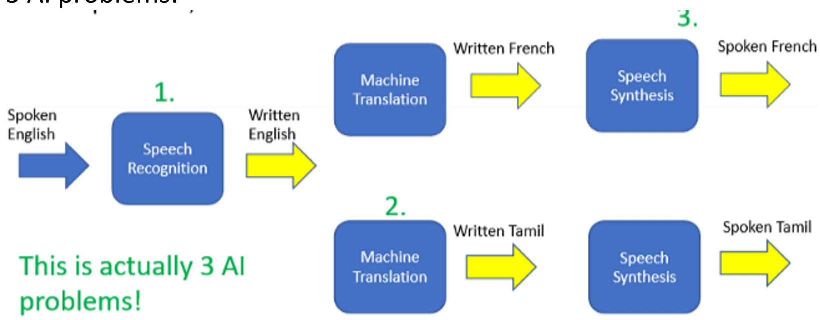
RNN and Natural Language Processing

Tuesday, December 14, 2021 6:52 PM

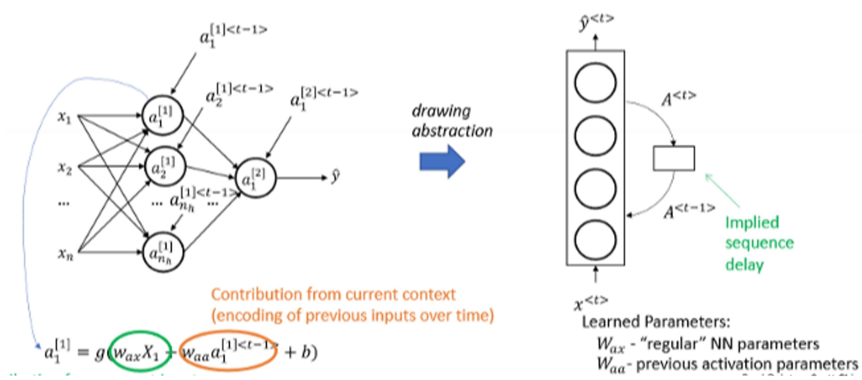
21. Recurrent Neural Networks

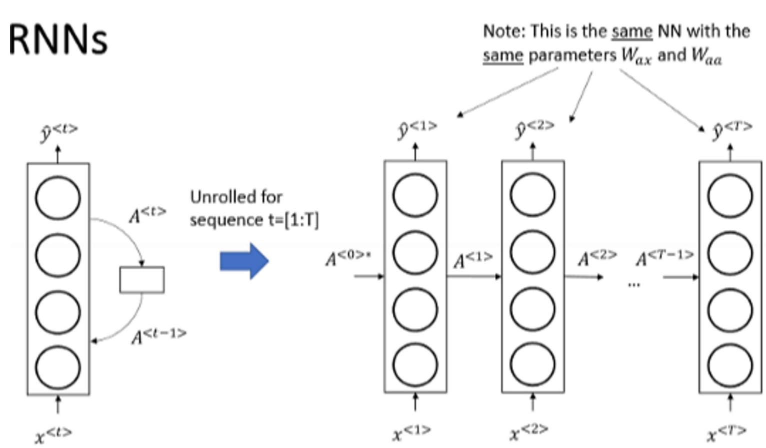
Translation

- 3 AI problems:

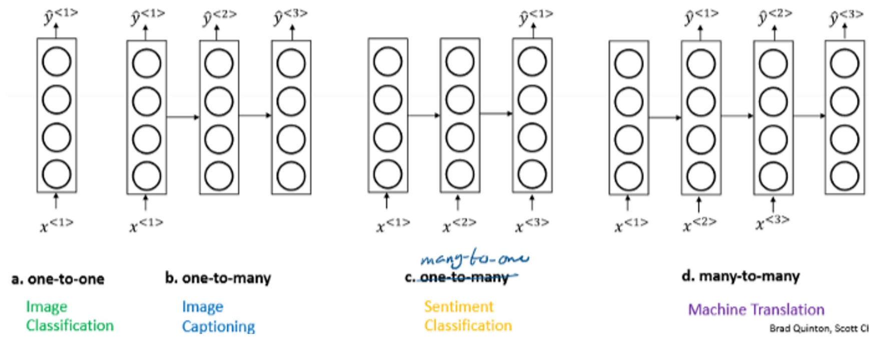


- information in individual components of data and their ordering
- need to consider the context!
  - eg. Michael Jordan made a basket
- **Giving ML Context**
  - understand new info
  - increasing inputs to system to reflect context (data from the past) doesn't scale!
  - activations from previous step in sequence can be used to "bias" activations in next step
    - could simultaneously learn amount of context required while we learn input to output mappings

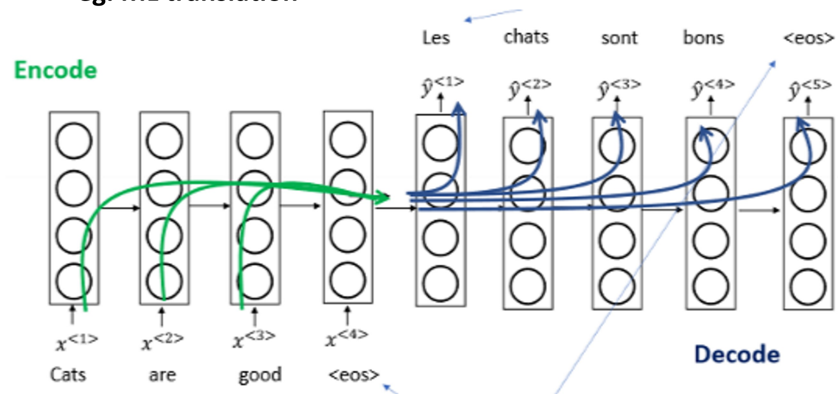




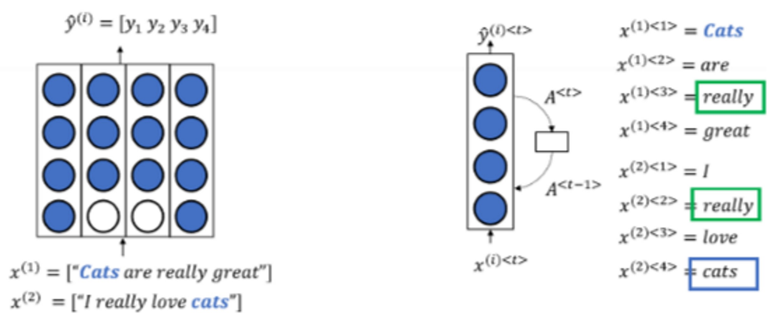
- **I/O Sequence Length Flexibility in RNNs**
  - Flexibility around size of input/output data



