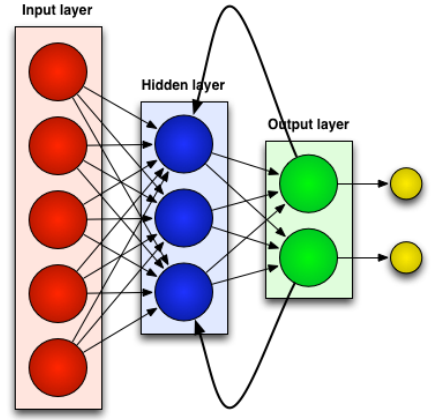
CS 615 – Deep Learning

Recurrent Neural Networks

Time Series Data

- We saw how Markov systems can be used on sequential data, for applications like:
  - Text processing
    - Data can be split into words or characters
  - Market Predictions
  - Video Processing
- Can we somehow tweak Neural Networks for this purpose?!



Recurrent Neural Networks (RNN)

- Recurrent Neural Networks (RNNs) are artificial neural networks that allow for processing sequential data.
- We feed in one sample at a time, in order, to train the network, "storing" gradients from previous observations of the sequence in some context layer.
- Uses of RNNs include:
  - Many to One
  - One to Many
  - Many to Many

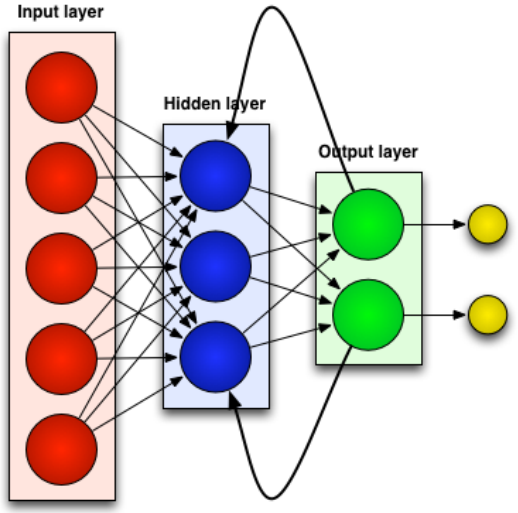
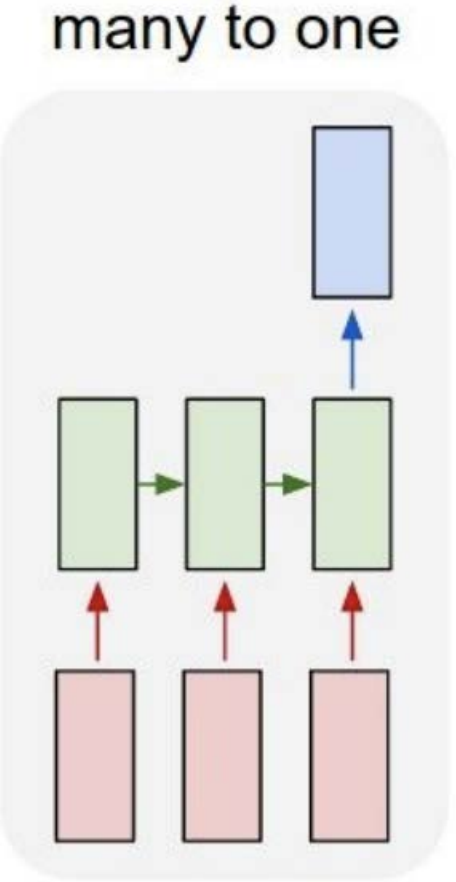


Image from towardsdatascience.com

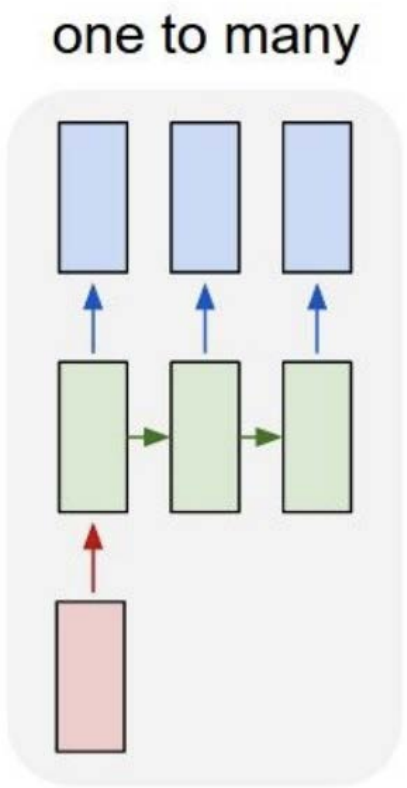
Many to One

- One type of RNN pushes a data sequence through our RNN to obtain a single value.
- Example Application: Video Classification
  - The “many” is the frames
  - The “one” is the classification of the video



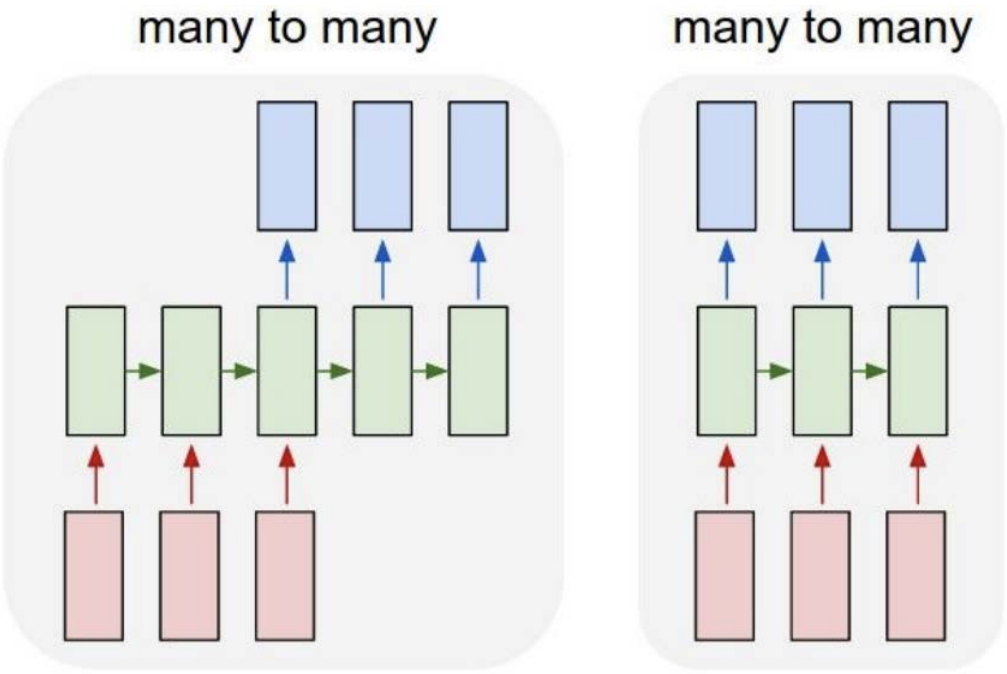
One to Many

- In another type we provide a single “seed” observation to the RNN and let it generate a sequence using the seed and the trained RNN
- Example Application: Generating Synthetic Data Sequence
  - The “one” is some starting value.
  - The “many” is the data sequence (pixels, words, etc..)



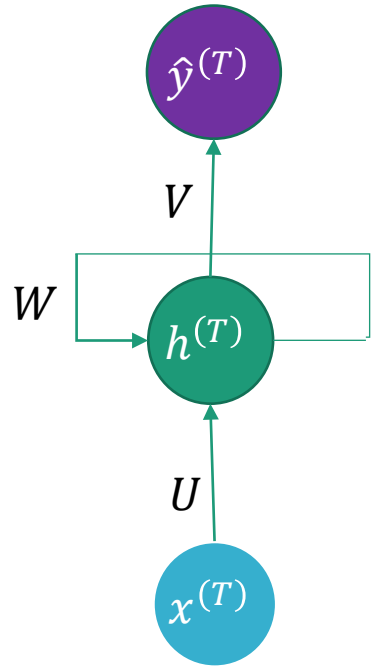
Many to Many

- And finally we might want to feed in a sequence and get out of sequence!
- Example applications: Translation
  - Provide a sequence of words
  - Get back a new sequence of words



