Sequential Models

RNN, LSTM

Presenters: Julius, Curtis

BOSTON UNIVERSITY
MACHINE INTELLIGENCE
COMMUNITY

Background

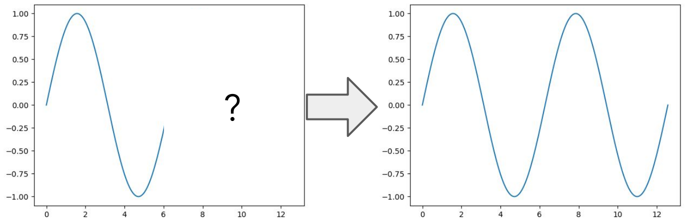
Conditional Probability and Sequential Data

Conditional Probability

1. Conditional Probability is the probability something happens given another event happened.
   a. P(passing | studying) > P(passing | gaming)
2. Conditional independence/dependance
   a. Conditional independence: $P(x) = P(x|y)$
   b. Conditional dependance: $P(x) \neq P(x|y)$

Sequential Data

1. Data is sequential if the probability of one data point is dependant on previous data
   a. $P(x_t) \neq P(x_t | x_{t-1}, \cdots, x_0)$
2. Example
   a. NLP
      i. The color of the bus is ______.
      ii. P(yellow | "The color of the bus is") > P(yellow)
   b. Sine wave



Sequential models

1. Sequential models predict the next data point given previous data points
2. Model Examples
   a. Recurrent Neural Networks (RNN)
   b. LSTM
   c. GRU
3. Applications
   a. Natural Language Processing
   b. Audio Signal Processing
   c. Reinforcement Learning

$$P(x_t | x_{t-1}, \cdots, x_0)$$

Recurrent Neural Networks

Combining new inputs with previous data

Recursive Functions

1. Our sequential data is $x_1, x_2, \cdots, x_t$
2. We wish to map each input, x, with a prediction, $y_1, y_2, \cdots, y_t$

We could learn a function that maps all of the previous inputs to one output, but that would require a function that takes a different number of inputs for each step of the sequence.

$$y_t = f(x_1, \cdots, x_t)$$

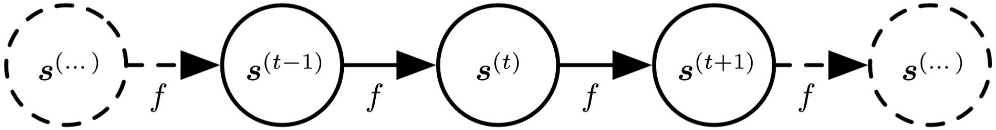
Instead, we want a function that takes only one state that summarizes all information necessary to make a prediction.

$$y_t = f(s_t)$$

Parameterized Recursive Functions

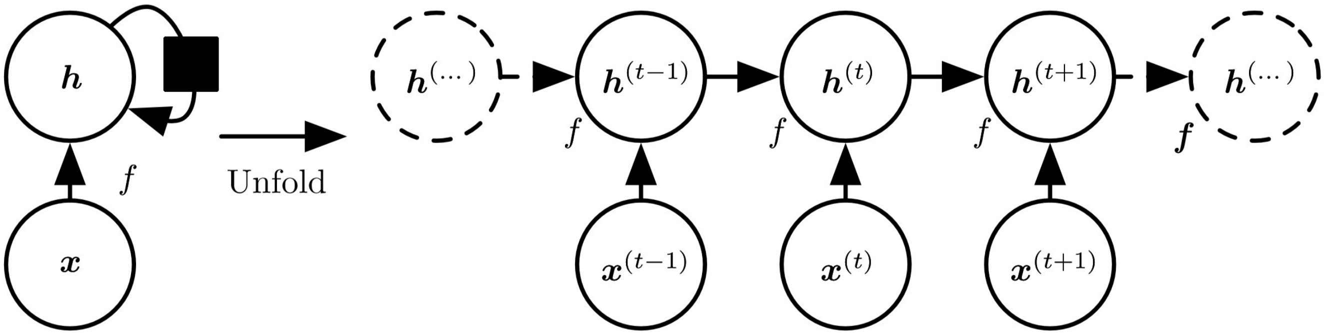
1. A recursive function takes an input as a result from itself
    a. We can add additional parameterization with the parameter theta
    b. This function could be a feed forward neural network

$$\underbrace{\boldsymbol{s}^{(t)}}_{\text{State at step t}} = f(\underbrace{\boldsymbol{s}^{(t-1)}}_{\text{State at step t-1}}; \boldsymbol{\theta})$$



