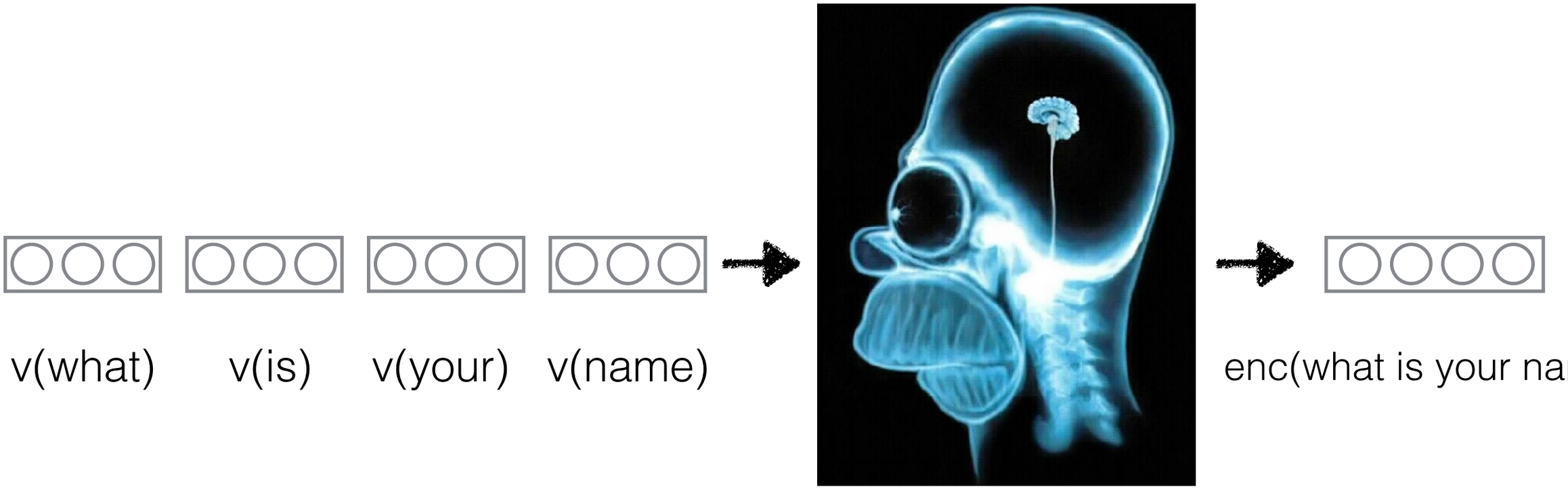
Recurrent Neural Networks (part 2 -- technical things)

Yoav Goldberg

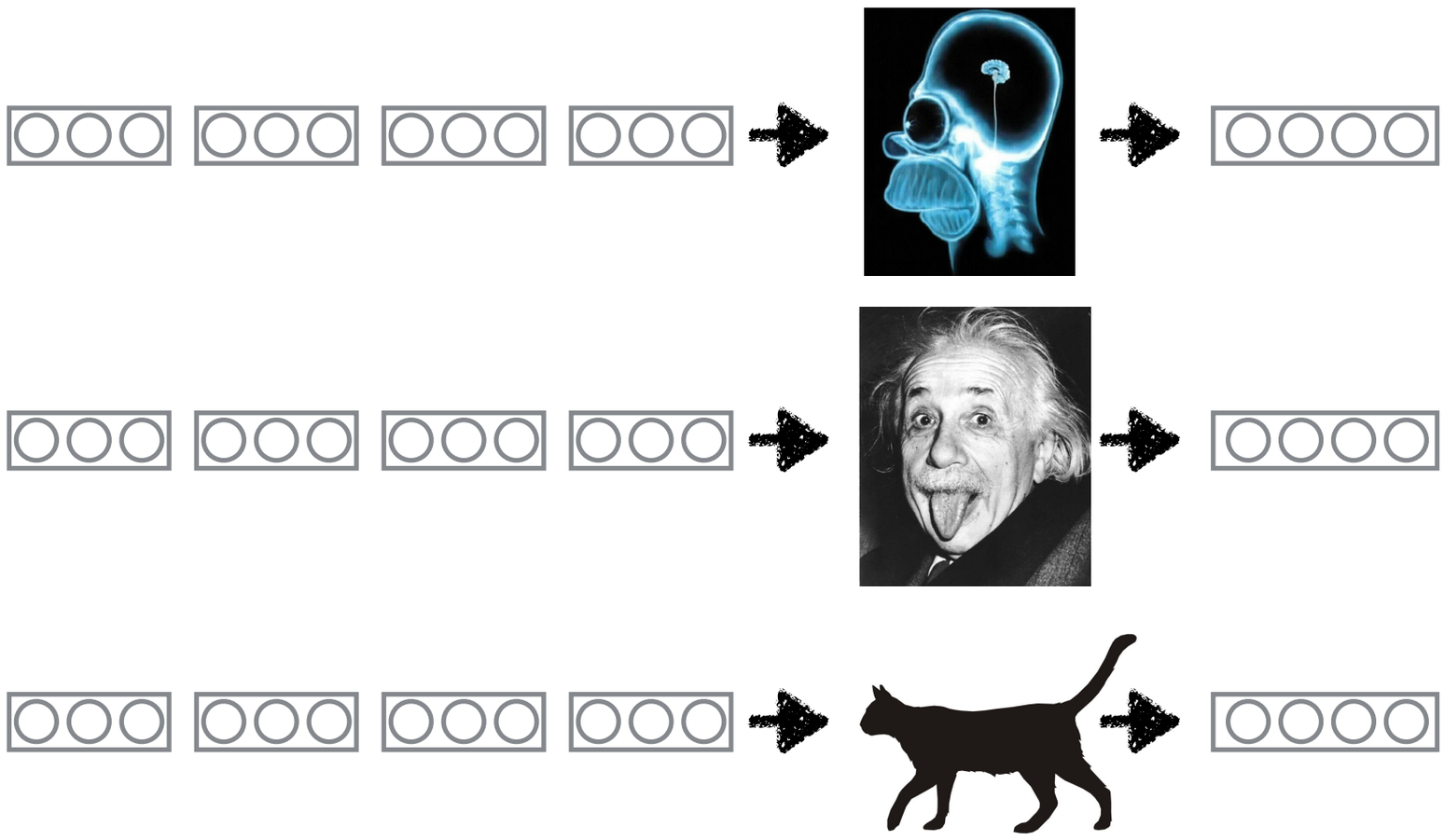
Concrete RNN Functions

Recurrent Neural Networks



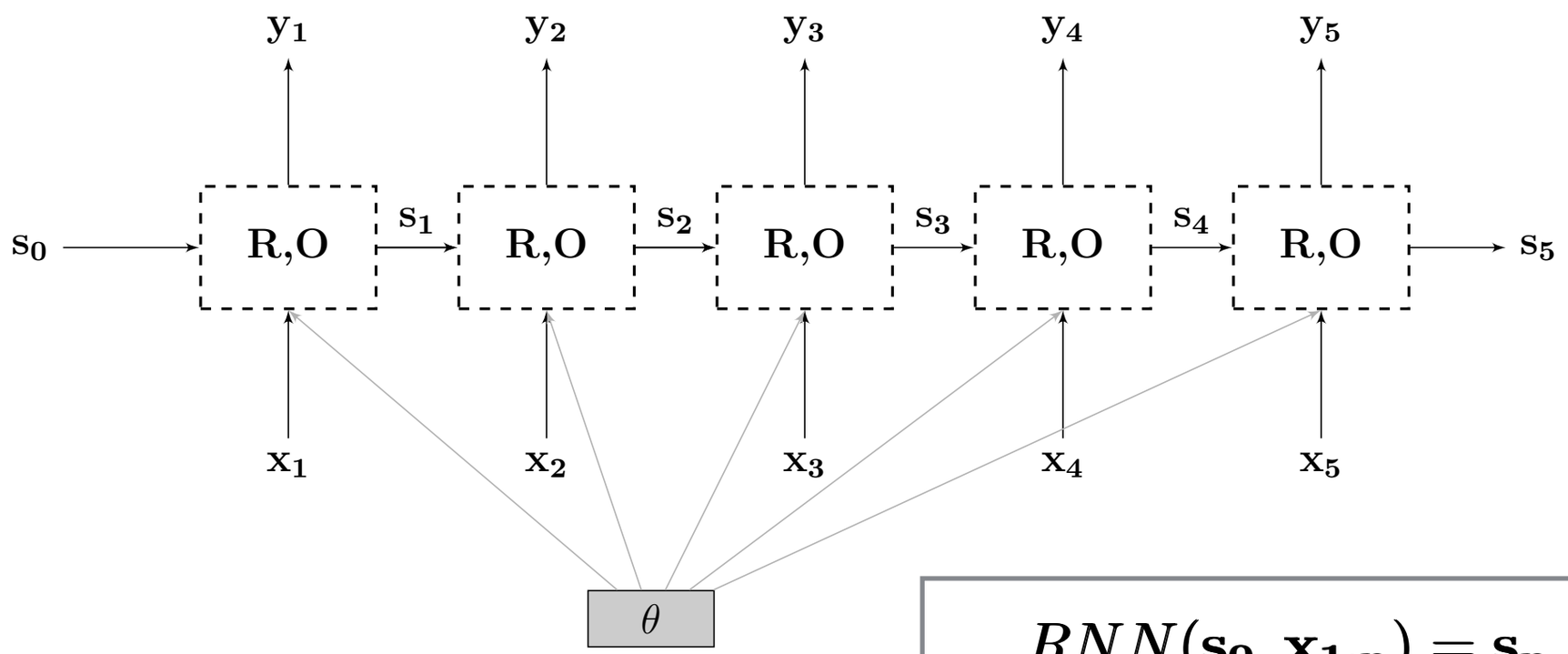
- Very strong models of sequential data.
- **Trainable** function from *n* vectors to a single vector.

Recurrent Neural Networks



- There are different variants (implementations).
- So far, we focused on the interface level.

Recurrent Neural Networks



for every finite input sequence,
can unroll the recursion.

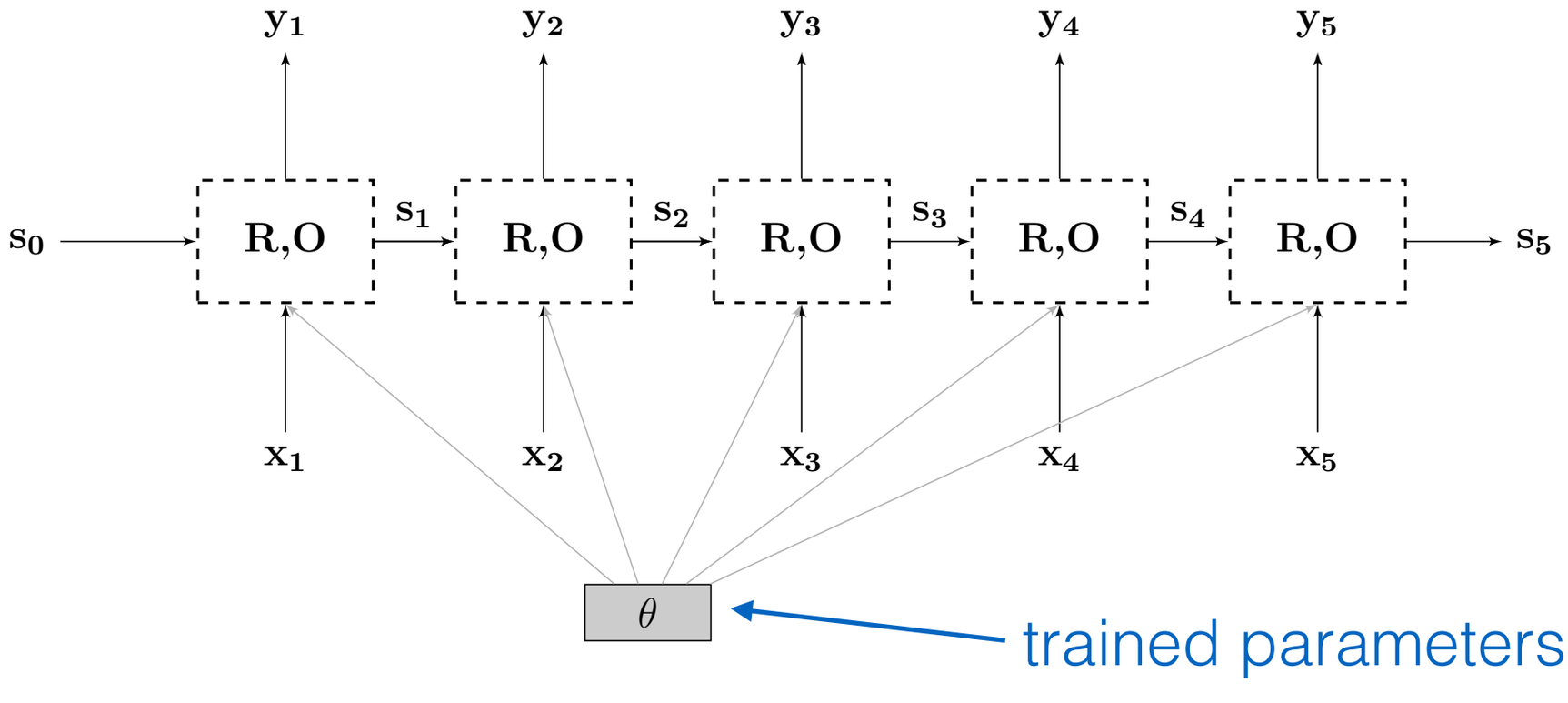
$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$
$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}},\ \mathbf{y_i} \in \mathbb{R}^{d_{out}},\ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- Recursively defined.
- There's a vector $\mathbf{y_i}$ for every prefix $\mathbf{x_{1:i}}$

Recurrent Neural Networks

- What do the RNN function look like?



- **But we can train them.**
  - **define function form**
  - define loss

CBOW as an RNN

$$R_{CBOW}(\mathbf{s_{i-1}}, \mathbf{x_i}) = \mathbf{s_{i-1}} + \mathbf{x_i}$$

(what are the parameters?)

CBOW as an RNN

$$R_{CBOW}(\mathbf{s_{i-1}}, \mathbf{x_i}) = \mathbf{s_{i-1}} + \mathbf{x_i}$$

(what are the parameters?)

$$R_{CBOW}(\mathbf{s_{i-1}}, x_i) = \mathbf{s_{i-1}} + \mathbf{E}_{[x_i]}$$

CBOW as an RNN

Is this a good parameterization?

$$R_{CBOW}(\mathbf{s_{i-1}}, x_i) = \mathbf{s_{i-1}} + \mathbf{E}_{[x_i]}$$

CBOW as an RNN

how about this modification?

$$R_{CBOW}(\mathbf{s_{i-1}}, x_i) = \tanh(\mathbf{s_{i-1}} + \mathbf{E}_{[x_i]})$$

Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

- Looks very simple.
- Theoretically very powerful.
- In practice not so much (hard to train).
- Why? Vanishing gradients.

Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

Another view on behavior:

- RNN as a "computer":
  input **xi** arrives, memory **s** is updated.

- In the Elman RNN, **entire memory is written** at each time-step.

Simple RNN (Elman RNN)

read previous state memory

write new input

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

Another view on behavior:

- RNN as a "computer":
  input **xi** arrives, memory **s** is updated.

- In the Elman RNN, **entire memory is written** at each time-step.

LSTM RNN

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) = & [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} = & \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} = & \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} = & \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} = & \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} = & \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} = & \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

$$
O_{LSTM}(\mathbf{s_j}) = O_{LSTM}([\mathbf{c_j}; \mathbf{h_j}]) = \mathbf{h_j}
$$

LSTM RNN

better controlled memory access

continuous gates

Differentiable "Gates"

- The main idea behind the LSTM is that you want to somehow control the "memory access".
- In a SimpleRNN:

$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

read previous state memory

write new input

- All the memory gets overwritten

Vector "Gates"

- We'd like to:
  * Selectively read from some memory "cells".
  * Selectively write to some memory "cells".

Vector "Gates"

- We'd like to:
  * Selectively read from some memory "cells".
  * Selectively write to some memory "cells".
- A gate function:

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix}$$

(element-wise multiplication)

$\mathbf{g}$ $\mathbf{x}$

- gate controls access
- vector of values

Vector "Gates"

- We'd like to:
  * Selectively read from some memory "cells".
  * Selectively write to some memory "cells".

- A gate function:

$$\mathbf{s_{i-1}} \odot \mathbf{g} \qquad \mathbf{g} \in \{0,1\}^d$$

vector of values

gate controls access

-

Vector "Gates"

- Using the gate function to control access:

$$\mathbf{s_i} \leftarrow \mathbf{s_{i-1}} \odot \mathbf{g^r} + \mathbf{x_i} \odot \mathbf{g^w} \qquad \mathbf{g} \in \{0,1\}^d$$

which cells to read

which cells to write

-

Vector "Gates"

- Using the gate function to control access:

$$\mathbf{s_i} \leftarrow \mathbf{s_{i-1}} \odot \mathbf{g^r} + \mathbf{x_i} \odot \mathbf{g^w} \qquad \mathbf{g} \in \{0,1\}^d$$

which cells to read

which cells to write

- (can also tie them: $\mathbf{g^r} = 1 - \mathbf{g^w}$)

Vector "Gates"

$$\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix} \leftarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}$$

$$\mathbf{s}' \qquad \mathbf{g} \quad \mathbf{x} \qquad (\mathbf{1}-\mathbf{g}) \quad \mathbf{s}$$

Differentiable "Gates"

- **Problem with the gates**:
  * they are fixed.
  * they don't depend on the input or the output.

Differentiable "Gates"

- **Problem with the gates**:
  * they are fixed.
  * they don't depend on the input or the output.

- Solution: make them smooth, input dependent, and trainable.

$$\mathbf{g^r} = \sigma(\mathbf{W} \cdot \mathbf{x_i} + \mathbf{U} \cdot \mathbf{s_{i-1}})$$

"almost 0"
or
"almost 1"

function of input and state

-

LSTM

(Long short-term Memory)

- The LSTM is a specific combination of gates.
-

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) &= [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} &= \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} &= \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} &= \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} &= \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

$$
O_{LSTM}(\mathbf{s_j}) = O_{LSTM}([\mathbf{c_j}; \mathbf{h_j}]) = \mathbf{h_j}
$$

GRU
(Gated Recurrent Unit)

- The GRU is a different combination of gates.
-

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (\mathbf{1} - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s}}_\mathbf{j}$$

$$\mathbf{z} = \sigma(\mathbf{x_j W^{xz}} + \mathbf{s_{j-1} W^{sz}})$$

$$\mathbf{r} = \sigma(\mathbf{x_j W^{xr}} + \mathbf{s_{j-1} W^{sr}})$$

$$\tilde{\mathbf{s}}_\mathbf{j} = \tanh(\mathbf{x_j W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

GRU vs LSTM

- The GRU and the LSTM are very similar ideas.
- Invented independently of the LSTM, almost two decades later.

GRU
(Gated Recurrent Unit)

- The GRU formulation:

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =$$

Proposal state: $\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j}\mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$

GRU
(Gated Recurrent Unit)

- The GRU formulation:

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =$$

gate controlling effect of prev on proposal:

$$\mathbf{r} = \sigma(\mathbf{x_j}\mathbf{W^{xr}} + \mathbf{s_{j-1}}\mathbf{W^{sr}})$$

$$\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j}\mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

GRU

(Gated Recurrent Unit)

blend of old state and proposal state

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (\mathbf{1} - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s}}_\mathbf{j}$$

$$\mathbf{r} = \sigma(\mathbf{x_j W^{xr}} + \mathbf{s_{j-1} W^{sr}})$$

$$\tilde{\mathbf{s}}_\mathbf{j} = \tanh(\mathbf{x_j W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

GRU

(Gated Recurrent Unit)

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (\mathbf{1} - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s}}_\mathbf{j}$$

gate for controlling the blend

$$\mathbf{z} = \sigma(\mathbf{x_j W^{xz}} + \mathbf{s_{j-1} W^{sz}})$$

$$\mathbf{r} = \sigma(\mathbf{x_j W^{xr}} + \mathbf{s_{j-1} W^{sr}})$$

$$\tilde{\mathbf{s}}_\mathbf{j} = \tanh(\mathbf{x_j W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

GRU

(Gated Recurrent Unit)

$$
\begin{aligned}
\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& (\mathbf{1} - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s}}_\mathbf{j} \\
\mathbf{z} =& \sigma(\mathbf{x_j W^{xz}} + \mathbf{s_{j-1} W^{sz}}) \\
\mathbf{r} =& \sigma(\mathbf{x_j W^{xr}} + \mathbf{s_{j-1} W^{sr}}) \\
\tilde{\mathbf{s}}_\mathbf{j} =& \tanh(\mathbf{x_j W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}}) \mathbf{W^{sg}})
\end{aligned}
$$

LSTM

(Long short-term Memory)

- The LSTM is formulation:

-

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) &= [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} &= \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} &= \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} &= \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} &= \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

$$
O_{LSTM}(\mathbf{s_j}) = O_{LSTM}([\mathbf{c_j}; \mathbf{h_j}]) = \mathbf{h_j}
$$

LSTM

(Long short-term Memory)

- The LSTM is formulation:

-

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) &= [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \odot \mathbf{o} \qquad \text{"y"} \\
\mathbf{i} &= \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \qquad \text{"input"} \\
\mathbf{f} &= \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \qquad \text{"forget"} \\
\mathbf{o} &= \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \qquad \text{"output"} \\
\mathbf{g} &= \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}}) \qquad \text{input repr}
\end{aligned}
$$

$$O_{LSTM}(\mathbf{s_j}) = O_{LSTM}([\mathbf{c_j}; \mathbf{h_j}]) = \mathbf{h_j}$$

Read More

- The gated architecture also helps the vanishing gradients problems.

- For a good explanation, see Kyunghyun Cho's notes: http://arxiv.org/abs/1511.07916 sections 4.2, 4.3

- Chris Olah's blog post (link in class website)

Other Variants

- Many other variants exist.
- Mostly perform similarly to each other.
  - Different tasks may work better with different variants.
- **The important idea is the differentiable gates.**

Differences?

