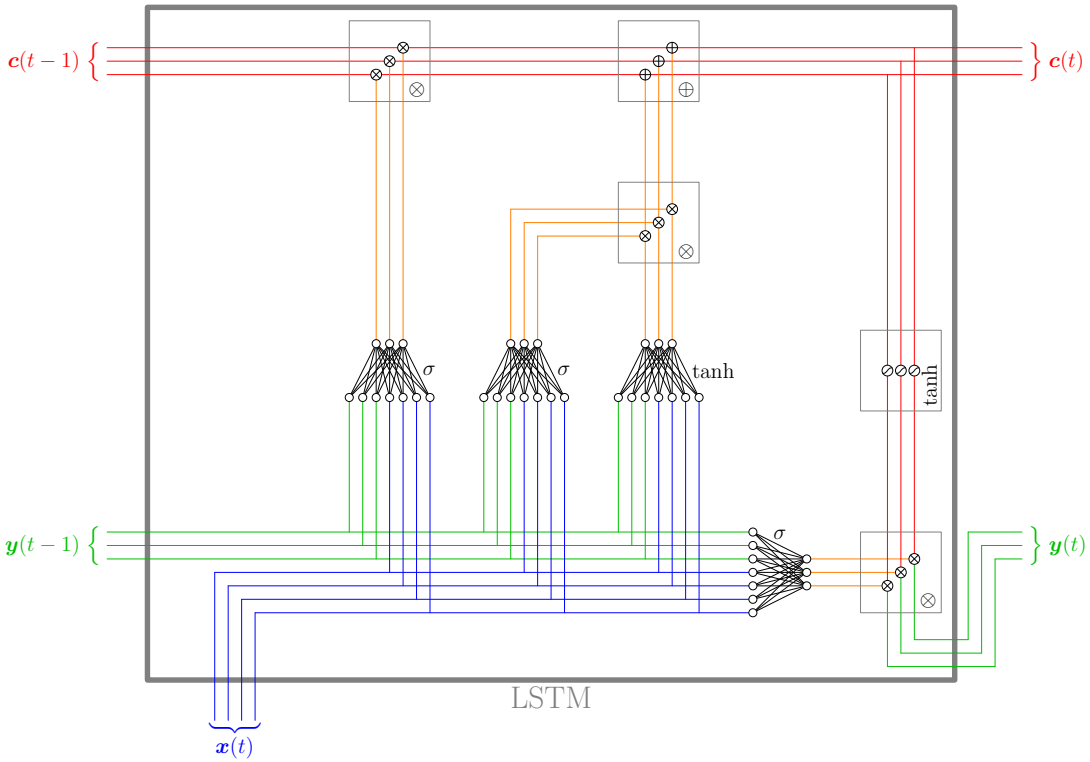
Advanced Machine Learning

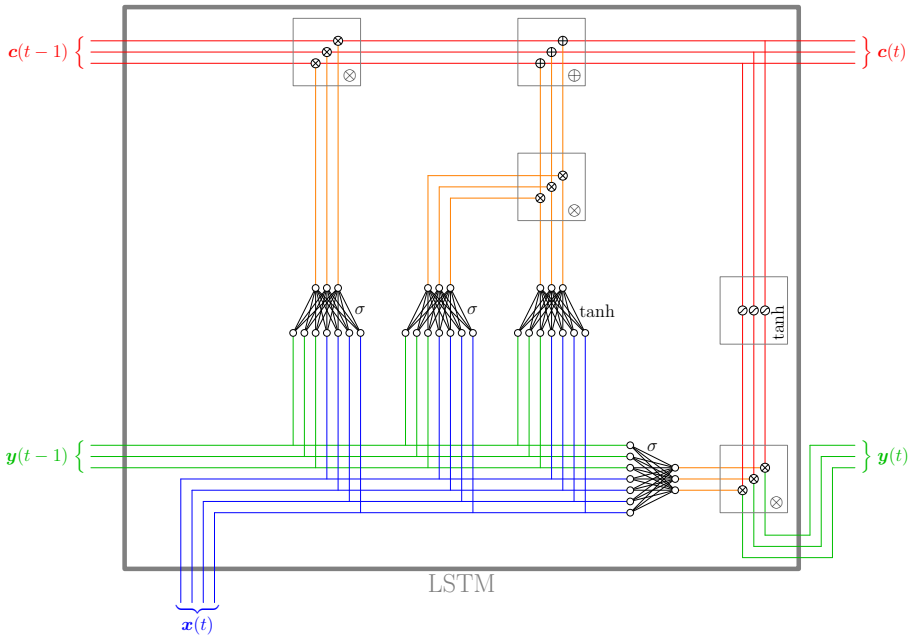
LSTM



Recurrent Neural Networks, Vanishing Gradients, LSTM

Outline

1. **More on CNNs**
2. Recurrent Networks
3. LSTM
4. Word Embedding
5. Applications



A Bit More on CNNs

- This lecture covers recurrent neural networks (RNNs)
- But, before we go there let we finish off with CNNs

Transfer Learning

- Although unsupervised pre-training of CNNs has never taken off, people use CNNs trained on other data sets (mostly imageNet)
- They will often decapitate a CNN trained on imageNet just before the dense layer
- The weights are publicly available for many classic networks (VGG, resNet, GoogLeNet)
- Often used and quite different problems (imageNet features seem to be generally useful)
- This is known as **transfer learning**

Transfer Learning

- Although unsupervised pre-training of CNNs has never taken off, people use CNNs trained on other data sets (mostly imageNet)
- They will often decapitate a CNN trained on imageNet just before the dense layer
- The weights are publicly available for many classic networks (VGG, resNet, GoogLeNet)
- Often used and quite different problems (imageNet features seem to be generally useful)
- This is known as **transfer learning**

Transfer Learning

- Although unsupervised pre-training of CNNs has never taken off, people use CNNs trained on other data sets (mostly imageNet)
- They will often decapitate a CNN trained on imageNet just before the dense layer
- The weights are publicly available for many classic networks (VGG, resNet, GoogLeNet)
- Often used and quite different problems (imageNet features seem to be generally useful)
- This is known as **transfer learning**

Transfer Learning

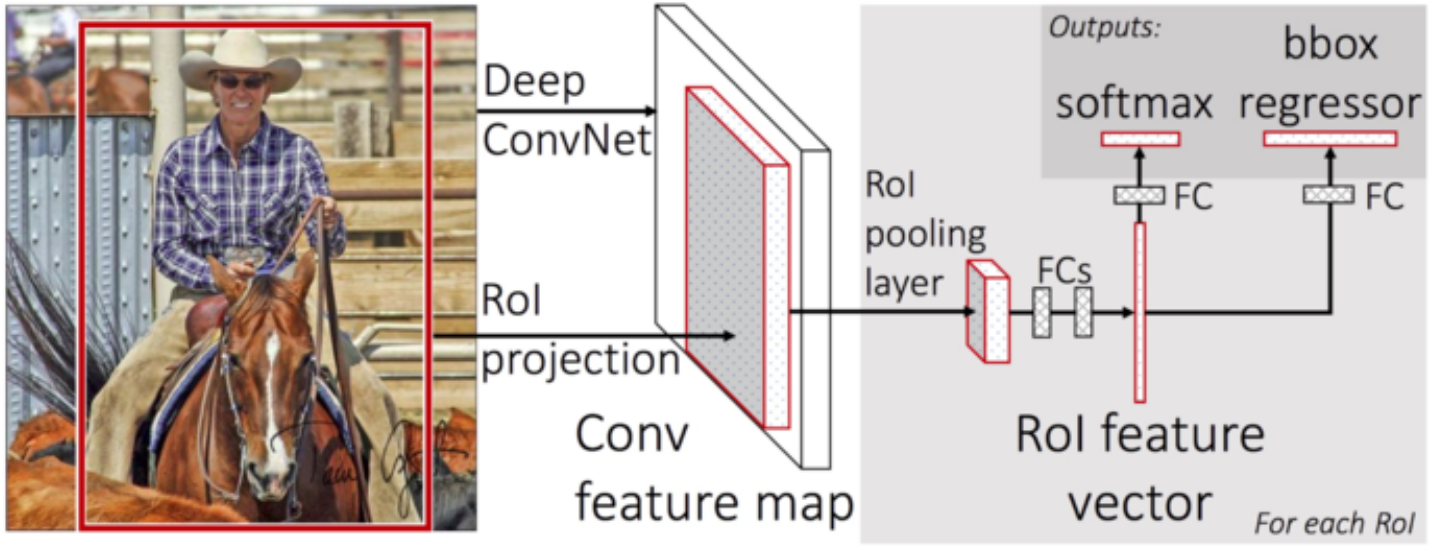
- Although unsupervised pre-training of CNNs has never taken off, people use CNNs trained on other data sets (mostly imageNet)
- They will often decapitate a CNN trained on imageNet just before the dense layer
- The weights are publicly available for many classic networks (VGG, resNet, GoogLeNet)
- Often used and quite different problems (imageNet features seem to be generally useful)
- This is known as **transfer learning**

Transfer Learning

- Although unsupervised pre-training of CNNs has never taken off, people use CNNs trained on other data sets (mostly imageNet)
- They will often decapitate a CNN trained on imageNet just before the dense layer
- The weights are publicly available for many classic networks (VGG, resNet, GoogLeNet)
- Often used and quite different problems (imageNet features seem to be generally useful)
- This is known as **transfer learning**

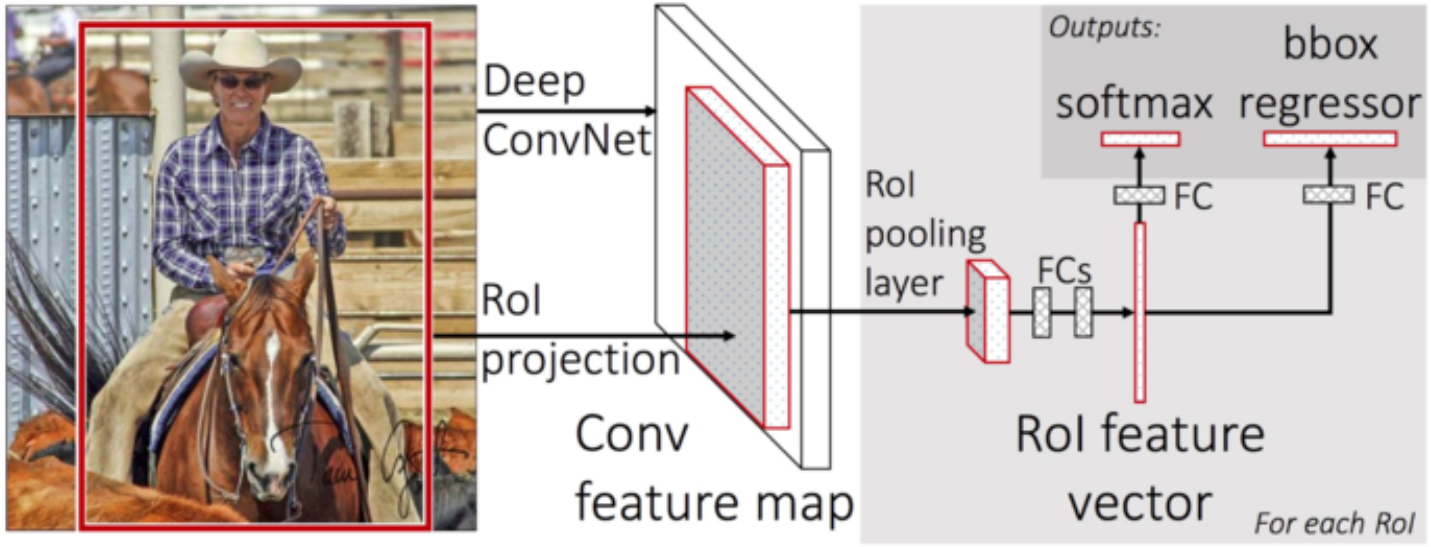
Regional CNN

- CNNs are also used for object location—putting a bounding box around subjects
- Use different data set with multiple objects—e.g. COCO
- Some of the best known networks for this are R-CNN, Fast R-CNN, Faster R-CNN and Masked R-CNN



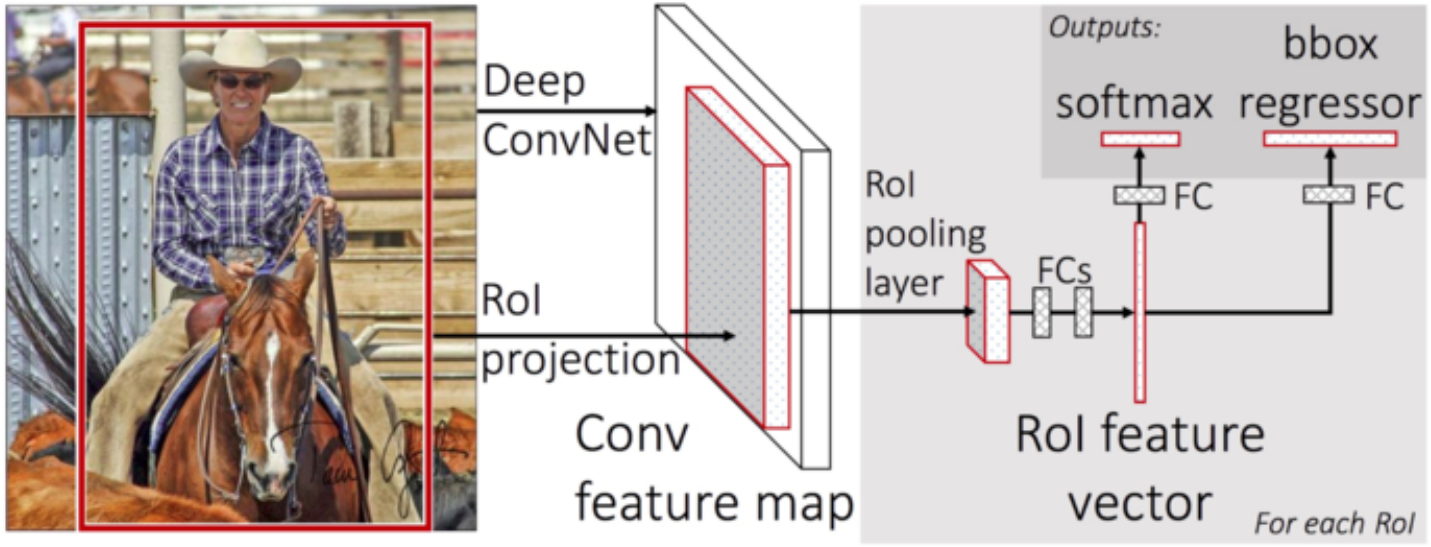
Regional CNN

- CNNs are also used for object location—putting a bounding box around subjects
- Use different data set with multiple objects—e.g. COCO
- Some of the best known networks for this are R-CNN, Fast R-CNN, Faster R-CNN and Masked R-CNN



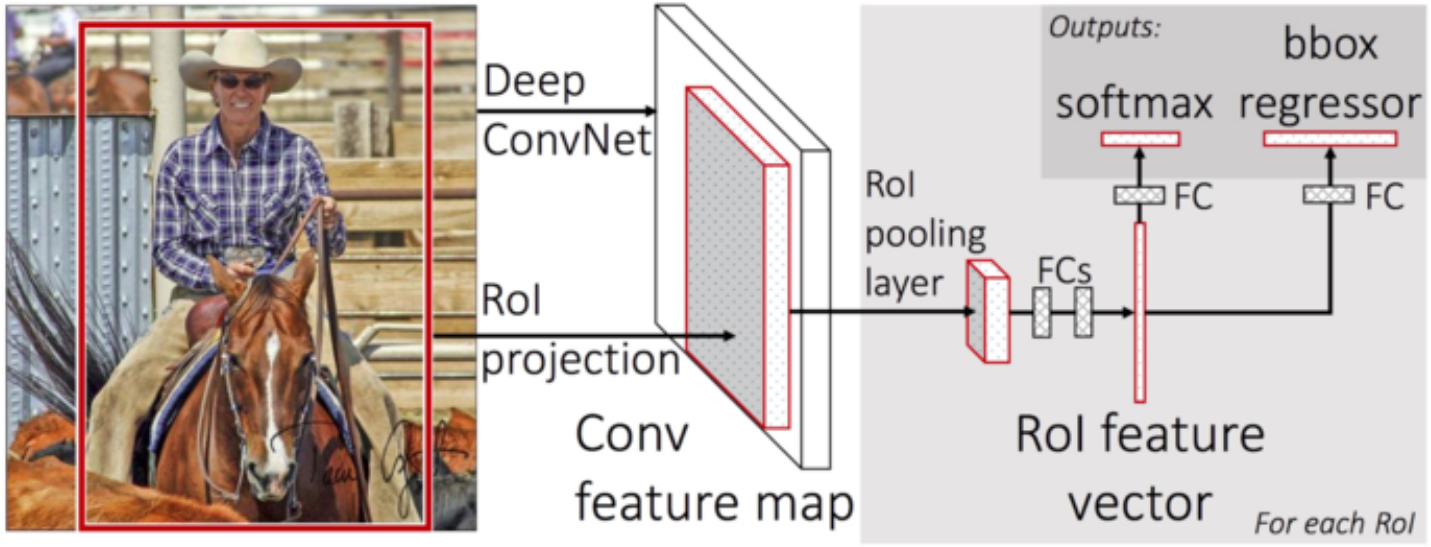
Regional CNN

- CNNs are also used for object location—putting a bounding box around subjects
- Use different data set with multiple objects—e.g. COCO
- Some of the best known networks for this are R-CNN, Fast R-CNN, Faster R-CNN and Masked R-CNN

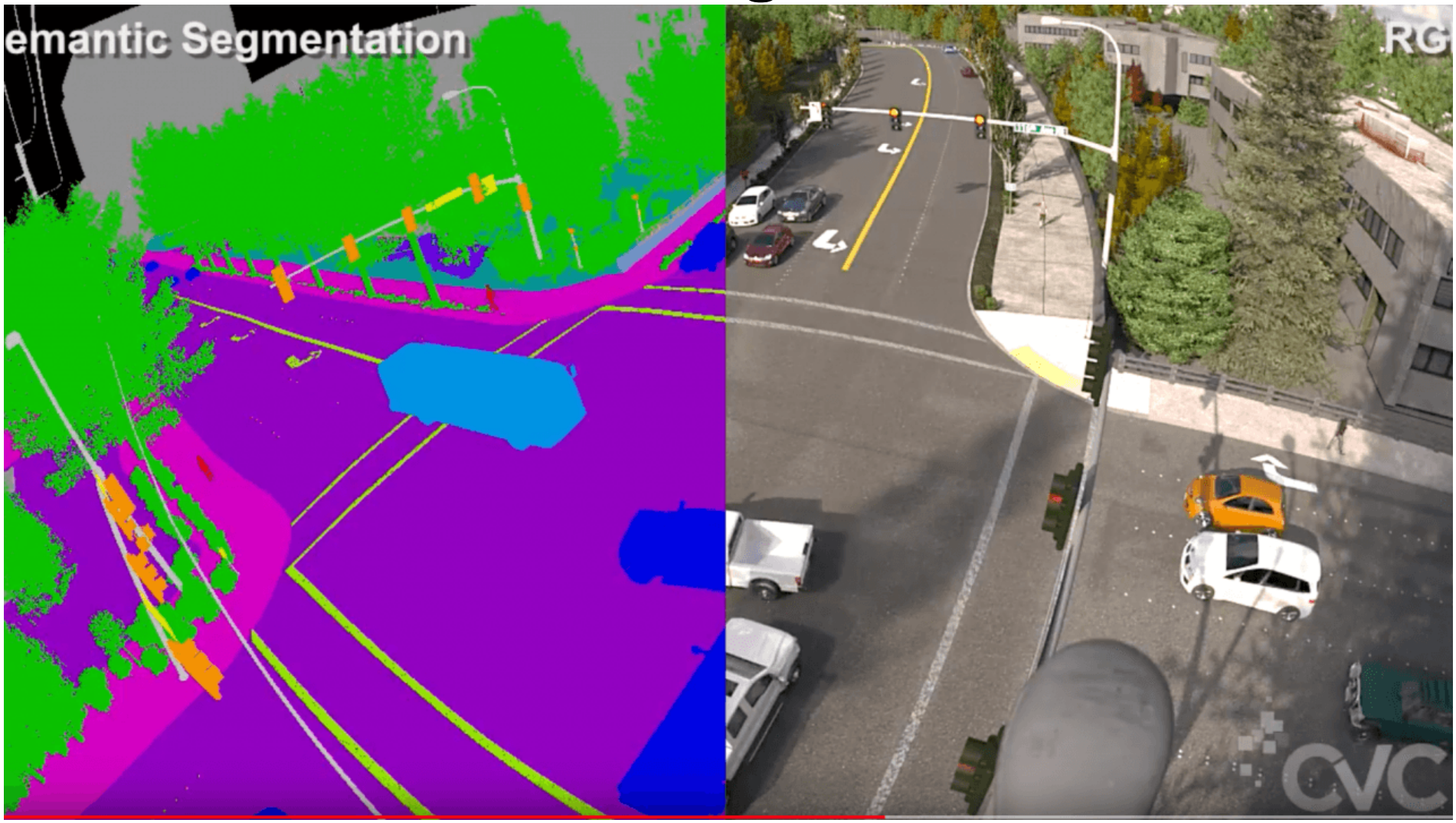


Regional CNN

- CNNs are also used for object location—putting a bounding box around subjects
- Use different data set with multiple objects—e.g. COCO
- Some of the best known networks for this are R-CNN, Fast R-CNN, Faster R-CNN and Masked R-CNN

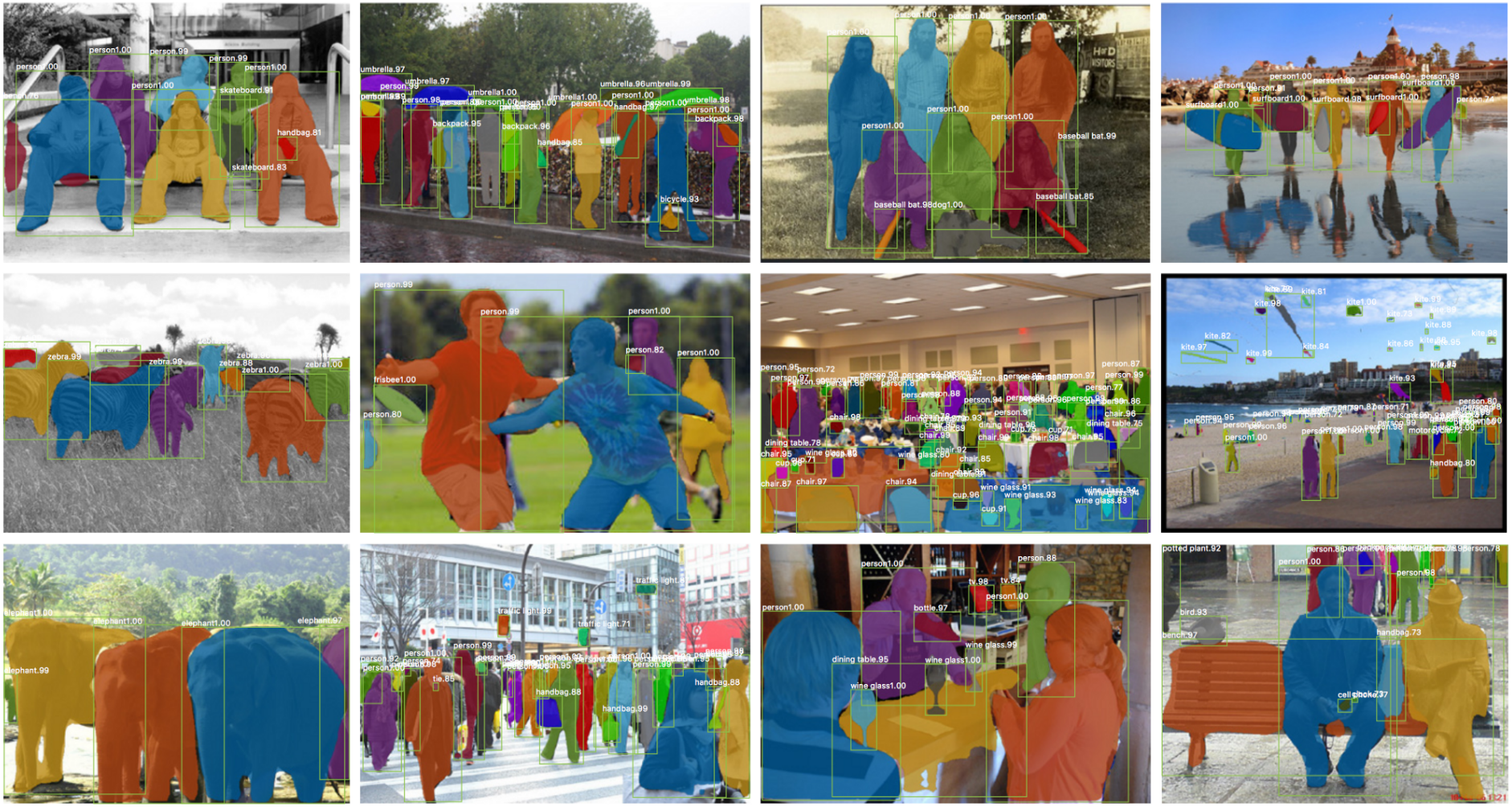


Semantic Segmentation



- Semantic segmentation is a pixel-level labelling of images

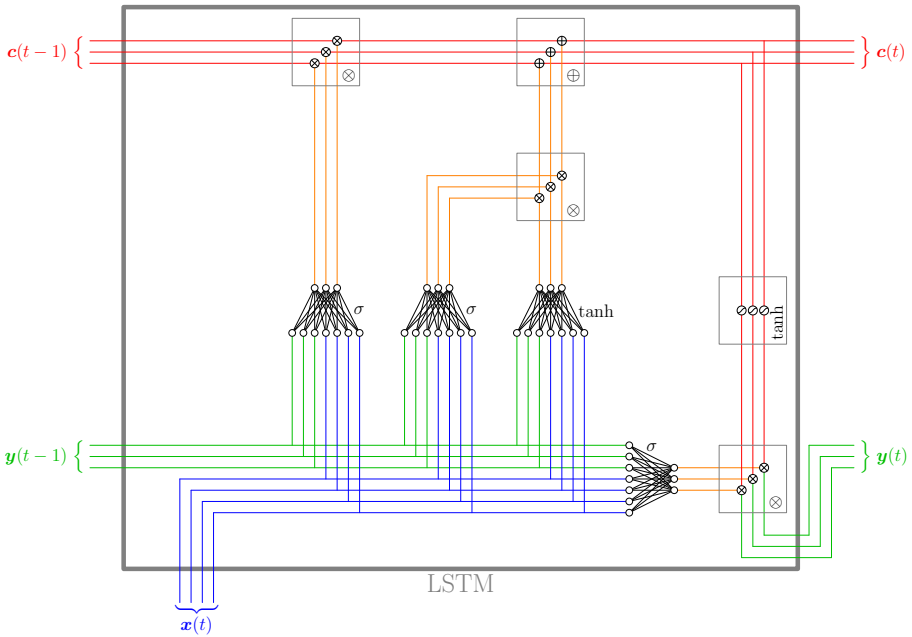
Semantic Segmentation



- Networks for achieving this include U-Net and Masked R-CNN

Outline

1. More on CNNs
2. **Recurrent Networks**
3. LSTM
4. Word Embedding
5. Applications



Recurrent Network

- Feed-forward networks are rather limited for some applications
- They are not particularly well matched for interpreting arbitrary length time series
- To interpret a sentence, or to predict tomorrows weather it is necessary to remember what happened in the past
- To facilitate this we would like to add a feedback loop delayed in time

Recurrent Network

- Feed-forward networks are rather limited for some applications
- They are not particularly well matched for interpreting arbitrary length time series
- To interpret a sentence, or to predict tomorrows weather it is necessary to remember what happened in the past
- To facilitate this we would like to add a feedback loop delayed in time

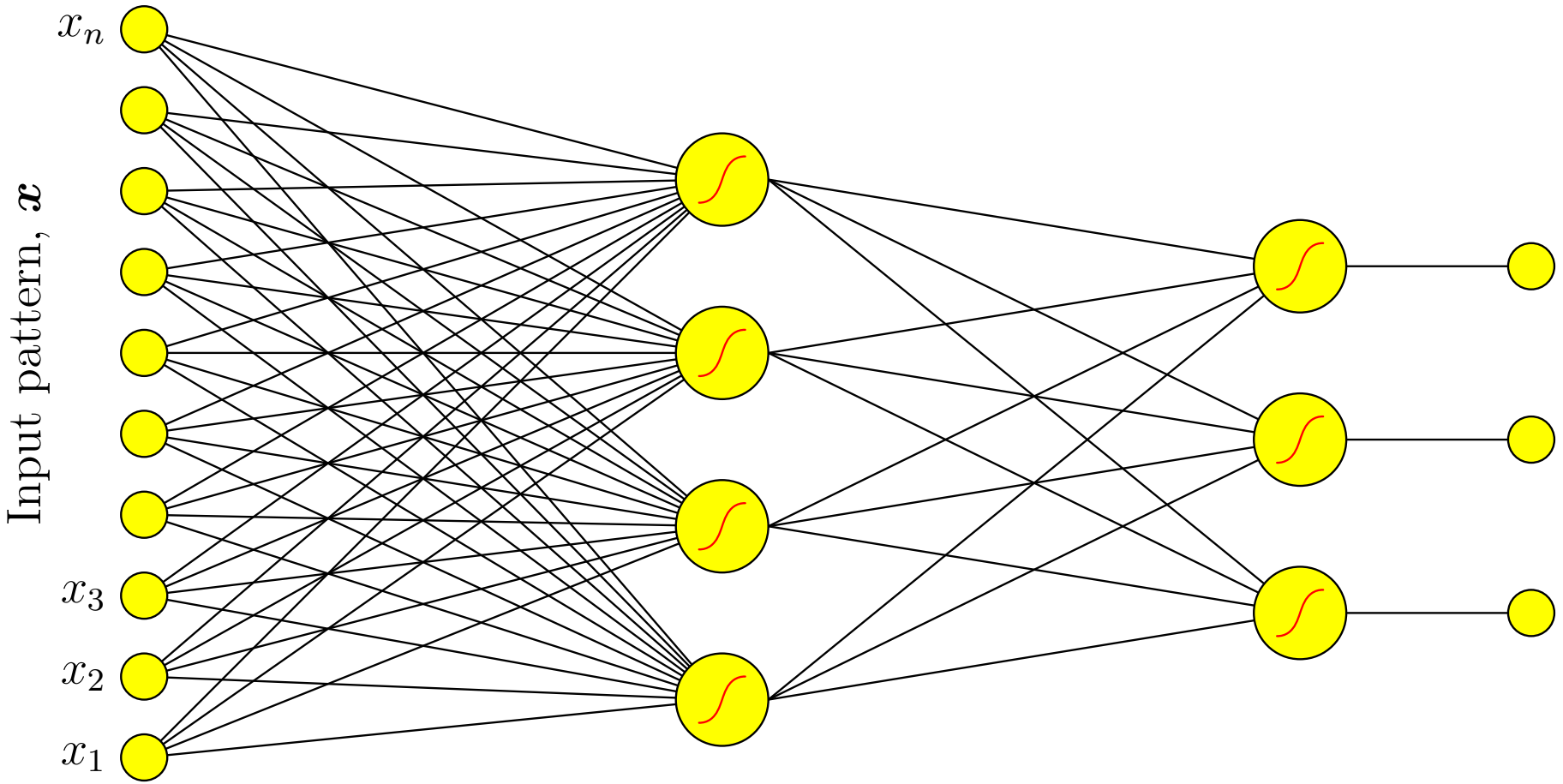
Recurrent Network

- Feed-forward networks are rather limited for some applications
- They are not particularly well matched for interpreting arbitrary length time series
- To interpret a sentence, or to predict tomorrows weather it is necessary to remember what happened in the past
- To facilitate this we would like to add a feedback loop delayed in time

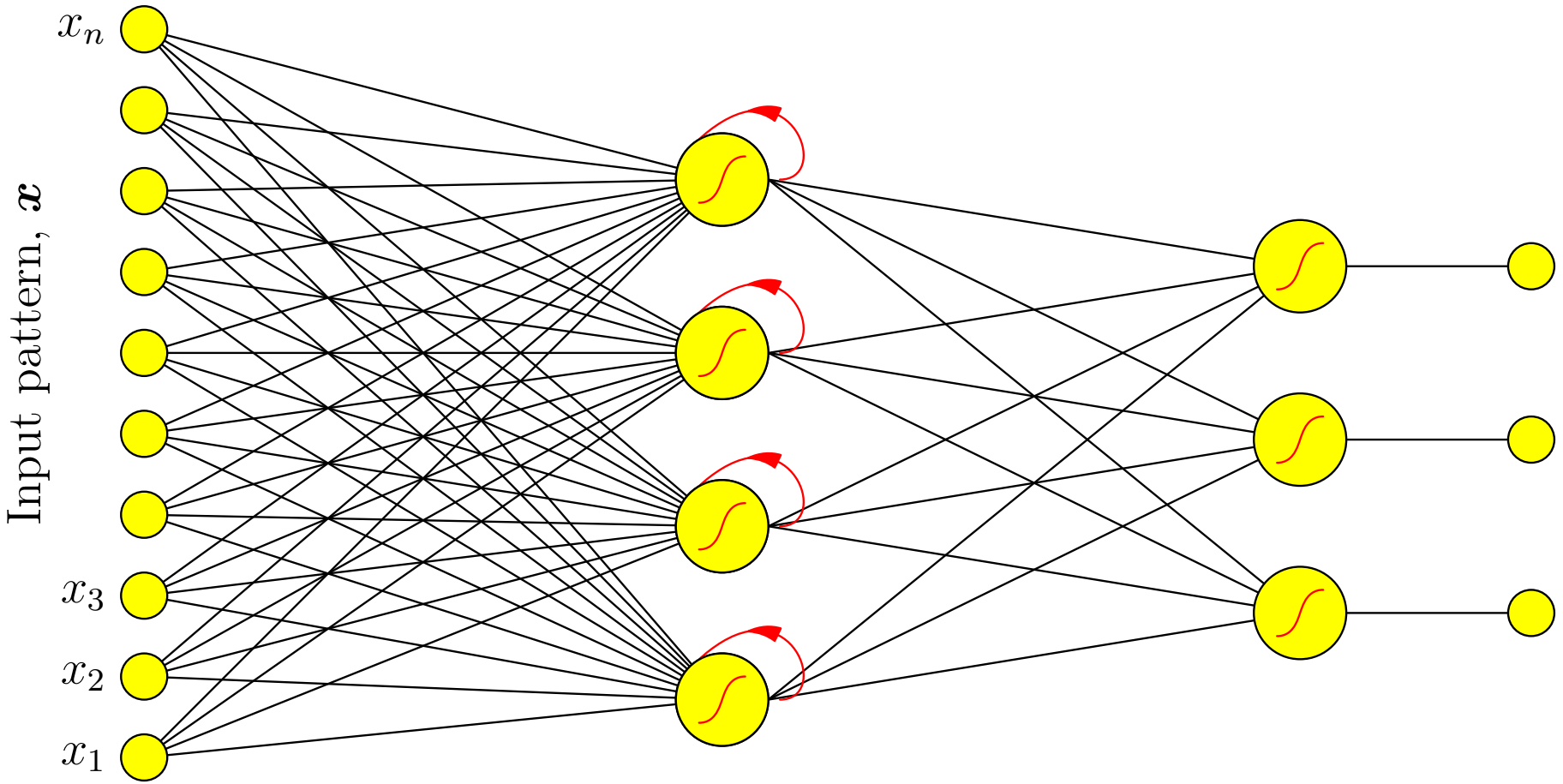
Recurrent Network

- Feed-forward networks are rather limited for some applications
- They are not particularly well matched for interpreting arbitrary length time series
- To interpret a sentence, or to predict tomorrows weather it is necessary to remember what happened in the past
- To facilitate this we would like to add a feedback loop delayed in time

