NAS via RNN + RL

Shusen Wang

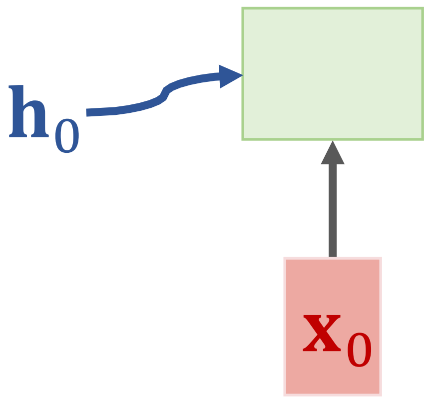
Prerequisites

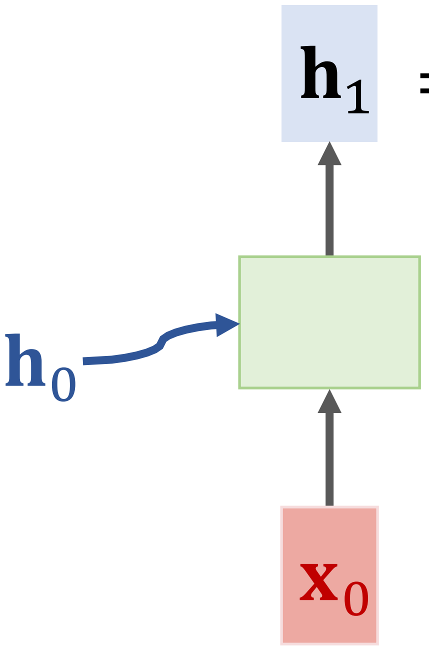
- Recurrent neural networks (RNNs).
- Policy-based reinforcement learning.

RNN for Generating CNN Architectures

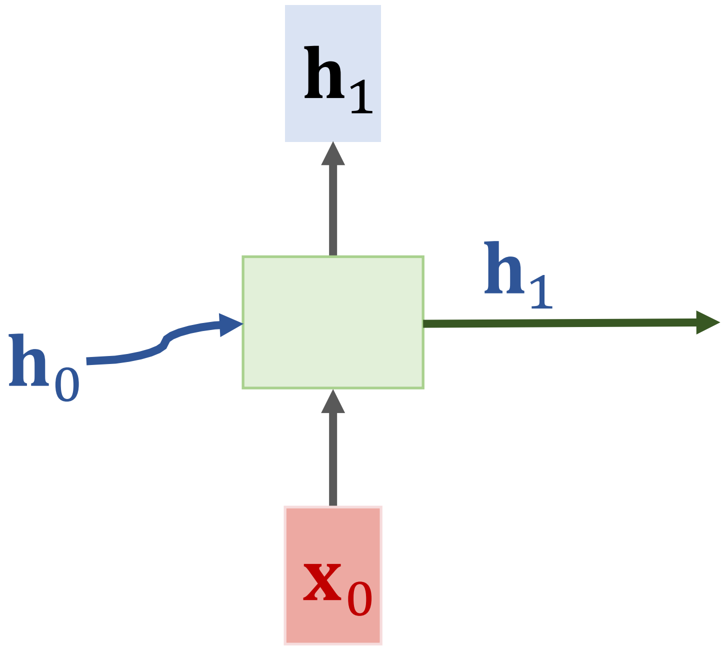
Reference:

- Zoph & Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

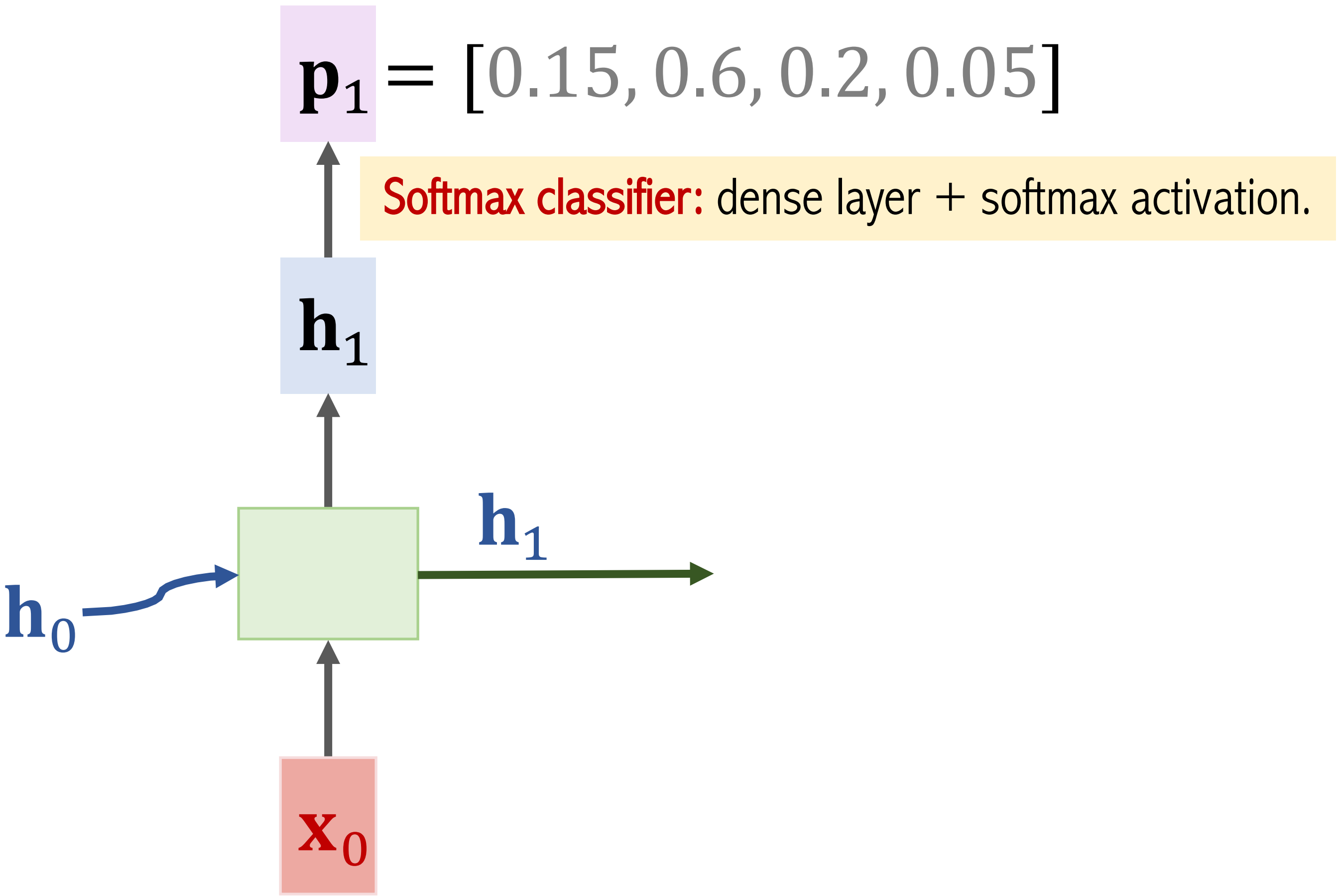

$\mathbf{h}_0$
$\mathbf{x}_0$



$$\mathbf{h}_1 = \boxed{\tanh\left(\mathbf{W}\cdot\begin{bmatrix}\mathbf{h}_0\\ \mathbf{x}_0\end{bmatrix}+\mathbf{b}\right)}$$

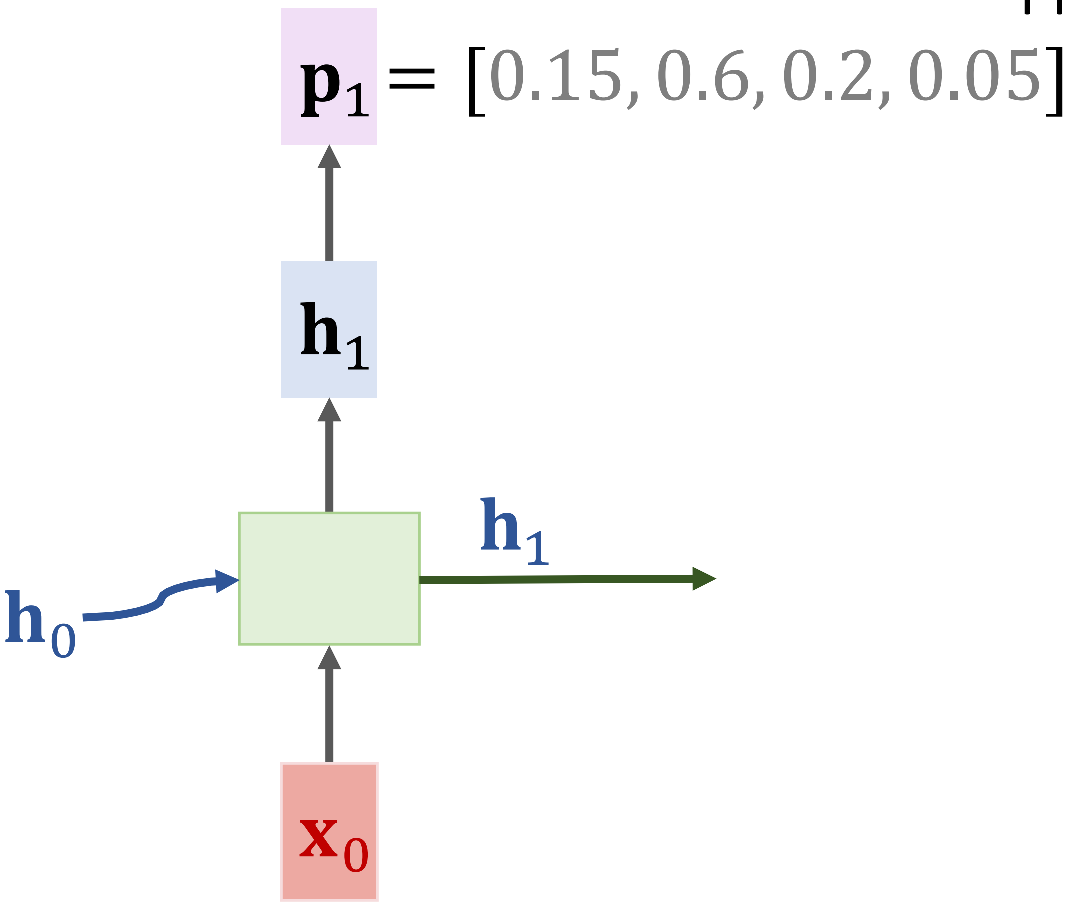


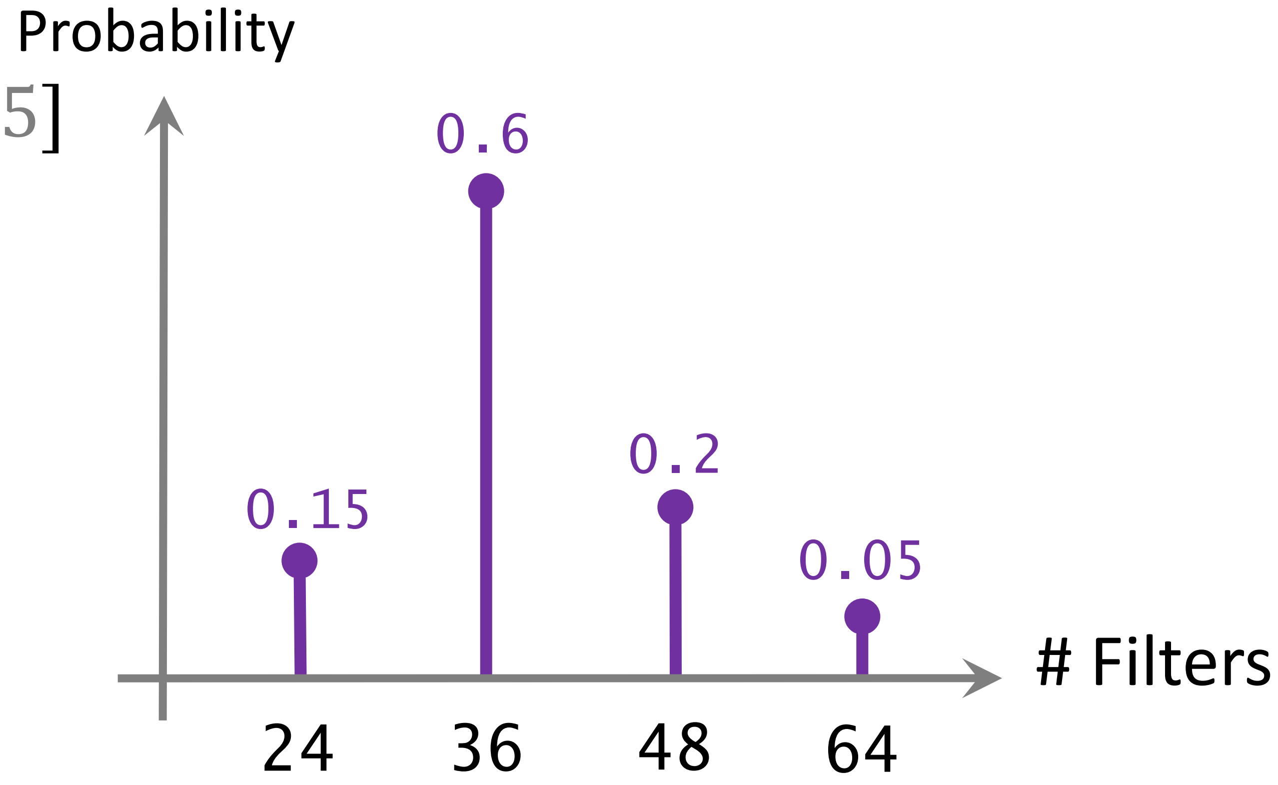
$\mathbf{h}_1$
$\mathbf{h}_1$
$\mathbf{h}_0$
$\mathbf{x}_0$



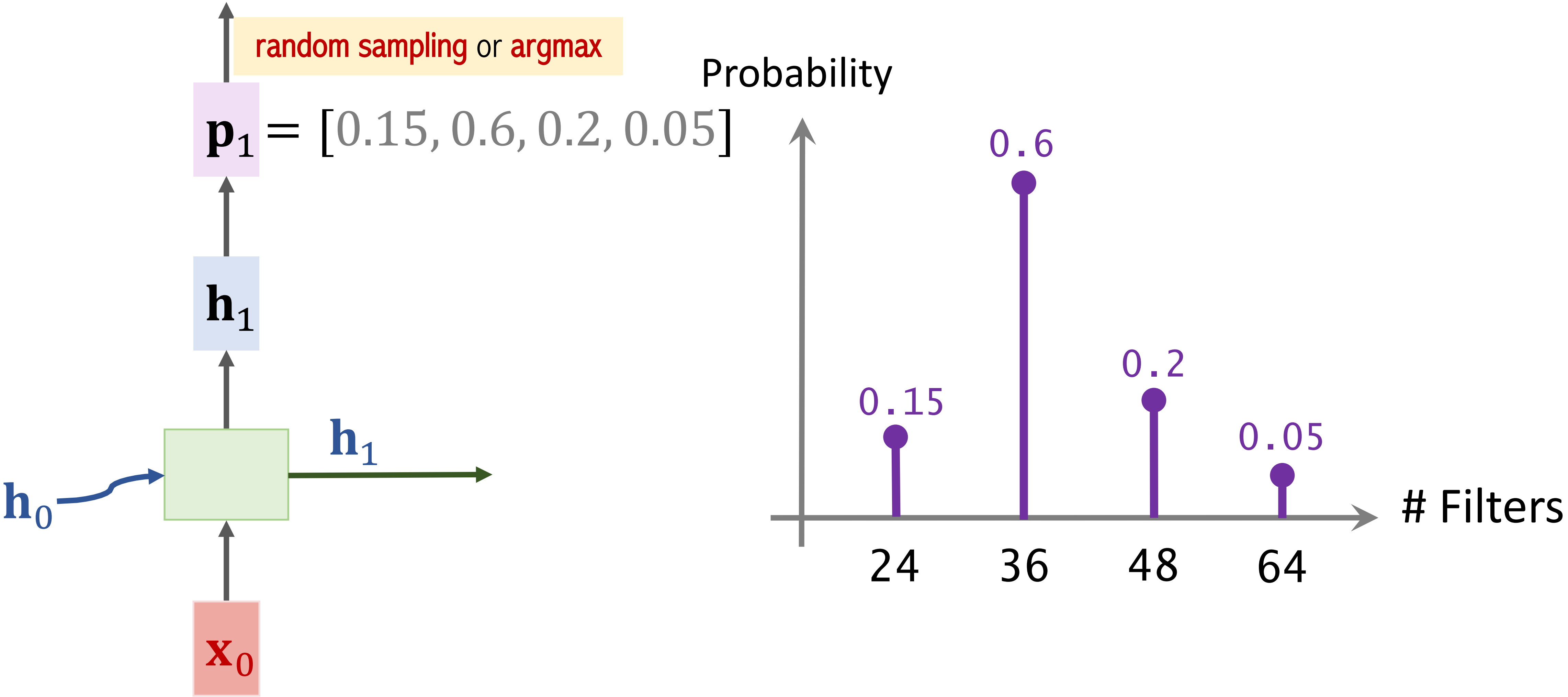
$\mathbf{p}_1 = [0.15, 0.6, 0.2, 0.05]$
Softmax classifier: dense layer + softmax activation.
$\mathbf{h}_1$
$\mathbf{h}_1$
$\mathbf{h}_0$
$\mathbf{x}_0$

