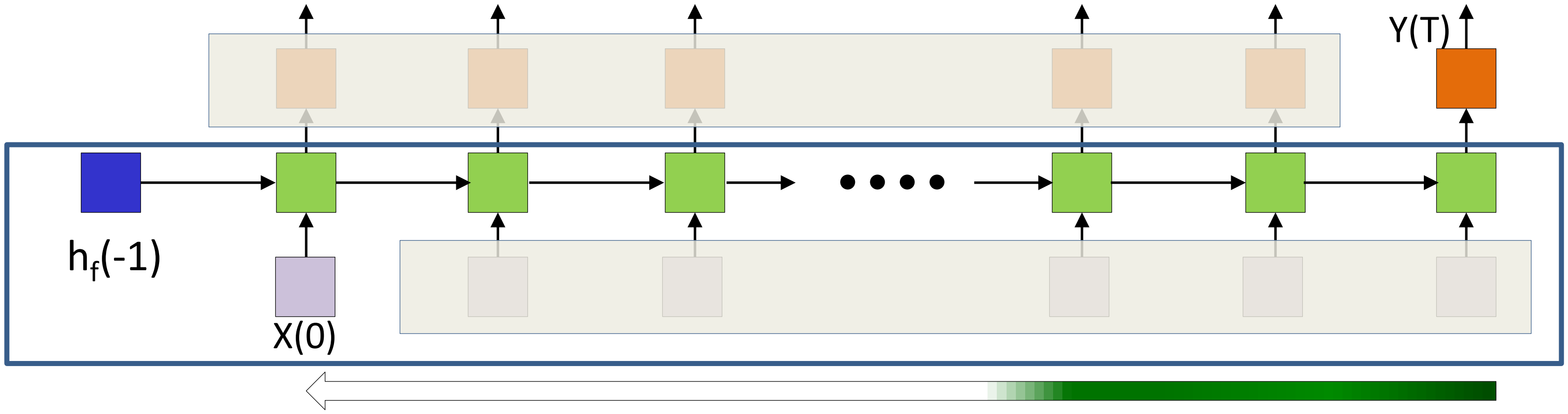
RNN Recitation

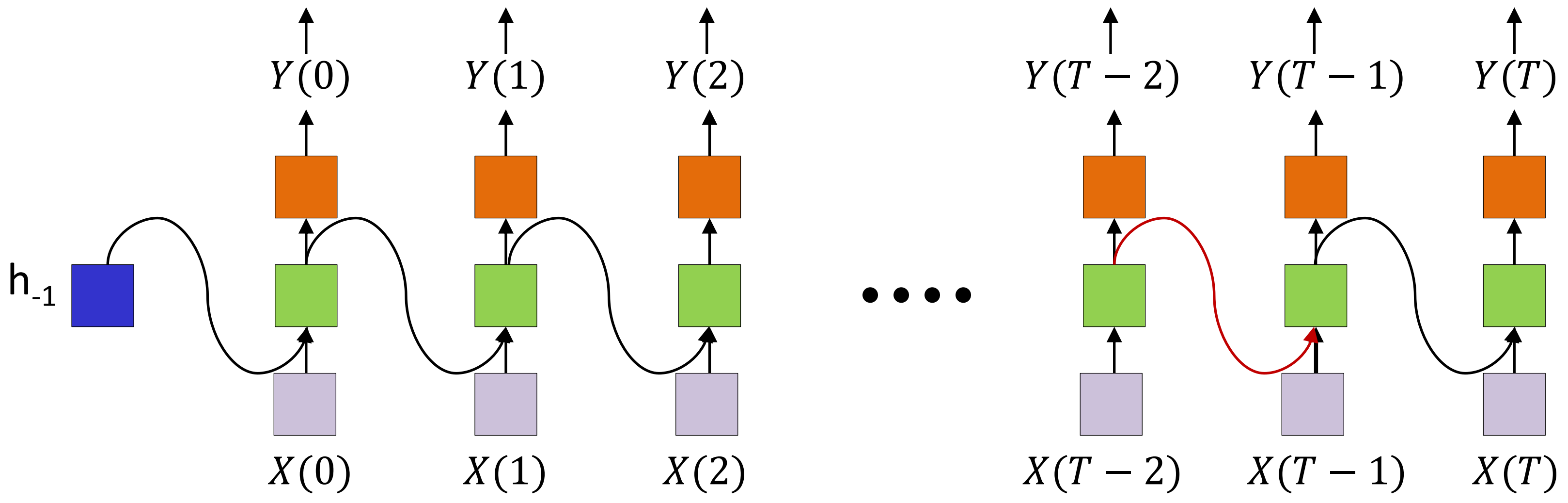
10/27/17

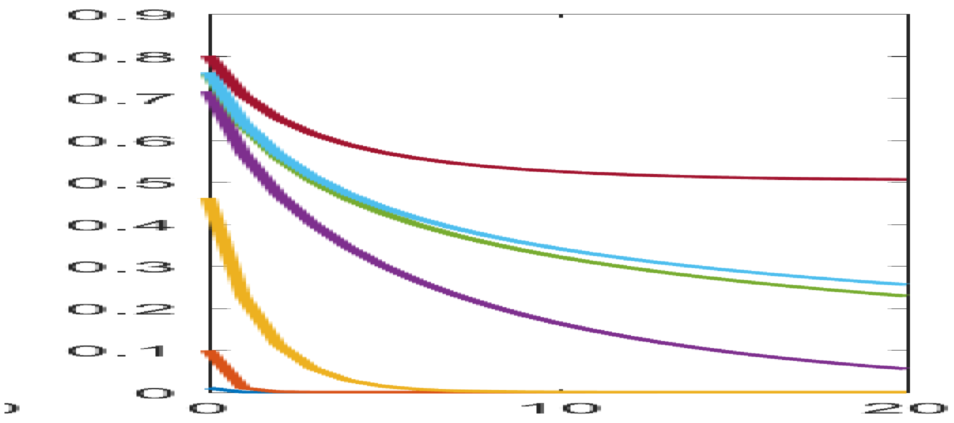
Recurrent nets are very deep nets



- The relation between and is one of a very deep network
  - Gradients from errors at will vanish by the time they're propagated to

Recall: Vanishing stuff..





- Stuff gets forgotten in the forward pass too

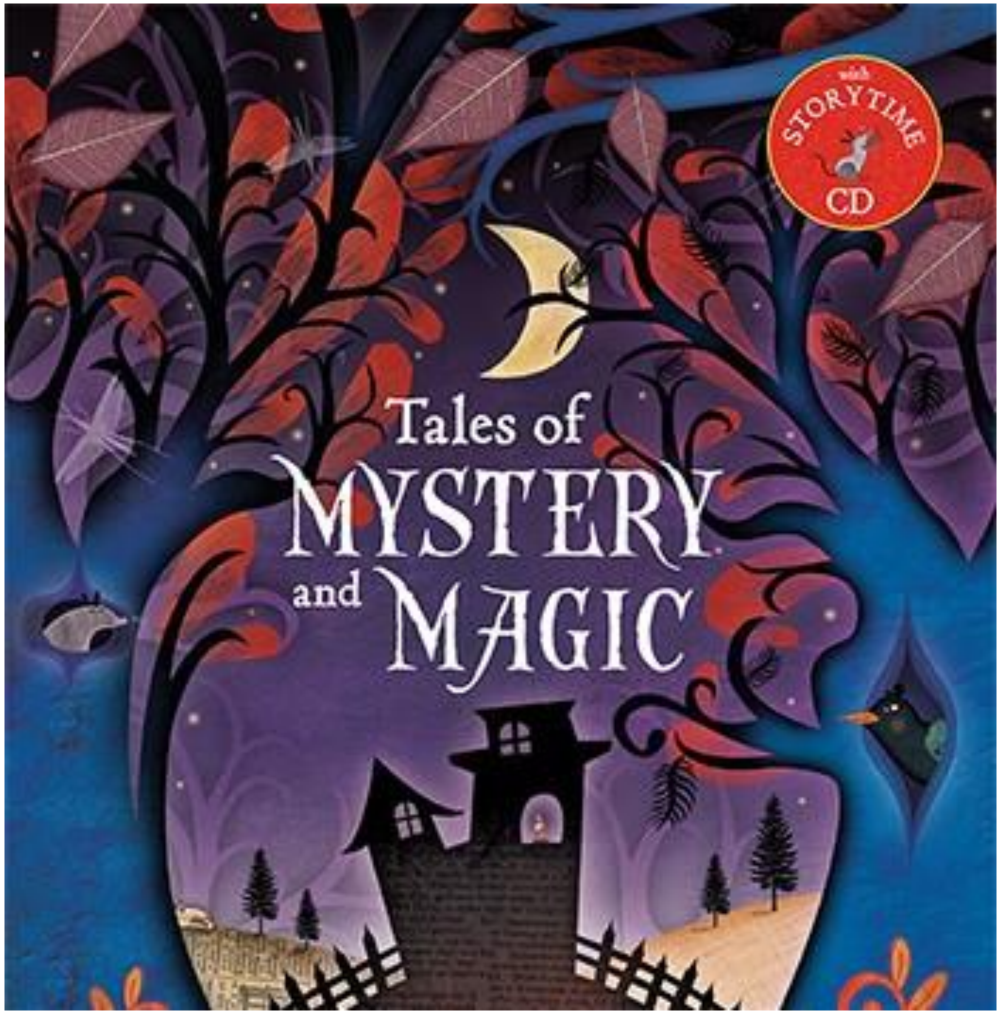
The long-term dependency problem

PATTERN1 [..............................] PATTERN 2

Jane had a quick lunch in the bistro. Then *she*..

- Any other pattern of any length can happen between pattern 1 and pattern 2
  - RNN will "forget" pattern 1 if intermediate stuff is too long
  - "Jane" → the next pronoun referring to her will be "she"
- Must know to "remember" for extended periods of time and "recall" when necessary
  - Can be performed with a multi-tap recursion, but how many taps?
  - Need an alternate way to "remember" stuff

And now we enter the domain of..

