Natural Language Processing
Neural sequence labeling

Sofia Serrano
sofias6@cs.washington.edu

Credit to Sachin Kumar, Graham Neubig, Emma Strubell, Wei Xu, and Greg Durrett for slides

Announcements

- A2 is due on Friday at 11:59pm
  - Careful with the number of late days you have left! If you used three late days on the last assignment, you have two late days left for the quarter.
  - Make sure to commit/push your **.preds files** by the deadline, in addition to your code and writeup
- Extra office hours this week– the course google calendar lists the OH schedule
- Quiz 5 goes out on Canvas Wednesday at 2:20pm
  - Available until Friday at 2:20pm; you'll have 15 minutes to complete it once you start.
  - Remember that you can use your notes during the quiz
  - Will cover material from last Wednesday's lecture through the end of today's lecture (so, Viterbi, CRFs, and neural sequence labeling)

Generative vs Discriminative Models

- Generative Models specify a *joint* distribution over the labels and the data. e.g. HMMs

$$p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$$

  - Training: Maximum Likelihood Estimation (Count and Divide)
  - Estimation: $\text{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \text{argmax}_{\mathbf{y}} \frac{P(\mathbf{y}, \mathbf{x})}{\cancel{P(\mathbf{x})}}$

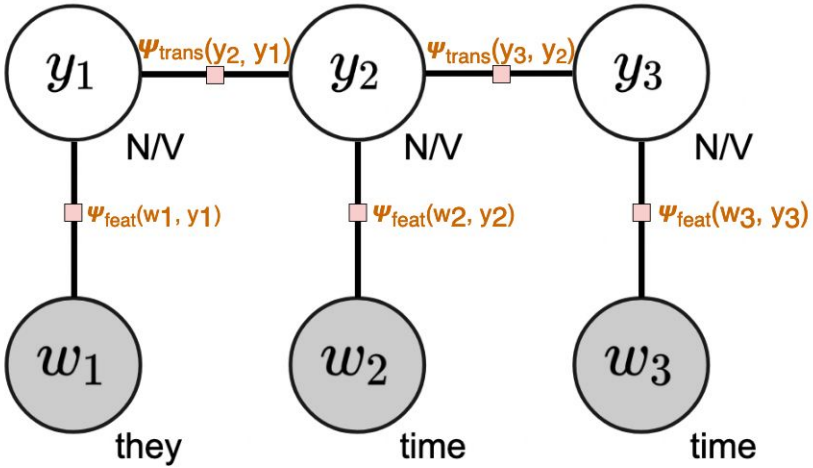
- Discriminative models compute the *conditional* distribution of the labels given the input. You want to **discriminate** between different labels.

$$p(\mathbf{y}|\mathbf{x})$$

An aside: MEMMs, another discriminative model for sequence labeling

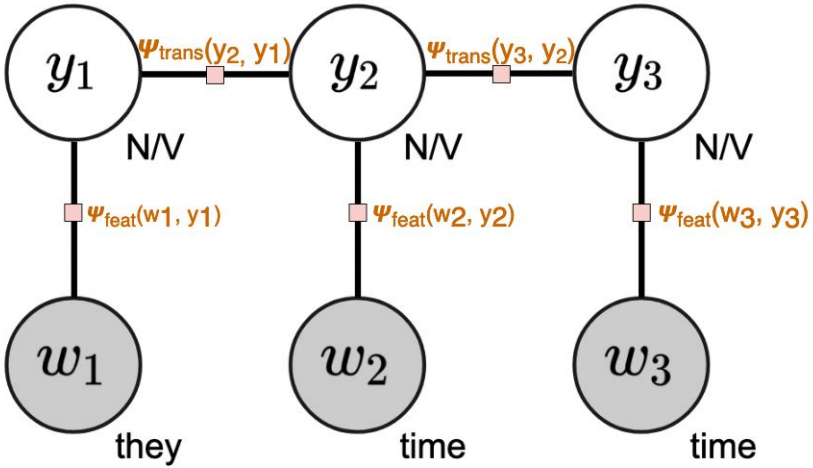
Last time we talked about a particular structure of CRF

The get-score-and-then-normalize approach works for other discriminative (non-CRF) models too!

