Recursive Neural Networks (RNNs)

CS 4740 (and crosslists): Introduction to Natural Language Processing
https://courses.cs.cornell.edu/cs4740/2025sp

Slides developed by:
Magd Bayoumi, Claire Cardie, Tanya Goyal, Dan Jurafsky, Lillian Lee, James Martin, Marten van Schijndel

Announcements

- **Midterm (=7:30pm Thurs)**
- Olin 155: netids **aa*-rt***
- Olin 165: netids **ru*-zz***
  - Bring your id card
- Weds class: midterm review

Last class

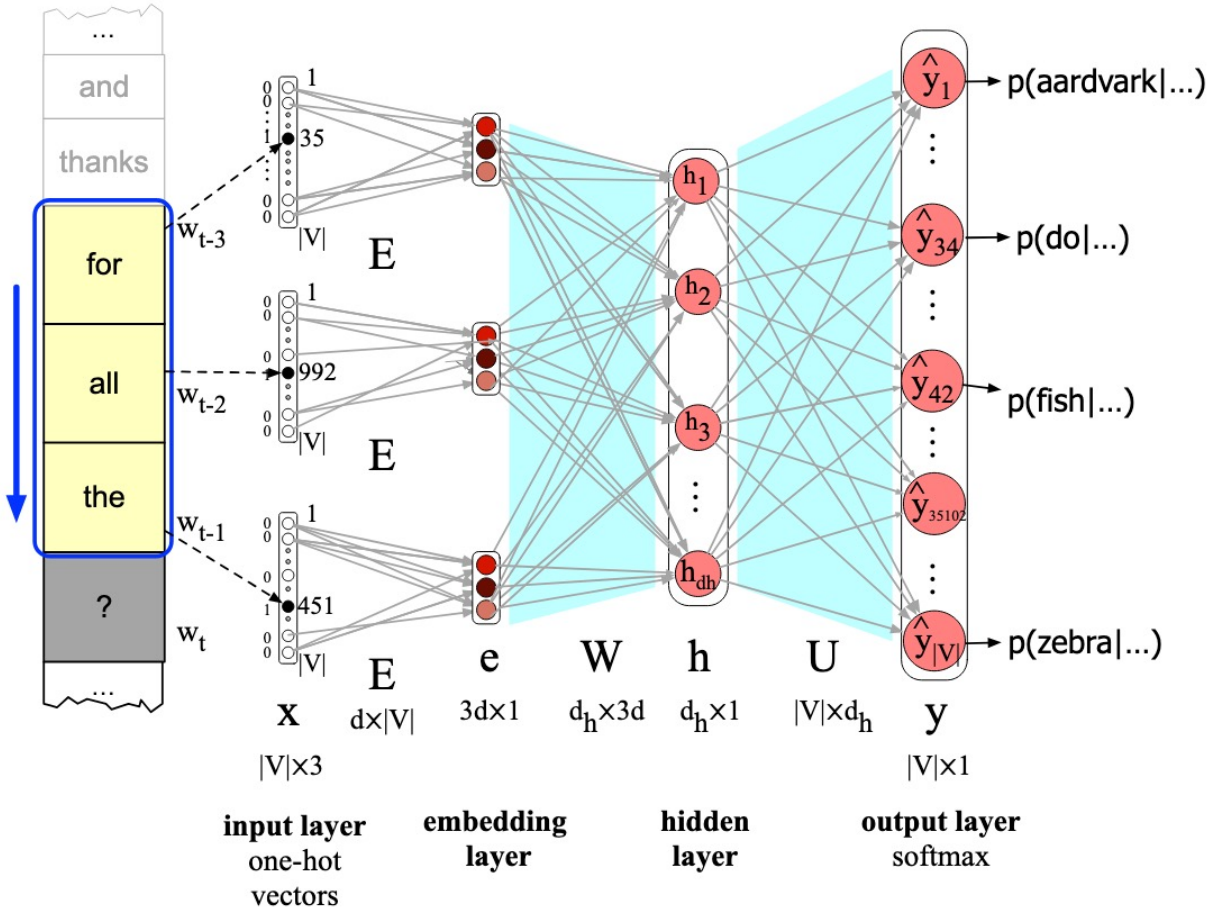
- Error backpropagation for FFNNs
- FFNNs as language models

Today

- Recurrent neural networks (RNNs)
  - What are they?
  - Training them
  - RNN architectures for NLP applications

Recall: FFNNs for language modeling

- Forward inference at each timestep



Recall: Incorporating embedding vector selection

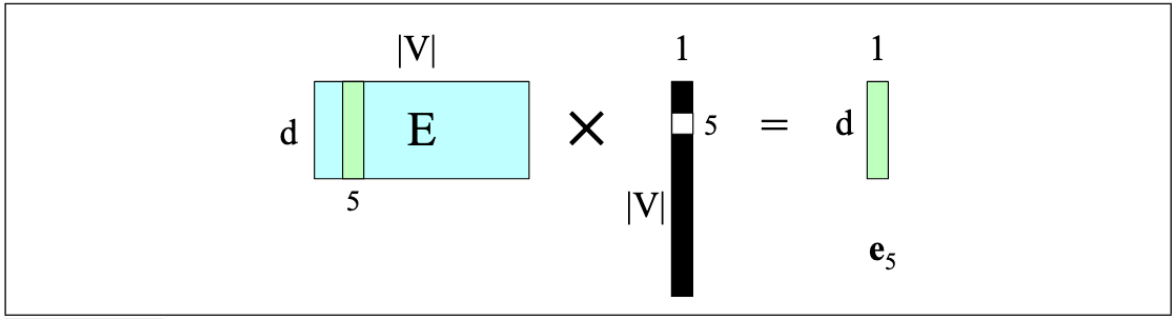
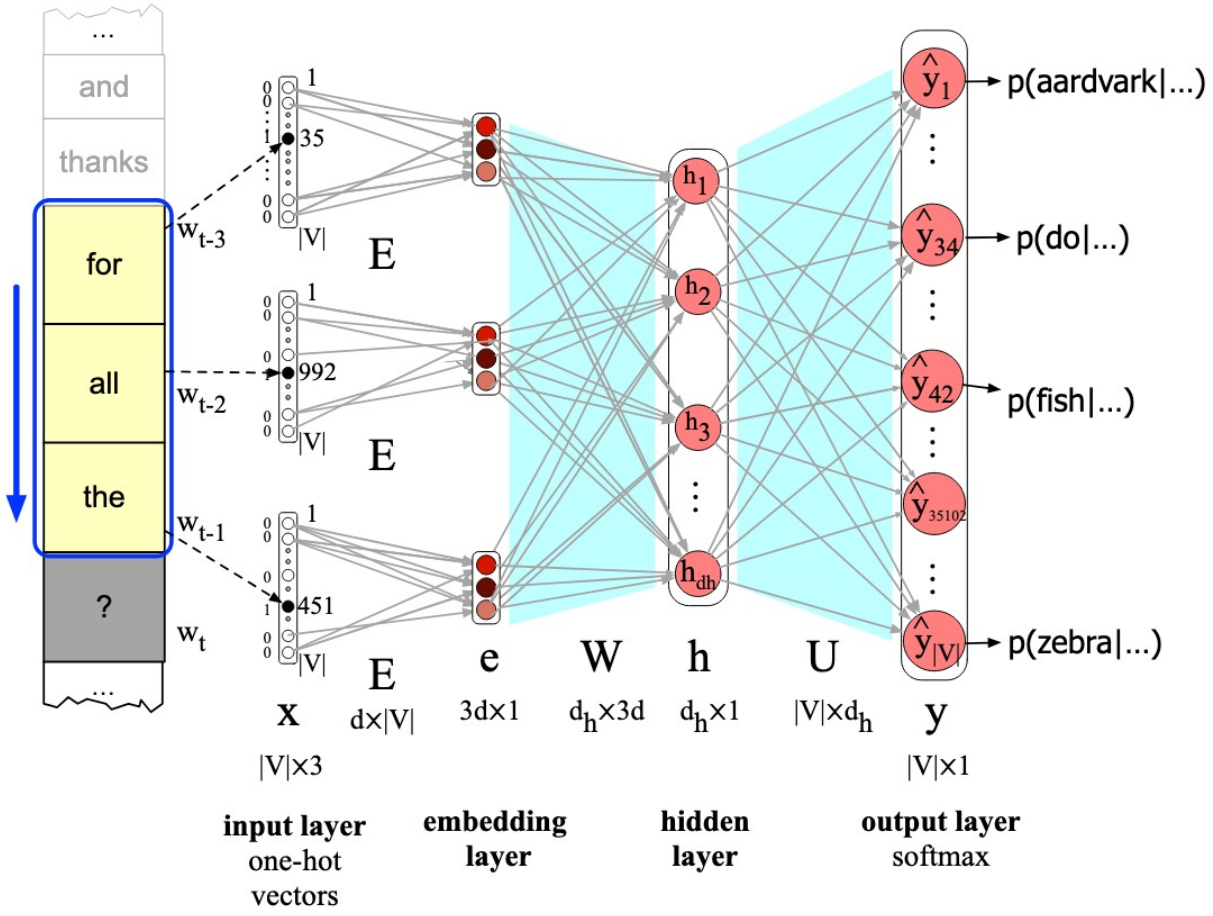


