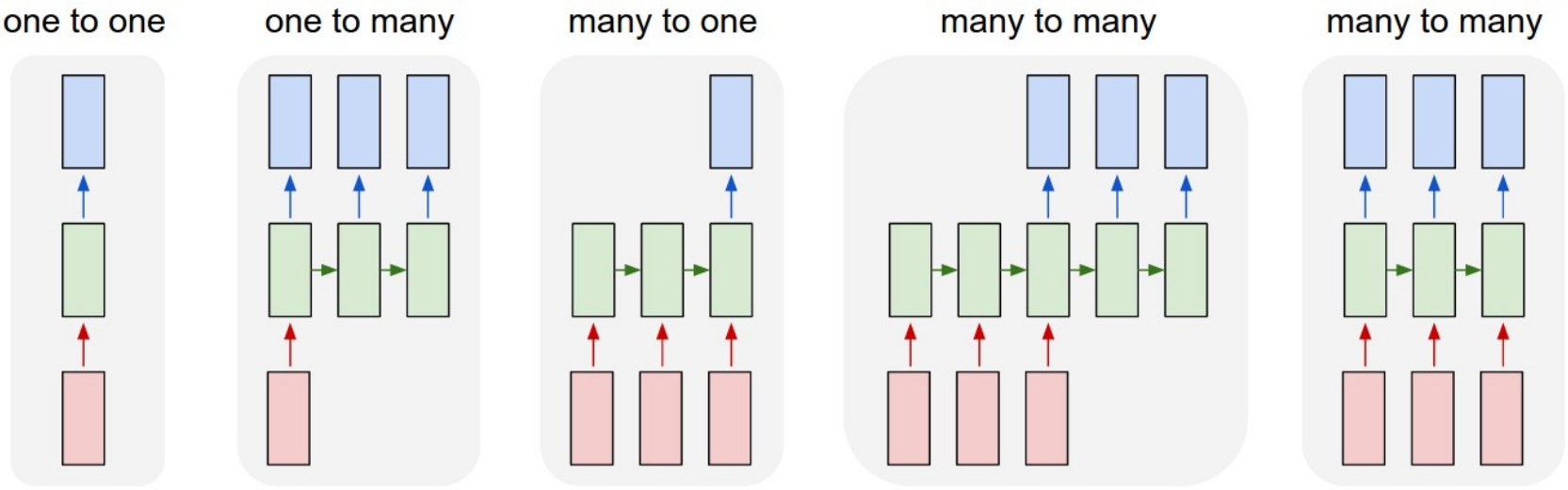
Recurrent Neural Networks

F. Noé[1]

Deep Learning Classes, FU Berlin 2018

Going beyond one-to-one mapping



1

- **One-to-one**: Classical (dense or convolutional) feedforward nets
- **One-to-many**: Given Image, generate sequence of words.
- **Many-to-one**: Given sequence of words, make classification (e.g., sentiment)
- **Many-to-many**: Text translation, on-line video segmentation

Recurrent Neural Networks (RNNs)

Rumelhart et al., 1986a

- **Specialized** for:
  - processing a sequence of values $[\mathbf{x}(1), ..., \mathbf{x}(t), ..., \mathbf{x}(T)]$.
    (t not necessarily physical time)
  - Sequences of variable length.
- **Example**: "*I went to Chile in 2013*" and "*In 2013, I went to Chile.*"
  - Extract the year the narrator went to Chile.
  - Traditional fully connected net would have separate parameters for each input feature, i.e. learn all of the rules of the language separately at each position in the sentence.
- **Key idea**: parameter sharing.
  - related idea: Convolution across a 1-D temporal sequence (time-delay neural networks – see Lang and Hinton, 1988; Waibel et al., 1989; Lang et al., 1990).

Recurrent Neural Networks (RNNs)

Rumelhart et al., 1986a

- **Specialized** for:
  - processing a sequence of values $[\mathbf{x}(1), ..., \mathbf{x}(t), ..., \mathbf{x}(T)]$. ($t$ not necessarily physical time)
  - Sequences of variable length.
- **Example**: "*I went to Chile in 2013*" and "*In 2013, I went to Chile.*"
  - Extract the year the narrator went to Chile.
  - Traditional fully connected net would have separate parameters for each input feature, i.e. learn all of the rules of the language separately at each position in the sentence.
- **Key idea**: parameter sharing.
  - related idea: Convolution across a 1-D temporal sequence (time-delay neural networks – see Lang and Hinton, 1988; Waibel et al., 1989; Lang et al., 1990).

Recurrent Neural Networks (RNNs)

Rumelhart et al., 1986a

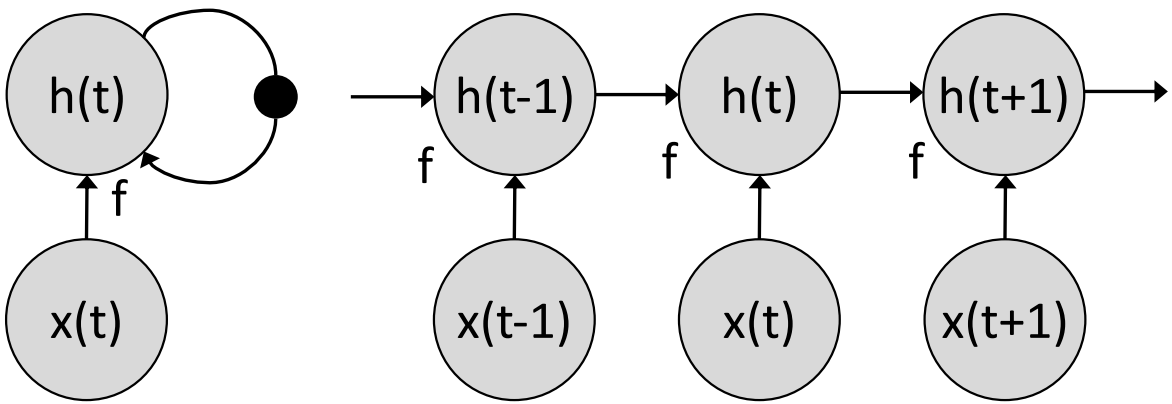
- **Specialized** for:
  - processing a sequence of values $[\mathbf{x}(1), ..., \mathbf{x}(t), ..., \mathbf{x}(T)]$. ($t$ not necessarily physical time)
  - Sequences of variable length.
- **Example**: "*I went to Chile in 2013*" and "*In 2013, I went to Chile.*"
  - Extract the year the narrator went to Chile.
  - Traditional fully connected net would have separate parameters for each input feature, i.e. learn all of the rules of the language separately at each position in the sentence.
- **Key idea**: parameter sharing.
  - related idea: Convolution across a 1-D temporal sequence (time-delay neural networks – see Lang and Hinton, 1988; Waibel et al., 1989; Lang et al., 1990).

RNN unfolding

- **Example**: dynamical system driven by external signal $\mathbf{x}(t)$:

$$\mathbf{h}(t) = f\left(\mathbf{h}(t-1), \mathbf{x}(t); \theta\right)$$

  - Train network to predict future from the past.
  - $\mathbf{h}(t)$ is summary of present and past. It is lossy as it represents an arbitrary-length sequence $(\mathbf{x}_1, ..., \mathbf{x}_t)$ with fixed length.



- **Unfolding**: represent recurrence after t steps with a function g_t:

$$\mathbf{h}(t) = g_t\left(\mathbf{x}(t), ..., \mathbf{x}(1)\right) = f\left(\mathbf{h}(t-1), \mathbf{x}(t); \theta\right)$$

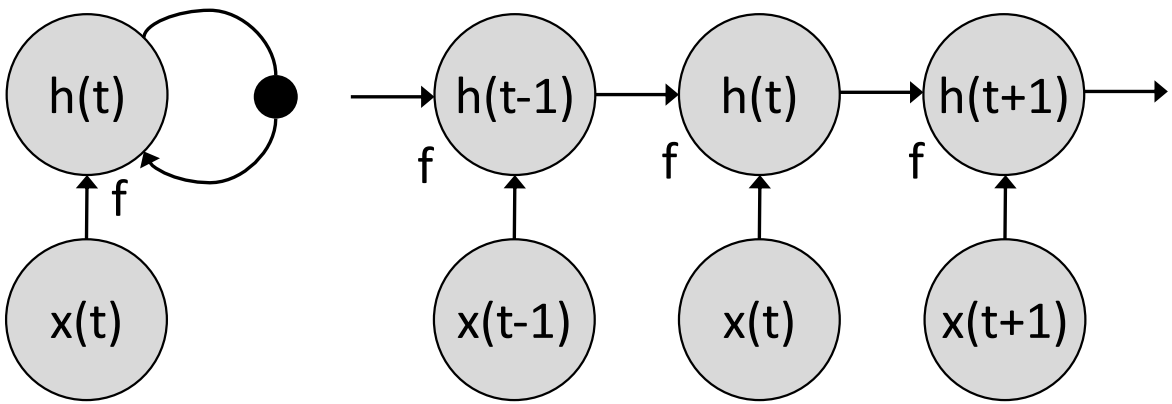
  - Allows us to factorize g_t into repeated application of a function f.
  - Can have same input size regardless of sequence length.
  - Parameter sharing: use same transition function f every time step.

RNN unfolding

- **Example**: dynamical system driven by external signal $\mathbf{x}(t)$:

$$\mathbf{h}(t) = f\left(\mathbf{h}(t-1), \mathbf{x}(t); \theta\right)$$

  - Train network to predict future from the past.
  - $\mathbf{h}(t)$ is summary of present and past. It is lossy as it represents an arbitrary-length sequence $(\mathbf{x}_1, ..., \mathbf{x}_t)$ with fixed length.

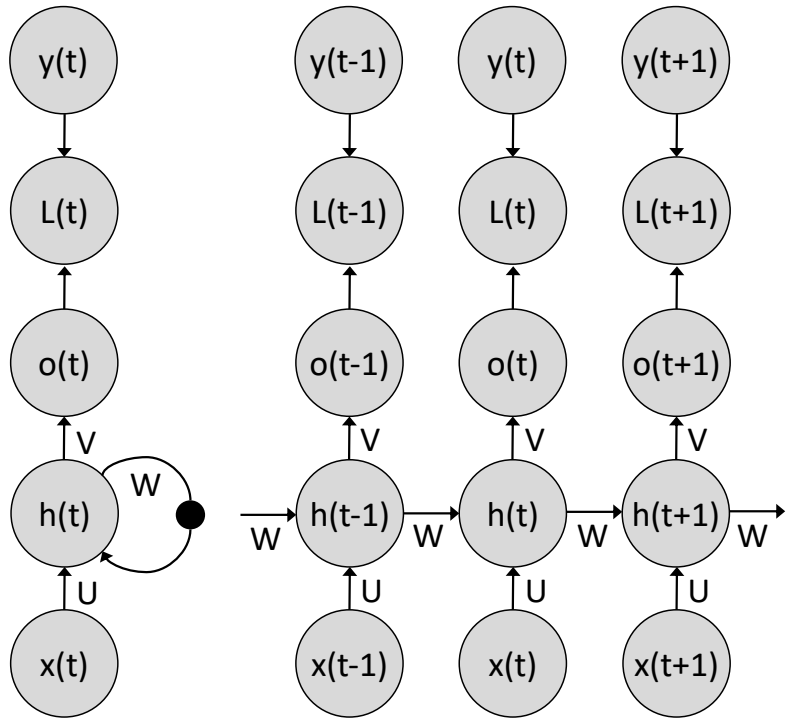


- **Unfolding**: represent recurrence after t steps with a function g_t:

$$\mathbf{h}(t) = g_t\left(\mathbf{x}(t), ..., \mathbf{x}(1)\right) = f\left(\mathbf{h}(t-1), \mathbf{x}(t); \theta\right)$$