Predict Number of Filters

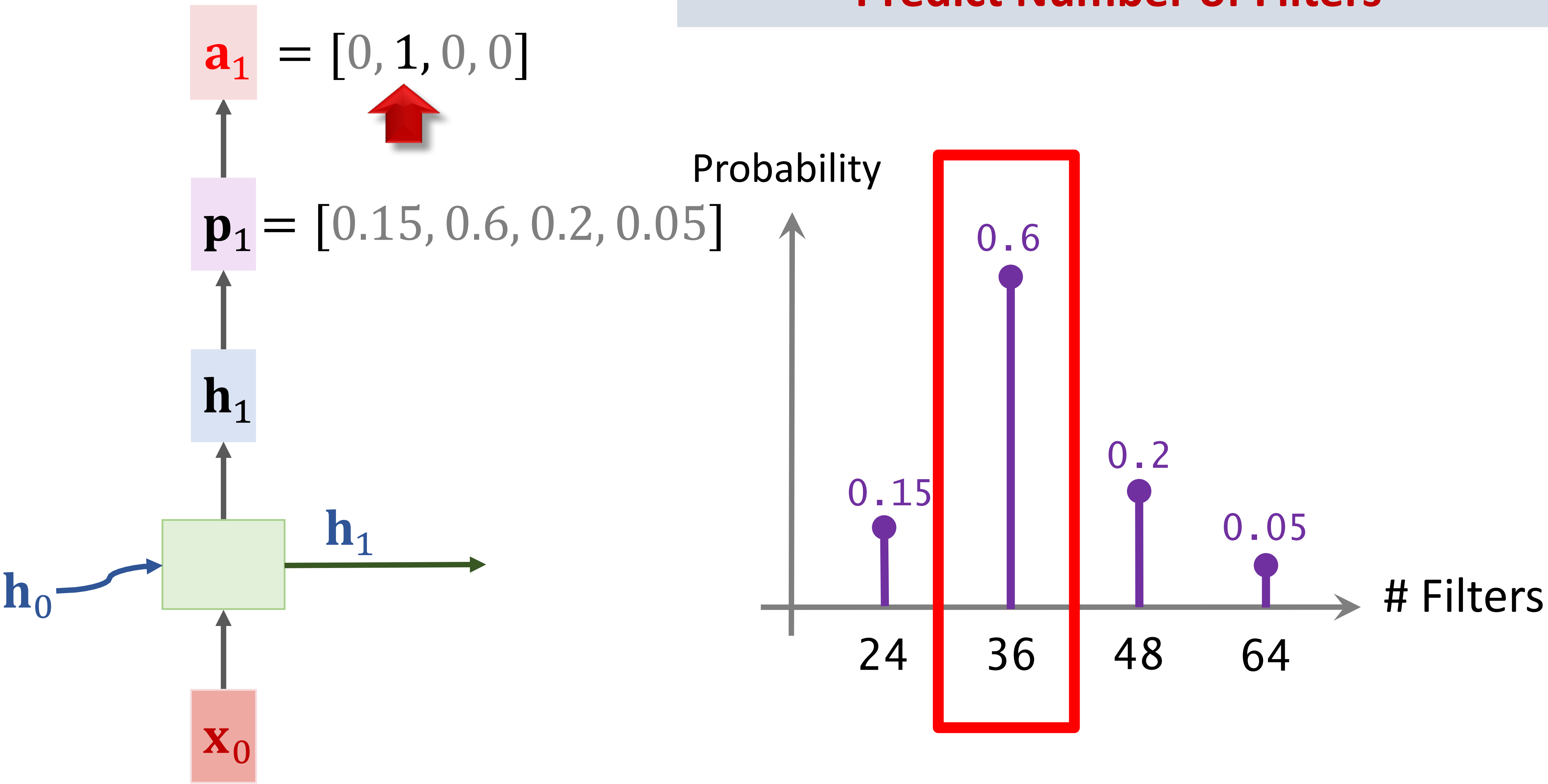


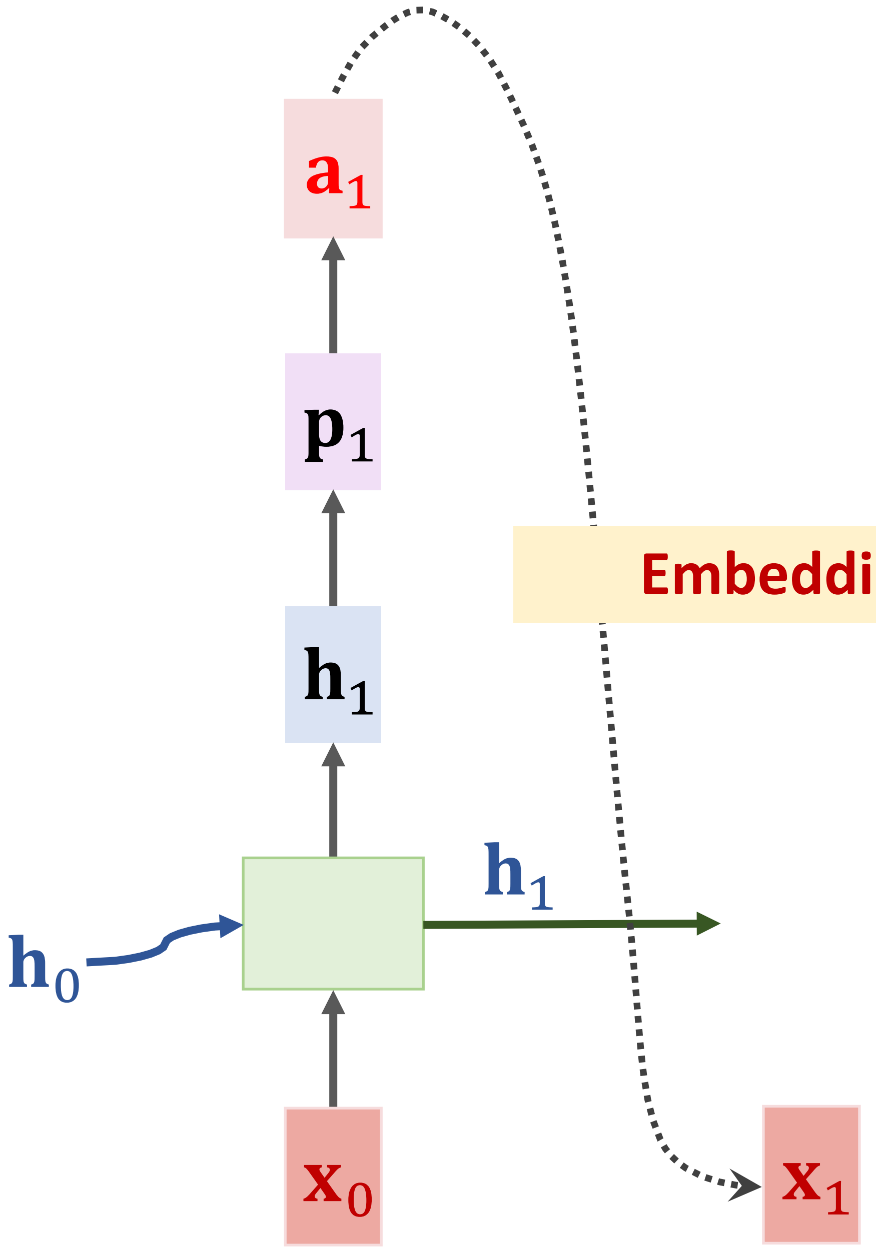


Predict Number of Filters

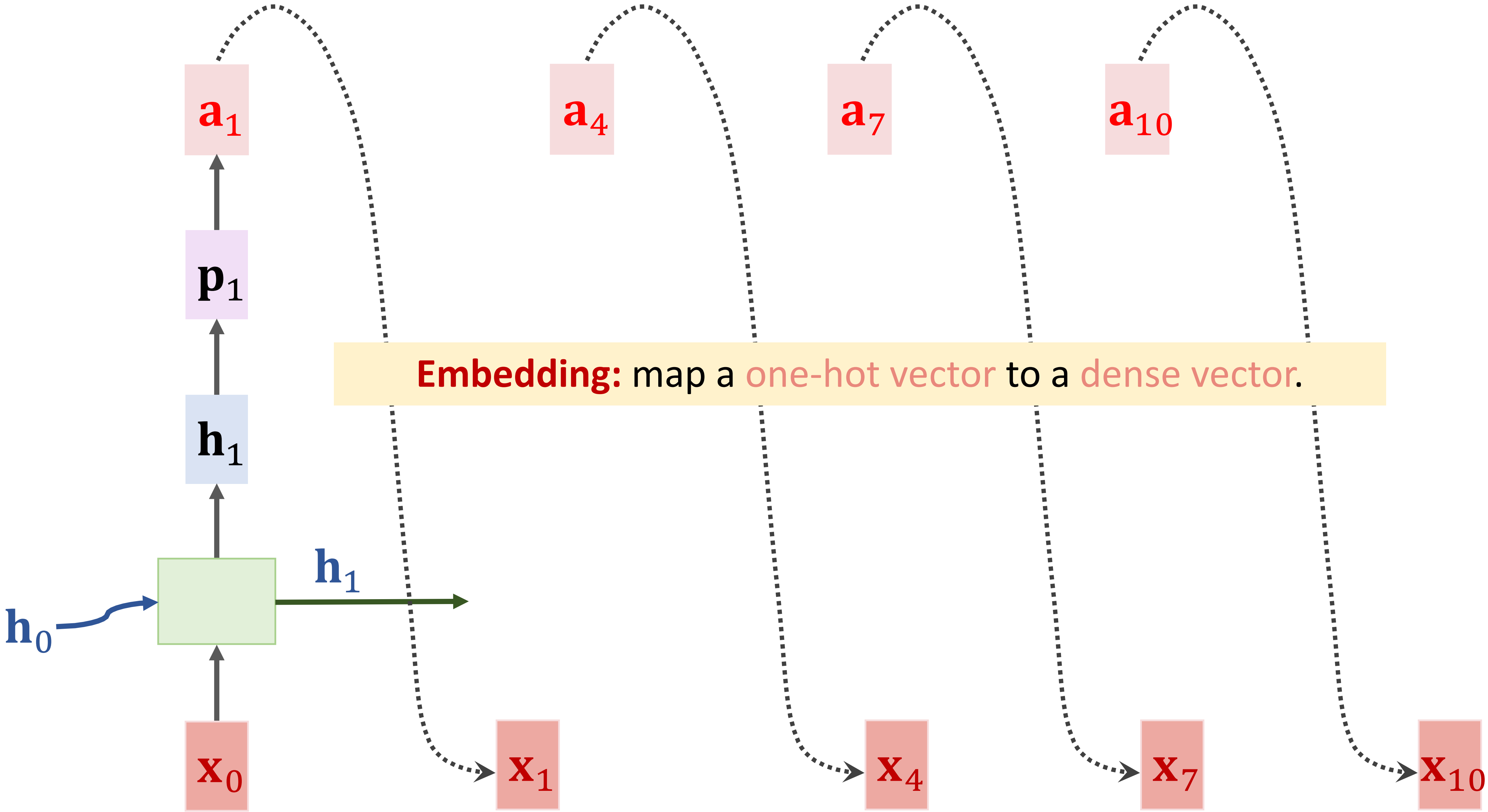


Predict Number of Filters

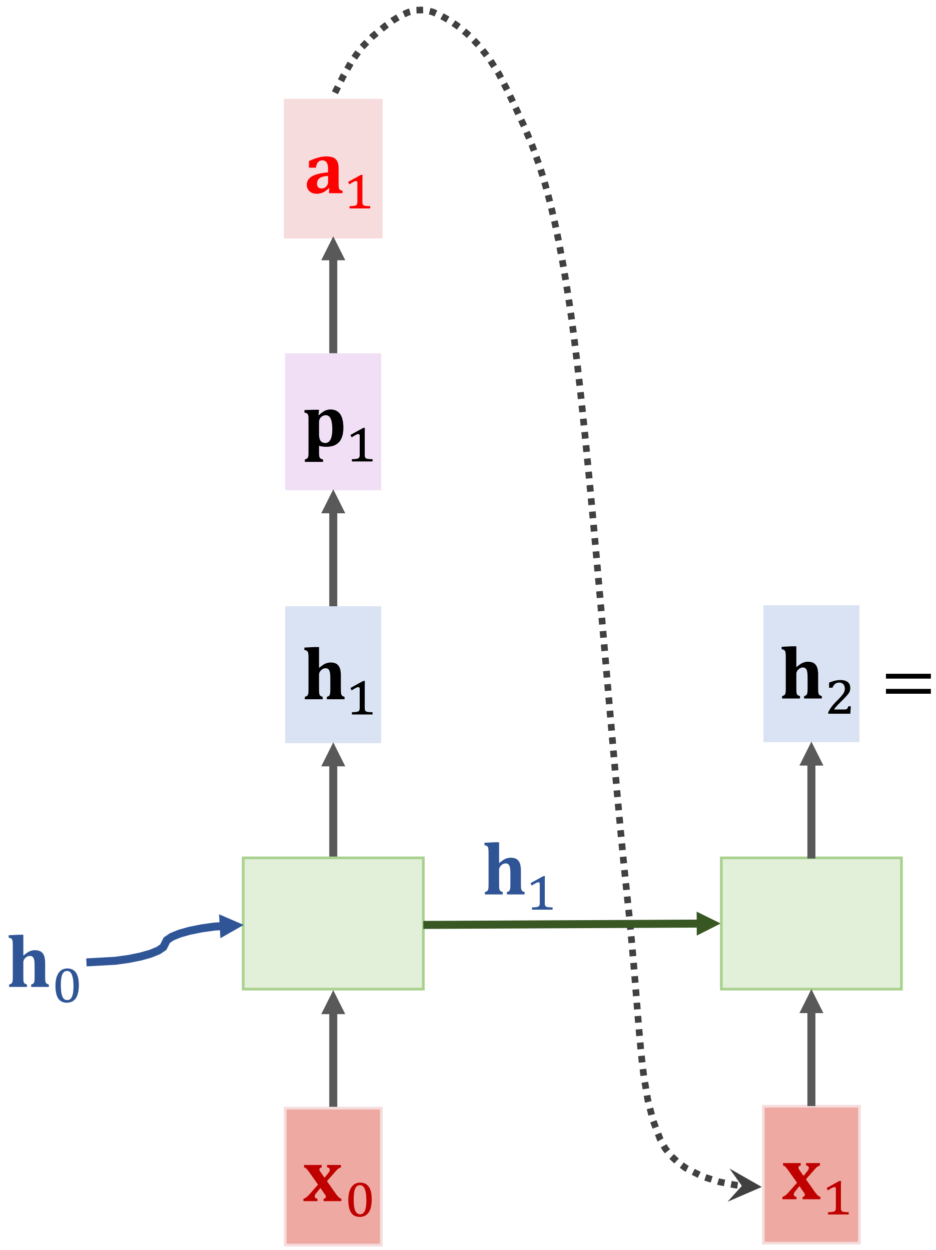




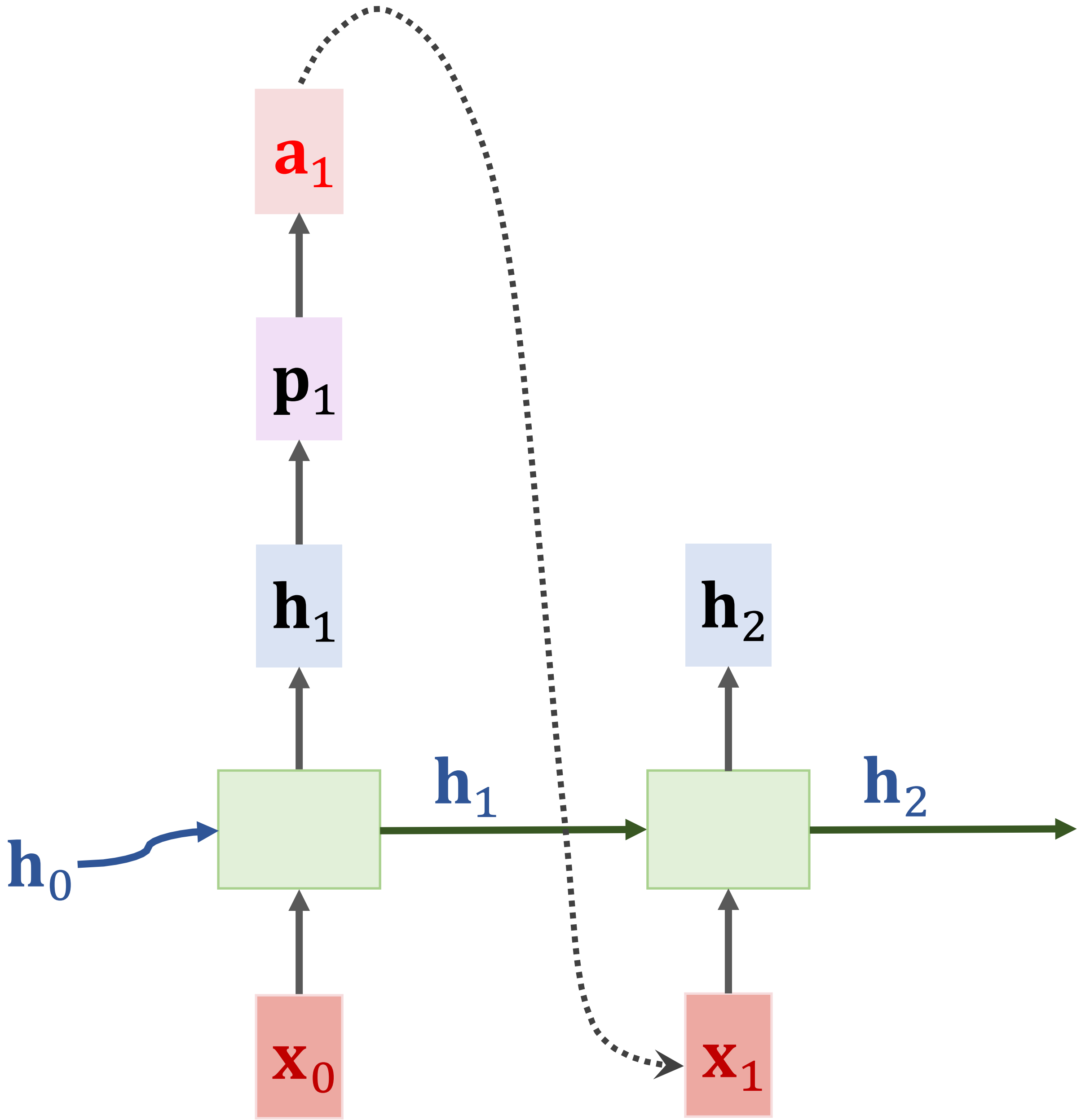
Embedding: map a one-hot vector to a dense vector.



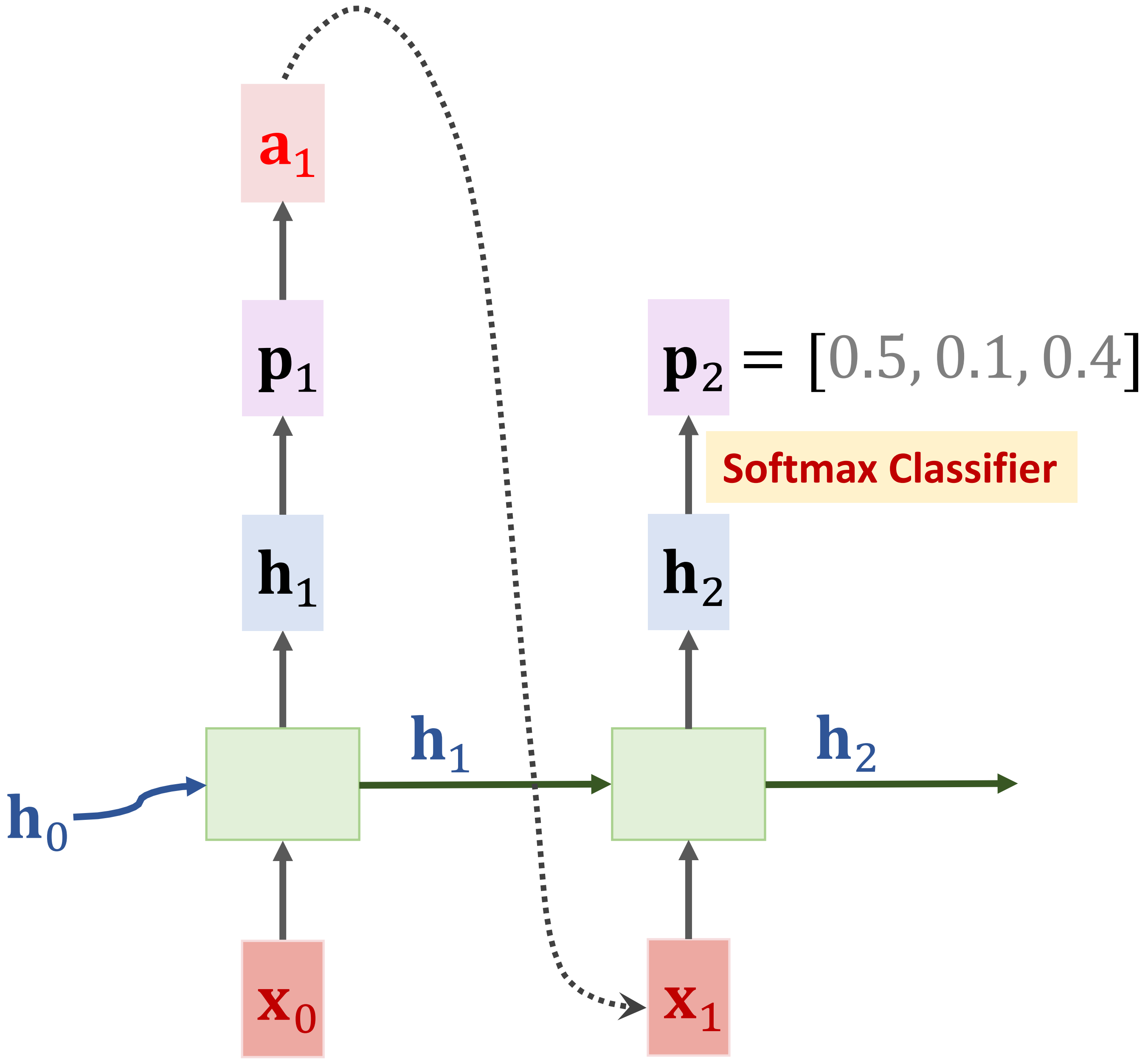
$\mathbf{a}_1$
$\mathbf{a}_4$
$\mathbf{a}_7$
$\mathbf{a}_{10}$
$\mathbf{p}_1$
Embedding: map a one-hot vector to a dense vector.
$\mathbf{h}_1$
$\mathbf{h}_1$
$\mathbf{h}_0$
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_4$