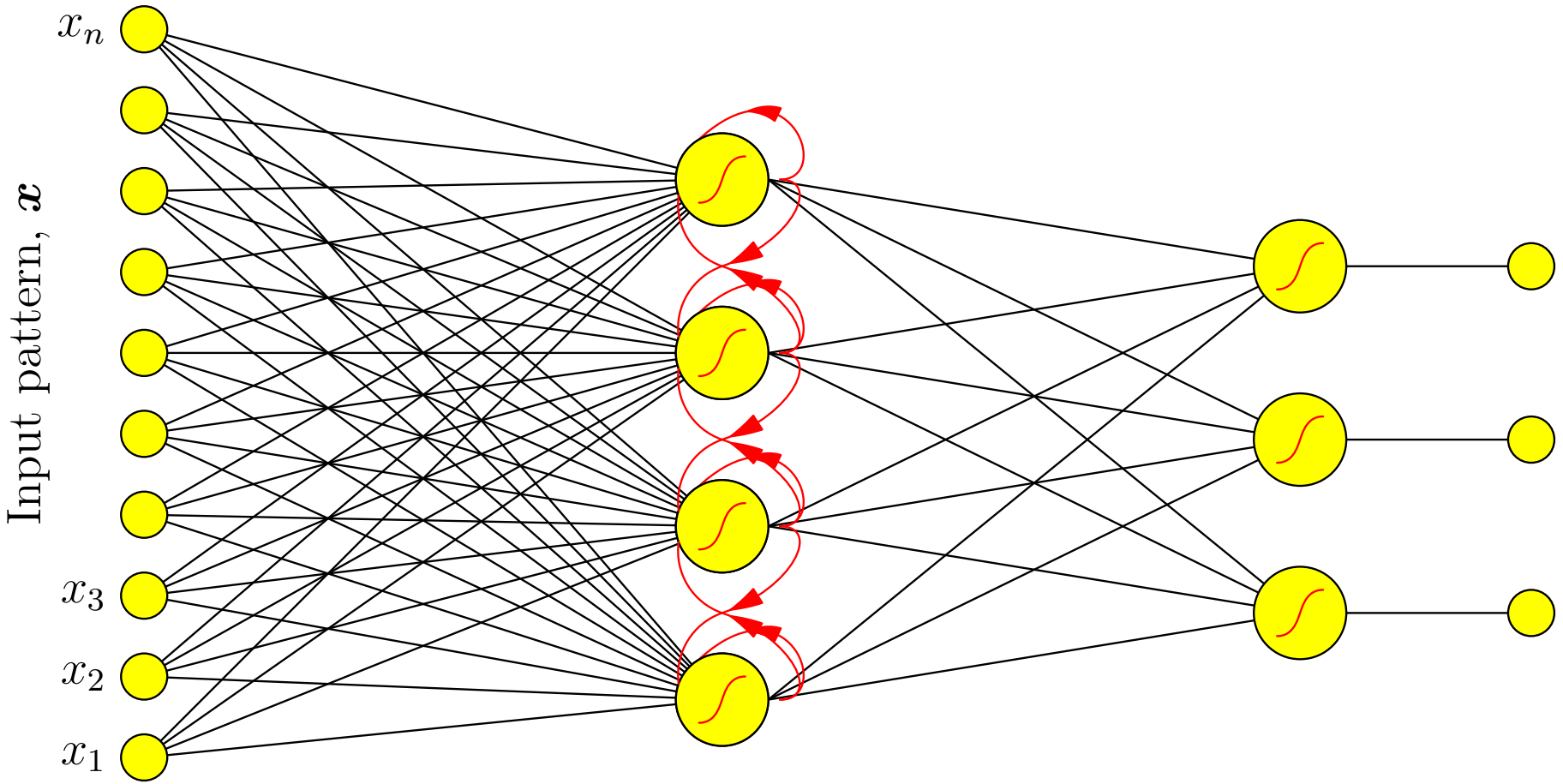
Recurrent Network



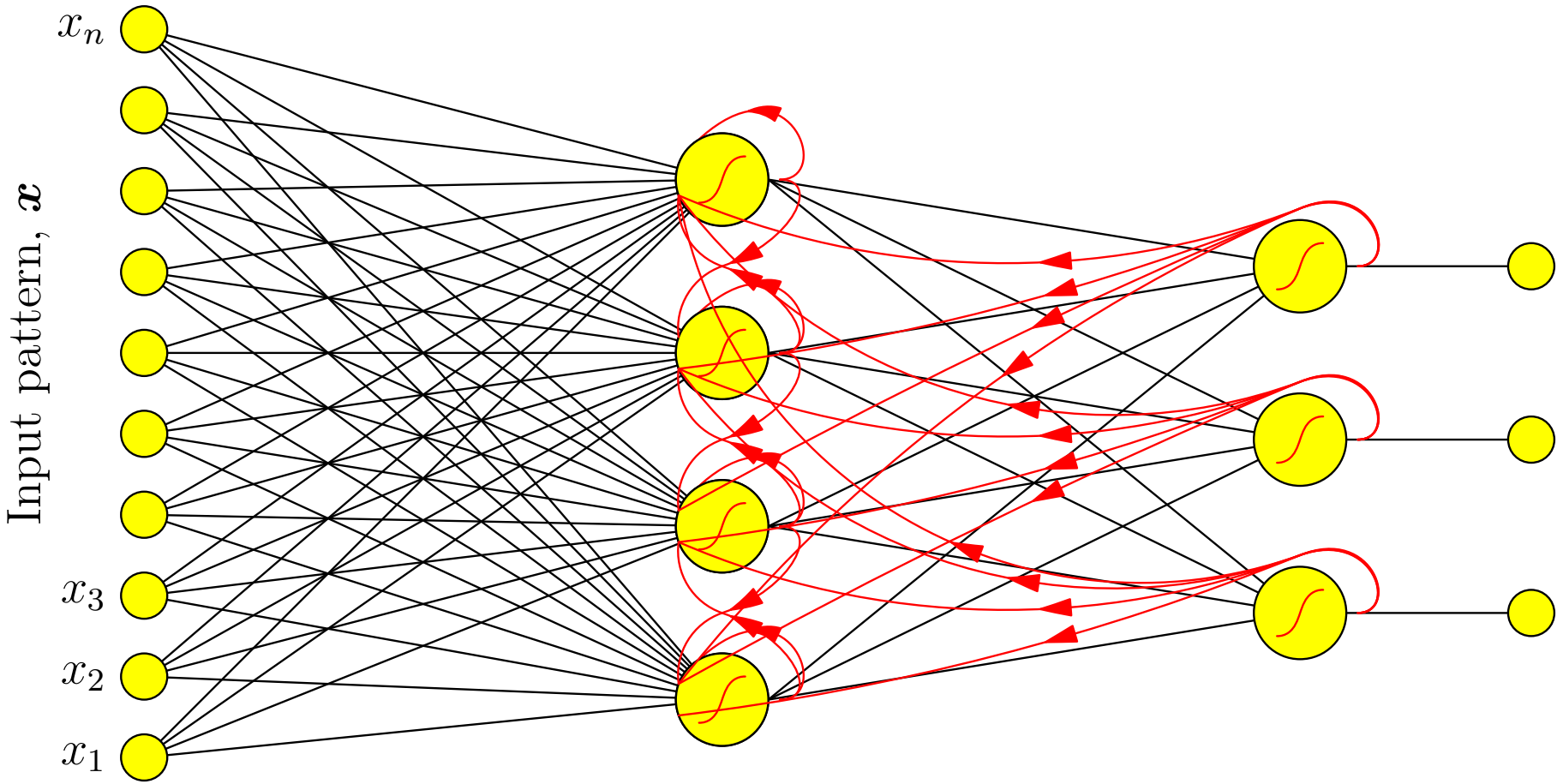
Recurrent Network



Recurrent Network

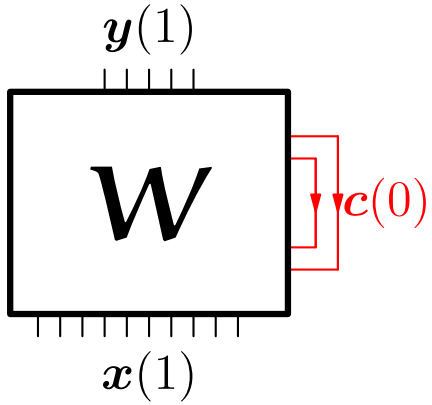


Recurrent Network



Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t)) \middle| t = 1, 2, \ldots, T \right)$



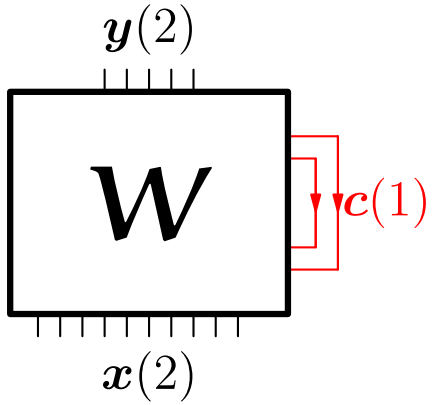
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1) | \boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



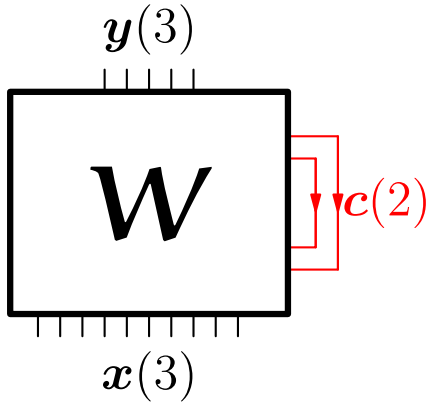
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



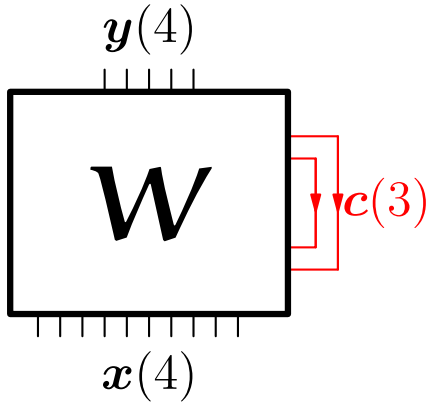
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



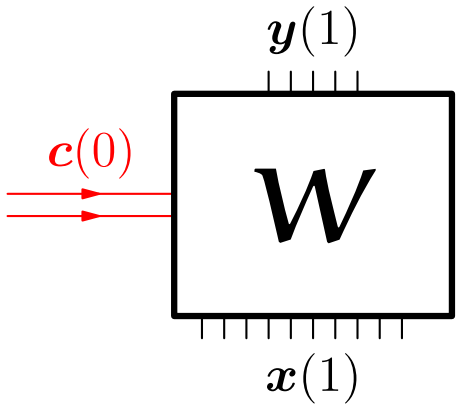
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



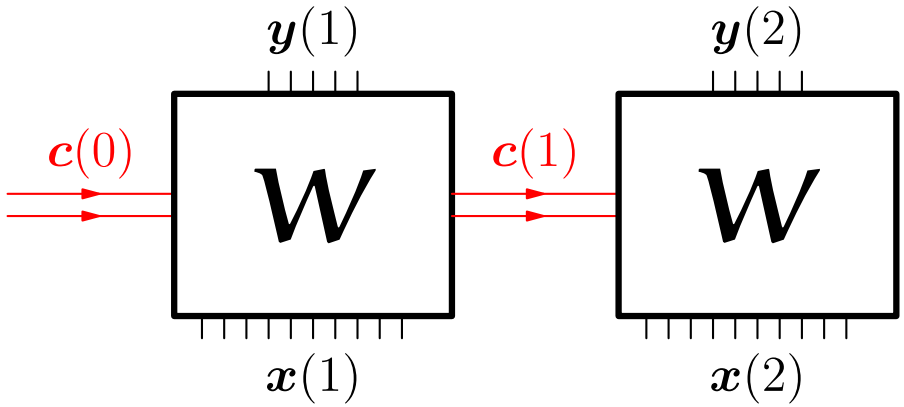
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



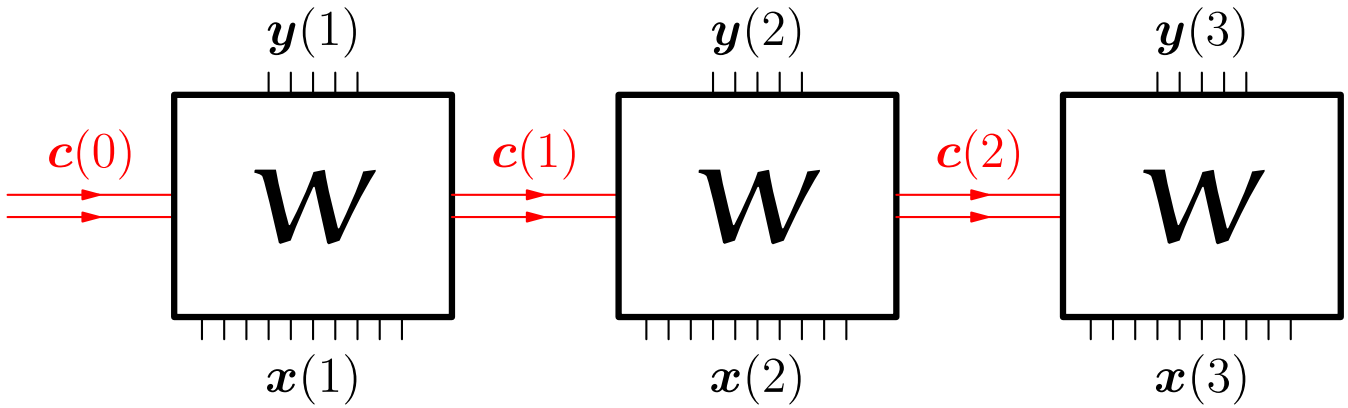
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



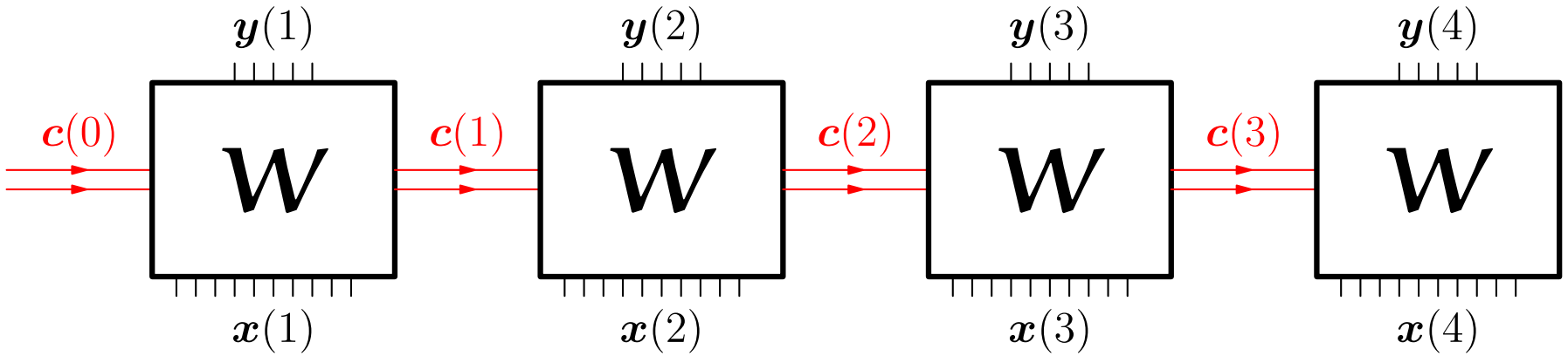
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



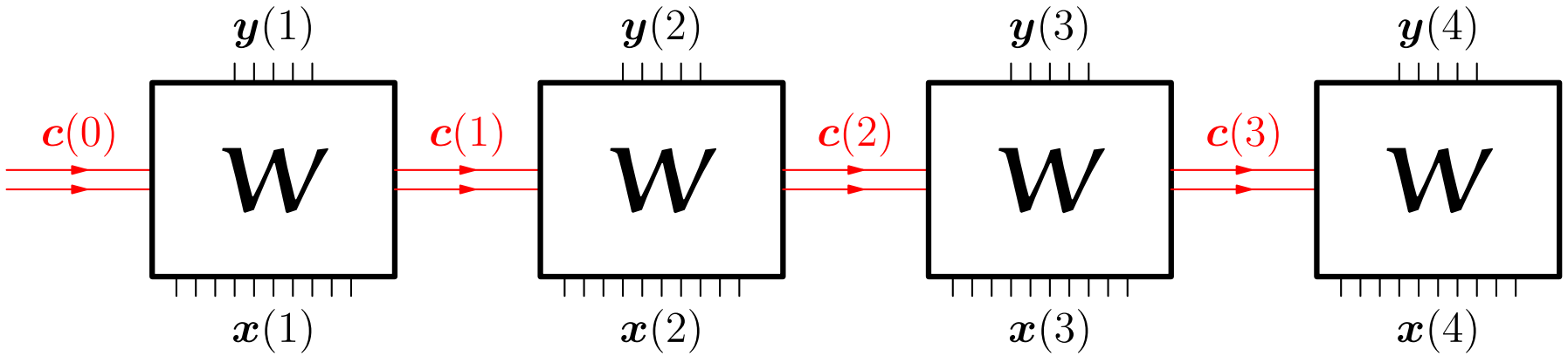
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



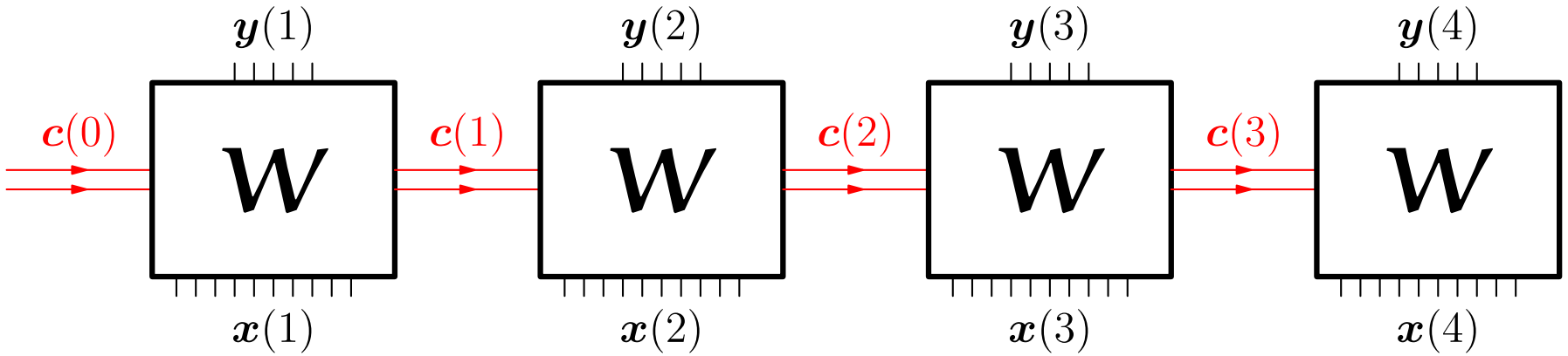
- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Training Recurrent Networks

- Given a set of inputs $\mathcal{D} = \left((\boldsymbol{x}(t), \boldsymbol{y}(t))\middle| t = 1, 2, \ldots, T\right)$



- Minimise the error

$$E(\boldsymbol{W}) = \sum_{t=1}^{T} \left\| \boldsymbol{y}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{c}(t-1)|\boldsymbol{W}) \right\|^2$$

- This is known as **back-propagation through time**

Vanishing and Exploding Gradients

- Note that $\boldsymbol{c}(t) = \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- If the output $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-3)$, say then prediction will be

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{f}_1(\boldsymbol{x}(t-2), \boldsymbol{f}_1(\boldsymbol{x}(t-3)|\boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W})$$

- The back-propagated error will involve applying $\boldsymbol{f}_1$ multiple times
- Each time the error will get multiplied by some factor a
- If $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-\tau)$ then the back-propagated signal will be proportional to $a^{\tau-1}$
- This either vanishes or explodes when τ becomes large

Vanishing and Exploding Gradients

- Note that $\boldsymbol{c}(t) = \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- If the output $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-3)$, say then prediction will be

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{f}_1(\boldsymbol{x}(t-2), \boldsymbol{f}_1(\boldsymbol{x}(t-3)|\boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W})$$

- The back-propagated error will involve applying $\boldsymbol{f}_1$ multiple times
- Each time the error will get multiplied by some factor a
- If $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-\tau)$ then the back-propagated signal will be proportional to $a^{\tau-1}$
- This either vanishes or explodes when τ becomes large

Vanishing and Exploding Gradients

- Note that $\boldsymbol{c}(t) = \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- If the output $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-3)$, say then prediction will be

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{f}_1(\boldsymbol{x}(t-2), \boldsymbol{f}_1(\boldsymbol{x}(t-3)|\boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W})$$

- The back-propagated error will involve applying $\boldsymbol{f}_1$ multiple times
- Each time the error will get multiplied by some factor a
- If $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-\tau)$ then the back-propagated signal will be proportional to $a^{\tau-1}$
- This either vanishes or explodes when τ becomes large

Vanishing and Exploding Gradients

- Note that $\boldsymbol{c}(t) = \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- If the output $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-3)$, say then prediction will be

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{f}_1(\boldsymbol{x}(t-2), \boldsymbol{f}_1(\boldsymbol{x}(t-3)|\boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W})$$

- The back-propagated error will involve applying $\boldsymbol{f}_1$ multiple times
- Each time the error will get multiplied by some factor a
- If $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-\tau)$ then the back-propagated signal will be proportional to $a^{\tau-1}$
- This either vanishes or explodes when τ becomes large

Vanishing and Exploding Gradients

- Note that $\boldsymbol{c}(t) = \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- If the output $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-3)$, say then prediction will be

$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{f}_1(\boldsymbol{x}(t-2), \boldsymbol{f}_1(\boldsymbol{x}(t-3)|\boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W})$$

- The back-propagated error will involve applying $\boldsymbol{f}_1$ multiple times
- Each time the error will get multiplied by some factor a
- If $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-\tau)$ then the back-propagated signal will be proportional to $a^{\tau-1}$
- This either vanishes or explodes when τ becomes large

Vanishing and Exploding Gradients

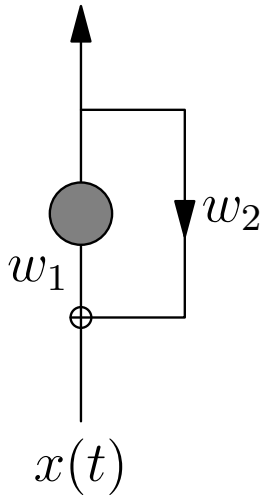
- Note that $\boldsymbol{c}(t) = \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{c}(t-1)|\boldsymbol{W})$
- If the output $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-3)$, say then prediction will be

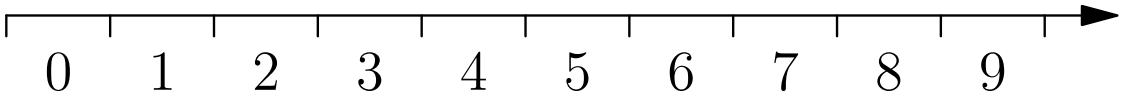
$$\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{f}_1(\boldsymbol{x}(t-1), \boldsymbol{f}_1(\boldsymbol{x}(t-2), \boldsymbol{f}_1(\boldsymbol{x}(t-3)|\boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W}), \boldsymbol{W})$$

- The back-propagated error will involve applying $\boldsymbol{f}_1$ multiple times
- Each time the error will get multiplied by some factor a
- If $\boldsymbol{y}(t)$ depends on the input $\boldsymbol{x}(t-\tau)$ then the back-propagated signal will be proportional to $a^{\tau-1}$
- This either vanishes or explodes when τ becomes large

Simple Recurrent Network

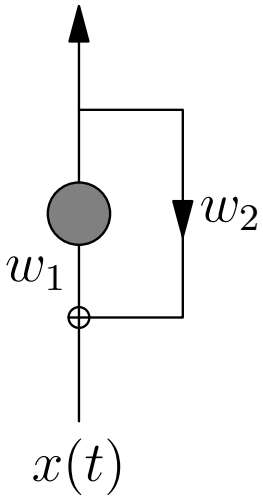
$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$