A state is equal to the function output of the previous state

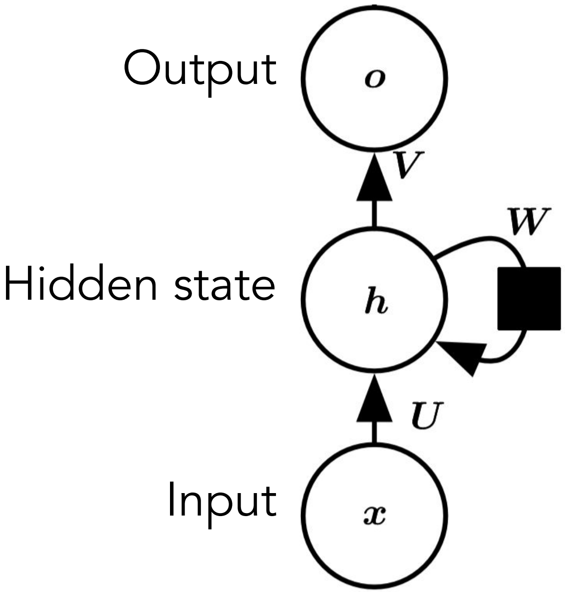
Unfolding



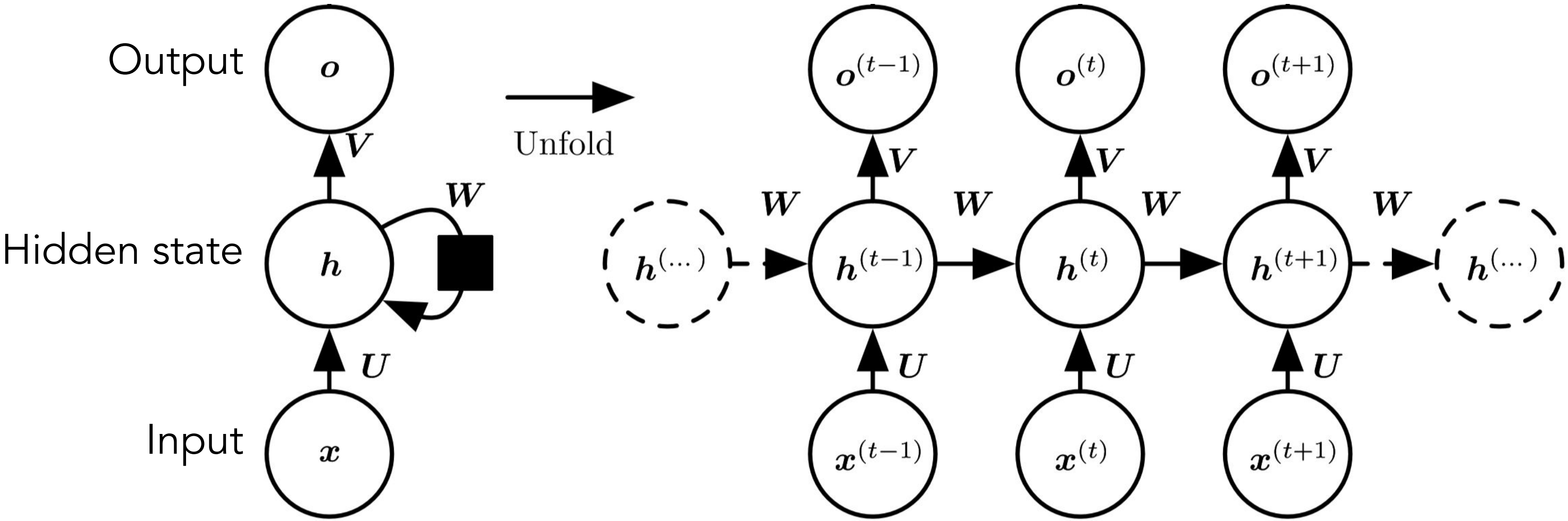
Recurrent Neural Network Goals

1. Summarize: Our network should take the previous state, h_{t-1} , and add the new input, x_t , to get the current state, h_t .
   a. This acts to summarize all previous inputs, $x_{1 \cdots t}$, into one state vector, h_t .
2. Predict: Our network should predict the output, o_t , from the current state, h_t
   a. These predictions are also noted as $\hat{y}_t$.
3. Train: The network should then compare the output, o_t, with the target value, y_t, and generate a loss, L_t .
   a. We then train the network through backpropagation using the gradient of the loss with respect to our weight matrices.

