RNN Language Models

① We have a lot of predictive power, as humans, when it comes to language. Suppose you're asked to guess the next letter in the sequence:

W A __

Likely you don't have a uniform probability over letters. Letters like S or N are quite likely (in English), but Q or E are pretty unlikely (this, despite the fact that E is generally a common letter).

② More context can sharpen the probability:

GEORGE ␣ WA __

BEES WA __

WHAT ␣ DO ␣ YOU ␣ WA __

③ A <u>language model</u> over an alphabet A is a probability distribution over letter $a_{n+1} \in A$ given the previous letters in a sequence $a_1, a_2, \ldots, a_n$.

$$\Pr(a_{n+1} \mid a_1, a_2, \ldots, a_n)$$

RNN Language Models

④ Language models are frequently used in natural language processing to determine how "fluent" a sequence sounds (relative to a particular language). For instance, a good language model (LM) for English should have

$$\Pr(\text{"N"} \mid \text{"W"}, \text{"A"}) > \Pr(\text{"E"} \mid \text{"W"}, \text{"A"})$$

⑤ A sequence's probability can be scored as:

$$\Pr(a_1, \ldots, a_K) = \prod_{k=1}^{K} \Pr(a_k \mid a_1, \ldots, a_{k-1})$$

A good LM for English should have

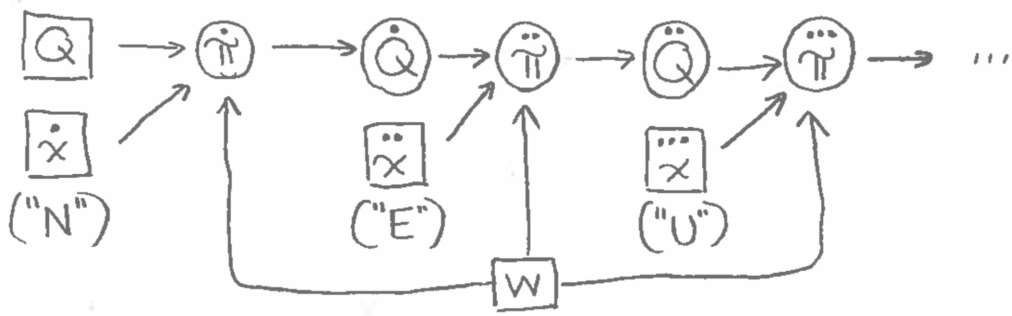
$$\Pr(\text{"THERE IS A"}) \gg \Pr(\text{"IHRA S EET"})$$

⑥ RNNs provide an effective way to create LMs.

The main difference between RNN LMs and the RNNs we've seen so far is that RNN LMs don't just take an input character at each time step, they also emit an output character.

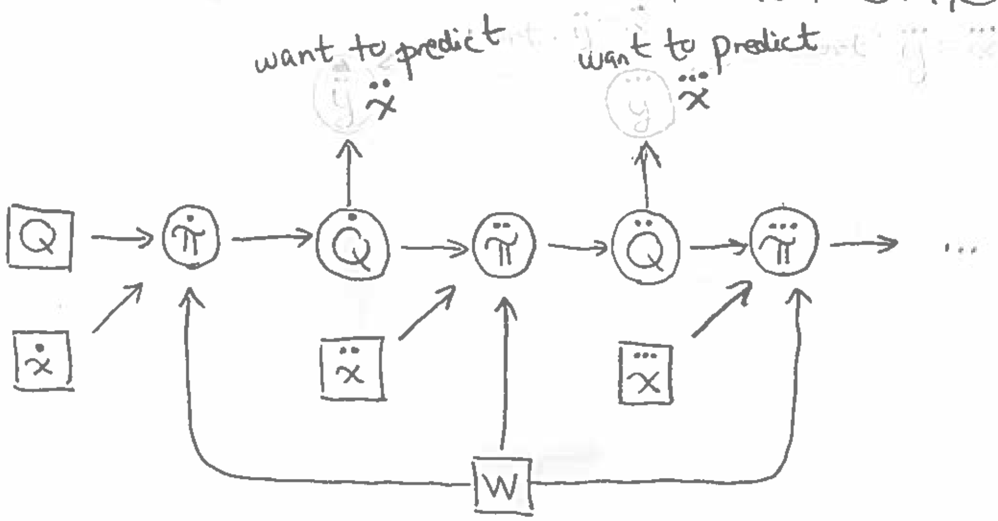
RNN Language Models

⑦ Consider our standard RNN:



As it reads in letters ("NEU..."), it updates the state vector Q.