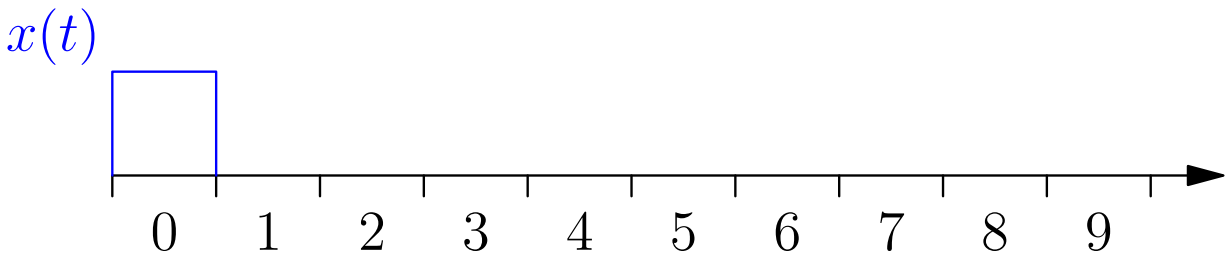


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

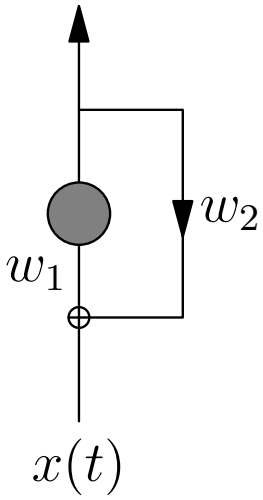


$$w_1 = w_2 = 0.9$$

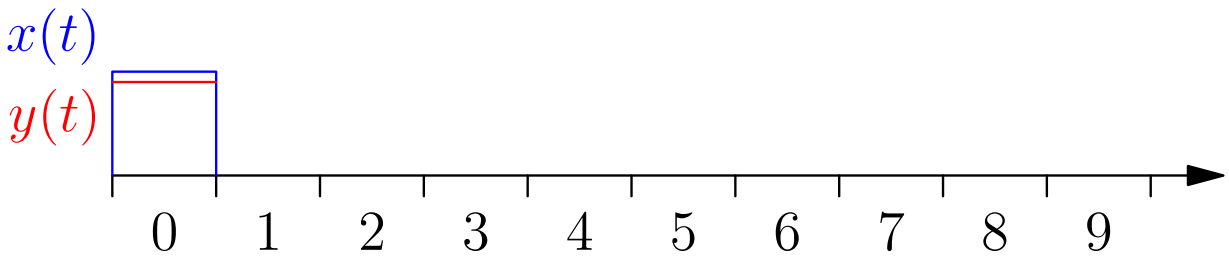


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

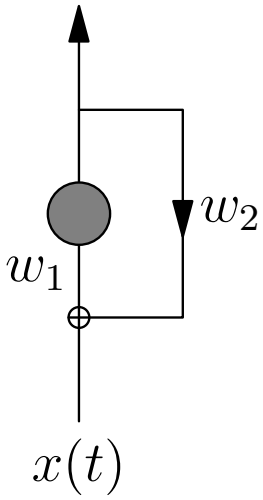


$$w_1 = w_2 = 0.9$$

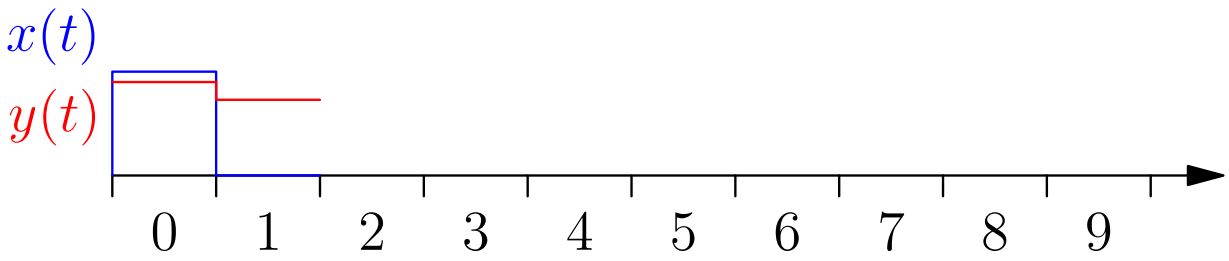


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

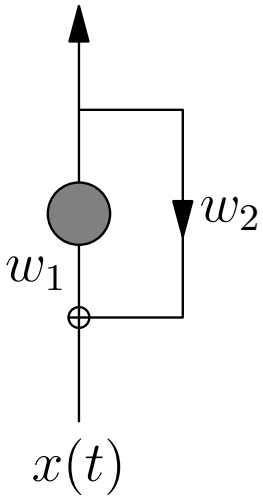


$$w_1 = w_2 = 0.9$$

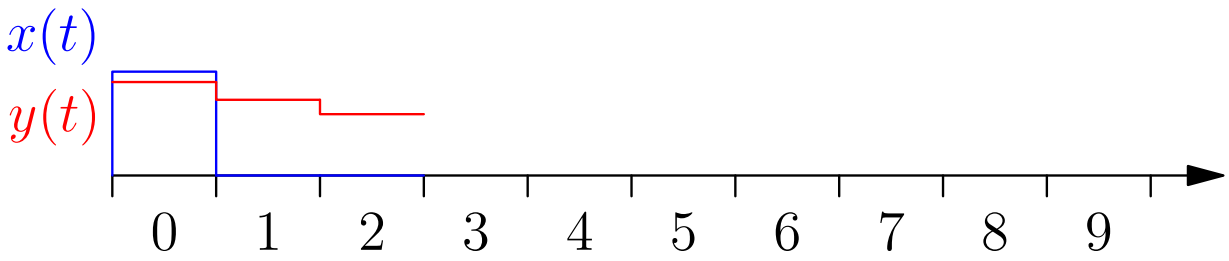


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

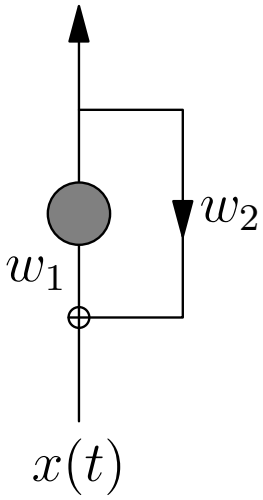


$$w_1 = w_2 = 0.9$$

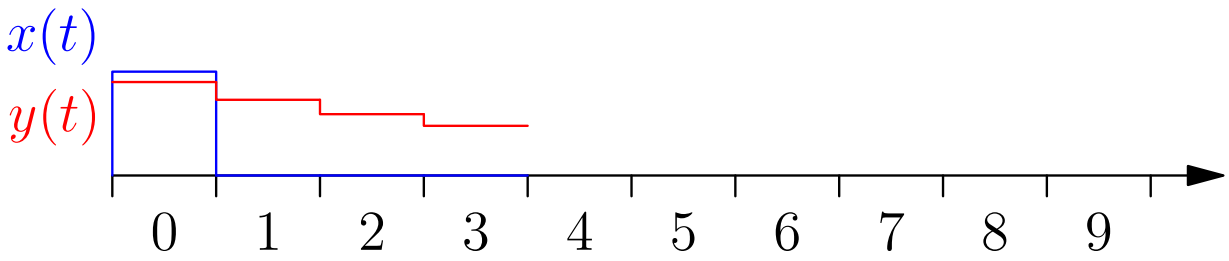


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

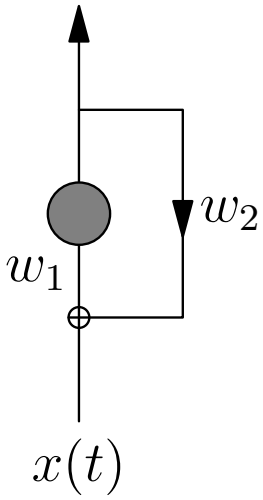


$$w_1 = w_2 = 0.9$$

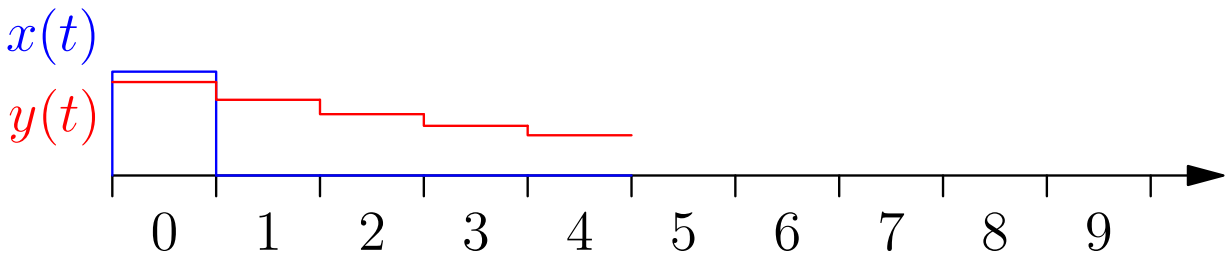


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

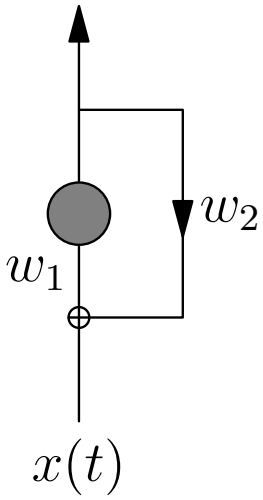


$$w_1 = w_2 = 0.9$$

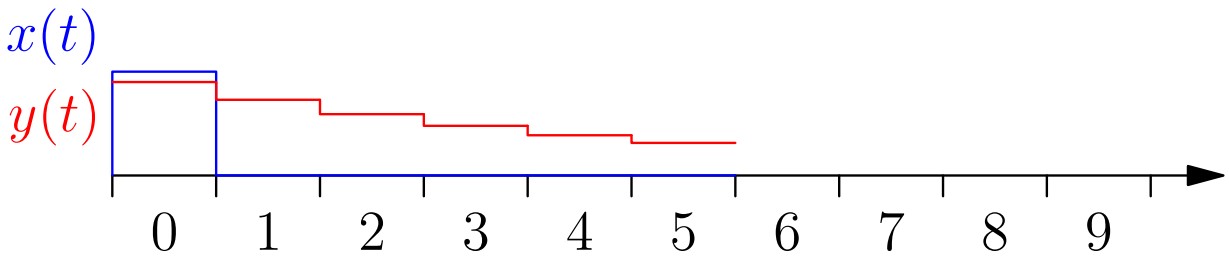


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

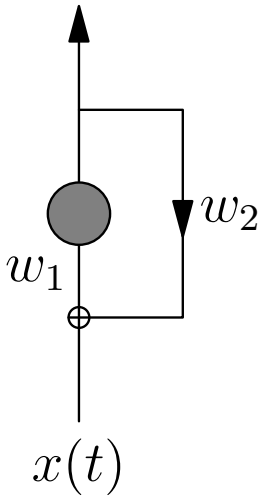


$$w_1 = w_2 = 0.9$$

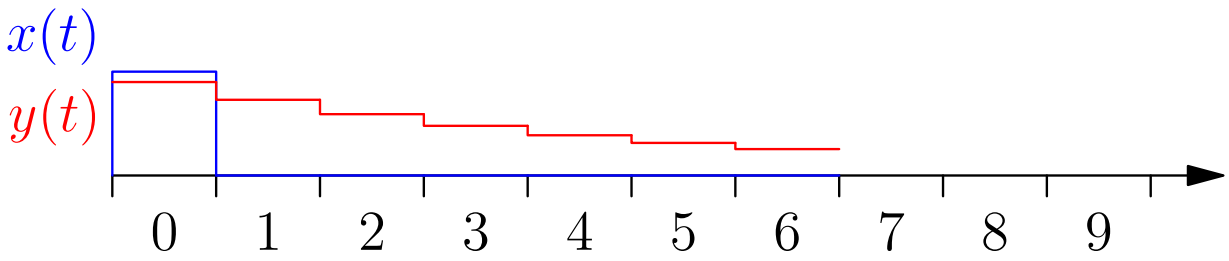


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

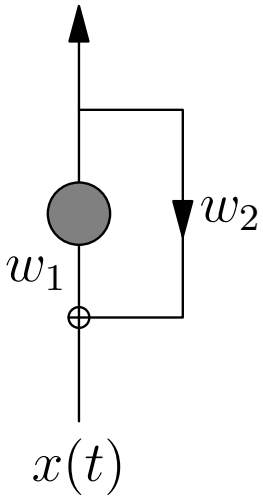


$$w_1 = w_2 = 0.9$$

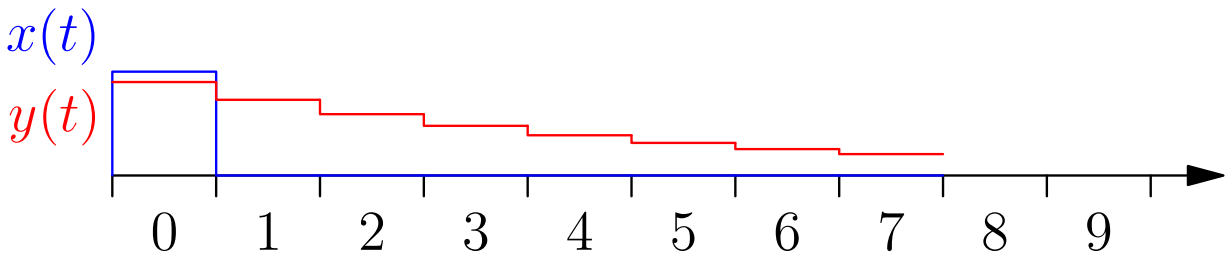


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

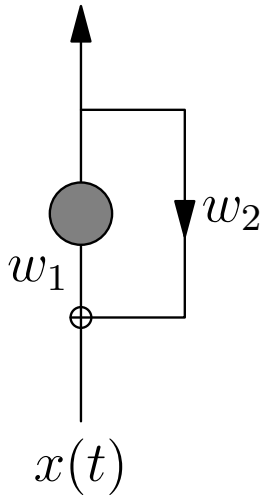


$$w_1 = w_2 = 0.9$$

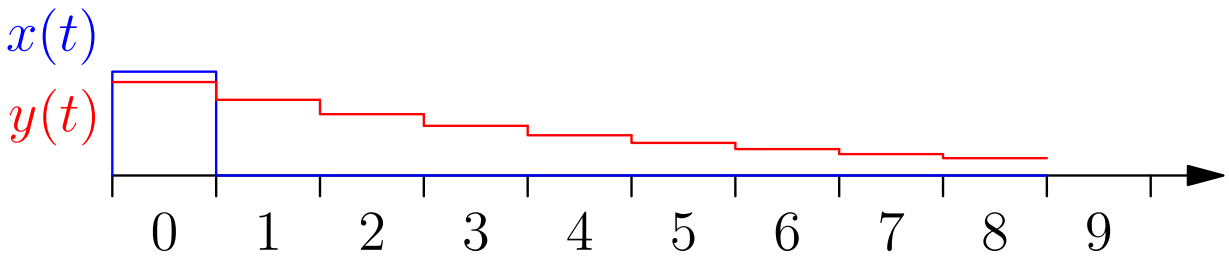


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

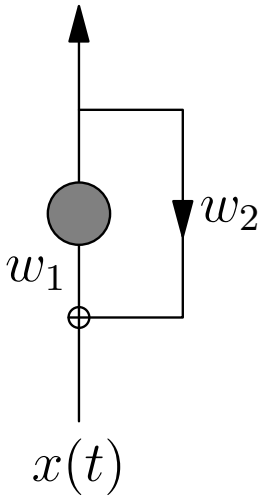


$$w_1 = w_2 = 0.9$$

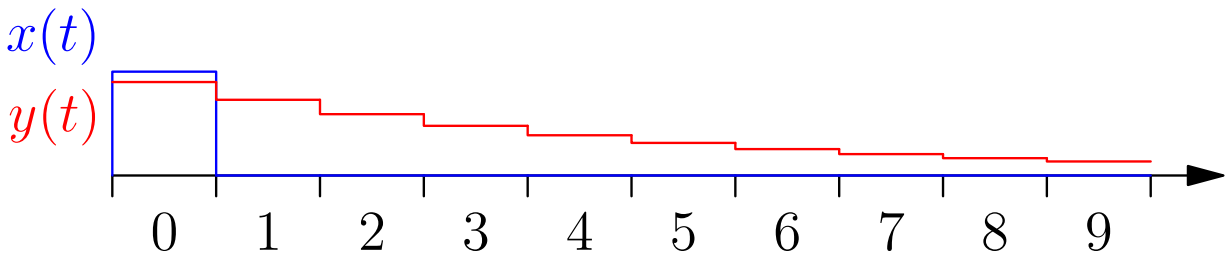


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

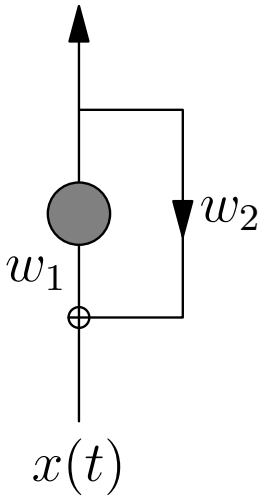


$$w_1 = w_2 = 0.9$$

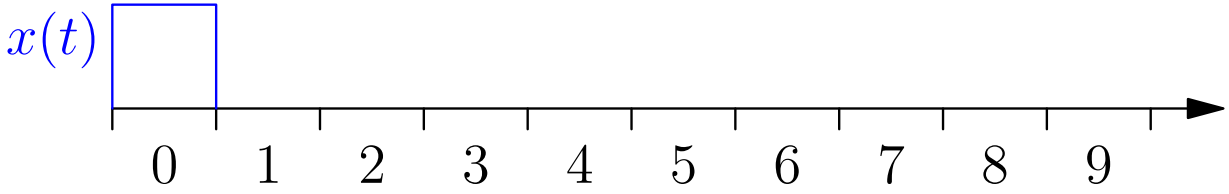


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

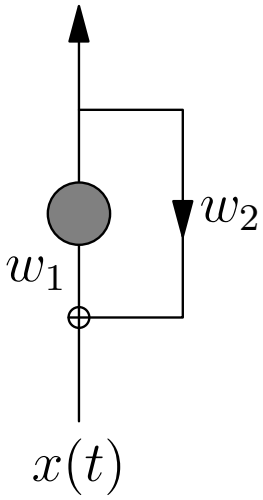


$$w_1 = w_2 = 1.1$$

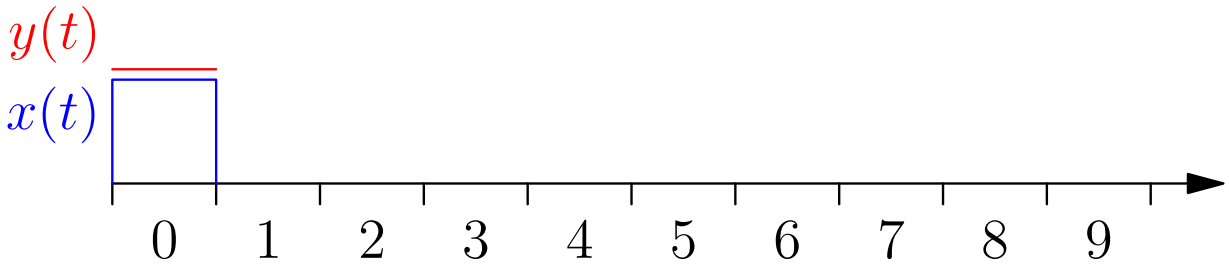


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

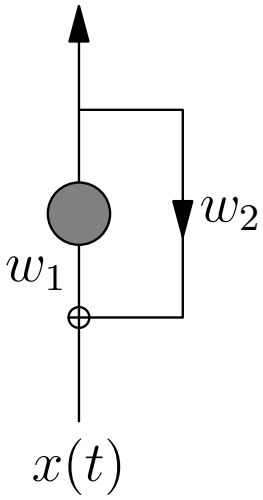


$$w_1 = w_2 = 1.1$$

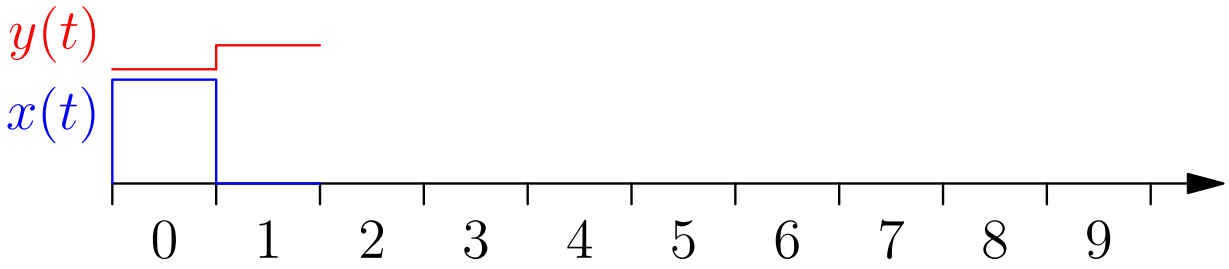


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

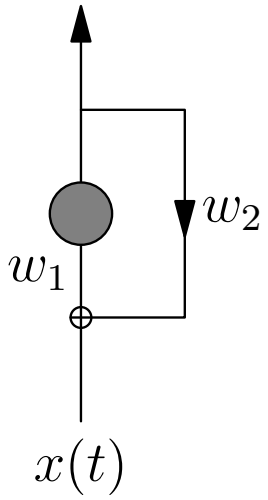


$$w_1 = w_2 = 1.1$$

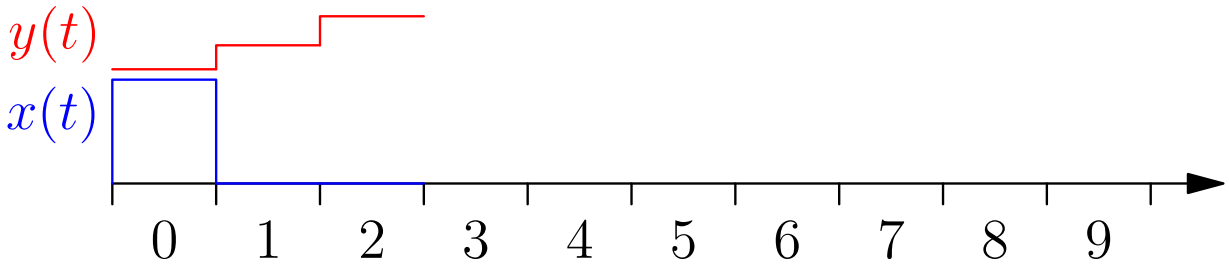


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

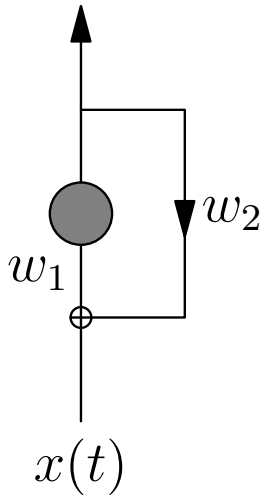


$$w_1 = w_2 = 1.1$$

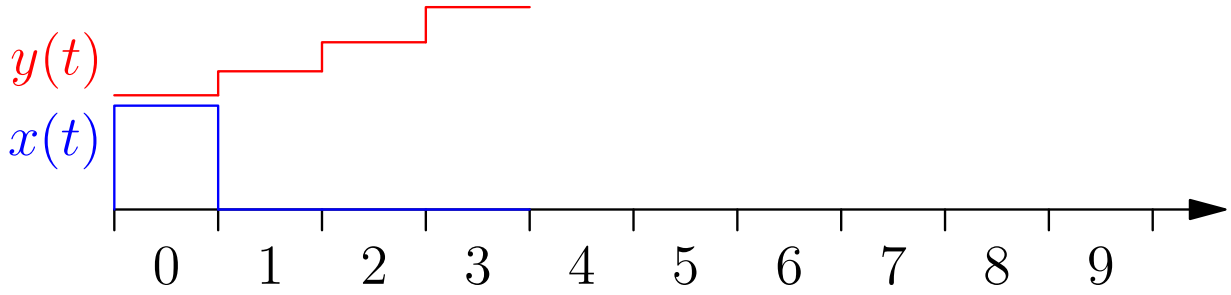


Simple Recurrent Network

$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

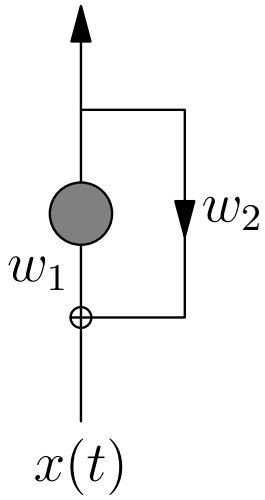


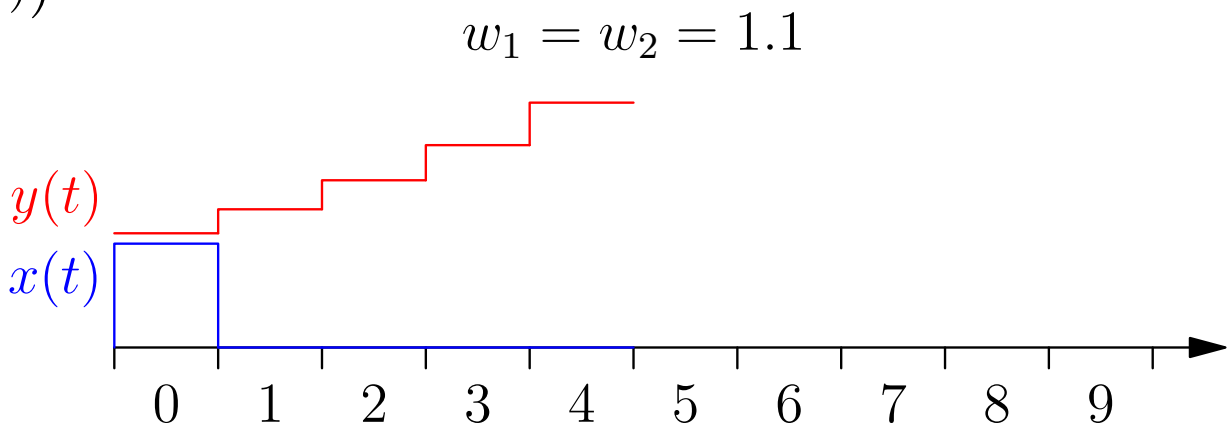
$$w_1 = w_2 = 1.1$$



Simple Recurrent Network

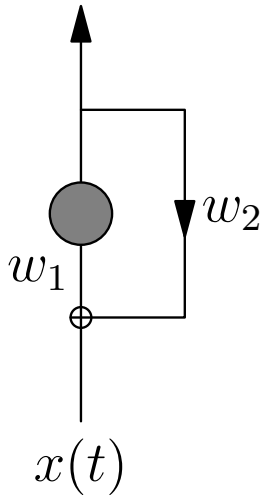
$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

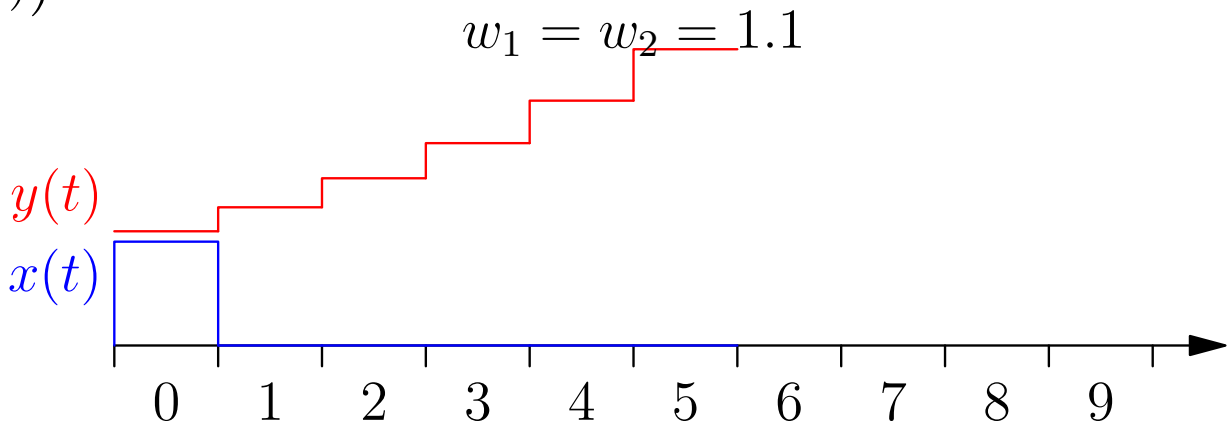




Simple Recurrent Network

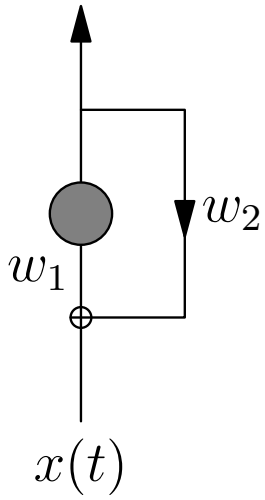
$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$

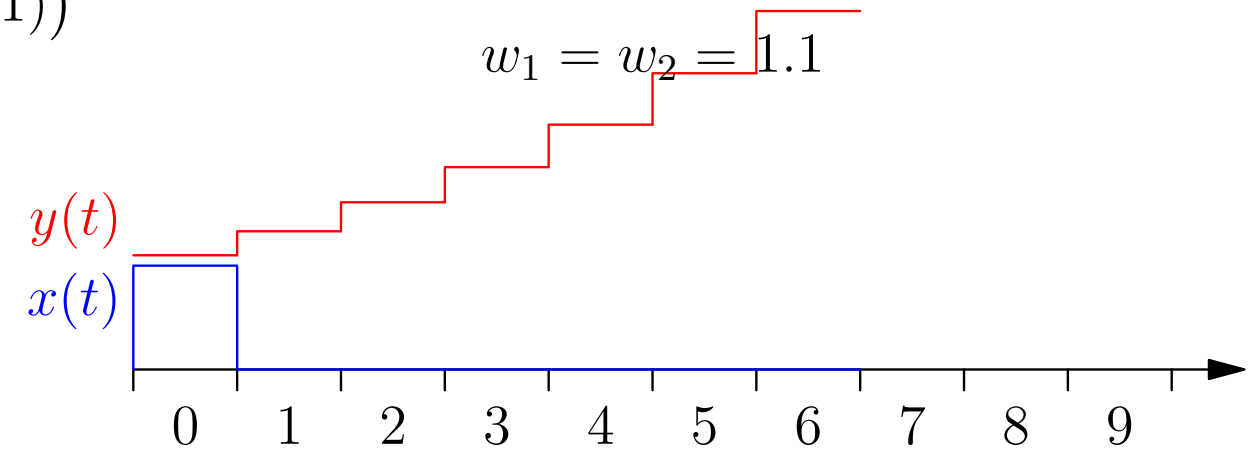




Simple Recurrent Network

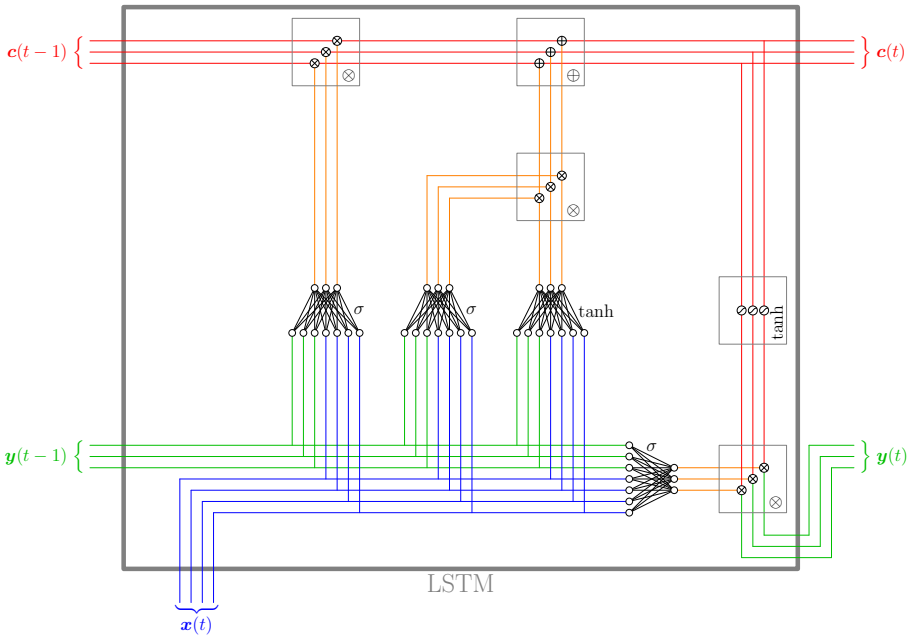
$$y(t) = w_1 \left(x(t) + w_2 \, y(t-1) \right)$$





Outline

1. More on CNNs
2. Recurrent Networks
3. **LSTM**
4. Word Embedding
5. Applications



LSTM Architecture

- LSTMs (long-short term memory) was designed to solve this problem
- It recognised that to retain a long time memory requires

$$\boldsymbol{c}(t) = \boldsymbol{c}(t-1)$$

- Sometimes we have to forget and sometimes we have to change a memory
- To do this we should use gates that saturate at 0 and 1
- Sigmoid functions naturally saturate at 0 and 1

LSTM Architecture

- LSTMs (long-short term memory) was designed to solve this problem
- It recognised that to retain a long time memory requires

$$\boldsymbol{c}(t) = \boldsymbol{c}(t-1)$$

- Sometimes we have to forget and sometimes we have to change a memory
- To do this we should use gates that saturate at 0 and 1
- Sigmoid functions naturally saturate at 0 and 1

LSTM Architecture

- LSTMs (long-short term memory) was designed to solve this problem
- It recognised that to retain a long time memory requires

$$\boldsymbol{c}(t) = \boldsymbol{c}(t-1)$$

- Sometimes we have to forget and sometimes we have to change a memory
- To do this we should use gates that saturate at 0 and 1
- Sigmoid functions naturally saturate at 0 and 1

LSTM Architecture

- LSTMs (long-short term memory) was designed to solve this problem
- It recognised that to retain a long time memory requires

$$\boldsymbol{c}(t) = \boldsymbol{c}(t-1)$$

- Sometimes we have to forget and sometimes we have to change a memory
- To do this we should use gates that saturate at 0 and 1
- Sigmoid functions naturally saturate at 0 and 1

LSTM Architecture

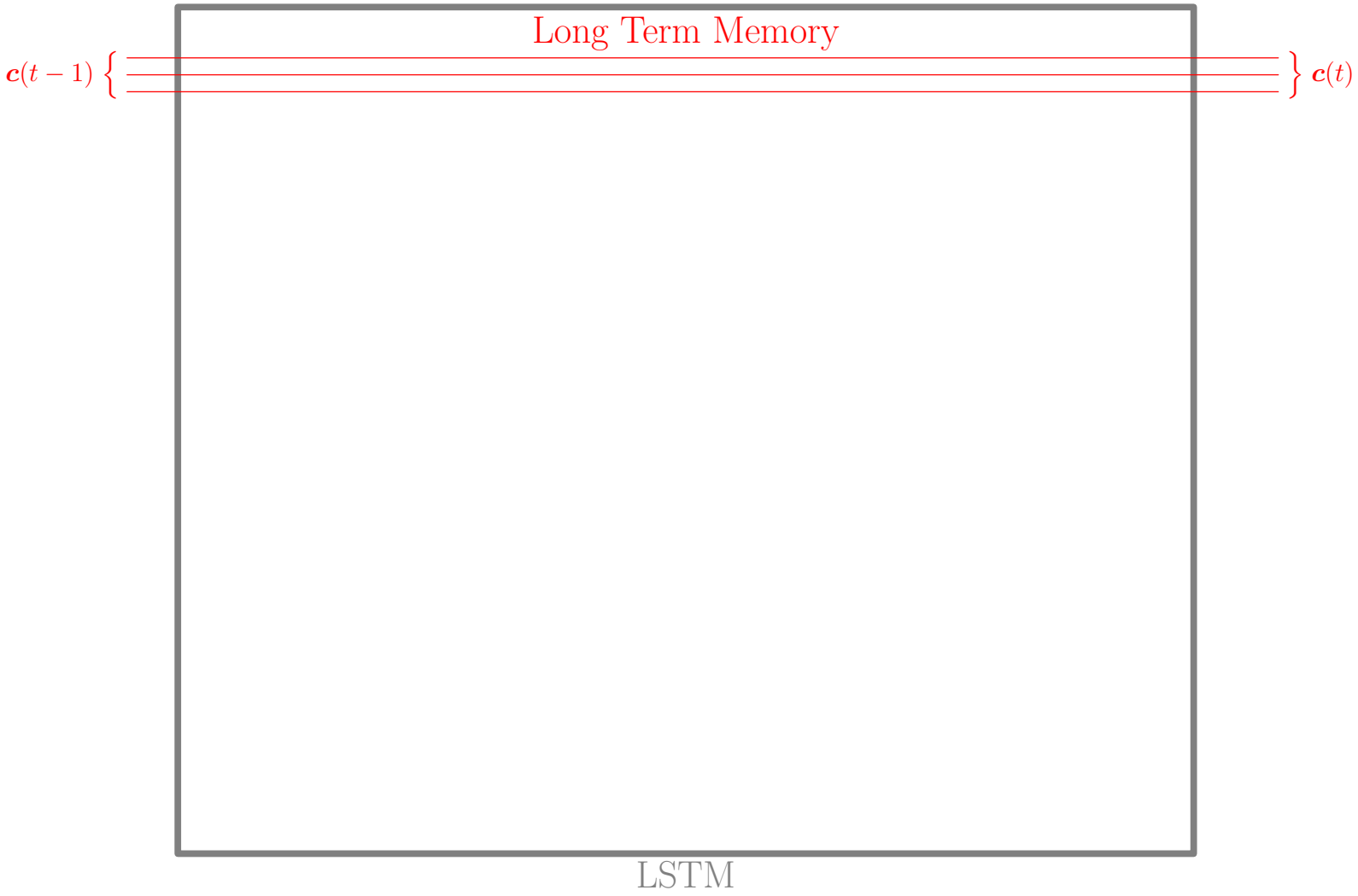
- LSTMs (long-short term memory) was designed to solve this problem
- It recognised that to retain a long time memory requires

$$\boldsymbol{c}(t) = \boldsymbol{c}(t-1)$$

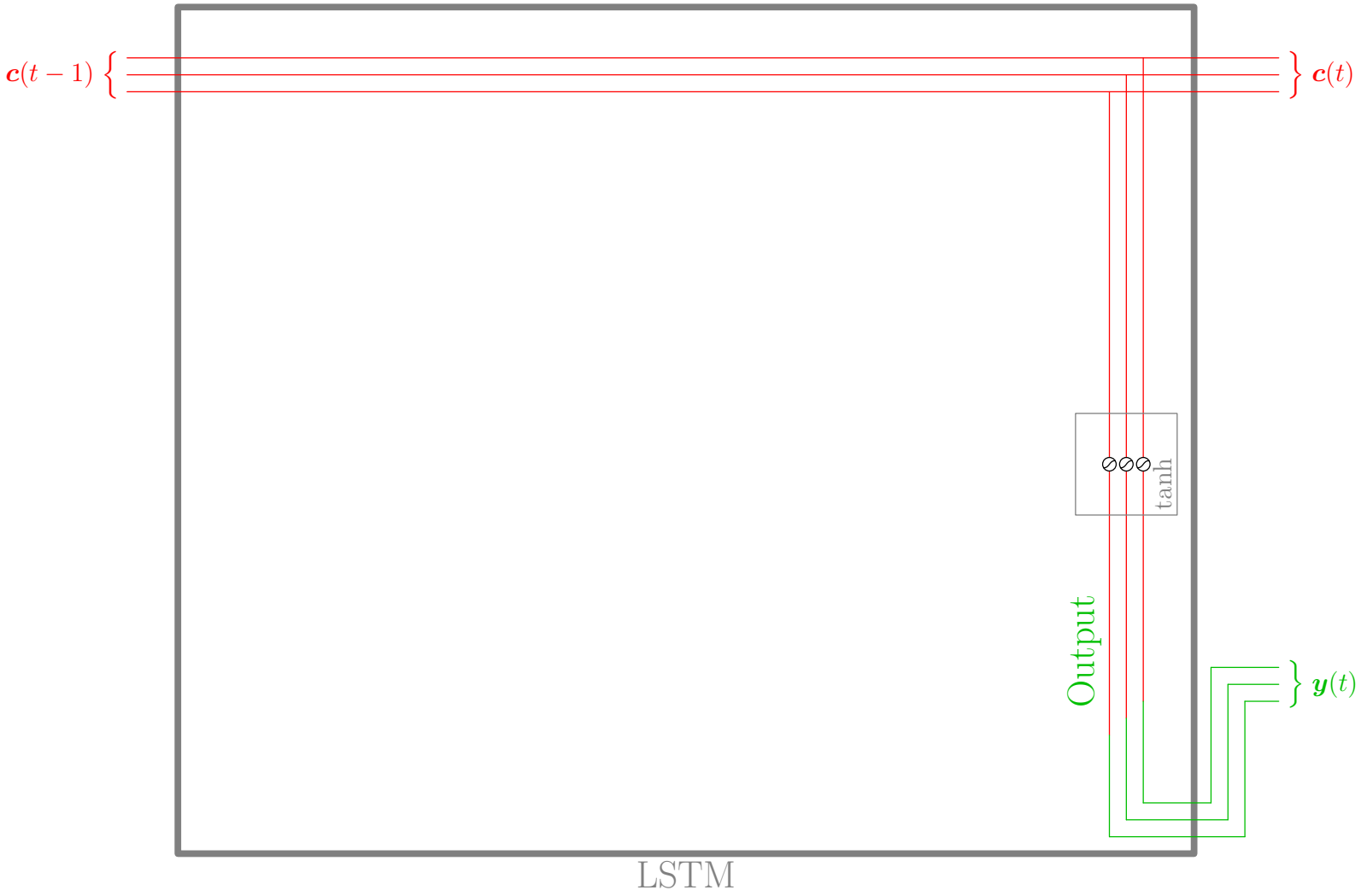
- Sometimes we have to forget and sometimes we have to change a memory
- To do this we should use gates that saturate at 0 and 1
- Sigmoid functions naturally saturate at 0 and 1

