Recurrent Neural Networks, I

LING 574 Deep Learning for NLP

Shane Steinert-Threlkeld

Today's Plan

- Last time:
  - Computation graphs + backpropagation
  - Deep Averaging Network (DAN)
- Quick notes on edugrad
- Neural Probabilistic Language Model (feed-forward model)
- Additional Training Notes
  - Regularization
  - Early stopping
  - Hyper-parameter searching
- Intro to *Recurrent* Neural Networks

Announcements

- HW2 reference code now available
- Tests: hwX/test_all.py. NB: necessary, but not sufficient, to check correctness of your code. From command-line, run `pytest` from your HW directory, with environment activated.
- Implementing ops in edugrad:
  - You can use any numpy operations you want; goal is to understand forward/backward API
  - https://github.com/shanest/edugrad
  - Log: base e, don't need to do special handling of bad input arguments (like 0)
- Edugrad is installed in the course conda environment, so be sure to activate it

Decorators

- @tensor_op in edugrad code: what is this??
  - This converts \`Operation\`s into methods on \`Tensor\`s
  - Handles dynamic graph construction, the \`ctx\` magic, etc.
- [Python decorator]() (similar to [decorator]() design pattern)
  - Design pattern to extend an object with more functionality
  - Decorators *wrap* their arguments, add features
    - e.g. registering in a central DB
- In python, syntactic sugar:
  - With more complicated use cases
- Canonical examples: @classmethod, @staticmethod

```python
@my_decorator
def fn(...):


def fn(...):

fn = my_decorator(fn)
```

Decorator Demo

```
def printer(method, *args):
    def fn(*args):
        output = method(*args)
        print(f"Output: {output}")
    return fn

@printer
def add(a, b):
    return a + b

add(1, 2) # prints "Output: 3"
```

@tensor_op

```
def tensor_op(op: Operation) -> Callable[[List[Tensor]], Tensor]:
    """
    Takes an operation, and turns it into a callable function on Tensors.

    The resulting function implicitly builds the dynamic computation graph,
    including populating the Tensors' _backward methods, when called.
    """

    def fn(*inputs: List[Tensor], **kwargs) -> Tensor:
        ctx = []
        new_tensor = Tensor(
            op.forward(ctx, *[tensor.value for tensor in inputs], **kwargs),
            inputs,
            op.__name__,
        )

        def _backward():
            grads = op.backward(ctx, new_tensor.grad)
            for idx in range(len(inputs)):
                inputs[idx].grad += grads[idx]

        new_tensor._backward = _backward
        return new_tensor

    return fn
```

Recurrent Neural Networks

RNNs: high-level

RNNs: high-level

- Feed-forward networks: fixed-size input, fixed-size output
  - Previous classifier: average embeddings of words
  - Previous LM: *n*-gram assumption (i.e. fixed-size context of word embeddings)

RNNs: high-level

- Feed-forward networks: fixed-size input, fixed-size output
  - Previous classifier: average embeddings of words
  - Previous LM: *n*-gram assumption (i.e. fixed-size context of word embeddings)
- RNNs process *sequences* of vectors
  - Maintaining “hidden” state
  - Applying the same operation at each step

RNNs: high-level

- Feed-forward networks: fixed-size input, fixed-size output
  - Previous classifier: average embeddings of words
  - Previous LM: *n*-gram assumption (i.e. fixed-size context of word embeddings)
- RNNs process *sequences* of vectors
  - Maintaining “hidden” state
  - Applying the same operation at each step
- Different RNNs:
  - Different operations at each step
  - Operation also called “recurrent cell”
  - Other architectural considerations (e.g. depth; bidirectionally)

Long-distance dependencies, I: number

- Language modeling (fill-in-the-blank)
  - The keys ____
  - The keys on the table ____
  - The keys next to the book on top of the table ____
- To get the number on the verb, need to look at the subject, which can be very far away
  - And number can disagree with linearly-close nouns

Selectional Restrictions

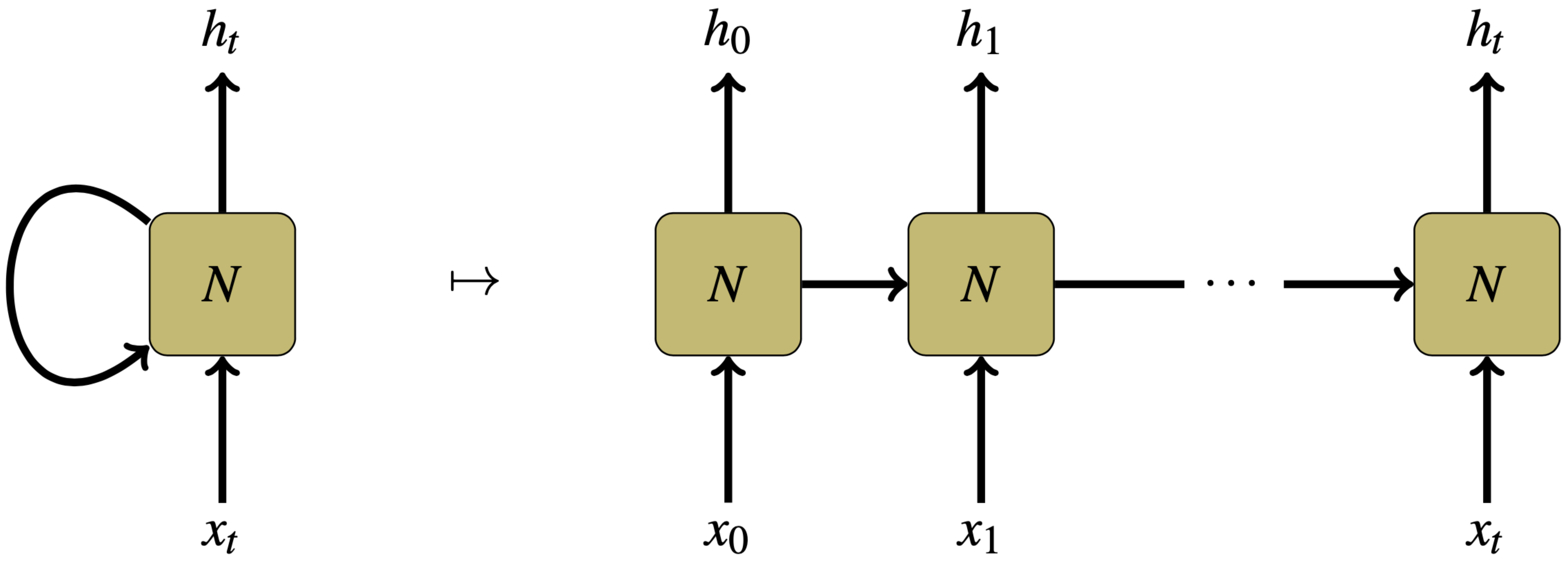
- The **family** moved from the city because they wanted a larger ____.
- The **team** moved from the city because they wanted a larger ____.

Selectional Restrictions

- The **family** moved from the city because they wanted a larger **house**.
- The **team** moved from the city because they wanted a larger **market**.

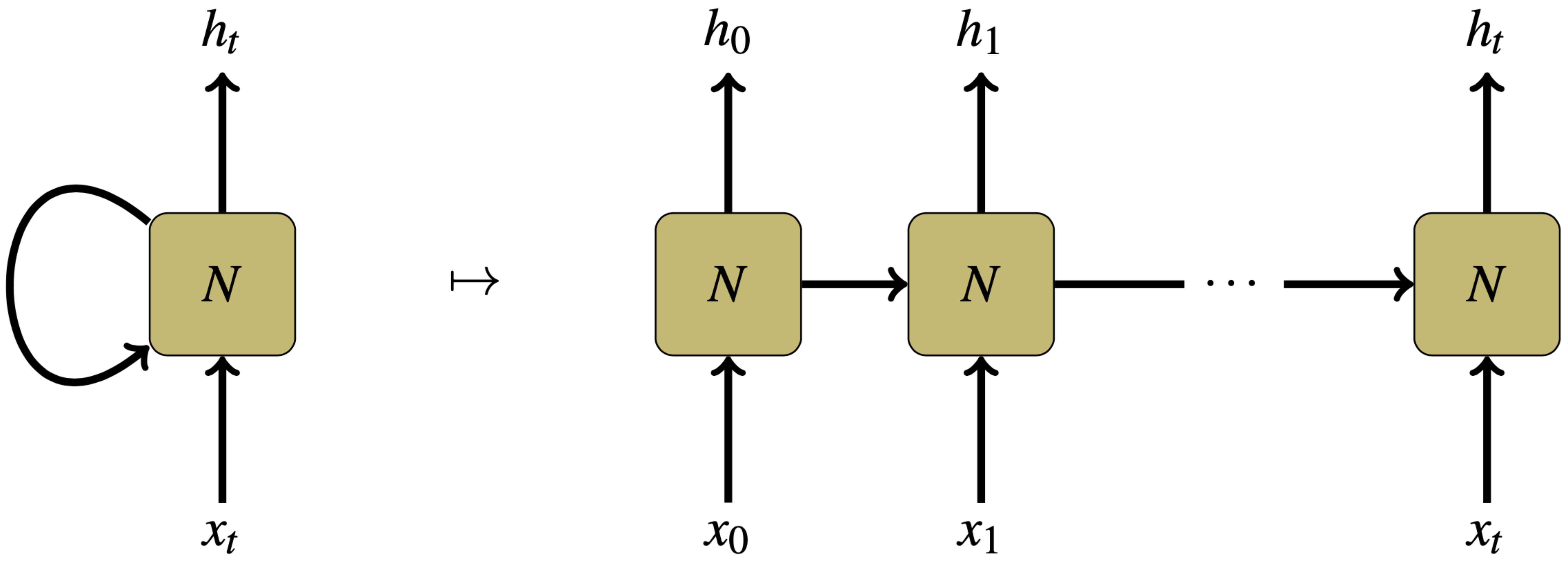
- Need models that can capture long-range dependencies like this.
- N-gram (whether count-based or neural) cannot. E.g., with n=4:
  - P(word | “they wanted a larger”)

