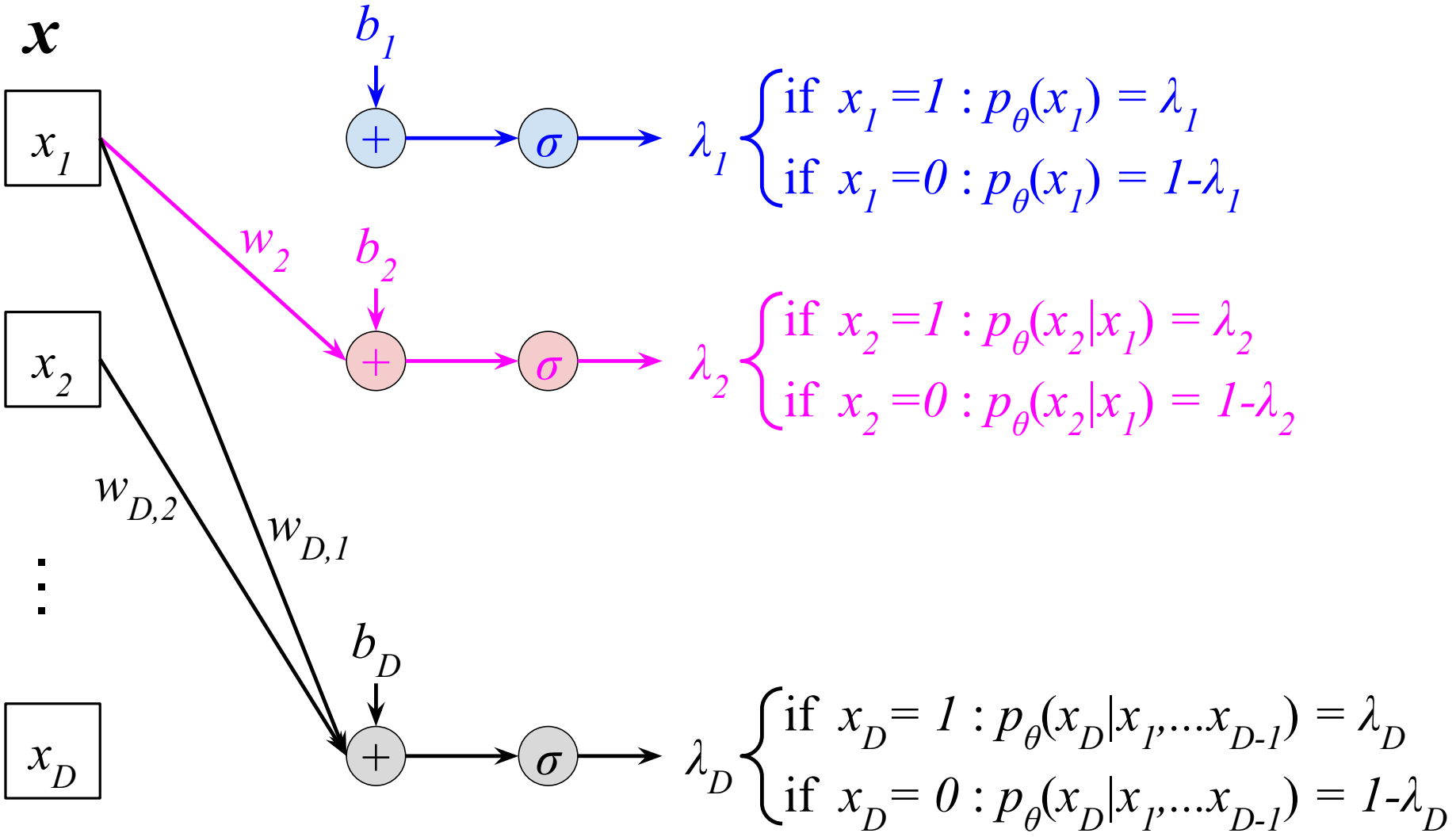


Figure: Calculations for $p_\theta(x_D|x_1 \dots x_{D-1})$

Density Estimation

For now, assume the parameters for FVSBN $\boldsymbol{\theta}$ are given and define:

$$\lambda_d(\boldsymbol{x}_{<d}) \triangleq p_\theta(X_d = 1|\boldsymbol{x}_{<d}) = \sigma(b_d + \boldsymbol{w}_d^T \boldsymbol{x}_{<d})$$

Then for a simple case with $n = 4$ we have:

$$p_\theta(\boldsymbol{x} = [0,1,1,0]^T) = \overbrace{p_\theta(X_1 = 0)}^{\text{Ber}(X_1|\lambda_1)} \overbrace{p_\theta(X_2 = 1|\boldsymbol{x}_{<2} = [0]^T)}^{\text{Ber}(X_2|\lambda_2(\boldsymbol{x}_{<2}=[0]^T))}$$

$$\overbrace{p_\theta(X_3 = 1|\boldsymbol{x}_{<3} = [0,1]^T)}^{\text{Ber}(X_3|\lambda_3(\boldsymbol{x}_{<3}=[0,1]^T))} \overbrace{p_\theta(X_4 = 0|\boldsymbol{x}_{<4} = [0,1,1]^T)}^{\text{Ber}(X_4|\lambda_4(\boldsymbol{x}_{<4}=[0,1,1]^T))}$$