LSTM Architecture

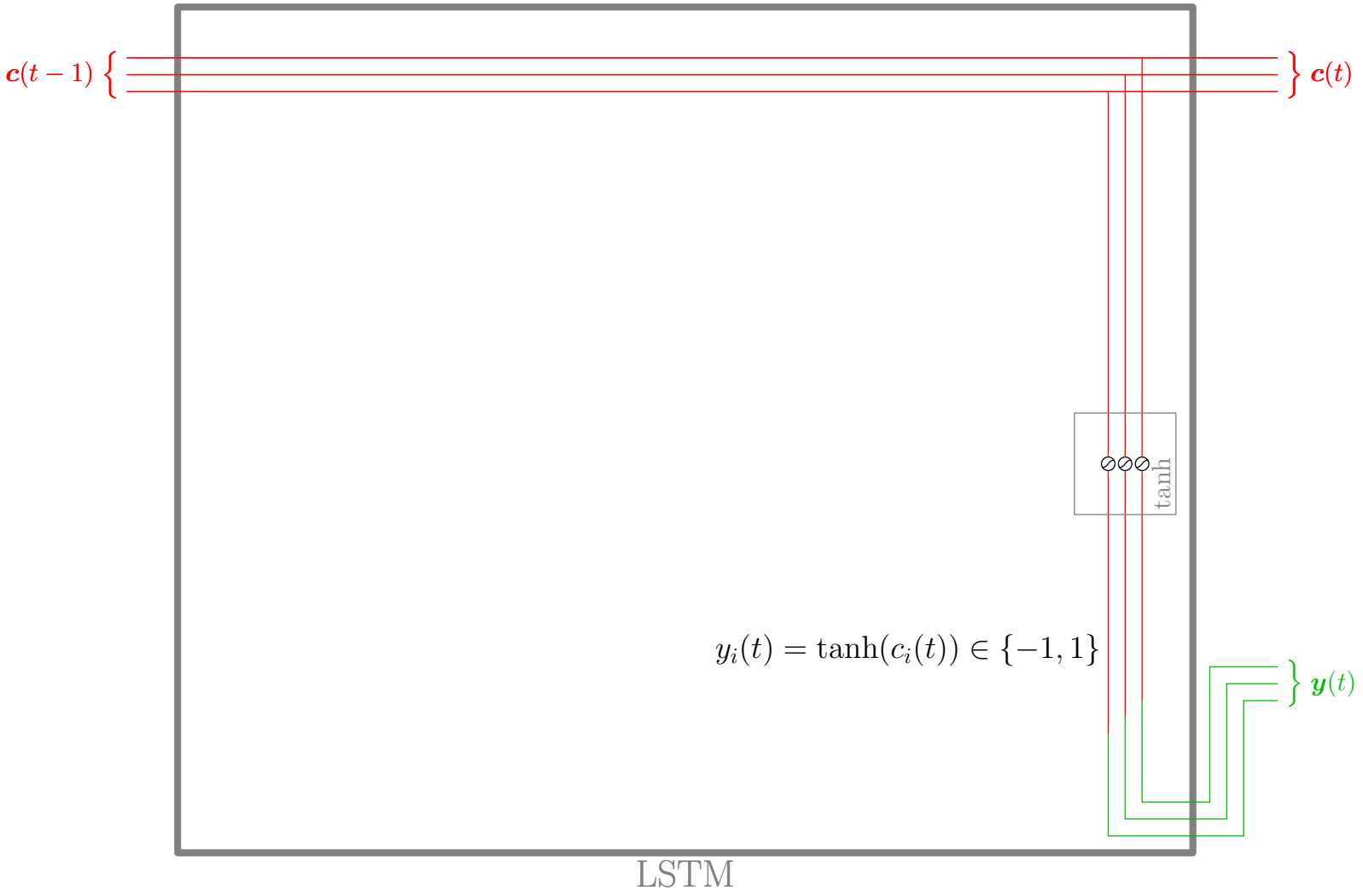
LSTM Architecture



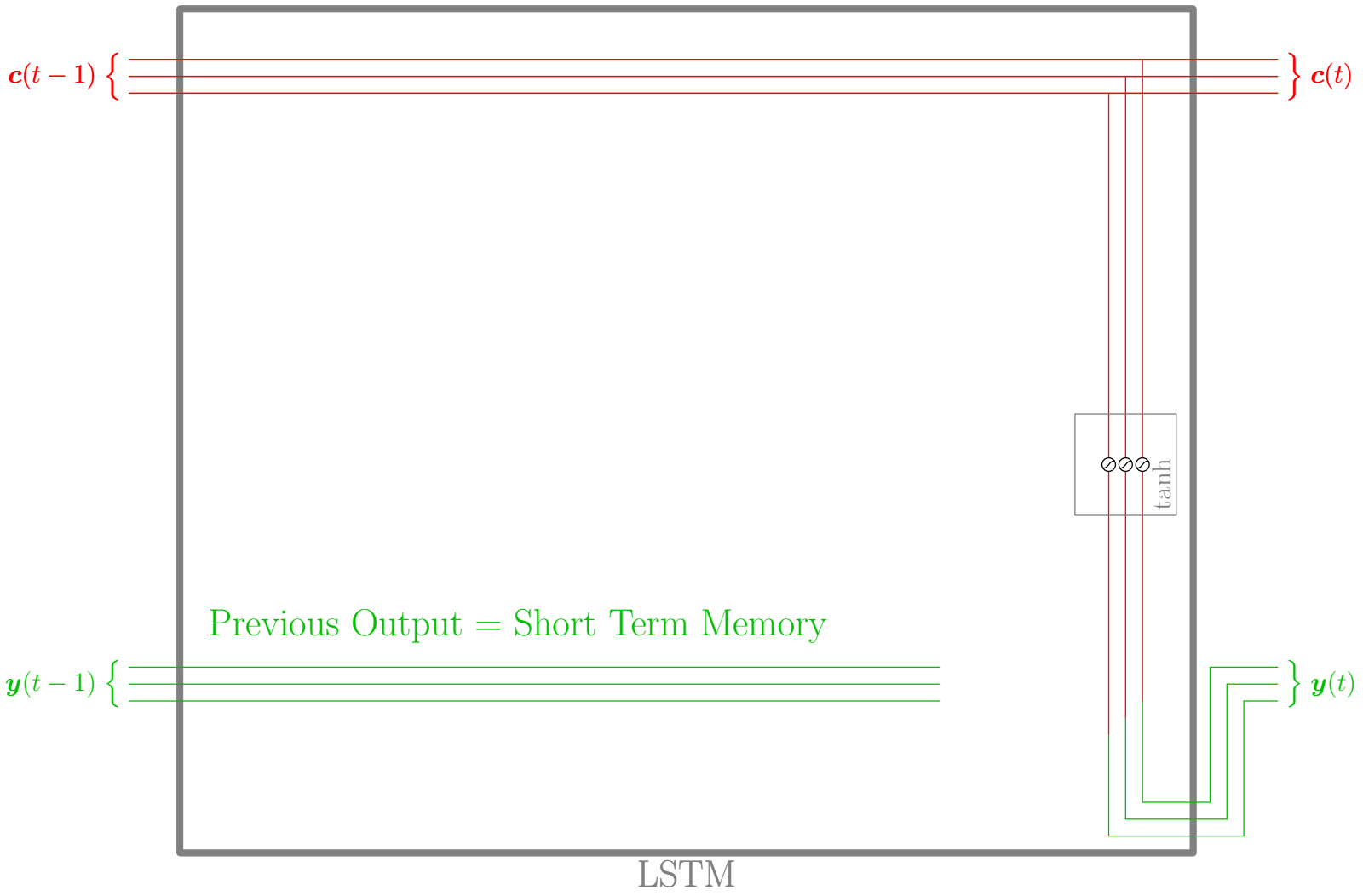
LSTM Architecture



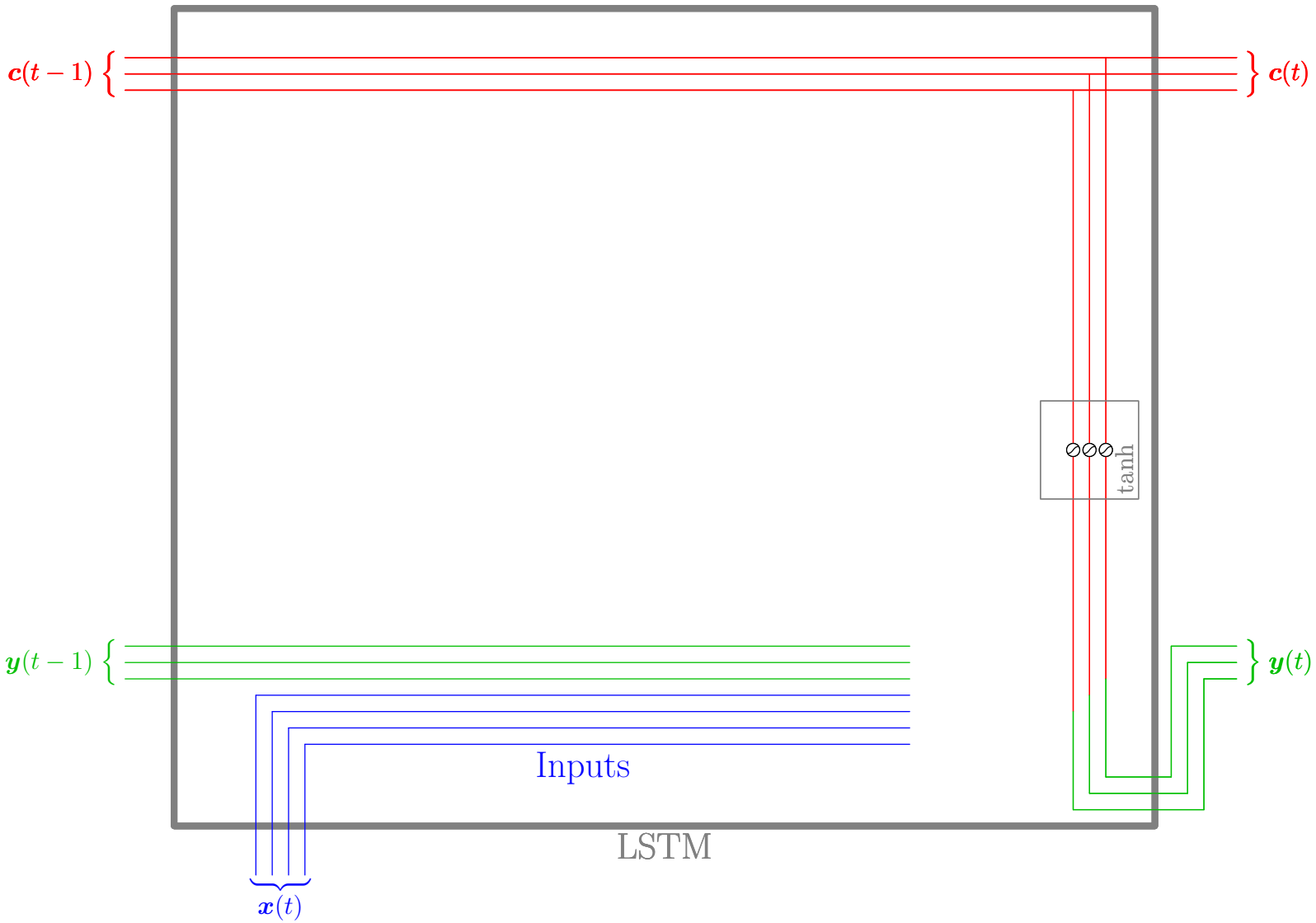
LSTM Architecture



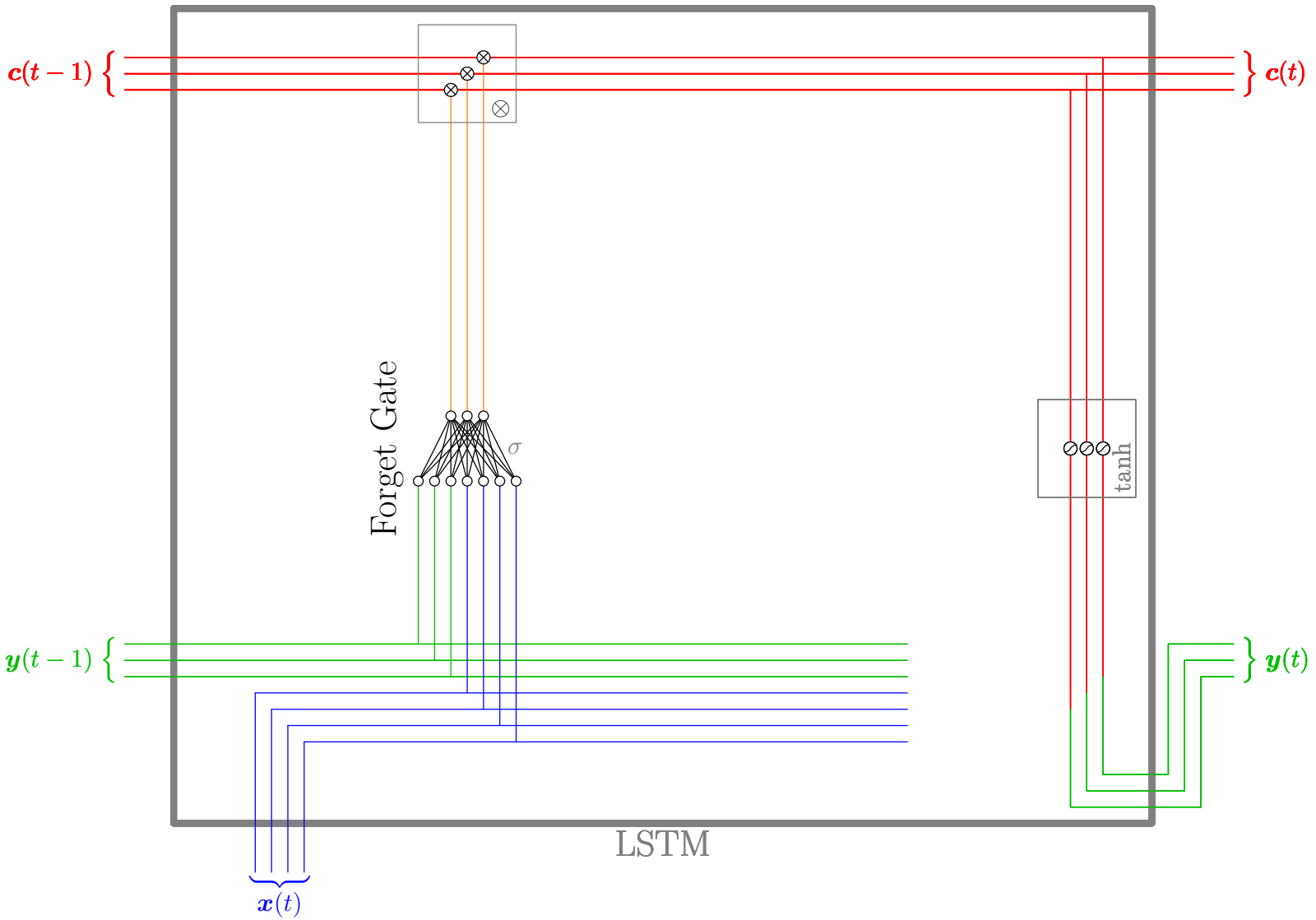
LSTM Architecture



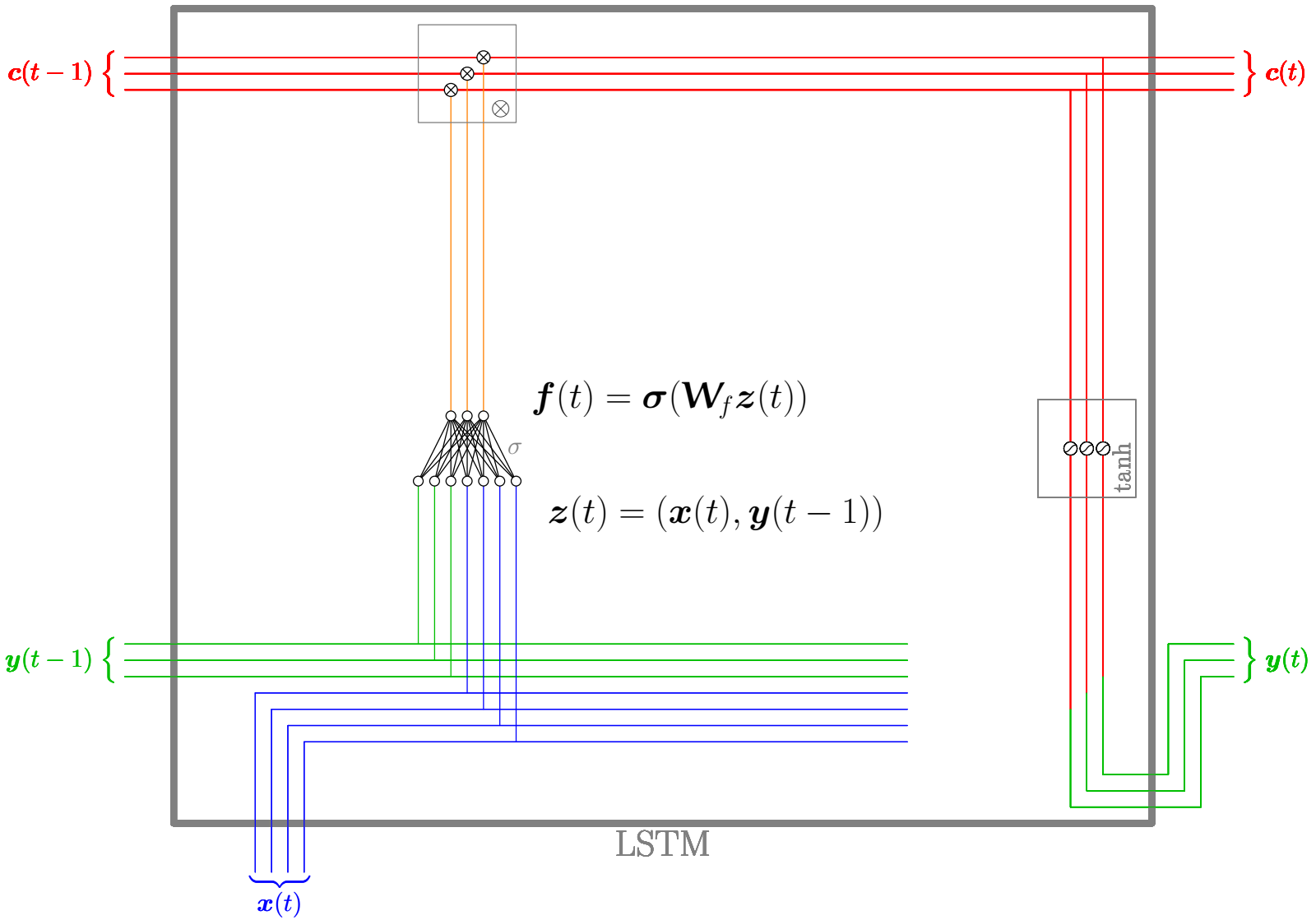
LSTM Architecture



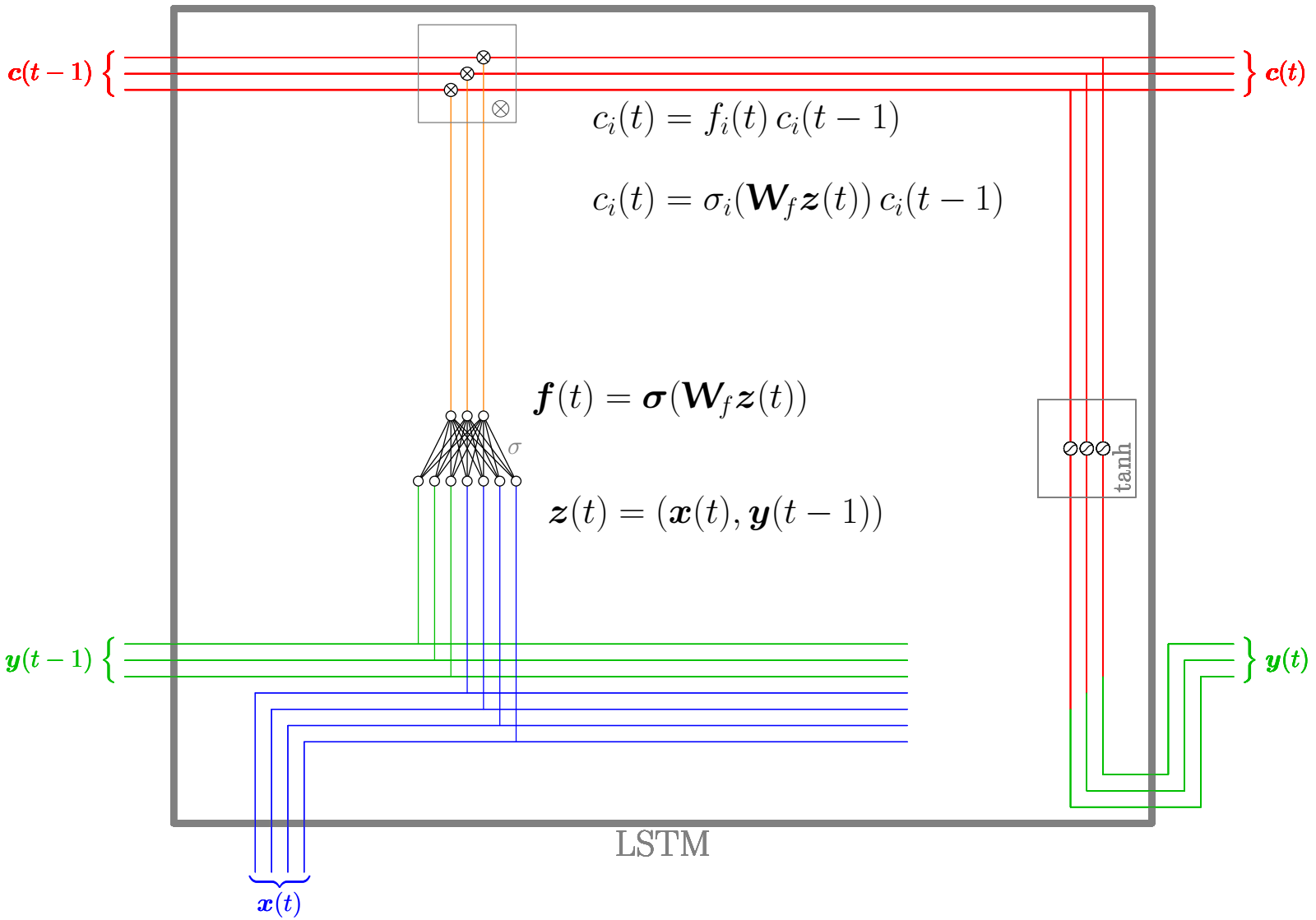
LSTM Architecture



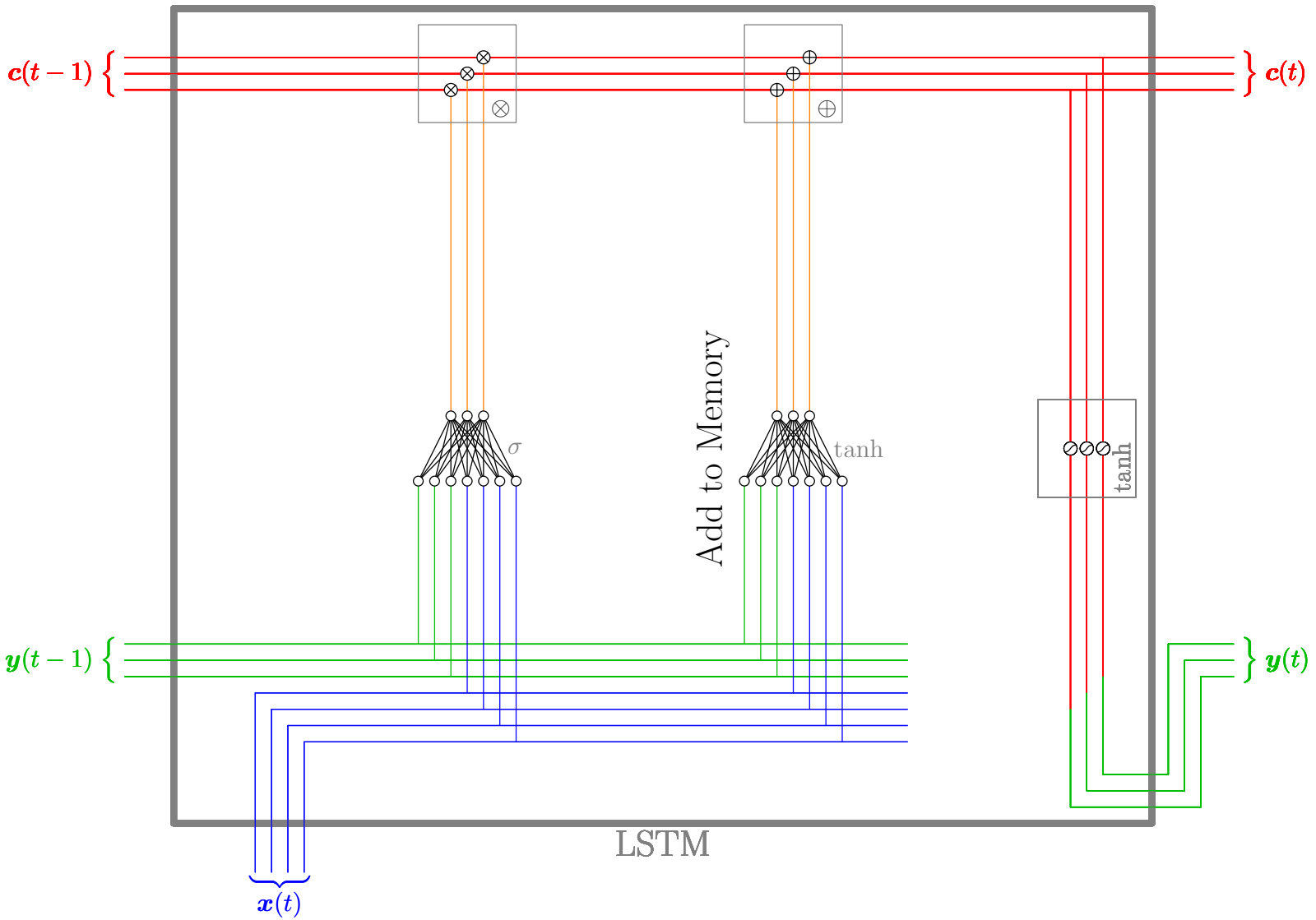
LSTM Architecture



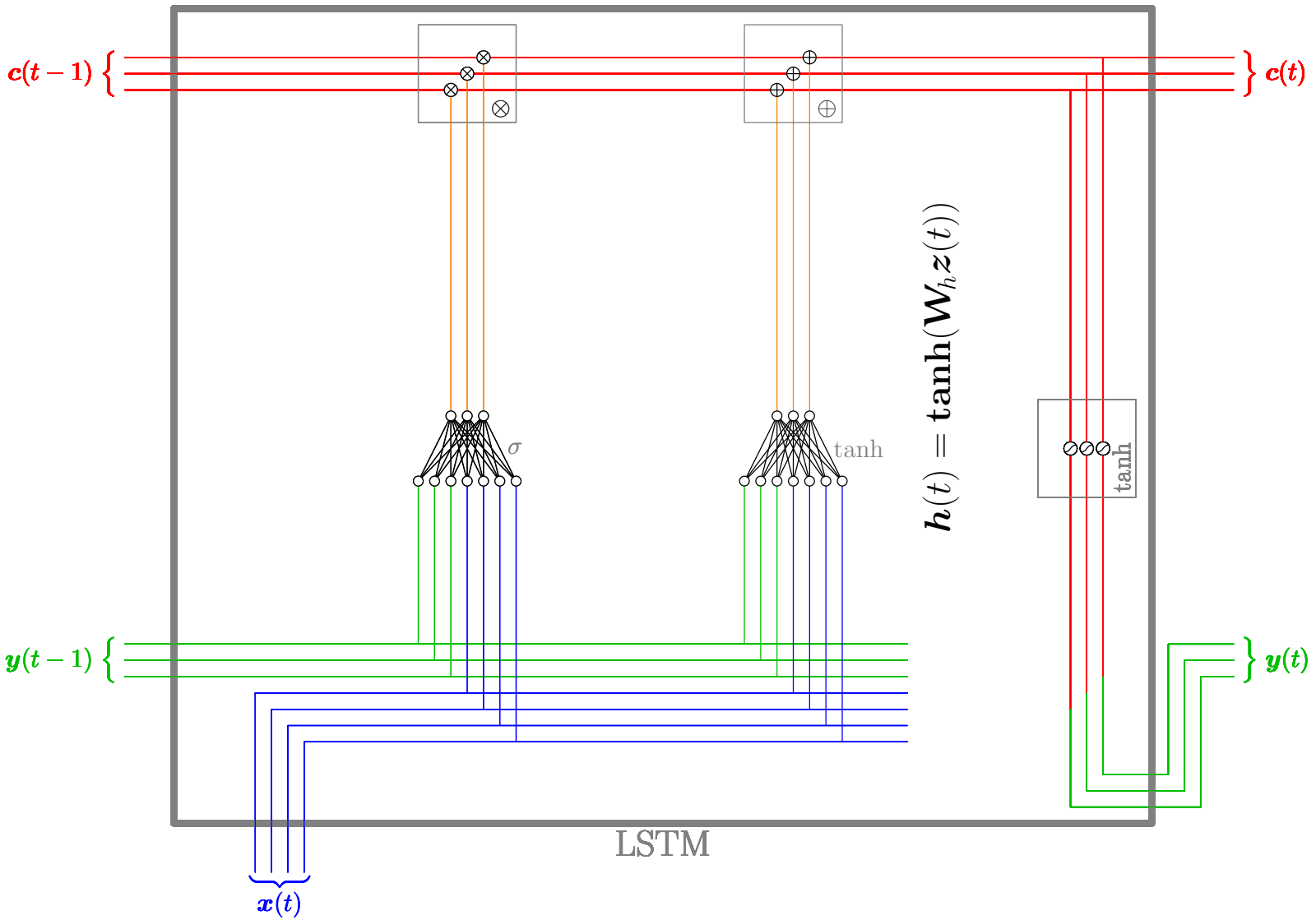
LSTM Architecture



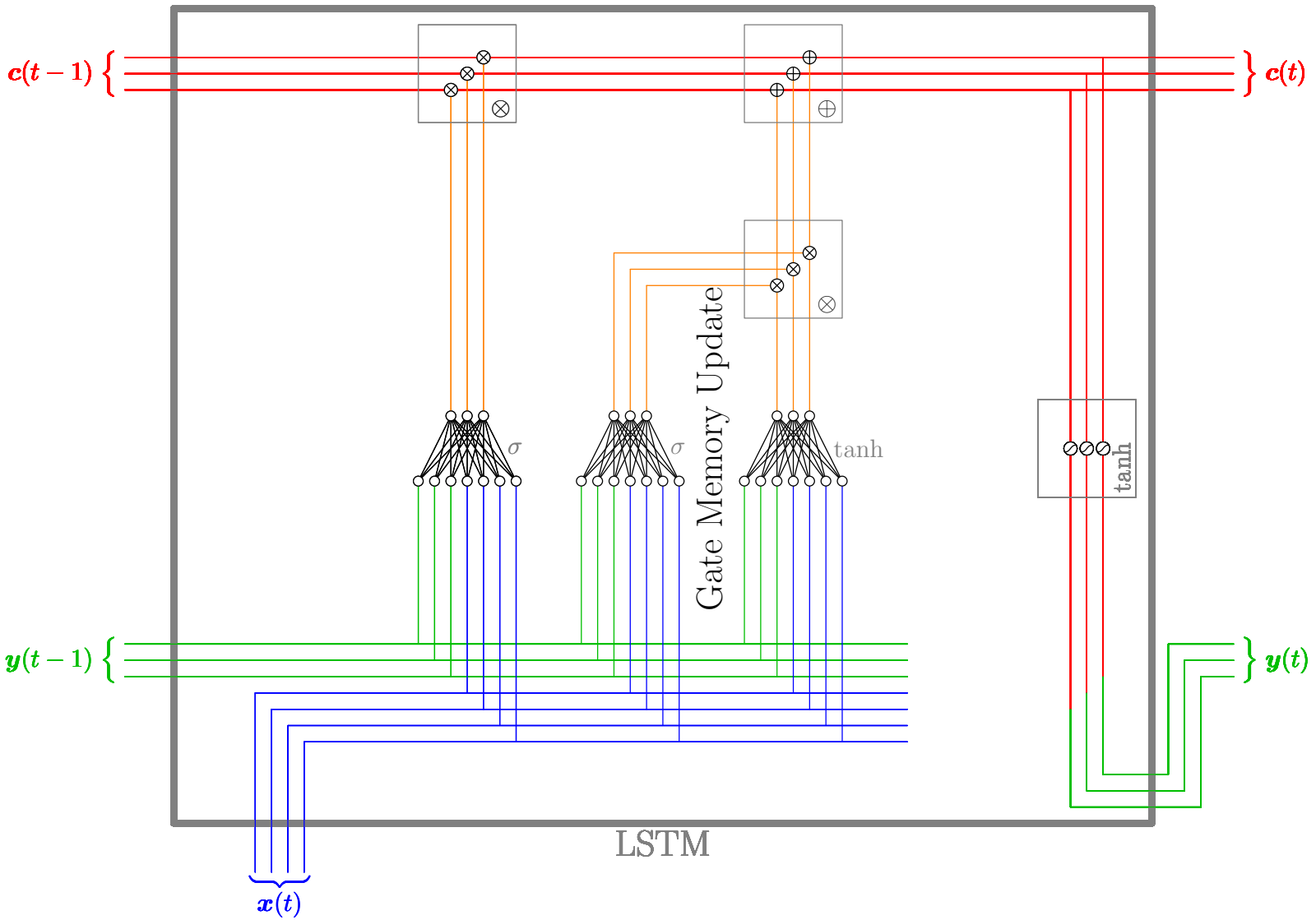
LSTM Architecture



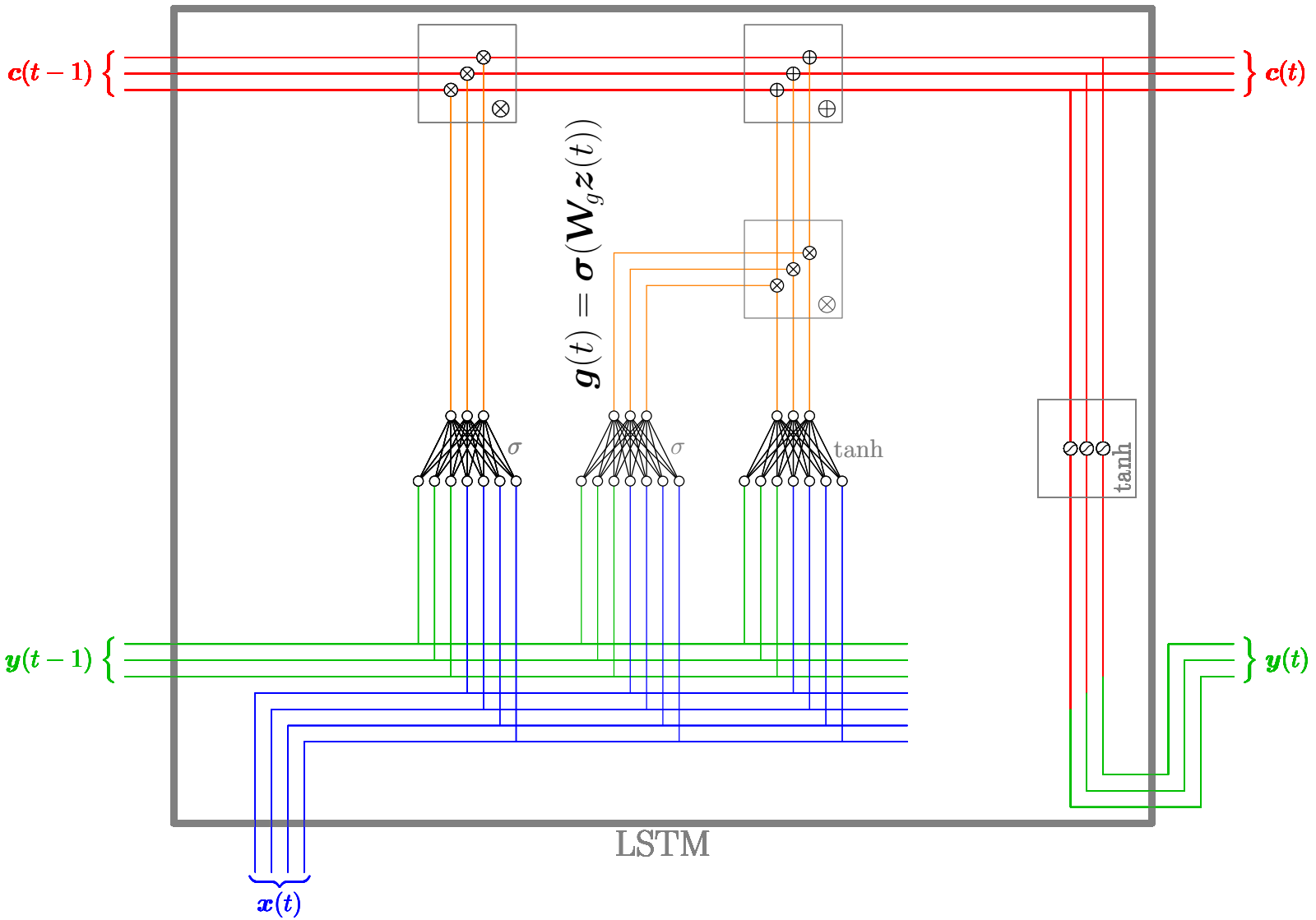
LSTM Architecture



LSTM Architecture

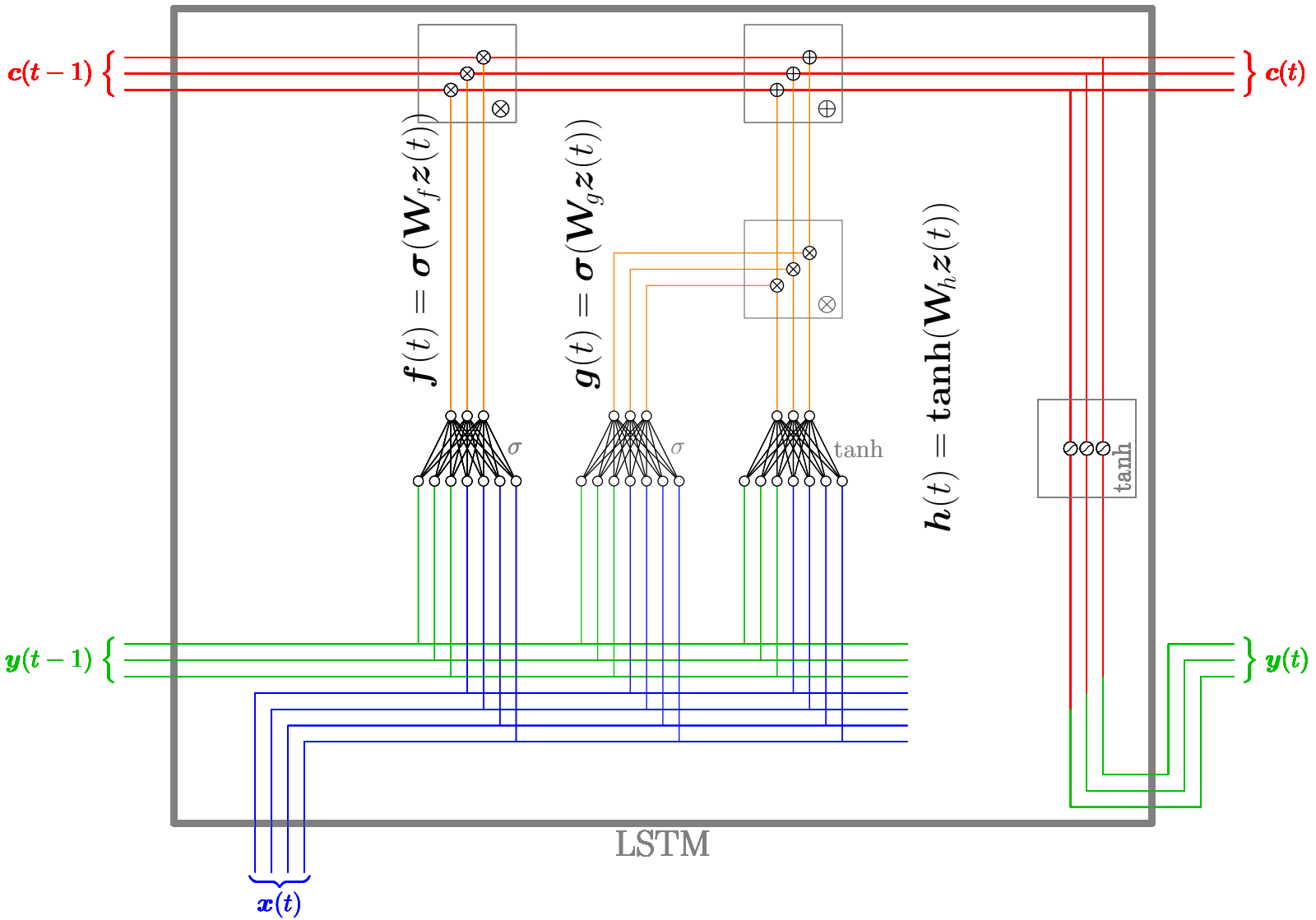


LSTM Architecture

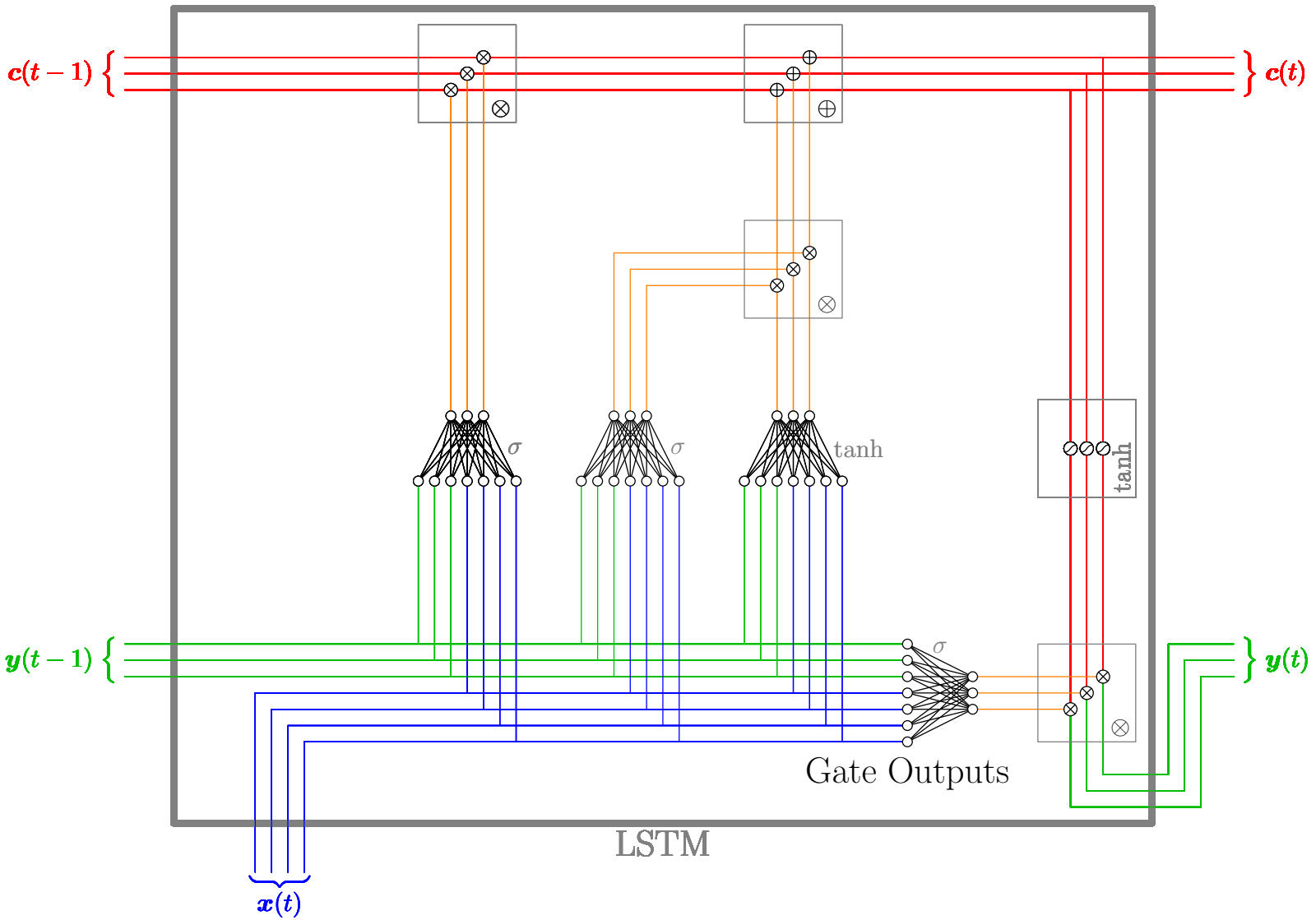


LSTM Architecture

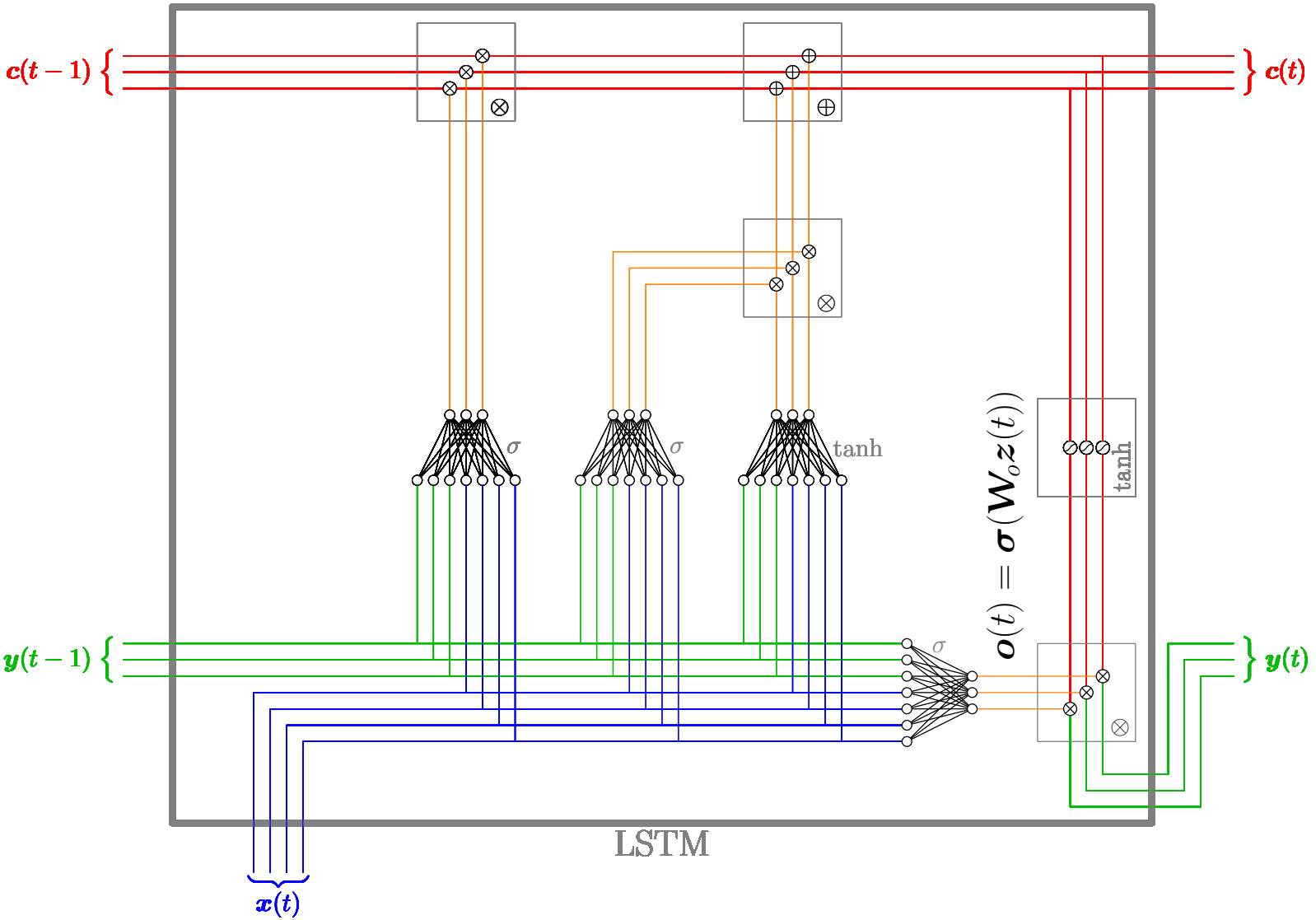
$$c_i(t) = f_i(t)\, c_i(t-1) + g_i(t)\, h_i(t)$$



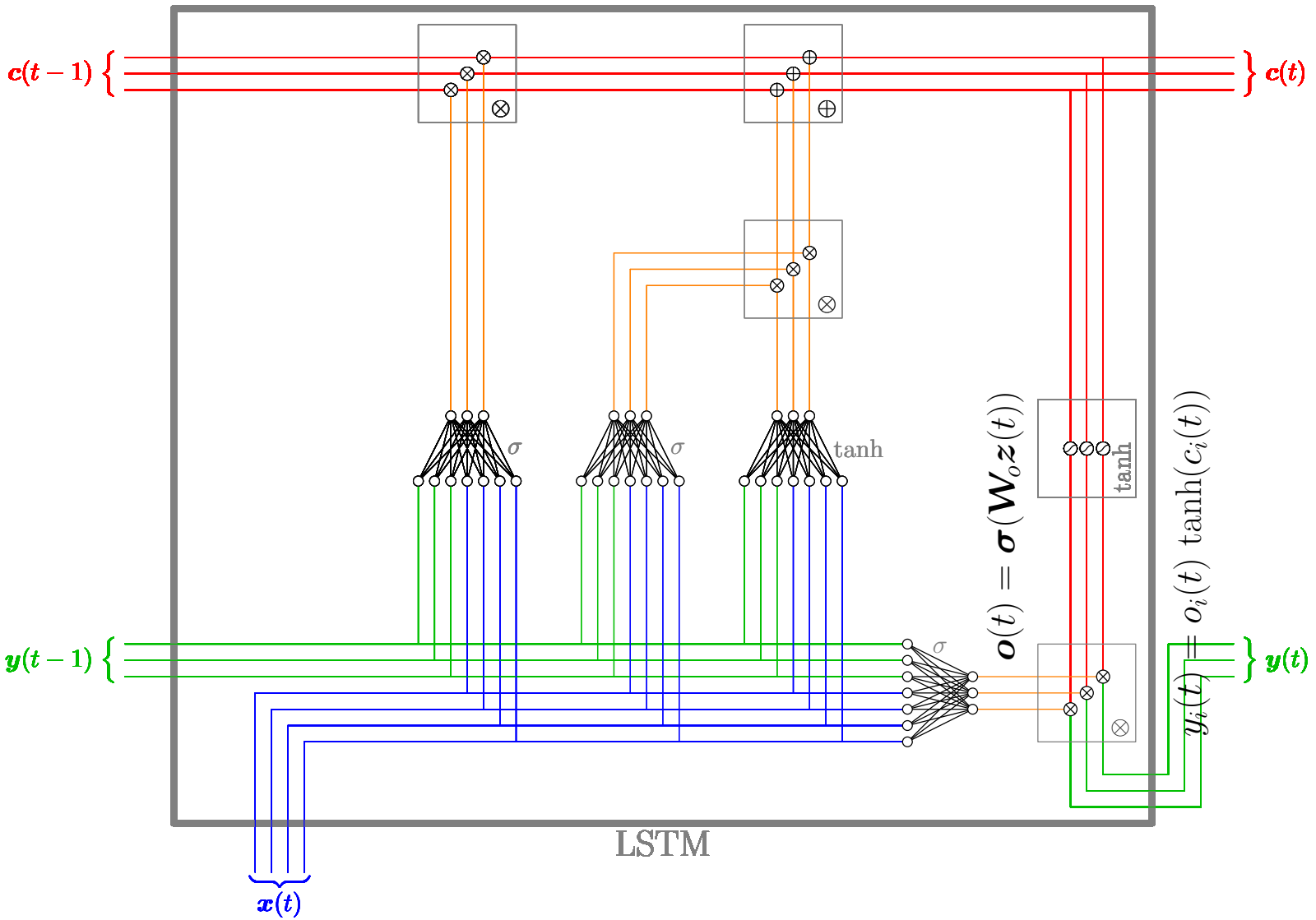
LSTM Architecture



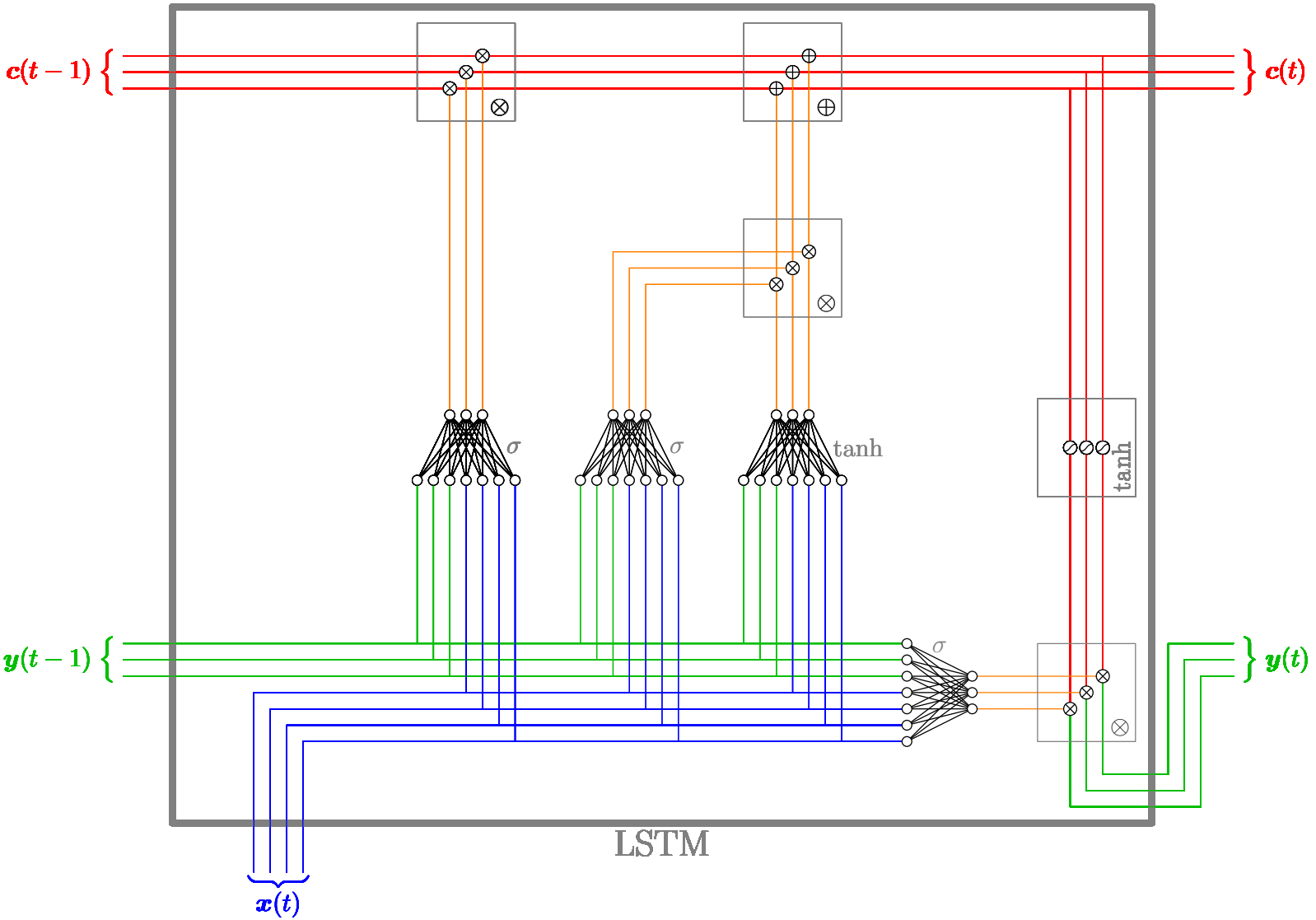
LSTM Architecture



LSTM Architecture



LSTM Architecture



Update Equations

- Inputs $\boldsymbol{z}(t) = \big(\boldsymbol{x}(t), \boldsymbol{y}(t-1)\big)$
- Network updates ($\boldsymbol{W}_*$ are the free parameters)

$$\boldsymbol{f}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_f\, \boldsymbol{z}(t)) \qquad \boldsymbol{g}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_g\, \boldsymbol{z}(t))$$
$$\boldsymbol{h}(t) = \mathbf{tanh}(\boldsymbol{W}_h\, \boldsymbol{z}(t)) \qquad \boldsymbol{o}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_o\, \boldsymbol{z}(t))$$

- Long-term memory update

$$\boldsymbol{c}(t) = \boldsymbol{f}(t) \otimes \boldsymbol{c}(t-1) + \boldsymbol{g}(t) \otimes \boldsymbol{h}(t)$$

- Output $\boldsymbol{y}(t) = \boldsymbol{o}(t) \otimes \mathbf{tanh}(\boldsymbol{c}(t))$

Update Equations

- Inputs $\boldsymbol{z}(t) = \big(\boldsymbol{x}(t), \boldsymbol{y}(t-1)\big)$
- Network updates ($\boldsymbol{W}_*$ are the free parameters)

$$\boldsymbol{f}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_f\, \boldsymbol{z}(t)) \qquad \boldsymbol{g}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_g\, \boldsymbol{z}(t))$$
$$\boldsymbol{h}(t) = \mathbf{tanh}(\boldsymbol{W}_h\, \boldsymbol{z}(t)) \qquad \boldsymbol{o}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_o\, \boldsymbol{z}(t))$$

- Long-term memory update

$$\boldsymbol{c}(t) = \boldsymbol{f}(t) \otimes \boldsymbol{c}(t-1) + \boldsymbol{g}(t) \otimes \boldsymbol{h}(t)$$

- Output $\boldsymbol{y}(t) = \boldsymbol{o}(t) \otimes \mathbf{tanh}(\boldsymbol{c}(t))$

Update Equations

- Inputs $\boldsymbol{z}(t) = \big(\boldsymbol{x}(t), \boldsymbol{y}(t-1)\big)$
- Network updates ($\boldsymbol{W}_*$ are the free parameters)

$$\boldsymbol{f}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_f\, \boldsymbol{z}(t)) \qquad \boldsymbol{g}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_g\, \boldsymbol{z}(t))$$
$$\boldsymbol{h}(t) = \mathbf{tanh}(\boldsymbol{W}_h\, \boldsymbol{z}(t)) \qquad \boldsymbol{o}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_o\, \boldsymbol{z}(t))$$

- Long-term memory update

$$\boldsymbol{c}(t) = \boldsymbol{f}(t) \otimes \boldsymbol{c}(t-1) + \boldsymbol{g}(t) \otimes \boldsymbol{h}(t)$$

- Output $\boldsymbol{y}(t) = \boldsymbol{o}(t) \otimes \mathbf{tanh}(\boldsymbol{c}(t))$

Update Equations

- Inputs $\boldsymbol{z}(t) = \big(\boldsymbol{x}(t), \boldsymbol{y}(t-1)\big)$
- Network updates ($\boldsymbol{W}_*$ are the free parameters)

$$\boldsymbol{f}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_f\, \boldsymbol{z}(t)) \qquad \boldsymbol{g}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_g\, \boldsymbol{z}(t))$$
$$\boldsymbol{h}(t) = \mathbf{tanh}(\boldsymbol{W}_h\, \boldsymbol{z}(t)) \qquad \boldsymbol{o}(t) = \boldsymbol{\sigma}(\boldsymbol{W}_o\, \boldsymbol{z}(t))$$

- Long-term memory update

$$\boldsymbol{c}(t) = \boldsymbol{f}(t) \otimes \boldsymbol{c}(t-1) + \boldsymbol{g}(t) \otimes \boldsymbol{h}(t)$$

- Output $\boldsymbol{y}(t) = \boldsymbol{o}(t) \otimes \mathbf{tanh}(\boldsymbol{c}(t))$

Training LSTM

- We can train an LSTM by unwrapping it in time
- Note that it involves four dense layers with sigmoidal (or tanh) outputs
- This means that typically it is very slow to train
- There are a few variants of LSTMs, but all are very similar. The most popular is probably Gated Recurrent Unit (GRU)

Training LSTM

- We can train an LSTM by unwrapping it in time
- Note that it involves four dense layers with sigmoidal (or tanh) outputs
- This means that typically it is very slow to train
- There are a few variants of LSTMs, but all are very similar. The most popular is probably Gated Recurrent Unit (GRU)

Training LSTM

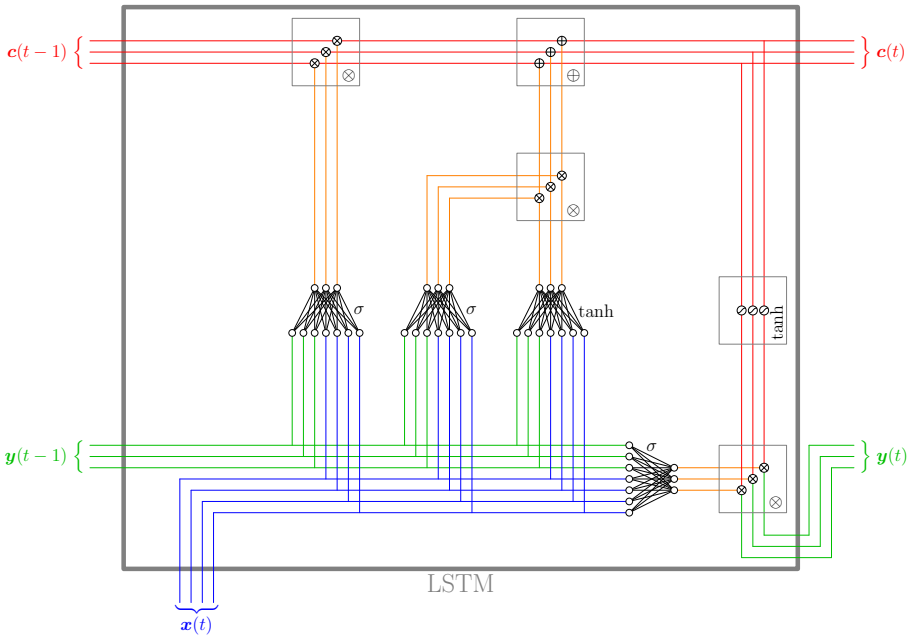
- We can train an LSTM by unwrapping it in time
- Note that it involves four dense layers with sigmoidal (or tanh) outputs
- This means that typically it is very slow to train
