

Long-Short Term Memory and Other Gated RNNs

Sargur Srihari

srihari@buffalo.edu

Topics in Sequence Modeling

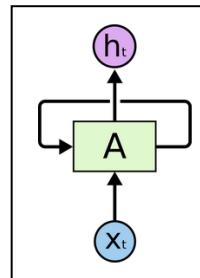
- Recurrent Neural Networks
 - 1. Unfolding Computational Graphs
 - 2. Recurrent Neural Networks
 - 3. Bidirectional RNNs
 - 4. Encoder-Decoder Sequence-to-Sequence Architectures
 - 5. Deep Recurrent Networks
 - 6. Recursive Neural Networks
 - 7. The Challenge of Long-Term Dependencies
 - 8. Echo-State Networks
 - 9. Leaky Units and Other Strategies for Multiple Time Scales
 - 10. **LSTM and Other Gated RNNs**
 - 11. Optimization for Long-Term Dependencies
 - 12. Explicit Memory

Topics in LSTM and Other Gated RNNs

- From RNN to LSTM
- Problem of Long-Term Dependency
- BPTT and Vanishing Gradients
- Key intuition of LSTM as “state”
- LSTM Cell (three gates and two activations)
- LSTM usage example
- Equations for: forget gate, state update, output gate
- Step-by-Step LSTM Walk-through
- Gated RNNs

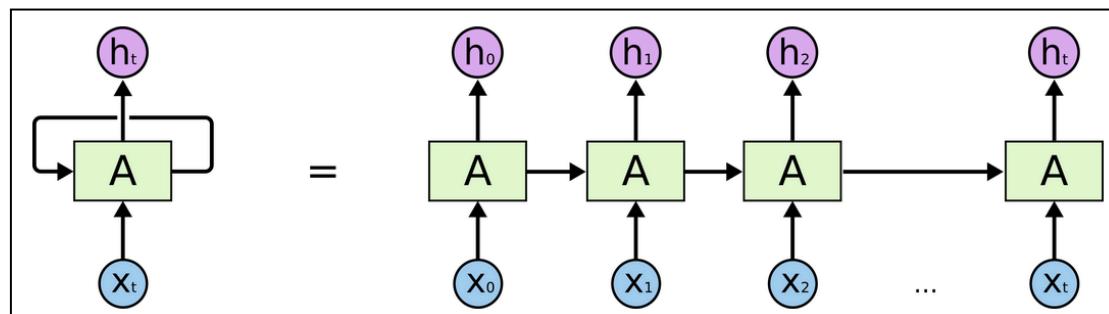
Recurrent Neural Networks

- RNNs have loops



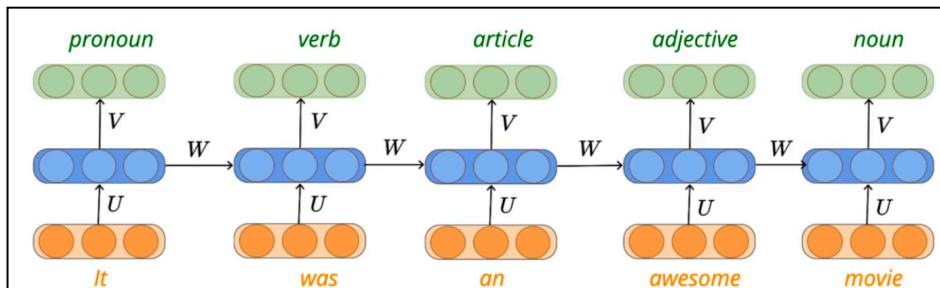
A chunk of neural network A looks at some input x_t and outputs a value h_t
 A loop allows information to be passed from one step of the network to the next

- An unrolled RNN



Chain-like structure reveals that RNNs are intimately related to sequences and lists

- Application to Part of Speech (POS) Tagging



Different Types of RNNs

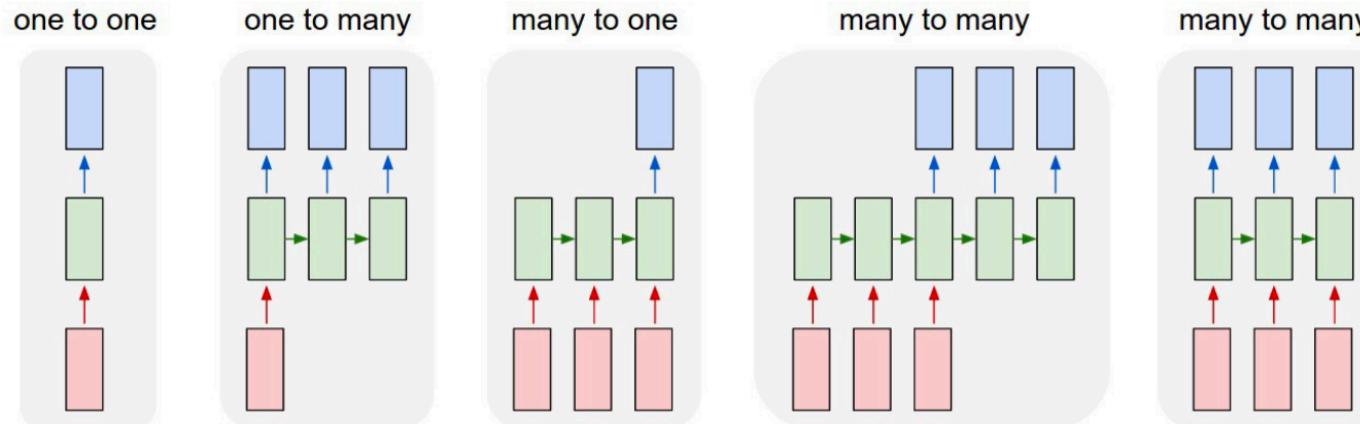


Image captioning takes image and outputs a sentence

Sentiment Analysis sentence classified as positive or negative

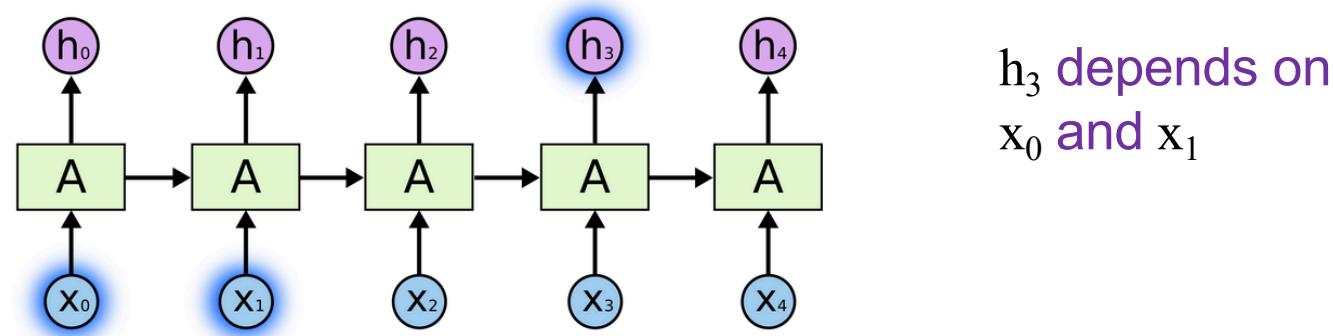
Machine Translation

Synced sequence input and output (e.g. video classification label each frame of video)

RNNs mainly used for
Sequence Classification
Sentiment & Video Classification
Sequence Labeling
POS & NE Tagging
Sequence Generation
MT & Transliteration

Prediction with only recent previous information

- RNNs connect previous information to the present task
 - Previous video frames may help understand present frame
- Sometimes we need only look at recent previous information to predict
 - To predict the last word of “The clouds are in the *sky*” we don’t need any further context. It is obvious that the word is “sky”

