Fundamentals of AI/ML

Recurrent Neural Networks

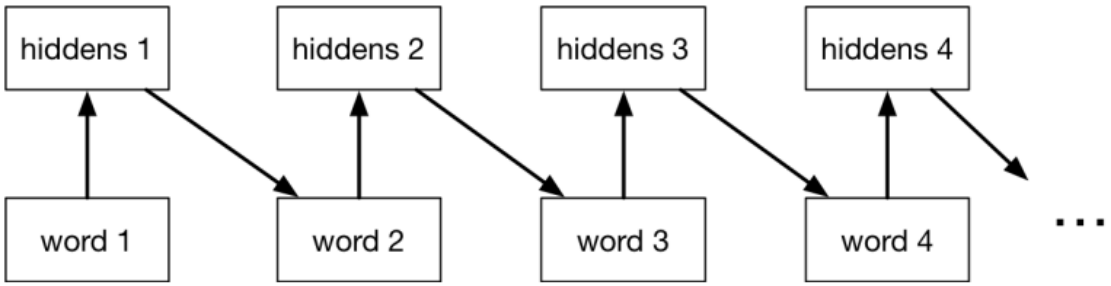
Ashley Y. Gao

William & Mary

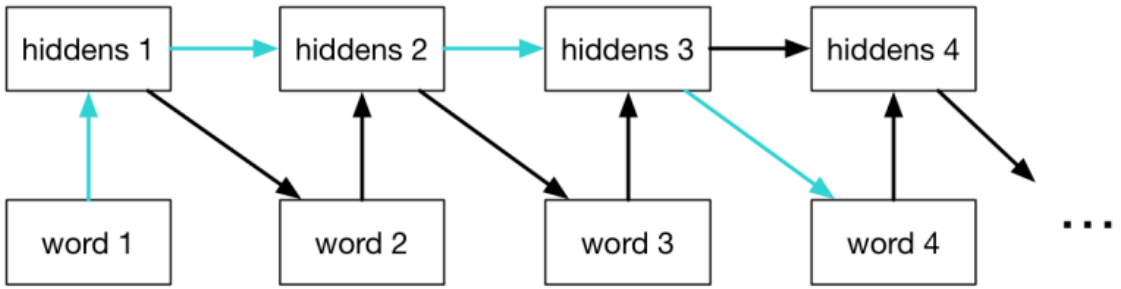
April 20, 2025

Ovierview

- Autoregressive models such as the neural language model are memoryless
  - They can only use information from their immediate context (in this figure, context length = 1)

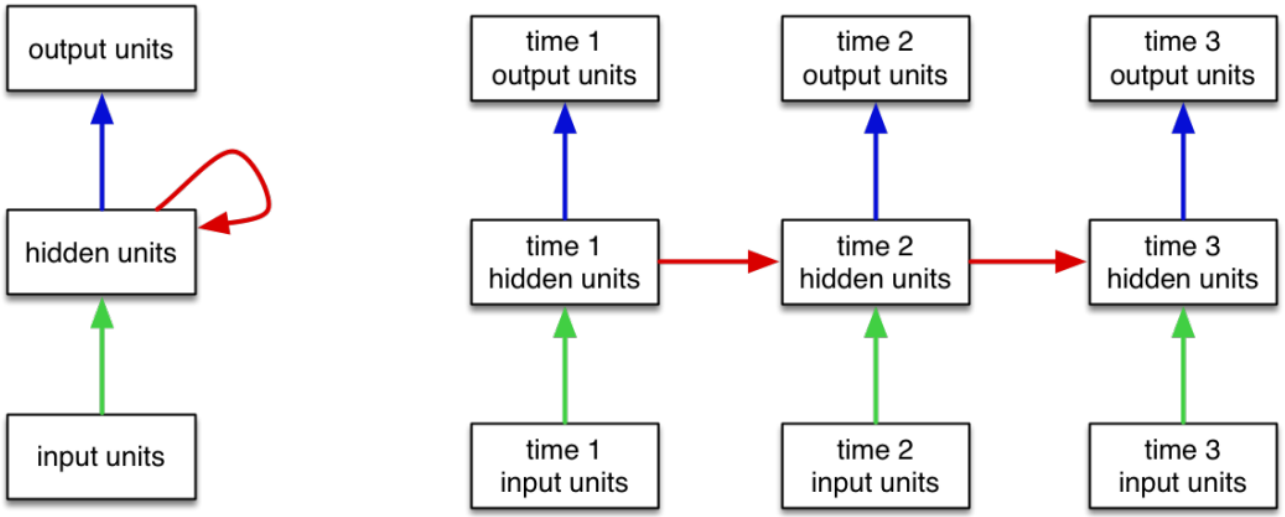


- If we add connections between the hidden units, it becomes a recurrent neural network (RNN)
  - Having a memory lets an RNN use longer-term dependencies:



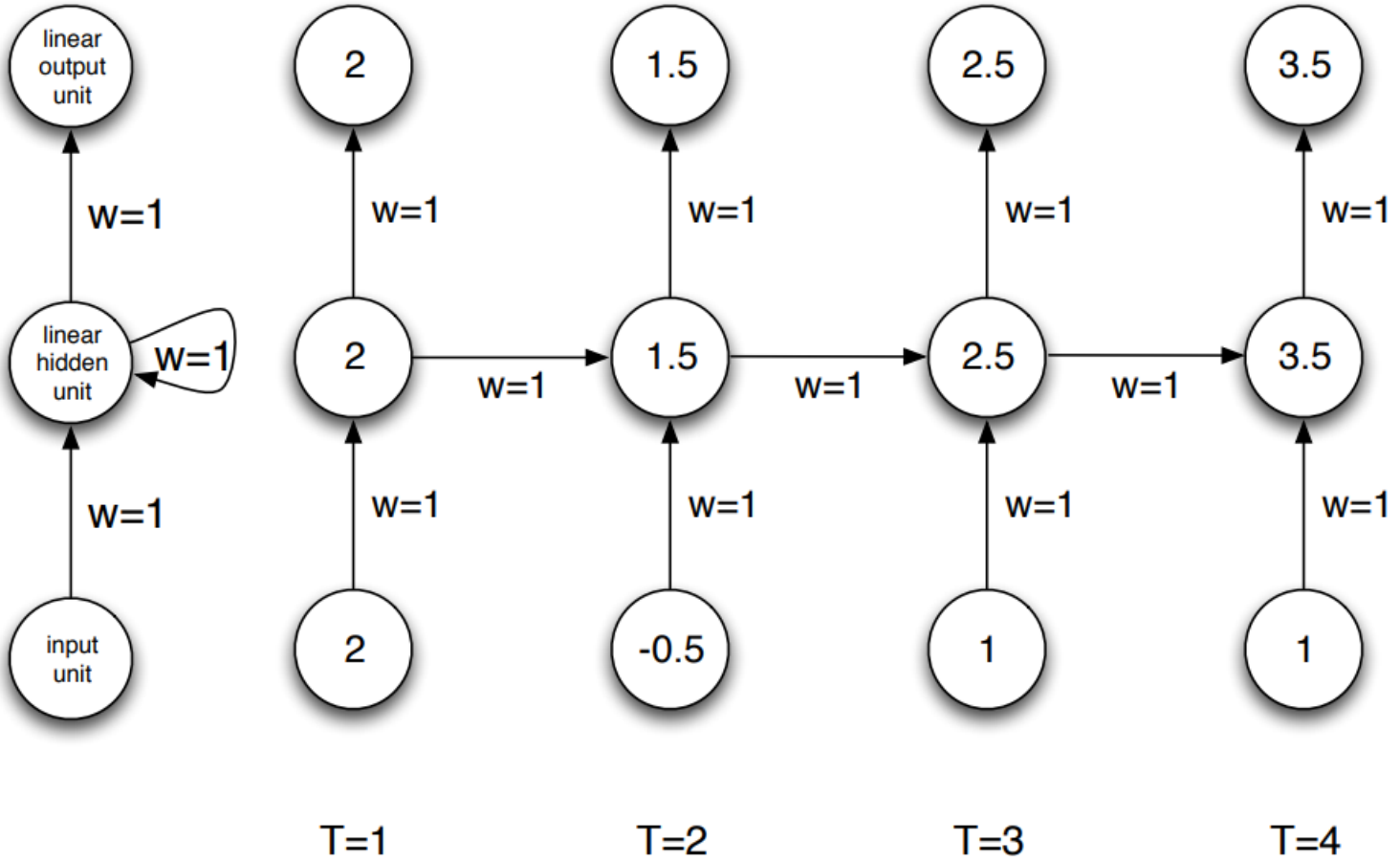
Recurrent Neural Nets

- We can think of an RNN as a dynamic system with one set of hidden units which feed into themselves
  - The network's graph would then have self-loops
- We can unroll the RNN's graph by explicitly representing the units at all time steps.
  - The weights and biases are shared between all time steps
  - Except there is typically a separate set of biases for the first time step



