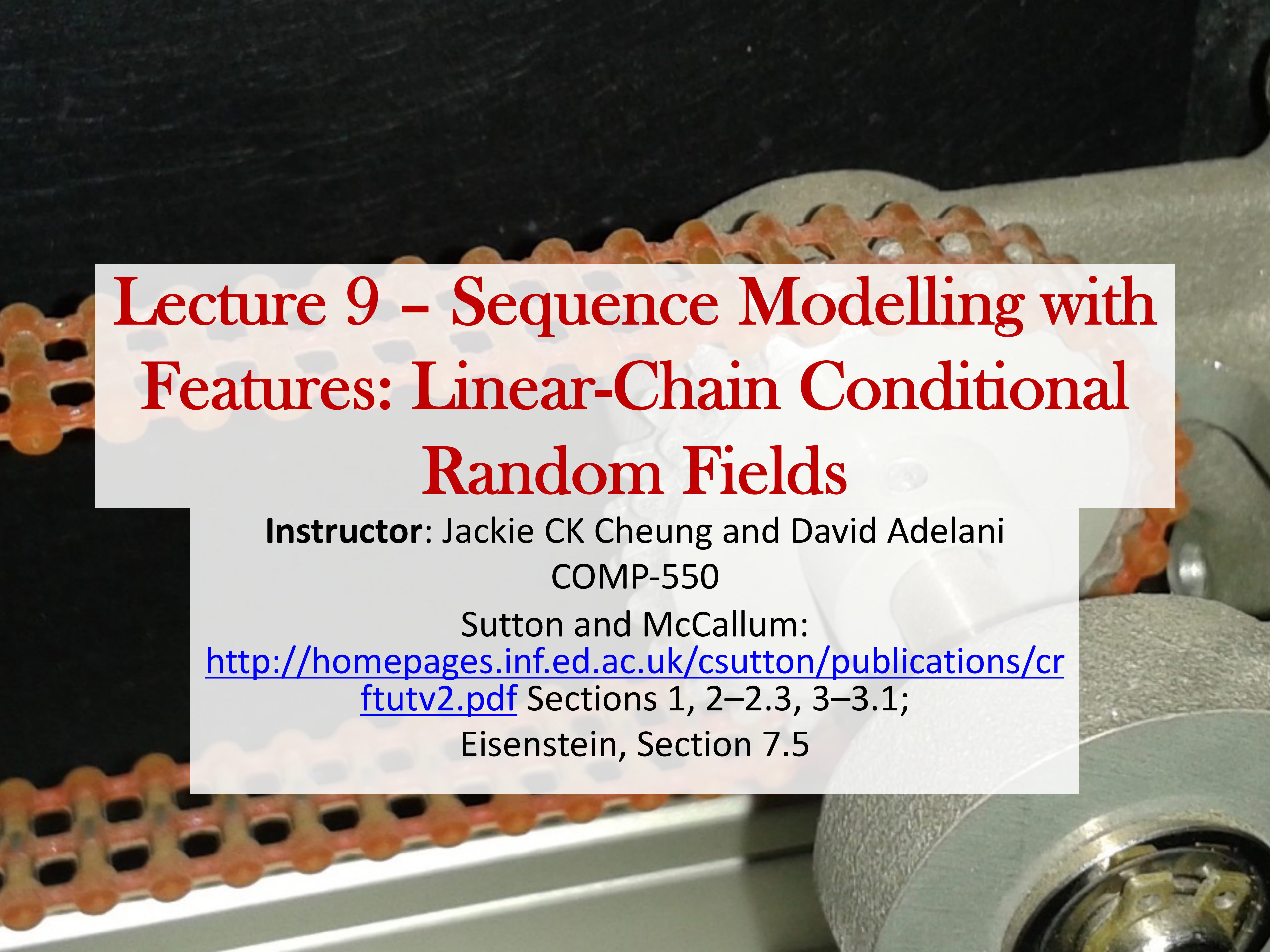


Lecture 9 – Sequence Modelling with Features: Linear-Chain Conditional Random Fields

