Lecture 8: Recurrent Neural Networks

Shuai Li

John Hopcroft Center, Shanghai Jiao Tong University

https://shuaili8.github.io

https://shuaili8.github.io/Teaching/VE445/index.html

Outline

- Motivation
- Basics of RNN
- Different RNNs
- Application examples
- Training
- LSTM

Motivation

- In traditional NN
    - Assume all inputs and outputs are independent of each other
    - Input and output length are fixed
- But this might be bad for many tasks
    - Predict next word in a sentence
        - “Context”: You better know which words came before it
    - Hard/Impossible to choose a fixed context window
        - Output YES if the number of 1s is even, else NO!
            - 1000010101 – YES, 100011 – NO, …
        - There can always be a new sample longer than anything seen

Basics

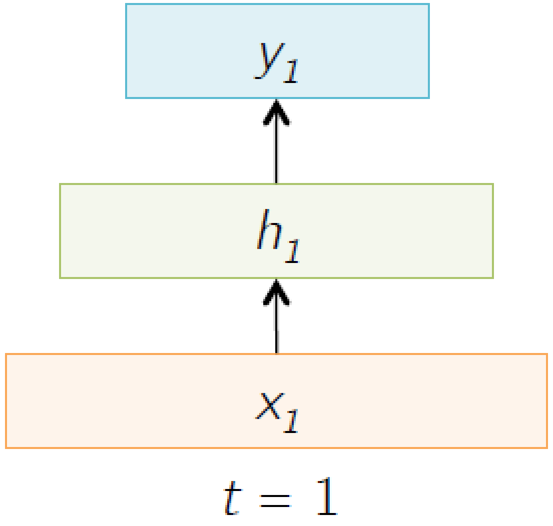
Recurrent neural networks (RNNs)

- Recurrent
  - Perform the same task for every element of a sequence, with the output being dependent on the previous computations

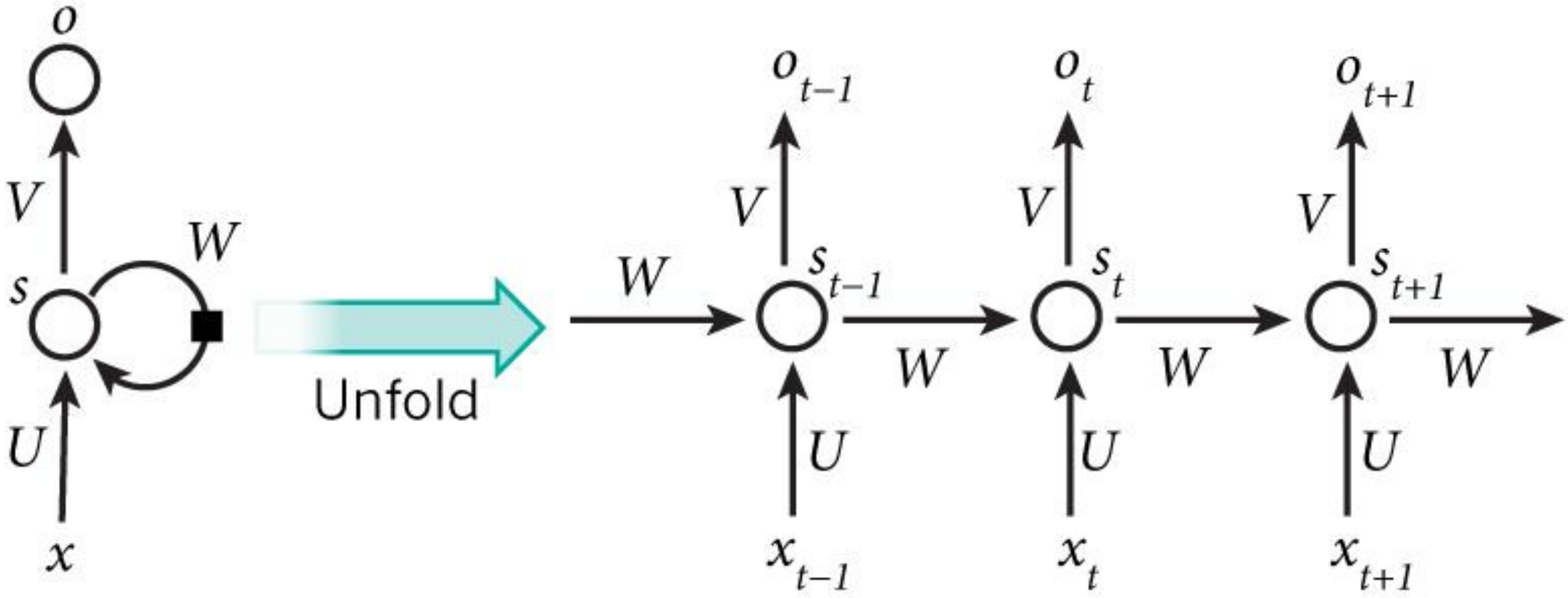
- They have a “memory” which captures information about what has been calculated so far

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

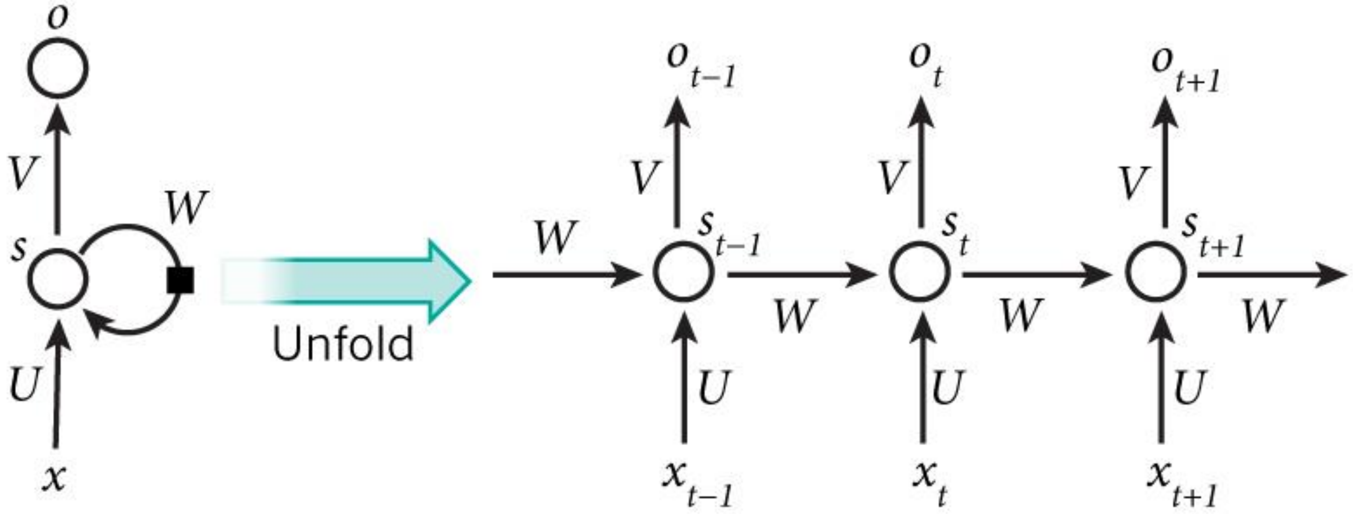
Feed-forward Network



RNN



RNN (cont.)

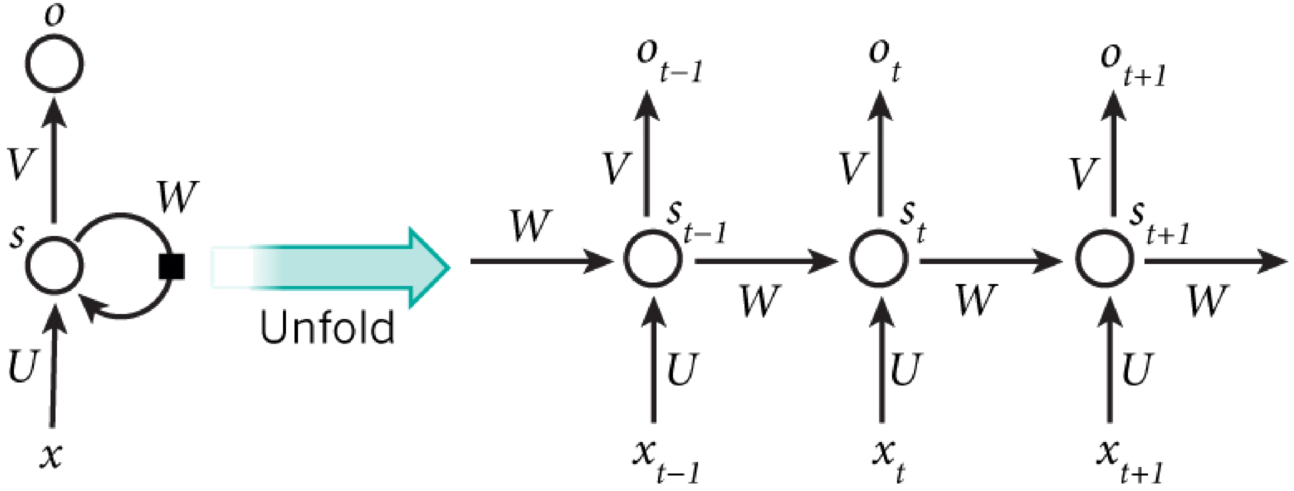


- x_t is the input at time t
- s_t is the hidden state at time t
  - It is the “memory” of the network
  - Is calculated based on previous hidden state and the input at the current step

    $$s_t = f(Ux_t + Ws_{t-1})$$

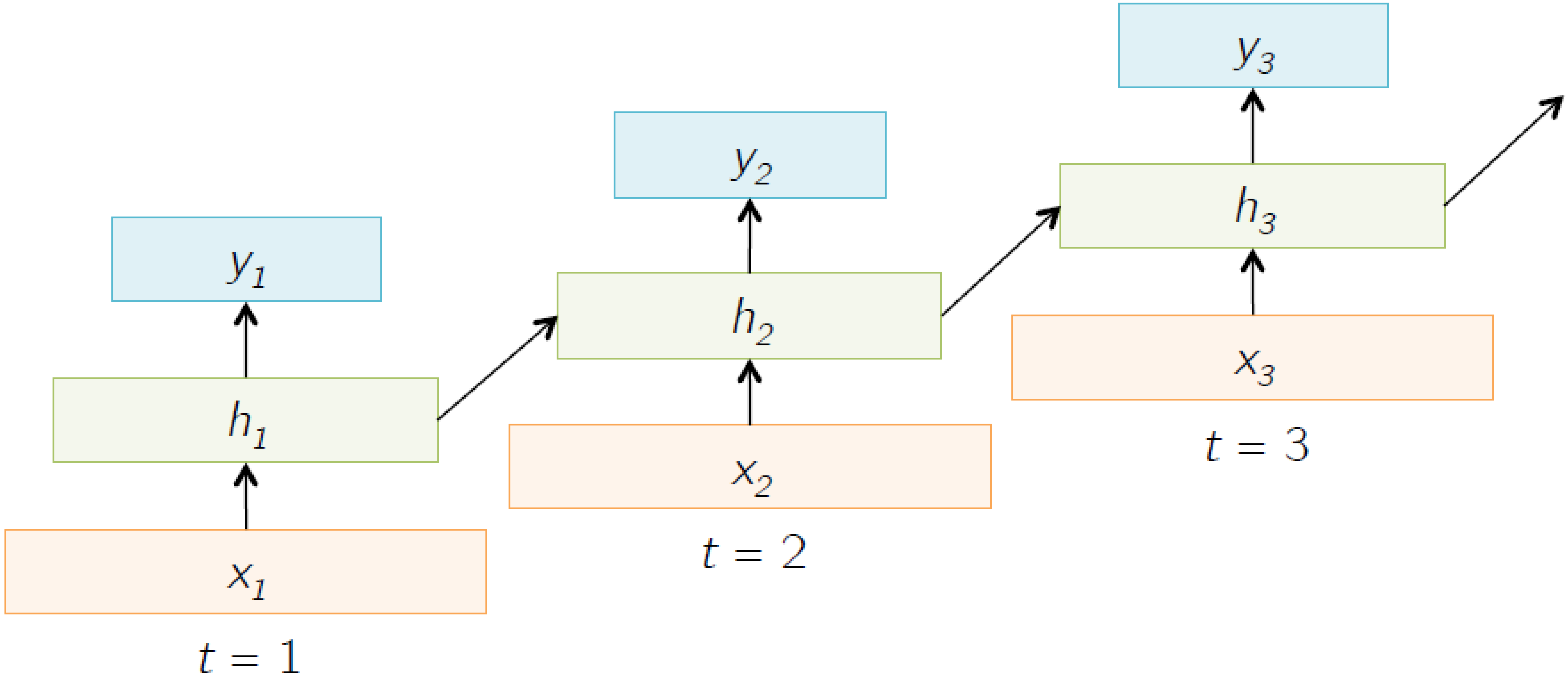
    where f is nonlinear activation function, such as tanh, ReLU
- o_t is the output at time t
  - E.g. If we want to predict the next word in a sentence, o_t is a vector of probabilities over certain vocabulary

RNN (cont.)

