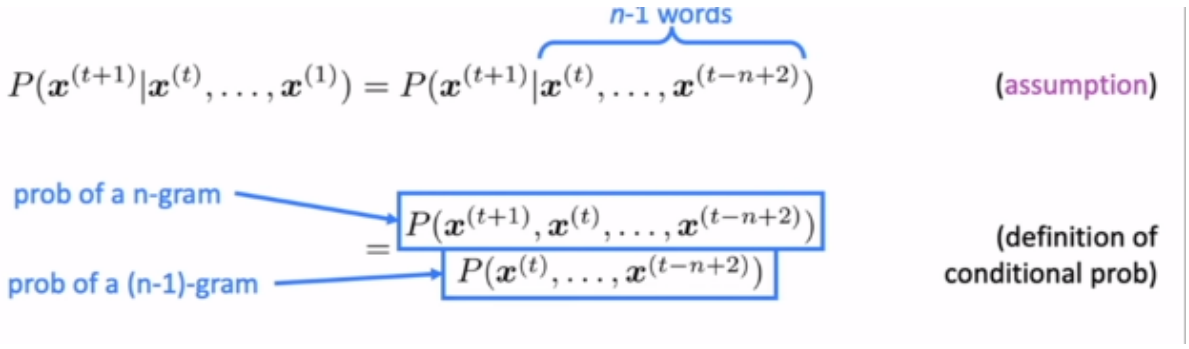
Language modelling:

It is the process of predicting what word comes next given the previous words.

N-gram language modeling:

- **Definition:** A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: "the", "students", "opened", "their"
  - bigrams: "the students", "students opened", "opened their"
  - trigrams: "the students opened", "students opened their"
  - 4-grams: "the students opened their"
- **Idea:** Collect statistics about how frequent different n-grams are, and use these to predict next word.

Probability:

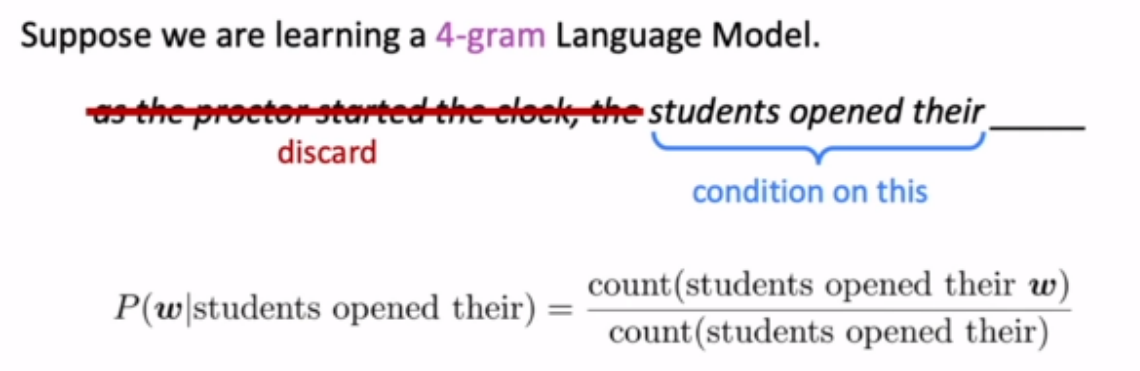


But how do we do this?

By just counting the number of times the above n-gram / (n-1)-gram has appeared in the corpus.

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

Example:



Sparsity problem 1:

The problem is, for example we want to know the probability of the case with w being petridish in the above example, if we don't have that case in our corpus the assigned probability will be zero but it is a perfectly valid case.

So what we do is we assign a small probability to all the words, this is called **smoothing,** this basically converts the graph from being a constant zero for most words with random spikes, to something more smooth.

