Topics in Natural Language Processing

Shay Cohen

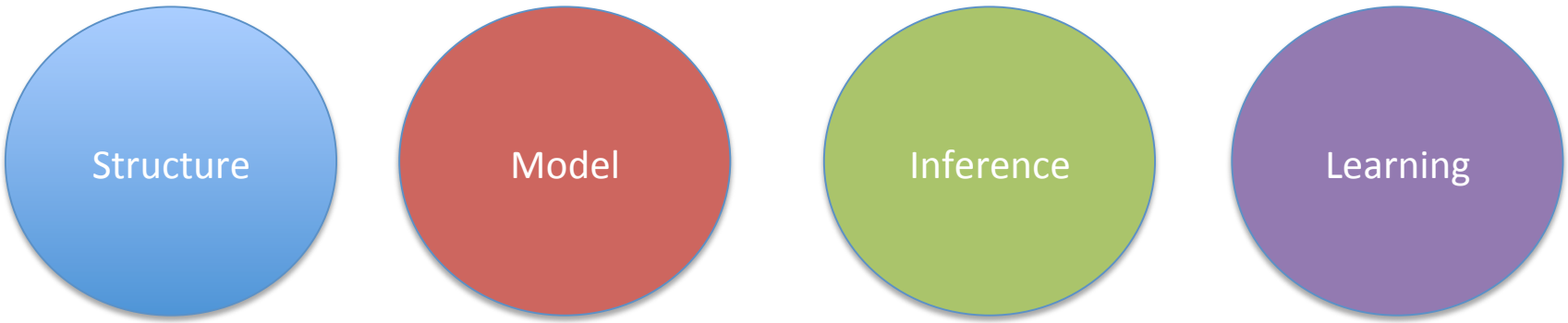
Institute for Language, Cognition and Computation

University of Edinburgh

Lecture 6

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**:

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

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:
(x - word, y - POS tag, type-based POS tagging)

$\phi(x, y)$ is 1 if x starts with a capital letter and y is a Proper Noun and 0 otherwise.
Named entity recognition:
(x - word, y - type of named entity)

$\phi(x, y)$ is 1 if x ends with “son” and y is a Person and 0 otherwise.

Training Log-Linear Models

Define the log-likelihood function:

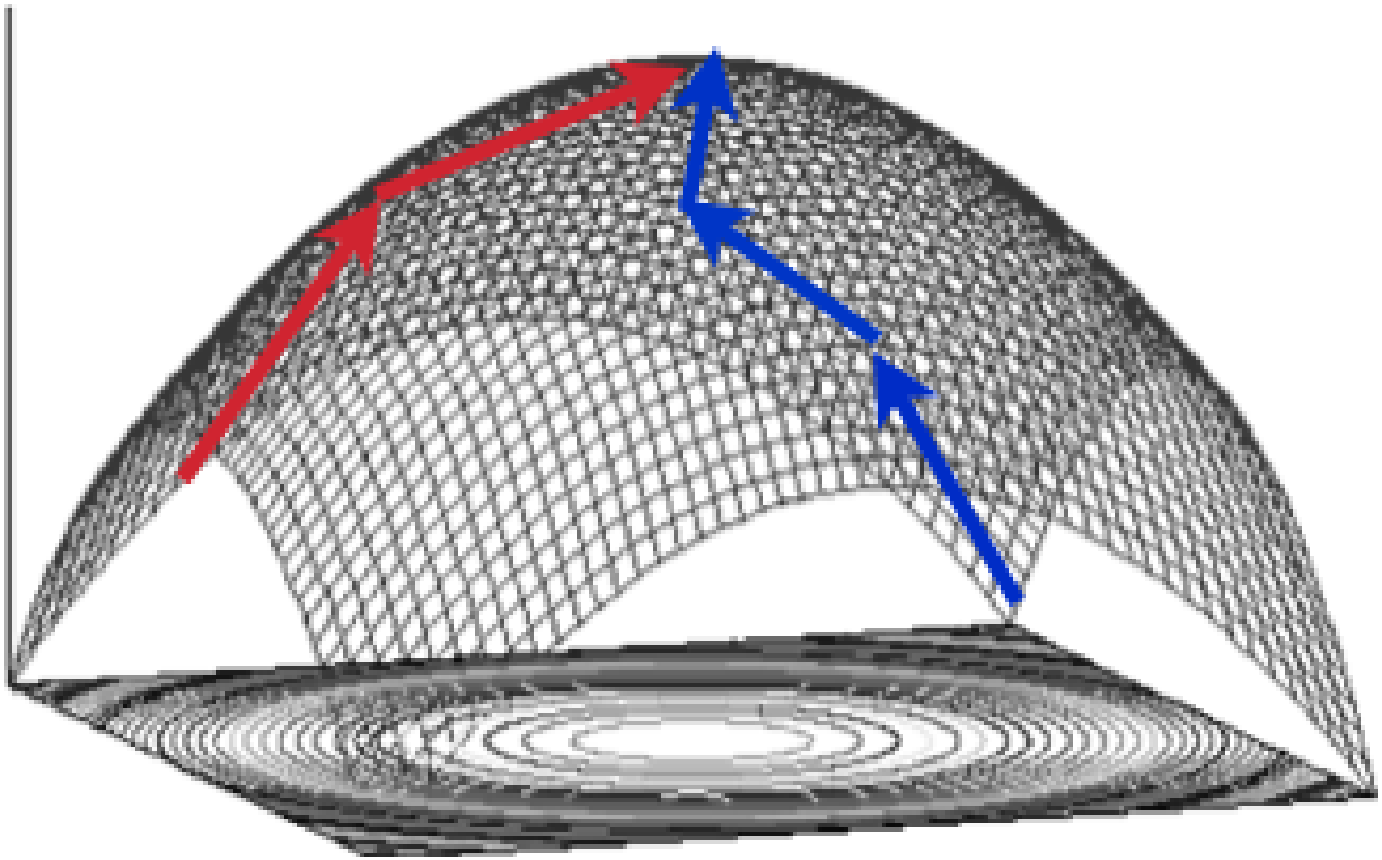
$$L(w \mid x_i, y_i, i \in [n]) = \sum_{i=1}^{n} \log p(y_i \mid x_i, w)$$

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

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

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

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

call each term $\ell_i(w)$

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

Gradient of a Single Summand

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

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

$$\frac{\partial Z(x_i, w)}{\partial w_j}(w) = \sum_{y'} \exp(w \cdot \phi(x_i, y'))\phi_j(x_i, y')$$

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

Gradient of average log-likelihood

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

$$\frac{1}{n} \sum_{i=1}^{n} \phi_j(x_i, y_i) = \mathbb{E}_{\tilde{p}}[\phi_j(x, y)]$$

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

Gradient of average log-likelihood

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

$$\frac{1}{n} \sum_{i=1}^{n} \phi_j(x_i, y_i) = \mathbb{E}_{\tilde{p}}[\phi_j(x, y)]$$

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

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

What are we trying to do with gradient descent? Essentially, find a place where the gradient is zero. **What does that mean?**

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:

$$y^* = \arg\max_y p(y \mid x, w) =$$

$$= \arg\max y \frac{\exp(w \cdot \phi(x, y))}{Z(x, w)}$$

($Z(x, w)$ is constant with respect to y and therefore:)

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

(and now we can just take the $\log$)

$$= \arg\max yw \cdot \phi(x, y)$$

Linear Score Models

A non-probabilistic model that only considers the score

$$\text{score}(x, y, w) = w \cdot \phi(x, y)$$

Decoding remains the same:

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

Can we train the model directly with this score?

Linear Score Models

A non-probabilistic model that only considers the score

$$\text{score}(x, y, w) = w \cdot \phi(x, y)$$

Decoding remains the same:

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

Can we train the model directly with this score?
Yes! For example, with the Perceptron.