Instructor: Jackie CK Cheung and David Adelani

COMP-550

Sutton and McCallum:
http://homepages.inf.ed.ac.uk/csutton/publications/crftutv2.pdf Sections 1, 2–2.3, 3–3.1;
Eisenstein, Section 7.5

Outline

Unsupervised learning of HMMs

Hidden Markov models: shortcomings

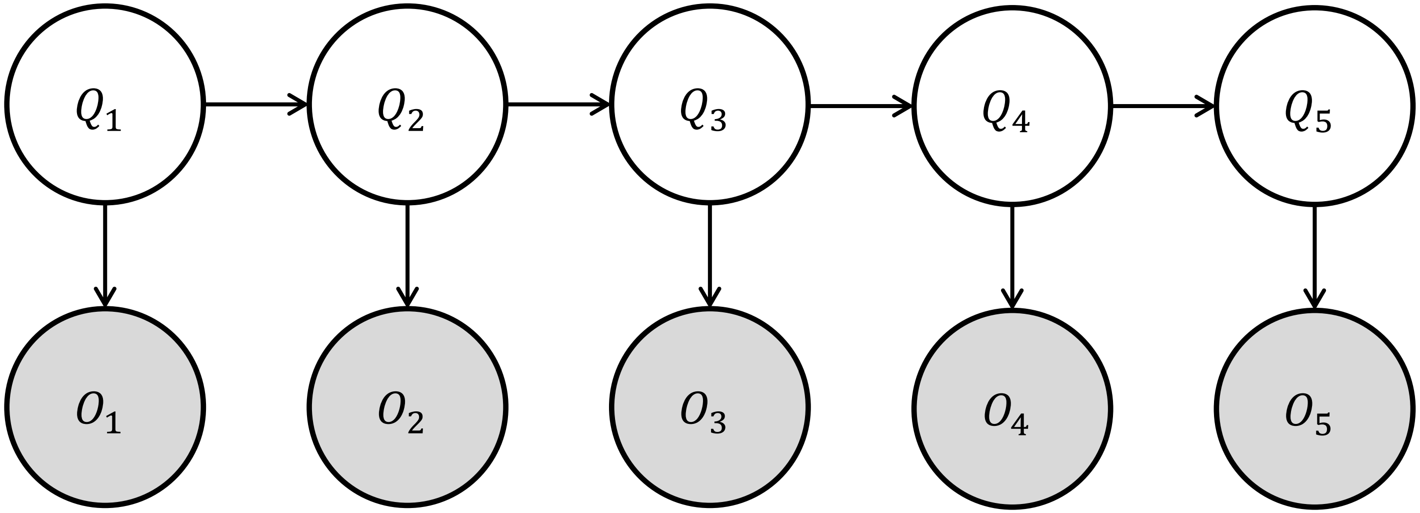
Generative vs. discriminative models

Linear-chain CRFs

- Inference and learning algorithms with linear-chain CRFs

Hidden Markov Model

Graph specifies how join probability decomposes



$$P(\boldsymbol{O},\boldsymbol{Q}) = P(Q_1)\prod_{t=1}^{T-1} P(Q_{t+1}|Q_t)\prod_{t=1}^{T} P(O_t|Q_t)$$

Initial state probability

State transition probabilities

Emission probabilities

Questions for an HMM

1. Compute likelihood of a sequence of observations, $P(\boldsymbol{O}|\theta)$

   Forward algorithm, backward algorithm

2. What state sequence best explains a sequence of observations?

   $\operatorname{argmax}_{\boldsymbol{Q}} P(\boldsymbol{Q}, \boldsymbol{O}|\theta)$

   Viterbi algorithm

3. Given an observation sequence (without labels), what is the best model for it?

   Forward-backward algorithm
   Baum-Welch algorithm
   Expectation Maximization

Q3: Unsupervised Training

Problem: No state sequences to compute above

Solution: Guess the state sequences

Initialize parameters randomly

Repeat for a while:

1. Predict the current state sequences using the current model
2. Update the current parameters based on current predictions

"Hard EM" or Viterbi EM

Initialize parameters randomly

Repeat for a while:

1. Predict the current state sequences using the current model with the Viterbi algorithm
2. Update the current parameters using the current predictions as in the supervised learning case

Can also use "soft" predictions; i.e., the probabilities of all the possible state sequences

Baum-Welch Algorithm

a.k.a., **Forward-backward** algorithm

EM algorithm applied to HMMs:

Expectation Get *expected* counts for hidden structures using current θ^k.