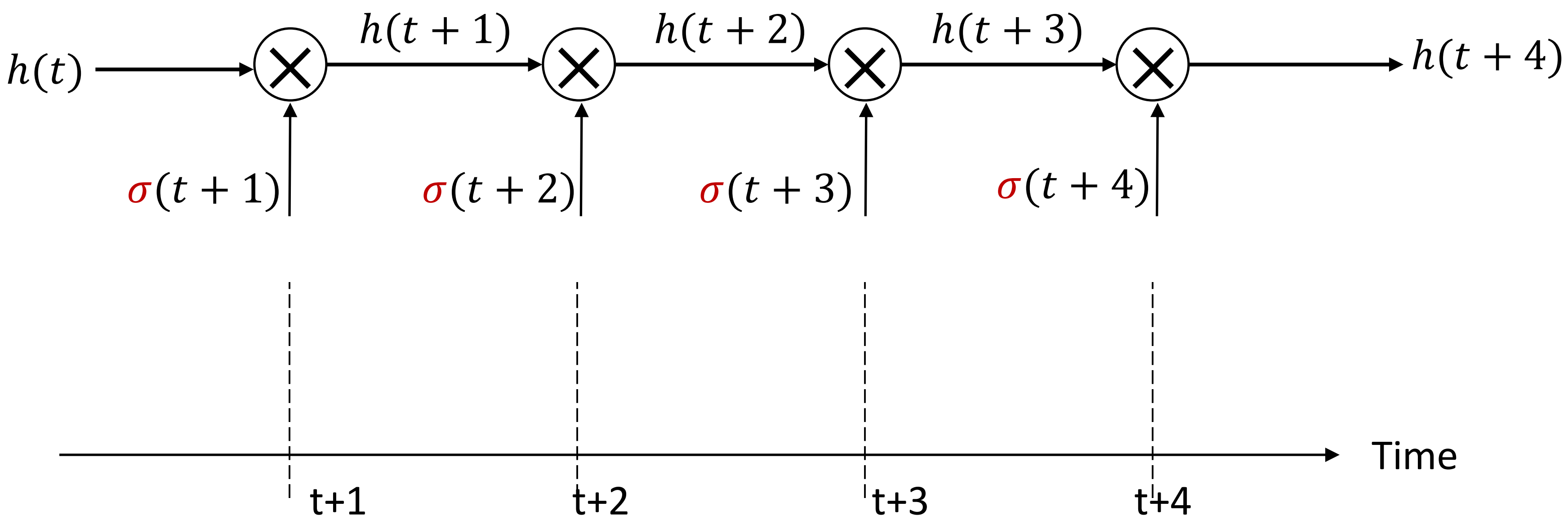


Exploding/Vanishing gradients

- Can we replace this with something that doesn’t fade or blow up?

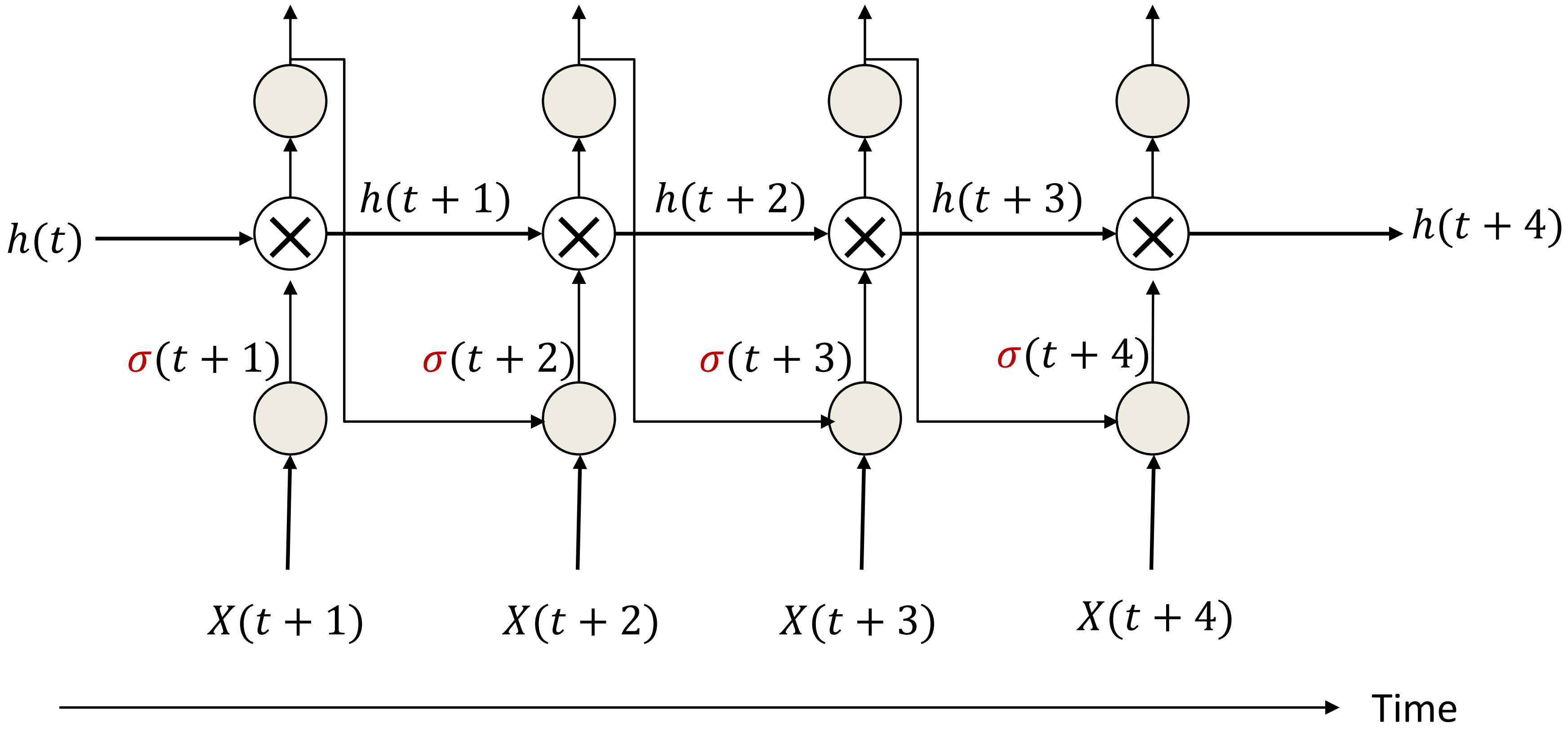
- Can we have a network that just “remembers” arbitrarily long, to be recalled on demand?

Enter – the constant error carousel



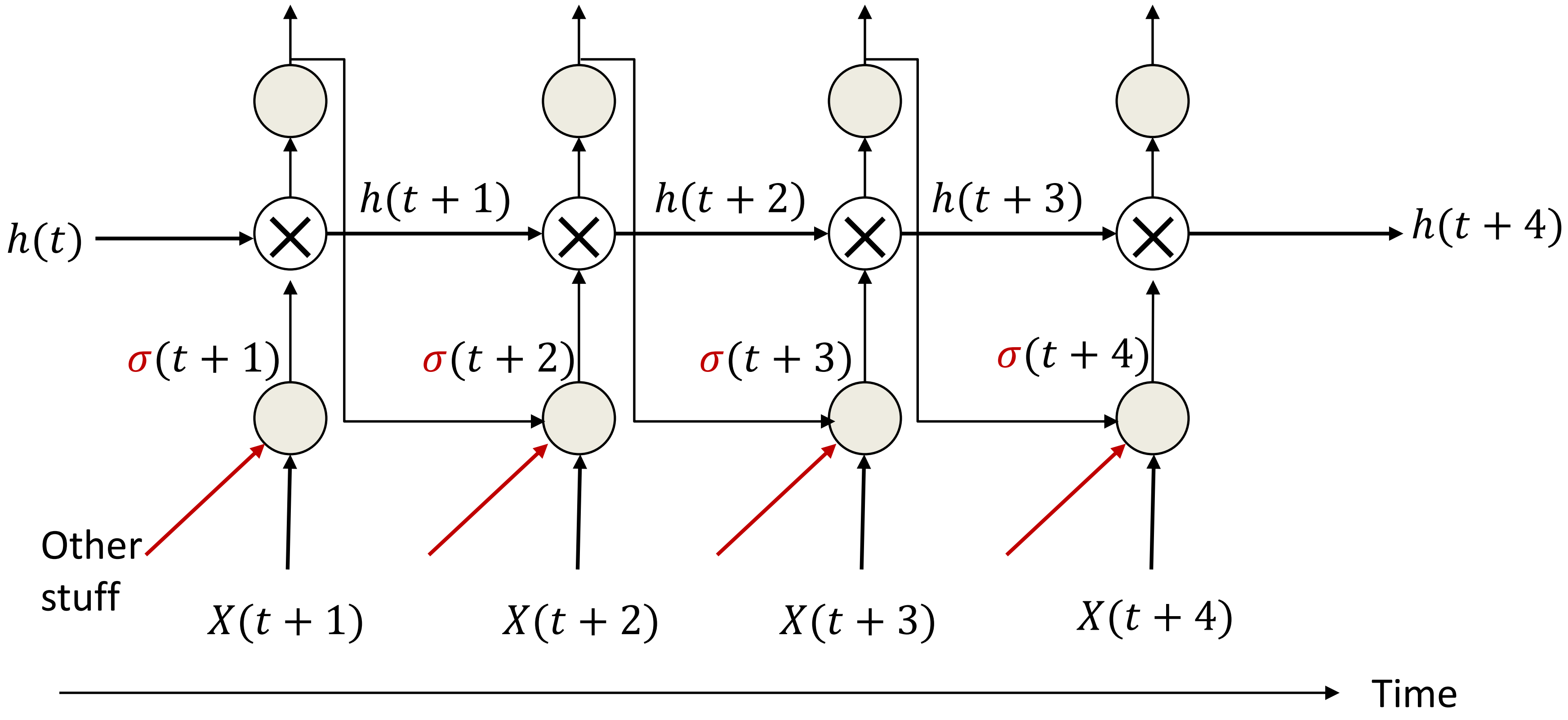
- History is carried through uncompressed
  - No weights, no nonlinearities
  - Only scaling is through the σ “gating” term that captures other triggers
  - E.g. “Have I seen Pattern2”?