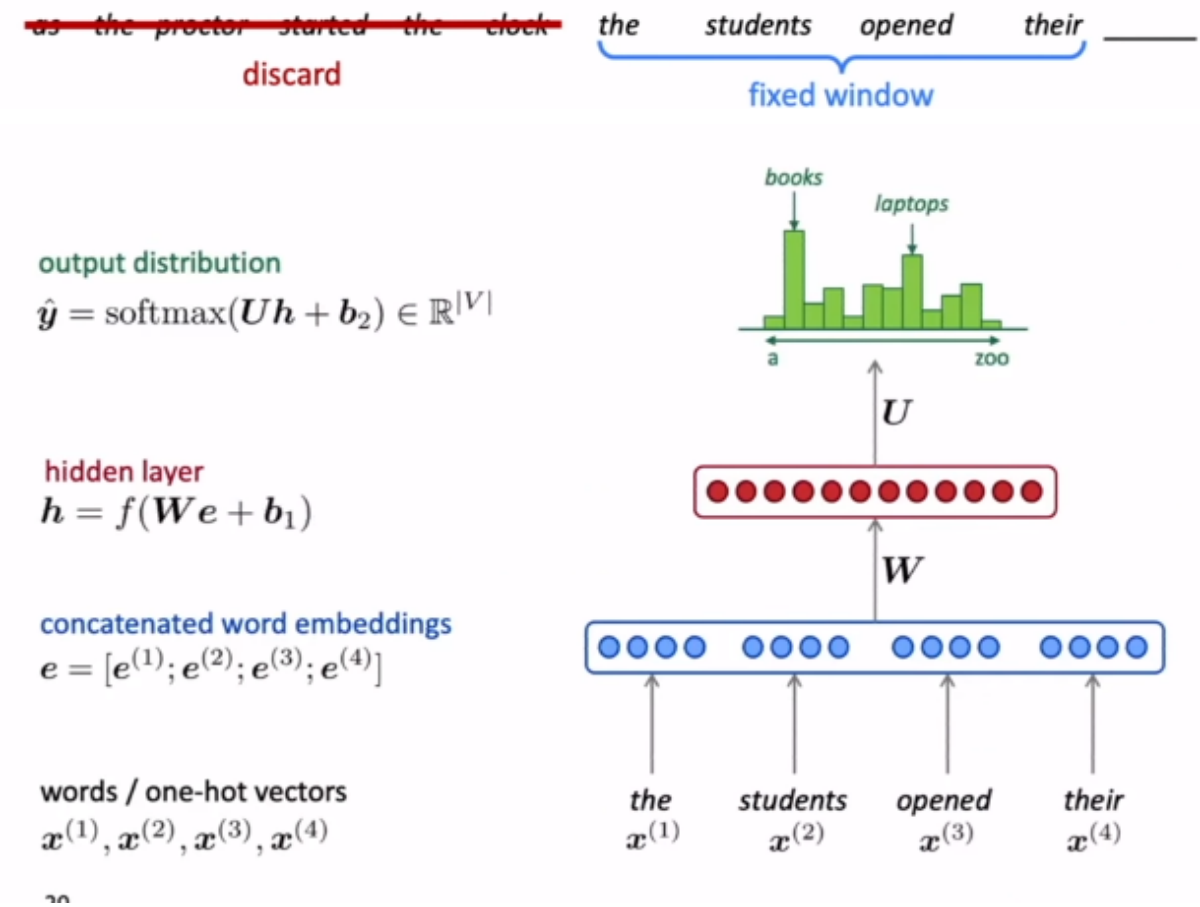
Sparsity problem 2:

What if we don't even have "students opened their" in the corpus? Then we just calculate the probability with "opened their", This is called **backoff.**

Note: The sparsity problem increases with the increase in the n of the n gram model, so any n more than 5 usually doesn't work.

Making it neural:

Just use a window based model as seen previously!



What are the advantages?

- It removes the sparsity problem!
- We don't have to store all the observed n-grams, we just have to store the word vectors and the nn.