$\mathbf{x}_7$
$\mathbf{x}_{10}$



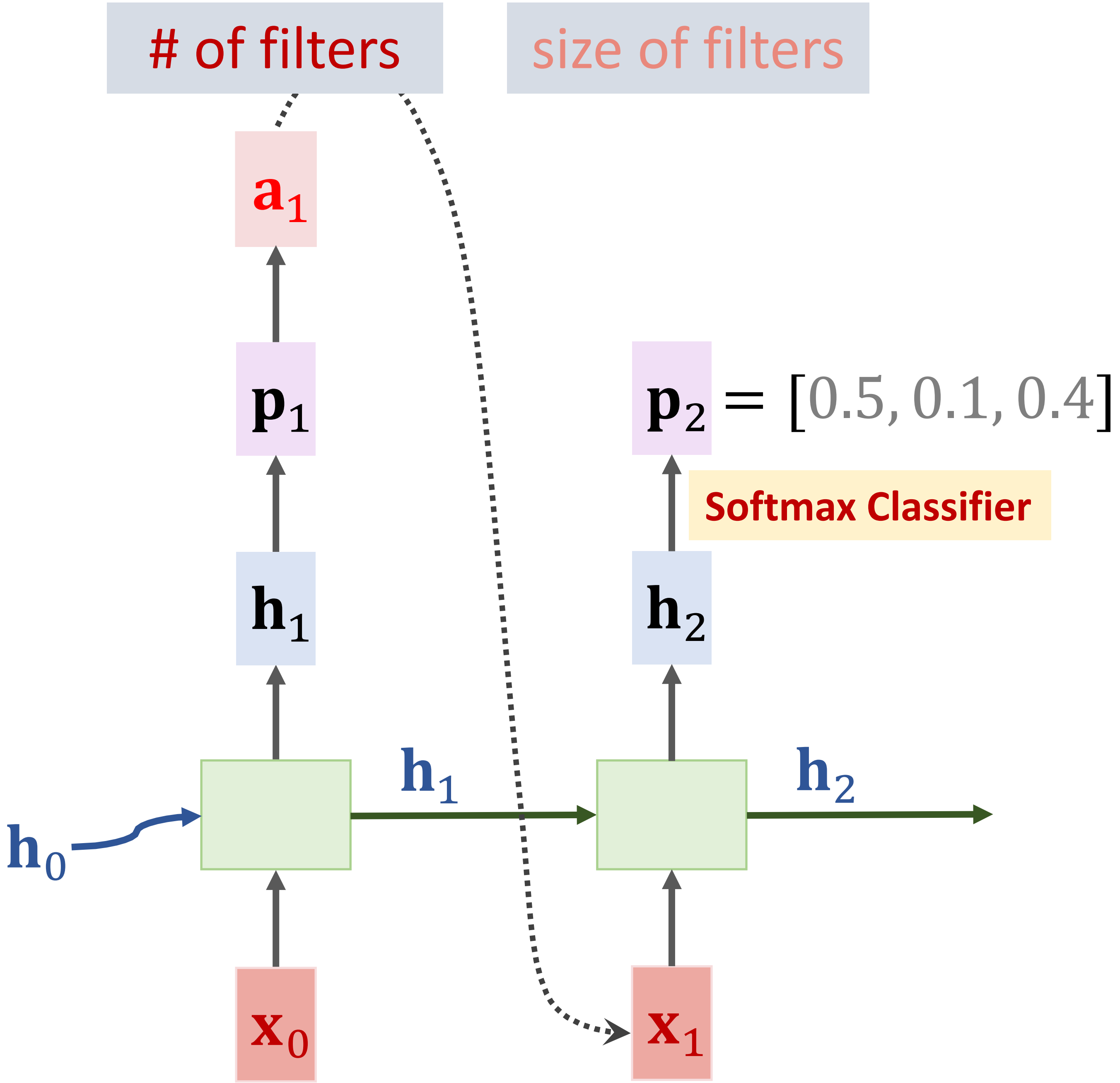
$$= \tanh\left(\mathbf{W} \cdot \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{x}_1 \end{bmatrix} + \mathbf{b}\right)$$



$\mathbf{a}_1$
$\mathbf{p}_1$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_0$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{x}_0$
$\mathbf{x}_1$

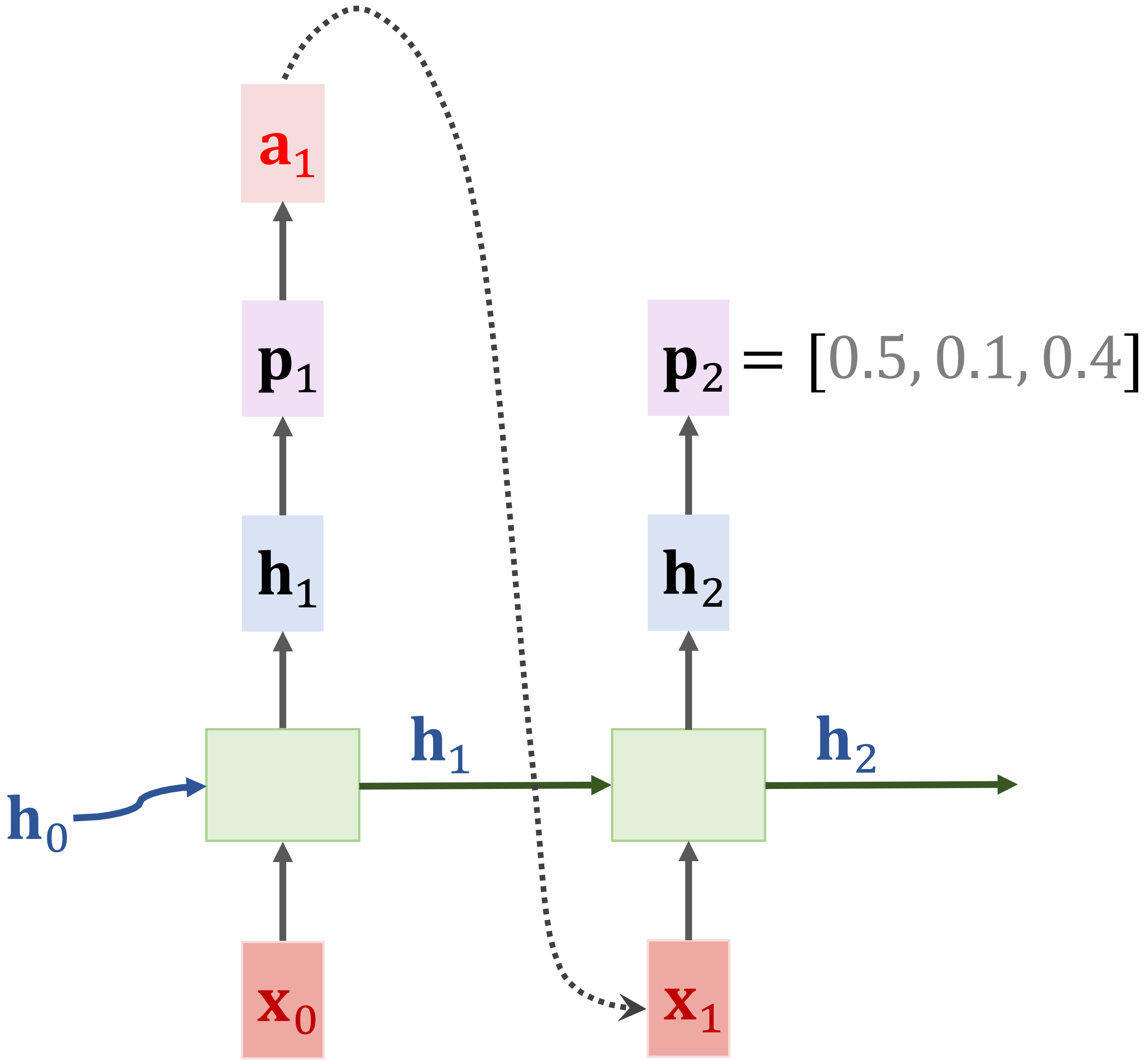


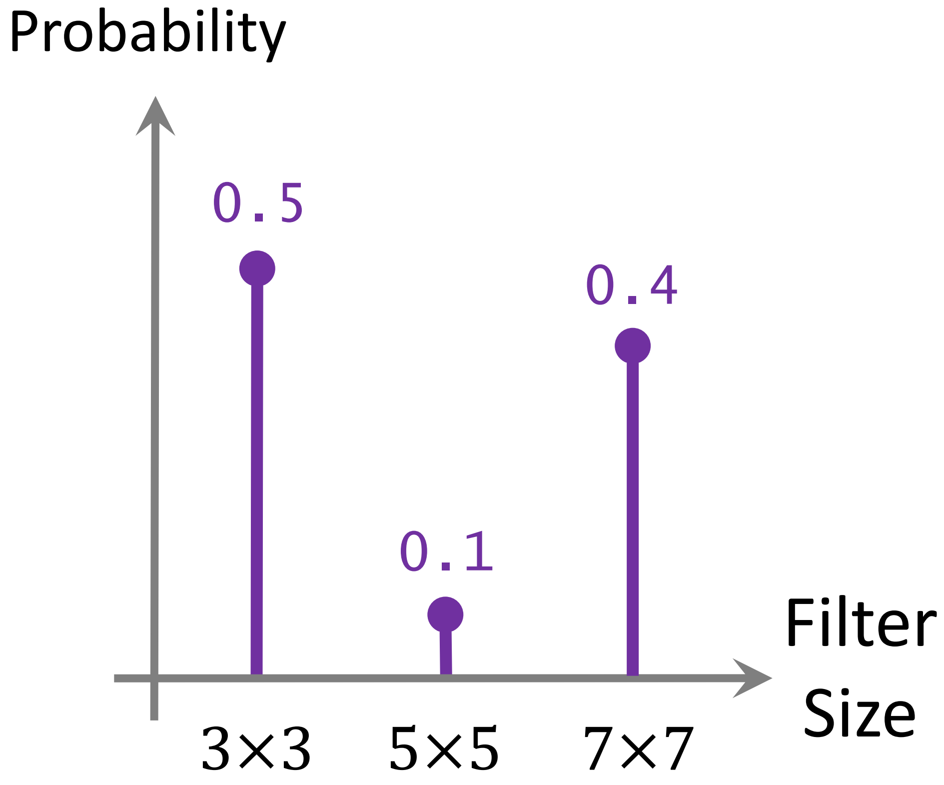
$\mathbf{a}_1$
$\mathbf{p}_1$
$\mathbf{p}_2 = [0.5, 0.1, 0.4]$
Softmax Classifier
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_0$
$\mathbf{x}_0$
$\mathbf{x}_1$