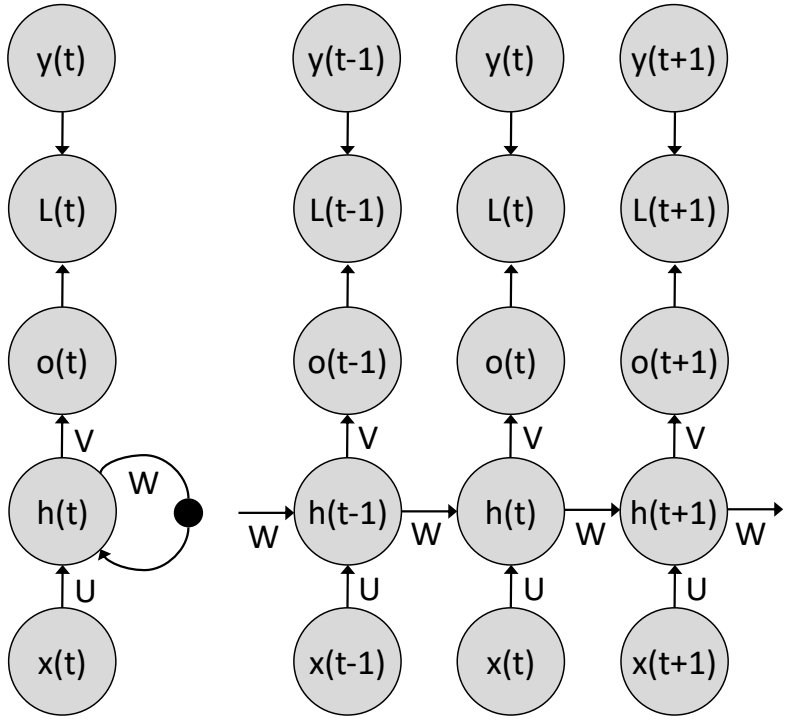
  - Allows us to factorize g_t into repeated application of a function f.
  - Can have same input size regardless of sequence length.
  - Parameter sharing: use same transition function f every time step.

A basic Universal RNN



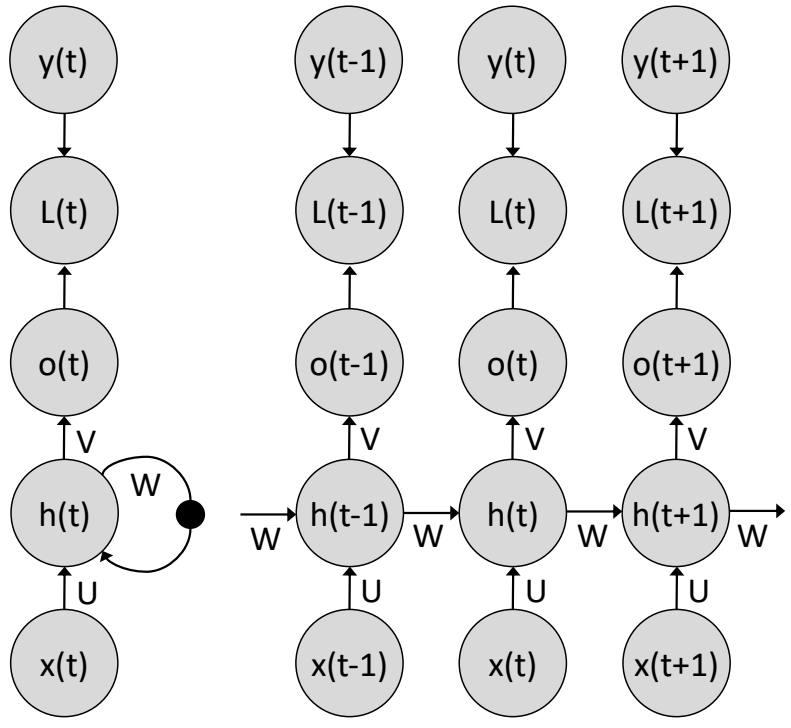
- Can with finite size compute any function computable by a Turing machine (Siegelmann and Sontag, 1991; Siegelmann, 1995; Siegelmann and Sontag, 1995; Hyotyniemi, 1996).
- Loss L measures distance of predictions $\mathbf{o}_t$ from training targets $\mathbf{y}_t$.

A basic Universal RNN



- Can with finite size compute any function computable by a Turing machine (Siegelmann and Sontag, 1991; Siegelmann, 1995; Siegelmann and Sontag, 1995; Hyotyniemi, 1996).
- Loss L measures distance of predictions $\mathbf{o}_t$ from training targets $\mathbf{y}_t$.

A basic Universal RNN



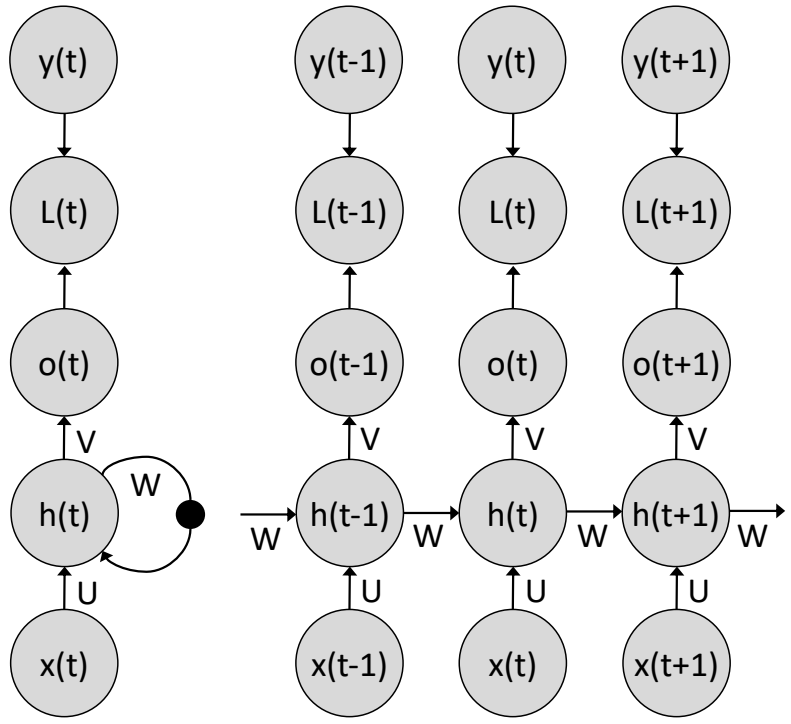
- **Update equations** (example):

$$\mathbf{h}(t) = \tanh\left(\mathbf{b} + \mathbf{W}\mathbf{h}(t-1) + \mathbf{U}\mathbf{x}(t)\right)$$
$$\mathbf{o}(t) = \mathrm{softmax}\left(\mathbf{c} + \mathbf{V}\mathbf{h}(t)\right)$$

  with $\mathrm{softmax}(\mathbf{a})_i = \exp(a_i) / \sum_j \exp(a_j)$

- **Parameters**: Bias vectors $\mathbf{b}, \mathbf{c}$. Weight matrices $\mathbf{U}, \mathbf{V}, \mathbf{W}$

A basic Universal RNN



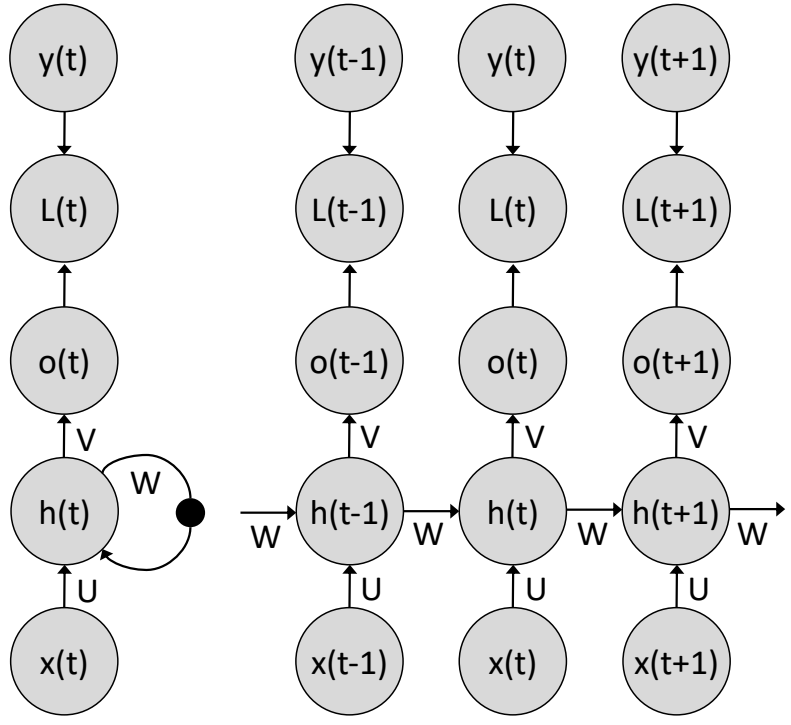
- **Update equations** (example):

$$\mathbf{h}(t) = \tanh\left(\mathbf{b} + \mathbf{W}\mathbf{h}(t-1) + \mathbf{U}\mathbf{x}(t)\right)$$
$$\mathbf{o}(t) = \mathrm{softmax}\left(\mathbf{c} + \mathbf{V}\mathbf{h}(t)\right)$$

  with $\mathrm{softmax}(\mathbf{a})_i = \exp(a_i) / \sum_j \exp(a_j)$

- **Parameters**: Bias vectors $\mathbf{b}, \mathbf{c}$. Weight matrices $\mathbf{U}, \mathbf{V}, \mathbf{W}$

A basic Universal RNN

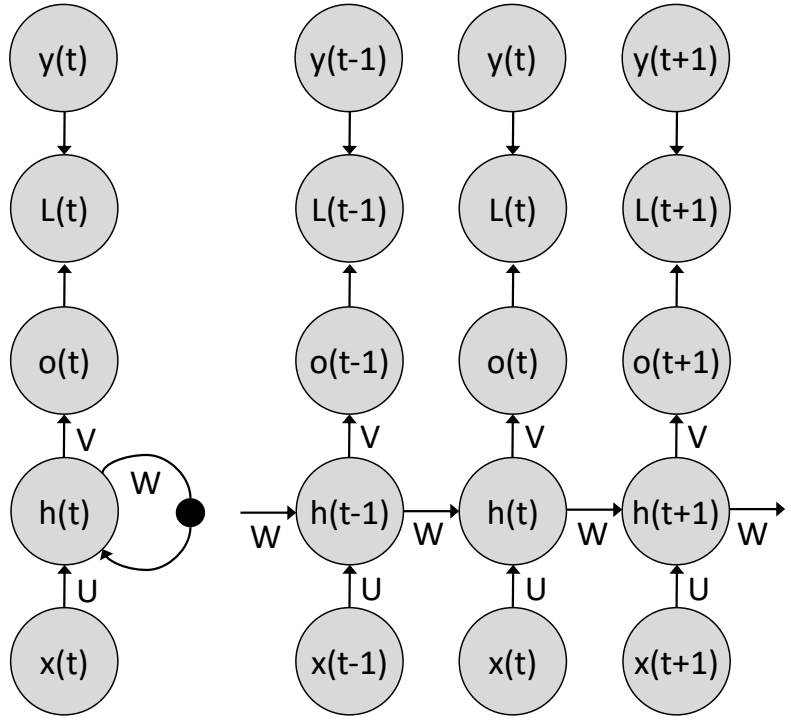


- RNN maps input to output of same length. Total loss:

$$L(\mathbf{x}(1),\ldots,\mathbf{x}(T);\mathbf{y}(1),\ldots,\mathbf{y}(T)) = -\sum_t \log p_{\text{model}}(\mathbf{y}(t) \mid \mathbf{x}(1),\ldots,\mathbf{x}(t))$$

  where p_{model} is computed from $\mathbf{y}(t)$ and $\mathbf{o}(t)$.
- **Training**: continguous minibatches $[\mathbf{x}(t),\ldots,\mathbf{x}(t+\tau)]$
- Gradient computation: **Back-propagation through time (BPTT)**. Forward and backward pass through unfolded graph. Runtime $O(\tau)$, cannot be reduced by parallelization.