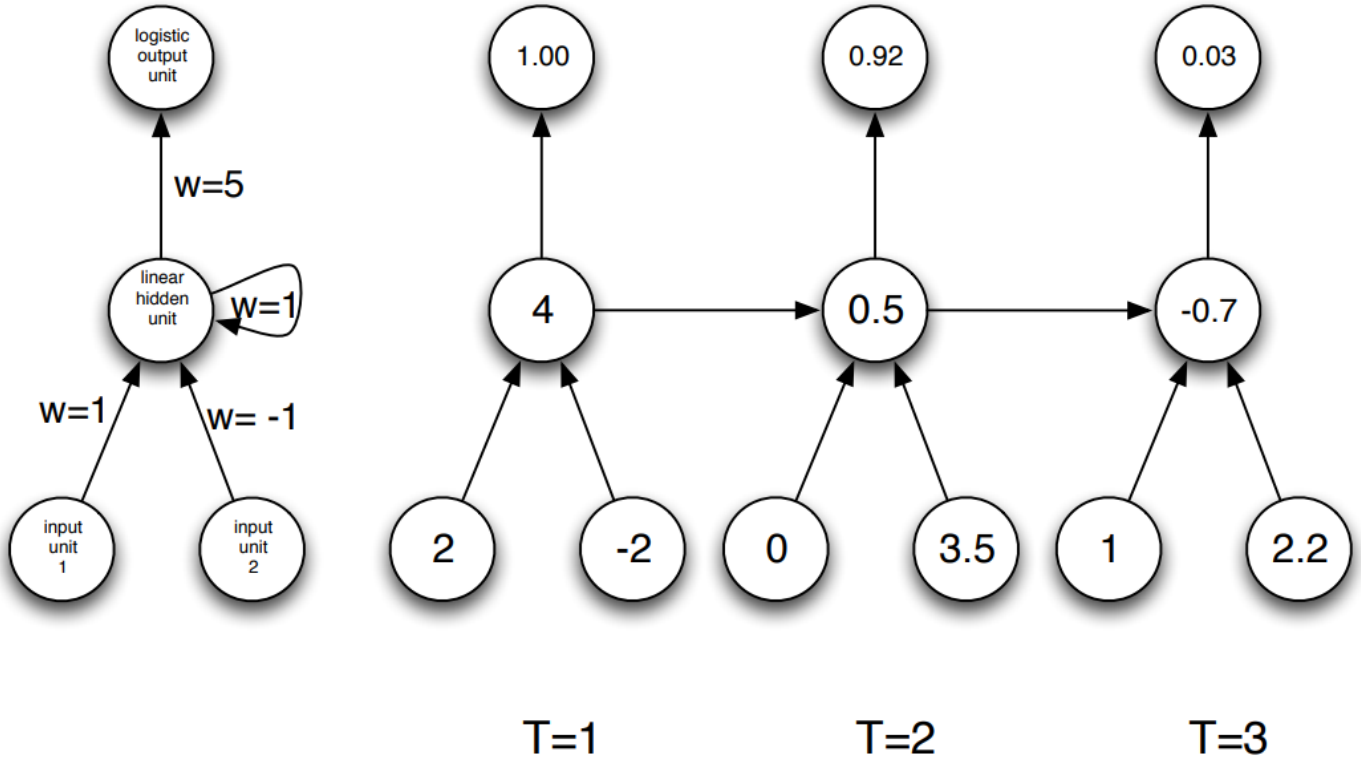
RNN Examples

- Now let's look at some simple examples of RNNs.
- This one sums its inputs



RNN Examples

- What about this one?



Example: Parity

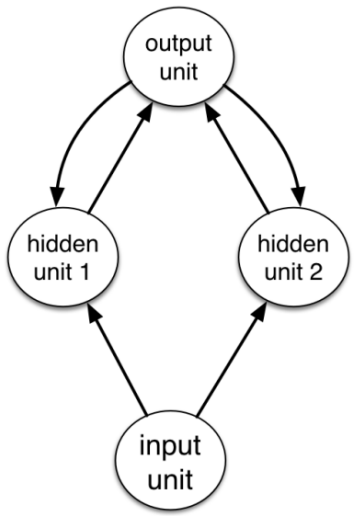
- Assume we have a sequence of binary inputs. We will consider how to determine the parity.
  - i.e. whether the number of 1's is even or odd
- We can compute parity incrementally by keeping track of the parity of the input so far.

$$\begin{array}{rl} \text{Parity bits:} & 0\ 1\ 1\ 0\ 1\ 1 \longrightarrow \\ \text{Input:} & 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \end{array}$$

- Each parity bit is the XOR of the input and the previous parity bit
- Parity is the classic example of a problem that is hard to solve with a shallow feed-forward net, but easy to solve with an RNN