RNNs



Steinert-Threlkeld and Szymanik 2019; Olah 2015

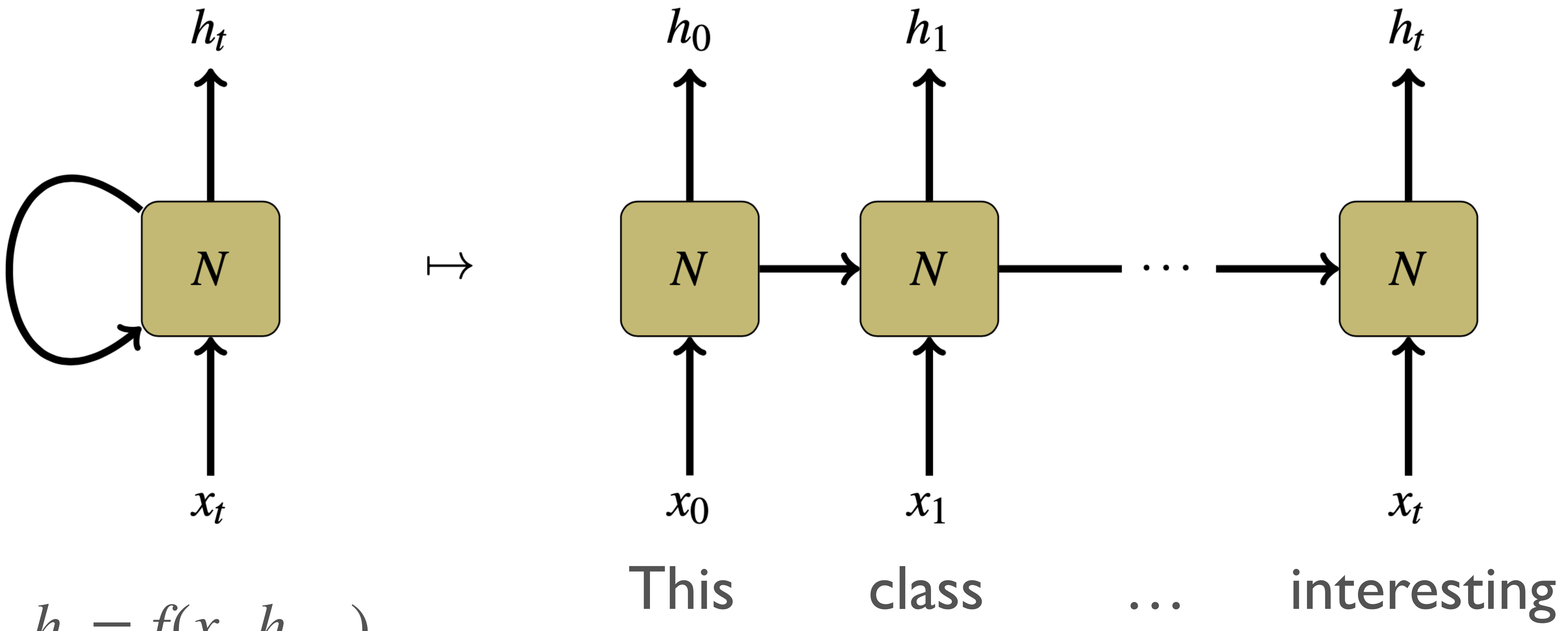
RNNs



$$h_t = f(x_t, h_{t-1})$$

Steinert-Threlkeld and Szymanik 2019; Olah 2015

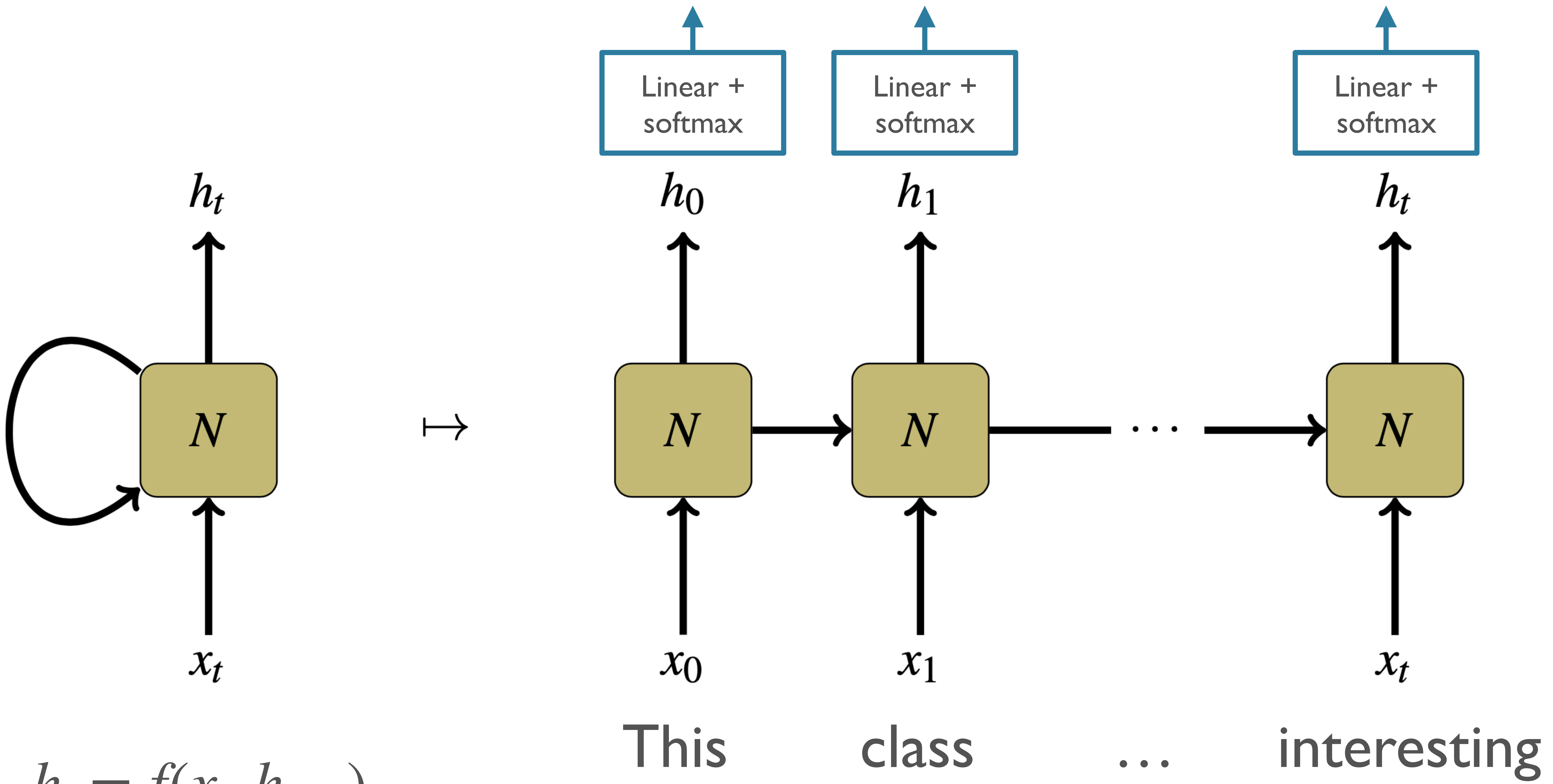
RNNs



$$h_t = f(x_t, h_{t-1})$$

Steinert-Threlkeld and Szymanik 2019; Olah 2015

RNNs



$$h_t = f(x_t, h_{t-1})$$

Steinert-Threlkeld and Szymanik 2019; Olah 2015

Simple / Vanilla / Elman RNNs

- Same kind of feed-forward computation we've been studying, but:
  - x_t: sequence element at time *t*
  - h_{t-1}: hidden state *of the model* at previous time *t-1*

Simple / Vanilla / Elman RNNs

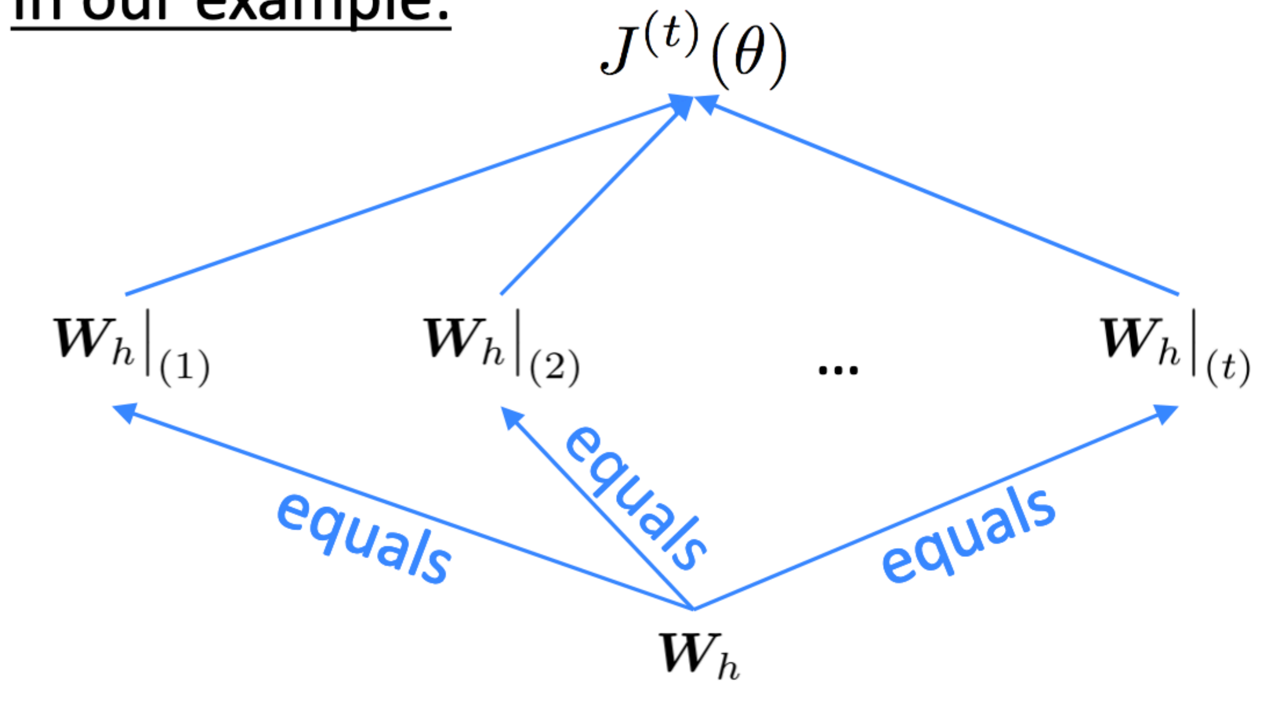
- Same kind of feed-forward computation we've been studying, but:
  - x_t: sequence element at time *t*
  - h_{t-1}: hidden state *of the model* at previous time *t-1*