Enter – the constant error carousel



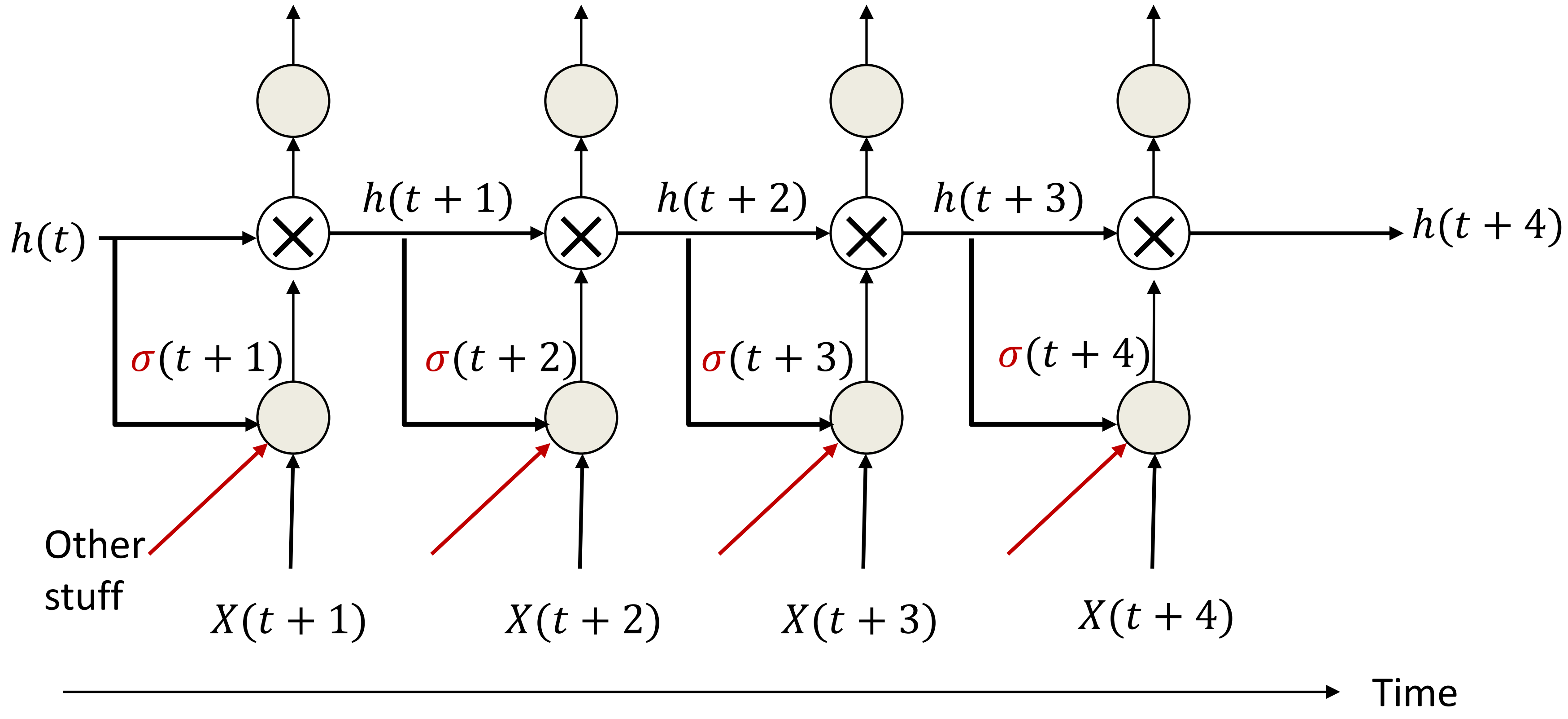
- Actual non-linear work is done by other portions of the network

Enter – the constant error carousel



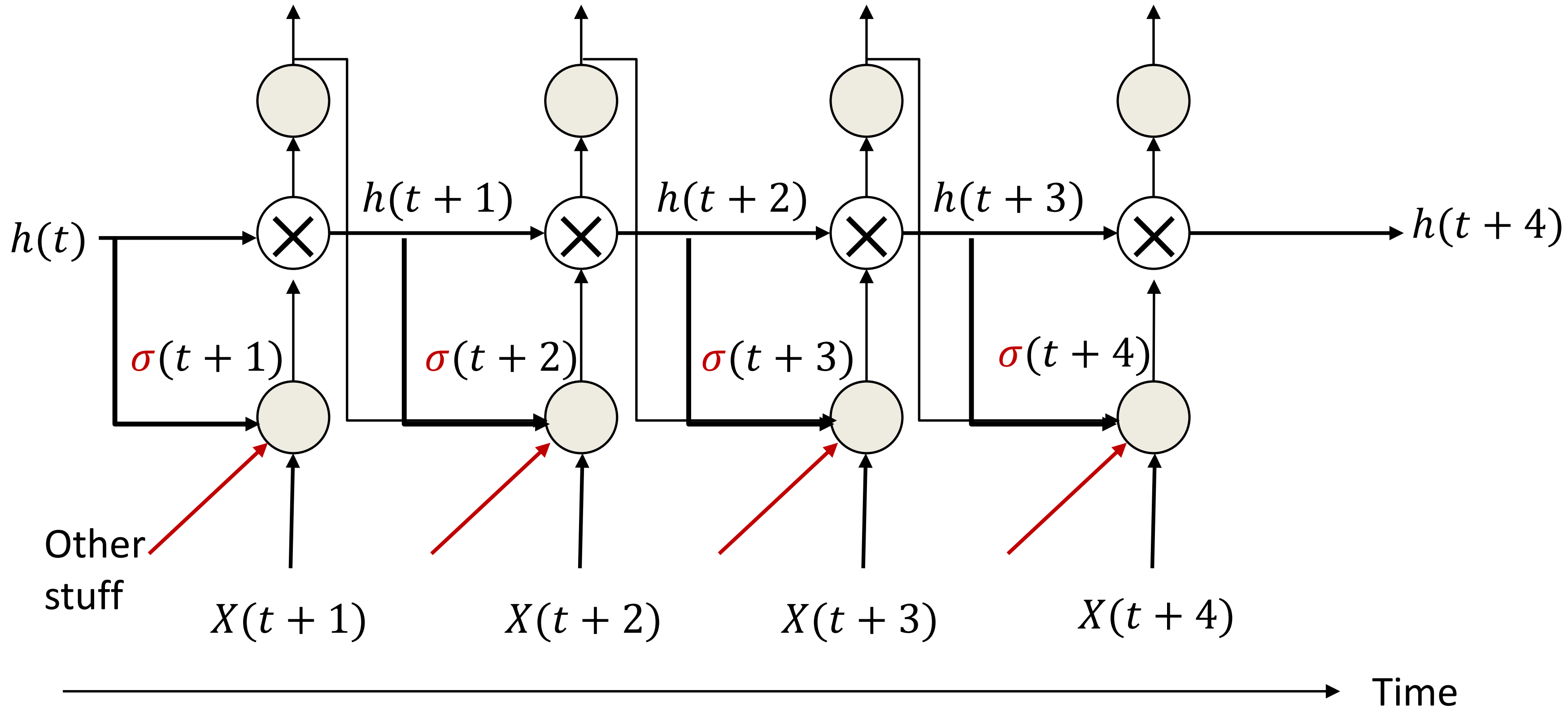
- Actual non-linear work is done by other portions of the network

Enter – the constant error carousel



- Actual non-linear work is done by other portions of the network

Enter – the constant error carousel



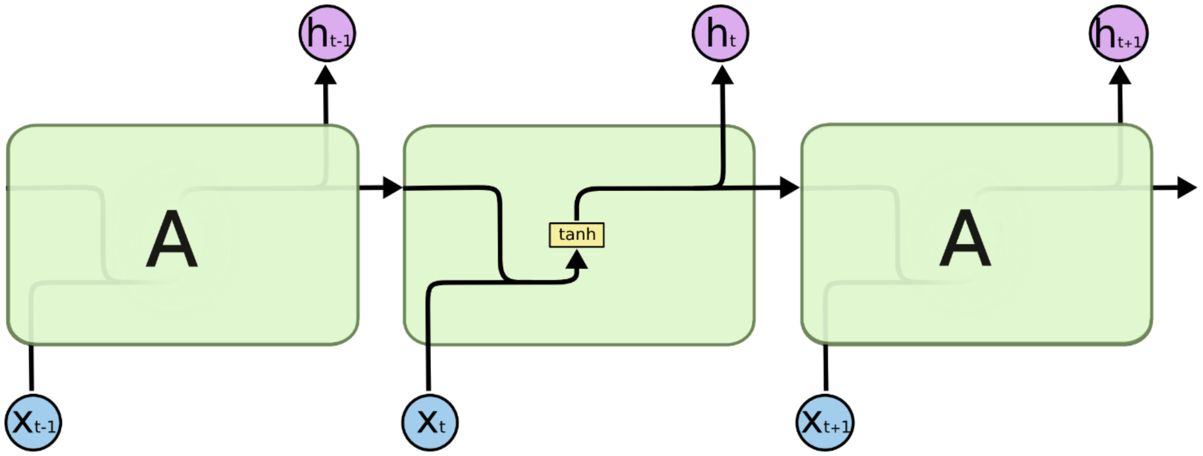
- Actual non-linear work is done by other portions of the network

Enter the *LSTM*

- *Long Short-Term Memory*
- Explicitly latch information to prevent decay / blowup

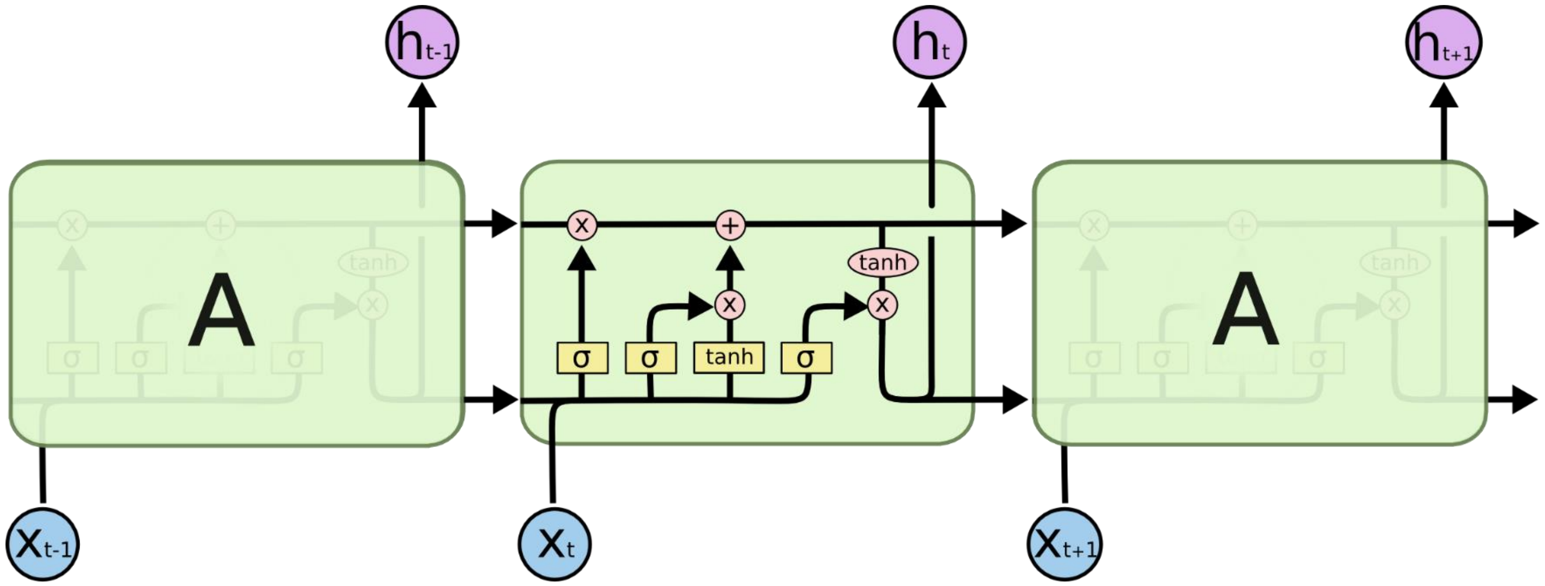
- Following notes borrow liberally from
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Standard RNN