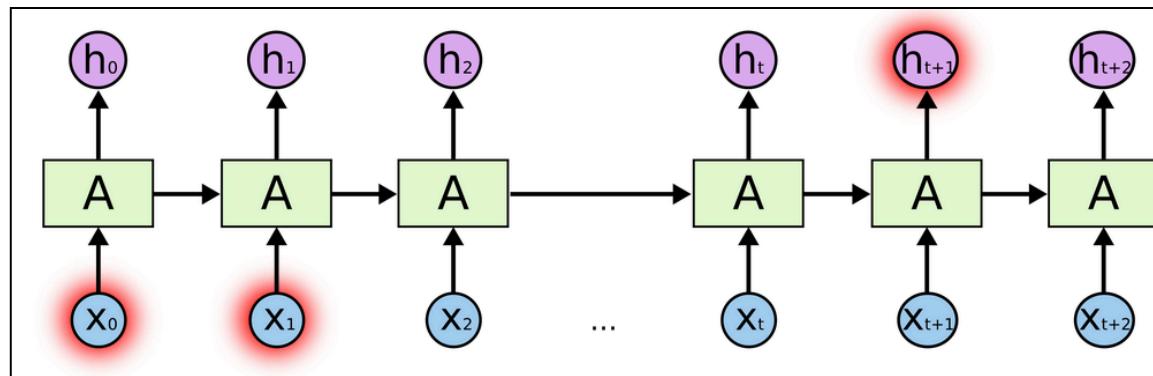


Long-Term Dependency

- RNNs work upon the fact that the result of an information is dependent on its previous state or previous n time steps.
- They have difficulty in learning long range dependencies.
- **Example sentence:** The man who ate my pizza has purple hair
 - **Note:** purple hair **is for the** man **and not** the pizza.
 - So this is a long dependency

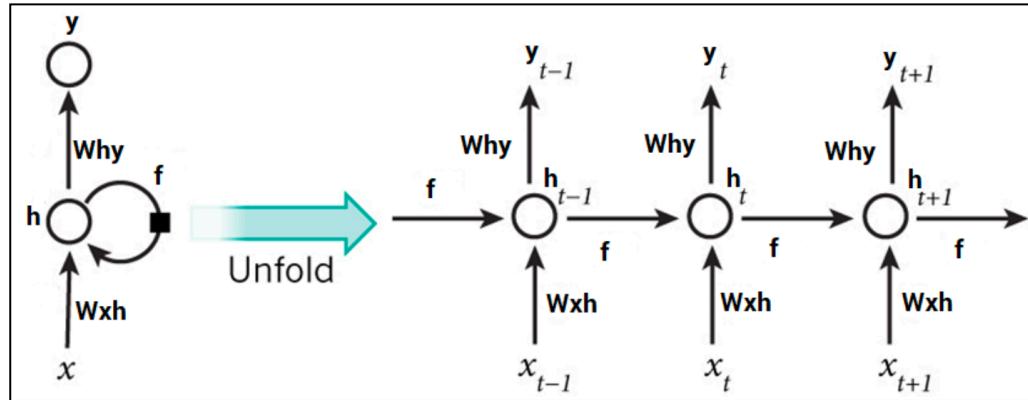
Long-term dependency and Backprop

- There are cases where we need even more context
 - To predict last word in I grew up in France.....I speak *French*
 - Using only recent information suggests that the last word is the name of a language. But more distant past indicates that it is *French*
- Gap between relevant information and where it is needed is large



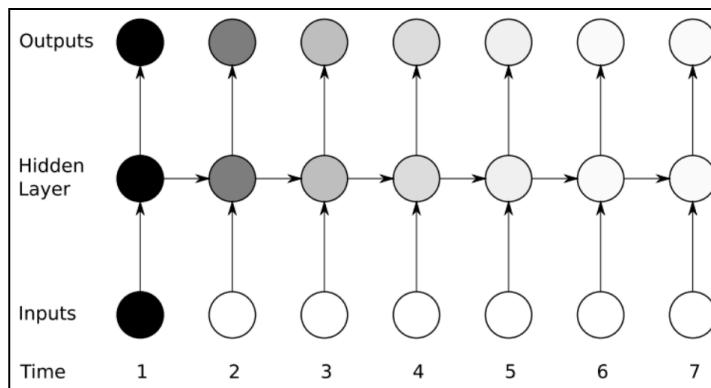
- Learning to store information over extended time intervals via RNN backpropagation takes a very long time due to decaying error backflow (discussed next)

Backpropagation Through Time (BPTT)



If y_t is predicted and \bar{y}_t is training value
the cross entropy loss is
 $E_t(\bar{y}_t, y_t) = -\bar{y}_t \log(y_t)$

Total error is the sum
 $E(\bar{y}, y) = -\sum \bar{y}_t \log(y_t)$



To backpropagate error, apply the chain rule.
The error after the third time step wrt the first:
 $\partial E / \partial W = \partial E / \partial y_3 * \partial y_3 / \partial h_3 * \partial h_3 / \partial y_2 * \partial y_2 / \partial h_1 \dots$
and there is a long dependency.

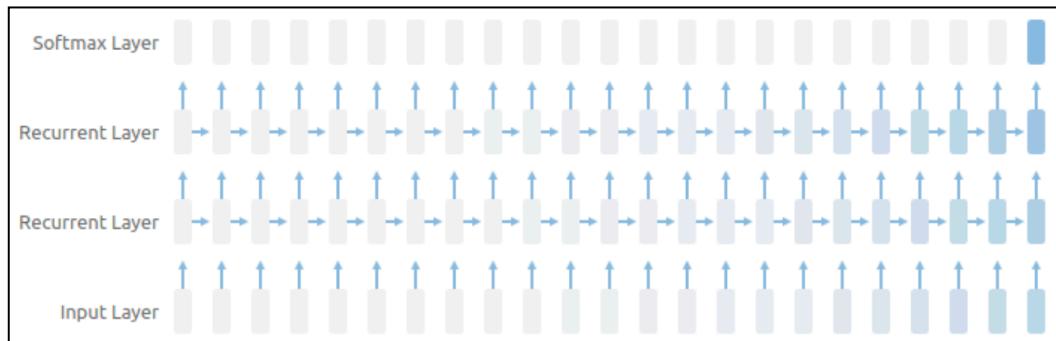
Vanishing gradient problem

If a single gradient approached 0, all gradients rush to zero exponentially fast due to the multiplication.

Such states would no longer help the network to learn anything.

Vanishing and Exploding Gradients

- Vanishing gradient problem

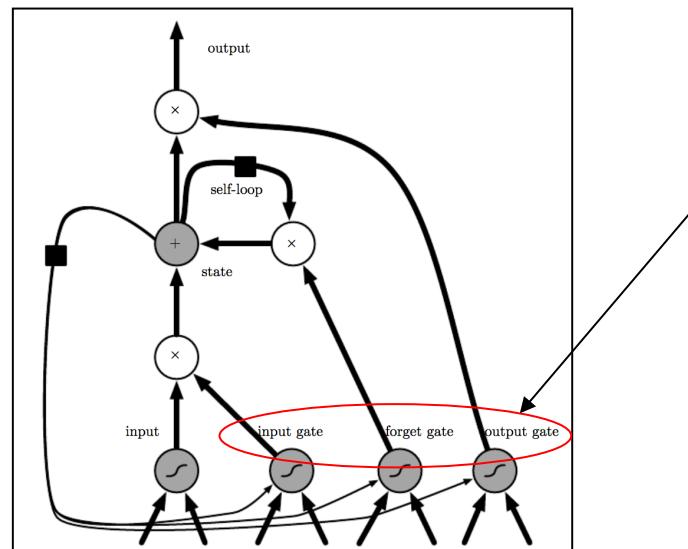


Contribution from the earlier steps becomes insignificant in the gradient

- Exploding gradient problem:
 - Gradients become very very large due to a single or multiple gradient values becoming very high.
- Vanishing gradient is more concerning
 - Exploding gradient easily solved by clipping the gradients at a predefined threshold value.
- Solution: gated RNNs
 - LSTM, GRU (Gated Recurrent Units), a variant of LSTM

Key intuition of LSTM is “State”

- A persistent module called the cell-state
- Note that “State” is a representation of past history
- It comprises a common thread through time
- Cells are connected recurrently to each other
 - Replacing hidden units of ordinary recurrent networks