A basic Universal RNN

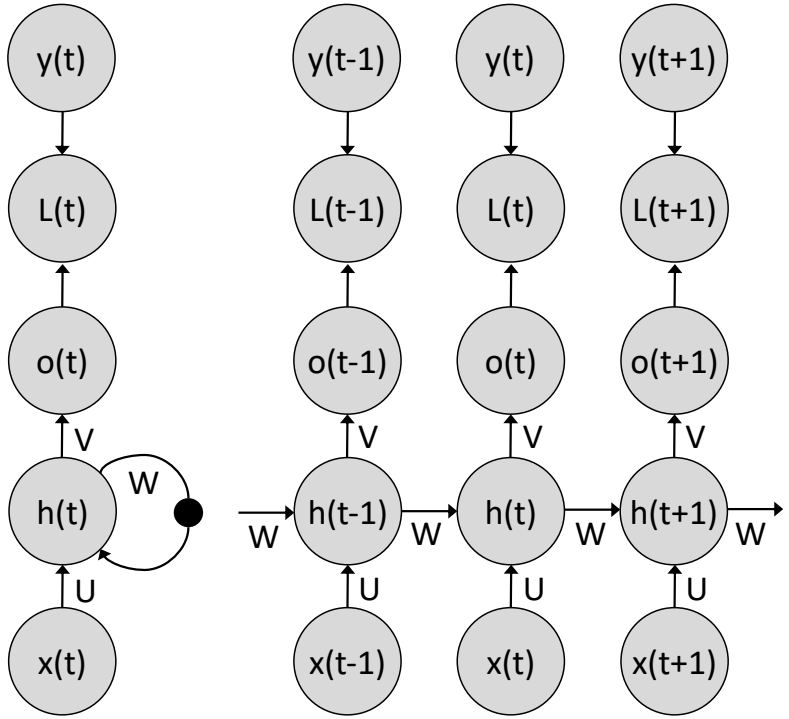


- RNN maps input to output of same length. Total loss:

$$L(\mathbf{x}(1),\ldots,\mathbf{x}(T);\mathbf{y}(1),\ldots,\mathbf{y}(T)) = -\sum_t \log p_{\text{model}}(\mathbf{y}(t) \mid \mathbf{x}(1),\ldots,\mathbf{x}(t))$$

  where p_{model} is computed from $\mathbf{y}(t)$ and $\mathbf{o}(t)$.
- **Training**: continguous minibatches $[\mathbf{x}(t),\ldots,\mathbf{x}(t+\tau)]$
- Gradient computation: **Back-propagation through time (BPTT)**. Forward and backward pass through unfolded graph. Runtime $O(\tau)$, cannot be reduced by parallelization.

A basic Universal RNN



- RNN maps input to output of same length. Total loss:

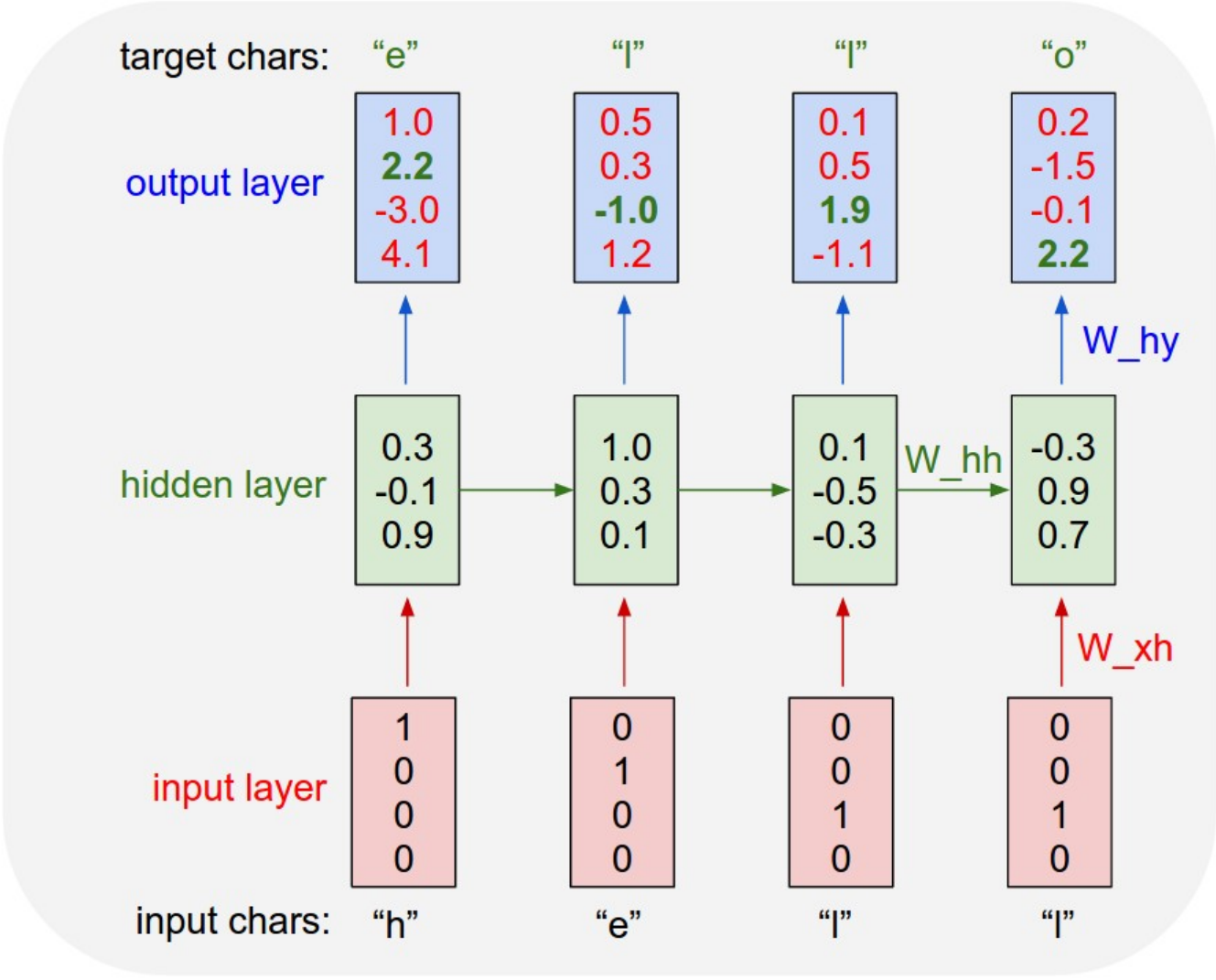
$$L(\mathbf{x}(1),...,\mathbf{x}(T);\mathbf{y}(1),...,\mathbf{y}(T)) = -\sum_t \log p_{\text{model}}(\mathbf{y}(t) \mid \mathbf{x}(1),...,\mathbf{x}(t))$$

  where p_{model} is computed from $\mathbf{y}(t)$ and $\mathbf{o}(t)$.
- **Training**: continguous minibatches $[\mathbf{x}(t),...,\mathbf{x}(t+\tau)]$
- Gradient computation: **Back-propagation through time (BPTT)**. Forward and backward pass through unfolded graph. Runtime $O(\tau)$, cannot be reduced by parallelization.

```
rnn = RNN()
o = rnn.step(x) # x input vector, y output vector

class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) +
                     np.dot(self.W_xh, x))
    # compute the output vector
    o = np.dot(self.W_hy, self.h)
    return y
```

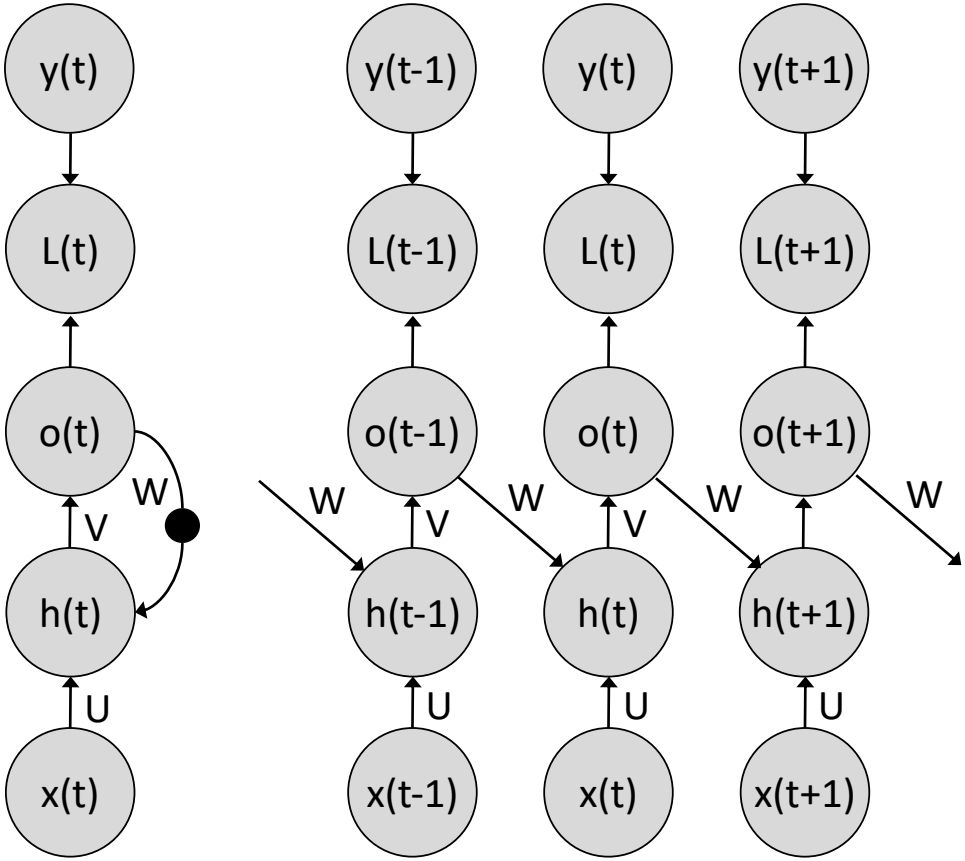
```
rnn = RNN()
o = rnn.step(x) # x input vector, y output vector

class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) +
                     np.dot(self.W_xh, x))
    # compute the output vector
    o = np.dot(self.W_hy, self.h)
    return y
```

