NLP 201: Logistic Regression and CRFs

Jeffrey Flanigan

University of California Santa Cruz
jmflanig@ucsc.edu

Many slides and figures from Greg Durret, David Bamman, and Noah Smith

Fall 2022

Plan

- Generative vs Discriminative Classifiers
- Multiclass logistic regression
- Conditional random fields (CRFs)
- Maximum entropy markov models (MEMMs) and the label bias problem

Generative Models

So far, most of our models have been **generative models** - models that specify the joint probability of the data.

- Naive Bayes: $Pr(x, y)$, where x is the input and y is the label
- Language models: $Pr(x)$, where x is a sentence
- HMMs: $Pr(x, y)$, where x is a sentence and y is a tag sequence
- Bayesian networks: $Pr(x_1, \ldots, x_n)$, where $x_1, \ldots x_n$ are random variables

With these models, you can **generate** more data by sampling from the joint probability (using ancestral sampling).

Generative Models

But, when predicting, we only care about the conditional probability:

$$\hat{y} = \operatorname*{argmax}_y p(y|x)$$

We could model $p(y|x)$ directly to get a more expressive model

Generative vs Discriminative Models

Generative models: probabilistic models of $p(x, y)$

Discriminative models: don't model $p(x, y)$, can't compute joint probabilities. May model $p(y|x)$, or may not be probabilistic at all!
Can only **discriminate** between different outputs y

Model this distribution $\rightarrow$	Generative $p(x, y)$	Conditional $p(y|x)$
Classification	Naive Bayes	Multi-class Logistic Regression
Sequences	HMM	Linear-chain CRF

This lecture: we will cover these conditional models

Relationships between NB, LR, HMMs and CRFs

- If you train a Naive Bayes classifier model to maximize $p(y|x)$ instead of $p(x, y)$, you get logistic regression with a specific set of features!
- If you train an HMM model to maximize $p(y|x)$ instead of $p(x, y)$, you get a CRF with a specific set of features!

Classification

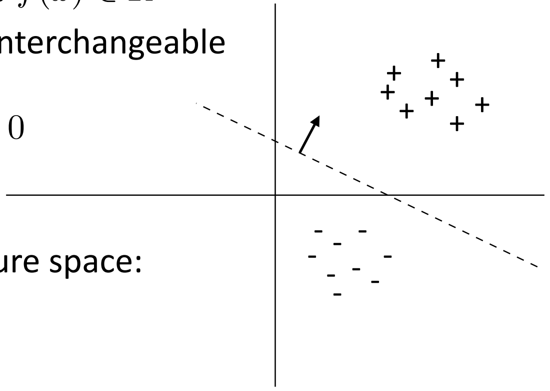
- Datapoint x with label $y \in \{0, 1\}$
- Embed datapoint in a feature space $f(x) \in \mathbb{R}^n$
  but in this lecture $f(x)$ and x are interchangeable
- Linear decision rule: $w^\top f(x) + b > 0$

  $w^\top f(x) > 0$
- Can delete bias if we augment feature space:

  $f(x)$ = [0.5, 1.6, 0.3]

  $\downarrow$

  [0.5, 1.6, 0.3, **1**]



Classification: Sentiment Analysis

this movie was great! would watch again Positive

that film was awful, I'll never watch again Negative

- Surface cues can basically tell you what's going on here: presence or absence of certain words (*great*, *awful*)
- Steps to classification:
  - Turn examples like this into feature vectors
  - Pick a model / learning algorithm
  - Train weights on data to get our classifier

Feature Representation

this movie was great! would watch again Positive

- Convert this example to a vector using *bag-of-words features*

[contains *the*]	[contains *a*]	[contains *was*]	[contains *movie*]	[contains *film*]	…
position 0	position 1	position 2	position 3	position 4	
$f(x)$ = [0	0	1	1	0	…

- Very large vector space (size of vocabulary), sparse features (how many?)
- Requires *indexing* the features (mapping them to axes)
- More sophisticated feature mappings possible (tf-idf), as well as lots of other features: n-grams, character n-grams, parts of speech, lemmas, …

Generative vs. Discriminative Modeling

- Data point $x = (x_1, ..., x_n)$, label $y \in \{0, 1\}$
- Generative models: probabilistic models of P(x,y)
  - Compute $P(y|x)$, predict $\text{argmax}_y P(y|x)$ to classify

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)} \propto P(y)P(x|y)$$

“proportional to”

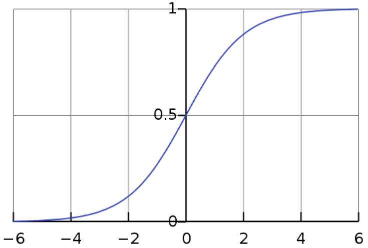
  - Examples: Naive Bayes (see textbook), Hidden Markov Models
- Discriminative models model P(y|x) directly, compute $\text{argmax}_y P(y|x)$
  - Examples: logistic regression
  - Cannot draw samples of *x*, but typically better classifiers

Logistic Regression

Logistic Regression

$$P(y=+|x) = \text{logistic}(w^\top x)$$

$$P(y=+|x) = \frac{\exp(\sum_{i=1}^n w_i x_i)}{1+\exp(\sum_{i=1}^n w_i x_i)}$$



- To learn weights: maximize discriminative log likelihood of data (log P(y|x))

$$\mathcal{L}(\{x_j, y_j\}_{j=1,\ldots,n}) = \sum_j \log P(y_j|x_j)$$ corpus-level LL

$$\mathcal{L}(x_j, y_j=+) = \log P(y_j=+|x_j)$$ one (positive) example LL

$$= \sum_{i=1}^n w_i x_{ji} - \log\left(1+\exp\left(\sum_{i=1}^n w_i x_{ji}\right)\right)$$

sum over features $\rightarrow$

Regularization

- Regularizing an objective can mean many things, including an L2-norm penalty to the weights:

$$\sum_{j=1}^{m} \mathcal{L}(x_j, y_j) - \lambda \|w\|_2^2$$

- Keeping weights small can prevent overfitting
- For most of the NLP models we build, explicit regularization isn't necessary
  - Early stopping
  - Large numbers of sparse features are hard to overfit in a really bad way
  - For neural networks: dropout and gradient clipping

Logistic Regression: Summary

- Model

$$P(y = +|x) = \frac{\exp(\sum_{i=1}^{n} w_i x_i)}{1 + \exp(\sum_{i=1}^{n} w_i x_i)}$$

- Inference

$$\text{argmax}_y P(y|x)$$

$$P(y = 1|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$$

- Learning: gradient ascent on the (regularized) discriminative log-likelihood

Multiclass Fundamentals

Text Classification

A Cancer Conundrum: Too Many Drug Trials, Too Few Patients

Breakthroughs in immunotherapy and a rush to develop profitable new treatments have brought a crush of clinical trials scrambling for patients.

