Sequence Labeling/Modeling (II)

CS 685, Spring 2021

Advanced Topics in Natural Language Processing

http://brenocon.com/cs685

https://people.cs.umass.edu/~brenocon/cs685_s21/

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

Feedback

- Comment complaining non-neural content is "outdated"
  - Sorry I disagree! As you know, this course takes an integrative approach to NLP.

Today

- Hidden Markov model, continued
    - HMM as LM: forward algo, neural HMM
    - HMMs vs RNNs
- Viterbi decoding - HMM as sequence labeler
- Conditional random fields - discriminative HMM

HMM forward inference

$$p(\mathbf{w}, \mathbf{y}) = \prod_t p(y_t \mid y_{t-1}) p(w_t \mid y_t)$$

- $P(w_1..w_T)$ whole-sentence scoring
- $P(w_t \mid w_1..w_{t-1})$ next-word prediction

$$= \sum_{y_t} P(w_t, y_t \mid w_1 \dots w_{t-1})$$

$$= \sum_{y_t, y_1 \dots y_{t-1}} P(w_t, y_t \mid w_1 \dots w_{t-1}) P(y_1 \dots y_{t-1} \mid w_1 \dots w_{t-1})$$

$$= P(w_t \mid y_t, w_1 \dots w_{t-1}) P(y_t \mid w_1 \dots w_{t-1})$$

Update after lec

Update after lecture

online fwd algo for next-word prediction

$$P(w_t \mid w_1 .. w_{t-1})$$

$$= \sum_{y_t} \underbrace{P(w_t \mid y_t, w_1 .. w_{t-1})}_{= P(w_t \mid y_t)} \underbrace{P(y_t \mid w_1 .. w_{t-1})}_{\text{recursively apply Fwd algo}}$$

$= P(w_t \mid y_t)$ by cond. indep. assumption

Emission Prob!

recursively apply Fwd algo

$= \alpha_t [y_t]$

Forward algo. as trellis

Dynamic Programming!

$P(\vec{w}) = \sum_{\vec{y} \in Cats^T} P(\vec{w}, \vec{y})$ $\quad (TK^2)$ time

$= \prod_t P(y_t | y_{t-1}) P(w_t | y_t)$

$\sum_{y_1} \sum_{y_2} \dots \sum_{y_T} [P(y_1|y_0) P(w_1|y_1)][P(y_2|y_1) P(w_2|y_2)] \dots$

$= \sum_{y_1 \in Cats} P(y_1|y_0) P(w_1|y_1) \sum_{y_2} P(y_2|y_1) P(w_2|y_2) \sum_{y_3} \dots$

Unsup. HMM for LM

- First-order HMM with thousands of states, and neural nets for transition & emission probs; block structure for inference efficiency
- Inference: forward algo for next-word prediction
- Learning: gradient of log marginal probability, **log p(x)** (alternative or related to EM; see their EMNLP 2018 tutorial, sec 5)

$$p(\mathbf{x}, \mathbf{z}; \theta) = \prod_{t=1}^{T} \overbrace{p(x_t \mid z_t)}^{O}\overbrace{p(z_t \mid z_{t-1})}^{A}. \quad (1)$$

Our parameterization uses an embedding for each state in $\mathcal{Z}$ ($\mathbf{E}_z \in \mathbb{R}^{|\mathcal{Z}| \times h}$) and each token in $\mathcal{X}$ ($\mathbf{E}_x \in \mathbb{R}^{|\mathcal{X}| \times h}$). From these we can create representations for leaving and entering a state, as well as emitting a word:

$$\mathbf{H}_{\text{out}}, \mathbf{H}_{\text{in}}, \mathbf{H}_{\text{emit}} = \text{MLP}(\mathbf{E}_z)$$

with all in $\mathbb{R}^{|\mathcal{Z}| \times h}$. The HMM distributional parameters are then computed as,[4]

$$\mathbf{O} \propto \exp(\mathbf{H}_{\text{emit}} \mathbf{E}_x^\top) \qquad \mathbf{A} \propto \exp(\mathbf{H}_{\text{in}} \mathbf{H}_{\text{out}}^\top)$$