Example: learn to pronounce "hello". Available classes: {'h', 'e', 'l', 'o'}



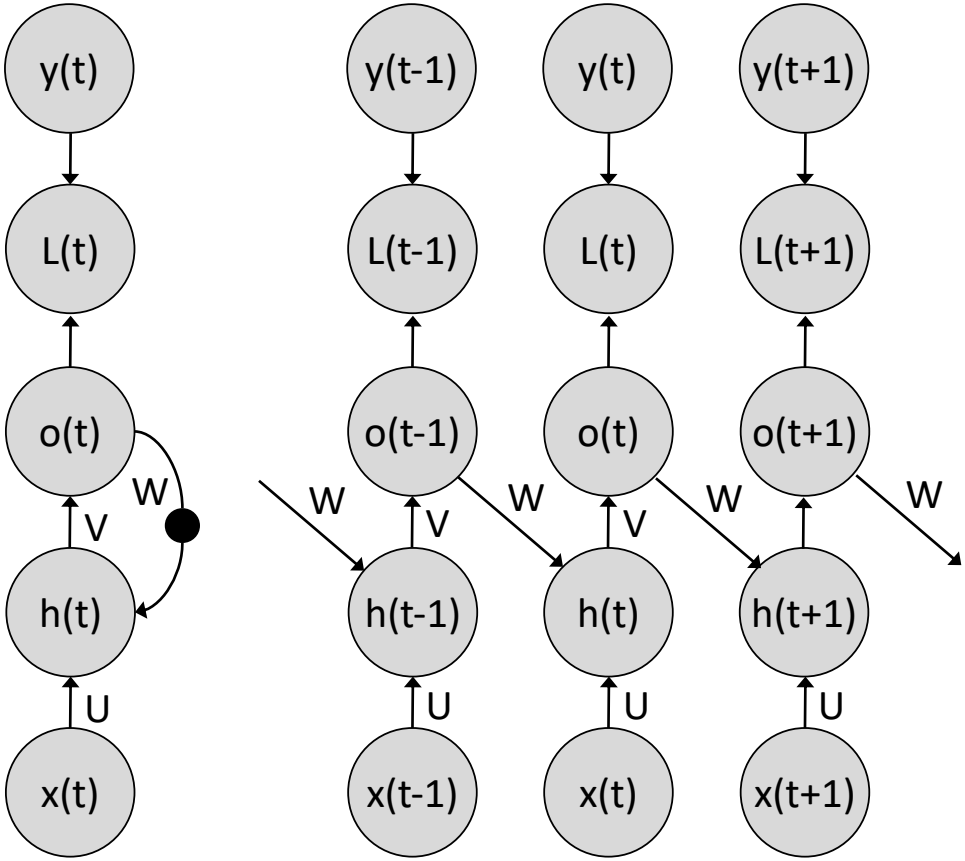
1

RNNs without direct hidden state feedback



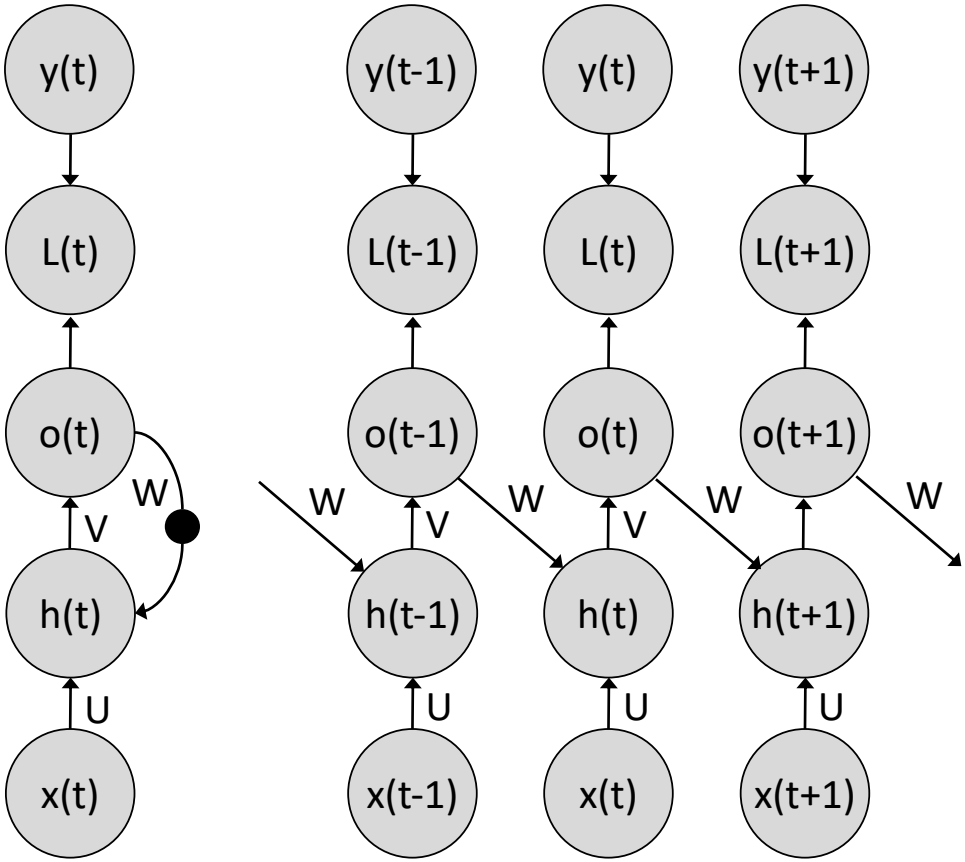
- Only recurrence is from output to hidden layer.
- Strictly less powerful (can express a smaller set of functions) than previous universal RNN.
- Previous universal RNN can put any information about the past into **h** and transmit **h** to the future. Present RNN is trained to produce outputs, and only **o** can be transmitted to the future.

RNNs without direct hidden state feedback



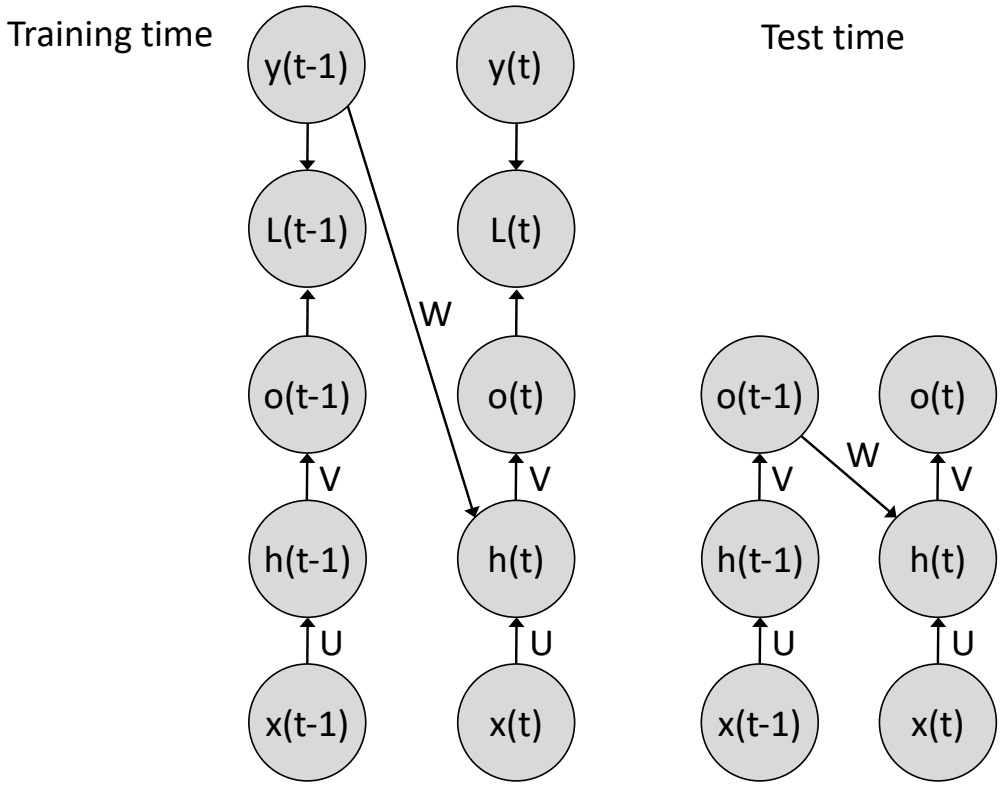
- Only recurrence is from output to hidden layer.
- Strictly less powerful (can express a smaller set of functions) than previous universal RNN.
- Previous universal RNN can put any information about the past into **h** and transmit **h** to the future. Present RNN is trained to produce outputs, and only **o** can be transmitted to the future.

RNNs without direct hidden state feedback



- Only recurrence is from output to hidden layer.
- Strictly less powerful (can express a smaller set of functions) than previous universal RNN.
- Previous universal RNN can put any information about the past into **h** and transmit **h** to the future. Present RNN is trained to produce outputs, and only **o** can be transmitted to the future.

Teacher forcing

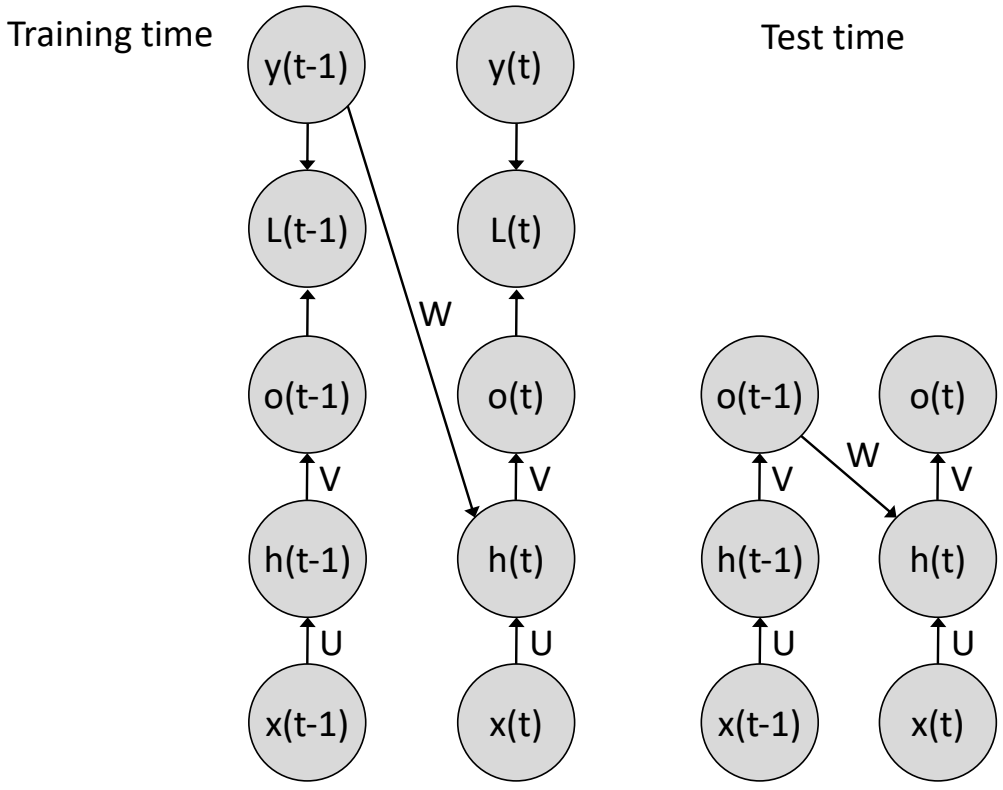


- For loss functions comparing $\mathbf{o}(t)$ and $\mathbf{y}(t)$, replace the feedback $\mathbf{o}(t) \to \mathbf{h}(t+1)$ by $\mathbf{y}(t) \to \mathbf{h}(t+1)$ at training time.
- Decouples all time steps, training can be parallelized over t
- Consistent with likelihood structure:

$$\log p\left(\mathbf{y}(1), \mathbf{y}(2) \mid \mathbf{x}(1), \mathbf{x}(2)\right) = \log p\left(\mathbf{y}(2), \mid \mathbf{y}(1), \mathbf{x}(1), \mathbf{x}(2)\right) + \log p\left(\mathbf{y}(1) \mid \mathbf{x}(1), \mathbf{x}(2)\right)$$

- Test / production time: Feedback is $\mathbf{o}(t) \to \mathbf{h}(t+1)$.