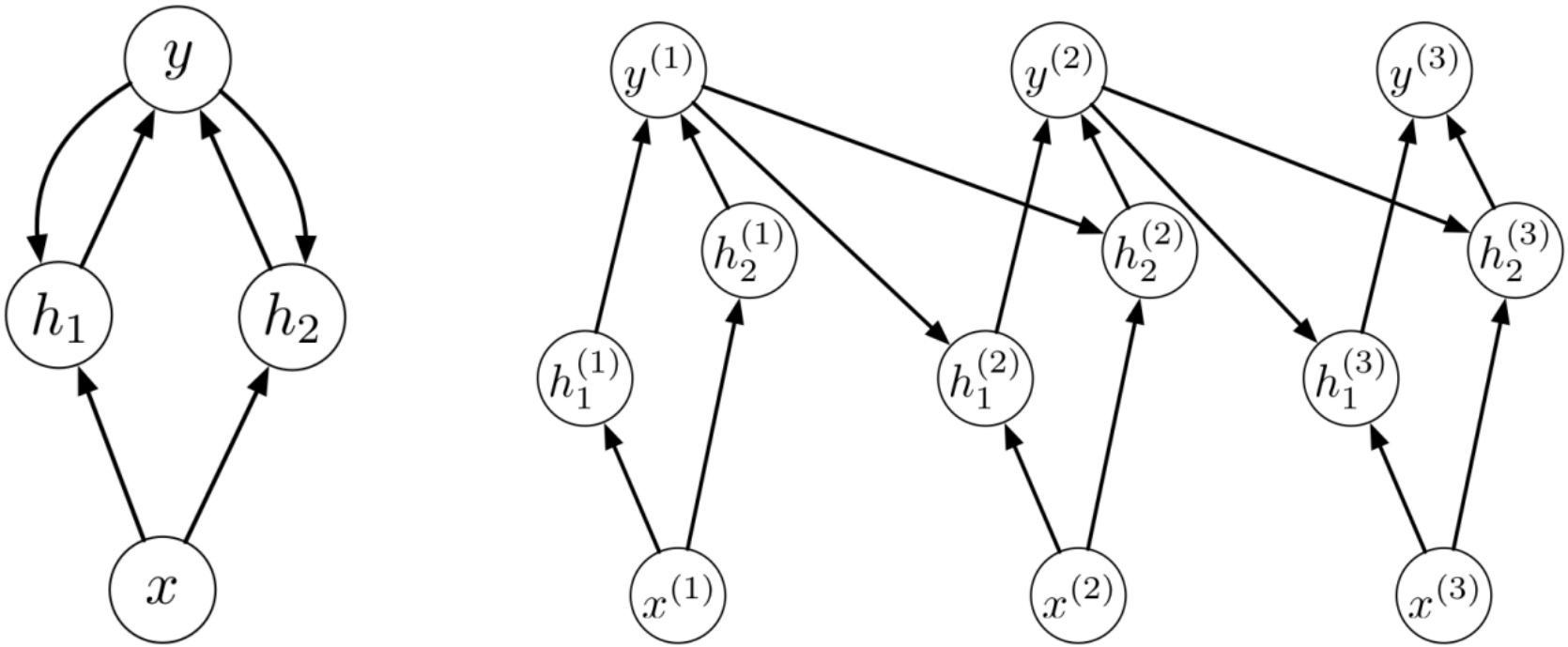
Example: Parity

- Assume we have a sequence of binary inputs. We will consider how to determine the parity.
- Let's find weights and biases for the RNN so that it computes the parity. All hidden and output units are binary threshold units.
- Strategy:
  - The output unit tracks the current parity, which is the XOR of the current input and its previous output
  - the hidden units help us compute the XOR



Example: Parity

- Unrolling the parity RNN:



Example: Parity

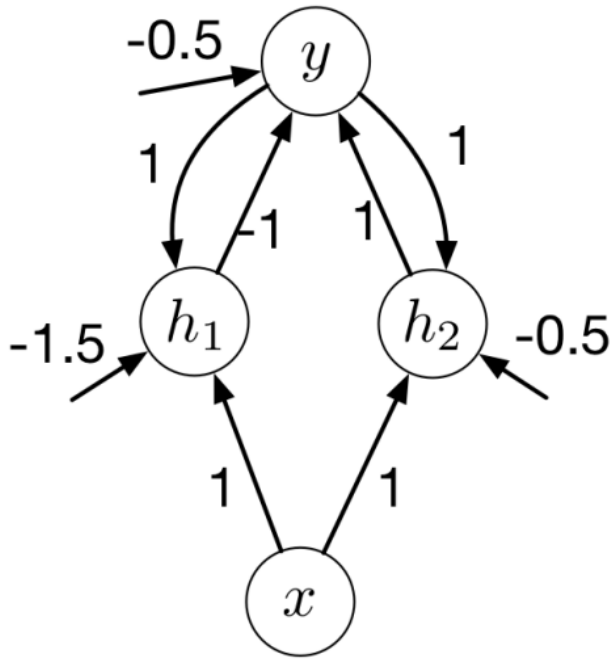
- The output unit should compute the XOR of the current input and previous output:

$y^{(t-1)}$	$x^{(t)}$	$y^{(t)}$
0	0	0
0	1	1
1	0	1
1	1	0

Example: Parity

- Let's use hidden units to help us compute XOR
  - Have one unit compute AND, and the other one compute OR
  - Then we can pick weights and biases just like we did for multilayer perceptrons

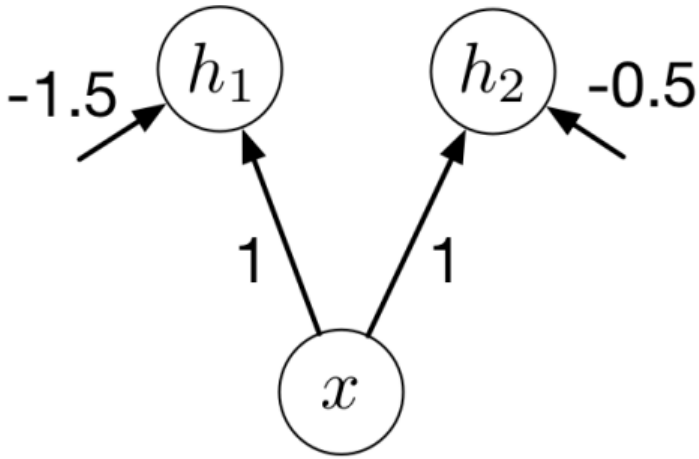
$y^{(t-1)}$	$x^{(t)}$	$h_1^{(t)}$	$h_2^{(t)}$	$y^{(t)}$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



Example: Parity

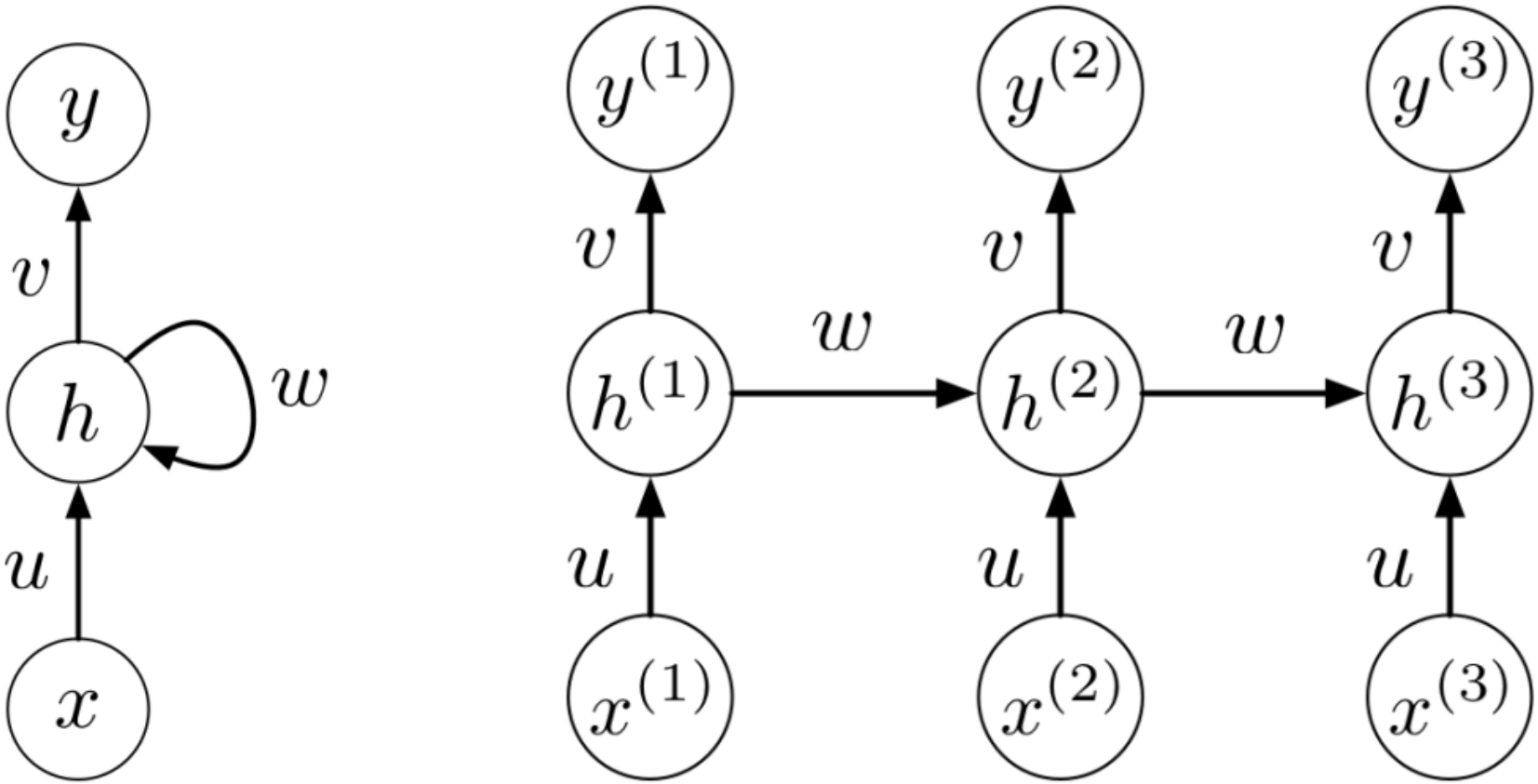
- We still need to determine hte hidden biases for the first time step
  - The network should behave as if the previous input was 0
  - This is represented with the following table

