Topics in Natural Language Processing

Shay Cohen

Institute for Language, Cognition and Computation

University of Edinburgh

Lecture 6

Semirings

A flexible framework for denoting the operations and values that a dynamic programme takes.

- Generalises CKY to the inside algorithm
- Can actually have richer semirings, such as ones that compute expected values on the output structure (“the probability that NP spans words 0 through 5”)
- Can be used in tandem with generic solvers of weighted logic programmes

Example of a Weighted Logic Programme

We are given a sequence $w_1, \ldots, w_n$ of some symbols.

$$\text{prob}(b, i) \oplus = \text{prob}(a, i-1) \otimes \text{transition}(a \to b) \otimes \text{emission}(b, w_i)$$

$$\text{prob}(a, 1) \oplus = \text{start\_state}(b) \otimes \text{emission}(a, w_1)$$

Hidden Markov models and the forward algorithm:

- emission are the emission probabilities
- transition are the transition probabilities
- start_state are the initial probabilities
- $\text{prob}(b, i)$ gives the probability $p(w_1, \ldots, w_i, S_i = b)$

Solving Weighted Logic Programmes

- Memoisation and dynamic programming
- Agenda algorithms. Roughly:
  - Keep a queue ("agenda") of unprocessed items and a chart of processed items
  - Dequeue an item x and create or update all items by using information from the chart together with x. Put all the new items in the queue
  - Depending on the order by which we dequeue from the agenda, we might process items several times
  - Examples of agenda priority values: the value of an item, the value of an item with A* heuristic

Reminder about Bayesian Inference

Bayesian inference:

$$p(\theta|x) = \frac{p(x|\theta)\,p(\theta)}{p(x)}$$

Bayesian inference with latent variables:

x, x — pure true sentence

$$p(z, \theta|x) = \frac{p(\theta)\,p(x, z|\theta)}{p(x)}$$

Sampling

Instead of finding the most likely structure, we randomly sample a structure from the underlying distribution

If the distribution is peaked, then it will be roughly the same as finding the highest scoring structure

One can also use “annealing” with sampling to find the highest scoring structure

Sampling Algorithms

Goal: sample from a target distribution $p(u)$

For example, $p(u)$ could be the posterior from a Bayesian model, in which case $u = (z, \theta)$

Most common sampling algorithms: Markov Chain Monte Carlo methods

Ideal for cases in which we cannot compute the “normalisation” constant such as with Bayesian models

Markov Chain Monte Carlo

The big picture:

- Our sample space becomes a space of “states”
- There is some strategy to probabilistically move between states
- The strategy ensures that the transition between states will at some point converge to the target distribution we are interested in
- The samples do *not* have to be independent, and usually are not!
- Very useful for cases in which we can calculate the target distribution up to its normalisation constant (such as with Bayesian inference)

Example of MCMC: Gibbs Sampling

Break a "state" into several parts, for example, two parts: z and θ

Algorithm:

Let θ^* and z_0 be some random values.
Repeat until convergence

- Sample z^* from $p(z \mid \theta^*)$
- Sample θ^* from $p(\theta \mid z_0)$
- Set z_0 to be z^*

Collect the samples θ^* and z^*

We can also break z into further parts and use "operators" to move between states, making small changes to a bigger structure.

Strong relationship to search algorithms

More General: Metropolis-Hastings Algorithm

We are interested in sampling from $p(u)$, but can only sample from a *proposal* distribution $q(u'|u)$

Algorithm:

Initialise u with some value from Ω

Repeat until convergence

$$p(u) = \frac{r(u)}{Z}$$

- Sample u' from $q(u'|u)$
- Calculate an acceptance ratio

$$\alpha = \min\left\{1, \frac{p(u')q(u|u')}{p(u)q(u'|u)}\right\}$$

- Set $u \leftarrow u'$ with probability α

Collect the samples u all through

Integer Linear Programming

Formulate the problem of inference as maximising a linear objective with constraints

The variables in the objective are “pieces” of the structure

Useful technique to add “global” constraints to the inference algorithm

Use off-the-shelf tools to find a solution, such as CPLEX or Gurobi

$$\max_{y} \; w \cdot \phi(x, y)$$

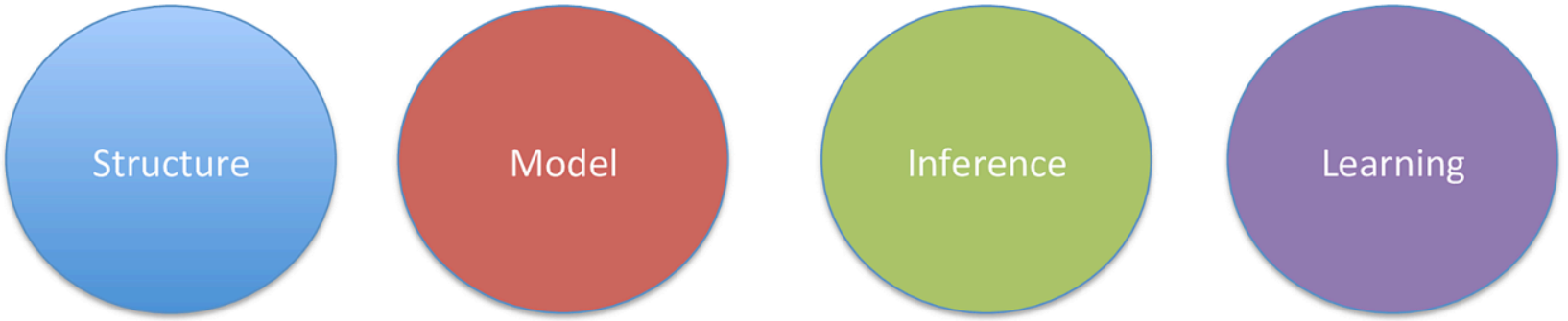
$$+ \text{ some constraints}$$

Summary

- Inference in our context refers to finding an output for a given input (decoding)
- ... or other type of information about the output
- Most common ways to do that in NLP: search algorithms, dynamic programming, sampling algorithms, integer linear programming

Solving an NLP Problem

When modelling a new problem in NLP, need to address four issues:



Log-Linear Models

To define a probability model over $\mathcal{X} \times \mathcal{Y}$, define a feature function $\phi(x, y)$ and then define a **log-linear model**:

weights feature function

$$p(y \mid x) = \frac{\exp\left(w \cdot \phi(x, y)\right)}{\sum_{y'} \exp\left(w \cdot \phi(x, y')\right)}$$

where w is a weight vector.

Also sometimes written as

$$p(y \mid x) = \frac{\exp\left(w \cdot \phi(x, y)\right)}{Z(x, w)}$$

- Given a set of data $(x_1, y_1), \ldots, (x_n, y_n)$ we need to find w. How?
- What is the role of ϕ?

Role of ϕ

- We see x and y through the glasses of ϕ
- ϕ should choose parts of x and y that indicate each other.

For example:
POS tagging:

$$\phi(x, y) = \begin{cases} 1 & \text{if } x \text{ starts with capital letter and } y \text{ is Proper Noun} \\ 0 & \text{o/w} \end{cases}$$

word, pos tag

Named entity recognition:

Training Log-Linear Models

Define the log-likelihood function:

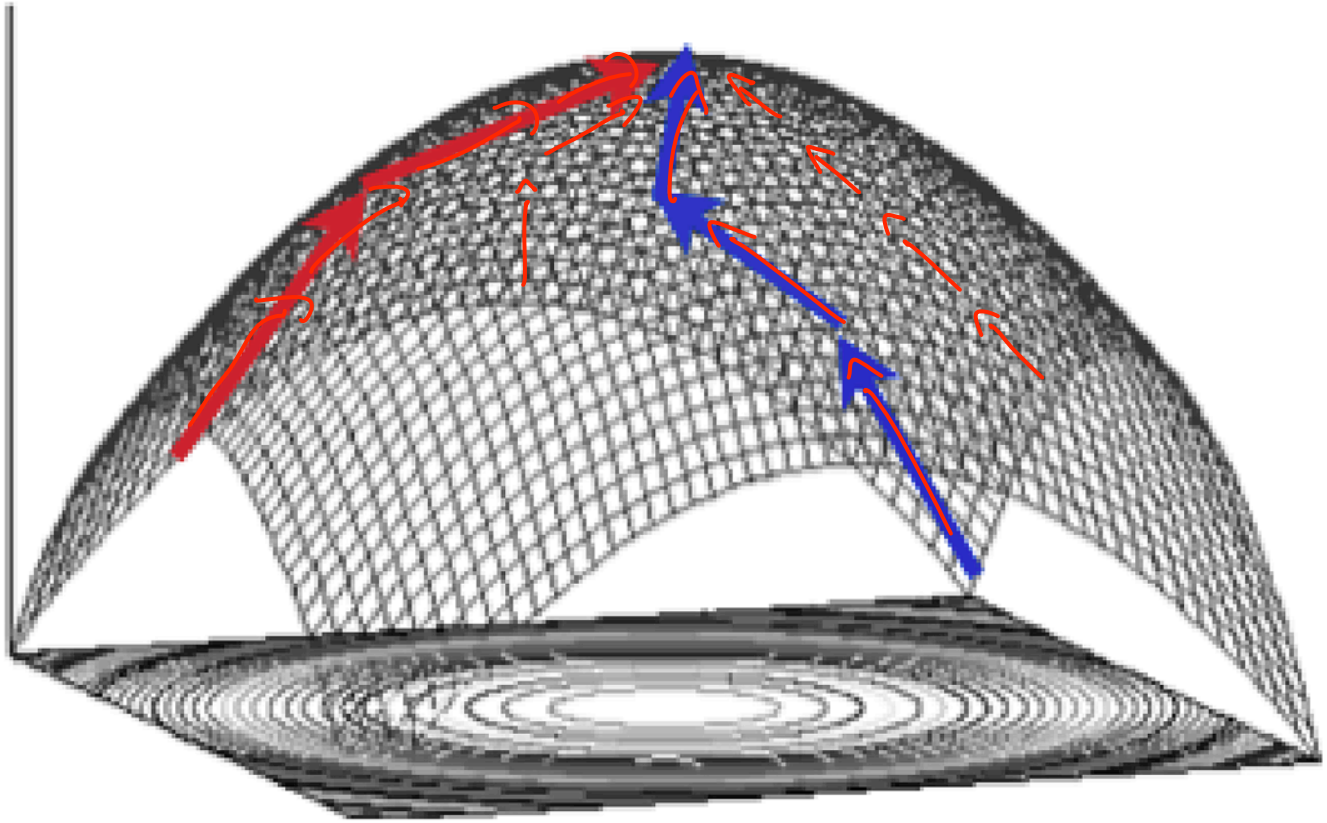
$$L(w \mid x_i, y_i, i \in [n]) = \sum_{i=1}^{n} \log p(y_i \mid x_i, w)$$

Gradient descent and other optimisation algorithms are our main tool

Need to be able to calculate the *gradient* of the log-likelihood.