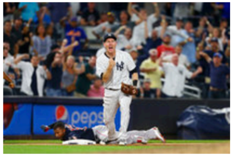
By GINA KOLATA

$\longrightarrow$ Health

Yankees and Mets Are on Opposite Tracks This Subway Series

As they meet for a four-game series, the Yankees are playing for a postseason spot, and the most the Mets can hope for is to play spoiler.

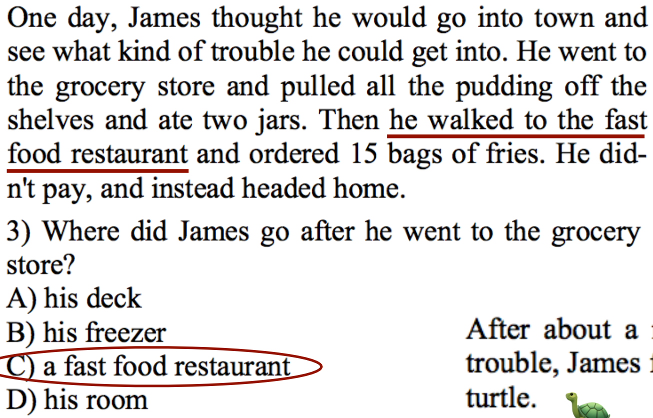
By FILIP BONDY



$\longrightarrow$ Sports

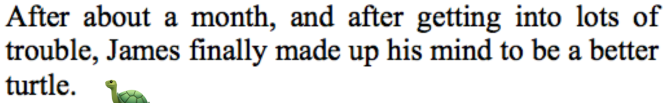
~20 classes

Reading Comprehension



One day, James thought he would go into town and see what kind of trouble he could get into. He went to the grocery store and pulled all the pudding off the shelves and ate two jars. Then he walked to the fast food restaurant and ordered 15 bags of fries. He did-n't pay, and instead headed home.

3) Where did James go after he went to the grocery store?
A) his deck
B) his freezer
[✓ C) a fast food restaurant]
D) his room



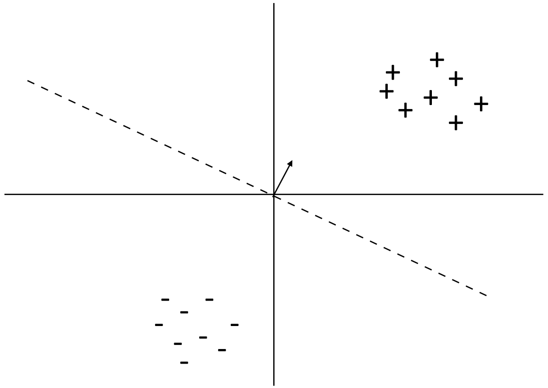
After about a month, and after getting into lots of trouble, James finally made up his mind to be a better turtle. 🐢

- Multiple choice questions, 4 classes (but classes change per example)

Richardson (2013)

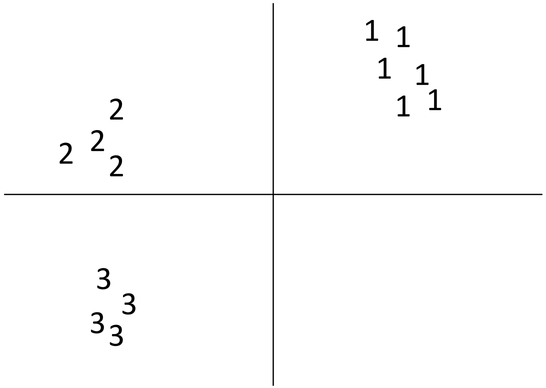
Binary Classification

- Binary classification: one weight vector defines positive and negative classes



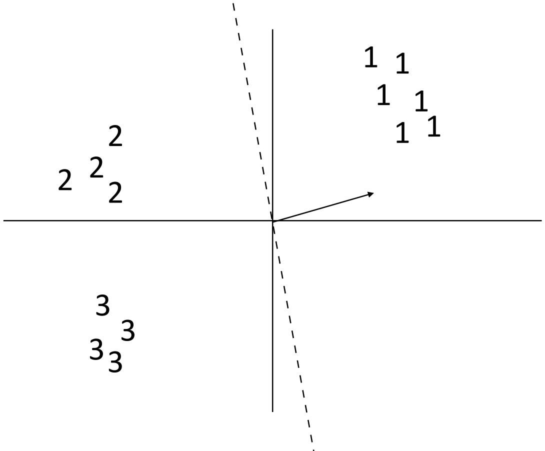
Multiclass Classification

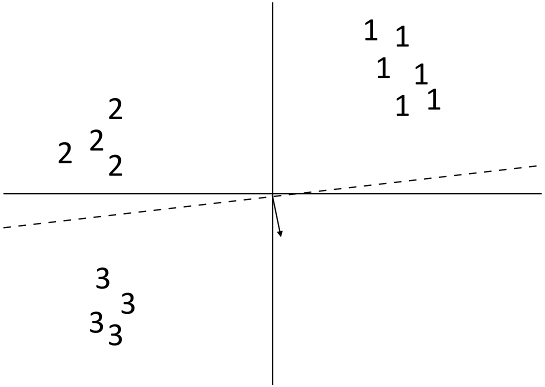
- Can we just use binary classifiers here?