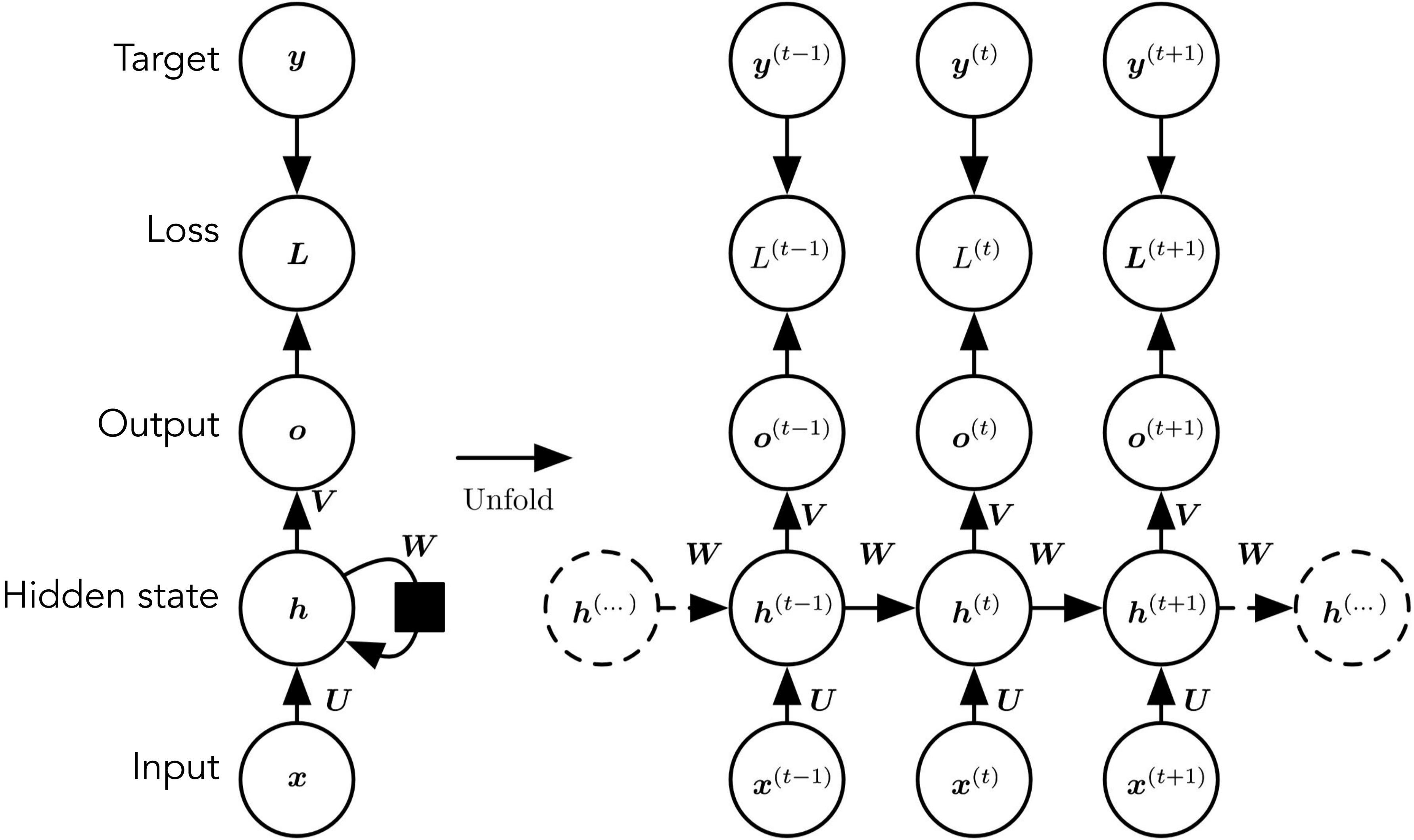
RNN Cell



RNN Graph



Output
Hidden state
Input
o
h
x
V
W
U
Unfold
$o^{(t-1)}$
$o^{(t)}$
$o^{(t+1)}$
$h^{(\dots)}$
$h^{(t-1)}$
$h^{(t)}$
$h^{(t+1)}$
$h^{(\dots)}$
$x^{(t-1)}$
$x^{(t)}$
$x^{(t+1)}$



Target
Loss
Output
Hidden state
Input
y
L
o
h
x
V
W
U
Unfold
$y^{(t-1)}$
$y^{(t)}$
$y^{(t+1)}$
$L^{(t-1)}$
$L^{(t)}$
$L^{(t+1)}$
$o^{(t-1)}$
$o^{(t)}$
$o^{(t+1)}$
$h^{(\dots)}$
$h^{(t-1)}$
$h^{(t)}$
$h^{(t+1)}$
$h^{(\dots)}$
$x^{(t-1)}$
$x^{(t)}$
$x^{(t+1)}$