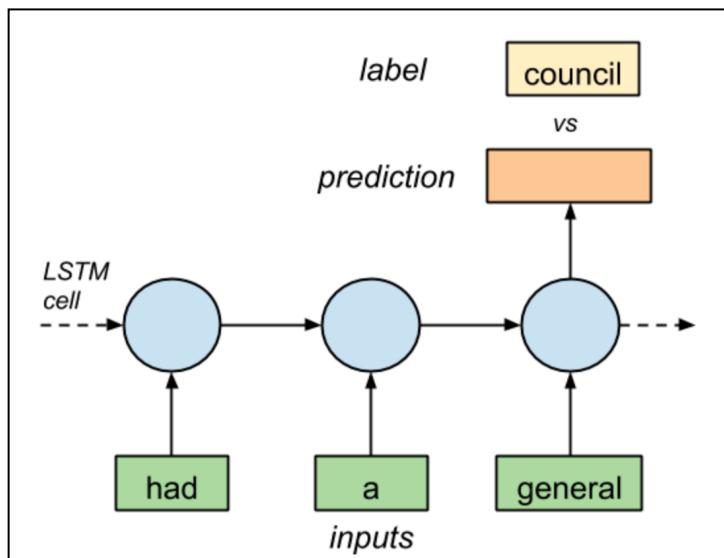


There are three sigmoid gates:
An input gate (i),
A forget gate (f)
An output gate (o)

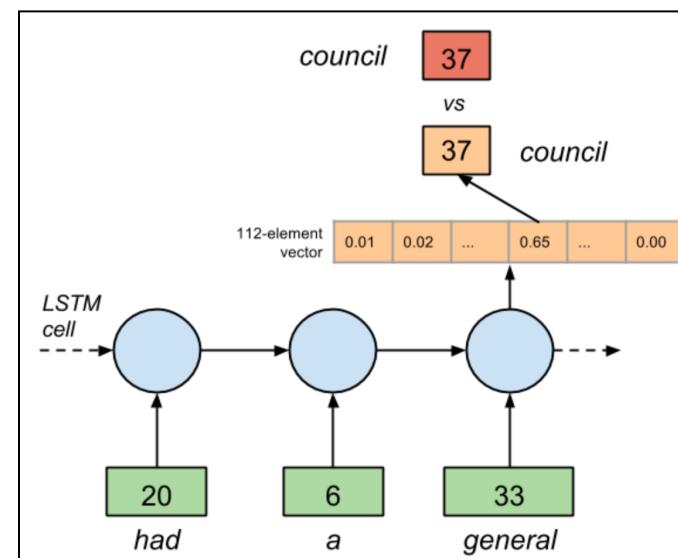
LSTM usage example

- “long ago , the mice had a general council to consider what measures they could take to outwit their common enemy , the cat.....” Aesop’s fables story with 112 words

Training



Prediction



LSTM in Tensorflow

Function for building the dictionary and reverse dictionary

```
def build_dataset(words):
    count = collections.Counter(words).most_common()
    dictionary = dict()
    for word, _ in count:
        dictionary[word] = len(dictionary)
    reverse_dictionary = dict(zip(dictionary.values(), dictionary.keys()))
    return dictionary, reverse_dictionary
```

Constants and Training parameters

```
vocab_size = len(dictionary)
n_input = 3

# number of units in RNN cell
n_hidden = 512

# RNN output node weights and biases
weights = {
    'out': tf.Variable(tf.random_normal([n_hidden, vocab_size]))
}
biases = {
    'out': tf.Variable(tf.random_normal([vocab_size]))
}
```

Symbols to vector of int as input

```
symbols_in_keys = [ [dictionary[ str(training_data[i])]] for i in
range(offset, offset+n_input) ]
```

Loss and Optimizer

```
pred = RNN(x, weights, biases)

# Loss and optimizer
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred,
labels=y))
optimizer =
tf.train.RMSPropOptimizer(learning_rate=learning_rate).minimize(cost)
```

Model with 512-unit LSTM

```
def RNN(x, weights, biases):

    # reshape to [1, n_input]
    x = tf.reshape(x, [-1, n_input])

    # Generate a n_input-element sequence of inputs
    # (eg. [had] [a] [general] -> [20] [6] [33])
    x = tf.split(x,n_input,1)

    # 1-layer LSTM with n_hidden units.
    rnn_cell = rnn.BasicLSTMCell(n_hidden)

    # generate prediction
    outputs, states = rnn.static_rnn(rnn_cell, x, dtype=tf.float32)

    # there are n_input outputs but
    # we only want the last output
    return tf.matmul(outputs[-1], weights['out']) + biases['out']
```

One hot vector as label

```
symbols_out_onehot = np.zeros([vocab_size], dtype=float)
symbols_out_onehot[dictionary[str(training_data[offset+n_input])]]=
1.0
```

Training set optimization

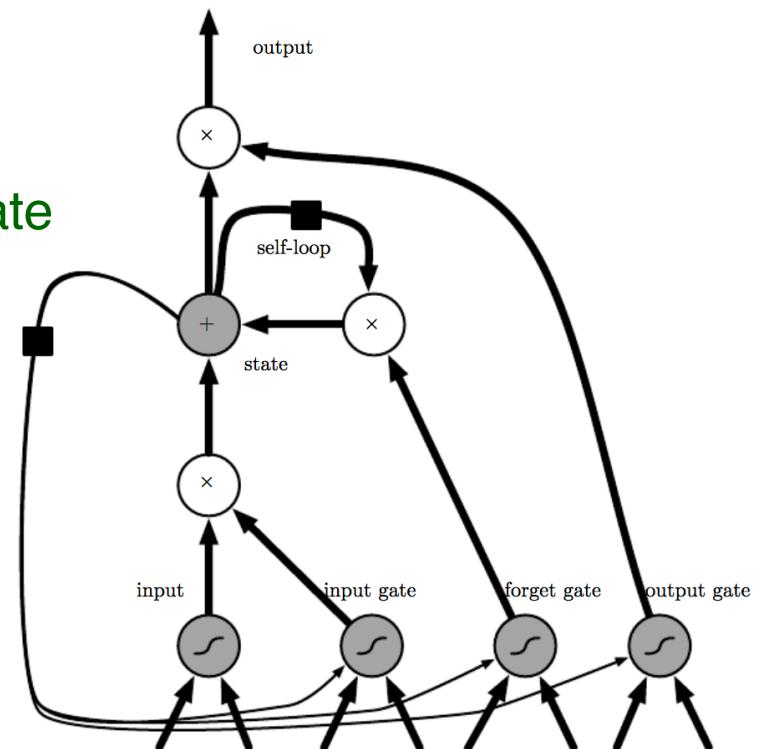
```
_, acc, loss, onehot_pred = session.run([optimizer, accuracy, cost,
pred], feed_dict={x: symbols_in_keys, y: symbols_out_onehot})
```

Sample prediction

```
...
Iter= 49000, Average Loss= 0.528684, Average Accuracy= 88.50%
['could', 'easily', 'retire'] - [while] vs [while]
Iter= 50000, Average Loss= 0.415811, Average Accuracy= 91.20%
['this', 'means', 'we'] - [should] vs [should]
```

LSTM Recurrent Network Cell

- Input feature computed with regular artificial neuron unit
 - Its value can be accumulated into the state
 - If the sigmoidal input gate allows it
- State unit has linear self-loop
 - Weight controlled by forget gate
- Output of cell can be shut off by output gate
- All units: sigmoid nonlinearity
 - Input gate can be any squashing
- State unit can also be used as an extra input to gating units
- Black square indicates delay of single time step

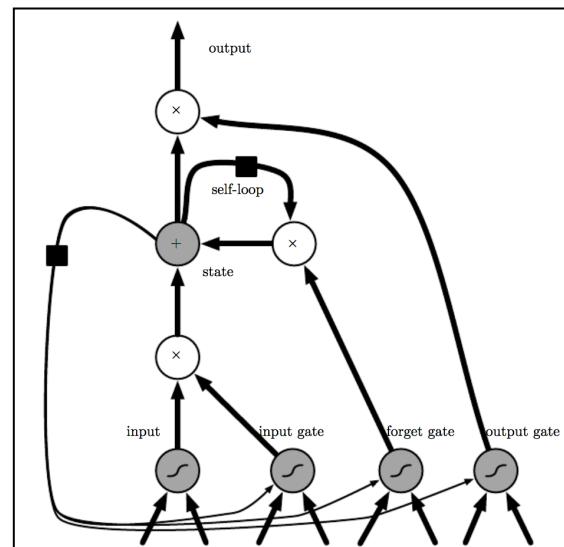


Core contribution of LSTM

- Clever idea of introducing self-loops to produce paths where the gradient can flow for long durations
- A crucial addition: make weight on this self-loop conditioned on the context, rather than fixed
 - By making weight of this self-loop gated (controlled by another hidden unit), time-scale can be changed dynamically
 - Even for an LSTM with fixed parameters, time scale of integration can change based on the input sequence
 - Because time constants are output by the model itself