Multiclass Classification

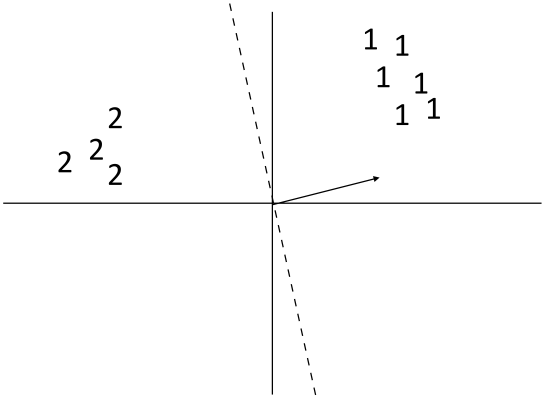
- One-vs-all: train *k* classifiers, one to distinguish each class from all the rest
- How do we reconcile multiple positive predictions? Highest score?

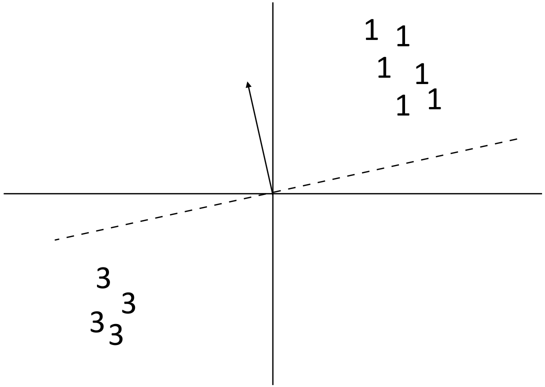




Multiclass Classification

- All-vs-all: train n(n-1)/2 classifiers to differentiate each pair of classes
- Again, how to reconcile?

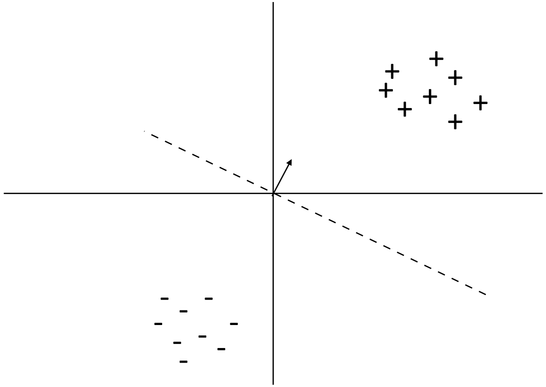


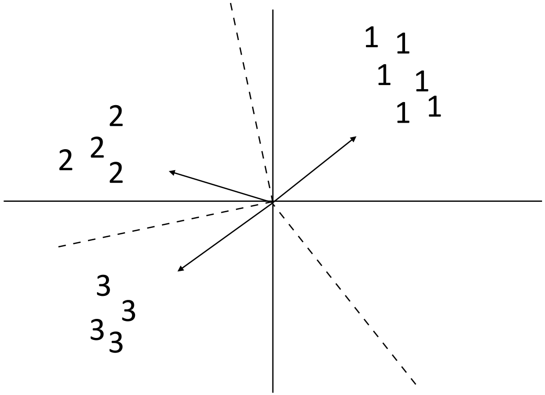


Multiclass Classification

- Binary classification: one weight vector defines both classes

- Multiclass classification: different weights and/or features per class





Multiclass Classification

- Formally: instead of two labels, we have an output space $\mathcal{Y}$ containing a number of possible classes
  - Same machinery that we'll use later for exponentially large output spaces, including sequences and trees
- Decision rule: $\text{argmax}_{y \in \mathcal{Y}} w^\top f(x, y)$ $\longleftarrow$ features depend on choice of label now! note: this isn't the gold label
  - Multiple feature vectors, one weight vector
- Can also have one weight vector per class: $\text{argmax}_{y \in \mathcal{Y}} w_y^\top f(x)$

Different Weights vs. Different Features

- Different features: $\text{argmax}_{y \in \mathcal{Y}} w^\top f(x, y)$
  - Suppose $\mathcal{Y}$ is a structured label space (part-of-speech tags for each word in a sentence). *f*(*x*,*y*) extracts features over shared parts of these
- Different weights: $\text{argmax}_{y \in \mathcal{Y}} w_y^\top f(x)$
  - Generalizes to neural networks: *f*(*x*) is the first *n*-1 layers of the network, then you multiply by a final linear layer at the end
- For linear multiclass classification with discrete classes, these are identical

Feature Extraction

Block Feature Vectors

- Decision rule: $\text{argmax}_{y \in \mathcal{Y}} w^\top f(x, y)$

$$\begin{tikzcd} & \text{Health} \\ \textit{too many drug trials, too few patients} \arrow[ur] \arrow[r] \arrow[dr] & \text{Sports} \\ & \text{Science} \end{tikzcd}$$

- Base feature function:

$f(x)$ = I[contains *drug*], I[contains *patients*], I[contains *baseball*] = [1, 1, 0]

feature vector blocks for each label

$f(x, y =$ Health $) =$ [1, 1, 0, 0, 0, 0, 0, 0, 0]   I[contains *drug* & label = Health]

$f(x, y =$ Sports $) =$ [0, 0, 0, 1, 1, 0, 0, 0, 0]

- Equivalent to having three weight vectors in this case
- We are NOT looking at the gold label! Instead looking at the candidate label

Making Decisions

$$\begin{tikzcd}
 & \text{Health} \\
\textit{too many drug trials, too few patients} \arrow[ur] \arrow[r] \arrow[dr] & \text{Sports} \\
 & \text{Science}
\end{tikzcd}$$

$f(x)$ = I[contains *drug*], I[contains *patients*], I[contains *baseball*]

$f(x, y =$ Health $) =$ [1, 1, 0, 0, 0, 0, 0, 0, 0]

$f(x, y =$ Sports $) =$ [0, 0, 0, 1, 1, 0, 0, 0, 0]

"word drug in Science article" = +1.1

$w =$ [+2.1, +2.3, -5, -2.1, -3.8, +5.2, +1.1, -1.7, -1.3]

$w^\top f(x, y)$ = Health: +4.4 Sports: -5.9 Science: -0.6

argmax

Features

- As a discriminative classifier, logistic regression doesn't assume features are independent like Naive Bayes does.

- Its power partly comes in the ability to create richly expressive features without the burden of independence.

- We can represent text through features that are not just the identities of individual words, but any feature that is scoped over the entirety of the input.

features
contains like
has word that shows up in positive sentiment dictionary
review begins with "I like"
at least 5 mentions of positive affectual verbs (like, love, etc.)