Figure 7.16 Selecting the embedding vector for word V_5 by multiplying the embedding matrix **E** with a one-hot vector with a 1 in index 5.

Recall: FFNNs for language modeling

- Forward inference at each timestep

$$\mathbf{e} = [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}]$$
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{e} + \mathbf{b})$$
$$\mathbf{z} = \mathbf{U}\mathbf{h}$$
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$



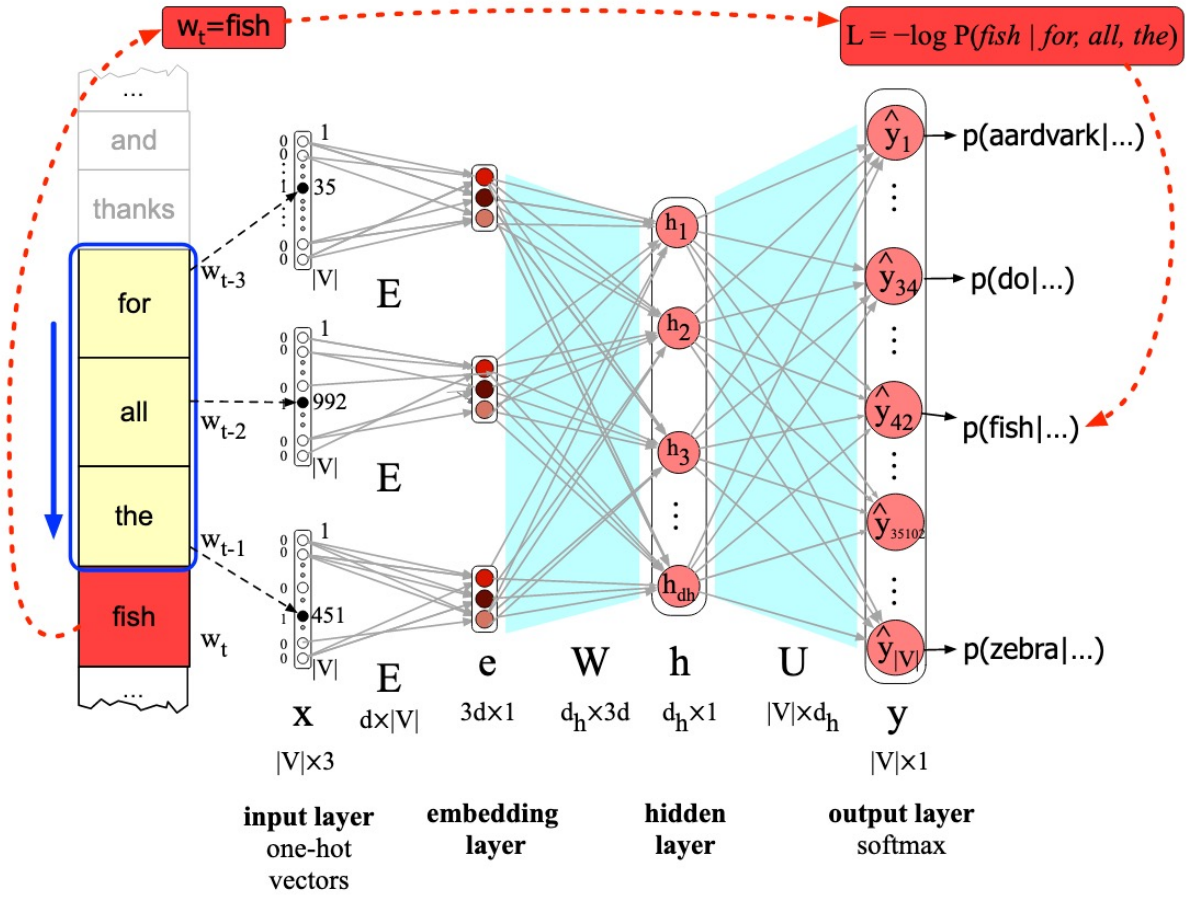
Training the NLM

- For some tasks, it's ok to **freeze** the embedding layer E with initial word2vec values.
  - Hold E constant while we only modify W, U, and b

- However, often we'd like to learn the embeddings simultaneously with training the network.
  - Tune the embeddings to the task the network is designed for (e.g., sentiment classification, or translation, or parsing)

Training procedure

- Take as input a very long text, concatenating all the sentences
- Start with random (or pretrained) weights
- Iteratively move through the text predicting each word w_t
  - Calculate loss
  - Backpropagate error
- **Key:** embedding matrix is shared among the context words



Result

- An algorithm/model for language modeling (a word predictor)
- A new set of embeddings *E* that can be used as word representations for other tasks

Neural LMs vs. n-gram LMs

- Advantages over N-gram-based LMs
  - Don’t need smoothing
    - Still need to handle unknown words
  - Can handle much longer histories
  - Can generalize over contexts of similar words
  - Higher predictive accuracy (given same training set size)
- Disadvantage over N-gram-based LMs
  - Way slower to train

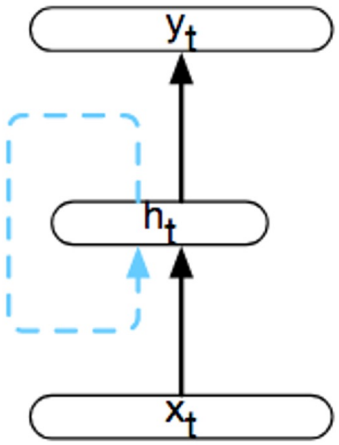
Feedforward neural LMs: Problems

- Limits the context to a fixed-size window
  - Many NLP tasks require access to information arbitrarily distant from the point at which a decision is made
- Difficult to learn systematic patterns
  - “the fish” at position one in the window is treated completely differently from “the fish” at position two

Can fix these problems with RNNs (Recurrent Neural Networks)