$$= (1 - \lambda_1)(\lambda_2)(\lambda_3)(1 - \lambda_4)$$

Task: Generation

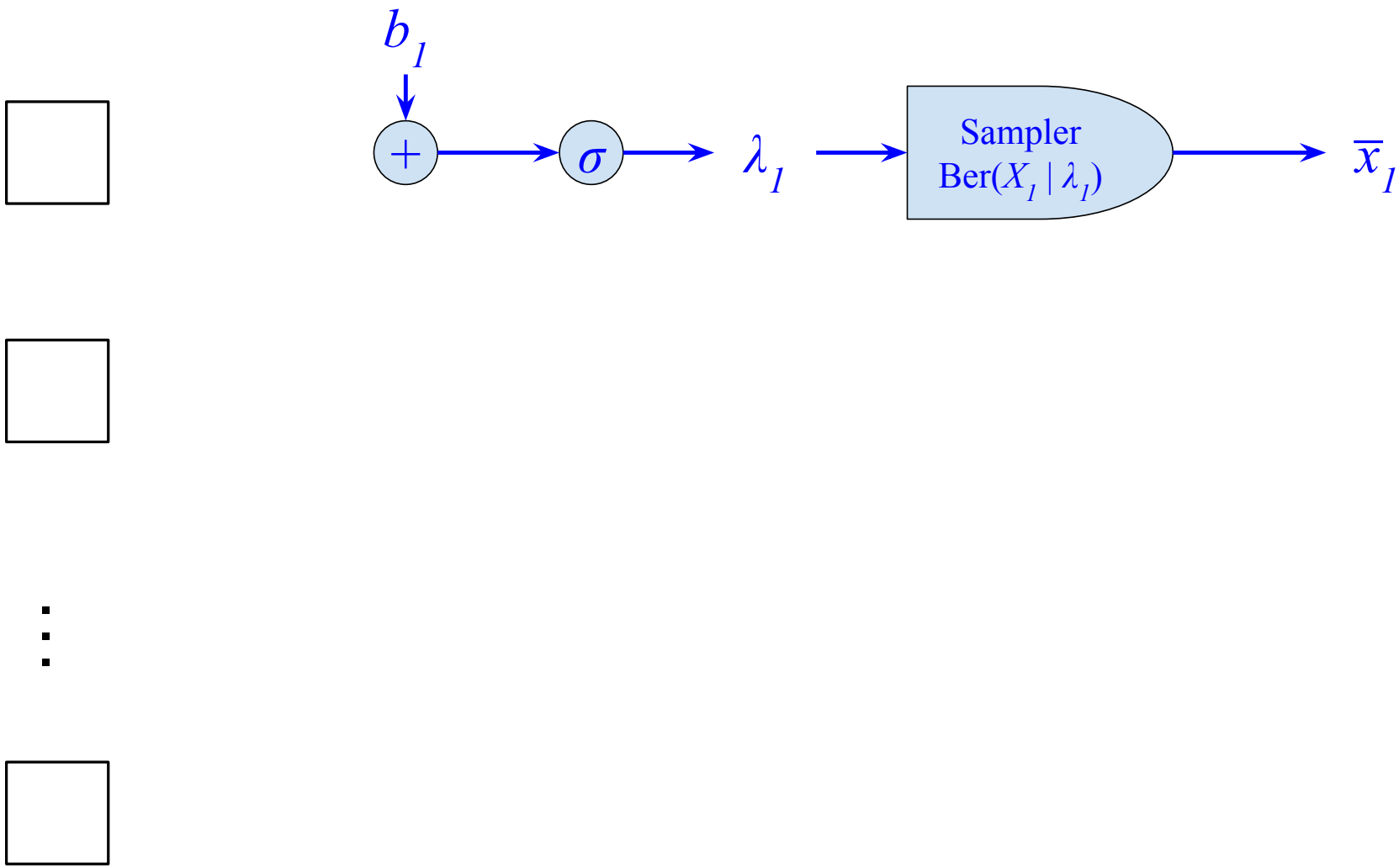


Figure: Sampling first element

Task: Generation

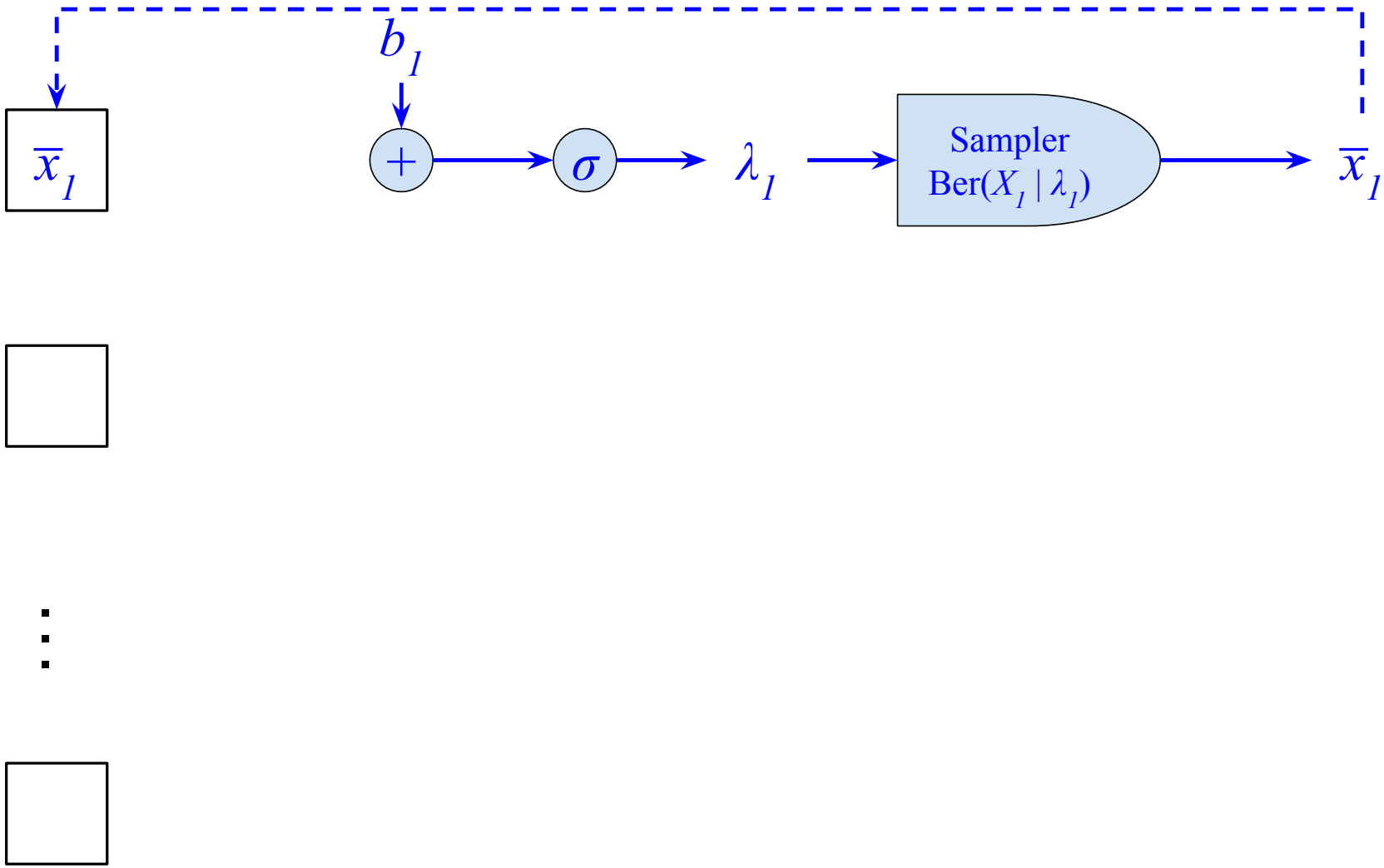


Figure: Entring the first sampled element in the vector

Task: Generation

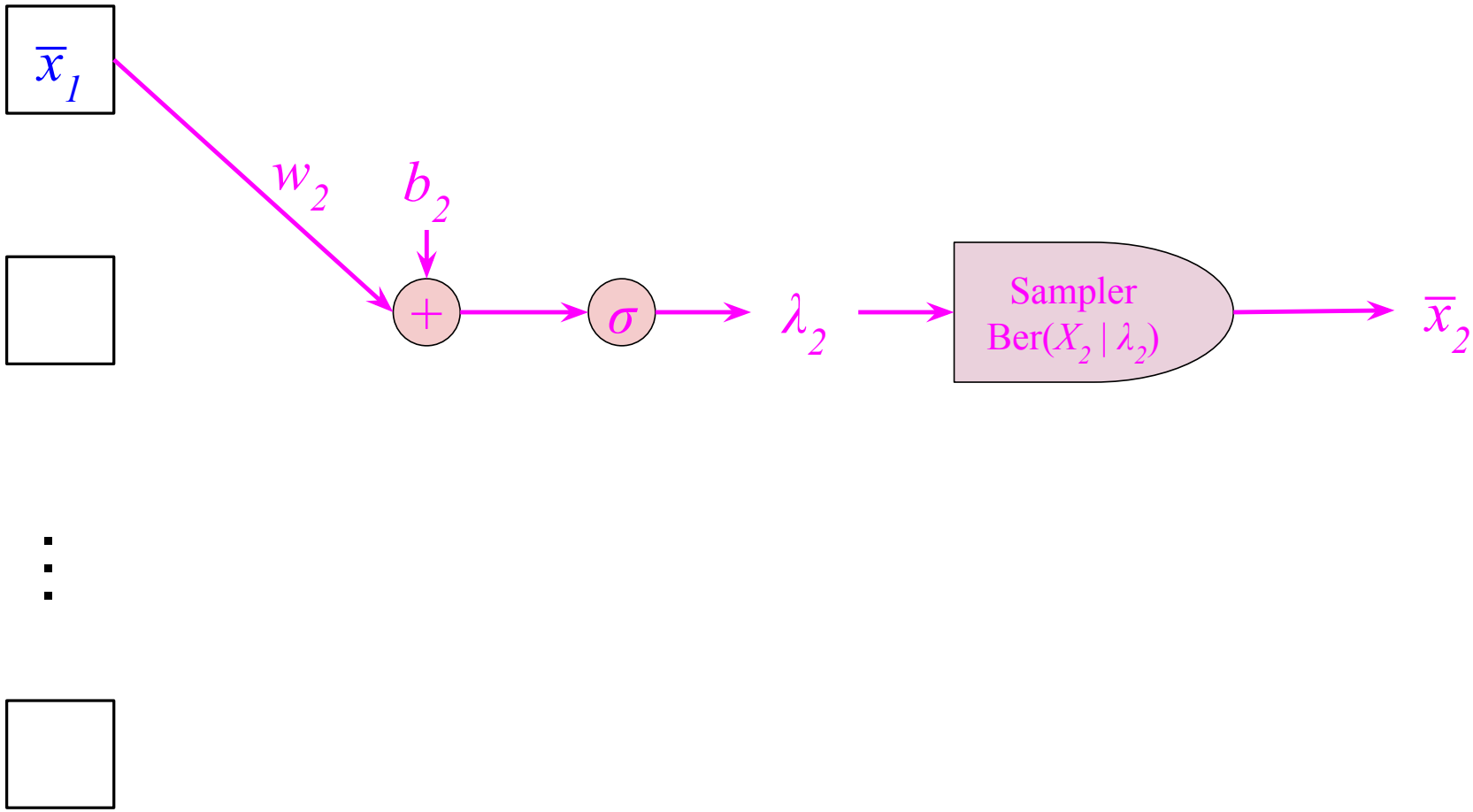


Figure: Sampling second element

Task: Generation

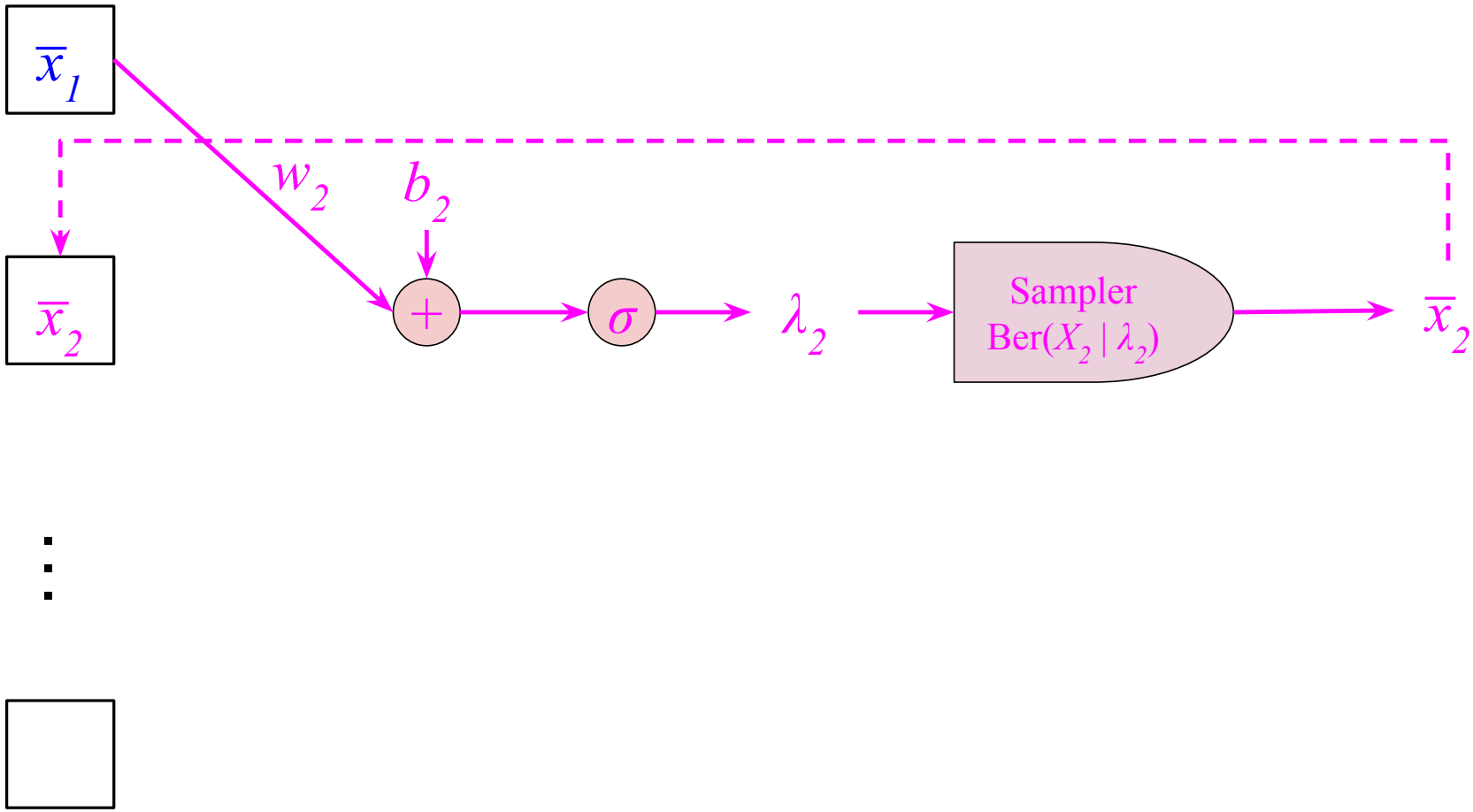


Figure: Entring the second sampled element in the vector

Task: Generation

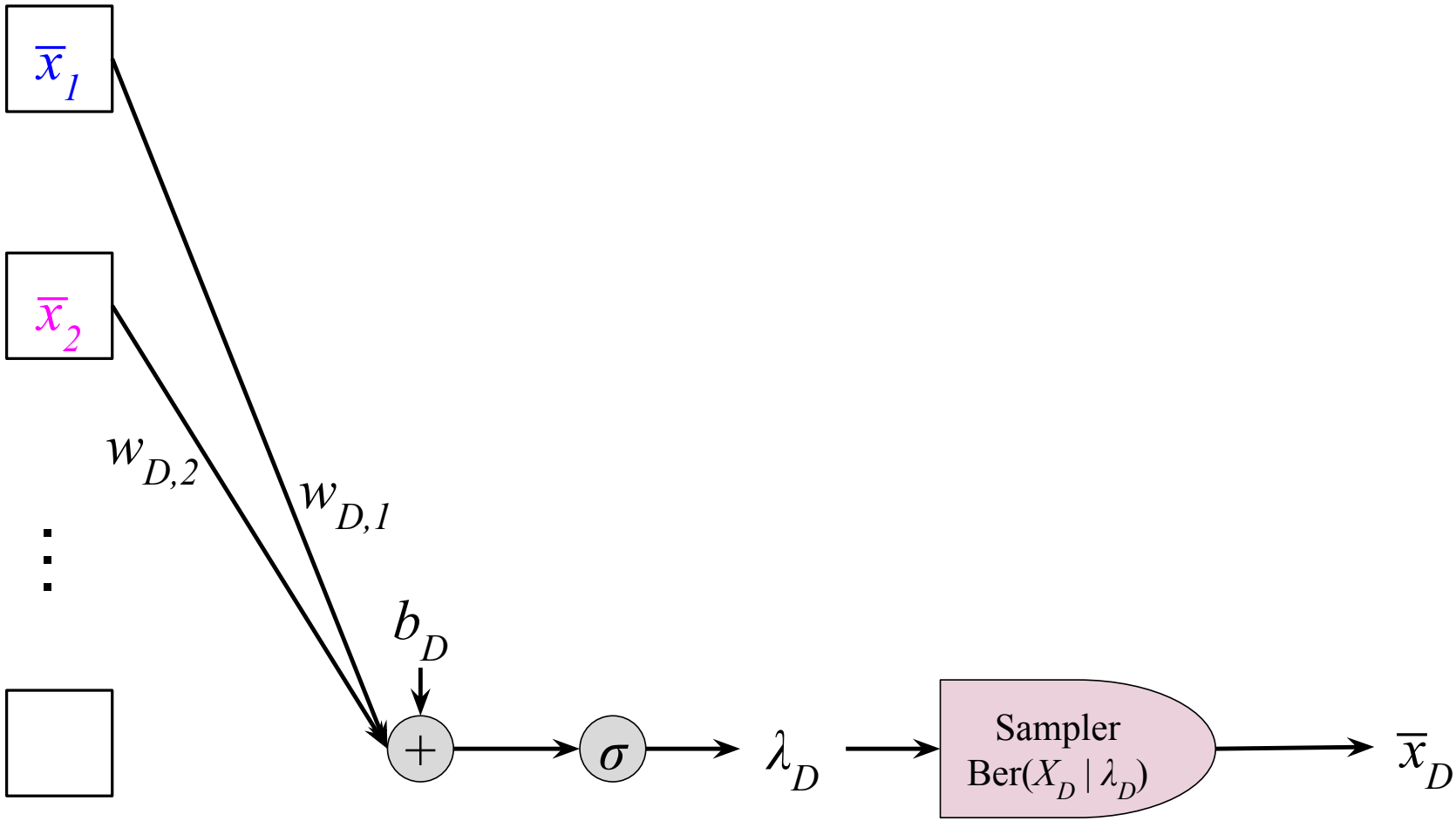


Figure: Sampling D-th element

Task: Generation

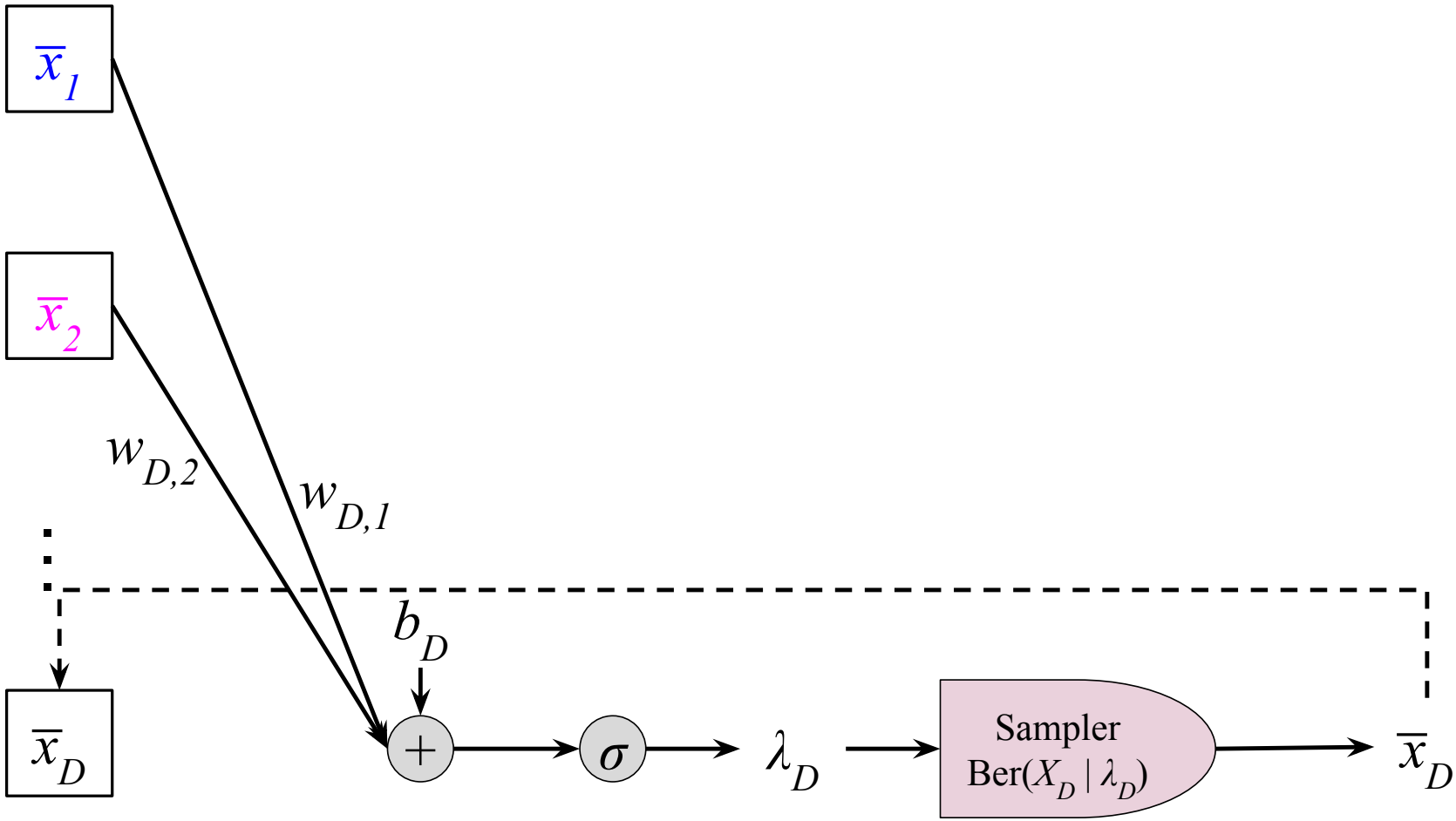


Figure: Entring the D-th sampled element in the vector

Generation

Again assume the parameters for FVSBN $\boldsymbol{\theta}$ are given and define:

$$\lambda_d(\boldsymbol{x}_{<d}) \triangleq p_\theta(X_d = 1|\boldsymbol{x}_{<d}) = \sigma(b_d + \boldsymbol{w}_d^T \boldsymbol{x}_{<d})$$

Then you can sample $p(x)$ as:

$$\bar{x}_1 \sim p_\theta(X_1) = \text{Ber}(X_1|\lambda_1)$$

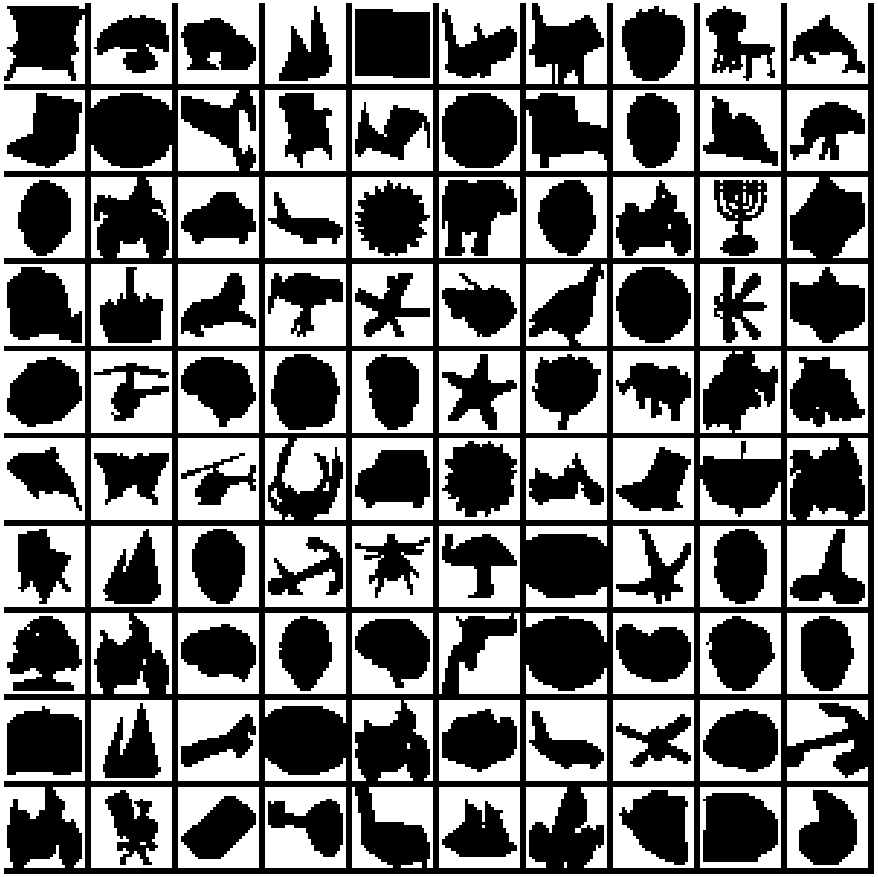
$$\bar{x}_2 \sim p(X_2|\boldsymbol{x}_{<2} = [\bar{x}_1]^T) = \text{Ber}(X_2|\lambda_2(\boldsymbol{x}_{<2} = [\bar{x}_1]^T))$$

$$\bar{x}_3 \sim p(X_3|\boldsymbol{x}_{<3} = [\bar{x}_1, \bar{x}_2]^T) = \text{Ber}(X_3|\lambda_3(\boldsymbol{x}_{<3} = [\bar{x}_1, \bar{x}_2]^T))$$

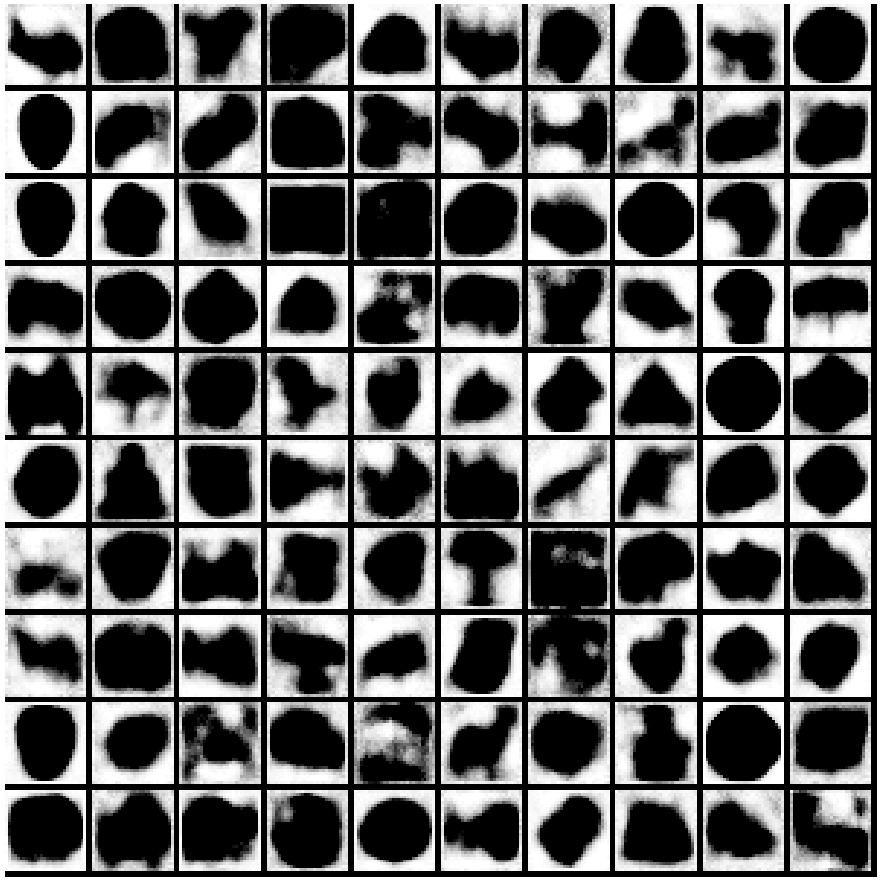
$$\vdots$$

$$\bar{x}_D \sim p(X_D|\boldsymbol{x}_{<D} = [\bar{x}_1, \ldots \bar{x}_{D-1}]^T) = \text{Ber}(X_D|\lambda_D(\boldsymbol{x}_{<D} = [\bar{x}_1, \ldots \bar{x}_{D-1}]^T))$$

FVSBN



(a) Dataset samples



(b) Generated samples

Figure: FVSBN performance over Caltech 101 dataset (source: [2])

Section 2

Neural Autoregressive Density Estimation

BLR Limitation

Again assume dataset $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\begin{cases} \boldsymbol{x}_n \in \{0,1\}^{28\times 28} \\ y_n \in \{0,1\} \end{cases}$, and BLR model as:

$$p_\theta(y|\boldsymbol{x}) = \text{Ber}(y|\sigma(\boldsymbol{w}^T\boldsymbol{x} + b))$$

One major limitation of the above model is linearity with respect to input feature space, in other words:

$$p_\theta(y = 1|\boldsymbol{x}) = \frac{1}{2} \Rightarrow \boldsymbol{w}^T\boldsymbol{x} + b = 0$$

To improve expressivity, we can use feature transformation:

$$p_\theta(y|\boldsymbol{x}) = \text{Ber}(y|\sigma(\boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x}) + b))$$

where $\boldsymbol{h}(\cdot)$ is the feature transformation function.

BLR and Feature Transformation

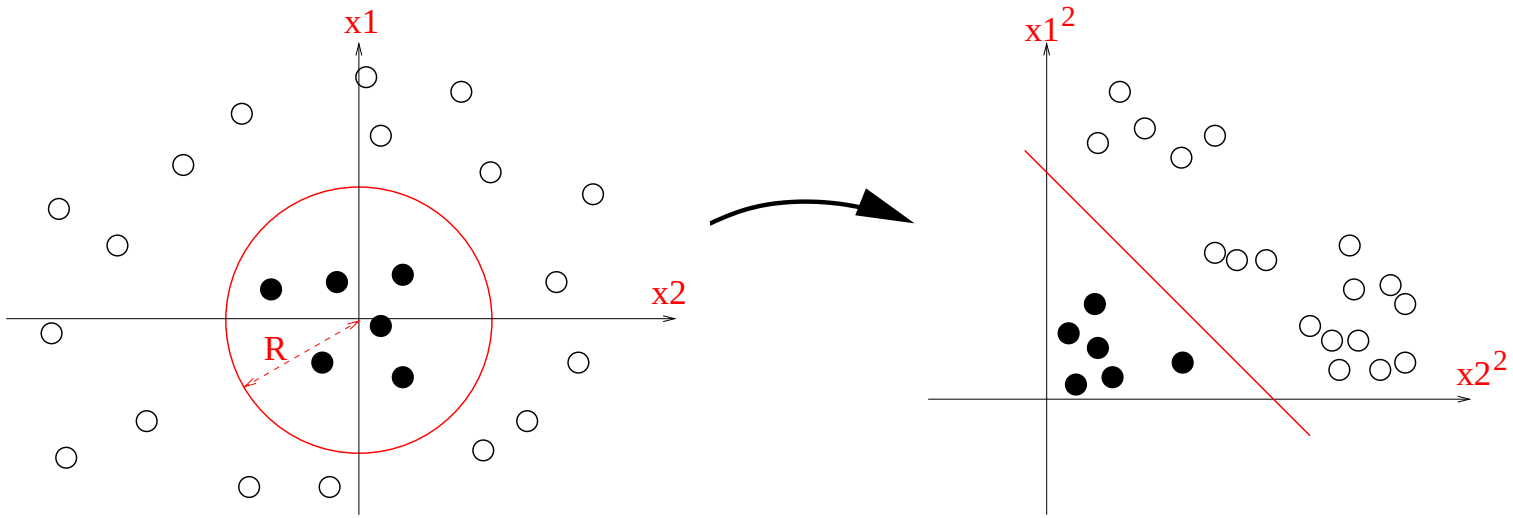


Figure: Original feature space $[x_1, x_2]^T$ (left) and transformed feature space $[x_1^2, x_2^2]^T$ (right) (source: [3]). Note that the decision boundary (red line) in the transformed space is the nonlinear decision boundary (red line) in the original space.

Generative Modeling

Based on the chain rule, we have:

$$p(\boldsymbol{x}) = p(x_1)p(x_2|\boldsymbol{x}_{<2}) \dots p(x_d|\boldsymbol{x}_{<d}) \dots p(x_D|\boldsymbol{x}_{<D})$$

Now assume transformed feature $\boldsymbol{h}_d$ defined as:

$$\boldsymbol{h}_d = \sigma(\boldsymbol{W}_d \boldsymbol{x}_{<d} + \boldsymbol{c}_d), \begin{cases} \boldsymbol{W}_d \in \mathbb{R}^{D_h \times (d-1)} \\ \boldsymbol{c}_d, \boldsymbol{h}_d \in \mathbb{R}^{D_h} \\ \sigma(\cdot) \text{ is arbitrary element-wise nonlinearity} \end{cases}$$

Generative Modeling

Thus for the first term, we have:

$$p_\theta(x_1) = \text{Ber}(x_1|\sigma(b_1)),\ b_1 \in \mathbb{R}$$

And for arbitrary d-th term with $1 < d \leq D$, we have:

$$p_\theta(x_d|\boldsymbol{x}_{<d}) = \text{Ber}(x_i|\sigma(\boldsymbol{v}_d^T \boldsymbol{h}_d + b_d))$$