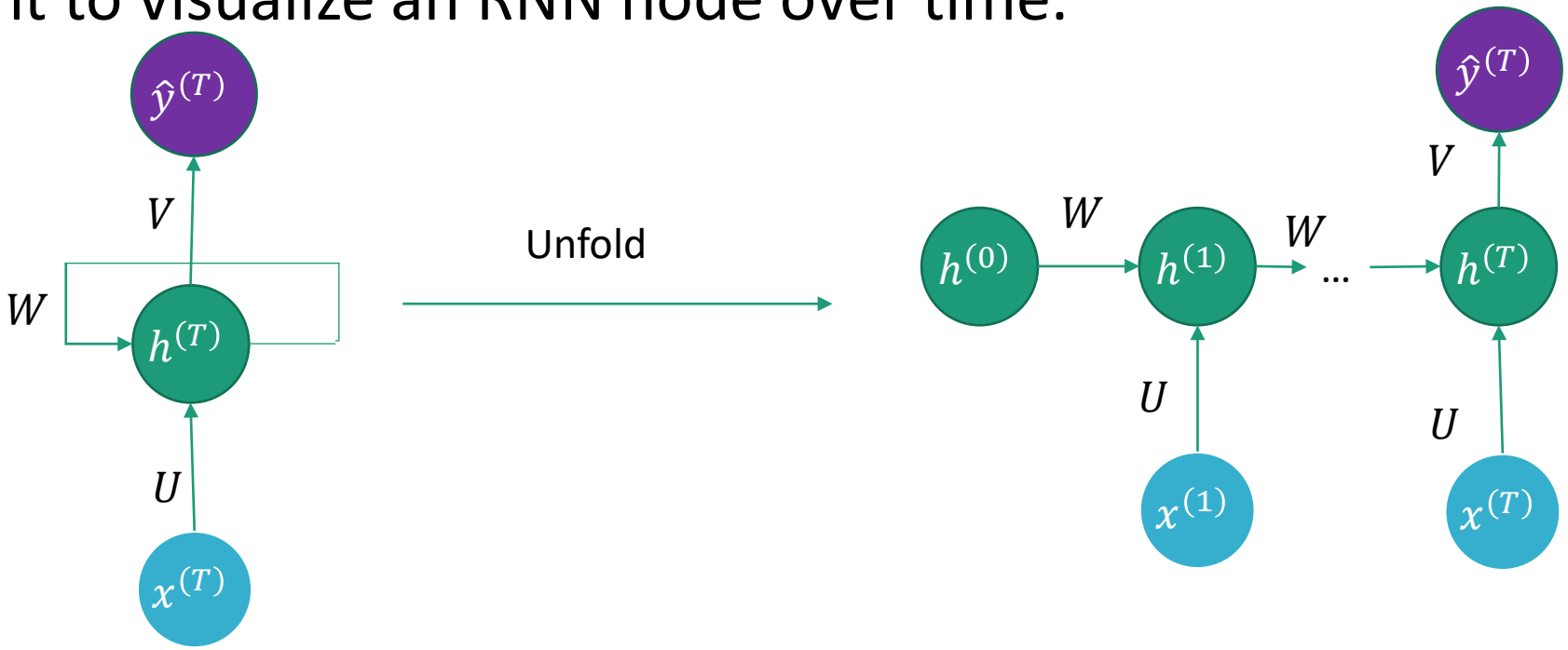
Recurrent Nodes

- A simple RNN has three sets of weights (we'll use different variables, to line up with the provided images).
- Like a traditional shallow ANN we have
  - Weights going from input layer to hidden layer, V
  - Weights going from hidden layer to output layer, U
- With an RNN we also have:
  - Weights going from hidden layer back to hidden layer, $\boldsymbol{W}$



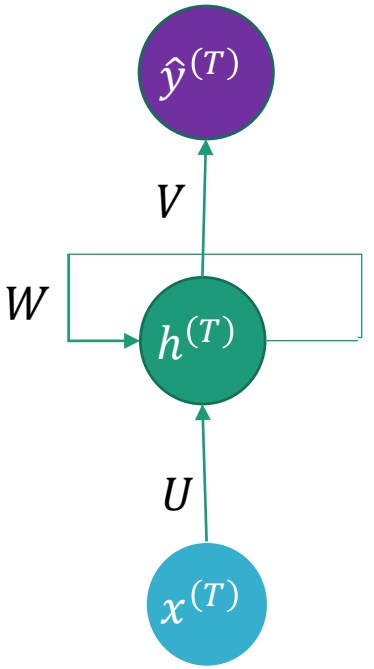
Unrolling

- Although there is no real “sequence” of units in RNNs, for illustrative and comprehension purposes, often you’ll see an RNN “unfolded”.
- This is meant to visualize an RNN node over time.



Notation

- Let
  - $\hat{y}^{(t)}$ be the output at time t
  - $neto^{(t)}$ be the input to the output layer at time t
  - $h^{(t)}$ be the output of the hidden layer at time t
  - $neth^{(t)}$ be the input to the hidden layer at time t
  - $x^{(t)}$ be the input at time t
  - V be the weights going from the hidden layer to the output layer
  - U be the weights going from the input layer to the hidden layer
  - W be the weights going from the hidden layer back to itself.
  - g_o, g_h be the activation functions for the output and hidden layers, respectively.



Forward Propagation

- As per usual, forward propagation is *relatively* easy.
- The only difference is that the **output of the hidden layer involves summing the previous weighed output of the hidden layer with the new weighed input**
- Then then input to the hidden layer at time t is:

$$neth^{(t)} = h^{(t-1)}W + x^{(t)}U$$

Feedback contribution

New contribution

- And the input to the output layer at time t is:

$$neto^{(t)} = h^{(t)}V$$

Counting Example

- As we work through our RNN, let's use a *one-to-many* example where we input 1 and have the RNN output 2 then 3
- So our input sequence is 1
- And our desired output sequence is 2,3
- For training purposes we'll do one-hot encoding:
  - $1 \Rightarrow [1, 0, 0]$
  - $2 \Rightarrow [0, 1, 0]$
  - $3 \Rightarrow [0, 0, 1]$
- Therefore our training set is:

$$x^{(1)} = [1, 0, 0]$$
$$y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example

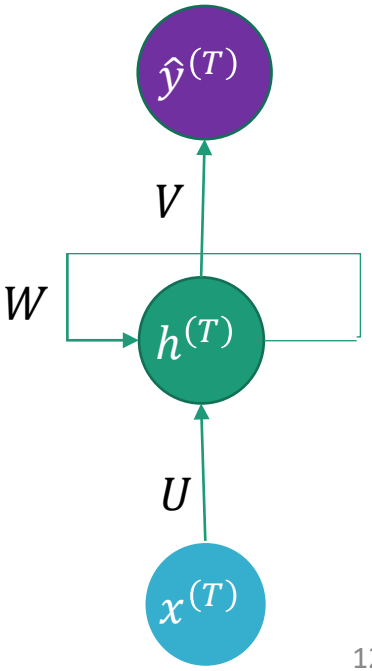
- Let's imagine that our weight matrices are:

$$U = \begin{bmatrix} -0.1565 & 0.9190 \\ 0.8315 & 0.3115 \\ 0.5844 & -0.9286 \end{bmatrix}, V = \begin{bmatrix} 0.6983 & 0.3575 & 0.4863 \\ 0.8680 & 0.5155 & -0.2155 \end{bmatrix}, W = \begin{bmatrix} 0.3110 & 0.4121 \\ -0.6576 & -0.9363 \end{bmatrix}$$

- Forward propagation results in:
  - Time $t = 0$
    - We need to initialize the "previous" h somehow, so let's just set it to zeros.
    - If we choose a hidden layer size of two, then we initialize $h^{(0)}$ to

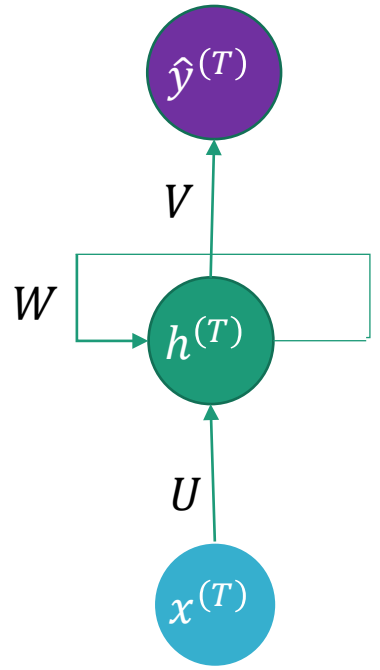
$$h^{(0)} = [0 \quad 0]$$

  - Time $t = 1$
    - $x^{(1)} = [1 \quad 0 \quad 0]$
    - $h^{(1)} = [-0.1565 \quad 0.9190]$
    - $\hat{y}^{(1)} = [0.6884 \quad 0.4178 \quad -0.2742]$



Example

- Forward propagation results in:
  - Time $t = 2$
    - Now we don't have any input x, so let's just set it to $x^{(2)} = [0\ 0\ 0]$
    - $x^{(2)} = [0\ 0\ 0]$
    - $h^{(1)} = [-0.1565 \quad 0.9190]$
    - $h^{(2)} = [-0.6530 \quad -0.9250]$
    - $\hat{y}^{(2)} = [-1.2588 \quad -0.7102 \quad -0.1182]$
- Use the argmax of $\hat{y}^{(t)}$ we get the sequence (1, 3)
- Our desired output was (2,3)
- So let's backpropagate to learn the weights better!