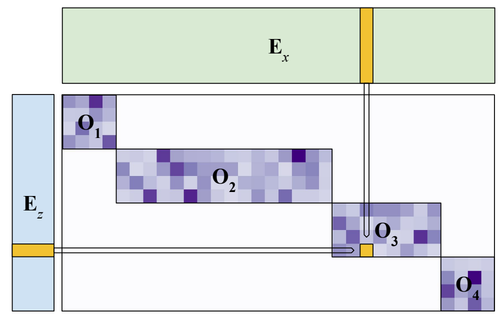


Figure 1: The emission matrix as a set of blocks $\mathbf{O}_1, \ldots, \mathbf{O}_4$ with fixed number of states k. The columns of each block may vary, as there is no constraint on the number of words a state can emit. Each non-zero cell is constructed from an MLP applied to word $\mathbf{E}_x$ and state $\mathbf{E}_z$ embeddings.

[Chiu and Rush 2020]

Model	Param	Val	Test
Penn Treebank			
KN 5-gram	2M	-	141.2
AWD-LSTM	24M	60.0	57.3
256 FF 5-gram	2.9M	159.9	152.0
2x256 dim LSTM	3.6M	93.6	88.8
HMM+RNN	10M	142.3	-
HMM $\lvert\mathcal{Z}\rvert = 900$	10M	284.6	-
VL-HMM $\lvert\mathcal{Z}\rvert = 2^{15}$	11.4M	125.0	116.0
WikiText			
KN 5-gram	5.7M	248.7	234.3
AWD-LSTM	33M	68.6	65.8
256 FF 5-gram	8.8M	210.9	195.0
2x256 LSTM	9.6M	124.5	117.5
VL-HMM $\lvert\mathcal{Z}\rvert = 2^{15}$	17.3M	166.6	158.2

Table 1: Perplexities on PTB / WikiText-2. The HMM+RNN and HMM of Buys et al. (2018) reported validation perplexity only for PTB.

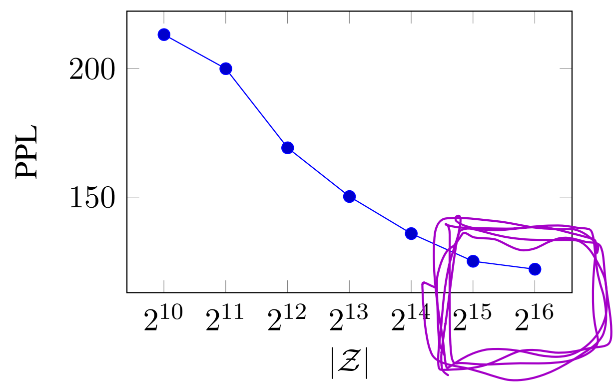
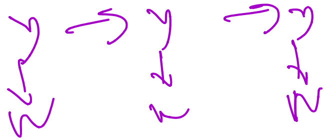


Figure 3: Perplexity on PTB by state size $\lvert\mathcal{Z}\rvert$ ($\lambda = 0.5$ and $M = 128$).

Are they related?

Kinda!

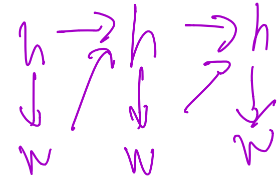
HMM



Prob. graph

RNN

Computation graph



State

Repr: $y \in \{1..K\}$

Inference: uncertainty over which state is at t

$P(y_t | w_1 ... w_{t-1}) \in S(K) \subset \mathbb{R}^K$

$P(y_t | w_1 ... w_T)$

State

Repr: $\vec{h} \in \mathbb{R}^K$

Inference? Conflated or the same

$\vec{h}$ is deterministic

$P(\vec{h}_{t+1} | w_1 ... w_t)$ = determ.

Viterbi algorithm

- Goal: given entire input sequence $w_1..w_T$, jointly predict best output sequence $y_1..y_T$
  - Why can't you do simply do this left-to-right?

$$\operatorname*{argmax}_{\vec{y}} P(\vec{y} \mid \vec{w}) \propto P(\vec{w}, \vec{y})$$