Task: Density Estimation

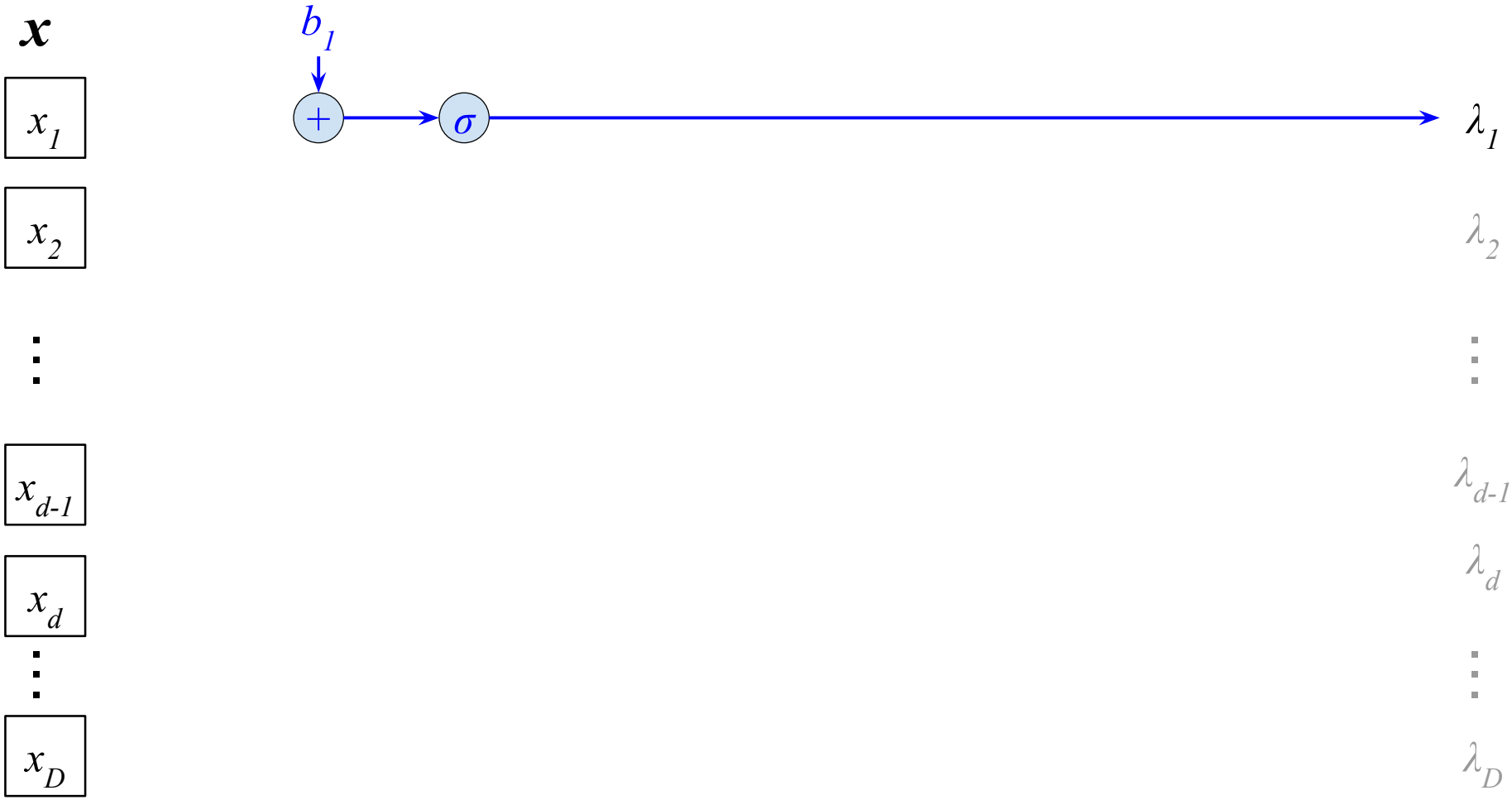


Figure: Calculations for $p_\theta(x_1)$

Task: Density Estimation

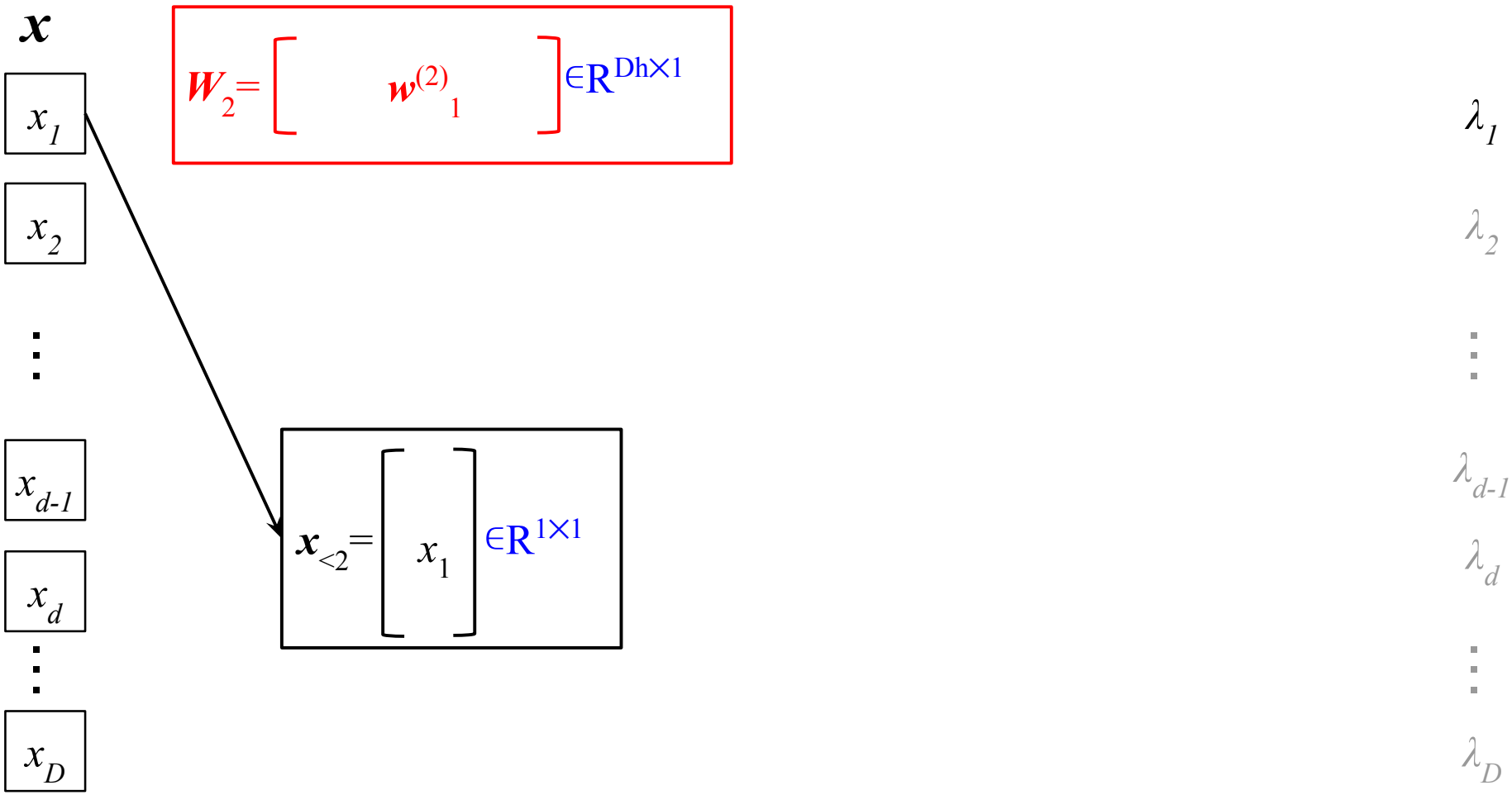


Figure: Calculations for $p_\theta(x_2|x_1)$

Task: Density Estimation

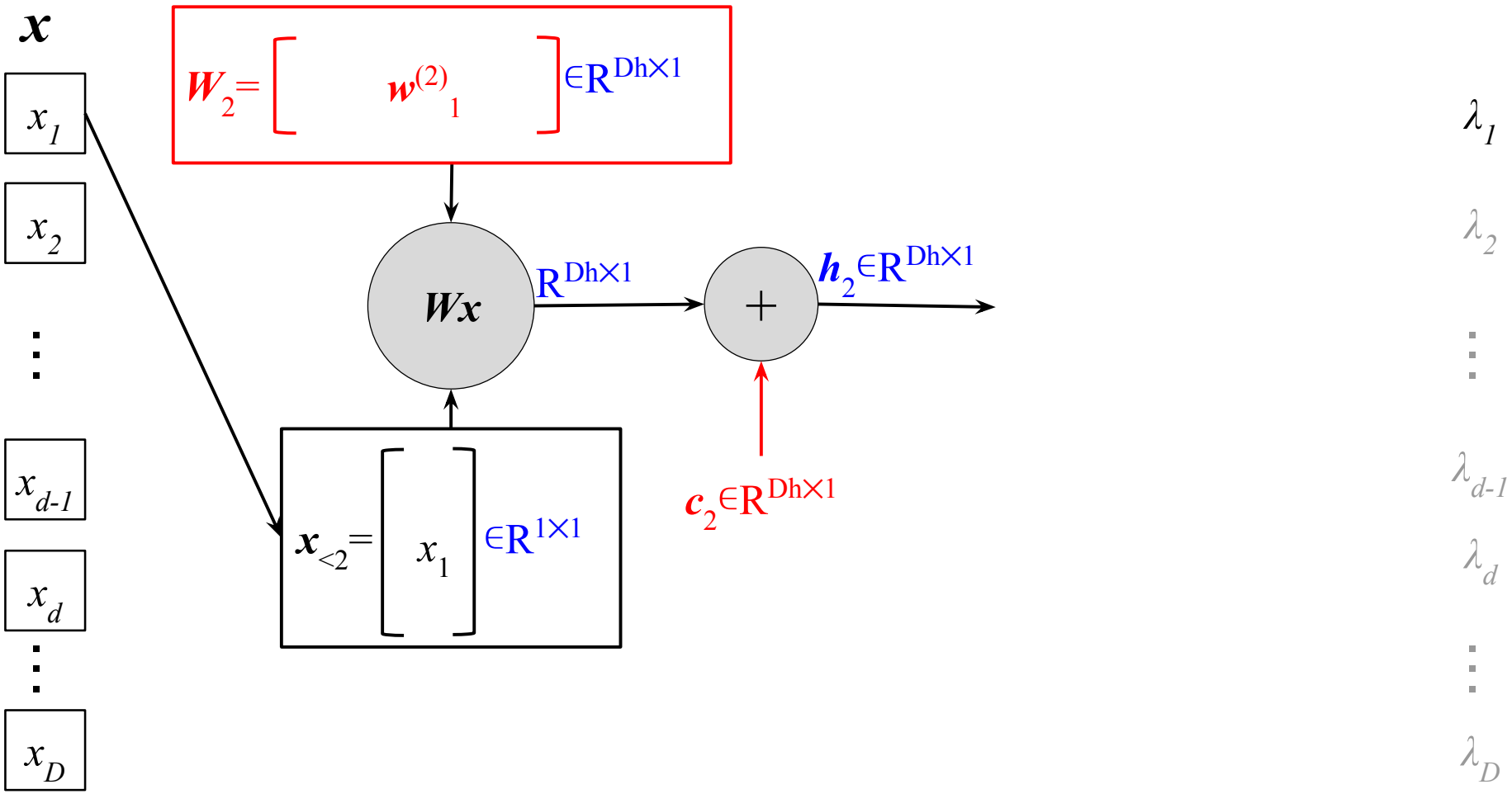


Figure: Calculations for $p_\theta(x_2|x_1)$

Task: Density Estimation

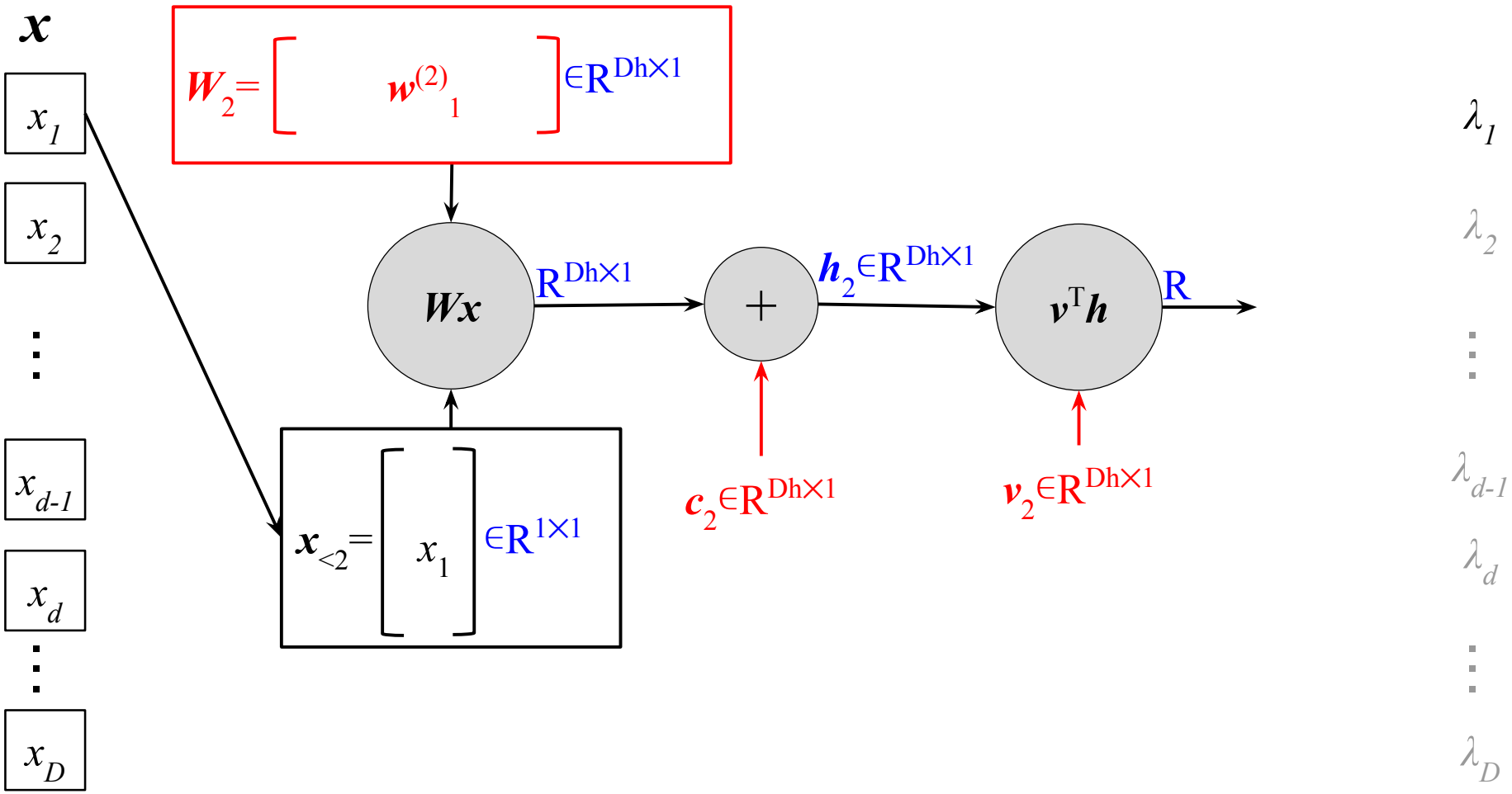


Figure: Calculations for $p_\theta(x_2|x_1)$

Task: Density Estimation

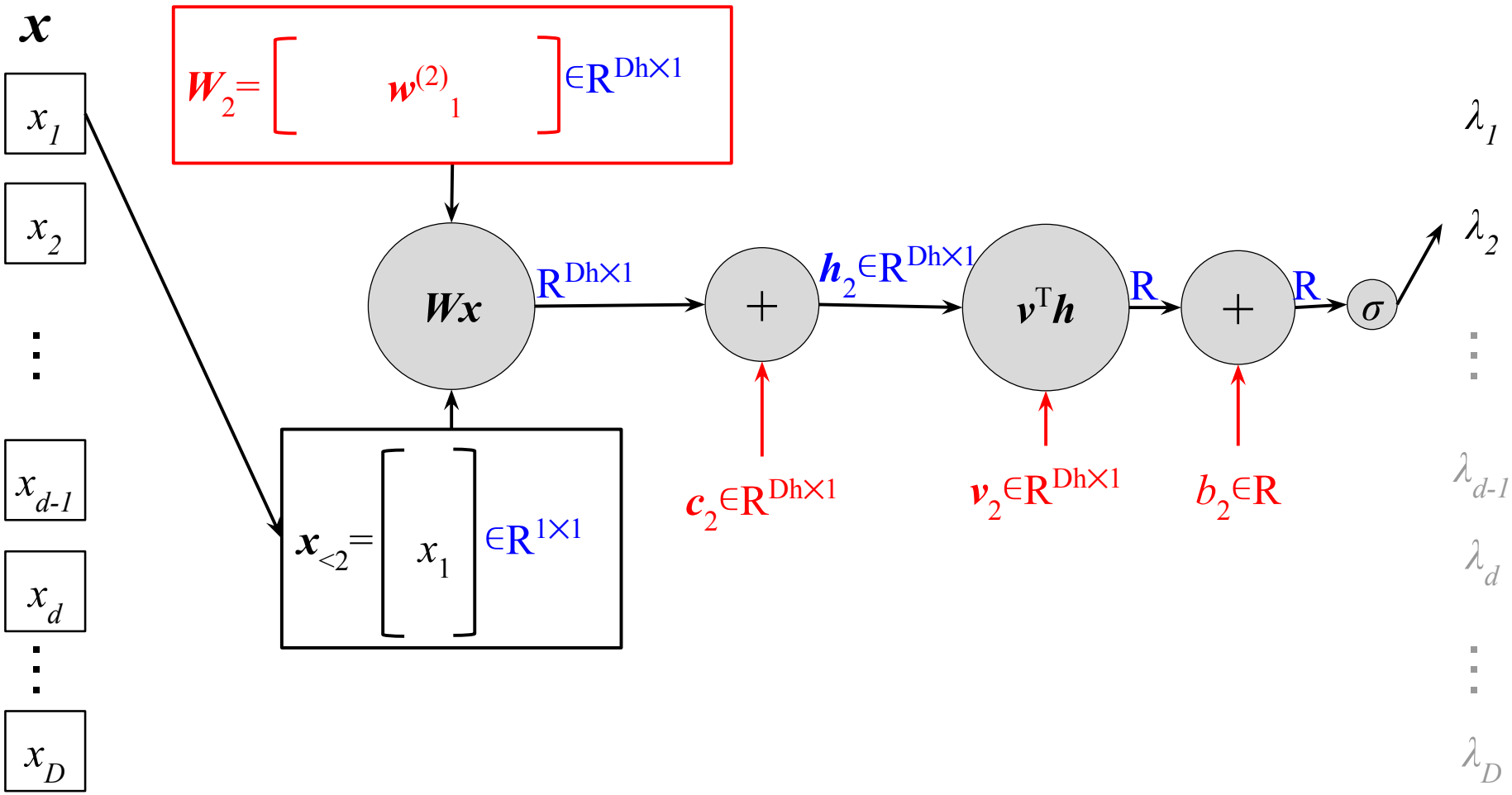


Figure: Calculations for $p_\theta(x_2|x_1)$

Task: Density Estimation

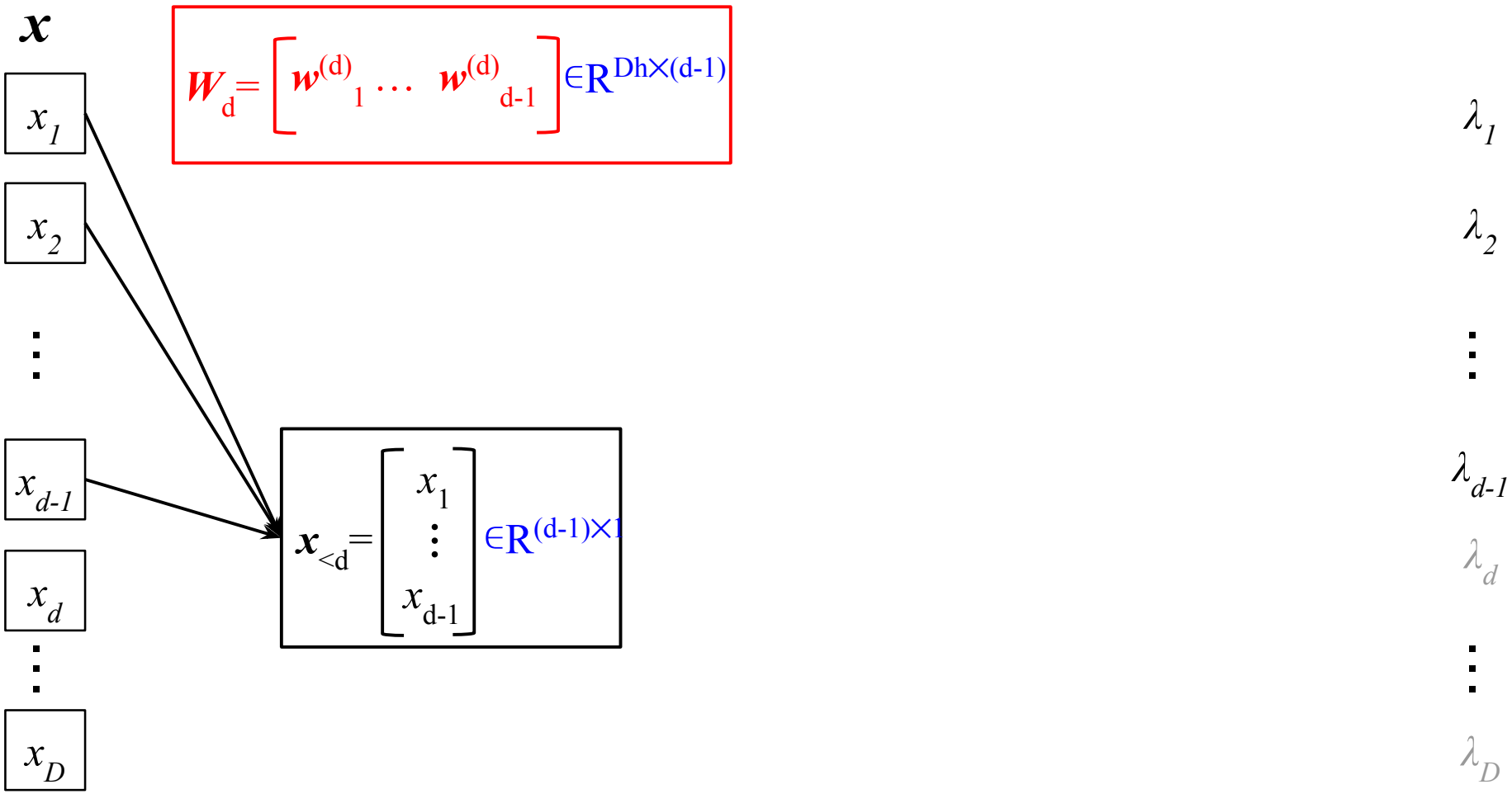


Figure: Calculations for $p_\theta(x_d|x_1, \dots x_{d-1})$

Task: Density Estimation

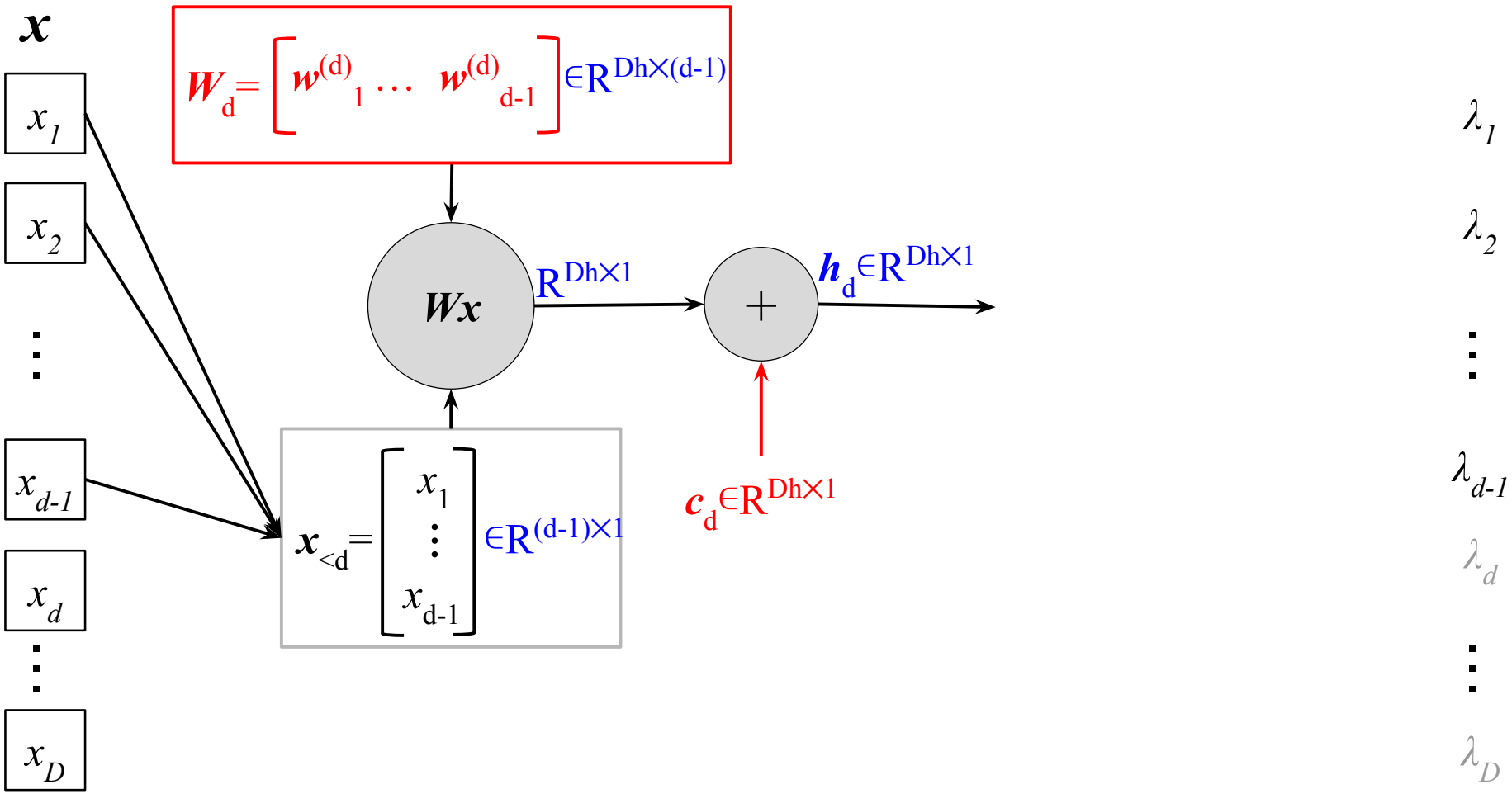


Figure: Calculations for $p_\theta(x_d|x_1, \ldots x_{d-1})$

Task: Density Estimation

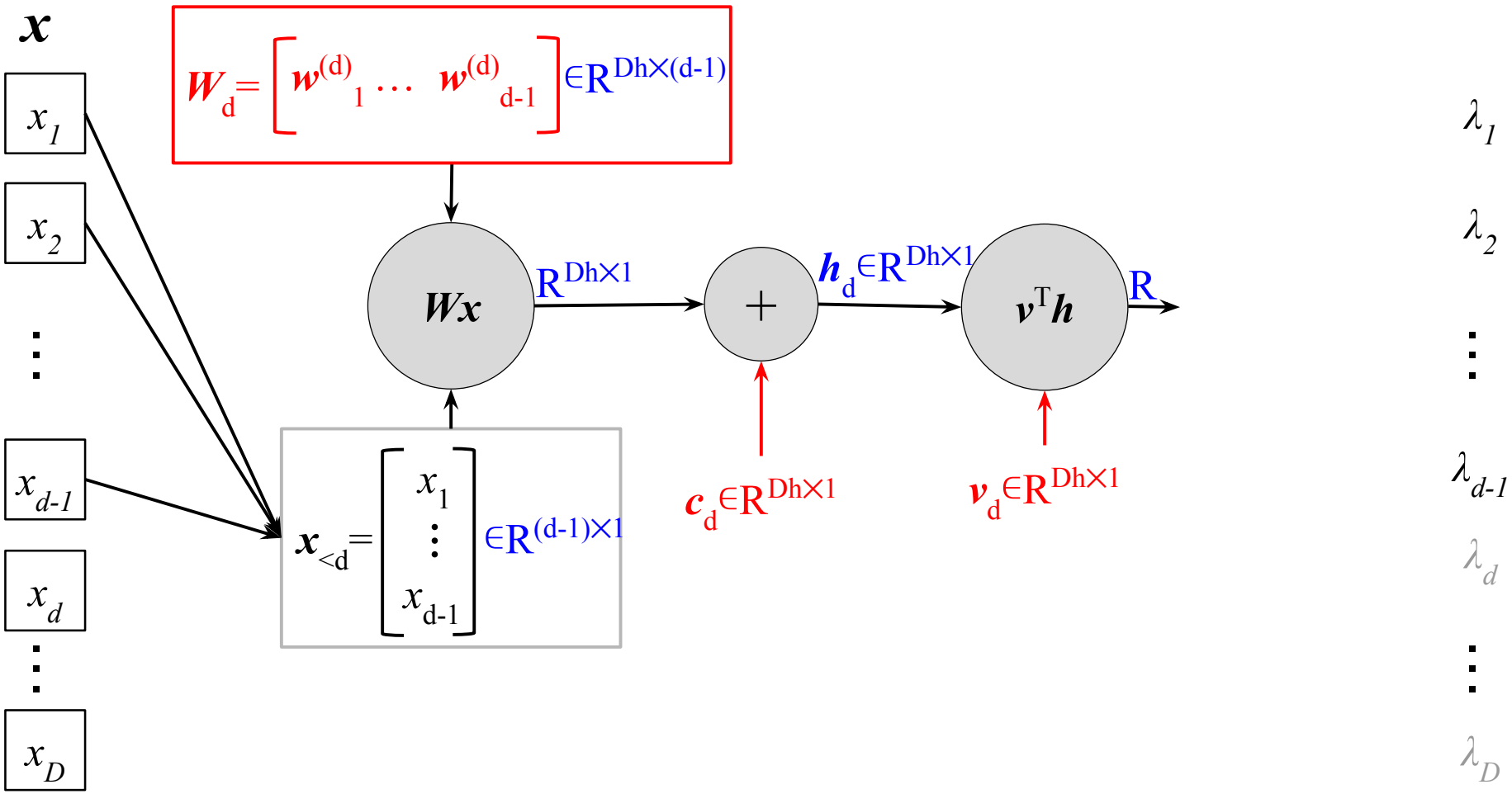


Figure: Calculations for $p_\theta(x_d|x_1, \dots x_{d-1})$

Task: Density Estimation

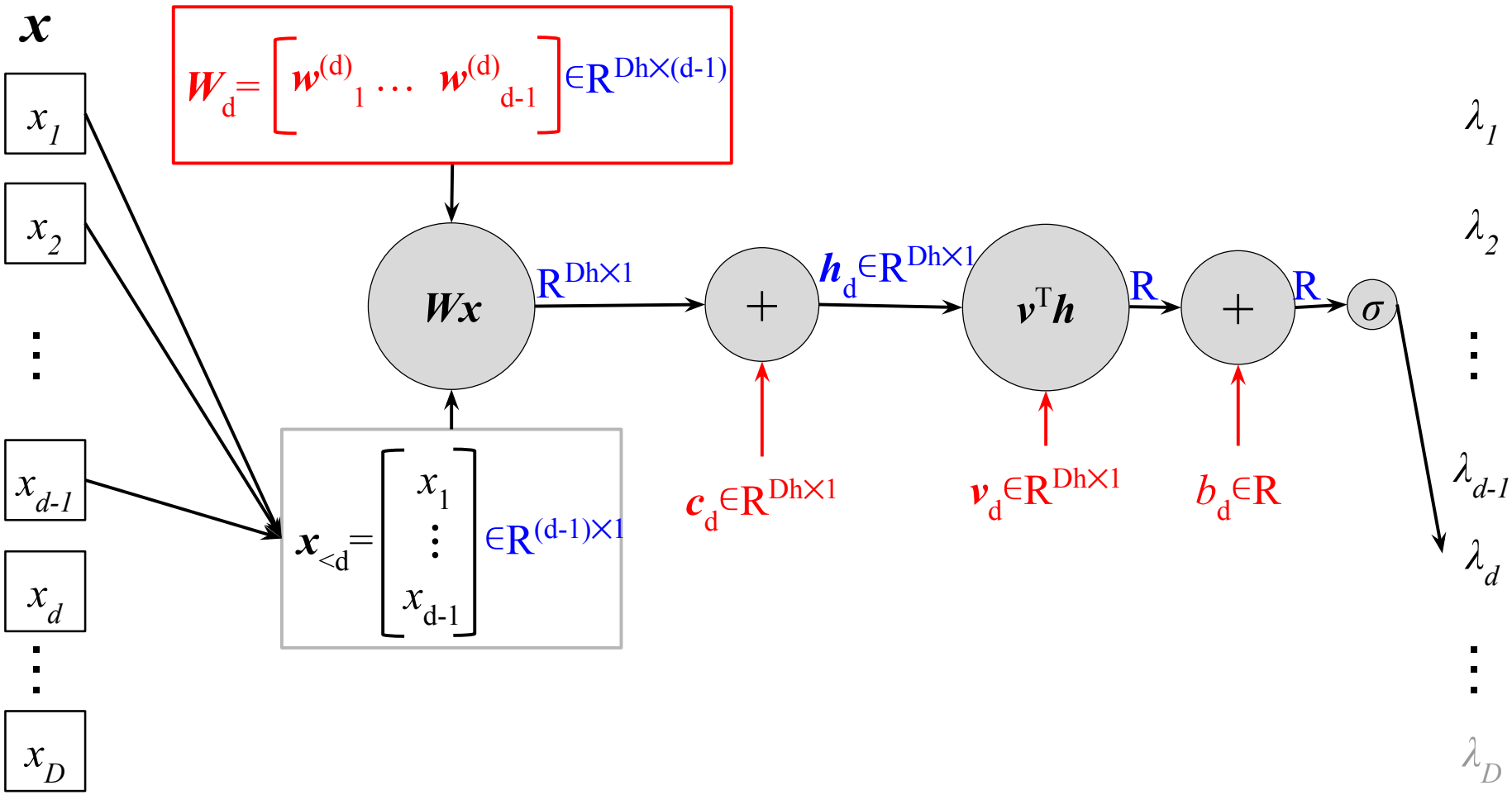


Figure: Calculations for $p_\theta(x_d|x_1, \dots x_{d-1})$

Task: Density Estimation

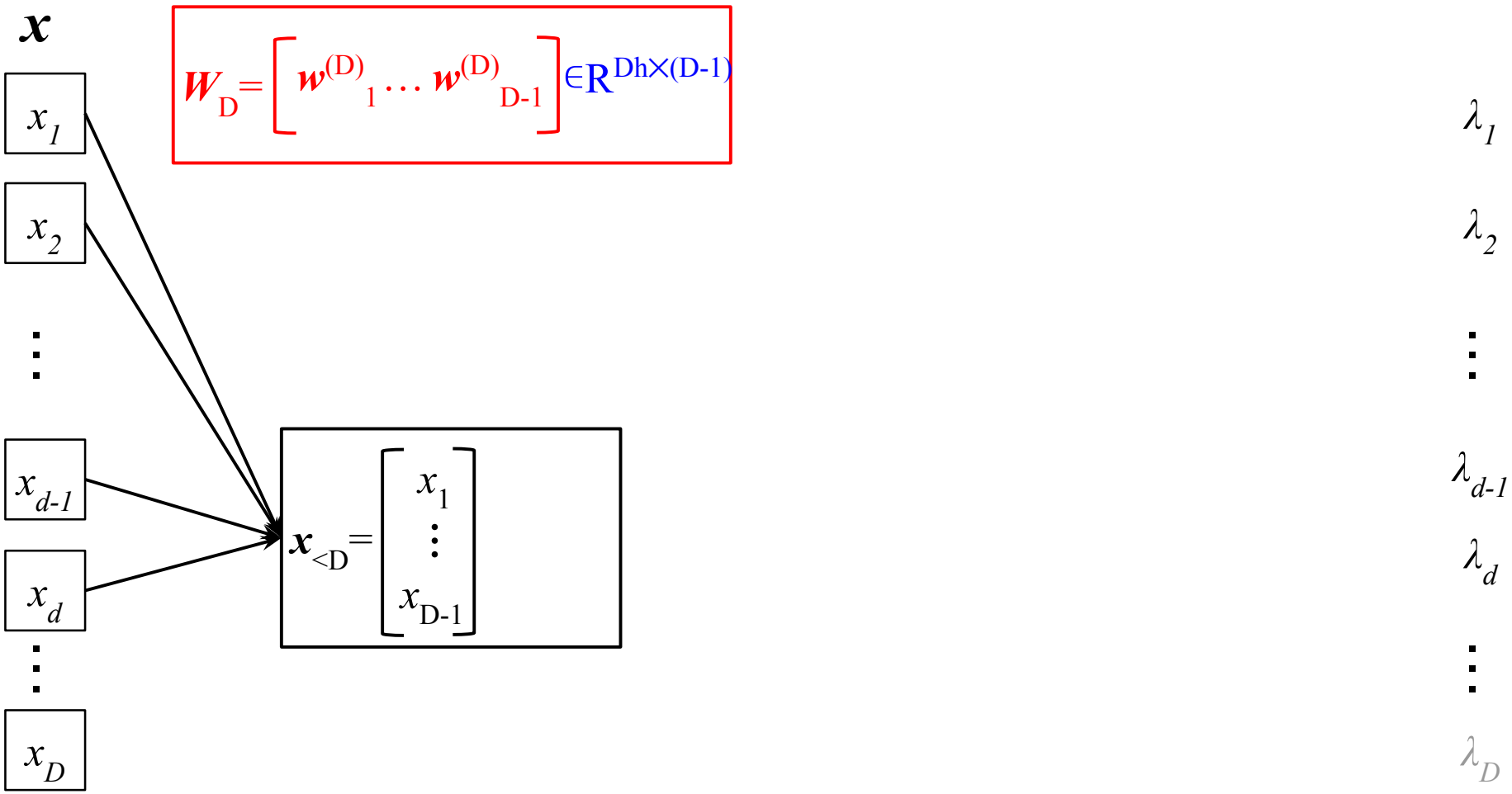


Figure: Calculations for $p_\theta(x_D|x_1, \dots x_{D-1})$

Task: Density Estimation

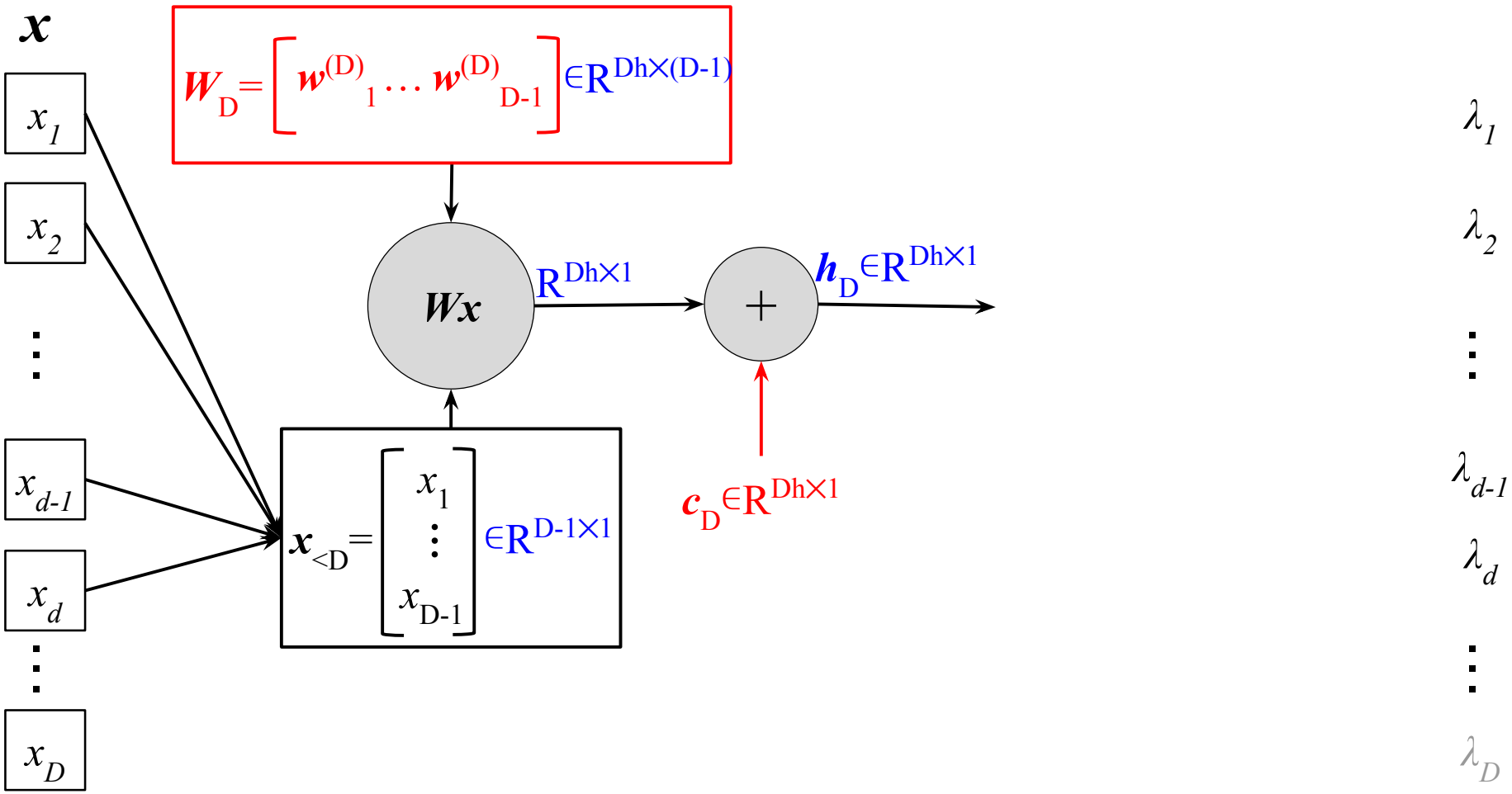


Figure: Calculations for $p_\theta(x_D|x_1, \dots x_{D-1})$