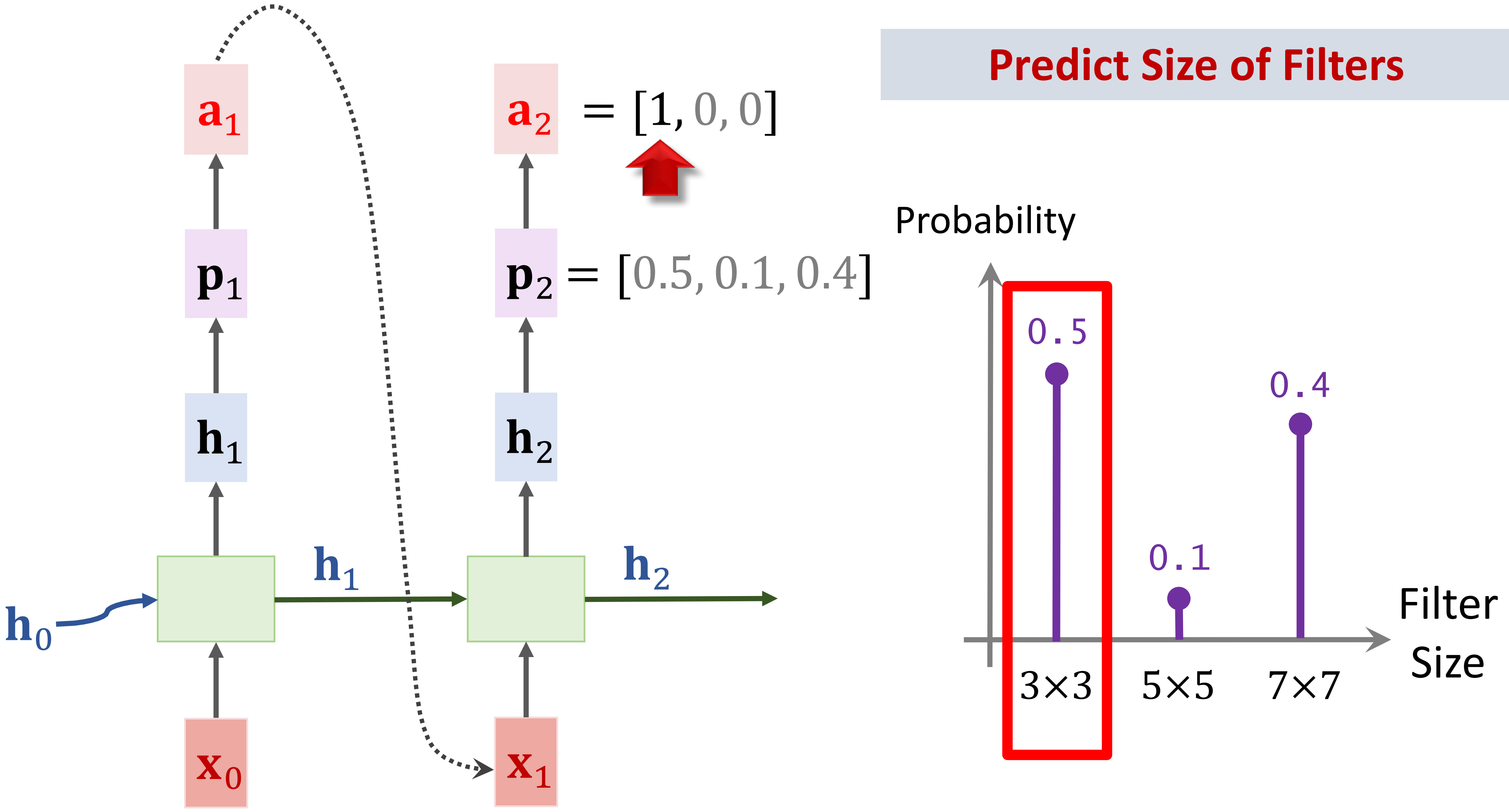
# of filters
size of filters
$\mathbf{a}_1$
$\mathbf{p}_1$
$\mathbf{p}_2 = [0.5, 0.1, 0.4]$
Softmax Classifier
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_0$
$\mathbf{x}_0$
$\mathbf{x}_1$

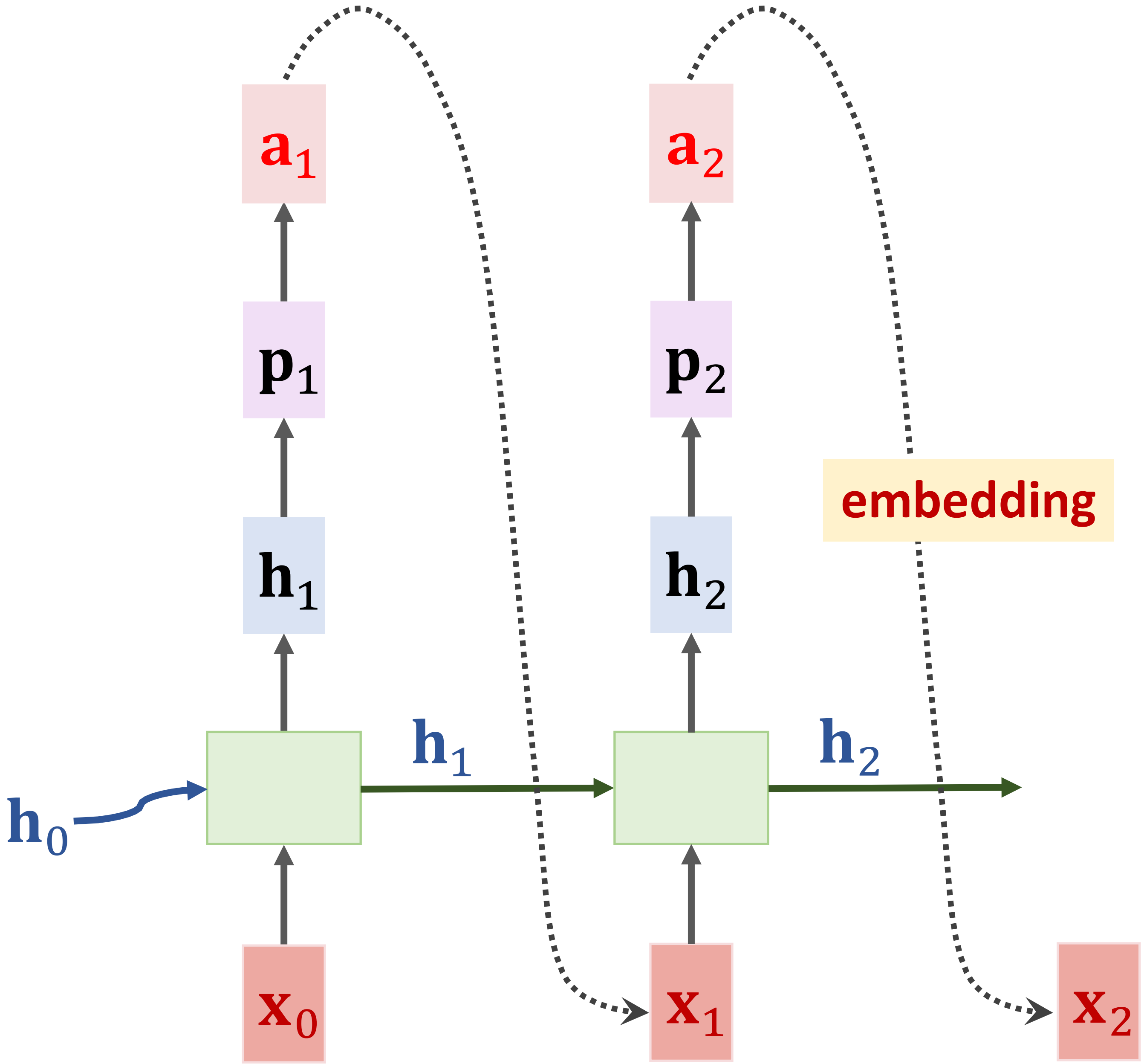
Predict Size of Filters



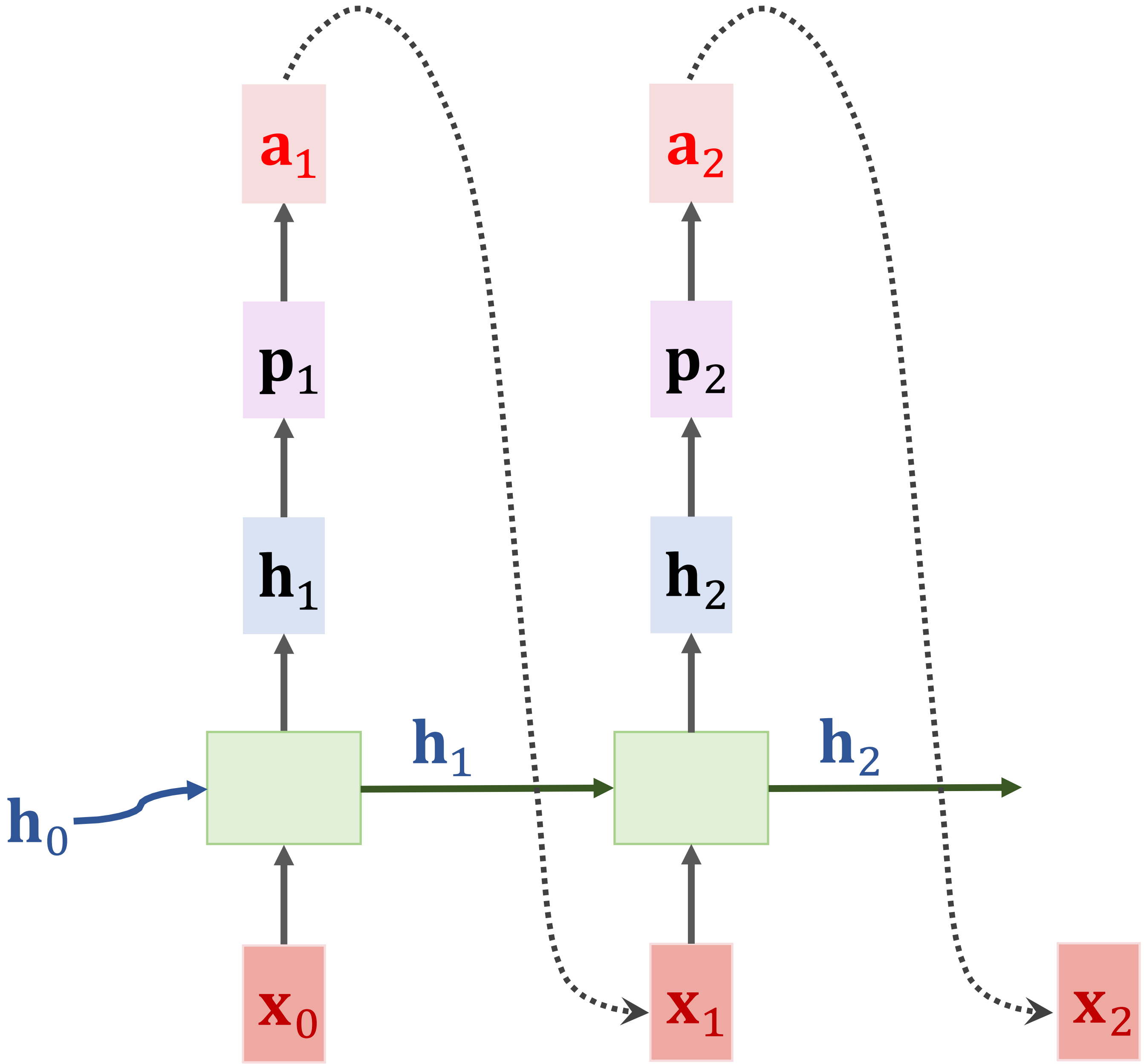


Predict Size of Filters



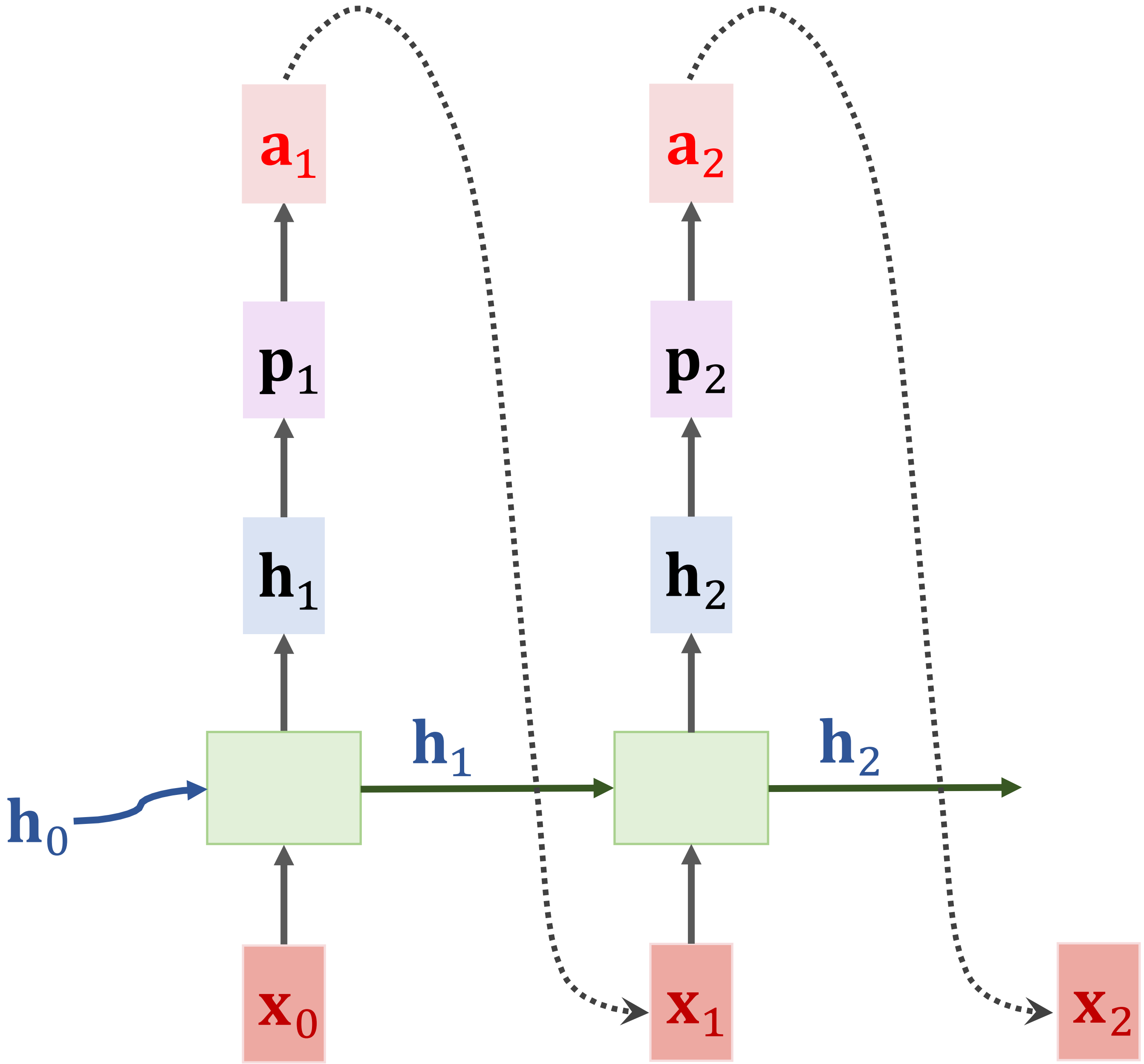


$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{p}_1$
$\mathbf{p}_2$
embedding
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_0$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$



