Deep Learning for Natural Language Processing

Conditional Random Fields

UNIVERSITY OF GOTHENBURG

CHALMERS

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Richard Johansson

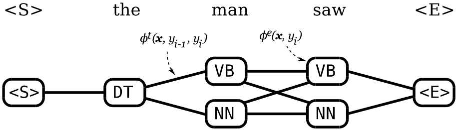
richard.johansson@gu.se

recap

- we saw how to define a function that computes a score for an input $\boldsymbol{x}$ and an output $\boldsymbol{y}$:

$$\text{score}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{L} \phi^e(\boldsymbol{x}, y_i) + \sum_{i=1}^{L} \phi^t(\boldsymbol{x}, y_{i-1}, y_i)$$

- the scoring function is "factorized" into **emission** scores ϕ^e and **transition** scores ϕ^t

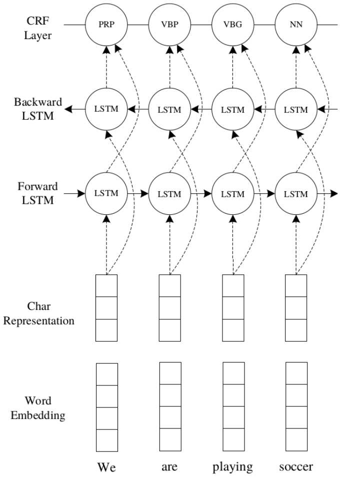


training the scoring function

- there are many training algorithms that can train the scoring function score($\boldsymbol{x}, \boldsymbol{y}$)
- some of them are generalizations of classification models:
  - perceptron $\rightarrow$ structured perceptron (Collins, 2002)
  - SVM $\rightarrow$ structured SVM (Tsochantaridis et al., 2005)
  - logistic regression $\rightarrow$ **conditional random field**

CRFs in NLP systems

- the CRF model was proposed by Lafferty et al. (2001) and has been popular in many NLP tasks since then
- in neural models, a CRF is typically used as the output layer (Ma and Hovy, 2016)



CRF and related models

		output type?	
		category	**sequence**
probability model?	**generative** $P(\boldsymbol{x}, \boldsymbol{y})$	naive Bayes, GMM	hidden Markov model
	discriminative $P(\boldsymbol{y}\|\boldsymbol{x})$	logistic regression	**conditional random field**

CRF: basic definition (Lafferty et al., 2001)

- the CRF defines a probability of an output sequence $\boldsymbol{y}$, given an input sequence $\boldsymbol{x}$, as follows

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp \ \text{score}(\boldsymbol{x}, \boldsymbol{y})}{\sum_{\boldsymbol{y}'} \exp \ \text{score}(\boldsymbol{x}, \boldsymbol{y}')} = \frac{\exp \ \text{score}(\boldsymbol{x}, \boldsymbol{y})}{Z(\boldsymbol{x})}$$

- to train the model, we minimize the negative log likelihood:

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}) = -\sum_{i=1}^{N} \log P(\boldsymbol{y}_i|\boldsymbol{x}_i)$$

CRF: basic definition (Lafferty et al., 2001)

- the CRF defines a probability of an output sequence $\boldsymbol{y}$, given an input sequence $\boldsymbol{x}$, as follows

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp\ \text{score}(\boldsymbol{x}, \boldsymbol{y})}{\sum_{\boldsymbol{y}'} \exp\ \text{score}(\boldsymbol{x}, \boldsymbol{y}')} = \frac{\exp\ \text{score}(\boldsymbol{x}, \boldsymbol{y})}{Z(\boldsymbol{x})}$$

- to train the model, we minimize the negative log likelihood:

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}) = -\sum_{i=1}^{N} \log P(\boldsymbol{y}_i|\boldsymbol{x}_i)$$

- but the probability is a giant softmax!
  - the sum in the denominator goes over all possible sequences!
  - the giant sum is called the **partition function** $Z(\boldsymbol{x})$

requirements for training and prediction

- for **predicting** or **decoding**, we need to compute the top-scoring output sequence $\hat{\boldsymbol{y}}$ for a given input $\boldsymbol{x}$:

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x})$$

- for **training**, we need to compute the log likelihood, including the log of the partition function:

$$\log P(\boldsymbol{y}|\boldsymbol{x}) = \text{score}(\boldsymbol{x}, \boldsymbol{y}) - \log Z(\boldsymbol{x})$$

finding the highest-probability sequence

- the partition function $Z(\boldsymbol{x})$ does not depend on $\boldsymbol{y}$
- so we don't need $Z(\boldsymbol{x})$ to find the top-scoring output:

$$\begin{aligned}
\arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) &= \arg\max_{\boldsymbol{y}} \frac{\exp \ \text{score}(\boldsymbol{x}, \boldsymbol{y})}{Z(\boldsymbol{x})} \\
&= \arg\max_{\boldsymbol{y}} \ \exp \ \text{score}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} \ \text{score}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}$$

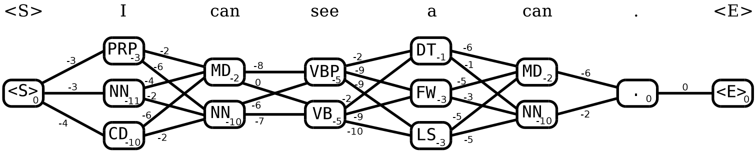
- we've seen that we can use the **Viterbi** algorithm to solve this

the forward algorithm

- the **forward** algorithm computes

$$\log Z(\boldsymbol{x}) = \log \sum_{\boldsymbol{y}} \exp \ \text{score}(\boldsymbol{x}, \boldsymbol{y})$$

- it is a special case of the **sum-product algorithm** for graphical models
- it uses a dynamic programming approach similar to Viterbi
- it computes log sums instead of maximizing



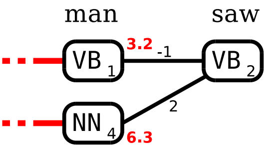
side note about working in the log domain

- when computing $\log Z(\boldsymbol{x})$ in the forward algorithm, we need to work in the **log domain** to avoid numerical overflows
- this is common when implementing probabilistic models in general
  - multiplying probabilities $\Rightarrow$ summing log probabilities
  - summing probabilities $\Rightarrow$ $\log \sum \exp$ with log probabilities
- in PyTorch, `torch.logsumexp` computes $\log \sum \exp$ in a numerically stable way

the forward algorithm: dynamic programming

- assume we have computed the log-sum-exp for all paths in the previous step
  - let $\alpha_{i-1,j}$ be the log-sum-exp of the scores of all paths ending in label j at position $i-1$
- then we compute the α scores in step i as follows:

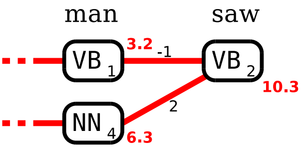
$$\alpha_{ij} = \phi^e(y_i) + \log \sum_k \exp\left[\alpha_{i-1,k} + \phi^t(y_{i-1}, y_i)\right]$$



the forward algorithm: dynamic programming

- assume we have computed the log-sum-exp for all paths in the previous step
  - let $\alpha_{i-1,j}$ be the log-sum-exp of the scores of all paths ending in label j at position $i-1$
- then we compute the α scores in step i as follows:

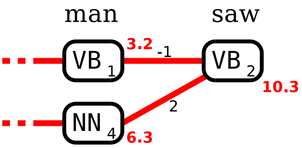
$$\alpha_{ij} = \phi^e(y_i) + \log \sum_k \exp \left[\alpha_{i-1,k} + \phi^t(y_{i-1}, y_i)\right]$$



the forward algorithm: dynamic programming

- assume we have computed the log-sum-exp for all paths in the previous step
  - let $\alpha_{i-1,j}$ be the log-sum-exp of the scores of all paths ending in label j at position $i-1$
- then we compute the α scores in step i as follows:

$$\alpha_{ij} = \phi^e(y_i) + \log \sum_k \exp \left[\alpha_{i-1,k} + \phi^t(y_{i-1}, y_i)\right]$$