State perturbed by linear operations

- Cell-state perturbed only by a few linear operations at each time step



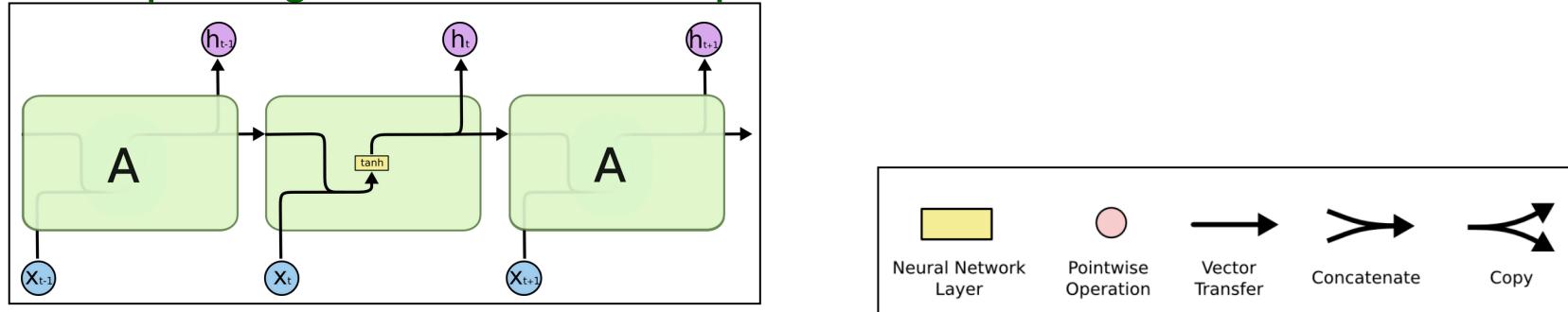
- Since cell state connection to previous cell states is interrupted only by the linear operations of multiplication and addition, LSTMs can remember short-term memories (*i.e.* activity belonging to the same “episode”) for a very long time

LSTM success

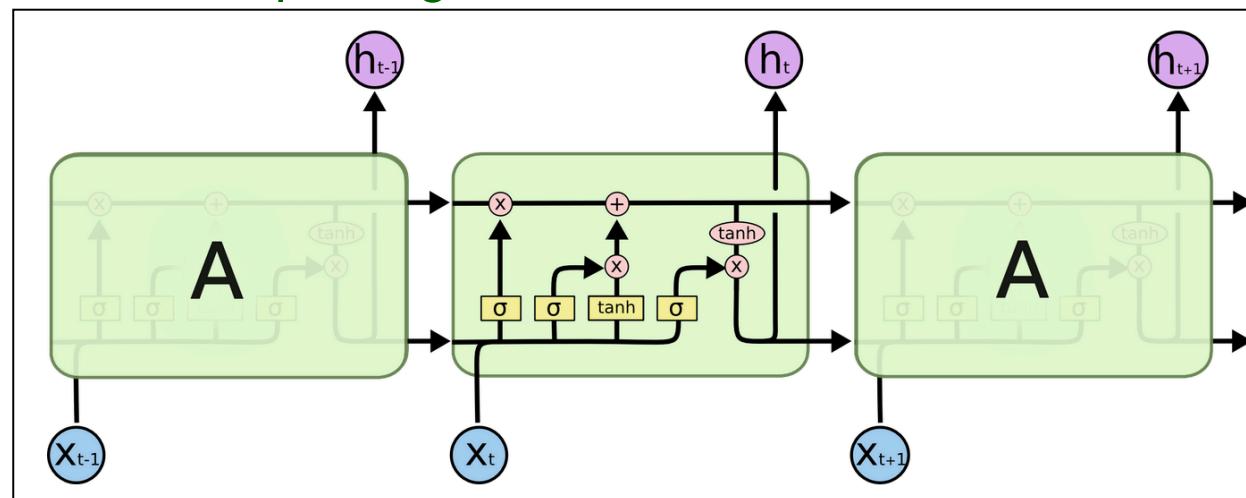
- LSTM found extremely successful in:
 - Unconstrained handwriting recognition
 - Speech recognition
 - Handwriting generation, Machine Translation
 - Image Captioning, Parsing

RNN vs LSTM cell

- RNNs have the form of a repeating chain structure
 - The repeating module has a simple structure such as tanh

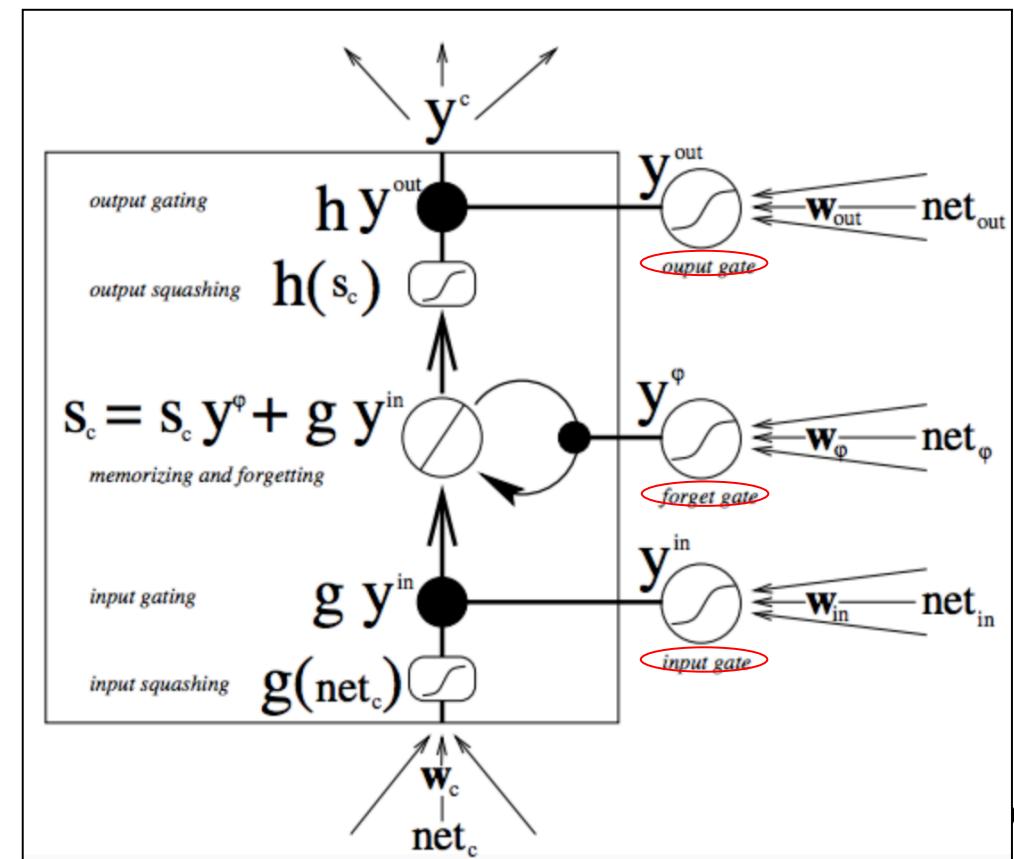
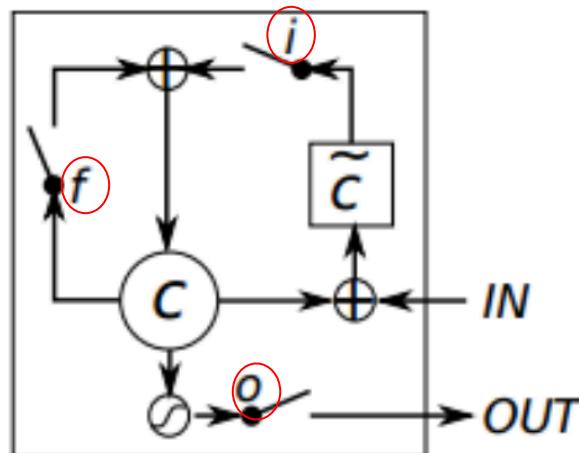