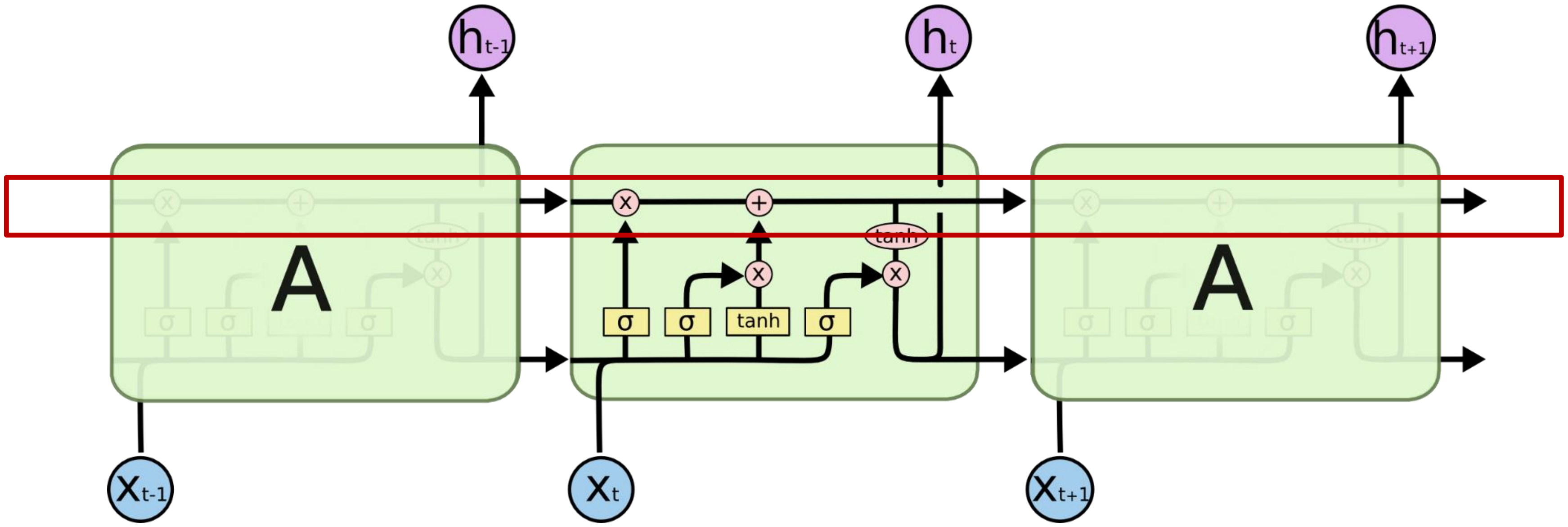
- Recurrent neurons receive past recurrent outputs and current input as inputs
- Processed through a tanh() activation function
  - As mentioned earlier, tanh() is the generally used activation for the hidden layer
- Current recurrent output passed to next higher layer and next time instant

Long Short-Term Memory



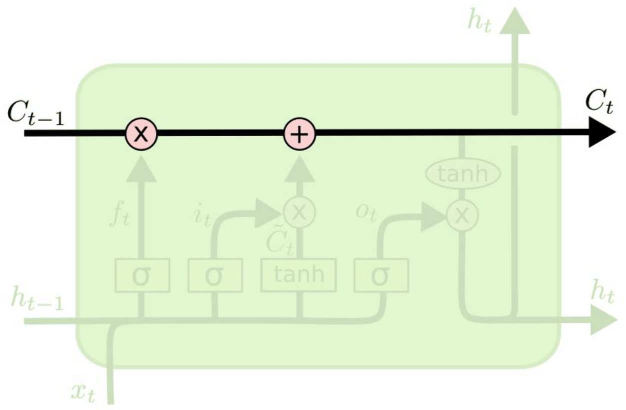
- The $\sigma()$ are *multiplicative gates* that decide if something is important or not
- Remember, every line actually represents a *vector*

LSTM: Constant Error Carousel



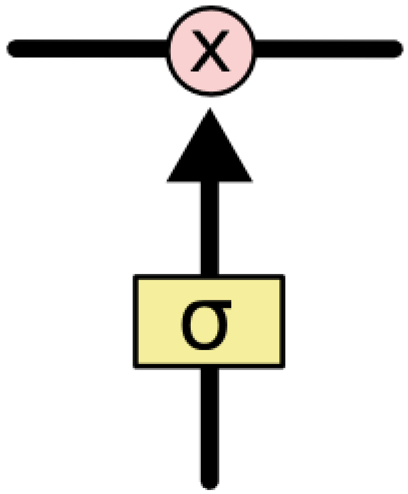
- Key component: a *remembered cell state*

LSTM: CEC



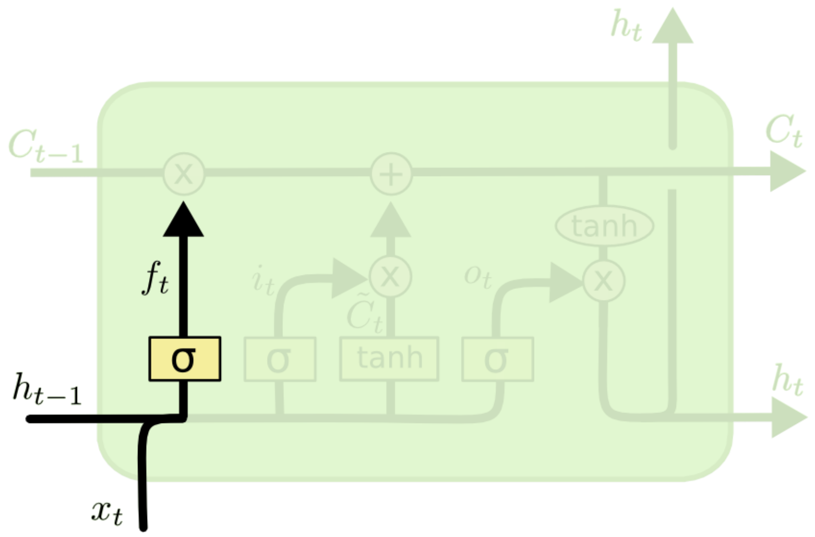
- C_t is the linear history carried by the *constant-error carousel*
- Carries information through, only affected by a gate
  – And *addition of history,* which too is gated..

LSTM: Gates



- Gates are simple sigmoidal units with outputs in the range (0,1)
- Controls how much of the information is to be let through

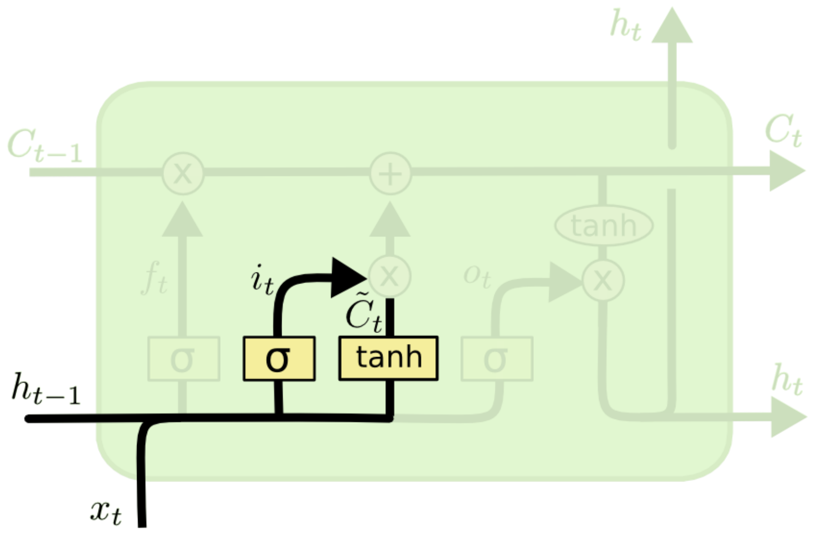
LSTM: Forget gate



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

- The first gate determines whether to carry over the history or to forget it
  - More precisely, how much of the history to carry over
  - Also called the "forget" gate
  - Note, we're actually distinguishing between the cell memory C and the state h that is coming over time! They're related though

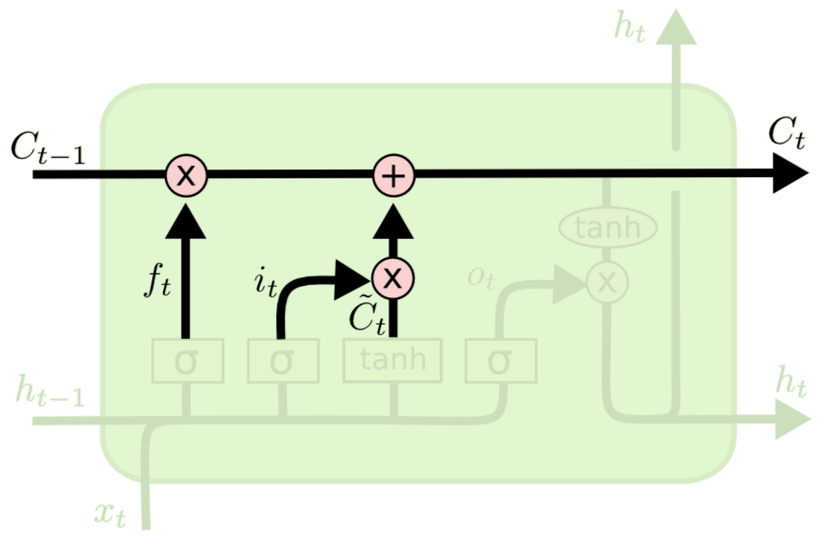
LSTM: Input gate



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

- The second gate has two parts
  - A perceptron layer that determines if there's something interesting in the input
  - A gate that decides if its worth remembering

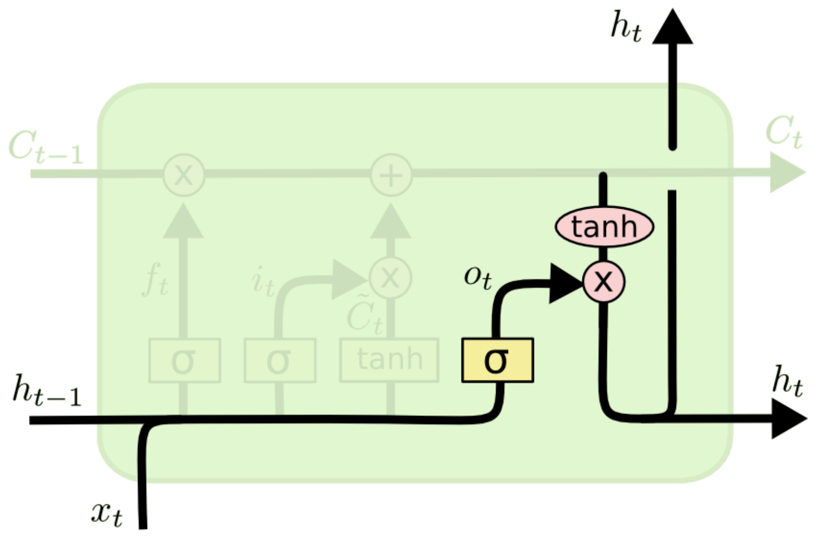
LSTM: Memory cell update



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

- The second gate has two parts
  - A perceptron layer that determines if there's something interesting in the input
  - A gate that decides if its worth remembering
  - **If so its added to the current memory cell**