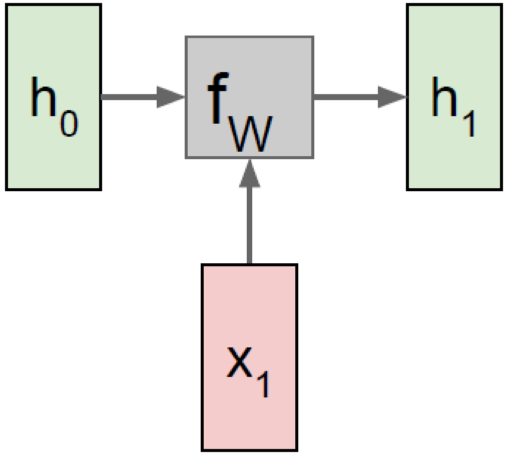


- s_t is the "memory" at time t
- o_t is based on memory at time t
- RNN share weights U and W
  - Unlike traditional NN
  - Greatly reduce the total number of parameters we need to learn
- The output at each time step might be unnecessary
  - E.g. When predicting the sentiment of a sentence we may only care about the final output, not the sentiment after each word
- We may not need inputs at each time step
- Main feature:
  - Is the hidden state, which captures some information about a sequence

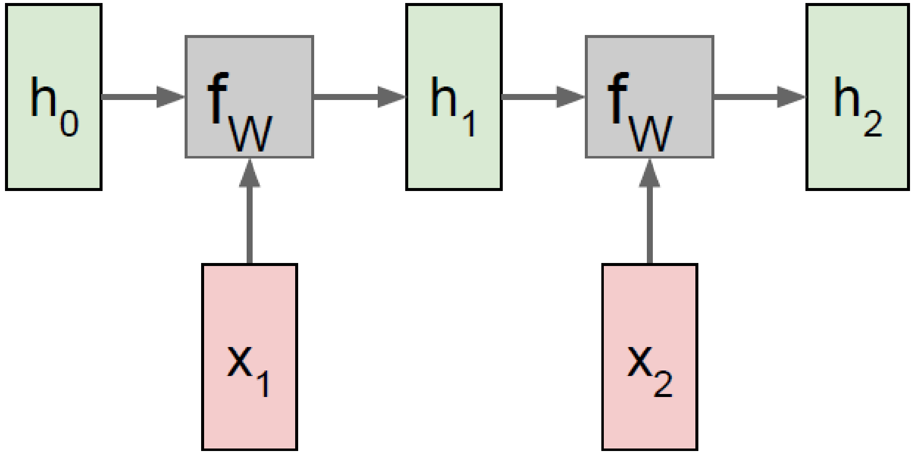
Recall RNN



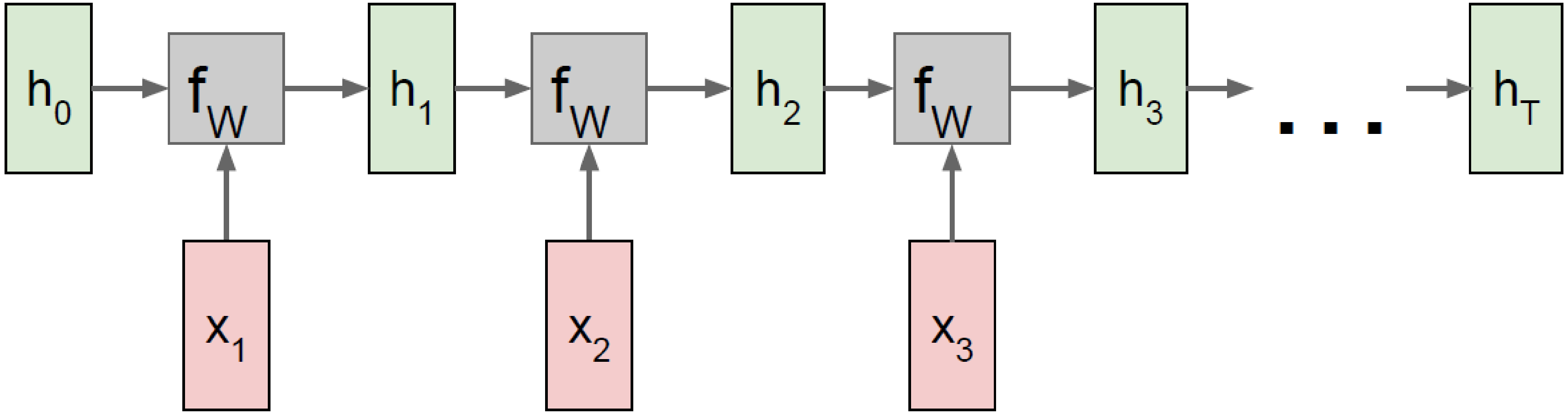
Computational graph of RNN



Computational graph of RNN

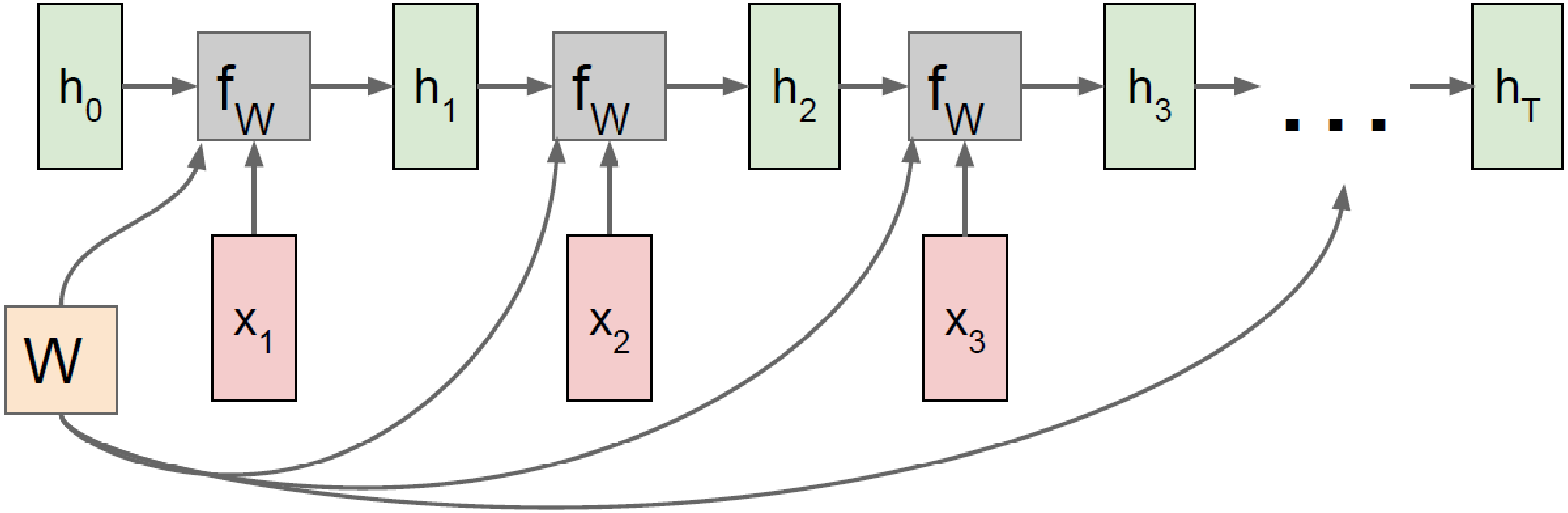


Computational graph of RNN



Computational graph of RNN

- Re-use the same weight matrix at every time-step



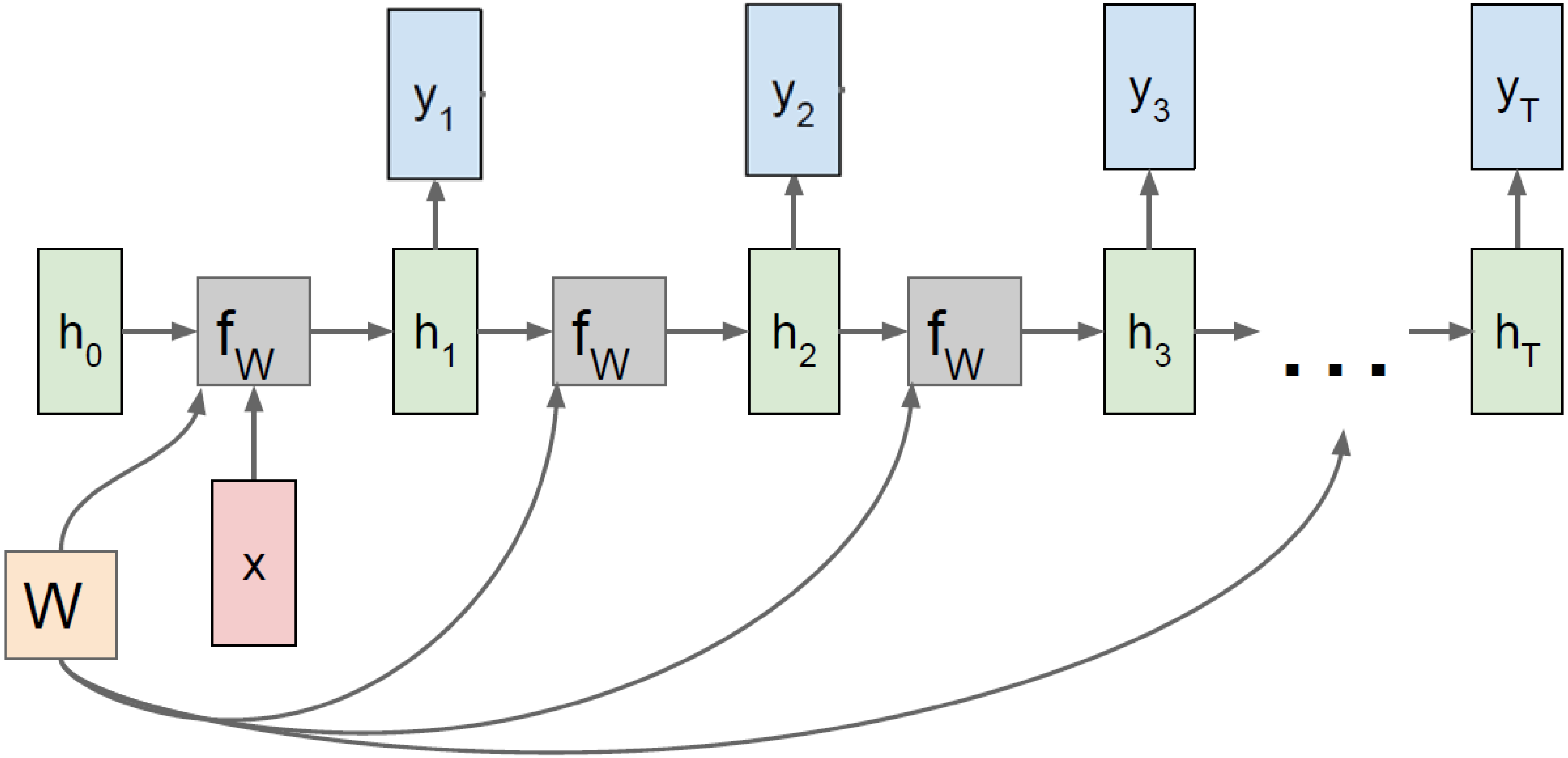
Different RNNs

Different RNNs: one to many

- E.g. Image captioning. Image -> sequence of words

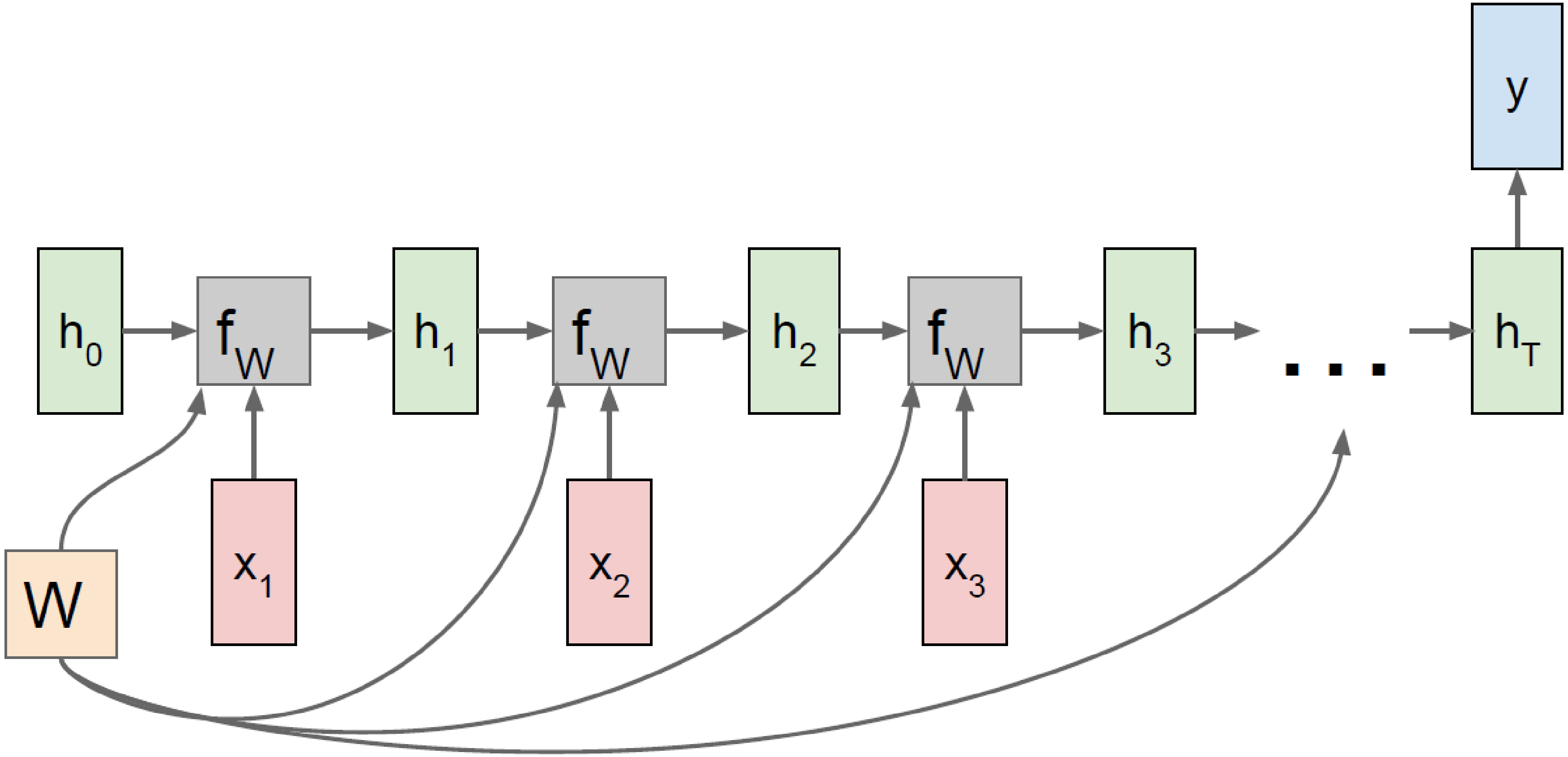
"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