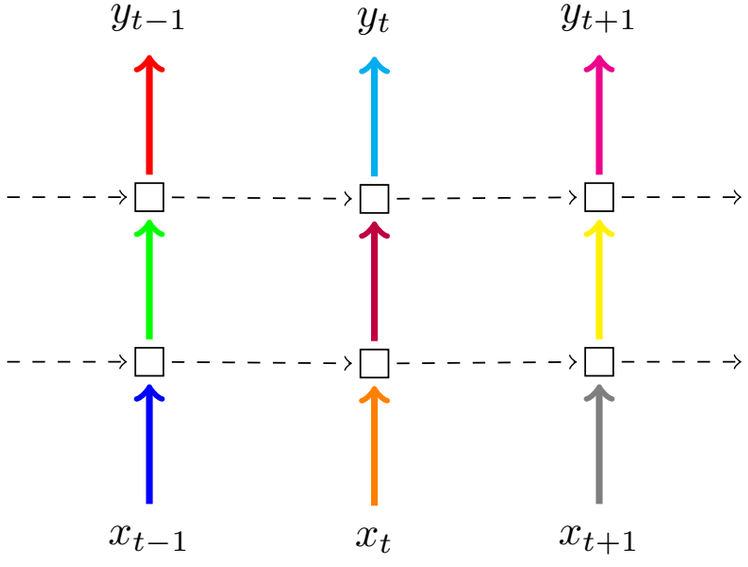
- **There ARE formal difference in power between the GRU and the LSTM.**

- The LSTM can count, the GRU cannot.
  See paper by Weiss, Goldberg and Yahav in reading material.
  https://arxiv.org/abs/1805.04908

Dropout in RNNs

Dropout in RNNs

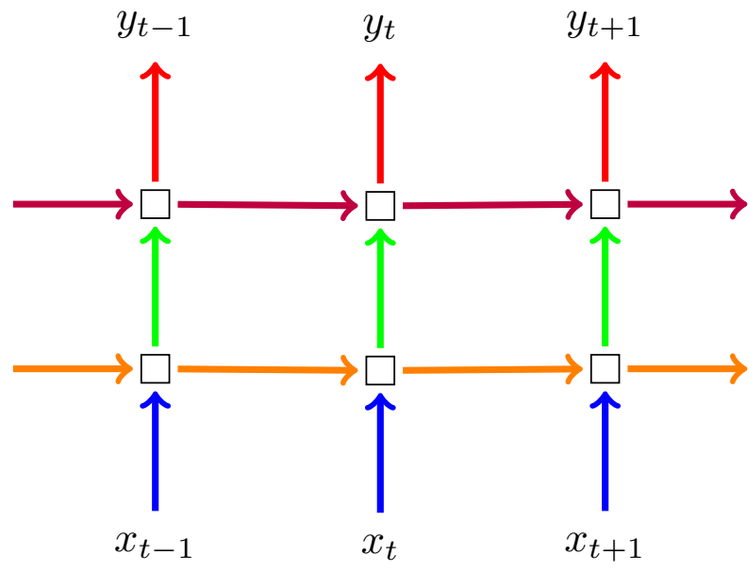
- Still an open question how to perform well.
- One suggestion:



each color is a different random dropout mask

Dropout in RNNs

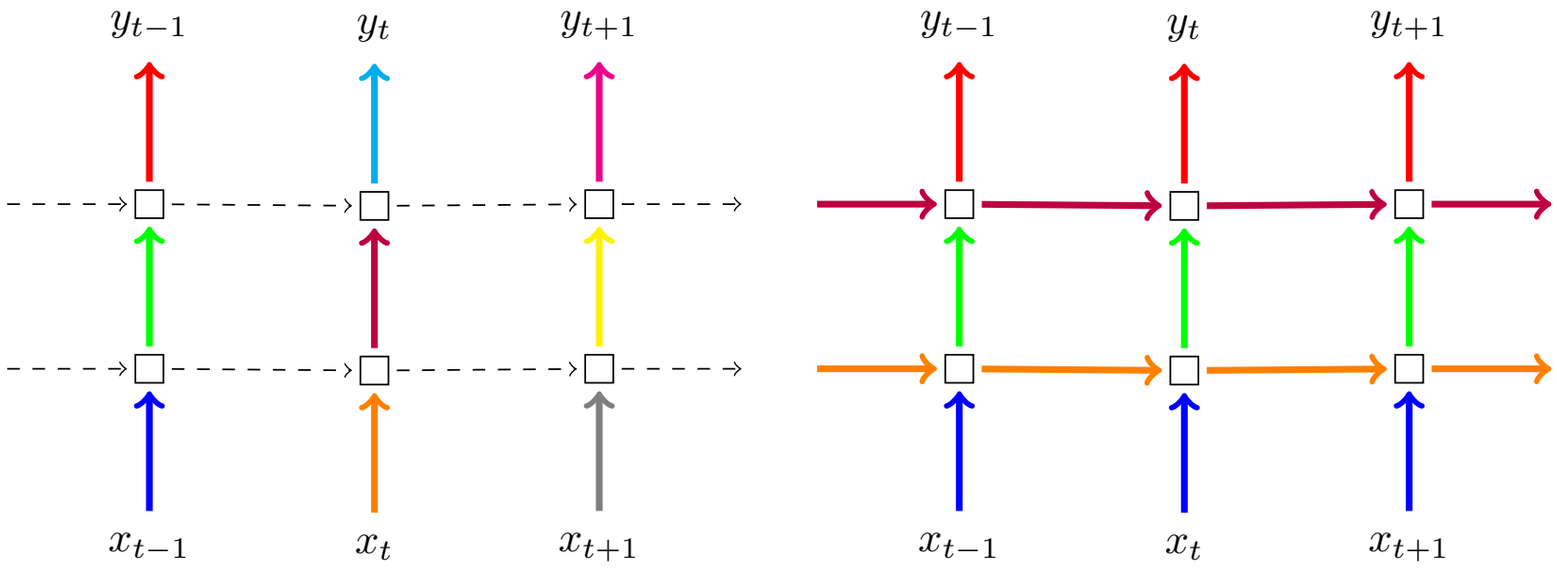
- Still an open question how to perform well.
- Yarin Gal's Dropout:



each color is a different random dropout mask

Dropout in RNNs

- Still an open question how to perform well.
- Yarin Gal's Dropout:



each color is a different random dropout mask

Practicalities

- Most toolkits require a **fixed** computation graph for all examples.

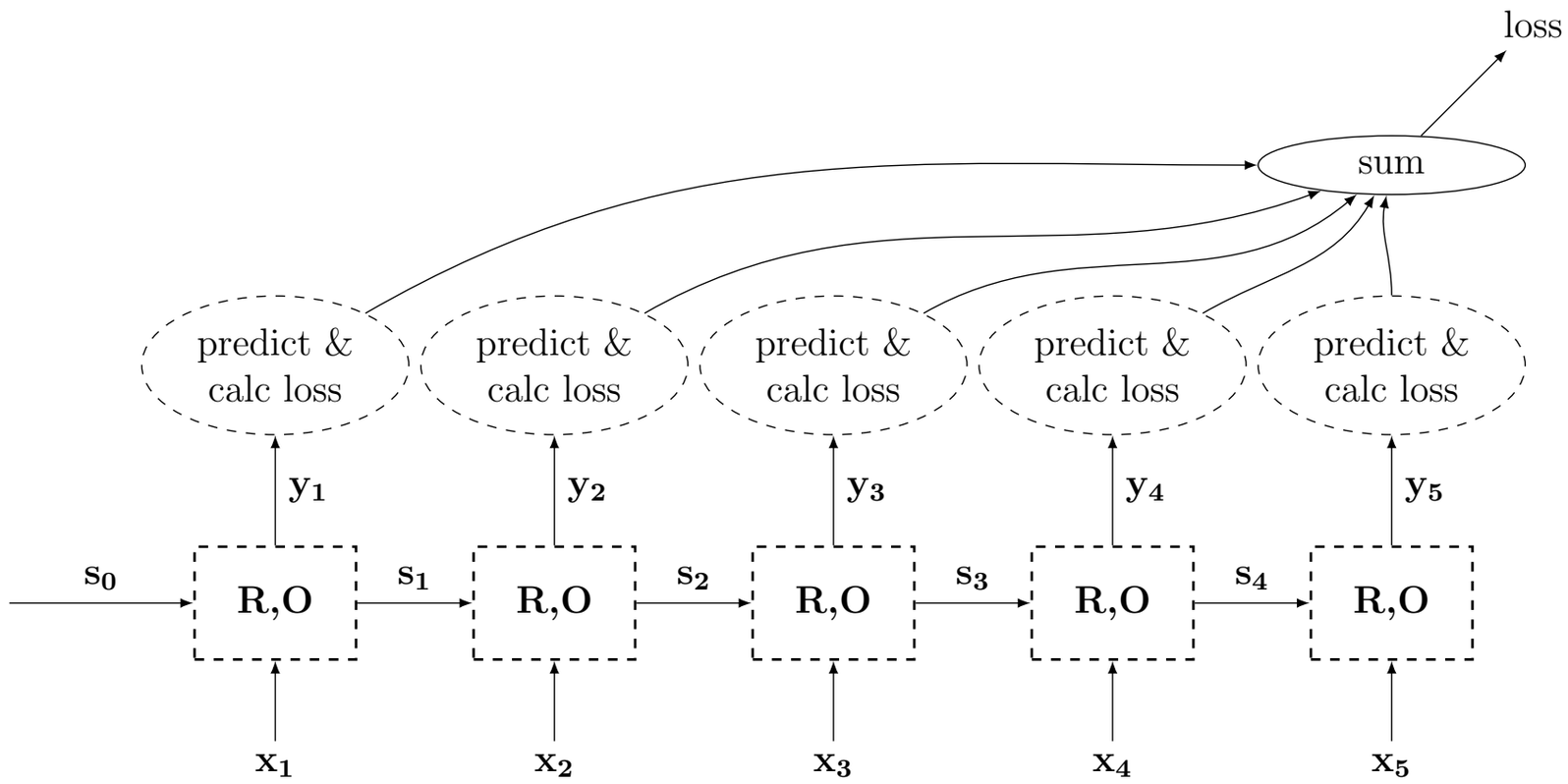
- But RNNs have **different** input lengths. What do we do?

  - Option 1:
    Use a tool that does not pose this limitation.

  - Option 2:
    Enforce max length + 0 padding for shorter sequences.

Bi-Directional RNNs

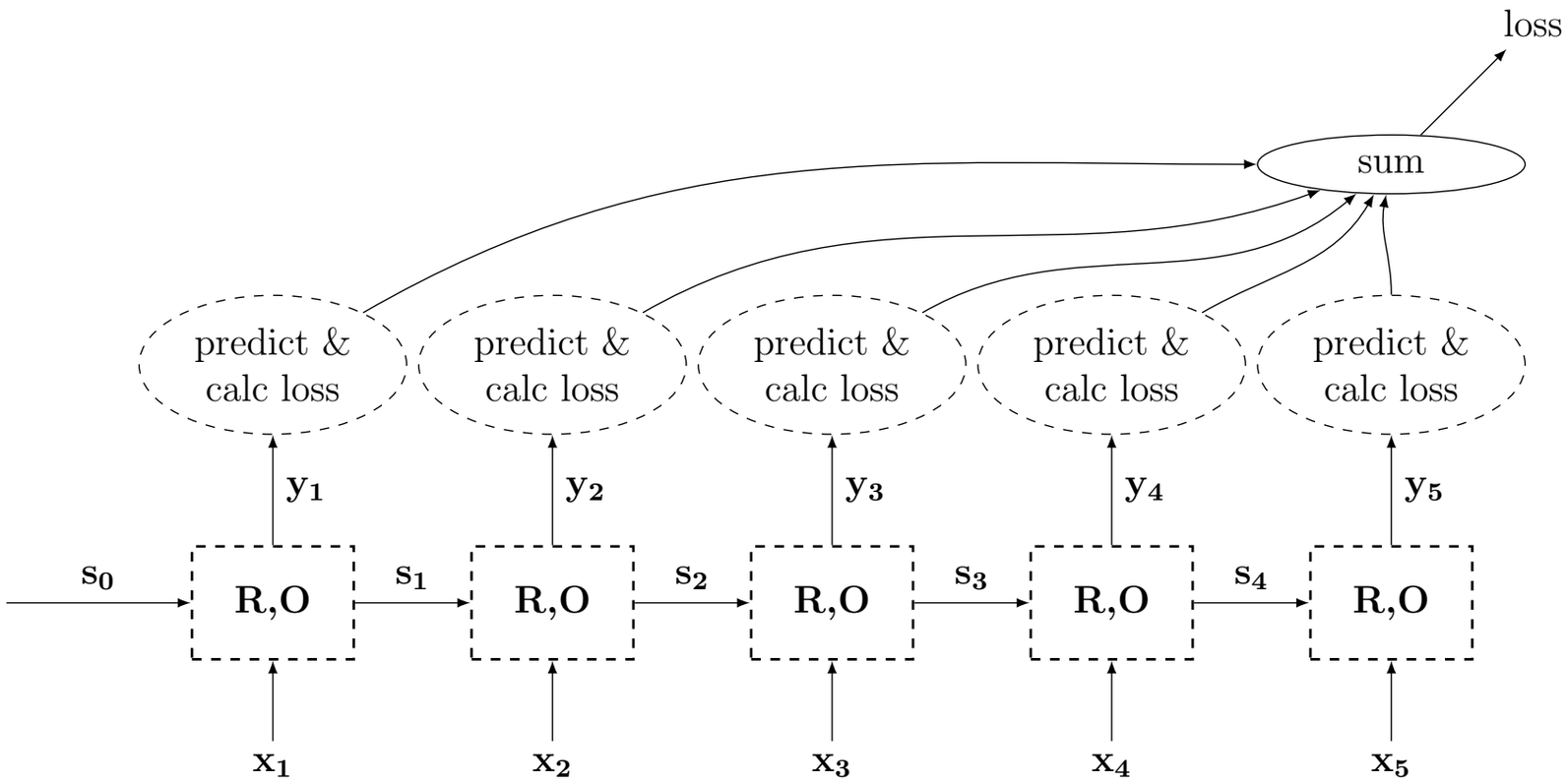
Transducers



Transducers for POS tagging?

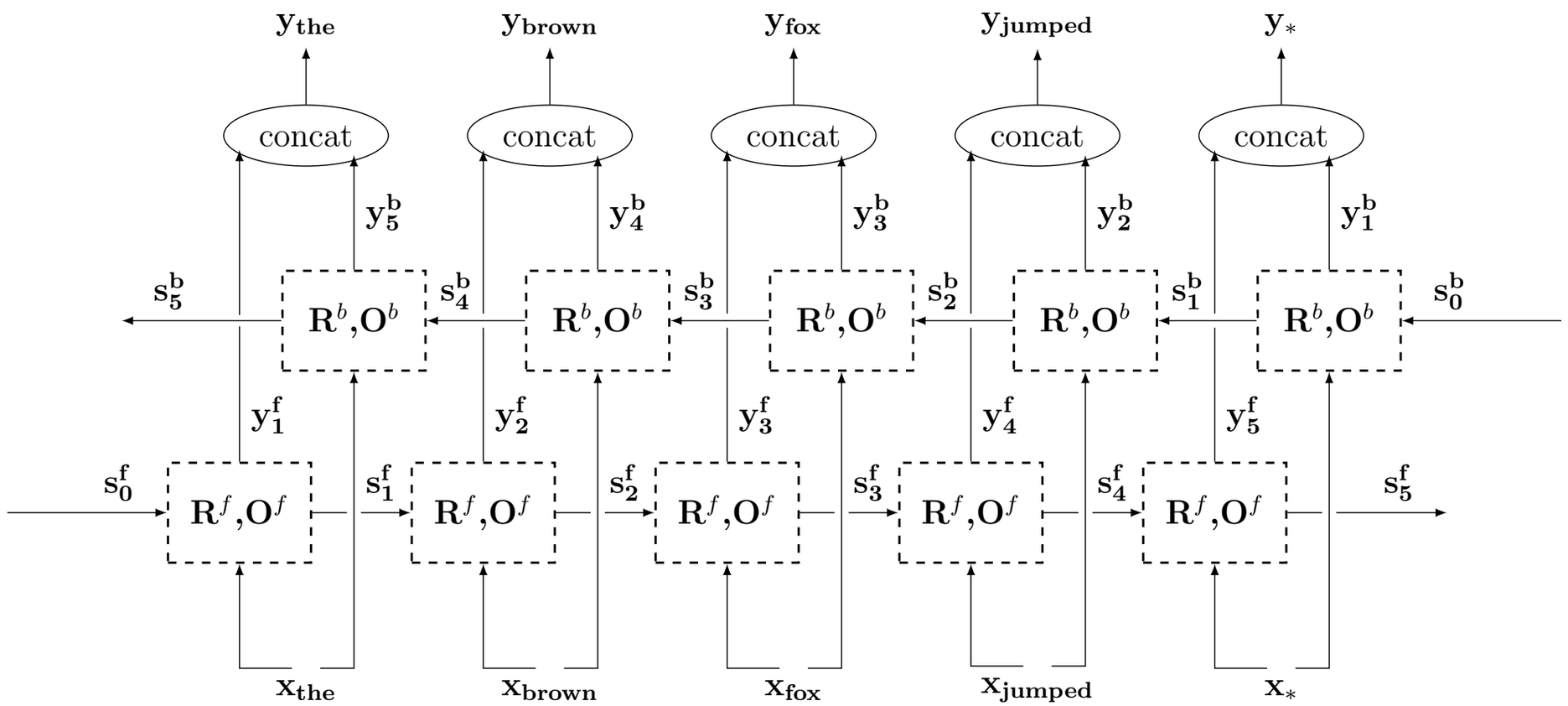
Can predict the tag of word i based on words 1,...,i-1, but...

Bidirectional RNNs



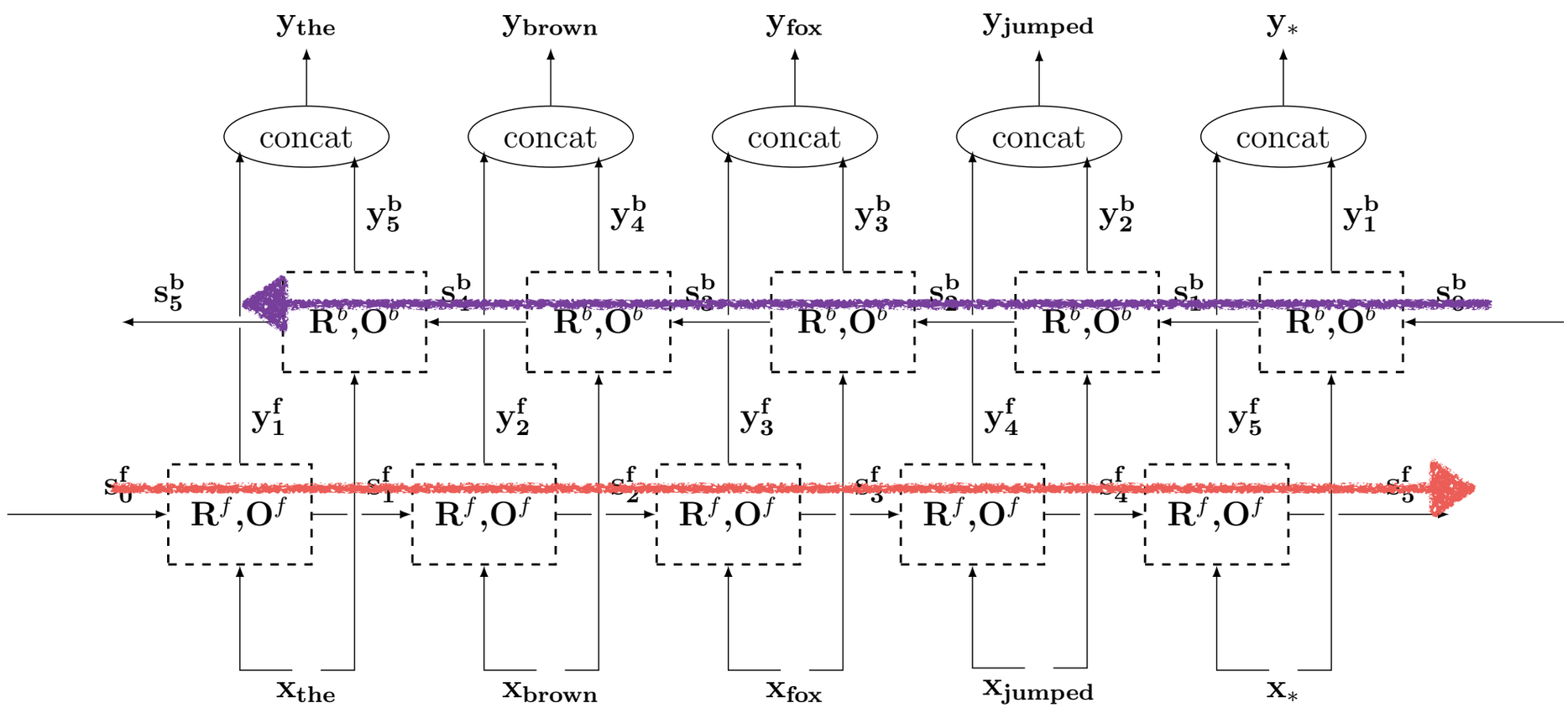
Each state encodes the entire history up to that state.
This is not bad. But what about the future?

Bidirectional RNNs



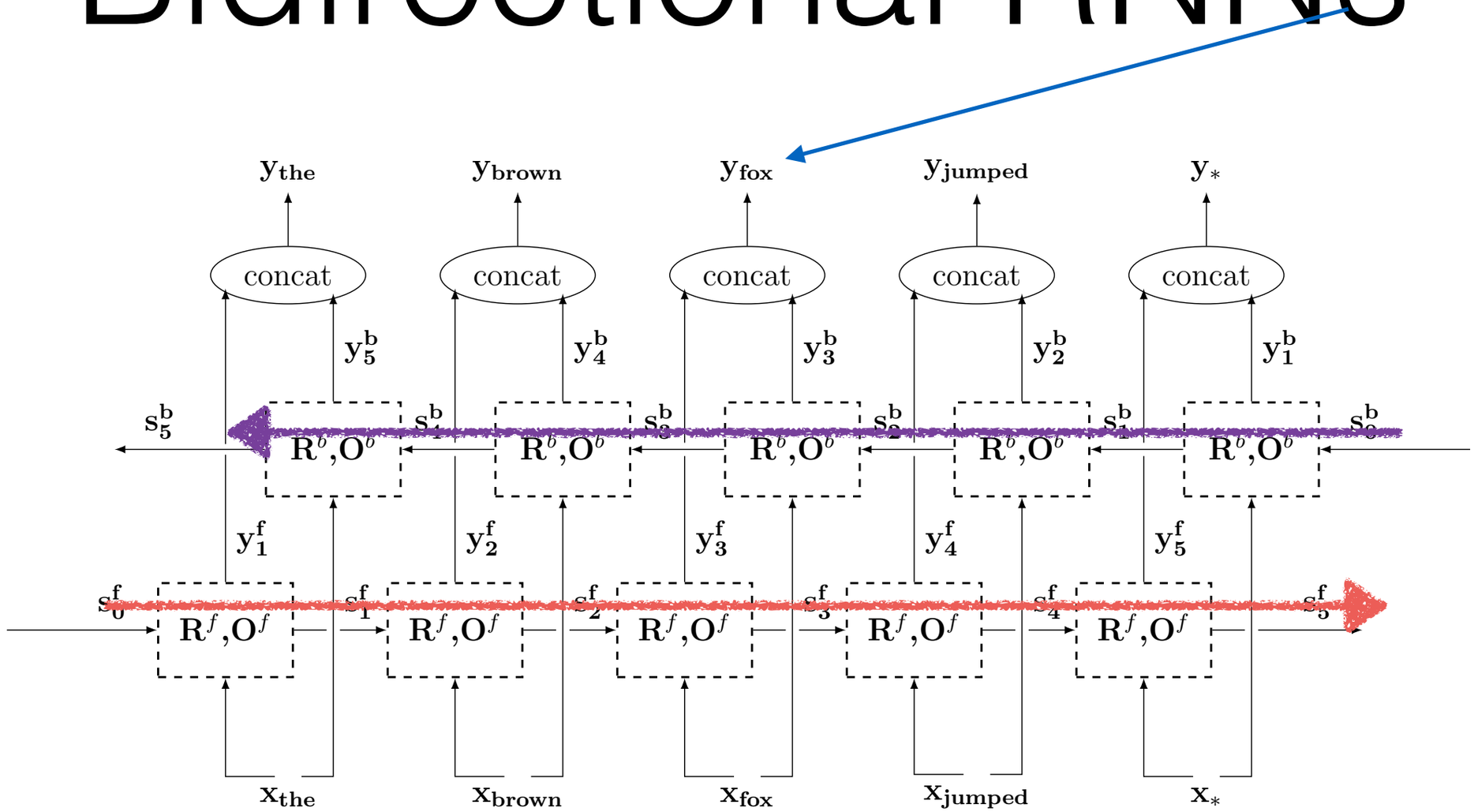
One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