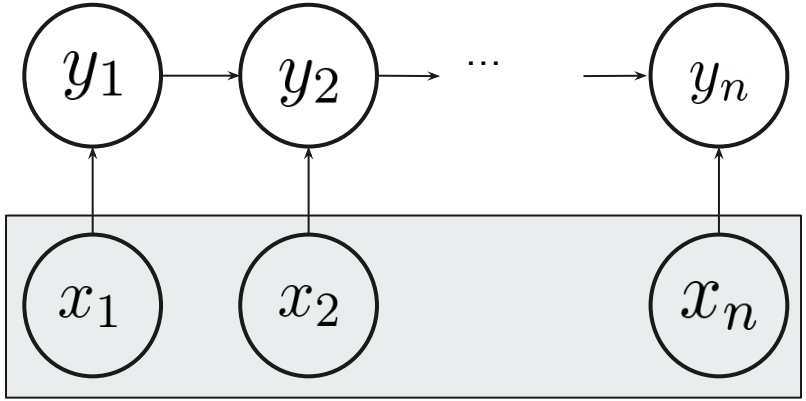


Switching from our CRF to a Maximum Entropy Markov Model

The get-score-and-then-normalize approach works for other discriminative (non-CRF) models too!

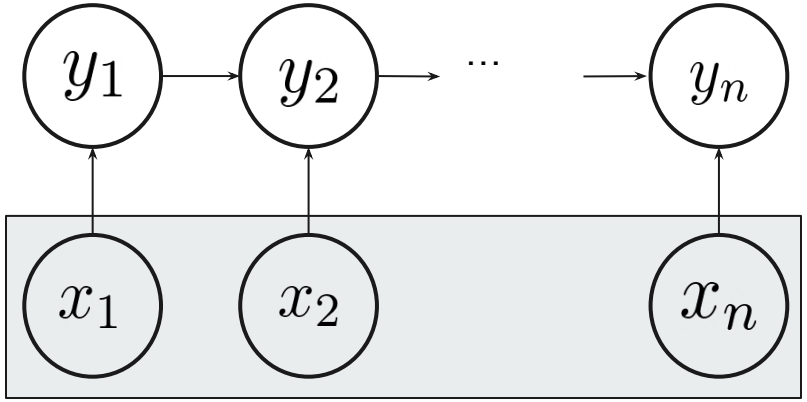


CRF



MEMM

What's changed?



$$p_{\text{MEMM}}(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p(y_t|y_{t-1}, \mathbf{x})$$

$$p(y_t|y_{t-1}, \mathbf{x}) = \frac{1}{Z_t(y_{t-1}, \mathbf{x})} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}$$

$$Z_t(y_{t-1}, \mathbf{x}) = \sum_{y'} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y', y_{t-1}, \mathbf{x}_t)\right\}$$

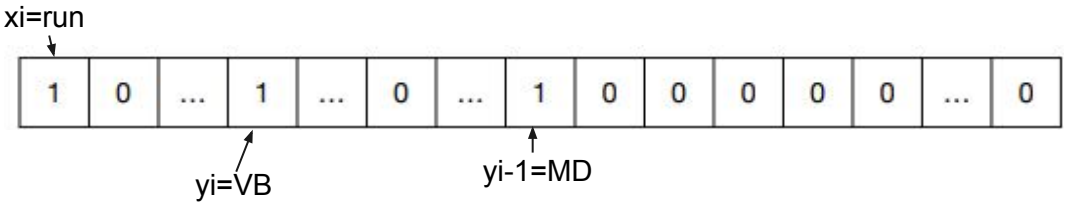
LEARNING.

We're back to learning a probability distribution over **next states**, not over only entire sequences.

Denominator involves only [number of features] terms– no forward algorithm necessary.

How to define features

$\phi(y_i, y_{i-1}, \mathbf{x}_t) =$



- What is the current word, xi?
  - Number of features: size of vocabulary

Example: I will run.

- What is the previous label y_{i-1}
  - Number of features: total number of tags

Basically, for MEMMs, any combination of information about y_i, y_{i-1}, and x_t you want.

Local Normalization to Global Normalization

Conditional Random Fields

- If we do global normalization, we get back to "conditional random fields" or CRFs.

$$\prod_{i=1}^{n} \frac{e^{\mathbf{w}\cdot\phi(y_i,y_{i-1},\mathbf{x})}}{\sum_{i=1}^{C} e^{\mathbf{w}\cdot\phi(y_i,y_{i-1},\mathbf{x})}} \longrightarrow p(\mathbf{y}|\mathbf{x}) = \frac{e^{\mathbf{w}\cdot\Phi(\mathbf{y},\mathbf{x})}}{\sum_{\mathbf{y}'} e^{\mathbf{w}\cdot\Phi(\mathbf{y},\mathbf{x})}}$$

Let’s revisit features

“Wait, you’re telling us THIS is a CRF now??”

