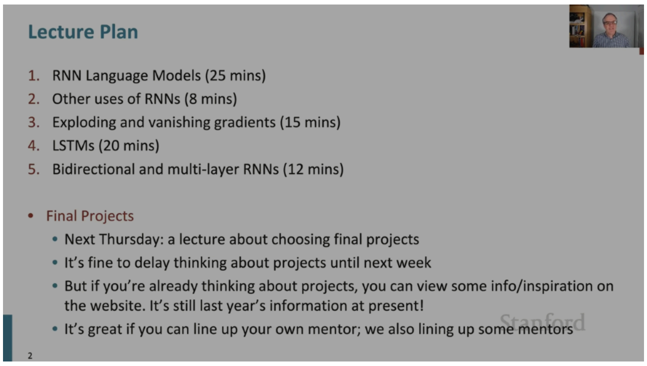
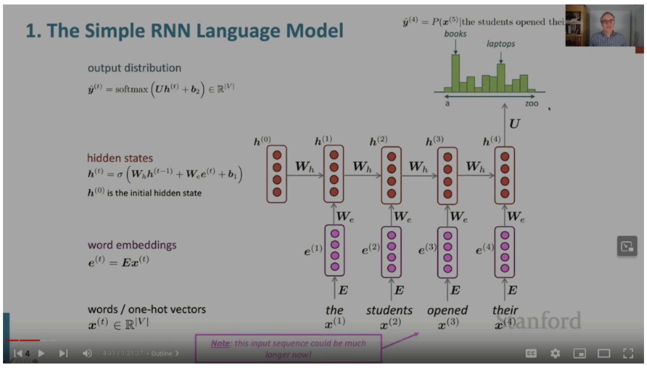
Notes

26 March 2024 07:40

Lecture Outline:

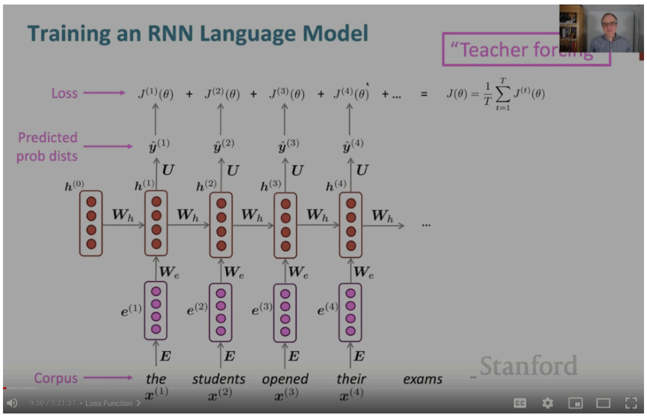


Simple RNN-based Language Model: Recap



Training an RNN-based LM:

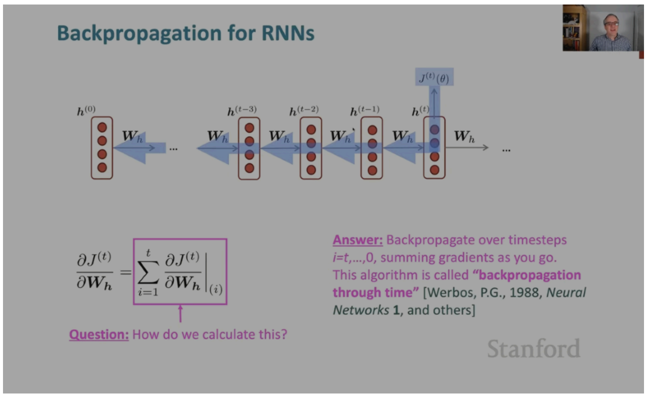
- Dataset: A big corpus of words
- Model: RNN
    - Feed a set of words (sentence if you will) to the model
    - Model predicts probability distribution (of next word) over the corpus
- Loss function:
    - Cross entropy between GT prob of next word and Pred prob at each time step
    - Total loss in one iteration: Avg. Of loss at each time step



- Teacher Forcing:
    - Calculating loss at each time step allows model to predict only the next word.

- Another way could be: Give model couple of words as input e.g. "the students" and let the model finish the sentence. The loss can be calculated over the sentence vs only one word.