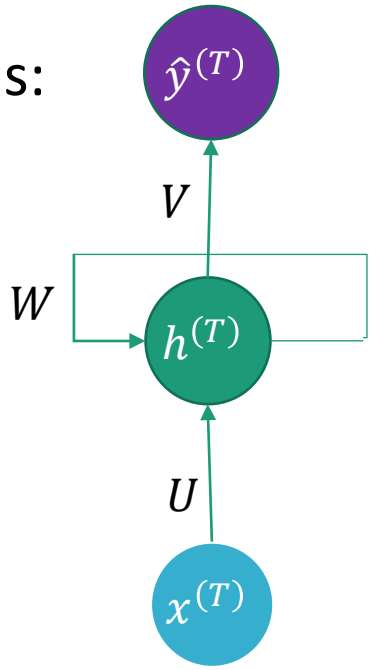
Example: Backpropagate

- For this example, we'll use a relatively simple configuration of a **squared error objective function** and **linear activation functions** at both the hidden and output layers.
- Since we care about the output at every time step, our objective function is:

$$J = \sum_{t=1}^{T} J^{(t)} = \sum_{t=1}^{T} \left(y^{(t)} - \hat{y}^{(t)}\right)^T \left(y^{(t)} - \hat{y}^{(t)}\right)$$

- Now it's time to backpropagate!
- Of course to do so we'll need to determine the gradient rules:

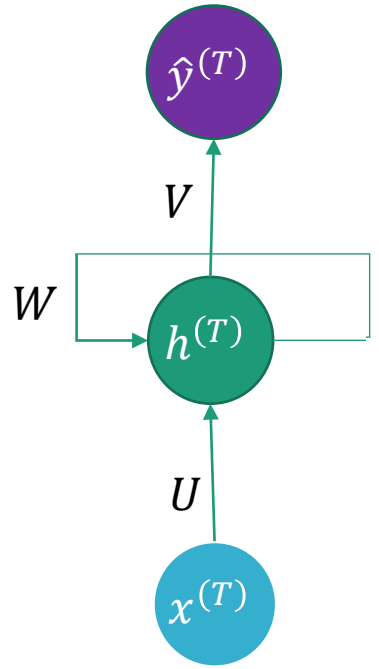
$$\frac{\partial J}{\partial V}, \frac{\partial J}{\partial W}, \frac{\partial J}{\partial U}$$



Single Input, Multiple Output: $\partial J / \partial V$

- As usual, our first gradient rule (the one nearest the output) is the easiest.

$$\frac{\partial J}{\partial V} = \sum_{t=1}^{T} \frac{\partial J^{(t)}}{\partial V} = \sum_{t=1}^{T} \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \cdot \frac{\partial \hat{y}^{(t)}}{\partial neto^{(t)}} \cdot \frac{\partial neto^{(t)}}{\partial V}$$



- And for the purposes of back propagation

$$\delta_o^{(t)} = \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \cdot \frac{\partial \hat{y}^{(t)}}{\partial neto^{(t)}}$$

$$\frac{\partial J}{\partial V} = \sum_{t=1}^{T} (h^{(t)})^T \delta_o^{(t)}$$

Example: $\partial J / \partial V$

- For our configuration:

$$\delta_o^{(t)} = \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \cdot \frac{\partial \hat{y}^{(t)}}{\partial net o^{(t)}} = 2\left(\hat{y}^{(t)} - y^{(t)}\right) \cdot 1$$

$$\frac{\partial J}{\partial V} = \sum_{t=1}^{T} {h^{(t)}}^T \delta_o^{(t)}$$

Example: $\partial J / \partial V$

- Plugging in numbers
  - $\delta_o^{(1)} = 2\left(\hat{y}^{(1)} - y^{(1)}\right) = [1.3768 \quad -1.1644 \quad -0.5483]$
  - $\frac{\partial J^{(1)}}{\partial V} = \left(h^{(1)}\right)^T \cdot \delta_o^{(1)} = \begin{bmatrix} -0.2154 & 0.1822 & 0.0858 \\ 1.2653 & -1.0701 & -0.5039 \end{bmatrix}$
  - $\delta_o^{(2)} = 2\left(\hat{y}^{(2)} - y^{(2)}\right) = [-2.5176 \quad -1.4205 \quad -0.2363]$
  - $\frac{\partial J^{(2)}}{\partial V} = \left(h^{(2)}\right)^T \cdot \delta_o^{(2)} = \begin{bmatrix} 1.6440 & 0.9276 & 1.4603 \\ 2.3287 & 1.3139 & 2.0685 \end{bmatrix}$
- So

$$\frac{\partial J}{\partial V} = \begin{bmatrix} 1.4286 & 1.1098 & 1.5461 \\ 3.5940 & 0.2438 & 1.5646 \end{bmatrix}$$

Single Input, Multiple Output: $\partial J/\partial W$

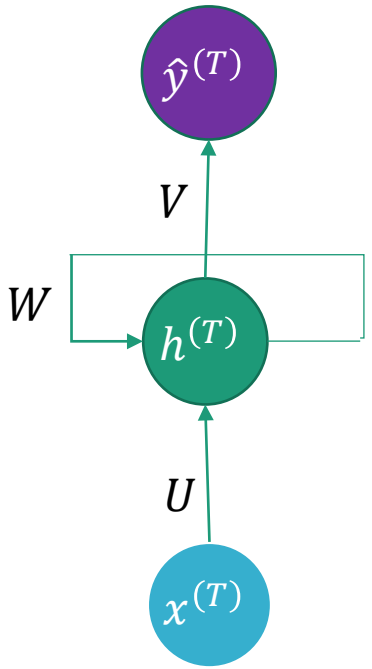
- Next, we have $\frac{\partial J}{\partial W}$

$$\frac{\partial J}{\partial W} = \sum_{t=1}^{T} \left(\frac{\partial \boldsymbol{J^{(t)}}}{\partial \boldsymbol{\hat{y}^{(t)}}} \cdot \frac{\partial \boldsymbol{\hat{y}^{(t)}}}{\partial \boldsymbol{neto^{(t)}}} \cdot \frac{\partial \boldsymbol{neto^{(t)}}}{\partial \boldsymbol{h^{(t)}}} \cdot \frac{\partial \boldsymbol{h^{(t)}}}{\partial \boldsymbol{neth^{(t)}}} \cdot \frac{\partial \boldsymbol{neth^{(t)}}}{\partial \boldsymbol{W}} \right)$$

$$\frac{\partial J}{\partial W} = \sum_{t=1}^{T} \left(\boldsymbol{\delta_o^{(t)}} \cdot \frac{\partial \boldsymbol{neto^{(t)}}}{\partial \boldsymbol{h^{(t)}}} \cdot \frac{\partial \boldsymbol{h^{(t)}}}{\partial \boldsymbol{neth^{(t)}}} \cdot \frac{\partial \boldsymbol{neth^{(t)}}}{\partial \boldsymbol{W}} \right)$$

- Of course $\frac{\partial neto^{(t)}}{\partial h^{(t)}} = V^T$

$$\frac{\partial J}{\partial W} = \sum_{t=1}^{T} \boldsymbol{\delta_o^{(t)}} \cdot \boldsymbol{V^T} \cdot \frac{\partial \boldsymbol{h^{(t)}}}{\partial \boldsymbol{neth^{(t)}}} \cdot \frac{\partial \boldsymbol{neth^{(t)}}}{\partial \boldsymbol{W}}$$



$$h^{(t)} = g\left(neth^{(t)}\right)$$

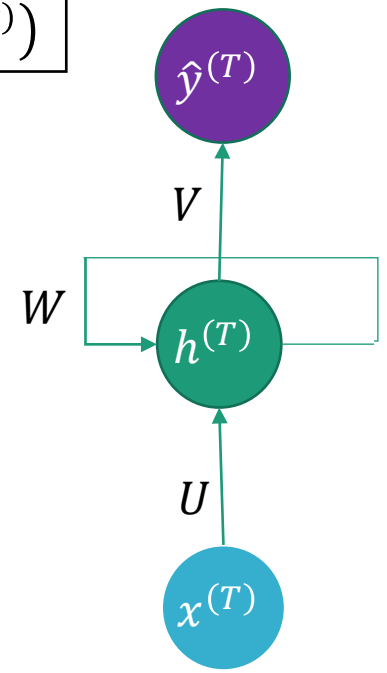
$$neth^{(t)} = h^{(t-1)}W + x^{(t)}U$$

Single Input, Multiple Output: $\partial J/\partial W$

$$neth^{(t)} = h^{(t-1)}W + x^{(t)}U$$

$$h^{(t)} = g\left(neth^{(t)}\right)$$

$$\frac{\partial J}{\partial W} = \sum_{t=1}^{T} \boldsymbol{\delta_o^{(t)}} \cdot \boldsymbol{V^T} \cdot \frac{\boldsymbol{\partial h^{(t)}}}{\boldsymbol{\partial neth^{(t)}}} \cdot \frac{\boldsymbol{\partial neth^{(t)}}}{\boldsymbol{\partial W}}$$



- Upon first inspection $\frac{\partial neth^{(t)}}{\partial W} = \left(h^{(t-1)}\right)^T$.
- However, changing W at time t will affect all previous values of $neth$ for $k = 1, \ldots, t$, which in turn changes $J^{(t)}$

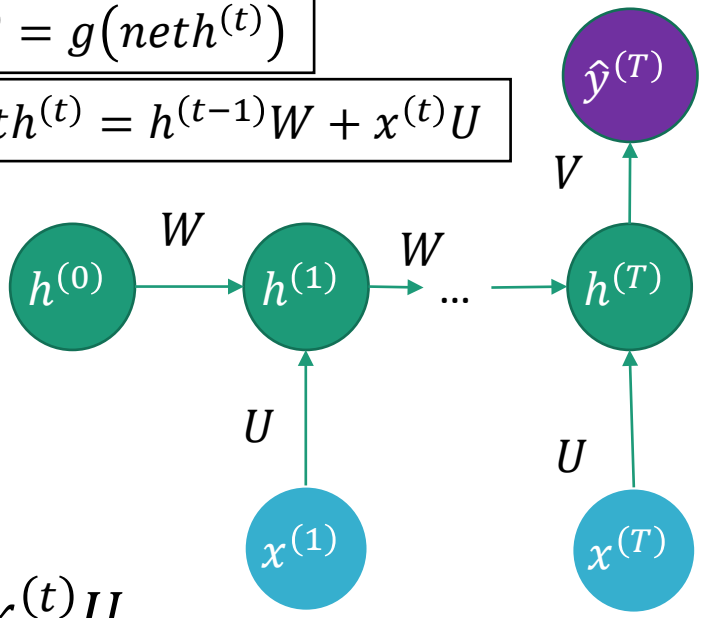
NOTE: The derivations for $\frac{\partial J}{\partial W}, \frac{\partial J}{\partial U}$ will likely be a bit different than in the resources. I think these are the most intuitive, although not necessarily the most efficient.