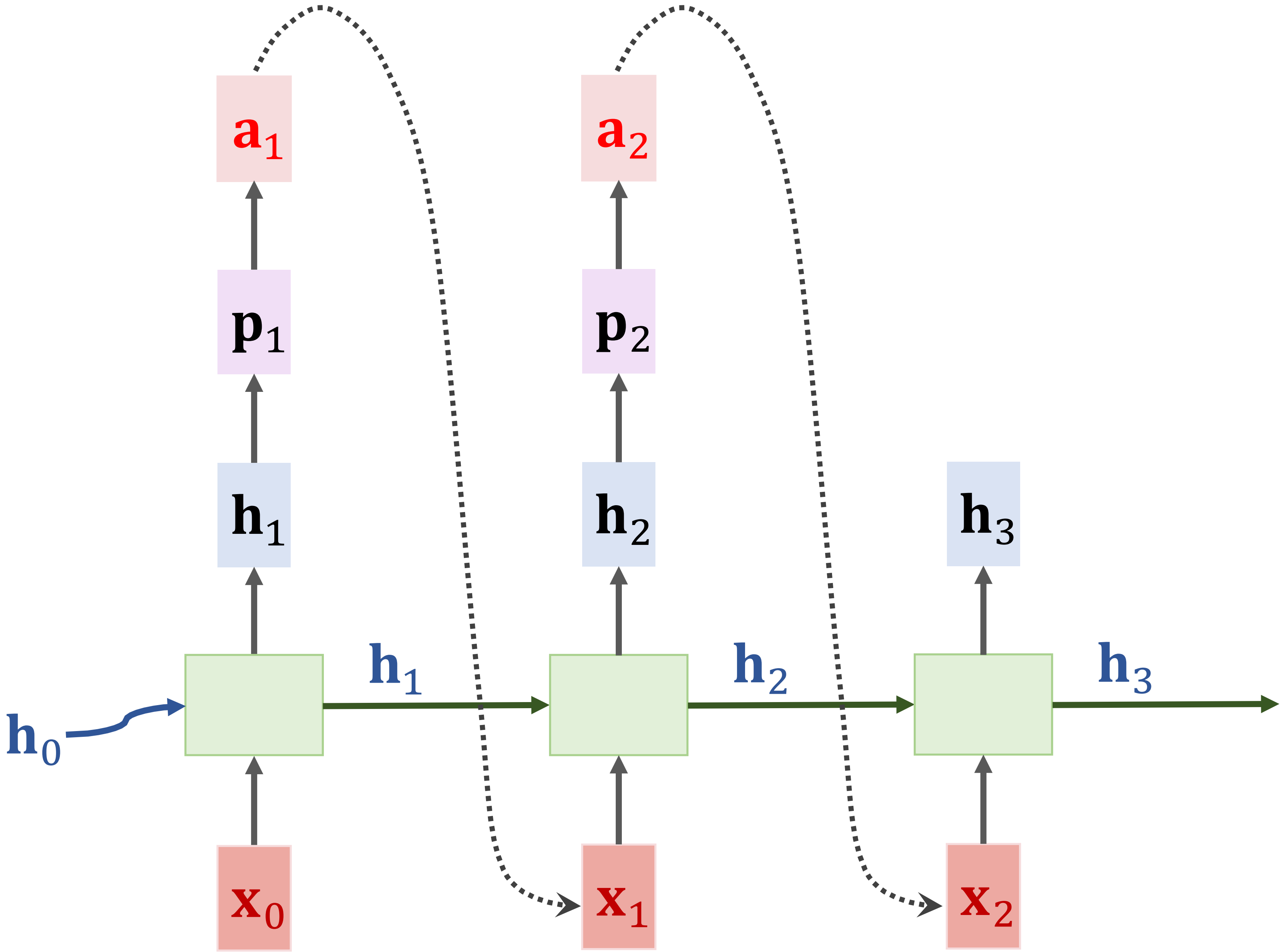
An embedding layer can be shared among the same task.

- E.g., $\mathbf{a}_2$, $\mathbf{a}_5$, $\mathbf{a}_8$, $\mathbf{a}_{11}$ are for size of filters.
- They can share embedding layer.

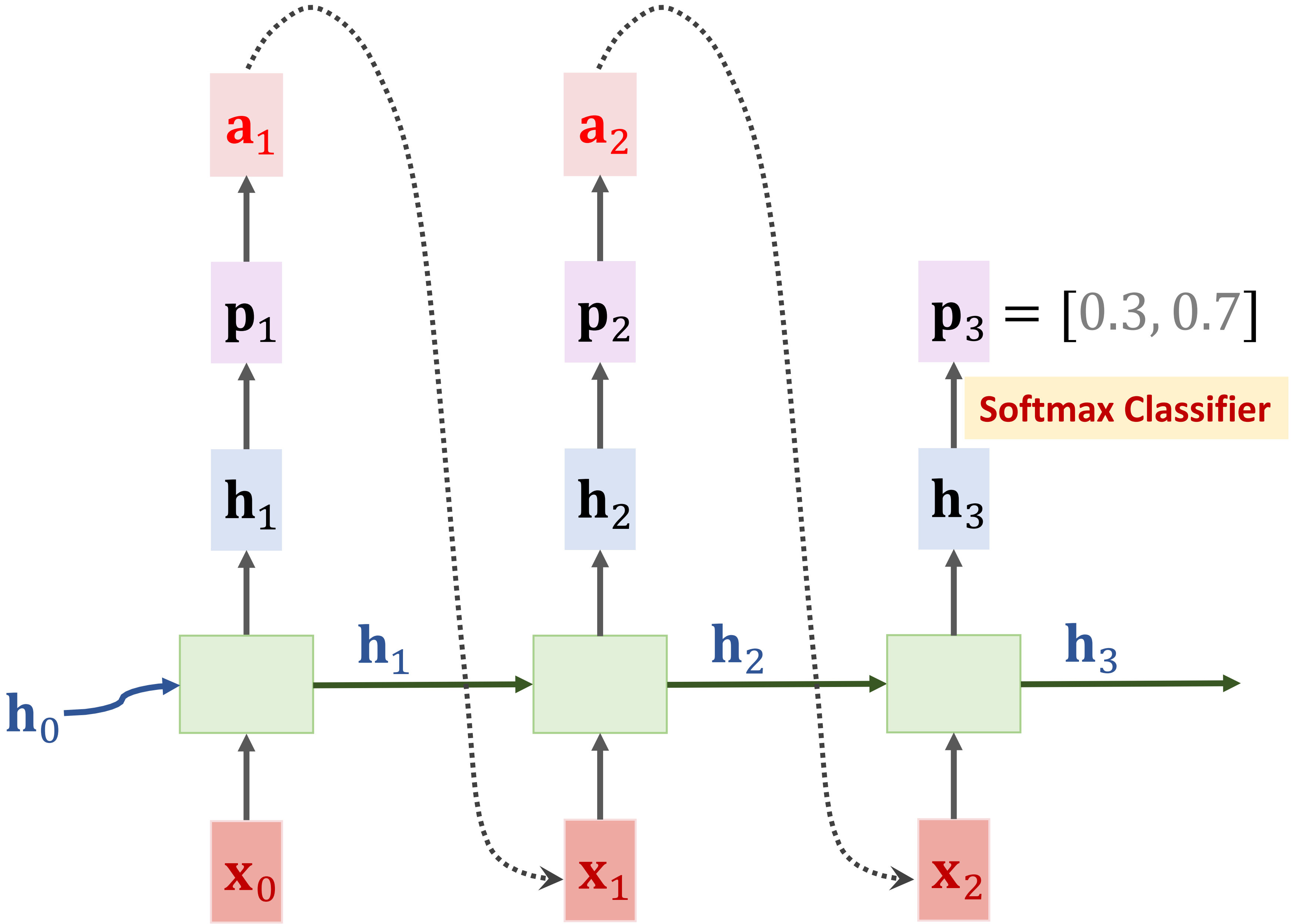


Different tasks do not share embedding layers.

- E.g., $\mathbf{a}_1$ and $\mathbf{a}_2$ are for different tasks.
- They cannot share embedding layer.