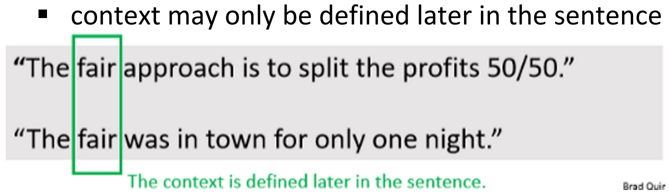
- **eg. ML translation**



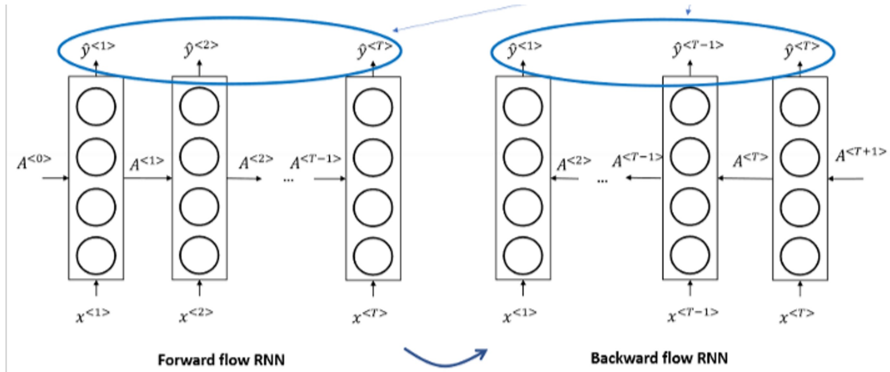
- **RNN Feature Extraction**

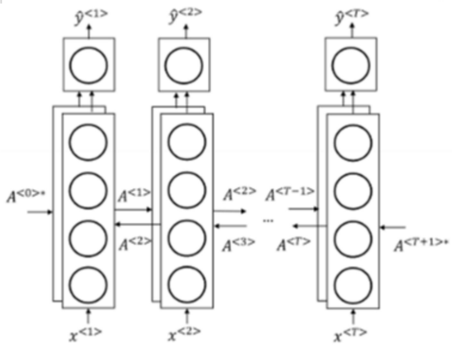


        - eg. neuron learned to activate strongly to patterns that make an 11x11 swatch of facial hair
    - RNNs isolate elements of sequences like conv filters isolate regions of an image
- **Bidirectional RNN**
    - can look forward and backward in time
    - buffer the inputs long enough to consider context in 2 directions
        - context may only be defined later in the sentence



    - create forward flow RNN and backward flow RNN then combine outputs
    - using BRNNs for each sentence = state of the art for NLP





23. Implementation of RNNs

RNN Notation

- **Inputs:** $x^{(i)<t>}$ $where\ i = 1{:}m\ and\ t = 1{:}T_x^{(i)}$
- **Outputs:** $y^{(i)<t>}$ $\underbrace{where\ i = 1{:}m}$ $and\ t = 1{:}T_y^{(i)}$

Now, each input and output in the training example has a sequence length T.

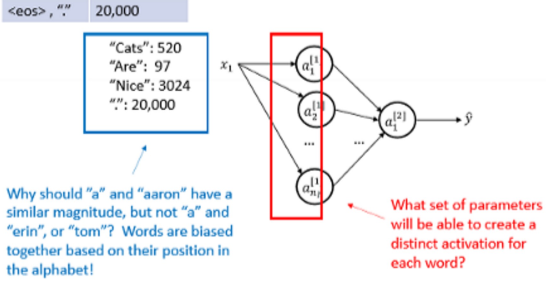
Like before, we have *m* training examples.

- eg. for x: "cats are nice", want to make each word an element of our sequence

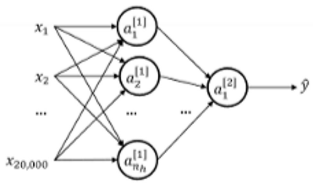
  $x^{<1>} = \text{“Cats”}, x^{<2>} = \text{“are”}, x^{<3>} = \text{“nice”}, x^{<4>} = \text{“.”}$

- need to assign each word a number
- can create an ordered dictionary, assign each word a num based on position in sequence
  - may add unintential bias, words are biased based on position in alphabet

Word	Position
a	1
aaron	2
...	
cats	520
....	
nice	3024
...	
zulu	19,999
<eos> , “.”	20,000



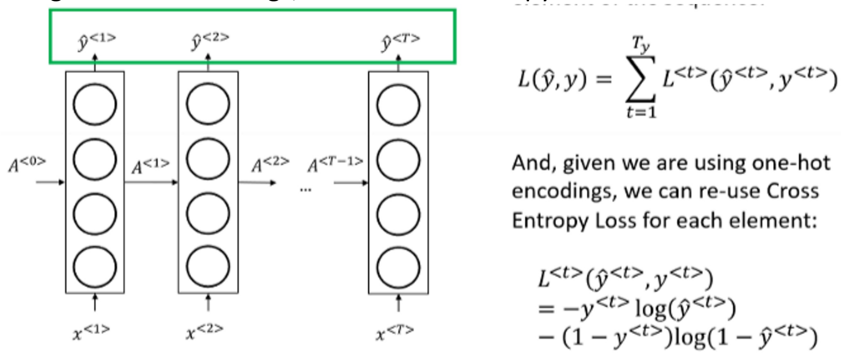
- need normalized representation and less compressed
  - instead use one-hot encoding, no order bias