- LSTMs also have a chain structure
 - but the repeating module has a different structure



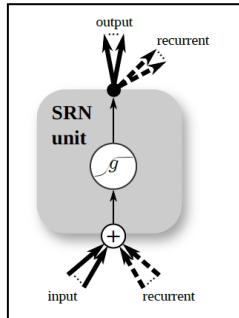
LSTM cell

- Like RNNs , LSTMs also have a chain like structure
 - But repeating module has a slightly different structure
- Instead of a single layer, multiple layers interact
 - They have three sigmoid gates:
 - An input gate (i),
 - A forget gate (f)
 - An output gate (o)



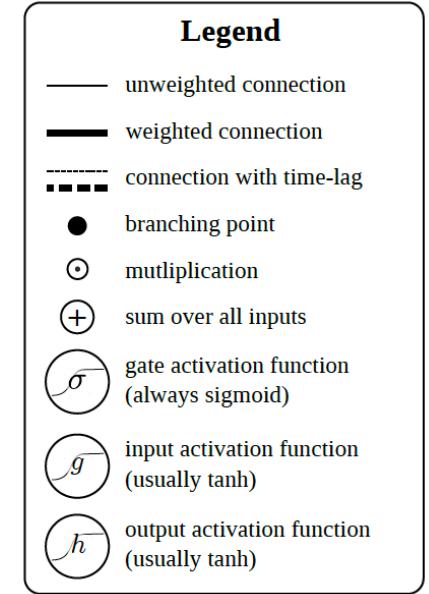
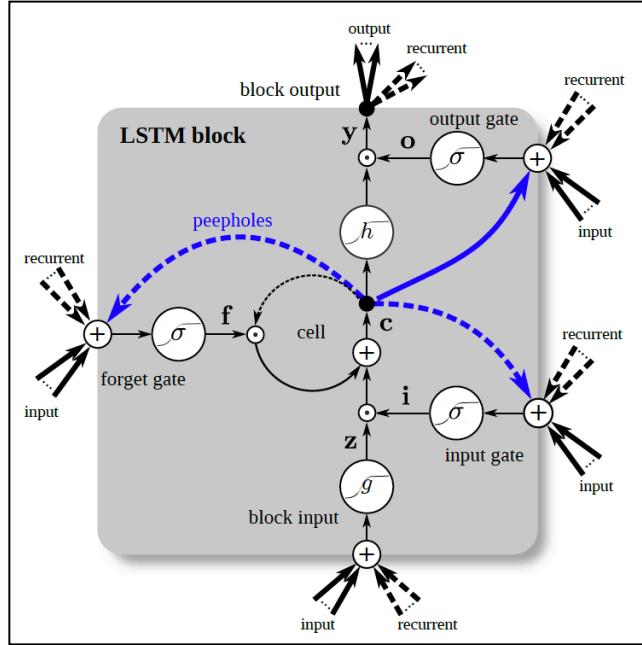
LSTM Cell equations

Simple RNN unit



Weights for an LSTM layer:

- Input weights: $\mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: $\mathbf{R}_z, \mathbf{R}_i, \mathbf{R}_f, \mathbf{R}_o \in \mathbb{R}^{N \times N}$
- Peephole weights: $\mathbf{p}_i, \mathbf{p}_f, \mathbf{p}_o \in \mathbb{R}^N$
- Bias weights: $\mathbf{b}_z, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^N$



Vector formulas for LSTM forward pass:

$\bar{\mathbf{z}}^t = \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z$	<i>block input</i>
$\mathbf{z}^t = g(\bar{\mathbf{z}}^t)$	
$\bar{\mathbf{i}}^t = \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i$	<i>input gate</i>
$\mathbf{i}^t = \sigma(\bar{\mathbf{i}}^t)$	
$\bar{\mathbf{f}}^t = \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f$	<i>forget gate</i>
$\mathbf{f}^t = \sigma(\bar{\mathbf{f}}^t)$	
$\mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t$	<i>cell</i>
$\bar{\mathbf{o}}^t = \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o$	
$\mathbf{o}^t = \sigma(\bar{\mathbf{o}}^t)$	<i>output gate</i>
$\mathbf{y}^t = h(\mathbf{c}^t) \odot \mathbf{o}^t$	

BPTT deltas inside the LSTM block:

$$\begin{aligned}
 \delta \mathbf{y}^t &= \Delta^t + \mathbf{R}_z^T \delta \mathbf{z}^{t+1} + \mathbf{R}_i^T \delta \mathbf{i}^{t+1} + \mathbf{R}_f^T \delta \mathbf{f}^{t+1} + \mathbf{R}_o^T \delta \mathbf{o}^{t+1} \\
 \delta \bar{\mathbf{o}}^t &= \delta \mathbf{y}^t \odot h(\mathbf{c}^t) \odot \sigma'(\bar{\mathbf{o}}^t) \\
 \delta \mathbf{c}^t &= \delta \mathbf{y}^t \odot \mathbf{o}^t \odot h'(\mathbf{c}^t) + \mathbf{p}_o \odot \delta \bar{\mathbf{o}}^t + \mathbf{p}_i \odot \delta \bar{\mathbf{i}}^{t+1} \\
 &\quad + \mathbf{p}_f \odot \delta \bar{\mathbf{f}}^{t+1} + \delta \mathbf{c}^{t+1} \odot \mathbf{f}^{t+1} \\
 \delta \bar{\mathbf{f}}^t &= \delta \mathbf{c}^t \odot \mathbf{c}^{t-1} \odot \sigma'(\bar{\mathbf{f}}^t) \\
 \delta \bar{\mathbf{i}}^t &= \delta \mathbf{c}^t \odot \mathbf{z}^t \odot \sigma'(\bar{\mathbf{i}}^t) \\
 \delta \bar{\mathbf{z}}^t &= \delta \mathbf{c}^t \odot \mathbf{i}^t \odot g'(\bar{\mathbf{z}}^t)
 \end{aligned}$$

Sigmoid used for gate activation, tanh used as input and output activation, point-wise multiplication of vectors is \odot

Accumulating Information over Longer Duration

- Leaky Units Allow the network to accumulate information
 - Such as evidence for a particular feature or category
 - Over a long duration
- However, once the information has been used, it might be useful to forget the old state
 - E.g., if a sequence is made of sub-sequences and we want a leaky unit to accumulate evidence inside each sub-sequence, we need a mechanism to forget the old state by setting it to zero
- Gated RNN:
 - Instead of manually deciding when to clear the state, we want the neural network to learn to decide when to do it