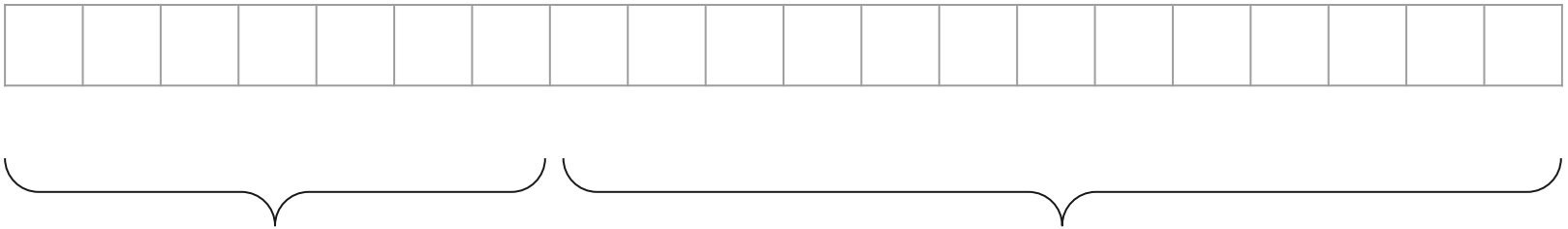
$$p(\mathbf{y}|\mathbf{x}) = \frac{e^{\mathbf{w}\cdot\Phi(\mathbf{y},\mathbf{x})}}{\sum_{\mathbf{y}'} e^{\mathbf{w}\cdot\Phi(\mathbf{y},\mathbf{x})}}$$

“We talked about CRFs on Friday! This doesn’t look like a sum of ψ_{trans}(yi-1, yi) and ψ_{feat}(xi, yi) terms!!” 😱

Let me explain!

On Friday, we were implicitly working with a feature vector I hadn't told you about:

Any hypothetical pairing of tags with our observed tokens got boiled down to the feature vector



(tag, tag) cells, one per possible (tag, tag) pairing
(where each holds the count of (tag, tag) transitions in our sequence of tags)

(token, tag) cells, one per possible (token, tag) pairing
(where each holds the count of (token, tag) emissions in our pairing of tags with our observed tokens)

Notice that this is equivalent to the CRF we talked about on Friday!

$$p(\mathbf{y}|\mathbf{x}) = \frac{e^{\mathbf{w}\cdot\Phi(\mathbf{y},\mathbf{x})}}{\sum_{\mathbf{y}'} e^{\mathbf{w}\cdot\Phi(\mathbf{y},\mathbf{x})}}$$

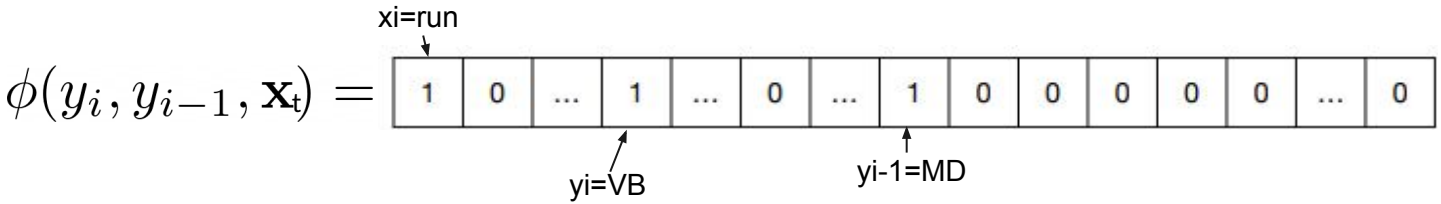
Any hypothetical pairing of tags with our observed tokens got boiled down to the feature vector

(tag, tag) cells, one per possible (tag, tag) pairing
(where each holds the count of (tag, tag) transitions in our sequence of tags)

(token, tag) cells, one per possible (token, tag) pairing
(where each holds the count of (token, tag) emissions in our pairing of tags with our observed tokens)

Now that’s got us thinking about additional options for features…

Features we gave as examples for a MEMM



- What is the current word, xi?
  - Number of features: size of vocabulary

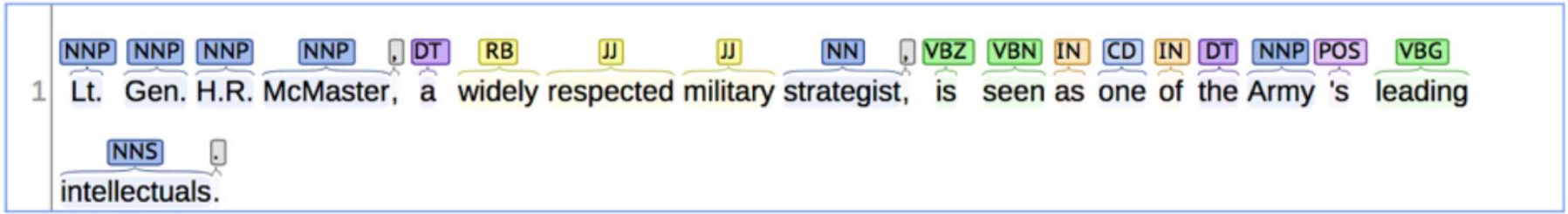
Example: I will run.