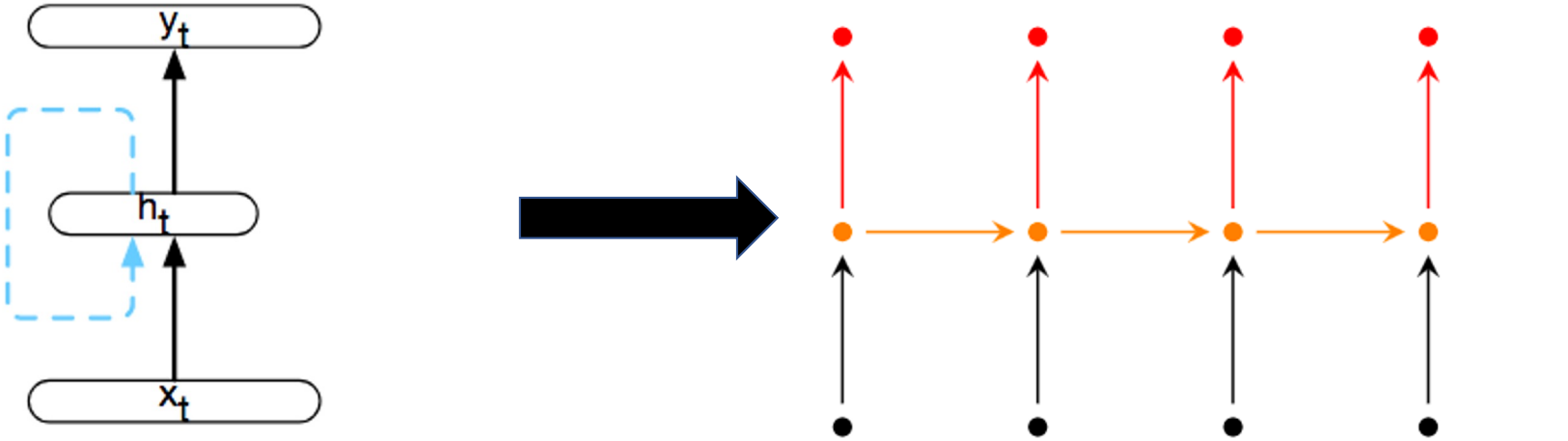
Recurrent NNs: the idea

- Any network that contains **a cycle**
- In the general case, networks with cycles are hard to train
  - Not the case for simple RNNs!
- Hidden layer activation depends on the
- input layer **and** the **activation of the hidden layer**
- **from the previous timestep**



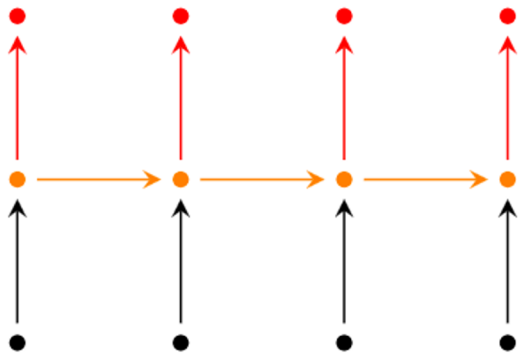
RNN unfolded in time (forward computation)

In this visualization of RNNs, **each node represents an entire layer; each edge represents a weight matrix**



Simple RNNs

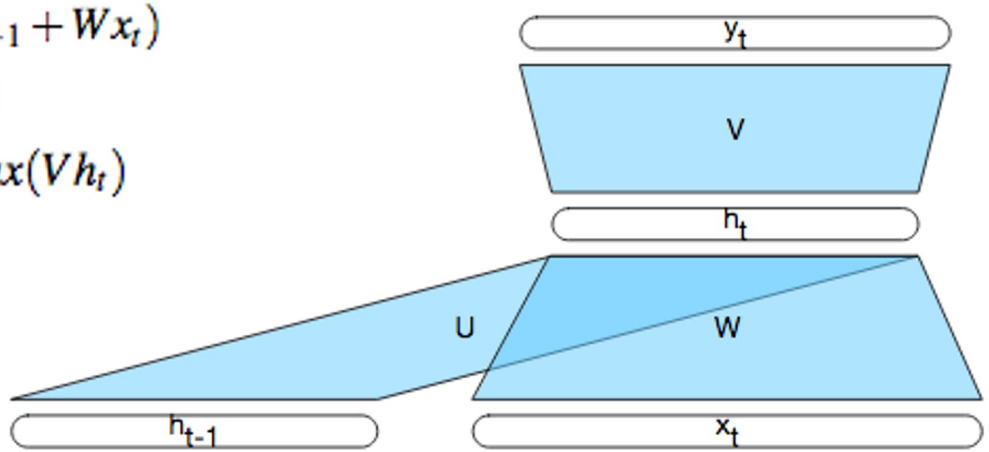
- Hidden layer from previous timestep acts as a form of memory (or context)
- Encodes earlier processing
- Informs decisions made at a later time
- Includes information extending back to the beginning of the sequence



RNNs: the computation structure

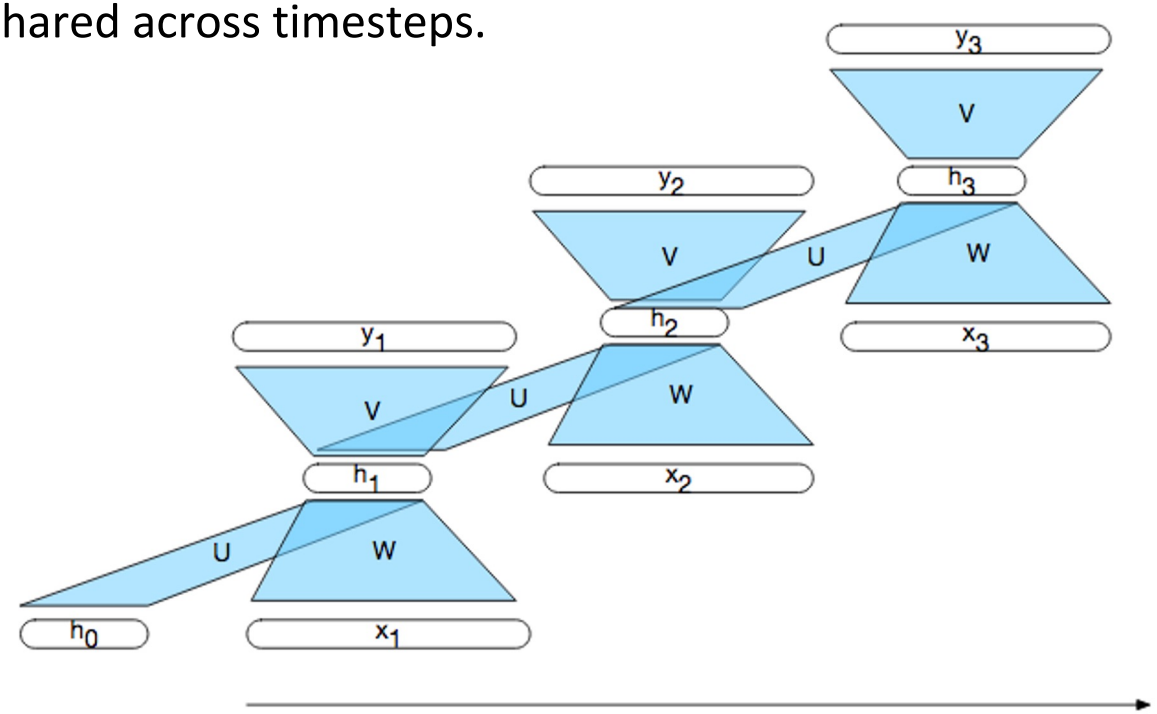
- Similar feedforward calculations
- U determines how the network should make use of past context.

$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

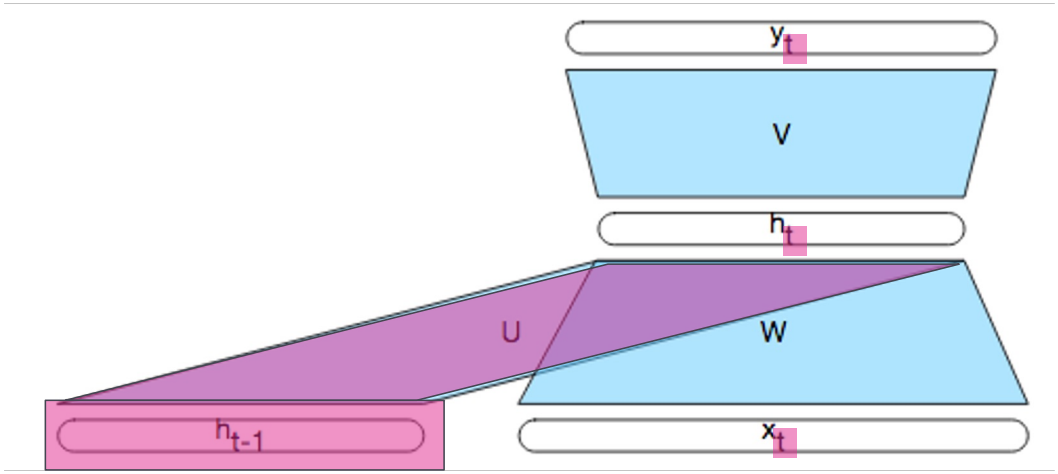


Unrolling the network in time

Layers have differing values over time.
Weights are shared across timesteps.