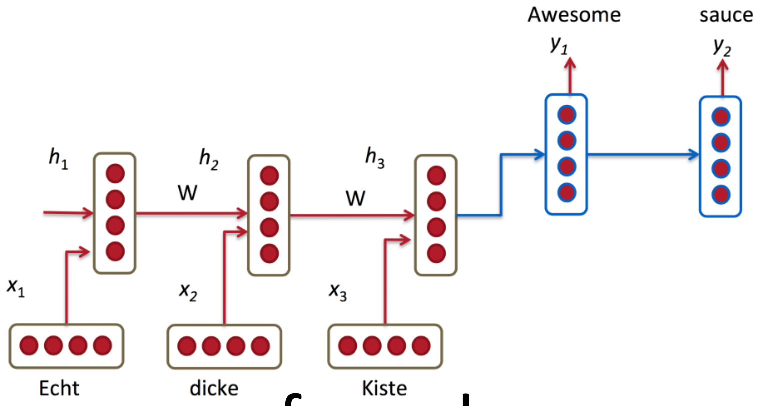


Different RNNs: many to one

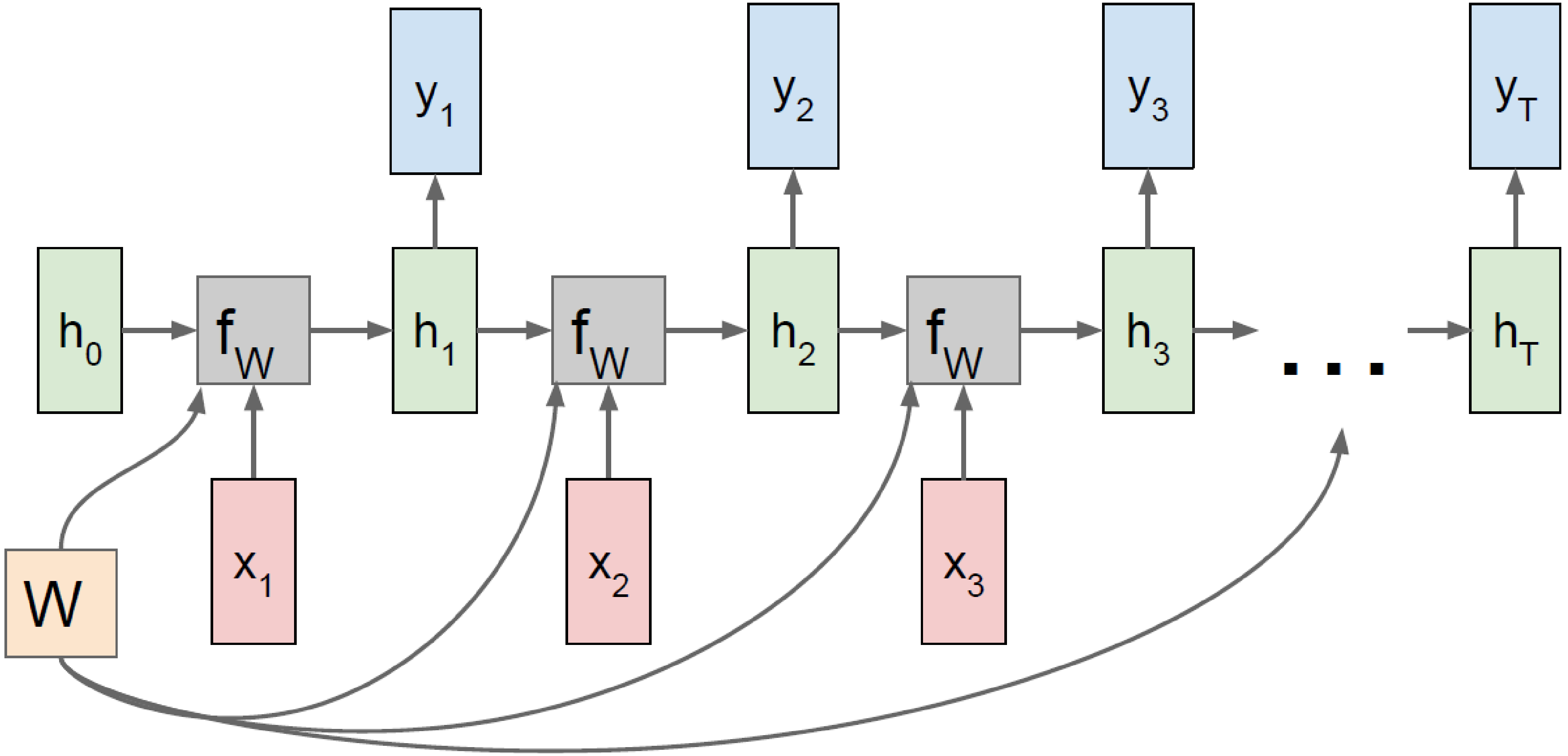
- E.g. Sentiment Classification. Sequence of words -> sentiment



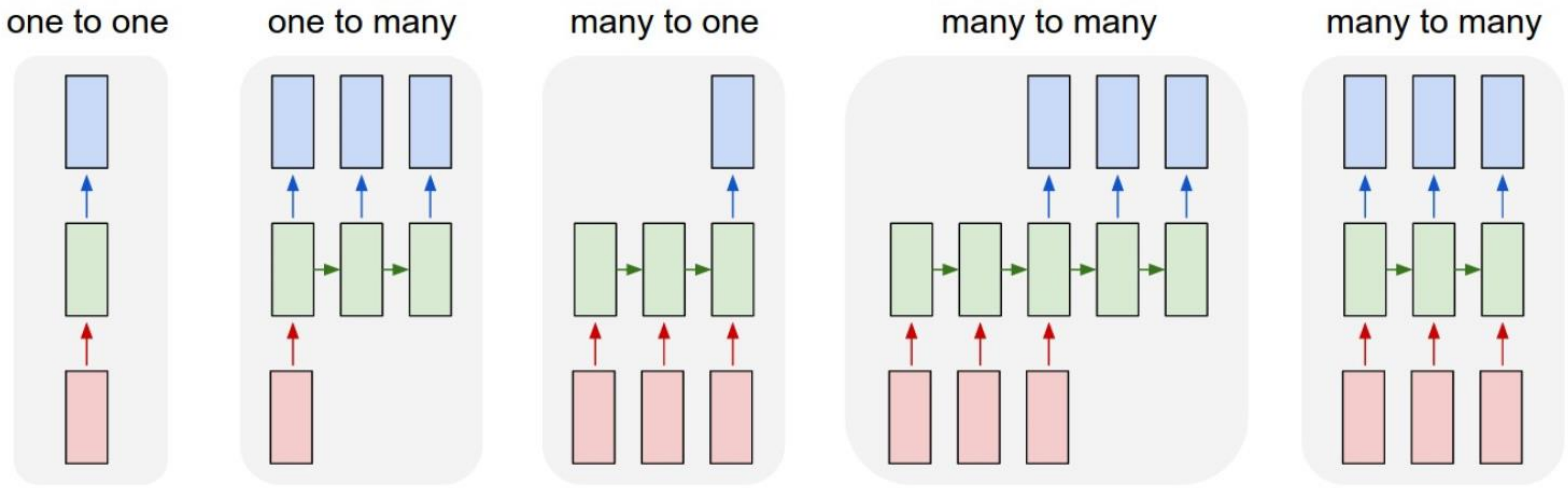
Different RNNs: many to many



- E.g. Machine translation. Sequence of words -> sequence of words



Different RNNs



Vanilla NN

Image captioning
Text generation

Text classification
Sentiment analysis

Machine translation
Dialogue system

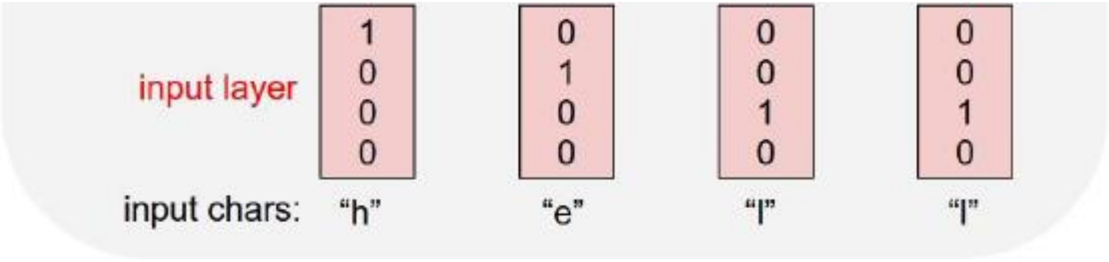
Stock price estimation
Video frame classification

Application Examples

Example 1: Character-level language model

- Given previous words/characters, predict the next
- Vocabulary: [h, e, l, o]
- Example of training sequence:

  “Hello”

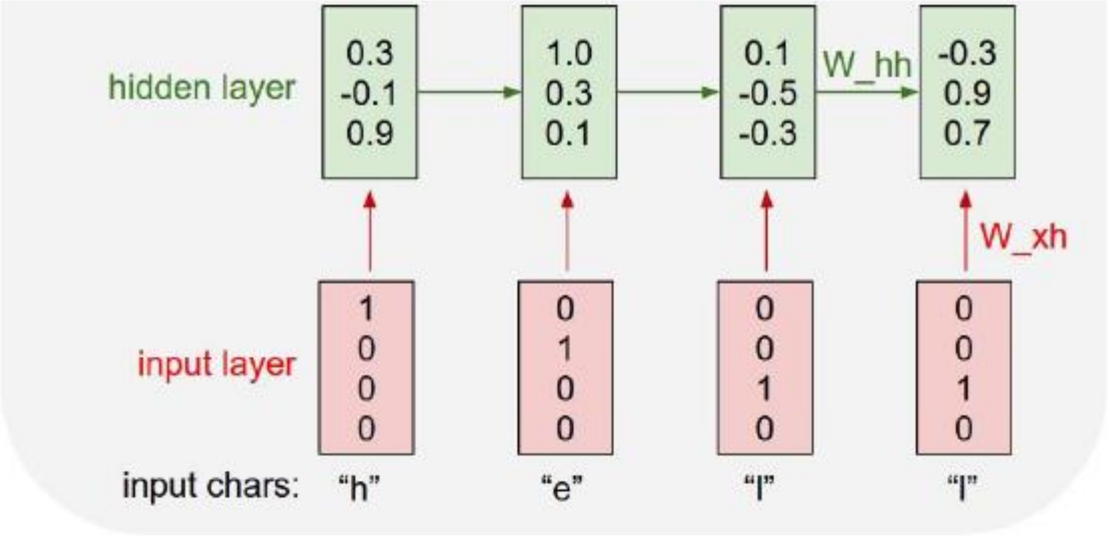


Example 1: Character-level language model

- Given previous words/characters, predict the next
- Vocabulary: [h, e, l, o]
- Example of training sequence:

  “Hello”

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



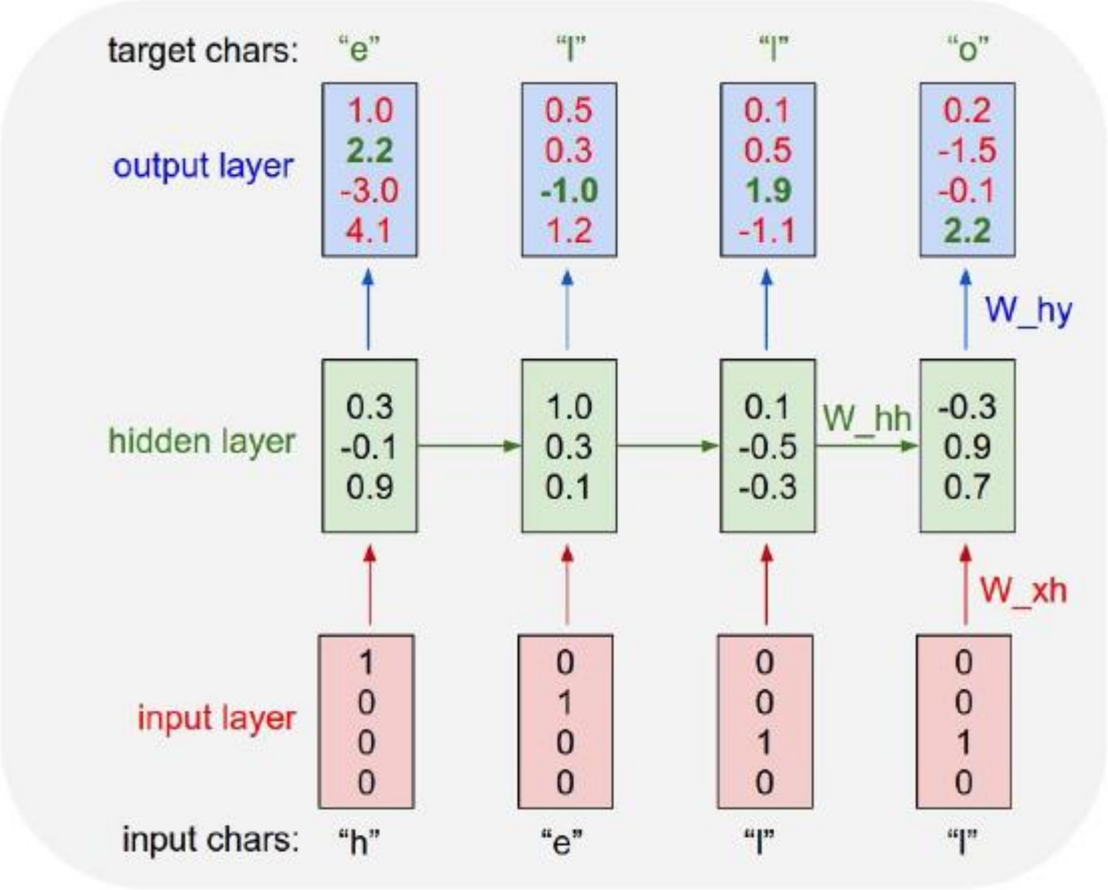
Example 1: Character-level language model

- Given previous words/characters, predict the next
- Vocabulary: [h, e, l, o]
- Example of training sequence:

  "Hello"

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

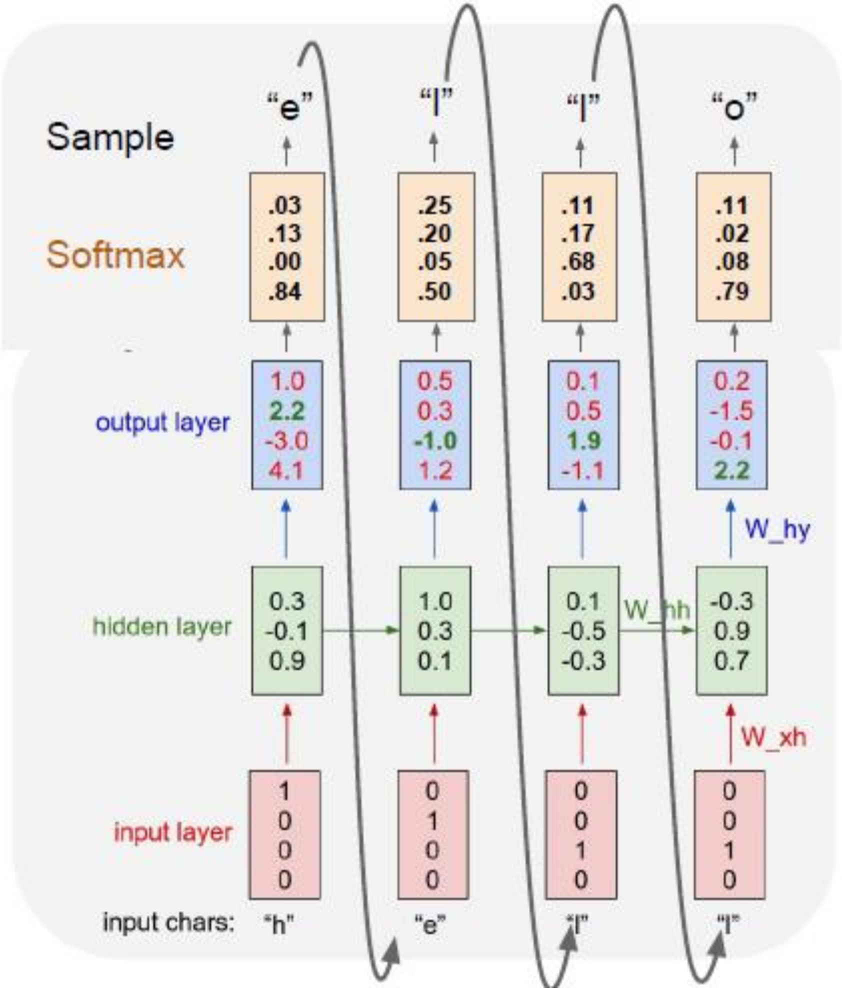


Example 1: Character-level language model

- Given previous words/characters, predict the next
- Vocabulary: [h, e, l, o]
- Example of training sequence:

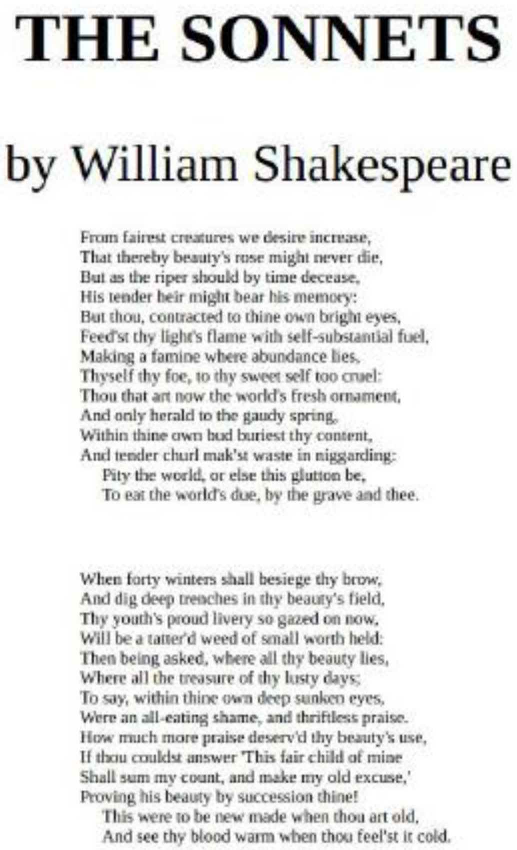
  “Hello”

- At test time, sample characters one at a time, feed back to model



Example 2: Text generating

- Can we generate sonnets like a poet, using RNN?