- What is the previous label y_{i-1}
  - Number of features: total number of tags

Recap: Sequence Labeling

Input $\mathbf{x} = (x_1, \ldots, x_n)$ Output $\mathbf{y} = (y_1, \ldots, y_n)$

Part-of-Speech:



What if we encounter a word we haven't seen? Could we still capture some information about that word?

More interesting (base) features

I will absolutely friend you on Facebook.

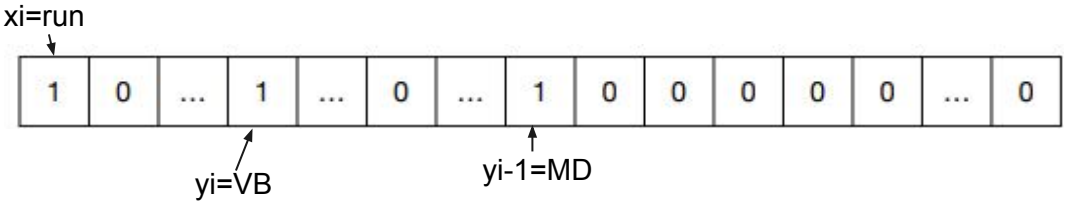
- Is the current word capitalized?
- Does the current (or previous) word end in -ly, -ed, …
- Does the current word contain digits, or a period?

And then consider every combination of these with each possible tag

How do we combine these feature vectors across multiple time steps?

Just add the feature vectors together.

$$\phi(y_i, y_{i-1}, \mathbf{x}_t) =$$



$$\Phi(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{n} \phi(y_i, y_{i-1}, \mathbf{x})$$

Global Feature vector

Local feature vector

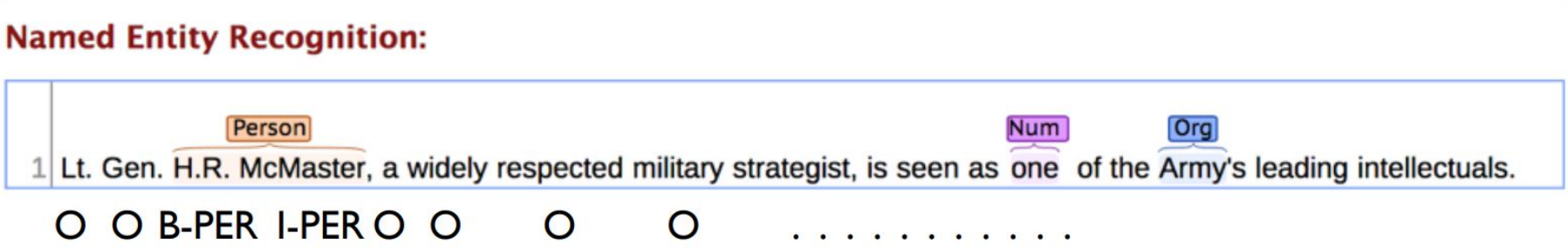
There's yet another type of feature we haven't considered... *context*

For example:

- What was the previous word x_{i-1}?

Consider a less informative tag set

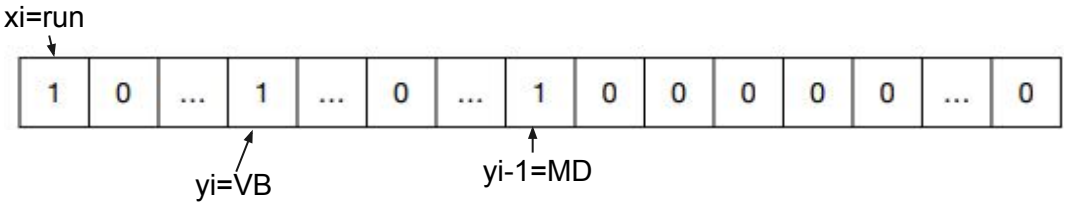
For example, BIO tagging. (Beginning, Inside, Outside)



Here, just crossing features of individual words with our really limited tag set might not be enough. We probably want information about the surrounding context of words!

One way of adding information about context…

$\phi(y_i, y_{i-1}, \mathbf{x}) =$



- What is the current word, xi?
  - Number of features: size of vocabulary

Example: I will run.