Bidirectional RNNs



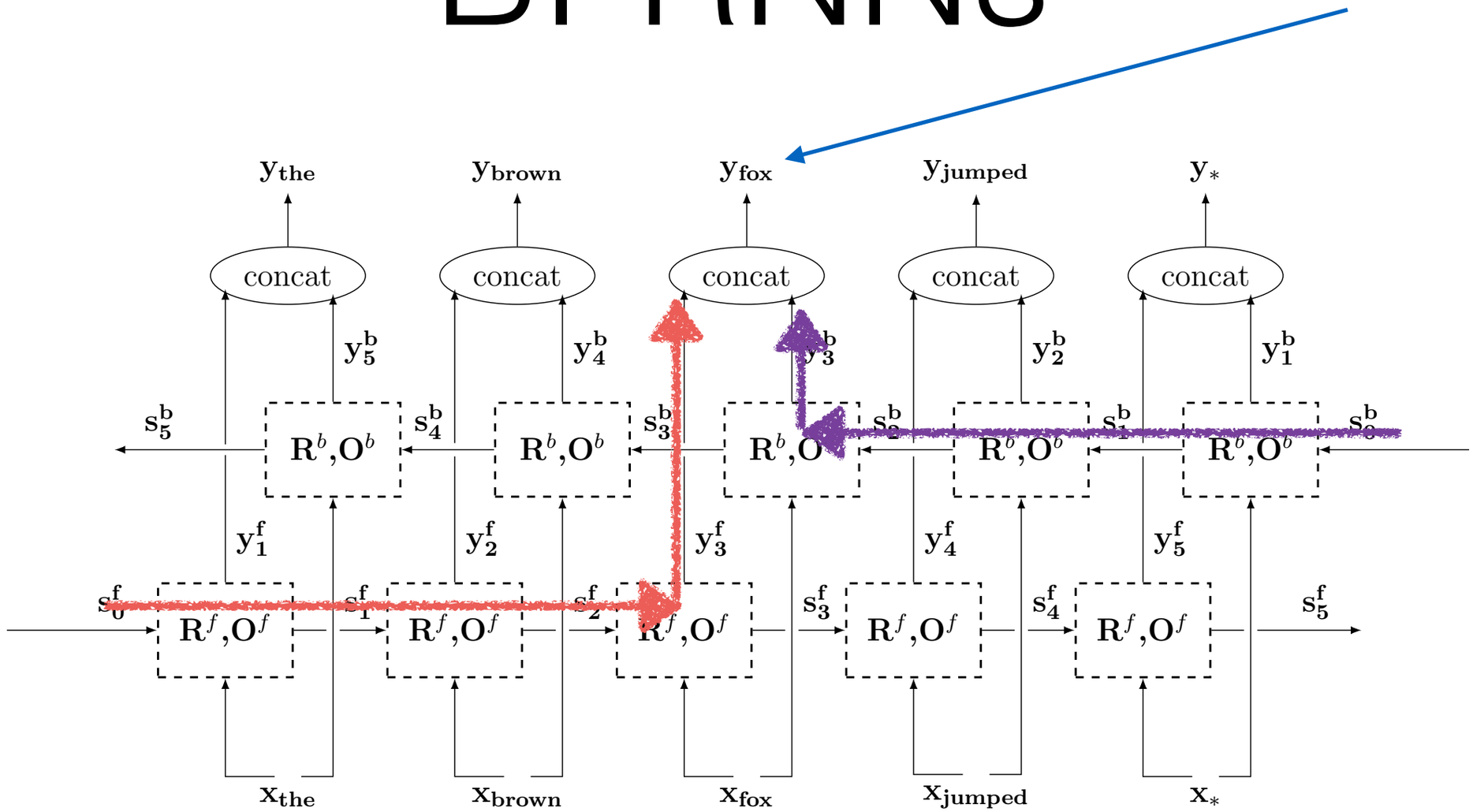
One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

Bidirectional RNNs



One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

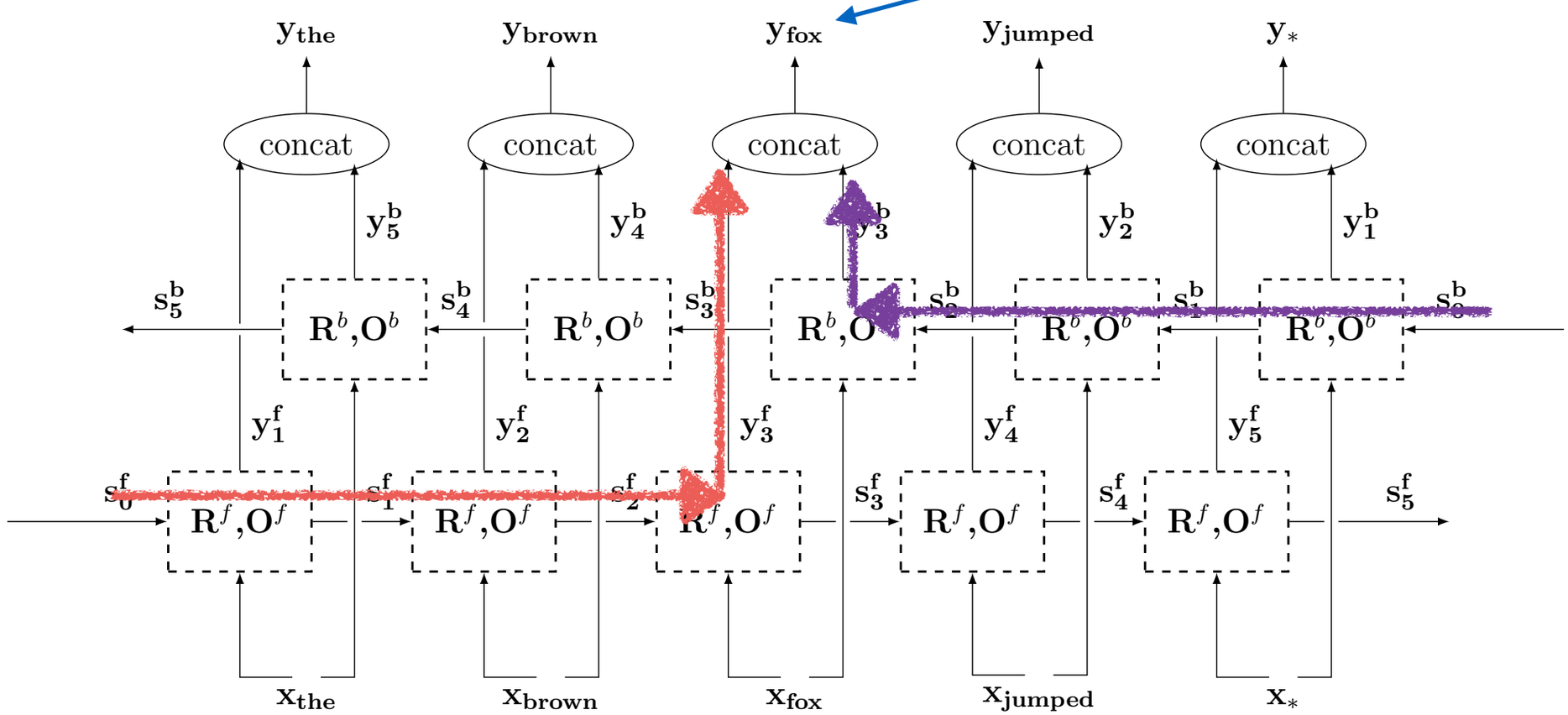
BI-RNNs



One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

BI-RNNs

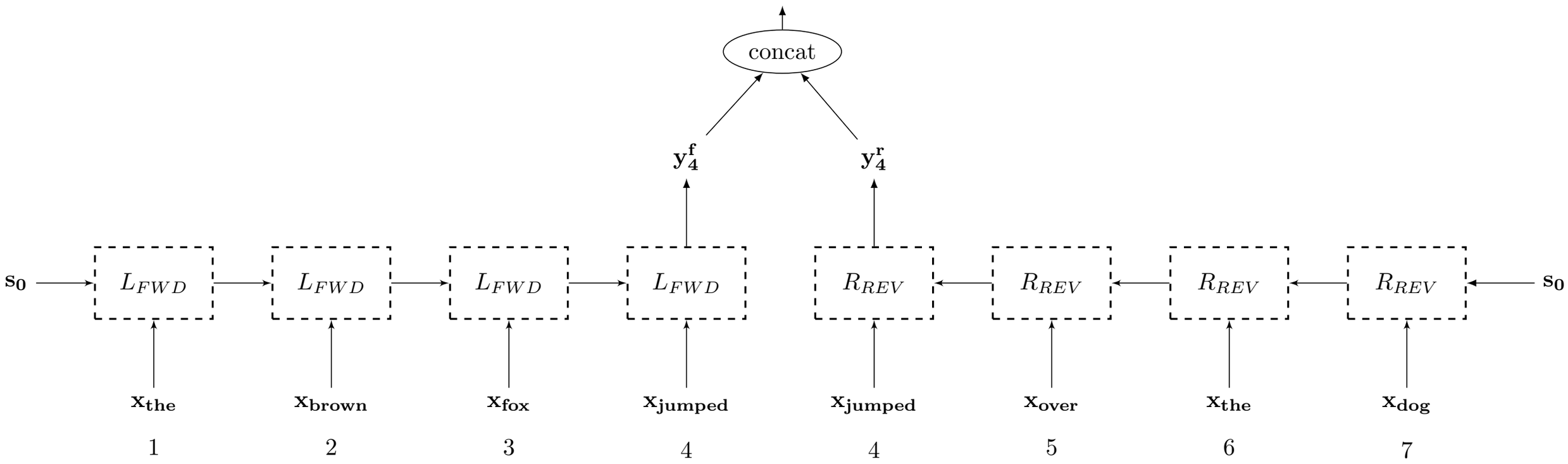
an infinite window around the word.



One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

BI-RNNs

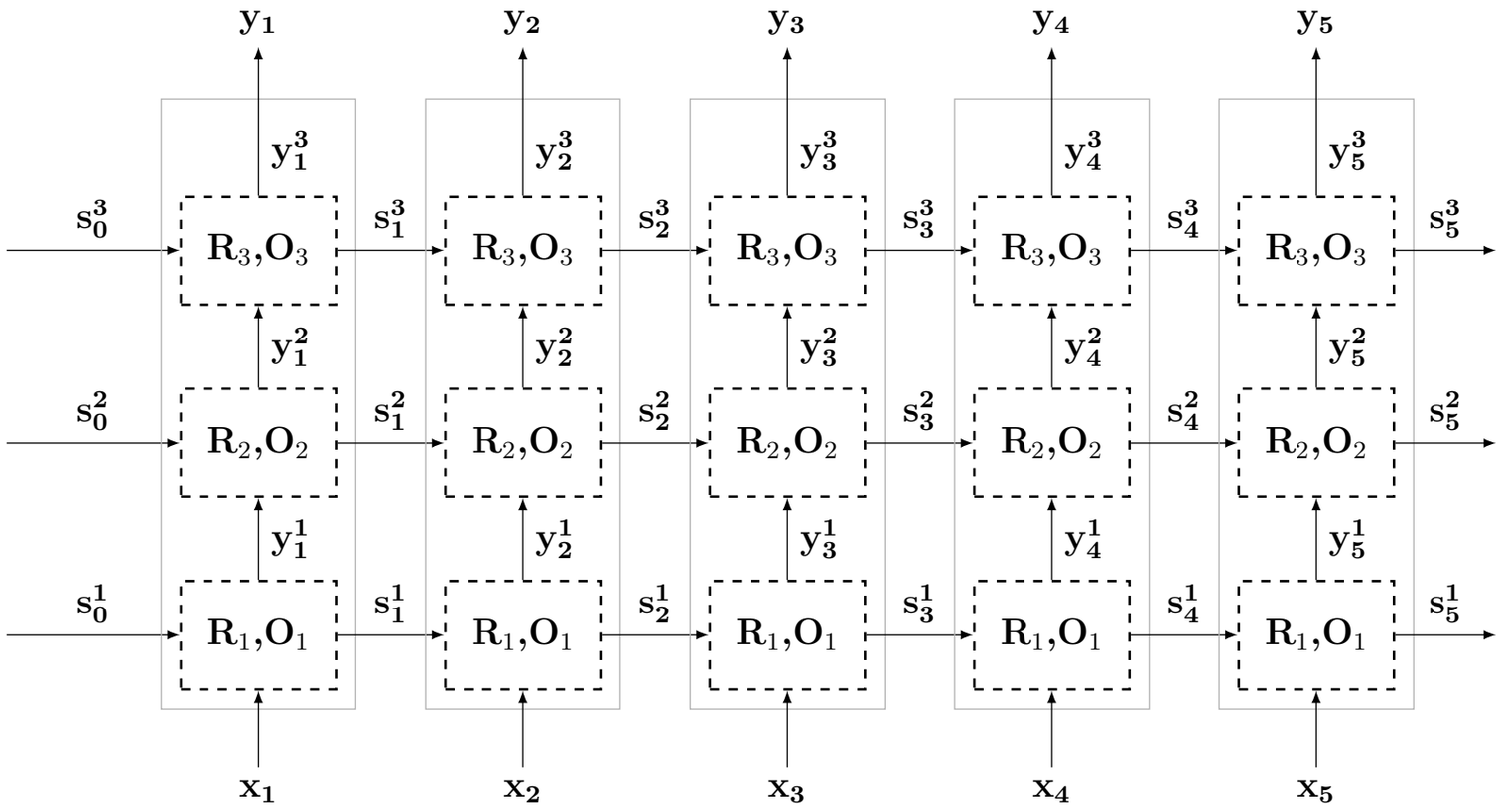
an infinite window around the word.



$$BiRNN(\mathbf{x_{1:7}}, 4) = [\mathbf{y_4^F}; \mathbf{y_4^R}]$$
$$\mathbf{y_4^F} = RNN_F(\mathbf{x_{1:4}})$$
$$\mathbf{y_4^R} = RNN_R(\mathbf{x_{7:4}})$$

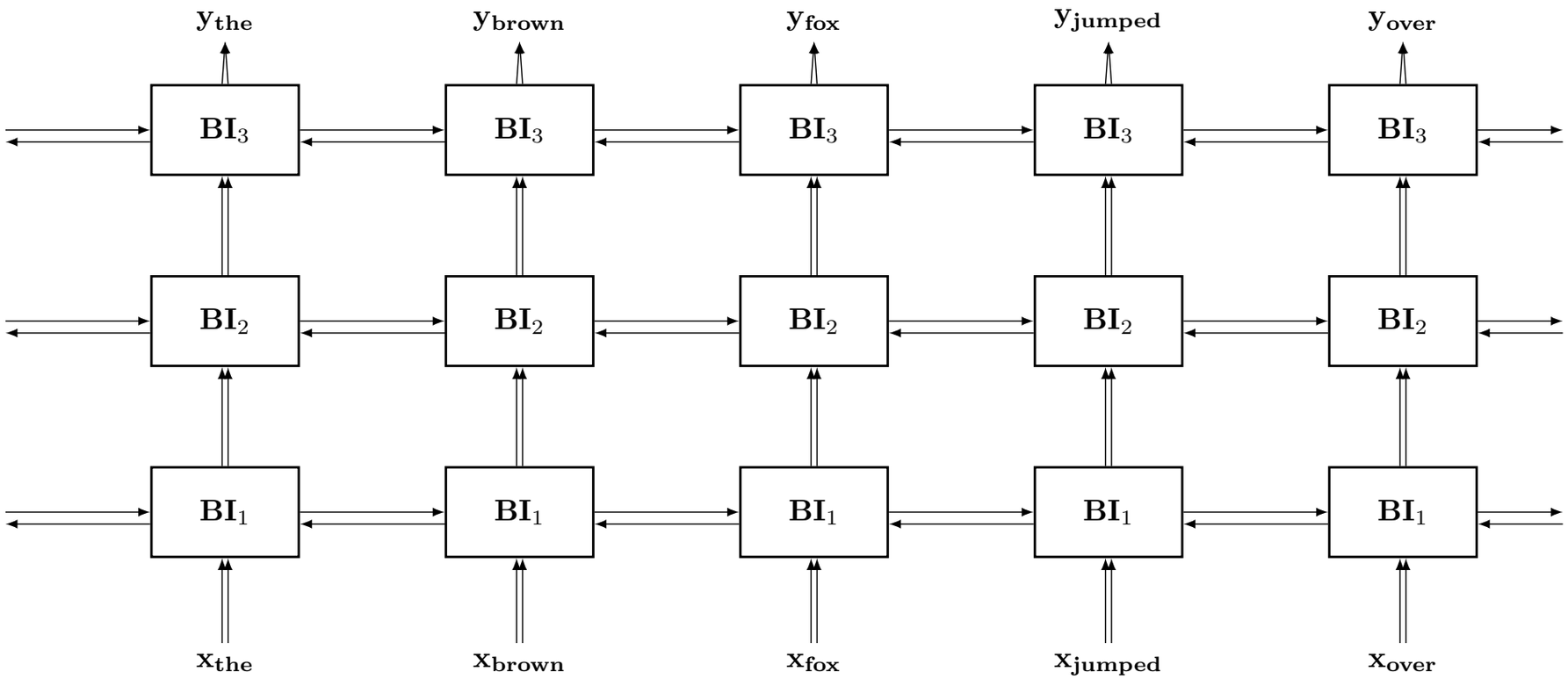
One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

"deep RNNs"



RNN can be stacked
deeper is better!
(better how?)

Deep Bi-RNNs

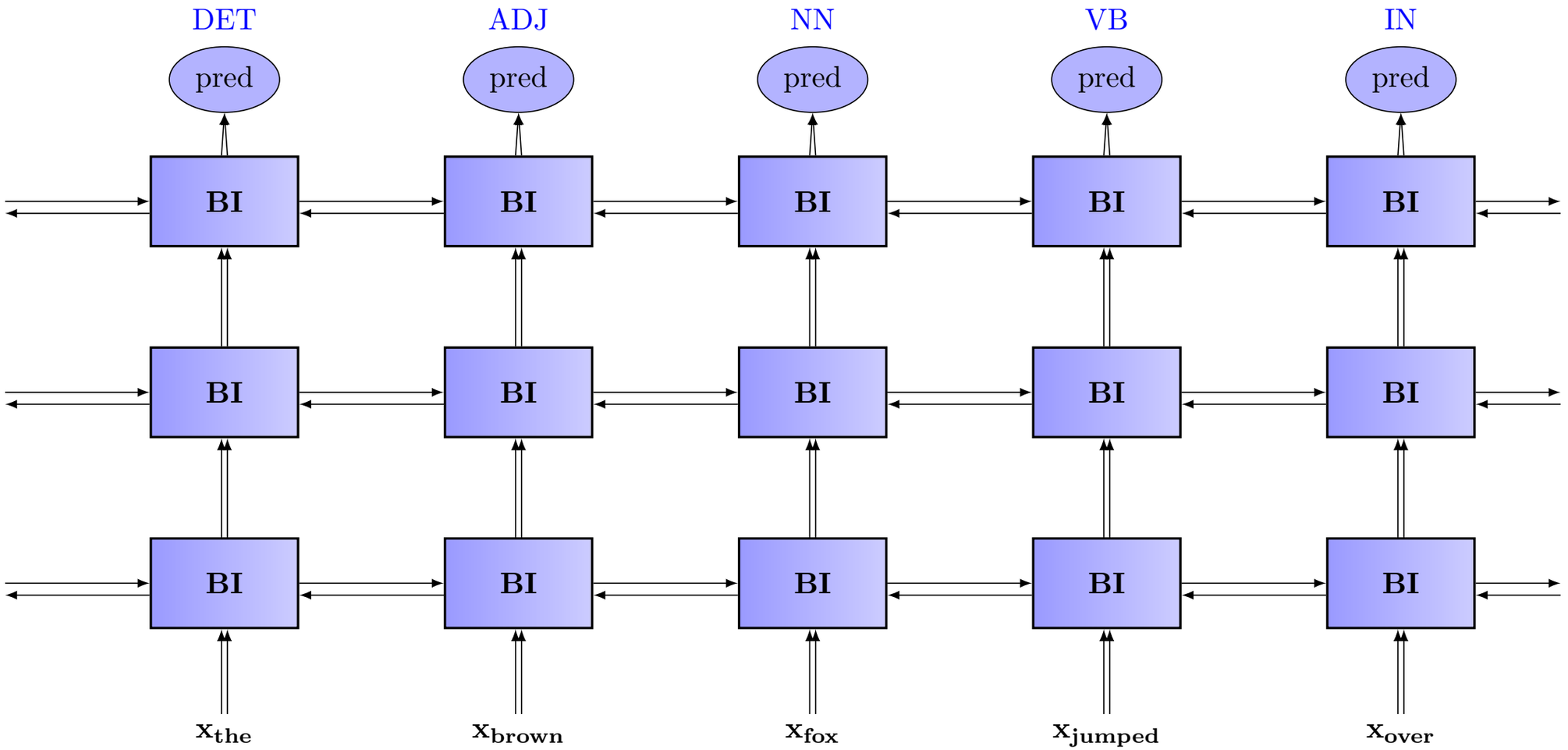


BI-RNN can also be stacked

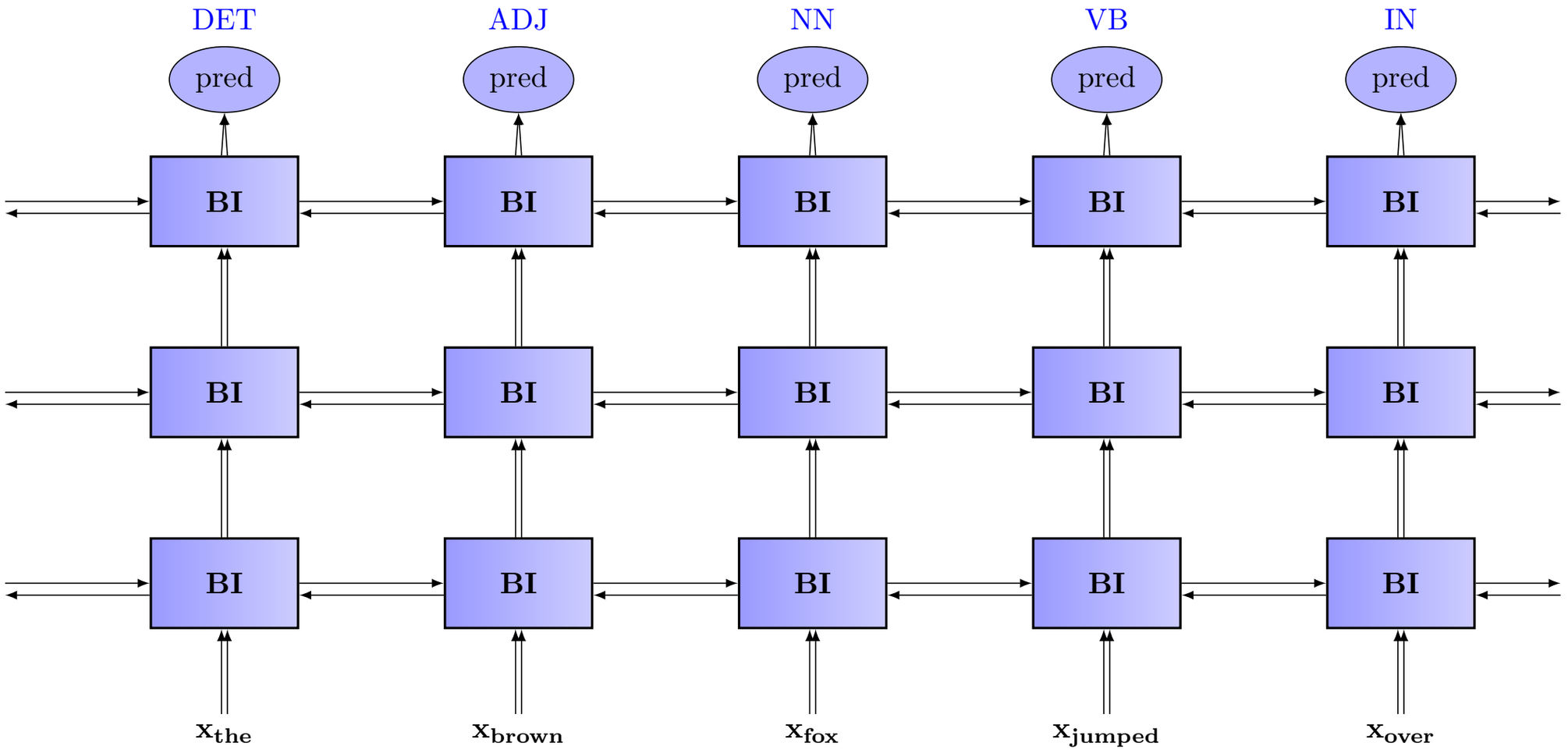
(Deep) BI-RNNs

- provide an "infinite" window around a focus word.
- learn to extract what's important.
- easy to train!
- very effective for sequence tagging.