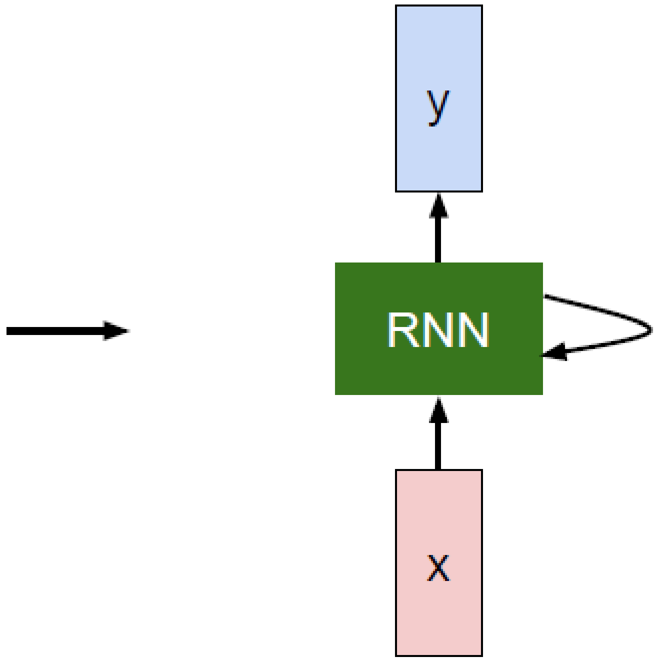


THE SONNETS

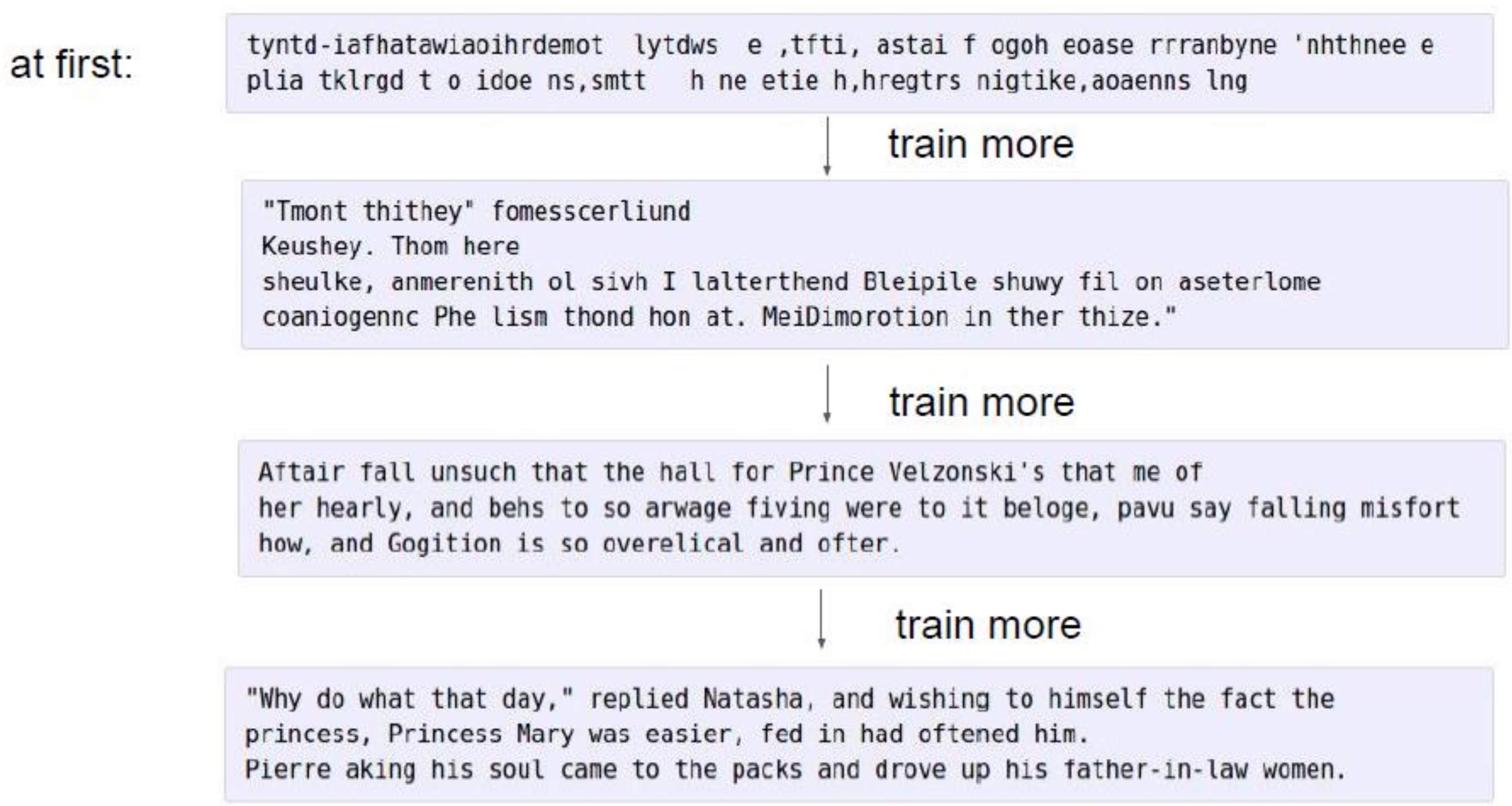
by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
  Pity the world, or else this glutton be,
  To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
  This were to be new made when thou art old,
  And see thy blood warm when thou feel'st it cold.