RNN Details

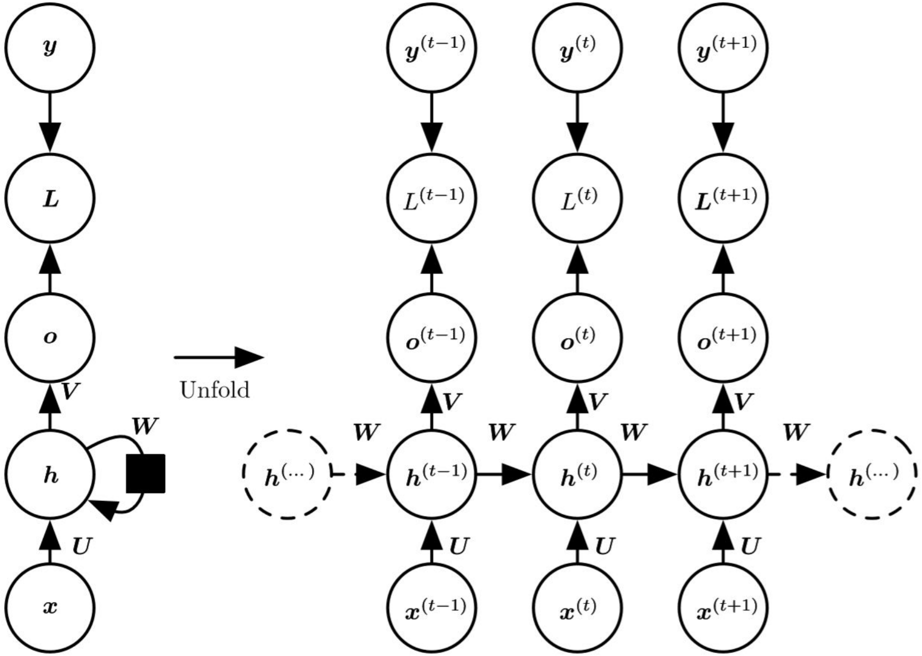
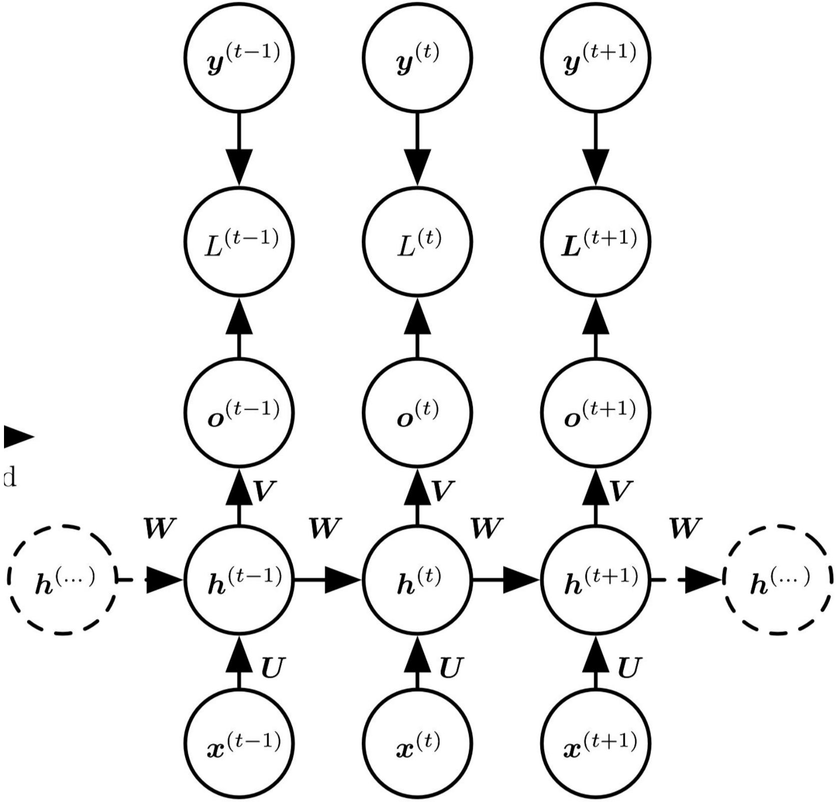


Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. When using softmax outputs, we assume $\boldsymbol{o}$ is the unnormalized log probabilities. The loss L internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. Equation 10.8 defines forward propagation in this model. *(Left)* The RNN and its loss drawn with recurrent connections. *(Right)* The same seen as a time-unfolded computational graph, where each node is now associated with one particular time instance.