Sequence tagger (here, POS)

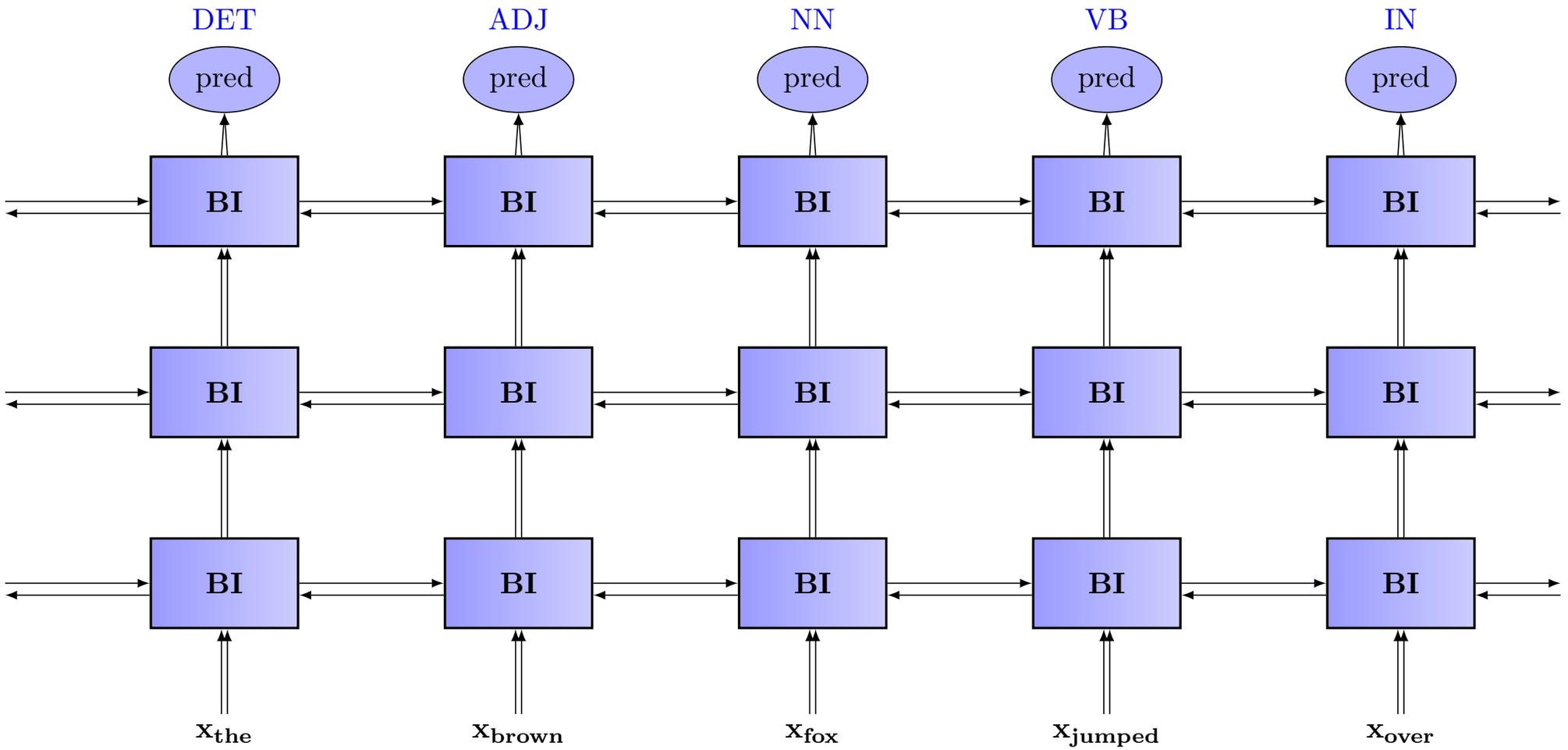


Sequence tagger (here, POS)



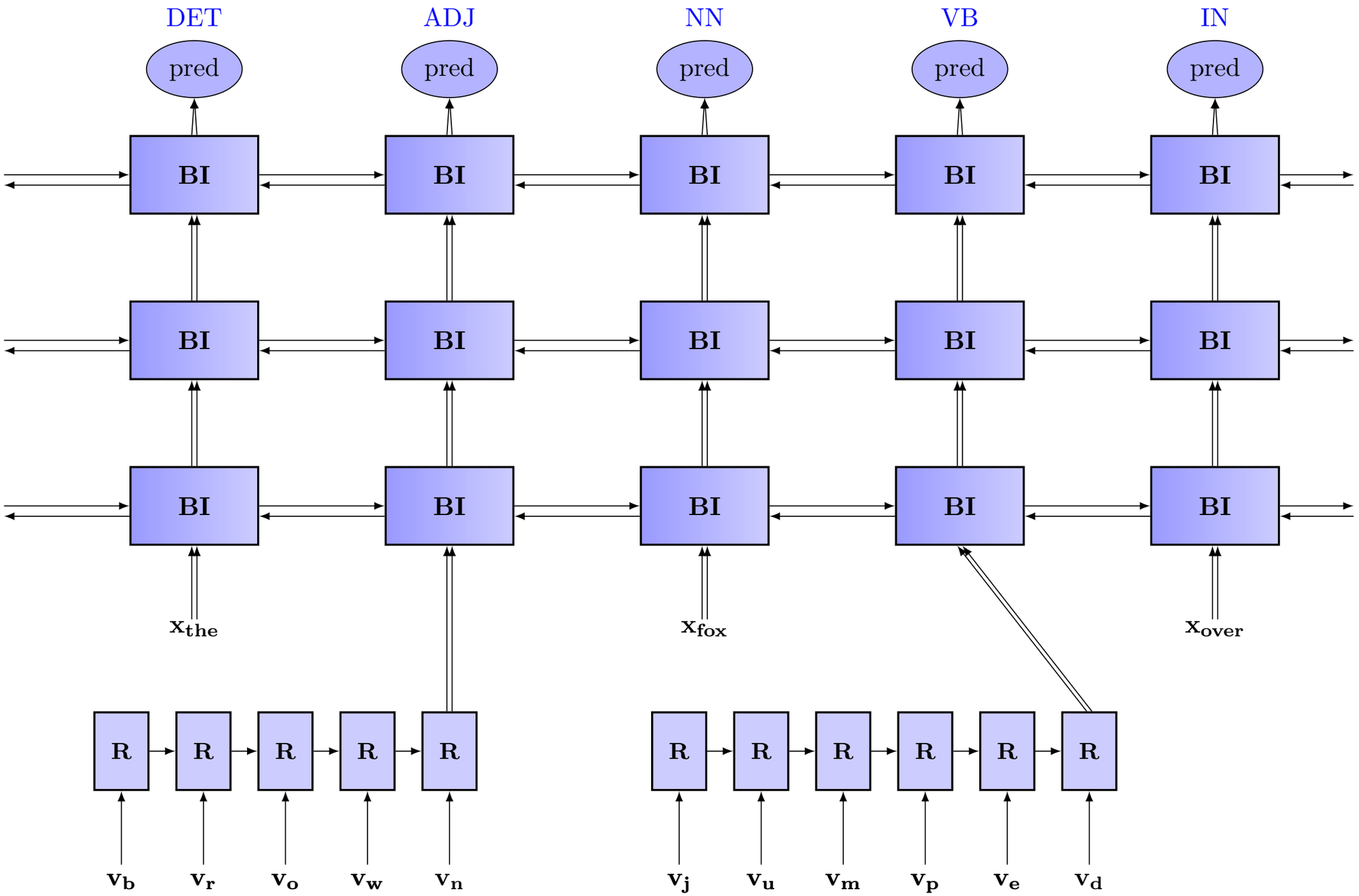
RNNs can be easily "nested".
Output of one RNN feeds into another RNN.

Sequence tagger (here, POS)



Back-off to char-level RNNs for Unknown Words:

(inspired by Ling et al, Ballesteros, Dyer and Smith)

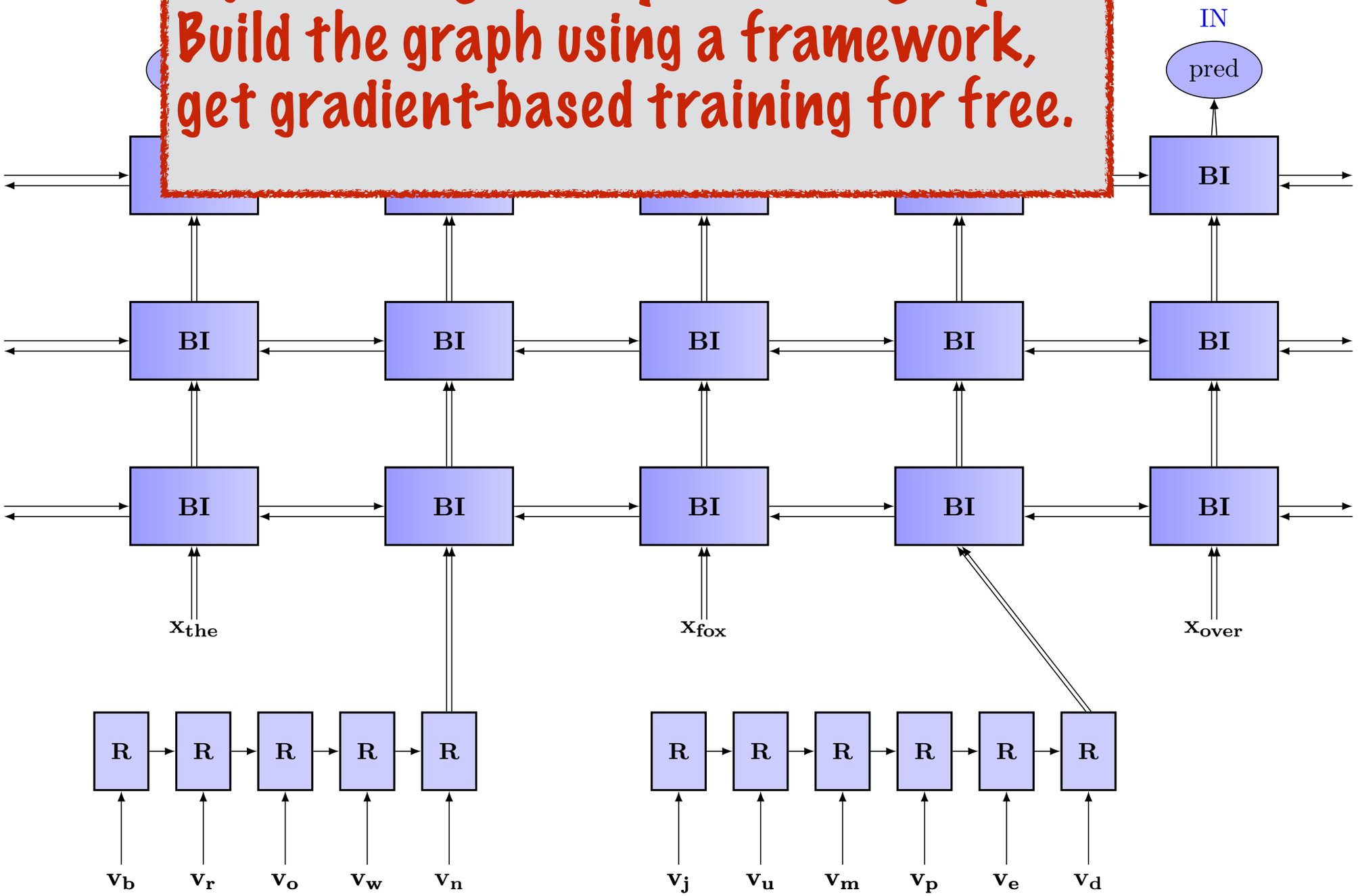


Back-off to

Reminder: this network
is just a large computation graph.
Build the graph using a framework,
get gradient-based training for free.

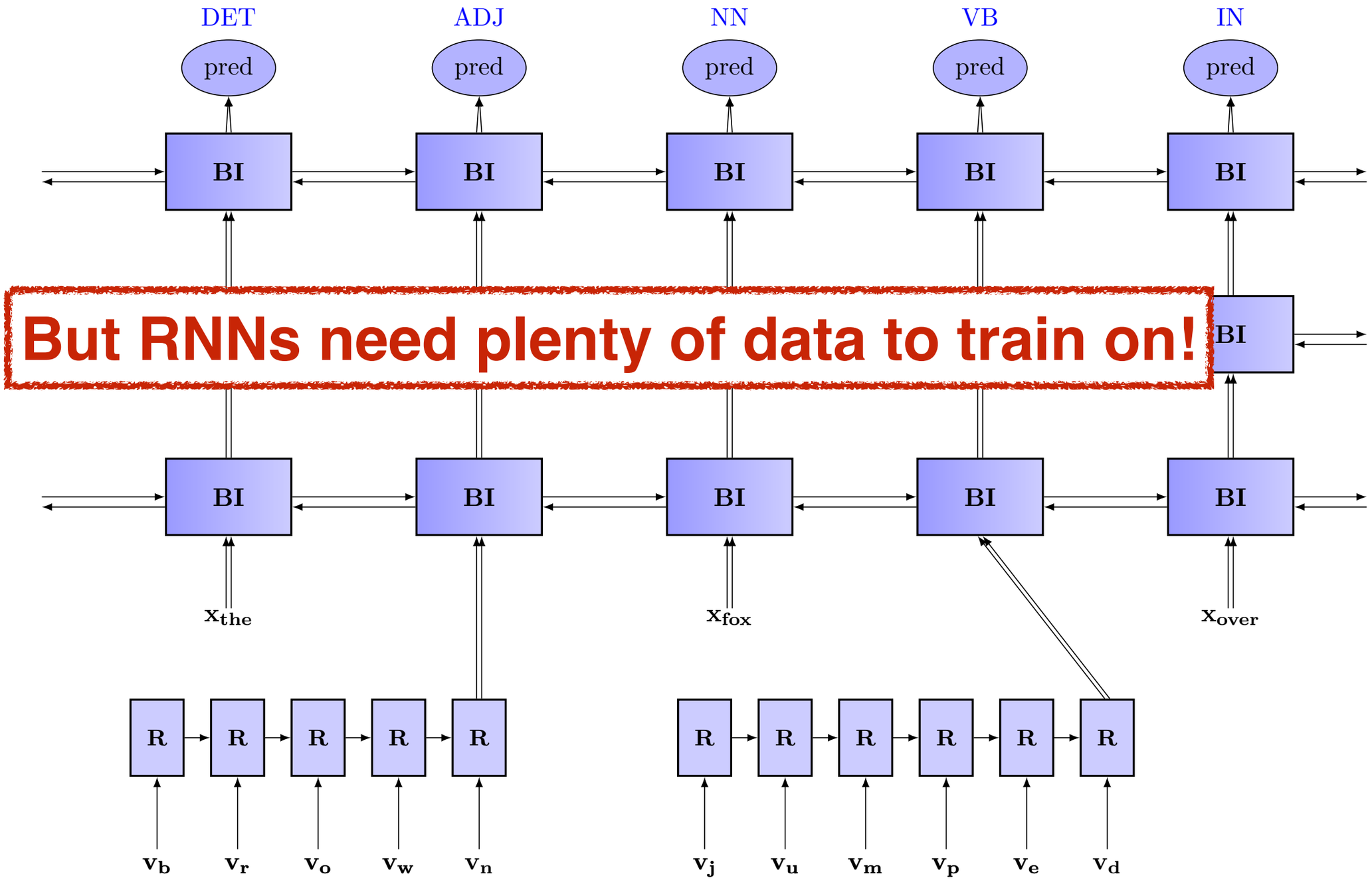
s:

yer and Smith)



Back-off to char-level RNNs for Unknown Words:

(inspired by Ling et al, Ballesteros, Dyer and Smith)



But RNNs need plenty of data to train on!

Do they, really?

Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss

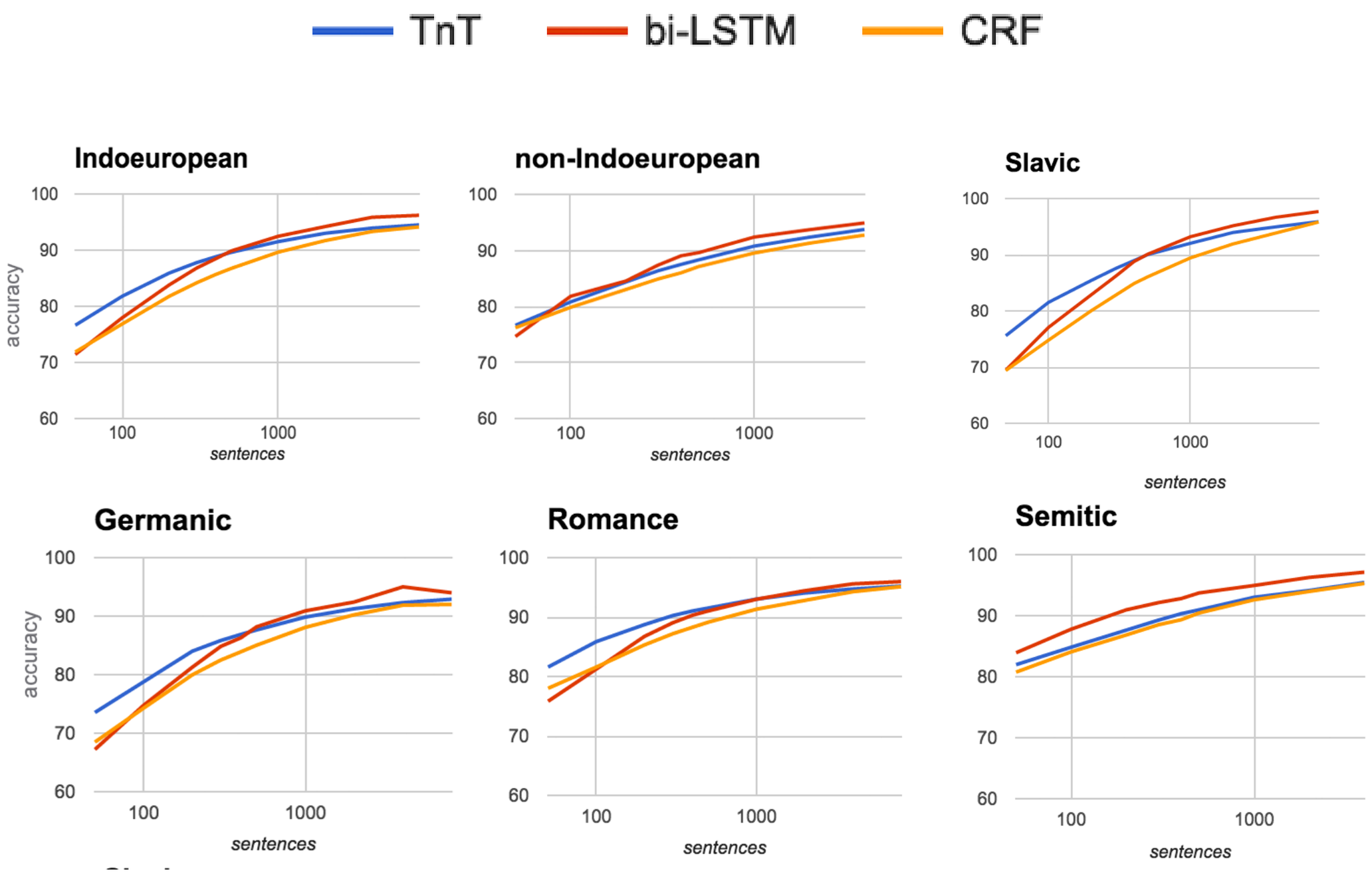
Barbara Plank
University of Groningen
The Netherlands
b.plank@rug.nl

Anders Søgaard
University of Copenhagen
Denmark
soegaard@hum.ku.dk

Yoav Goldberg
Bar-Ilan University
Israel
yoav.goldberg@gmail.com

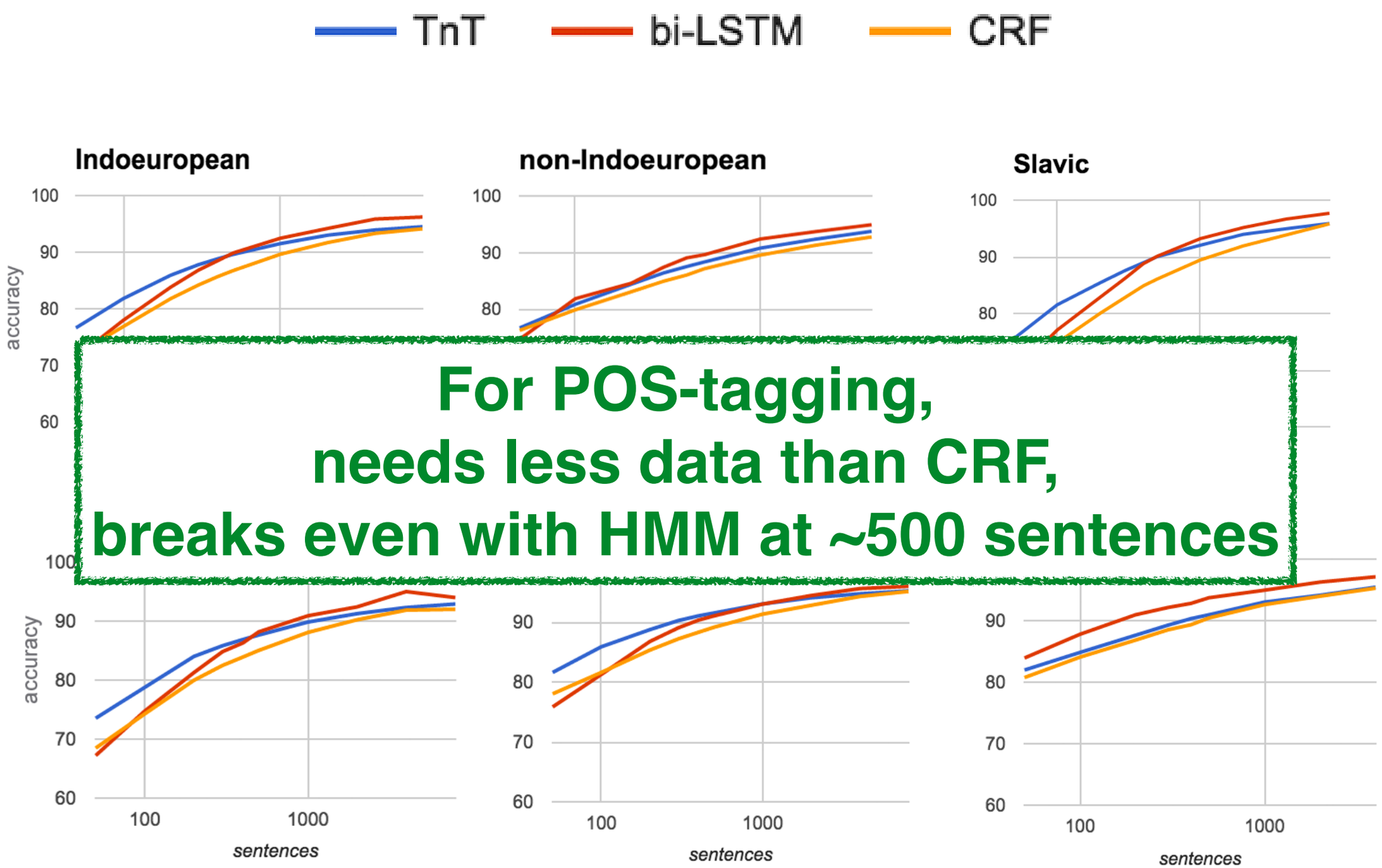
But RNNs need plenty of data to train on!

Do they, really?



But RNNs need plenty of data to train on!

Do they, really?



RNN Recap

- Trainable encoders of sequential data.
- Allow capturing non-markovian (Infinite) history.
- Used as "lego bricks", feed into other components.
- Very effective for language modeling.
- Very effective for feature extraction.

BI-RNN's Recap

- Represent the history up to a point in the sequence, and the future from a point in a sequence.

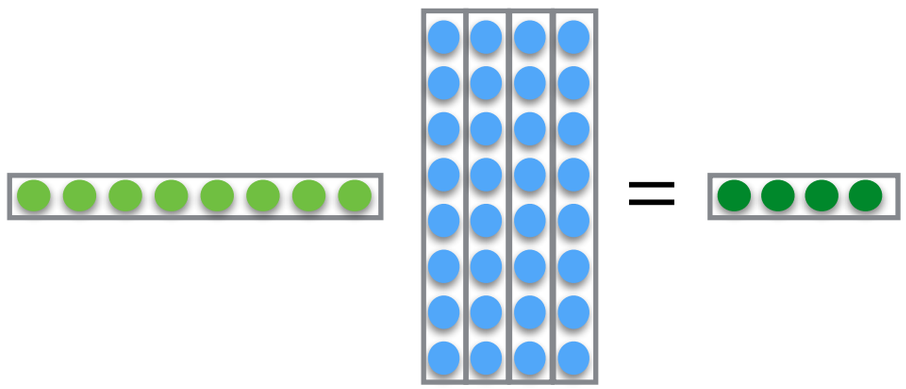
- Feed into an MLP (or linear classifier) to classify the point based on history and future.