Teacher forcing

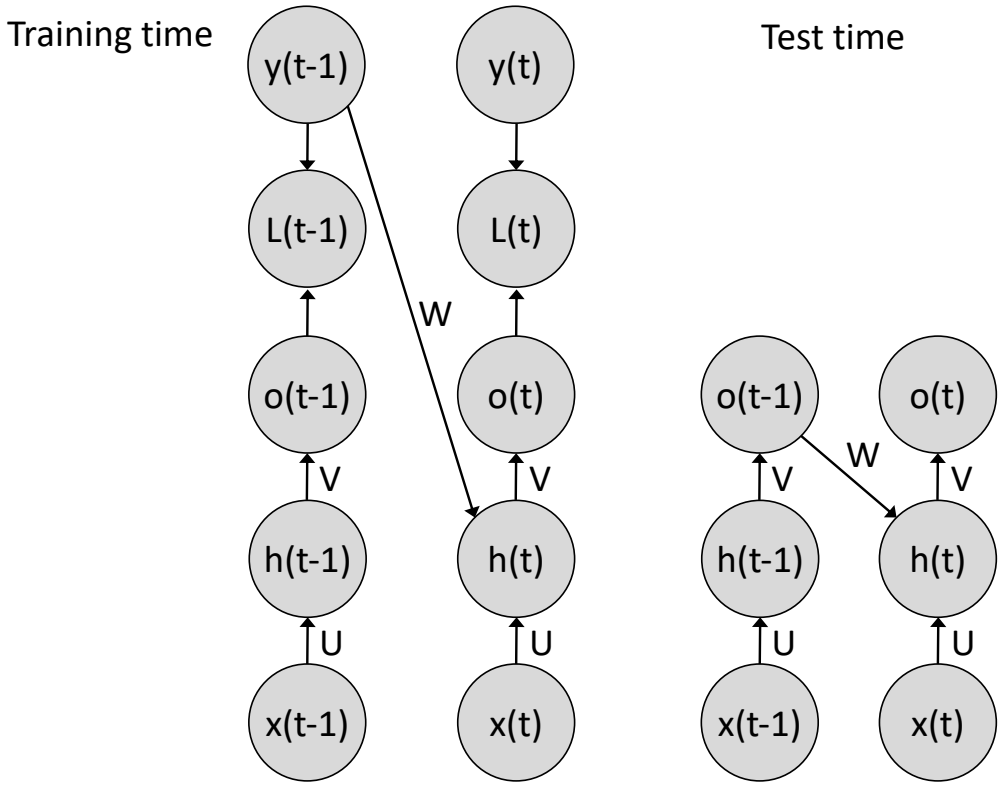


- For loss functions comparing $\mathbf{o}(t)$ and $\mathbf{y}(t)$, replace the feedback $\mathbf{o}(t) \to \mathbf{h}(t+1)$ by $\mathbf{y}(t) \to \mathbf{h}(t+1)$ at training time.
- Decouples all time steps, training can be parallelized over t
- Consistent with likelihood structure:

$$\log p(\mathbf{y}(1), \mathbf{y}(2) \mid \mathbf{x}(1), \mathbf{x}(2)) = \log p(\mathbf{y}(2), \mid \mathbf{y}(1), \mathbf{x}(1), \mathbf{x}(2)) + \log p(\mathbf{y}(1) \mid \mathbf{x}(1), \mathbf{x}(2))$$

- Test / production time: Feedback is $\mathbf{o}(t) \to \mathbf{h}(t+1)$.

Teacher forcing

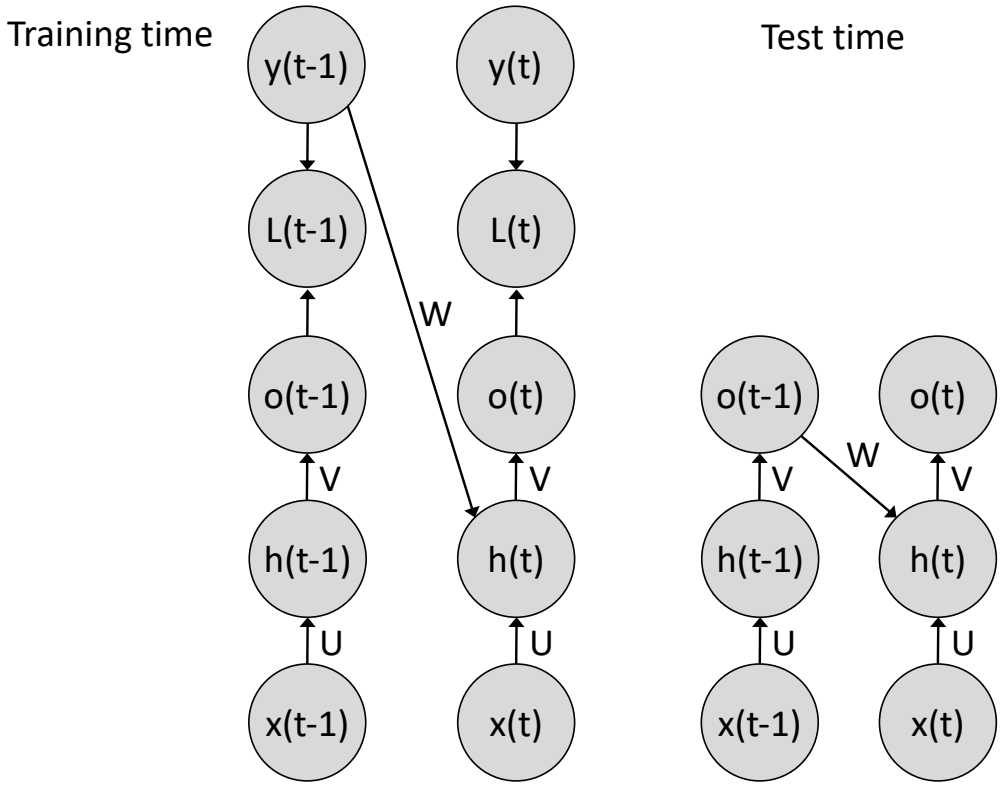


- For loss functions comparing $\mathbf{o}(t)$ and $\mathbf{y}(t)$, replace the feedback $\mathbf{o}(t) \to \mathbf{h}(t+1)$ by $\mathbf{y}(t) \to \mathbf{h}(t+1)$ at training time.
- Decouples all time steps, training can be parallelized over t
- Consistent with likelihood structure:

$$\log p(\mathbf{y}(1), \mathbf{y}(2) \mid \mathbf{x}(1), \mathbf{x}(2)) = \log p(\mathbf{y}(2), \mid \mathbf{y}(1), \mathbf{x}(1), \mathbf{x}(2)) + \log p(\mathbf{y}(1) \mid \mathbf{x}(1), \mathbf{x}(2))$$

- Test / production time: Feedback is $\mathbf{o}(t) \to \mathbf{h}(t+1)$.

Teacher forcing

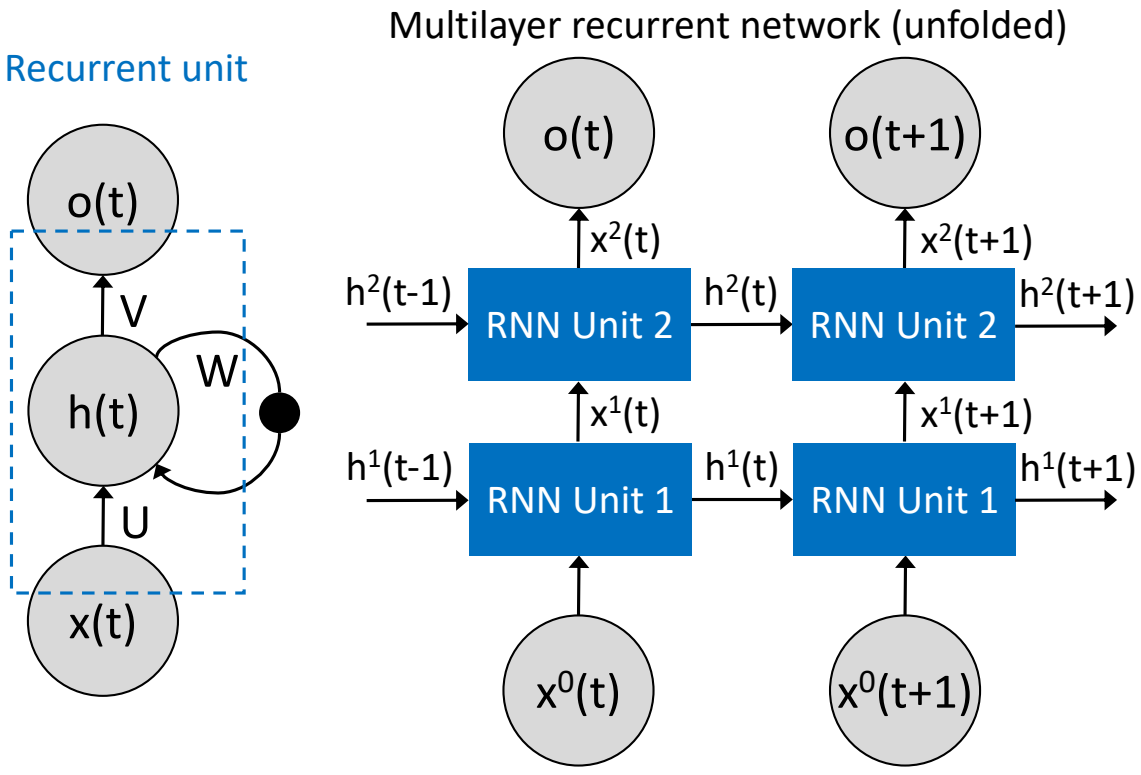


- For loss functions comparing $\mathbf{o}(t)$ and $\mathbf{y}(t)$, replace the feedback $\mathbf{o}(t) \to \mathbf{h}(t+1)$ by $\mathbf{y}(t) \to \mathbf{h}(t+1)$ at training time.
- Decouples all time steps, training can be parallelized over t
- Consistent with likelihood structure:

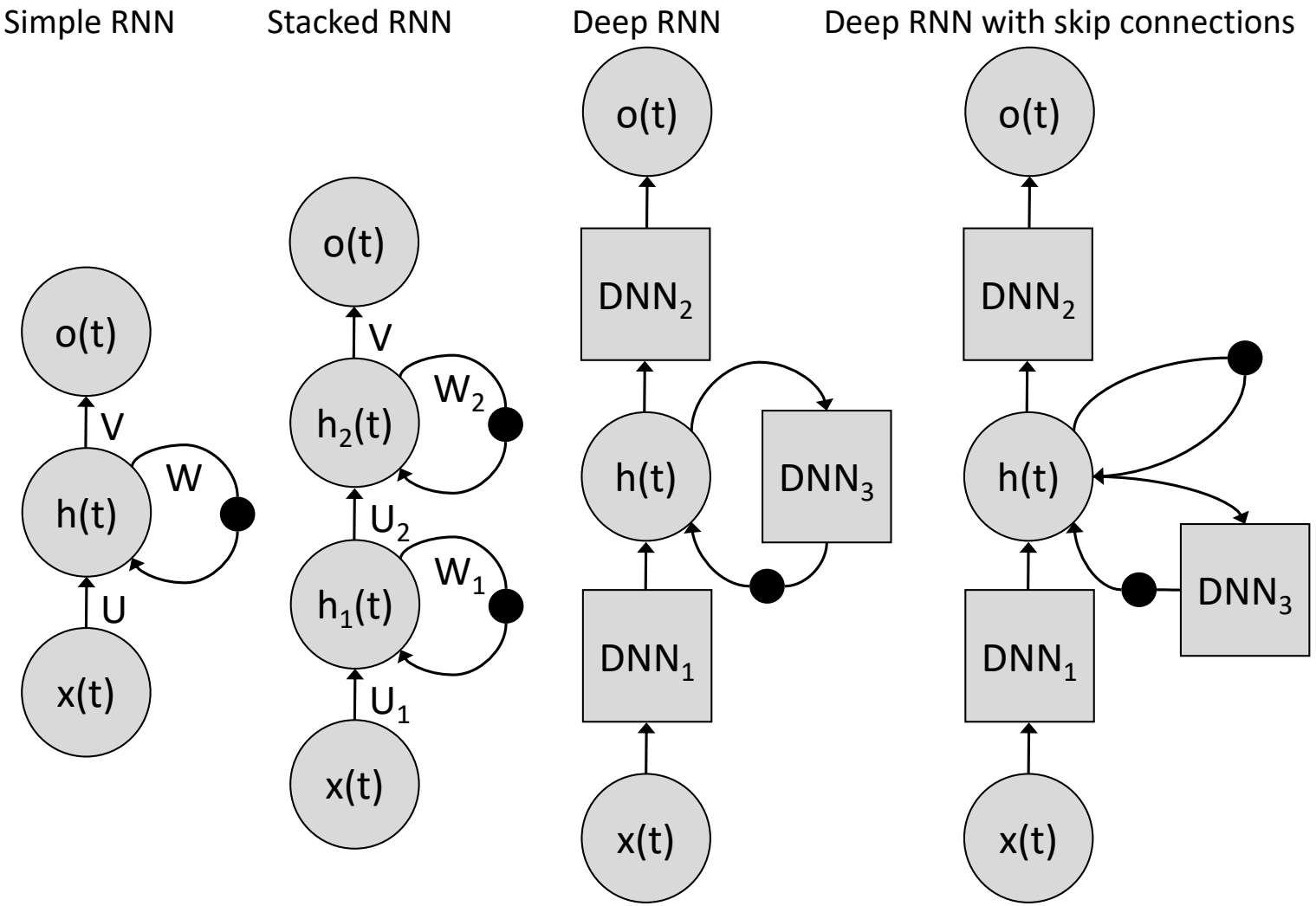
$$\log p(\mathbf{y}(1), \mathbf{y}(2) \mid \mathbf{x}(1), \mathbf{x}(2)) = \log p(\mathbf{y}(2), \mid \mathbf{y}(1), \mathbf{x}(1), \mathbf{x}(2)) + \log p(\mathbf{y}(1) \mid \mathbf{x}(1), \mathbf{x}(2))$$