RNN Feedforward Explanation

1. h_0 is initialized
2. x_1 is combined with h_0 to get h_1
3. o_1 is obtained from h_1
4. y_hat_1 probabilities is obtained from o_1
5. L_1 is computed from y_1, y_hat_1 with softmax
6. Process repeats for each t
7.



RNN Feedforward

Affine
$$\boldsymbol{a}^{(t)} = \underbrace{\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},}_{\text{Affine function (hidden network)}}$$

Hidden state
$$\boldsymbol{h}^{(t)} = \underbrace{\tanh(\boldsymbol{a}^{(t)}),}_{\text{Activation function}}$$

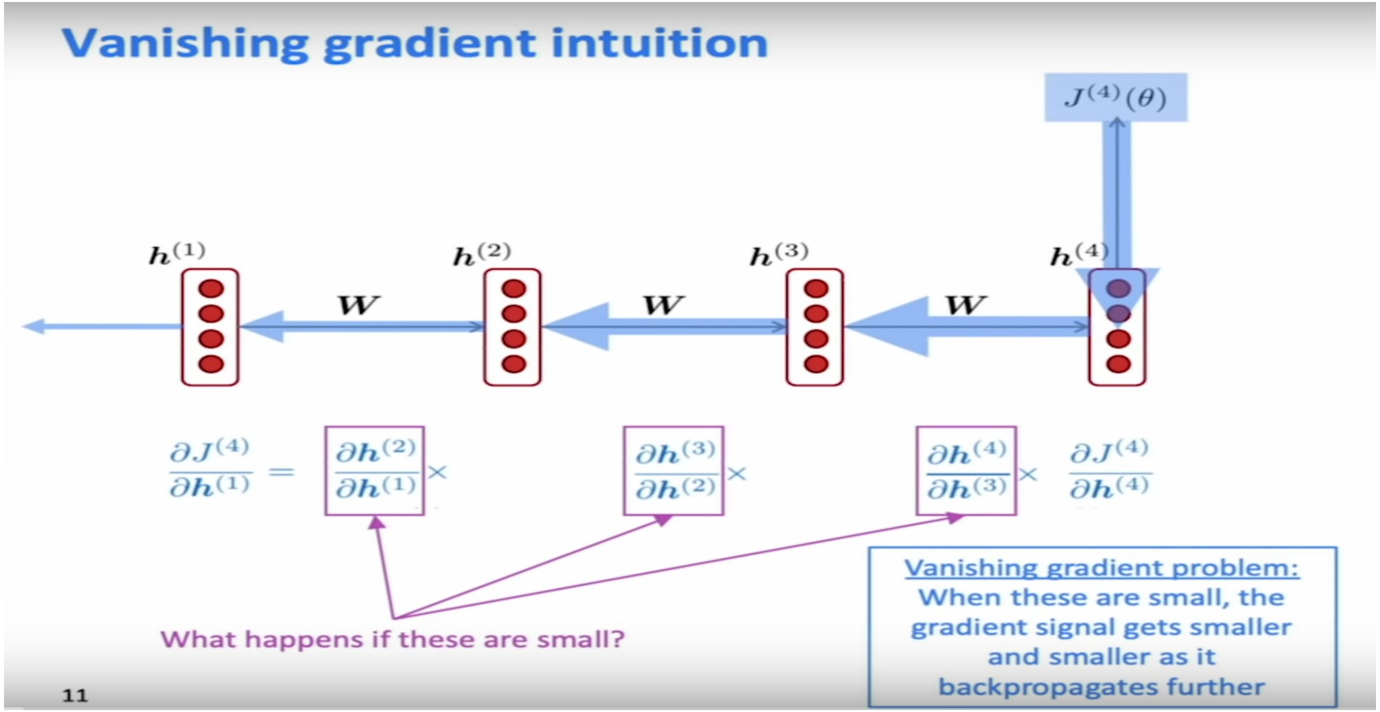
Output
$$\boldsymbol{o}^{(t)} = \underbrace{\boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},}_{\text{Output network}}$$

RNN Feedforward

Predictions
$$\hat{\boldsymbol{y}}^{(t)} = \underbrace{\operatorname{softmax}(\boldsymbol{o}^{(t)})}_{\text{Prediction probabilities}},$$

Loss
$$L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right)$$
$$= \sum_t L^{(t)}$$
$$= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right),$$

Backpropagation Through Time



Backpropagation Through Time

Gradient formulas for reference:

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L. \tag{10.19}$$

We can then iterate backward in time to back-propagate gradients through time, from $t = \tau - 1$ down to $t = 1$, noting that $\boldsymbol{h}^{(t)}$ (for $t < \tau$) has as descendents both $\boldsymbol{o}^{(t)}$ and $\boldsymbol{h}^{(t+1)}$. Its gradient is thus given by

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L) \tag{10.20}$$

$$= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{(t)}} L), \tag{10.21}$$

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, \tag{10.22}$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L, \tag{10.23}$$

$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L)\, \boldsymbol{h}^{(t)\top}, \tag{10.24}$$

375

CHAPTER 10

$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \tag{10.25}$$

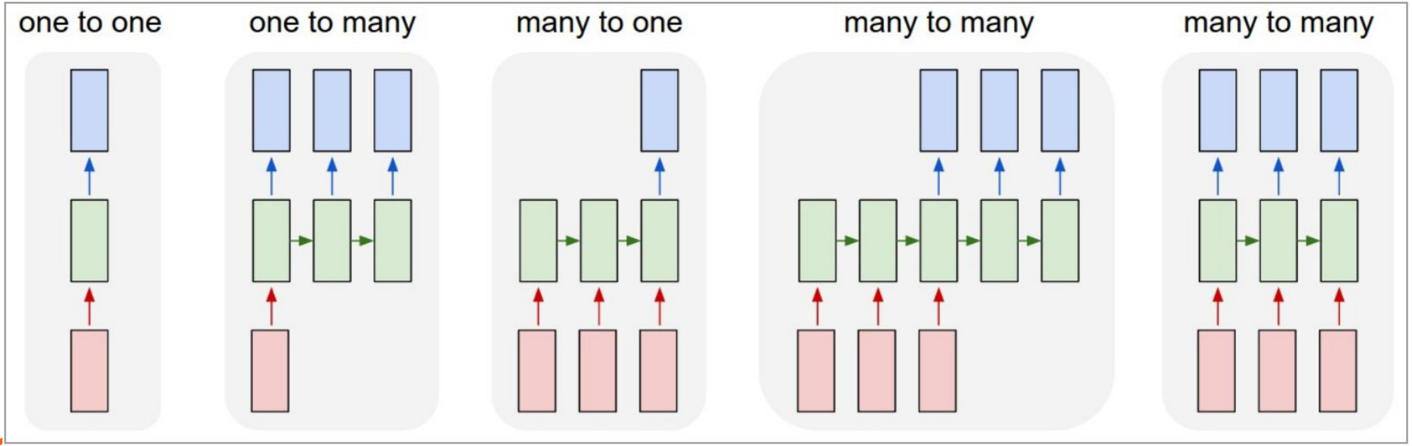
$$= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)\, \boldsymbol{h}^{(t-1)\top}, \tag{10.26}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \tag{10.27}$$

$$= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)\, \boldsymbol{x}^{(t)\top}, \tag{10.28}$$

Types of RNN problems

1. One-one: (given an image, is this a bird or not)
2. One-many: (given an image, give a description of this image)
3. Many-one: (given video, classify genre; predict sentiment of sentence)
4. Many-many: (given video, predict label for each frame; translate a sentence)



Deep RNNs

1. Why not just make RNNs more deep? Doesn’t solve...
    a. Vanishing gradient problem
    b. The network “forgets” information in the hidden state over long time steps
2. Adding Convolutions
    a. Use CNNs to have a “sliding window” over time steps
    b. Takes advantage of extracting information from features close in time with less computational overhead
3. Adding Attention
    a. Makes the network focus on particular parts of the input that is relevant to prediction

Long short-term memory (LSTM)