Example 2: Text generating results



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

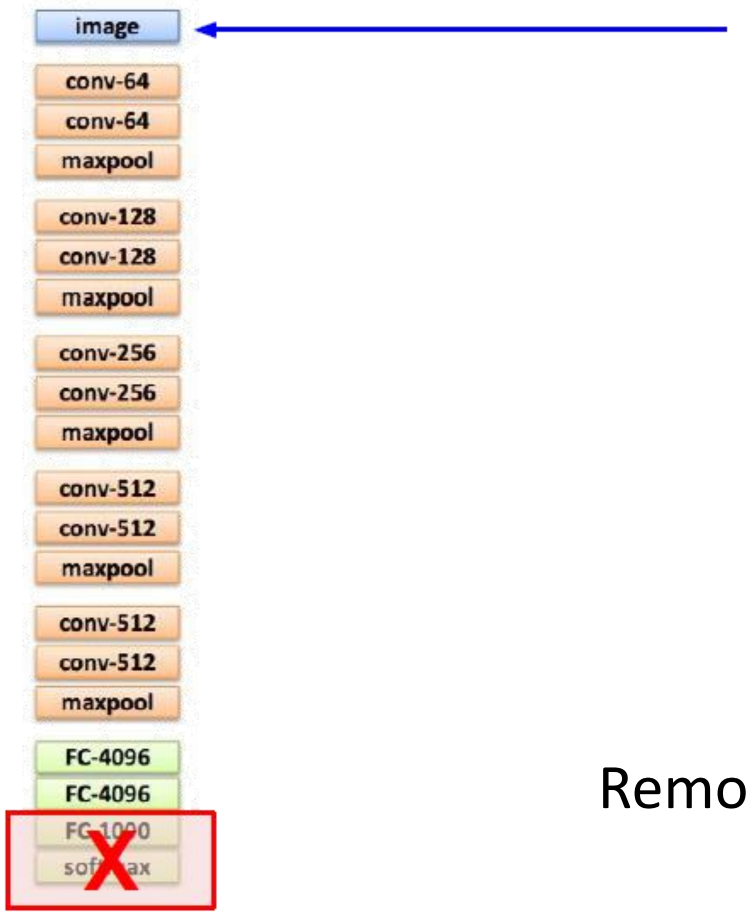
```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

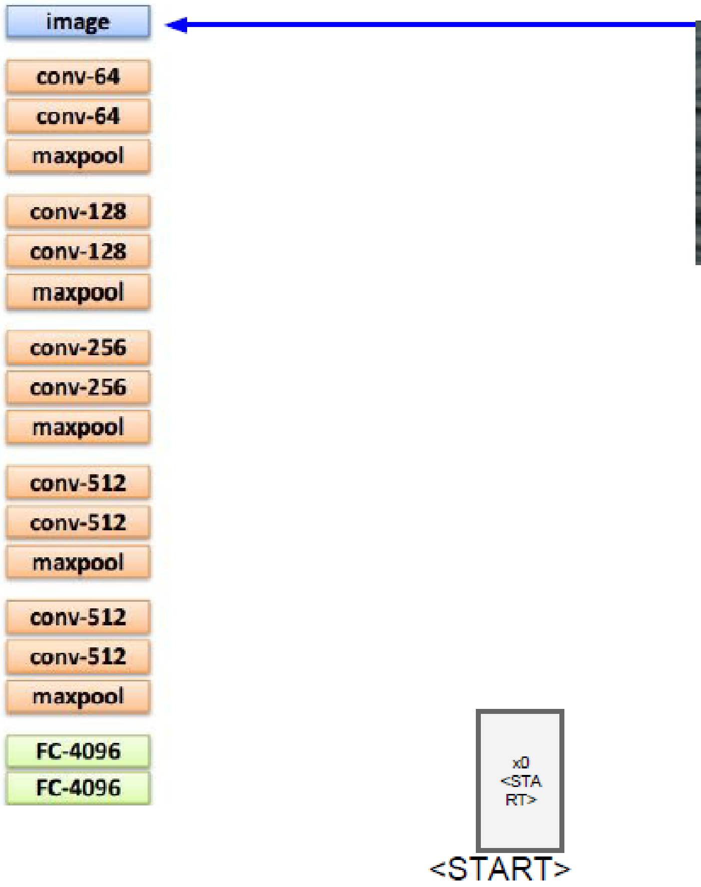
Example 3: Image captioning

- Using RNN to explain the content in an Image



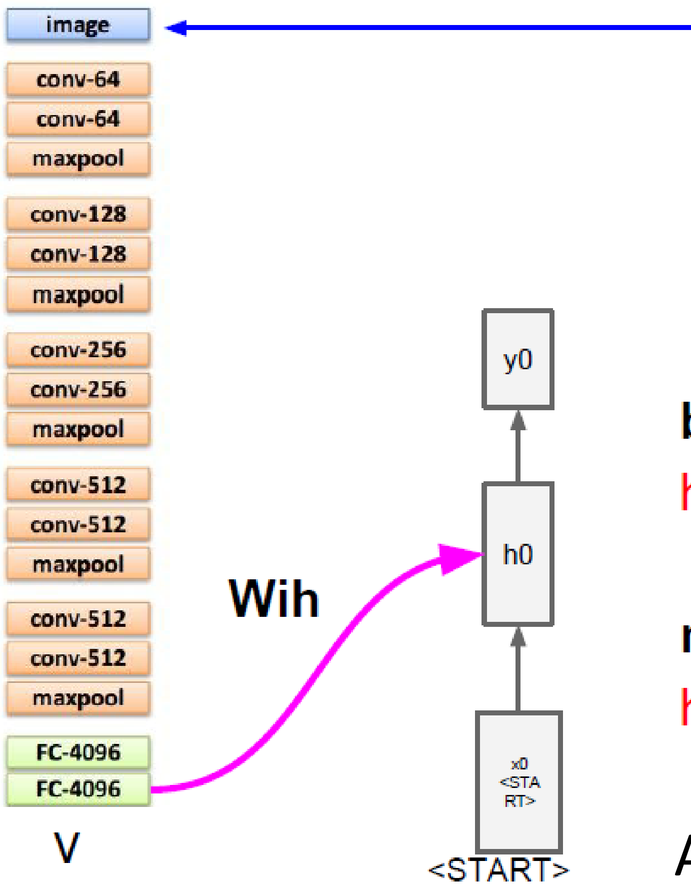
Remove the last two layers in CNN

Example 3: Image captioning (cont.)



Preparing the initial input for RNN

Example 3: Image captioning (cont.)



test image

before:

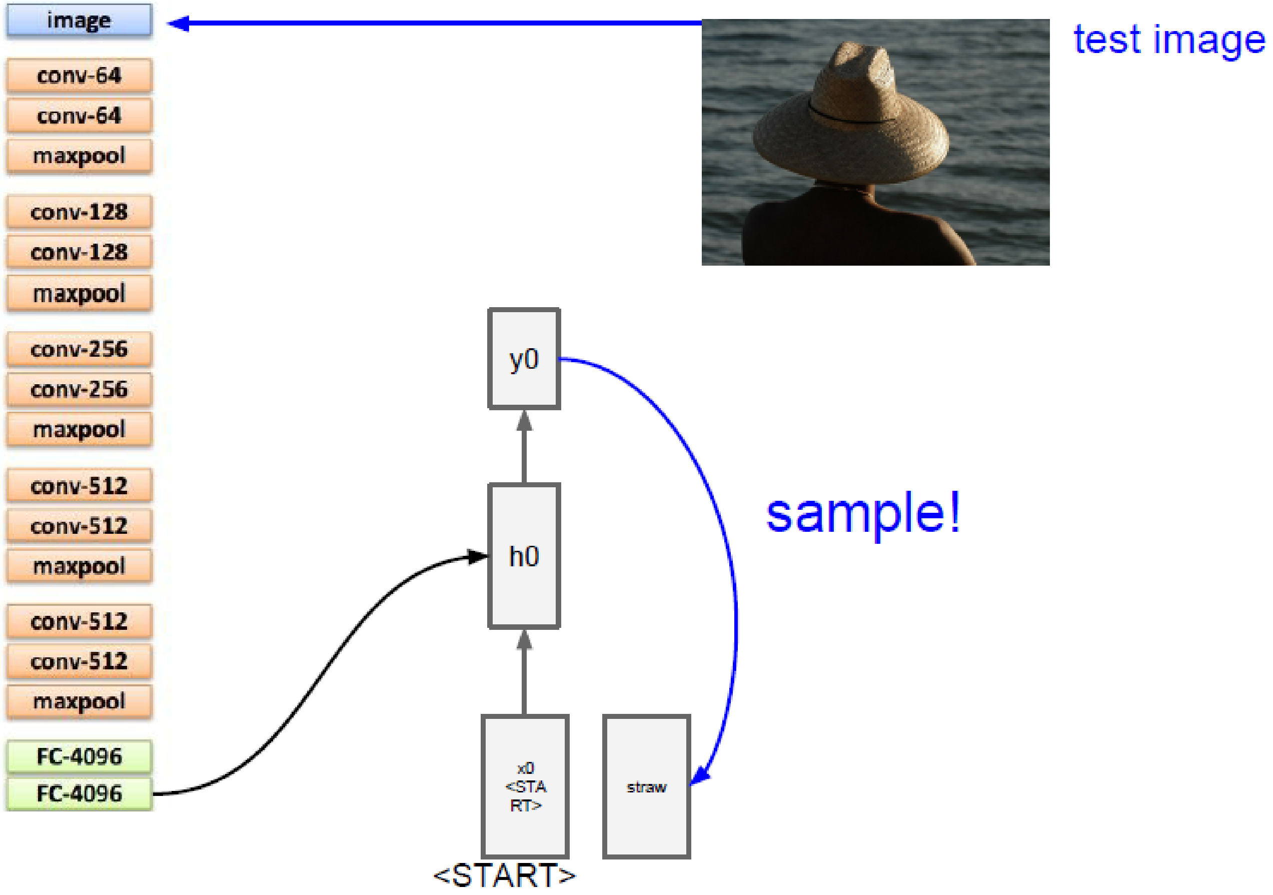
h = tanh(Wxh * x + Whh * h)

now:

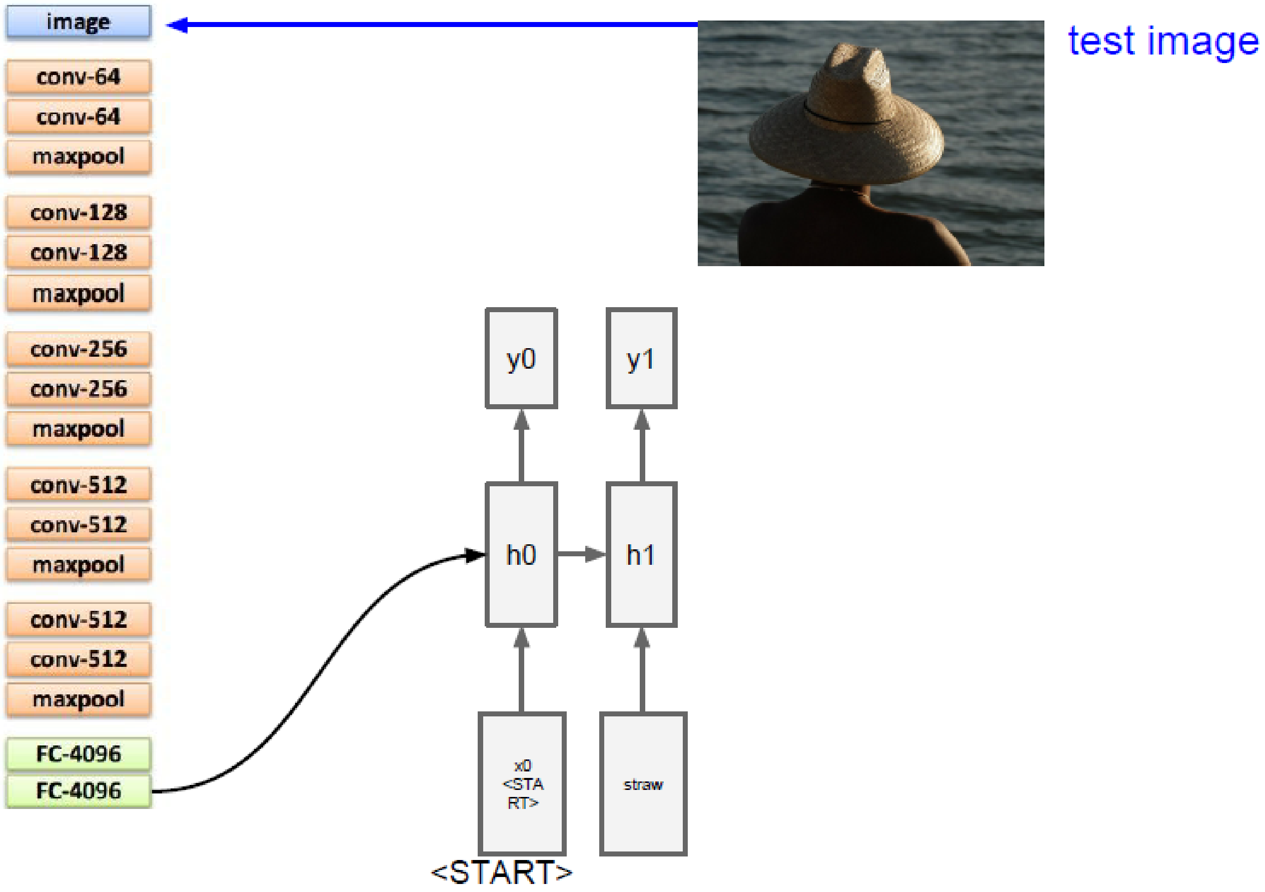
h = tanh(Wxh * x + Whh * h + Wih * v)

Add a new item to connect CNN to RNN

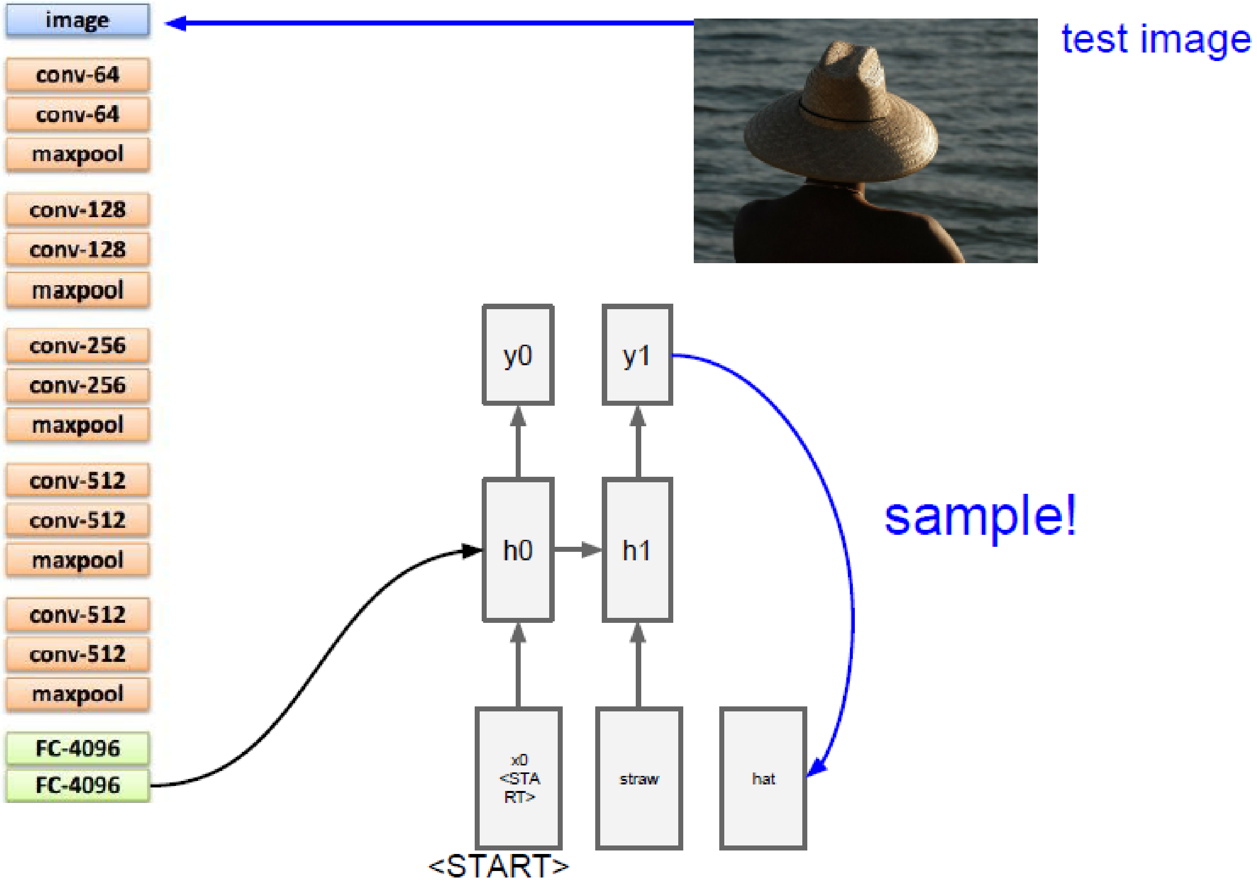
Example 3: Image captioning (cont.)



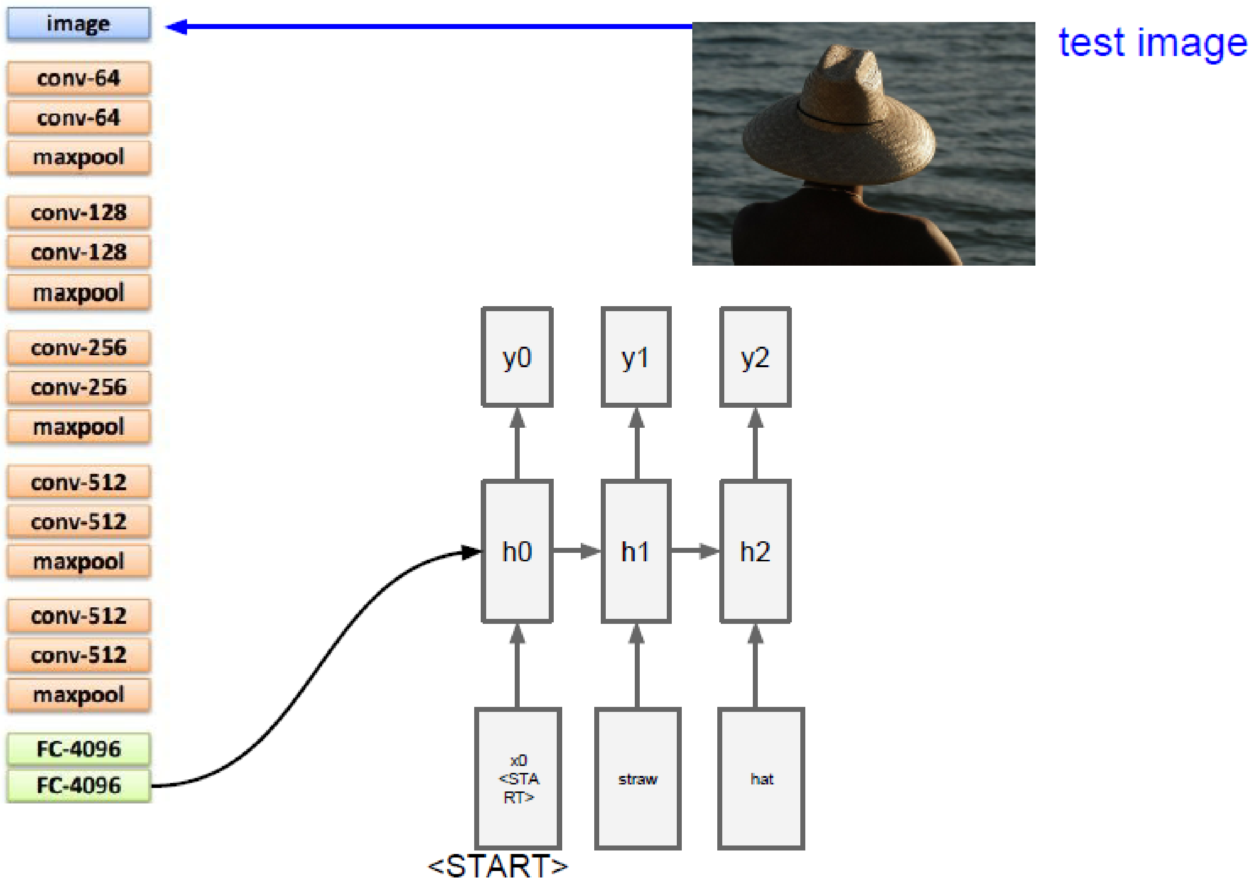
Example 3: Image captioning (cont.)



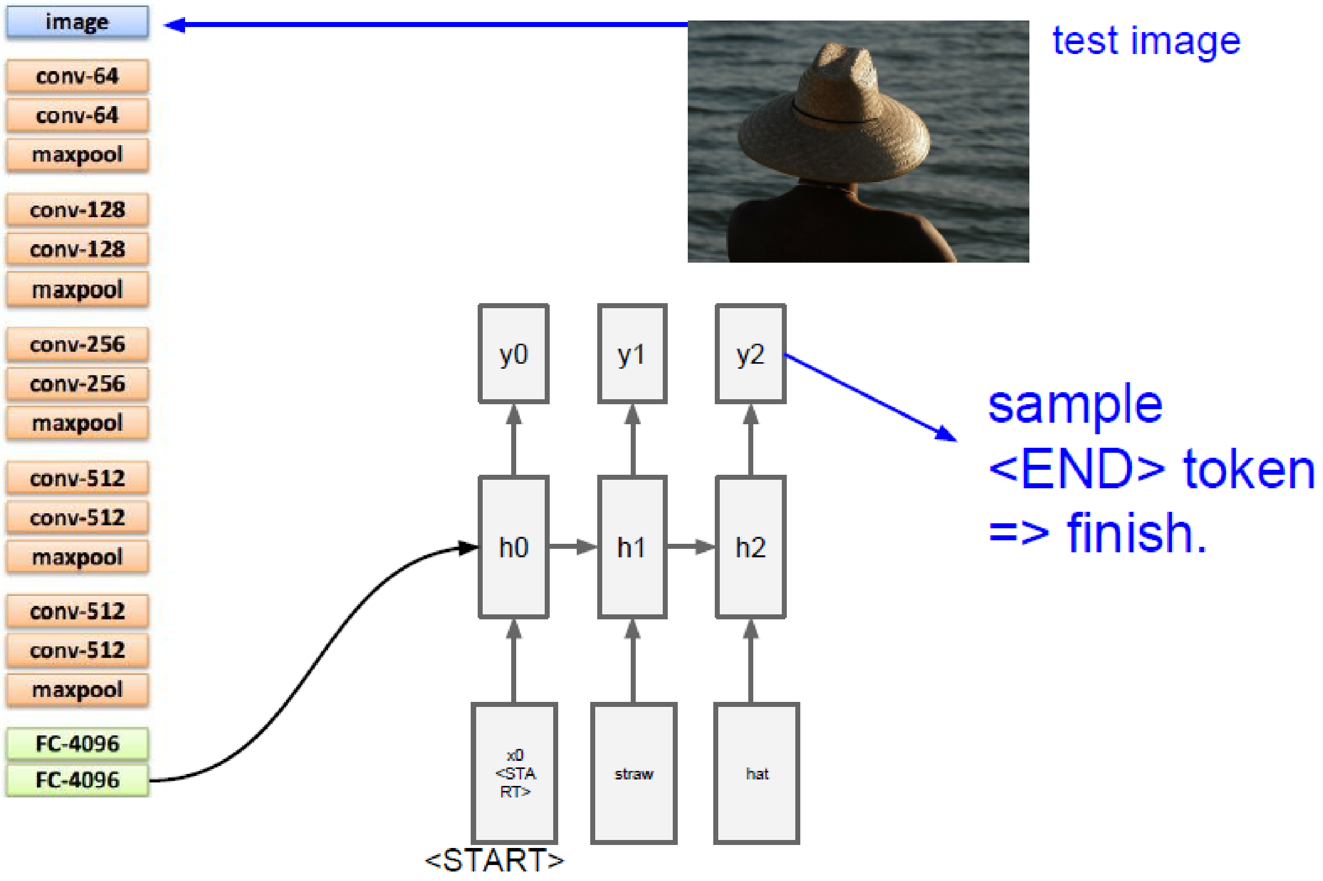
Example 3: Image captioning (cont.)



Example 3: Image captioning (cont.)



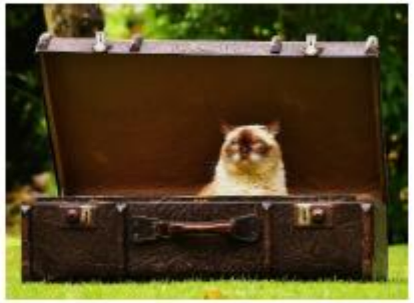
Example 3: Image captioning (cont.)



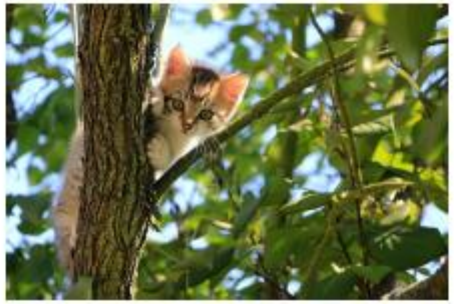
test image

sample <END> token => finish.

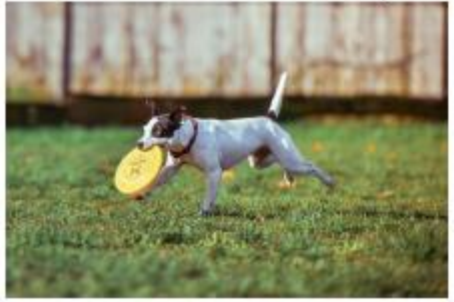
Example 3: More results



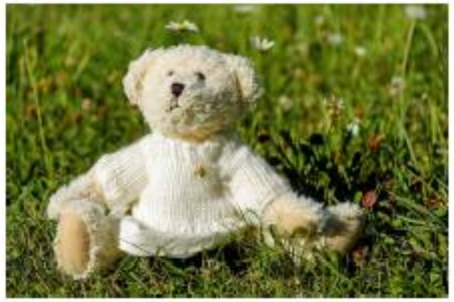
A cat sitting on a suitcase on the floor



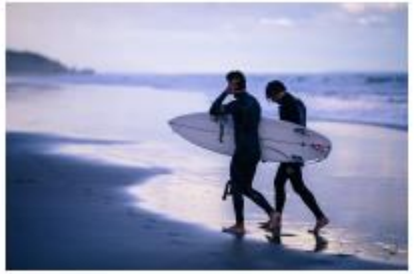
A cat is sitting on a tree branch



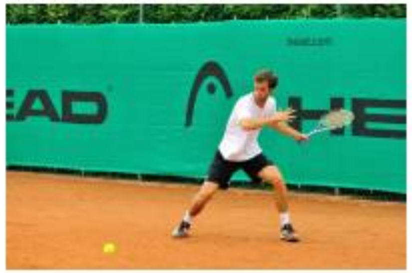
A dog is running in the grass with a frisbee



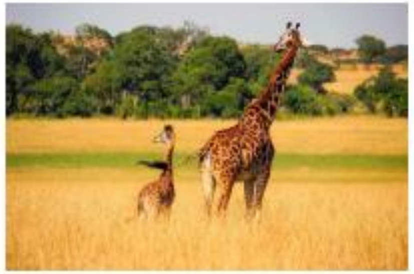
A white teddy bear sitting in the grass



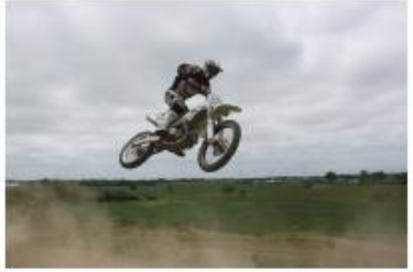
Two people walking on the beach with surfboards



A tennis player in action on the court

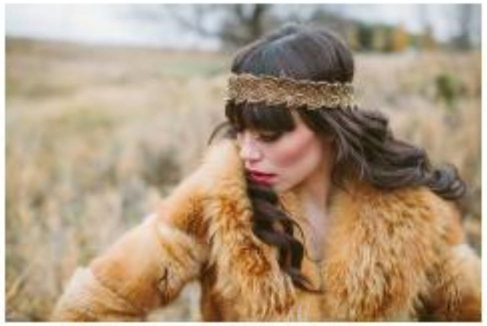


Two giraffes standing in a grassy field

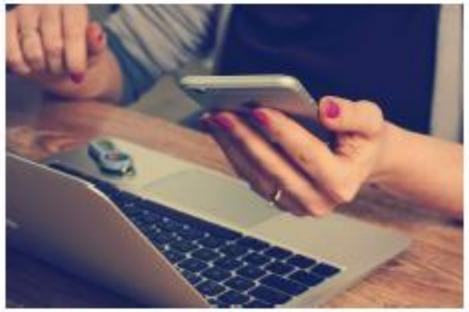


A man riding a dirt bike on a dirt track

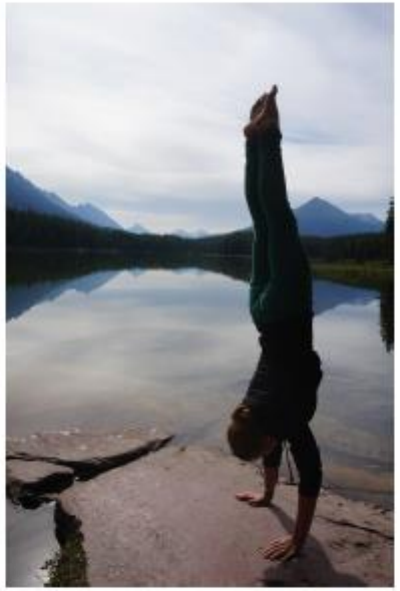
Example 3: Failure cases



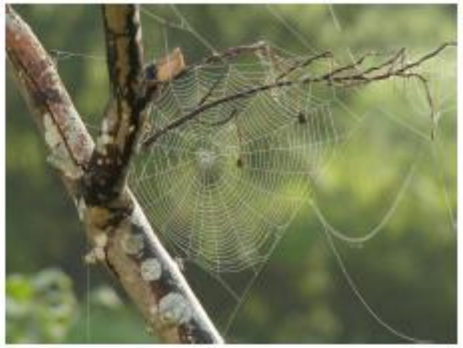
A woman is holding a cat in her hand



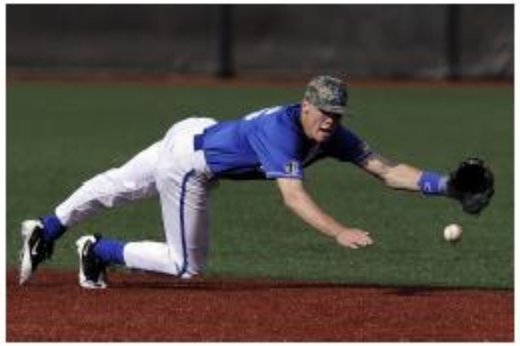
A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard



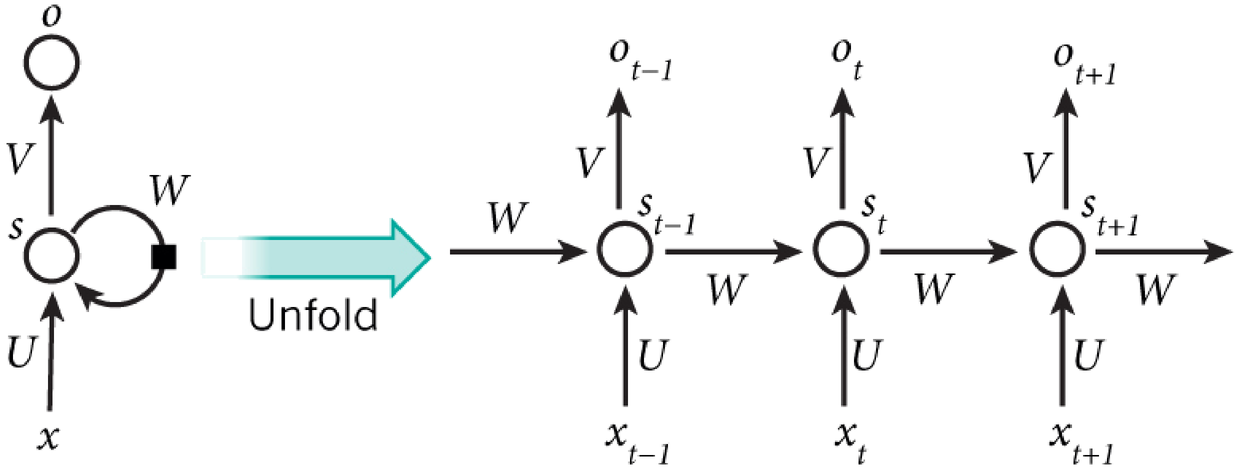
A bird is perched on a tree branch



A man in a baseball uniform throwing a ball

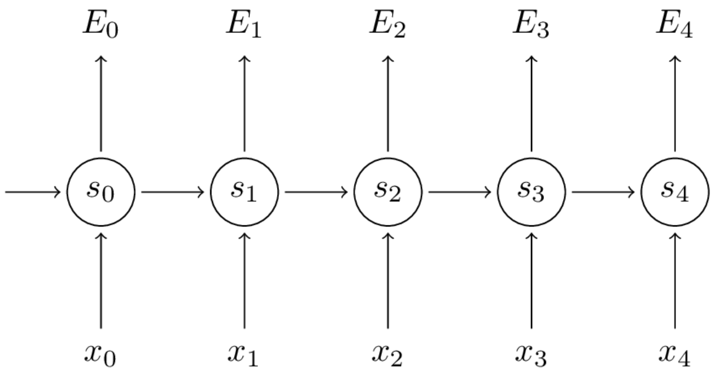
Training

Training



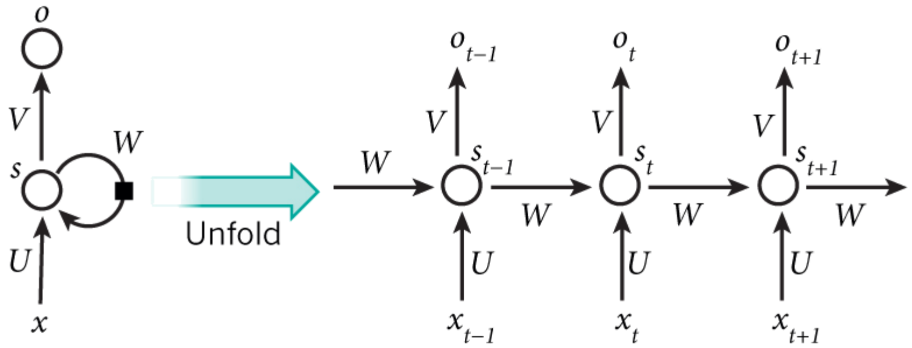