Single Input, Multiple Output: $\partial J/\partial W$

$$\frac{\partial J^{(t)}}{\partial W} = \sum_{t=1}^{T} \boldsymbol{\delta_o^{(t)} \cdot V^T} \cdot \frac{\boldsymbol{\partial h^{(t)}}}{\boldsymbol{\partial neth^{(t)}}} \cdot \frac{\boldsymbol{\partial neth^{(t)}}}{\boldsymbol{\partial W}}$$

$$h^{(t)} = g\left(neth^{(t)}\right)$$

$$neth^{(t)} = h^{(t-1)}W + x^{(t)}U$$



- We can write $neth^{(t)}$ recursively as

$$\begin{aligned} neth^{(t)} &= g_h\left(neth^{(t-1)}\right)W + x^{(t)}U \\ &= g_h\left(h^{(t-2)}W + x^{(t-1)}U\right)W + x^{(t)}U \\ &= g_h\left(g_h(h^{(t-3)}W + x^{(t-2)}U)\, W + x^{(t-1)}U\right)W + x^{(t)}U \\ &= \ \ldots \end{aligned}$$

- Notice how we must also propagate backwards *in time*!
- Therefore, learning $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial U}$ is often called ***back propagation through time (bptt).***

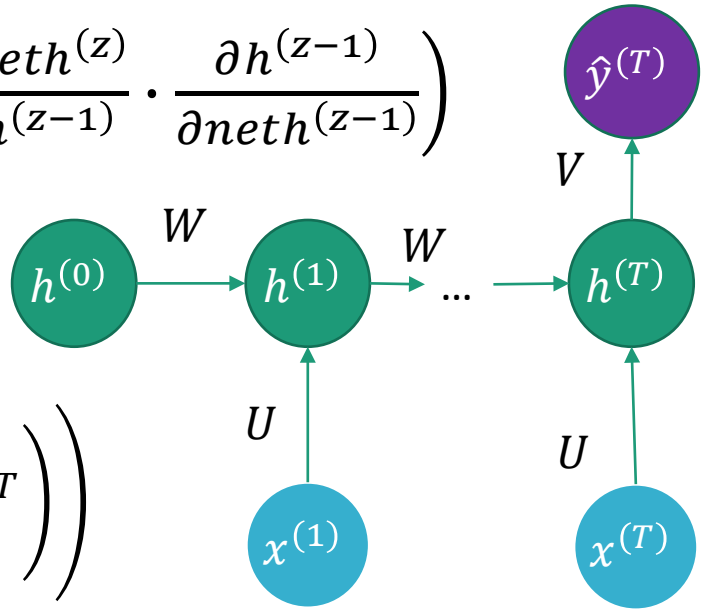
Single Input, Multiple Output: $\partial J/\partial W$

$$neth^{(t)} = g_h\left(g_h\left(h^{(t-3)}W + x^{(t-2)}U\right)W + x^{(t-1)}U\right)W + x^{(t)}U = \cdots$$

- Via the chain rule:

$$\frac{\partial neth^{(t)}}{\partial W} = \frac{\partial neth^{(t)}}{\partial W}\prod_{z=2}^{t}\left(\frac{\partial neth^{(z)}}{\partial neth^{(z-1)}}\right) = \frac{\partial neth^{(t)}}{\partial W}\prod_{z=2}^{t}\left(\frac{\partial neth^{(z)}}{\partial h^{(z-1)}}\cdot\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}}\right)$$

$$= \left(h^{(t-1)}\right)^T \cdot \prod_{z=2}^{t}\left(\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}}\cdot W^T\right)$$



- So finally:

$$\frac{\partial J}{\partial W} = \sum_{t=1}^{T}\left(h^{(t-1)^T}\cdot\delta_o^{(t)}\cdot V^T\cdot\prod_{z=2}^{t}\left(\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}}W^T\right)\right)$$

$$h^{(t)} = g\left(neth^{(t)}\right)$$

$$neth^{(t)} = h^{(t-1)}W + x^{(t)}U$$

Example: $\partial J / \partial W$

$$\frac{\partial J}{\partial W} = \sum_{t=1}^{T} \left(h^{(t-1)^T} \cdot \delta_o^{(t)} \cdot V^T \cdot \prod_{z=2}^{t} \left(\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}} W^T \right) \right)$$

- For our example…
- $\frac{\partial J^{(1)}}{\partial W} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} [1.3768 \quad -1.1644 \quad -0.5483] \begin{bmatrix} 0.6983 & 0.8680 \\ 0.3575 & 0.5155 \\ 0.4863 & -0.2155 \end{bmatrix} (\quad) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
- $\frac{\partial J^{(2)}}{\partial W} = \begin{bmatrix} -0.1565 \\ 0.9190 \end{bmatrix} [-2.5176 \quad -1.4205 \quad -2.2363] \begin{bmatrix} 0.6983 & 0.8680 \\ 0.3575 & 0.5155 \\ 0.4863 & -0.2155 \end{bmatrix} \left(\begin{bmatrix} 0.3110 & -0.6576 \\ 0.4121 & -0.9363 \end{bmatrix} \right) = \begin{bmatrix} 0.3203 & -0.7019 \\ -1.8805 & 4.1222 \end{bmatrix}$
- So

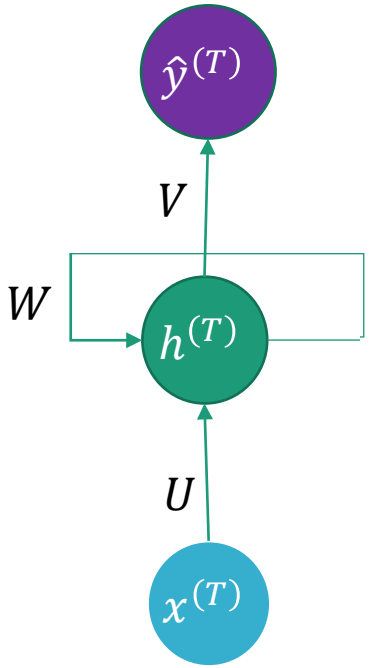
$$\frac{\partial J}{\partial W} = \begin{bmatrix} 0.3203 & -0.7019 \\ -1.8805 & 4.1222 \end{bmatrix}$$

Single Input, Multiple Output: $\partial J/\partial U$

- Next, we have $\frac{\partial J}{\partial U}$
- Changes in U at time t also affects h at previous times and therefore $J^{(t)}$
- Similarly:

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \left(\frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \cdot \frac{\partial \hat{y}^{(t)}}{\partial neto^{(t)}} \cdot \frac{\partial neto^{(t)}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial neth^{(t)}} \cdot \frac{\partial neth^{(t)}}{\partial U} \right)$$

$$= \sum_{t=1}^{T} \delta_o^{(t)} \cdot V^T \cdot \frac{\partial h^{(t)}}{\partial neth^{(t)}} \cdot \frac{\partial neth^{(t)}}{\partial U}$$



$$h^{(t)} = g\left(neth^{(t)}\right)$$

$$neth^{(t)} = h^{(t-1)}W + x^{(t)}U$$

Single Input, Multiple Output: $\partial J/\partial U$

$$neth^{(t)} = g_h\left(g_h\left(h^{(t-3)}W + x^{(t-2)}U\right)W + x^{(t-1)}U\right)W + x^{(t)}U = \cdots$$

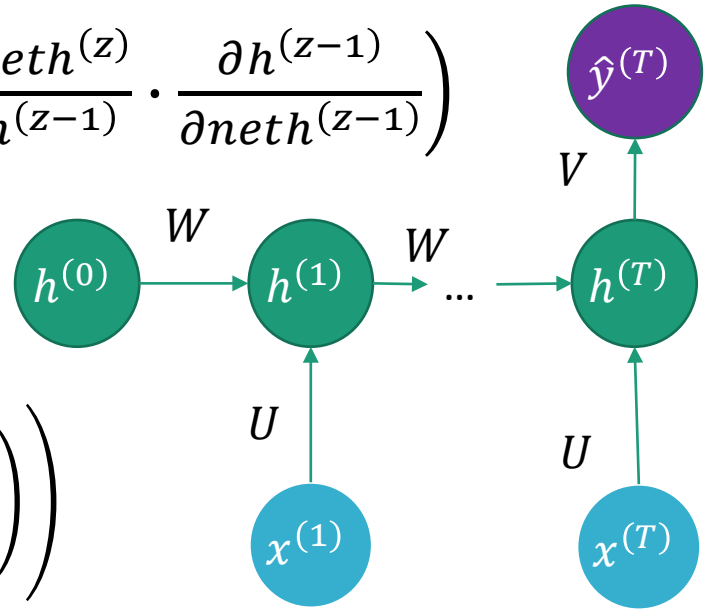
- Via the chain rule:

$$\frac{\partial neth^{(t)}}{\partial U} = \frac{\partial neth^{(t)}}{\partial U}\prod_{z=2}^{t}\left(\frac{\partial neth^{(z)}}{\partial neth^{(z-1)}}\right) = \frac{\partial neth^{(t)}}{\partial U}\prod_{z=2}^{t}\left(\frac{\partial neth^{(z)}}{\partial h^{(z-1)}}\cdot\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}}\right)$$

$$= \left(x^{(1)}\right)^T \cdot \prod_{z=2}^{t}\left(\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}}\cdot W^T\right)$$

- So finally:

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T}\left(x^{(t)^T}\cdot\delta_o^{(t)}\cdot V^T\cdot\prod_{z=2}^{t}\left(\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}}W^T\right)\right)$$



$$h^{(t)} = g\left(neth^{(t)}\right)$$

$$neth^{(t)} = h^{(t-1)}W + x^{(t)}U$$

Example: $\partial J / \partial U$

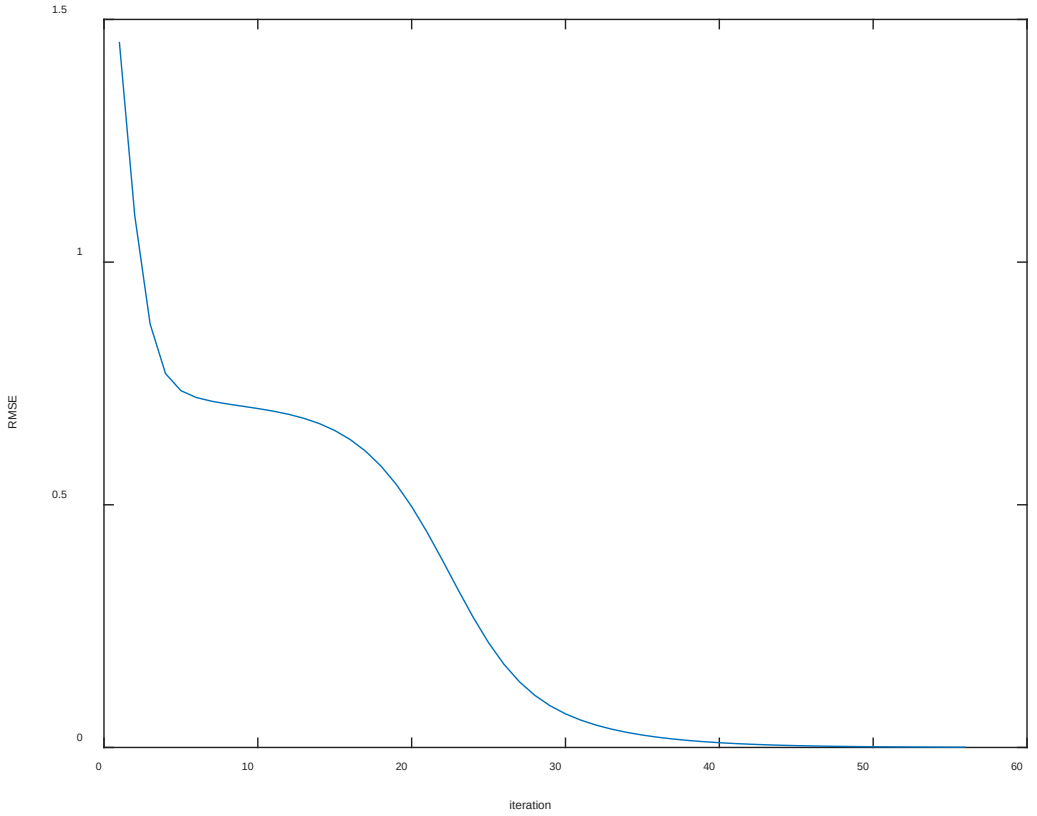
$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \left(x^{(t)^T} \cdot \delta_o^{(t)} \cdot V^T \cdot \prod_{z=2}^{t} \left(\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}} W^T \right) \right)$$

- For our example…
- $\frac{\partial J^{(1)}}{\partial U} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} [1.3768 \quad -1.1644 \quad -0.5483] \begin{bmatrix} 0.6983 & 0.8680 \\ 0.3575 & 0.5155 \\ 0.4863 & -0.2155 \end{bmatrix} (\quad) = \begin{bmatrix} 0.2785 & 0.7130 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
- $\frac{\partial J^{(2)}}{\partial U} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} [-2.5176 \quad -1.4205 \quad -2.2363] \begin{bmatrix} 0.6983 & 0.8680 \\ 0.3575 & 0.5155 \\ 0.4863 & -0.2155 \end{bmatrix} \left(\begin{bmatrix} 0.3110 & -0.6576 \\ 0.4121 & -0.9363 \end{bmatrix} \right) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
- So

$$\frac{\partial J}{\partial U} = \begin{bmatrix} 0.2785 & 0.7130 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Example

- Terminated if $|RMSE(end) - RMSE(end-1)| < 10^{-4}$
  - Using learning rate $\eta = 0.1$
  - This occurred after 56 iterations.
- Final Model:
  - $V = \begin{bmatrix} 0.0007 & -0.9555 & -0.4747 \\ -0.0001 & 0.3239 & -0.4533 \end{bmatrix}$
  - $W = \begin{bmatrix} 0.2870 & 0.4962 \\ -0.4082 & -1.5386 \end{bmatrix}$
  - $U = \begin{bmatrix} -0.7724 & 0.8089 \\ 0.8315 & 0.3115 \\ 0.5844 & -0.9286 \end{bmatrix}$
- Forward propagating using the learned model
  - $\hat{y}^{(1)} = [-0.0007 \quad 1.0001 \quad -0.0001]$
  - $\hat{y}^{(2)} = [-0.0003 \quad 0.0001 \quad 0.9999]$
- Choosing argmax, we get the sequence $(2, 3)$



Exploding/Vanishing Gradients

- If you noticed, by recurring through time we are often multiplying W by itself several times.

$$\prod_{z=2}^{t}\left(\frac{\partial h^{(z-1)}}{\partial neth^{(z-1)}}W^{T}\right)$$

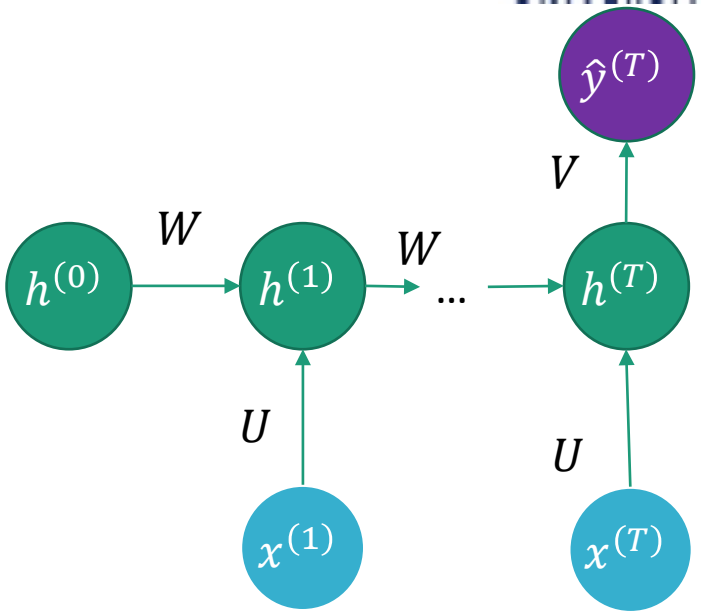
- In fact, t temporal units forward, we'll have to multiply W by itself t times!
- As a result, if $|W|$ is greater than one, we get exponential "blow up".
- Likewise if $|W|$ is very small, then we get exponential decay (vanish).
- There are various creative work-around for this, but we'll leave that up to you to research.
- One idea/fix is to limit how far back you go (don't go all the way back to $t = 1$)

Addition RNN Scenarios

- We just went through the one-to-many scenario.
- Recall that some applications call for a many-to-one situation.
- Here we only evaluate at the end:

$$J = J^{(T)}$$

- However, for all times $x^{(t)}$ we have valid inputs
  - Recall that for the one-to-many situation when $t > 1$ x was just a vector of zeros (no input).
- So we'd need to define/tweak our gradients accordingly.
- Likewise, a many-to-many application would almost be a hybrid of these two.

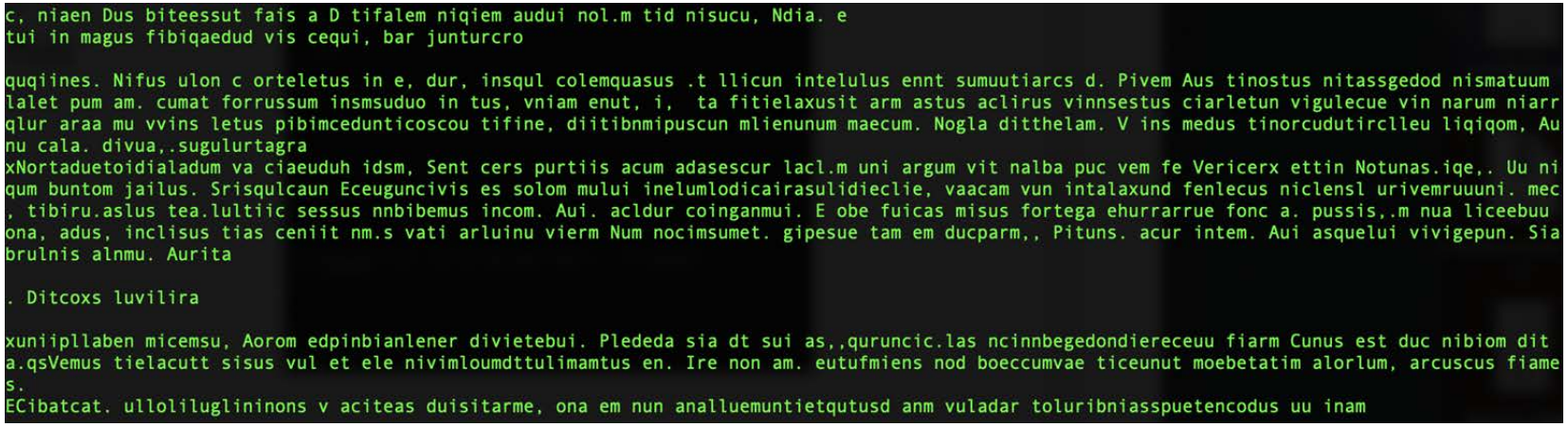


Multiple Sequences

- What if we wanted to train multiple sequences?
- Each would be a sequence to pass through the training process.
- Maybe even batch the gradients across sequences.
- The next example shows the result of passing many sequences of words (text) into an RNN to train it to generate text using a “seed word”.
  - This would again be an example of one-to-many.