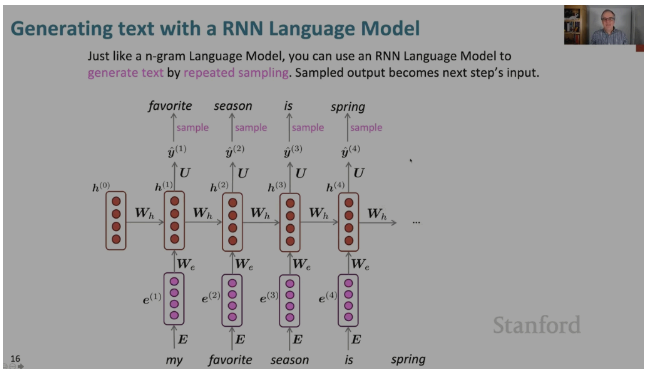
- Backpropagation and Gradient for RNNs:
  - (if it is not clear until now) There is only one/shared weight matrix W_h.
  - However (downstream)gradient at each timestep t will be different as upstream (gradient) is different.
  - What to do? Sum all the gradients at all timesteps and apply all of them at once (Do not chain the updates as during forward pass each W_h was used as a separate matrix)



Other Uses of RNNs:

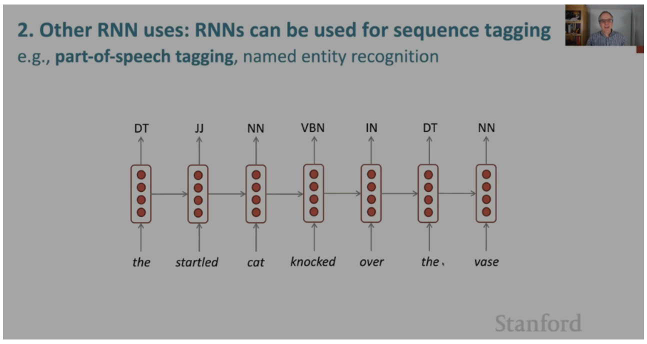
Text Generation:

- Notice how output at time t is being fed back as input of time t+1.



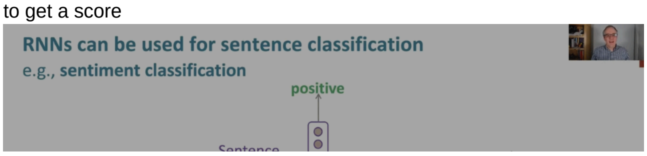
Sequence Tagging:

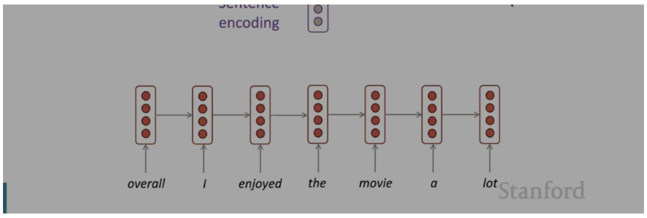
- Assigning Part of speech to each word in the sentence



Sentiment Analysis:

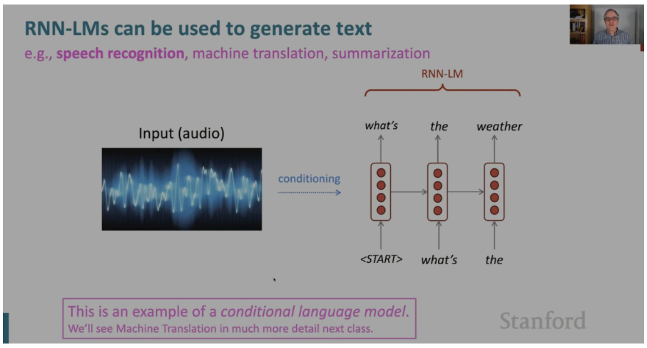
- Using RNN to get encoding of sentence and use a classifier layer to get a score





Speech-to-Text/Text Generation:

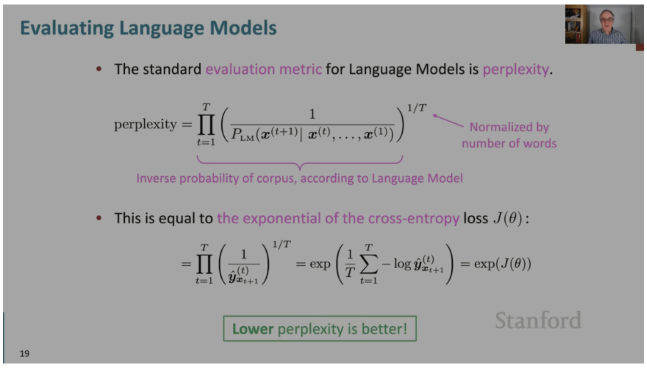
- Using a NN as conditioning layer (to transform audio signal into embedding vectors)