- Still use backpropagation
- But now weights are shared by all time steps
- Backpropagation Through Time (BPTT)
  - E.g., in order to calculate the gradient at $t = 4$, we would need to backpropagate 3 steps and sum up the gradients
  - $s_t = \tanh(Ux_t + Ws_{t-1})$
    $\hat{y}_t = \mathrm{softmax}(Vs_t)$
  - Loss, by cross entropy
    - y_t is the correct word at time t
    - $\hat{y}_t$ is prediction

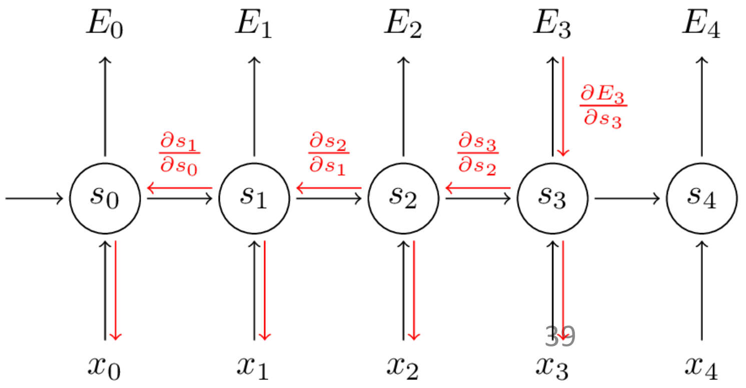
$$\begin{aligned} E_t(y_t, \hat{y}_t) &= -y_t \log \hat{y}_t \\ E(y, \hat{y}) &= \sum_t E_t(y_t, \hat{y}_t) \\ &= -\sum_t y_t \log \hat{y}_t \end{aligned}$$



Backpropagation Through Time (BPTT)

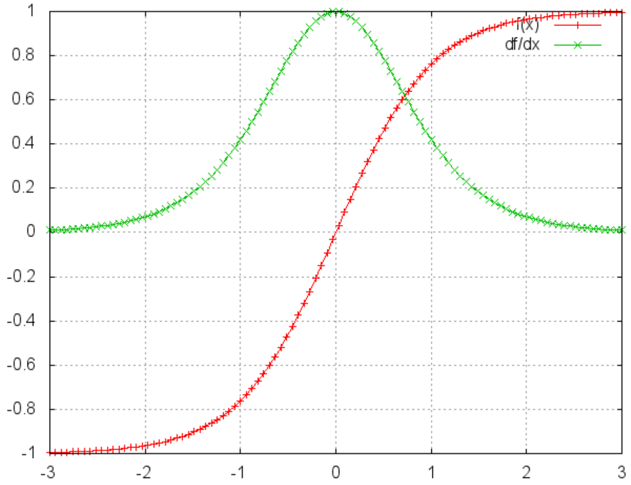
- Recall that our goal is to calculate the gradients of the error with respect to parameters U, V, W
- Also the gradients sum over time $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$
- E.g. $\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\frac{\partial s_3}{\partial W} + \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W} + \cdots \right)$
  - $s_3 = \tanh(Ux_t + Ws_2)$
- So to get gradients of time $t = 3$, we need to backpropagate gradients to $t = 0$





E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

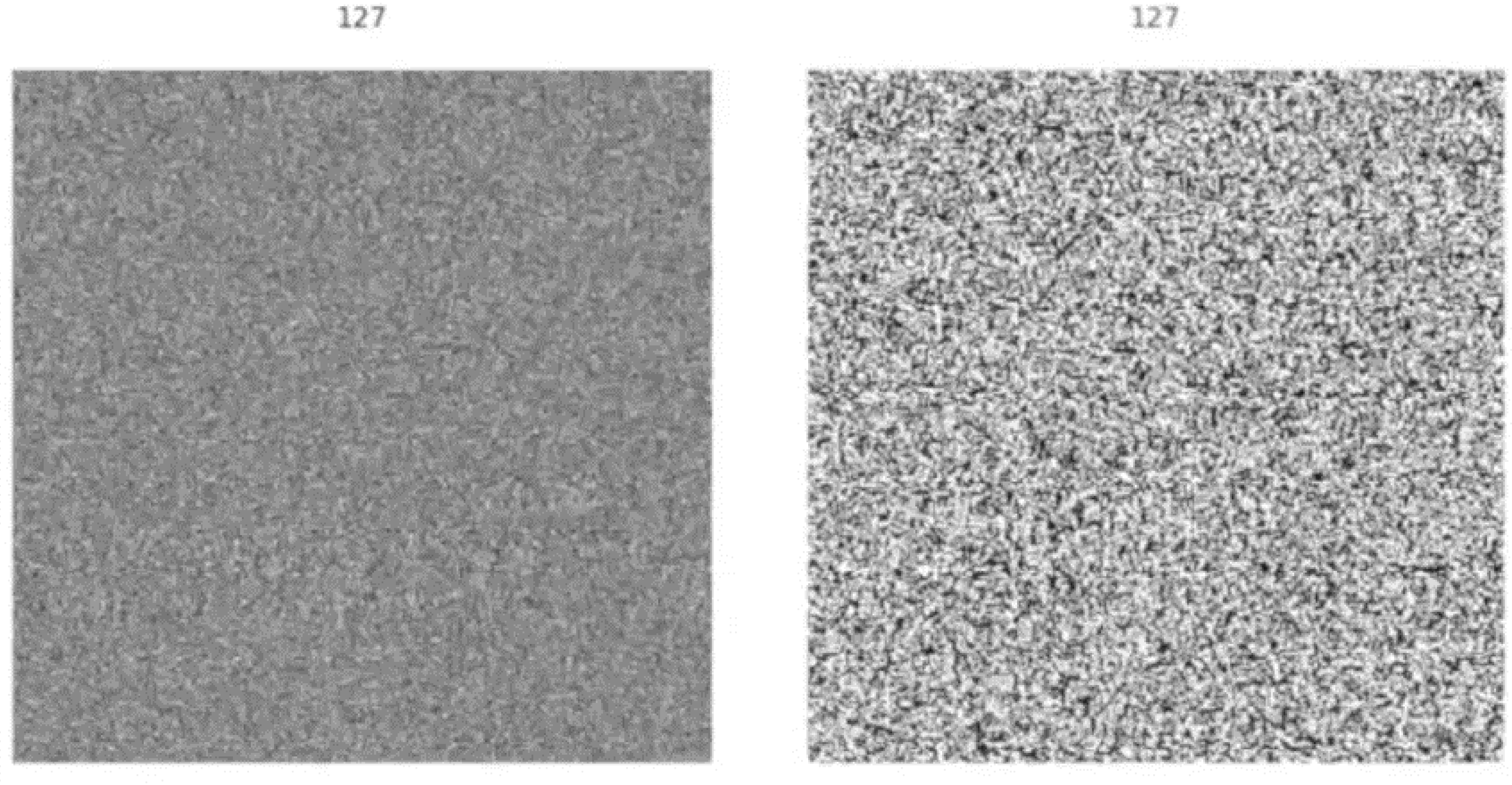
Summary of BPTT



- It is just backpropagation on the unrolled RNN by summing over all gradients for W and U at each time step
- The vanishing gradient problem
  - $\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\frac{\partial s_3}{\partial W} + \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W} + \cdots \right)$

    $\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$
  - This Jacobian matrix (derivative w.r.t. a vector) has 2-norm less than 1
    - As an intuition, the derivative of tanh is bounded by 1
  - Thus, with small values in multiple matrix multiplications, the gradient values are shrinking exponentially fast, eventually vanishing completely after a few time steps
  - Gradient contributions from “far away” steps become zero
  - You end up not learning long-range dependencies
  - Not exclusive to RNNs. It’s just that RNNs tend to be very deep (as sentence length)

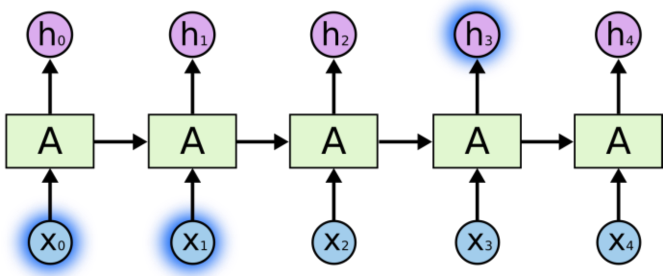
Demo of RNN vs LSTM: Vanishing gradients

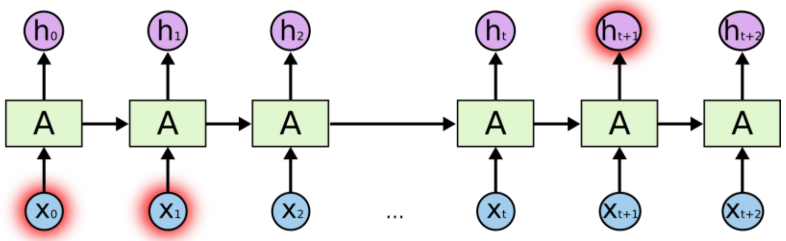