Simple/“Vanilla” RNN: $$h_t = \tanh(x_t W_x + h_{t-1} W_h + b)$$

Training: BPTT

- Backpropagation Through Time
- “Unroll” the network across time-steps
- Apply backprop to the “wide” network
  - Each cell has the *same* parameters
  - Gradients sum across time-steps
    - Multi-variable chain rule

In our example:



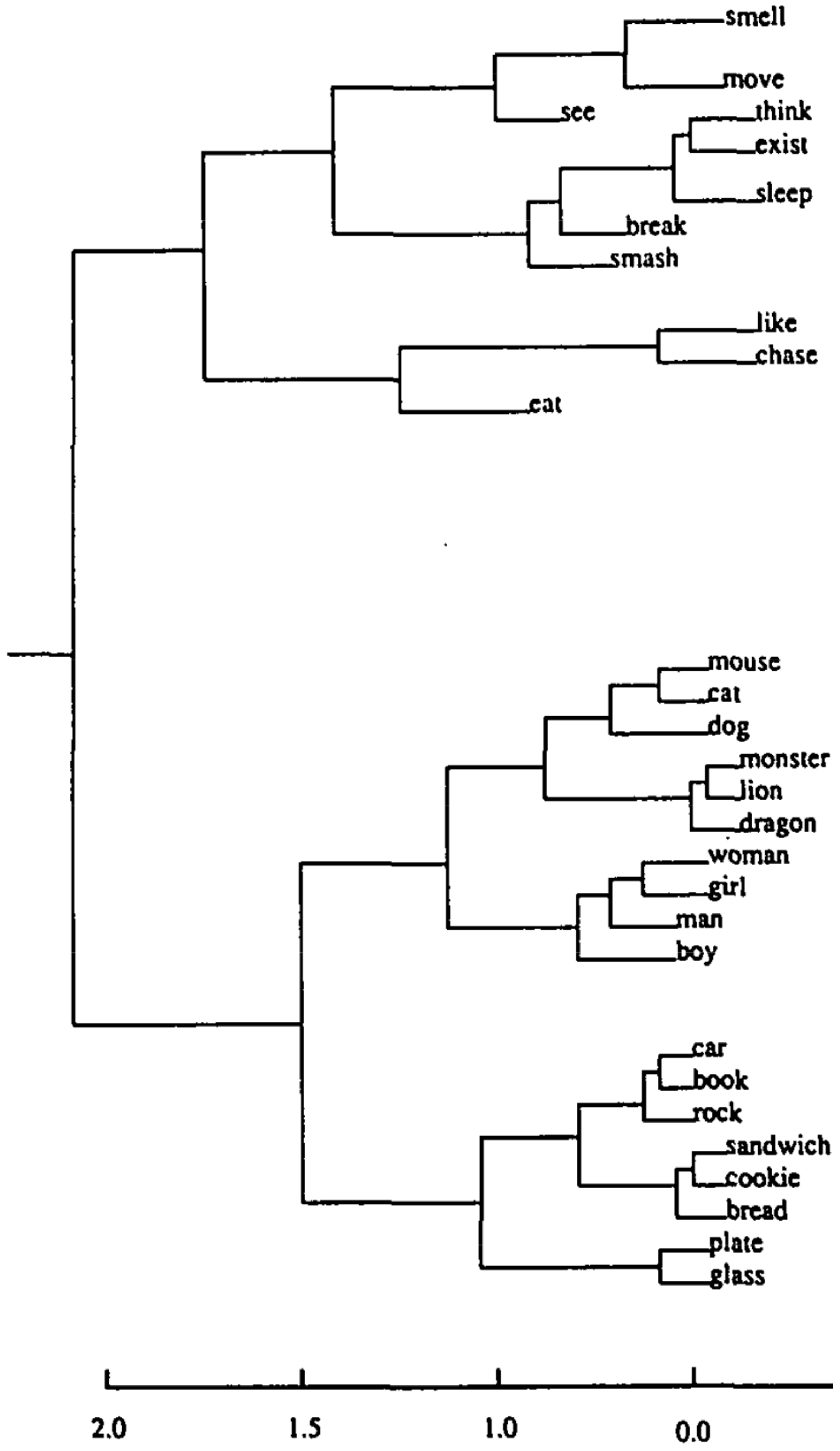
Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)} \overset{= 1}{\boxed{\frac{\partial \boldsymbol{W}_h\big|_{(i)}}{\partial \boldsymbol{W}_h}}}$$

$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)}$$

Power of RNNs

Hierarchical clustering of Vanilla RNN hidden states trained as LM on synthetic data:

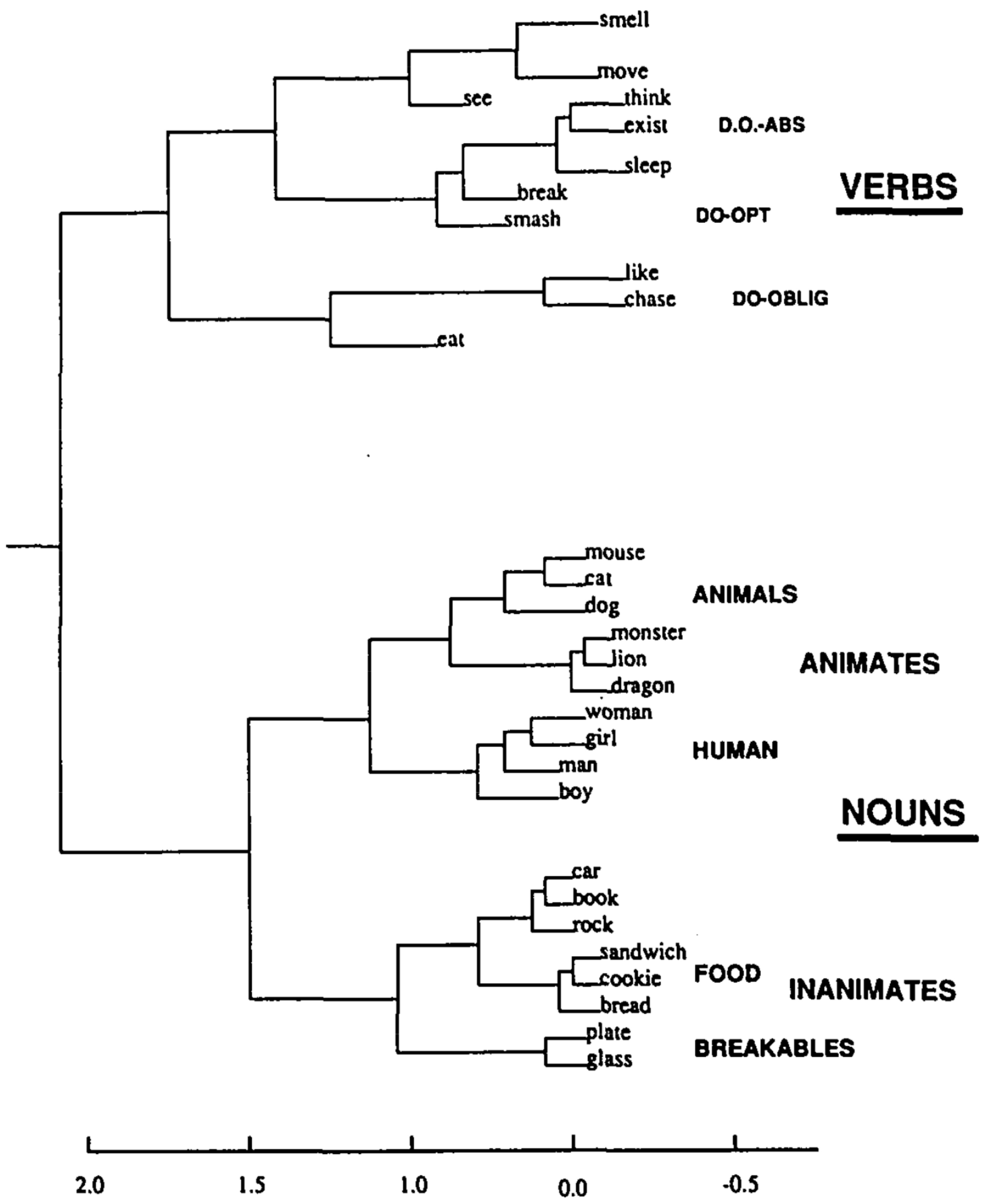


What trends do you notice?

[Elman 1990]

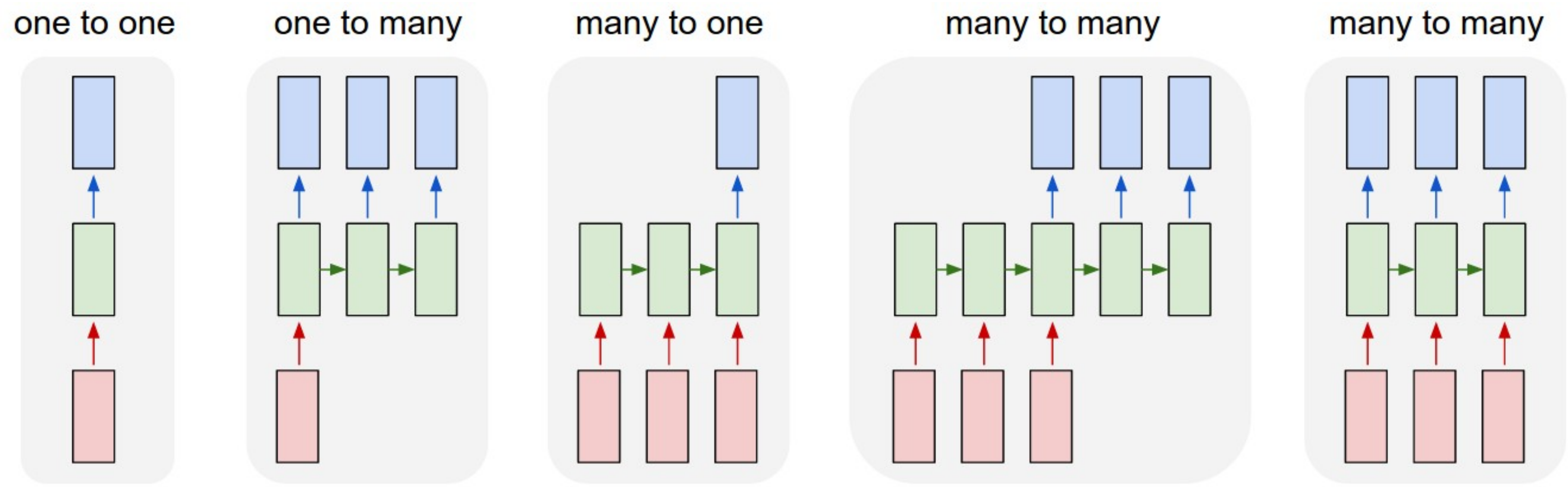
Power of RNNs

Hierarchical clustering of Vanilla RNN hidden states trained as LM on synthetic data:



Elman 1990

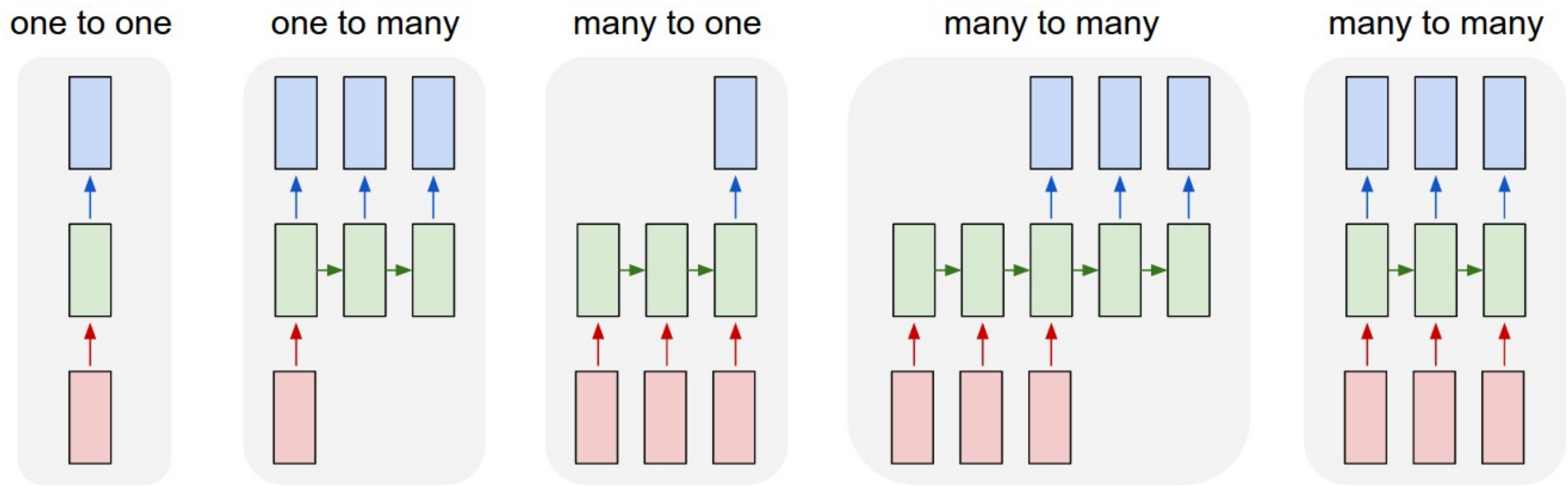
Using RNNs



MLP

e.g. image captioning

Using RNNs

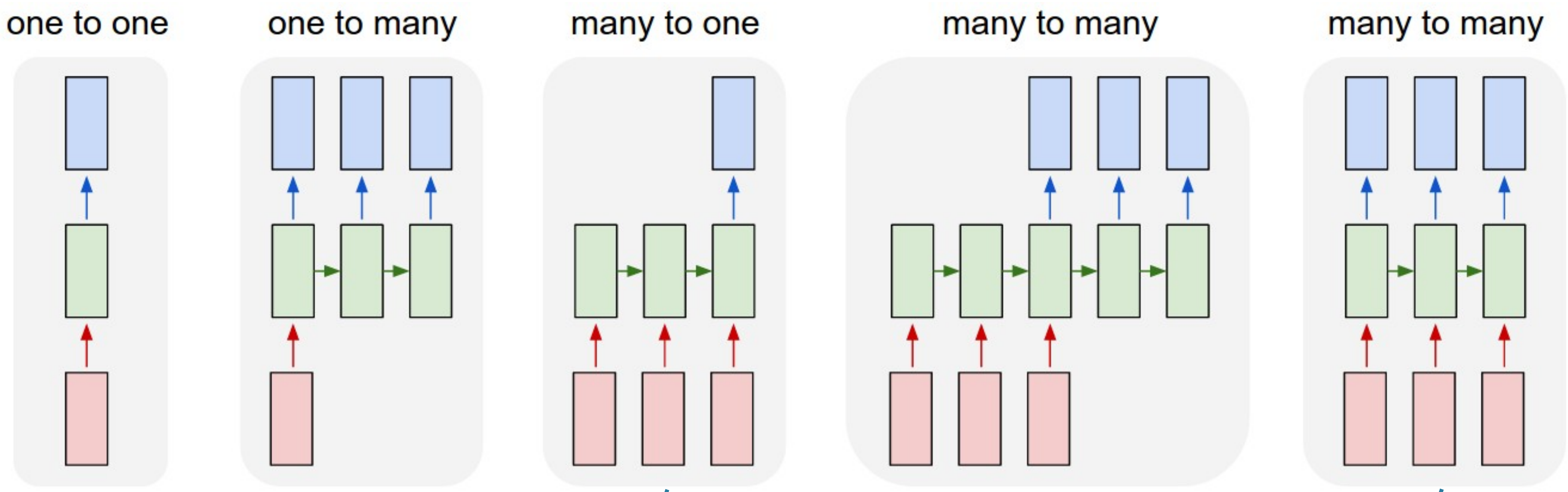


MLP

e.g. image captioning

e.g. text classification

Using RNNs



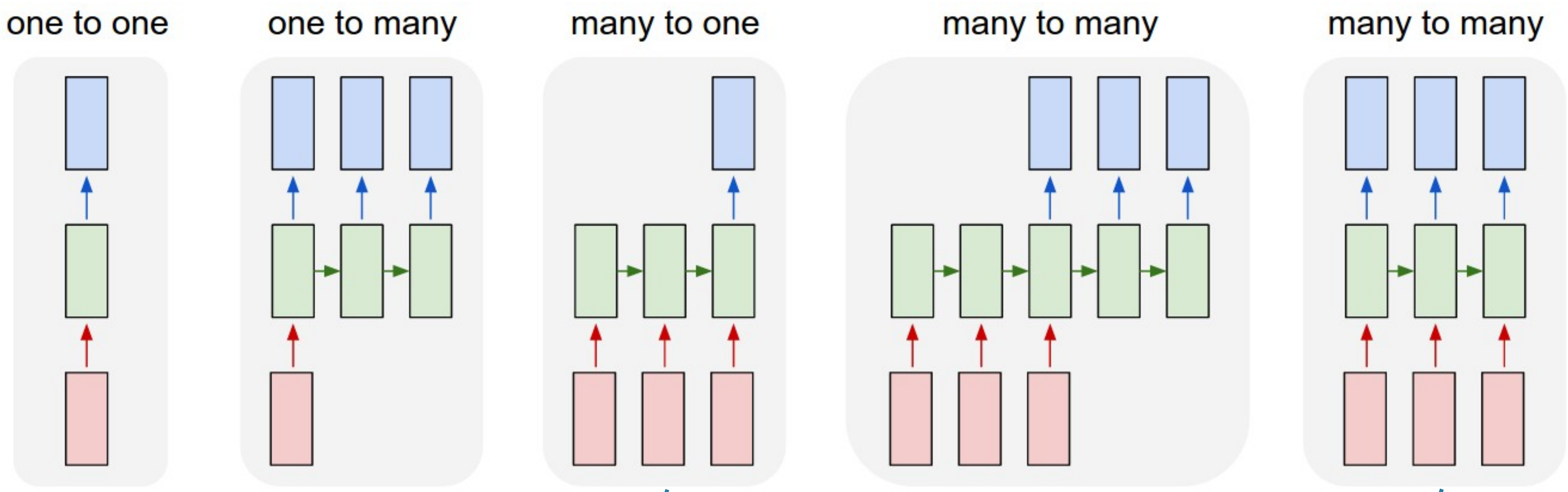
MLP

e.g. image captioning

e.g. text classification

e.g. POS tagging, LM O_O

Using RNNs



MLP

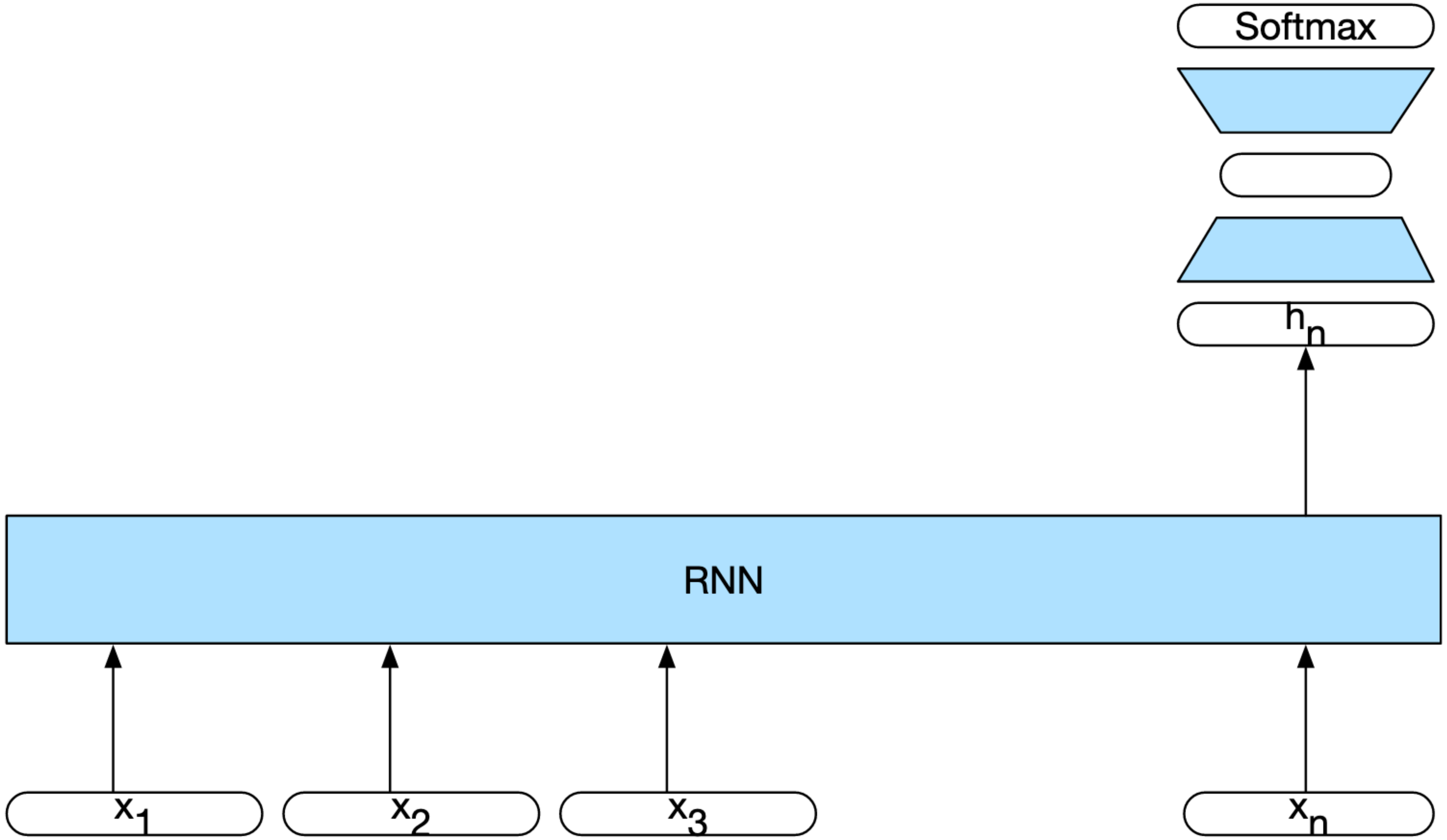
e.g. image captioning

seq2seq (later)