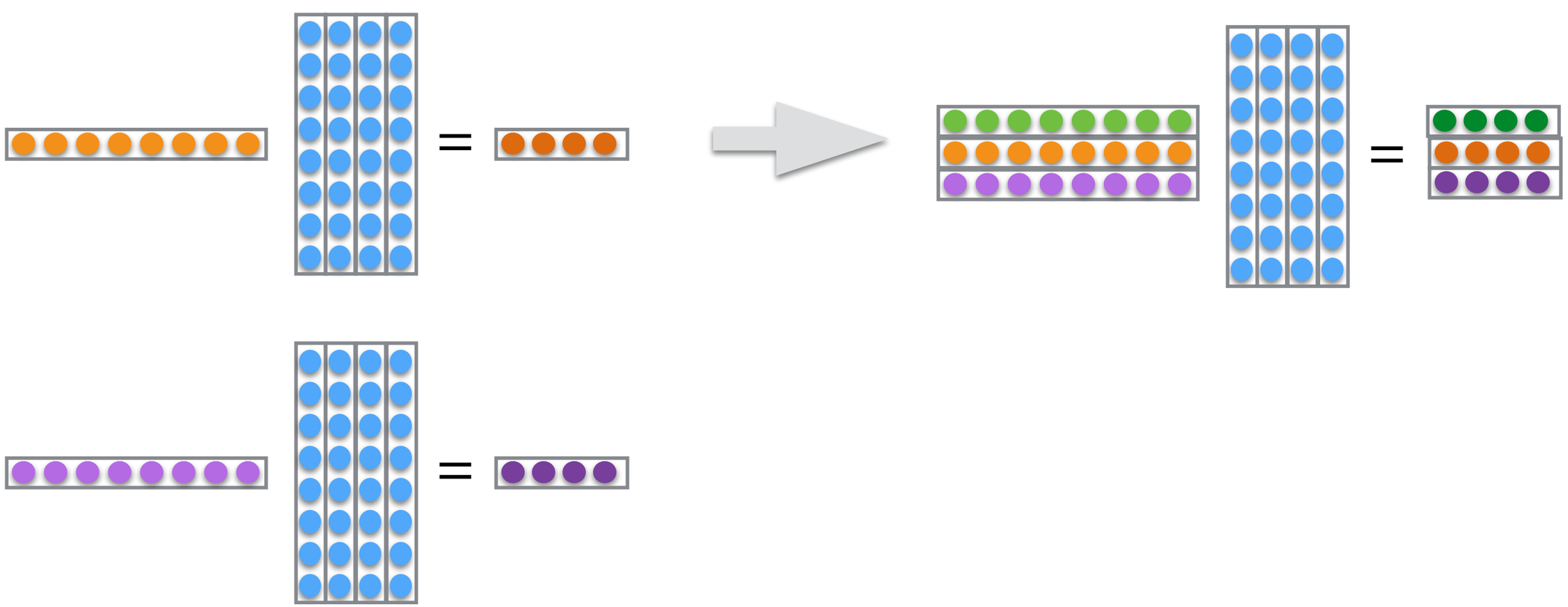
- The network learns what's important in the history and future for the given prediction task.

- **"Infinite window"**

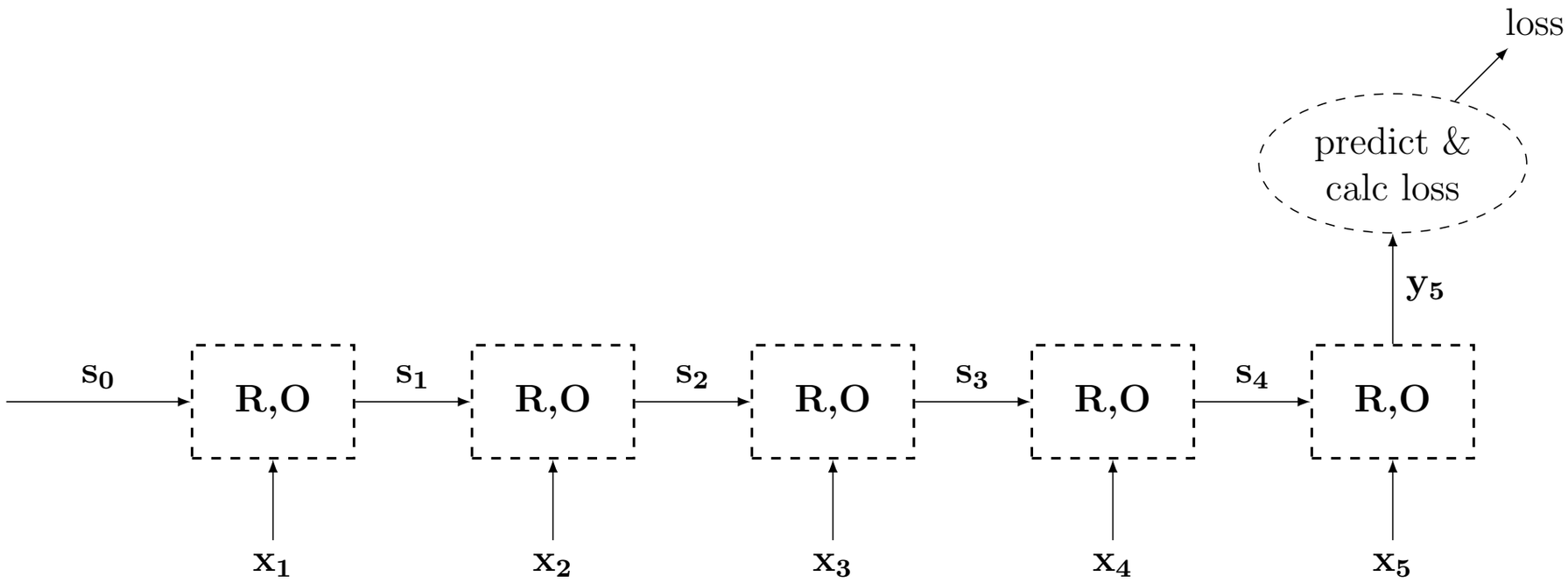
Batching in RNNs

Batching Reminder





Batching in RNNs

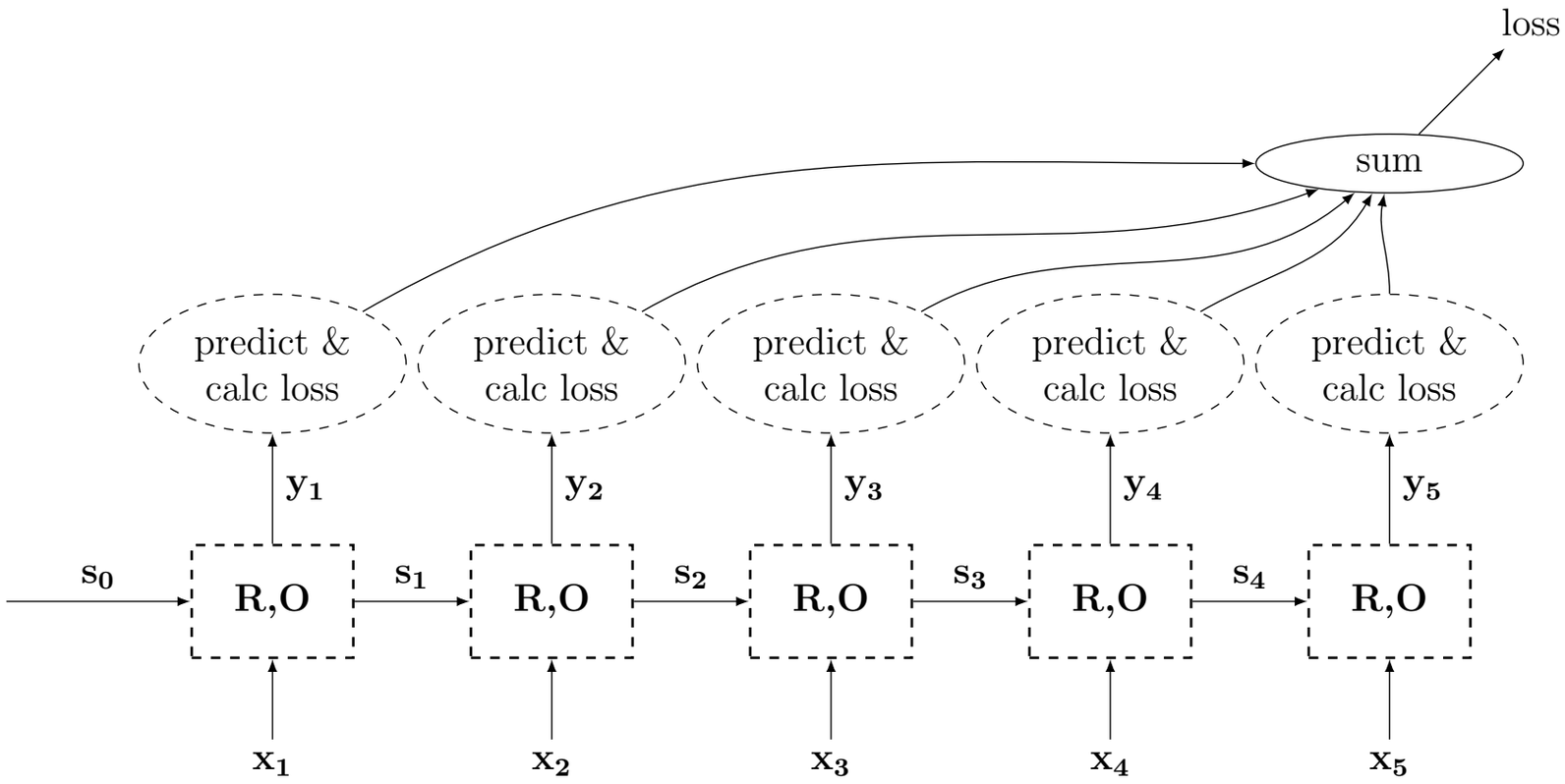


Batching in RNNs

- Sequential in nature, very little parallelism.

- (Compare, e.g., to a Convolutional Network)

Batching in RNNs



in transduction mode, can batch the final output predictions

Batching in RNNs

- **proposals:**
  newer architectures, that try to make the sequence-dependent parts cheaper.

QUASI-RECURRENT NEURAL NETWORKS

James Bradbury,* Stephen Merity,* Caiming Xiong & Richard Socher
Salesforce Research
Palo Alto, California
{james.bradbury,smerity,cxiong,rsocher}@salesforce.com

Batching in RNNs

- **proposals:**
  newer architectures, that try to make the sequence-dependent parts cheaper.

QUASI-RECURRENT NEURAL NETWORKS

James Bradbury,* Stephen Merity,* Caiming Xiong & Richard Socher
Salesforce Research
Palo Alto, California
{james.bradbury,smerity,cxiong,rsocher}@salesforce.com

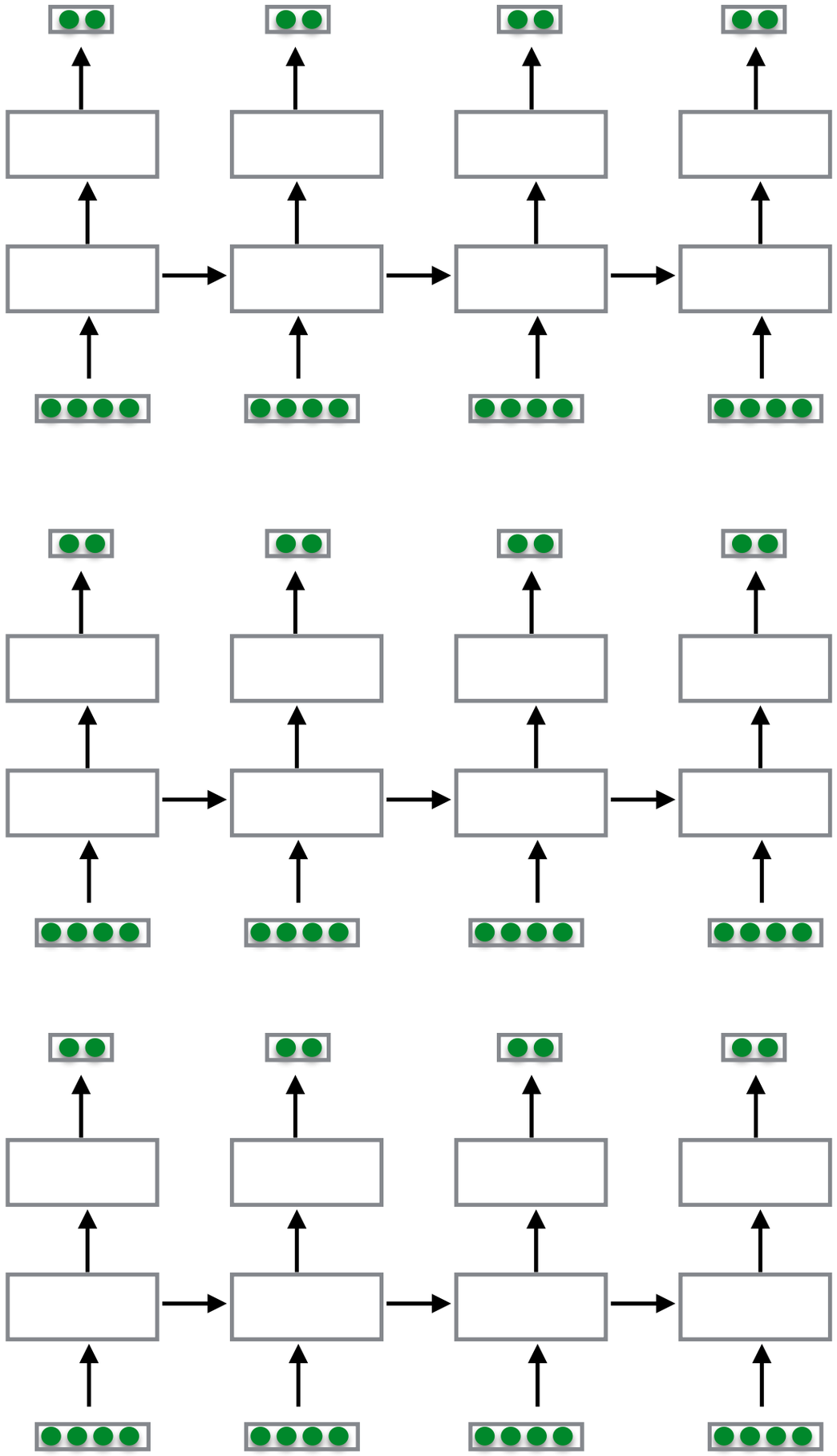
Attention Is All You Need

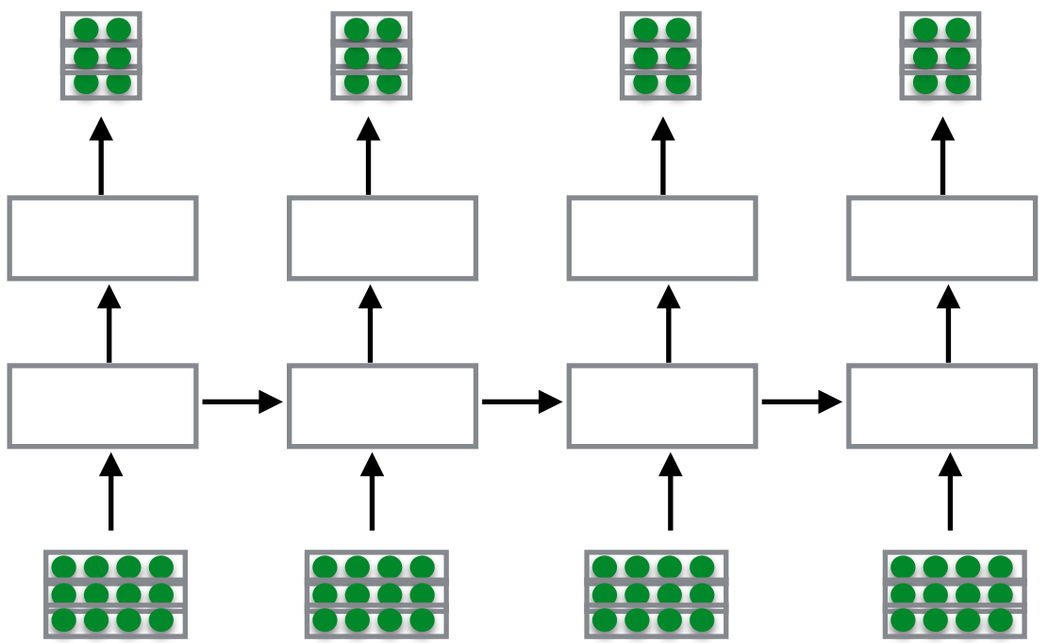
("transformers")

Batching in RNNs

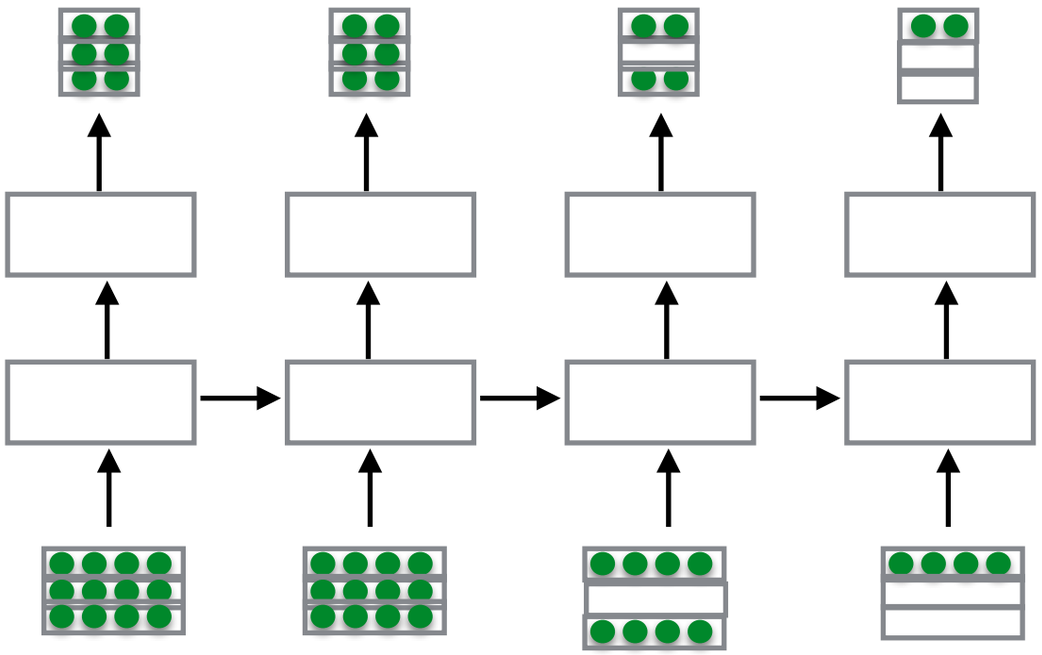
- **Can batch across sequences.**

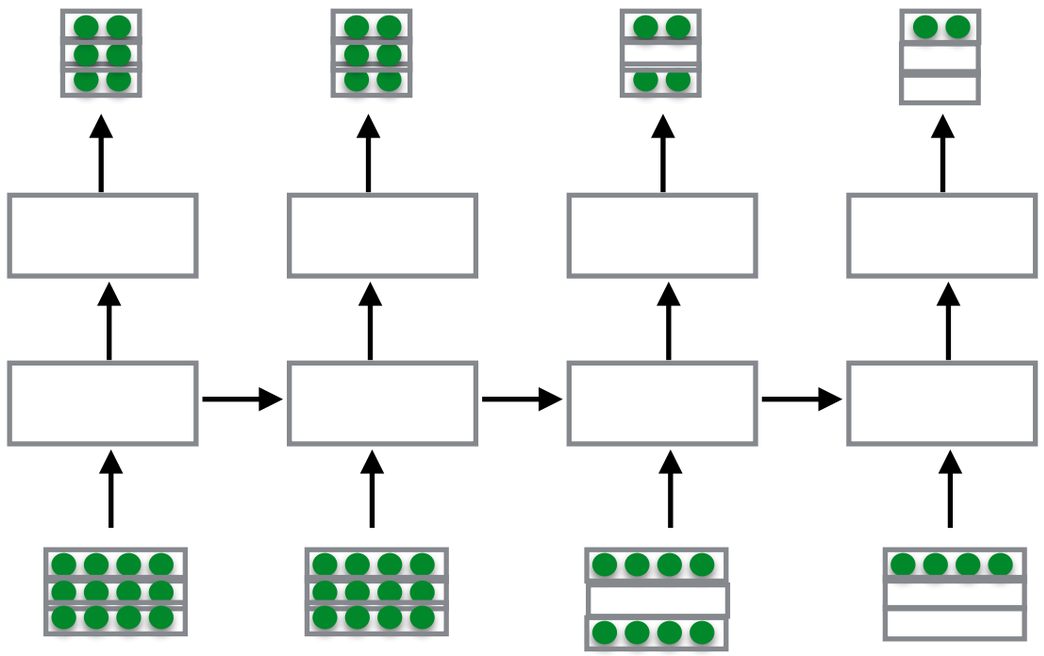
B I U
N L P





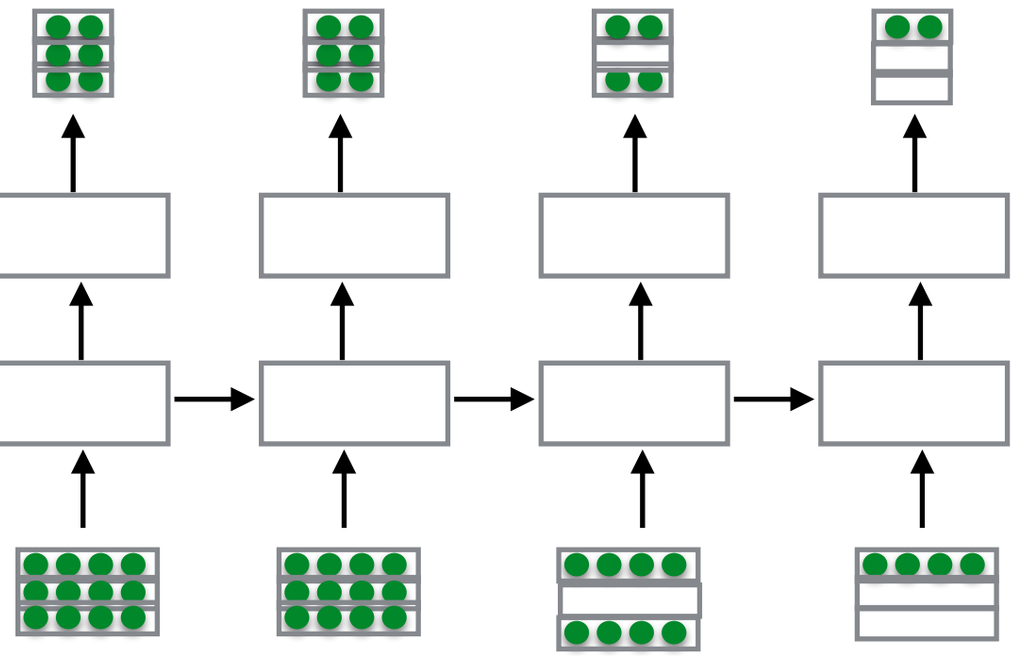
what if the sequences are different lengths?