- Test / production time: Feedback is $\mathbf{o}(t) \to \mathbf{h}(t+1)$.

Multilayer (stacked) RNNs

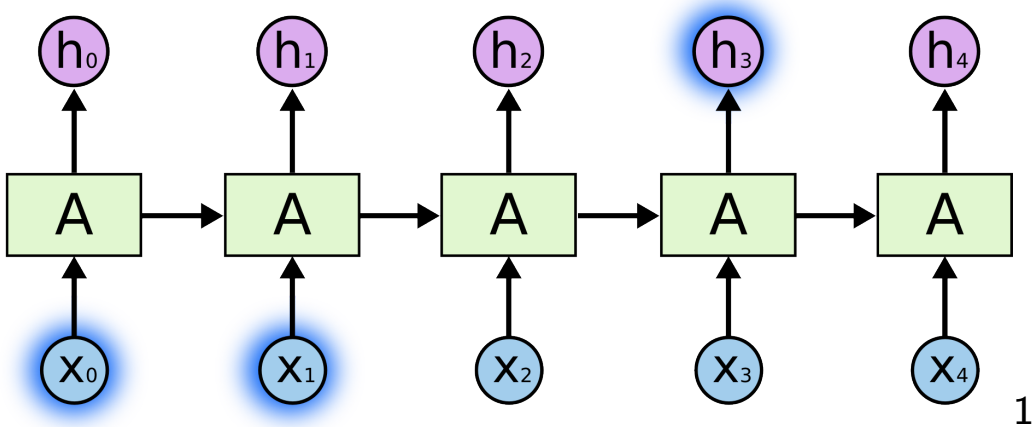


Deep RNNs


Simple RNN
Stacked RNN
Deep RNN
Deep RNN with skip connections
o(t)
V
h(t)
W
U
x(t)
o(t)
V
h2(t)
W2
U2
h1(t)
W1
U1
x(t)
o(t)
DNN2
h(t)
DNN3
DNN1
x(t)
o(t)
DNN2
h(t)
DNN3
DNN1
x(t)

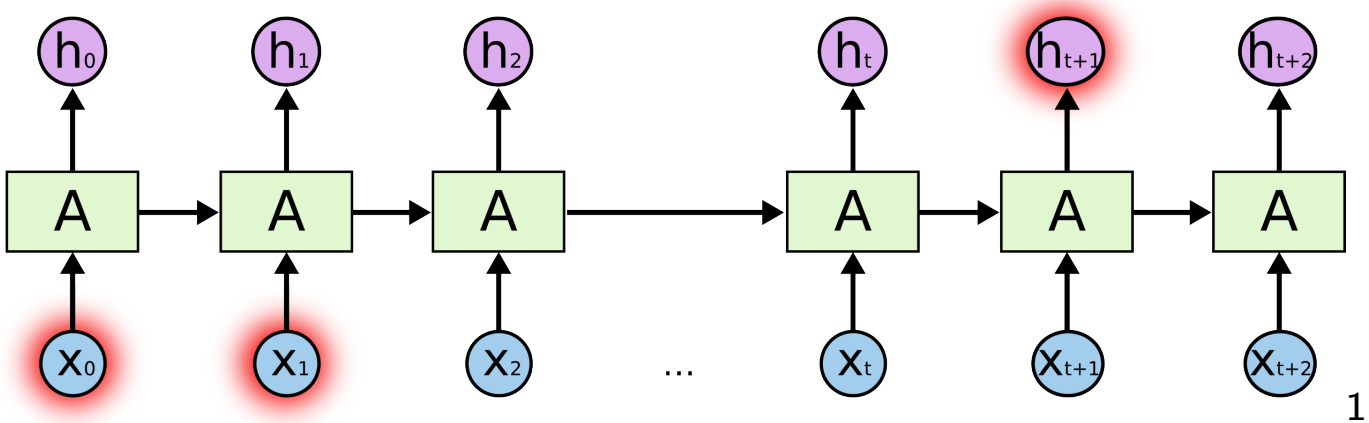
RNNs are deep – gradient annihilation

short "time" dependency: *The clouds are in the sky.*



1

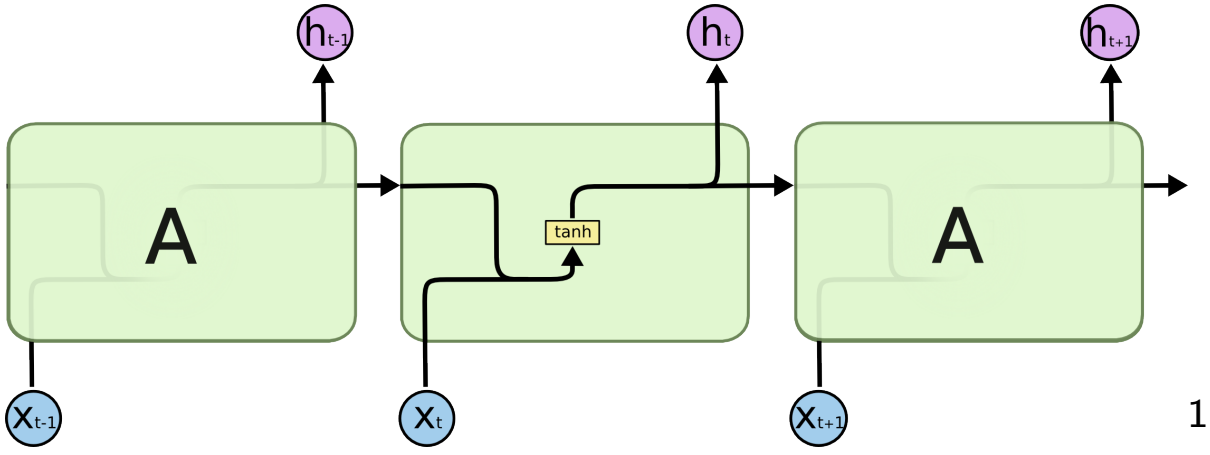
long "time" dependency: *I was born in France. Over the years, I have learned many foreign languages, but my native language is french.*



1

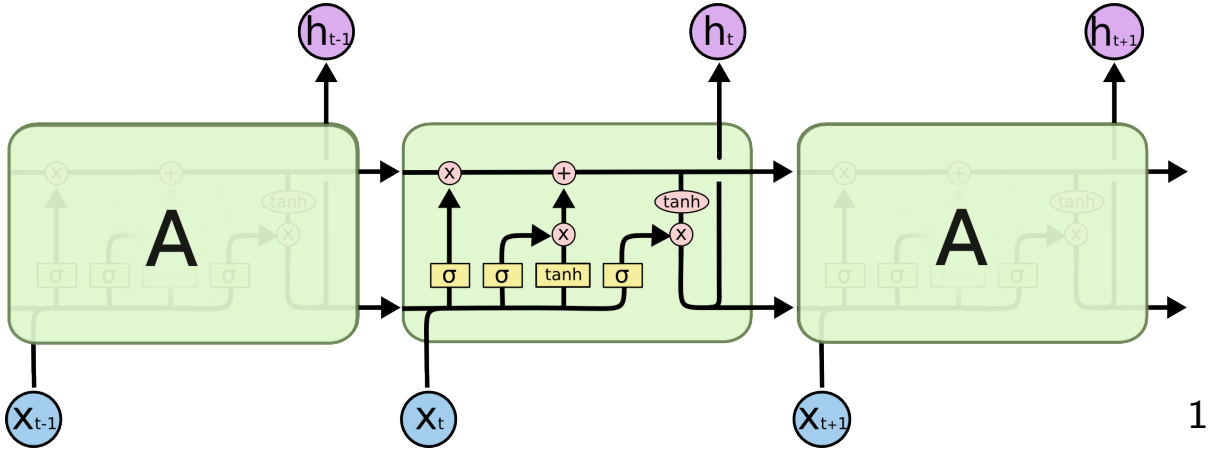
RNN structures

Simple RNN

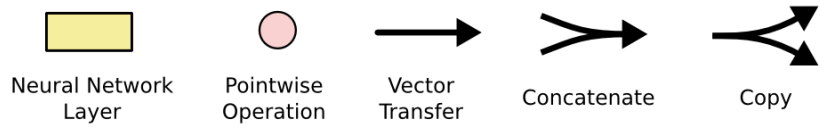


1

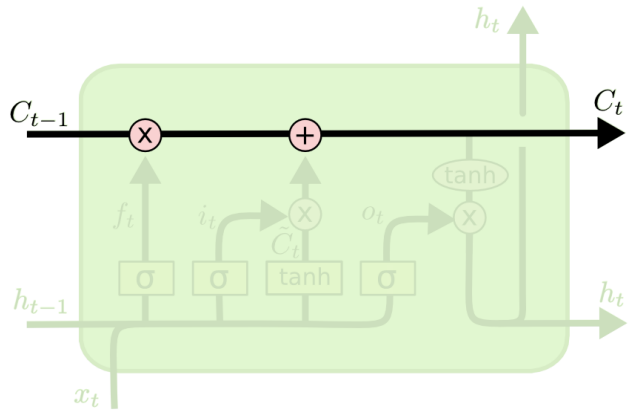
Long Short Time Memory (**LSTM**, Hochreiter & Schmidhuber 1997)



1

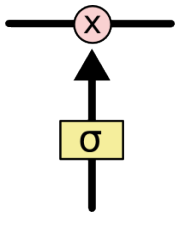


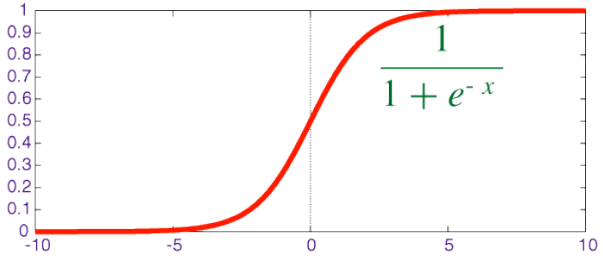
Core idea: cell state. Like a "conveyor belt"