e.g. text classification

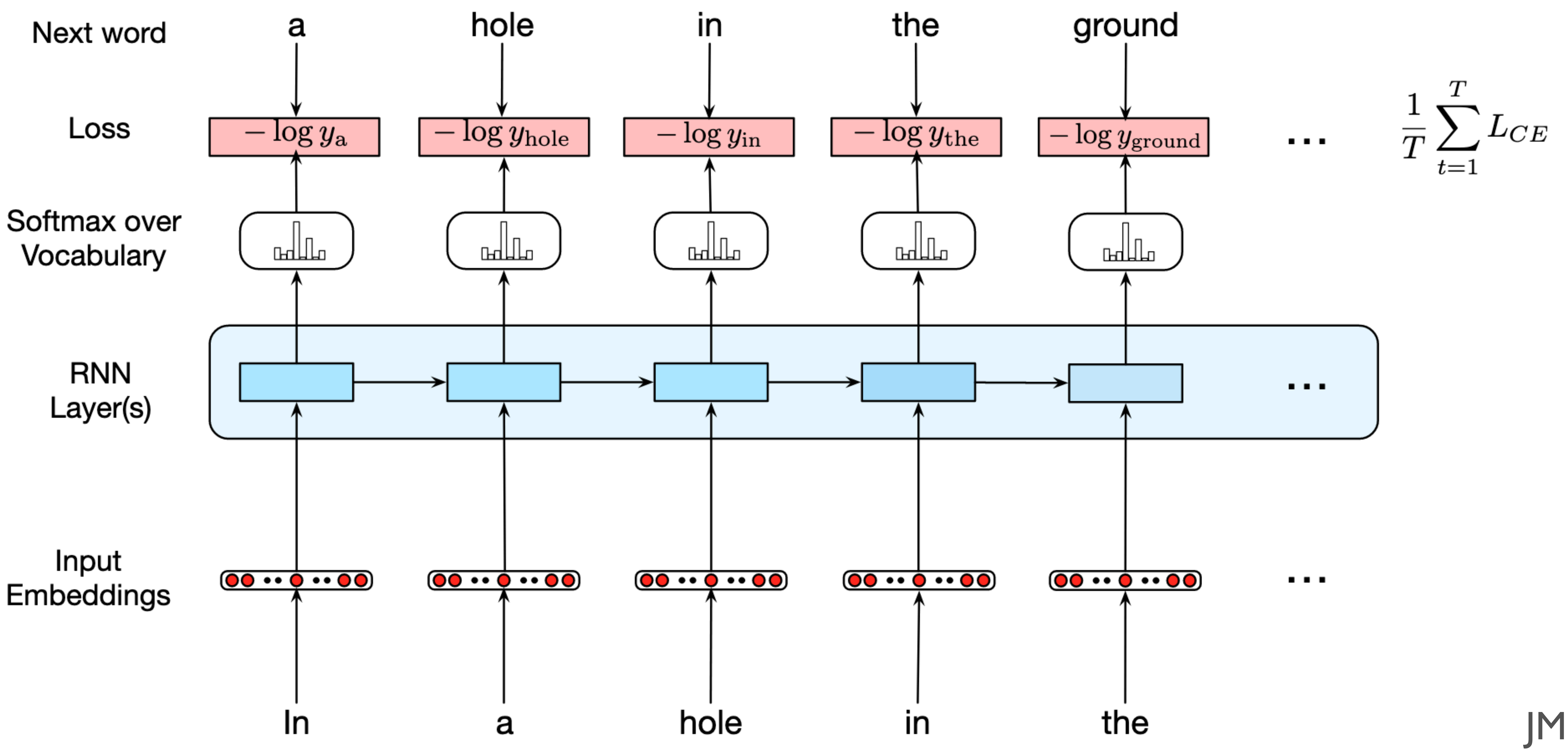
e.g. POS tagging, LM O_O

RNN for Text Classification



JM sec 9.2.5

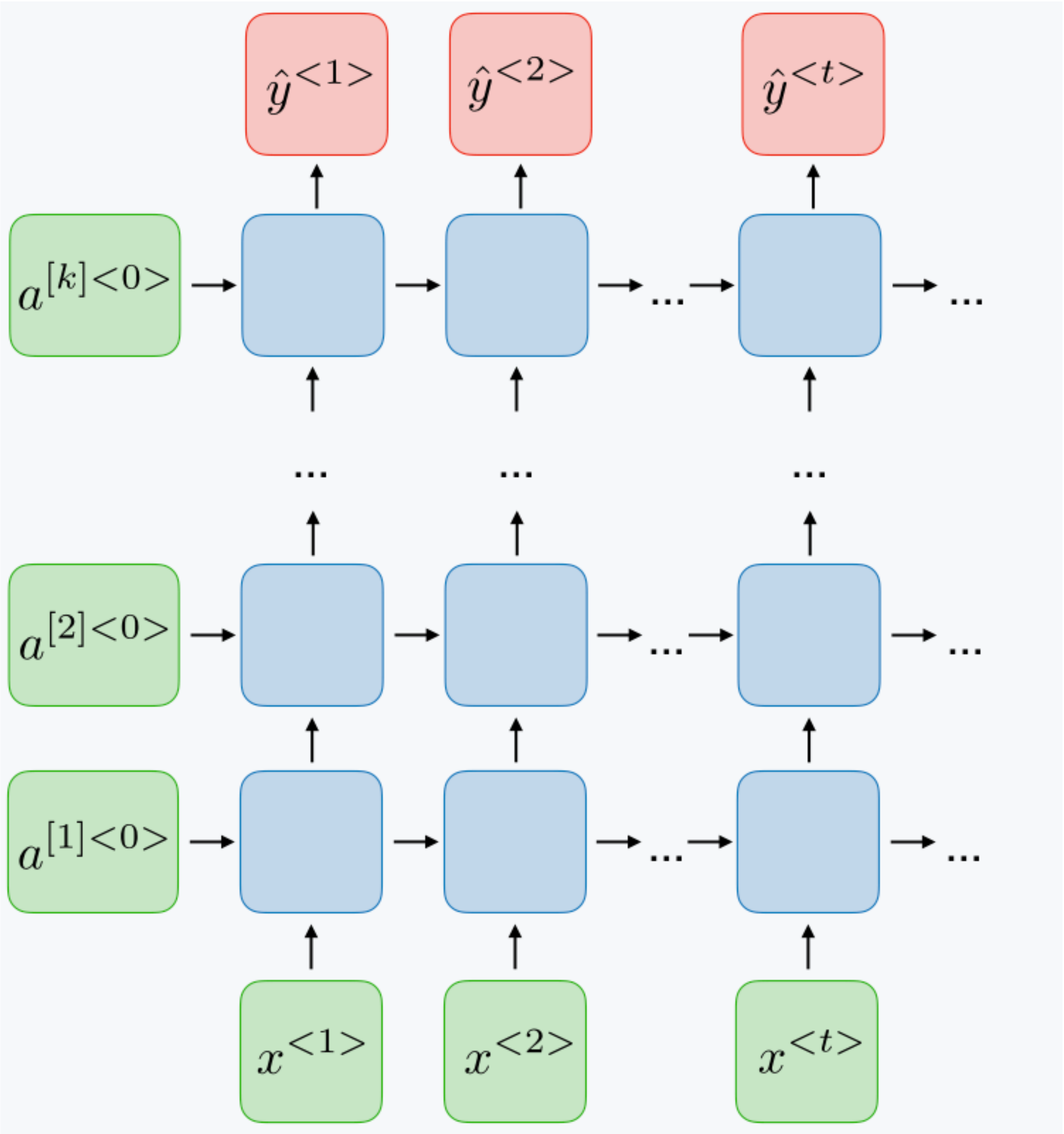
RNNs for Language Modeling



JM sec 9.2.3