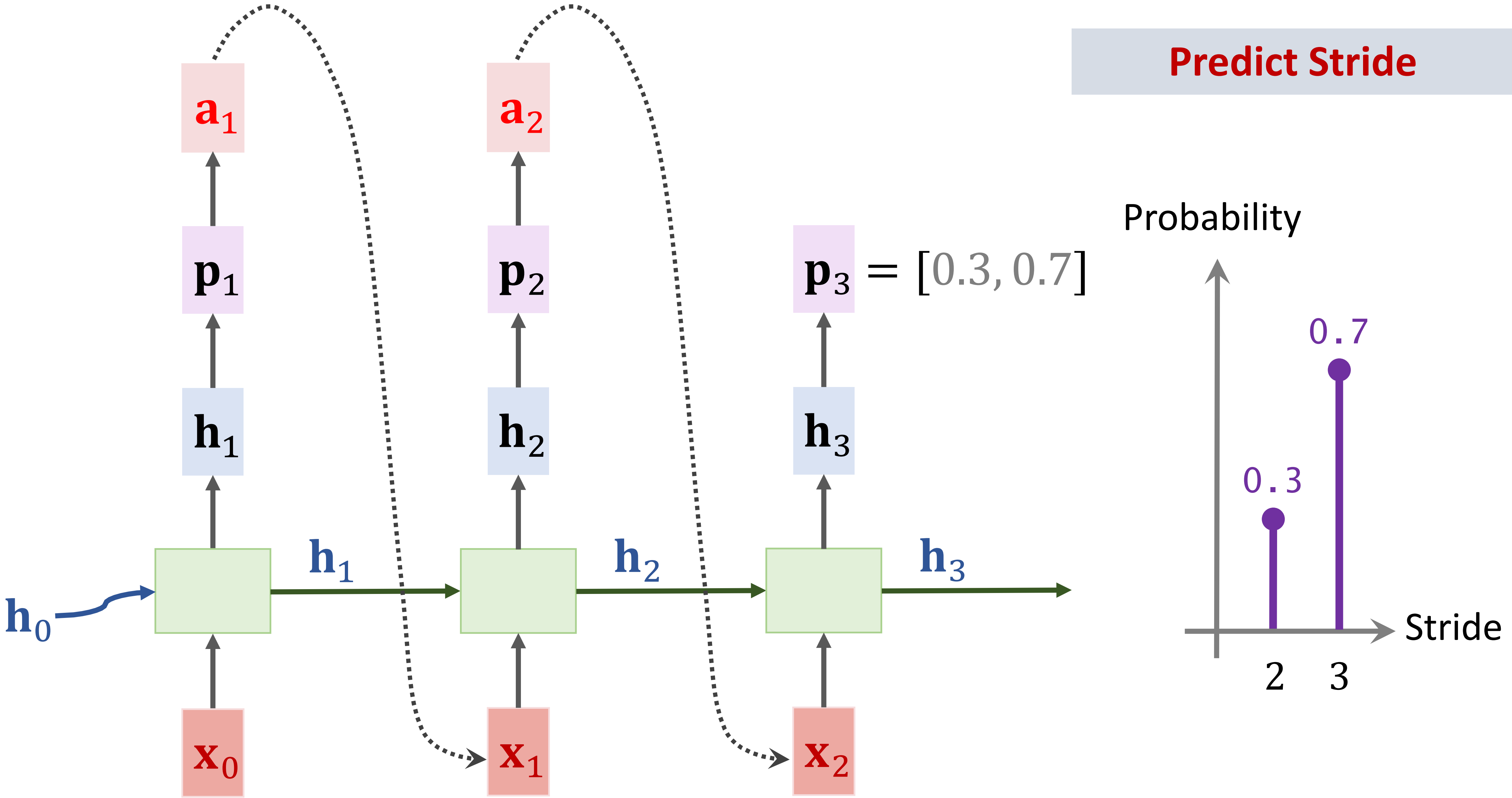


a1
a2
p1
p2
h1
h2
h3
h0
h1
h2
h3
x0
x1
x2

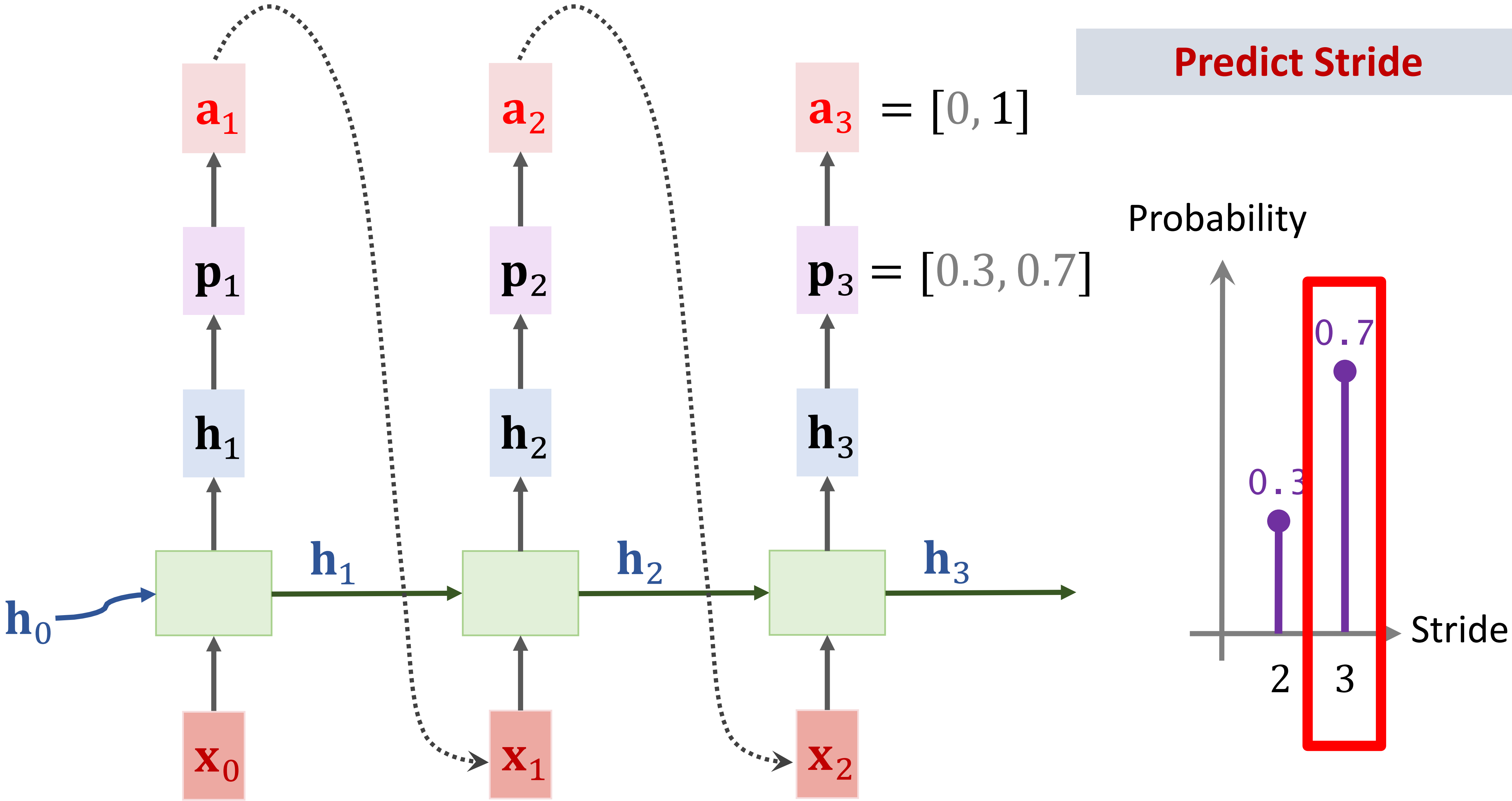


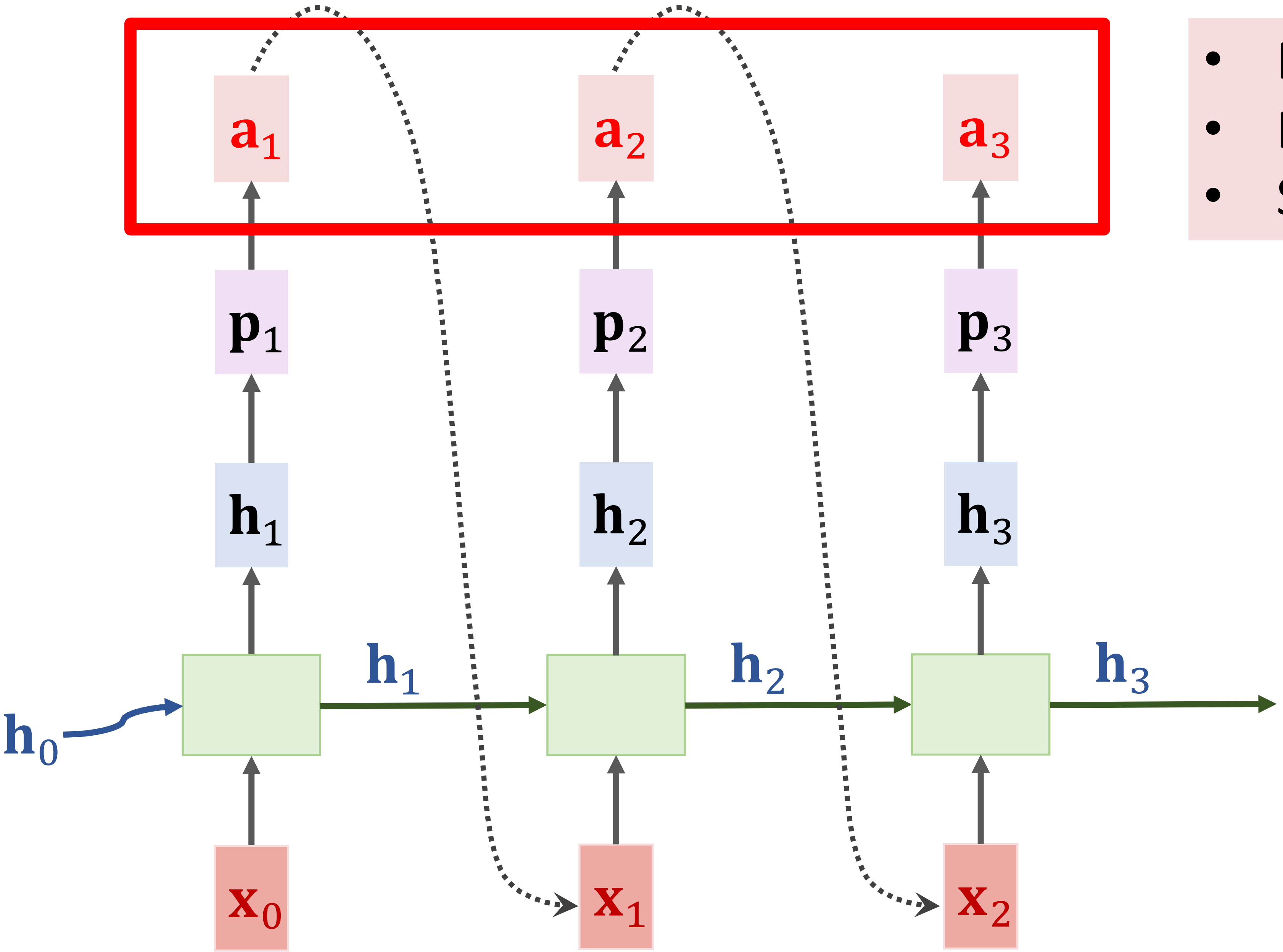
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{p}_1$
$\mathbf{p}_2$
$\mathbf{p}_3 = [0.3, 0.7]$
Softmax Classifier
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_0$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$

Predict Stride

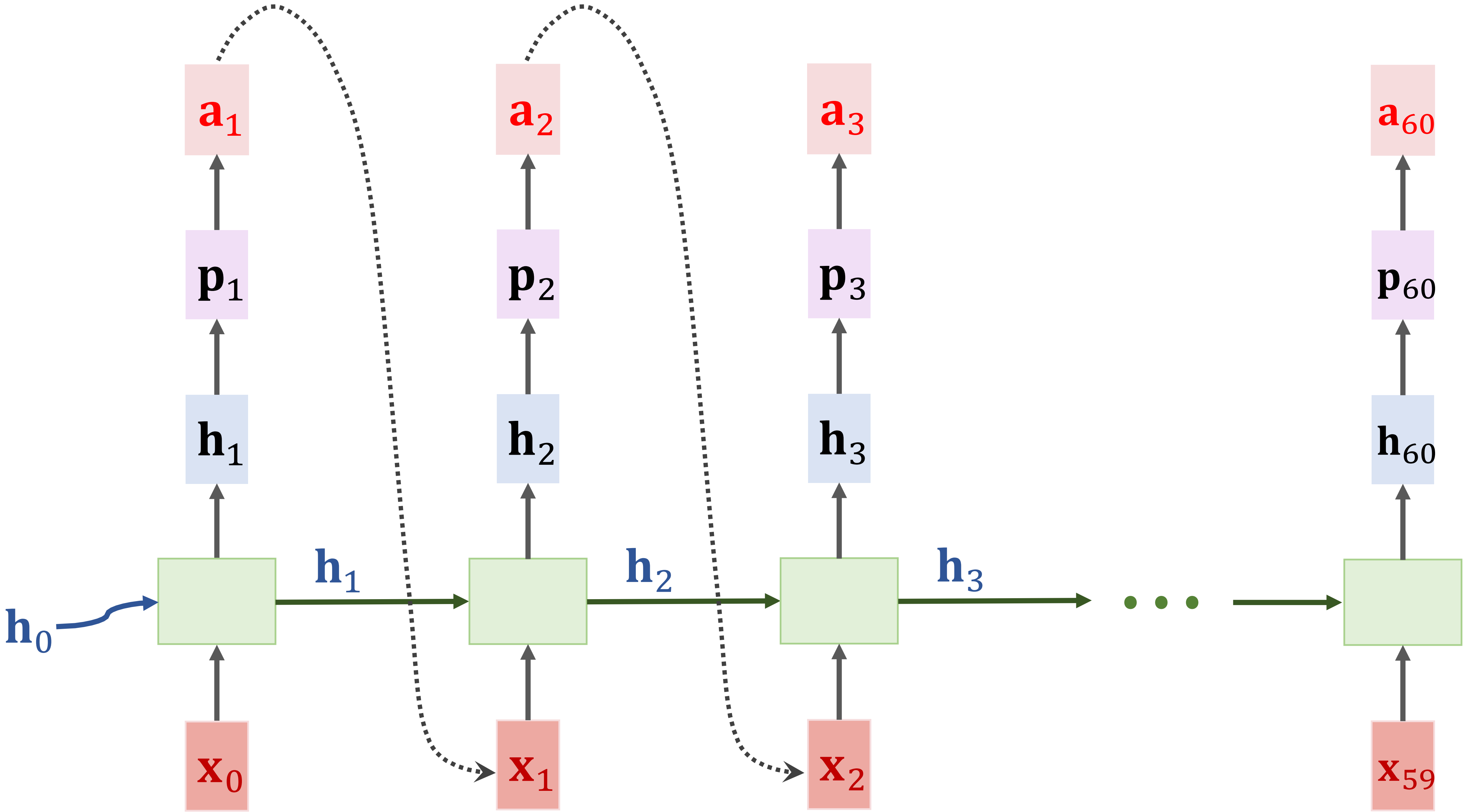


Predict Stride

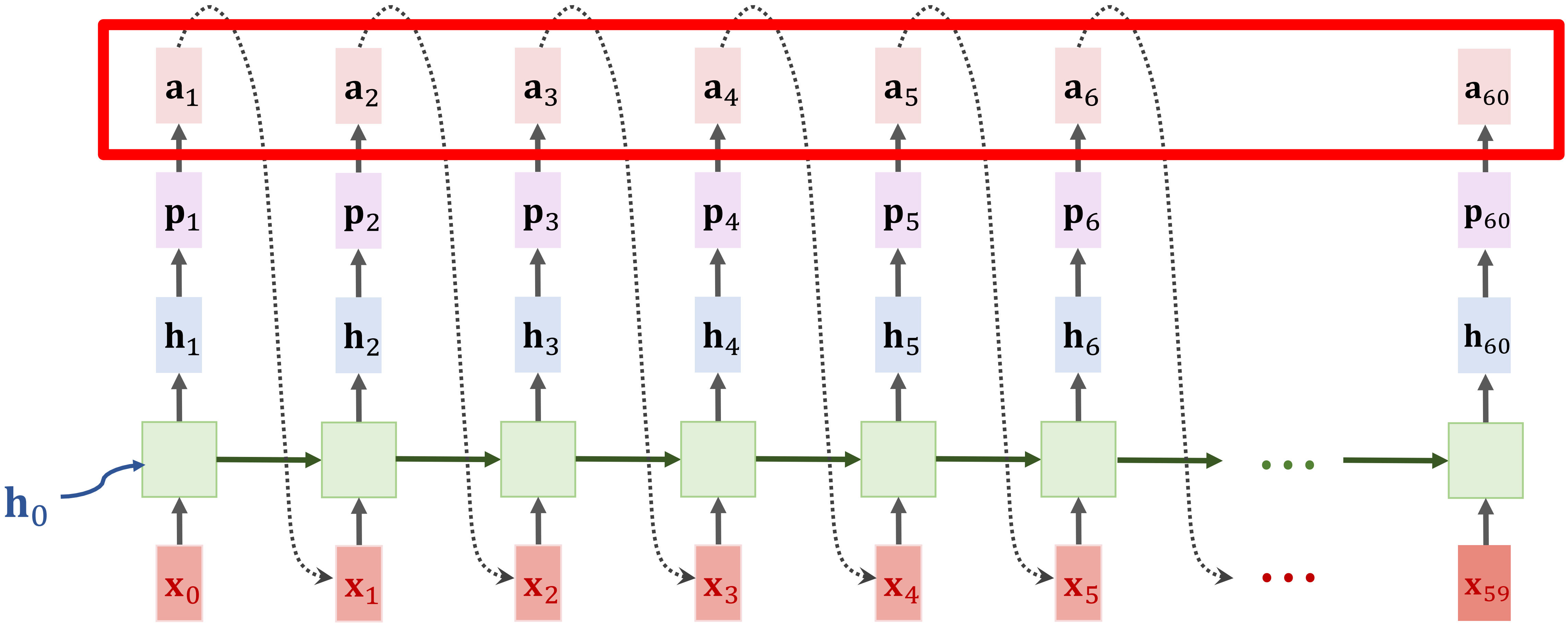




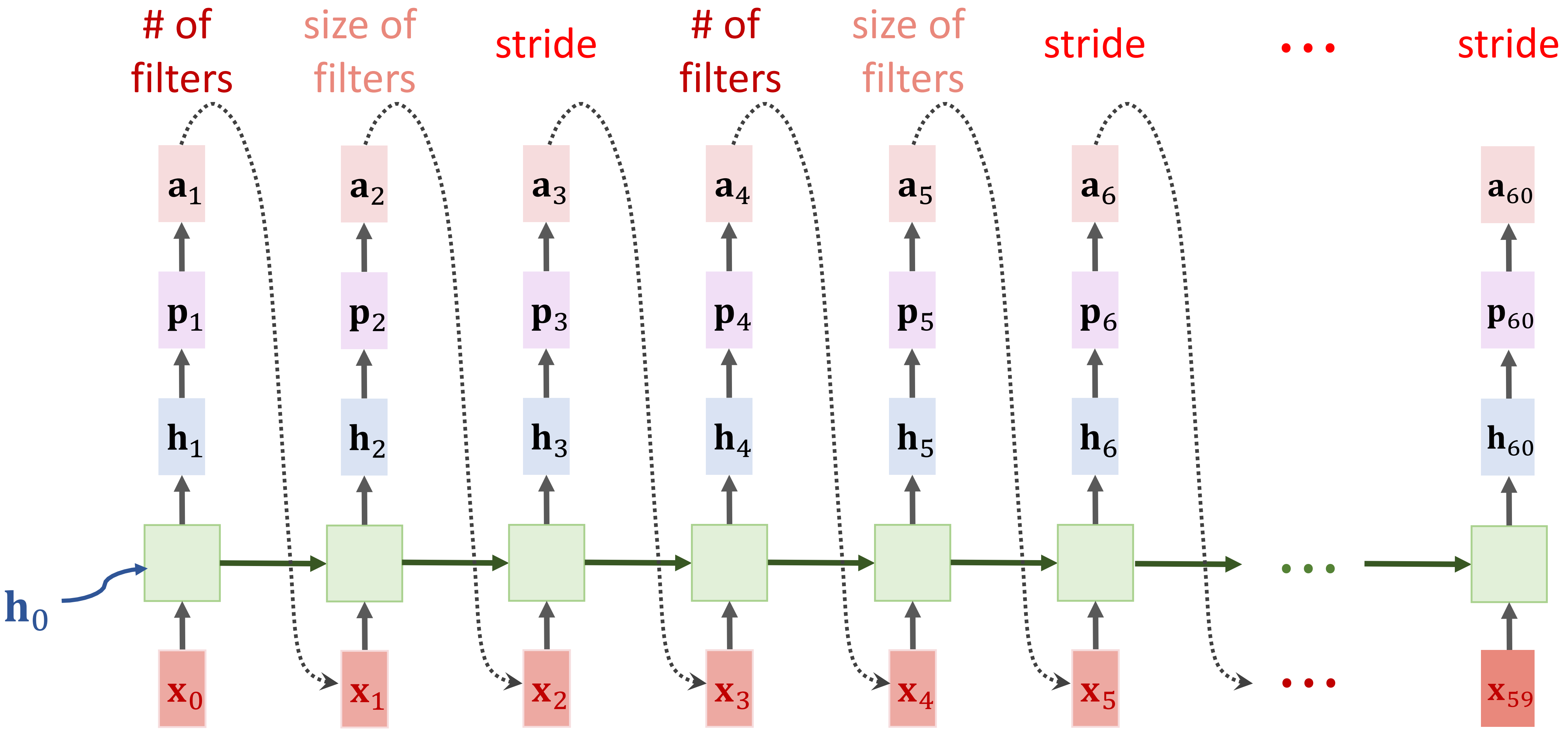
- Filter number = 36
- Filter size = 3×3
- Stride = 3



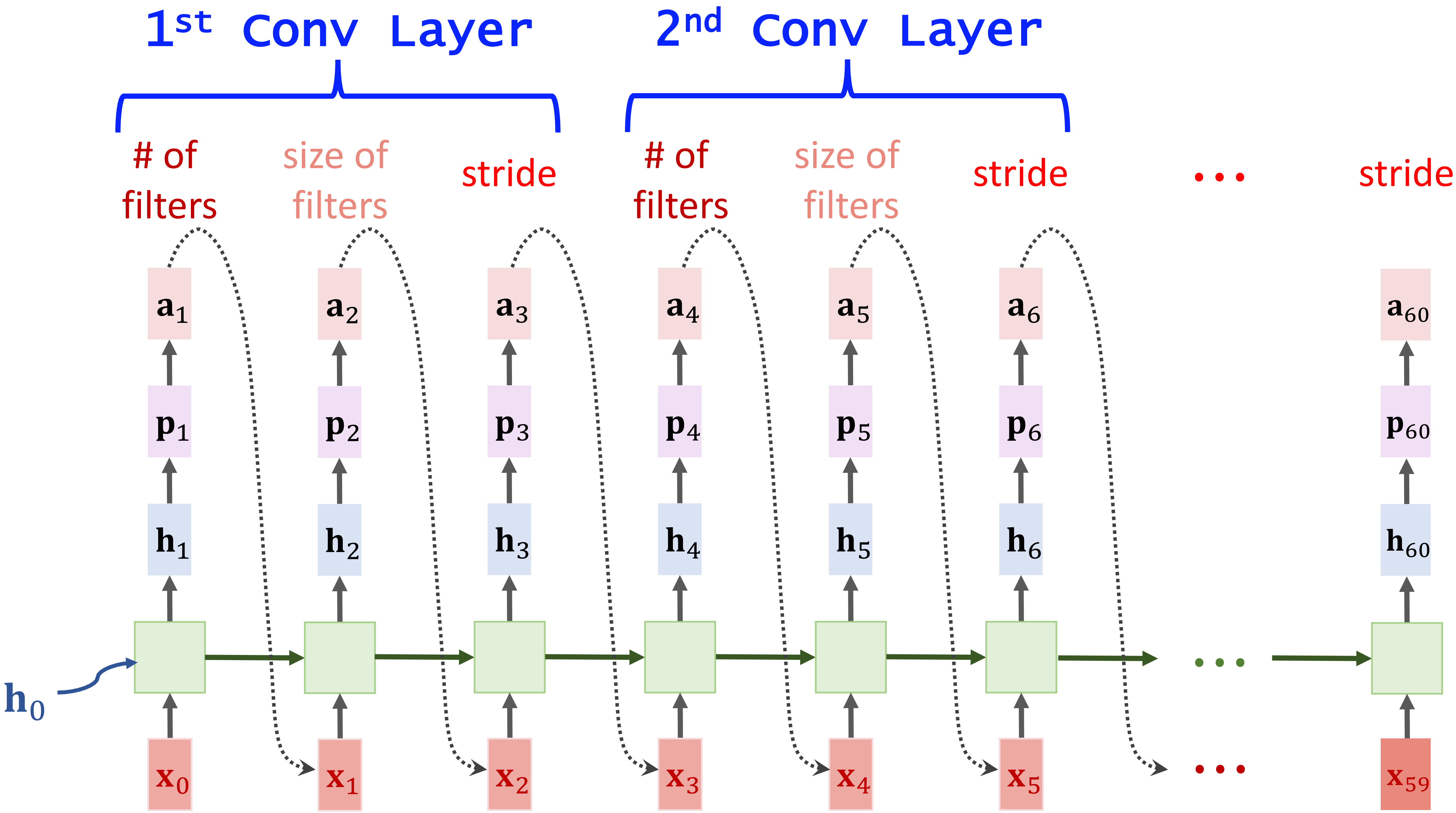
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
$\mathbf{a}_{60}$
$\mathbf{p}_1$
$\mathbf{p}_2$
$\mathbf{p}_3$
$\mathbf{p}_{60}$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_{60}$
$\mathbf{h}_0$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_{59}$