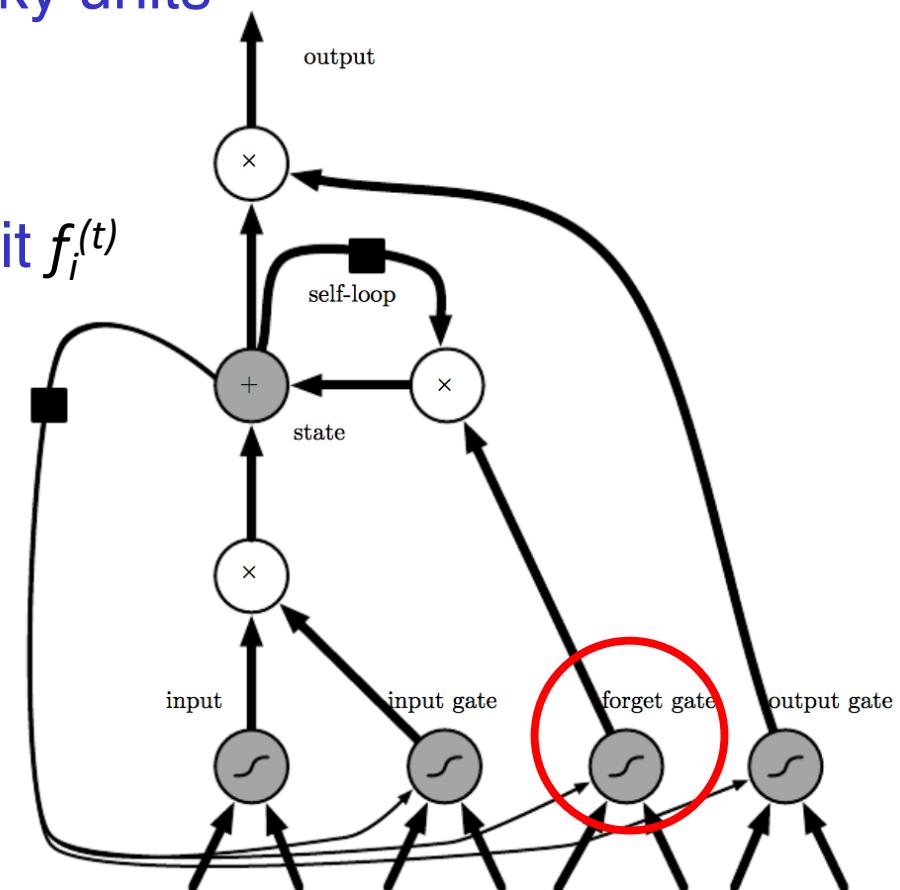
Description of the LSTM cell

- Forward propagation equations for the LSTM cell are given in the next slide
 - In the case of a shallow recurrent network architecture
 - Deeper architectures have also been successively used
- Instead of a unit that simply applies an elementwise nonlinearity to the affine transformation of inputs and recurrent units LSTM recurrent units have *LSTM cells* that have an internal recurrence (a self-loop) in addition to the outer recurrence of the RNN
- Each cell has the same inputs and outputs as an ordinary RNN
 - But has more parameters and a system of gating units that controls the flow of information

Equation for LSTM forget gate

- The most important component is the state unit $s_i^{(t)}$ that has a linear self-loop similar to the leaky units
- However the self-loop weight
(or the associated time constant)
is controlled by a forget gate unit $f_i^{(t)}$
(for time step t and cell i)
that sets this weight to a
value between 0 and 1 via a
sigmoid unit

$$f_i^{(t)} = \sigma \left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)} \right)$$



where $x^{(t)}$ is the current input vector and $h^{(t)}$ is the current hidden layer vector containing the outputs of all the LSTM cells, and b^f , u^f and w^f are respectively biases, input weights and recurrent weights for forget gates

Equation for LSTM internal state update

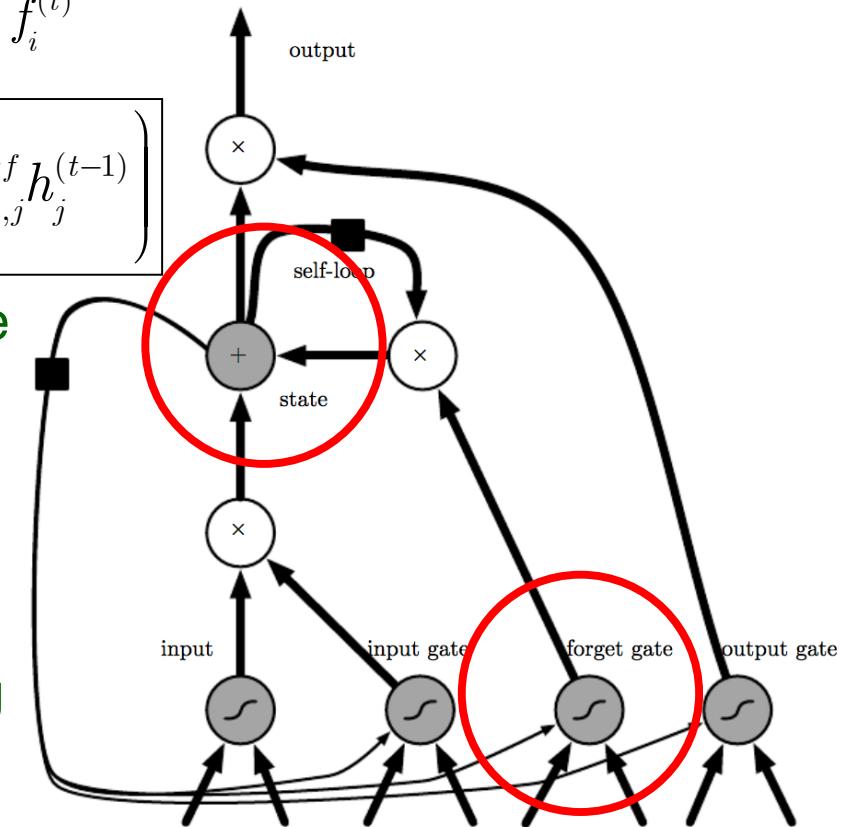
- The LSTM cell internal state is updated as follows

- But with conditional self-loop weight $f_i^{(t)}$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)} \right)$$

- where b , u and w respectively denote the biases, input weights and recurrent weights into the LSTM cell
- External input gate unit $g_i^{(t)}$ is
 - computed similar to forget gate
 - With a sigmoid unit to obtain a gating value between 0 and 1 but with its own parameters

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right)$$



Equation for output of LSTM cell

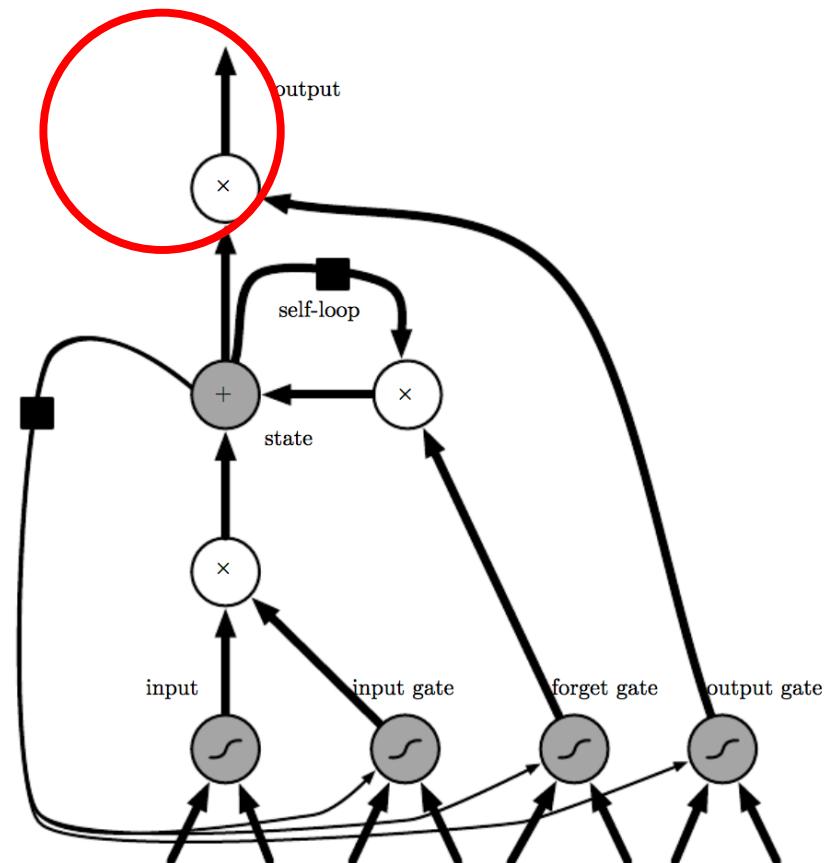
- Output $h_i^{(t)}$ of the LSTM cell can be shut off via the output gate $q_i^{(t)}$ which also uses a sigmoid unit for gating

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)}$$

$$q_i^{(t)} = \sigma(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)})$$

b^0 , U^0 and W^0 are biases, input weights and recurrent weights

- Among the variants one can choose to use the cell state $s_i^{(t)}$ as an extra input (with its weight) into the three gates of the i -th unit. This would require three additional parameters