Task: Density Estimation

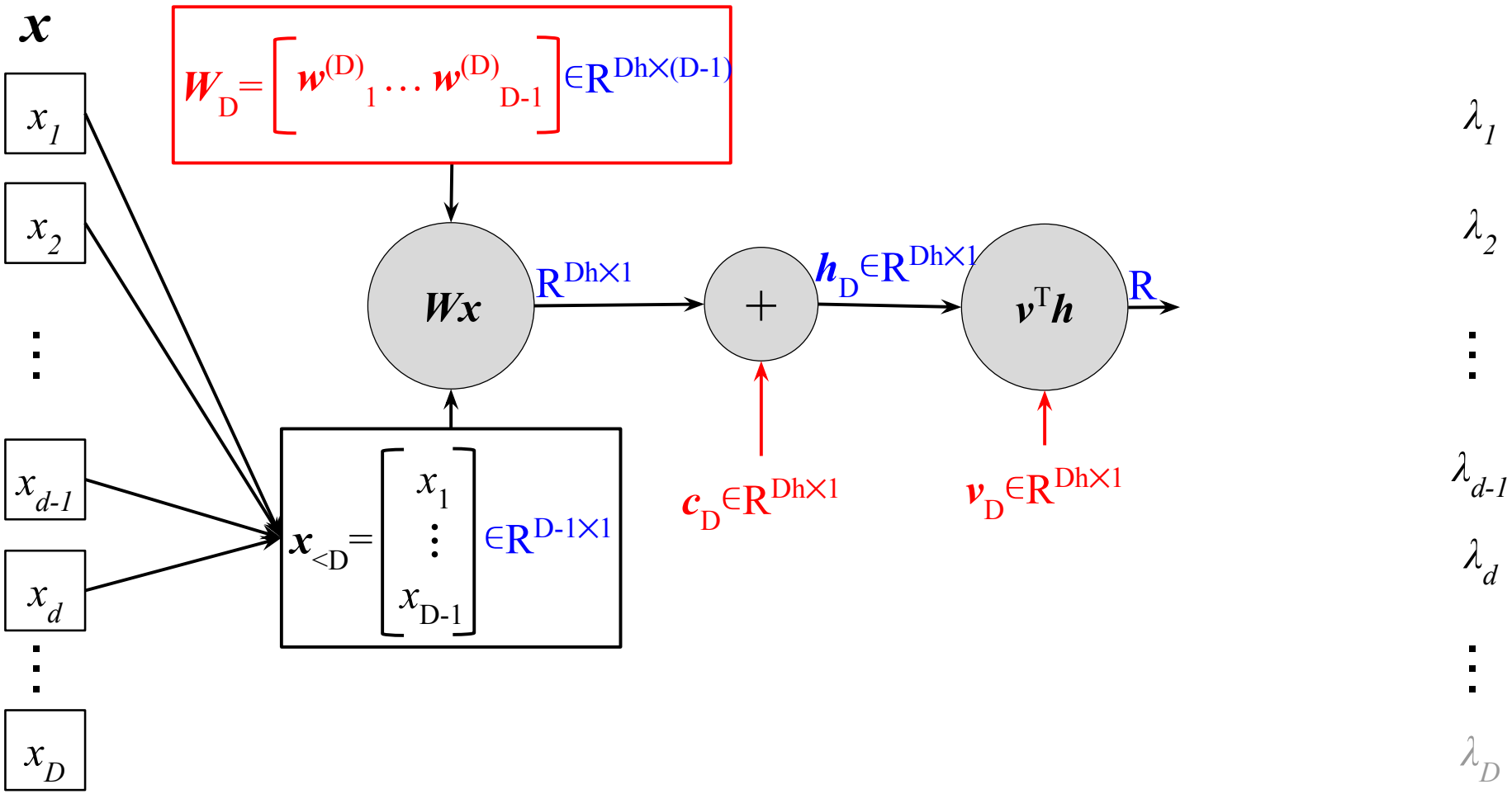


Figure: Calculations for $p_\theta(x_D|x_1, \dots x_{D-1})$

Task: Density Estimation

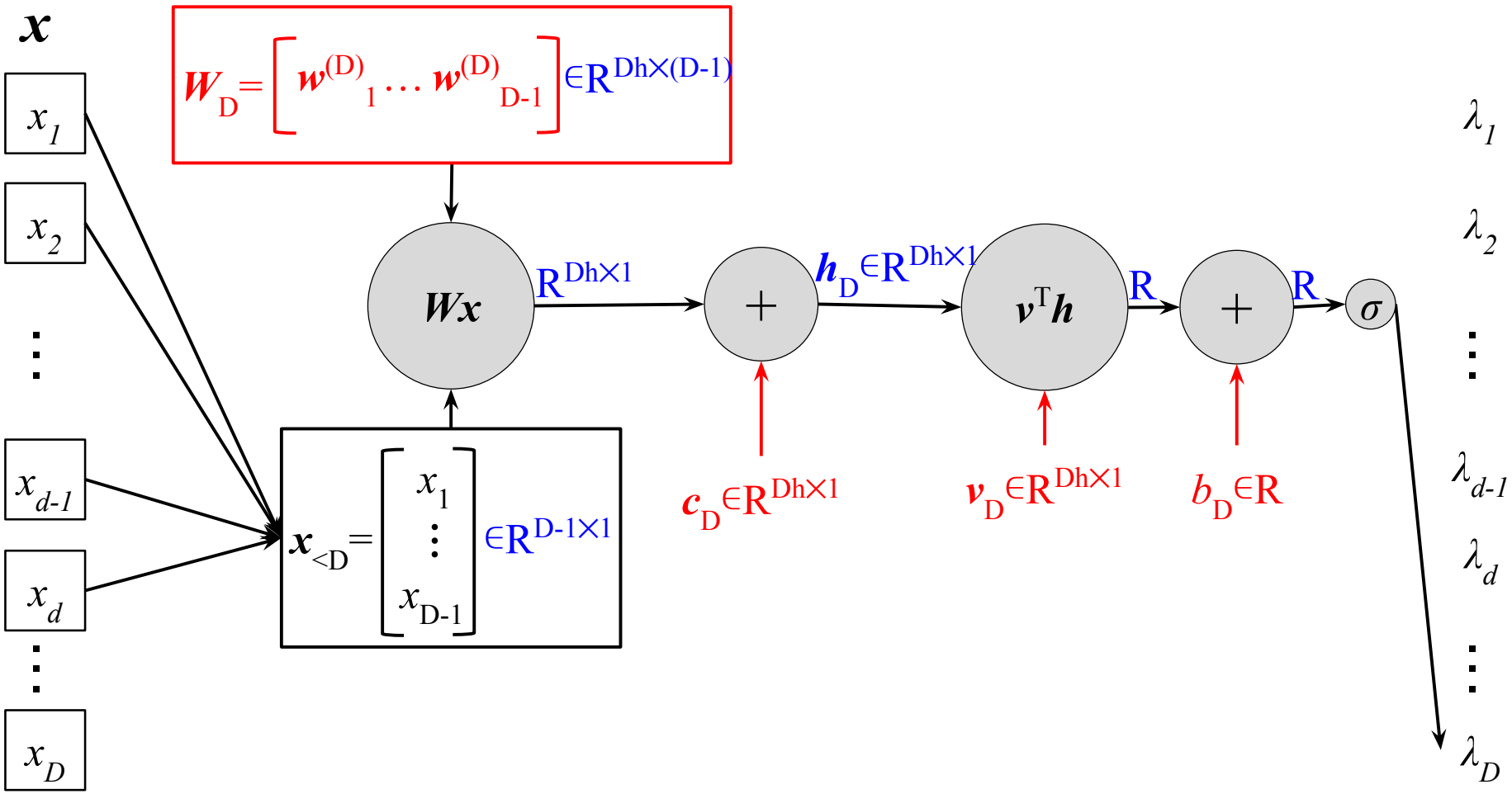


Figure: Calculations for $p_\theta(x_D|x_1, \dots x_{D-1})$

Weight Sharing

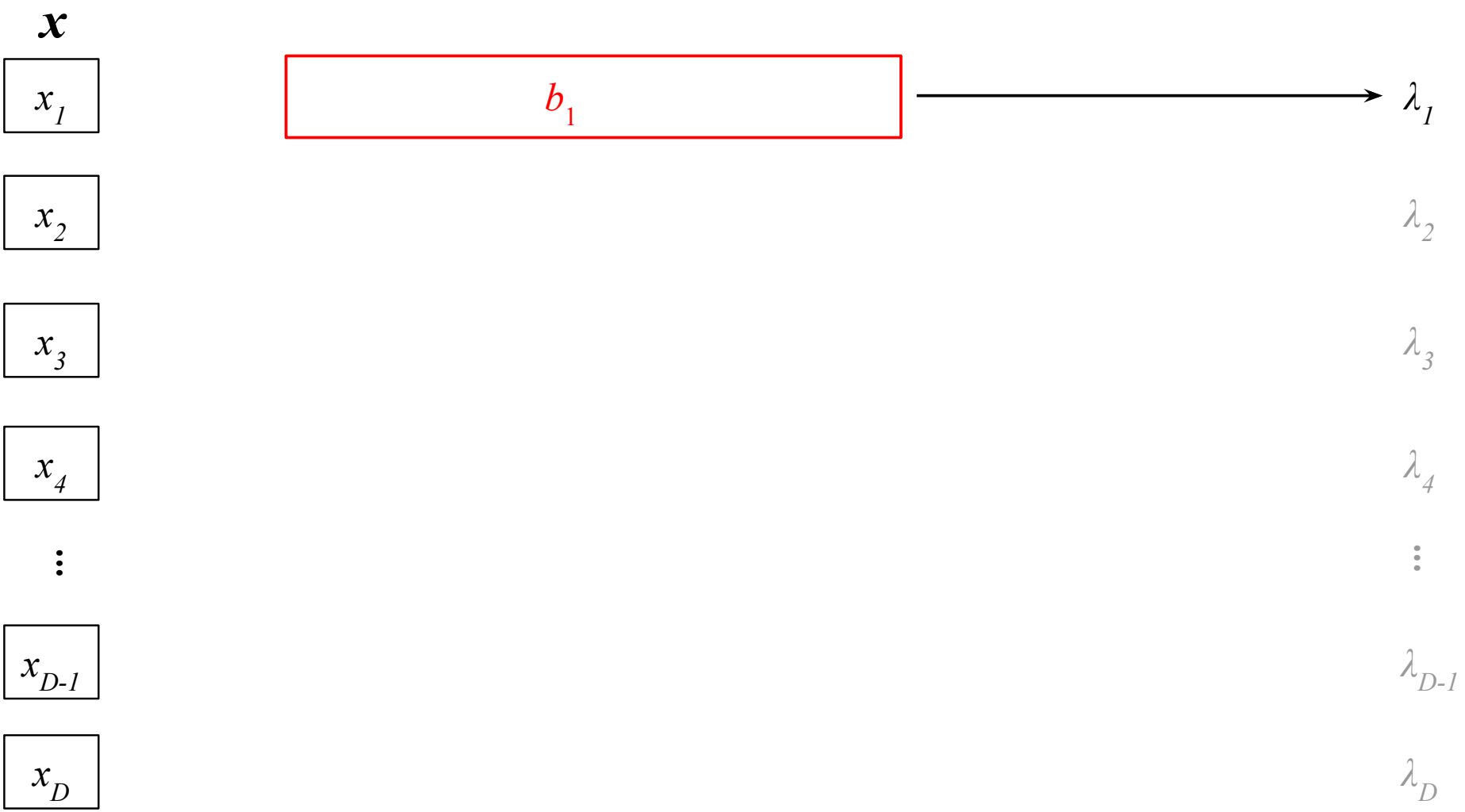


Figure: Calculations for $p_\theta(x_1)$

Weight Sharing

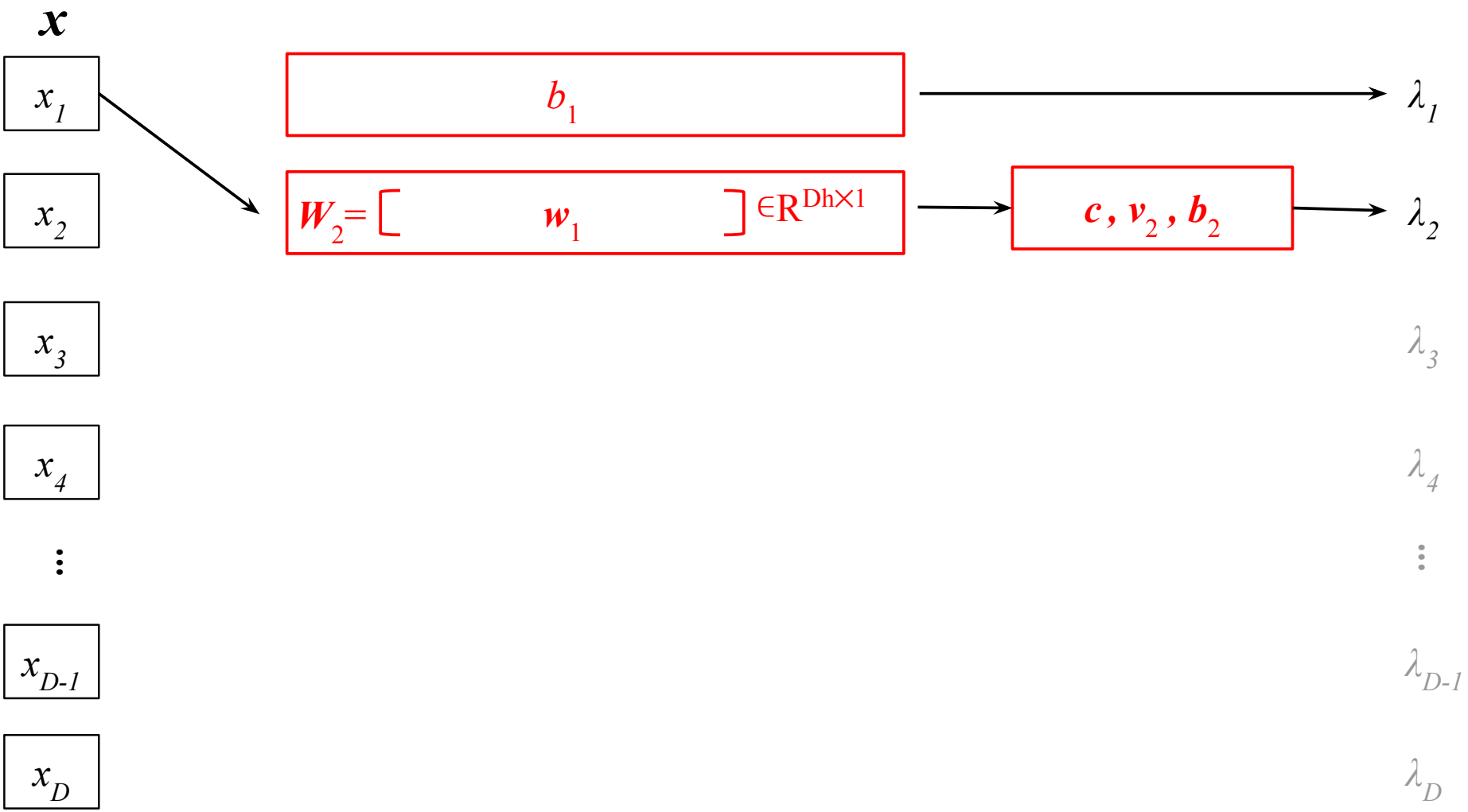


Figure: Calculations for $p_\theta(x_2|x_1)$

Weight Sharing

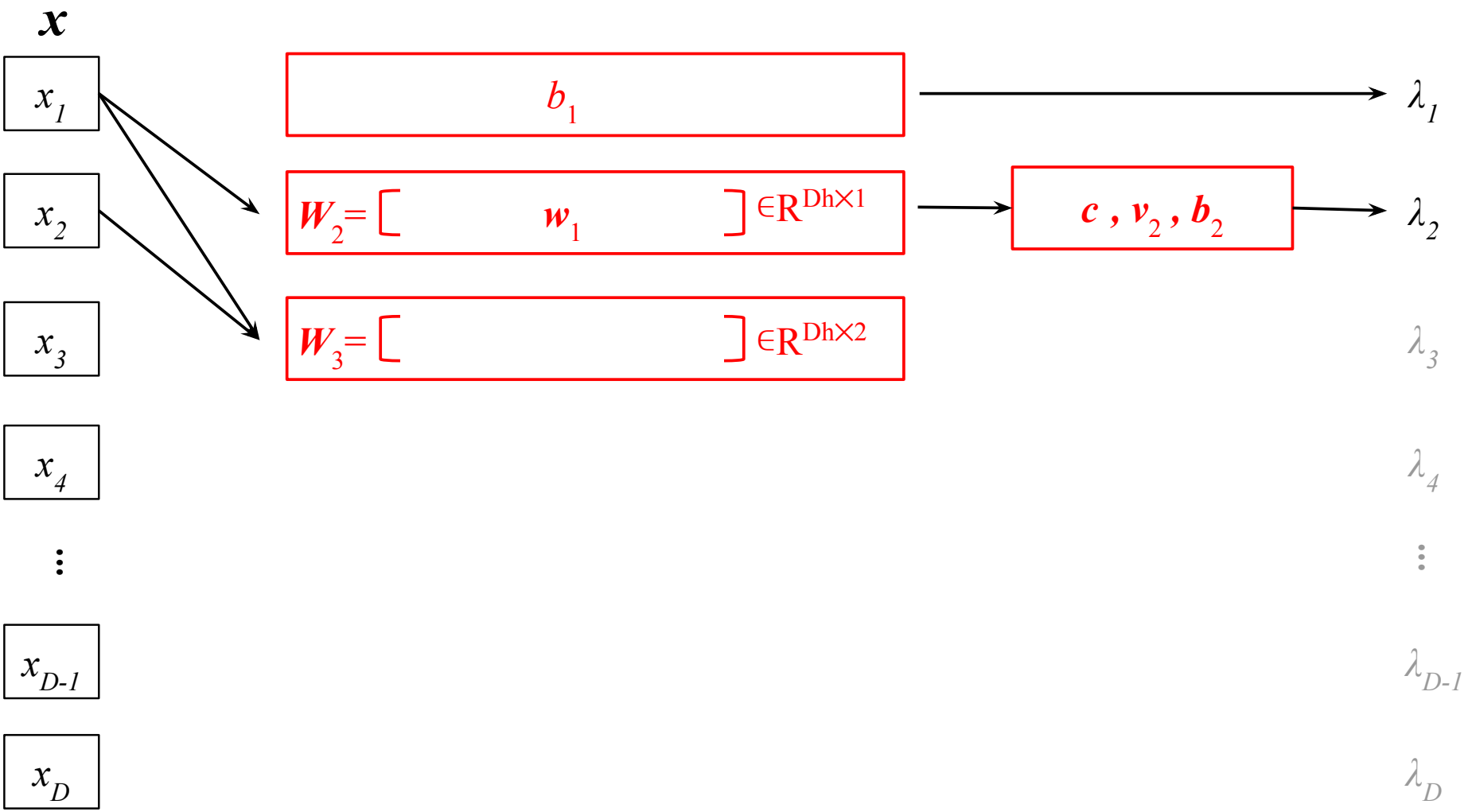


Figure: Calculations for $p_\theta(x_3|x_1, x_2)$

Weight Sharing

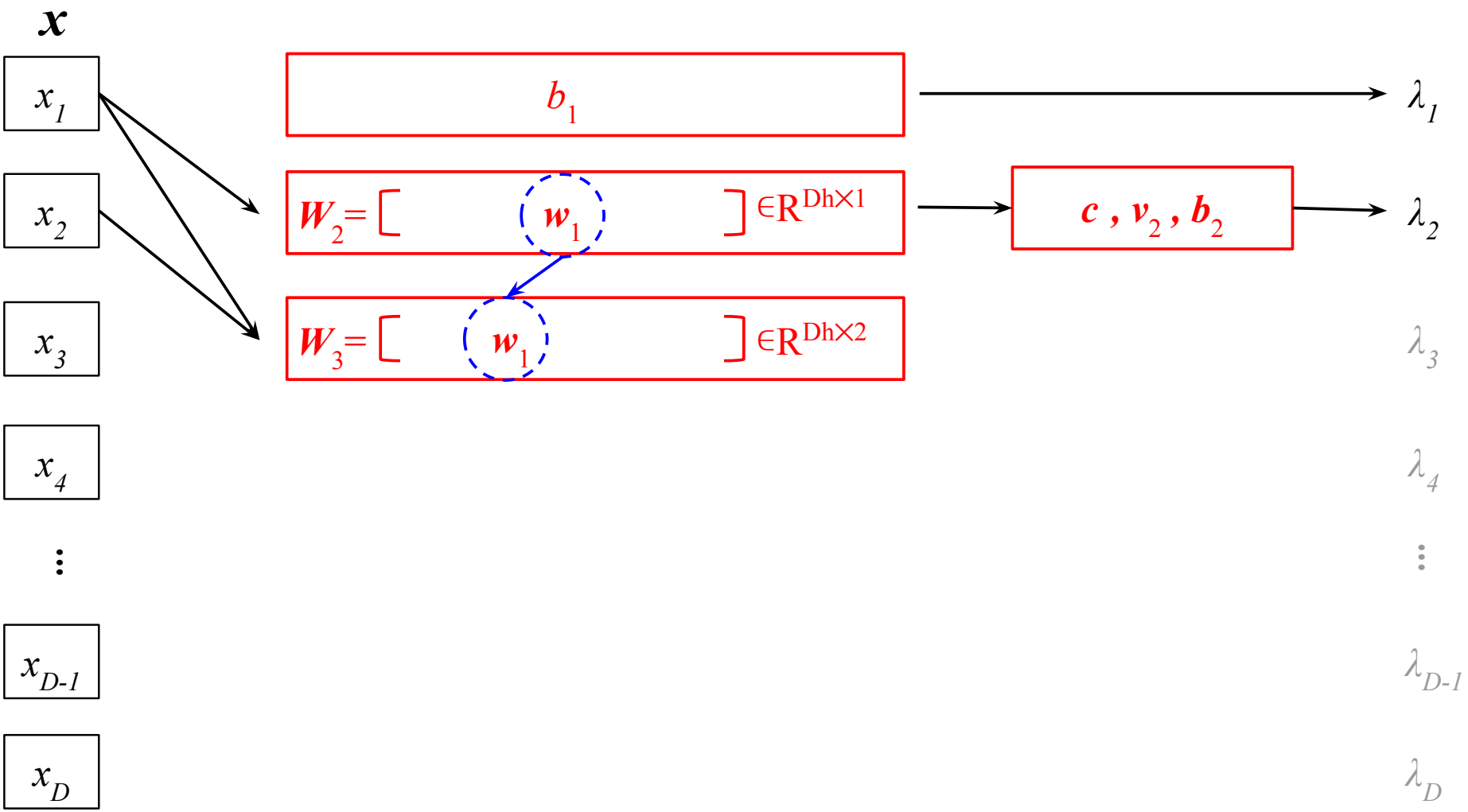


Figure: Calculations for $p_\theta(x_3|x_1, x_2)$

Weight Sharing

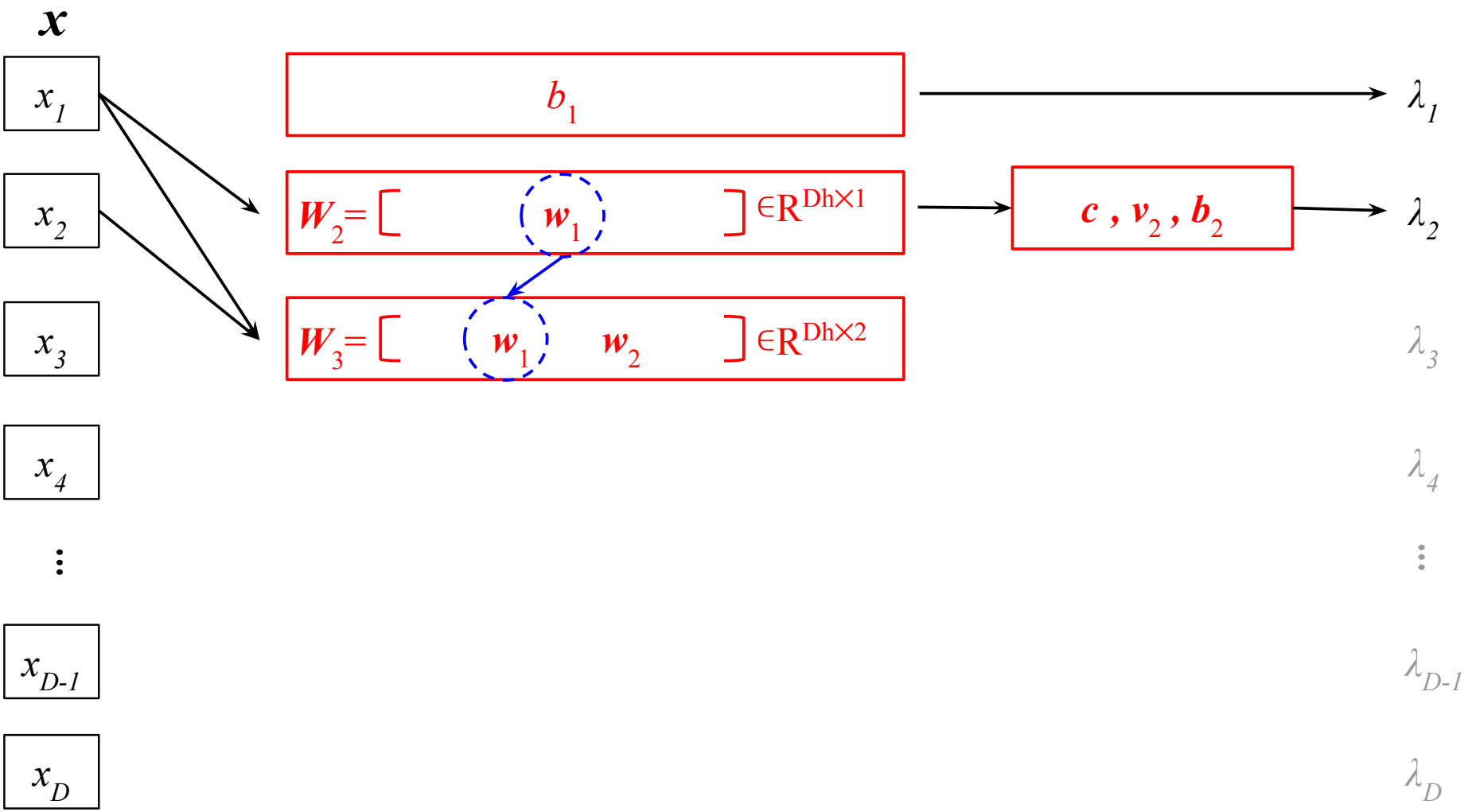


Figure: Calculations for $p_\theta(x_3|x_1, x_2)$

Weight Sharing

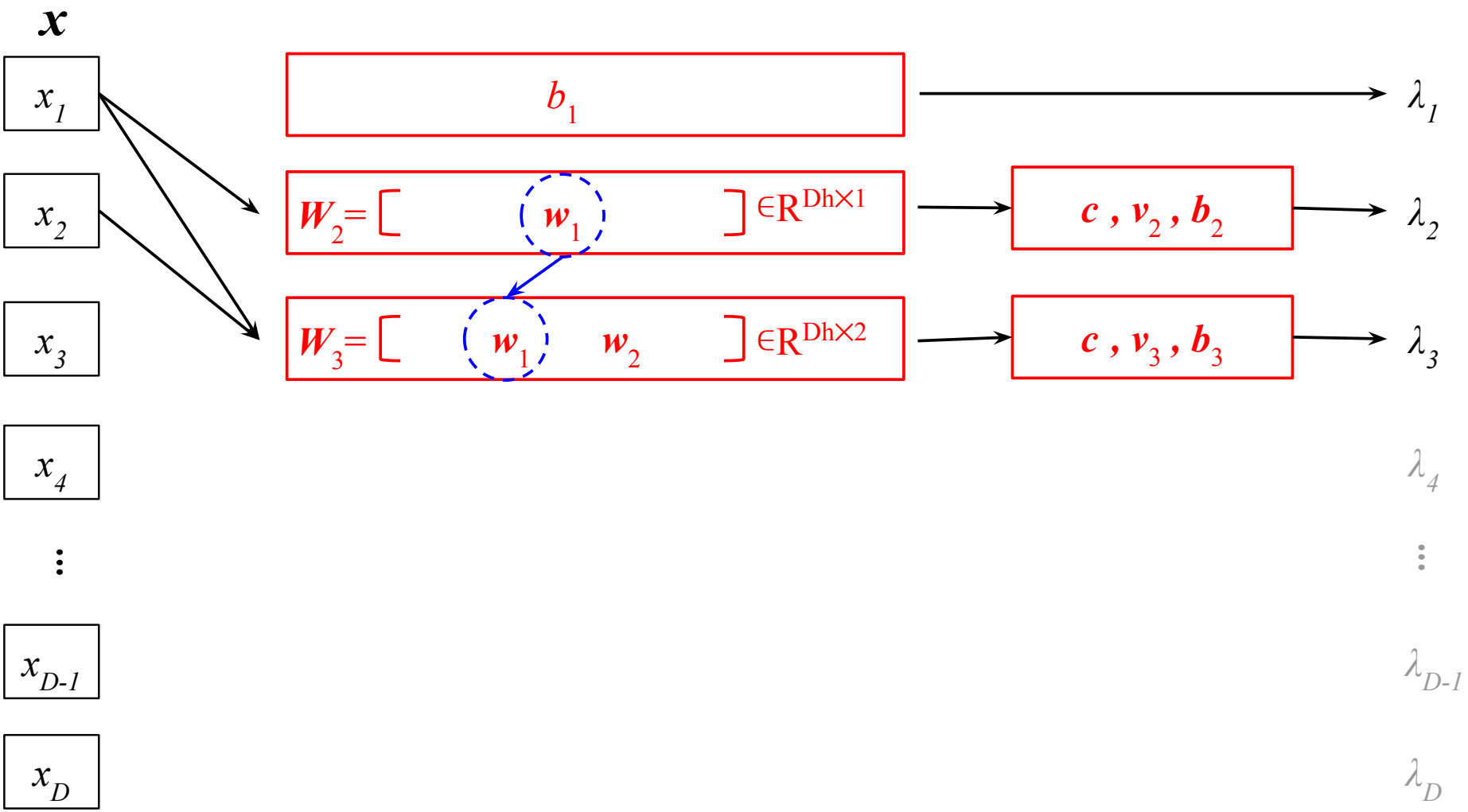


Figure: Calculations for $p_\theta(x_3|x_1, x_2)$

Weight Sharing

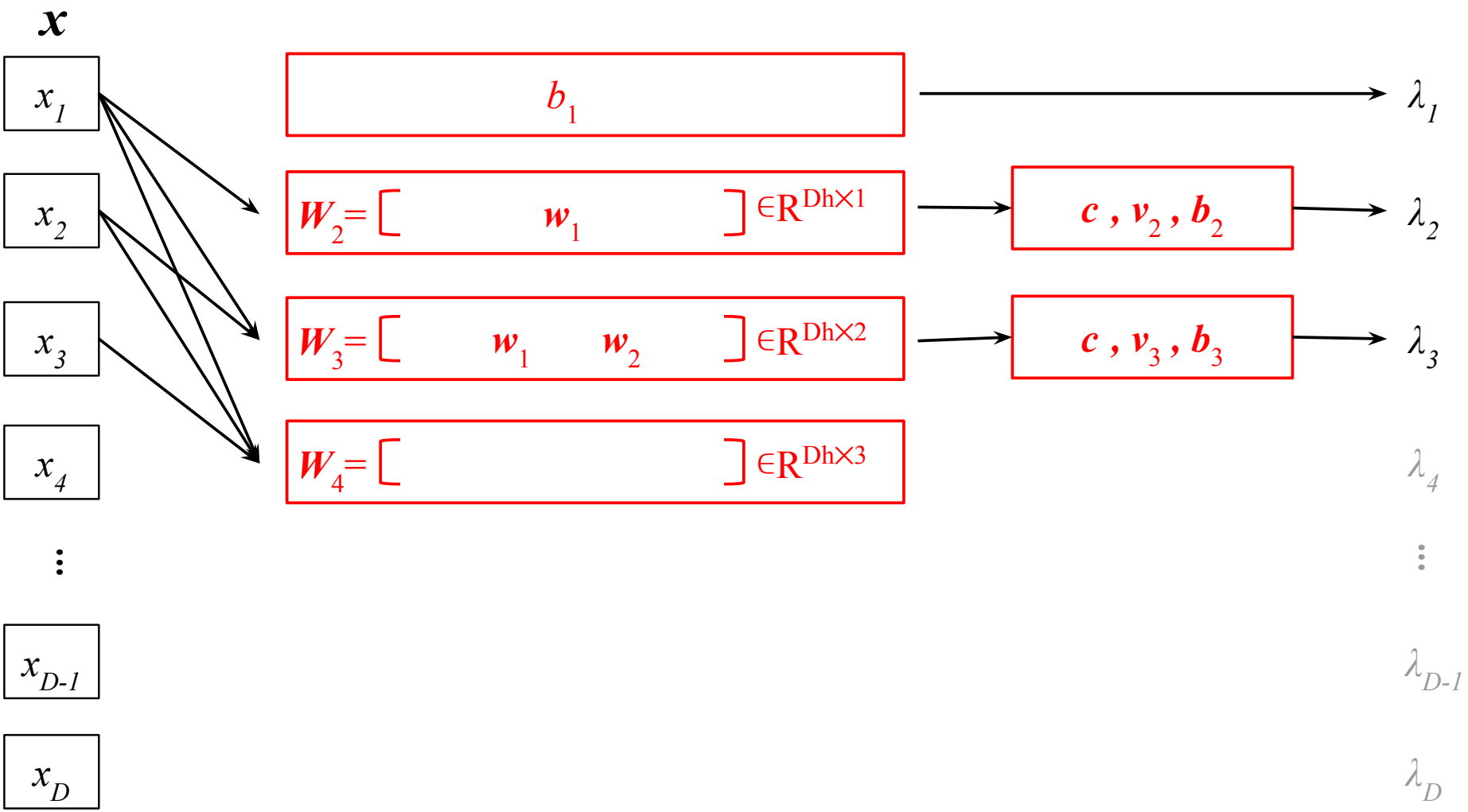


Figure: Calculations for $p_\theta(x_4|x_1, x_2, x_3)$

Weight Sharing

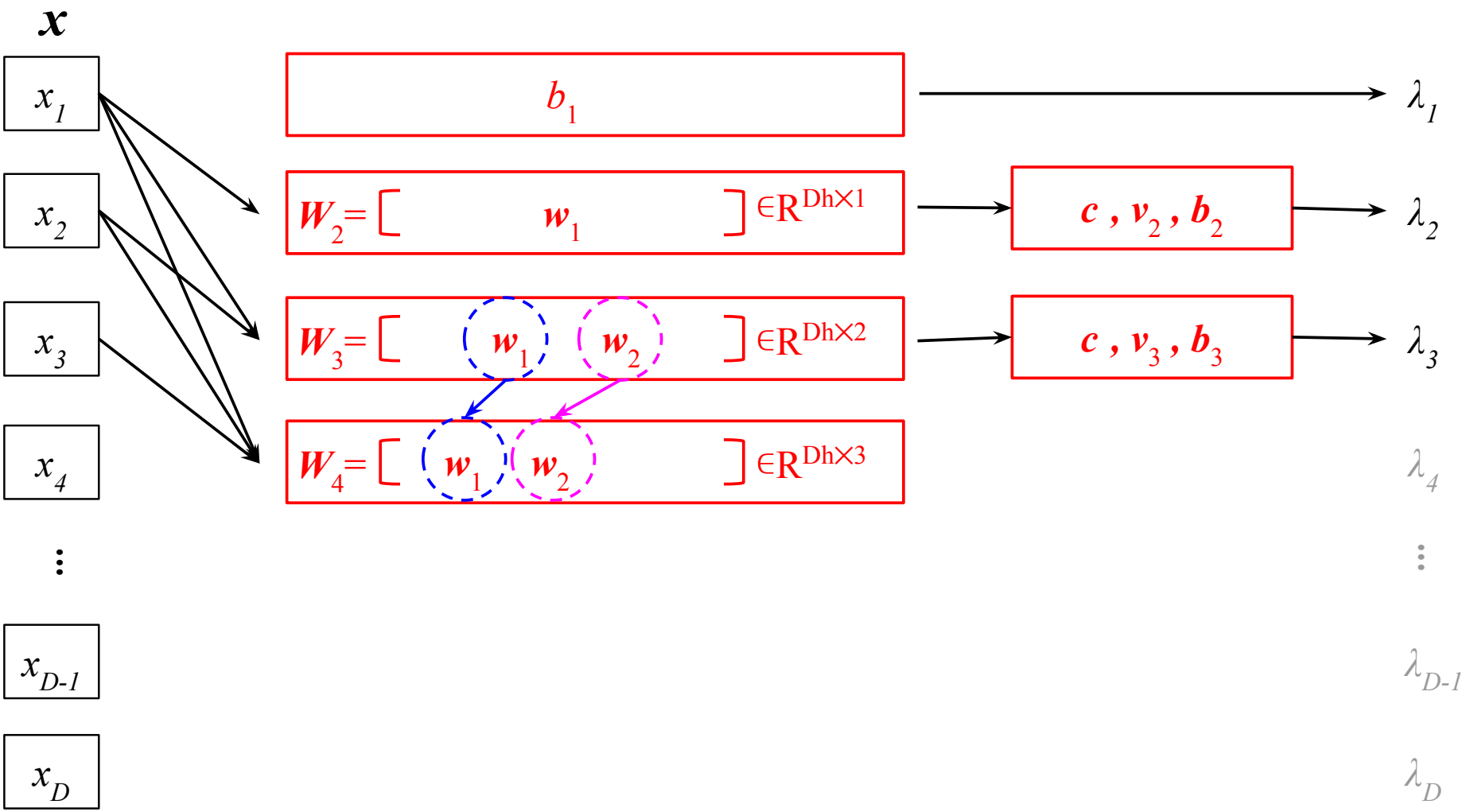


Figure: Calculations for $p_\theta(x_4|x_1, x_2, x_3)$

Weight Sharing

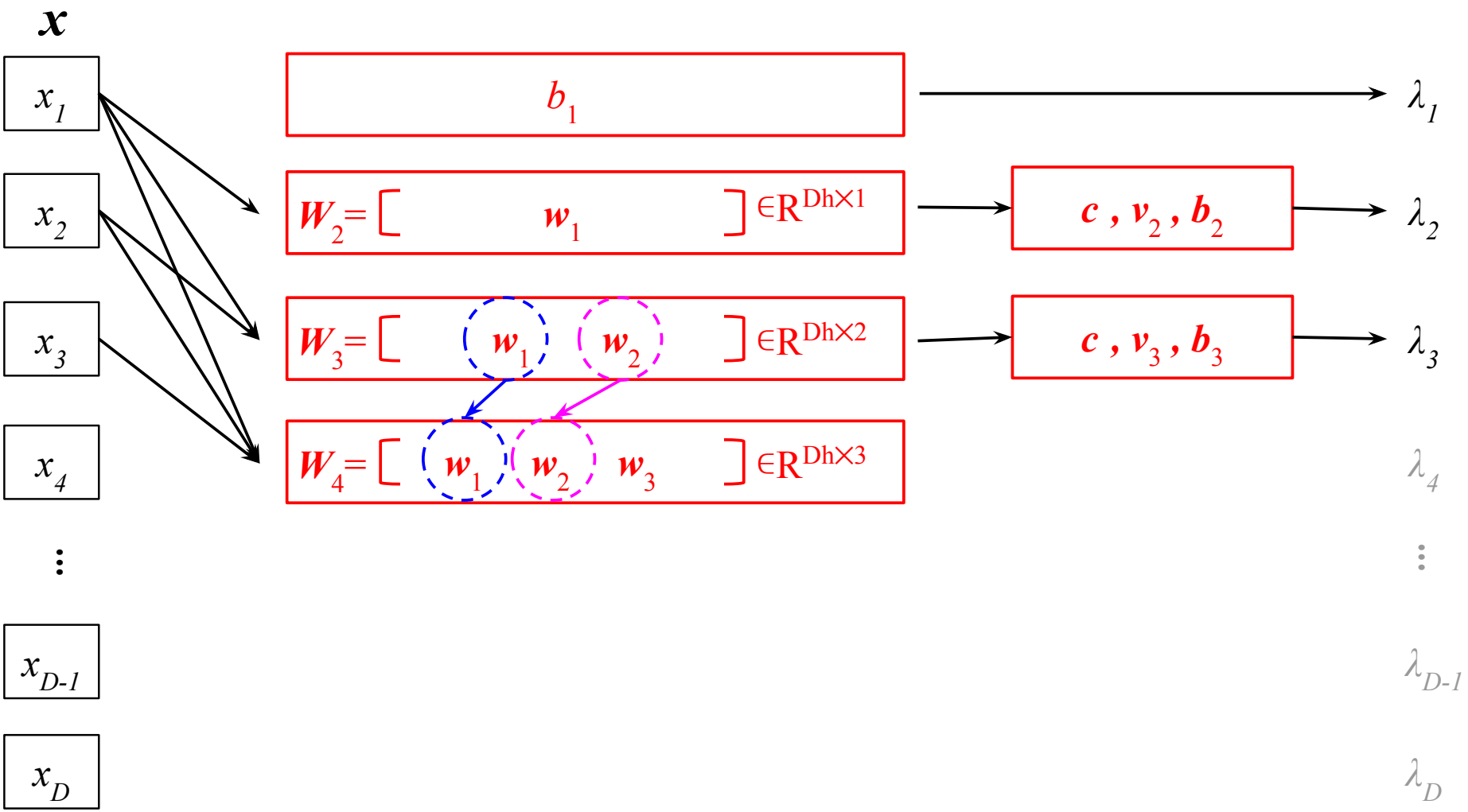


Figure: Calculations for $p_\theta(x_4|x_1, x_2, x_3)$

Weight Sharing

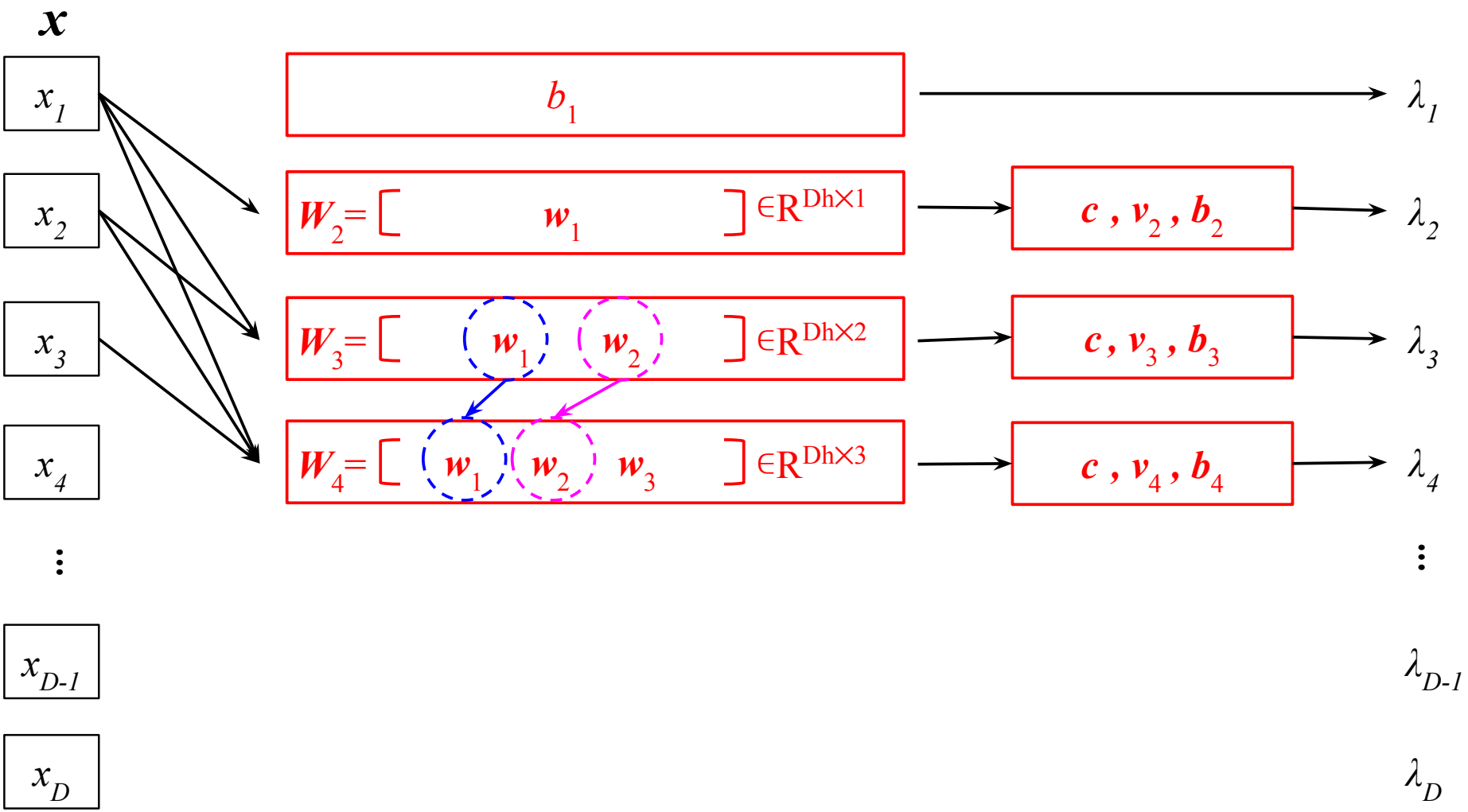


Figure: Calculations for $p_\theta(x_4|x_1, x_2, x_3)$