$x^{(1)}$	$h_1^{(1)}$	$h_2^{(1)}$
0	0	0
1	0	1



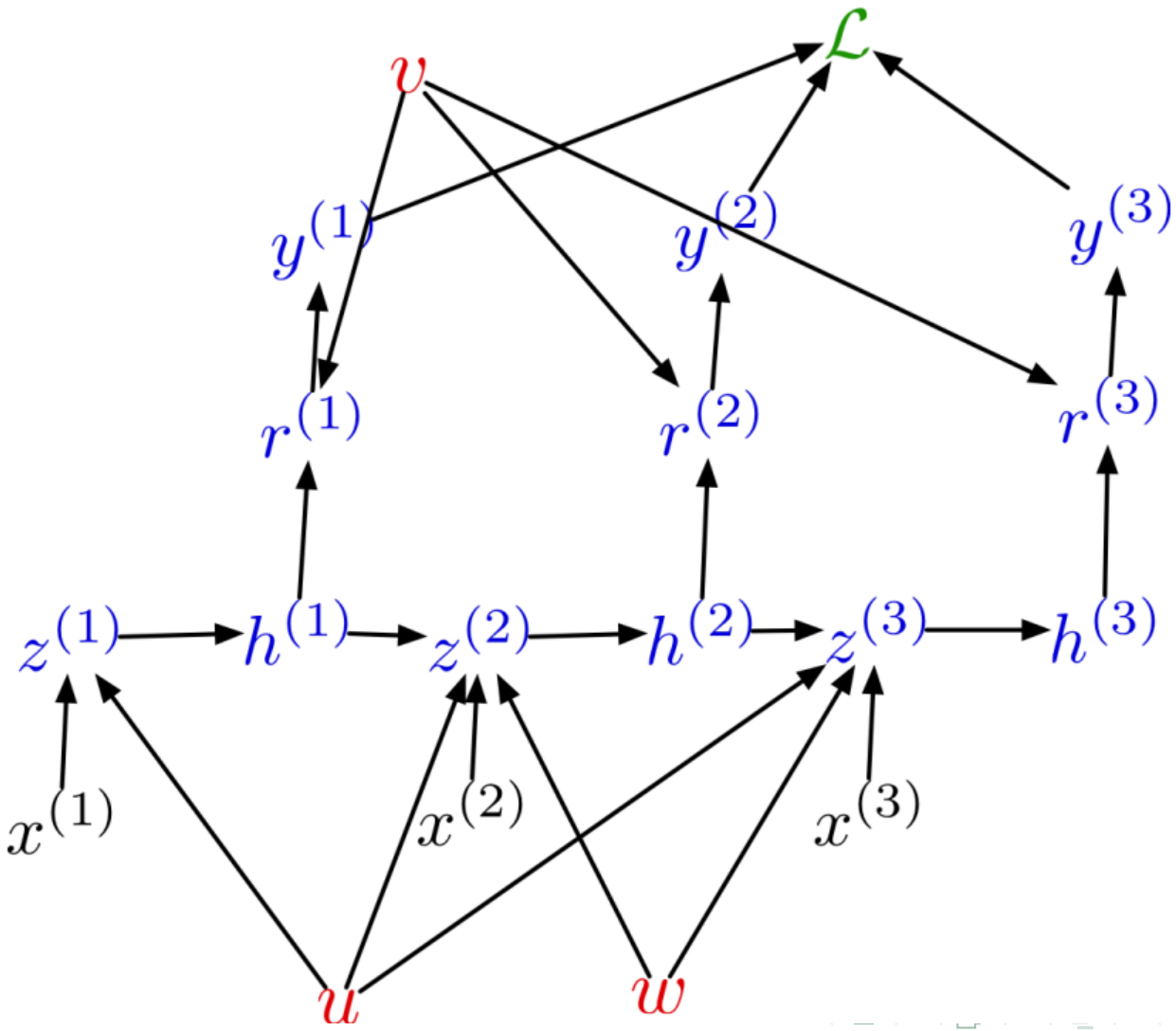
Backpropagation through Time

- As you can guess, we don't usually set RNN weights by hand
  - Instead, we learn them using backpropagation
- In particular, we do backpropagation on the unrolled network. This is known as backpropagation through time.



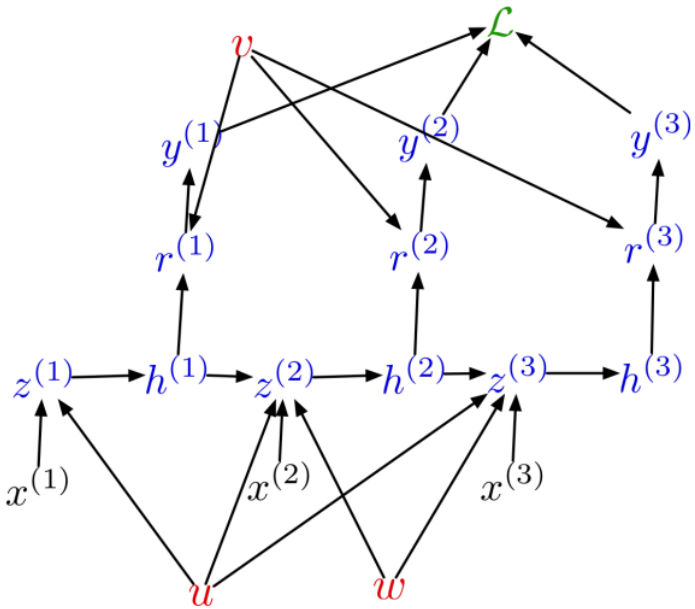
Backpropagation through Time

- Here's the unrolled computation graph. Notice the weight sharing



Backpropagation through Time

- Backpropagation formulas, according to the computation graph



Activations:

$$\overline{\mathcal{L}} = 1$$

$$\overline{y^{(t)}} = \overline{\mathcal{L}} \frac{\partial \mathcal{L}}{\partial y^{(t)}}$$

$$\overline{r^{(t)}} = \overline{y^{(t)}}\, \phi'(r^{(t)})$$

$$\overline{h^{(t)}} = \overline{r^{(t)}}\, v + \overline{z^{(t+1)}}\, w$$

$$\overline{z^{(t)}} = \overline{h^{(t)}}\, \phi'(z^{(t)})$$

Parameters:

$$\overline{u} = \sum_t \overline{z^{(t)}}\, x^{(t)}$$

$$\overline{v} = \sum_t \overline{r^{(t)}}\, h^{(t)}$$

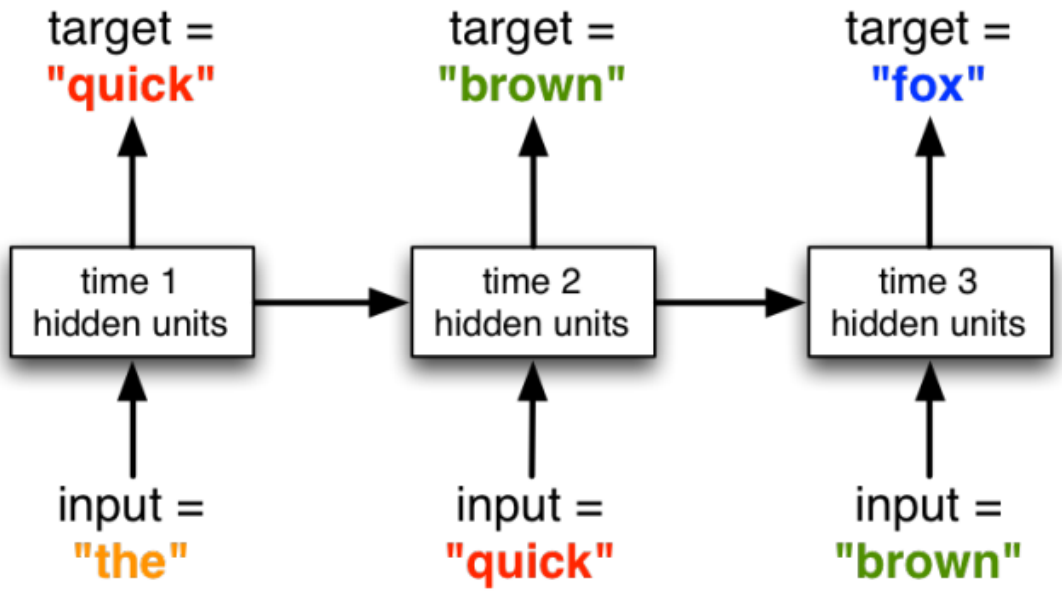
$$\overline{w} = \sum_t \overline{z^{(t+1)}}\, h^{(t)}$$

Backpropagation through Time

- Now you know how to compute the derivatives using backpropagation through time
- How are RNNs used in applications?