Features

- Features are where you can encode your own domain understanding of the problem.

feature classes
unigrams (“like”)
bigrams (“not like”), higher order ngrams
prefixes (words that start with “un-”)
has word that shows up in positive sentiment dictionary

Feature Representation Revisited

this movie was great! would watch again Positive

[contains *the*] [contains *a*] [contains *was*] [contains *movie*] [contains *film*]
position 0 position 1 position 2 position 3 position 4 …

- Bag-of-words features are position-insensitive
- What about for tasks like classifying a word as a given part-of-speech?

this movie was great! would watch again

- Want features extracted with respect to this particular position
- [curr word = *was*], [prev word = *movie*], [next word = *great*].
  - How many features?

Multiclass Logistic Regression

Multiclass Logistic Regression

$$P_w(y|x) = \frac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$$

↗ sum over output space to normalize

- Compare to binary:

$$P(y=1|x) = \frac{\exp(w^\top f(x))}{1+\exp(w^\top f(x))}$$

negative class implicitly had *f*(*x, y*=0) = the zero vector

- exp/sum(exp): also called *softmax*
- Training: maximize $\mathcal{L}(x,y) = \sum_{j=1}^{n} \log P(y_j^*|x_j)$

$$= \sum_{j=1}^{n} \left(w^\top f(x_j, y_j^*) - \log \sum_{y} \exp(w^\top f(x_j, y)) \right)$$

Training

- Multiclass logistic regression $P_w(y|x) = \frac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$

- Likelihood $\mathcal{L}(x_j, y_j^*) = w^\top f(x_j, y_j^*) - \log \sum_y \exp(w^\top f(x_j, y))$

$$\frac{\partial}{\partial w_i}\mathcal{L}(x_j, y_j^*) = f_i(x_j, y_j^*) - \frac{\sum_y f_i(x_j, y)\exp(w^\top f(x_j, y))}{\sum_y \exp(w^\top f(x_j, y))}$$

$$\frac{\partial}{\partial w_i}\mathcal{L}(x_j, y_j^*) = f_i(x_j, y_j^*) - \sum_y f_i(x_j, y) P_w(y|x_j)$$

$$\frac{\partial}{\partial w_i}\mathcal{L}(x_j, y_j^*) = f_i(x_j, y_j^*) - \mathbb{E}_y[f_i(x_j, y)]$$

model's expectation of feature value

gold feature value

Logistic Regression $\rightarrow$ CRFs

Logistic regression:

$$P(y|x) = \frac{\exp(\theta \cdot f(x,y))}{\sum_{y'} \exp(\theta \cdot f(x,y'))}$$

Trained to maximize conditional log probability of the data:

$$\sum_{i=1}^{N} \log P(y_i|x_i)$$

Predictions:

$$\hat{y} = \operatorname*{argmax}_{y} P(y|x)$$

Logistic Regression $\rightarrow$ CRFs

Logistic regression:

$$P(y|x) = \frac{\exp(\theta \cdot f(x, y))}{\sum_{y'} \exp(\theta \cdot f(x, y'))}$$

CRF:

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

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are now **sequences**.
Trained to maximize conditional log probability of the data (same as logistic regression):

$$\sum_{i=1}^{N} \log P(\boldsymbol{y}_i|\boldsymbol{x}_i)$$

Predictions: $\hat{\boldsymbol{y}} = \operatorname{argmax}_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x})$ (found using Viterbi algorithm)

Viterbi revisited: arbitrary scoring functions

- In log space, Viterbi finds

$$
\begin{aligned}
\hat{\mathbf{t}} &= \operatorname*{argmax}_{\mathbf{t}} \log(P(\mathbf{t}|\mathbf{w})) \\
&= \operatorname*{argmax}_{\mathbf{t}} \sum_{i=1}^{n} \log(P(w_i|t_i)P(t_i|t_{i-1})) \\
&= \operatorname*{argmax}_{\mathbf{t}} \sum_{i=1}^{n} s(w_i, t_i, t_{i-1})
\end{aligned}
$$

  where $s(w_i, t_i, t_{i-1}) = \log(P(w_i|t_i)P(t_i|t_{i-1}))$

- Viterbi algorithm will work with any scoring function, even if it depends on the position i and the entire input sequence $\mathbf{w}$ (you can check that the proof still works)

Viterbi and Forward with arbitrary scoring functions

- Viterbi algorithm can find the exact argmax

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y}} \sum_{i=1}^{n} s(\mathbf{x}, i, y_i, y_{i-1})$$

- Works for any scoring function and any semiring
- In particular, Forward algorithm can compute:

$$\sum_{\mathbf{y}} \prod_{i=1}^{n} e^{s(\mathbf{x}, i, y_i, y_{i-1})}$$
$$= \sum_{\mathbf{y}} e^{\sum_{i=1}^{n} s(\mathbf{x}, i, y_i, y_{i-1})}$$

(We are using Forward algorithm with the local scoring function $s'(\mathbf{x}, i, y_i, y_{i-1}) = e^{s(\mathbf{x}, i, y_i, y_{i-1})}$)

How do we compute $P(\boldsymbol{y}|\boldsymbol{x})$ during training?

Forward algorithm!
CRF:

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

Using Forward algorithm, we can compute:

$$Z = \sum_{\boldsymbol{y}'} e^{score(\boldsymbol{x}, \boldsymbol{y}')}$$

as long as $score$ is a sum of "local parts":

$$score(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} s(\mathbf{x}, i, y_i, y_{i-1})$$

Summary: CRFs

Training: maximize $\sum_{i=1}^{N} \log P_\theta(\boldsymbol{y}_i|\boldsymbol{x}_i)$ (using gradient ascent)

$$\log(P_\theta(\boldsymbol{y}|\boldsymbol{x})) = score_\theta(\boldsymbol{x}, \boldsymbol{y}')) - \log(Z_\theta)$$

where $Z_\theta = \sum_{\boldsymbol{y}'} \exp(score_\theta(\boldsymbol{x}, \boldsymbol{y}'))$ is computed using the **Forward algorithm**.
The $score$ is a sum of "local parts":

$$score_\theta(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$$

Predictions: $\hat{\boldsymbol{y}} = \text{argmax}_{\boldsymbol{y}} P_\theta(\boldsymbol{y}|\boldsymbol{x}) = \text{argmax}_{\boldsymbol{y}} \log(P_\theta(\boldsymbol{y}|\boldsymbol{x}))$
$= \text{argmax}_{\mathbf{y}} \sum_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$ (found using **Viterbi algorithm**)