Brief introduction to LSTM’s

Issues with vanilla RNN's

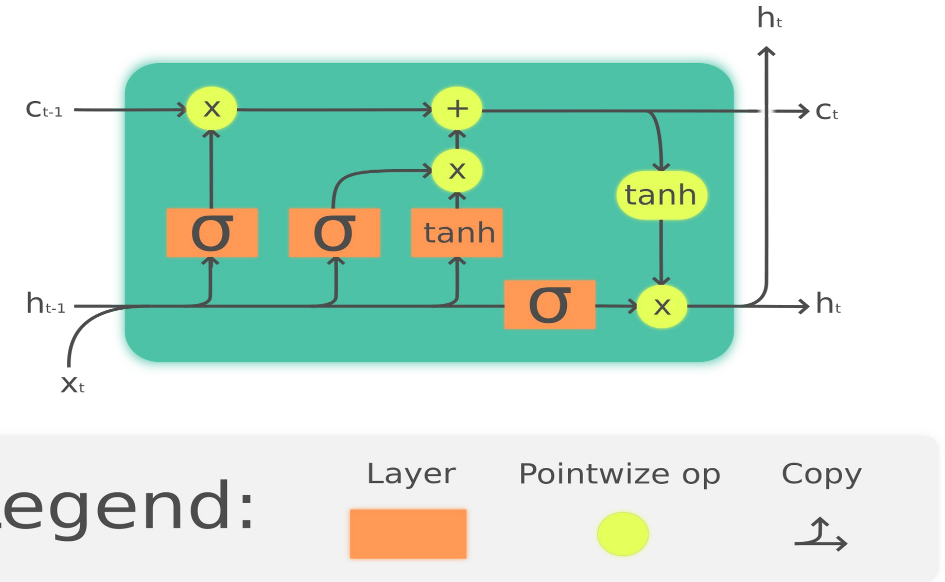
1. Vanishing Gradients
2. Difficult to retain long term memory within our hidden state h

Solution

1. Long Short Term Memory cells which maintain a hidden state, and a cell state.
2. RNN is able to read, write and erase information from cell state through gates.
   a. Gates are nearly open or nearly closed, open meaning information can pass through, closed meaning information cannot pass through.

Vanilla LSTM Cell

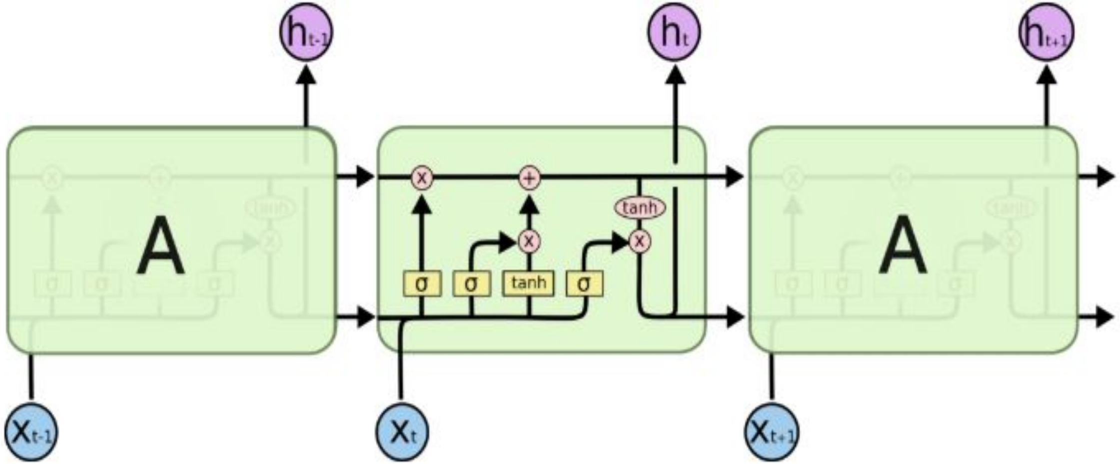
1.



https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:The_LSTM_cell.png

Multiple LSTM Cells

1.

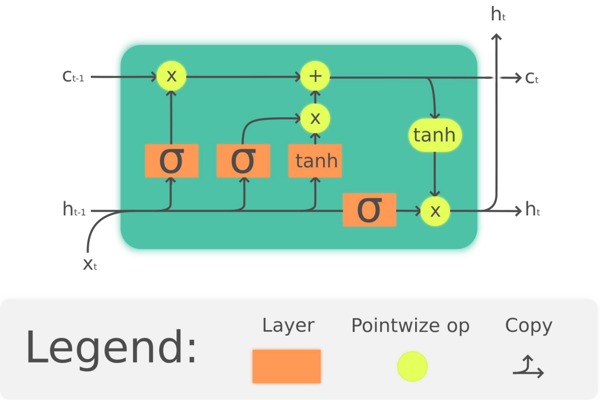


The repeating module in an LSTM contains four interacting layers.

https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4

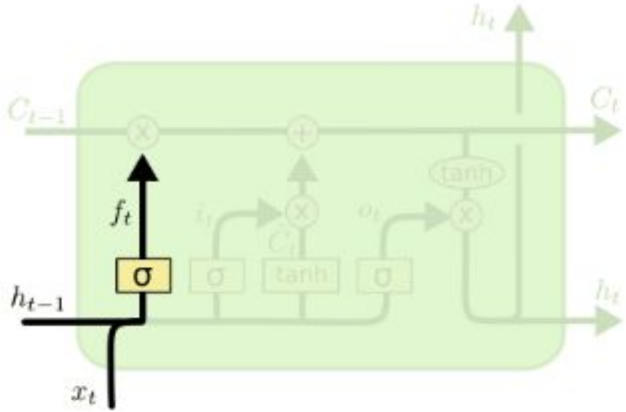
Vanilla LSTM Cell

1. Composed of several smaller neural networks
2. Each layer/nn have their own weights
3. Layers typically end in sigmoid or tanh
    a. Sigmoid squashes inputs between 0 and 1 deciding what information should be forgotten.
    b. Tanh rescales inputs between -1 and 1.