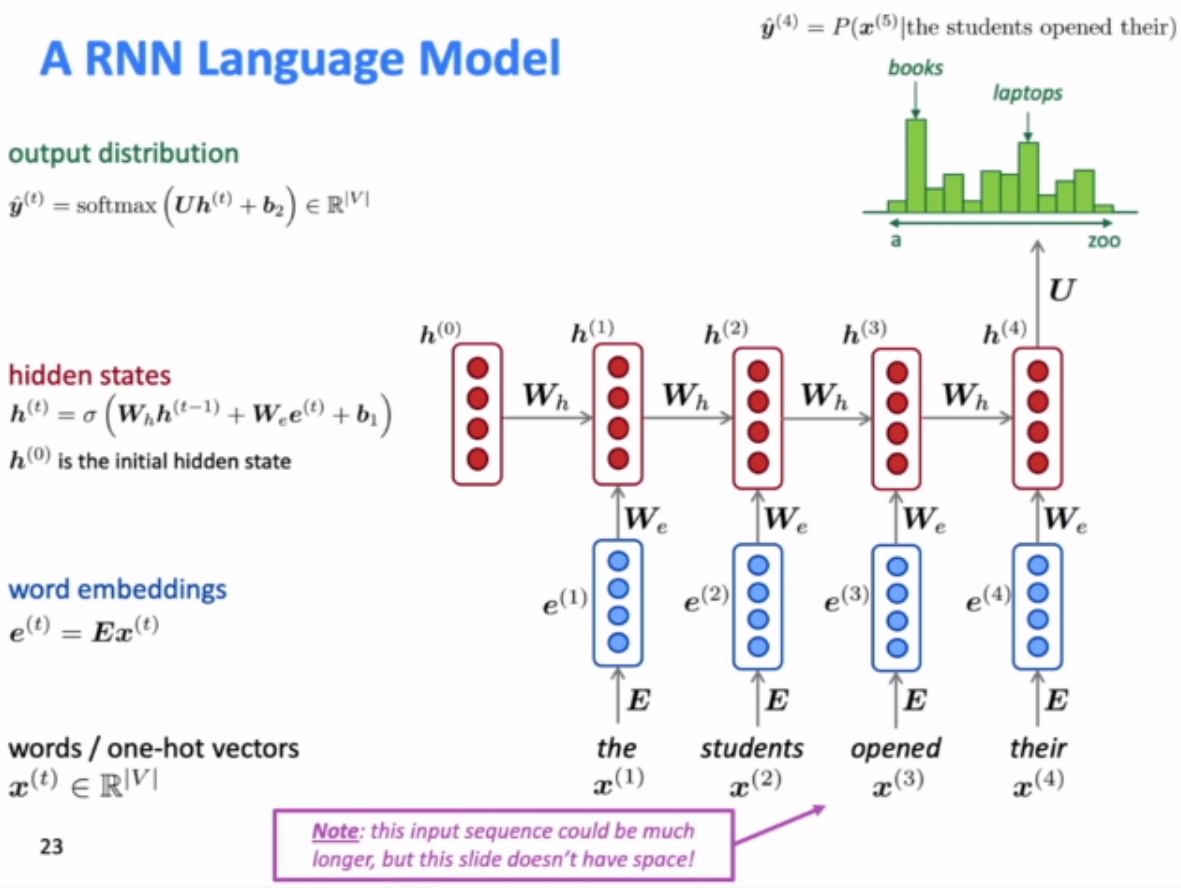
What are the negatives that still remain?

- The word widow is still probably too small.
- There is no symmetry in how the input words are processed. The weights multipied with x1 is totally independent of weights multiplied with w2. We are learning weights for all these words separately.

This is why we learn about recurrent neural networks !

Recurrent Neural Networks.

H_0 can be a vector that we learn as a parameter of the RNN language model or just a zero vector.



Embeddings are basically just the word vectors, we can just download it from the internet and make it fixed or we can also download it and use it as an initial state and fine tune it.

The U is putting the last hidden state through a linear layer and trying to find the final probabilities of all the remaining words.

RNN advantages:

- Can process any length input
- In theory model uses words from long back to determine next word.
- Model sizes don't increase with increasing size of words, the model is only the weights Wh, We and the bias b1 (can also include the word vector as part of the model).
- Same weights is applied on all steps. There is symmetry in processing words.

RNN disadvantages:

- Recurrent computations are slow.
- In practice words far behind diminishes in value due to adding biases and putting it through nonlinear functions and adding bias again…

Training RNN:

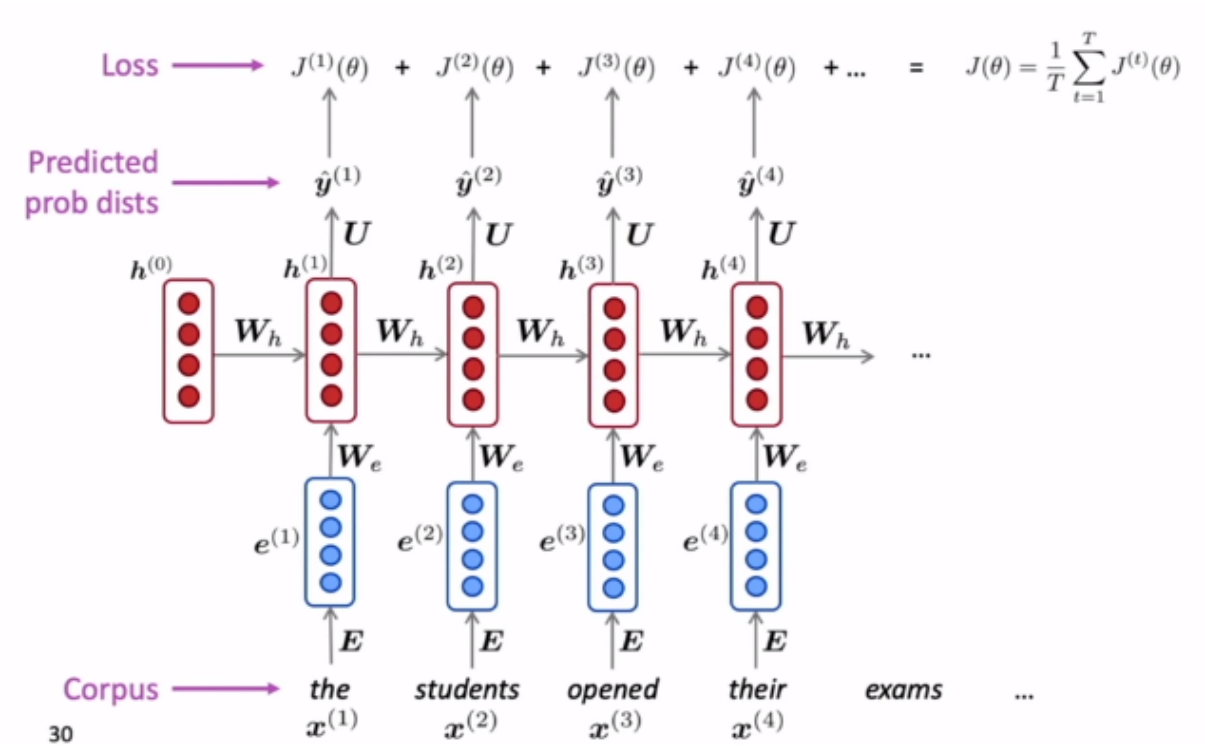
- Get a big corpus of text which is a sequence of words $x^{(1)}, \ldots, x^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for *every step t*.
  - i.e. predict probability dist of *every word*, given words so far
- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

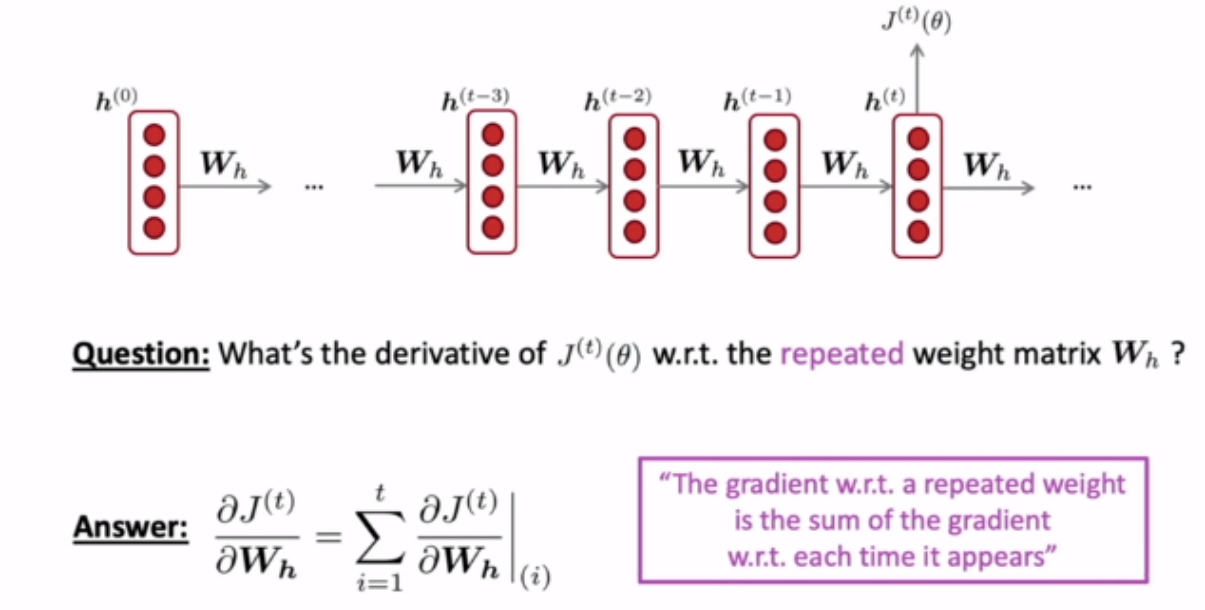
$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