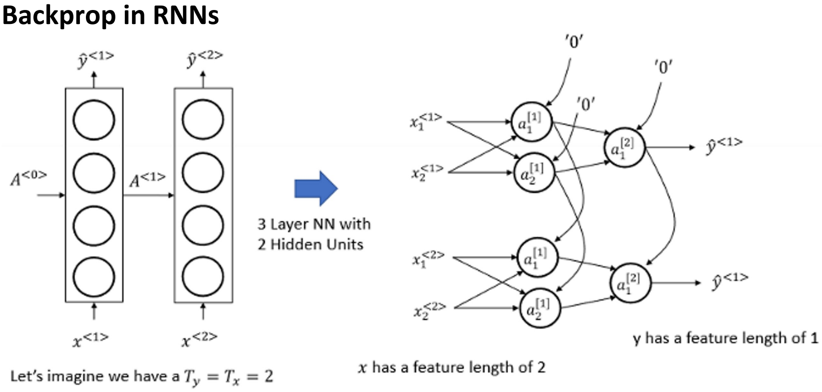
- **Unknown words**
  - one more vector element called unknown word ("UKW")
  - works as long as all important words NP task uses are included in vocabulary

RNN Loss Function

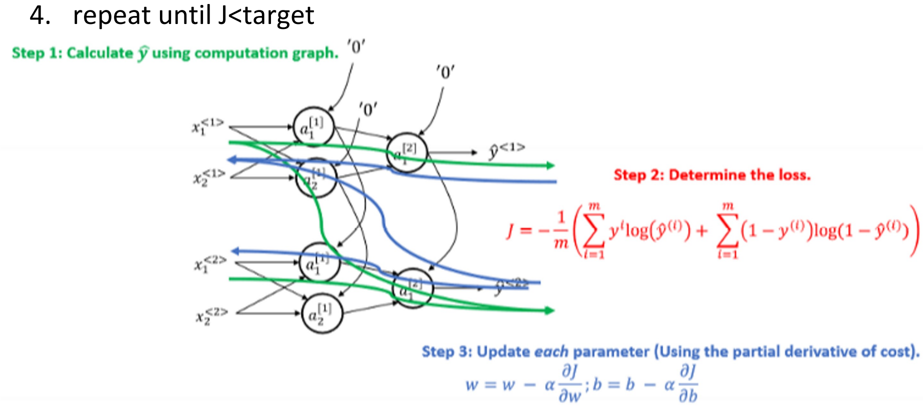
- define loss function to optimize later
- define overall loss to be sum of each element of sequence
- using one-hot encodings, re-use Cross Entropy Loss for each element



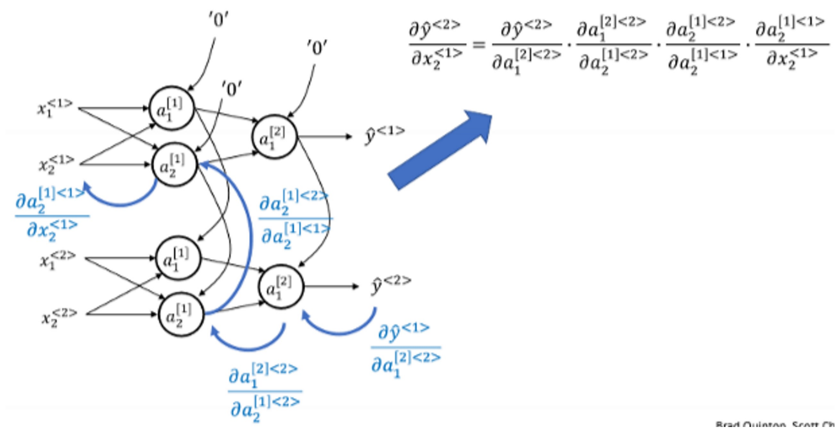
- **Backprop in RNNs**



1. calculate y_hat using computation graph
2. determine the loss
3. update each parameter (using partial derivative of cost)
4. repeat until J<target