Weight Sharing

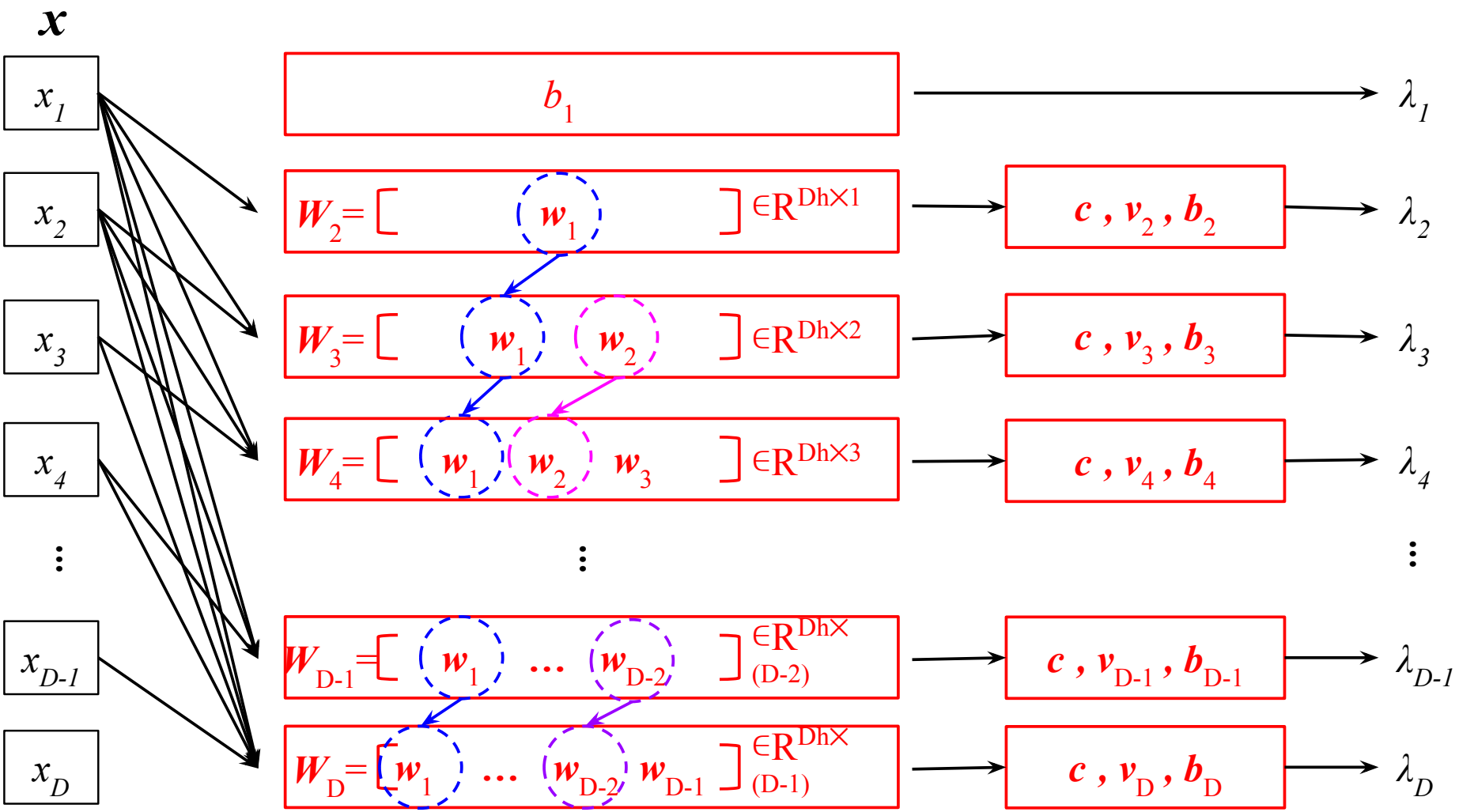


Figure: Calculations for $p_\theta(\boldsymbol{x})$

Weight Sharing

To reduce the number of parameters and speed up computations, the weights for generating hidden representation are shared as follows:

$$\boldsymbol{h}_2 = \sigma\left(\overbrace{\begin{bmatrix}\boldsymbol{w}_1\end{bmatrix}}^{\boldsymbol{W}_2}\overbrace{\begin{bmatrix}x_1\end{bmatrix}}^{\boldsymbol{x}_{<2}} + \boldsymbol{c}\right), \boldsymbol{h}_3 = \sigma\left(\overbrace{\begin{bmatrix}\boldsymbol{w}_1 & \boldsymbol{w}_2\end{bmatrix}}^{\boldsymbol{W}_3}\overbrace{\begin{bmatrix}x_1\\ x_2\end{bmatrix}}^{\boldsymbol{x}_{<3}} + \boldsymbol{c}\right)$$

$$\boldsymbol{h}_D = \sigma\left(\overbrace{\begin{bmatrix}\boldsymbol{w}_1 & \dots & \boldsymbol{w}_{D-1}\end{bmatrix}}^{\boldsymbol{W}_D}\overbrace{\begin{bmatrix}x_1\\ \dots\\ x_{D-1}\end{bmatrix}}^{\boldsymbol{x}_{<D}} + \boldsymbol{c}\right)$$

Thus all the weights needed to calculate $\boldsymbol{h}_2, \dots, \boldsymbol{h}_D$ are available in $\boldsymbol{W}_D$ which we represent by $\boldsymbol{W}$ and also $\boldsymbol{c}_1 = \boldsymbol{c}_2 = \dots = \boldsymbol{c}_D = \boldsymbol{c}$. So we can write:

$$\boldsymbol{h}_d = \sigma(\boldsymbol{W}_{:<d}\, \boldsymbol{x}_{<d} + \boldsymbol{c})$$

Number of Parameters

The total number of parameters in NADE is :