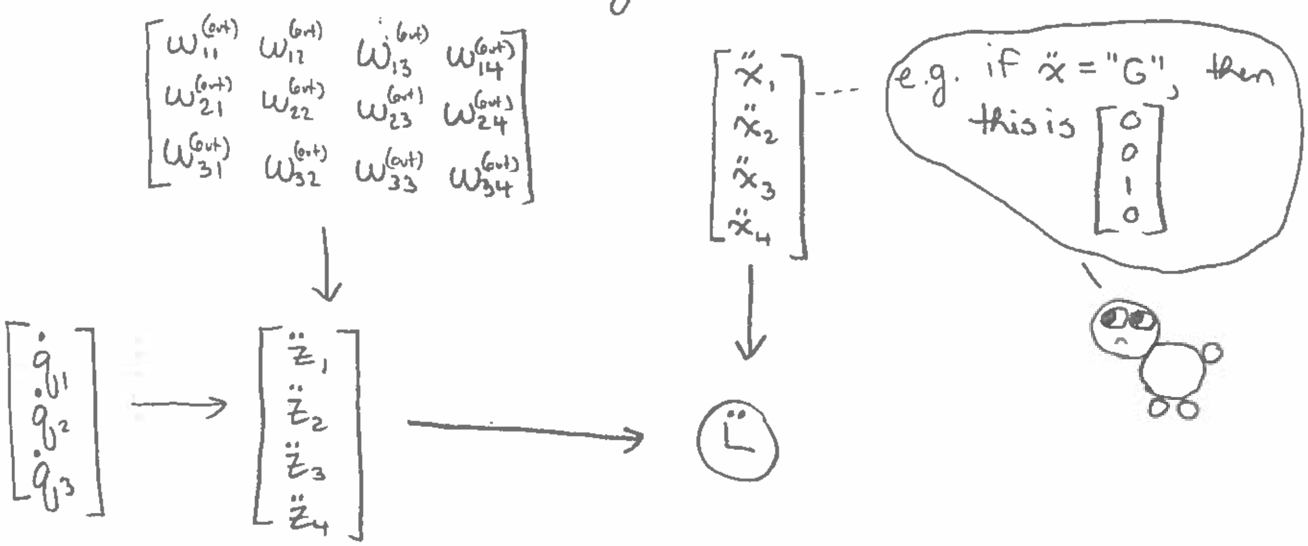
⑧ To make this into a language model, we will try to predict the next letter from each state vector:



RNN Language Models

(9) To show some simple concrete examples, let's assume we're learning a language model over DNA sequences (so our alphabet is $\{A, C, G, T\}$).

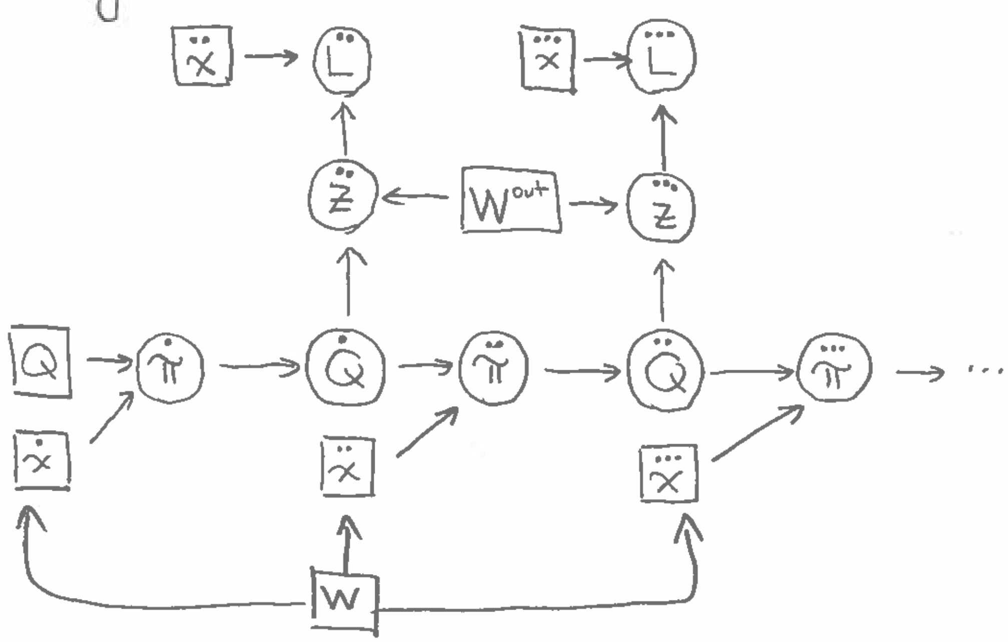
To predict the next letter from the current state vector $\dot{Q}$, we can use a simple dense layer:



where: $L(\ddot{z}, \ddot{x}) = -\log\left(\ddot{x}^T \cdot \text{softmax}(\ddot{z})\right)$

$$\ddot{z} = W^T \dot{q}$$

(10) Putting this into our RNN:



RNN LANGUAGE MODELS

11 It's mildly awkward that our loss is distributed across each layer, but it's easy to reconcile using probability and logs. Essentially, we want to maximize the predicted probability of the actual sequence:

$$\prod_k \Pr(a_k)$$

$$= \prod_k \overset{k}{x}^T \cdot \mathrm{softmax}(\overset{k}{z})$$

This is equivalent to maximizing the log probability

$$\log \prod_k \overset{k}{x}^T \cdot \mathrm{softmax}(\overset{k}{z})$$

$$= \sum_k \log \overset{k}{x}^T \cdot \mathrm{softmax}(\overset{k}{z})$$

Or alternatively, <u>minimizing</u> the <u>negative</u> log probability:

$$-\log \prod_k \overset{k}{x}^T \cdot \mathrm{softmax}(\overset{k}{z})$$

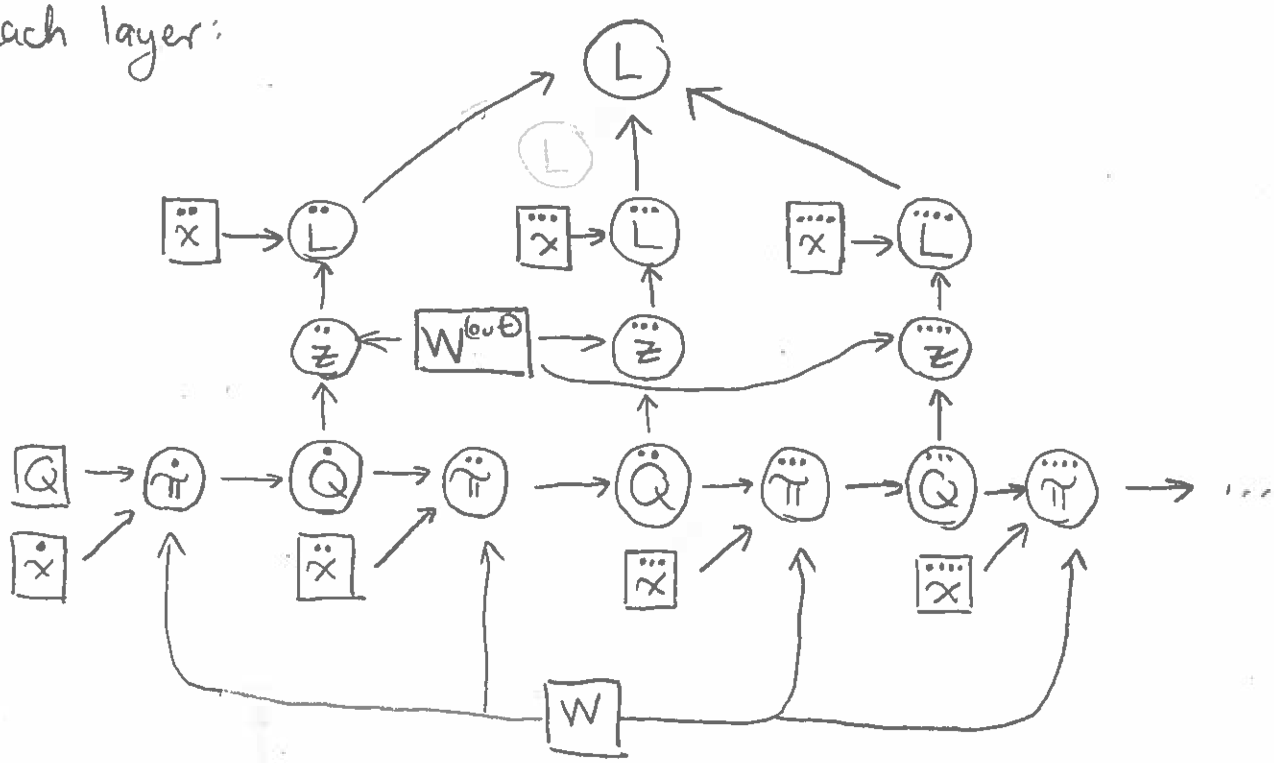
$$= -\sum_k \log \overset{k}{x}^T \cdot \mathrm{softmax}(\overset{k}{z})$$

$$= \sum_k -\log \overset{k}{x}^T \cdot \mathrm{softmax}(\overset{k}{z})$$

$$= \sum_k L(\overset{k}{z}, \overset{k}{x})$$

RNN Language Models

12) So the overall loss is just the sum of the losses at each layer:



13) From this depiction, it is straightforward (though arduous by hand) to compute $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial W^{(out)}}$

for each sequence $\dot{x}, \ddot{x}, \dddot{x}, \ldots$ in our training data.

e.g. $$\frac{\partial L}{\partial W} = \sum_{k=2}^{K} \frac{\partial L}{\partial \overset{(k)}{L}} \cdot \frac{\partial \overset{(k)}{L}}{\partial W}$$

$$= \ldots \text{ etc.}$$