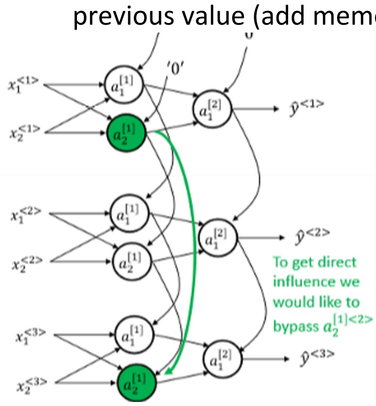
- **Partial Derivatives on RNN**



- **Vanishing gradients in RNNs**
  - as sequences get long, can be difficult to enable earlier elements to correctly influence later outputs
  - influence of early elements of the sequences are "washed out" by repeat multiply accumulates in each iteration
  - activation unit for RNN:

$$a_1^{[1]<t>} = g(w_{ax}X_1 + w_{aa}a_1^{[1]<t-1>} + b)$$

  - repeated applying same RNN activation units. To bypass current activation, we hold previous value (add memory to RNN)



- **Simplified Gated Recurrent Unit (GRU)**
  - get a value between 0 and 1 based on learned param and standard RNN unit inputs

$$\Gamma_\mu = \sigma(w_{\mu x}X_1 + w_{\mu a}a_1^{[1]<t-1>} + b_\mu)$$

  - Use function to decide whether keep previous activation or update

$$a_1^{[1]<t>} = \Gamma_\mu * \tilde{a}_1^{[1]<t>} + (1 - \Gamma_\mu) * a_1^{[1]<t-1>}$$

  - standard activation calculation now a candidate:

$$\tilde{a}_1^{[1]<t>} = g(w_{ax}X_1 + w_{aa}a_1^{[1]<t-1>} + b)$$

    - similar to creating a mux function
- **Long Short Term Memory (LSTM)**
  - maintian influence of value across many sequences
  - manage vanishing gradient problem in RNN
  - create new var c, expression of internal memory
  - gating: update, forget, output

$$\Gamma_\mu = \sigma(w_{\mu x}X_1 + w_{\mu a}a_1^{[1]<t-1>} + b_\mu) \qquad \text{Update}$$

$$\Gamma_f = \sigma(w_{fx}X_1 + w_{fa}a_1^{[1]<t-1>} + b_f) \qquad \text{Forget}$$

$$\Gamma_o = \sigma(w_{ox}X_1 + w_{oa}a_1^{[1]<t-1>} + b_o) \qquad \text{Output}$$

- use funcs to manage internal memory and output vals
- eg. create candidate memory:

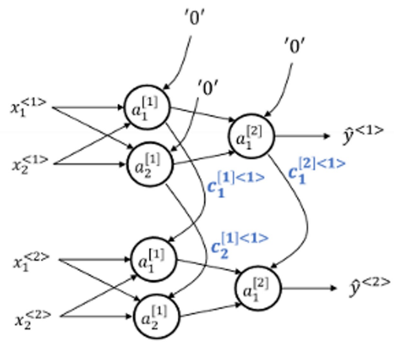
$$\tilde{c}_1^{[1]<t>} = g(w_{ax}X_1 + w_{aa}a_1^{[1]<t-1>} + b)$$

  - update internal memory, update and forget

$$c_1^{[1]<t>} = \Gamma_\mu * \tilde{c}_1^{[1]<t>} + \Gamma_f * c_1^{[1]<t-1>}$$

  - create output separate from internal memory

$$a_1^{[1]<t>} = \Gamma_o * \tanh(c_1^{[1]<t>})$$



- structure of many sequential data sets, key element critical for period of time, then no longer relevant

24. Natural Language Processing (NLP)