Power of LSTMs

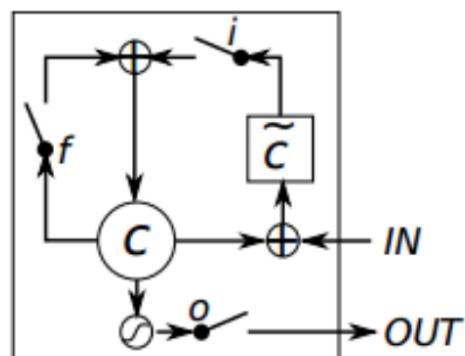
- LSTM networks have been shown to learn long-term dependencies more easily than simple recurrent architectures
 - First on artificial data sets
 - Then on challenging sequential tasks
- Variants and alternatives to LSTM have been studied and used and are discussed next

Other gated RNNs

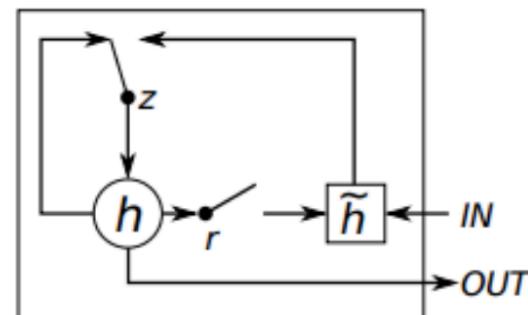
- What pieces of the LSTM architecture are actually necessary?
- What other successful architectures could be designed that allow the network to dynamically control the time scale and forgetting behavior of different units?
- Some answers to these questions are given with the recent work on gated RNNs, whose units are also known as gated recurrent units or GRUs
- Main difference with LSTM
 - Single gating unit simultaneously controls the forgetting factor and decision to update state unit

GRUs

- GRUs have simpler structure and are easier to train.
- Their success is primarily due to the gating network signals that control how the present input and previous memory are used, to update the current activation and produce the current state.
- These gates have their own sets of weights that are adaptively updated in the learning phase.
- We have just two gates here, the reset and the update gate



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

Idea of Gated RNNs

- Like leaky units, gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode
- Leaky units do this with connection weights that are manually chosen or were parameters α
generalizing the concept of discrete skipped connections
- Gated RNNs generalize this to connection weights that may change with each time step

Update equations for Gated RNNs

- Update equations are

$$h_i^{(t)} = u_i^{(t)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)} \right)$$

- Where u stands for the update gate and r for reset gate. Their value is defined as usual:

$$u_i^{(t)} = \sigma \left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)} \right)$$

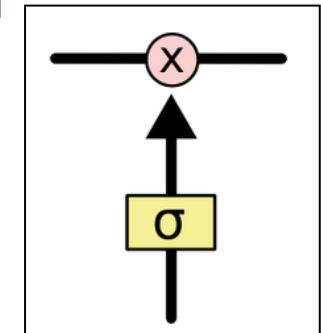
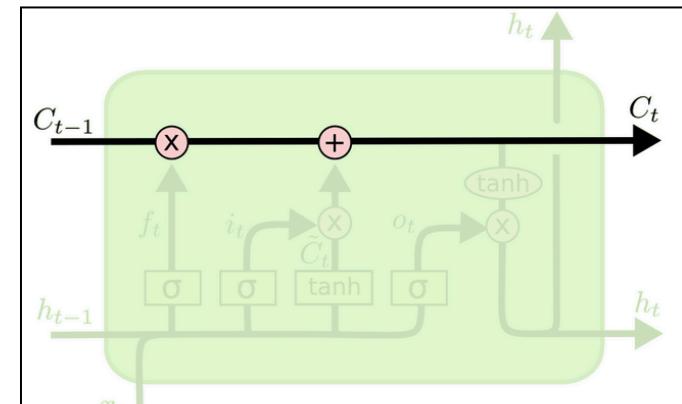
and

$$r_i^{(t)} = \sigma \left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)} \right)$$

- Reset and update gates can individually ignore parts of the state vector
 - Update gates act conditional leaky integrators that can linearly gate any dimension thus choosing to copy it (at one extreme of sigmoid) or completely ignore it (at the other extreme) by replacing it by the new target state value (towards which the leaky integrator converges)
 - Reset gates control which parts of the state get used to compute next target state, introducing an additional nonlinear effect in the relationship between past state and future state

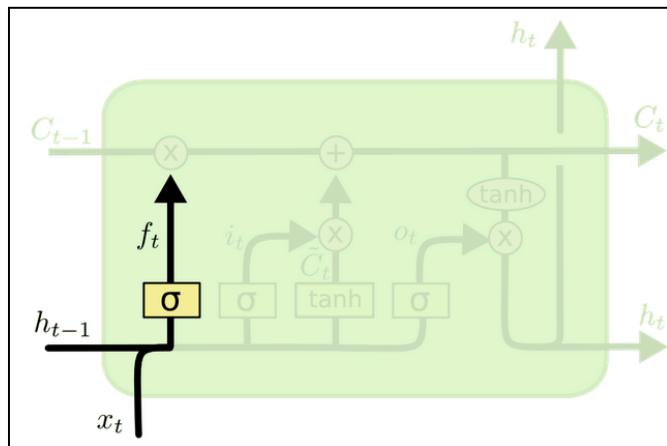
Core idea behind LSTM

- The key to LSTM is the cell state, C_t , the horizontal line running through the top of the diagram
- Like a conveyor belt
 - Runs through entire chain with minor interactions
 - LSTM does have the ability to remove/add information to cell state regulated by structures called gates
- Gates are an optional way to let information through
- Consist of a sigmoid and a multiplication operation
- Sigmoid outputs a value between 0 and 1
 - 0 means let nothing through
 - 1 means let everything through
- LSTM has three of these gates, to protect and control cell state



Step-by-step LSM walk through

- Example of language model: predict next word based on previous ones
 - Cell state may include the gender of the present subject
- First step: information to throw away from cell state



Called *forget gate layer*

It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each member of C_{t-1} for whether to forget

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

In language model

consider trying to predict the next word based on all previous ones

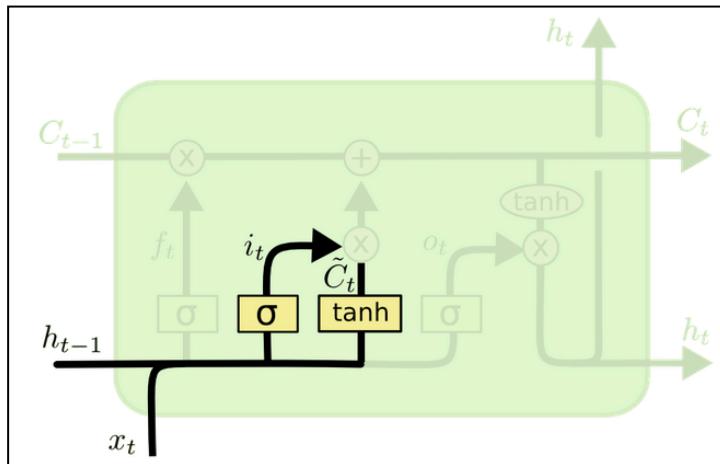
The cell state may include the gender of the present subject

so that the proper pronouns can be used

When we see a new subject we want to forget old subject

LSM walk through: Second step

- Next step is to decide as to what new information we're going to store in the cell state



In the Language model, we'd want to add the gender of the new subject to the cell state, to replace the old One we are forgetting

This has two parts:

first a sigmoid layer called *Input gate layer*: decides which values we will update
Next a tanh layer creates a vector of new candidate values \tilde{C}_t that could be added to the state.