$\mathbf{a}_1$ $\mathbf{a}_2$ $\mathbf{a}_3$ $\mathbf{a}_4$ $\mathbf{a}_5$ $\mathbf{a}_6$ $\mathbf{a}_{60}$
$\mathbf{p}_1$ $\mathbf{p}_2$ $\mathbf{p}_3$ $\mathbf{p}_4$ $\mathbf{p}_5$ $\mathbf{p}_6$ $\mathbf{p}_{60}$
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$ $\mathbf{h}_5$ $\mathbf{h}_6$ $\mathbf{h}_{60}$
$\mathbf{h}_0$
$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\mathbf{x}_5$ $\mathbf{x}_{59}$



# of filters
size of filters
stride
# of filters
size of filters
stride
stride
$\mathbf{a}_1$ $\mathbf{a}_2$ $\mathbf{a}_3$ $\mathbf{a}_4$ $\mathbf{a}_5$ $\mathbf{a}_6$ $\mathbf{a}_{60}$
$\mathbf{p}_1$ $\mathbf{p}_2$ $\mathbf{p}_3$ $\mathbf{p}_4$ $\mathbf{p}_5$ $\mathbf{p}_6$ $\mathbf{p}_{60}$
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$ $\mathbf{h}_5$ $\mathbf{h}_6$ $\mathbf{h}_{60}$
$\mathbf{h}_0$
$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\mathbf{x}_5$ $\mathbf{x}_{59}$



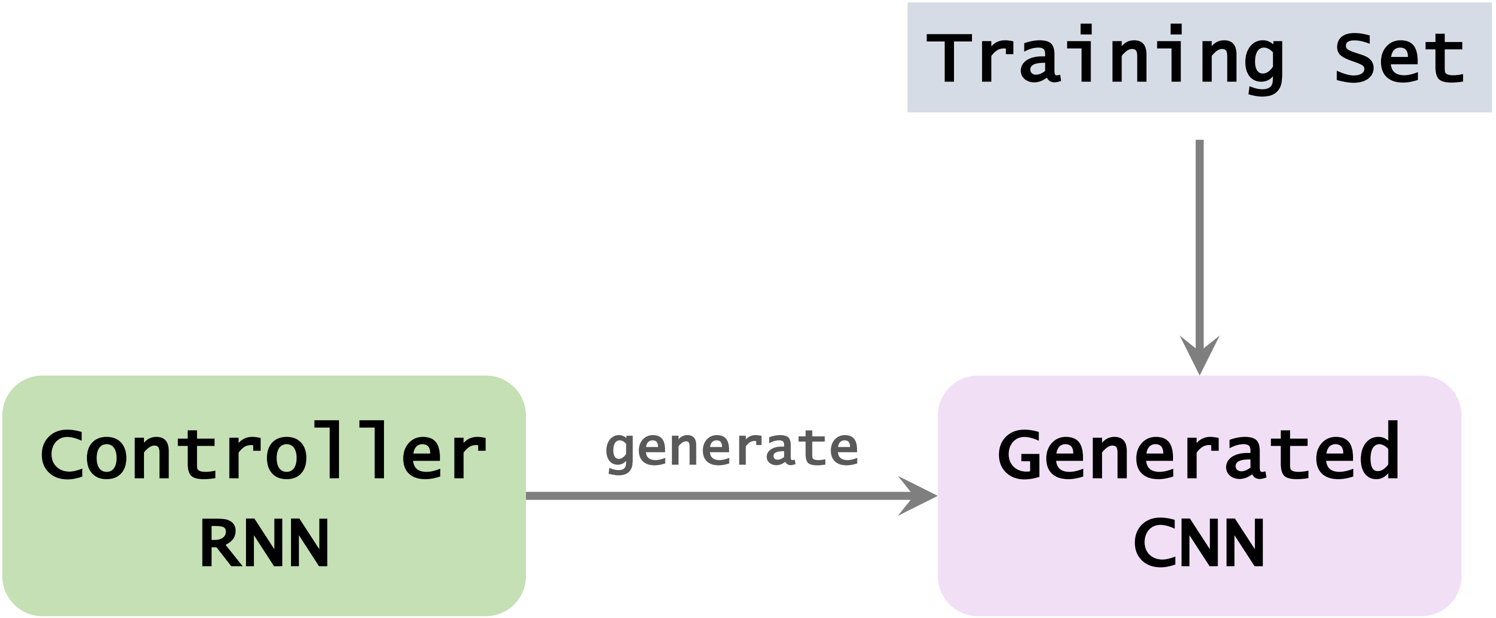
1st Conv Layer
2nd Conv Layer
# of filters
size of filters
stride
# of filters
size of filters
stride
...
stride
a_1 a_2 a_3 a_4 a_5 a_6 a_{60}
p_1 p_2 p_3 p_4 p_5 p_6 p_{60}
h_1 h_2 h_3 h_4 h_5 h_6 h_{60}
h_0
...
x_0 x_1 x_2 x_3 x_4 x_5 ... x_{59}

Training Controller RNN

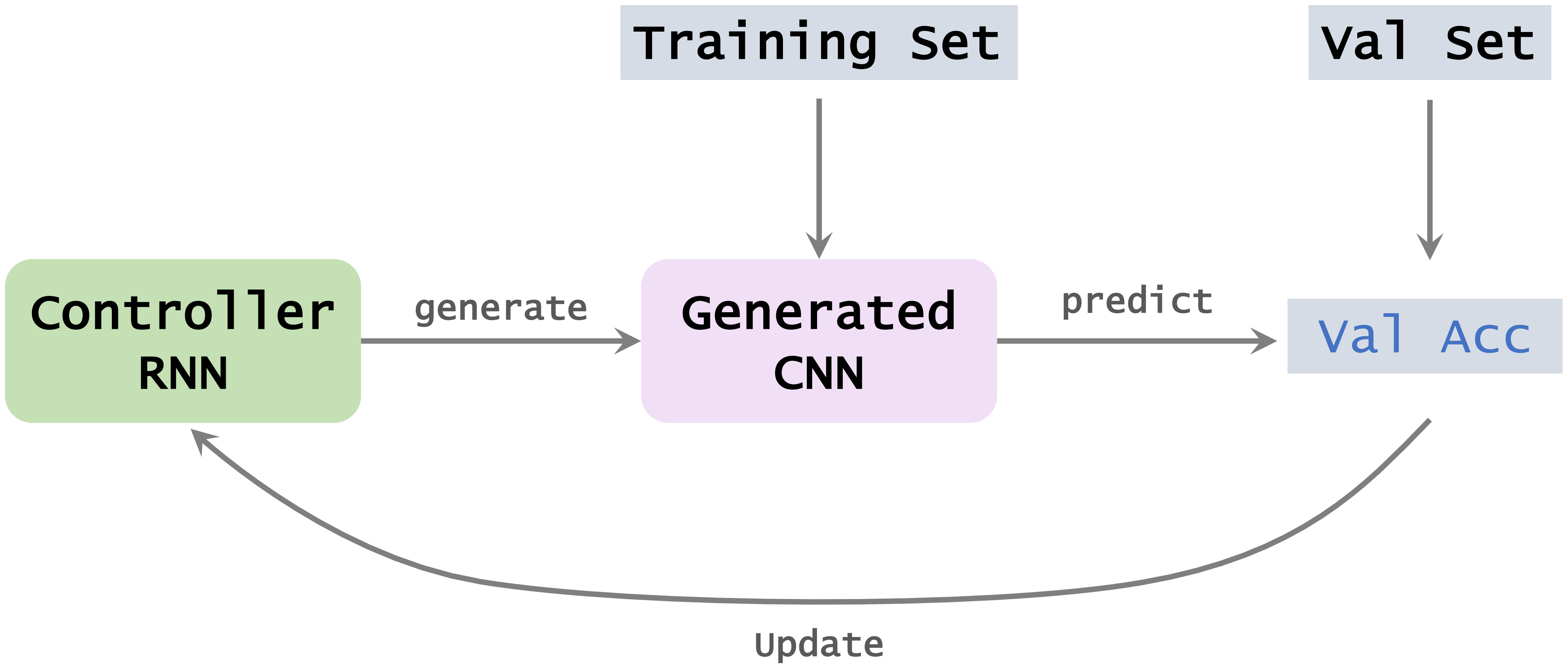
How to train the controller RNN?

- The controller RNN outputs the hyper-parameters of a CNN.
- With the hyper-parameters at hand, instantiate a CNN.
- Train the CNN on a dataset, e.g., CIFAR-10, ImageNet, etc.
- Compute validation accuracy on a held-out dataset.
- Validation accuracy is the supervision for training the controller RNN.

How to train the controller RNN?



How to train the controller RNN?



Challenges

- r: objective function (to maximize).
- $\boldsymbol{\theta}$: optimization variable.

Challenges

- r: objective function (to maximize).
- $\boldsymbol{\theta}$: optimization variable.

- What if r is a differentiable function of $\boldsymbol{\theta}$?
- Update $\boldsymbol{\theta}$ by gradient ascent:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \boxed{\frac{\partial r}{\partial \boldsymbol{\theta}}.}$$

Challenges

- r: objective function (to maximize).
- $\boldsymbol{\theta}$: optimization variable.

- What if r is a differentiable function of $\boldsymbol{\theta}$?
- Update $\boldsymbol{\theta}$ by gradient ascent:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \cdot \frac{\partial\, r}{\partial\, \boldsymbol{\theta}}.$$

- However, if r is **not** a differentiable function of $\boldsymbol{\theta}$, then we cannot use the gradient to update $\boldsymbol{\theta}$.

Challenges

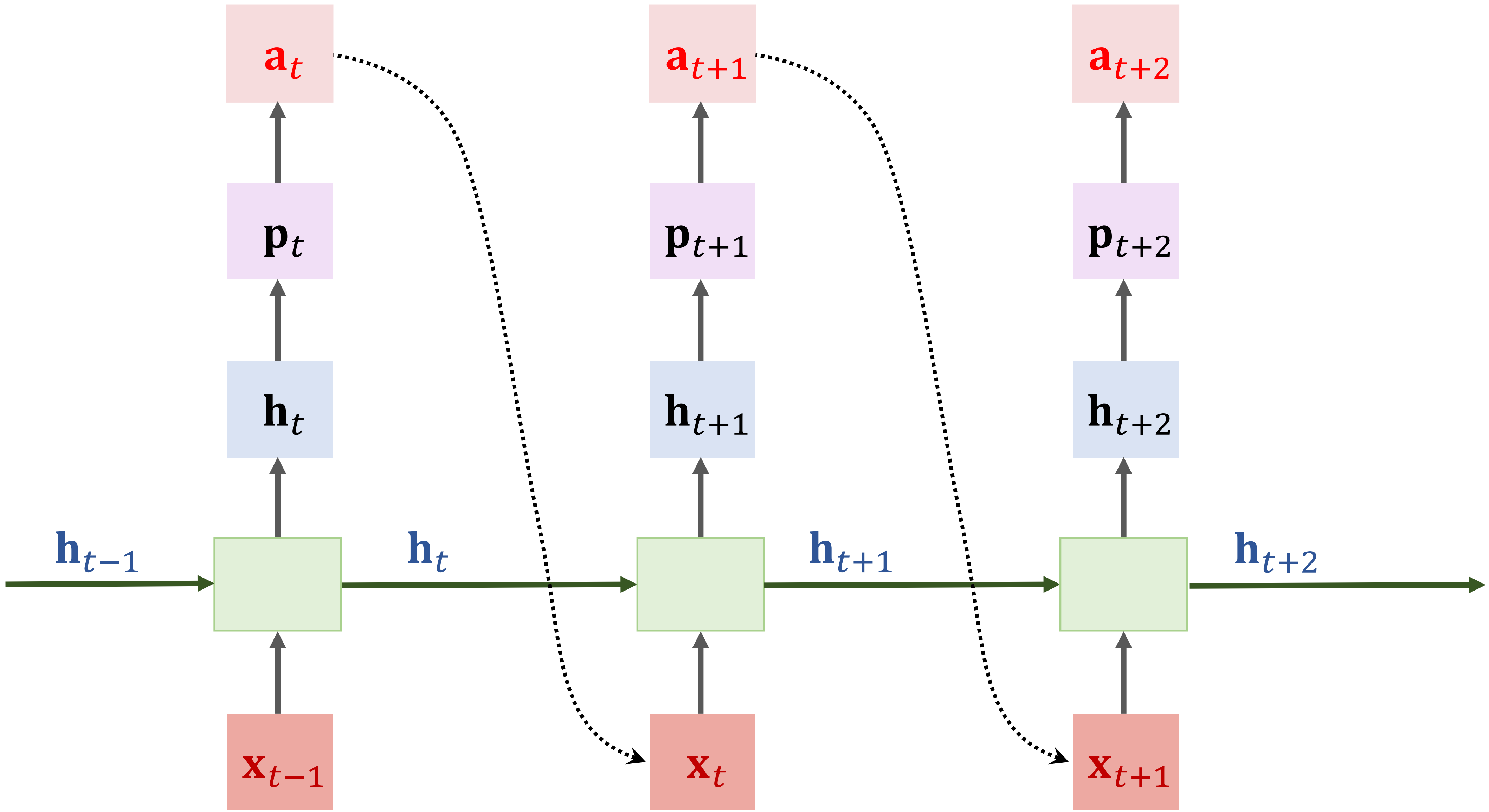
- r: objective function (to maximize). ⟺ Validation Accuracy
- $\boldsymbol{\theta}$: optimization variable. ⟺ Controller RNN Parameters

- Validation accuracy (r) is **not** a differentiable function of the controller RNN parameters ($\boldsymbol{\theta}$).
- They have to use reinforcement learning.