Language Model

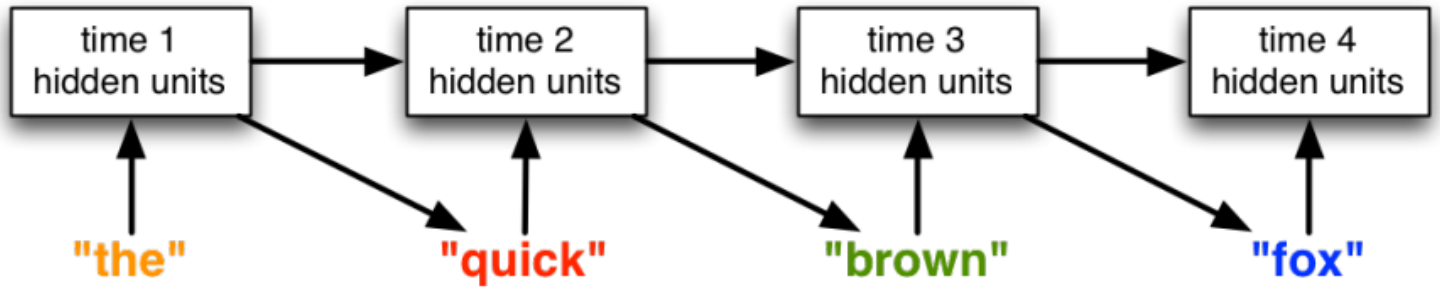
- One way to use RNNs as a language model:



- As with our language model, each word is represented as an indicator vector
  - The model is trained using cross-entropy loss
- This model can learn long-distance dependencies

Language Model

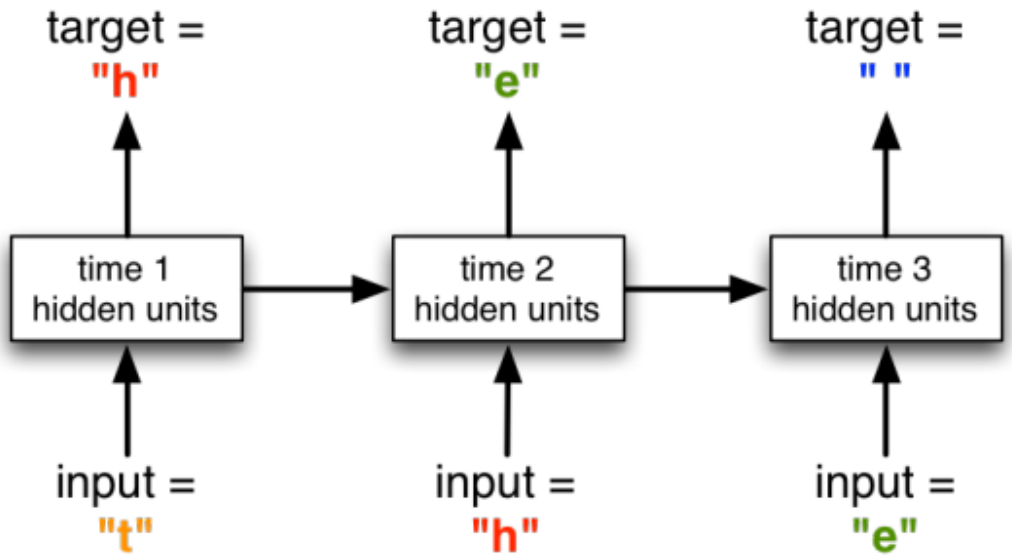
- We we generate from the model (i.e. compute samples from its distribution over sentences), the outputs feed back into the network as inputs



- Solve remaining challenges:
  - The vocabulary can be very large once you include people, places, etc.
    - It's computationally difficult to predict distributions over millions of words
  - How do we deal with words we haven't seen before?
  - In some languages (e.g. German) it is hard to define what should be considered a word

Language Model

- Another approach is to model text one character at a time



- This solves the problem of what to do about previously unseen words
  - Note that long-term memory is essential at character level

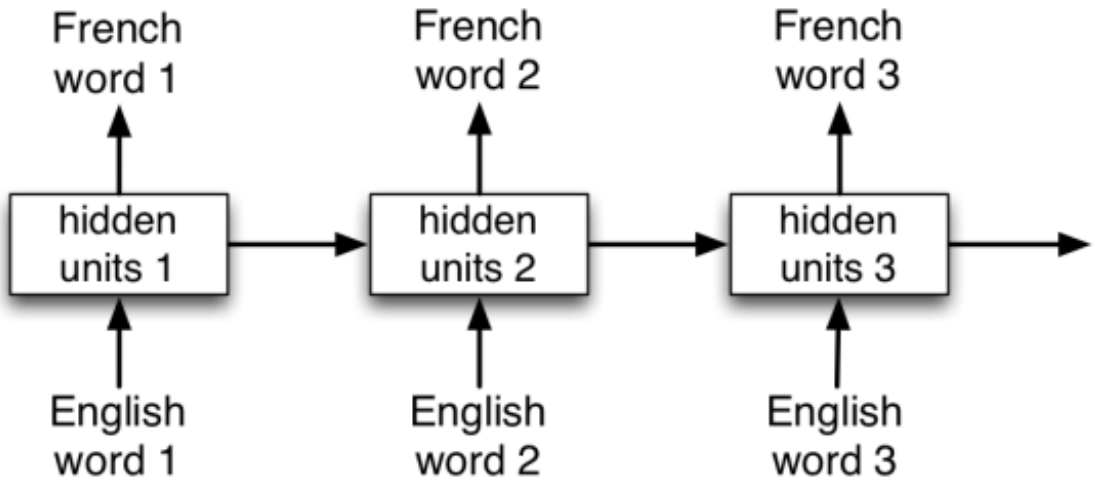
Language Model

- From Geoff Hinton's Coursera course, an example of a paragraph generated by an RNN language model one character at a time:

> He was elected President during the Revolutionary War and forgave Opus Paul at Rome. The regime of his crew of England, is now Arab women's icons in and the demons that use something between the characters‘ sisters in lower coil trains were always operated on the line of the ephemerable street, respectively, the graphic or other facility for deformation of a given proportion of large segments at RTUS). The B every chord was a "strongly cold internal palette pour even the white blade."

Neural Machine Translation

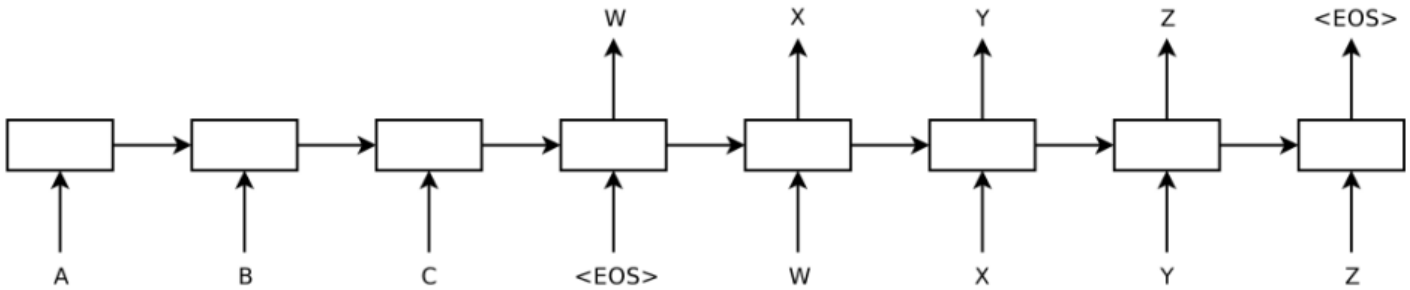
- We'd like to translate, e.g., English to French sentences, and we have pairs of translated sentences to train on
- What's wrong with the following setup?