Recurrent NNs: allowing previous "state" to affect the decision for the next input.

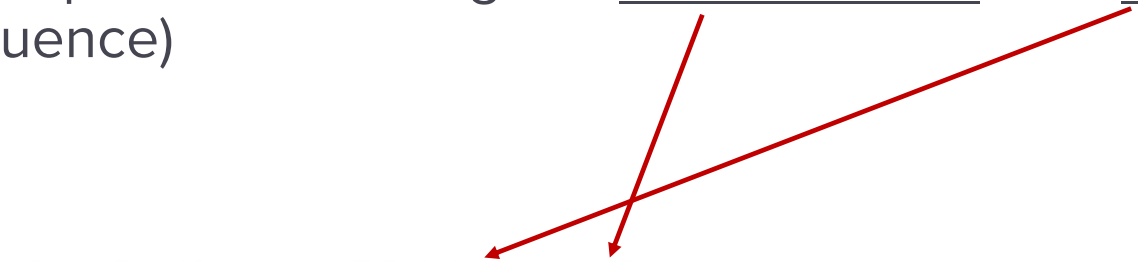


$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

As the highlighted subscripts indicate, the input "time" (position in the input sequence $\mathbf{x}_1\mathbf{x}_2$, ...) now matters much more, in comparison with FFNNs.
(We need some h_0; let's not worry about it now.)

(Simple) RNNs in algorithm format

- make predictions using the trained model on a *test* instance (a sequence)

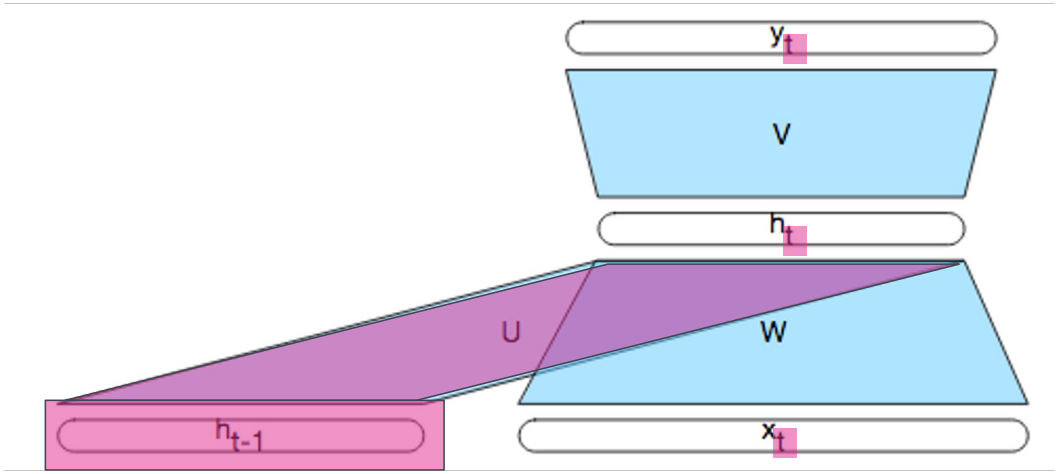


function FORWARDRNN($\mathbf{x}$, *network*) **returns** output sequence $\mathbf{y}$

$\mathbf{h}^0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH($\mathbf{x}$) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

Input is a sequence of $\mathbf{x}$ = ..., $\mathbf{x}_t$, ... vectors.
Output is a sequence $\mathbf{y}$ where $\mathbf{y}_t$ for $\mathbf{x}_t$ depends on $\mathbf{h}_t$.

Q1: does the dimension of the individual input word vectors need to be pre-specified?



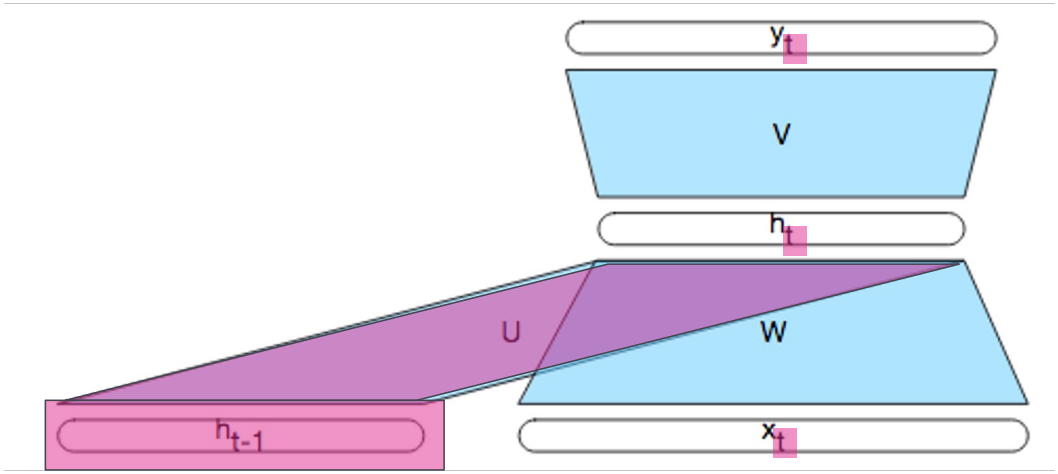
$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

A: …

Q': is this the same as or different from the case for FFNNs?

A': …

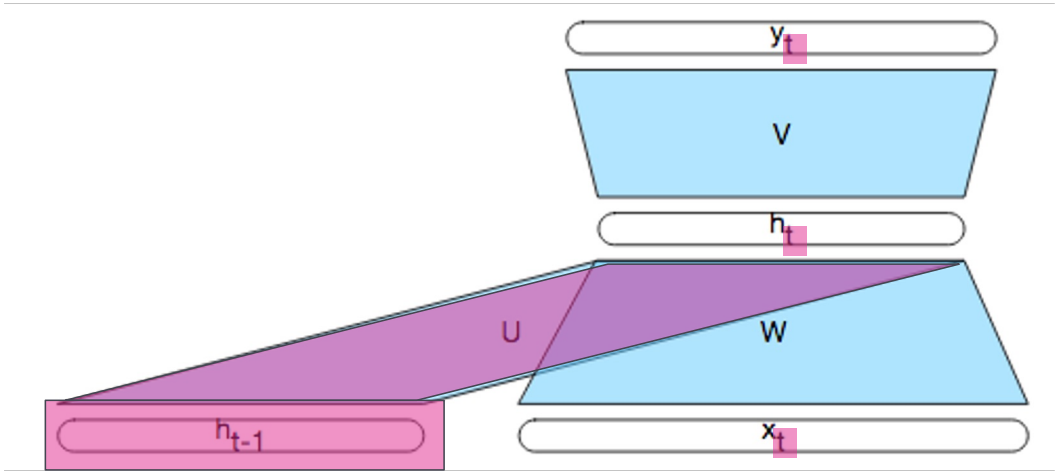
Q1: does the dimension of the individual input word vectors need to be pre-specified?



$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

A: Yes.
Q': is this the same as or different from the case for FFNNs?
A': It's the same.

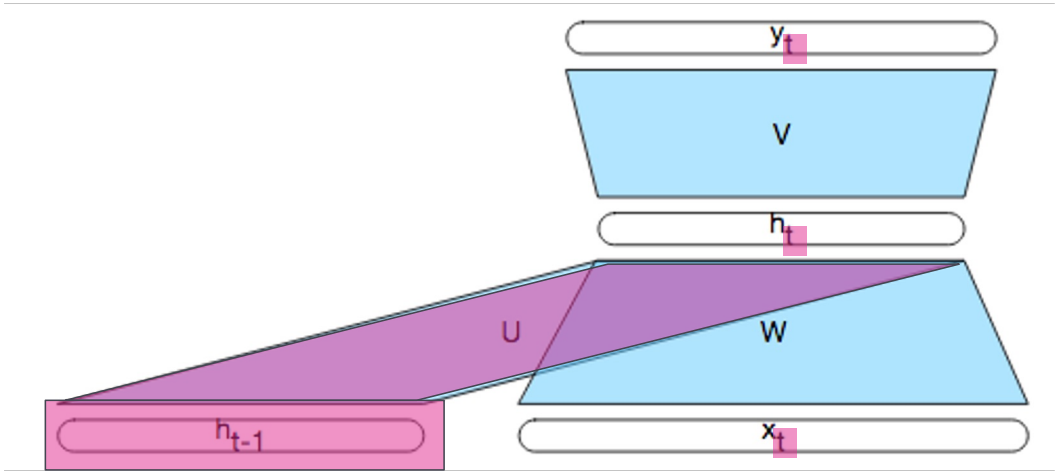
Q2: do the W, U, V matrices change for different input x_t?