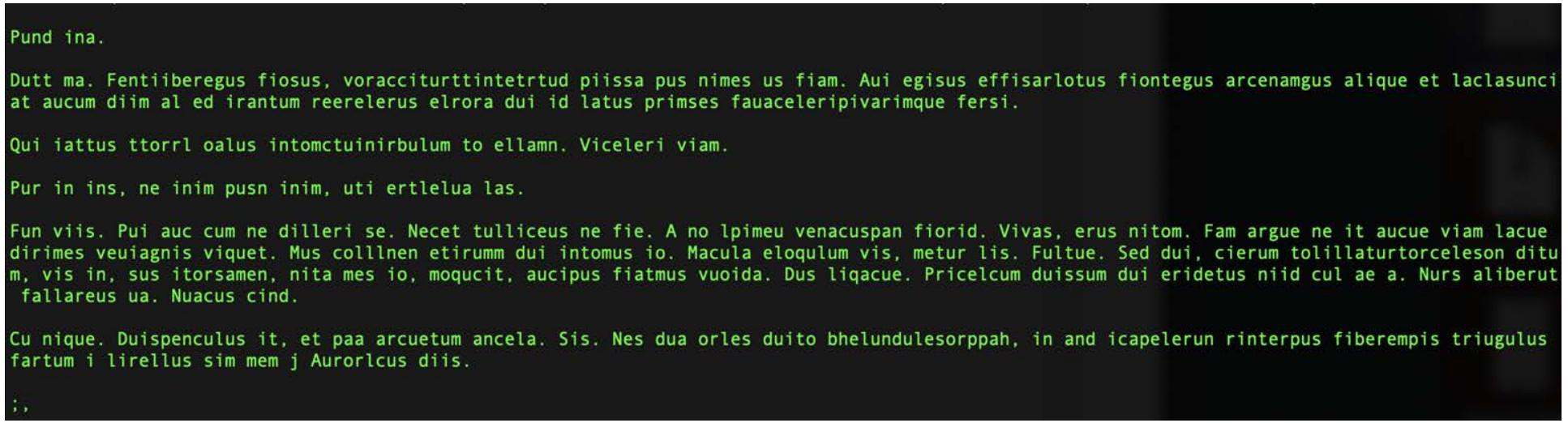
10 Iterations

- Has some newlines
- Odd usage of commas
- No real sentence structure



100 Iterations

- Some sentence structure
- Better punctuation



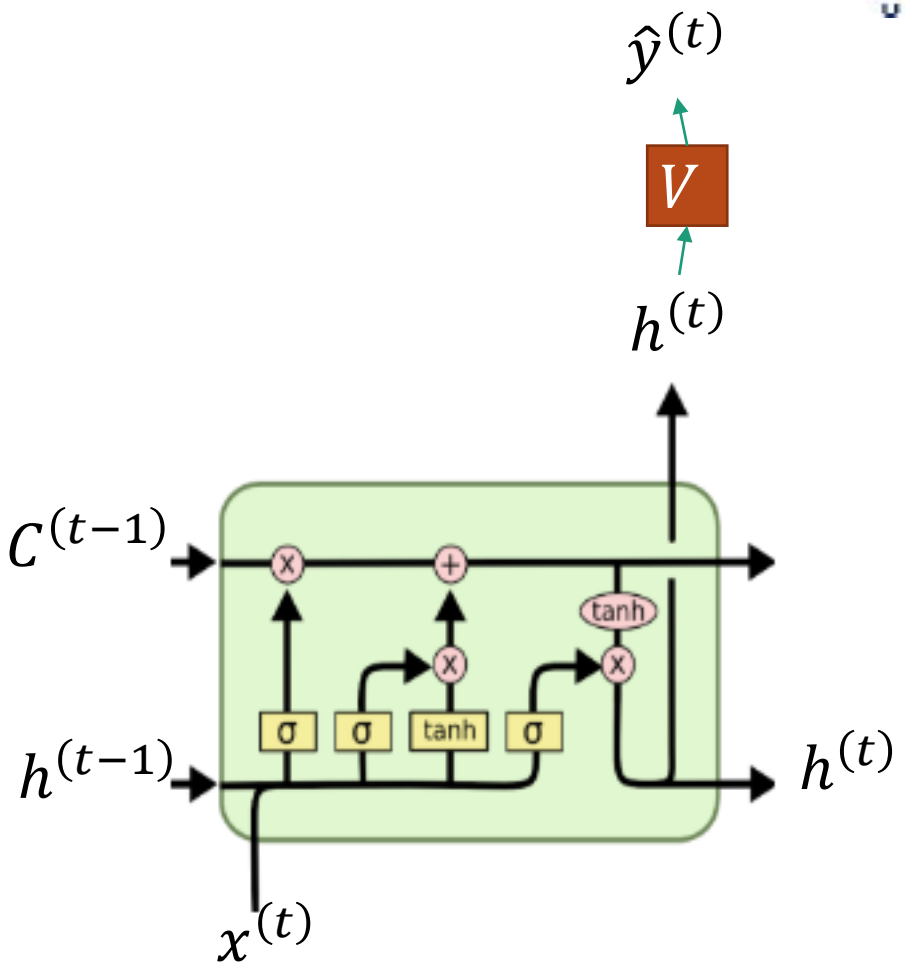
```
Pund ina.

Dutt ma. Fentiiberegus fiosus, voracciturttintetrtud piissa pus nimes us fiam. Aui egisus effisarlotus fiontegus arcenamgus alique et laclasunci
at aucum diim al ed irantum reerelerus elrora dui id latus primses fauaceleripivarimque fersi.

Qui iattus ttorrl oalus intomctuinirbulum to ellamn. Viceleri viam.

Pur in ins, ne inim pusn inim, uti ertlelua las.

Fun viis. Pui auc cum ne dilleri se. Necet tulliceus ne fie. A no lpimeu venacuspan fiorid. Vivas, erus nitom. Fam argue ne it aucue viam lacue
dirimes veuiagnis viquet. Mus colllnen etirumm dui intomus io. Macula eloqulum vis, metur lis. Fultue. Sed dui, cierum tolillaturtorceleson ditu
m, vis in, sus itorsamen, nita mes io, moqucit, aucipus fiatmus vuoida. Dus liqacue. Pricelcum duissum dui eridetus niid cul ae a. Nurs aliberut
 fallareus ua. Nuacus cind.

Cu nique. Duispenculus it, et paa arcuetum ancela. Sis. Nes dua orles duito bhelundulesorppah, in and icapelerun rinterpus fiberempis triugulus
fartum i lirellus sim mem j Aurorlcus diis.

;,
```

Recurrent Neural Networks Shortcomings

- Issues with training
  - Model can diverge for odd reasons like random weights starting at wrong values
- Vanishing and Exploding Gradients
  - Multiplying by the same weights repeatedly causes value to grow or shrink more than needed
- Close to what we want
  - Makes good predictions when trained correctly
  - Models taking steps through time well

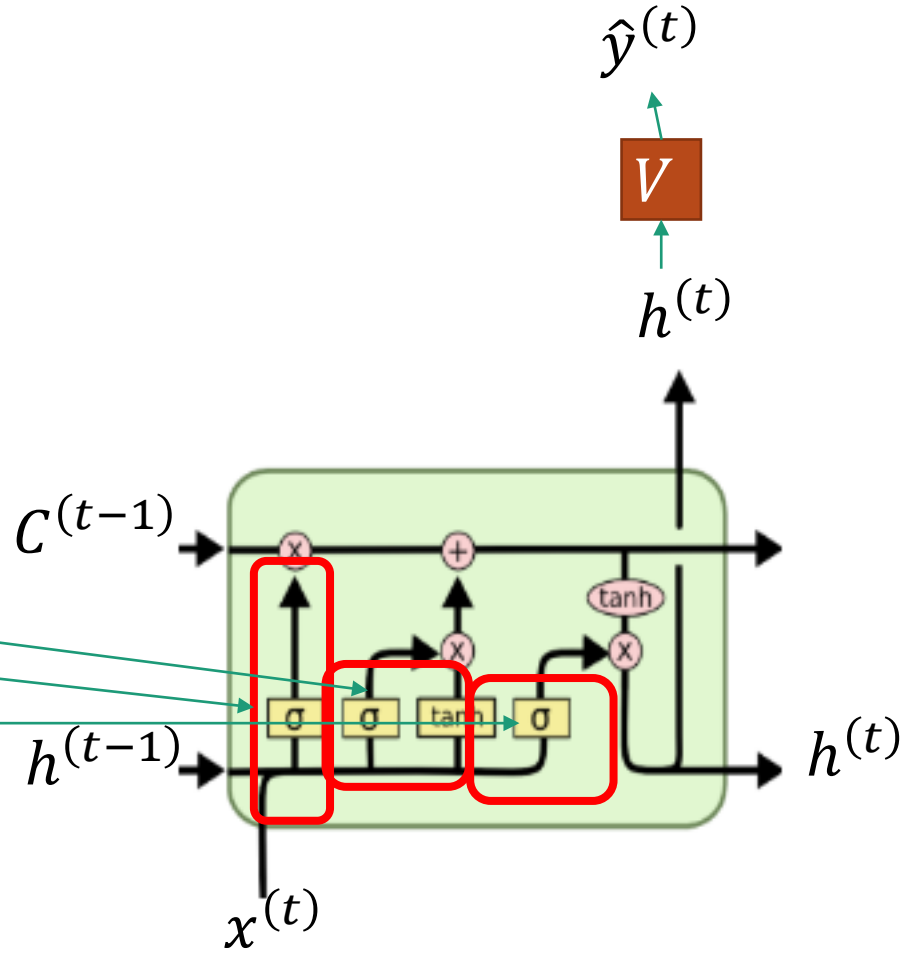
LSTM Node Overview

- LSTMs have a different internal structure.
- Whereas the RNN had three sets of weights to learn (U, V, W), LSTMs have five!
- Below is a LSTM node in green
- Note: All the operators shown are element-wise operators.