- What is the previous label y_{i-1}
  - Number of features: total number of tags

- What is the previous word x_{i-1} … ?
  - Number of features: size of vocabulary

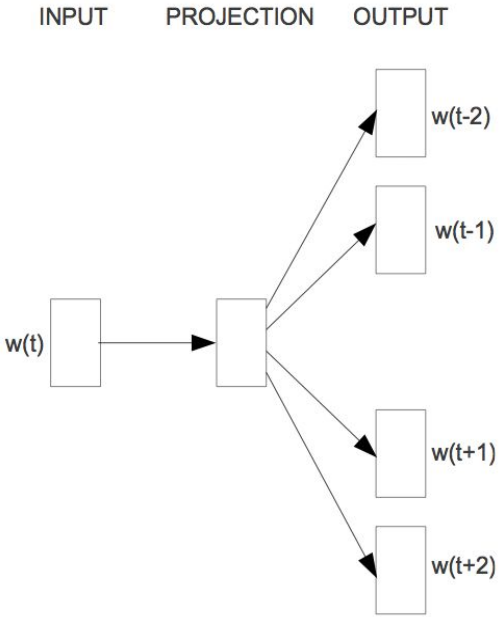
But that vector will be *enormous!*

We don’t want to have to learn that many different parameters!

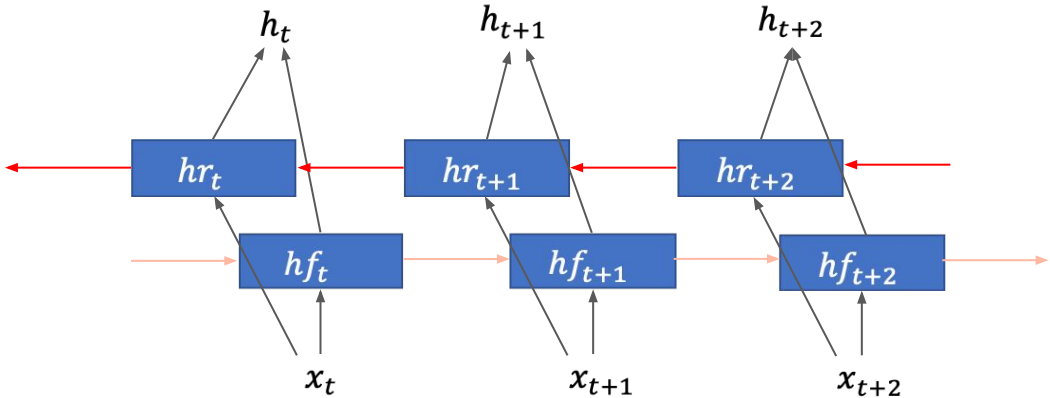
What if there were a more condensed way of creating a vector that added information about the context of a word?

… what does that ring a bell from?

Word embeddings and neural networks that use them!



Skip-gram



Features can also be learned

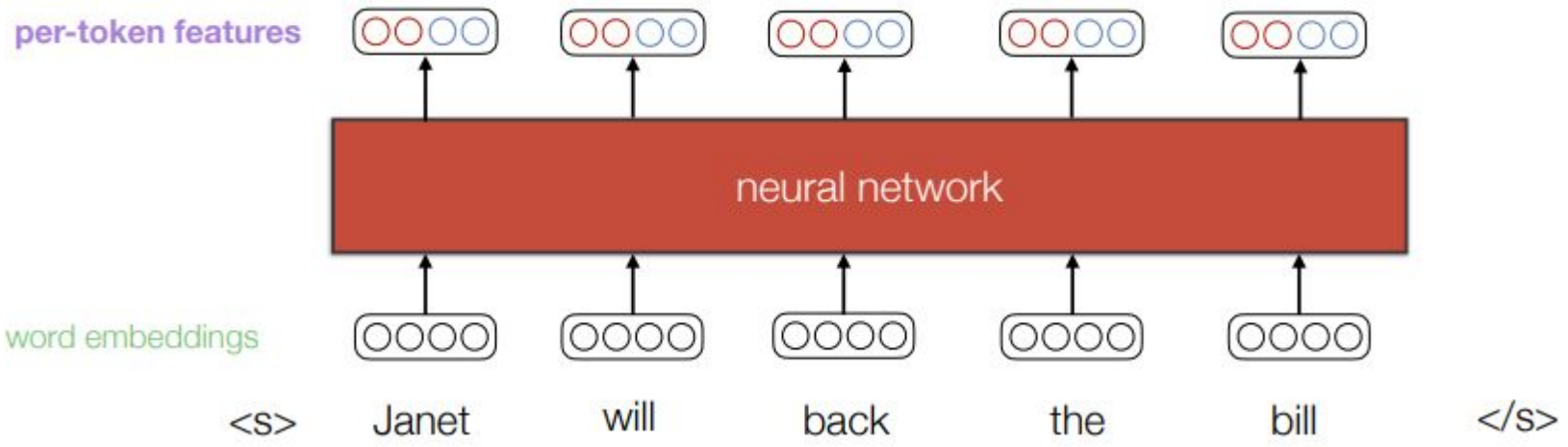
Using neural networks.

Basic Idea: encode the sequence of words into a sequence of vectors

Features can also be learned

Using neural networks.