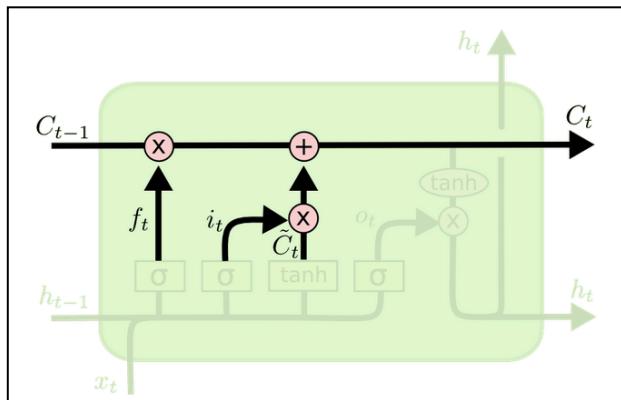
In the third step we will combine these two to create an update to the state

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

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

LSTM walk-through: Third Step

- It's now time to update old cell state C_{t-1} into new cell state C_t
 - The previous step decided what we need to do
 - We just need to do it



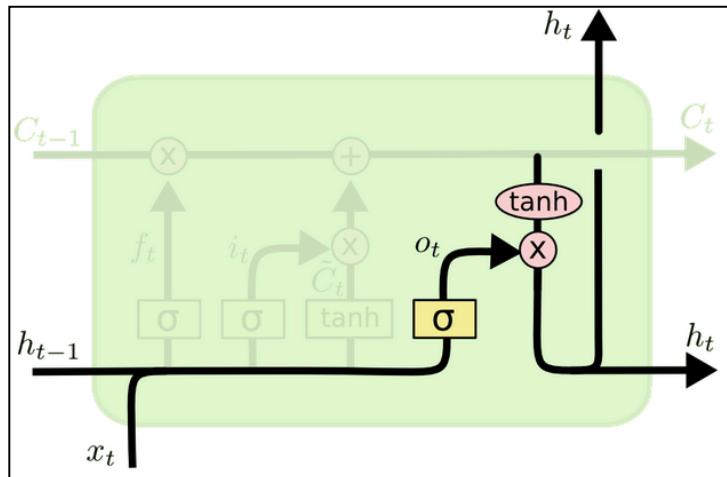
We multiply the old state by f_t , forgetting the things we decided to forget earlier.
 Then we add $i_t * \tilde{C}_t$
 This is the new candidate values, scaled by how much we decided to update each state value

In the Language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in previous steps

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

LSTM walk-through: Fourth Step

- Finally we decide what we are going to output



This output will be based on our cell state, but will be a filtered version. First we run a sigmoid layer which decides what parts of cell state we're going to output. Then we put the cell state through tanh (to push values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we output only the parts we decided to

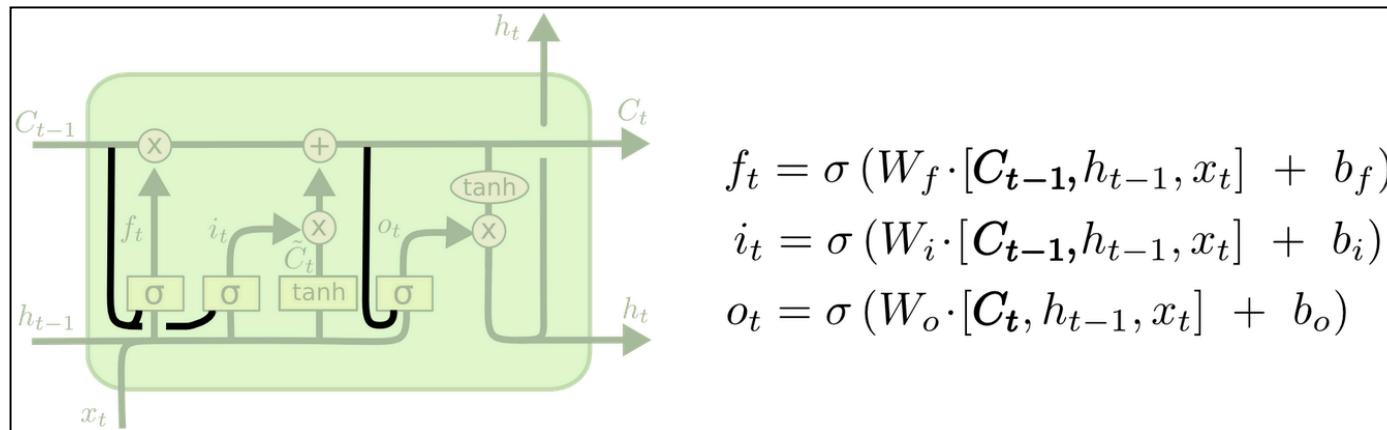
For the Language model, Since it just saw a subject it might want to output information relevant to a verb, in case that is what is coming next, e.g., it might output whether the subject is singular or plural so that we know what form a verb should be conjugated into if that's what follows next.

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

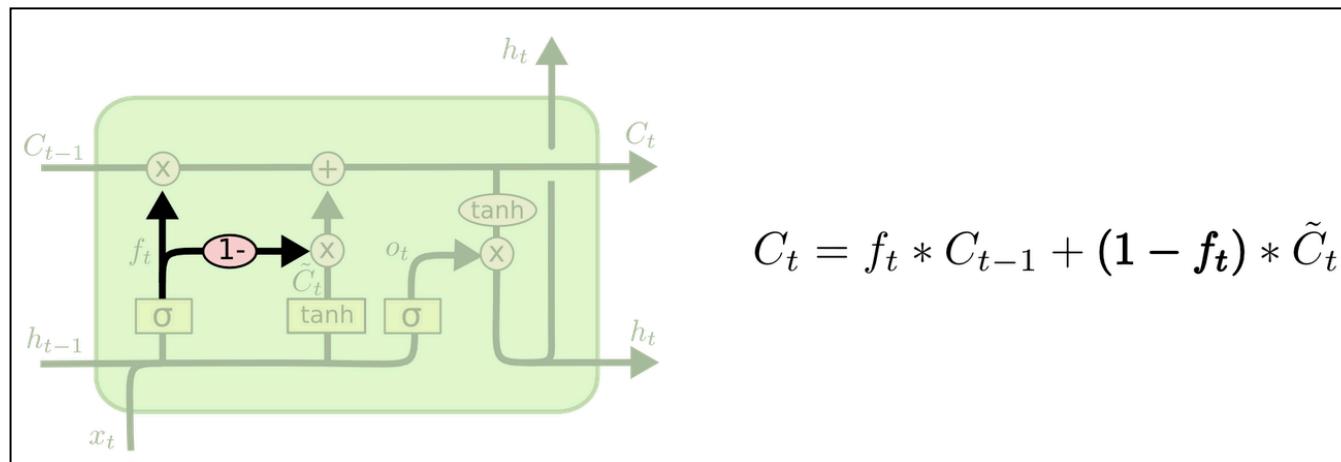
$$h_t = o_t * \tanh (C_t)$$

Variants of LSTM

- There are several minor variants of LSTM
 - LSTM with “peephole” connections



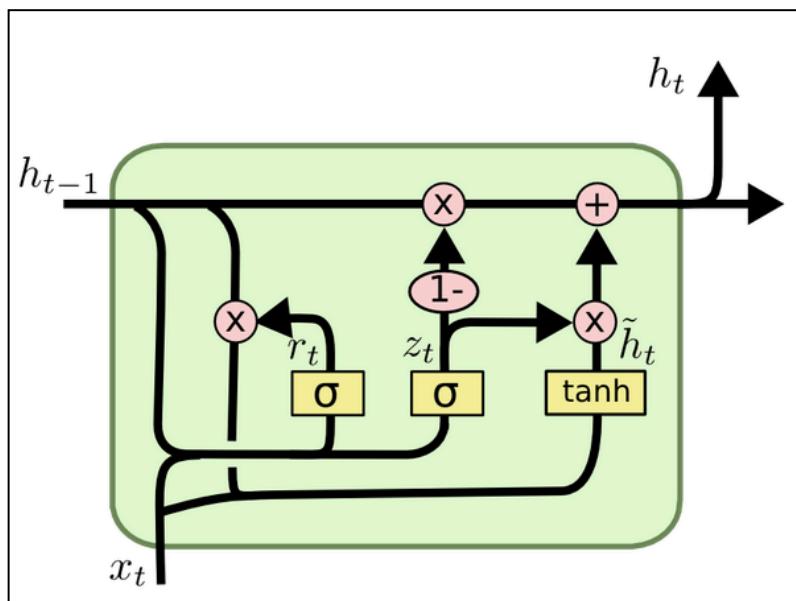
- Coupled forget and input gates



Gated Recurrent Unit (GRU)

- A dramatic variant of LSTM

- It combines the forget and input gates into a single update gate
- It also merges the cell state and hidden state, and makes some other changes
- The resulting model is simpler than LSTM models
- Has become increasingly popular



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

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

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

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