- The sentences might not be the same length, and the words might not align perfectly
- You might need to resolve ambiguities using information from later in the sentence

Neural Machine Translation

- Instead, the network first reads and memorizes the sentence
- When it sees the END token, it starts outputting the translation



What Can RNNs Compute

- In 2014, Google built an RNN that learns to execute simple Python programs, one character at a time

Input:

```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```

Target: 25011.

Input:

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else
    5308)
```

Target: 1218.

Example training inputs

Input:

```
vqppkn
sqdvfljmnc
y2vxdddsepnimcbvubkomhrpliibtwztbljipcc
```

Target: hkhpg

A training input with characters scrambled

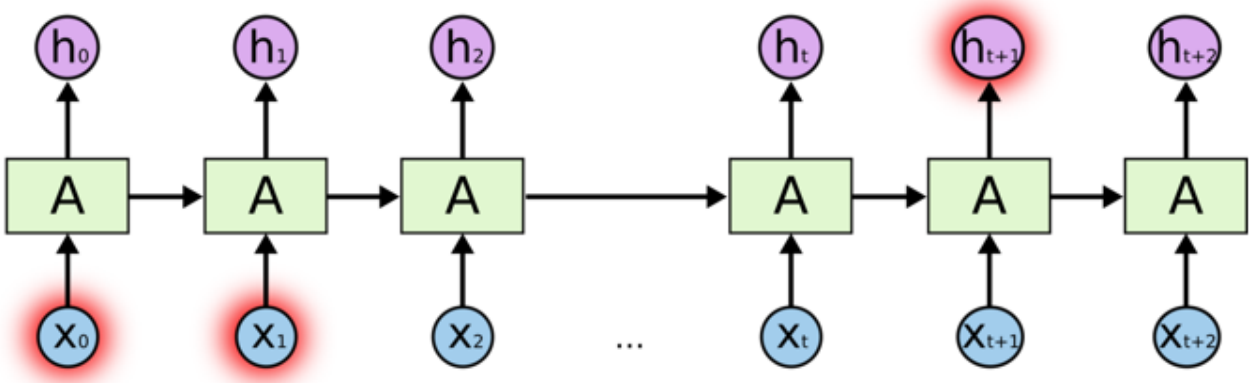
Long Short Term Memory (LSTM)

Vanishing and Exploding Gradient Problem

- Backpropagated errors multiply at each layer, resulting in exponential decay (if the derivative is small) or growth (if the derivative is large)
- Makes it very difficult train deep networks, or simple recurrent networks over many time steps
- We won't cover a lot about this topic in this class

Long Distance Dependencies

- It's difficult to train a vanilla RNN to retain information over many time steps
- The RNN might have trouble remembering stuff, such as the subject-verb agreement

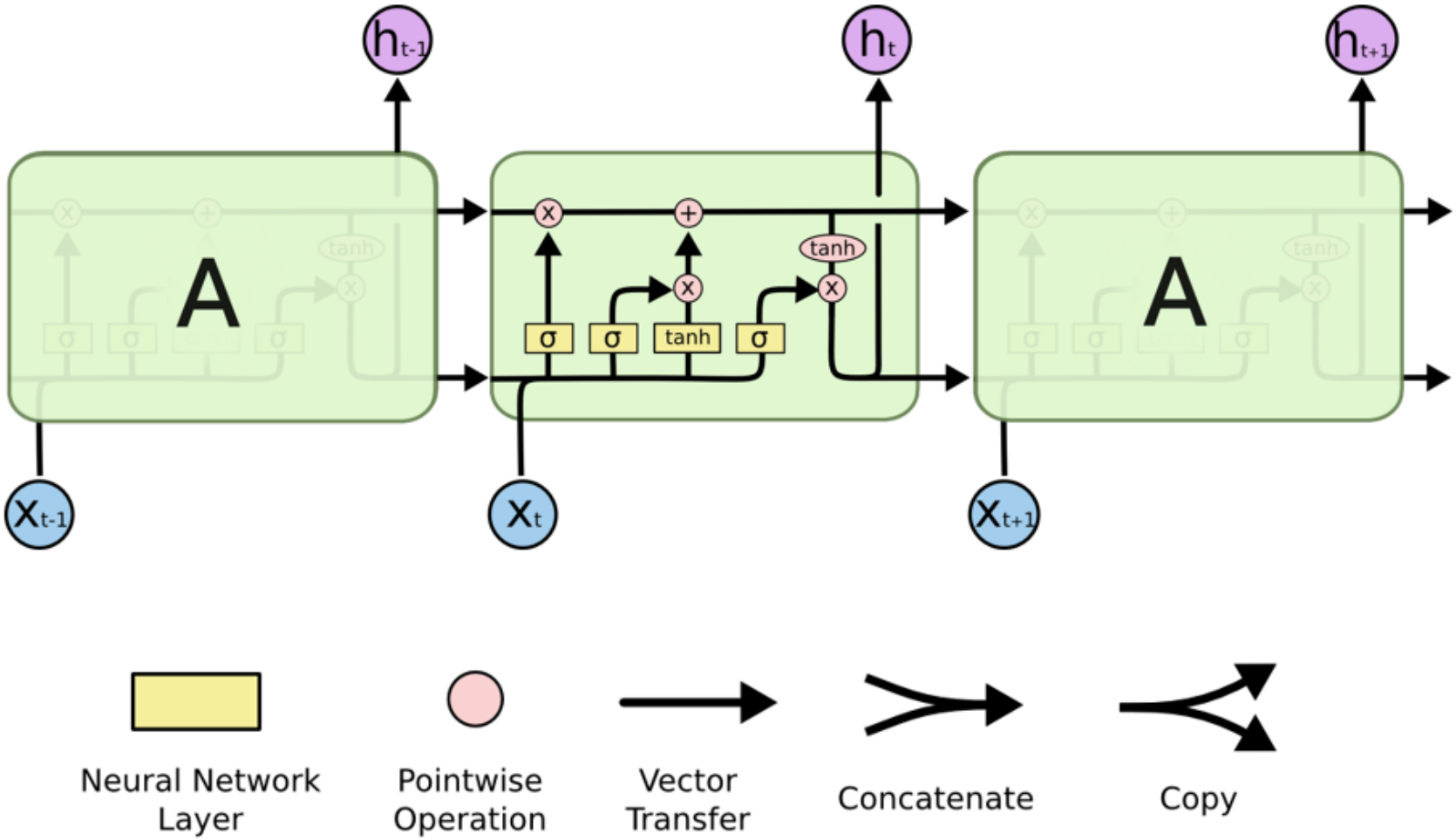


Long Short Term Memory

- LSTM networks, add additional gating units in each memory cell.
  - Forget gate
  - Input gate
  - Output gate
- Prevents vanishing/exploding gradient problem and allows network to retain state information over longer periods of time.