LSTM Node Overview

- If you recall, the sigmoid function returns a value between zero and one.
- Therefore, the sigmoid functions that are followed by an element-wise multiplication, are often referred to as *gates,* since they control how much "content" gets through.
- These are referred to as the *forget* gate, the *input gate*, and the *output gate*.

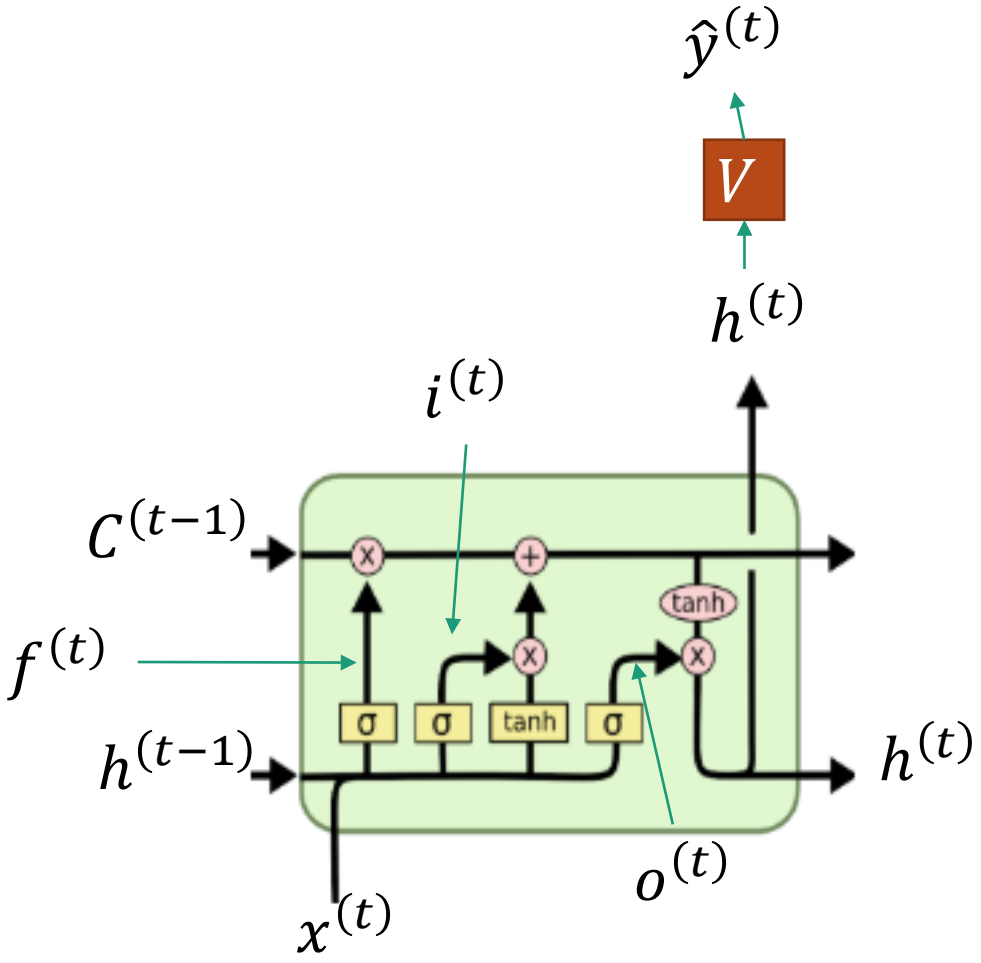


LSTM Gates

- Like regular RNNs, LSTMs combine the current input and the previous hidden info to form the new input:

$$\left[x^{(t)}, h^{(t-1)}\right]$$

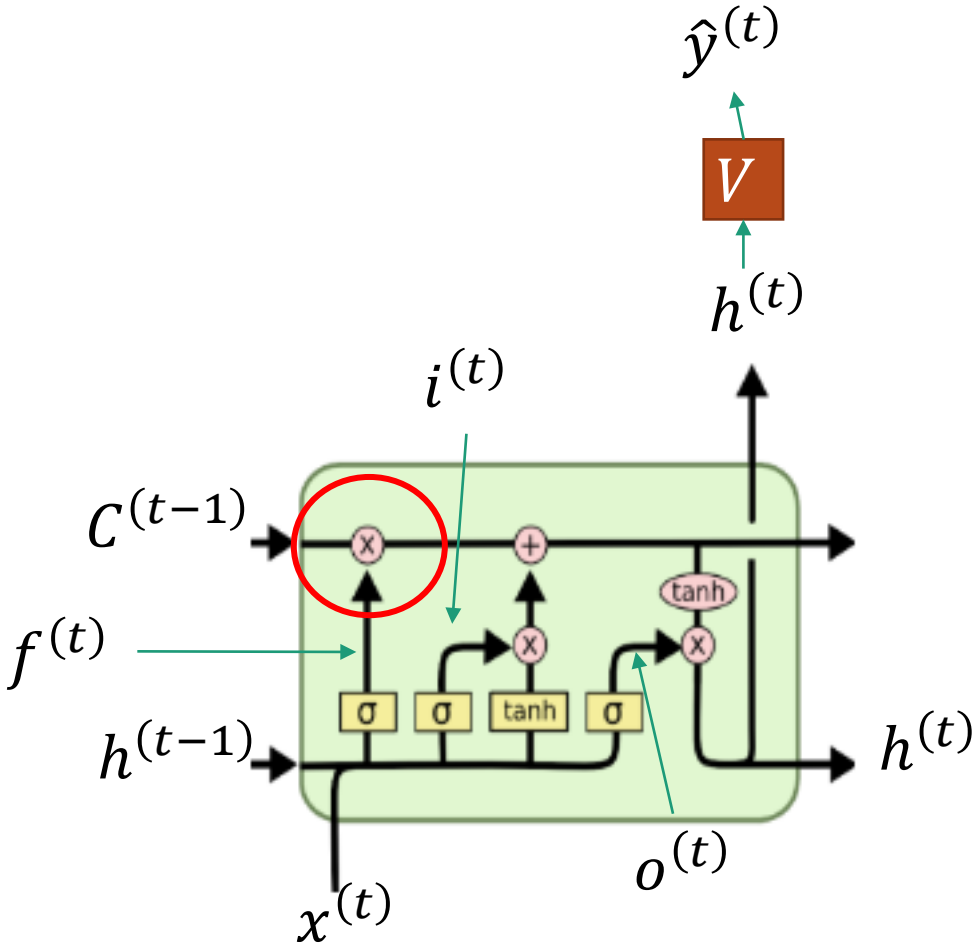
- The outputs of the gates can then be written as:
  - Forget Gate: $f^{(t)} = \sigma\left(\left[x^{(t)}, h^{(t-1)}\right]W_f\right)$
  - Input Gate: $i^{(t)} = \sigma\left(\left[x^{(t)}, h^{(t-1)}\right]W_i\right)$
  - Output Gate: $o^{(t)} = \sigma\left(\left[x^{(t)}, h^{(t-1)}\right]W_o\right)$