masking

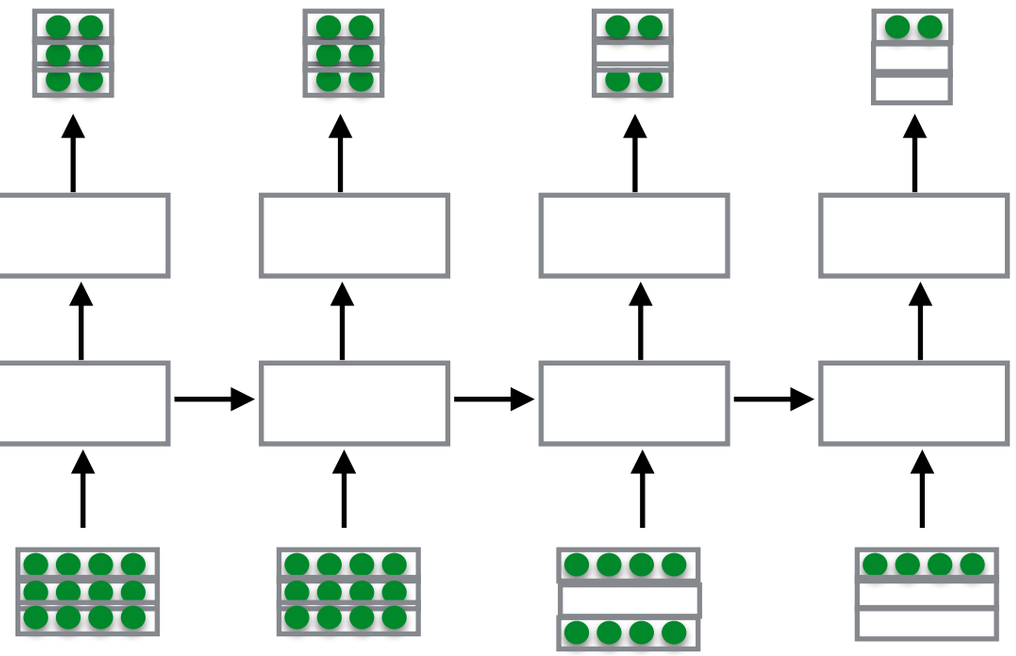
padding



masking

padding

this is how its done in TF, PyTorch.
supported also in DyNet, but...



masking

padding

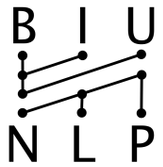
really annoying

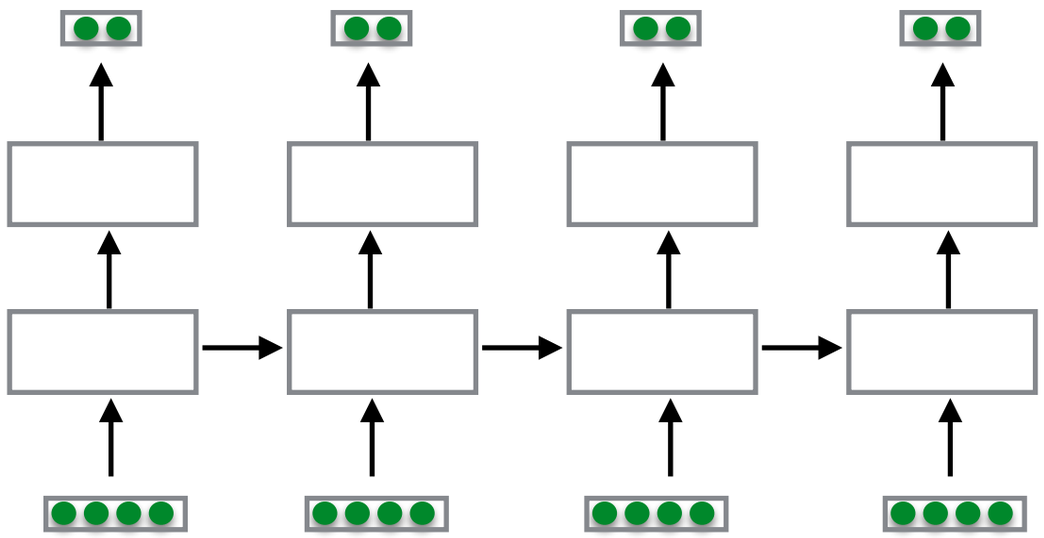
super-confusing with biLSTMs

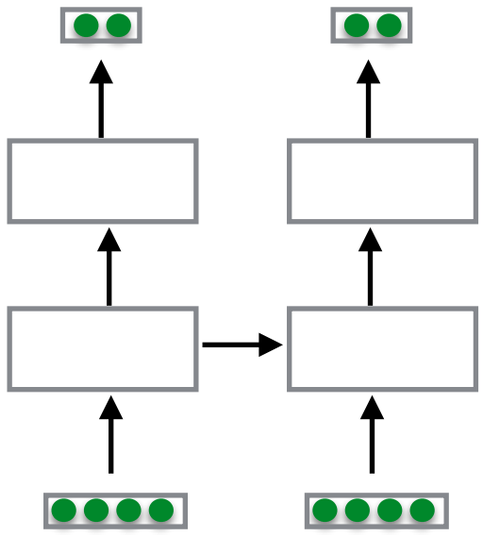
practically impossible with our complex networks

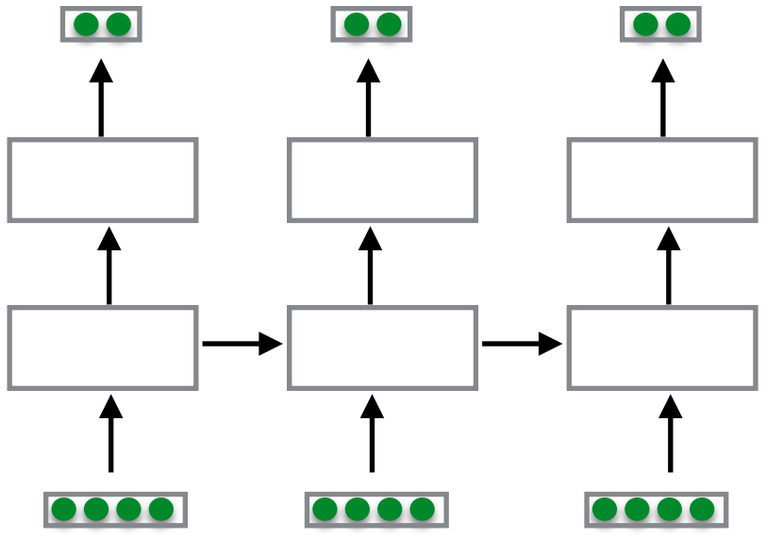
Auto Batching

what if the sequences are different lengths?



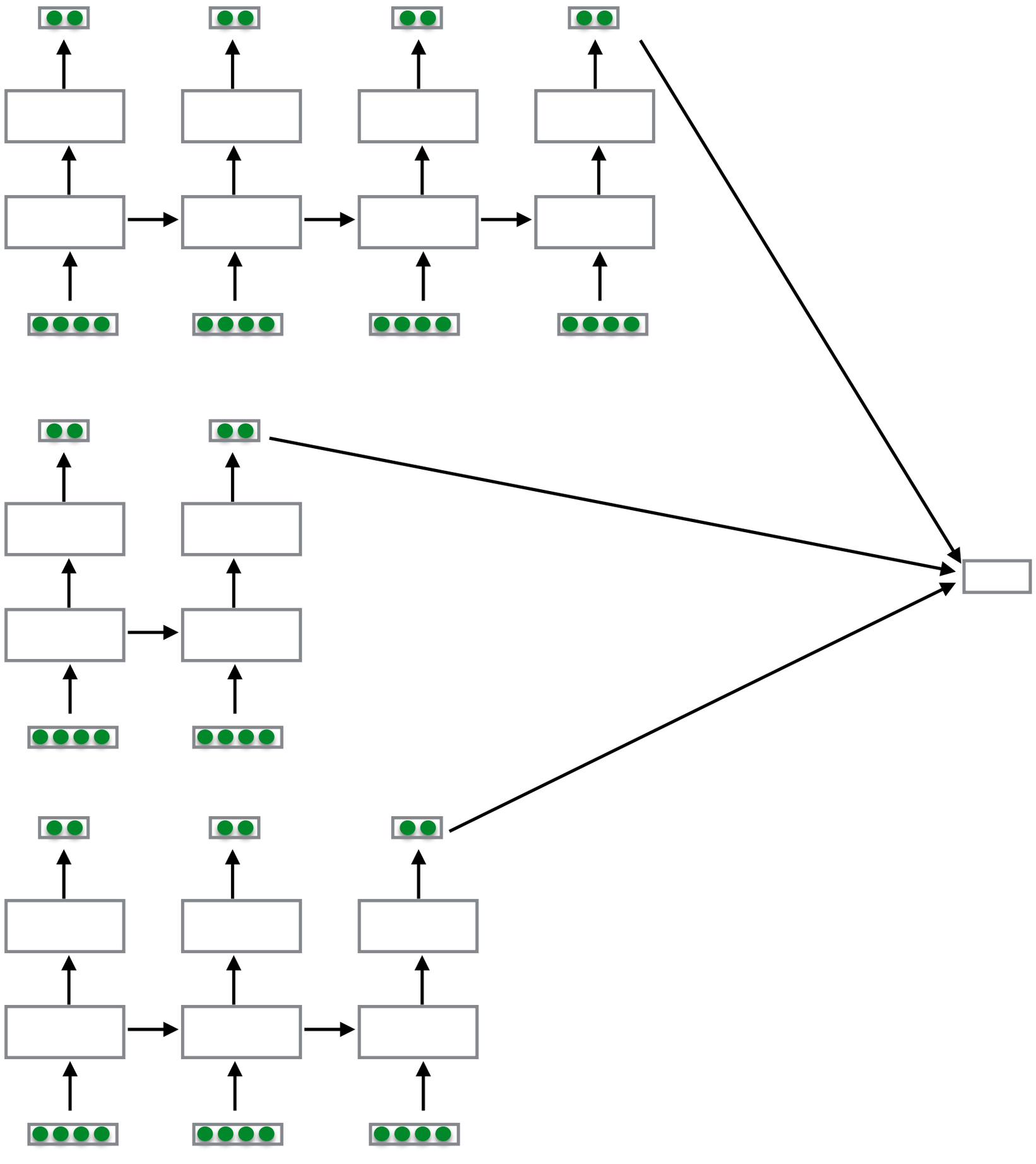




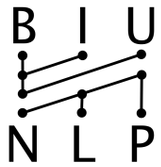


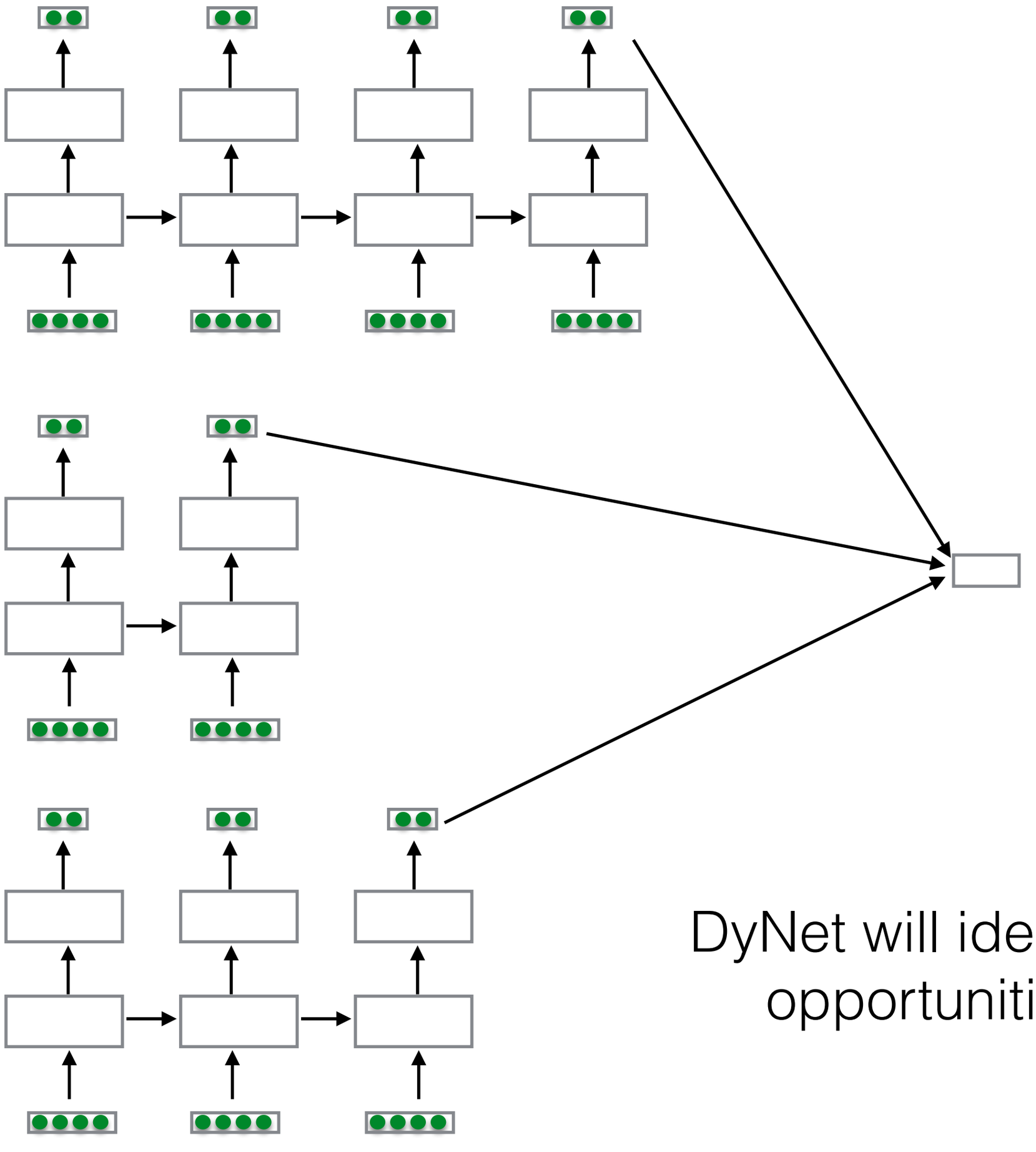
then create a separate network for each (easy)

B I U
N L P



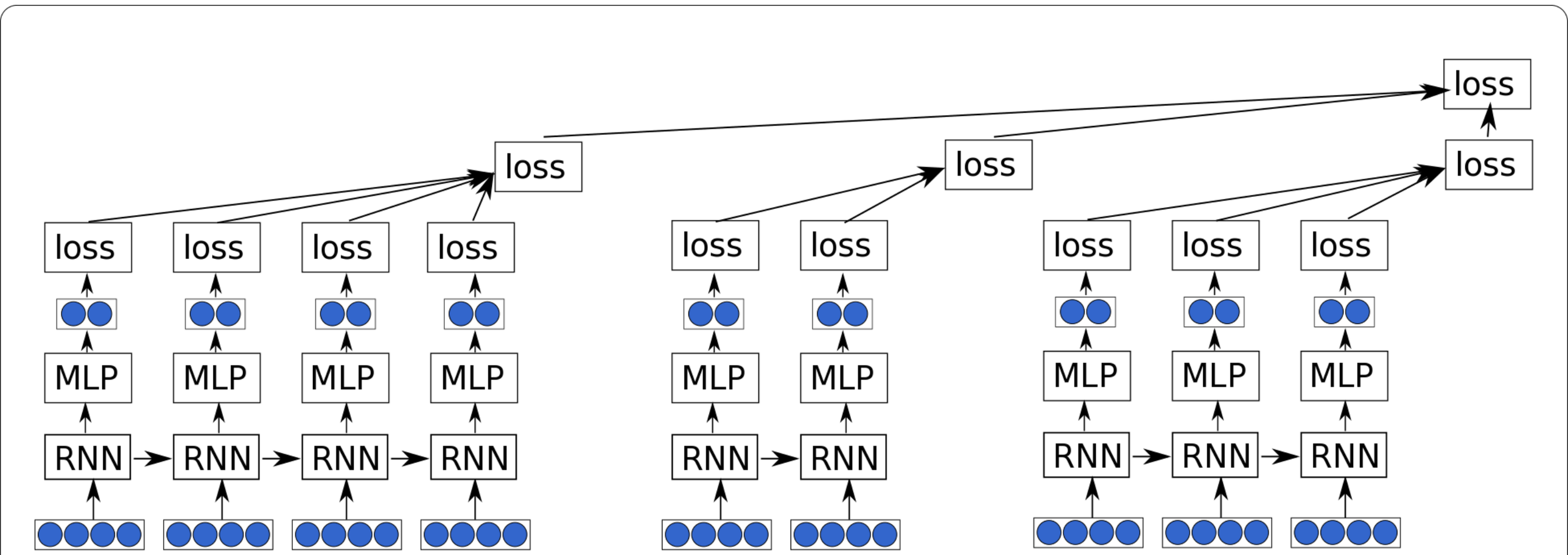
treat them as a single graph



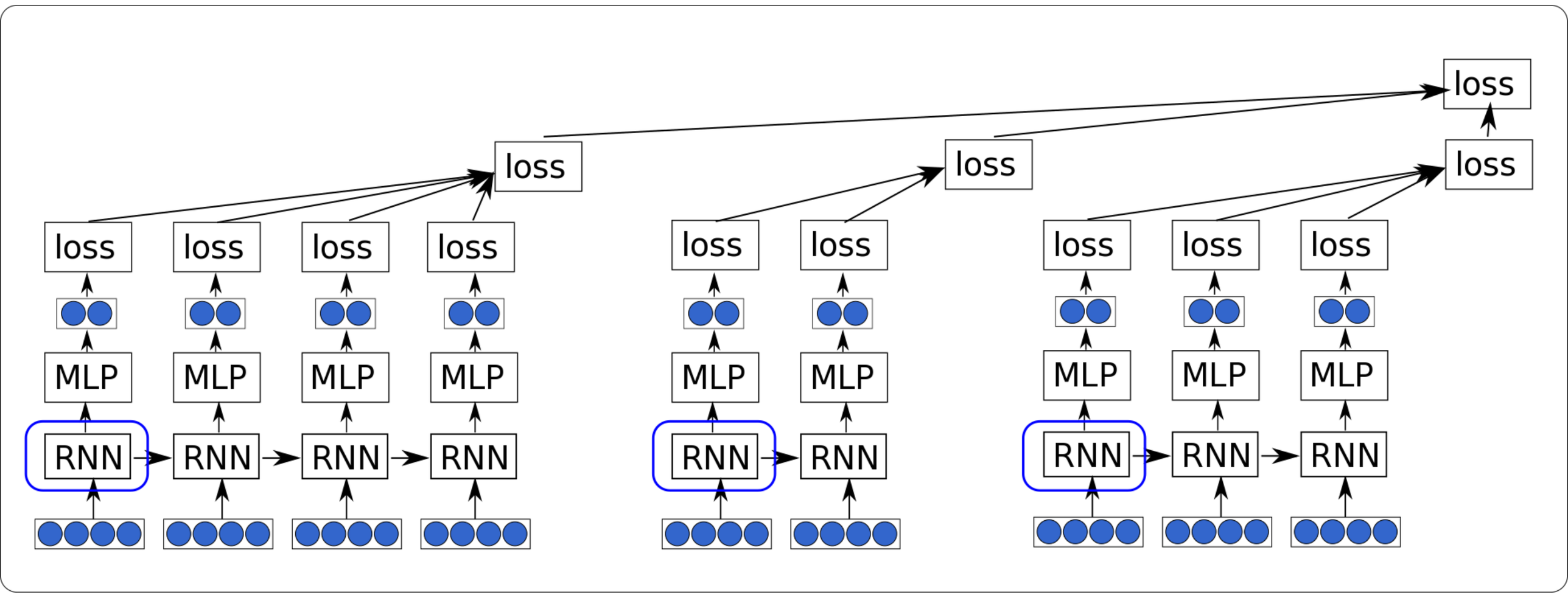


treat them as a single graph

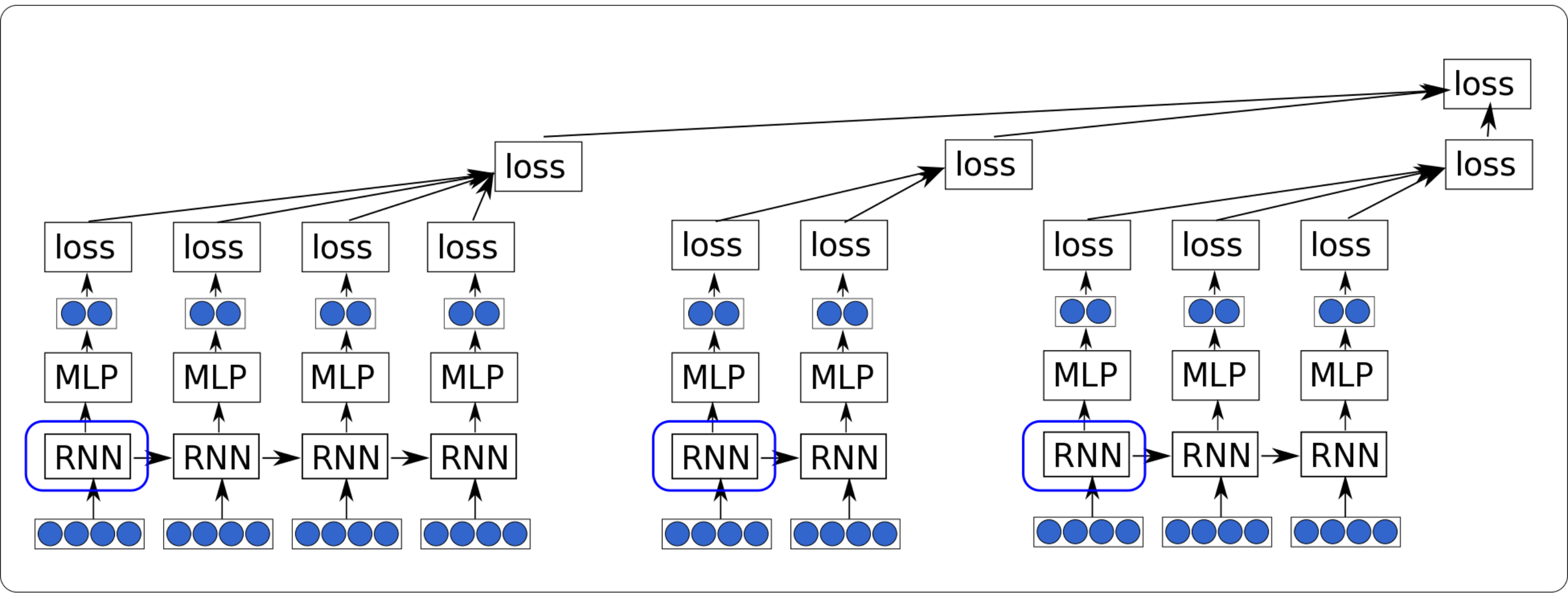
DyNet will identify batching opportunities for you.