$$\left.\begin{array}{r} \left.\begin{array}{r} \boldsymbol{W} \in \mathbb{R}^{D_h \times (D-1)} \\ \\ \boldsymbol{c} \in \mathbb{R}^{D_h} \end{array}\right\} \Rightarrow D \times D_h \\ \\ \left.\begin{array}{r} \{\boldsymbol{v}_d \in \mathbb{R}^{D_h}\}_{d=2}^{D} \\ \\ \{b_d \in \mathbb{R}\}_{d=1}^{D} \end{array}\right\} \Rightarrow (D-1)D_h + D \end{array}\right\} \Rightarrow D \times D_h + (D-1)D_h + D$$

Density Estimation and Generation

Similar to FVSBN, one can generate samples from $p_\theta(\boldsymbol{x})$ assuming we have access to model parameters.

NADE

(a) Generated samples

Real NADE

Generative Modeling

As usual, based on the chain rule, we have:

$$p_\theta(\boldsymbol{x}) = p_\theta(x_1)p_\theta(x_2|\boldsymbol{x}_{<2})\dots p_\theta(x_d|\boldsymbol{x}_{<d})\dots p_\theta(x_D|\boldsymbol{x}_{<D})$$

Now assume all the dimensions are real-valued numbers, then the modeling is:

$$p_\theta(x_1) = \mathcal{N}(x_1|\mu_1, \exp(s_1))$$

$$\vdots$$

$$p_\theta(x_d|\boldsymbol{x}_{<d}) = \mathcal{N}(x_d|\boldsymbol{v}_d^T\boldsymbol{h}_d + b_d, \exp(\boldsymbol{u}_d^T\boldsymbol{h}_d + d_d))$$

$$\begin{cases} \boldsymbol{W} \in \mathbb{R}^{D_h \times (D-1)} \\ \boldsymbol{c}, \boldsymbol{v}_d, \boldsymbol{u}_d \in \mathbb{R}^{D_h} \\ b_d, d_d \in \mathbb{R} \end{cases}$$

For more interesting continuous modeling, you can see [4].