$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

A: According to the notation …
Q': is this the same as or different from the case for FFNNs w.r.t. their two matrices?
A': …

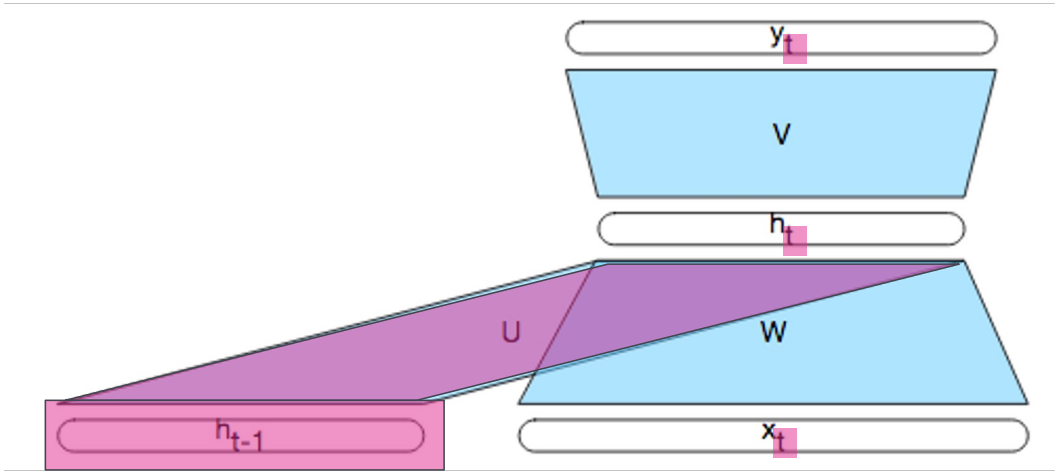
Q2: do the W, U, V matrices change for different input x_t?



$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

A: No, so the number of parameters is constant in the length of the input sequence.
Q': is this the same as or different from the case for FFNNs w.r.t. their two matrices?
A': It's the same.

Q3: Is there a limit on how much previous input influences the current computation?



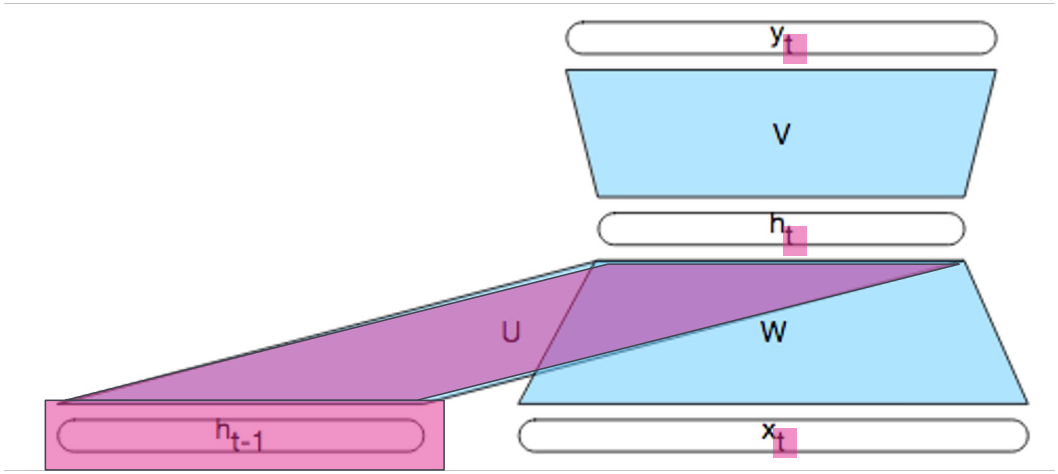
$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

A: …

Q': is this the same as or different from the case for FFNNs?

A':

A3: For RNNs, the entire history has an influence (in principle)

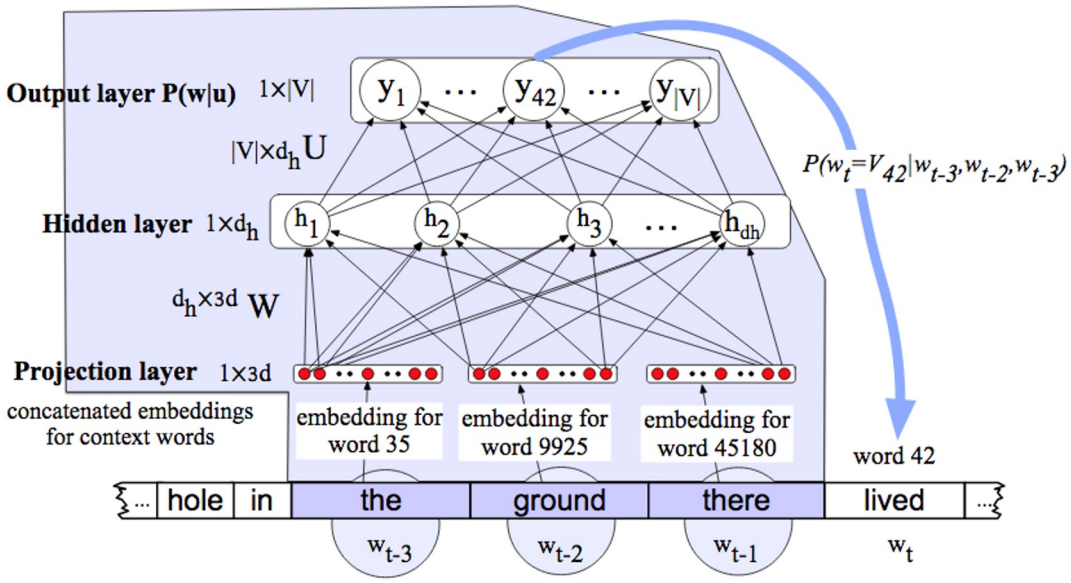


$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

$\mathbf{h}_t = g(U\mathbf{h}_{t-1} + \ldots)$
$= g(Ug(U\mathbf{h}_{t-2}+\ldots) + \ldots)$
$= g(Ug(Ug(U\mathbf{h}_{t-3}+ \ldots) +\ldots) + \ldots) = \ldots = g(Ug(Ug(Ug(\ldots.. Ug(\mathbf{h}_0+ \ldots))) +\ldots) + \ldots)$
for any t!

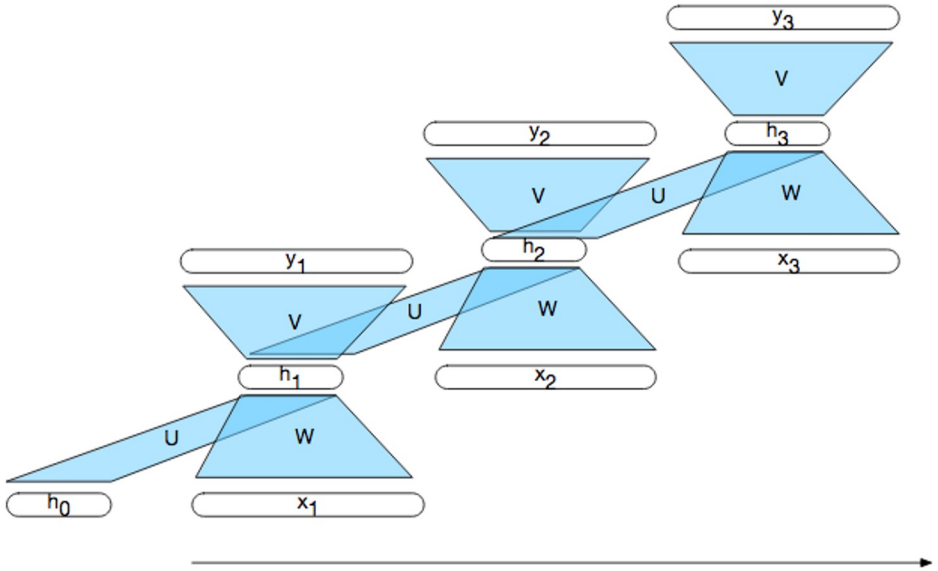
Recall the “FFNN Solution” to incorporating previous context.