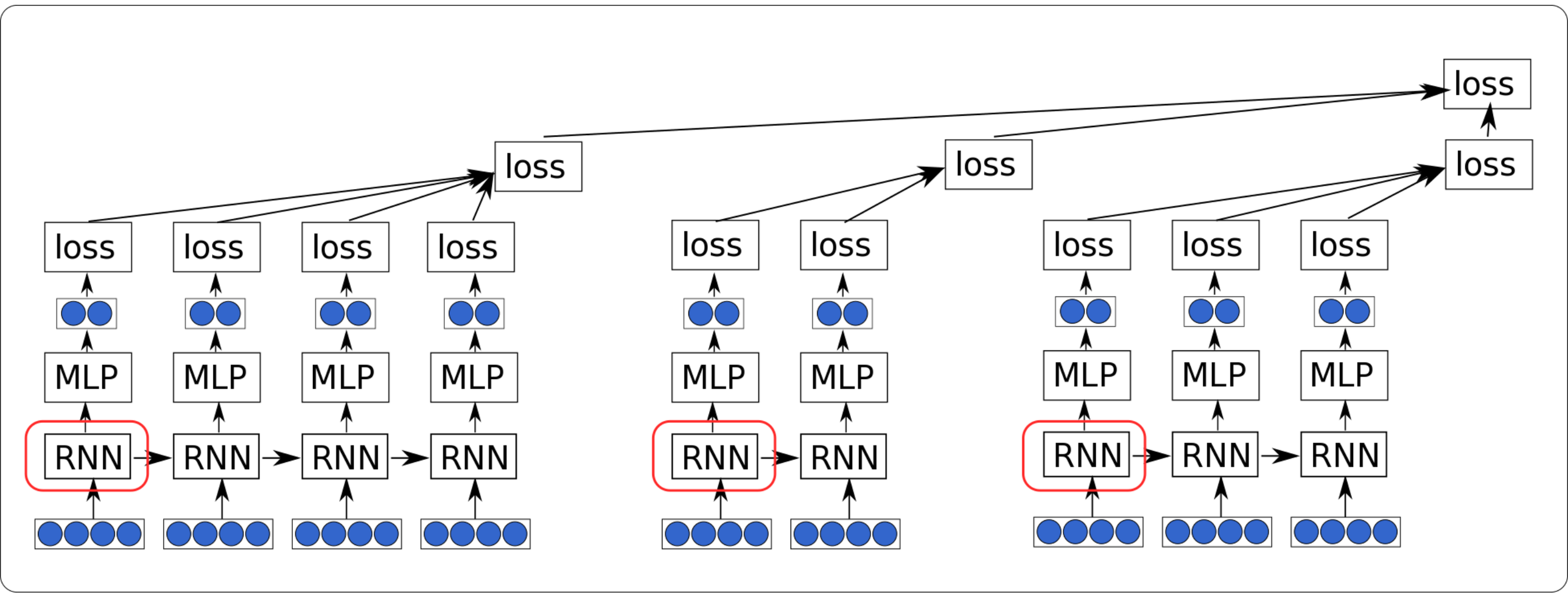

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

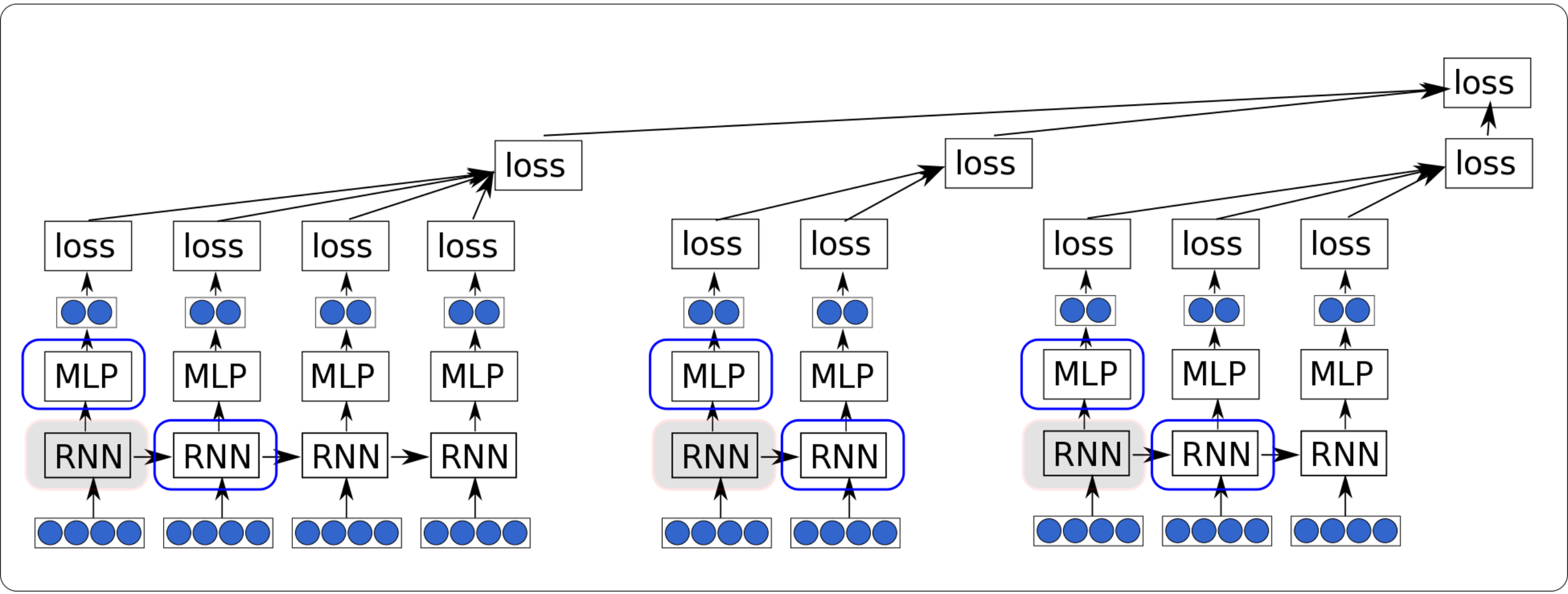
nodes in blue are ready
to be executed

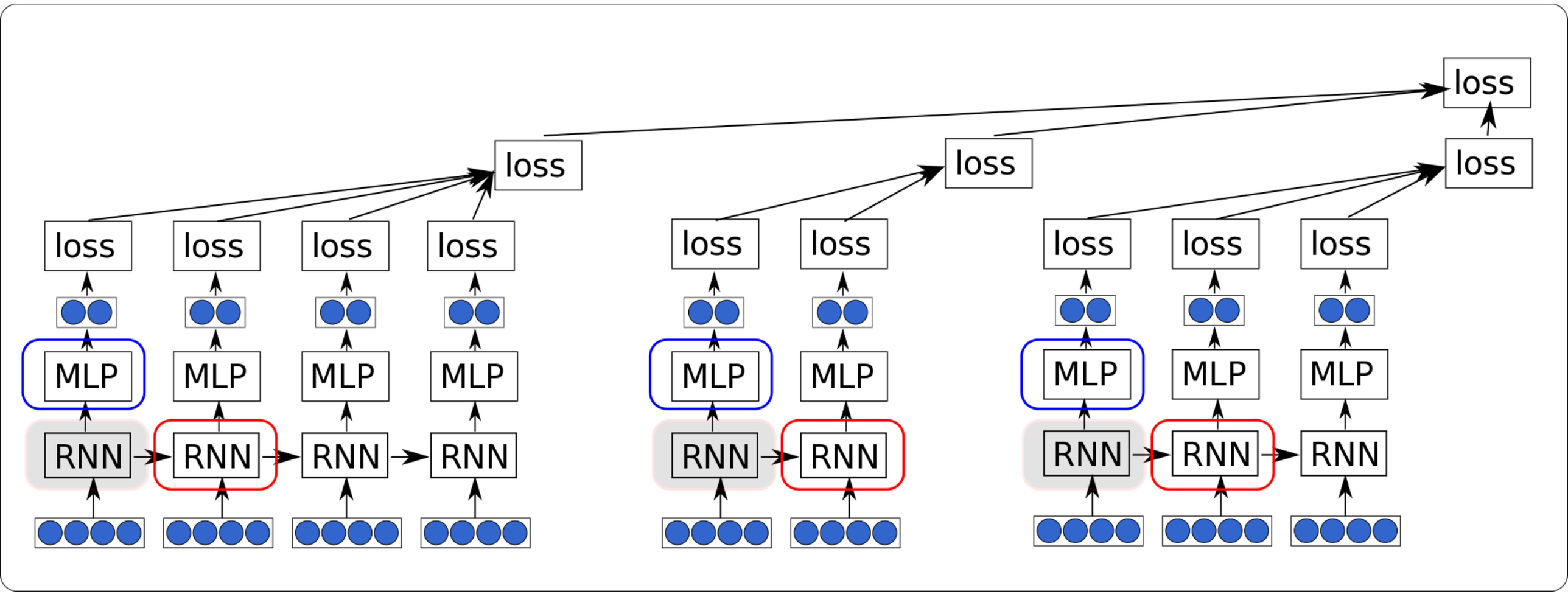

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

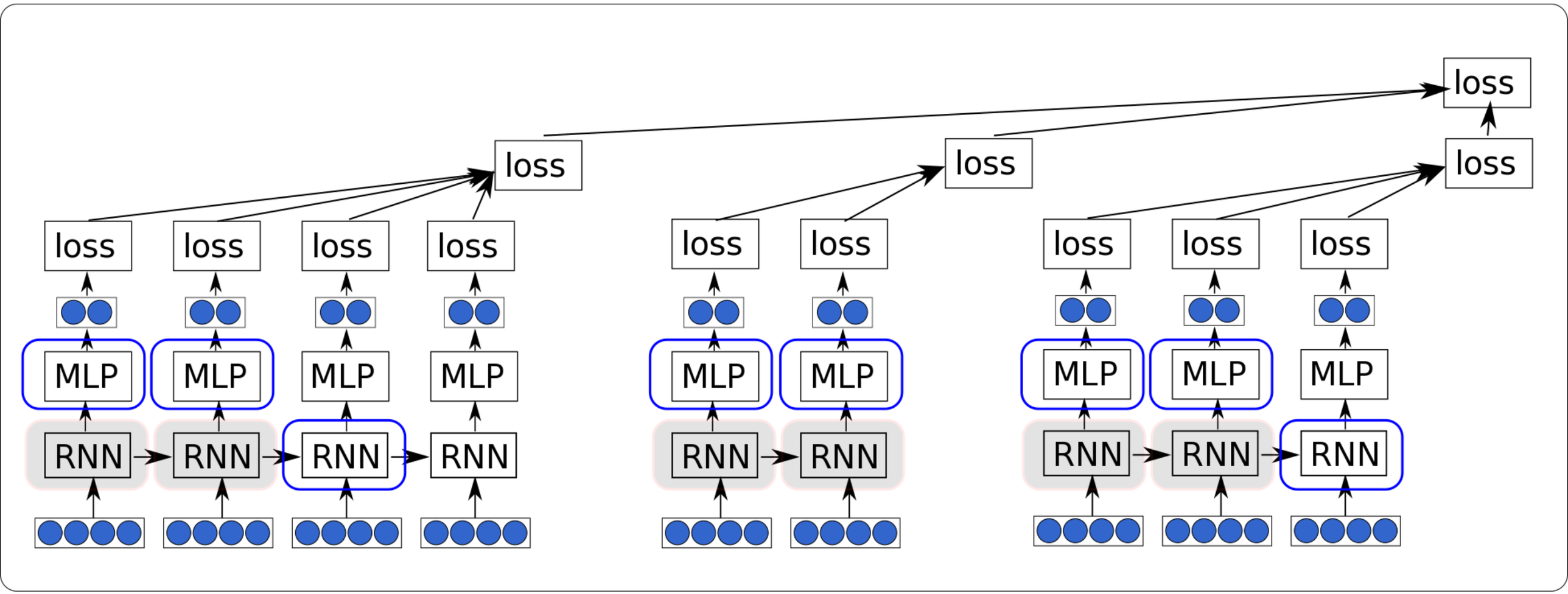
nodes in red will be executed
using batch operations