Two Extensions

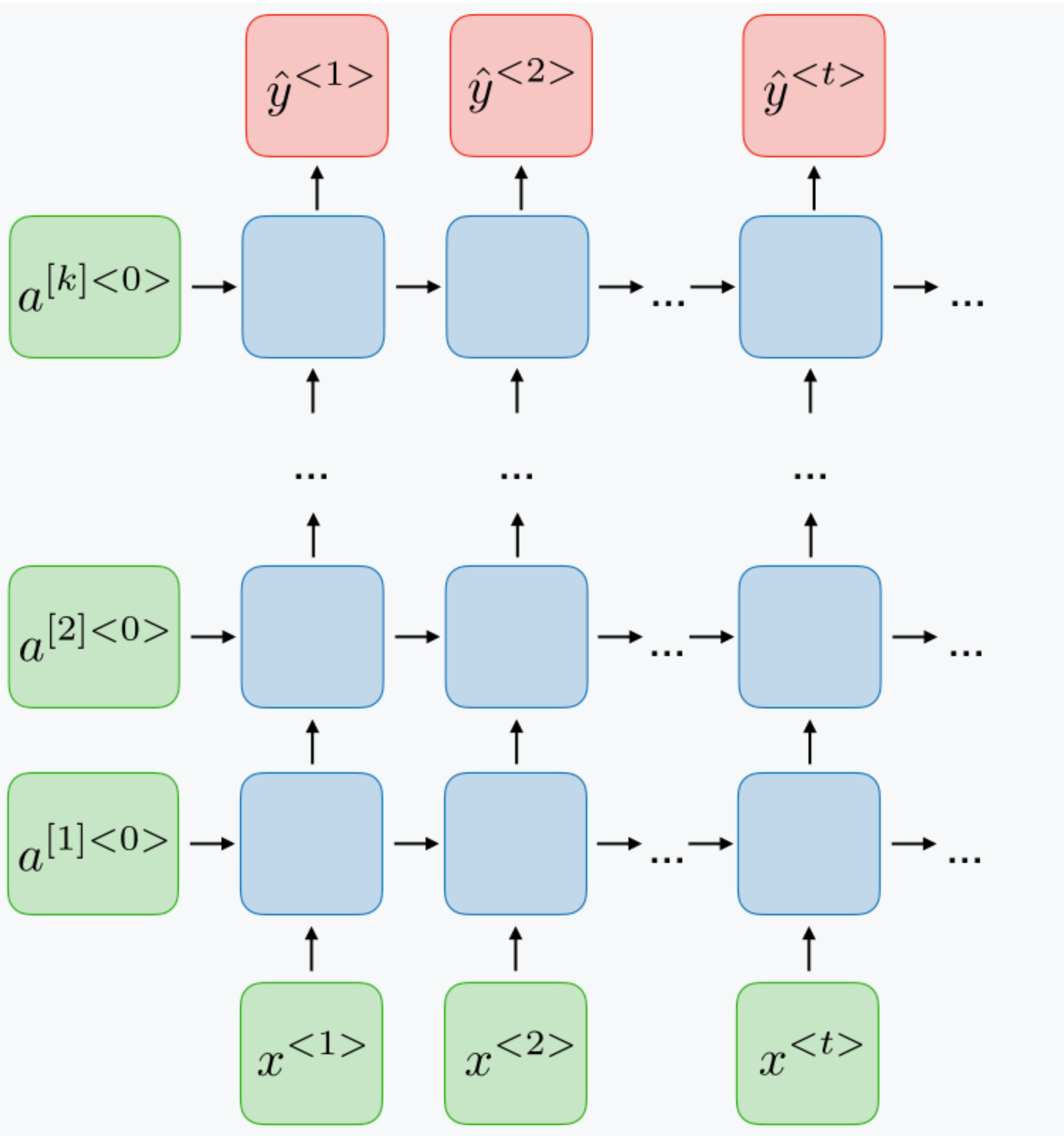
- Deep RNNs:



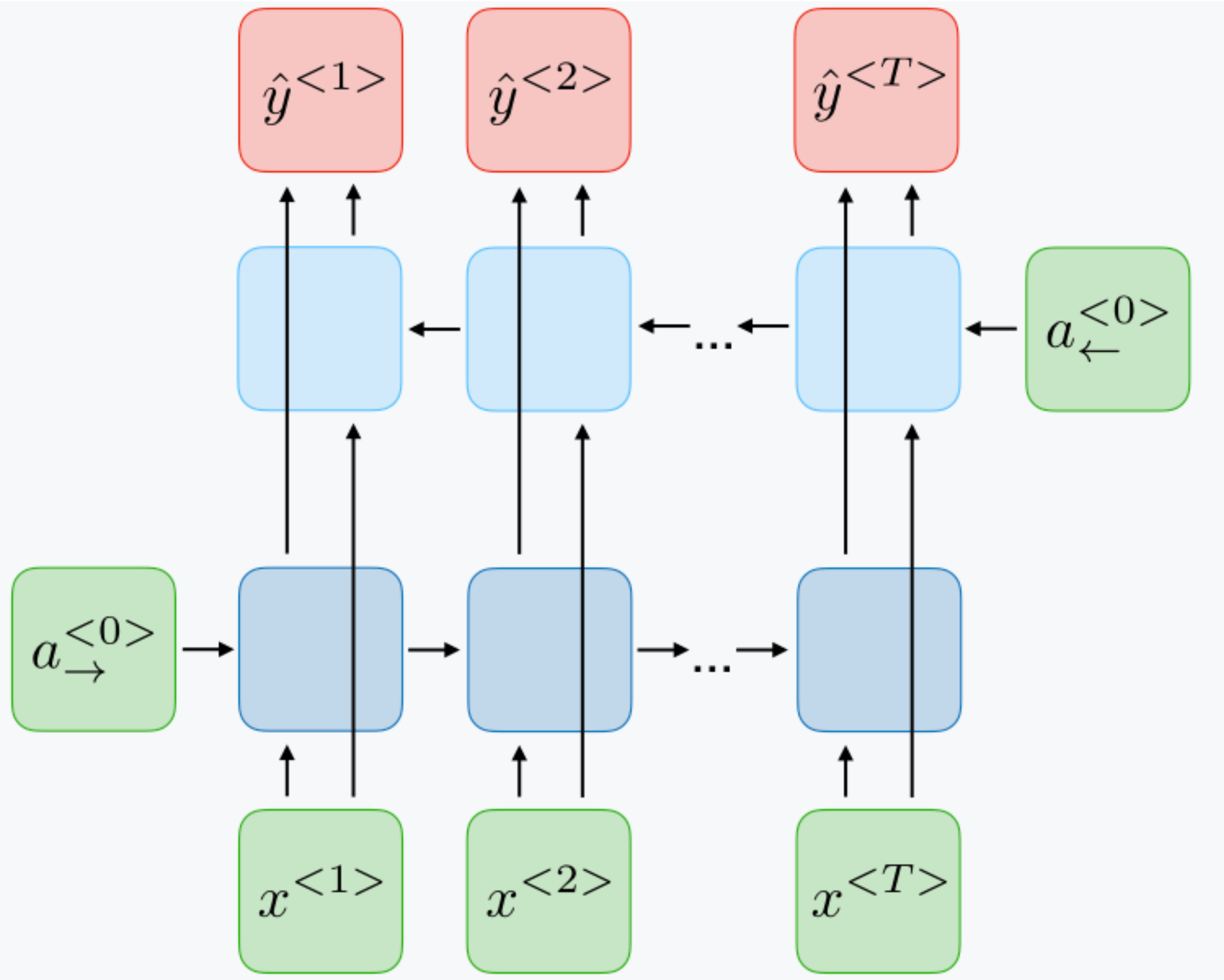
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



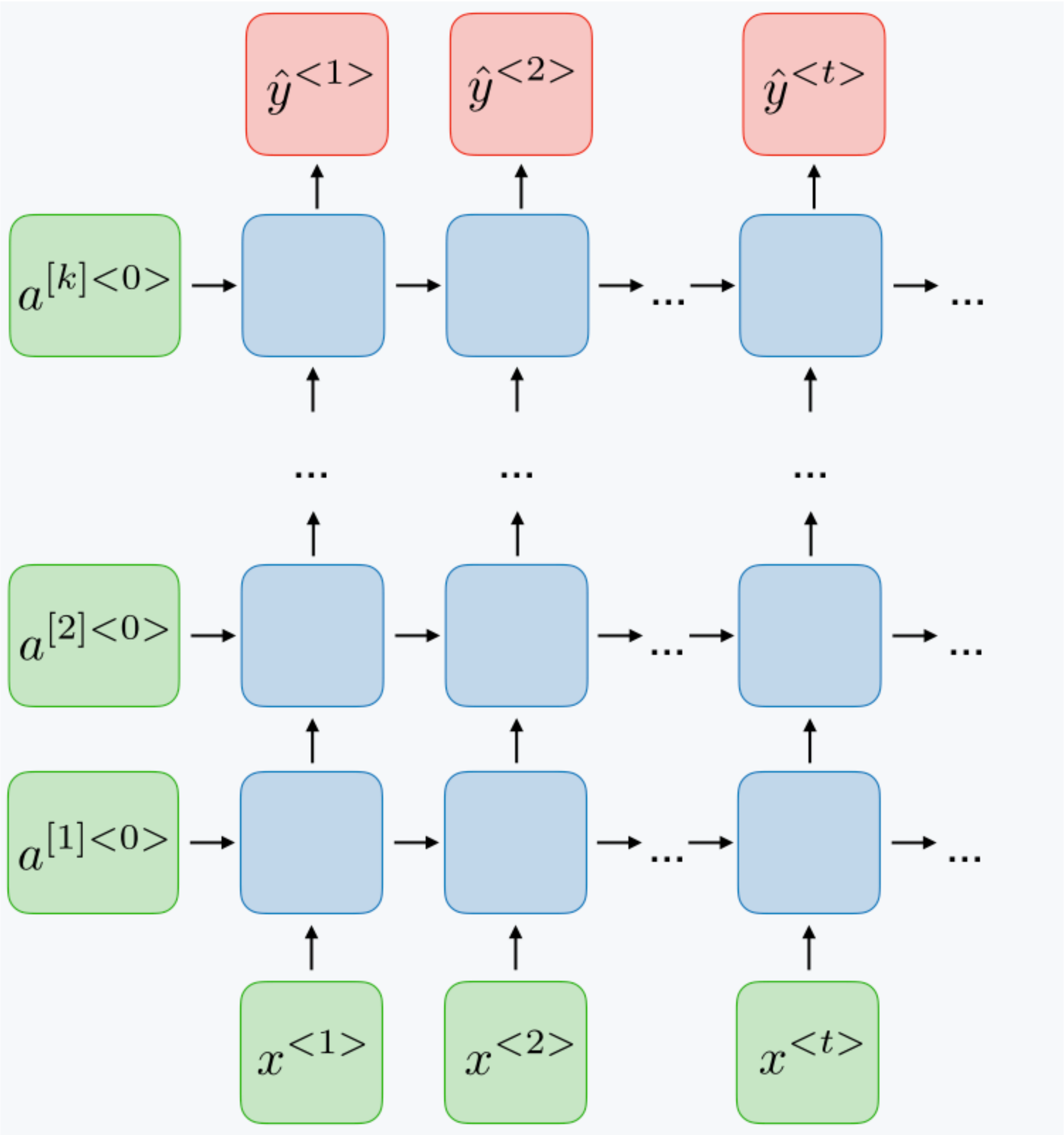
- Bidirectional RNNs:



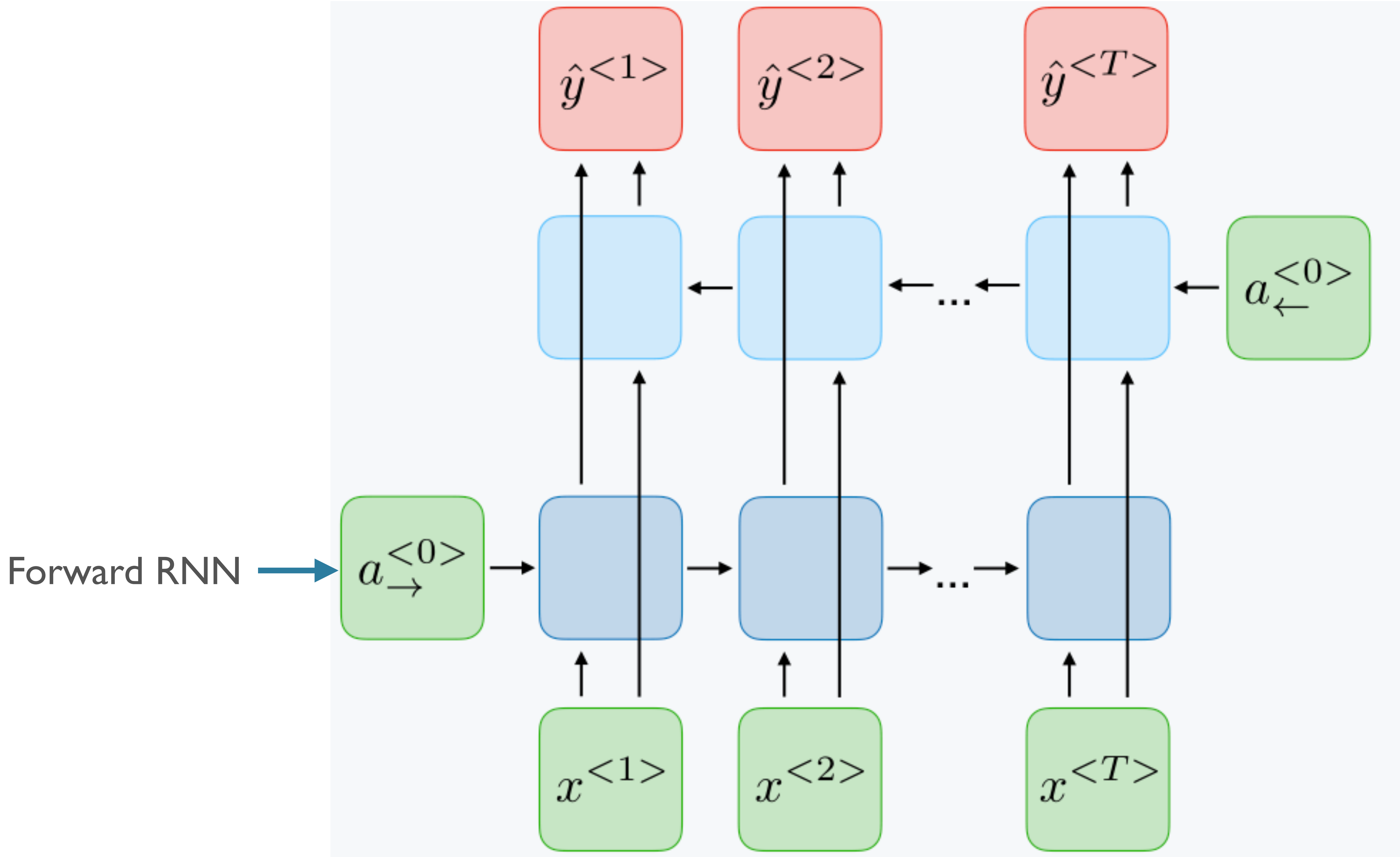
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



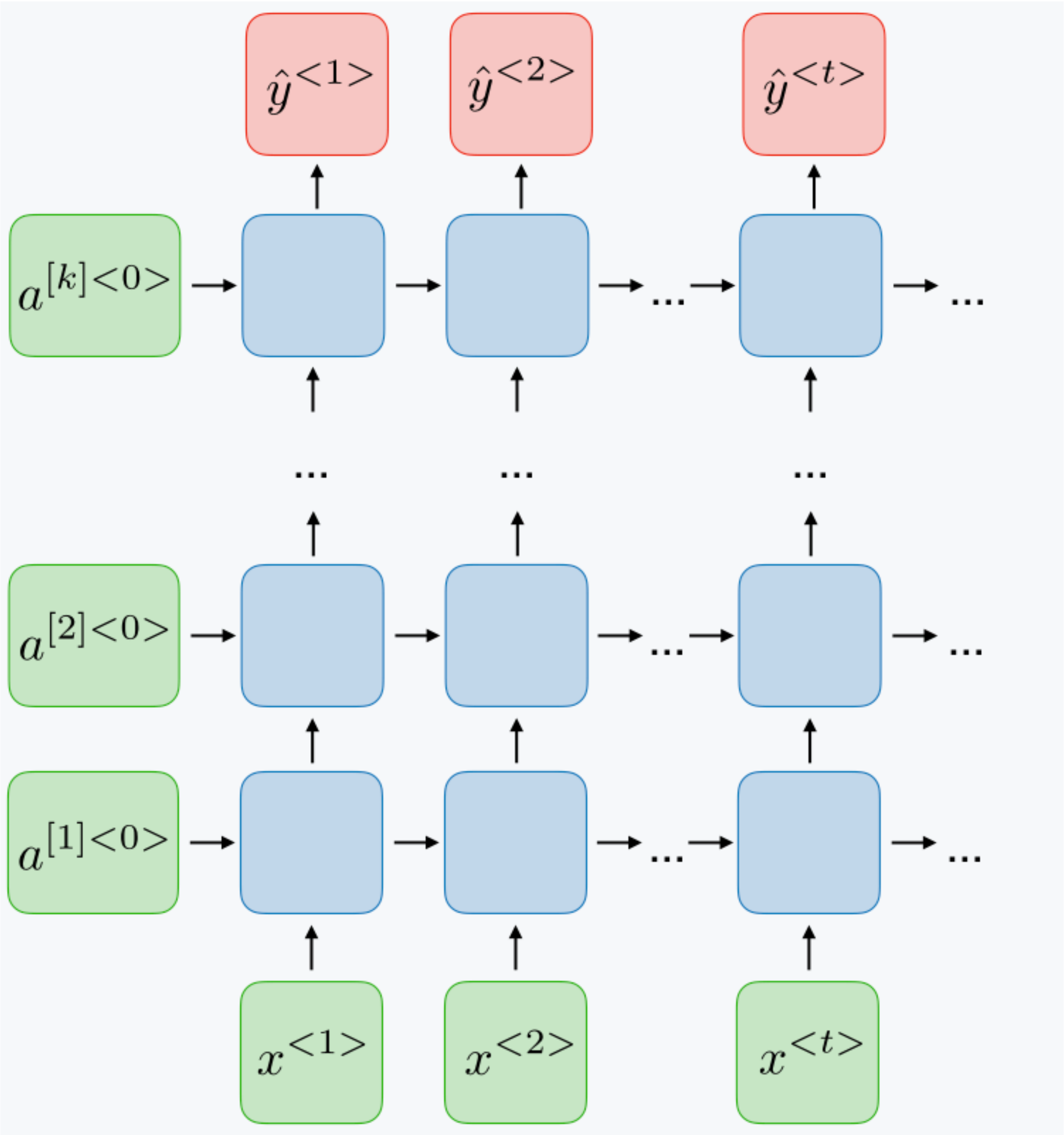
- Bidirectional RNNs:



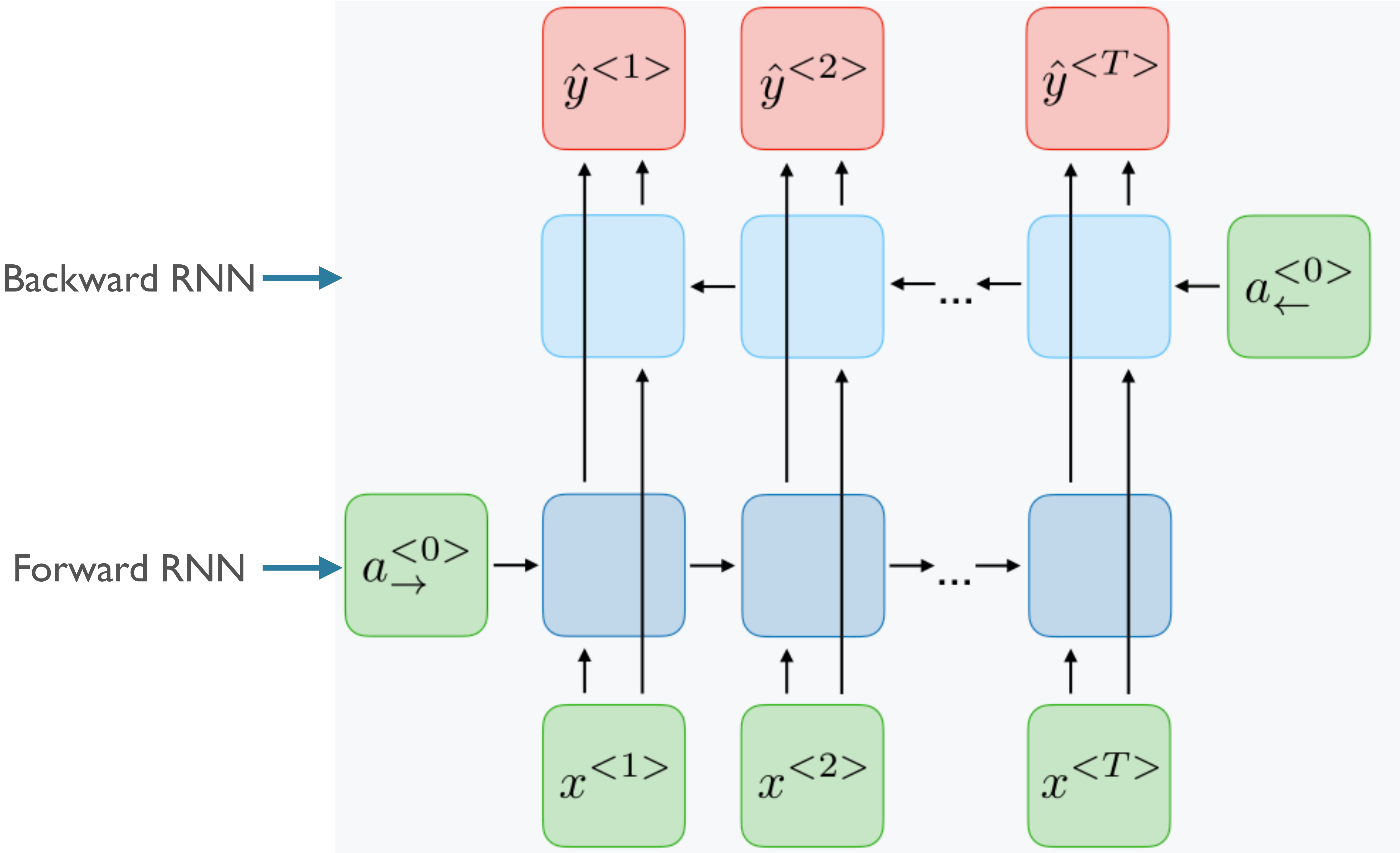
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