- There are a few variants of LSTMs, but all are very similar. The most popular is probably Gated Recurrent Unit (GRU)

Training LSTM

- We can train an LSTM by unwrapping it in time
- Note that it involves four dense layers with sigmoidal (or tanh) outputs
- This means that typically it is very slow to train
- There are a few variants of LSTMs, but all are very similar. The most popular is probably Gated Recurrent Unit (GRU)

Outline

1. More on CNNs
2. Recurrent Networks
3. LSTM
4. **Word Embedding**
5. Applications



Working with Words

- Words often need some preprocessing (removing punctuation, removing stop words, stemming, etc)—depends on application
- Usually use a 1-hot vector representation of words
- Document can be represented by a **Bag of Words** representation which is a weighted sum of the 1-hot vectors
- Often use TF-IDF weighting of words and clustering (codebooks) to reduce dimensionality

Working with Words

- Words often need some preprocessing (removing punctuation, removing stop words, stemming, etc)—depends on application
- Usually use a 1-hot vector representation of words
- Document can be represented by a **Bag of Words** representation which is a weighted sum of the 1-hot vectors
- Often use TF-IDF weighting of words and clustering (codebooks) to reduce dimensionality

Working with Words

- Words often need some preprocessing (removing punctuation, removing stop words, stemming, etc)—depends on application
- Usually use a 1-hot vector representation of words
- Document can be represented by a **Bag of Words** representation which is a weighted sum of the 1-hot vectors
- Often use TF-IDF weighting of words and clustering (codebooks) to reduce dimensionality

Working with Words

- Words often need some preprocessing (removing punctuation, removing stop words, stemming, etc)—depends on application
- Usually use a 1-hot vector representation of words

$$\text{deep} = (0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0)$$

a
aardvark
abacus
acrobat
Adam
...
deep
...
zebra

- Document can be represented by a **Bag of Words** representation which is a weighted sum of the 1-hot vectors
- Often use TF-IDF weighting of words and clustering (codebooks) to reduce dimensionality

Working with Words

- Words often need some preprocessing (removing punctuation, removing stop words, stemming, etc)—depends on application
- Usually use a 1-hot vector representation of words

$$\text{acrobat} = (0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

a
aardvark
abacus
acrobat
Adam
...
deep
...
zebra

- Document can be represented by a **Bag of Words** representation which is a weighted sum of the 1-hot vectors
- Often use TF-IDF weighting of words and clustering (codebooks) to reduce dimensionality

Working with Words

- Words often need some preprocessing (removing punctuation, removing stop words, stemming, etc)—depends on application
- Usually use a 1-hot vector representation of words

$$\text{aardvark} = (0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

a
aardvark
abacus
acrobat
Adam
...
deep
...
zebra

- Document can be represented by a **Bag of Words** representation which is a weighted sum of the 1-hot vectors
- Often use TF-IDF weighting of words and clustering (codebooks) to reduce dimensionality

Working with Words

- Words often need some preprocessing (removing punctuation, removing stop words, stemming, etc)—depends on application
- Usually use a 1-hot vector representation of words

$$\text{aardvark} = (0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

a
aardvark
abacus
acrobat
Adam
deep
zebra

- Document can be represented by a **Bag of Words** representation which is a weighted sum of the 1-hot vectors
- Often use TF-IDF weighting of words and clustering (codebooks) to reduce dimensionality

Working with Words

- Words often need some preprocessing (removing punctuation, removing stop words, stemming, etc)—depends on application
- Usually use a 1-hot vector representation of words

$$\text{aardvark} = (0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

a
aardvark
abacus
acrobat
Adam
...
deep
...
zebra

- Document can be represented by a **Bag of Words** representation which is a weighted sum of the 1-hot vectors
- Often use TF-IDF weighting of words and clustering (codebooks) to reduce dimensionality

Word Embeddings

- For LSTMs or GRUs we need a lower dimensional representation than the 1-hot vectors
- This is usually achieved using a pre-trained word embedding
- These are trained on a large corpus of words by associating words with their context within in a sentence
- A famous example of this is Google's **word2vec**

Word Embeddings

- For LSTMs or GRUs we need a lower dimensional representation than the 1-hot vectors
- This is usually achieved using a pre-trained word embedding
- These are trained on a large corpus of words by associating words with their context within in a sentence
- A famous example of this is Google's **word2vec**