- RNN vs LSTM gradients on the input weight matrix https://imgur.com/gallery/vaNahKE
- Error is generated at 128th step and propagated back. No error from other steps
- Initial weights sampled from Normal Distribution in (-0.1, 0.1)

The problem of long-term dependencies

- One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame
- RNN usually works well for recent information
  - E.g. The clouds are in the *sky*.
- But might work worse for further back context
  - E.g. I grew up in France… I speak fluent *French*.

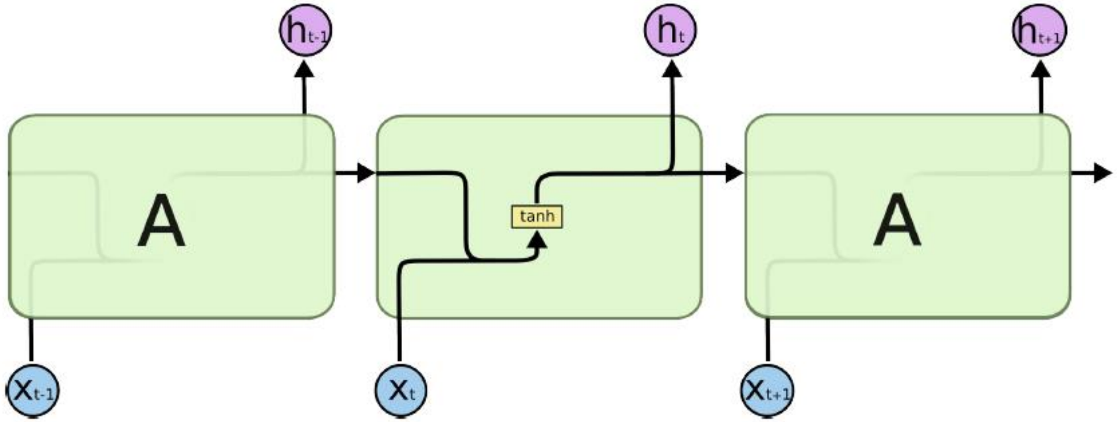




Long Short-Term Memory

Long short-term memory (LSTM)

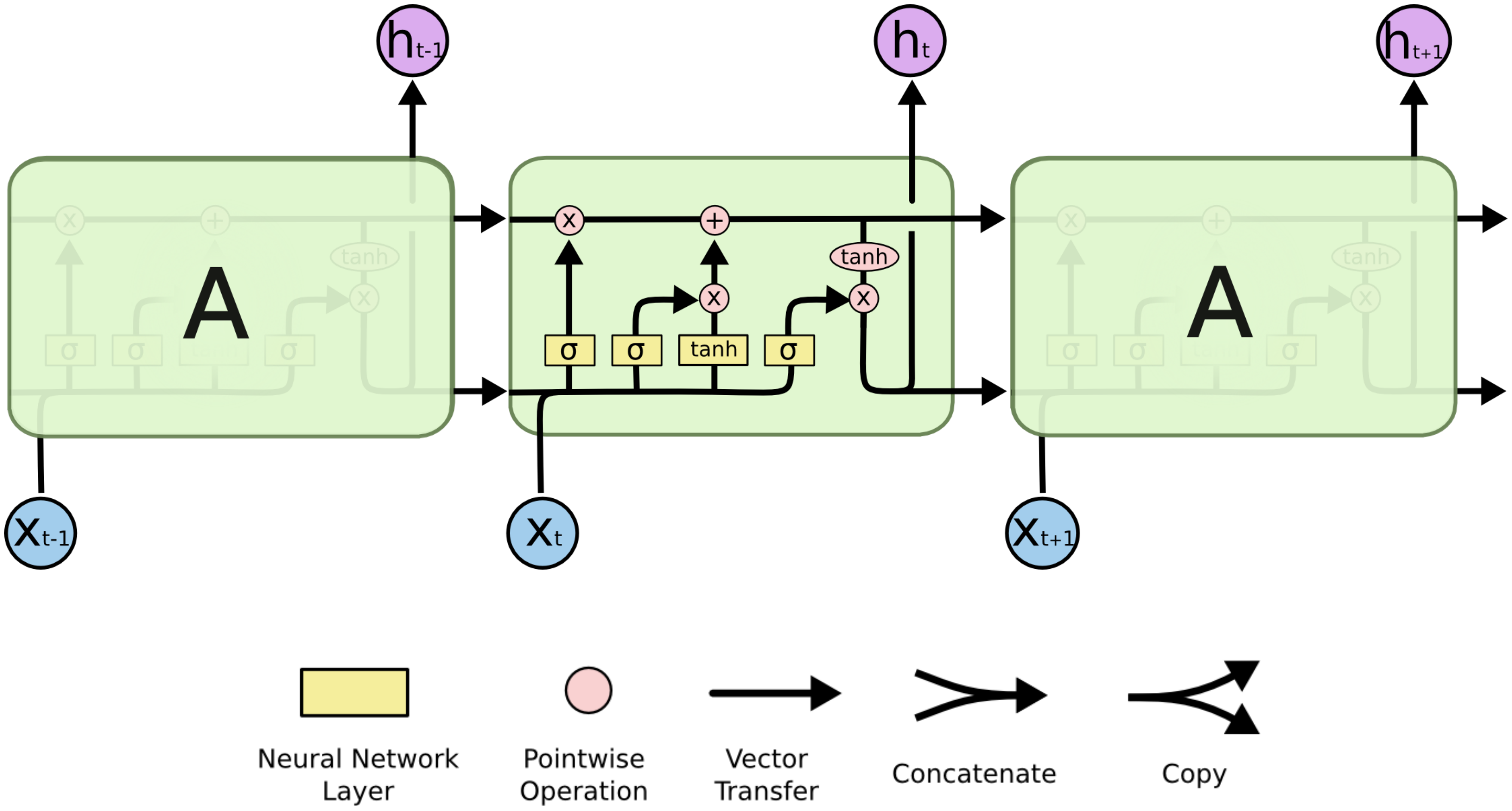
- Hochreiter & Schmidhuber [1997]
- A variant of RNN, capable of learning long-term dependencies
- Can remember information for long periods of time



Remember the architecture of RNN cells, we will make many changes on it

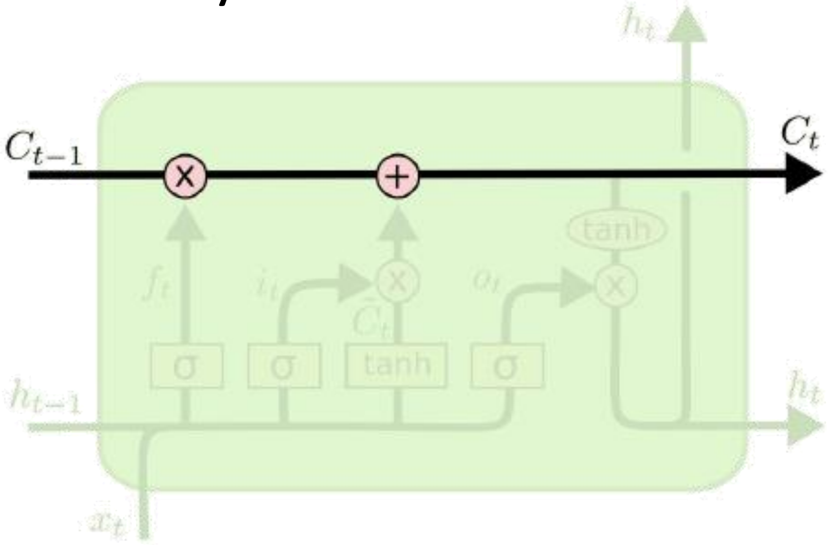
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM at a glance



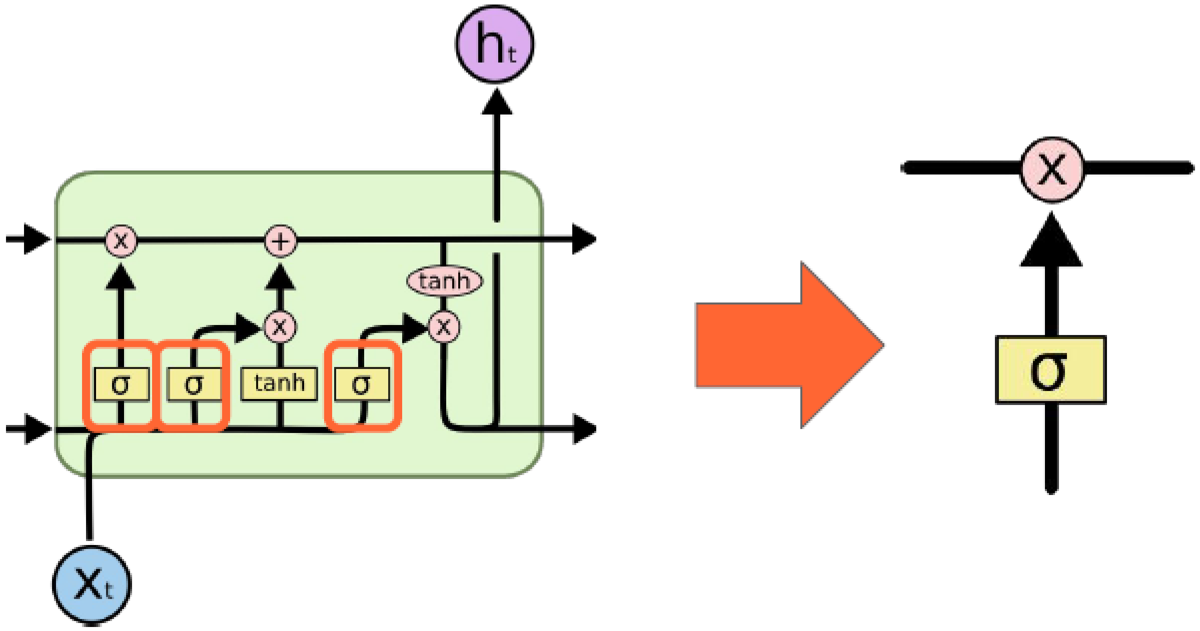
LSTM

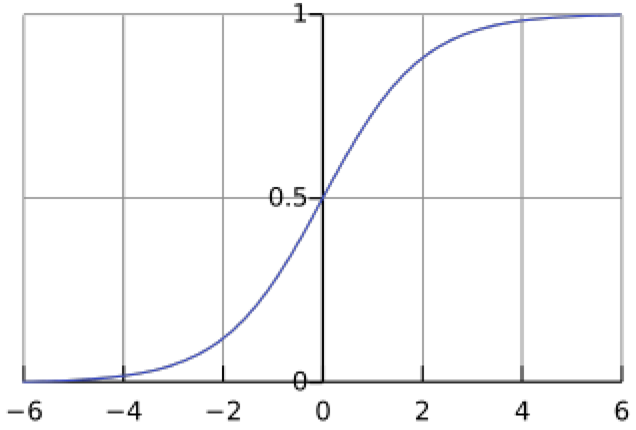
- C_t is the cell state
  - Horizontal line running through the top of the diagram
  - Like a conveyer
  - Run straight down the entire chain, with only some minor linear interactions
  - LSTM uses gates to carefully remove or add information to the cell state



LSTM (cont.)

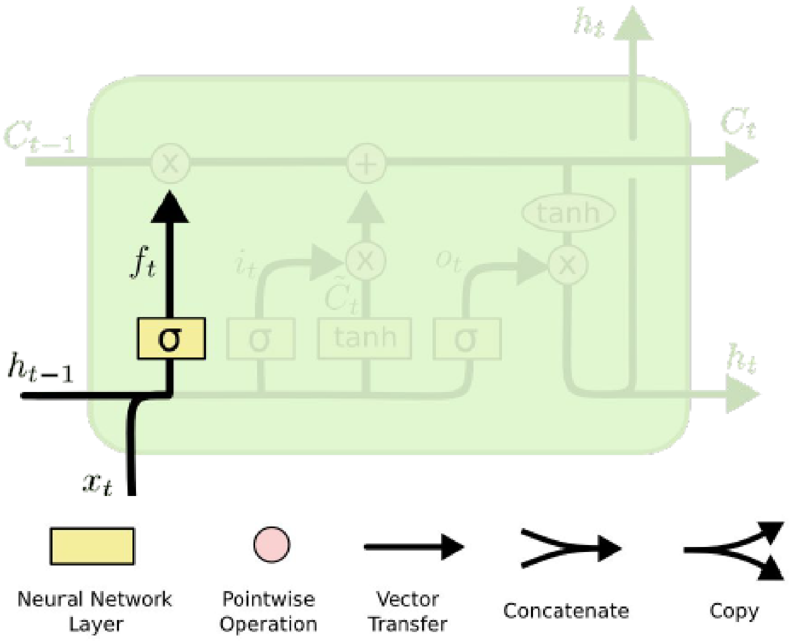
- Three gates are governed by *sigmoid* units
- Describing how much of each component should be let through
- “let nothing through” (zero value)/“let everything through” (one value)





LSTM (cont.)

- The first gate is forget gate layer
  - Decide what information we’re going to throw away from the cell state
  - Output a number in $(0,1)$ for each number in the cell state
  - 1/0: completely keep/remove this

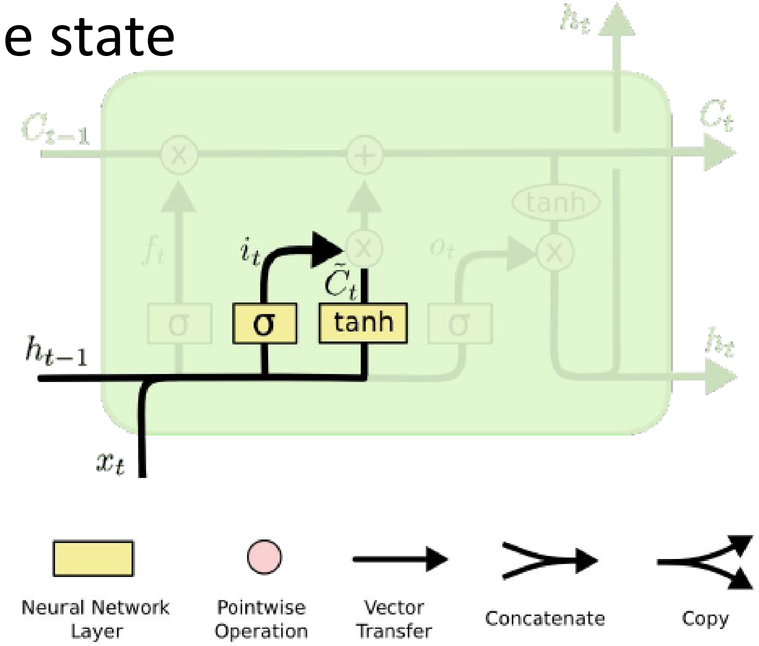


Forget Gate:

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

LSTM (cont.)

- Next is to decide what new information we’re going to store in the cell state
  - The sigmoid layer is the input gate layer, deciding which values we’ll update
  - The tanh layer creates a vector of new candidate values $\tilde{C}_t$ that should be added to the state



Input Gate Layer

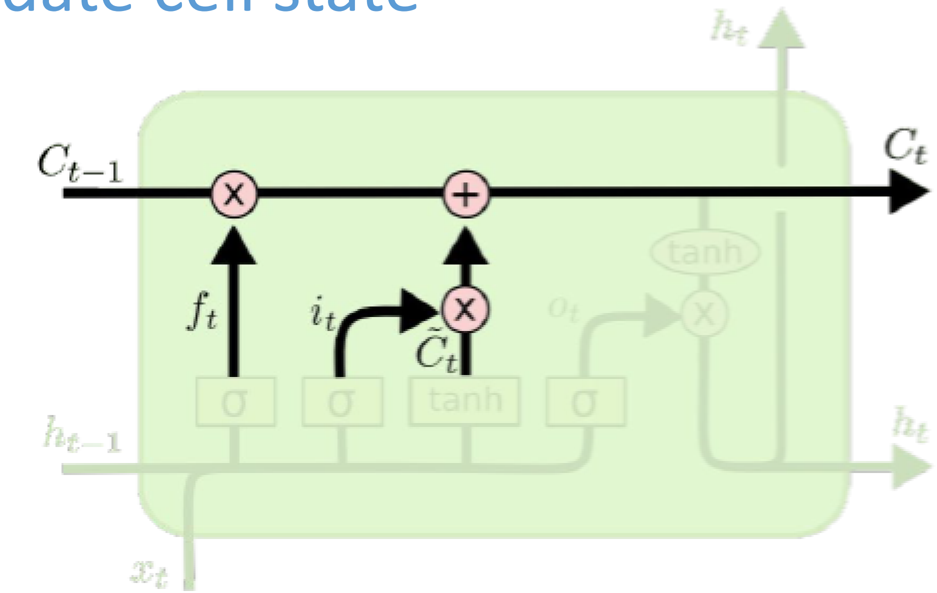
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

New contribution to cell state