Section 3

Masked Autoencoder for Distribution Estimation

Intuition From Autoencoder

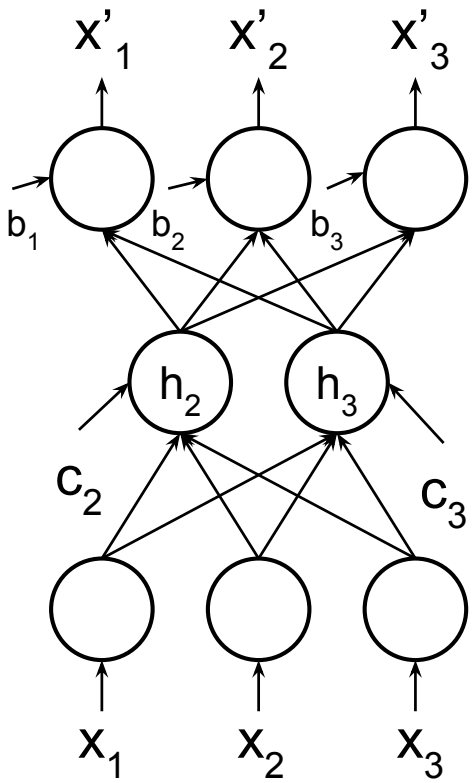


Figure: Autoencoder as unsupervised learning architecture

Autoencoder

An Autoencoder (AE) is characterized by:

- Encoding: $\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{c}$
- Decoding: : $\boldsymbol{x}' = \boldsymbol{V}\boldsymbol{h} + \boldsymbol{b}$
- Constraints to avoid identity mapping
- Loss:

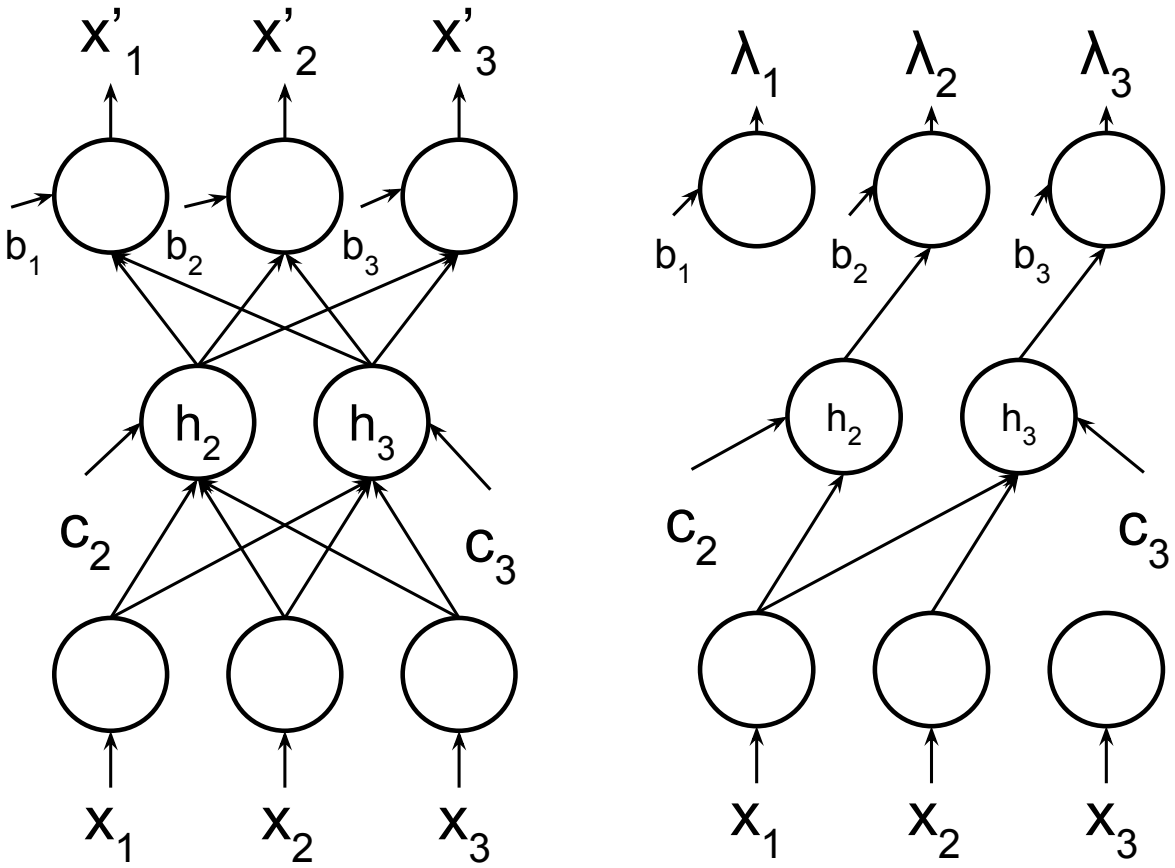
Binary input:

$$L_b(\boldsymbol{\theta}) = -\sum_{\boldsymbol{x}\in\mathcal{D}}\sum_{i=1}^{n}[x_i \log x'_i + (1-x_i)\log(1-x'_i)]$$

Continuous input:

$$L_c(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}\in\mathcal{D}}\sum_{i=1}^{n}(x_i - x'_i)^2$$

Autoencoder vs NADE



(a) Autoencoder (b) NADE

Figure: Comparison of AE and NADE (Pay attention to the architecture pruning to validate autoregressive condition)

MADE

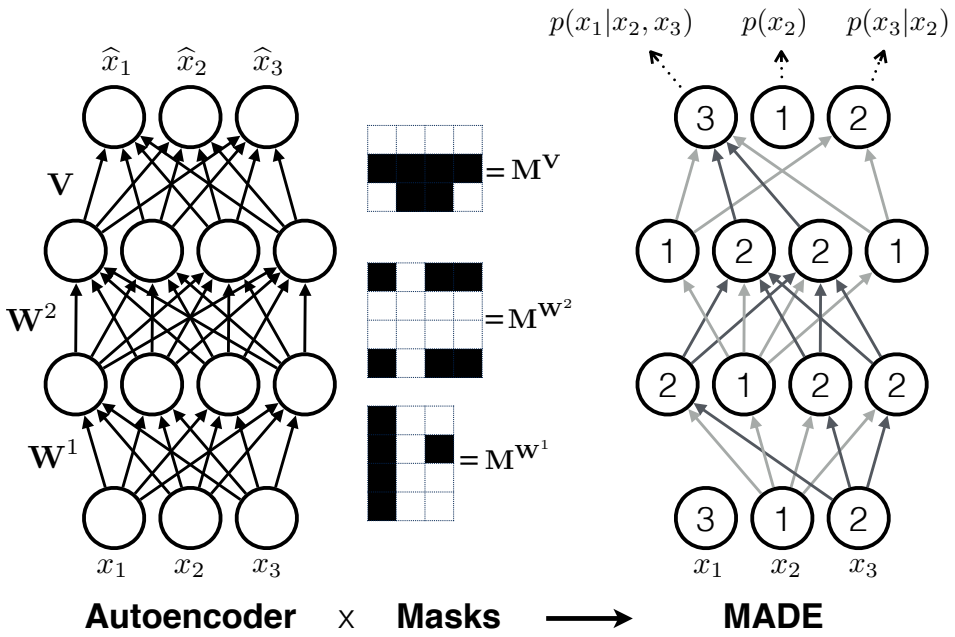


Figure: Converting an AE into a MADE (source: [5])

AE to MADE

1. Input ordering

$$x_2, x_3, x_1$$

2. Set output ordering as input
3. For each unit in the hidden layer, pick a random integer index from $[1, D-1]$
4. Each unit can be connected to previous layer units with equal or smaller index in hidden layers and strictly smaller index in the output layer (masks)

MADE

- In MADE you can compute all the conditional needed to evaluate $p(\boldsymbol{x})$ in parallel.
- Sampling from MADE is sequentially
- Assume ordering x_2, x_3, x_1, then the chain rule is:

$$p_\theta(\boldsymbol{x}) = p_\theta(x_2)p_\theta(x_3|x_2)p_\theta(x_1|x_2, x_3)$$

  - Because distribution over x_2 is unconditional, the second unit in the output layer has no input connection.
  - Because x_1 is in none of the conditions, the first unit in the input layer has no output connection.

Section 4

Recurrent Neural Networks

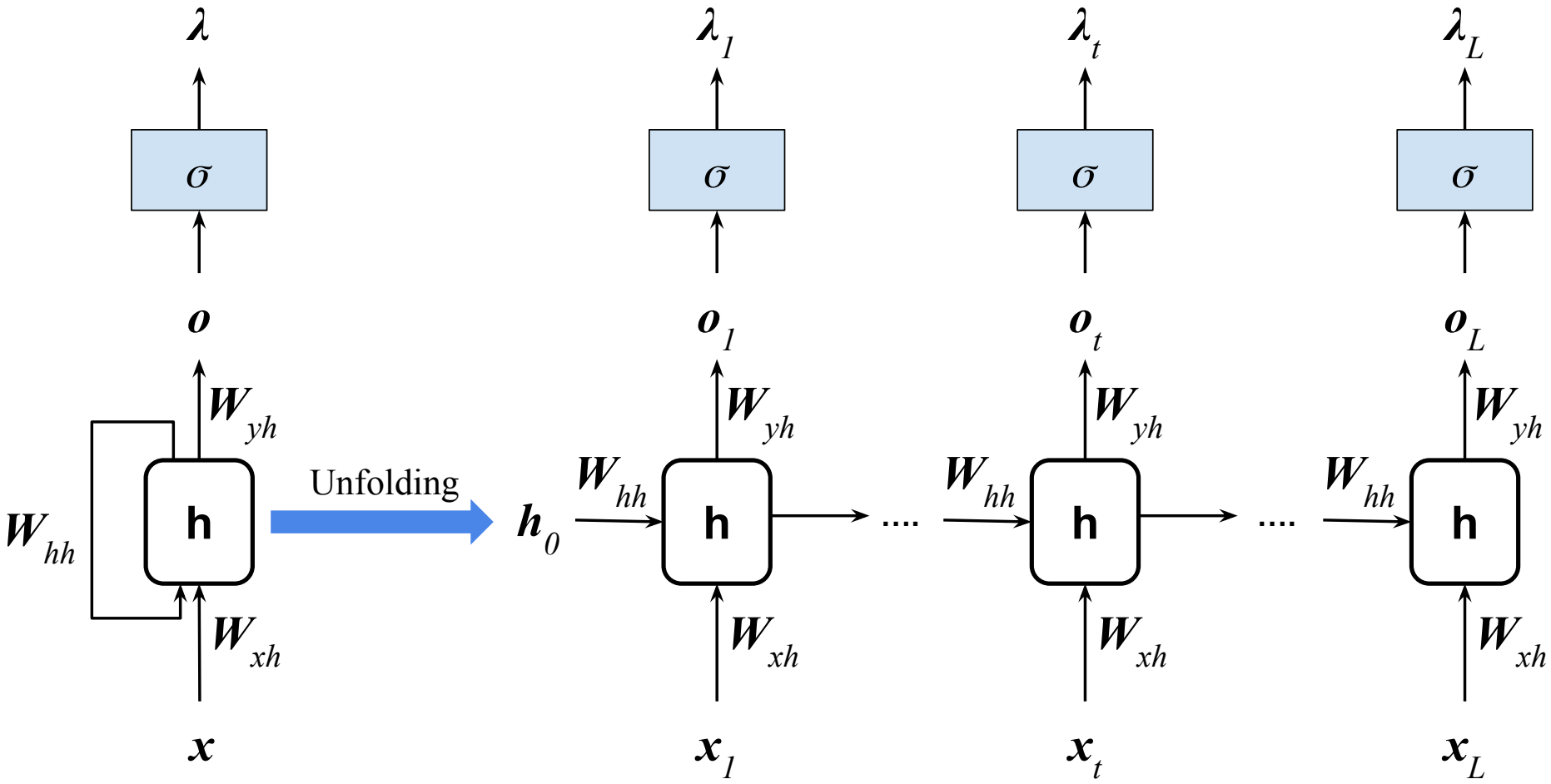


Figure: RNN architecture with its unfolded version

RNN

Assume: $\boldsymbol{x} = \overbrace{\text{I}}^{\boldsymbol{x}_1, y_1=0} \overbrace{\text{am}}^{\boldsymbol{x}_2, y_2=0} \overbrace{\text{going}}^{\boldsymbol{x}_3, y_3=1} \overbrace{\text{to}}^{\boldsymbol{x}_4, y_4=0} \overbrace{\text{Bank}}^{\boldsymbol{x}_3, y_3=0}$

Thus you have problem to relate two sequences of length l where l is variable and your dataset is $\mathcal{D} = \{(\boldsymbol{X}_i \in \mathbb{R}^{D \times l_i}, \boldsymbol{y}_i \in \mathbb{R}^{l_i})\}_{i=1}^{N}$. RNNs can capture this scenario.

RNN

- Initialization: $\boldsymbol{h}_0 = \mathbf{0}$
- For $i = 1$ to l:
  - Hidden state update:$\boldsymbol{h}_i = f(\boldsymbol{W}_{hh}\boldsymbol{h}_{i-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_i)$
  - Output calculation: $\boldsymbol{o}_i = \sigma(\boldsymbol{W}_{yh}\boldsymbol{h}_i)$

Generative RNN

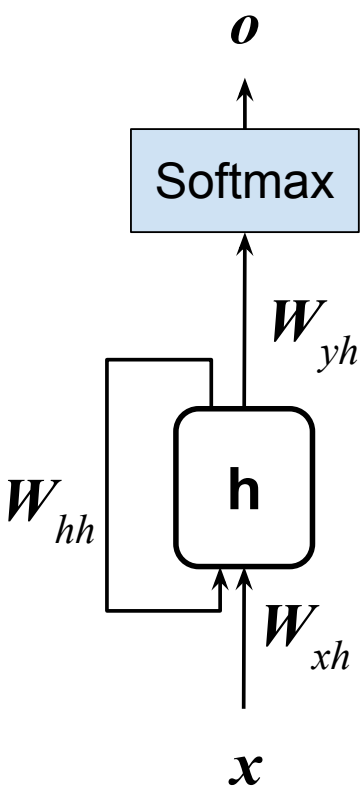


Figure: Compact generative RNN

Generative RNN

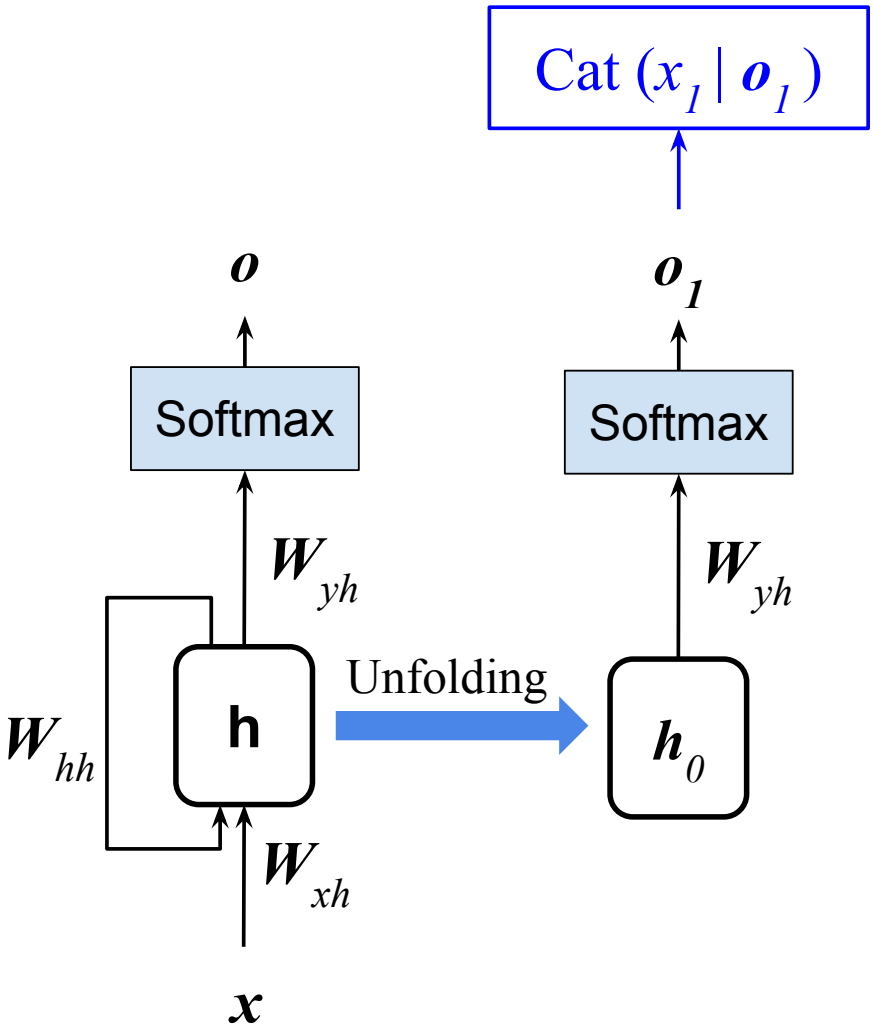


Figure: Calculations for $p_\theta(x_1)$

Generative RNN

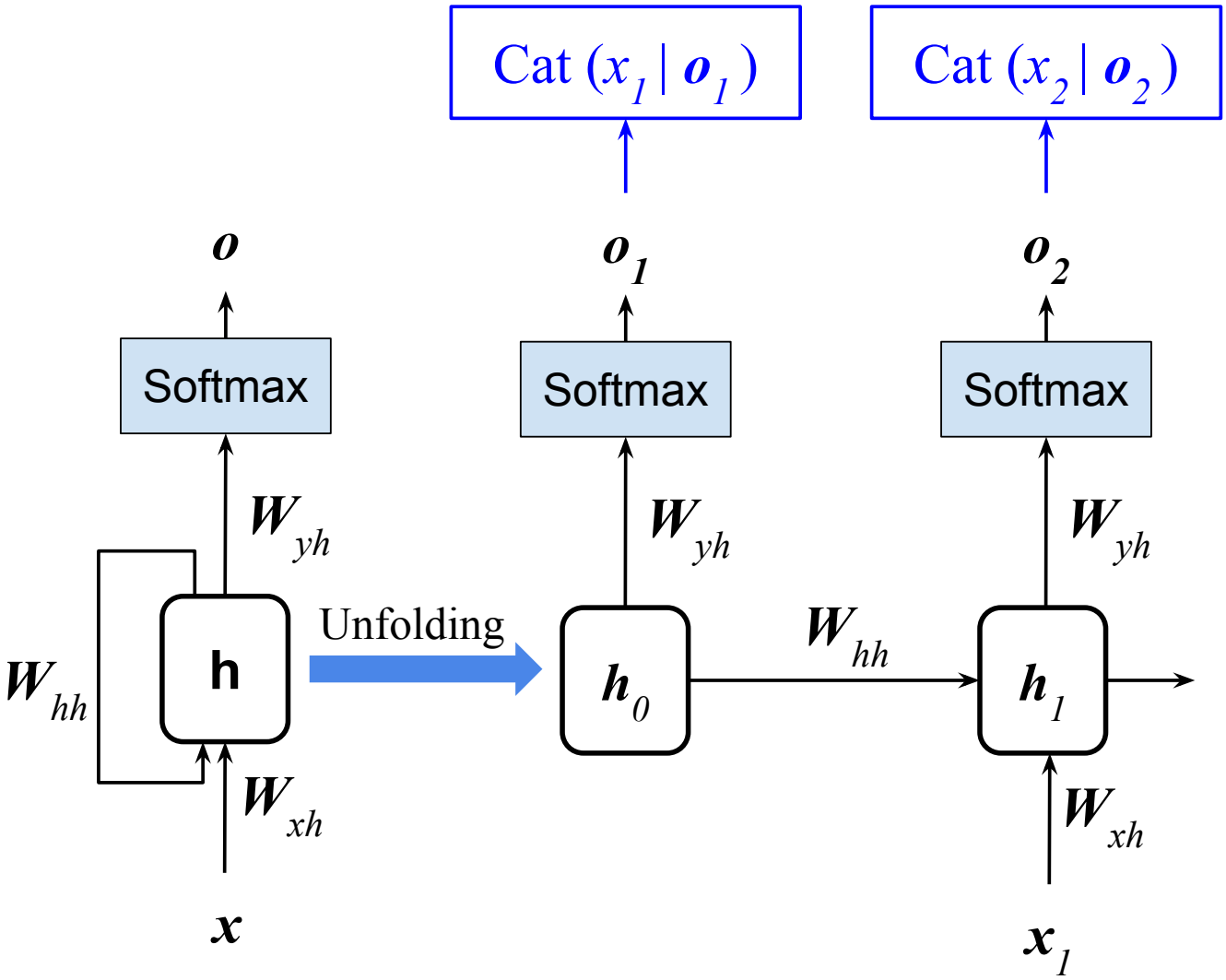


Figure: Calculations for $p_\theta(x_2|x_1)$

Generative RNN

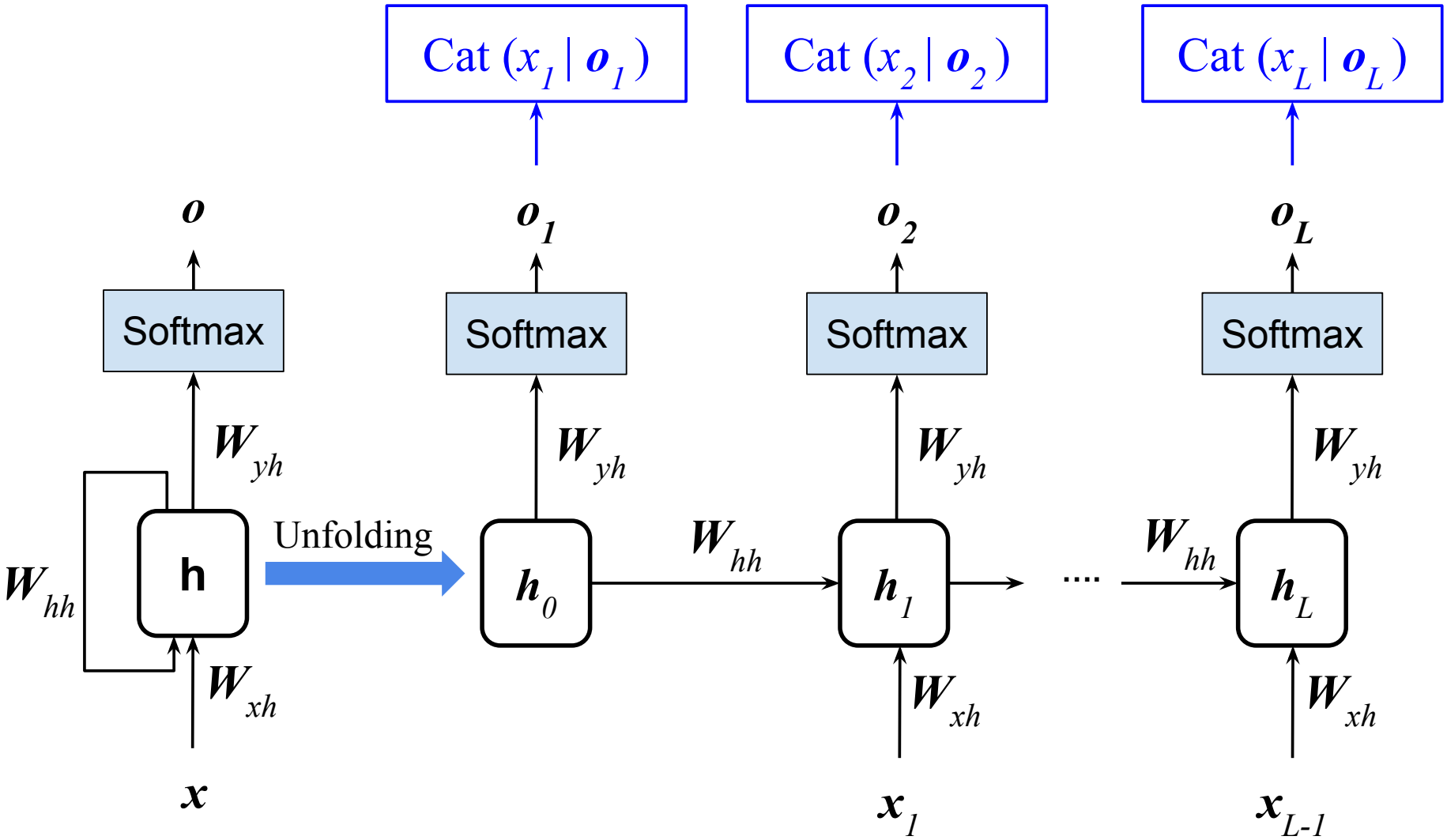


Figure: Calculations for $p_\theta(x_D|x_1, \ldots, x_{D-1})$

Generative Modeling Based on RNN Architecture

Pixel RNN

Assume we want to design an autoregressive generative model for MNIST dataset $\mathcal{D} = \left\{\boldsymbol{x}_i \in \{0, 1, \ldots, 255\}^{784}\right\}_{i=1}^{N}$ using RNN architecture:

$$\boldsymbol{h}_0 = \boldsymbol{b}_0 \Rightarrow \boldsymbol{o}_0 = \text{softmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_0) \Rightarrow p(x_1) = \text{Cat}(x_1|\boldsymbol{o}_1)$$

$$\vdots$$

$$\boldsymbol{h}_{i-1} = f(\boldsymbol{W}_{hh}\boldsymbol{h}_{i-2} + \boldsymbol{W}_{xh}\boldsymbol{x}_{i-1}) \Rightarrow$$
$$\boldsymbol{o}_{i-1} = \text{softmax}(\boldsymbol{W}_{yh}\boldsymbol{h}_{i-1}) \Rightarrow p(x_i|\boldsymbol{x}_{<i}) = \text{Cat}(x_i|\boldsymbol{o}_{i-1})$$

Thus we can calculate conditionals while RNN guarantees that the autoregression constraint is met.