The context width of “3” has to be fixed ahead of time, in contrast to the RNN case.



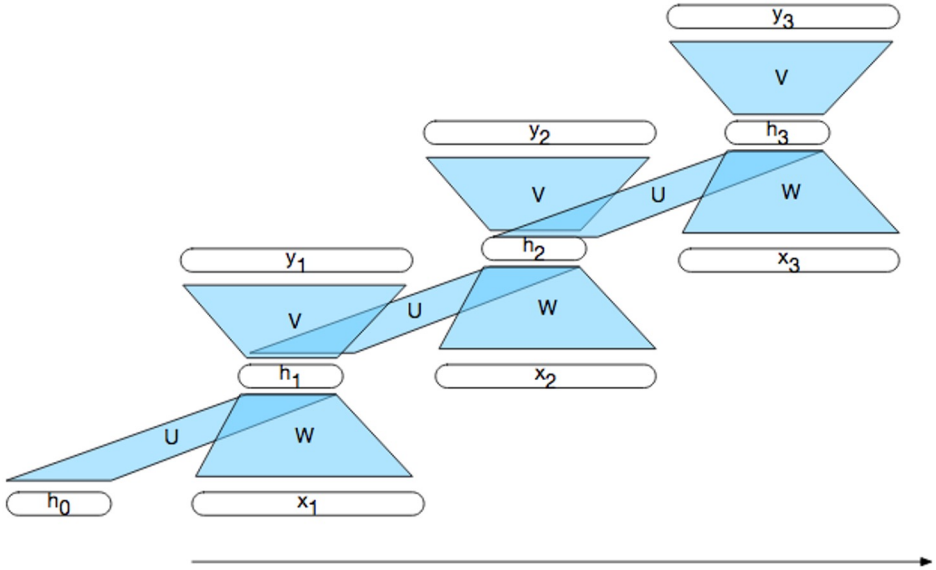
But wait: can't I reduce an RNN to a really wide FFNN…?

For input sequence $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, this wide FFNN "unrolls" the RNN's computation:



But wait: can't I reduce an RNN to a really wide FFNN…?

For input sequence $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, this wide FFNN "unrolls" the RNN's computation:



Oh: that's if we **fix the length of the input sequence**(e.g., to 3).

RNNs don't have this limitation.

Note: this is indeed an FFNN because there are no cycles, even if it doesn't look like the FFNNs we've been depicting so far.

So RNNs can *really* take arbitrarily long history into account?

In practice, no.

- Some problems with *vanishing gradients* in training (we'll discuss this later)
- It takes a long time to start the computation at input 1 and get to input 10000.

  Break the real input sequence up into subsequences, and only process one subsequence at a time.

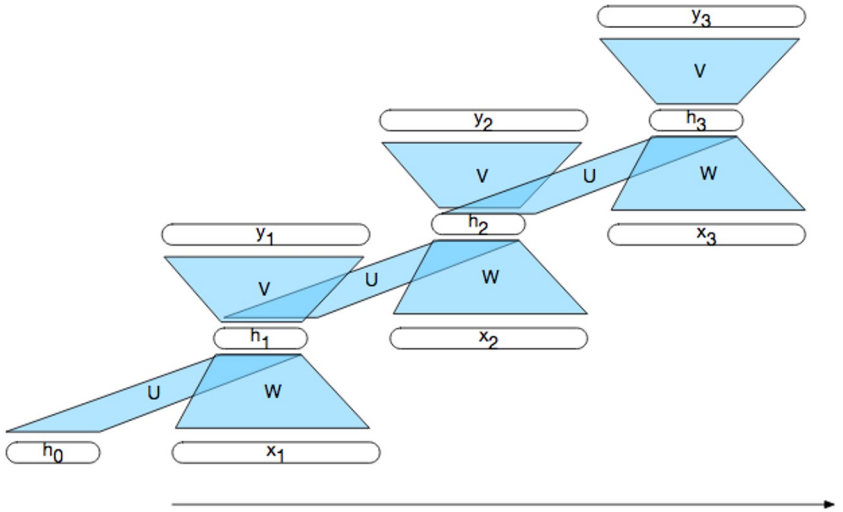
  FFNNs, in contrast, can process all the input tokens in parallel.

Today

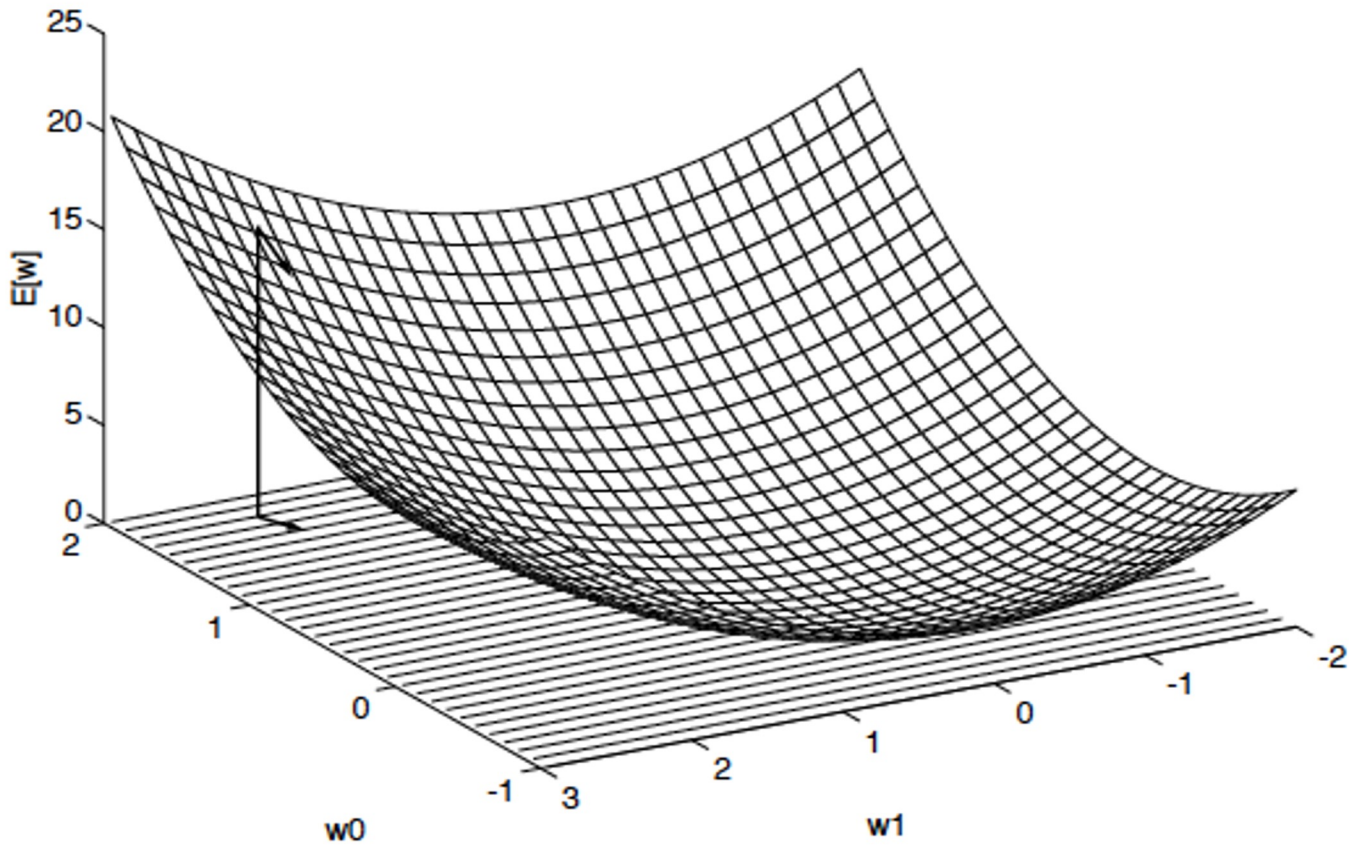
- Recurrent neural networks (RNNs)
  - What are they?
  - **Training them**
  - RNN architectures for NLP applications