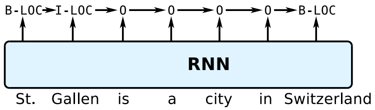
- $\log Z(\boldsymbol{x})$ is the α score for the dummy end token

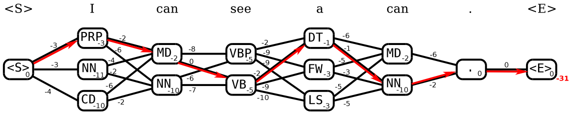
CRF implementations for PyTorch

- PyTorch does not include a CRF module out of the box
- a couple of implementations:
  - **pytorch-crf**: a standalone CRF module
  - **AllenNLP**: the CRF module is part of a larger library
  - **NCRF++**: also part of a library

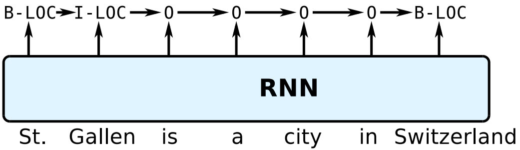
sequence labeling: summary of approaches

- why have we spent so much time on sequence models?
- a fairly simple task, but we could explore different approaches to solving **structured prediction problems**





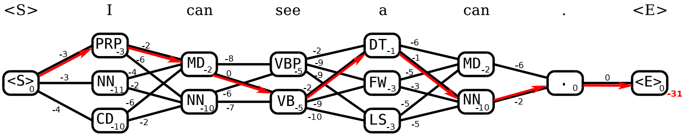
summary: greedy sequential models



- **pros**:
  - a bit more flexible, no need to design special algorithms
  - since we reduce the task to classification, we can rely on standard machinery for classification
- **cons**:
  - algorithm is greedy and does not necessarily find the highest-scoring sequence
  - because prediction is sequential, we can't use any information about future predictions

summary: global scoring approaches

$$\text{score}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{p \in \text{parts}} \text{part-score}(\boldsymbol{x}, p) = \sum_{i=1}^{L} \phi^e(\boldsymbol{x}, y_i) + \sum_{i=1}^{L} \phi^t(\boldsymbol{x}, y_{i-1}, y_i)$$



- **pros**:
  - a bit more accurate (generally)
  - finds the globally best solution, does not go to a dead end
- **cons**:
  - need specialized algorithms for training and prediction
  - computational complexity may be higher

exercise 2

- we will continue our NER experiments

Manchester	United	will	return	to	the	United	States
B-ORG	I-ORG	O	O	O	O	B-LOC	I-LOC

- we will investigate autoregressive models and CRFs

reading

- Goldberg doesn't have a chapter on sequence tagging
  - chapter 19 includes a bit, but also some parts that are more relevant when we discuss parsing
- Eisenstein's chapter 7 covers CRF and related models
  - you can skim or skip the parts on HMMs and structured perceptrons and SVMs
- Reimers and Gurevych (2017b) gives a good overview of different engineering choices
  - ...and more in another paper (Reimers and Gurevych, 2017a)

state of the art for sequence labeling tasks

- Ruder collects state-of-the-art results for many NLP tasks: `https://nlpprogress.com/`
- examples of some of the tasks we have discussed:
  - part-of-speech tagging
  - named entity recognition
  - semantic role labeling
- quite English-centric...

references

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*.

X. Ma and E. Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *ACL*.

N. Reimers and I. Gurevych. 2017a. Optimal hyperparameters for deep LSTM-networks for sequence labeling tasks. arXiv:1707.06799.

N. Reimers and I. Gurevych. 2017b. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *EMNLP*.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. 2005. Large margin methods for structured and interdependent output variables. *JMLR* 6(Sep):1453–1484.