$$\tilde{C}_t = \underbrace{\tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)}_{\text{Classic neuron}}$$

LSTM (cont.)

- Multiply the old state by f_t
  - Forgetting the things from previous state
- Add new candidate values
- Update cell state

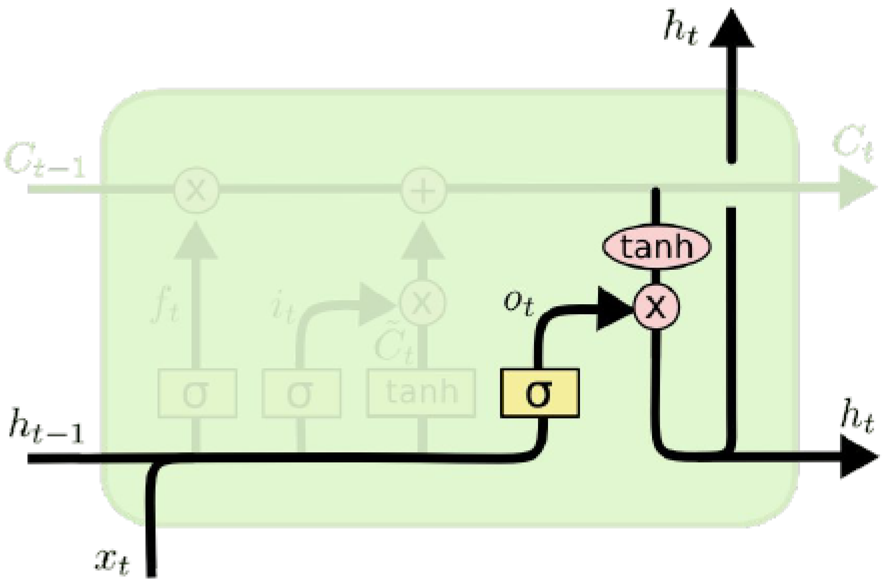


Update Cell State (memory):

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

LSTM (cont.)

- Last decide the output h_t
  - Filtered (tanh) version of the cell state
  - The sigmoid layer decides what parts of the cell state we're going to output h_{t-1}, x_t are like *context*



Output Gate Layer

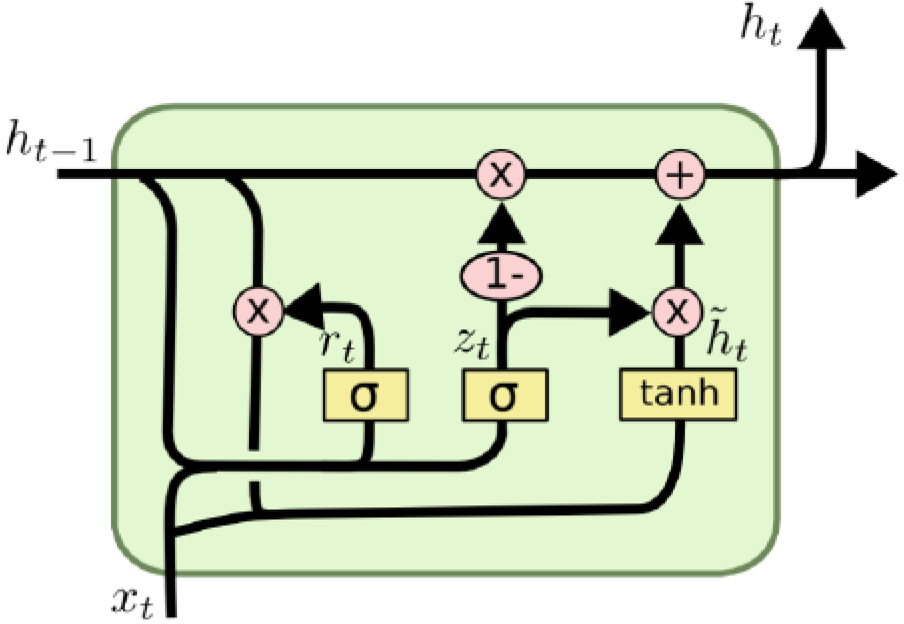
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

Output to next layer

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

Gated recurrent unit (GRU)

- GRU is a variants of LSTM
  - Combines the forget and input gates into a single “update gate”
  - Also merges the cell state and hidden state



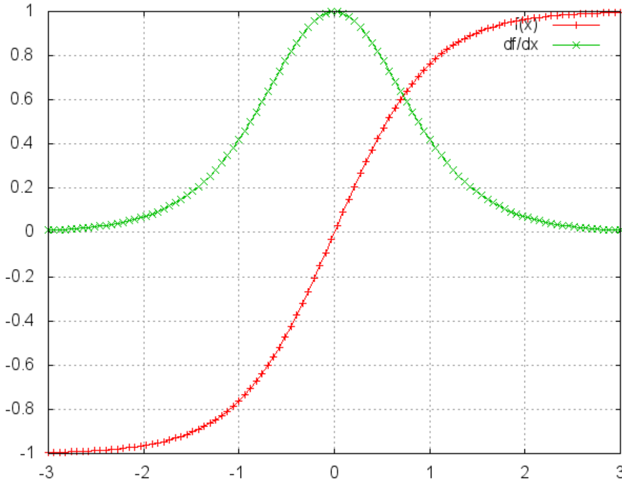
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

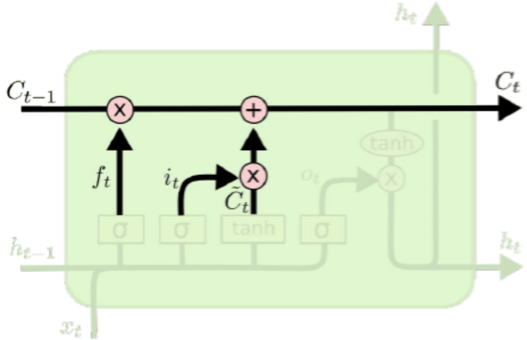
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

How LSTM avoids gradient vanishing

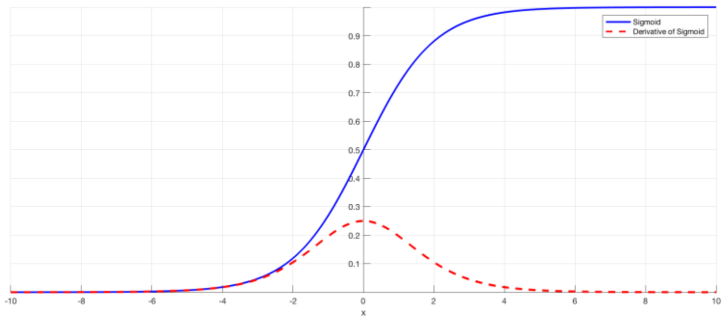


- RNN $\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$
    - $s_j = \tanh(Ux_t + Ws_{j-1})$
    - $\frac{\partial s_j}{\partial s_{j-1}} = \tanh'(\cdot)W$
    - Vanishing since
      $$\prod_{j=k+1}^{k\prime} \frac{\partial s_j}{\partial s_{j-1}} = (\tanh'(\cdot)W\,)^{k'-k}$$
      as $k' - k$ is large
- LSTM
    - $\frac{\partial C_t}{\partial C_{t-1}} = f_t = \sigma(\cdot)$ where the value can be trained around 1 (a large part of σ near 1)
    - Also the sum operation help ease the problem
    - Will also vanish, but much slower than RNN



Update Cell State (memory):

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



LSTM hyperparameter tuning

- Watch out for *overfitting*
- Regularization helps: regularization methods include L1, L2, and dropout among others.
- The larger the network, the more powerful, but also easier to overfit. Don’t want to try to learn a million parameters from 10,000 examples
  - parameters > examples = trouble
- More data is usually better
- Train over multiple epochs (complete passes through the dataset)
- The learning rate is the single most important hyper parameter