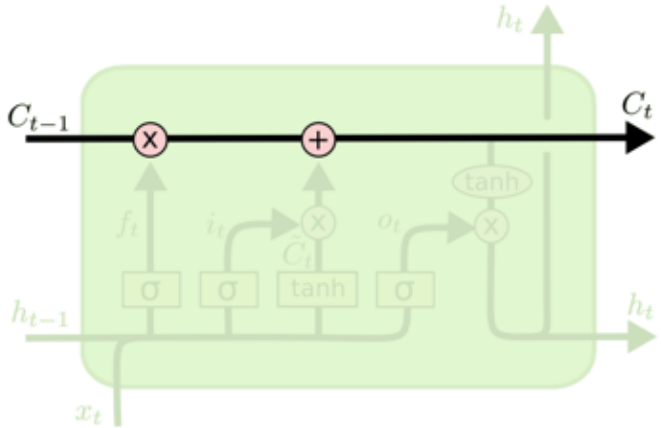
LSTM Network Architecture

- Here's the architecture of a typical LSTM network



Cell State

- Maintains a vector C_t that is the same dimensionality as the hidden state, h_t
- Information can be added or deleted from this state vector via the forget and input gates

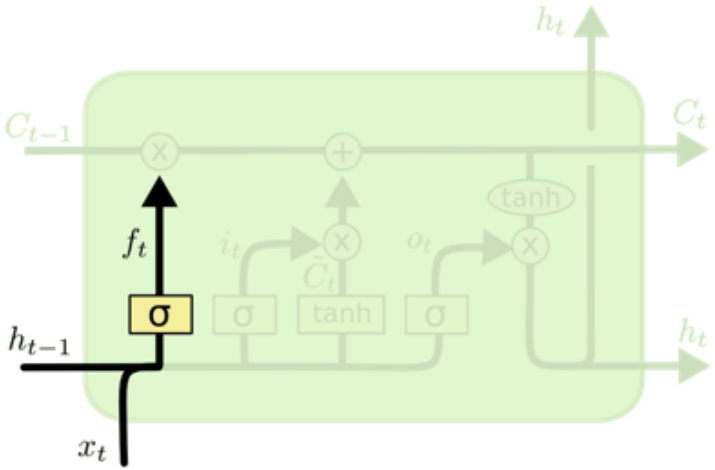


Cell State Example

- Want to remember the person & number of a subject noun so that it can be checked to agree with the person and number of verb when it is eventually encountered.
- Forget gate will remove existing information of a prior subject when a new one is encountered.
- Input gate “adds” in the information for the new subject.

Forget Gate

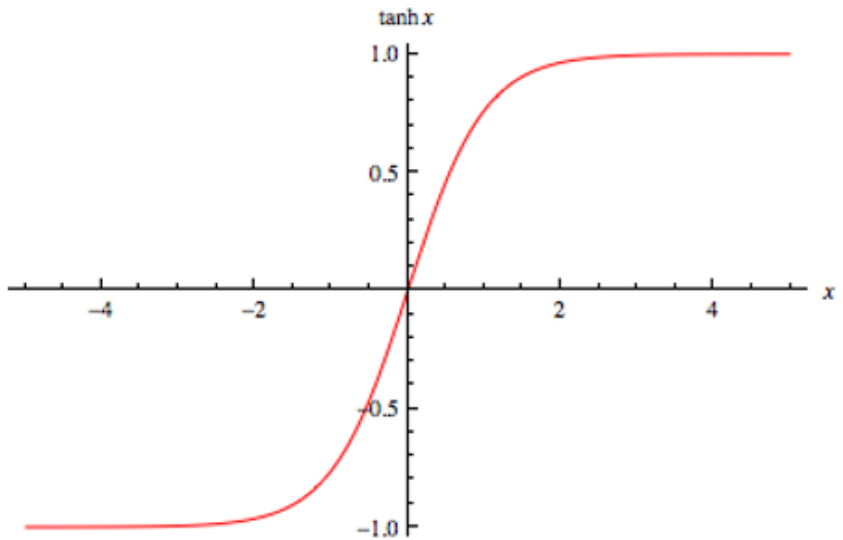
- Forget gate computes a 0-1 value using a logistic sigmoid output function from the input, x_t. and the current hidden state, h_t
- Multiplicatively combined with cell state, "forgetting" information where the gate outputs something close to 0



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

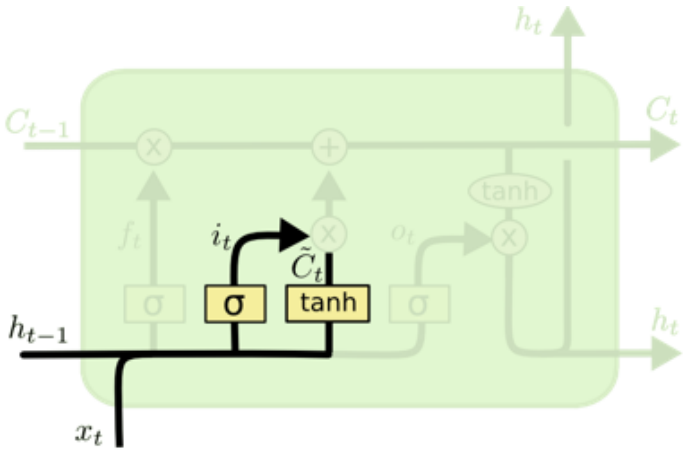
Hyperbolic Tangent Units

- Tanh can be used as an alternative nonlinear function to the sigmoid logistic (0-1) output function
- Used to produce thresholded output between –1 and 1



Input Gate

- First, determine which entries in the cell state to update by computing 0-1 sigmoid putput
- Then determine what amount to add or substract from these entries by computing a tanh output (valued -1 to 1) function of the input and hidden state

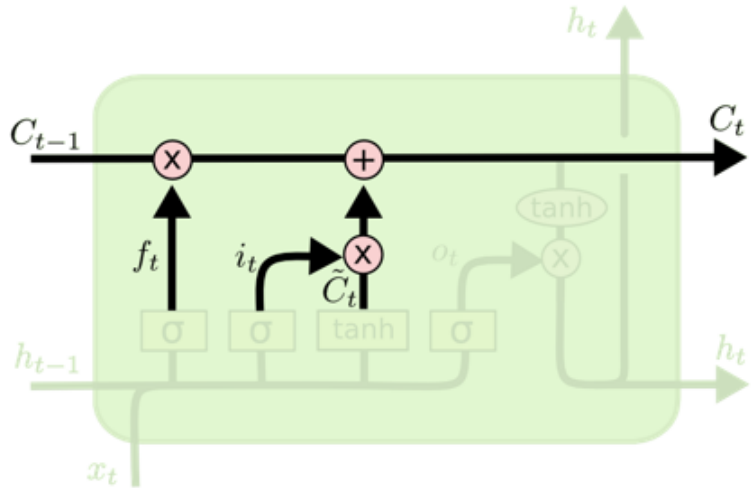


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Updating the Cell State

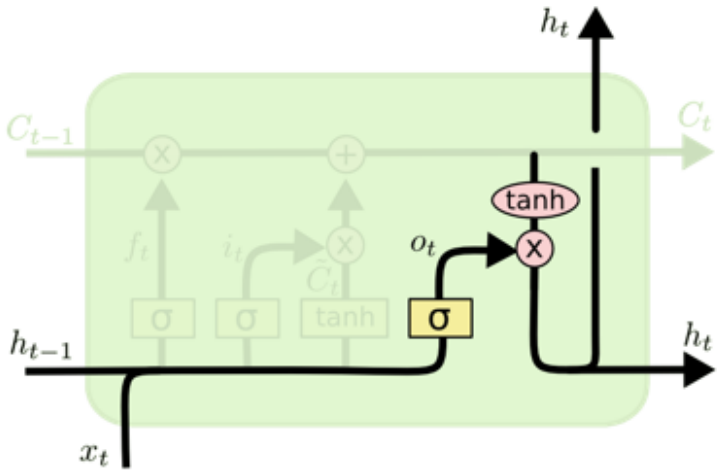
- Cell state is updated by using component-wise vector multiply to "forget" and vector addition to "input" new information.