Maximization Find θ^{k+1} to maximize the likelihood of the training data given the expected counts from the E-step.

Responsibilities Needed

Supervised/Hard EM:	A single tag
Baum-Welch:	*Distribution* over tags

Call such probabilities **responsibilities**.

$\gamma_i(t) = P(Q_t = i | \boldsymbol{O}, \theta^k)$

- Probability of being in state i at time t given the observed sequence under the current model.

$\xi_{ij}(t) = P(Q_t = i, Q_{t+1} = j | \boldsymbol{O}, \theta^k)$

- Probability of transitioning from i at time t to j at time $t + 1$.

E-Step γ

$$\gamma_i(t) = P(Q_t = i|\boldsymbol{O}, \theta^k)$$

$$= \frac{P(Q_t = i, \boldsymbol{O}|\theta^k)}{P(\boldsymbol{O}|\theta^k)}$$

$$= \frac{\alpha_i(t)\beta_i(t)}{P(\boldsymbol{O}|\theta^k)}$$

E-Step ξ

$$\xi_{ij}(t) = P(Q_t = i, Q_{t+1} = j | \boldsymbol{O}, \theta^k)$$

$$= \frac{P(Q_t = i, Q_{t+1} = j, \boldsymbol{O} | \theta^k)}{P(\boldsymbol{O}|\theta^k)}$$

$$= \frac{\alpha_i(t) a_{ij} b_j(O_{t+1}) \beta_j(t+1)}{P(\boldsymbol{O}|\theta^k)}$$

Beginning to state i at time t

Transition from i to j

Emit O_{t+1}

Rest of the sequence

M-Step

"Soft" versions of the MLE updates:

$$\pi_i^{k+1} = \gamma_i(1)$$

$$a_{ij}^{k+1} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

$$b_i^{k+1}(w_k) = \frac{\sum_{t=1}^{T} \gamma_i(t)|_{O_t=w_k}}{\sum_{t=1}^{T} \gamma_i(t)}$$

Compare:

$$\frac{\#(i,j)}{\#(i)}$$

$$\frac{\#(\text{word } k, \text{tag } i)}{\#(i)}$$

- With multiple sentences, sum up expected counts over all sentences

When to Stop EM?

Multiple options

- When training set likelihood stops improving
- When prediction performance on held-out **development** or **validation** set stops improving

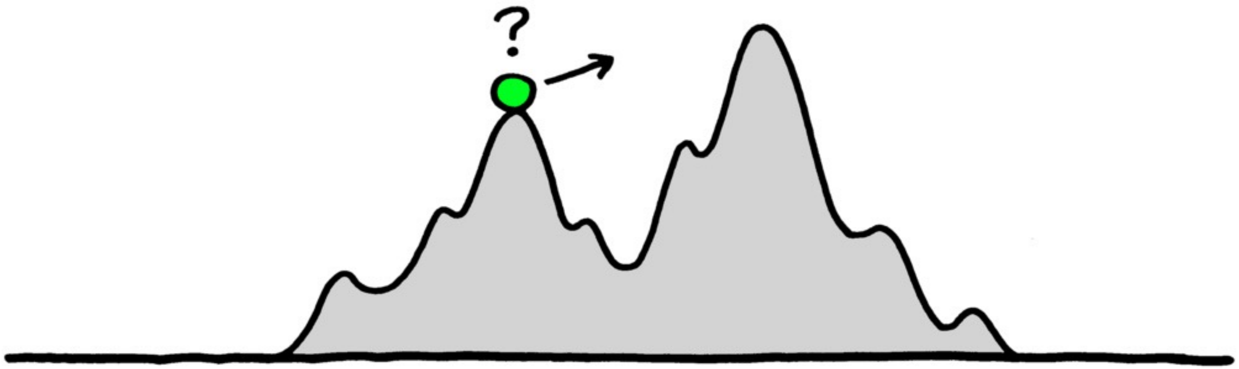
Why Does EM Work?

EM finds a local optimum in $P(\boldsymbol{O}|\theta)$.

You can show that after each step of EM:

$$P\left(\boldsymbol{O}\middle|\theta^{k+1}\right) > P(\boldsymbol{O}|\theta^{k})$$

However, this is not necessarily a global optimum.



Proof of Baum-Welch Correctness

Part 1: Show that in our procedure, one iteration corresponds to this update:

$$\theta^{k+1} = \underset{\theta}{\operatorname{argmax}} \sum_{\boldsymbol{Q}} \log[P(\boldsymbol{O}, \boldsymbol{Q}|\theta)] P(\boldsymbol{Q}|\boldsymbol{O}, \theta^{k})$$

https://stephentu.github.io/writeups/hmm-baum-welch-derivation.pdf

Part 2: Show that improving the quantity

$\sum_{\boldsymbol{Q}} \log[P(\boldsymbol{O}, \boldsymbol{Q}|\theta)] P(\boldsymbol{Q}|\boldsymbol{O}, \theta^{k})$

corresponds to improving $P(\boldsymbol{O}|\theta)$

https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm#Proof_of_correctness

Dealing with Local Optima

Random restarts

- Train multiple models with different random initializations
- Model selection on development set to pick the best one

Biased initialization

- Bias your initialization using some external source of knowledge (e.g., external corpus counts or clustering procedure, expert knowledge about domain)
- Further training will hopefully improve results

Caveats

Baum-Welch with no labelled data generally gives poor results, at least for linguistic structure (~40% accuracy, according to Johnson, (2007))

Semi-supervised learning: combine small amounts of labelled data with larger corpus of unlabelled data

In Practice

Per-token (i.e., per word) accuracy results on WSJ corpus:

Most frequent tag baseline	~90—94%
HMMs (Brants, 2000)	96.5%
Stanford tagger (Manning, 2011)	97.32%
Current best	97.85%+

- E.g. (Akbik et al., 2018), based on algorithms we will discuss today and next class

Other Sequence Modelling Tasks

Chunking (a.k.a., **shallow parsing**)

- Find syntactic chunks in the sentence; not hierarchical

[$_{NP}$The chicken] [$_{V}$crossed] [$_{NP}$the road] [$_{P}$across] [$_{NP}$the lake].

Named-Entity Recognition (**NER**)

- Identify elements in text that correspond to some high level categories (e.g., PERSON, ORGANIZATION, LOCATION)

[$_{ORG}$McGill University] is located in [$_{LOC}$Montreal, Canada].

- Problem: need to detect spans of multiple words

First Attempt

Simply label words by their category

ORG ORG ORG - ORG - - - LOC

McGill, UQAM, UdeM, and Concordia are located in Montreal.

What is the problem with this scheme?

IOB Tagging

Label whether a word is inside, outside, or at the beginning of a span as well.

For n categories, 2n+1 total tags.

B_{ORG} I_{ORG} O O O B_{LOC} I_{LOC}

McGill University is located in Montreal, Canada

B_{ORG} B_{ORG} B_{ORG} O B_{ORG} O O O B_{LOC}

McGill, UQAM, UdeM, and Concordia are located in Montreal.

Shortcomings of Standard HMMs

How do we add more features to HMMs?

Might be useful for POS tagging:

- Word position within sentence (1st, 2nd, last…)
- Capitalization
- Word prefix and suffixes (*-tion*, *-ed*, *-ly*, *-er*, *re-*, *de-*)
- Features that depend on more than the current word or the previous words.

Discriminative Models

HMMs are **generative models**

- Build a model of the joint probability distribution $P(\boldsymbol{O}, \boldsymbol{Q})$,
- Let's rename the variables
- Generative models specify $P(X, Y; \theta^{\text{gen}})$

If we are only interested in POS tagging, we can instead train a **discriminative model**

- Model specifies $P(Y|X; \theta^{\text{disc}})$
- Now a task-specific model for sequence labelling; cannot use it for generating new samples of word and POS sequences

Discriminative Sequence Model

The parallel to an HMM in the discriminative case: **linear-chain conditional random fields (linear-chain CRFs)** (Lafferty et al., 2001)

$$P(Y|X) = \frac{1}{Z(X)} \exp \sum_{t} \sum_{k} \theta_k f_k(y_t, y_{t-1}, x_t)$$

sum over all features

sum over all time-steps

Z(X) is a normalization constant:

$$Z(X) = \sum_{y} \exp \sum_{t} \sum_{k} \theta_k f_k(y_t, y_{t-1}, x_t)$$

sum over all possible sequences of hidden states

Intuition

Standard HMM: product of probabilities; these probabilities are defined over the identity of the states and words

- Transition from state DT to NN: $P(y_{t+1} = NN|y_t = DT)$
- Emit word the from state DT: $P(x_t = the|y_t = DT)$

Linear-chain CRF: replace the products by numbers that are NOT probabilities, but linear combinations of weights and feature values.

Features in CRFs

Standard HMM probabilities as CRF features:

- Transition from state DT to state NN

  $f_{DT\rightarrow NN}(y_t, y_{t-1}, x_t) = \mathbf{1}(y_{t-1} = DT)\,\mathbf{1}(y_t = NN)$
- Emit *the* from state DT

  $f_{DT\rightarrow the}(y_t, y_{t-1}, x_t) = \mathbf{1}(y_t = DT)\,\mathbf{1}(x_t = the)$
- Initial state is DT

  $f_{DT}(y_1, x_1) = \mathbf{1}(y_1 = DT)$

Indicator function:

Let $\mathbf{1}(condition) = \begin{cases} 1 & \text{if } condition \text{ is true} \\ 0 & \text{otherwise} \end{cases}$

Features in CRFs

Additional features that may be useful

- Word is capitalized

  $f_{cap}(y_t, y_{t-1}, x_t) = \mathbf{1}(y_t = ?)\mathbf{1}(x_t \text{ is capitalized})$

- Word ends in *–ed*

  $f_{-ed}(y_t, y_{t-1}, x_t) = \mathbf{1}(y_t = ?)\mathbf{1}(x_t \text{ ends with } ed)$

- Let's brainstorm and propose more features
  - $f_{len<5}(y_t, y_{t-1}, x_t) = \mathbf{1}(y_t = ?)\mathbf{1}(len(x_t) < 5)$
  - $f_{1st}(y_t, y_{t-1}, x_t) = \mathbf{1}(y_t = ?)\mathbf{1}(t = 1)$

Inference with LC-CRFs

Dynamic programming still works – modify the forward and the Viterbi algorithms to work with the weight-feature products.

	HMM	LC-CRF
Forward algorithm	$P(X\|\theta)$	$Z(X)$
Viterbi algorithm	$\underset{Y}{\operatorname{argmax}}\, P(X, Y\|\theta)$	$\underset{Y}{\operatorname{argmax}}\, P(Y\|X, \theta)$

Forward Algorithm for HMMs

```
Create trellis $\alpha_i(t)$ for $i = 1 \ldots N, t = 1 \ldots T$
$\alpha_j(1) = \pi_j b_j(O_1)$ for j = 1 … N
for t = 2 … T:
    for j = 1 … N:
        $\alpha_j(t) = \sum_{i=1}^{N} \alpha_i(t-1) a_{ij} b_j(O_t)$
$P(\boldsymbol{O}|\theta) = \sum_{j=1}^{N} \alpha_j(T)$
Runtime: O($N^2 T$)
```

Forward Algorithm for LC-CRFs

```
Create trellis $\alpha_i(t)$ for $i = 1 \ldots N, t = 1 \ldots T$
$\alpha_j(1) = \exp \sum_k \theta_k^{init} f_k^{init}(y_1 = j, x_1)$ for j = 1 … N
for t = 2 … T:
    for j = 1 … N:
        $\alpha_j(t) = \sum_{i=1}^{N} \alpha_i(t-1) \exp \sum_{k} \theta_k f_k(y_t = j, y_{t-1}, x_t)$
$Z(X) = \sum_{j=1}^{N} \alpha_j(T)$
```

Transition and emission probabilities replaced by exponent of weighted sums of features.

Runtime: O(N^2T)

Having $Z(X)$ allows us to compute $P(Y|X)$

Viterbi Algorithm for HMMs

```
Create trellis $\delta_i(t)$ for $i = 1 \ldots N, t = 1 \ldots T$
$\delta_j(1) = \pi_j b_j(O_1)$ for j = 1 … N
for t = 2 … T:
    for j = 1 … N:
        $\delta_j(t) = \max_i \delta_i(t-1) a_{ij} b_j(O_t)$
Take $\max_i \delta_i(T)$
```

Runtime: O(N^2T)

Viterbi Algorithm for LC-CRFs

```
Create trellis $\delta_i(t)$ for $i = 1 \ldots N, t = 1 \ldots T$
$\delta_j(1) = \exp \sum_k \theta_k^{init} f_k^{init}(y_1 = j, x_1)$ for j = 1 … N
for t = 2 … T:
    for j = 1 … N:
        $\delta_j(t) = \max_i \delta_i(t-1) \exp \sum_k \theta_k f_k(y_t = j, y_{t-1}, x_t)$
Take $\max_i \delta_i(T)$
```

Runtime: O(N^2T)

Remember that we need backpointers.