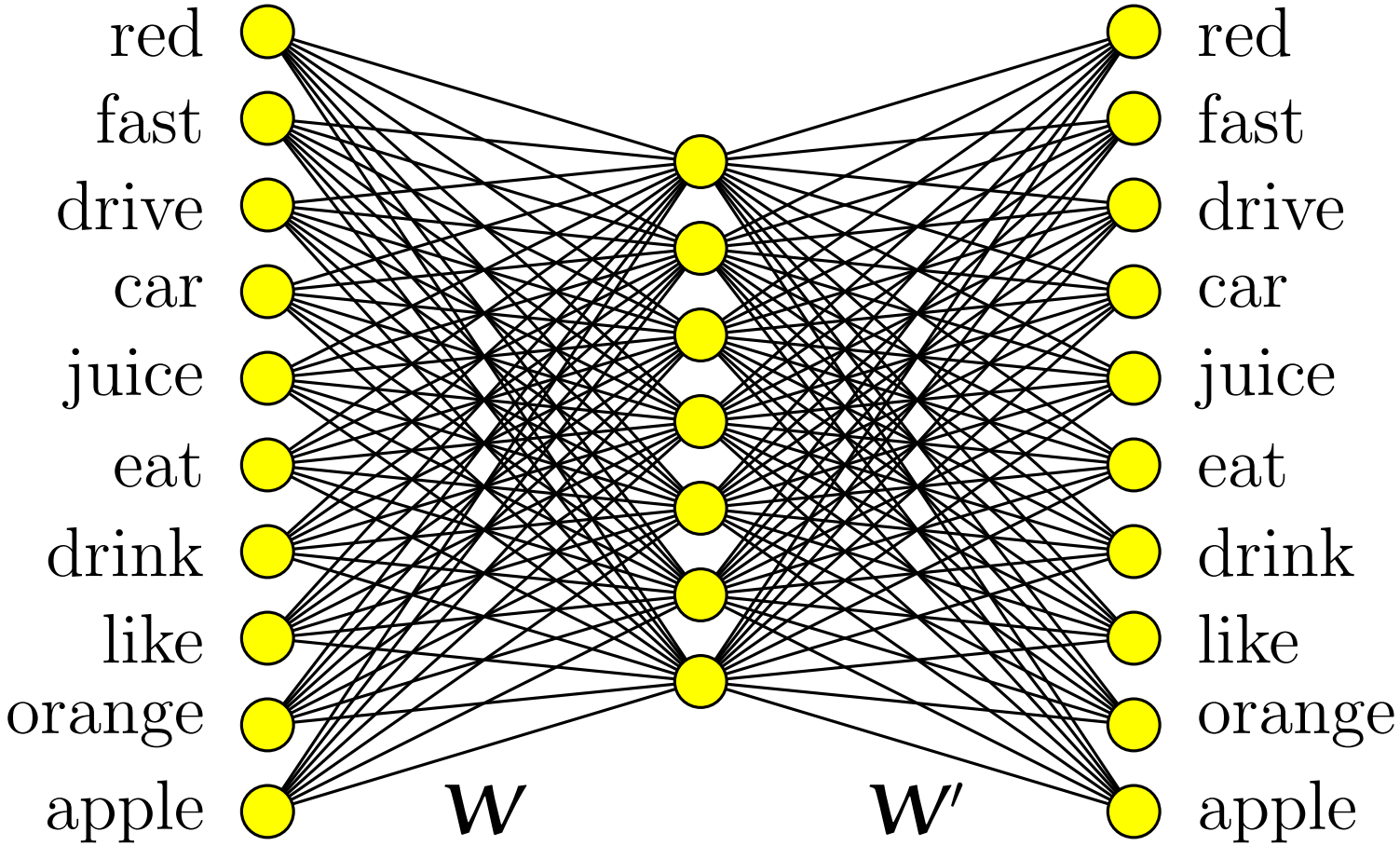
Word Embeddings

- For LSTMs or GRUs we need a lower dimensional representation than the 1-hot vectors
- This is usually achieved using a pre-trained word embedding
- These are trained on a large corpus of words by associating words with their context within in a sentence
- A famous example of this is Google's **word2vec**

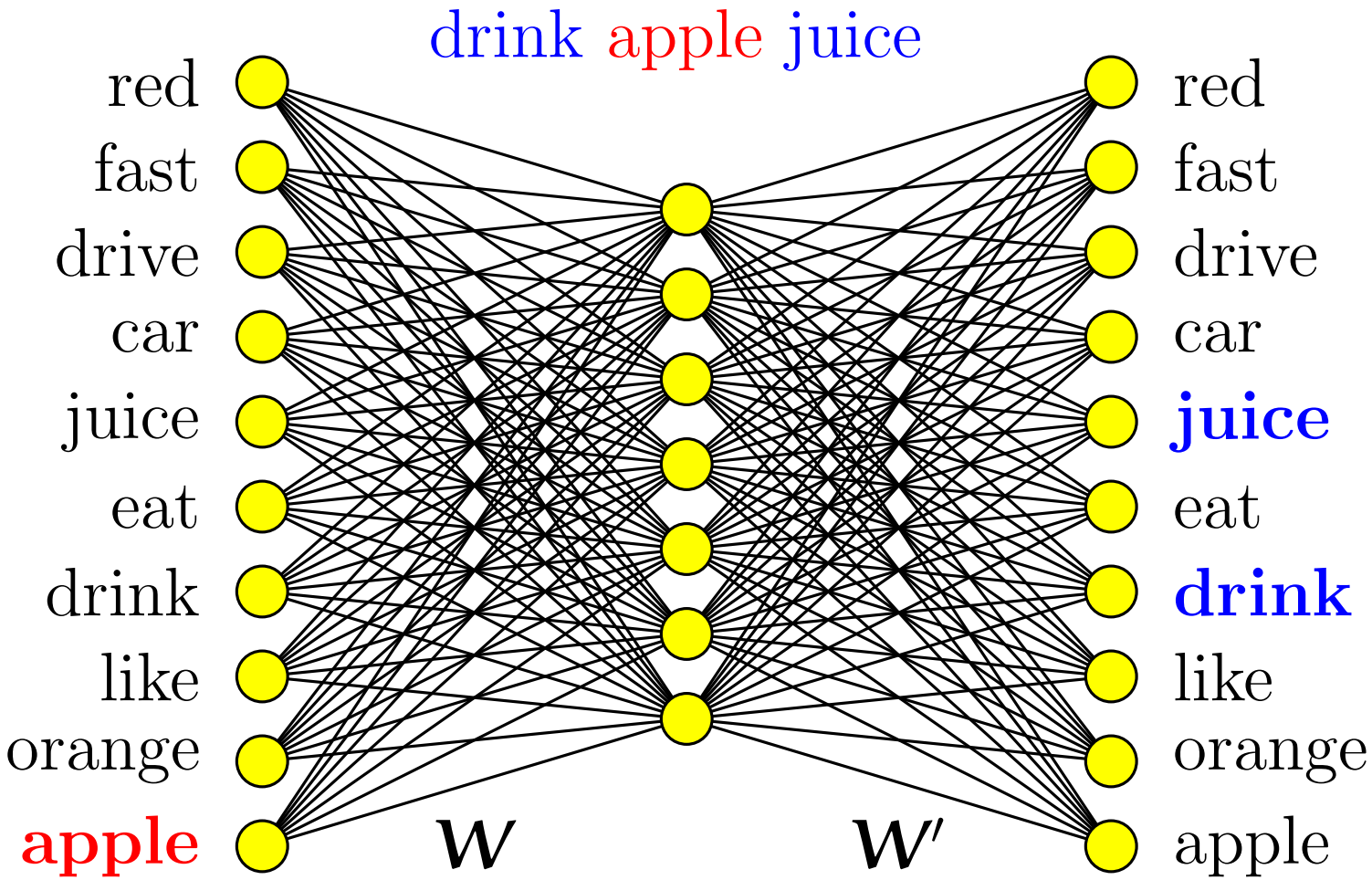
Word Embeddings

- For LSTMs or GRUs we need a lower dimensional representation than the 1-hot vectors
- This is usually achieved using a pre-trained word embedding
- These are trained on a large corpus of words by associating words with their context within in a sentence
- A famous example of this is Google's **word2vec**

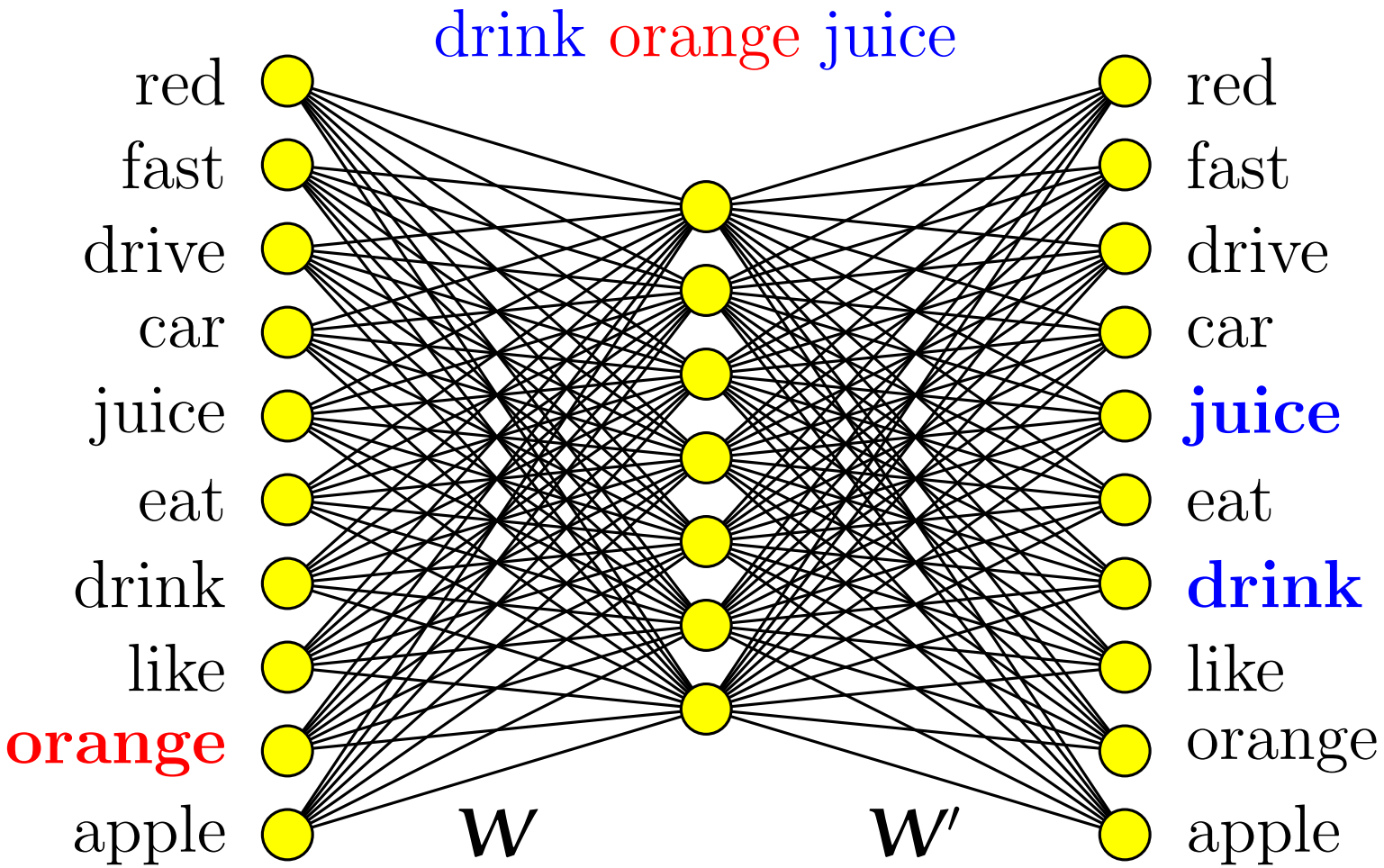
Word2Vec


red
fast
drive
car
juice
eat
drink
like
orange
apple
W
W′
red
fast
drive
car
juice
eat
drink
like
orange
apple

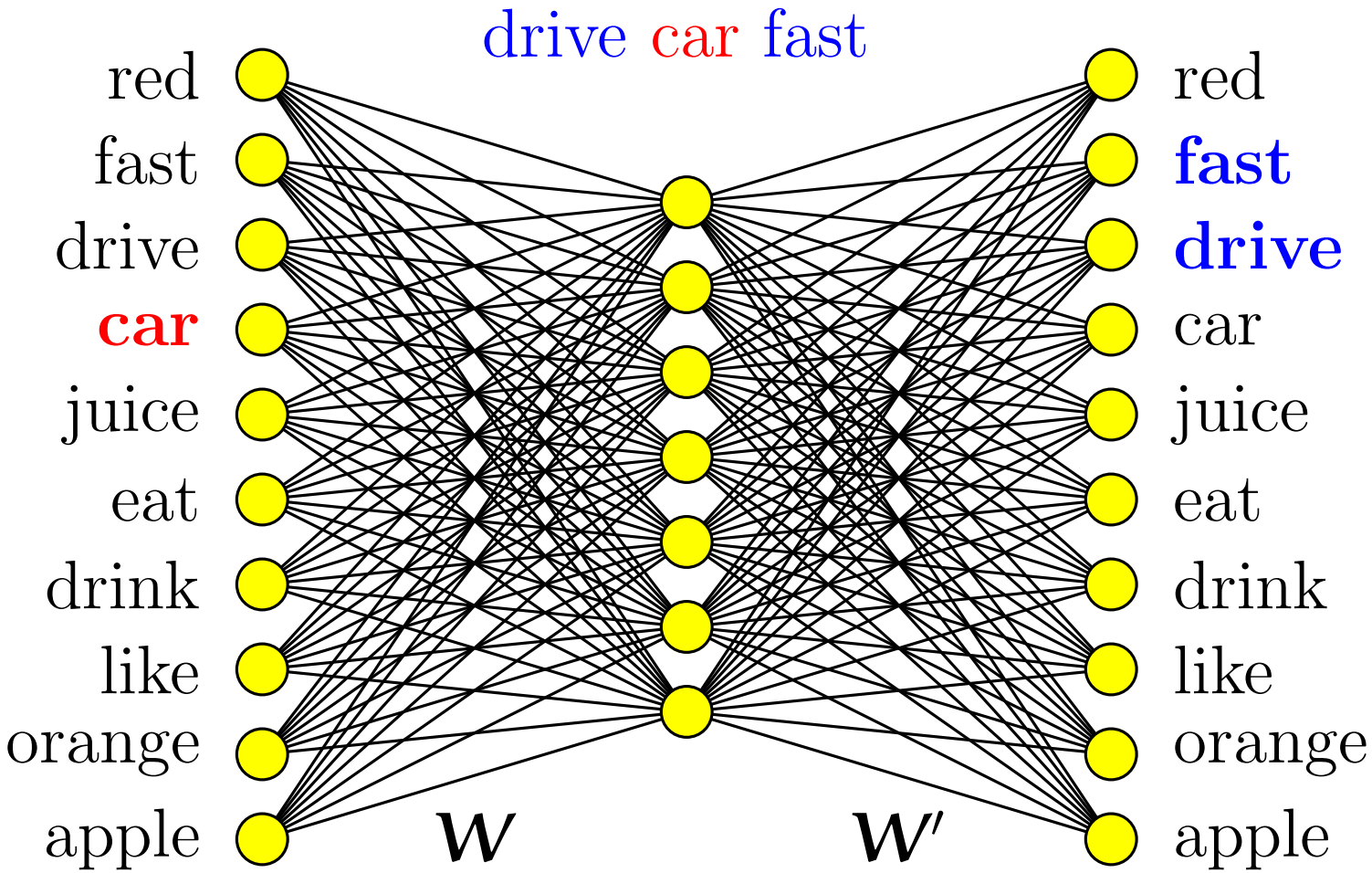
Word2Vec


drink apple juice
red
fast
drive
car
juice
eat
drink
like
orange
apple
W
W′
red
fast
drive
car
juice
eat
drink
like
orange
apple

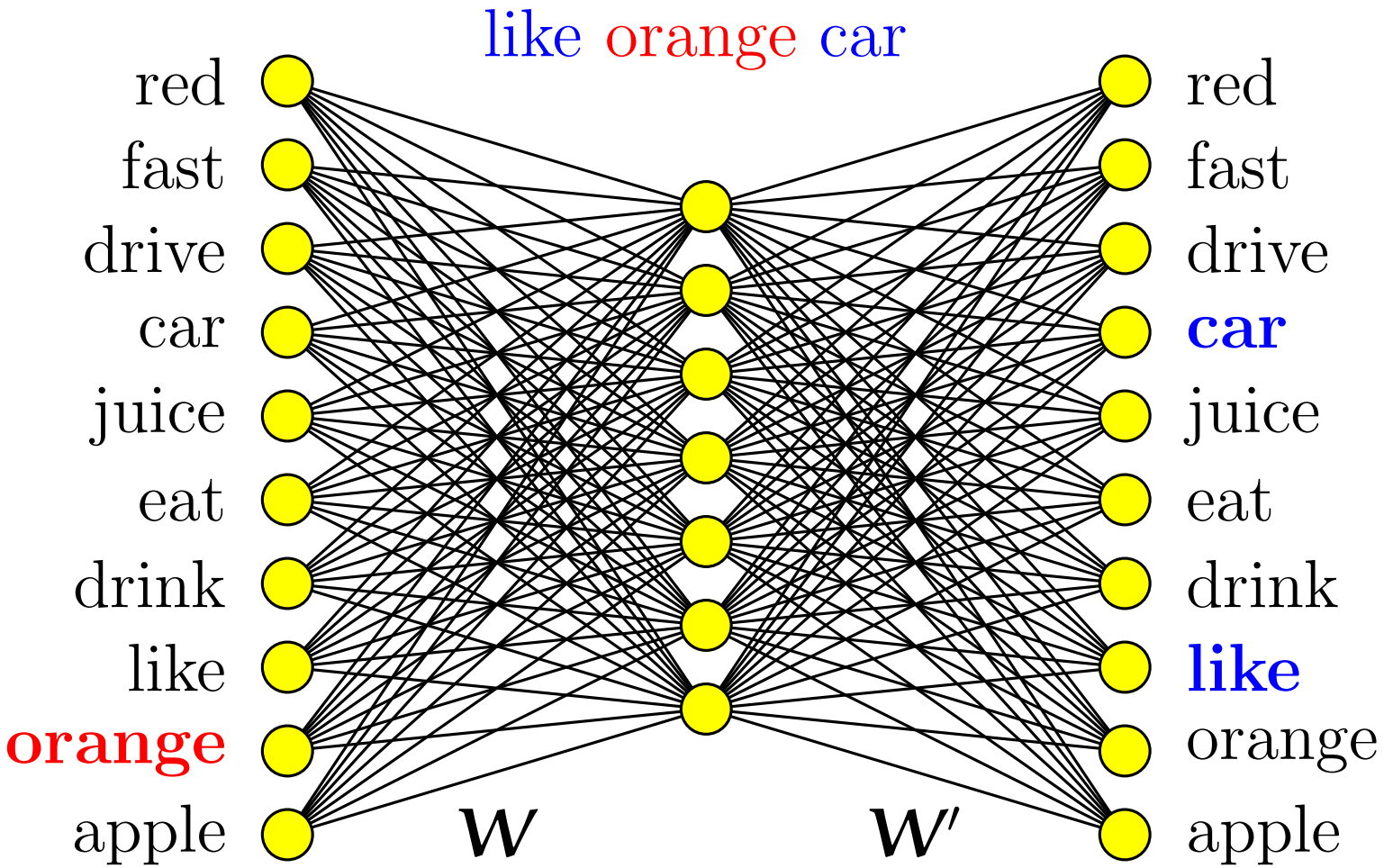
Word2Vec


drink orange juice
red
fast
drive
car
juice
eat
drink
like
orange
apple
W
W′
red
fast
drive
car
juice
eat
drink
like
orange
apple

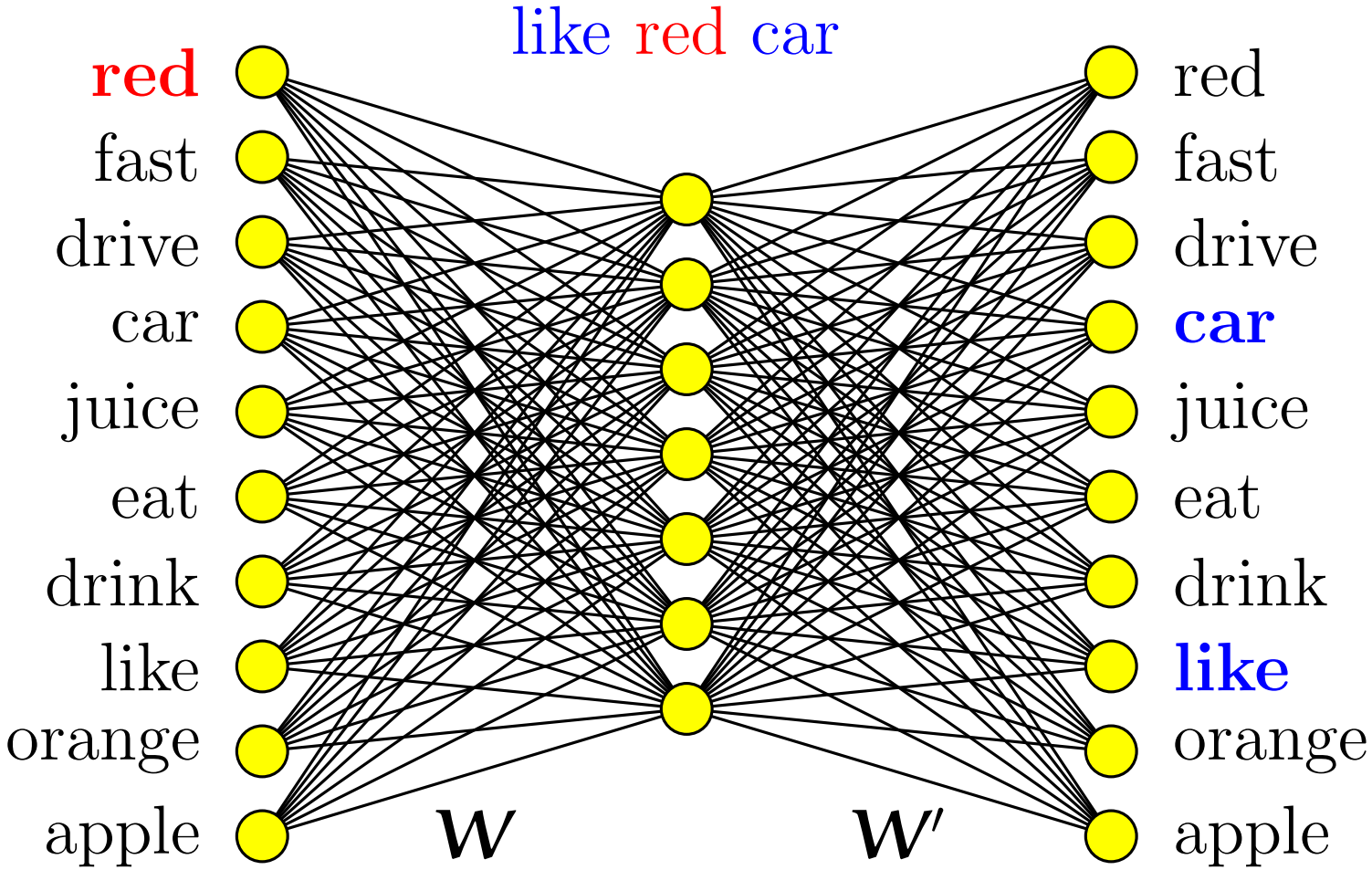
Word2Vec


drive car fast
red
fast
drive
car
juice
eat
drink
like
orange
apple
W
W′
red
fast
drive
car
juice
eat
drink
like
orange
apple

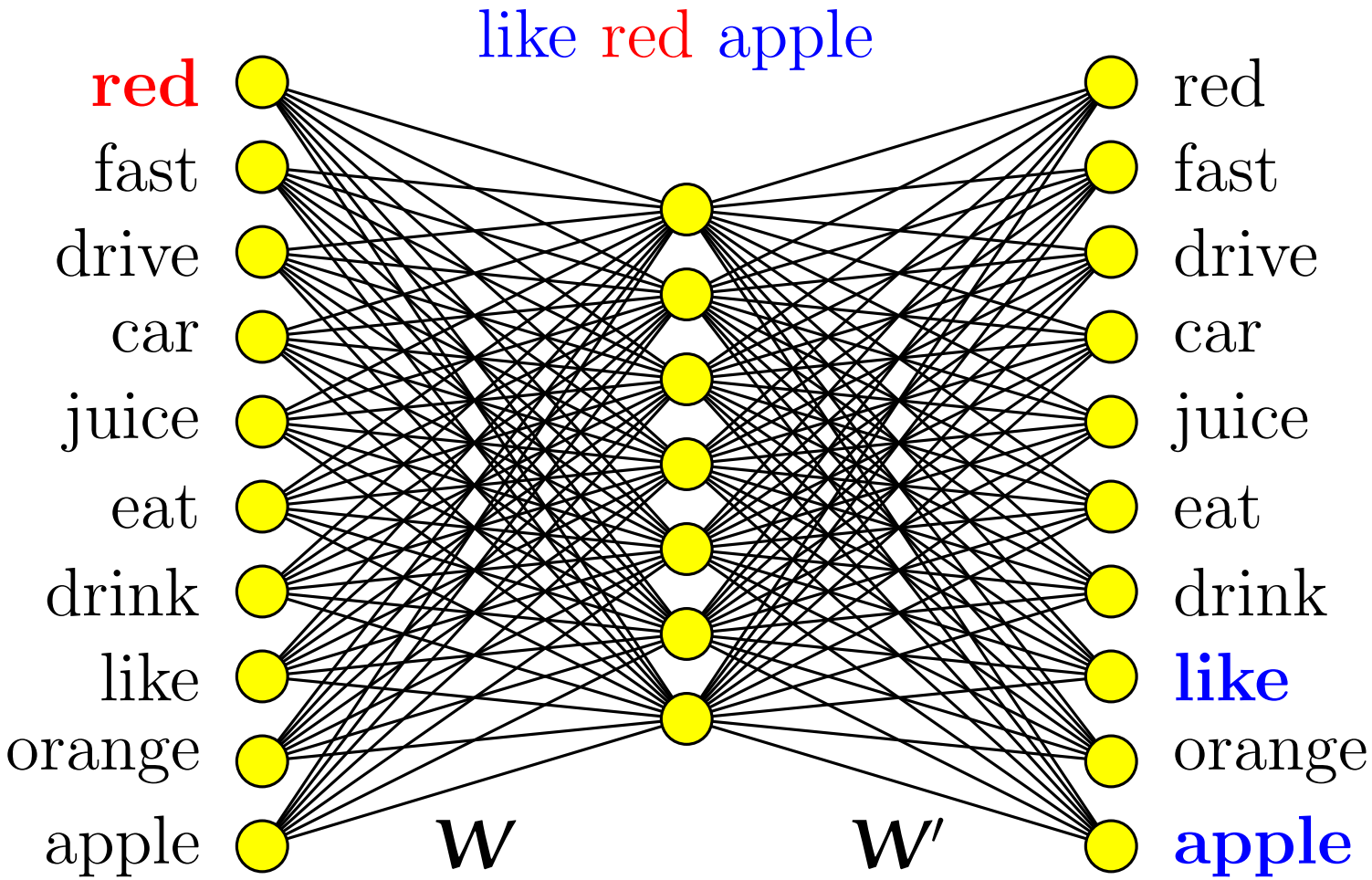
Word2Vec


like orange car
red
fast
drive
car
juice
eat
drink
like
orange
apple
W
W′
red
fast
drive
car
juice
eat
drink
like
orange
apple

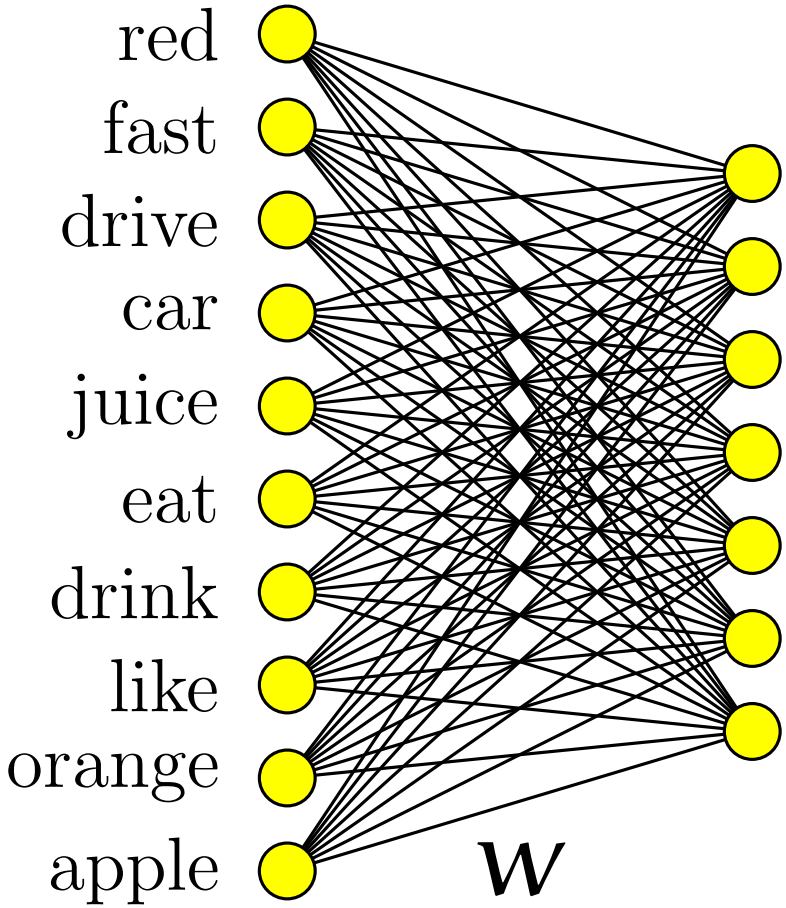
Word2Vec


like red car
red
fast
drive
car
juice
eat
drink
like
orange
apple
W
W′
red
fast
drive
car
juice
eat
drink
like
orange
apple

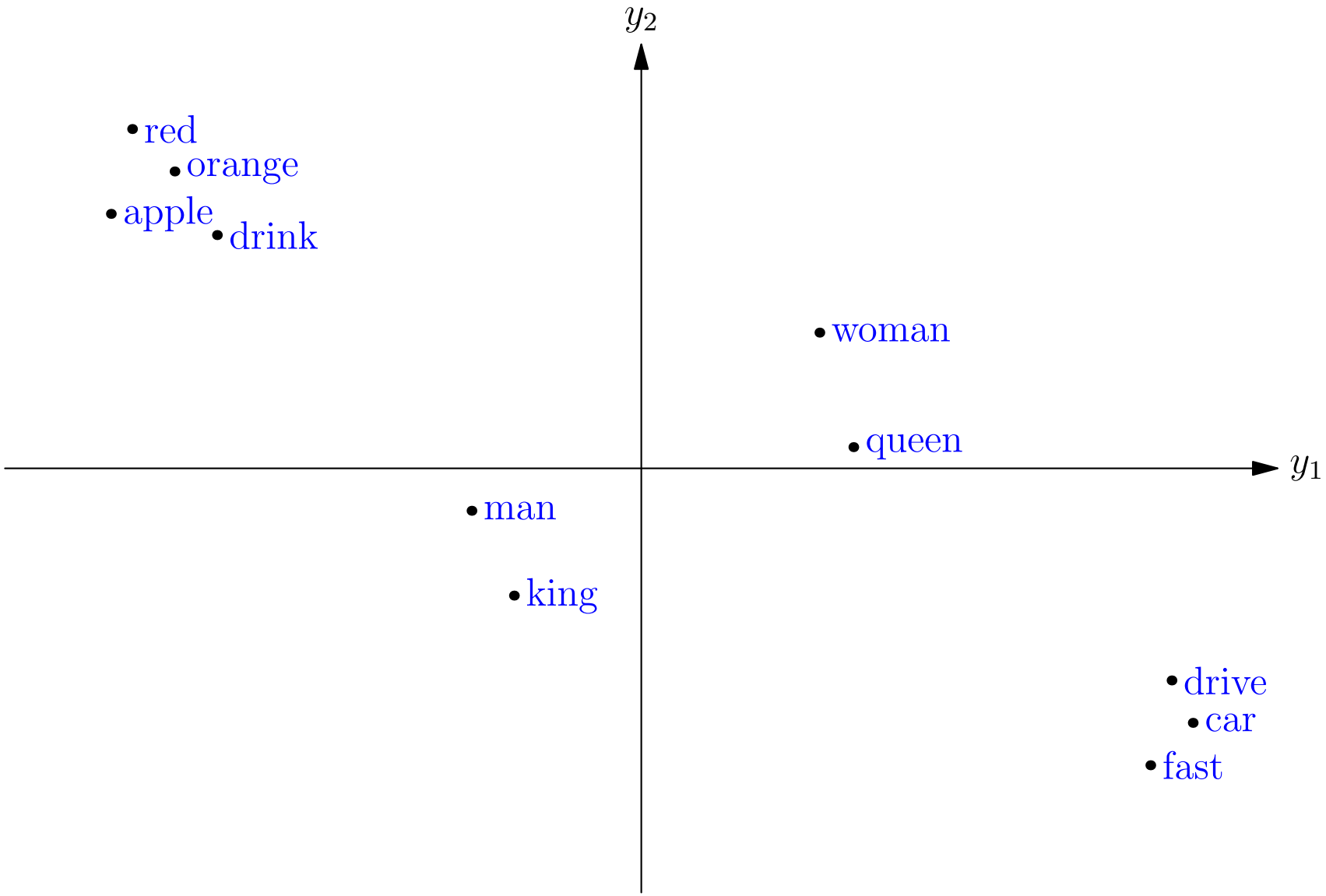
Word2Vec


like red apple
red
fast
drive
car
juice
eat
drink
like
orange
apple
W
W′
red
fast
drive
car
juice
eat
drink
like
orange
apple