Training a simple recurrent network

- As with feedforward networks, we'll use:
  - a **training set**,
  - a **loss function** (distance between the system output and the gold output),
  - **backpropagation** to adjust the sets of weights
- Three sets of weights to adjust: U, V, W



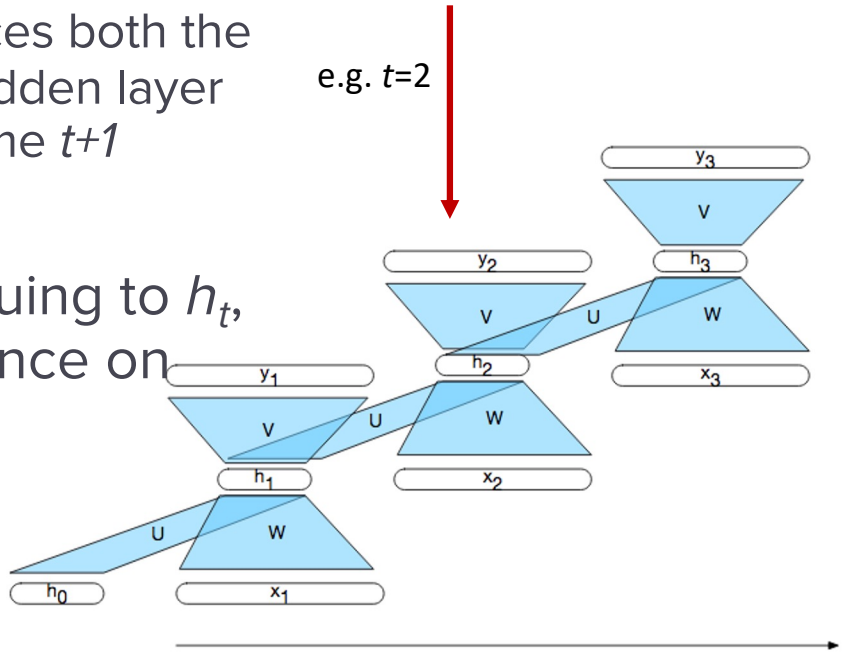
Gradient Descent Through Weight Space



Training a simple recurrent network (SRN)

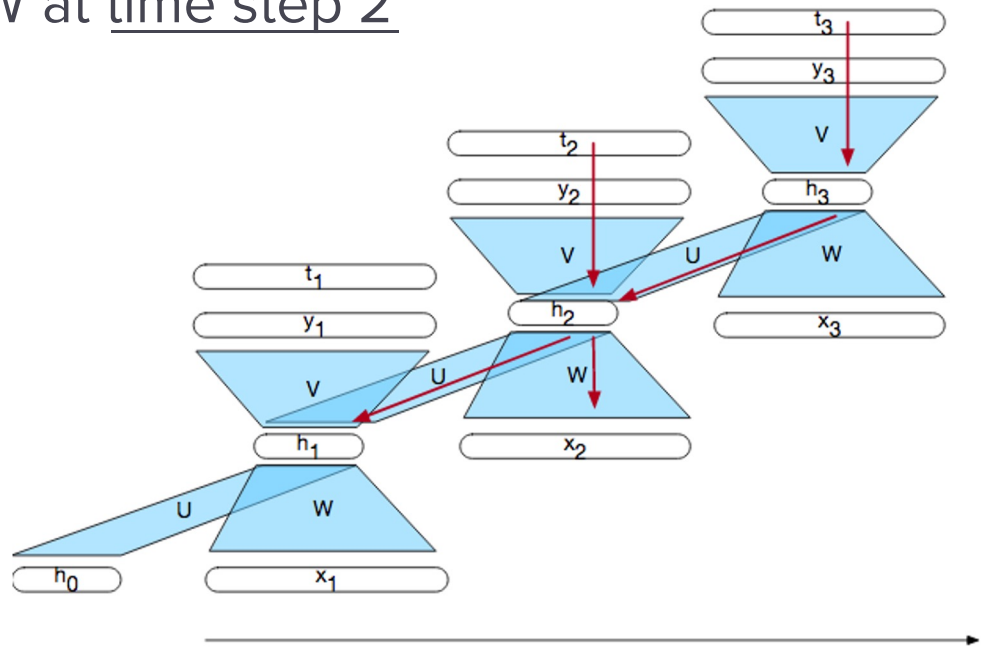
- Complications
  - to compute the loss function for the output at time *t* we need the hidden layer from time *t−1*
  - hidden layer at time *t* influences both the output at time *t* and the the hidden layer (and the output and loss) at time *t+1*

- So…to assess the error accruing to h_t, we'll need to know its influence on both the output at *t* and the output at *t+1*



Backpropagation through time (BPTT)

- The t_i vectors represent the target (desired output)
- Shows the flow of backpropagated errors needed for updating U, V, W at time step 2



Summary

function BACKPROPTHROUGHTIME(*sequence, network*) **returns** gradients for weight updates
forward pass to gather the loss
backward pass compute error terms and assess blame

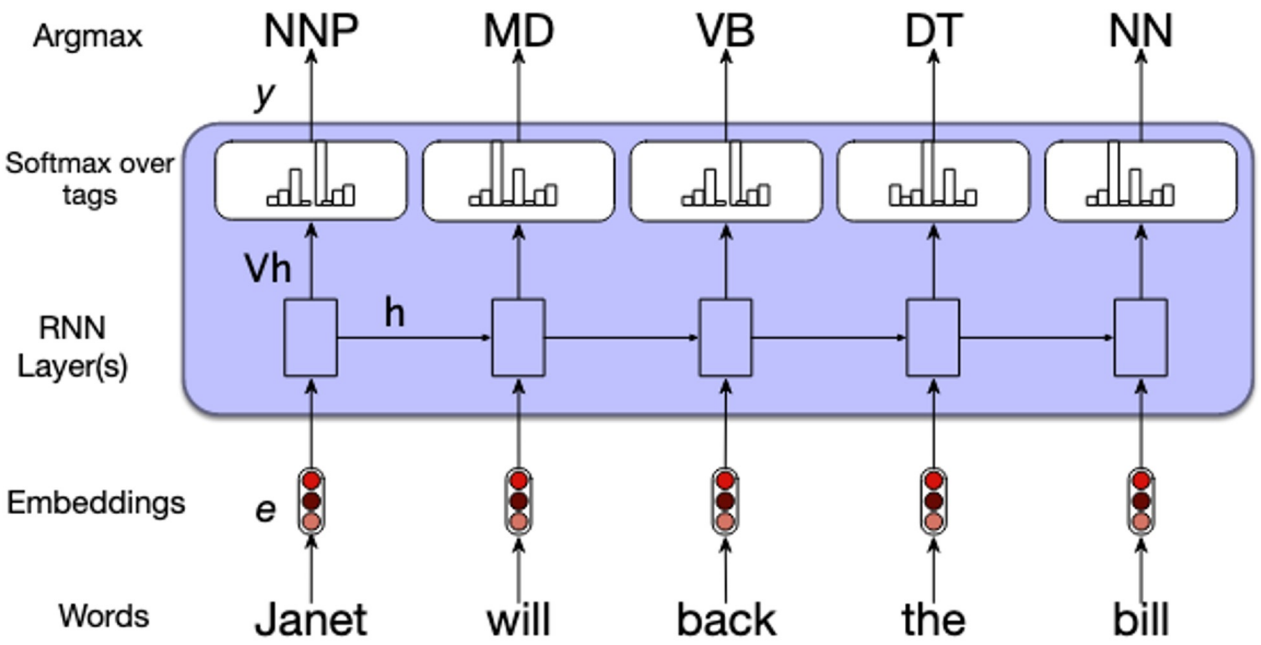
RNNs form the basic building blocks for many NLP tasks!!!!