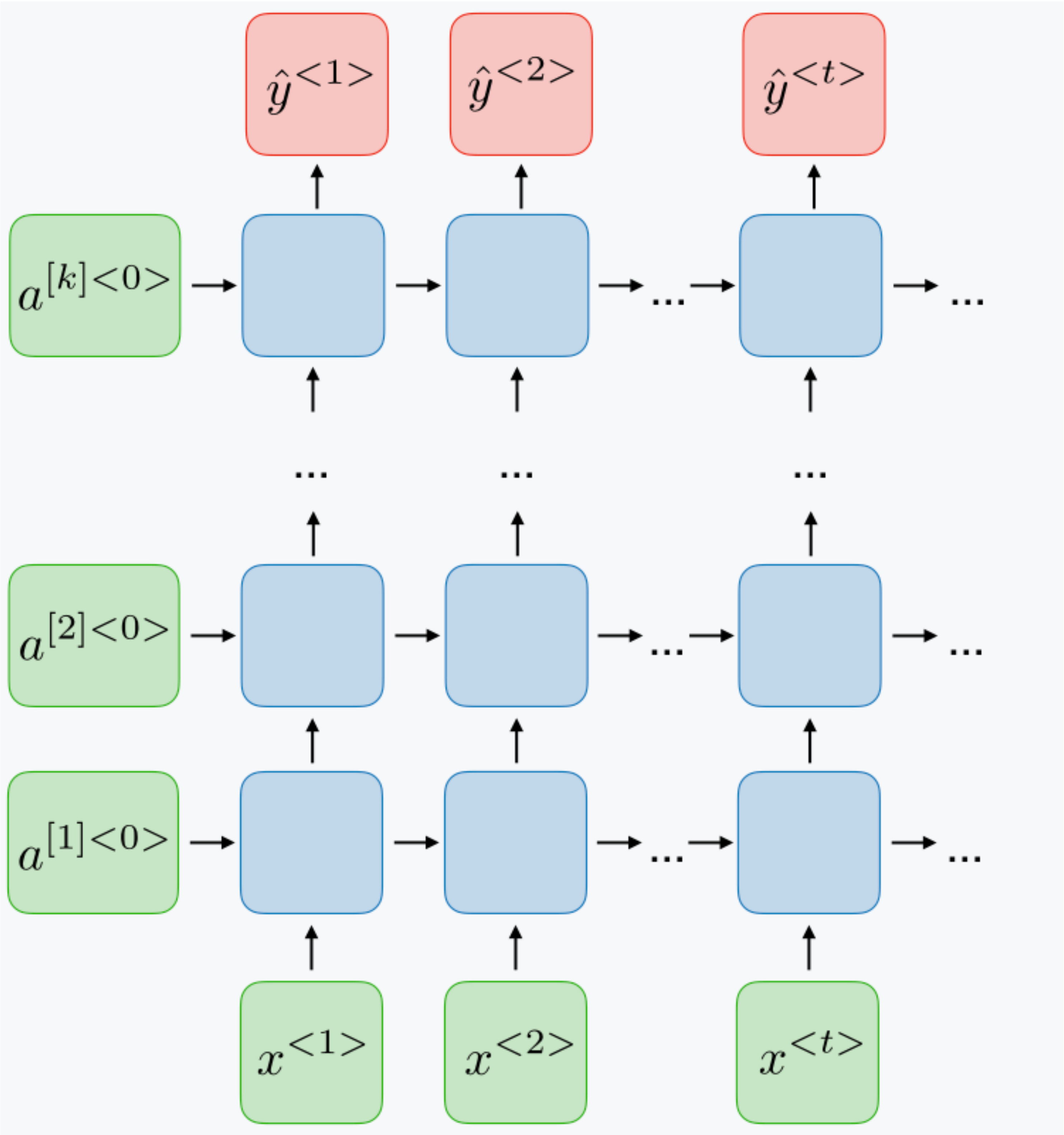
- Bidirectional RNNs:



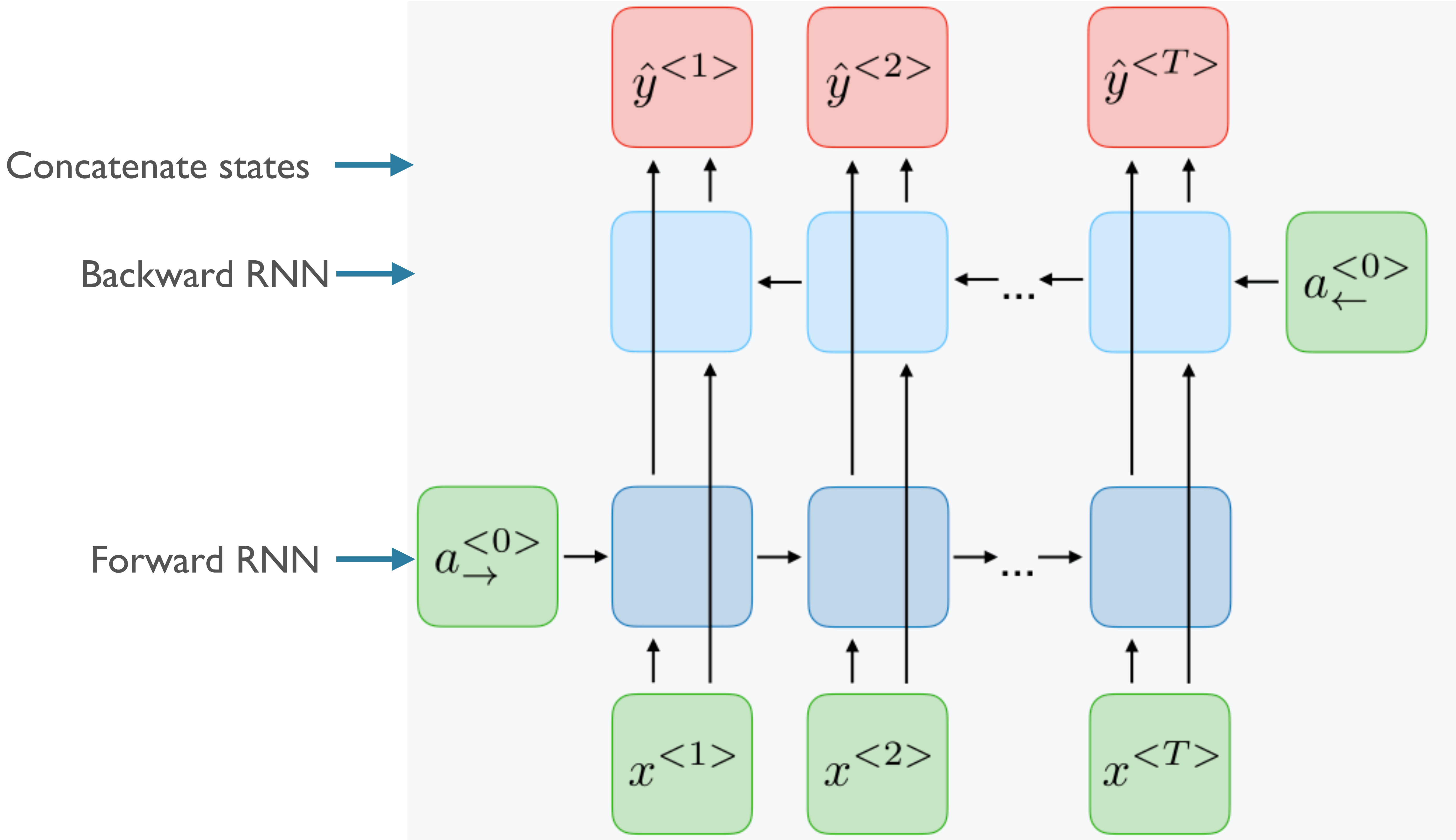
[Source: RNN cheat sheet]

Two Extensions

- Deep RNNs:



- Bidirectional RNNs:



Source: RNN cheat sheet

Batching in RNNs

- Intuitively, shape of inputs: [batch_size, seq_len, vocab_size]
- But what is sequence length??
  - “This is the first example </s>”: 6
  - “This is another </s>”: 4

Padding and Masking

- Step 1: *pad* all sequences in batch to be of the same length (PAD = special token)
  - “This is the first example </s>”: 6
  - “This is another </s> PAD PAD”: 6
- Step 2: build a “mask” (1 = True token, 0 = padding)

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

- Step 3: use mask to tell model what to ignore, either
  - Select correct final states [classification]
  - Multiply losses in tagging tasks [LM]

Summary

- RNNs allow for neural processing of *sequential data*
- In principle, should help models capture long-distance dependencies (e.g. number agreement, selectional preferences, …)
  - Maintain a state over time
  - Repeatedly apply the same weights
  - as opposed to n-gram models, which cannot build such dependencies
- Uses: classification, tagging
- Extensions: deep, bidirectional

Next Time

- Discuss a technical problem in training Vanilla RNNs
  - Vanishing gradients
- Introduce *gating-based* RNNs
  - LSTMs
  - GRUs
  - Strengths, weaknesses, differences