LSTM: Output and Output gate

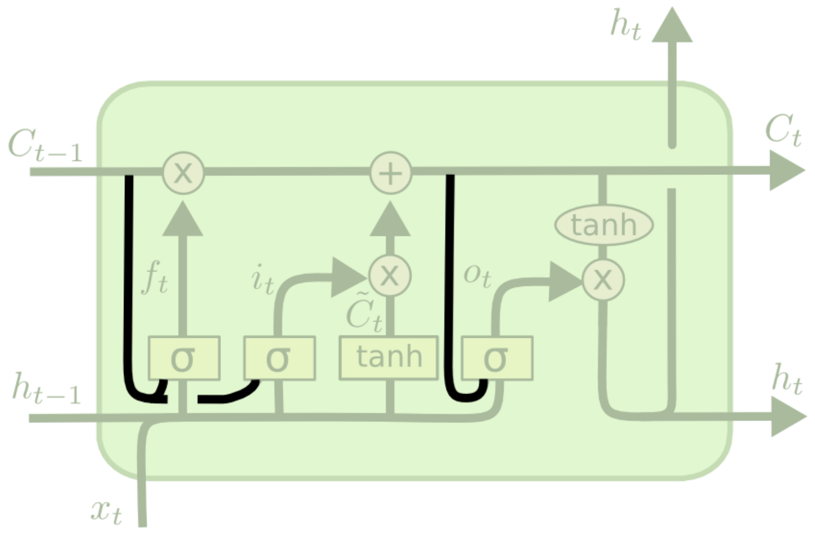


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

- The *output* of the cell
  - Simply compress it with tanh to make it lie between 1 and -1
    - Note that this compression no longer affects our ability to *carry* memory forward
  - While we're at it, lets toss in an output gate
    - To decide if the memory contents are worth reporting at *this* time

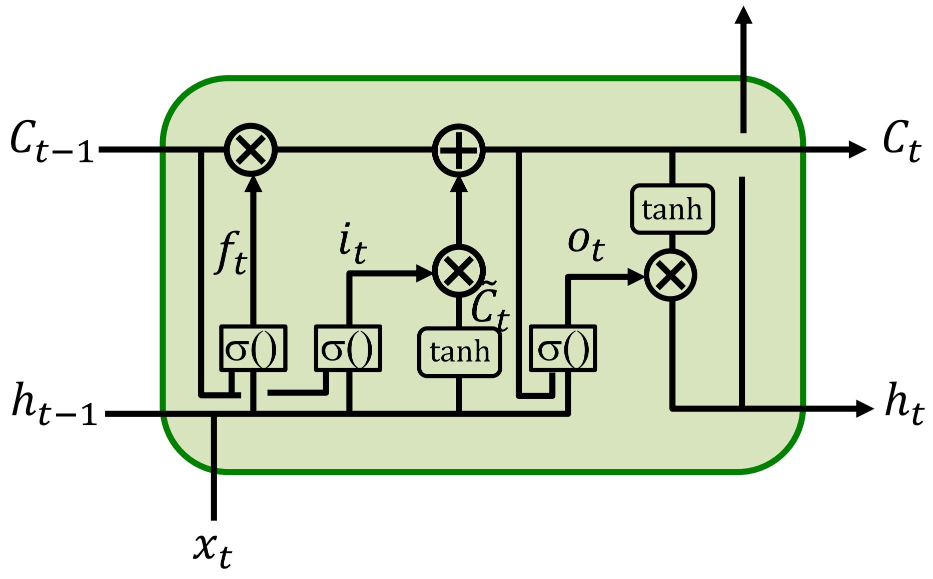
LSTM: The “Peephole” Connection



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

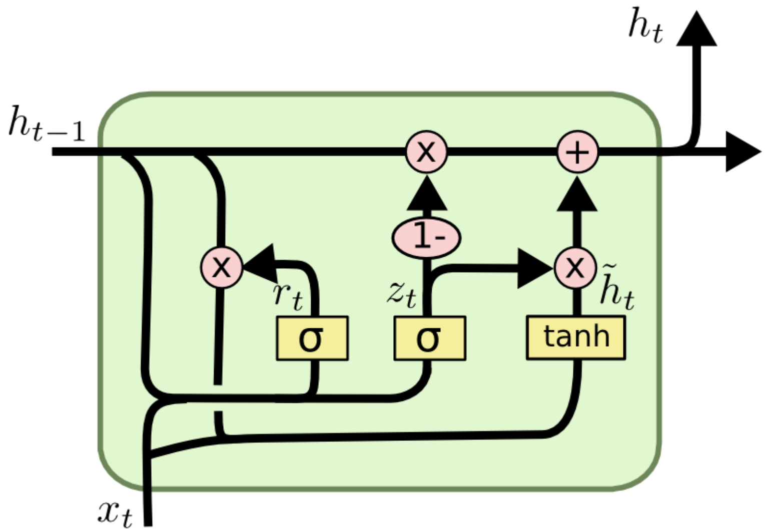
- Why not just let the cell directly influence the gates while at it
  - Party!!

The complete LSTM unit



- With input, output, and forget gates and the peephole connection..

Gated Recurrent Units: Lets simplify the LSTM



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

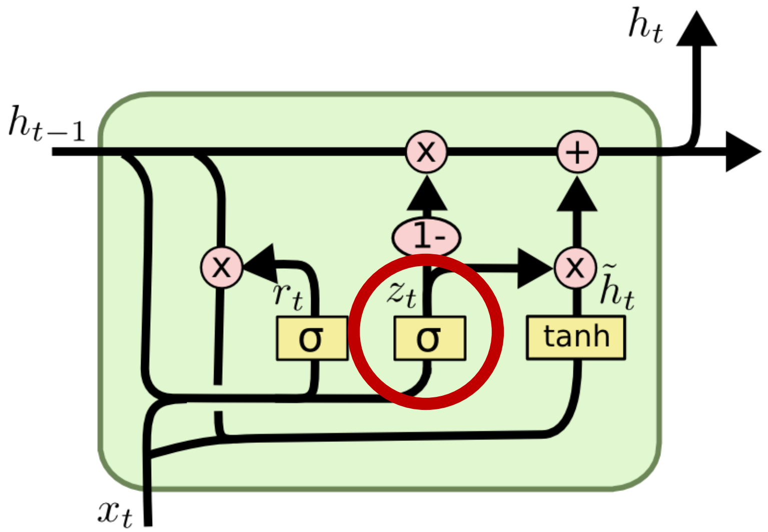
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

- Simplified LSTM which addresses some of your concerns of *why*

Gated Recurrent Units: Lets simplify the LSTM



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

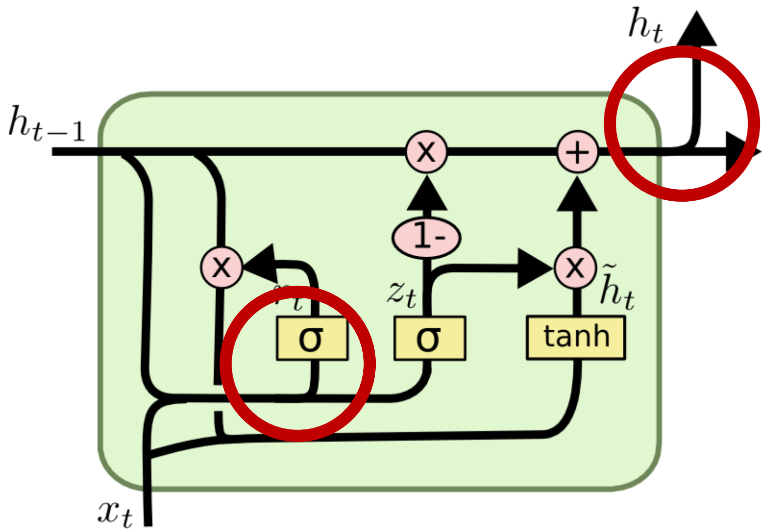
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

- Combine forget and input gates
  - In new input is to be remembered, then this means old memory is to be forgotten
    - Why compute twice?

Gated Recurrent Units: Lets simplify the LSTM



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

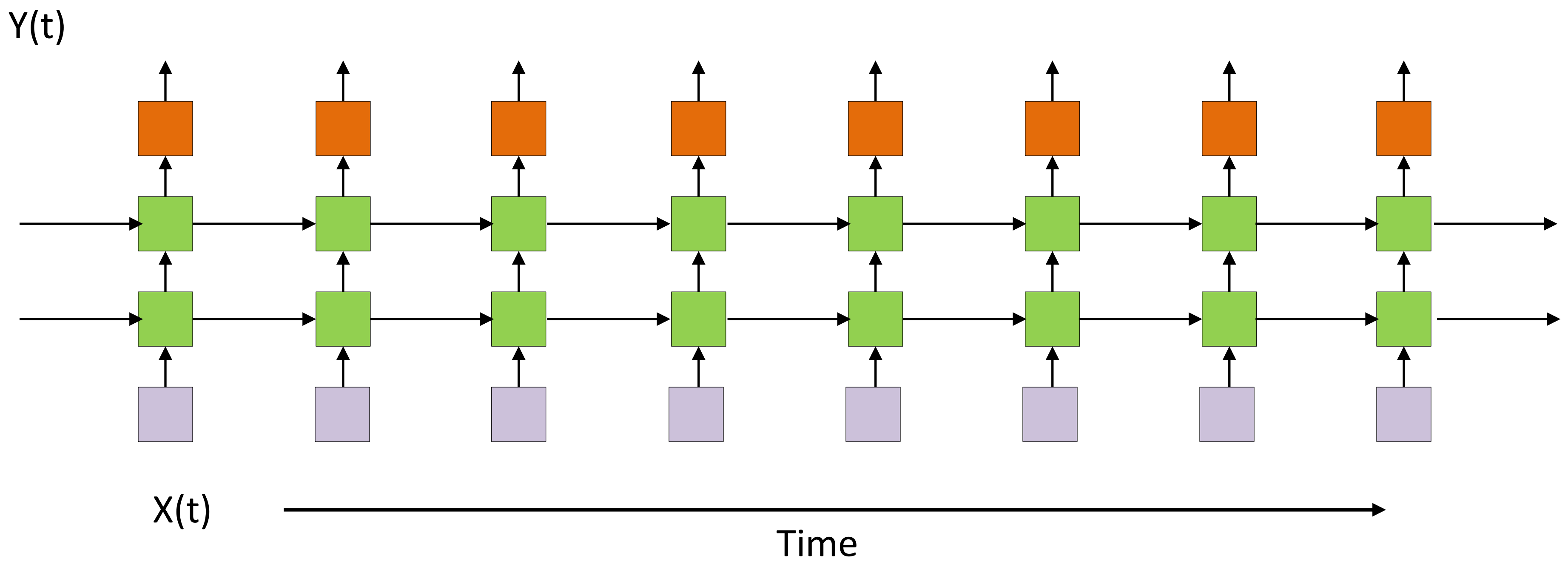
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