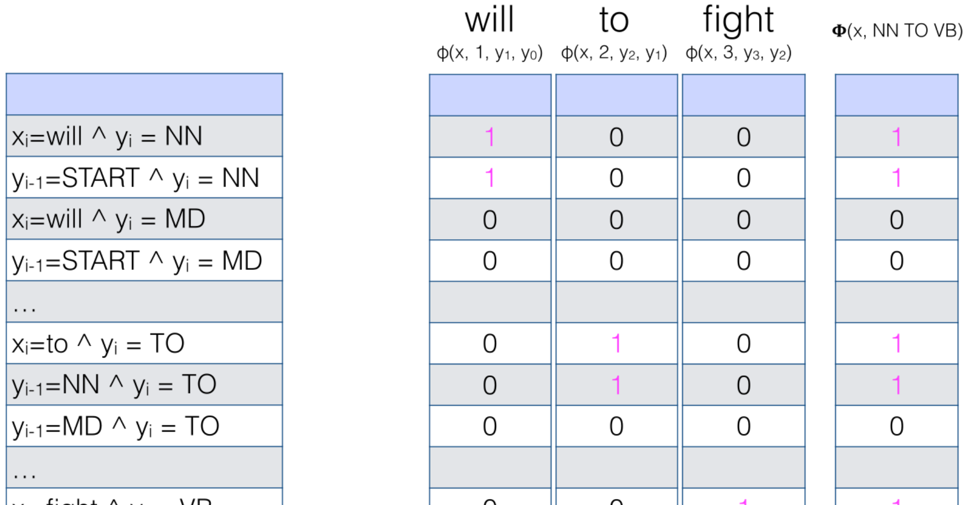
Sequence Labeling with Linear Scoring Function

$$\begin{aligned} score_\theta(\boldsymbol{x}, \boldsymbol{y}) &= \sum_{i=0}^{n} s_\theta(\boldsymbol{x}, i, y_i, y_{i-1}) \\ &= \sum_{i=0}^{n} \theta \cdot f(\boldsymbol{x}, i, y_i, y_{i-1}) \\ &= \theta \cdot \sum_{i=0}^{n} f(\boldsymbol{x}, i, y_i, y_{i-1}) \\ &= \theta \cdot F(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

$s_\theta(\boldsymbol{x}, i, y_i, y_{i-1}) = \theta \cdot f(\boldsymbol{x}, i, y_i, y_{i-1})$ are **local factors**
$f(\boldsymbol{x}, i, y_i, y_{i-1})$ are **local features**
$F(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} f(\boldsymbol{x}, i, y_i, y_{i-1})$ is the total feature vector

Implementation: Features in Linear Models

In linear models, the easiest way to represent feature vectors is a **dictionary from strings to features values**.



	will $\phi(x, 1, y_1, y_0)$	to $\phi(x, 2, y_2, y_1)$	fight $\phi(x, 3, y_3, y_2)$	$\Phi(x, \text{NN TO VB})$
x_i=will ^ y_i = NN	1	0	0	1
y_{i-1}=START ^ y_i = NN	1	0	0	1
x_i=will ^ y_i = MD	0	0	0	0
y_{i-1}=START ^ y_i = MD	0	0	0	0
…				
x_i=to ^ y_i = TO	0	1	0	1
y_{i-1}=NN ^ y_i = TO	0	1	0	1
y_{i-1}=MD ^ y_i = TO	0	0	0	0
…				
x_i=fight ^ y_i = VB	0	0	1	1

Learning for CRFs

For CRFs implemented with a NN library, you can just compute

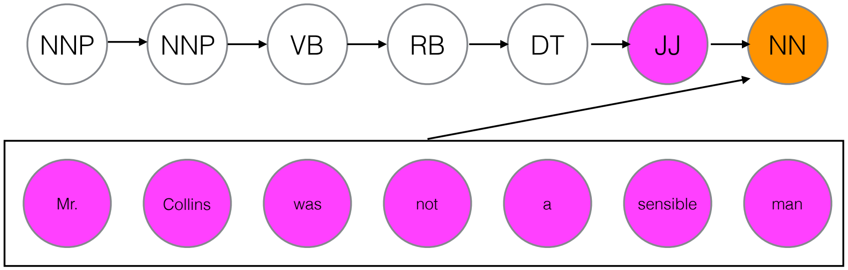
$$Z = \sum_{\boldsymbol{y}'} e^{score(\boldsymbol{x},\boldsymbol{y}')}$$

using the Forward algorithm, and use automatic differentiation to compute the gradient.

Learn the parameters using stochastic gradient descent, Adam, etc.

Maximum-Entropy Markov Models (MEMMs) and the label bias problem

MEMM



Features

$$f(y_i, y_{i-1}; x_1, \ldots, x_n)$$

Features are scoped over the previous predicted tag and the entire observed input

feature	example
x_i = man	1
y_{i-1} = JJ	1
i=n (last word of sentence)	1
x_i ends in -ly	0

Training

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

For all training data, we want probability of the true label y_i conditioned on the previous true label y_{i-1} to be high.

This is simply multiclass logistic regression

Decoding

- With logistic regression, our prediction is simply the argmax y:

$$P(y \mid x, \beta)$$

- With an MEMM, we know the true y_{i-1} during training but we never of course know it at test time

$$P(y_i \mid y_{i-1}, x, \beta)$$

Greedy decoding

- A i=1, predict the argmax given START:

$$P(y_1 \mid START, x, \beta)$$

- For each subsequent time step, condition on the y just predicted during the step before

$$P(y_i \mid y_{i-1}, x, \beta)$$

Viterbi decoding

Viterbi for HMM: max joint probability $P(y)P(x \mid y) = P(x, y)$

$$v_t(y) = \max_{u \in \mathcal{Y}} \left[v_{t-1}(u) \times P(y_t = y \mid y_{t-1} = u) P(x_t \mid y_t = y) \right]$$

Viterbi for MEMM: max conditional probability $P(y \mid x)$

$$v_t(y) = \max_{u \in \mathcal{Y}} [v_{t-1}(u) \times P(y_t = y \mid y_{t-1} = u, x, \beta)]$$

MEMM Training

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

For all training data, we want probability of the true label y_i conditioned on the previous true label y_{i-1} to be high.