1

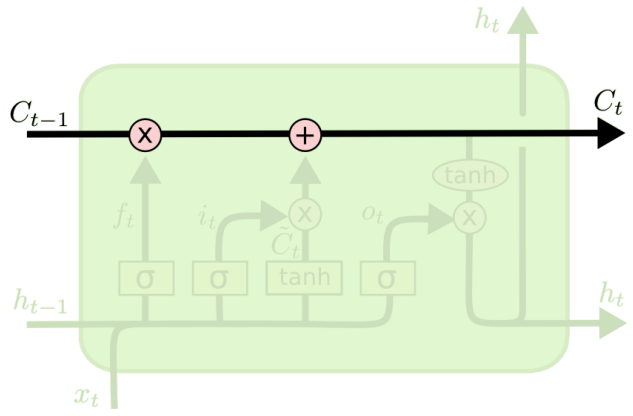
- Carries a signal downstream, with minor linear interactions.
- Easy for information to flow unchanged, avoids gradient annihilation.
- LSTM can remove or add information to the cell state.
- Gates regulate which information passes. Composed of sigmoid layer and pointwise multiplication:





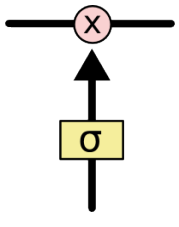
LSTM

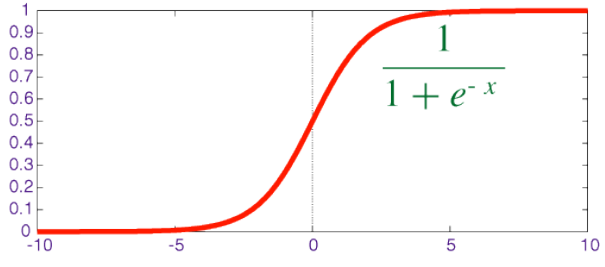
Core idea: cell state. Like a "conveyor belt"



1

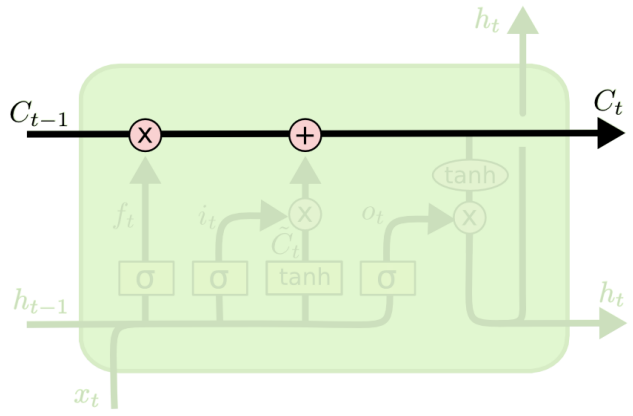
- Carries a signal downstream, with minor linear interactions.
- Easy for information to flow unchanged, avoids gradient annihilation.
- LSTM can remove or add information to the cell state.
- Gates regulate which information passes. Composed of sigmoid layer and pointwise multiplication:





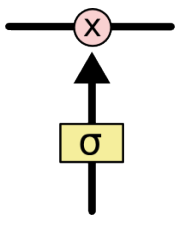
LSTM

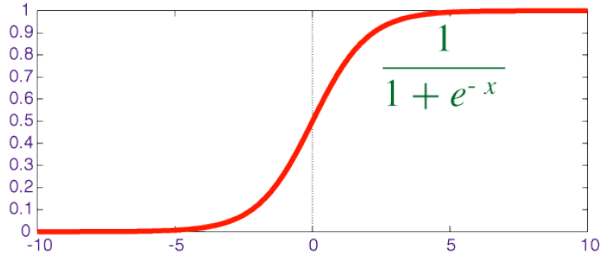
Core idea: cell state. Like a "conveyor belt"



1

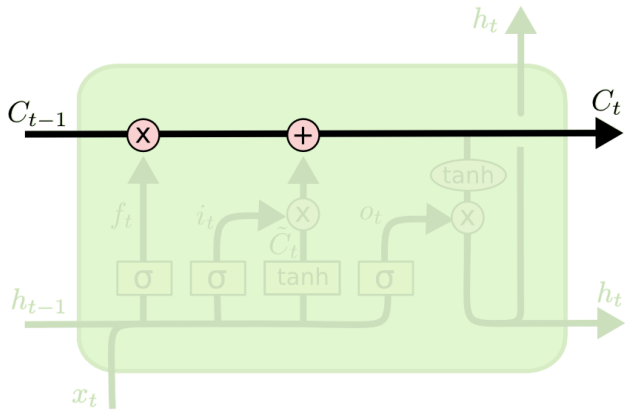
- Carries a signal downstream, with minor linear interactions.
- Easy for information to flow unchanged, avoids gradient annihilation.
- LSTM can remove or add information to the cell state.
- Gates regulate which information passes. Composed of sigmoid layer and pointwise multiplication:





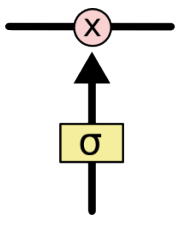
LSTM

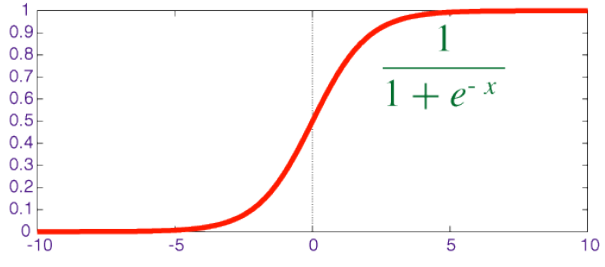
Core idea: cell state. Like a "conveyor belt"



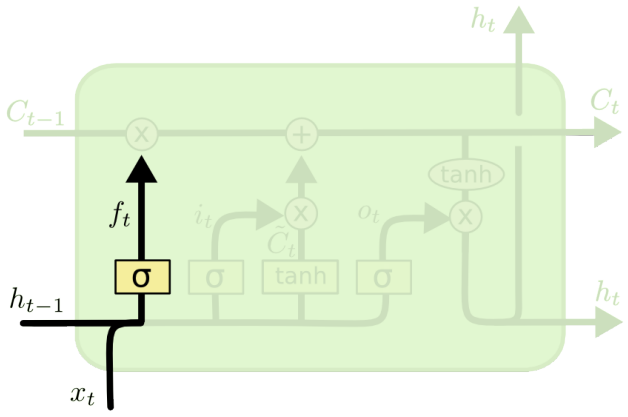
1

- Carries a signal downstream, with minor linear interactions.
- Easy for information to flow unchanged, avoids gradient annihilation.
- LSTM can remove or add information to the cell state.
- Gates regulate which information passes. Composed of sigmoid layer and pointwise multiplication:





Forget gate layer

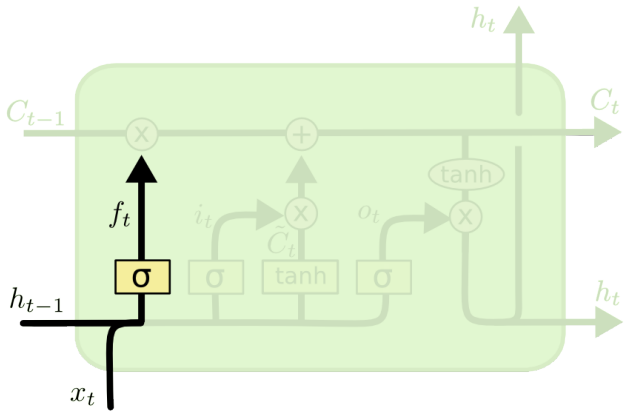


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

1

- Purpose: **Keep / discard part of cell state**
- Given h_{t-1} and x_t, output a number $[0, 1]$ for each number in cell state C_{t-1}:
  - 1: "completely keep this"
  - 0: "completely forget about this."
- **Example**: word-based text prediction. Cell state might include the gender of the current subject to use the correct pronouns. When we see a new subject, forget the gender of the old subject.

Forget gate layer

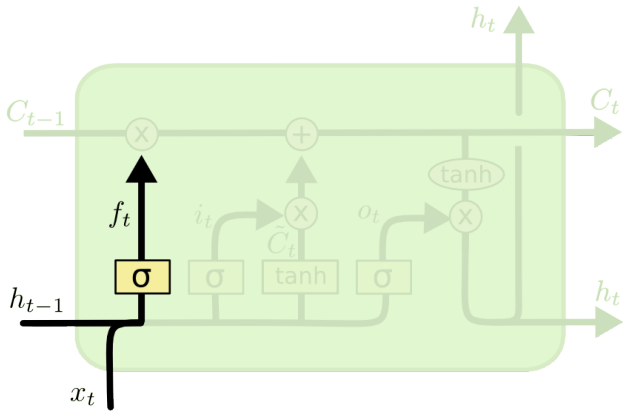


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

1

- Purpose: **Keep / discard part of cell state**
- Given h_{t-1} and x_t, output a number $[0, 1]$ for each number in cell state C_{t-1}:
  - 1: "completely keep this"
  - 0: "completely forget about this."
- **Example**: word-based text prediction. Cell state might include the gender of the current subject to use the correct pronouns. When we see a new subject, forget the gender of the old subject.

Forget gate layer

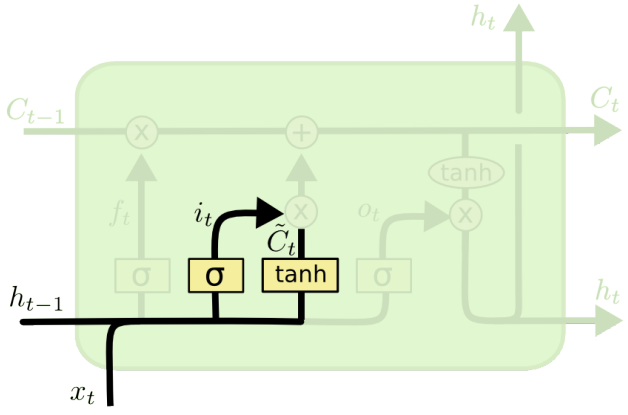


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

1

- Purpose: **Keep / discard part of cell state**
- Given h_{t-1} and x_t, output a number $[0, 1]$ for each number in cell state C_{t-1}:
  - 1: "completely keep this"
  - 0: "completely forget about this."
- **Example**: word-based text prediction. Cell state might include the gender of the current subject to use the correct pronouns. When we see a new subject, forget the gender of the old subject.

Prepare new cell state

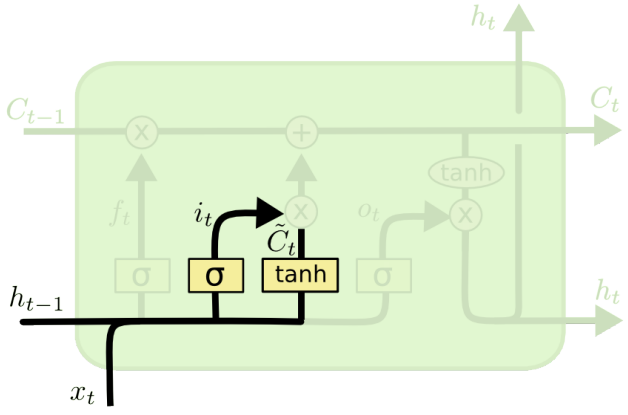


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

1

- What new information to store in cell state?
- **Input gate layer**: Sigmoid layer decides which values will be updated.
- tanh layer creates a vector of new candidate values $\tilde{C}_t$
- **Example**: word-based text prediction: adopt gender of the new subject in the new cell state, to replace the old one we're forgetting.