Evaluating RNN LM:

Perplexity:

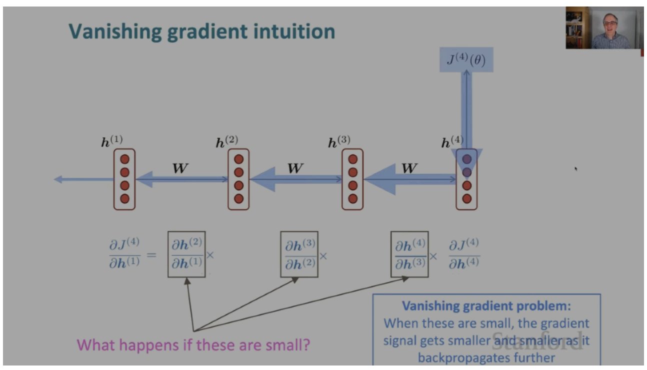
- Standard eval metric.
- Geometric representation of inverse probability.
- Indicates how well the model predicted the next word? In Ideal case, the model should have perplexity of 1.
- It is a unitless measure. Ranging from 1 to positive inf.
- e.g. perplexity value of 53 sort of means that model is choosing the next word from 53 near equiprobable words (out of all output prob. Values 53 values are very close to each other) -- representing model's uncertainty.



Gradient Problems: Vanishing and Exploding

Vanishing Gradients:

- No update at all to the model weights.



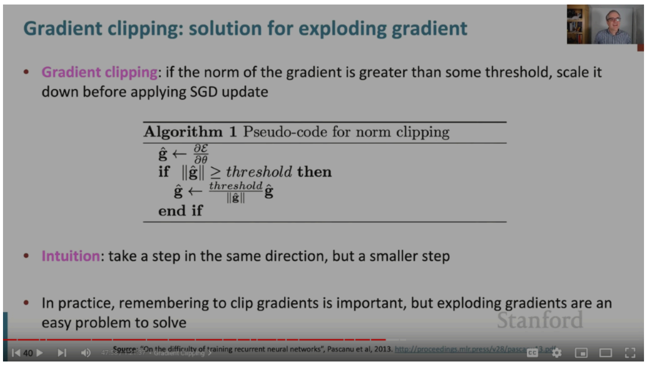
- Why the gradients vanish?
  - Visible reason – if the partial derivatives of hidden stages are small enough in magnitude, the upstream gradient becomes small and small as it gets propagated.
  - Architectural Reasons(not talked in the lecture):
    - Reuse of weights: during initialization/training if weights are small enough then upstream gradients will become smaller and smaller at each time step
    - Choice of Activation function: Commonly used tanh/sigmoid have saturation regions where the gradient is very small.
  - What is too small gradients/weights?
    - If eigen values of gradients/weights matrix are < 1 then it can be considered as too small

- Why is this a problem?
  - Model is rendered incapable of understanding long-term dependencies – as upstream gradient vanishes, timesteps in the beginning receive almost zero gradients and then the weights are not modified for the errors in those time steps.

Exploding Gradient:

- Weight updates becoming too big

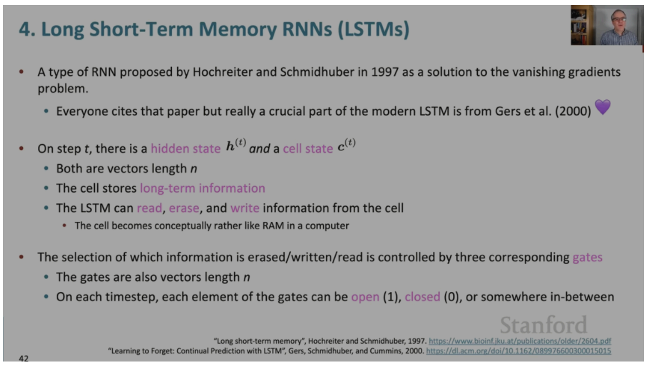
- Solution: Make gradients manageable (not too big) -- Gradient Clipping