This is simply multiclass logistic regression

MEMM Training

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

Locally normalized — at each time step, each conditional distribution sums to 1

Label bias

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

- For a given conditioning context, the probability of a tag (e.g., VBZ) only competes against other tags with that same context (e.g., NN)

Bottou 2001; Lafferty et al. 2001

Label bias

NN TO VB

will to fight

	NN	MD
x_i=will	10	40
y_{i-1}=START	-1	7
BIAS	7	-2

Modals show up much more frequently at the start of the sentence than nouns do (e.g., questions)

Toutanova et al. 2003

Label bias

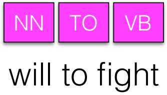
NN	TO	VB
will	to	fight

But we know that MD + TO is very rare

- *can to eat
- *would to eat
- *could to eat
- *may to eat

Toutanova et al. 2003

Label bias



	TO
x_i=to	10000000
y_{i-1}=NN	0
y_{i-1}=MD	0

to is relatively deterministic (almost always TO) so it doesn't matter what tag precedes it.

Toutanova et al. 2003

Label bias

NN	TO	VB
will	to	fight

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

Because of this local normalization, P(TO | context) will always be 1 if x=“to”

Toutanova et al. 2003

Label bias

NN	TO	VB
will	to	fight

That means our prediction for *to* can't help us disambiguate *will*. We lose the information that MD + TO sequences rarely happen.

Toutanova et al. 2003

Label bias

Viterbi decoding doesn’t help in this case

$$v_t(y) = \max_{u \in \mathcal{Y}} [v_{t-1}(u) \times P(y_t = y \mid y_{t-1} = u, x, \beta)]$$

$$P(y_t = \text{TO} \mid y_{t-1} = \text{MD}, x, \beta) = 1$$

$$P(y_t = \text{TO} \mid y_{t-1} = \text{NN}, x, \beta) = 1$$

Conditional random fields

- We can solve this problem using global normalization (over the entire sequences) rather than locally normalized factors.

MEMM

$$P(y \mid x, \beta) = \prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

CRF

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x, y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x, y')^\top \beta)}$$

Conditional random fields

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x, y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x, y')^\top \beta)}$$

Feature vector scoped over the entire input and label sequence

$$\Phi(x, y) = \sum_{i=1}^{n} \phi(x, i, y_i, y_{i-1})$$

ϕ is the same feature vector we used for local predictions using MEMMs

Features

$$\phi(x, i, y_i, y_{i-1})$$

Features are scoped over the previous predicted tag and the entire observed input

feature	example
x_i = man	1
y_{i-1} = JJ	1
i=n (last word of sentence)	1
x_i ends in -ly	0

In a CRF, we use features from the entire sequence (by summing the individual features at each time step)

	will $\phi(x, 1, y_1, y_0)$	to $\phi(x, 2, y_2, y_1)$	fight $\phi(x, 3, y_3, y_2)$	$\boldsymbol{\Phi}(x, \text{NN TO VB})$
x_i=will ^ y_i = NN	1	0	0	1
y_{i-1}=START ^ y_i = NN	1	0	0	1
x_i=will ^ y_i = MD	0	0	0	0
y_{i-1}=START ^ y_i = MD	0	0	0	0
…				
x_i=to ^ y_i = TO	0	1	0	1
y_{i-1}=NN ^ y_i = TO	0	1	0	1
y_{i-1}=MD ^ y_i = TO	0	0	0	0
…				
x_i=fight ^ y_i = VB	0	0	1	1
y_{i-1}=TO ^ y_i = VB	0	0	1	1

Implementation: Features in Linear Models

In linear models, the easiest way to represent feature vectors is a **dictionary from strings to features values**.

	will	to	fight	$\Phi(x, \text{NN TO VB})$
	$\phi(x, 1, y_1, y_0)$	$\phi(x, 2, y_2, y_1)$	$\phi(x, 3, y_3, y_2)$	
x_i=will ^ y_i = NN	1	0	0	1
y_{i-1}=START ^ y_i = NN	1	0	0	1
x_i=will ^ y_i = MD	0	0	0	0
y_{i-1}=START ^ y_i = MD	0	0	0	0
...				
x_i=to ^ y_i = TO	0	1	0	1
y_{i-1}=NN ^ y_i = TO	0	1	0	1
y_{i-1}=MD ^ y_i = TO	0	0	0	0
...				
x_i=fight ^ y_i = VB	0	0	1	1

Conditional random fields

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x,y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x,y')^\top \beta)}$$

- In MEMMs, we normalize over the set of 45 POS tags
- CRFs are globally normalized, but the normalization complexity is huge — every possible sequence of labels of length n.

Forward algorithm (CRF)

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x,y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x,y')^\top \beta)}$$

- Calculating the denominator naively would involve a summation over K^N terms
- But we can do this efficiently in NK^2 time using the forward algorithm

For details, see: Collins, “The Forward-Backward Algorithm”

Forward algorithm (CRF)

END							
DT							
NNP							
VB							
NN							
MD							
START							
	^	Janet	will	back	the	bill	$

$$\alpha(1, y) = \exp\left(\phi(x, i, y, \text{START})^\top \beta\right)$$

$$\alpha(i, y) = \sum_{y' \in \mathcal{S}} \alpha(i-1, y') \times \exp\left(\phi(x, i, y, y')^\top \beta\right)$$

Forward algorithm (CRF)

	^	Janet	will	back	the	bill	$
END							
DT							
NNP							
VB							
NN							
MD							
START							

$$Z = \sum_{y' \in \mathcal{Y}} \exp\left(\Phi(x, y')^\top \beta\right) = \sum_{s \in \mathcal{S}} \alpha(n, s)$$

Conditional random fields

With a CRF, we have exactly the same parameters as we do with an equivalent MEMM; but we learn the best values of those parameters that leads to the best probability of the sequence overall (in our training data)