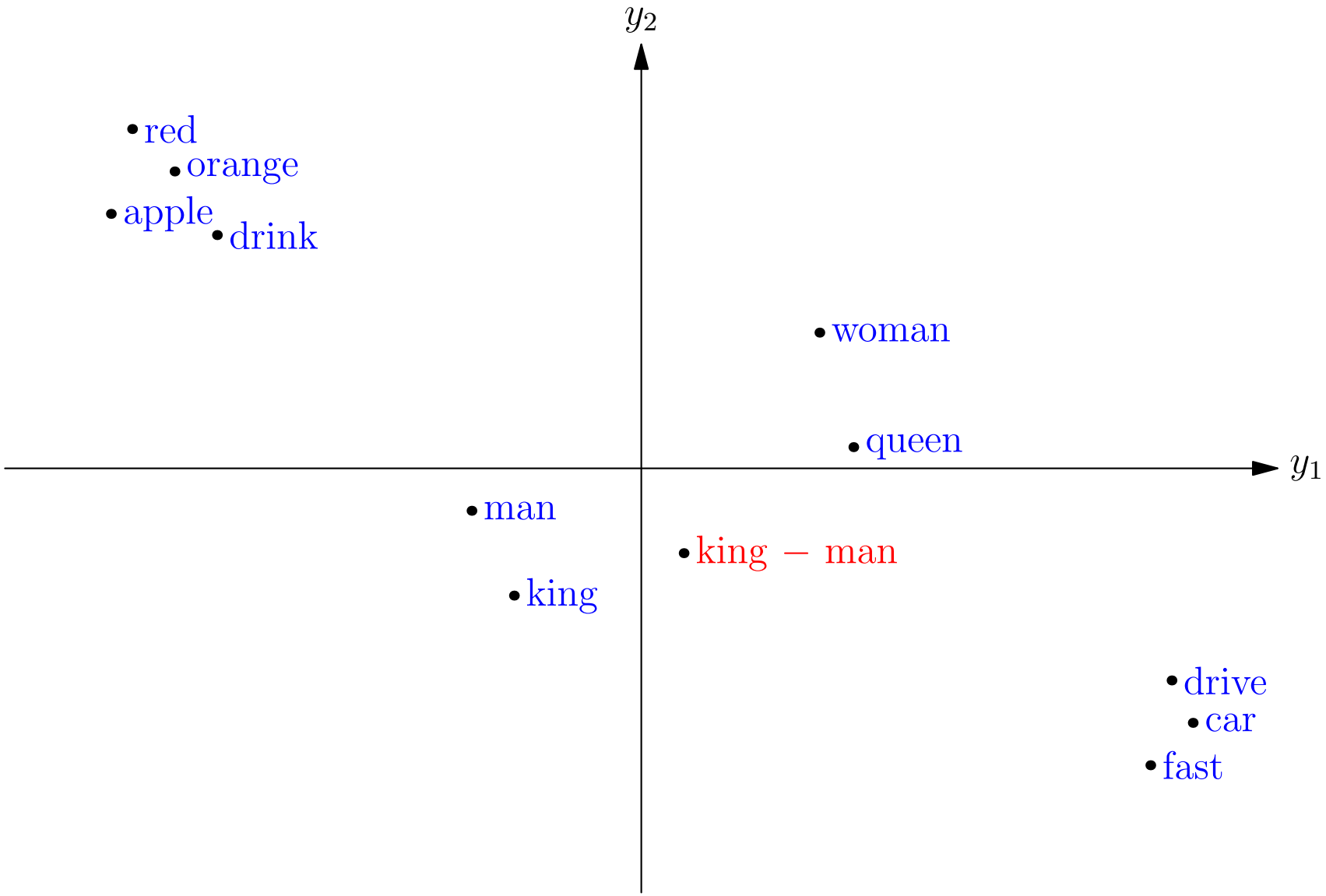
Word2Vec


red
fast
drive
car
juice
eat
drink
like
orange
apple
$\boldsymbol{W}$

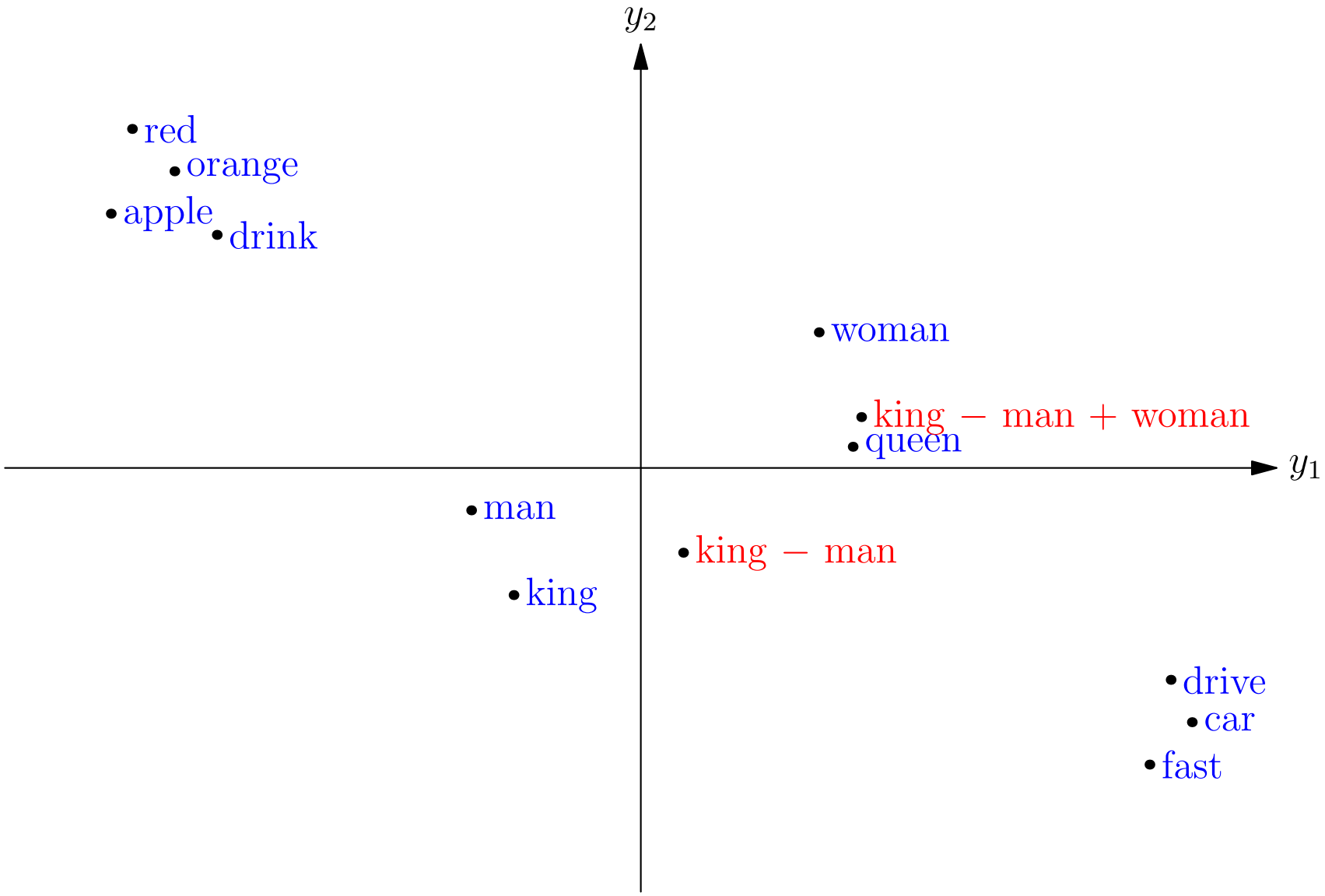
Word Embedding



Word Embedding

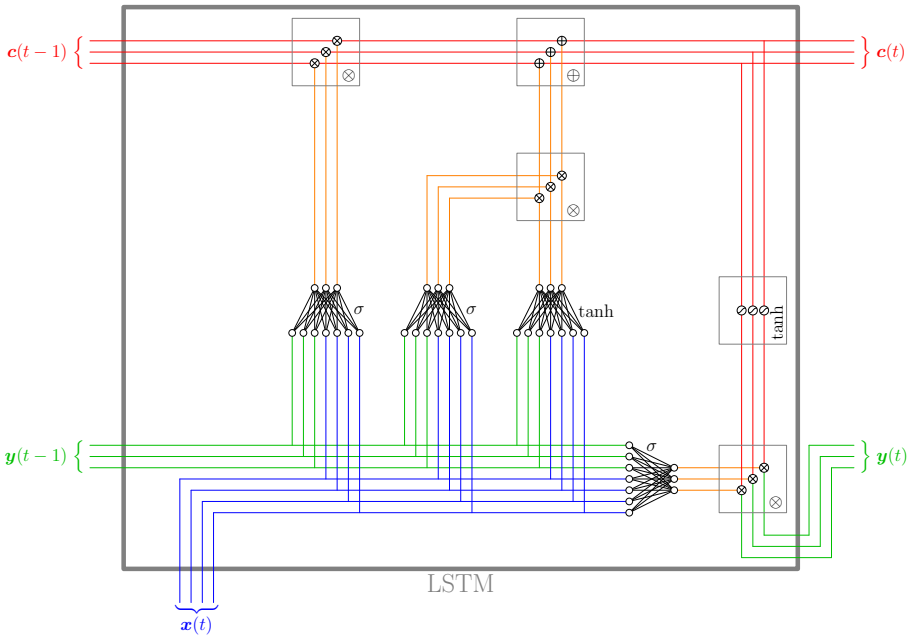


Word Embedding



Outline

1. More on CNNs
2. Recurrent Networks
3. LSTM
4. Word Embedding
5. **Applications**



Sequence-to-Sequence

- An application of LSTMs that have captured peoples attention is a sequence to sequence architecture
- This has been used in various applications such as generating answers to questions, chatbots, etc.
- Here we focus on machine translation from English to French
- Used a database with 12M sentences
- Used 160 000 English words and 80 000 French words

Sequence-to-Sequence

- An application of LSTMs that have captured peoples attention is a sequence to sequence architecture
- This has been used in various applications such as generating answers to questions, chatbots, etc.
- Here we focus on machine translation from English to French
- Used a database with 12M sentences
- Used 160 000 English words and 80 000 French words

Sequence-to-Sequence

- An application of LSTMs that have captured peoples attention is a sequence to sequence architecture
- This has been used in various applications such as generating answers to questions, chatbots, etc.
- Here we focus on machine translation from English to French
- Used a database with 12M sentences
- Used 160 000 English words and 80 000 French words

Sequence-to-Sequence

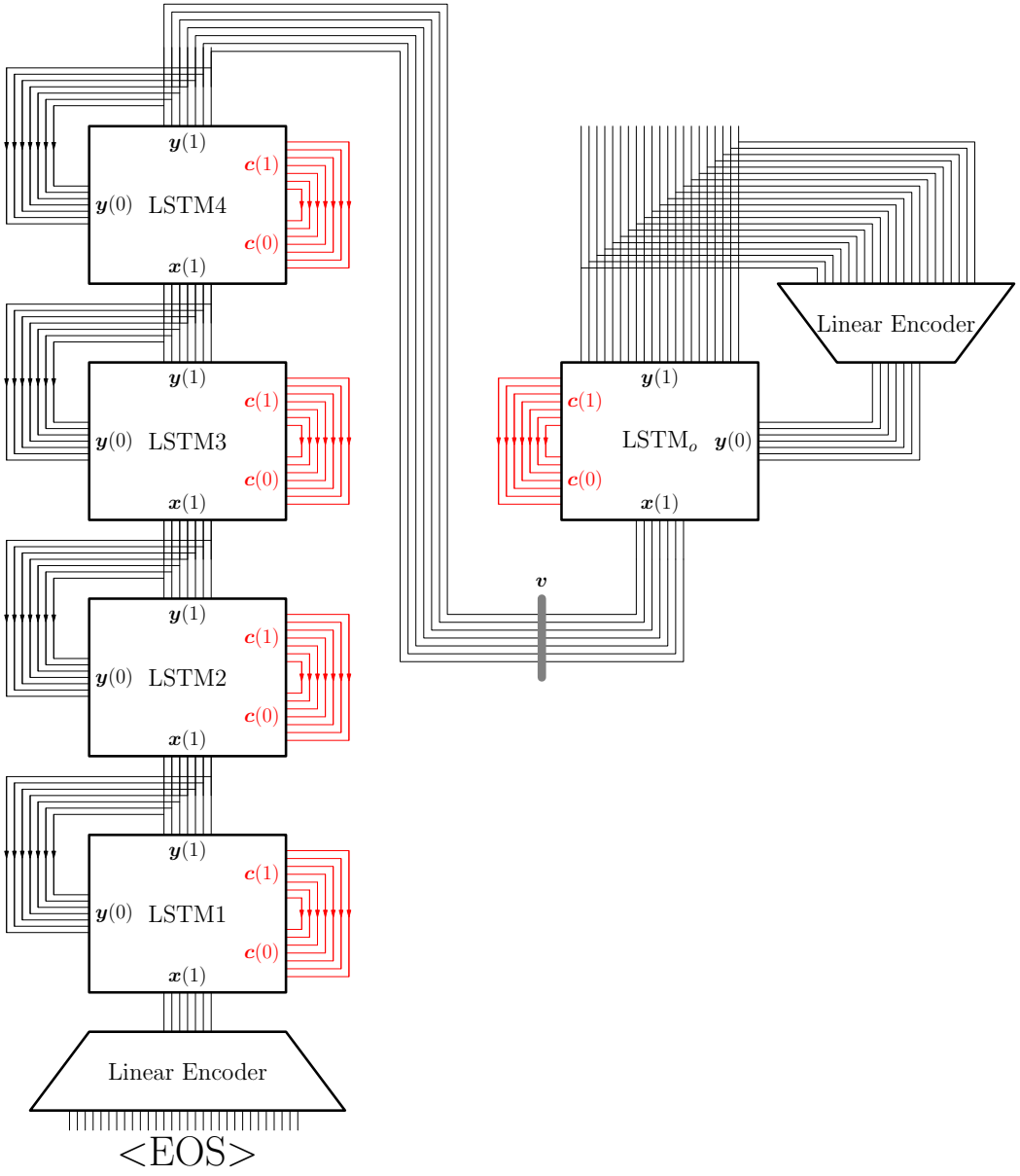
- An application of LSTMs that have captured peoples attention is a sequence to sequence architecture
- This has been used in various applications such as generating answers to questions, chatbots, etc.
- Here we focus on machine translation from English to French
- Used a database with 12M sentences
- Used 160 000 English words and 80 000 French words

Sequence-to-Sequence

- An application of LSTMs that have captured peoples attention is a sequence to sequence architecture
- This has been used in various applications such as generating answers to questions, chatbots, etc.
- Here we focus on machine translation from English to French
- Used a database with 12M sentences
- Used 160 000 English words and 80 000 French words

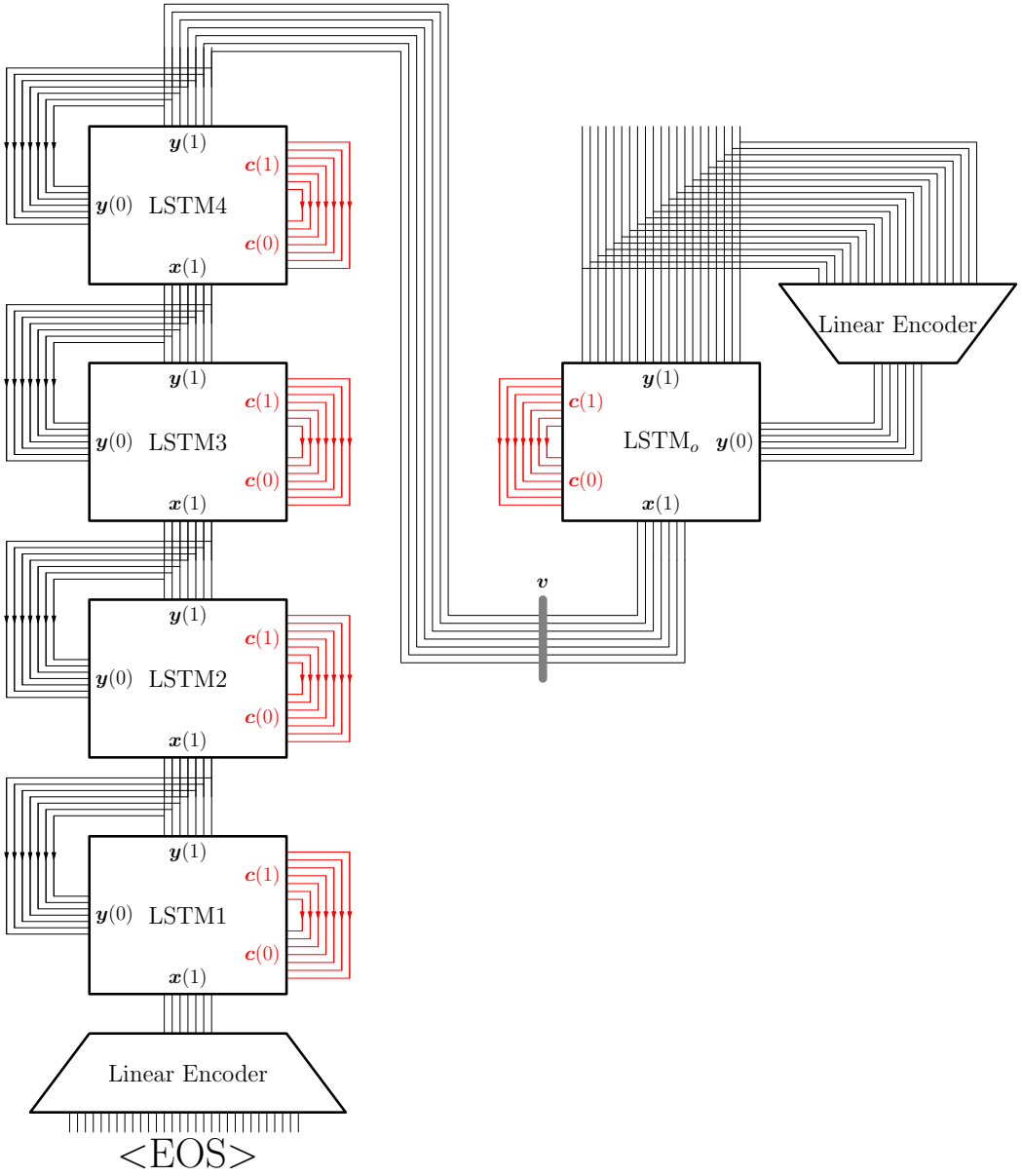
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



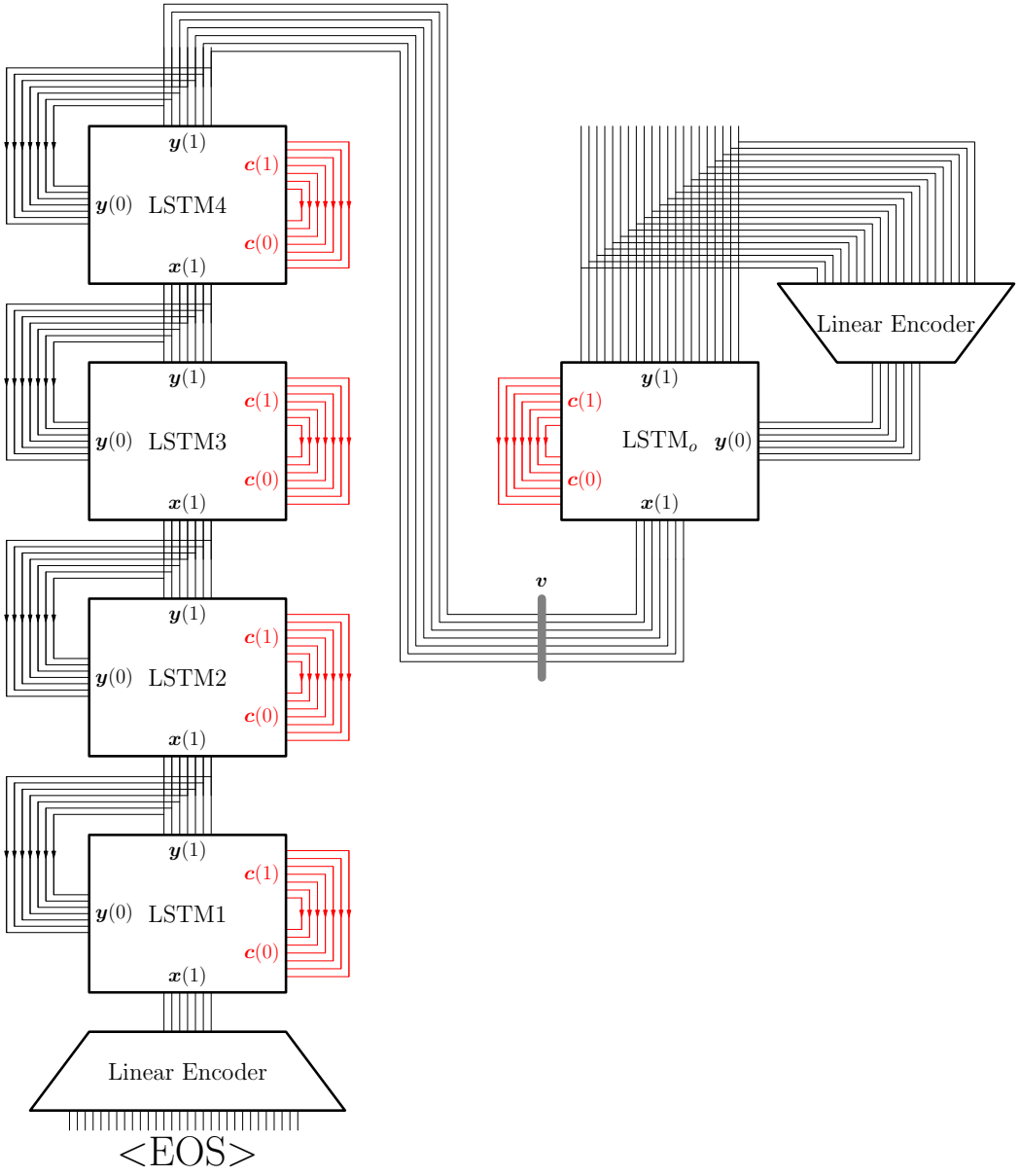
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



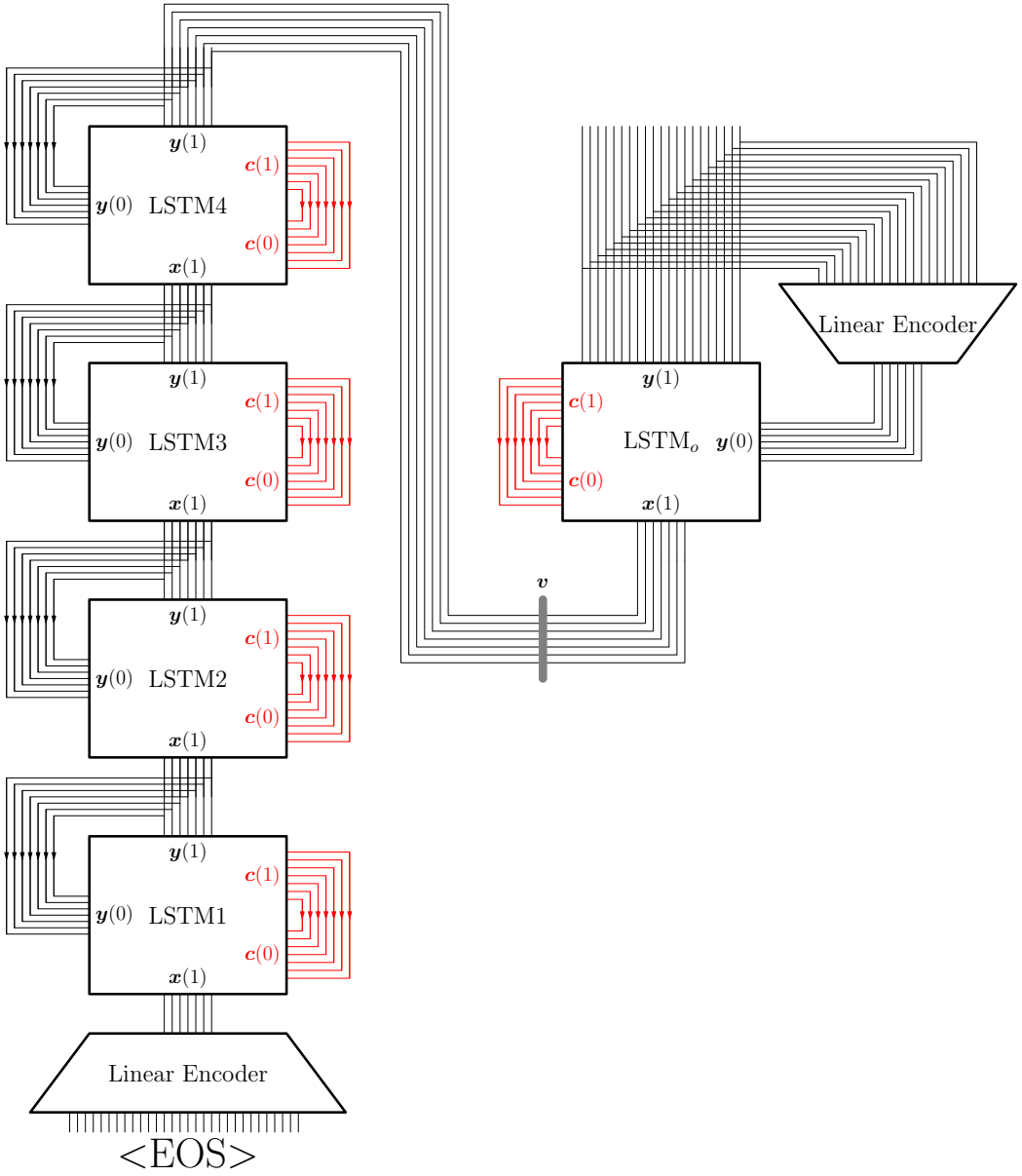
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



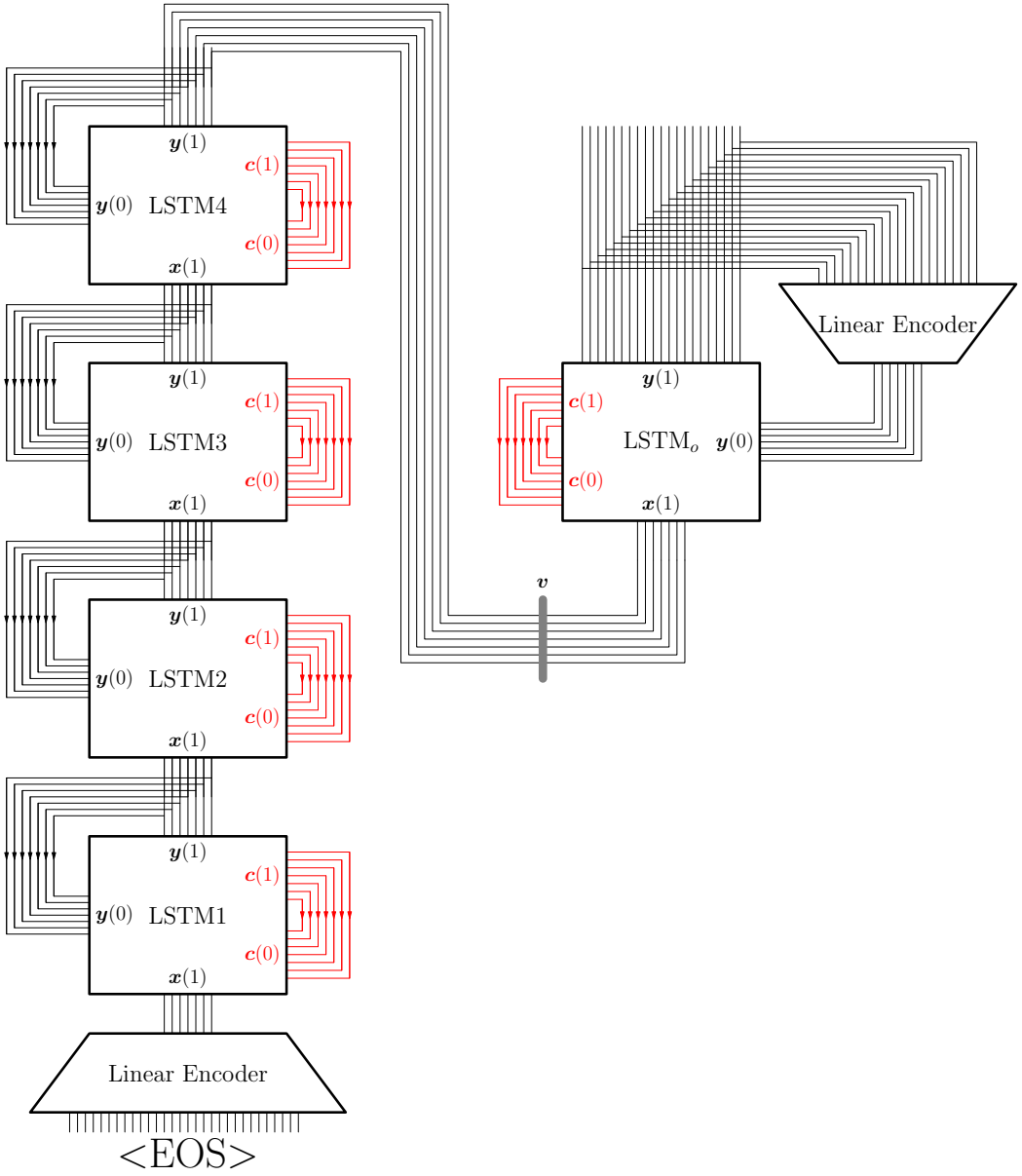
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



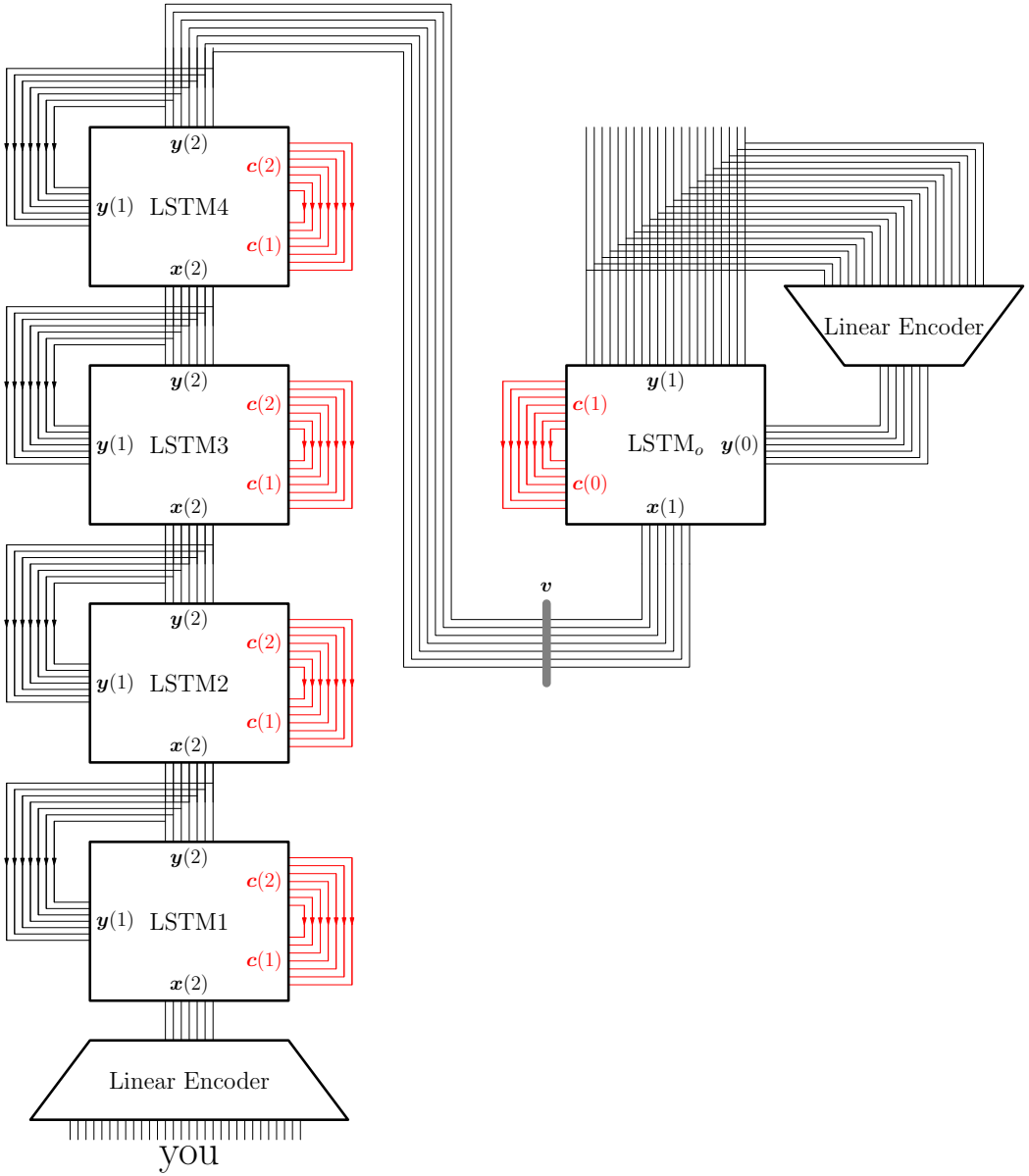
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