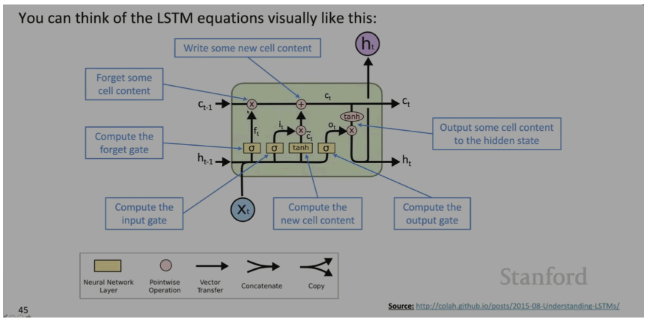
LSTM:

- As hidden states are almost completely overwritten at each time step --
  - RNNs not able to retain information for long
  - Also contributes to vanishing gradients – frequent changes might zero out some elements of hidden states

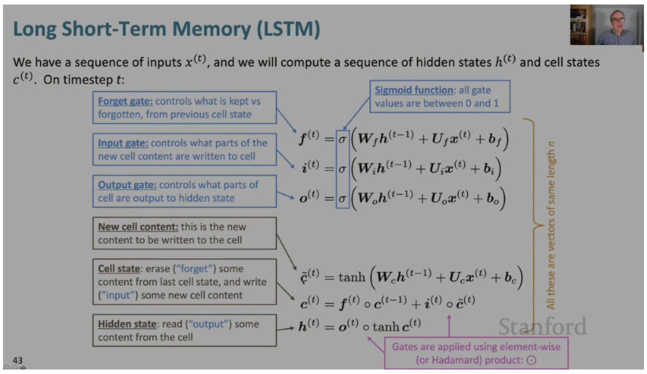
- How about adding some memory to RNNs so that hidden states can be changed selectively?



- Graphical representation of LSTM Block :



- Mathematical representation of LSTM Block

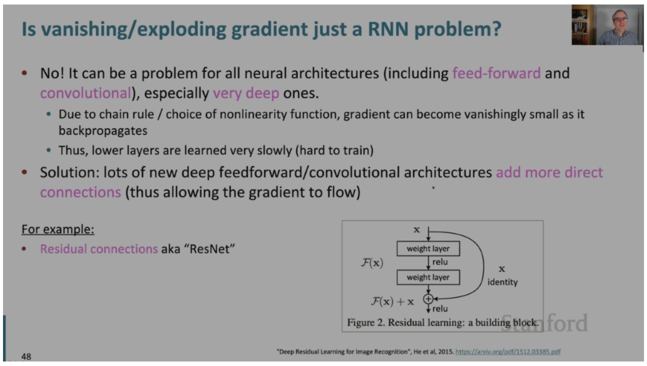


- Key difference to simple RNNs:
  - In LSTM: Not all is changed (added/forgotten) in each iteration
  - When new information is added to hidden states (in simple RNN) it is added by multiplication vs addition when new information is added to Cell states in (LSTMs)
  - In RNNs – multiplication carries out drastic changes vs in LSTMs addition carries out gradual changes
  - This makes retaining information lot easier for LSTMs

- How LSTMs help to control/reduce vanishing gradients problem?
  - By controlling rate of change of information in cell states, LSTM create a longer term version of short term memory
  - This prevents frequent changes to hidden states allowing upstream gradient to flow for until timestamp 0 for longer duration of time

Vanishing/Exploding Gradients in general DNN:

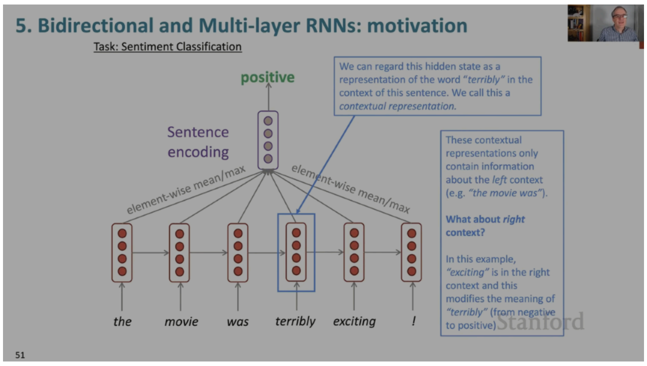
- Not only RNN specific. Every Deep Network suffers from such issues due to chaining the upstream gradient for longer depths
- Solution:



Bi-directional and Multi-layer RNNs:

- Adding bi-directional context to RNNs: not only left context (from words on the left/previous words) but also right context (from words on the right of the current word)
- Only applicable when you have access to entire sentence e.g. when in sentiment analysis. Not applicable in e.g. language

modelling



- How?? -- Add a second backward RNN and concatenate both RNN's hidden states