The same on an image:



Note: But finding the cross entropy loss over the whole corpus will be costly right ? Yes, it is so what we do is take a sentence of Xs and do stochastic gradient descent (use only a random part of the corpus).

Backpropagation for RNNs:

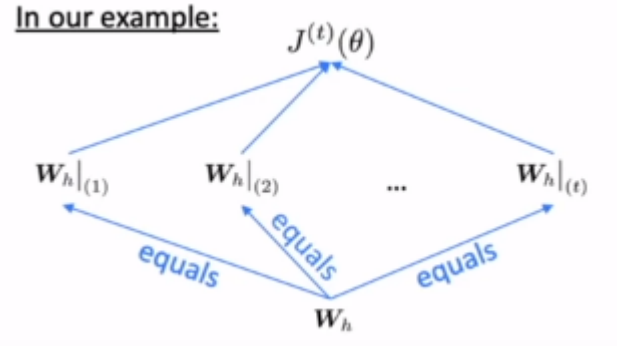


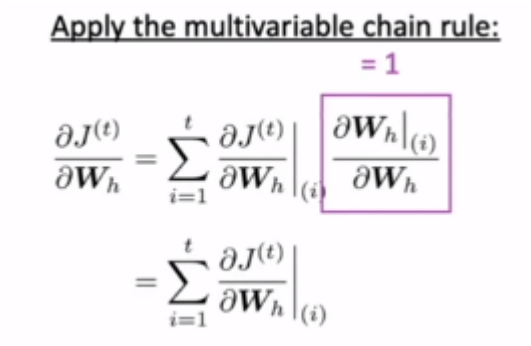
Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix W_h ?

Answer: $\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$

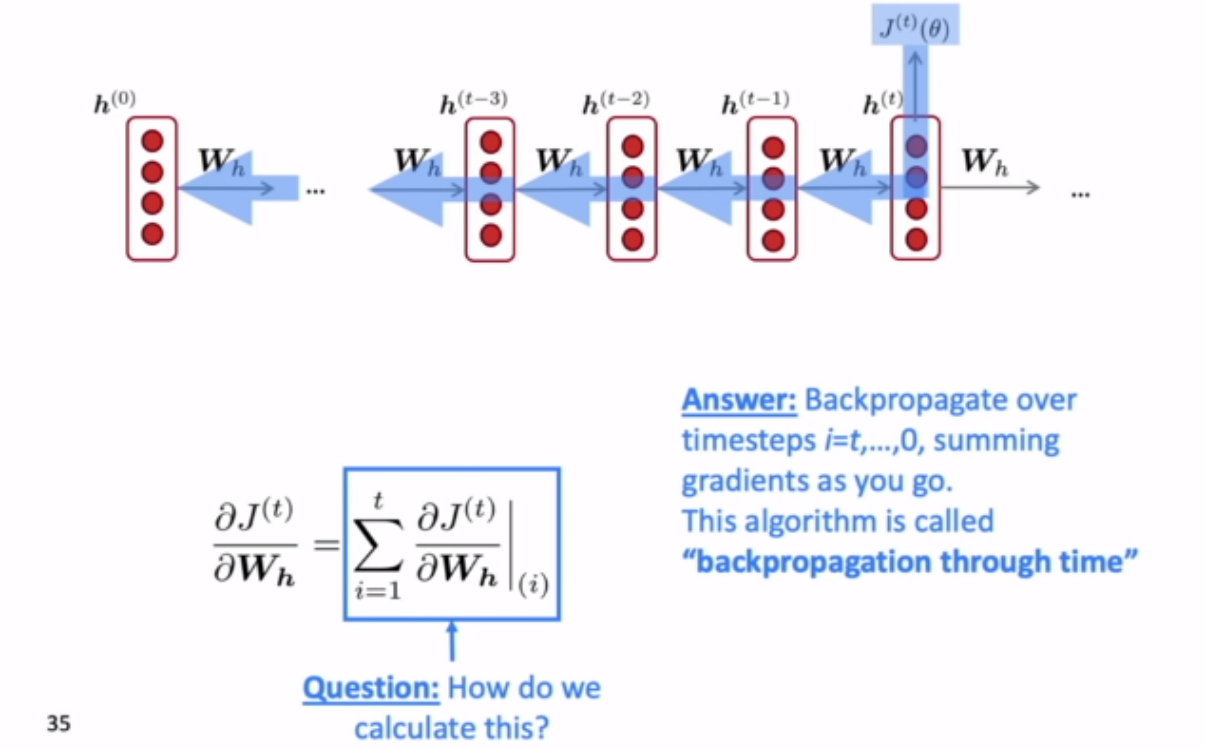
"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