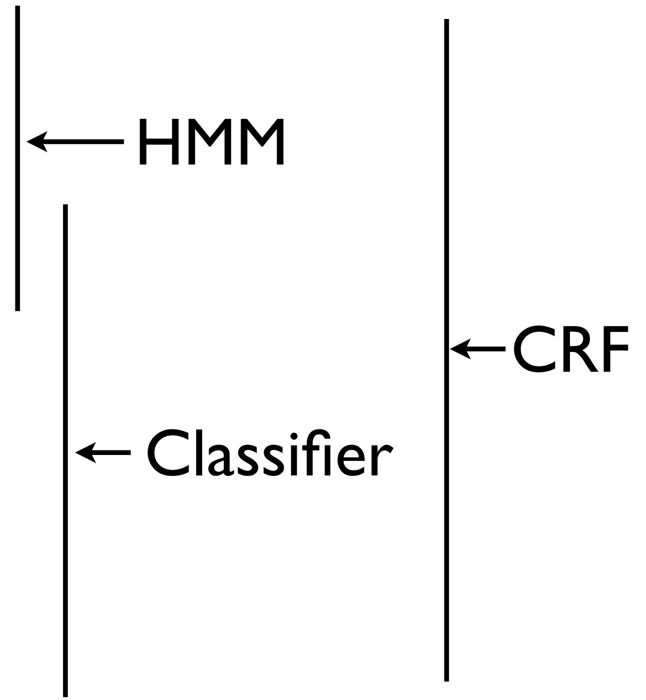
Left-to-right decoder "greedy"

$y_1 =$ VERB NOUN ??? $y_2 =$ PREP

$w_1 =$ Attack $w_2 =$ at

How to build a POS tagger?

- Sources of information:
  - POS tags of surrounding words: syntactic context
  - The word itself
  - Features, etc.!
    - Word-internal information
    - Features from surrounding words
    - External lexicons
    - Embeddings, NN states



[BERT/ELMO may be sufficient alternatives to sharing contextual information?]

- Seq. labeling as log-linear ***structured prediction***

$$\hat{\boldsymbol{y}}_{1:M} = \underset{\boldsymbol{y}_{1:M} \in \mathcal{Y}(\boldsymbol{w}_{1:M})}{\operatorname{argmax}} \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}_{1:M}, \boldsymbol{y}_{1:M}),$$

- Example: the ***Hidden Markov model***

$$p(\mathbf{w}, \mathbf{y}) = \prod_t p(y_t \mid y_{t-1}) p(w_t \mid y_t)$$

  - w: Text Data
  - y: Proposed class or sequence
  - θ: Feature weights (model parameters)
  - f(x,y): Feature extractor, produces feature vector

- Seq. labeling as log-linear ***structured prediction***

$$\hat{\boldsymbol{y}}_{1:M} = \underset{\boldsymbol{y}_{1:M} \in \mathcal{Y}(\boldsymbol{w}_{1:M})}{\operatorname{argmax}} \quad \boldsymbol{\theta}^\top \boldsymbol{f}(\boldsymbol{w}_{1:M}, \boldsymbol{y}_{1:M}),$$

- Example: the ***Hidden Markov model***

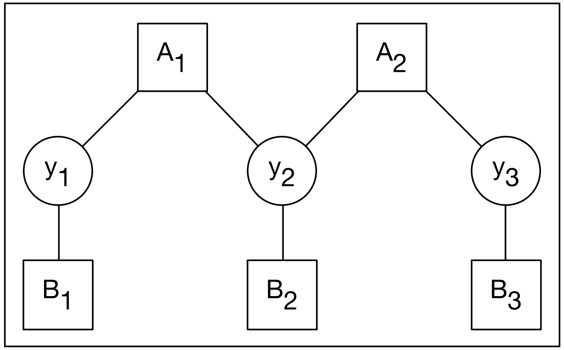
$$p(\mathbf{w}, \mathbf{y}) = \prod_t p(y_t \mid y_{t-1}) p(w_t \mid y_t)$$

- Next: ***Conditional Random Fields***

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp(\theta^\mathsf{T} \mathbf{f}(\mathbf{w}_{1:M}, \mathbf{y}_{1:M}))}{\sum_{\mathbf{y}'_{1:M} \in \mathcal{Y}(\mathbf{w}_{1:M})} \exp(\theta^\mathsf{T} \mathbf{f}(\mathbf{w}_{1:M}, \mathbf{y}'_{1:M}))}$$