MEMM

	TO
x_i=to ^ y_i = TO	10000000
y_{i-1}=NN ^ y_i = TO	0
y_{i-1}=MD ^ y_i = TO	0

CRF

	TO
x_i=to ^ y_i = TO	7.8
y_{i-1}=NN ^ y_i = TO	1.4
y_{i-1}=MD ^ y_i = TO	-5.8

Learning for CRFs

To train a CRF, we minimize

$$\hat{\beta} = \min_{\beta} - \log P(\mathcal{D}, \beta) = \min_{\beta} \sum_{i=1}^{N} - \log P(y_i | x_i, \beta))$$

$$P(y|x, \beta) = \frac{\exp(\Phi(x, y)^T \beta)}{\sum_{y' \in Y} \exp(\Phi(x, y')^T \beta}$$

$$\begin{aligned} \log P(y|x, \beta) &= \Phi(x, y)^T \beta - \log(\sum_{y' \in Y} \exp(\Phi(x, y')^T \beta)) \\ &= \Phi(x, y)^T \beta - \log(Z) \end{aligned}$$

Z is called the **normalizer** or **partition function**.

Learning for CRFs

For CRFs implemented with a NN library, you can just compute

$$Z = \sum_{y' \in Y} \exp(\Phi(x, y')^T \beta))$$

using the Forward algorithm, and use automatic differentiation to compute the gradient.

Learn the parameters using stochastic gradient descent, Adam, etc.

Why does this work? Viterbi with arbitrary scoring functions

- Viterbi algorithm can find the exact argmax

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y}} \sum_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$$

- Forward algorithm can find the exact sum

$$\hat{\mathbf{y}} = \sum_{\mathbf{y}} \prod_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$$

- **Works for any scoring function and any semiring**
- We can use the Viterbi algorithm to make predictions for a CRF!
- We can use the Forward algorithm to compute Z for a CRF!

CRF with Linear Scoring Function

$$\begin{aligned}
\log(P(\boldsymbol{y}|\boldsymbol{x})) &= \log(\prod_{i=0}^{n} s_\theta(\boldsymbol{x}, i, y_i, y_{i+1})) - \log(Z) \\
&= \sum_{i=0}^{n} \beta \cdot \phi(\boldsymbol{x}, i, y_i, y_{i+1}) - \log(Z) \\
&= \beta \cdot \sum_{i=0}^{n} \phi(\boldsymbol{x}, i, y_i, y_{i+1}) - \log(Z) \\
&= \beta \cdot \Phi(\boldsymbol{x}, \boldsymbol{y}) - \log(Z)
\end{aligned}$$

$\phi(\boldsymbol{x}, i, y_i, y_{i+1})$ are **local features**
$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} \phi(\boldsymbol{x}, i, y_i, y_{i+1})$ is the total feature vector

CRFs with Neural Models

Learning CRFs with a neural model:

- Compute Z(X) (the partition function) with the forward algorithm
- Compute derivative of log likelihood with automatic differentiation

Learning CRFs with a Linear Model

(Old method: Forward-Backward Algorithm)

Parameter estimation

- Just like logistic regression/MEMM, we can find the optimal values for the parameters using stochastic gradient descent for a given sequence x and true labels y.

$$\frac{\partial L}{\partial \beta_k} = \Phi_k(x, y) - \sum_{y' \in \mathcal{Y}} P(y' \mid x, \beta) \Phi_k(x, y')$$

Features for the true label y

Expected feature counts under the probability for a sequence assigned by current β

Parameter estimation

- Just like logistic regression/MEMM, we can find the optimal values for the parameters using stochastic gradient descent for a given sequence x and true labels y.

$$\frac{\partial L}{\partial \beta_k} = \Phi_k(x, y) - \sum_{y' \in \mathcal{Y}} P(y' \mid x, \beta)\Phi_k(x, y')$$

If current model assigns probability of 1 to true sequence and 0 to all other K^N sequences, then gradient = 0

Parameter estimation

$$\sum_{y' \in \mathcal{Y}} P(y' \mid x, \beta) \Phi_k(x, y')$$

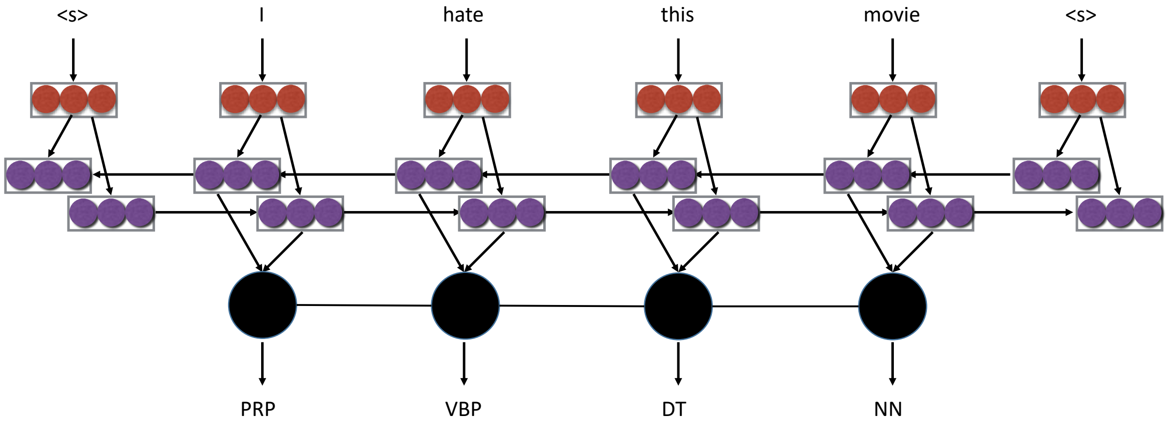
$$= \sum_{i=1}^{n} \sum_{a \in \mathcal{S}, b \in \mathcal{S}} \phi_k(x, i, a, b) \sum_{y' \in \mathcal{Y} : y'_{i-1} = a, y'_i = b} P(y' \mid x, \beta)$$

This entire sum can be found in NK^2 time using the forward-backward algorithm

For details, see: Collins, "The Forward-Backward Algorithm"