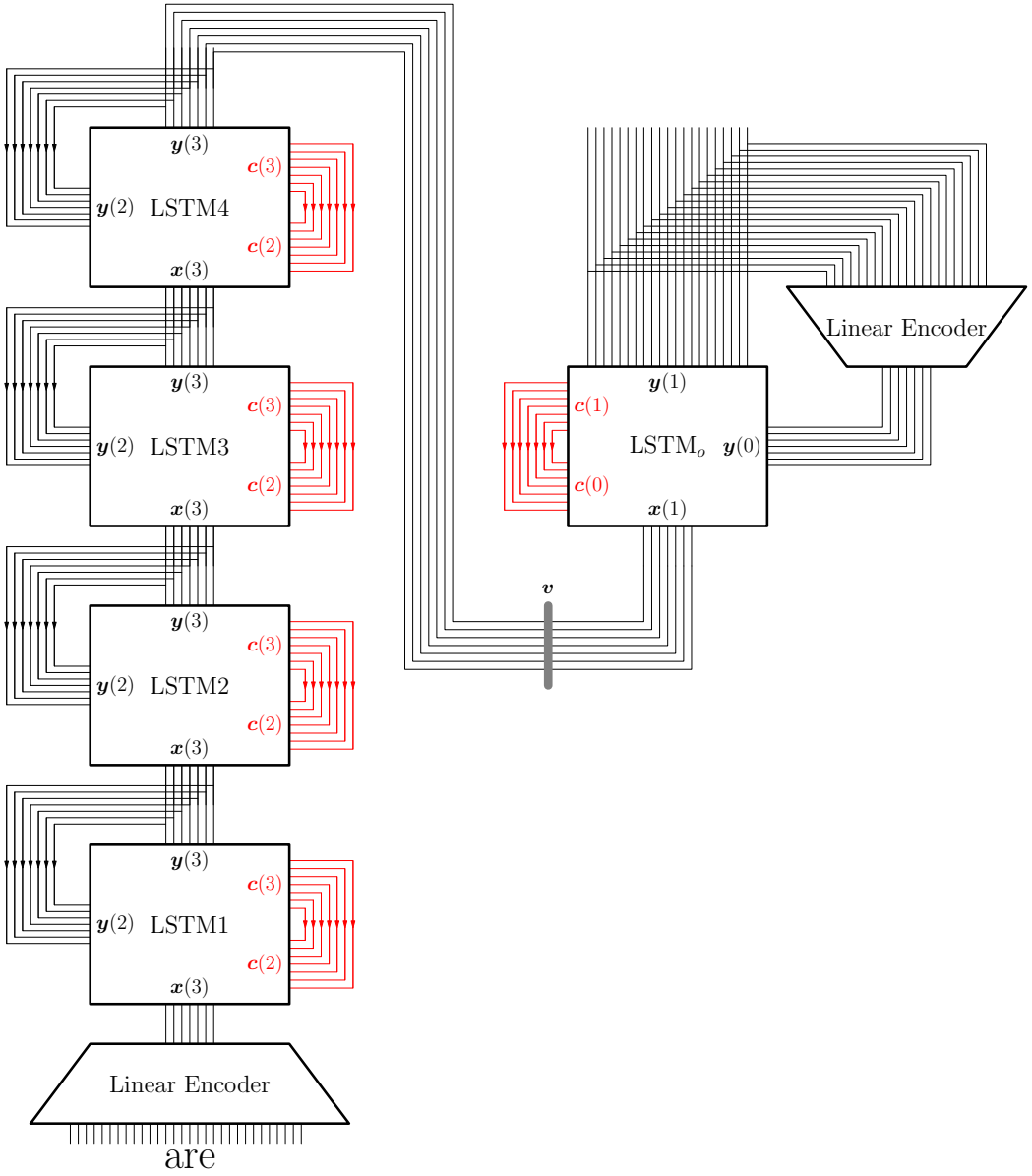
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



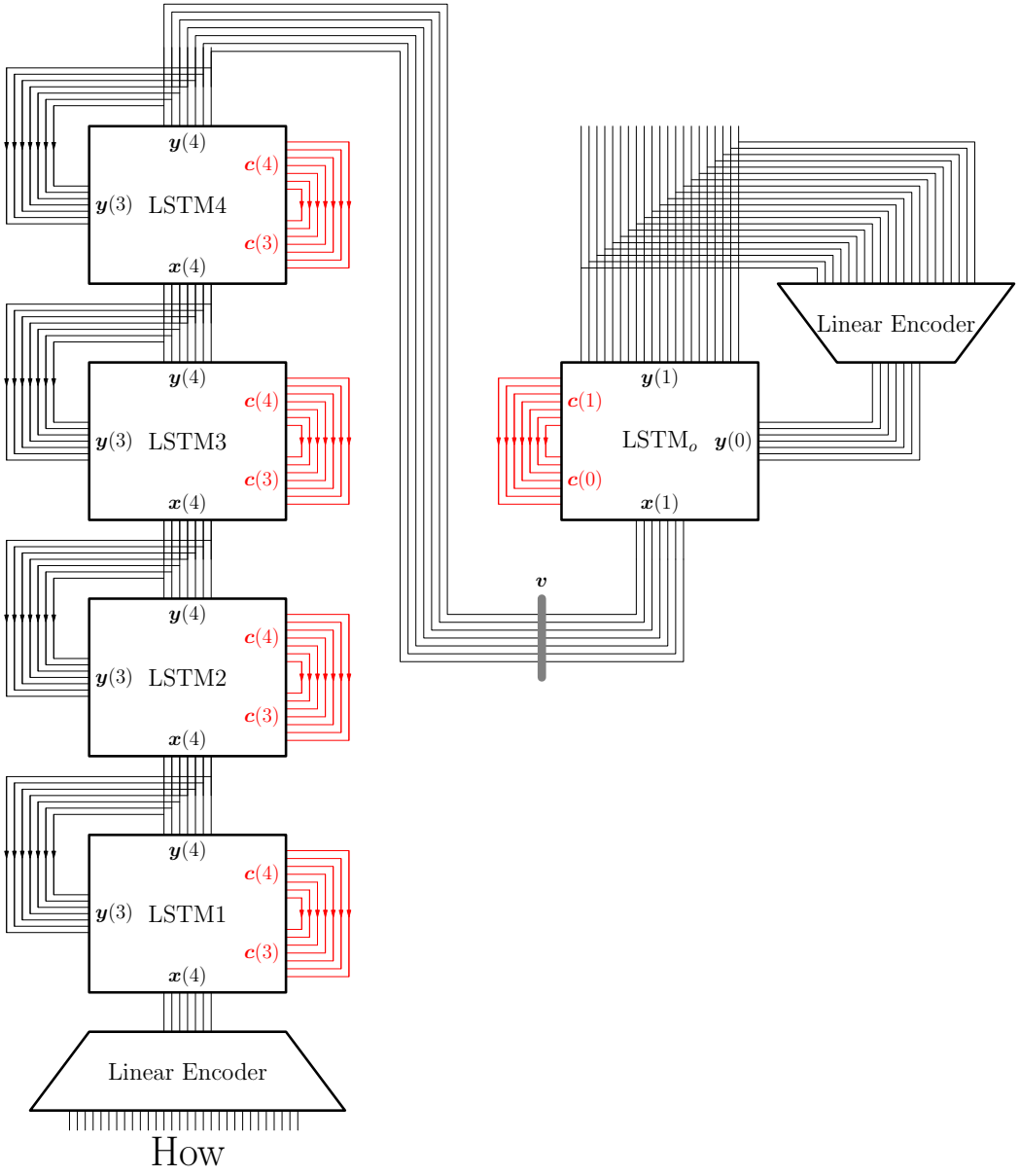
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



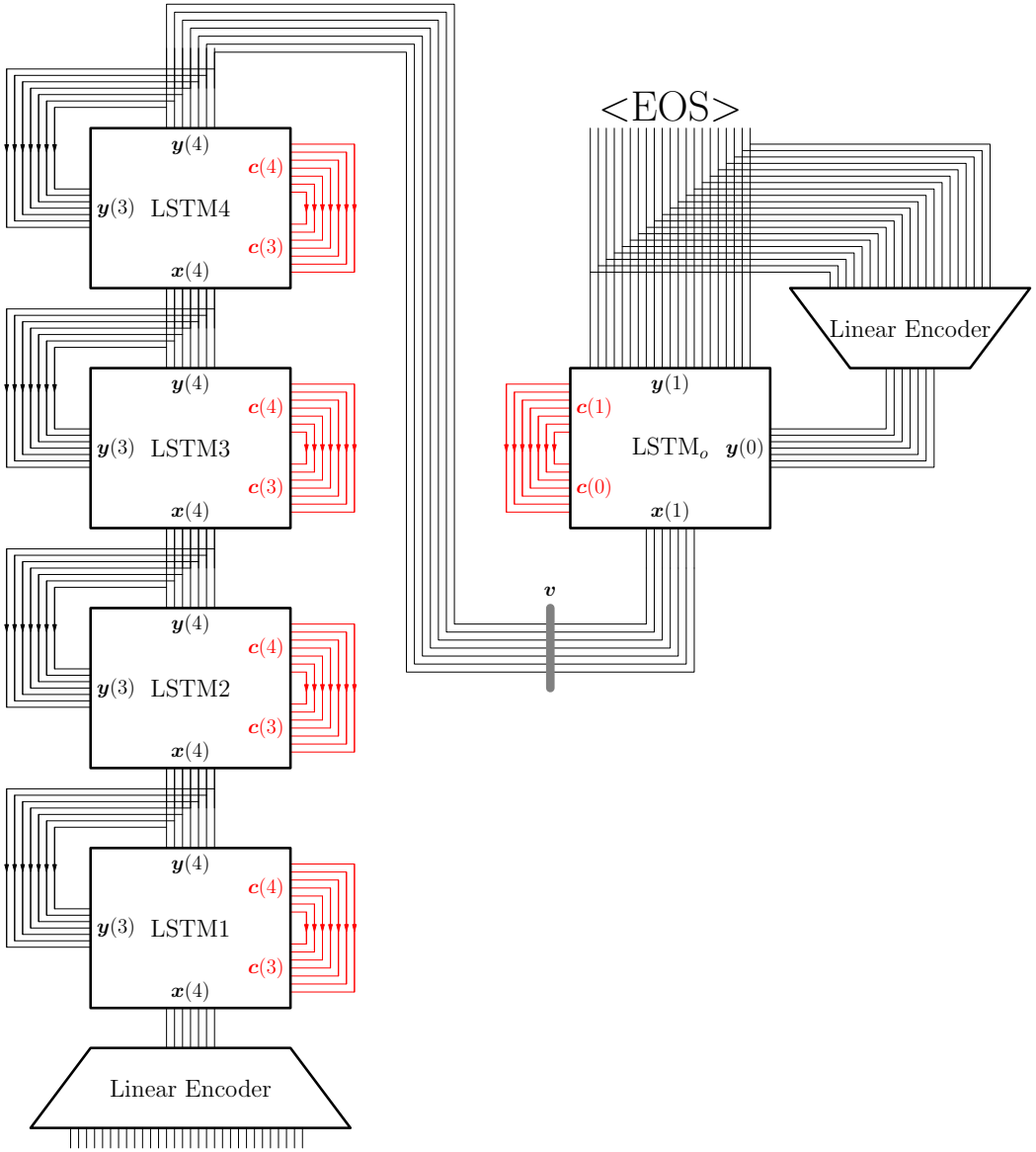
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



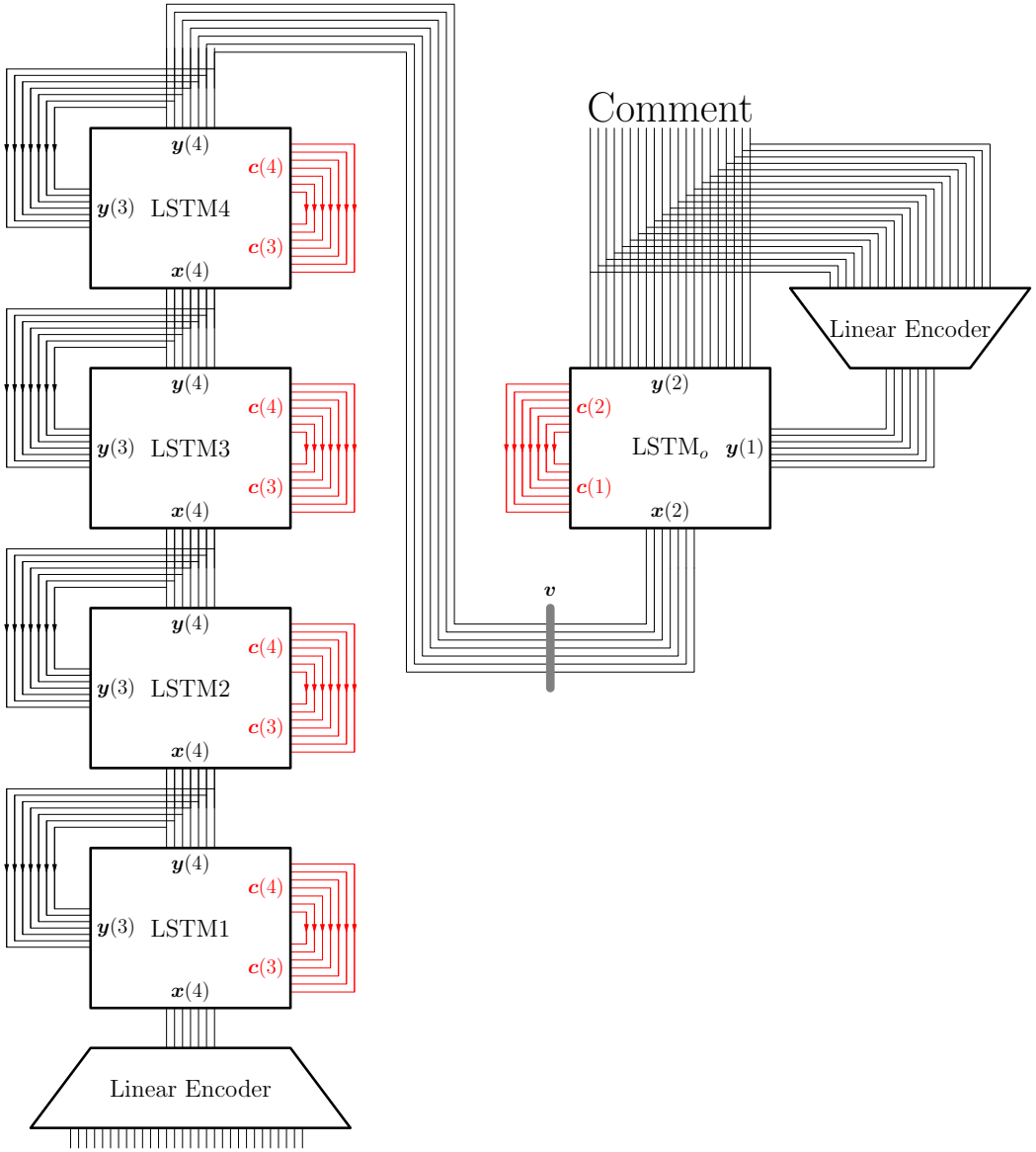
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



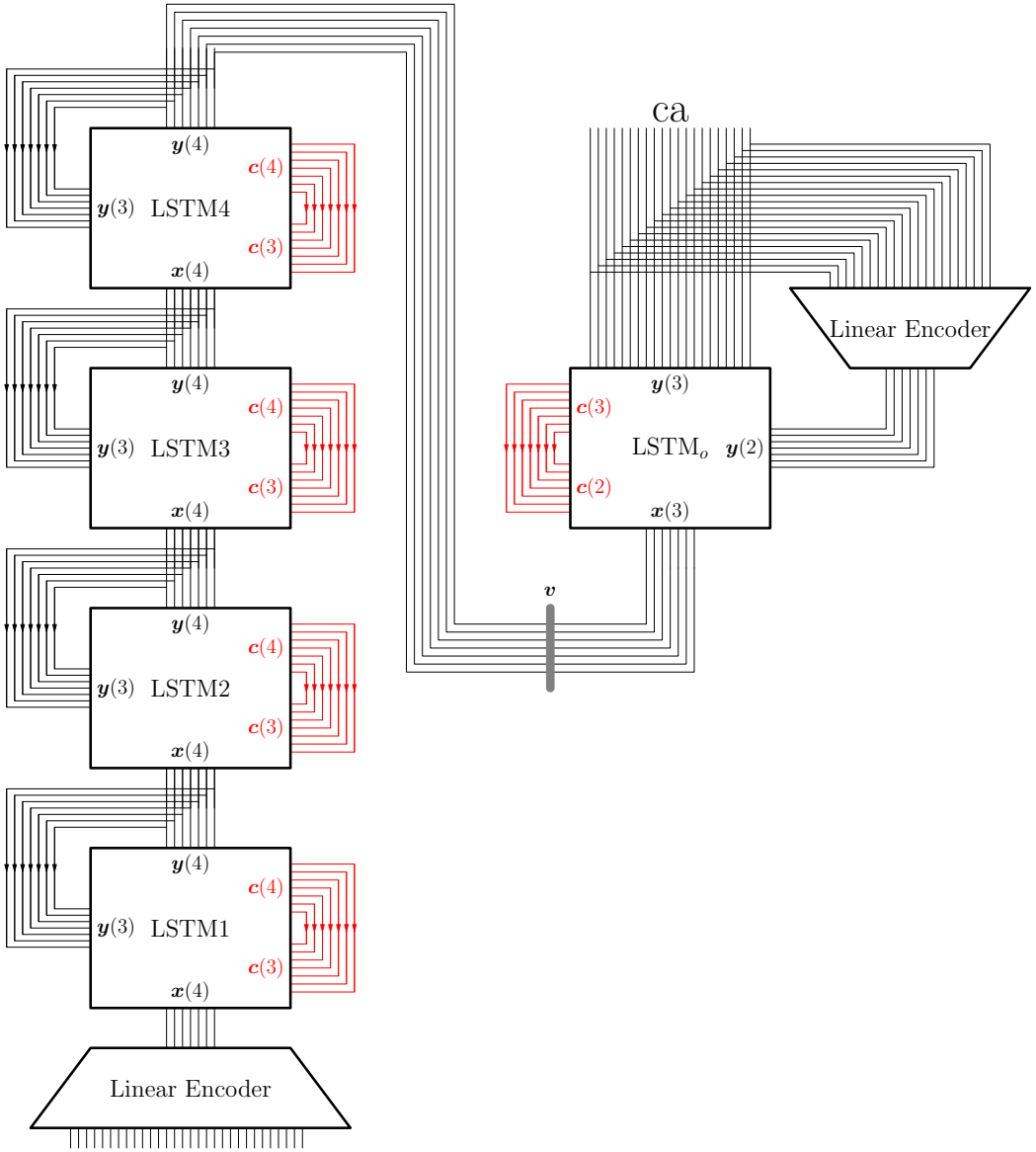
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



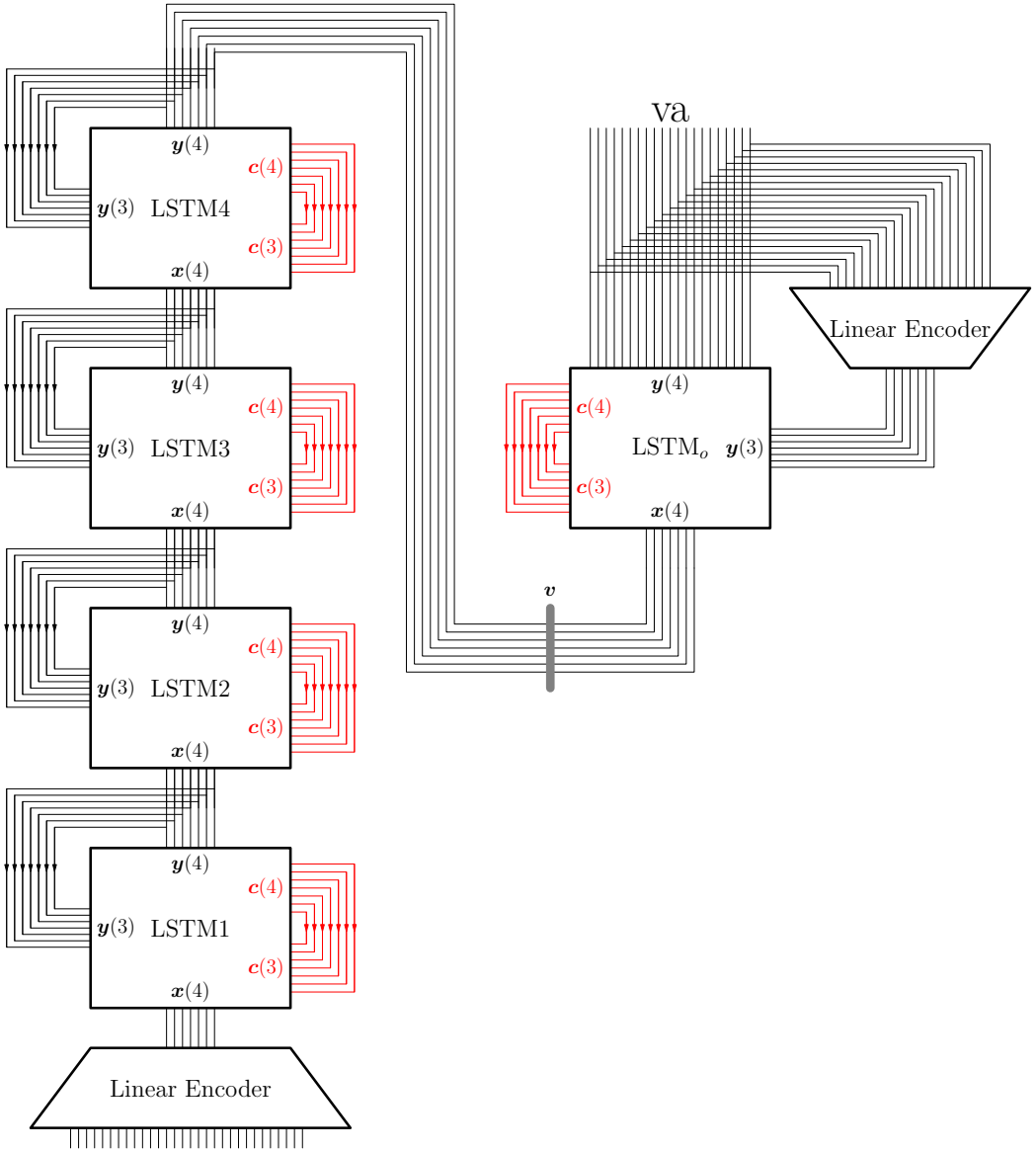
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



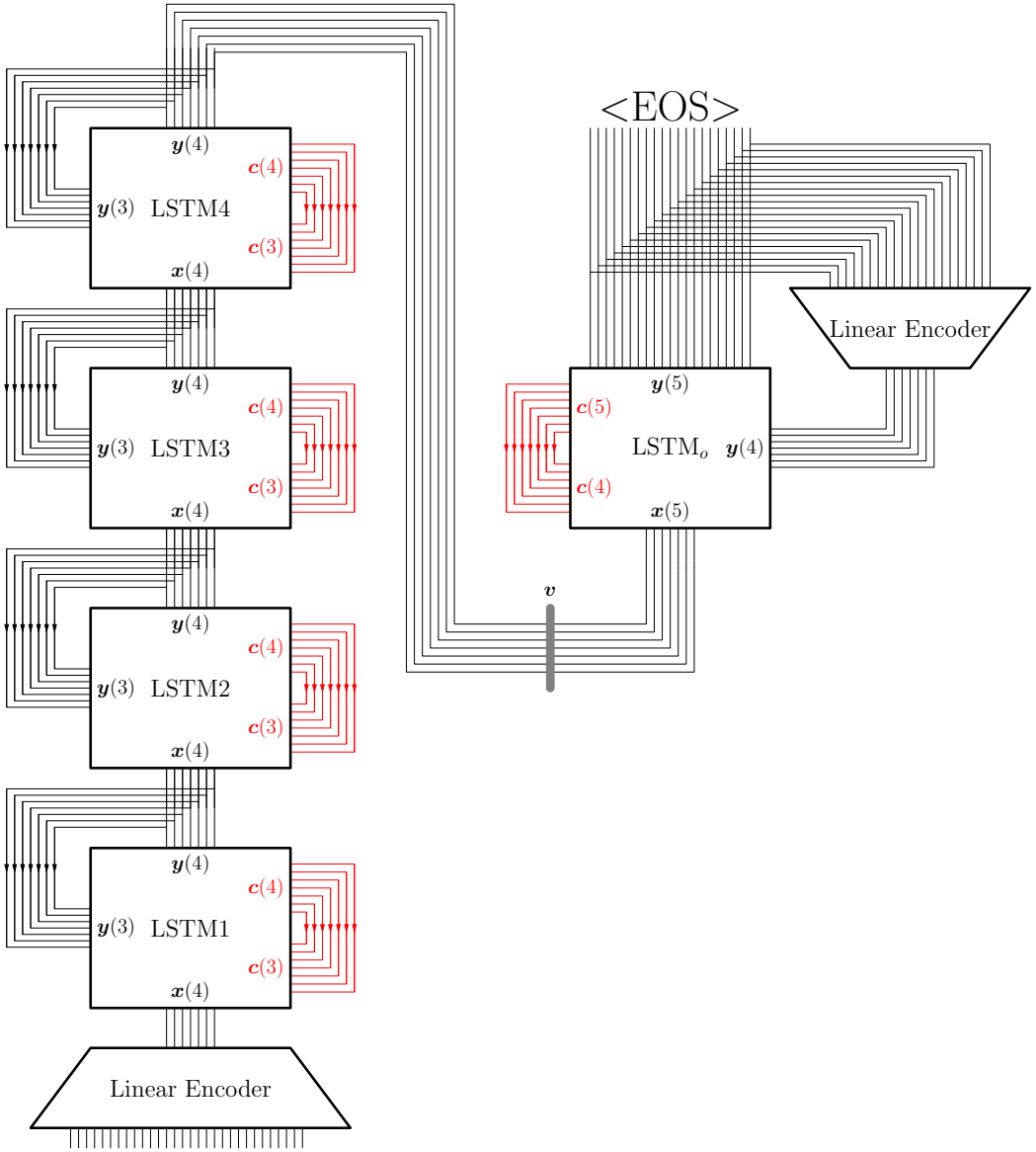
How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



How Are You?

- Used 4 layers of LSTM
- Internal vectors are of length 1000
- English put in backwards
- <EOS> token used to mark end of sentence
- All information stored in a static vector $\boldsymbol{v}$



Training Deep LSTM

- The English is put in backwards so that corresponding words in English and French are next to each other
- This speeds up learning
- Nevertheless they only managed 7.5 training epochs
- This took 10 days of 8 GPUs
- To achieve near state of art performance use an ensemble of 5 LSTMs

Training Deep LSTM

- The English is put in backwards so that corresponding words in English and French are next to each other
- This speeds up learning
- Nevertheless they only managed 7.5 training epochs
- This took 10 days of 8 GPUs
- To achieve near state of art performance use an ensemble of 5 LSTMs

Training Deep LSTM

- The English is put in backwards so that corresponding words in English and French are next to each other
- This speeds up learning
- Nevertheless they only managed 7.5 training epochs
- This took 10 days of 8 GPUs
- To achieve near state of art performance use an ensemble of 5 LSTMs

Training Deep LSTM

- The English is put in backwards so that corresponding words in English and French are next to each other
- This speeds up learning
- Nevertheless they only managed 7.5 training epochs
- This took 10 days of 8 GPUs
- To achieve near state of art performance use an ensemble of 5 LSTMs

Training Deep LSTM

- The English is put in backwards so that corresponding words in English and French are next to each other
- This speeds up learning
- Nevertheless they only managed 7.5 training epochs
- This took 10 days of 8 GPUs
- To achieve near state of art performance use an ensemble of 5 LSTMs

Other Applications

- LSTMs are widely used in time vary problems
- E.g. video interpretation
- In this case, typically use CNNs to preprocess the images and then use LSTMs to extract dynamic data

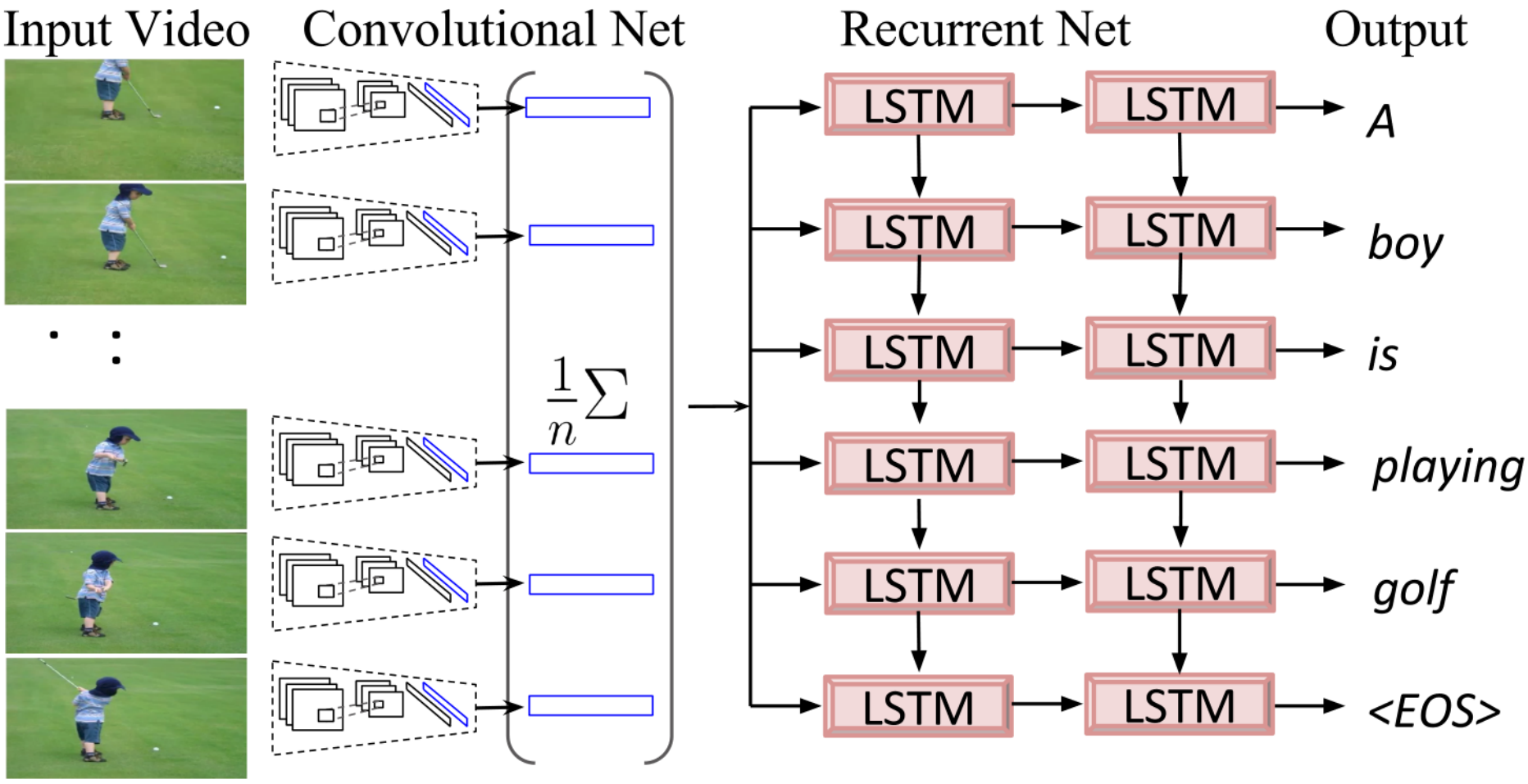
Other Applications

- LSTMs are widely used in time vary problems
- E.g. video interpretation
- In this case, typically use CNNs to preprocess the images and then use LSTMs to extract dynamic data

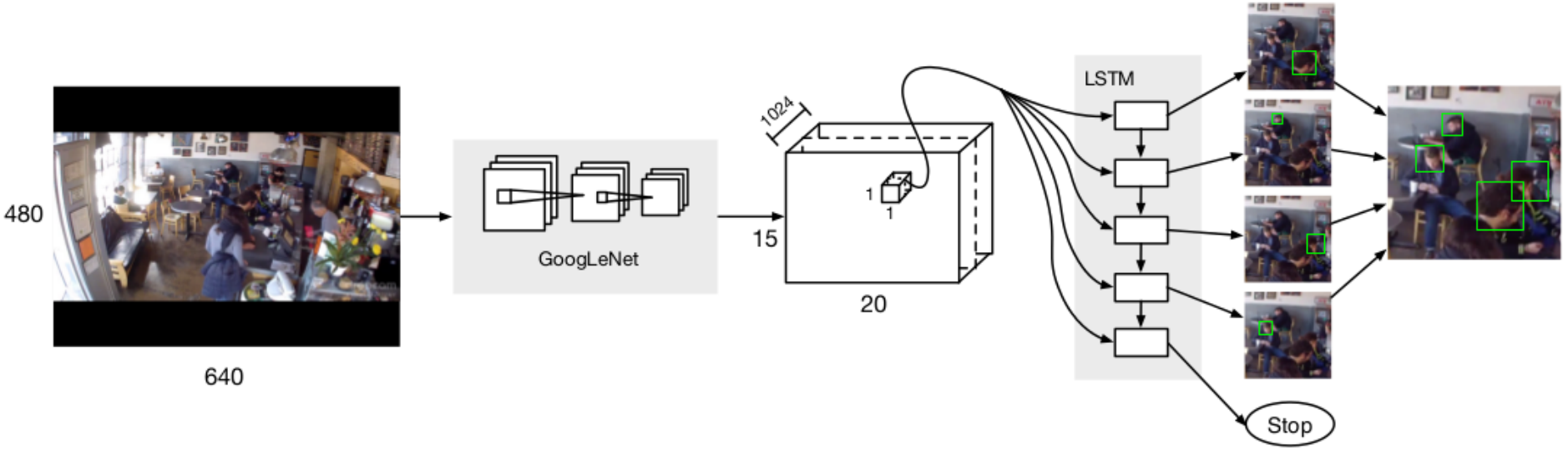
Other Applications

- LSTMs are widely used in time vary problems
- E.g. video interpretation
- In this case, typically use CNNs to preprocess the images and then use LSTMs to extract dynamic data

CNN and LSTM


Input Video
Convolutional Net
Recurrent Net
Output
$\frac{1}{n}\sum$
LSTM
A
boy
is
playing
golf
<EOS>

End-to-End People Detection



Conclusions

- LSTMs have become a major component of "deep" learning
- They allow a richer set of functionality than the traditional feed-forward networks
- Because the involve dense networks they are usually very slow to train
- There are variants that try to simplify the architecture
- But the key to storing long-term memory is to allow signals to propagate through time unaltered

Conclusions

- LSTMs have become a major component of "deep" learning
- They allow a richer set of functionality than the traditional feed-forward networks
- Because the involve dense networks they are usually very slow to train
- There are variants that try to simplify the architecture
- But the key to storing long-term memory is to allow signals to propagate through time unaltered

Conclusions

- LSTMs have become a major component of "deep" learning
- They allow a richer set of functionality than the traditional feed-forward networks
- Because the involve dense networks they are usually very slow to train
- There are variants that try to simplify the architecture
- But the key to storing long-term memory is to allow signals to propagate through time unaltered

Conclusions

- LSTMs have become a major component of "deep" learning
- They allow a richer set of functionality than the traditional feed-forward networks
- Because the involve dense networks they are usually very slow to train
- There are variants that try to simplify the architecture
- But the key to storing long-term memory is to allow signals to propagate through time unaltered

Conclusions

- LSTMs have become a major component of "deep" learning
- They allow a richer set of functionality than the traditional feed-forward networks
- Because the involve dense networks they are usually very slow to train
- There are variants that try to simplify the architecture
- But the key to storing long-term memory is to allow signals to propagate through time unaltered