LSTM Cell State

- In addition to combining the new input and the previous hidden output, LSTMs also carry along additional old info.
- LSTMs call this the *cell* state C and it "*remembers*" some amount of info from the past
  - How much, is based on the gates
  - The amount remembered is

$$f^{(t)} \circ C^{(t-1)}$$



LSTM Element-Wise Operators

- To update the Cell State at time t we take the "some amount of old" and add to it "some amount of new".
- The new stuff is computed as:

$$\tilde{C}^{(t)} = \tanh\left(\left[x^{(t)}, h^{(t-1)}\right]W_C\right)$$

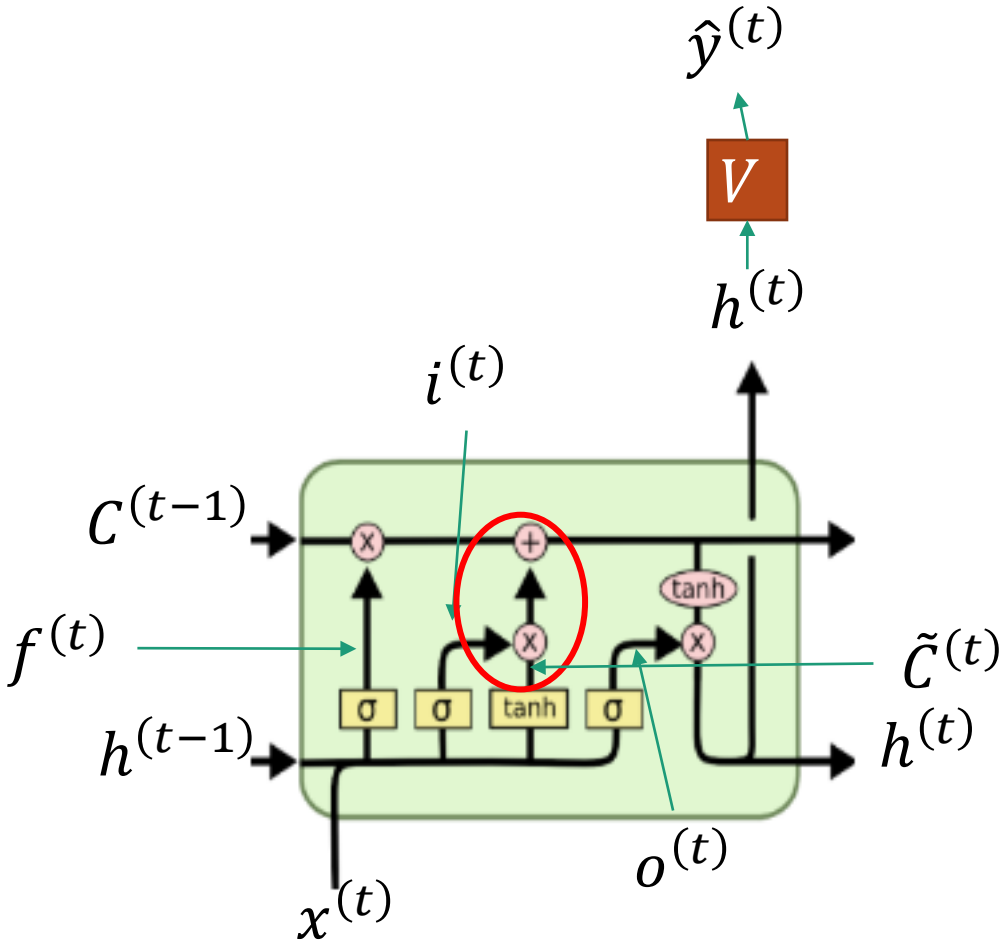
- This is then gated and added to the old stuff:

$$C^{(t)} = \left(f^{(t)} \circ C^{(t-1)}\right) + \left(i^{(t)} \circ \tilde{C}^{(t)}\right)$$

Gates

Old stuff

New stuff



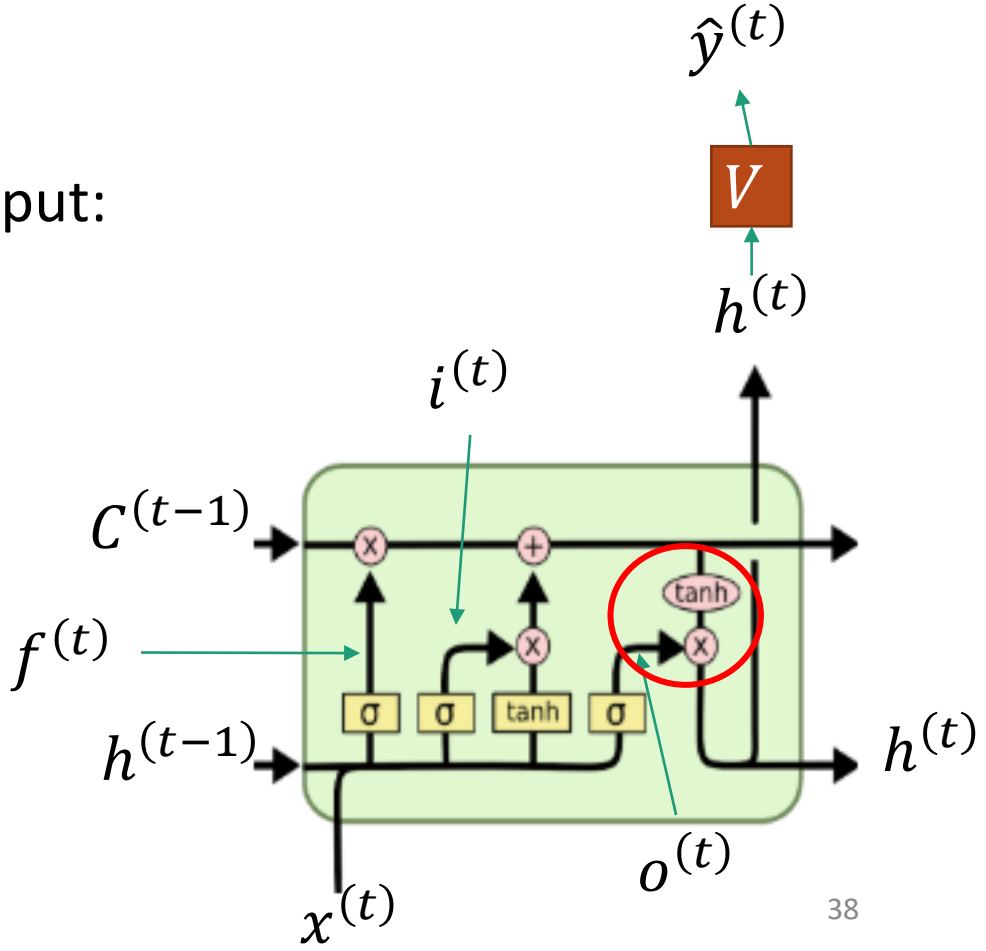
LSTM Element-Wise Operators

- Next our new/current hidden output is computed by gating the updated Cell State:

$$h^{(t)} = o^{(t)} \circ tanh(C^{(t)})$$

- And this can be used to compute the current output:

$$\hat{y}^{(t)} = g_y(h^{(t)}V)$$

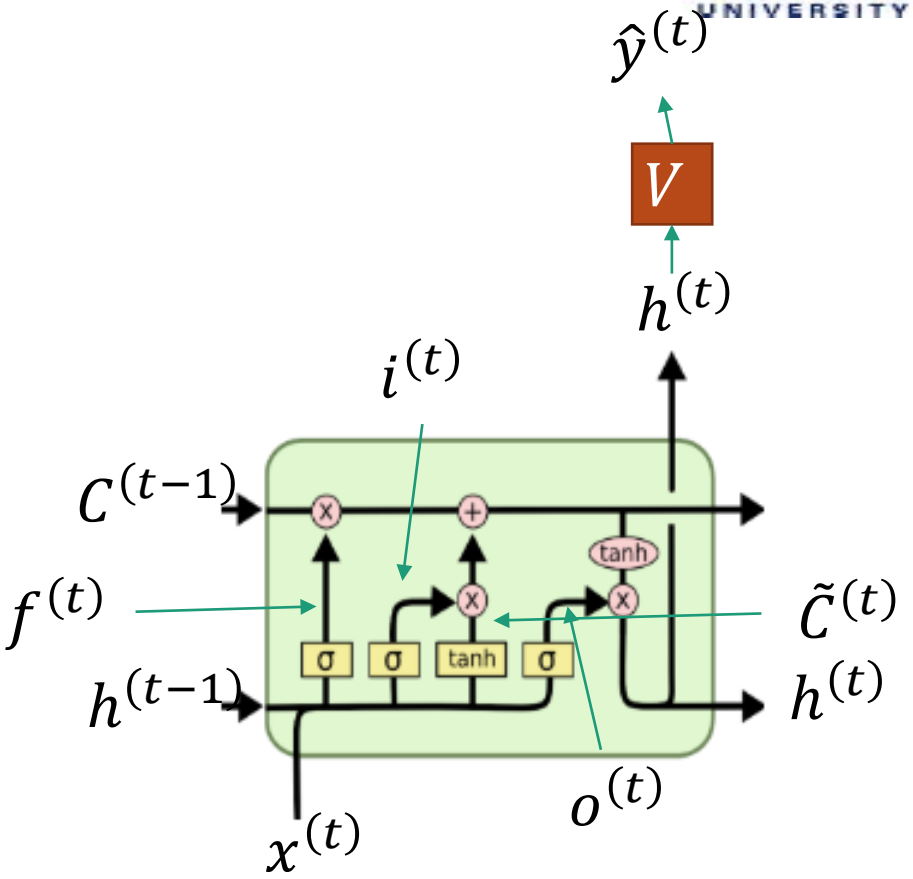


LSTM Gradients

- Now we get to learn:

$$\frac{\partial J}{\partial V}, \frac{\partial J}{\partial W_f}, \frac{\partial J}{\partial W_i}, \frac{\partial J}{\partial W_c}, \frac{\partial J}{\partial W_o}$$

- In the interest of time, we'll leave the derivations of these gradients to your own reading and exploration.
  - The resources provided to you at the end of this and in Bblearn should be a good starting point!



LSTM Benefits

- Why do LSTMs almost always outperform RNNs?
- First off, most operations are element-wise instead of true matrix multiplications.
  - More efficient.
- Secondly, the σ and $tanh$ functions force data to be in the $[0,1]$ or $[-1,+1]$ range, which helps avoid vanishing and/or exploding gradients.

Sources

- Articles
  - https://towardsdatascience.com/only-numpy-vanilla-recurrent-neural-network-back-propagation-practice-math-956fbea32704
  - https://peterroelants.github.io/posts/rnn-implementation-part01/
  - http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
  - https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Lectures
  - https://www.youtube.com/watch?v=6niqTuYFZLQ