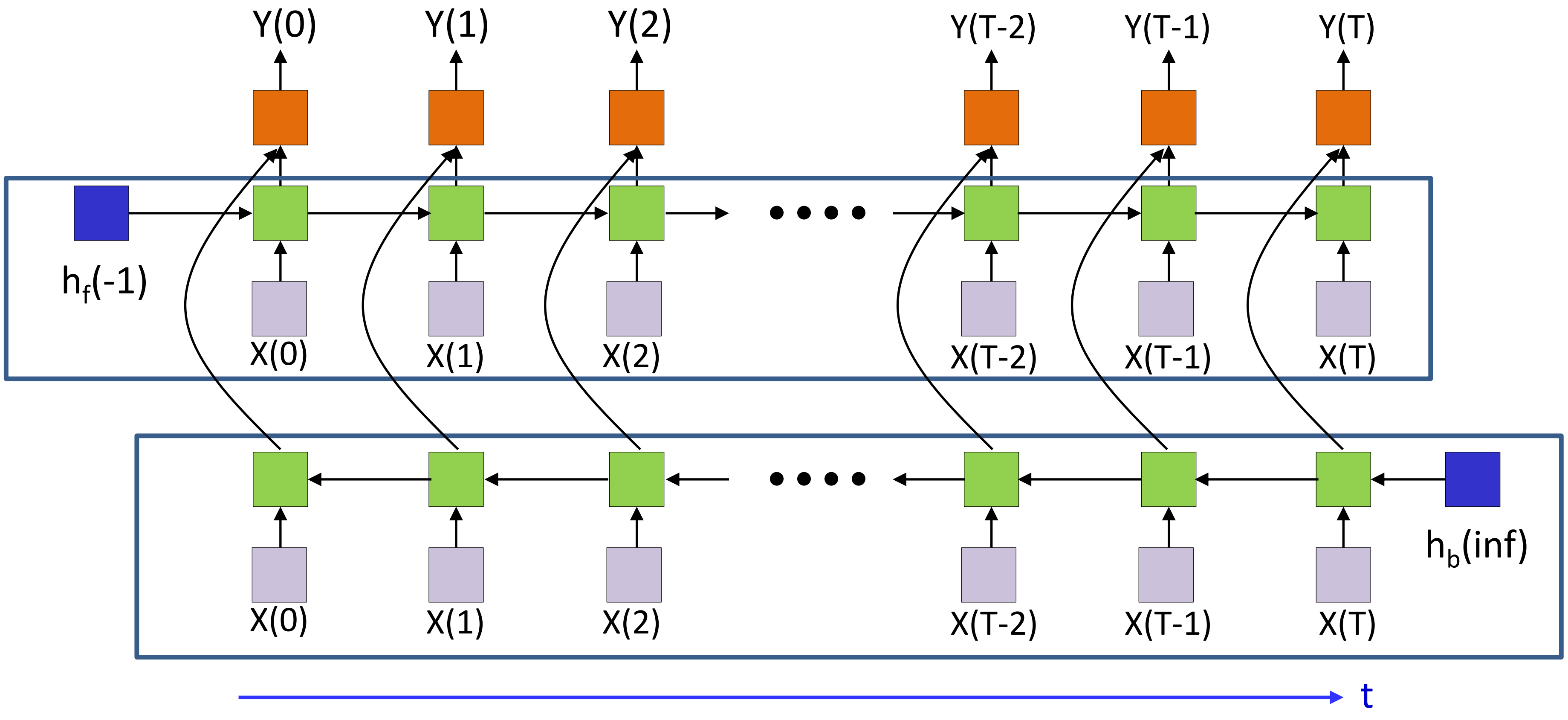
- Don’t bother to separately maintain compressed and regular memories
  - Pointless computation!
- But compress it before using it to decide on the usefulness of the current input!

LSTM architectures example



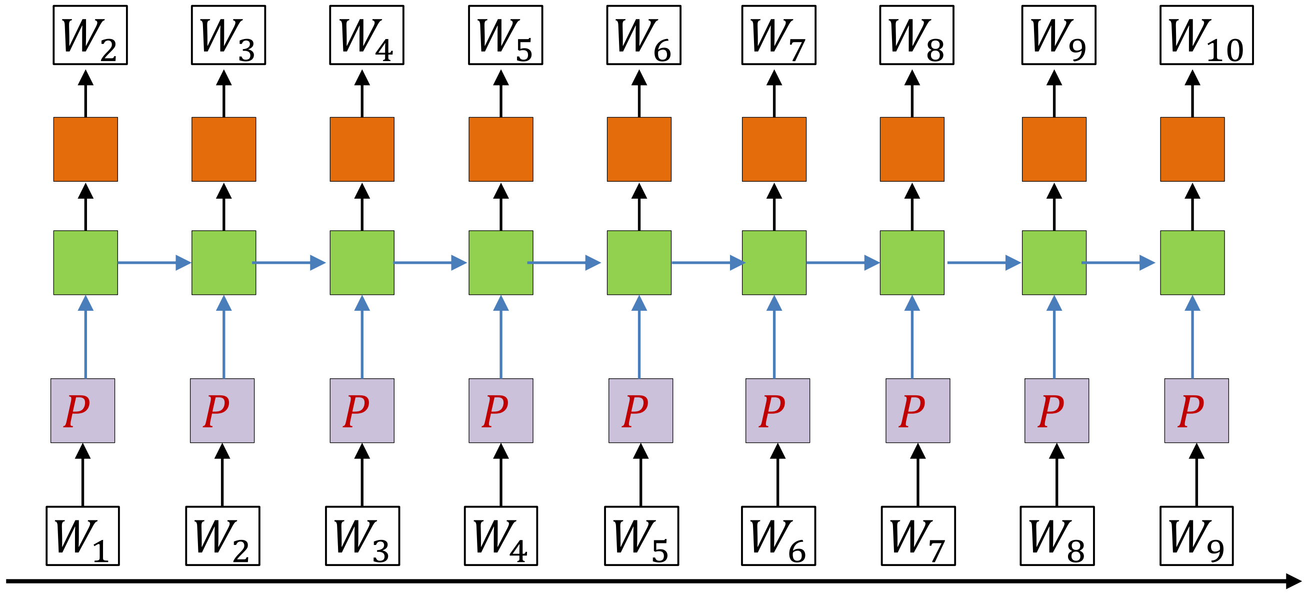
- Each green box is now an entire LSTM or GRU unit
- Also keep in mind each box is an *array* of units

Bidirectional LSTM



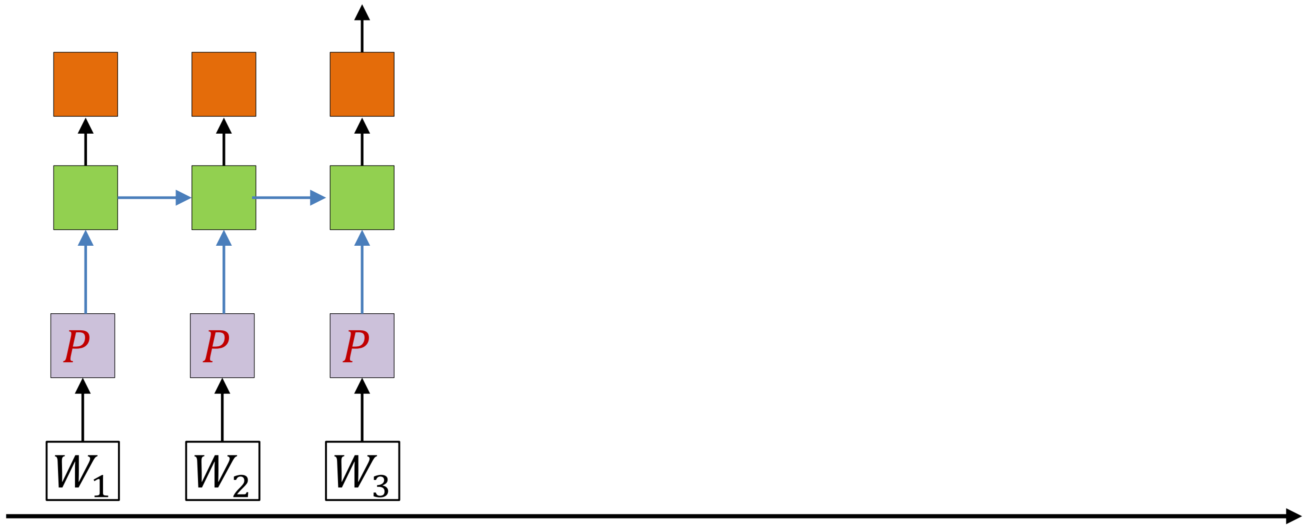
- Like the BRNN, but now the hidden nodes are LSTM units.
- Can have multiple layers of LSTM units in either direction
  - Its also possible to have MLP feed-forward layers between the hidden layers..
- The output nodes (orange boxes) may be complete MLPs

Generating Language: The model



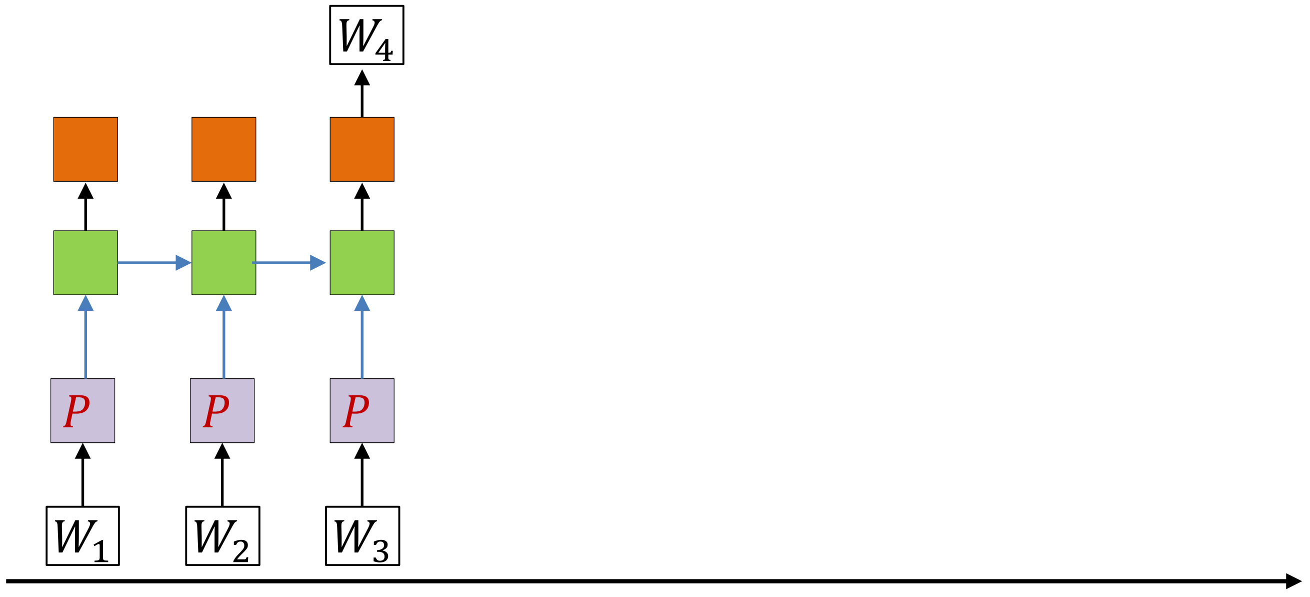
- The hidden units are (one or more layers of) LSTM units
- Trained via backpropagation from a lot of text