H := Previous Output, X := new input, C := Cell State

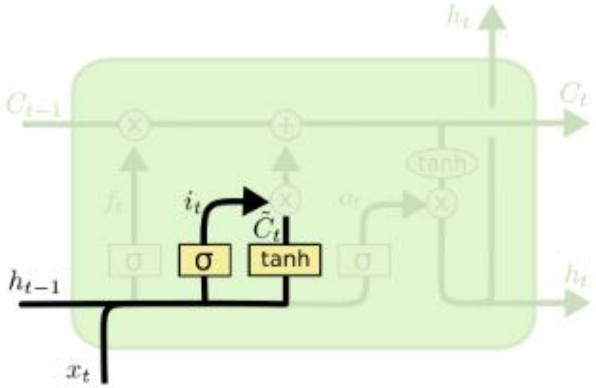
Intuition: Forget Gate



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

- Based on the previous output and current input, decides what information is relevant by squashing the combined feature vector [h_t-1, x_t] with a sigmoid function (squashes inputs between 0 and 1, 0 being forgotten input).
- Uses element-wise multiplication to effectively forget unimportant cell state features based on the relevancy matrix f_t

Intuition: Input Gate

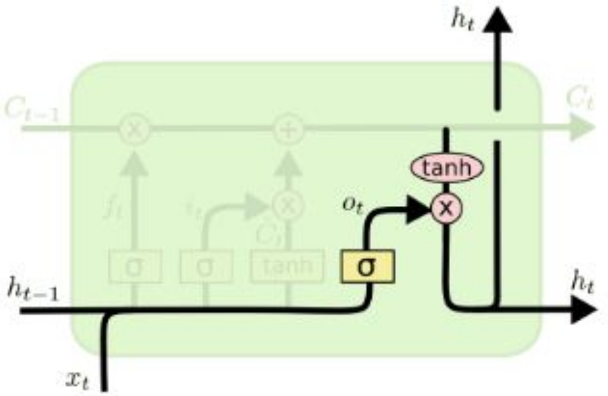


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

- Adds in new information to the cell state. Note, cell state acts as long term memory within a lstm cell.
- Does this by determining what is relevant like in the forget gate (i_t) and element wise multiplying that by rescaled inputs (C_t prime) then adding it to the current cell state (C_t).

Intuition: Output Gate



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

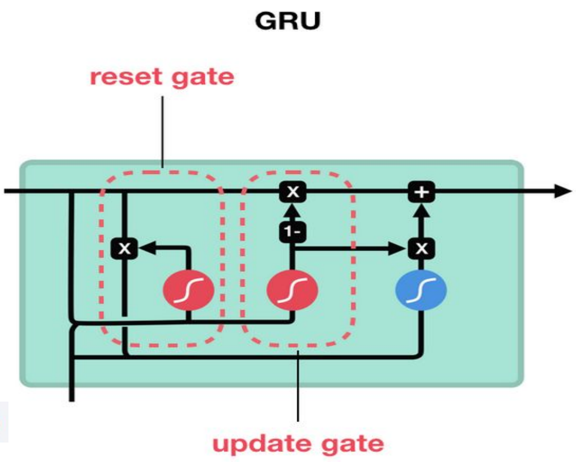
$$h_t = o_t * \tanh\left(C_t\right)$$

- Uses the existing cell state to decide what information will be propagated onwards to the next step/what the general output will be.
- Once again decides what's relevant (sigmoid) then forgets/reinforces relevant information through a dot product.

Conclusion

1. LSTM's allow long term memory to be stored for longer in comparison to vanilla RNN's
2. LSTM's do not avoid vanishing gradients entirely

GRUs act similarly to LSTM's with fewer weights and fewer gates.



Remember you can just do :)

```
CLASS torch.nn.LSTM(*args, **kwargs)
CLASS torch.nn.GRU(*args, **kwargs)
```

Thank you for coming!

References

Goodfellow, Ian, et al. *Deep Learning*. MIT Press, 2017. http://www.deeplearningbook.org