Basically,





Now, lets see how we do backprop to achieve the above equation:

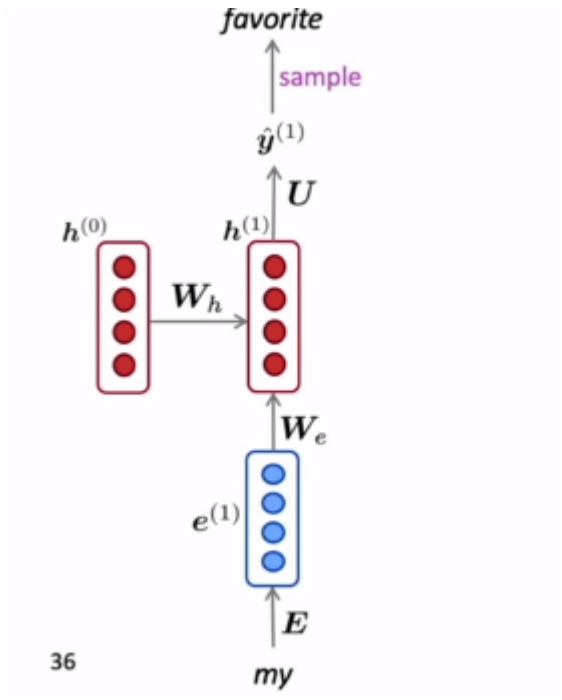


Note: we have to remember to not do backprop for every step individually but do it as some kind of increment where result of previous step is used in the calculation.

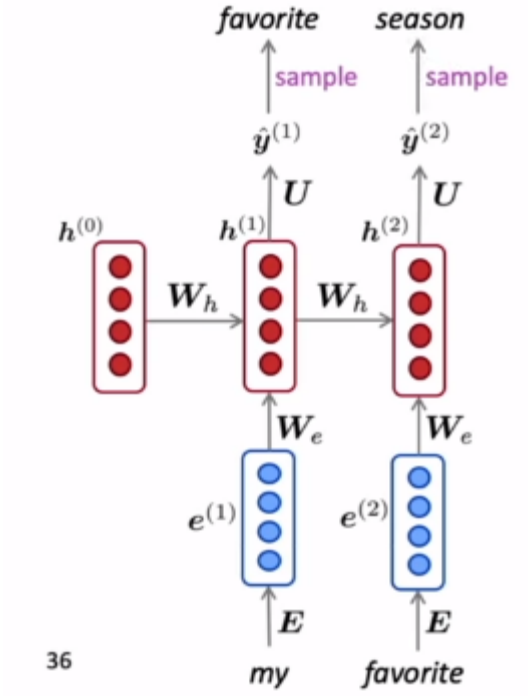
Text Generation with a RNN:

Let’s say we feed in the word my,

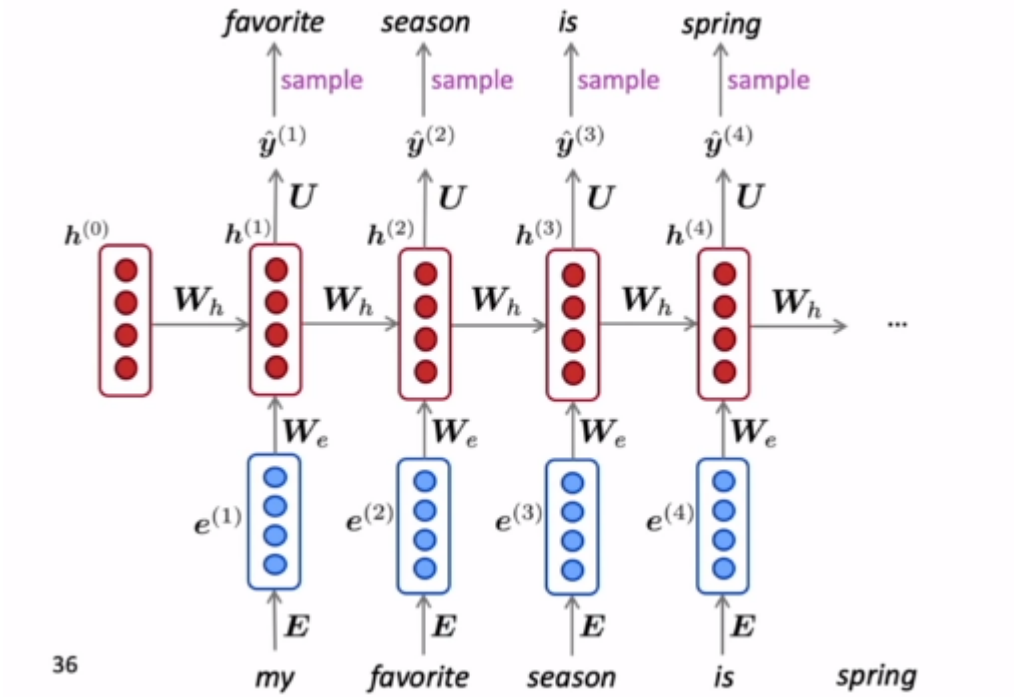
Step1:



Step2:



StepN:



Evaluation RNN:

For evaluation RNNs we use **Perplexity:**

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

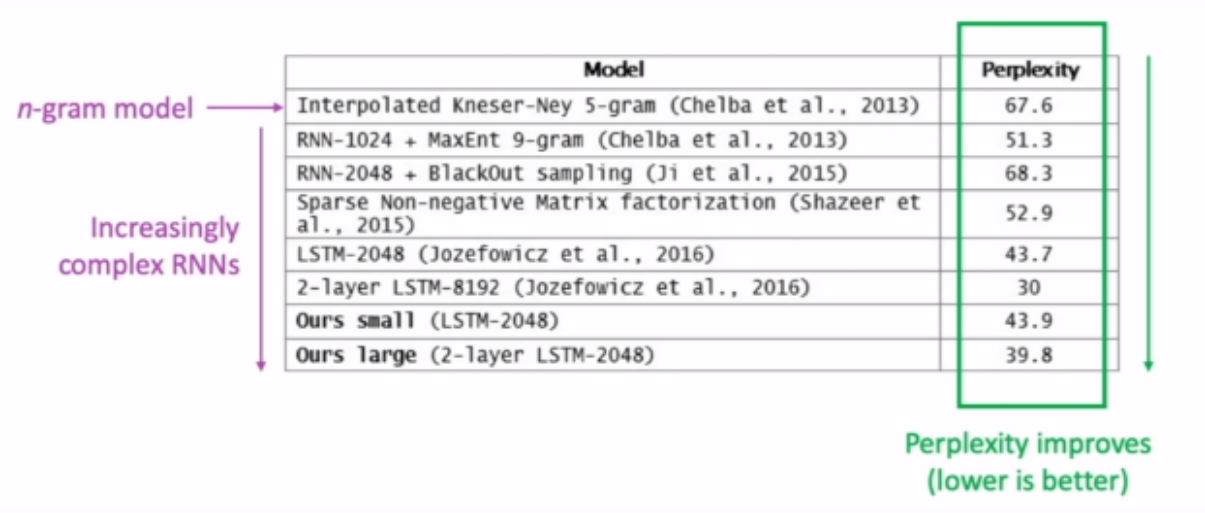
If we notice we can realize perplexity is just the exponent of the cost function.

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

Some perplexity numbers:



Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

Note: we can also use RNN for word tagging like named entity recognition or sentiment recognition.