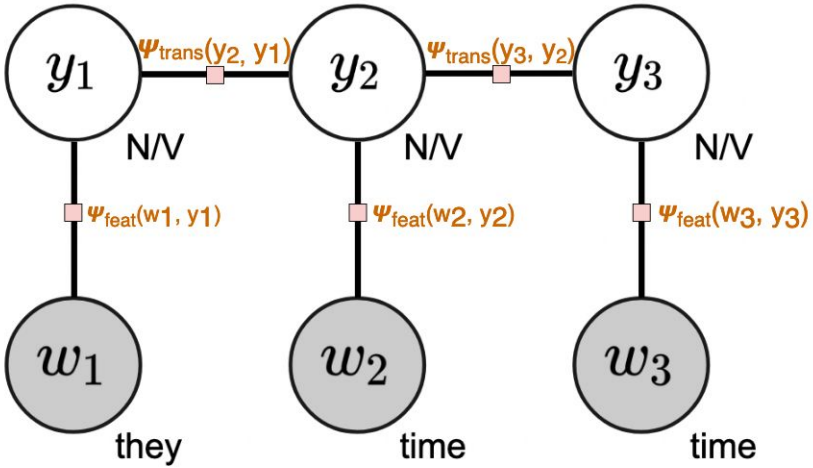
Basic Idea: encode the sequence of words into a sequence of vectors

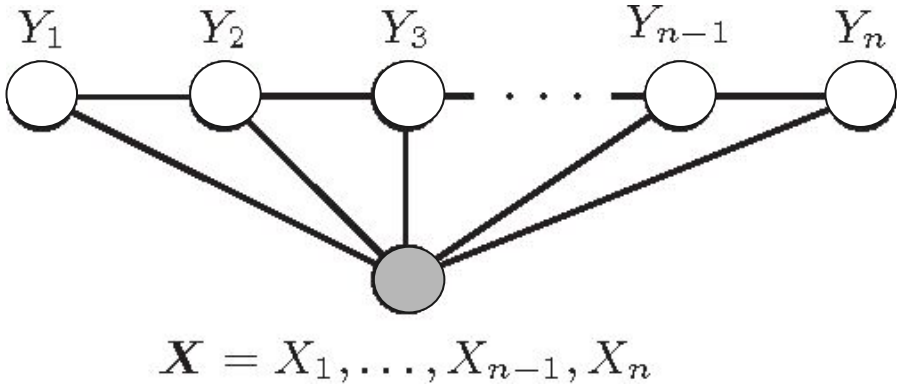


Implications of this switch for inference and learning

Does introducing a dependence between tokens affect our graphical model?

... Yes.

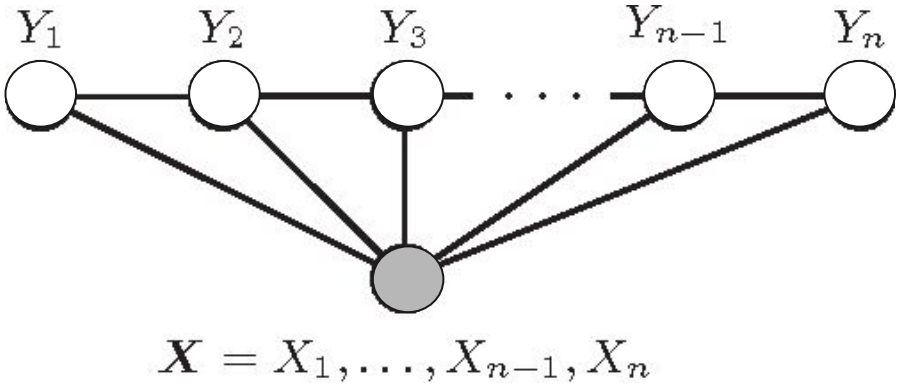


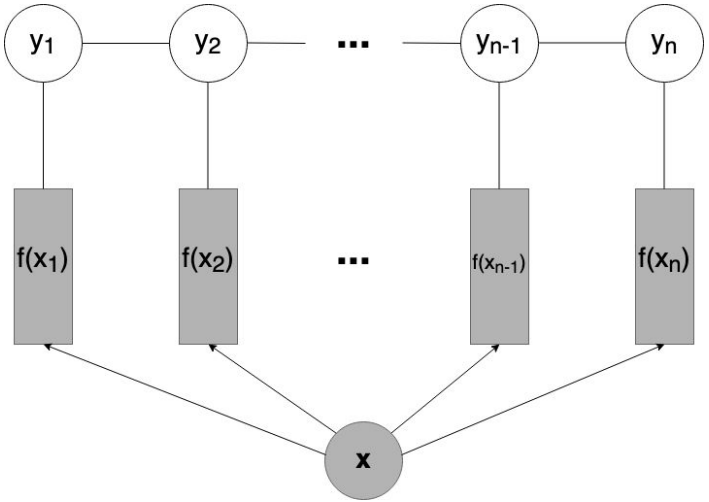


Does this change in the graphical model change our strategy for inference?