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

Output Gate

- Hidden state is updated based on a "filtered" version of the cell state, scaled to –1 to 1 using tanh
- Output gate computes a sigmoid function of the input and current hidden state to determine which elements of the cell state to "output"

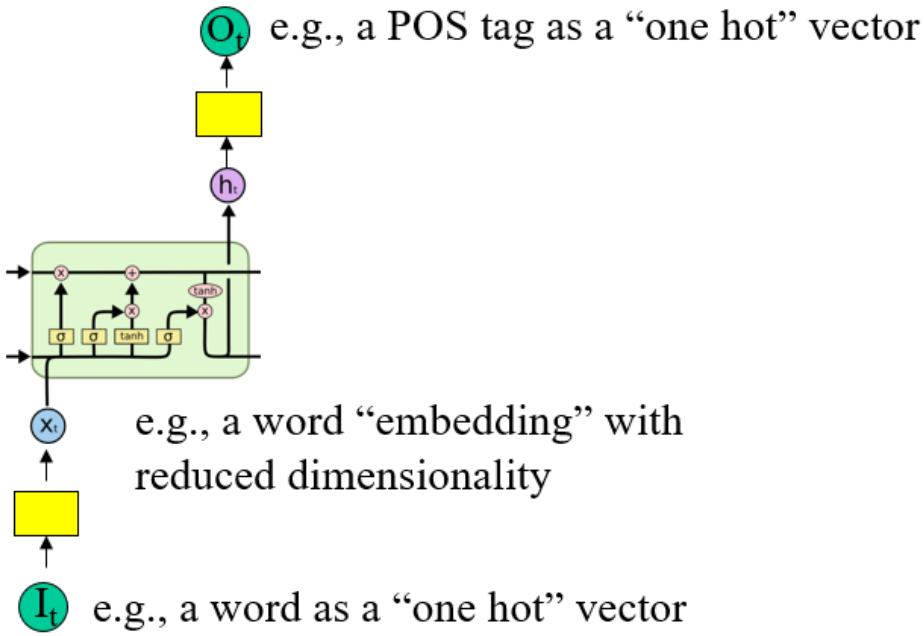


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

Overall Network Architecgture

- Single or multilayer networks can compute LSTM inputs from problem inputs and problem outputs from LSTM outputs.



LSTM Training

- Trainable with backpropagation derivatives such as:
  - Stochastic gradient descent (randomize order of examples in each epoch) with momentum (bias weight changes to continue in same direction as last update)
  - ADAM optimizer (Kingma and Ma, 2015)
- Each cell has many parameters ($\boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_C, \boldsymbol{W}_o$)
  - Generally requires lots of training data
  - Requires lots of computation time that exploits GPU clusters

General Problems Solved with LSTMs

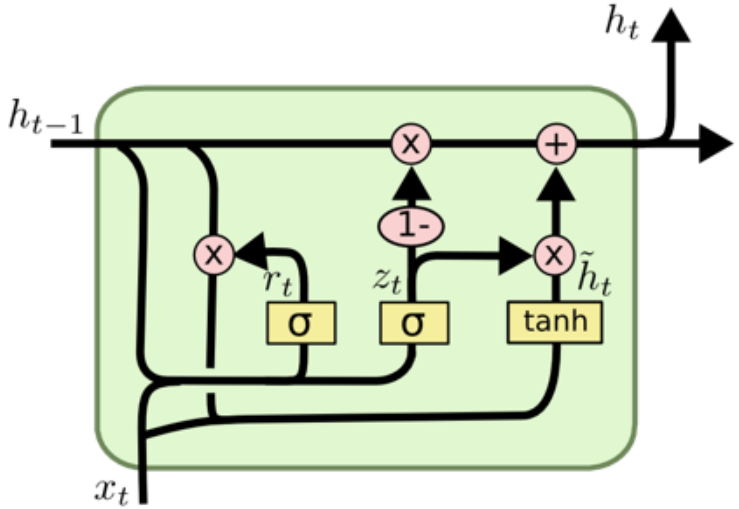
- Sequence labeling
  - Train with supervised output at each time step computed using a single or multilayer network that maps the hidden state (h_t) to an output vector (O_t)
- Language modeling
  - Train to predict the next input ($O_t = I_{t+1}$)
- Sequence (e.g. text) classification
  - Train a single or multilayer network that maps the final hidden state (h_n) to an output vector (O).

Successful Applications of LSTMs

- Speech recognition: Language and acoustic modeling
- Sequence labeling
  - POS Tagging
- Image captioning: CNN output vector to sequence
- Sequence to sequence
  - Machine translation
  - Video captioning (input sequence of CNN fame outputs)

Gated Recurrent Unit (GRU)

- Alternative RNN to LSTM that uses fewer gates (Cho, et al., 2014)
  - Combines forget and input gates into "update" gate
  - Eliminates cell state vector



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

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

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

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

GRU vs. LSTM

- GRU has significantly fewer parameters and trains faster
- Experimental results comparing the two are still inconclusive, many problems they perform the same, but each has problems on which they work better

Attention

- For many applications, it helps to add “attention” to RNNs
- Allows network to learn to attend to different parts of the input at different time steps, shifting its attention to focus on different aspects during its processing
- Used in image captioning to focus on different parts of an image when generating different parts of the output sentence
- In MT, allows focusing attention on different parts of the source sentence when generating different parts of the translation
- Will be covered in the following lectures!

Attention for Image Captioning (Xu, et al. 2015)

- Attention shifting based on the words being generated

Figure 2. Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. "soft" (top row) vs "hard" (bottom row) attention. (Note that both models generated the same captions in this example.)

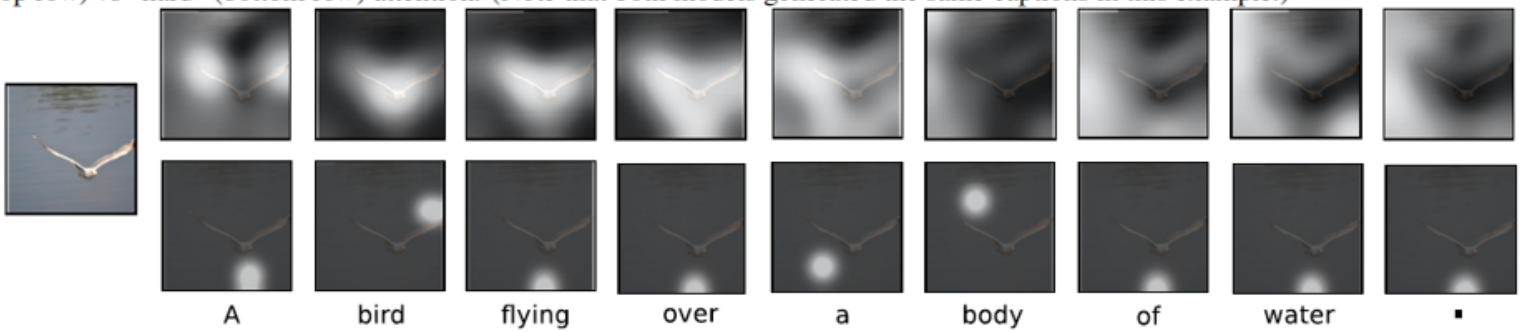
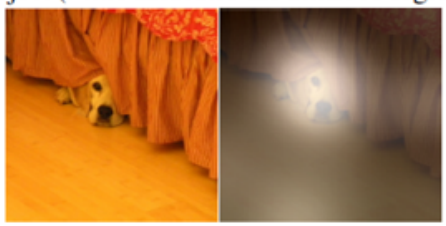


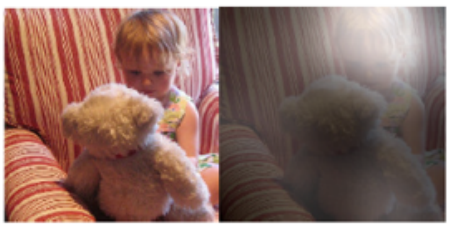
Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)

A woman is throwing a frisbee in a park.

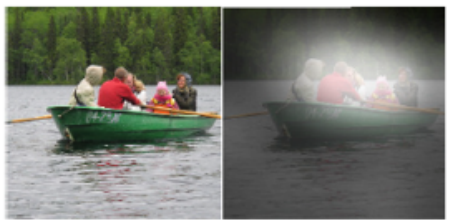


A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Conclusion

- By adding “gates” to an RNN, we can prevent the vanishing/exploding gradient problem.
- Trained LSTMs/GRUs can retain state information longer and handle long-distance dependencies
- Recent impressive results on a range of challenging NLP problems (with attention)