Generative RNN

Generative RNN

- Pros:
  - Variable-length input
  - Fixed model size
  - History abstraction via hidden representation $\boldsymbol{h}_d$
  - Weight sharing
- Cons
  - Sequential computations
  - Difficult to carry information for long sequences
  - Ordering required

Pixel RNN

Figure: Pixel RNN results in image completion (source: [6])

Section 5

Learning

Distance Metric

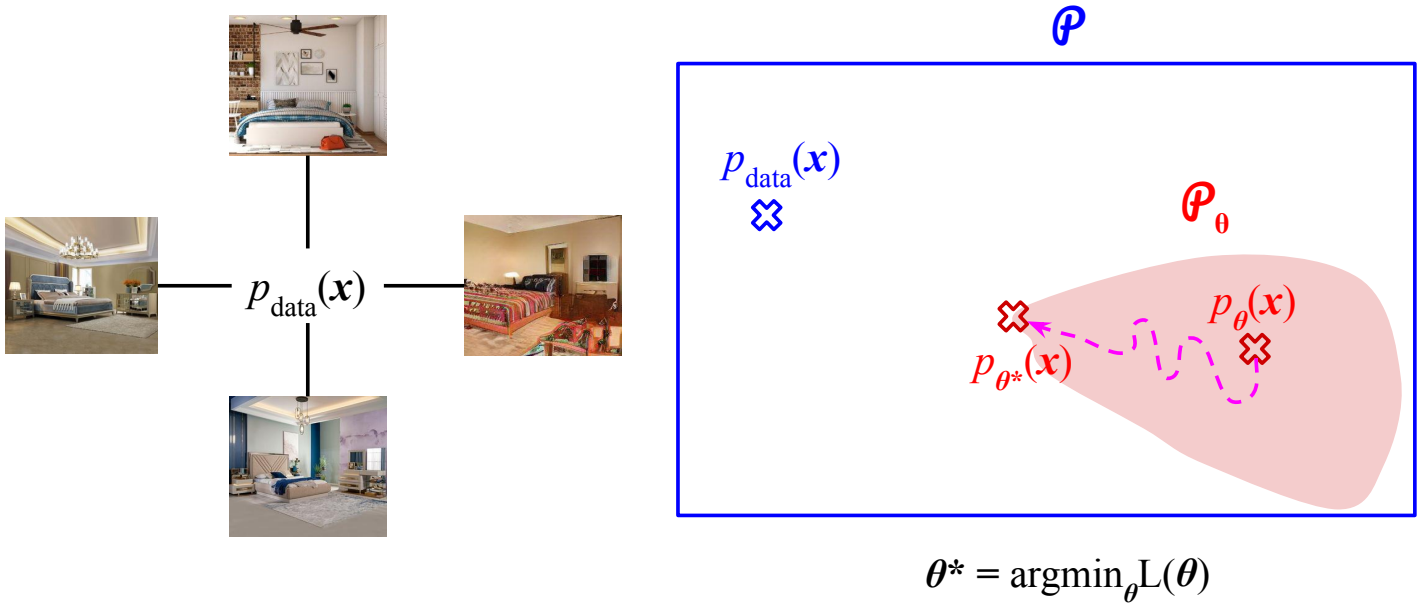


Figure: General view to generative modeling

So Far

- Characterizing $\mathcal{P}$
- Characterizing $\mathcal{P}_\theta$
  - FVSBN, NADE, MADE, RNN, ...

From now on

- Determining the distance $L(\boldsymbol{\theta})$
- Optimizing $\boldsymbol{\theta}$ (Finding $\boldsymbol{\theta}^\star$)

Distance Metric

We want to comapre two distribution p_{data} and p_θ, thus we can use KL divergence as:

$$L(\boldsymbol{\theta}) = \text{dist}(p_{\text{data}}, p_\theta) = \text{KL}(p_{\text{data}} \| p_\theta) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\mathbb{X})} \left[\log \left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} \right) \right]$$

$$= \sum_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}$$

Using above definition, we know $L(\boldsymbol{\theta}) = 0$ iff $p_\theta(\mathbb{X}) = p_{\text{data}}(\mathbb{X})$. We can rewrite $L(\boldsymbol{\theta})$ as:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log p_{\text{data}}(\boldsymbol{x}) \right] - \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log p_\theta(\boldsymbol{x}) \right]$$

Because the first term on the right-hand side is independent of $\boldsymbol{\theta}$, we have:

$$\boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}}{\text{argmin}}\, \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log \left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} \right) \right] \equiv \underset{\boldsymbol{\theta}}{\text{argmax}}\, \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log p_\theta(\boldsymbol{x}) \right]$$

From KL divergence to Model Likelihood

Model Likelihood

We see:

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log p_{\theta}(\boldsymbol{x}) \right]$$

Thus:

- Desirable situation is when $p_{\theta}(\mathbb{X})$ assign high probability to probable regions in $p_{\text{data}}(\mathbb{X})$
- We have yet a problem: No access to p_{data}
- $\mathbb{H}(p_{\text{data}}(\mathbb{X})) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\mathbb{X})} \left[\log p_{\text{data}}(\boldsymbol{x}) \right]$ is the maximum accessible objective value where $\mathbb{H}(p_{\text{data}}(\mathbb{X}))$ is the *entropy* defined as:

$$\mathbb{H}(p_{\text{data}}(\mathbb{X})) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log p_{\text{data}}(\boldsymbol{x}) \right]$$

Model Likelihood Estimation

We are interested in solving the following problem:

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\mathbb{X})} \left[\log p_{\theta}(\boldsymbol{x})\right]$$

but we don't have access to p_{data} and instead, we have access to independent samples from the distribution $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}$.

Solution via Monte Carlo Estimate

Using the Monte Carlo estimate we have:

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\mathbb{X})} \left[\log p_{\theta}(\boldsymbol{x})\right] \simeq \frac{1}{N} \sum_{n=1}^{N} \log p_{\theta}(\boldsymbol{x}_n)$$

Thus:

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \frac{1}{N} \sum_{n=1}^{N} \log p_{\theta}(\boldsymbol{x}_n)$$

Optimization

By adding a minus sign to the objective function, we have the following minimization problem:

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -\frac{1}{|\mathcal{D}|} \sum_{n=1}^{N} \log p_{\theta}(\boldsymbol{x}_n)$$

and we can use different methods to find $\boldsymbol{\theta}^{\star}$ such as:

- Gradient descent
- Stochastic gradient descent
- ...

Section 6

Examples

Coin Tossing

Problem

Assume you have a biased coin and toss it for 4 times and your dataset is:

$$\mathcal{D} = \{1, 0, 1, 1\}, \begin{cases} 1 & \text{head} \\ 0 & \text{tail} \end{cases}$$

Suggest the parameterized model and find the optimum parameters.

Solution

The model is a Bernoulli model as: $p_\theta(x) = \theta^x (1-\theta)^{1-x}$
The optimum parameter is the result of the following optimization problem:

$$\theta^\star = \underset{\theta}{\operatorname{argmin}} -\frac{1}{4} \sum_{i=1}^{4} [x_i \log \theta + (1 - x_i) \log(1 - \theta)]$$

Coin Tossing

Solution (Cont.)

Taking into account the dataset $\mathcal{D} = \{1, 0, 1, 1\}$, we have:

$$\begin{aligned}\theta^\star &= \operatorname*{argmin}_\theta -\frac{1}{4}(\log\theta + \log(1-\theta) + \log\theta + \log\theta) \\ &= -\frac{3}{4}\log\theta - \frac{1}{4}\log(1-\theta)\end{aligned}$$

By taking the derivative of the objective with respect to θ, we can find:

$$\theta^\star = \frac{3}{4}$$

Thus your generative model is: $p_\theta(x) = \text{Ber}(x|\frac{3}{4})$ and you can:

- Sample from this distribution
- Calculate the probability of an input x

Coin Tossing

General Case

Now assume you have access to a dataset with N samples where N_h samples are head and N_t samples are tail ($N = N_h + N_t$), then we can write:

$$\theta^\star = \underset{\theta}{\text{argmin}} - \left(\frac{N_h}{N} \log \theta + \frac{N_t}{N} \log(1-\theta) \right)$$

which results in:

$$\theta^\star = \frac{N_h}{N}$$

and thus your generative model is:

$$p_\theta(x) = \text{Ber}\left(x \mid \frac{N_h}{N} \right)$$

Learning General Autoregressive Model

Learning

Assume you're given an Autoregressive model (such as NADE, MADE, PixelRNN, etc.), and you are asked to train the model using $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$. Then:

- Select a (mini)batch $\mathcal{B}$ of samples with size N'.
- Compute the following loss function in a parametric form:

$$L(\boldsymbol{\theta}) = -\frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}_i \in \mathcal{B}} \log p_\theta(\boldsymbol{x}_i)$$

- Take the gradient with respect to parameters $\nabla_\theta L$
- Update parameters using (stochastic) gradient descent or any other gradient-based optimization method
- Repeat the above steps until a convergence criterion is met.

Avoid Overfitting

The following strategies can be used to avoid overfitting:

- Decreasing the size of $p_\theta(\mathbb{X})$ set (Toward independence assumption) by:
  - Limiting the number of processors that each variable can depend on:

$$p_\theta(x_i|\boldsymbol{x}_{<i}) = p_\theta(x_i|x_{i-1}, x_{i-2})$$

  - Weight sharing (similar to NADE and RNN)
  - Regularization such as Gaussian prior over parameters (Weight decay)
- Early stopping via monitoring likelihood over a validation set

Section 7

Conditional Generation

Conditional Autoregressive Generation

Conditional Generation Vs. Supervised Learning

Assume dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ where $\boldsymbol{y}_i$ is a text and $\boldsymbol{x}_i$ is the corresponding speech. For this problem:

- Similar to supervised learning, we are interested in modeling $p(\mathbb{X}|\boldsymbol{y})$. Thus the unconditional distribution $p(\mathbb{Y})$ is not modeled and $\boldsymbol{y}$ is assumed to be given.
- In contrast to supervised learning, $\mathbb{X}$ is a high-dimensional random vector (note that for binary classification X is Bernoulli and for multi-class classification, X is Categorical).

Conditional Autoregressive Generation

Modeling

Keeping in mind that $\boldsymbol{y} \in \mathbb{R}^{D_g}$ is a give vector and we want to model $p_\theta(\mathbb{X}|\boldsymbol{y})$ using Autoregressive model, where $\mathbb{X}$ is D dimensional random vector, we have:

$$p_\theta(\mathbb{X}|\boldsymbol{y}) = p_\theta(X_1|\boldsymbol{y})p_\theta(X_2|\boldsymbol{x}_{<2}, \boldsymbol{y}) \dots p_\theta(X_d|\boldsymbol{x}_{<d}, \boldsymbol{y}) \dots p_\theta(X_D|\boldsymbol{x}_{<D}, \boldsymbol{y})$$

and we can use any of the autoregressive modeling approaches to model the conditionals.

Density Estimation and Generation

Similar to unconditional Autoregressive models, you can sample new $\boldsymbol{x}$ given $\boldsymbol{y}$ (assume different response $\boldsymbol{x}$ that can ChatGPT give to a fixed question $\boldsymbol{y}$), or estimate density for a $\boldsymbol{x}$ vector while $\boldsymbol{y}$ is given.

Distance Metric

One option for distance metric is:

$$
\begin{aligned}
L(\boldsymbol{\theta}) &= \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}(\mathbb{Y})}\left[\text{KL}\left(p_{\text{data}}(\mathbb{X}|\boldsymbol{y}) \parallel p_\theta(\mathbb{X}|\boldsymbol{y})\right)\right] \\
&= \sum_{\boldsymbol{y}} p_{\text{data}}(\boldsymbol{y})\left[\sum_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}|\boldsymbol{y}) \log \frac{p_{\text{data}}(\boldsymbol{x}|\boldsymbol{y})}{p_\theta(\boldsymbol{x}|\boldsymbol{y})}\right] \\
&= \underbrace{\sum_{\boldsymbol{y}}\sum_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x},\boldsymbol{y}) \log p_{\text{data}}(\boldsymbol{x}|\boldsymbol{y})}_{\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_{\text{data}}(\mathbb{X},\mathbb{Y})}[\log p_{\text{data}}(\boldsymbol{x}|\boldsymbol{y})]} - \underbrace{\sum_{\boldsymbol{y}}\sum_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x},\boldsymbol{y}) \log p_\theta(\boldsymbol{x}|\boldsymbol{y})}_{\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_{\text{data}}(\mathbb{X},\mathbb{Y})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{y})]}
\end{aligned}
$$

While the second term is a function of your model parameters, the first one is independent of the selected Autoregressive model and thus can be omitted in optimization.

Distance Metric

Altogether:

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \; \mathbb{E}_{\boldsymbol{y} \sim p_{\text{data}}(\mathbb{Y})} \Big[\operatorname{KL} \Big(p_{\text{data}}(\mathbb{X}|\boldsymbol{y}) \parallel p_{\theta}(\mathbb{X}|\boldsymbol{y}) \Big) \Big]$$

$$\equiv \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \; \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}) \sim p_{\text{data}}(\mathbb{X},\mathbb{Y})} \Big[\log p_{\theta}(\boldsymbol{x}|\boldsymbol{y}) \Big]$$

The above can be translated as:
Minimizing the expected KL divergence of true conditional distribution and model conditional distribution over true $\mathbb{X}$ is equivalent to maximizing the expected likelihood of model conditional distribution over true joint distribution.

Model Likelihood Estimation

Model Conditional Likelihood Estimation

We are interested in solving the following problem:

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \; \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_{\text{data}}(\mathbb{X},\mathbb{Y})} \big[\log p_{\theta}(\boldsymbol{x}|\boldsymbol{y}) \big]$$

but we don't have access to $p_{\text{data}}(\mathbb{X}, \mathbb{Y})$ and instead, we have access to independent samples from the distribution $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$.

Solution via Monte Carlo Estimate

Using the Monte Carlo estimate we have:

$$\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_{\text{data}}(\mathbb{X},\mathbb{Y})} \big[\log p_{\theta}(\boldsymbol{x}|\boldsymbol{y}) \big] \simeq \frac{1}{N} \sum_{n=1}^{N} \log p_{\theta}(\boldsymbol{x}_n|\boldsymbol{y}_n)$$

Thus:

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \frac{1}{N} \sum_{n=1}^{N} \log p_{\theta}(\boldsymbol{x}_n|\boldsymbol{y}_n)$$

Section 8

Applications

Adversarial Robustness

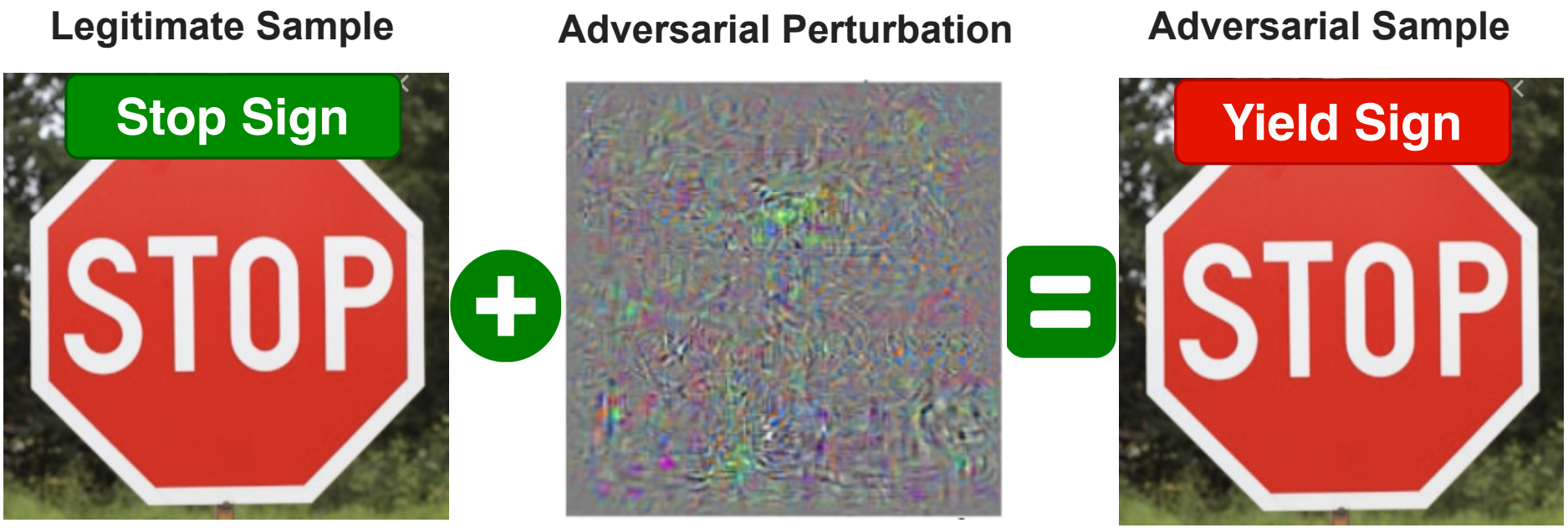


Figure: Different adversarial attacks to Frog image from Cifar10 dataset (source: [7])

Adversarial Robustness

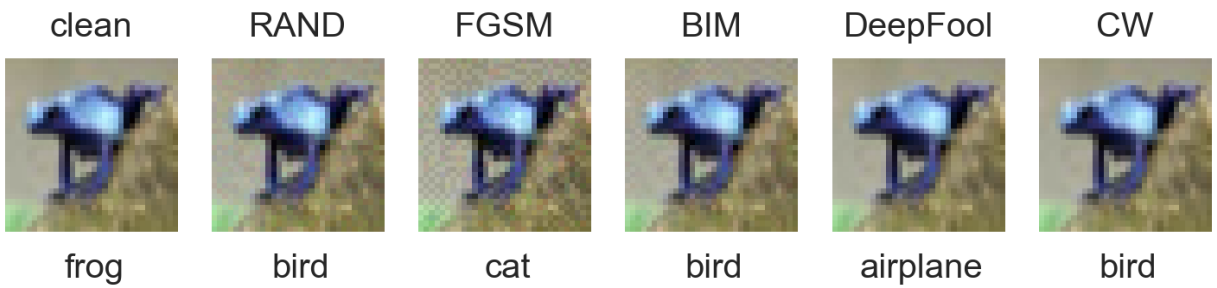
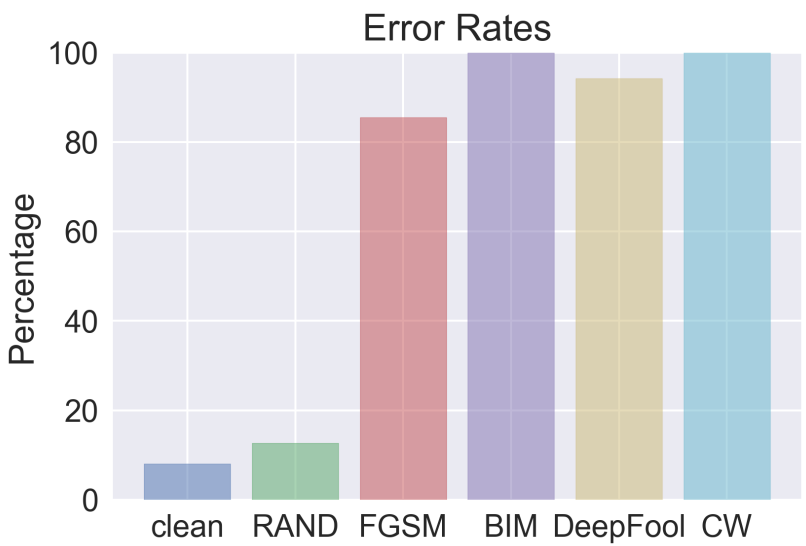
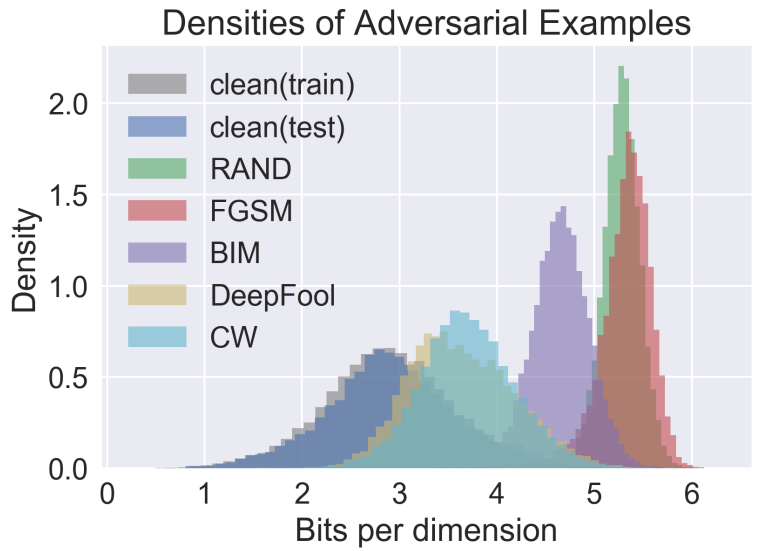


Figure: Sample adversarial attack to deep learning architectures (source: [7])



(a) Error rate



(b) Density change

Figure: Using autoregressive models to detect adversarial samples (source: [7])

Section 9

Summary

Summary

Idea

- Using chain rule to factorize the joint
- Parameterizing the factors in the chain rule using neural networks
- Using KL divergence as the distance metric
- Using Monte Carlo estimation to approximate the distance metric

Pros and Cons

- Pros:
  - Easy probability estimation
  - Easy to train and monitor training via validation likelihood
- Cons:
  - Ordering requirement
  - Sequential generation
  - No high-level representation $\boldsymbol{z}$

List of Abbreviations

Complete	Abbreviation
Binary Logistic Regression	BLR
Fully Visible Sigmoid Belief Network	FVSBN
Masked Autoencoder for Distribution Estimation	MADE
Neural Autoregressive Density Estimation	NADE
Recurrent Neural Networks	RNN

References I


Kevin P Murphy,
Machine learning: a probabilistic perspective,
MIT press, 2012.


Zhe Gan, Ricardo Henao, David Carlson, and Lawrence Carin,
"Learning deep sigmoid belief networks with data augmentation,"
in *Artificial Intelligence and Statistics*. PMLR, 2015, pp. 268–276.


Kevin P Murphy,
Probabilistic machine learning: an introduction,
MIT press, 2022.


Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle,
"Neural autoregressive distribution estimation,"
The Journal of Machine Learning Research, vol. 17, no. 1, pp. 7184–7220, 2016.


Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle,
"Made: Masked autoencoder for distribution estimation,"
in *International conference on machine learning*. PMLR, 2015, pp. 881–889.


Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu,
"Pixel recurrent neural networks,"
in *International conference on machine learning*. PMLR, 2016, pp. 1747–1756.

References II

Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman,
"Pixeldefend: Leveraging generative models to understand and defend against adversarial examples,"
arXiv preprint arXiv:1710.10766, 2017.