Today

- Recurrent neural networks (RNNs)
  - What are they?
  - Training them
  - **RNN architectures for NLP applications**

Sequence tagging/labeling tasks

Part of speech tagging



Almost *any* sequence tagging task

- NE recognition
- Chunking
- Opinion extraction
- SRL (semantic role labelling)

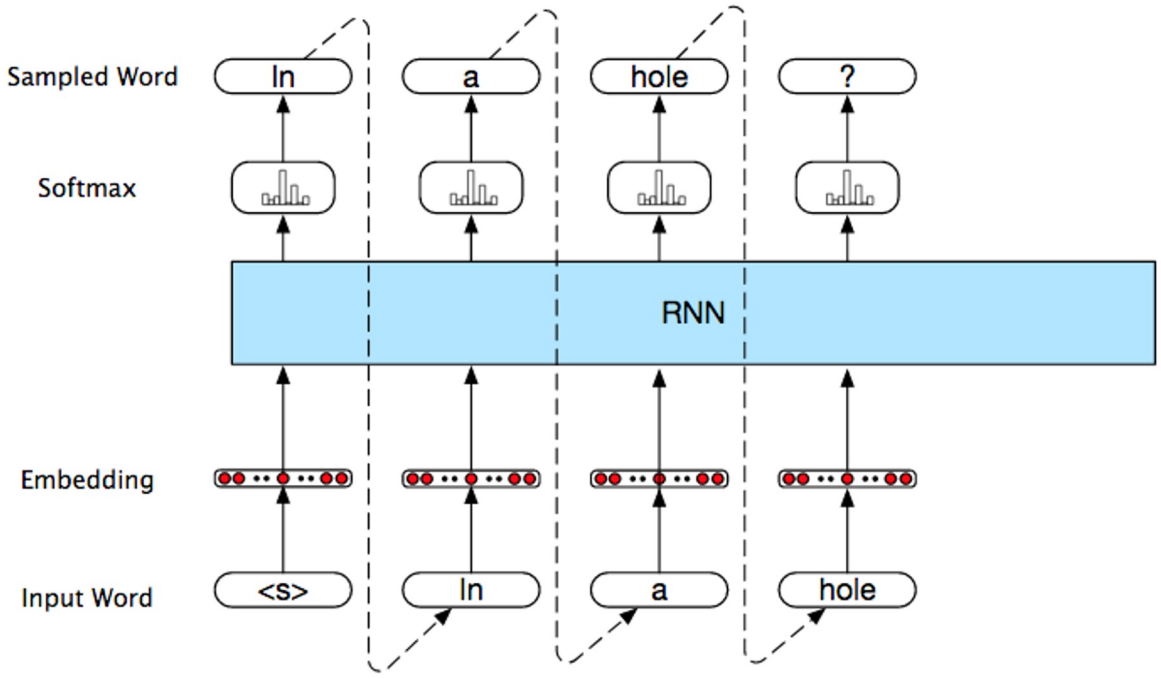
Application: language modeling/generation

Autoregressive generation:

word generated at each time step is conditioned on the word generated by the network at the previous step.

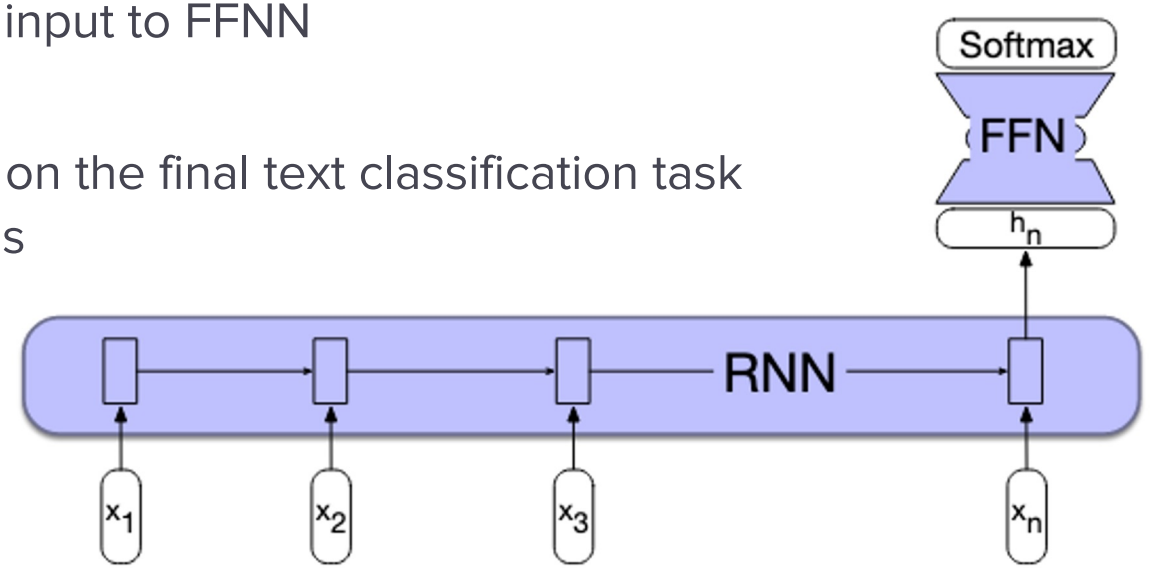
Training the RNN:

Typically use **teacher forcing**, i.e., use the predicted word at each time step for backprop, BUT supply the gold sequence of tokens for input.



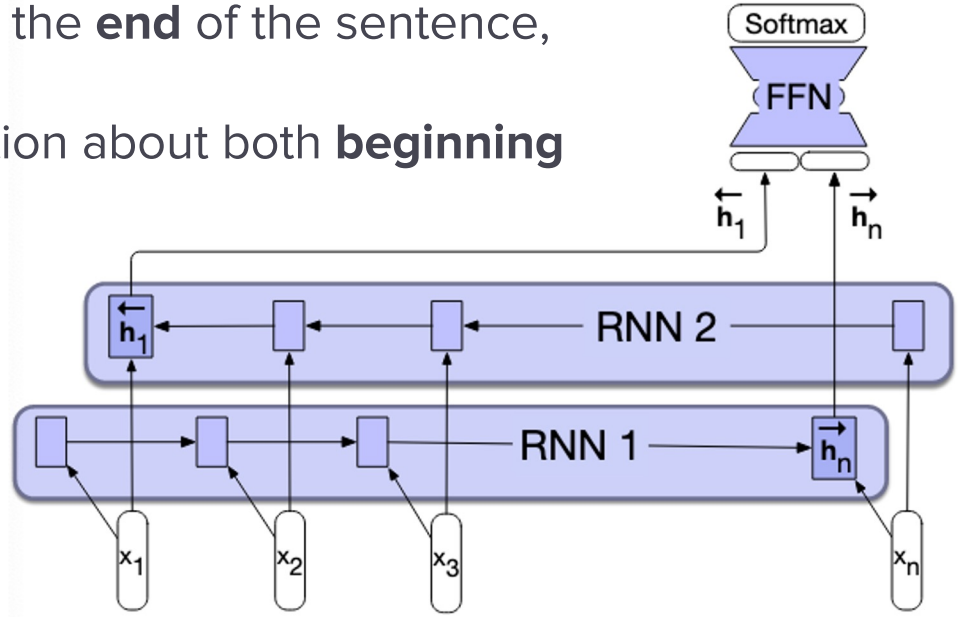
Application: one label for the entire sequence

- To classify the entire sequence
  - E.g., Is the sentence +/-/neutral in sentiment? Is the review real/fake?
  - No intermediate token-level outputs
  - Final hidden state is input to FFNN
- End-to-end training
  - Loss function based on the final text classification task
  - No token-level losses



Bidirectional RNNs

- Train two models, one in **forward** direction, one in **backward** direction
- Especially helpful for sequence classification
  - h_n reflects more information about the **end** of the sentence, h_1, about the beginning
  - bidiRNNs allow capturing information about both **beginning** and **end** of input sequence
  - Normally use concatenation



See you at the midterm!!!