Maximising the log-likelihood

$$w^* = \arg\max_w L(w|x_1, y_1, \ldots, x_n, y_n)$$



Gradient of Log-Linear Model

$$L(w \mid x_i, y_i, i \in [n]) = \frac{1}{n}\sum_{i=1}^{n} \log p(y_i \mid x_i, w) =$$

$$= \frac{1}{n} \sum_{i=1}^{n} \log \left[\frac{\exp(w \cdot \phi(x_i, y_i))}{Z(x_i, w)} \right]$$

$$= \frac{1}{n} \left[\sum_{i=1}^{n} w \cdot \phi(x_i, y_i) - \log Z(x_i, w) \right]$$

$$\left(\ell_i(w) = w \cdot \phi(x_i, y_i) - \log Z(w, x_i) \right)$$

$$= \frac{1}{n} \sum_{i=1}^{n} \ell_i(w)$$

Gradient of a Single Summand

$$\ell_i(w \mid x_i, y_i) = w \cdot \phi(x_i, y_i) - \log\left(\sum_{y'} \exp\left(w \cdot \phi(x, y')\right)\right)$$

$$\frac{\partial \ell_i}{\partial w_j} = \phi_j(x_i, y_i) - \frac{\partial \log z(x_i, w)}{\partial w_j}$$

$$\frac{\partial z}{\partial w_j}(w) = \sum_{y'} \exp\left(w \cdot \phi(x, y')\right) \cdot \phi_j(x, y')$$

$$\frac{\partial \log z}{\partial w_j}(w) = \frac{1}{z(x_i, w)} \cdot \frac{\partial z(x_i, w)}{\partial w_j}$$

Gradient of average log-likelihood

$$\frac{\partial L}{\partial w_j} = \left(\frac{1}{n}\sum_{i=1}^{n}\phi_j(x_i, y_i)\right) - \sum_i \sum_y \frac{\exp(\sum_{k=1}^{d} w_k \phi_k(x_i, y))}{Z(x_i, w)}\phi_j(x_i, y)$$

$$\frac{1}{n}\sum_{i=1}^{n}\phi_j(x_i, y_i) = E_{\tilde{p}}\left[\phi_j(x, y)\right]$$

$$\sum_i \sum_y \frac{\exp(\sum_{k=1}^{d} w_k \phi_k(x, y))}{Z(x_i, w)}\phi_j(x_i, y) = E_{p_w}\left[\phi_j(x, y)\right]$$

Gradient of average log-likelihood

$$\frac{\partial L}{\partial w_j} = \left(\frac{1}{n} \sum_{i=1}^{n} \phi_j(x_i, y_i) \right) - \sum_i \sum_y \frac{\exp(\sum_{k=1}^{d} w_k \phi_k(x_i, y))}{Z(x_i, w)} \phi_j(x_i, y)$$

$$\frac{1}{n} \sum_{i=1}^{n} \phi_j(x_i, y_i) =$$

$$\sum_i \sum_y \frac{\exp(\sum_{k=1}^{d} w_k \phi_k(x, y))}{Z(x_i, w)} \phi_j(x_i, y) =$$

Therefore, the gradient is the difference between empirical expectations and expectations under the model

MaxEnt Modelling and Log-linear Models

Principle of maximum entropy: Keep the uncertainty about what you didn't observe as high as possible

MaxEnt Modelling and Log-linear Models

Principle of maximum entropy: Keep the uncertainty about what you didn't observe as high as possible

Maximising the entropy of the model while keeping the feature expectations according to the model identical to the feature expectations according to the data

is equivalent to

Maximising the log-likelihood of a log-linear model with the same feature functions

Regularisation

To avoid overfitting, add a term that ensures that weights do not become too large in absolute value:

$$L(w \mid x_i, y_i, i \in [n]) = \sum_{i=1}^{n} \log p(y_i \mid x_i, w) - R(w)$$

Examples for $R(w)$:

- $R(w) = ||w||_2^2 = \sum_i w_i^2$
- $R(w) = ||w||_1 = \sum_i |w_i|$

Decoding with Log-Linear Models

Given an x, want to find the most likely y given an x:

$$y^* = \arg\max_y p(y \mid x, w) = \arg\max_y \frac{\exp(w \cdot \phi(x,y))}{Z(x,w)} =$$

$$= \arg\max_y \exp(w \cdot \phi(x,y))$$

$$= \arg\max_y w \cdot \phi(x,y)$$