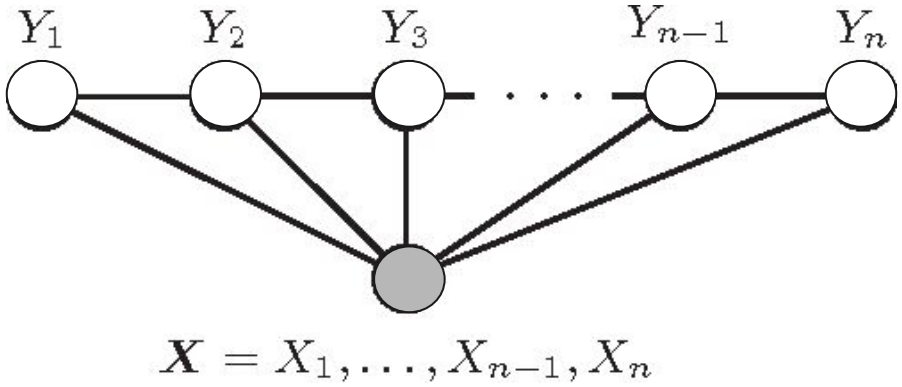
We're used to thinking of Viterbi as finding us the best *path*, but it looks like there are now cycles!! Is this a problem??

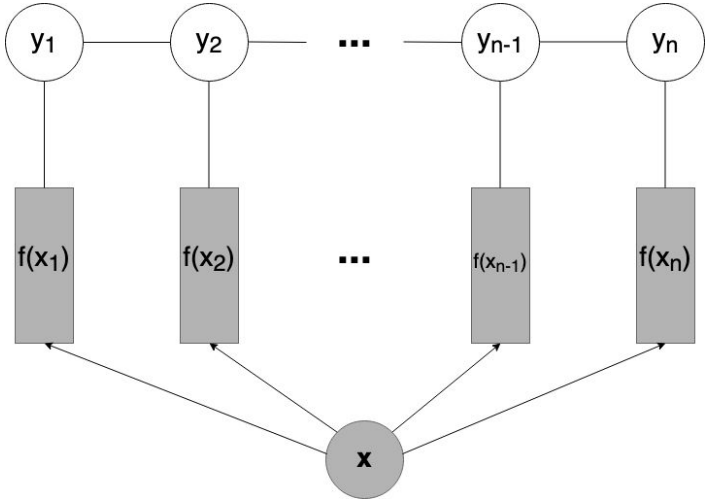
No, because any cycle in our graph goes through observed variables in our data (in other words, the variables that we're not trying to infer values for– our input tokens).





Does this change in the graphical model change our strategy for inference?





Given observed tokens, $f(x_i)$ is *constant*.

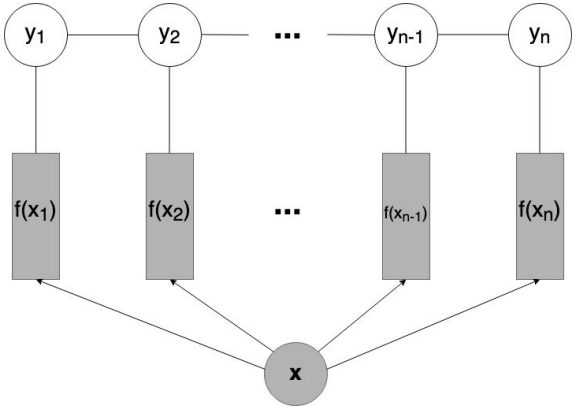
(In A2, each $f(x_i)$ is an input to BiLSTM.hidden2tag)

→ As long as we process our sequence of hidden variables from one end to the other (as usual), Viterbi is still guaranteed to find us the best-scoring sequence.

Does this change in the graphical model change our strategy for inference?

Nope! Still Viterbi algorithm.

Does this change in the graphical model change our strategy for learning?



"Ok but wouldn't this be a problem for learning? If we're learning parameters in our featurizing function, they're definitely not constant then!"

True, but… the same is true when we're moving all the parameters of a deep neural network at once, and that doesn't stop us from using gradient descent then!

So, provided we take reasonably sized small steps after calculating each gradient, we're still good to use the forward algorithm and then backpropagate.

Does this change in the graphical model change our strategy for learning?

Nope! Still Forward algorithm (just with extra backpropagation that goes further back into the network).

What kinds of featurizing functions can we use?

We have lots of options!