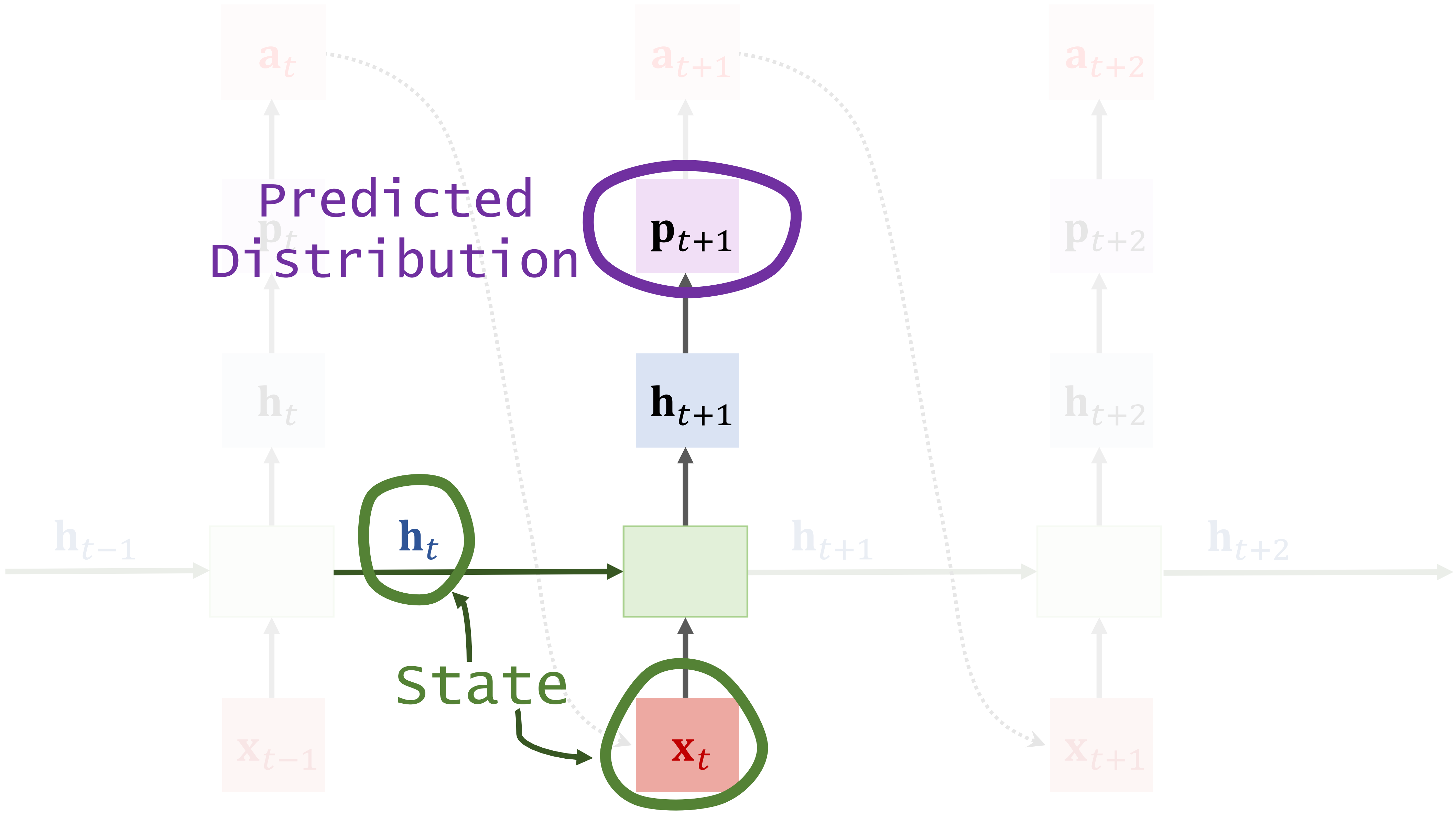
Reinforcement Learning

- **Objective:** Improve the controller RNN so that validation accuracies improve over time.
- **Rewards:** validation accuracies.
- **Policy function:** the controller RNN.
- Improve the policy function by policy gradient ascent.

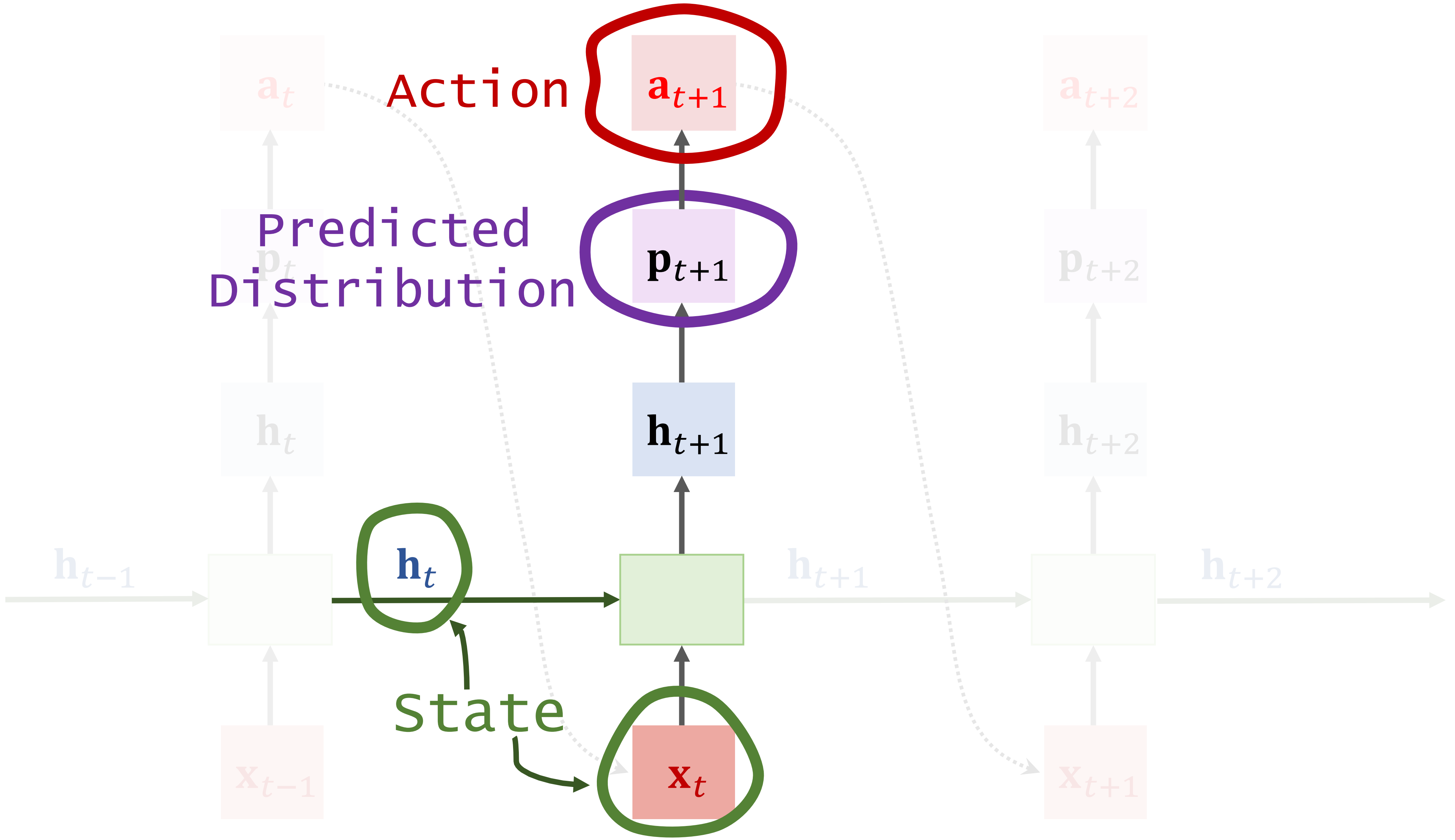
Policy Function



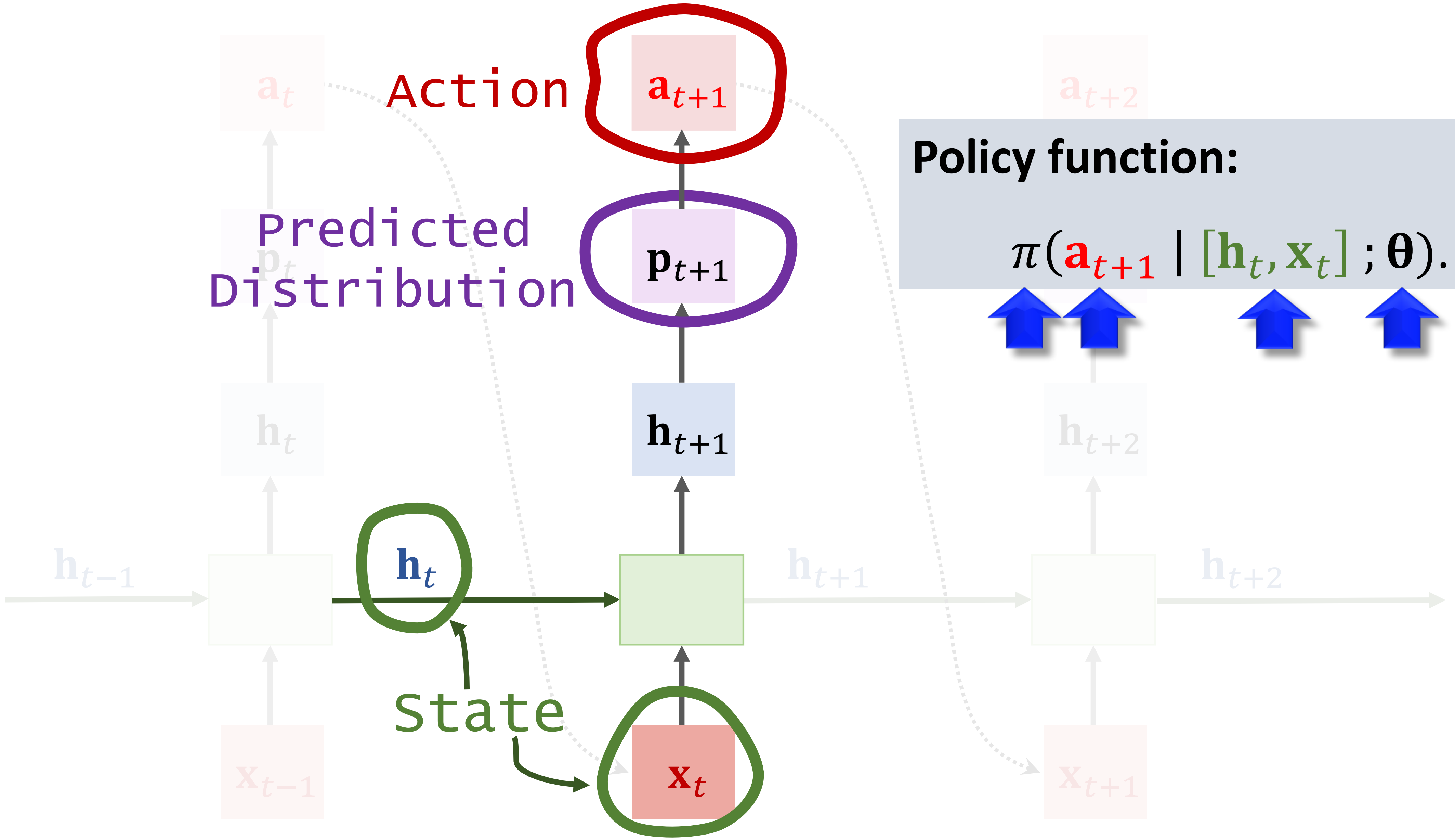
Policy Function



Policy Function



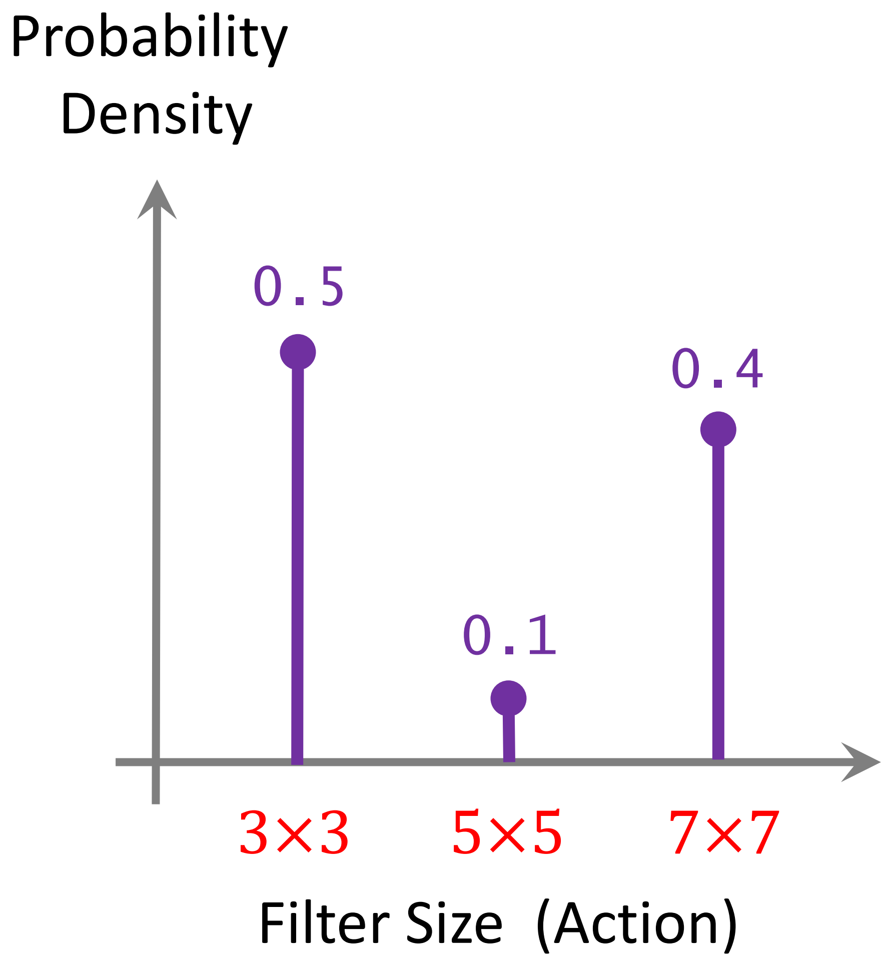
Policy Function



Policy function:

$$\pi(\mathbf{a}_{t+1} \mid [\mathbf{h}_t, \mathbf{x}_t]\,; \boldsymbol{\theta}).$$

Policy Function



Policy function:

$$\pi(\mathbf{a}_{t+1} \mid [\mathbf{h}_t, \mathbf{x}_t]\,; \boldsymbol{\theta}).$$

- $\pi(\text{“}3\times3\text{”} \mid [\mathbf{h}_t, \mathbf{x}_t]\,; \boldsymbol{\theta}) = 0.5.$
- $\pi(\text{“}5\times5\text{”} \mid [\mathbf{h}_t, \mathbf{x}_t]\,; \boldsymbol{\theta}) = 0.1.$
- $\pi(\text{“}7\times7\text{”} \mid [\mathbf{h}_t, \mathbf{x}_t]\,; \boldsymbol{\theta}) = 0.4.$

Reward & Return

- Suppose the controller RNN runs 60 steps.
- The first 59 rewards are zeros: $r_1 = r_2 = \cdots = r_{59} = 0.$
- The last reward is the validation accuracy: $r_{60} = \text{ValAcc}.$

Reward & Return

- Suppose the controller RNN runs 60 steps.
- The first 59 rewards are zeros: $r_1 = r_2 = \cdots = r_{59} = 0.$
- The last reward is the validation accuracy: $r_{60} = \text{ValAcc}.$
- Return (aka cumulative reward) is defined as:

$$\underline{u_t} = \underline{r_t + r_{t+1} + r_{t+2} + \cdots + r_{59} + r_{60}}.$$

- Thus, all the returns are equal:

$$\underline{u_1 = u_2 = \cdots = u_{60} = \text{ValAcc}.}$$

REINFORCE Algorithm

- Approximate policy gradients by:

$$\frac{\partial \log \pi(\mathbf{a}_{t+1} \mid [\mathbf{h}_t, \mathbf{x}_t] ; \boldsymbol{\theta})}{\partial \, \boldsymbol{\theta}} \cdot u_t .$$

REINFORCE Algorithm

- Approximate policy gradients by:

$$\frac{\partial \log \pi(\mathbf{a}_{t+1} \mid [\mathbf{h}_t, \mathbf{x}_t] ; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot u_t .$$

- Update trainable parameters by:

$$\underline{\boldsymbol{\theta}} \leftarrow \underline{\boldsymbol{\theta}} + \underline{\beta} \cdot \underline{\sum_{t=1}^{60} \frac{\partial \log \pi(\mathbf{a}_{t+1} \mid [\mathbf{h}_t, \mathbf{x}_t] ; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot u_t .}$$

Recap

- Run the controller RNN to generate the hyper-parameters of the 20 convolutional layers.
- Instantiate a CNN, train the CNN, and then obtain a validation accuracy (to be used as a reward.)
- REINFORCE algorithm uses the reward to update the policy function (i.e., the controller RNN.)
- Repeat this process thousands of times.

NAS is expensive!

- To update the controller RNN once, we need to train a CNN from scratch. (Once is already expensive.)
- 10,000+ updates to train the RNN well ➔ Train 10,000+ CNNs from scratch. (Extremely expensive!)
- The controller RNN itself has tuning hyper-parameters.
  - E.g., # of layers, size of $\mathbf{x}$, size of $\mathbf{h}$, etc.
  - Hyper-parameter tuning will make the overall time cost many times higher!

Thank You!