Neural CRF Example

BiLSTM-CRF for Sequence Labeling



Potential Functions

- $\psi_i(y_{i-1}, y_i, X) = \exp\left(W^T T(y_{i-1}, y_i, X, i) + U^T\ S(y_i, X, i) + b_{y_{i-1}, y_i}\right)$
  - Using neural features in DNN:
    $$\psi_i(y_{i-1}, y_i, X) = \exp\left(W^T_{y_{i-1}, y_i} F(X, i) + U^T_{y_i}\ F(X, i) + b_{y_{i-1}, y_i}\right)$$
    - Number of parameters: $O(|Y|^2 d_F)$
  - Simpler version:
    $$\psi_i(y_{i-1}, y_i, X) = \exp\left(W_{y_{i-1}, y_i} + U^T_{y_i} F(X, i) + b_{y_{i-1}, y_i}\right)$$
    - Number of parameters: $O(|Y|^2 + |Y| d_F)$

CRF Training & Decoding

- $P(Y|X) = \frac{\prod_{i=1}^{L} \psi_i(y_{i-1}, y_i, X)}{\sum_{Y\prime} \prod_{i=1}^{L} \psi_i(y\prime_{i-1}, y\prime_i, X)} = \frac{\prod_{i=1}^{L} \psi_i(y_{i-1}, y_i, X)}{Z(X)}$
- Training: computing the partition function Z(X)

$$Z(X) = \sum_{Y} \prod_{i=1}^{L} \psi_i(y_{i-1}, y_i, X)$$

- Decoding

$$y^* = argmax_Y P(Y|X)$$

Go through the output space of Y which grows exponentially with the length of the input sequence.

Viterbi Algorithm

- $\pi_t(y|\text{X})$ is the partition of sequence with length equal to t and end with label y:

$$\pi_t(y|X) = \sum_{y_i,\dots,y_{t-1}} \left(\prod_{i=1}^{t-1} \psi_i(y_{i-1}, y_i, X) \right) \psi_t(y_{t-1}, y_t = y, X)$$

$$= \sum_{y_{t-1}} \psi_t(y_{t-1}, y_t = y, X) \sum_{y_i,\dots,y_{t-2}} \left(\prod_{i=1}^{t-2} \psi_i(y_{i-1}, y_i, X) \right) \psi_{t-1}(y_{t-2}, y_{t-1}, X)$$

$$= \sum_{y_{t-1}} \psi_t(y_{t-1}, y_t = y, X) \pi_{t-1}(y_{t-1}|X)$$

- Computing partition function $Z(X) = \sum_y \pi_L(y|X)$

Viterbi Algorithm

- Decoding is performed with similar dynamic programming algorithm
- Calculating gradient: $l_{ML}(X,Y;\theta) = -\log P(Y|X;\theta)$

$$\frac{\partial l_{ML}(X,Y;\theta)}{\partial\theta} = F(Y,X) - E_{P(Y|X;\theta)}[F(Y,X)]$$

  - Forward-backward algorithm (Sutton and McCallum, 2010)
    - Both $P(Y|X;\theta)$ and $F(Y,X)$ can be decomposed
    - Need to compute the marginal distribution:

$$P(y_{i-1} = y', y_i = y|X;\theta) = \frac{\alpha_{i-1}(y'|X)\psi_i(y',y,X)\beta_i(y|X)}{Z(X)}$$

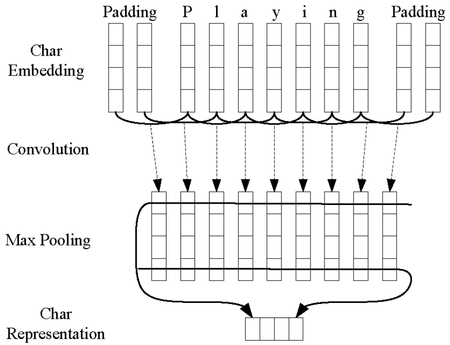
  - Not necessary if using DNN framework (auto-grad)

Case Study: BiLSTM-CNN-CRF for Sequence Labeling (Ma et al, 2016)

- Goal: Build a truly end-to-end neural model for sequence labeling task, requiring no feature engineering and data pre-processing.
- Two levels of representations
  - Character-level representation: CNN
  - Word-level representation: Bi-directional LSTM

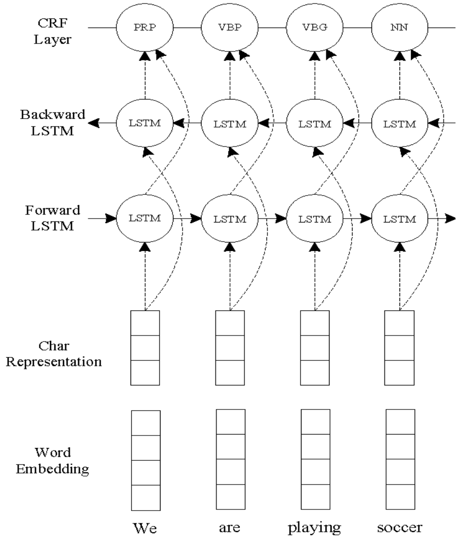
CNN for Character-level representation

- We used CNN to extract morphological information such as prefix or suffix of a word



Bi-LSTM-CNN-CRF

- We used Bi-LSTM to model word-level information.
- CRF is on top of Bi-LSTM to consider the co-relation between labels.



Training Details

- Optimization Algorithm:
  - SGD with momentum (0.9)
  - Learning rate decays with rate 0.05 after each epoch.
- Dropout Training:
  - Applying dropout to regularize the model with fixed dropout rate 0.5
- Parameter Initialization:
  - Parameters: Glorot and Bengio (2010)
  - Word Embedding: Stanford’s GloVe 100-dimentional embeddings
  - Character Embedding: uniformly sampled from $[-\sqrt{\frac{3}{dim}}, +\sqrt{\frac{3}{dim}}]$, where $dim = 30$

Experiments

	POS		**NER**					
	Dev	**Test**	**Dev**			**Test**		
Model	Acc.	Acc.	Prec.	Recall	F1	Prec.	Recall	F1
BRNN	96.56	96.76	92.04	89.13	90.56	87.05	83.88	85.44
BLSTM	96.88	96.93	92.31	90.85	91.57	87.77	86.23	87.00
BLSTM-CNN	97.34	97.33	92.52	93.64	93.07	88.53	90.21	89.36
BLSTM-CNN-CRF	97.46	97.55	94.85	94.63	94.74	91.35	91.06	91.21