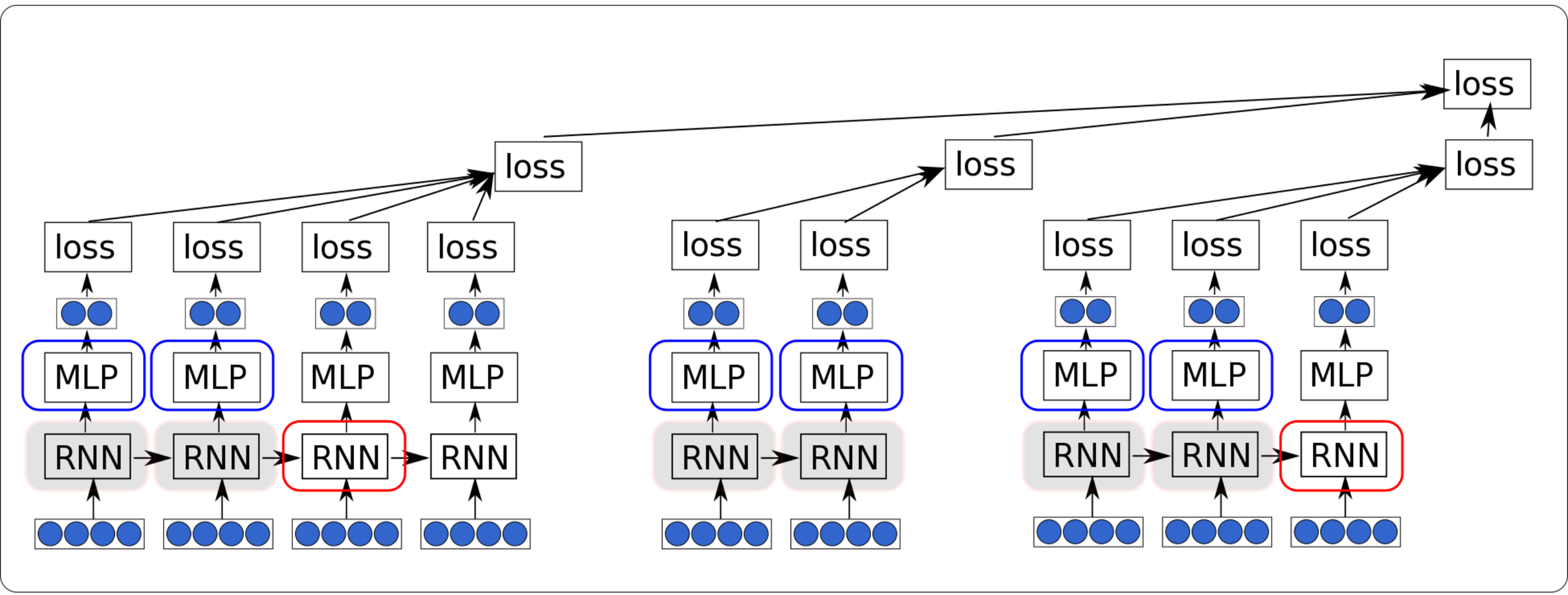

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

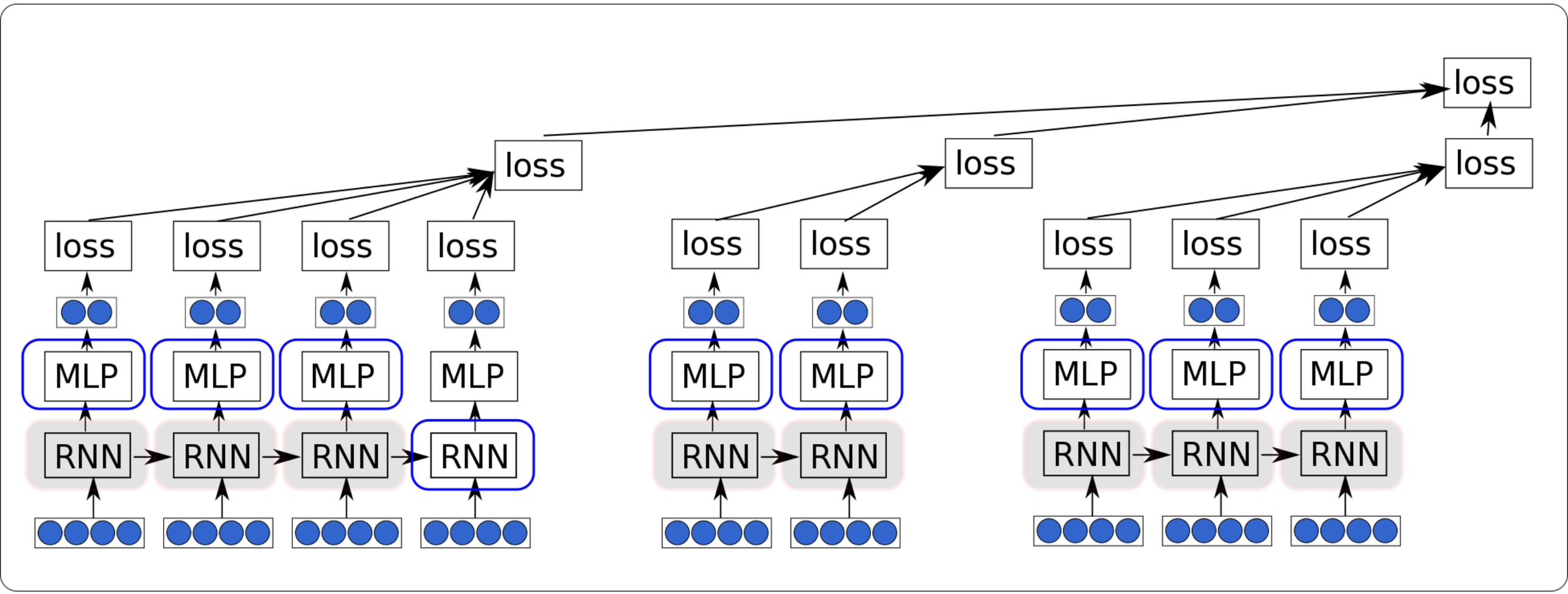

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

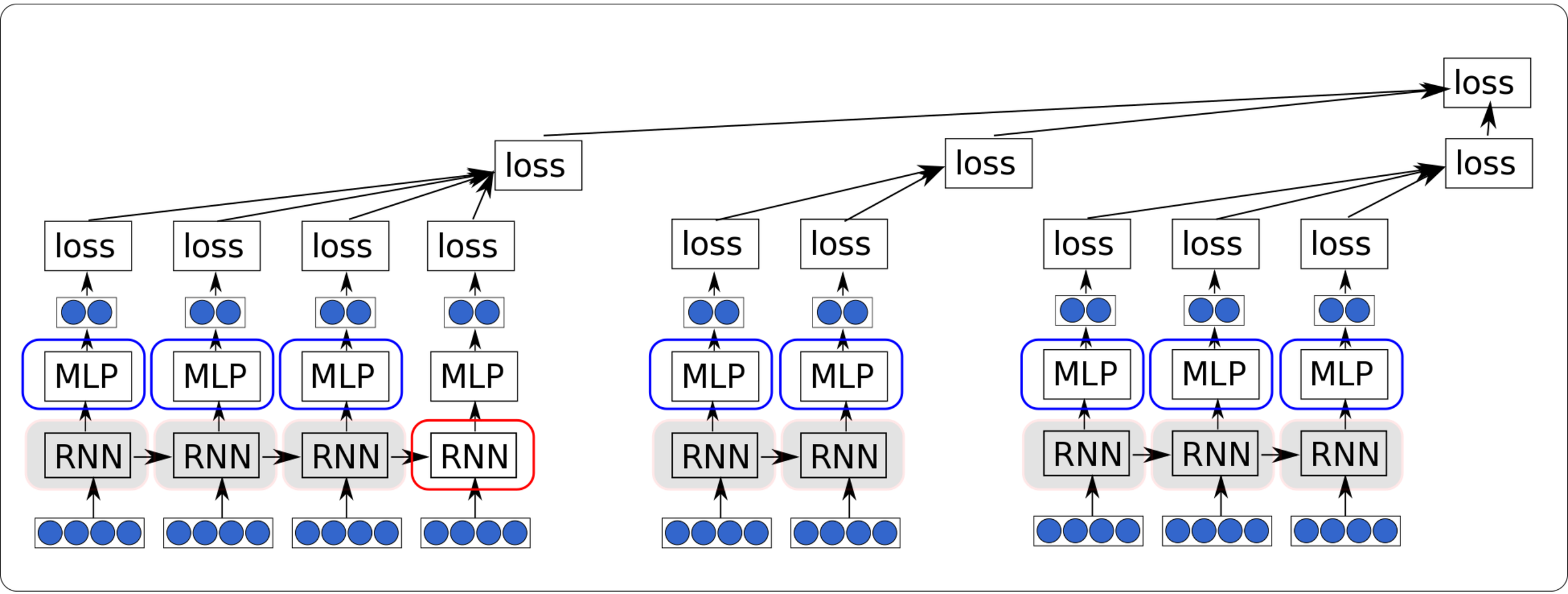

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

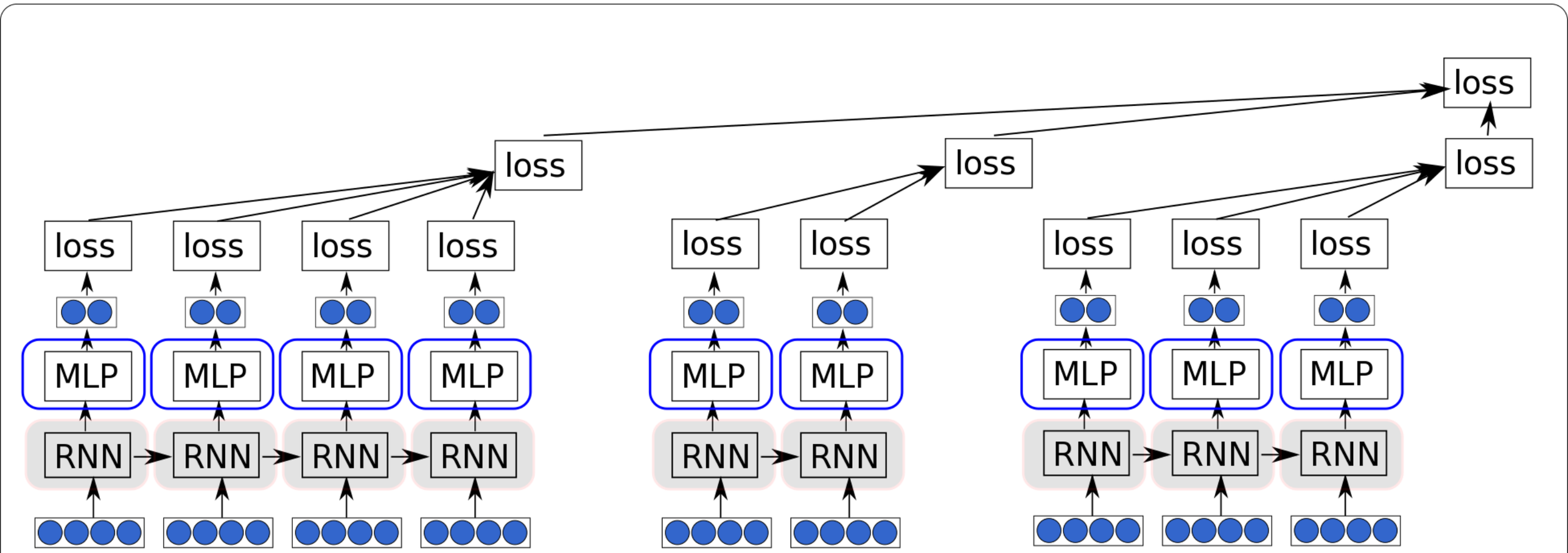

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

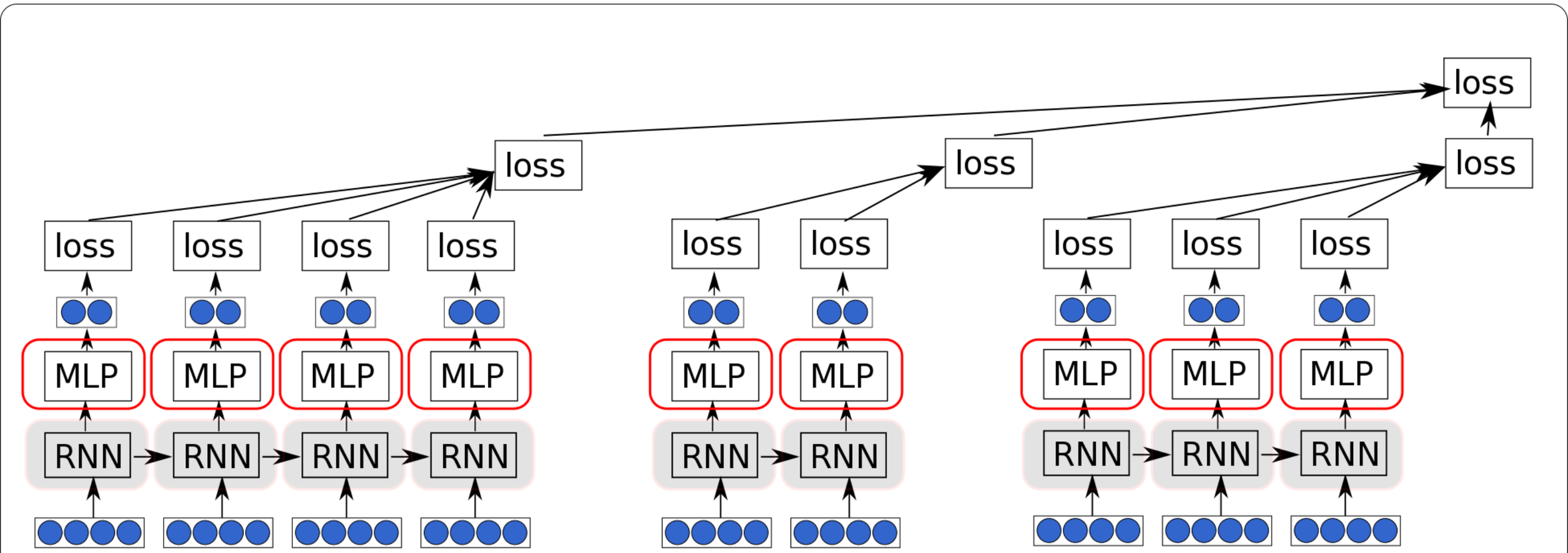

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

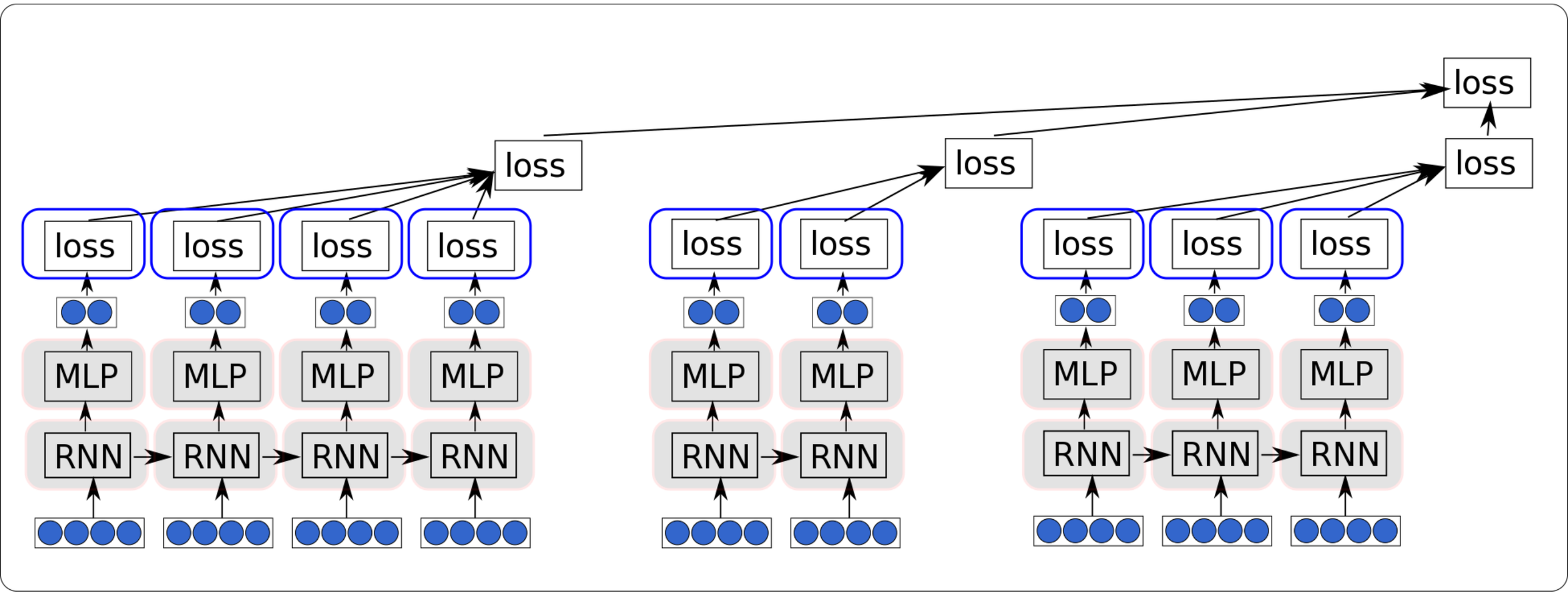

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

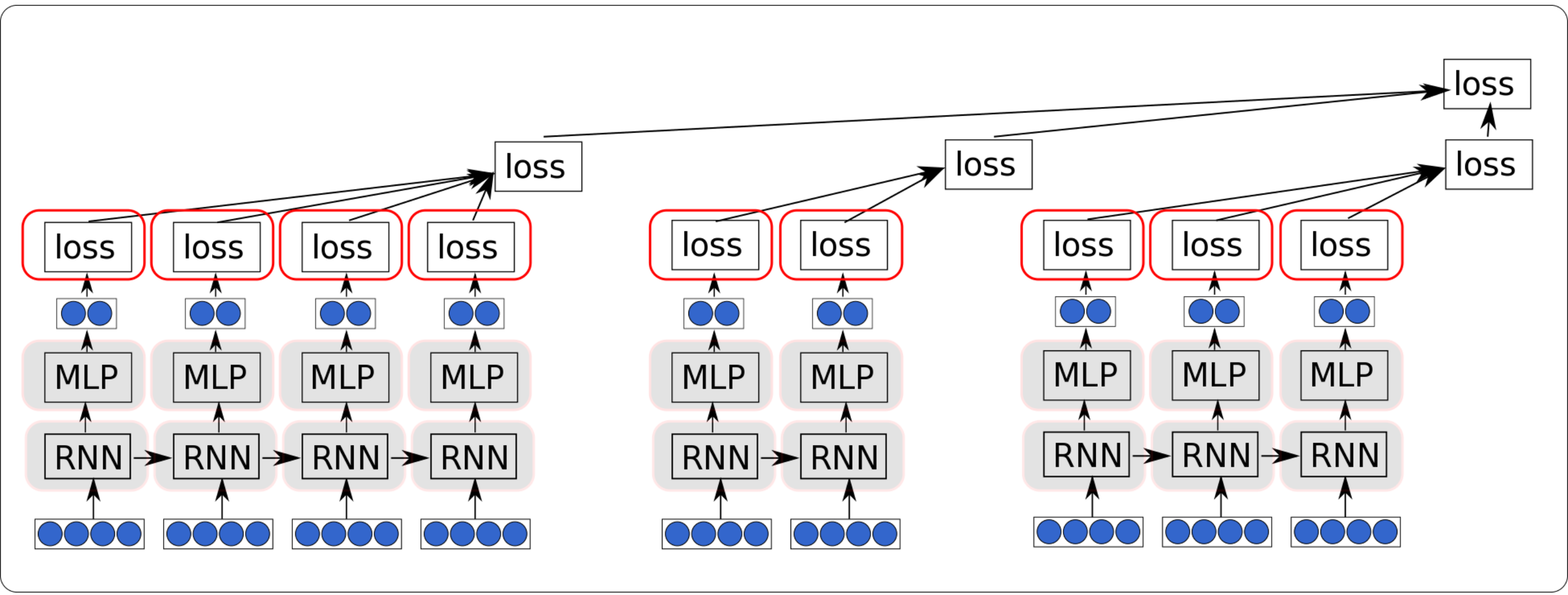

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

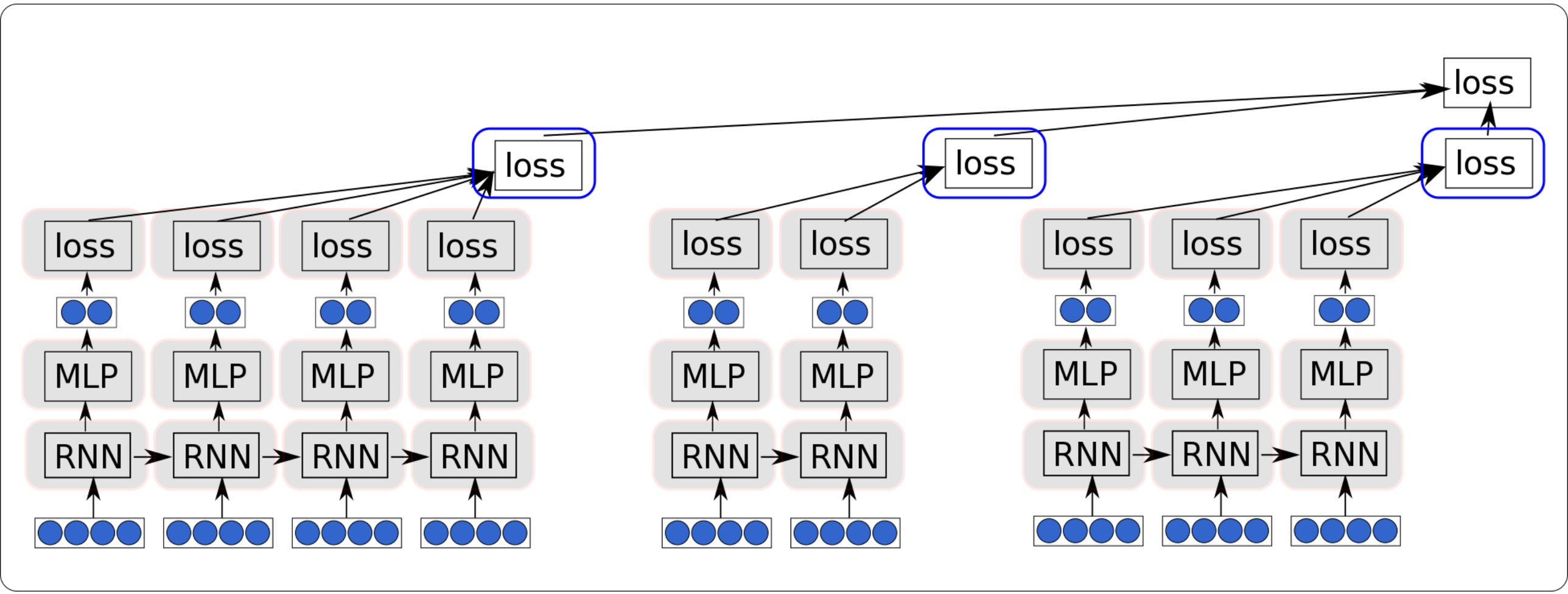

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

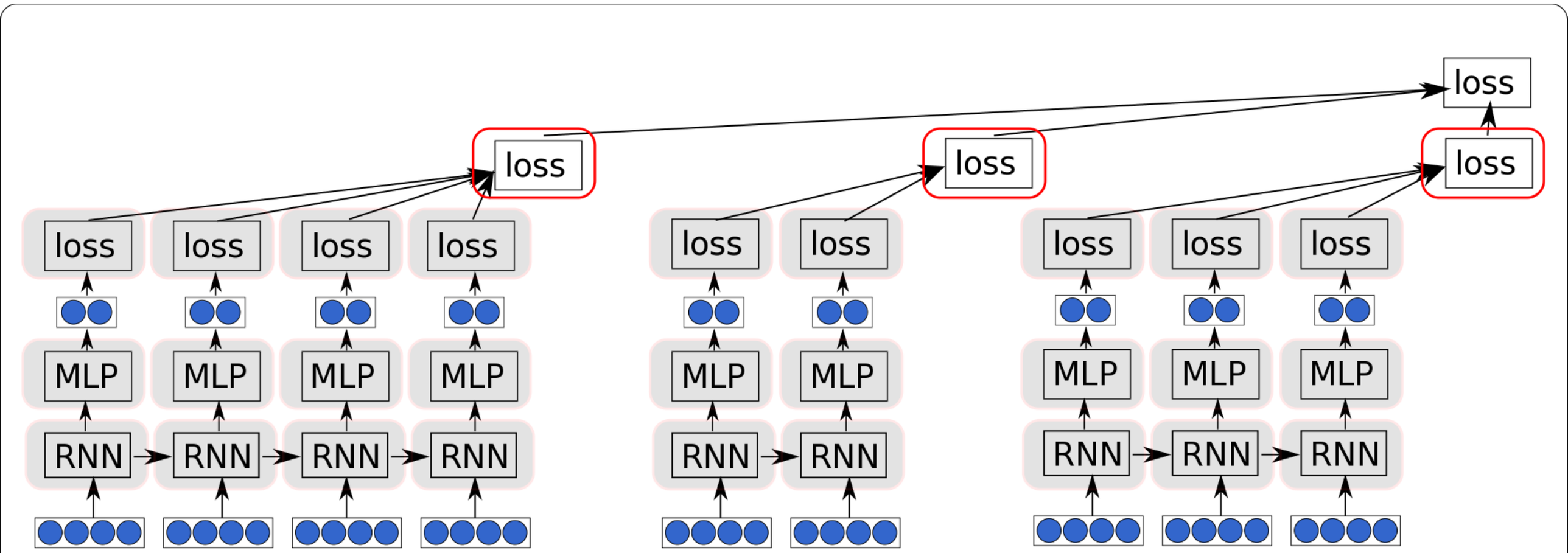

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

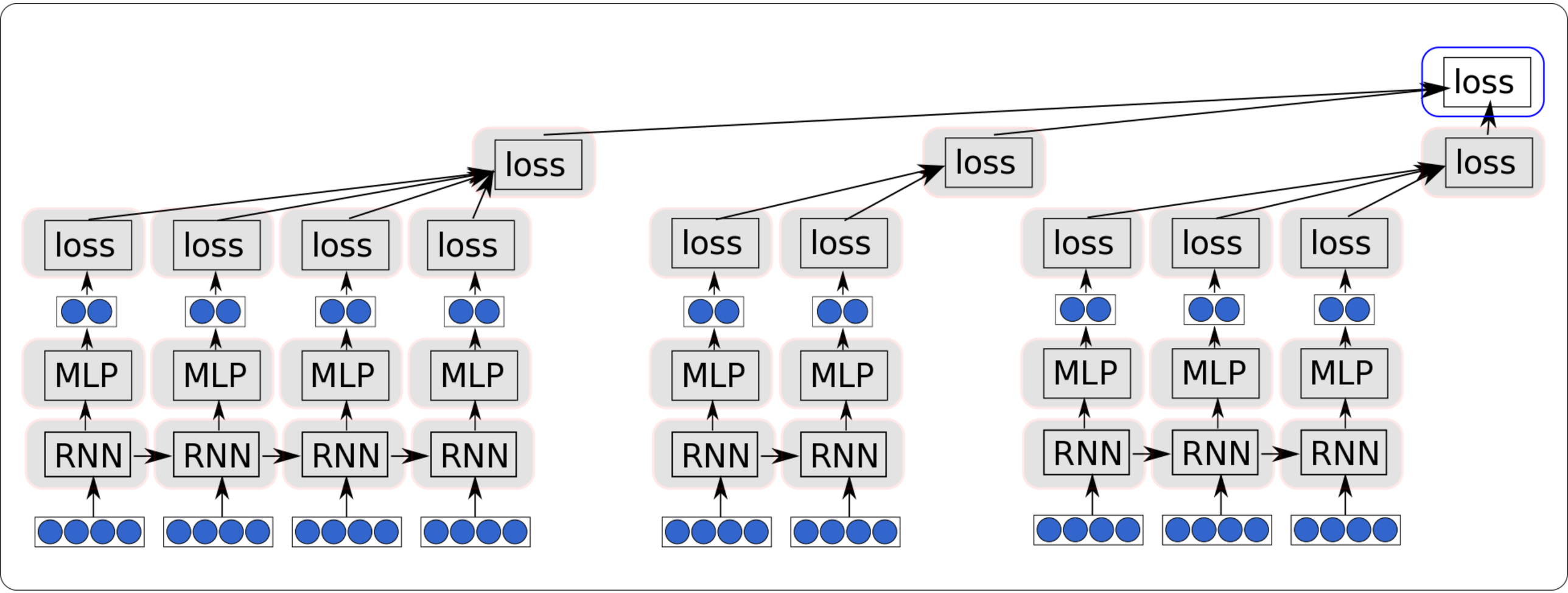

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

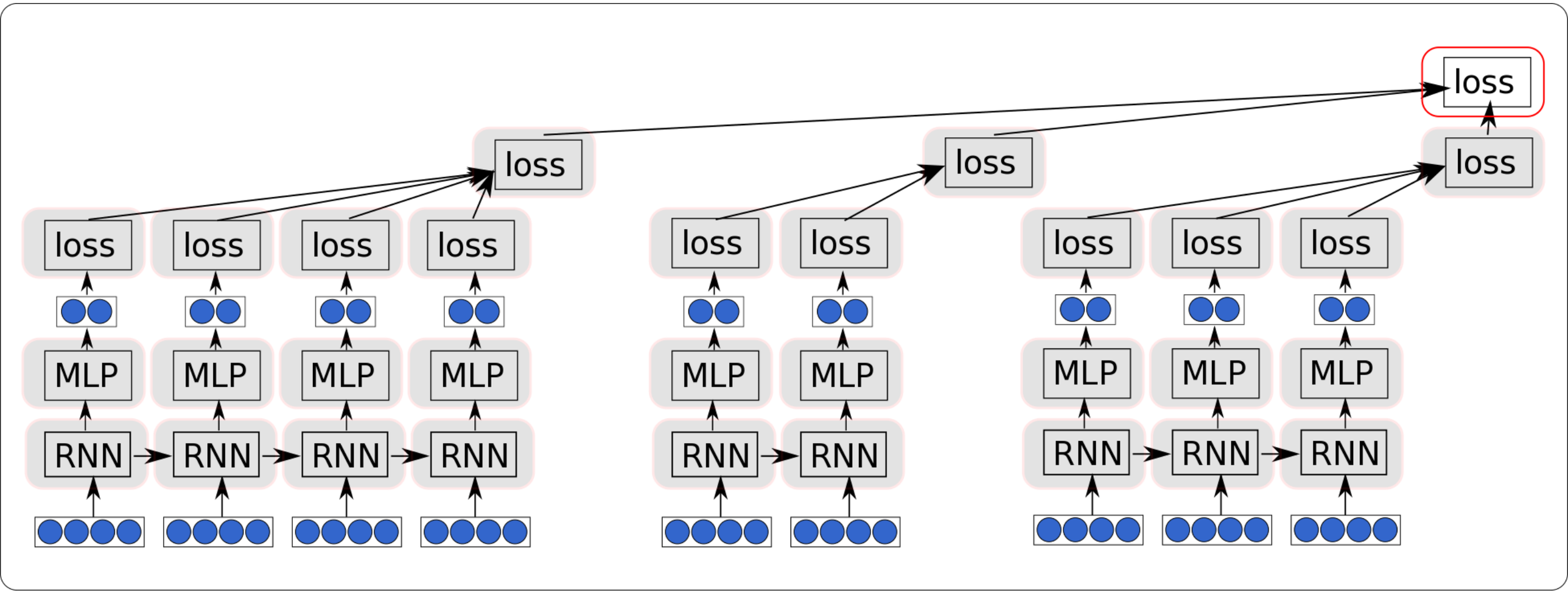

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

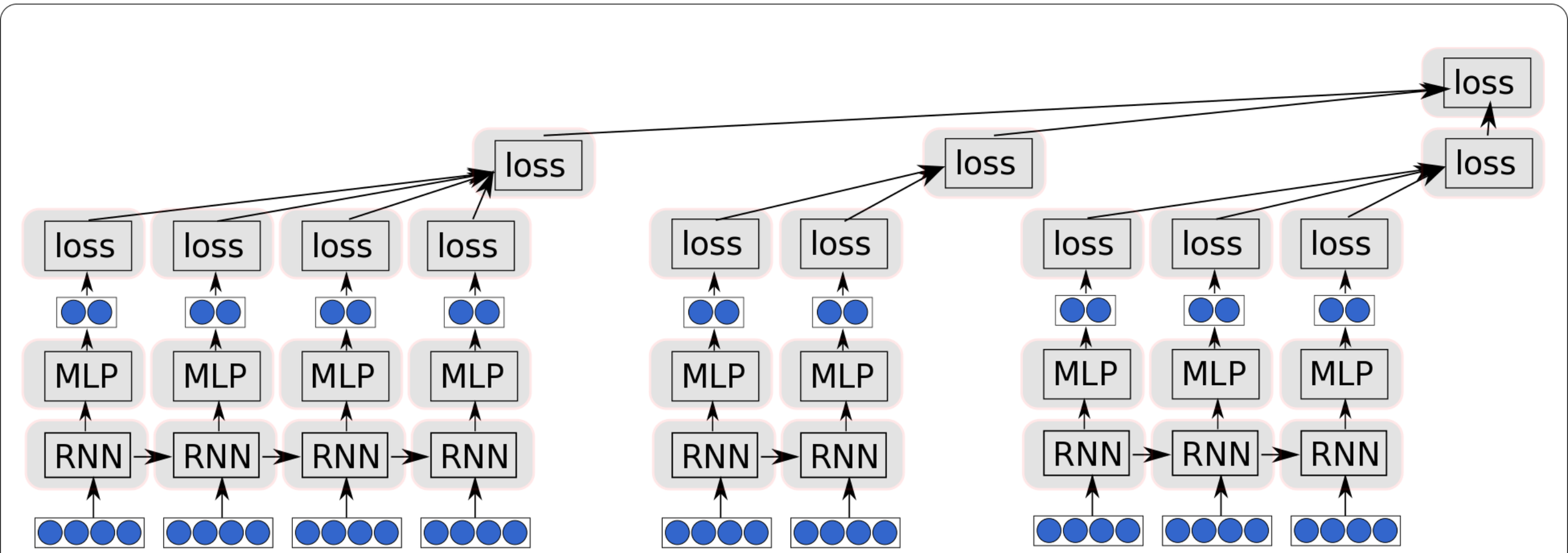

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN

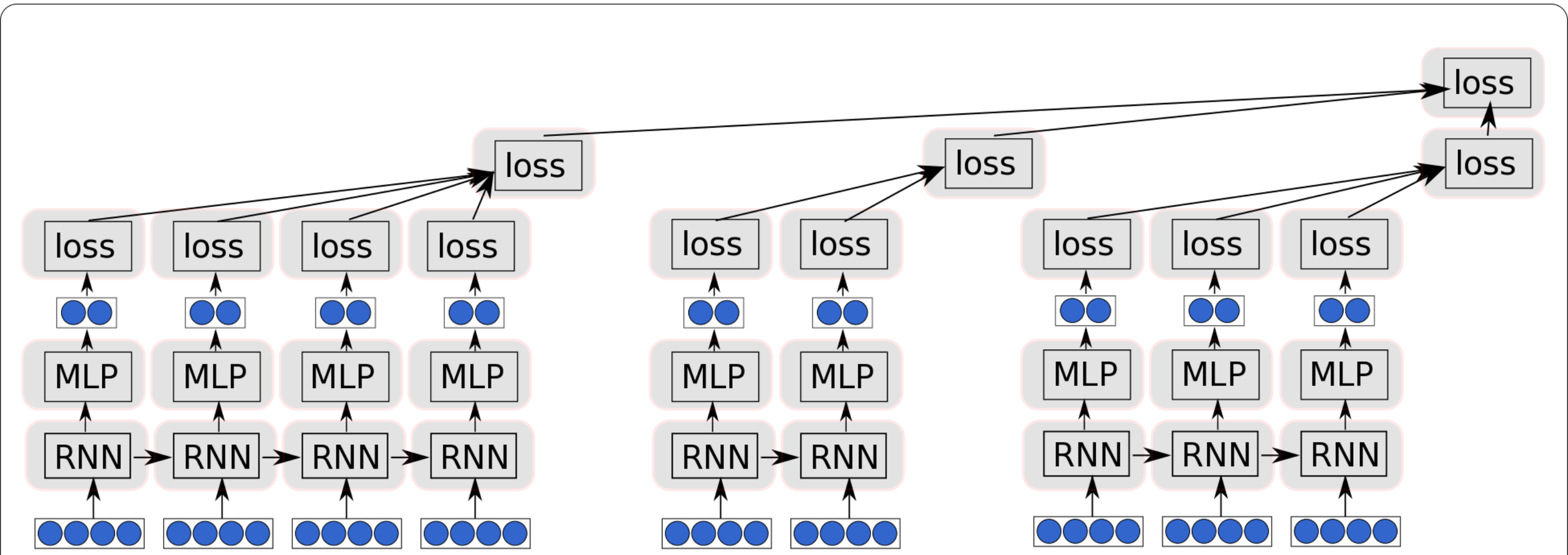

loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN


loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN


loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN


loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN


loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
loss
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
MLP
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN



note: batching operations, not inputs.

Recent Efficiency Benchmarks

For some tasks, GPU is much better.

For other, CPU still wins. (why, how?)

Efficiency Considerations when Implementing an LSTM

- What can be batched?

- What can be done in efficient math operations?

Efficiency Considerations when Implementing an LSTM

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) &= [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} &= \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} &= \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} &= \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} &= \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

Efficiency Considerations when Implementing an LSTM

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} =& \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} =& \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} =& \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} =& \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} =& \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} =& \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

in transduction mode, can batch all inputs (not states)

Efficiency Considerations when Implementing an LSTM

$$
\begin{aligned}
R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} =& \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} =& \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} =& \sigma(\mathbf{W^{xi}} \cdot \mathbf{x_j} + \mathbf{W^{hi}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{f} =& \sigma(\mathbf{W^{xf}} \cdot \mathbf{x_j} + \mathbf{W^{hf}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{o} =& \sigma(\mathbf{W^{xo}} \cdot \mathbf{x_j} + \mathbf{W^{ho}} \cdot \mathbf{h_{j-1}}) \\
\mathbf{g} =& \tanh(\mathbf{W^{xg}} \cdot \mathbf{x_j} + \mathbf{W^{hg}} \cdot \mathbf{h_{j-1}})
\end{aligned}
$$

all gates computations can be done in single mat-mat op.