* for carefully chosen **f**

HMM as log-linear



$$p(y, w) = \prod_t p(w_y|y_t)\ p(y_t|y_{t-1})$$

$$\underbrace{\log p(y, w)}_{\substack{G(y)\\ \textit{goodness}}} = \sum_t \underbrace{\log p(w_t|y_t)}_{\substack{B_t(y_t)\\ \textit{emission factor score}}} + \underbrace{\log p(y_t|y_{t-1})}_{\substack{A(y_{t-1},\, y_t)\\ \textit{transition factor score}}}$$

$$\log p(y, w) = \sum_t \underbrace{\phi_t(y_{t-1}, y_t)}_{\textit{pair factor score}}$$

Decoding problem
(Viterbi algorithm) $\arg \max_{y^* \in outputs(x)} G(y^*)$

HMM as log-linear

- HMM as a joint log-linear model

$$P(y, w) = \prod_t P(y_t \mid y_{t-1}) P(w_t \mid y_t)$$

$$P(y, w) = \exp(\theta^\mathsf{T} f(y, w))$$

$$f(y, w) = \sum_t f(y_{t-1}, y_t, w_t)$$

Local features only!
(Allows efficient inference)

e.g. {(N,V):1, (V,dog):1}
What are the weights?

- This implies the conditional is also log-linear

$$P(y \mid w) \propto \exp(\theta^\mathsf{T} f(y, w))$$

From HMMs to CRFs

- **1. Discriminative learning**: take HMM features, but set weights to maximize *conditional LL* of labels
- **2. More features**: affix, positional, feature templates, embeddings, etc.
  - For efficient inference: make sure to **preserve Markovian structure** within the feature function (e.g. first-order CRF)

		finna	bless	us
y =	**[S]**	**V**	**V**	**V**

Tags: "**V**"erb and pr"**O**"noun (and **[S]**tart)

Let's use three feature templates:

Transition features: for example $f_{VV}(x,y)$ = number of V-V transitions in y

Word-tag observation features: for example $f_{V,dog}(x,y)$ = number of tokens that are word "dog" under a Verb tag

"ends with s"–tag features: $f_{V\text{-}s}(x,y)$ = number of tokens that end with -s and are tagged as Verb

(Global features have to be COUNTS: the reason why is further below.)
For 3 word vocabulary and 2 tag types, that's J=14 total features.
Assume we have fixed model weights θ and would like to score the goodness of the above tag sequence.

Global feature vector f(x,y) =

f_{SV}	f_{SO}	f_{VV}	f_{VO}	f_{OV}	f_{OO}	$f_{V,finna}$	$f_{V,bless}$	$f_{V,us}$	$f_{O,finna}$	$f_{O,bless}$	$f_{O,us}$	$f_{V,-s}$	$f_{O,-s}$
1	0	2	0	0	0	1	1	1	0	0	0	2	0

Model parameters θ =

θ_{SV}	θ_{SO}	θ_{VV}	θ_{VO}	θ_{OV}	θ_{OO}	$\theta_{V,finna}$	$\theta_{V,bless}$	$\theta_{V,us}$	$\theta_{O,finna}$	$\theta_{O,bless}$	$\theta_{O,us}$	$\theta_{V,-s}$	$\theta_{O,-s}$
-0.2	-0.8	+0.1	+0.5	+4.3	-0.3	-1.2	-0.1	+0.1	+5.3	-4.1	-0.3	+1.1	+2.2

Goodness score G(y) $= \theta' f(x,y) = \sum_{j=1}^{J} \theta_j f_j(x,y)$

$= \; -0.2 + 0 + 0.2 + 0 + 0 + 0 \;\; -1.2 + 0 + 0.1 + 5.3 + 0 + 0 + 2.2 + 0$

Global feature vector is from the sum of local feature vectors

$$f(x,y) = \sum_t f_t(y_{t-1}, y_t, x_t)$$

$f_t(y_{t-1}, y_t, x_t) =$ local feature vector including the transition between these two tags, and the observation of word at position t.

The local features are, for example:

f_{VV}(yprev, ycur, curword) = {1 if yprev=V and ycur=V, else 0}

$f_{V,dog}$(yprev, ycur, curword) = {1 if ycur=V and curword="dog", else 0}

$f_{V,-s}$(yprev, ycur, curword) = {1 if ycur=V and curword ends in "s", else 0}

And so on, repeated for different tags and words.