Prepare new cell state

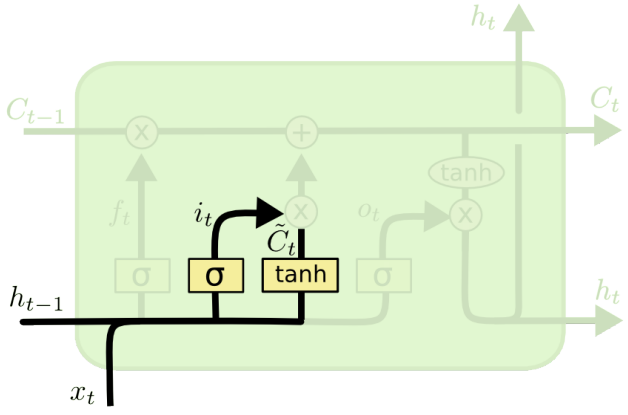


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

1

- What new information to store in cell state?
- **Input gate layer**: Sigmoid layer decides which values will be updated.
- tanh layer creates a vector of new candidate values $\tilde{C}_t$
- **Example**: word-based text prediction: adopt gender of the new subject in the new cell state, to replace the old one we're forgetting.

Prepare new cell state

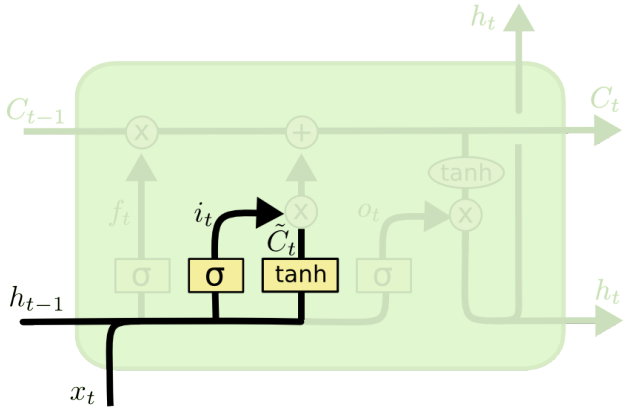


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

1

- What new information to store in cell state?
- **Input gate layer**: Sigmoid layer decides which values will be updated.
- tanh layer creates a vector of new candidate values $\tilde{C}_t$
- **Example**: word-based text prediction: adopt gender of the new subject in the new cell state, to replace the old one we're forgetting.

Prepare new cell state

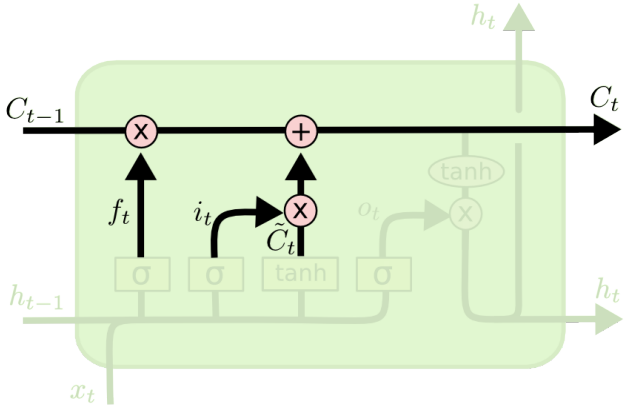


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

1

- What new information to store in cell state?
- **Input gate layer**: Sigmoid layer decides which values will be updated.
- tanh layer creates a vector of new candidate values $\tilde{C}_t$
- **Example**: word-based text prediction: adopt gender of the new subject in the new cell state, to replace the old one we're forgetting.

Update cell state

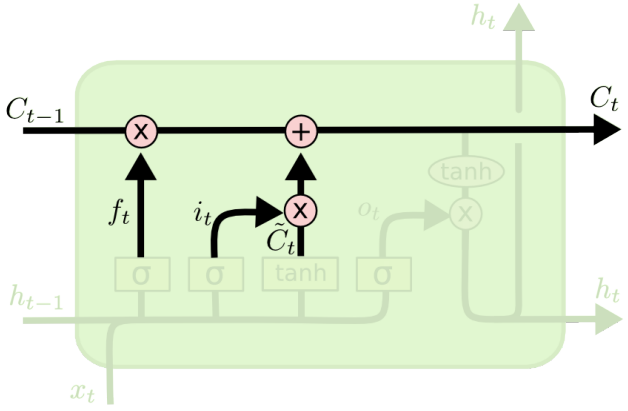


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

1

- Perform update of old cell state, C_{t-1}, into new cell state C_t
- Forget: $f_t * C_{t-1}$
- Add new information: $i_t * \tilde{C}_t$
- **Example**: word-based text prediction: drop the information about the old subject's gender, add the new gender.

Update cell state

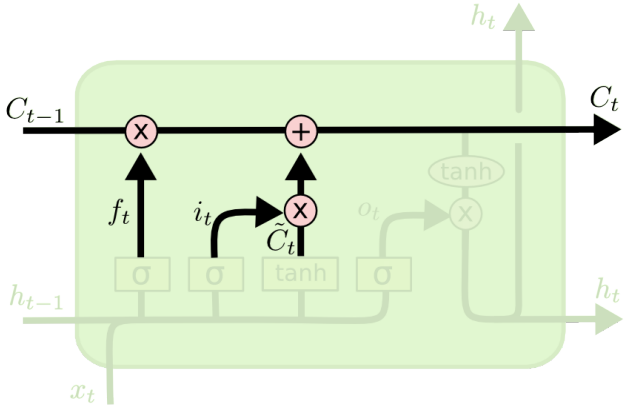


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

1

- Perform update of old cell state, C_{t-1}, into new cell state C_t
- Forget: $f_t * C_{t-1}$
- Add new information: $i_t * \tilde{C}_t$
- **Example**: word-based text prediction: drop the information about the old subject's gender, add the new gender.

Update cell state

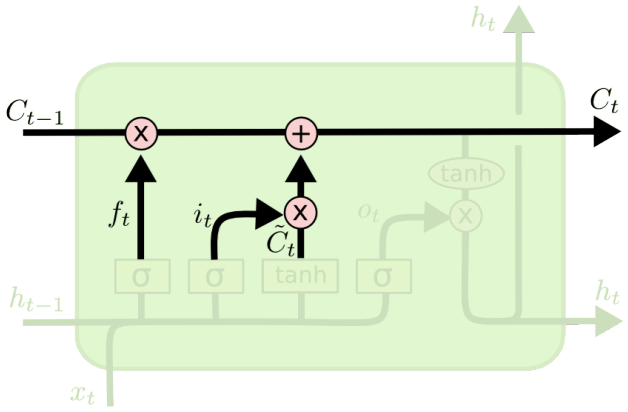


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

1

- Perform update of old cell state, C_{t-1}, into new cell state C_t
- Forget: $f_t * C_{t-1}$
- Add new information: $i_t * \tilde{C}_t$
- **Example**: word-based text prediction: drop the information about the old subject's gender, add the new gender.

Update cell state

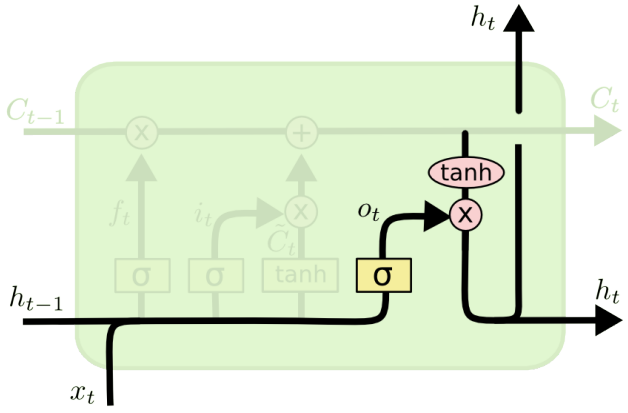


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

1

- Perform update of old cell state, C_{t-1}, into new cell state C_t
- Forget: $f_t * C_{t-1}$
- Add new information: $i_t * \tilde{C}_t$
- **Example**: word-based text prediction: drop the information about the old subject's gender, add the new gender.

Output



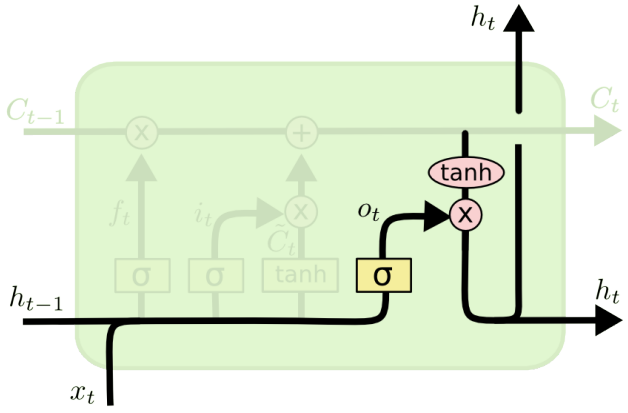
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

1

- Output o_t: filtered version of our hidden state.
- New hidden state:
  - put cell state through tanh (bounds values by $[-1, 1]$)
  - multiply result by o_t, to save only keep output-relevant parts of cell state in h_t.
- **Example**: word-based text prediction: we just saw a subject, now we might want to output information relevant to the subsequent verb. E.g., it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

Output



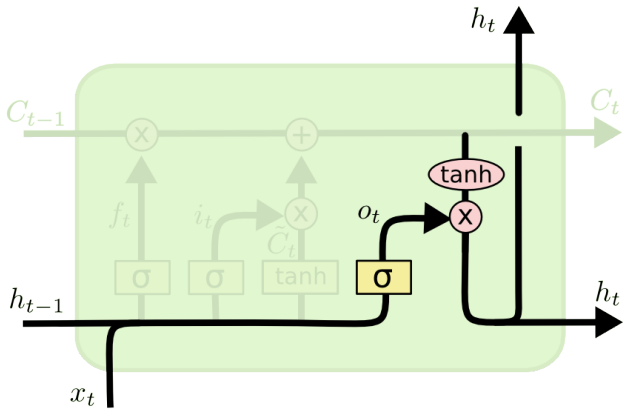
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

1

- Output o_t: filtered version of our hidden state.
- New hidden state:
  - put cell state through tanh (bounds values by $[-1, 1]$)
  - multiply result by o_t, to save only keep output-relevant parts of cell state in h_t.
- **Example**: word-based text prediction: we just saw a subject, now we might want to output information relevant to the subsequent verb. E.g., it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

Output



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

1

- Output o_t: filtered version of our hidden state.
- New hidden state:
  - put cell state through tanh (bounds values by $[-1, 1]$)
  - multiply result by o_t, to save only keep output-relevant parts of cell state in h_t.
- **Example**: word-based text prediction: we just saw a subject, now we might want to output information relevant to the subsequent verb. E.g., it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

- Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- 1 MB text input: http://www.paulgraham.com/articles.html

The surprised in investors weren't going to raise money. I'm not the company with the time there are all interesting quickly, don't have to get off the same programmers. There's a super-angel round fundraising, why do you can do. If you have a different physical investment are become in people who reduced in a startup with the way to argument the acquirer could see them just that you're also the founders will part of users' affords that and an alternation to the idea. [2] Don't work at first member to see the way kids will seem in advance of a bad successful startup. And if you have to act the big company too.

Examples

Generate Shakespeare[1]

- 4.4 MB text input,
- 3-layer RNN, 512 hidden nodes per layer.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour."

- Source: 16 MB LaTeX code *http://stacks.math.columbia.edu/*
- Sample: almost compiles, minor corrections needed.

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciol y in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

Examples

Pretending Math[1]

- Source: 16 MB LaTeX code *http://stacks.math.columbia.edu/*
- Sample: almost compiles, minor corrections needed.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

Lemma 0.3. *In Situation ??. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □