Sums of logs

Issue: computing $\alpha$ forward probabilities can undeflow

- Normally we’d fix this using logs
- But $\alpha$ requires a sum of probabilities
  - Not easy to do in log-space

Solution 1: scale/exponentiate

```python
def logplus(xx, yy):
    M = max(xx, yy)
    return M + log(exp(xx - M) + exp(yy - M))
```

Because:

- Subtracting $M$ in logspace is equivalent to dividing
- So we have $\frac{x}{M} + \frac{y}{M}$ (one of these is 1)
  - Doesn’t underflow
- Computes $\frac{1}{M}(x + y)$
- Then take the log again, and multiply by $M$ to get $\log(x + y)$
Solution 2

Normalize the $\alpha$ values at each timestep.

- At a single timestep, we’d get:

$$A(i) = \sum_t \alpha_t(i) = P(W_{1:i})$$

- Divide all the $\alpha_t(i)$ by $A(i)$

$$\alpha_t'(i) = \frac{\alpha_t(i)}{A(i)} = P(T_i = t | W_{1:i})$$

Now the $\alpha'$ are not so small (since they have to sum to 1). Applying this recursively, we get that:

$$A'(i) = \sum_t \alpha_t'(i) = P(W_i)$$

We can extract the probability of all the words:

$$\log P(W_{1:n}) = \log \prod_i A'(i) = \sum_i \log A'(i)$$
Conditional random fields

- A model which handles sequences
  - Like the HMM
- And can use many correlated features
  - Like the max-entropy classifier
- By combining the two ideas

Text here: Sutton and McCallum “An introduction to conditional random fields”

- Has everything you want to know
- Relatively accessible
- Really, really long
  - Basics: sections 2.2-2.4
  - Applications: 2.5-2.7
  - Algorithms: 4.1, 4.3, 5.1
Review: generative vs conditional

Generative model

- Model of the observed data (features $F$, tag $T$)
- Params maximize $P(F, T; \theta)$
  - The joint probability of all observations

Conditional model

- Model of the tag as a function of the data
- Params maximize $P(T|F; \theta)$
  - The probability of the tag
Deriving the conditional model

- Move away from having a *probability* over feature values
- Instead have *weights* for feature values
- Want to optimize using gradient (calculus)
  - Because weights are correlated so estimation isn’t obvious
- Easier to take derivatives if everything is a sum
- So: use exp log to transform products into sums
- End up with dot product
Making an HMM into a dot product

\[
P(T_{1:n}|W_{1:n}) = \frac{\prod_{i=0}^{n} P_{\text{trans}}(T_{i+1}|T_i)P_{\text{emit}}(W_i|T_i)}{P(W_{1:n})}
\]

\[
= \exp \log \prod_{i=0}^{n} P_{\text{trans}}(T_{i+1}|T_i)P_{\text{emit}}(W_i|T_i)
\]

\[
= \frac{\exp (\sum_{i=0}^{n} \log P_{\text{trans}}(T_{i+1}|T_i) \log P_{\text{emit}}(W_i|T_i))}{P(W_{1:n})}
\]

- Let \( f_{t\rightarrow t'}(T) = (T_{i+1} = t', T_i = t) \) and \( \theta_{t\rightarrow t'} = \log P(t'|t) \)
- Let \( f_{t\rightarrow w}(T) = (T_i = t, W_i = w) \) and \( \theta_{t\rightarrow w} = \log P(w|t) \)
- Defines a feature vector \( F(T) \) (specific to tags \( T \))

\[
= \frac{\exp(\Theta \cdot F(T))}{P(W_{1:n})}
\]
Making it conditional

\[
P(T|W) = \frac{\exp(\Theta \cdot F(T))}{P(W_{1:n})} = \frac{\exp(\Theta \cdot F(T))}{\sum_{T_{1:n}} \exp(\Theta \cdot F(T'))}
\]

To make this conditional:

- Drop constraint that \( \theta \) are log probabilities
- Choose \( \theta \) to maximize probability of training \( P(T|W) \)
- Normalizer is no longer interpretable as \( P(W_{1:n}) \)

\[
Z(\Theta) \text{ or } Z = \sum_{T'_{1:n}} \exp(\Theta \cdot F(T'))
\]

- The “normalizing constant” or “partition function”
- Notation \( Z \) and partition name from statistical physics
(Linear-chain) CRF

\[ P(T|W) = \frac{1}{Z} \exp(\Theta \cdot F(T)) \]

Graphical model:

Undirected graphical model

- No arrows
- Arcs in the graph represent weights
- No interpretation of arc as conditional prob...
- Just a number that gets multiplied!
Adding features

- We can add features of the word just like in max-entropy
- Capitalization, numeric, suffix, etc
Inference

What we want:

$$\max P(T_{1:n} | W_{1:n}) \propto \exp \Theta \cdot F(T)$$

We can simply maximize the weight $\Theta \cdot F(T)$

- This breaks down into a sum of the $f$ features and $\theta$ weights
- Can maximize with Viterbi
- Write initial weight of 0 under $<s>$
- On an arc, add transition and emission $\theta$

$$\mu_t(i) = \max_{t'} \theta_{t \rightarrow w} + \theta_{t' \rightarrow t} + \mu_{t'}(i - 1)$$
Estimation

Using gradient:

- As in max-entropy model, gradient is difference of:
  - Count of feature \( f \) in true labelings (from numerator)
  - Expected count of feature \( f \) (from denominator)

- Numerator term is easy (just count!)
- Denominator term is harder

\[
E_{P(\theta)}[f_{t' \rightarrow t}(T)] = \sum_{T', W \in D} P(T' | W; \Theta) \#(f_{t' \rightarrow t}(T))
\]

Sum over all possible tag sequences of probability times whether feature is active

- Can compute using forward-backward
Want to compute:

\[ \sum_{T', W \in D} P(T' | W; \Theta) \#(f_{t' \rightarrow t}(T)) \]

Basic question: probability that \( f_{t' \rightarrow t} \) is active at time \( i \)

Very similar to tag marginals!

For time \( i \):

\[
P(T_i = t', T_{i+1} = t) \propto \alpha_{t'}(i) \theta_{t' \rightarrow t} \theta_{t \rightarrow w_{i+1}} \beta_t(i + 1)
\]
Feature design

Consequence: we can compute gradient easily for:
- Features which look at a single $T_i \in T_{1:n}$
  - And any words $W$ (since they don’t depend on $T$)
- Features which look at $T_i, T_{i+1}$
  - And any words $W$

Hard(er) to compute for:
- Features which look at more than two tags
- Features for non-adjacent tags
- These require altering the dynamic program somehow

Features which only look at $W$ and not $T$ don’t do anything:
- Since we only care about $P(T|W)$
CRF training can be slow

- To compute the gradient, we have to run forward-backward
- CRF training can be as slow as your HMM program *times* your max-ent program
- Using someone else’s software highly recommended
CRF Suite

Recommended software for the project:

- Like Megam, you transform your data into a feature file
- It learns the weights

Data format:

```
LABEL FEAT FEAT ...
LABEL FEAT FEAT ...
<------ end of sentence
LABEL FEAT FEAT ...
```

CRF Suite automatically computes $\theta$ values for:

- All $\theta_{t' \rightarrow t}$ (based on the labels you use)
- All $\theta_{t \rightarrow f}$ (based on the features you use)
- No way to specify other feature types such as $t'$, $t$, $w$ in this software
Using CRFs as chunkers

Typical problem: find and label subsequences of text:

Named entity recognition
Pick out all strings referring to specific individual in the world
  ▶ Identify entity type (person/organization/place/date...)
  “[PERS Mary ] works for [ORG The Ohio State University ]”

Sequence labeling:
  ▶ Because labeling of name words affected by context
  ▶ “[PLACE Ohio ]”, and “a university”
Standard problem setup

- Four tags per NE label
- *BEGIN-ORG, IN-ORG, LAST-ORG, UNIT-ORG* (one word)
- And neutral *OUTSIDE* tag

“Students *OUT* at *OUT* Ohio *BEGIN-ORG* State *IN-ORG* University *LAST-ORG* are *OUT* studying *OUT*”
“Students *OUT* at *OUT* OSU *UNIT-ORG* are *OUT* studying *OUT*”
(Sometimes called *BILOU* coding)
- Could just use *ORG* and *OUT*, but not as good
Relation extraction

“Extracting Relation Descriptors with Conditional Random Fields”, Li, Jiang, Chieu, Chai 2011
Task: extract entity pairs that have some relationship
  ▶ Employment: “said ARG-1, a vice president at ARG-2”
  ▶ Personal: “ARG-1 later married ARG-2”
Basic solution: treat as sequence labeling with tags
  ARG-1-EMPLOYMENT, ARG-2-EMPLOYMENT, ARG-1-PERSONAL etc
  ▶ Using BILOU-like coding scheme
Issues

Task has long-distance dependencies (only one ARG-1 and ARG-2 per relation per sentence, etc)

▶ Markov property violated
▶ Modify dynamic program by adding some long-distance features
  ▶ Exponential slowdown, but this is research, I guess
▶ Sequence of words between ARG tags
▶ Standard CRF scores 73% F-score
▶ Modified CRF up to 80%
▶ No timing figures in paper...
Hindi NER


Find all PERSON, LOCATION, ORGANIZATION in Hindi news:

▶ Throw in word, characters from word, prefix/suffix, lists of Hindi NEs
▶ Use stepwise feature selection procedure
  ▶ Add batch of features
  ▶ Calculate approximate gain in LL from each one
  ▶ Keep the good ones
  ▶ Repeat
▶ Search over features, conjunctions of features

∼70% F-score
Transliterating Chinese

“Forward-backward Machine Transliteration between English and Chinese Based on Combined CRFs”, Qin and Chen 2011 Workshop on Named Entities

Use two CRFs:
- First to find “chunks” of characters that represent a sound
  - BILOU-like encoding
- Second to label each chunk with some foreign characters

Results aren’t great:
- En->Zh: ~70% characters correct
- Zh->En: ~76% characters correct
- Whole name about 30% of the time