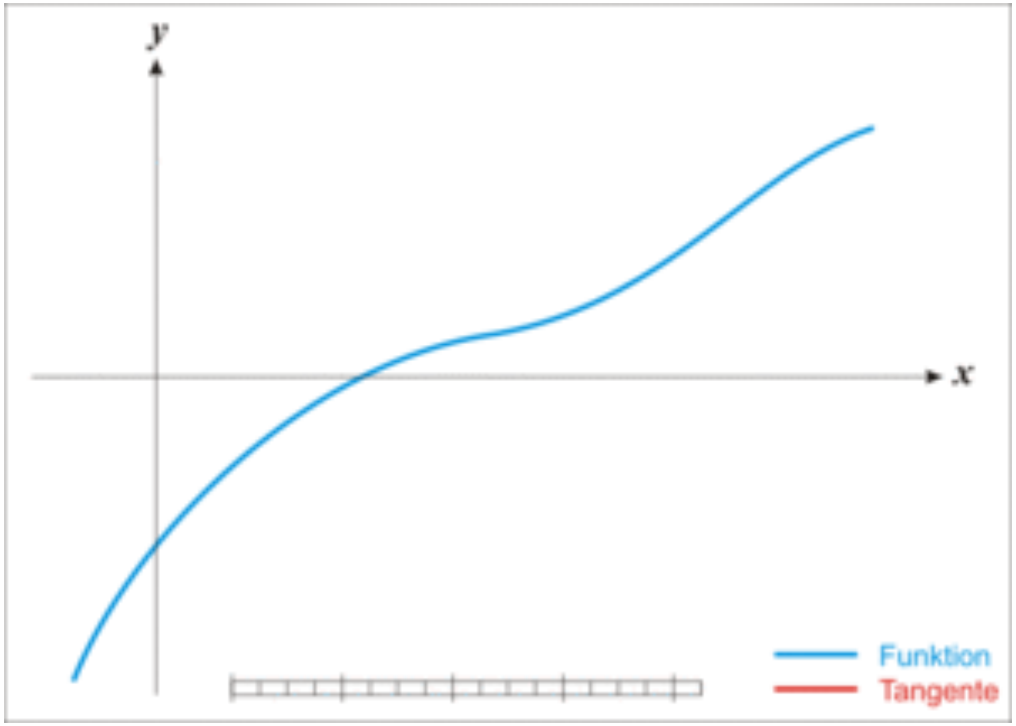
Training LC-CRFs

Unlike for HMMs, no analytic MLE solution

Use iterative method to improve data likelihood

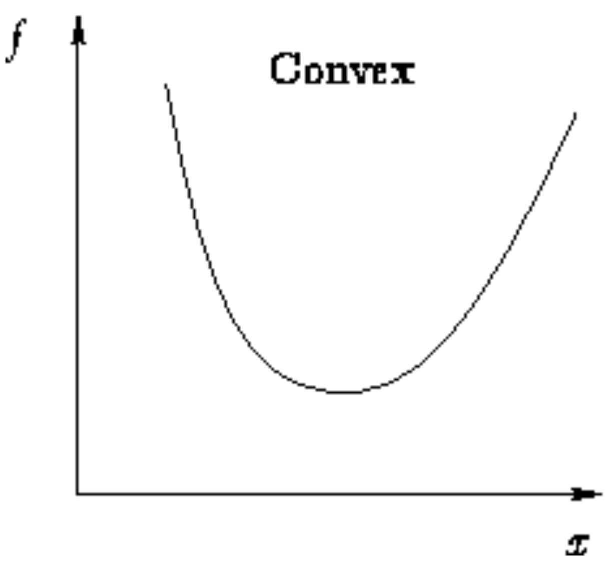
Gradient descent

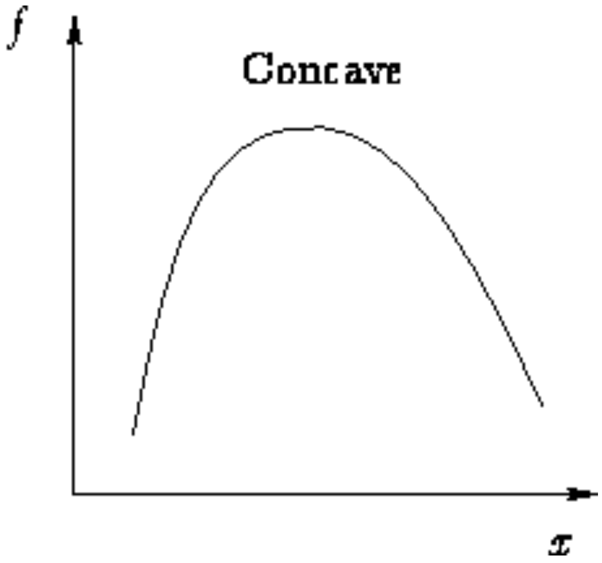
A version of Newton's method to find where the gradient is 0



Convexity

Fortunately, $l(\theta)$ is a **concave** function (equivalently, its negation is a **convex** function). That means that we will find the global maximum of $l(\theta)$ with gradient ascent (equivalently, the global minimum of $-l(\theta)$ with gradient descent).





Gradient Ascent

Walk in the direction of the gradient to maximize $l(\theta)$

- a.k.a., gradient descent on a loss function

$$\theta^{\text{new}} = \theta^{\text{old}} + \gamma \nabla l(\theta)$$

$\gamma$is a learning rate that specifies how large a step to take.

There are more sophisticated ways to do this update:

- Conjugate gradient
- L-BFGS (approximates using second derivative)

Gradient Descent Summary

Descent vs ascent

Convention: think about the problem as a minimization problem

Minimize the *negative* log likelihood

- $\theta \leftarrow \theta - \gamma(-\nabla l(\theta))$

Initialize $\theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$ randomly

Do for a while:

Compute $\nabla l(\theta)$, which will require dynamic programming (i.e., forward algorithm)

$\theta \leftarrow \theta + \gamma \nabla l(\theta)$

Gradient of Log-Likelihood

Find the gradient of the log likelihood of the training corpus:

$$l(\theta) = \log \prod_i P(Y^{(i)} | X^{(i)})$$

Here it is:

$$\sum_i \sum_t f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)} \right) - \sum_i \sum_t \sum_{y,y\prime} f_k \left(y, y', x_t^{(i)} \right) P(y, y' | X^{(i)})$$

Derivation on the next slide.