Training Linear Score Models

The Perceptron algorithm:

- Initialise w to 0.
- For T iterations
  - For each labelled pair (x, y_{gold}) in the data
    - $y_{\text{predict}} = \arg\max_y w \cdot \phi(x, y)$
    - $w \leftarrow w + \phi(x, y_{\text{gold}}) - \phi(x, y_{\text{predict}})$

Intuition Behind Perceptron

Main update rule in step t:

$$w \leftarrow w + \phi(x_t, y_{\text{gold}}) - \phi(x, y_{\text{predict}})$$

- Features that fire in the correct structure positively predict the structure, and if they don't fire in the prediction, we need to increase their weight to make them more "important" (increase the score when they fire)
- If we don't make a mistake, there won't be an update!

Avoiding Overfitting with the Perceptron

The *averaged* Perceptron: maintain a w_{average} which is the average of all weight vectors w after each update (whether it happened or not).

Return w_{average} as the final weight vector

Each w is most adapted to the last example it has seen. Averaging treats each w as a separate classifier, and then takes the average of all scores from all of these classifiers

Two Interpretations of the Perceptron

- The Mistake Bound Model
- Optimising an objective function

Stochastic Gradient Descent

The Perceptron algorithm can be viewed as a *stochastic subgradient descent* algorithm.

- Stochastic: instead of making an update to the full objective function, summing over all examples, we make an update for an example at a time

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

  Update at each step with the gradient $\frac{\partial \ell_i}{\partial w}$.

- How is the Perceptron related to SSGD?

Perceptron as Objective Maximisation

We usually think in terms of optimising an objective function (like with log-linear models). Does the Perceptron optimise any function for a training set $(x_1, y_1), \ldots, (x_n, y_n)$?
Consider

$$P(w, x_i, i \in [n]) = \arg\min_w \lambda ||w||_2^2 + \frac{1}{n} \sum_i \max_{y'} \{0, \left(\phi(x_i, y') - \phi(x_i, y_i)\right) \cdot w\}$$

- Maximising $\left(\phi(x_i, y') - \phi(x_i, y_{\text{gold}})\right) \cdot w$ gives the y' that is closest to y_{gold} in its score.
- Minimising the whole objective function tries to minimise this score difference
- The $||w||_2^2$ term is for regularisation

Subgradient

To optimise this function P we can calculate its "subgradient"

$$P(w, x_i, i \in [n]) = \arg\min_w \lambda ||w||_2^2 + \frac{1}{n} \sum_i \max_{y'} \{0, \left(\phi(x_i, y') - \phi(x_i, y_i)\right) \cdot w\}$$

This is a generalisation of the notion of gradient (because the $\max$ function is not differentiable according to standard gradient calculations), and it gives the update of the Perceptron with $\lambda = 2$:
$w \leftarrow w + \phi(x, y_{\text{gold}}) - \phi(x, y_{\text{predict}})$

Note that we are doing "stochastic optimisation" – at each step updating the weights with a single example

Support Vector Machines

Similar to the Perceptron, only with a slightly different objective function:

$$\arg\min_{w} \lambda||w||_2^2 + \frac{1}{n}\sum_i \max_{y'}\{0, \Delta(y_i, y') + \big(\phi(x_i, y') - \phi(x_i, y_i)\big) \cdot w\}$$

$\Delta(y_i, y')$ is a loss function that tells how far y_i is from y (for example, accuracy of labels in a sequence)

Idea: We take into account not just the linear score, but also how well y' is according to some evaluation metric

Same update as the Perceptron, only we need to find y_{predict} such that

$$y_{\text{predict}} = \arg\max_{y'} \Delta(y_i, y') + \phi(x_i, y_i) - \phi(x_i, y')$$

Summary

- Different types of algorithms for linear model learning
- We learned about: log-linear models, linear models with the Perceptron and linear models with Support Vector Machines
- There is an active area of research that adds nuances to these ideas for better optimisation and other learning objectives