Generating Language: Synthesis



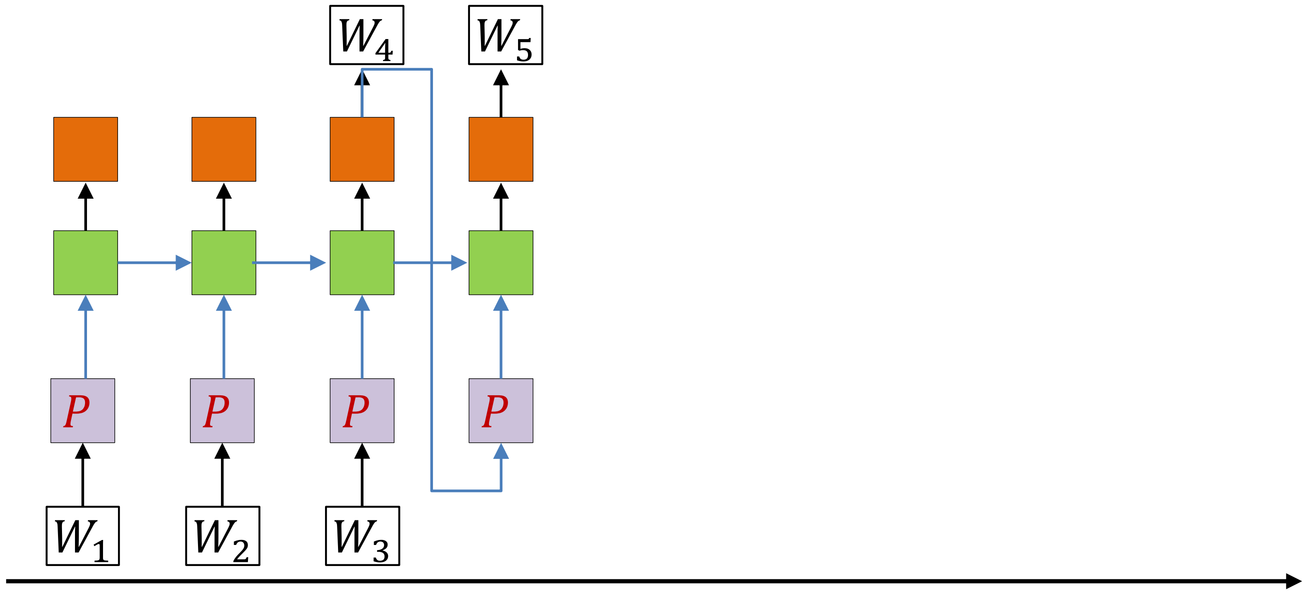
- On trained model : Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector

Generating Language: Synthesis



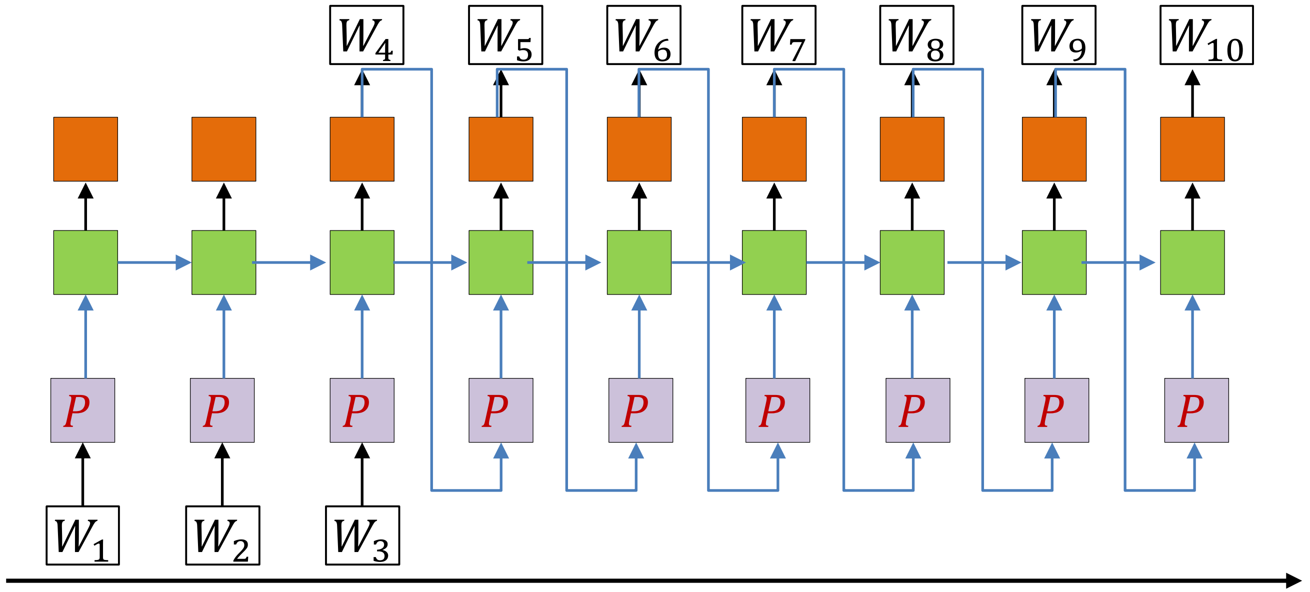
- On trained model : Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector
- Draw a word from the distribution
  - And set it as the next word in the series

Generating Language: Synthesis



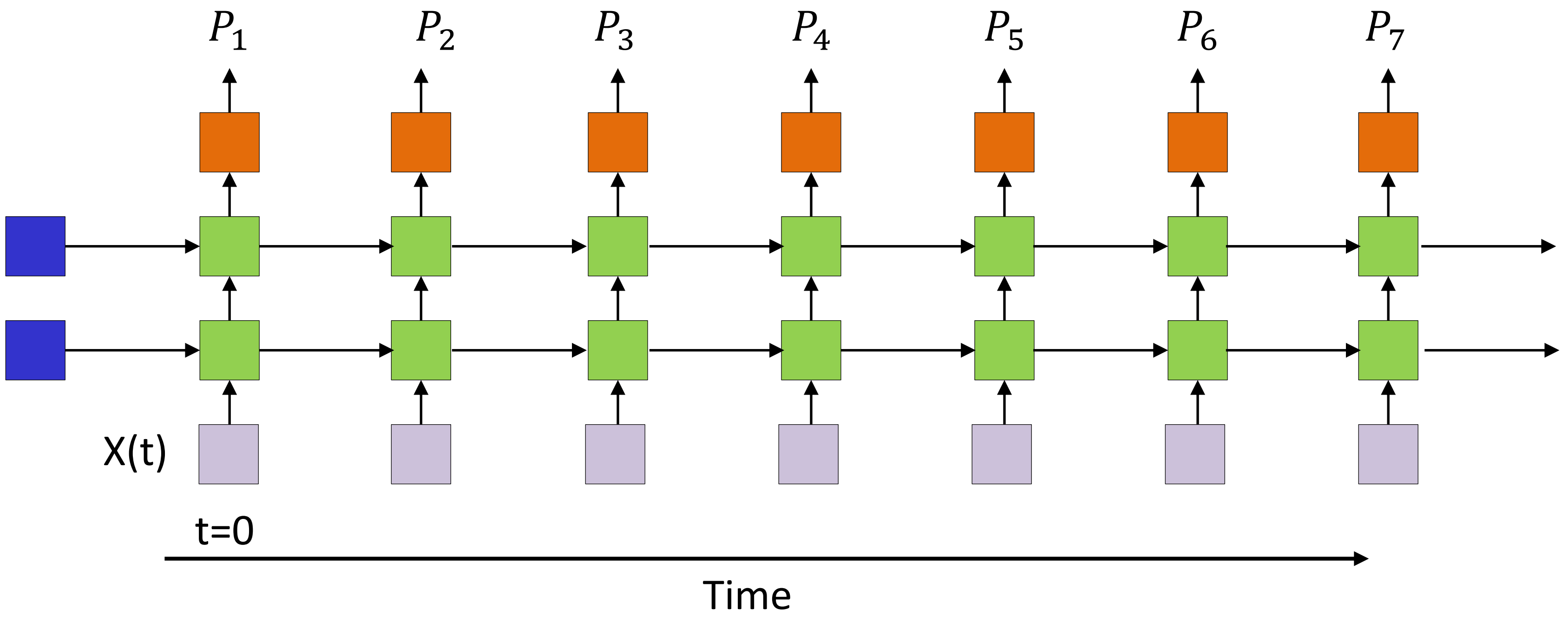
- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution

Generating Language: Synthesis



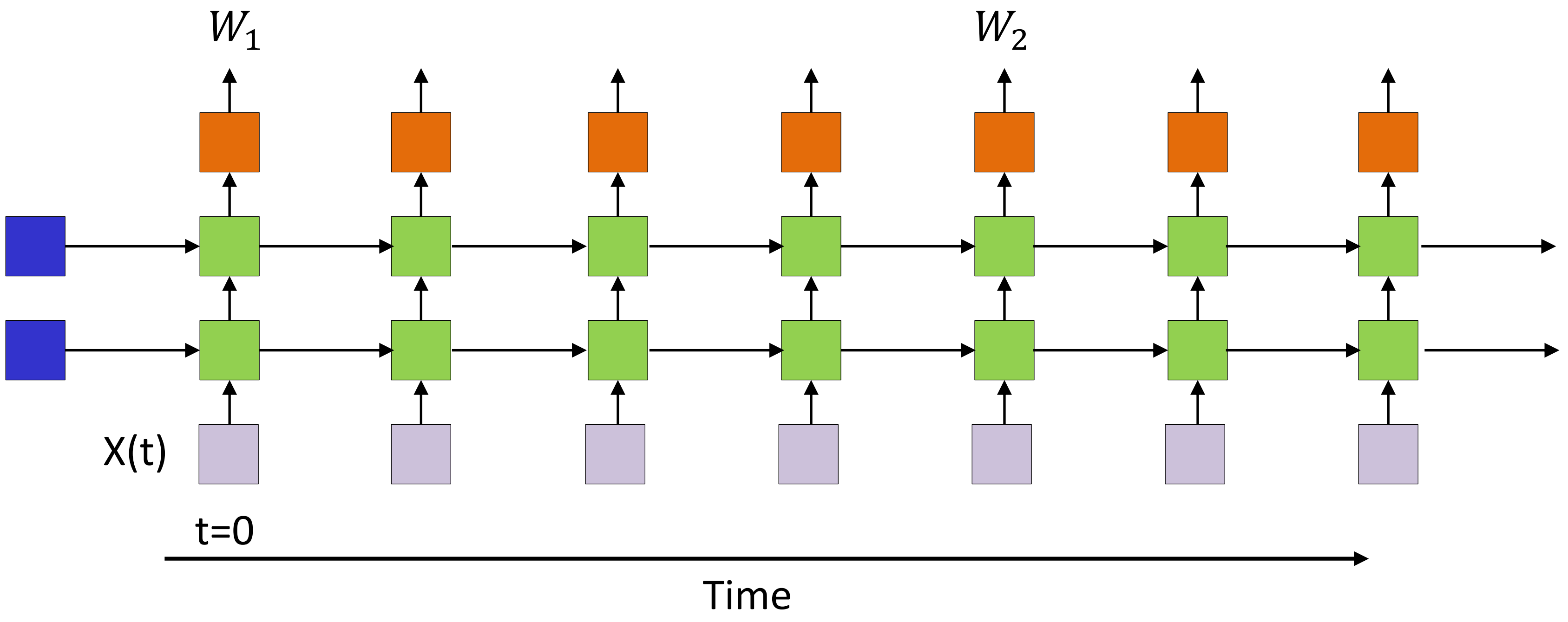
- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution
- Continue this process until we terminate generation
  - In some cases, e.g. generating programs, there may be a natural termination

Speech recognition using Recurrent Nets



- Recurrent neural networks (with LSTMs) can be used to perform speech recognition
  - Input: Sequences of audio feature vectors
  - Output: Phonetic label of each vector

Speech recognition using Recurrent Nets



- Alternative: Directly output phoneme, character or word sequence
- Challenge: How to define the loss function to optimize for training
  - Future lecture
  - Also homework

Problem: Ambiguous labels

- Speech data is continuous but the labels are discrete.
- Forcing a one-to-one correspondence between time steps and output labels is artificial.

Enter: CTC (Connectionist Temporal Classification)

A sophisticated loss layer that gives the network sensible feedback on tasks like speech recognition.

The idea

- Add “blanks” to the possible outputs of the network.
- Effectively serve as a pass on assigning a new label to the data meaning if a label has already been outputted it “leaves it as is”
- Analogous to a transcriber pausing in writing.

The implementation: Cost

Define the Label Error Rate as the mean edit distance.

$$LER(h, S') = \frac{1}{|S'|} \sum_{(\mathbf{x},\mathbf{z}) \in S'} \frac{ED(h(\mathbf{x}), \mathbf{z})}{|\mathbf{z}|}$$

Where S’ is the test set.

The implementation: Cost

This differs from errors used by other speech models in being characterize rather than word or wise.

This causes it to only indirectly learn a language model but also makes it more suitable for use with RNNs since they can Simply output the character or a blank.

The implementation: Path

The formula for the probability of a path π is given by

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y_{\pi_t}^{t}, \ \forall \pi \in L'^{T}.$$

For sequence x and outputs $y^t_{\pi t}$ at time step t for the value in path π

The implementation: Probability of Labeling

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}).$$

Where l is a labeling of x and Beta is the set of possible labeling of length less than or equal to the length of l

The implementation: Path

Great we have a closed for solution for the probability of a labeling!

The implementation: Path

Great we have a closed for solution for the probability of a labeling!

Problem: This is exponential in size.

It is on the order of the number of paths through the labels.

The implementation: Efficiency

The Solution: Dynamic programming

The probability of each label in the labeling is dependent on all other labels.

These can be computed with two variable Alpha and Beta corresponding to the probabilities of a valid prefix and suffix respectively.

The implementation: Efficiency

Alpha is the forward probability for s. It is defined as the sum over the probabilities of all possible prefixes for which s is a viable label in position t.

$$\alpha_t(s) \overset{\text{def}}{=} \sum_{\substack{\pi \in N^T: \\ \mathcal{B}(\pi_{1:t}) = \mathbf{l}_{1:s}}} \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'} .$$

The implementation: Efficiency

This can be implemented recursively as follows on l' the modified target label sequence with a blank in-between every symbol and at the beginning and end.

$$\alpha_1(1) = y_b^1$$

$$\alpha_1(2) = y_{\mathbf{l}_1}^1$$

$$\alpha_1(s) = 0, \ \forall s > 2$$

$$\bar{\alpha}_t(s) \overset{\text{def}}{=} \alpha_{t-1}(s) + \alpha_{t-1}(s-1).$$

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y_{\mathbf{l}'_s}^t & \text{if } \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s-2} = \mathbf{l}'_s \\ \big(\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)\big) y_{\mathbf{l}'_s}^t & \text{otherwise} \end{cases}$$

The implementation: Efficiency

The backwards pass:

$$\beta_t(s) \overset{\text{def}}{=} \sum_{\substack{\pi \in N^T: \\ \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}|}}} \prod_{t'=t}^{T} y^{t'}_{\pi_{t'}}$$

The implementation: Efficiency

The backwards pass:

$$\beta_t(s) \overset{\text{def}}{=} \sum_{\substack{\pi \in N^T: \\ \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}|}}} \prod_{t'=t}^{T} y^{t'}_{\pi_{t'}}$$

Analogous to forward pass

The implementation: Efficiency

Recursive definition

$$\beta_T(|\mathbf{l}'|) = y_b^T$$

$$\beta_T(|\mathbf{l}'|-1) = y_{\mathbf{l}_{|\mathbf{l}|}}^T$$

$$\beta_T(s) = 0, \ \forall s < |\mathbf{l}'| - 1$$

$$\bar{\beta}_t(s) \overset{\text{def}}{=} \beta_{t+1}(s) + \beta_{t+1}(s+1).$$

$$\beta_t(s) = \begin{cases} \bar{\beta}_t(s) y_{\mathbf{l}'_s}^t & \text{if } \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s+2} = \mathbf{l}'_s \\ \left(\bar{\beta}_t(s) + \beta_{t+1}(s+2)\right) y_{\mathbf{l}'_s}^t & \text{otherwise} \end{cases}$$

The implementation: Efficiency

Recursive definition

$$\beta_T(|\mathbf{l}'|) = y_b^T$$

$$\beta_T(|\mathbf{l}'|-1) = y_{\mathbf{l}_{|\mathbf{l}|}}^T$$

$$\beta_T(s) = 0, \ \forall s < |\mathbf{l}'| - 1$$

$$\bar{\beta}_t(s) \stackrel{\text{def}}{=} \beta_{t+1}(s) + \beta_{t+1}(s+1).$$

$$\beta_t(s) = \begin{cases} \bar{\beta}_t(s) y_{\mathbf{l}'_s}^t & \text{if } \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s+2} = \mathbf{l}'_s \\ \left(\bar{\beta}_t(s) + \beta_{t+1}(s+2)\right) y_{\mathbf{l}'_s}^t & \text{otherwise} \end{cases}$$

The implementation: Efficiency

What this gets us and intuition

The implementation: Efficiency

What this gets us and intuition

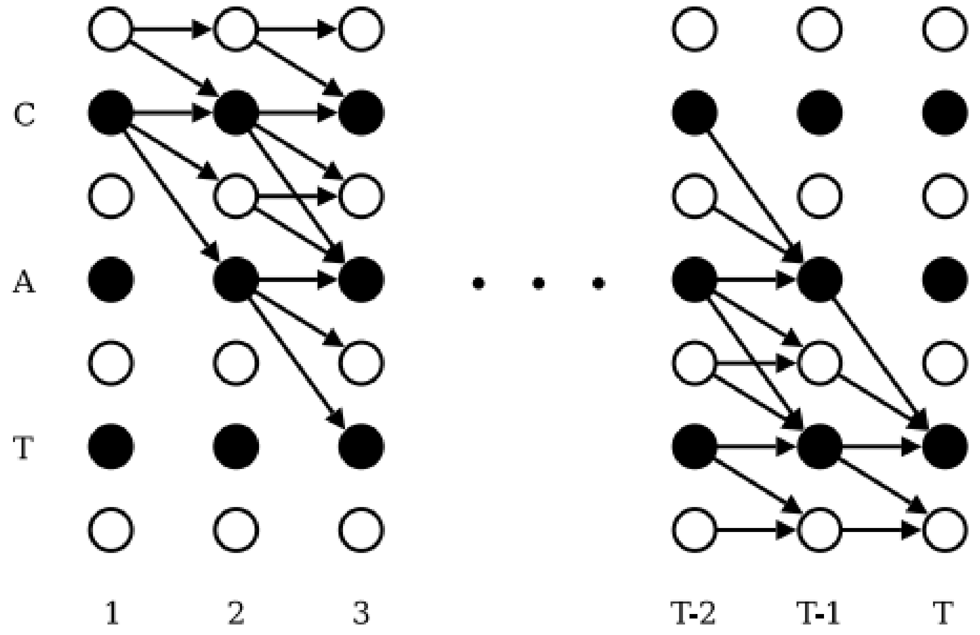


Illustration of CTC

Implementation

Rescale to avoid underflow

$$C_t \stackrel{\text{def}}{=} \sum_s \alpha_t(s), \qquad \hat{\alpha}_t(s) \stackrel{\text{def}}{=} \frac{\alpha_t(s)}{C_t}$$

$$D_t \stackrel{\text{def}}{=} \sum_s \beta_t(s), \qquad \hat{\beta}_t(s) \stackrel{\text{def}}{=} \frac{\beta_t(s)}{D_t}$$

By substituting Alpha for Alpha-hat and Beta for Beta-hat

Implementation

Returning to the task at hand the maximum likelihood objective function is:

$$O^{ML}(S, \mathcal{N}_w) = - \sum_{(\mathbf{x},\mathbf{z}) \in S} ln(p(\mathbf{z}|\mathbf{x}))$$

Implementation

Recall

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y_{\pi_t}^{t}, \ \forall \pi \in L'^{T}.$$

Implementation

Recall

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi_t}, \ \forall \pi \in L'^T.$$

Consider

$$\alpha_t(s)\beta_t(s) = \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = \mathbf{l}'_s}} y^t_{\mathbf{l}'_s} \prod_{t=1}^{T} y^t_{\pi_t}.$$

This is the product of the forward and backwards probabilities.

By substitution

$$\frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}} = \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = \mathbf{l}'_s}} p(\pi|\mathbf{x}).$$

Implementation

Recall
$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}).$$

Thus
$$p(\mathbf{l}|\mathbf{x}) = \sum_{s=1}^{|\mathbf{l}'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}}.$$

This is a cost we can compute