Gradient of Log-Likelihood

Find the gradient of the log likelihood of the training corpus:

$$l(\theta) = \log \prod_i P(Y^{(i)}|X^{(i)})$$

$$= \sum_i \log P\left(Y^{(i)}|X^{(i)}\right)$$

$$= \sum_i \log \left[exp \sum_t \sum_k \theta_k f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}\right) / \mathrm{Z}(X) \right]$$

$$= \sum_i \sum_t \sum_k \theta_k f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}\right) - \log \mathrm{Z}(X)$$

$$\frac{\partial L}{\theta_k} = \sum_i \sum_t f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}\right) - \frac{\partial L}{\theta_k} \log \mathrm{Z}(X)$$

$$\frac{\partial L}{\theta_k} \log \mathrm{Z}(X) = \frac{1}{Z(X)} \frac{\partial}{\theta_k} \sum_{\hat{y}} exp \sum_t \sum_k \theta_k f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}\right)$$

$$= \frac{1}{Z(X)} \sum_{\hat{y}} \left[exp \sum_t \sum_k \theta_k f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}\right) \right] \left[\sum_t f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}\right) \right]$$

$$= \sum_{\hat{y}} P(\hat{y}|X^{(i)}) \sum_t f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}\right)$$

Can re-arrange in terms of local state transition with f_k

$$\sum_i \sum_t \sum_{y,y\prime} f_k \left(y, y', x_t^{(i)}\right) P(y, y'|X^{(i)})$$

Here it is:

$$\frac{\partial L}{\theta_k} = \sum_i \sum_t f_k \left(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}\right) - \sum_i \sum_t \sum_{y,y\prime} f_k \left(y, y', x_t^{(i)}\right) P(y, y'|X^{(i)})$$

Interpretation of Gradient

Overall gradient is the difference between:

$$\sum_i \sum_t f_k(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)})$$

the empirical distribution of feature f_k in the training corpus

and:

$$\sum_i \sum_t \sum_{y,y\prime} f_k\left(y, y', x_t^{(i)}\right) P(y, y' | X^{(i)})$$

the expected distribution of f_k as predicted by the current model

Interpretation of Gradient

When the corpus likelihood is maximized, the gradient is zero, so the difference is zero.

Intuitively, this means that finding parameter estimate by gradient descent is equivalent to telling our model to predict the features in such a way that they are found in the same distribution as in the gold standard.

Regularization

To avoid overfitting, we can encourage the weights to be close to zero.

Add term to corpus log likelihood:

$$l^*(\theta) = \log \prod_i P(Y^{(i)}|X^{(i)}) \; - \sum_k \frac{\theta_k^2}{2\sigma^2}$$

σ controls the amount of **regularization**

Results in extra term in gradient:

$$-\frac{\theta_k}{\sigma^2}$$

Stochastic Gradient Descent (SGD)

In the standard version of the algorithm, the gradient is computed over the entire training corpus.

- Weight update only once per iteration through training corpus.

Alternative: calculate gradient over a small mini-batch of the training corpus and update weights

SGD is when mini-batch size is one.

- Many weight updates per iteration through training corpus
- Usually results in much faster convergence to final solution, without loss in performance

Stochastic Gradient Descent

```
Goal: Minimize $-l^*(\theta)$
Initialize $\theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$ randomly
Do for a while:
    Randomize order of samples in training corpus
    For each mini-batch (of size one) in the training corpus:
        Compute $\nabla l^*(\theta)$ over this mini-batch
        $\theta \leftarrow \theta + \gamma \nabla l^*(\theta)$
```