Example

	fSV	fSO	fVV	fVO	fOV	fOO	fV,finna	fV,bless	fV,us	fO,finna	fO,bless	fO,us	fV,-s	fO,-s
	........trans. feats.........						...............obs. feats...............							
f(START, V, finna)	1	0	0	0	0	0	1	0	0	0	0	0	0	0
+ f(V, V, bless)	0	0	1	0	0	0	0	1	0	0	0	0	1	0
+ f(V, V, us)	0	0	1	0	0	0	0	0	1	0	0	0	1	0
= f(x=finna bless us, y=V V V) =	1	0	2	0	0	0	1	1	1	0	0	0	2	0

Local feature decomposition implies that the scoring function decomposes, too.

$$G(y) = \theta' f(x,y) = \theta' \sum_t f_t(y_{t-1}, y_t, x_t) = \sum_t \theta' f_t(y_{t-1}, y_t, x_t)$$

= θ'f(START, V, finna) + θ'f(V, V, bless) + θ'f(V, V, us)

= dotprod (

-0.2	-0.8	+0.1	+0.5	+4.3	-0.3	-1.2	-0.1	+0.1	+5.3	-4.1	-0.3	+1.1	+2.2
1	0	0	0	0	0	1	0	0	0	0	0	0	0

)

\+ dotprod (

-0.2	-0.8	+0.1	+0.5	+4.3	-0.3	-1.2	-0.1	+0.1	+5.3	-4.1	-0.3	+1.1	+2.2
0	0	1	0	0	0	0	1	0	0	0	0	1	0

)

\+ dotprod (

-0.2	-0.8	+0.1	+0.5	+4.3	-0.3	-1.2	-0.1	+0.1	+5.3	-4.1	-0.3	+1.1	+2.2
0	0	1	0	0	0	0	0	1	0	0	0	1	0

)

Learning a CRF

- Gradient descent on negative **conditional LL**
  - Log-linear gradient:
    sum over all possible predicted structures
    (Forward-Backward for marginalization)
- Non-probabilistic losses: compare gold structure to only one predicted structure
  - Structured perceptron algorithm:
    Collins, 2002 (recent Test of Time award)
  - Structured SVM (hinge loss)
  - (Viterbi for best-structure)

Learning a CRF: max CLL

$$\log p_\theta(y \mid w) = \theta^\mathsf{T} f(y,w) - \log \sum_{y'} \exp(\theta^\mathsf{T} f(y,w))$$

$$\frac{\partial \log p_\theta(...)}{\partial \theta_j} = f_j(y,w) - \sum_{y'} p_\theta(y' \mid w) f_j(y',w)$$

- Apply local decomposition

$$= \left(\sum_t f_j(y_{t-1}, y_t, w_t) \right) - \sum_{y'} p_\theta(y' \mid w) \sum_t f_j(y'_{t-1}, y'_t, w_t)$$

Real feature value

Expected feature value

$$= \sum_t \left(f_j(y_{t-1}, y_t, w_t) - \sum_{y'_t, y'_{t-1}} p_\theta(y'_{t-1}, y'_t \mid w) f_j(y'_{t-1}, y'_t, w_t) \right)$$

Tag marginals (to compute: forward-backward)

Seq. Labeling inference

- P(w): Likelihood (generative model)
  - Forward algorithm. Each step: *sum* over all possible prefixes
- P(y | w): Predicted sequence ("decoding")
  - Viterbi algorithm. Each step: consider each best possible prefix
  - Need for **supervised struct perceptron / SSVM** learning
- $P(y_m \mid w)$ and $P(y_m, y_{m-1} \mid w)$:
  Predicted tag (and tag pair) marginals
  - Forward-Backward algorithm
  - Need for **supervised CRF** learning
  - Need for **unsupervised HMM** learning

Forward-Backward

Want: a pair marginal

$$\Pr(Y_{m-1} = k', Y_m = k \mid \boldsymbol{w}) = \frac{\sum_{\boldsymbol{y}:Y_m=k, Y_{m-1}=k'} \prod_{n=1}^{M} \exp s_n(y_n, y_{n-1})}{\sum_{\boldsymbol{y}'} \prod_{n=1}^{M} \exp s_n(y'_n, y'_{n-1})}.$$