- Feedforward neural networks, perhaps including k tokens to either side of current token
- Convolutional neural networks
- RNNs, including BiRNNs
- Transformers

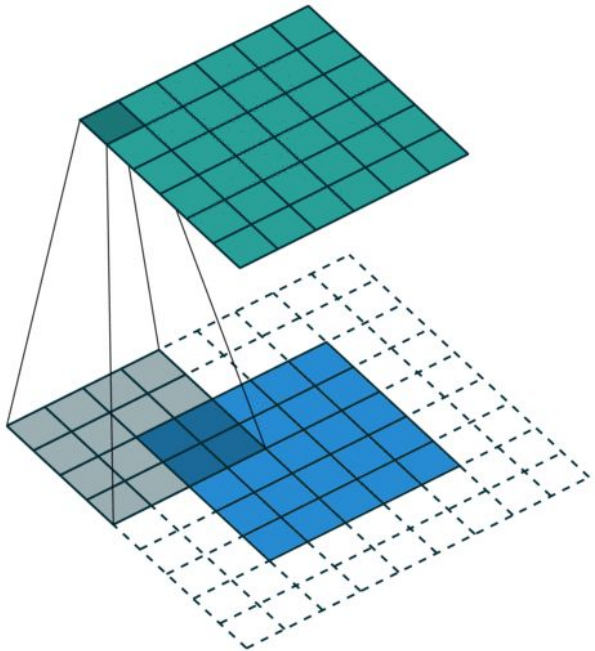
Convolutional neural networks (CNNs)

Commonly used in computer vision models.

Idea: slide a linear learned function (a “kernel”) over cross sections of your input (in our case, a matrix of word embeddings representing the sequence tokens)

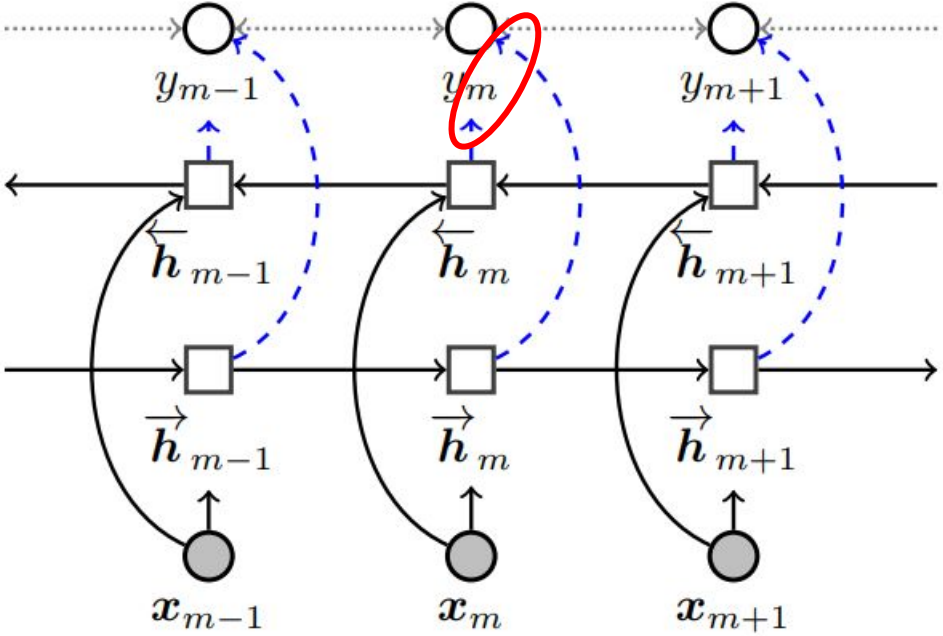
- For NLP purposes, usually sweep over whole word vectors for parts of the sequence

Can do a bunch of these in parallel and max pool across them or average pool across them



Visualization by vdumoulin

Bi-RNN-CRF


y_{m-1}
y_m
y_{m+1}
$\overleftarrow{h}_{m-1}$
$\overleftarrow{h}_m$
$\overleftarrow{h}_{m+1}$
$\overrightarrow{h}_{m-1}$
$\overrightarrow{h}_m$
$\overrightarrow{h}_{m+1}$
x_{m-1}
x_m
x_{m+1}

To summarize:

CRFs:

$$p(\mathbf{y}|\mathbf{x}) = \frac{e^{\mathbf{w}\cdot\Phi(\mathbf{y},\mathbf{x})}}{\sum_{\mathbf{y}'} e^{\mathbf{w}\cdot\Phi(\mathbf{y},\mathbf{x})}} \qquad \Phi(\mathbf{y},\mathbf{x}) = \sum_{i=1}^{n} \phi(y_i, y_{i-1}, \mathbf{x})$$

Features:

- Hand engineered or based on neural networks. (Though you have a similar sort of choice for how to featurize tokens for other discriminative graphical models!)

Training:

- Cross Entropy Loss (and Gradient Descent) + Forward Algorithm

Decoding

- Viterbi Algorithm