To get: sum out left/right-side paths

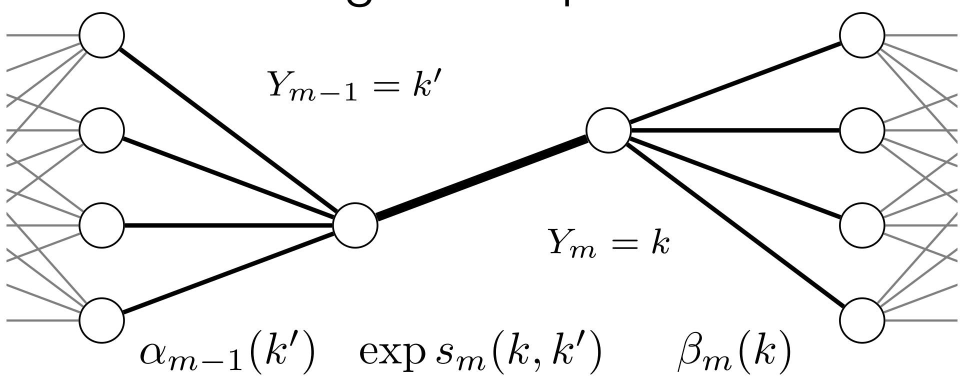


Figure 7.3: A schematic illustration of the computation of the marginal probability $\Pr(Y_{m-1} = k', Y_m = k)$, using the forward score $\alpha_{m-1}(k')$ and the backward score $\beta_m(k)$.

[Diagram: Jacob Eisenstein]

Forward recurrence

$$\alpha_m(y_m) = \sum_{\boldsymbol{y}_{1:m-1}} \prod_{n=1}^{m} \exp s_n(y_n, y_{n-1})$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \sum_{\boldsymbol{y}_{1:m-2}} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1})$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \times \alpha_{m-1}(y_{m-1}).$$

Backward recurrence

$$\beta_m(k) \triangleq \sum_{\boldsymbol{y}_{m:M}:Y_m=k} \prod_{n=m}^{M+1} \exp s_n(y_n, y_{n-1})$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \times \beta_{m+1}(k').$$

Baum-Welch: EM for HMMs

- When complete LL is easy to maximize, as in the simple count-based HMM
  - It's an optimization method for the marginal LL
  - For linear or NN parameterizations, backprop implicitly does an E-step for you; no need for explicit E/M alternation
- **E-step**: calculate marginals with forward-backward
  - $p(y_{t-1}, y_t \mid w_1..w_T)$
  - $p(y_t \mid w_1..w_T)$
- **M-step**: re-estimate parameters from expected counts
  - Transitions: will use pair marginals
  - Emissions: will use tag marginals
- Learns soft clusters kind-of-like POS tags

Structured Perceptron

- Viterbi is very common for decoding. Inconvenient that you also need forward-backward for CRF learning
- Collins 2002: actually you can directly train only using Viterbi: **structured perceptron**
  - Theoretical results hold from the usual perceptron...
- Important extension in NLP: **Structured SVM**
  - a.k.a. **Structured large-margin/hinge-loss** energy network
    a.k.a. **Cost-augmented perceptron**
- SP, SSVM, CRF training are variants of highly related objective functions and SSGD updates

Structured/multiclass Perceptron

(for any log-linear model)

- For ~10 iterations
  - For each (x,y) in dataset
    - PREDICT

$$y^* = \arg\max_{y'} \theta^\mathsf{T} f(x, y')$$

    - IF y=y*, do nothing
    - ELSE update weights

$$\theta := \theta + r[f(x,y) - f(x,y^*)]$$

learning rate constant
e.g. r=1

Features for
TRUE output

Features for
PREDICTED output

Does this look similar to the CRF CLL gradient?

Perceptron notes/issues

- Issue: does it converge? (generally no)
  - Solution: the *averaged* perceptron
- Can you regularize it? No... use SSVM instead (cost-augmented perceptron)
- StructPerc and CRF perform similarly in practice

CRF extensions

- Not just chains
  - 2nd-order, 3rd-order Markov assumptions…
  - Trees…
  - Grids, social networks, etc… any situation where you want interdependence of the latent (predicted) variables
  - Best: a low-treewidth DGM (why?)

Structured Pred. and NNs

Avg structure

- Tradeoffs
  - Complex output model + simple input model? (CRF and linear features)
    vs.
  - Simple output model + complex input model? (Indiv. classifier with LSTM "features")
- Can combine both! (e.g. BiLSTM-CRF)

- RNNs as *alternative* to probabilistic model-based message passing
  - Success of BERT representation + indep classifier suggests BERT (or similar) is already combining significant information
- Other work (e.g. VAEs): NNs for inference