Sequential Data Modeling - Conditional Random Fields

Graham Neubig
Nara Institute of Science and Technology (NAIST)

Prediction Problems

Given x, predict y

Prediction Problems

Given x,	predict y	
A book review Oh, man I love this book! This book is so boring...	Is it positive? yes no	Binary Prediction (2 choices)
A tweet On the way to the park! 公園に行くなう！	Its language English Japanese	Multi-class Prediction (several choices)
A sentence I read a book	Its parts-of-speech N VBD DET NN I read a book	Structured Prediction (millions of choices)

Logistic Regression

Example we will use:

- Given an introductory sentence from Wikipedia
- Predict whether the article is about a person

Given		Predict
Gonso was a Sanron sect priest (754-827) in the late Nara and early Heian periods.	→	Yes!
Shichikuzan Chigogataki Fudomyoo is a historical site located at Magura, Maizuru City, Kyoto Prefecture.	→	No!

- This is binary classification (of course!)

Review: Linear Prediction Model

- Each element that helps us predict is a *feature*

contains “priest”	contains “(<#>-<#>)”
contains “site”	contains “Kyoto Prefecture”

- Each feature has a weight, *positive* if it indicates “yes”, and *negative* if it indicates “no”

$w_{\text{contains “priest”}} = 2$ $w_{\text{contains “(<\#>-<\#>)”}} = 1$

$w_{\text{contains “site”}} = -3$ $w_{\text{contains “Kyoto Prefecture”}} = -1$

- For a new example, sum the weights

Kuya (903-972) was a priest born in Kyoto Prefecture. → $2 + -1 + 1 = 2$

- If the sum is at least 0: “yes”, otherwise: “no”

Review: Mathematical Formulation

$$
\begin{aligned}
y &= \operatorname{sign}(\mathbf{w} \cdot \boldsymbol{\varphi}(x)) \\
&= \operatorname{sign}\left(\sum_{i=1}^{I} w_i \cdot \varphi_i(x)\right)
\end{aligned}
$$

- **x**: the input
- **φ(x)**: vector of feature functions $\{\varphi_1(x), \varphi_2(x), \ldots, \varphi_I(x)\}$
- **w**: the weight vector $\{w_1, w_2, \ldots, w_I\}$
- **y**: the prediction, +1 if “yes”, -1 if “no”
  - (sign(v) is +1 if v >= 0, -1 otherwise)

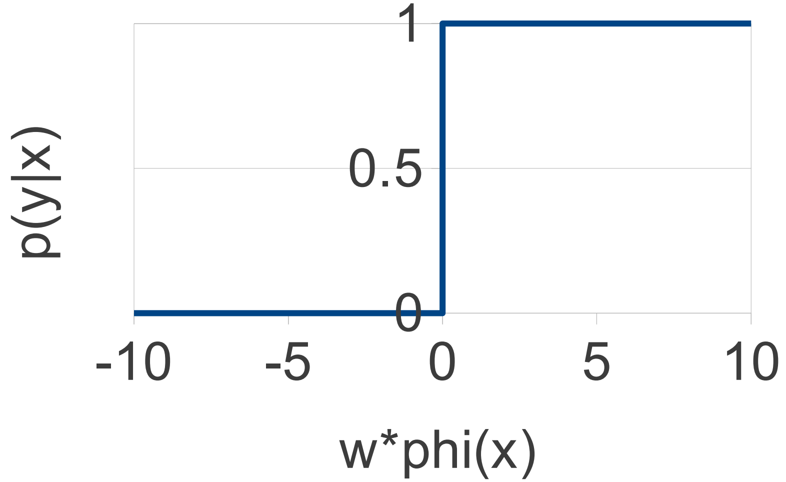
Perceptron and Probabilities

- Sometimes we want the probability $P(y|x)$
  - Estimating confidence in predictions
  - Combining with other systems
- However, perceptron only gives us a prediction

$$y = \text{sign}(\mathbf{w} \cdot \varphi(x))$$

In other words:

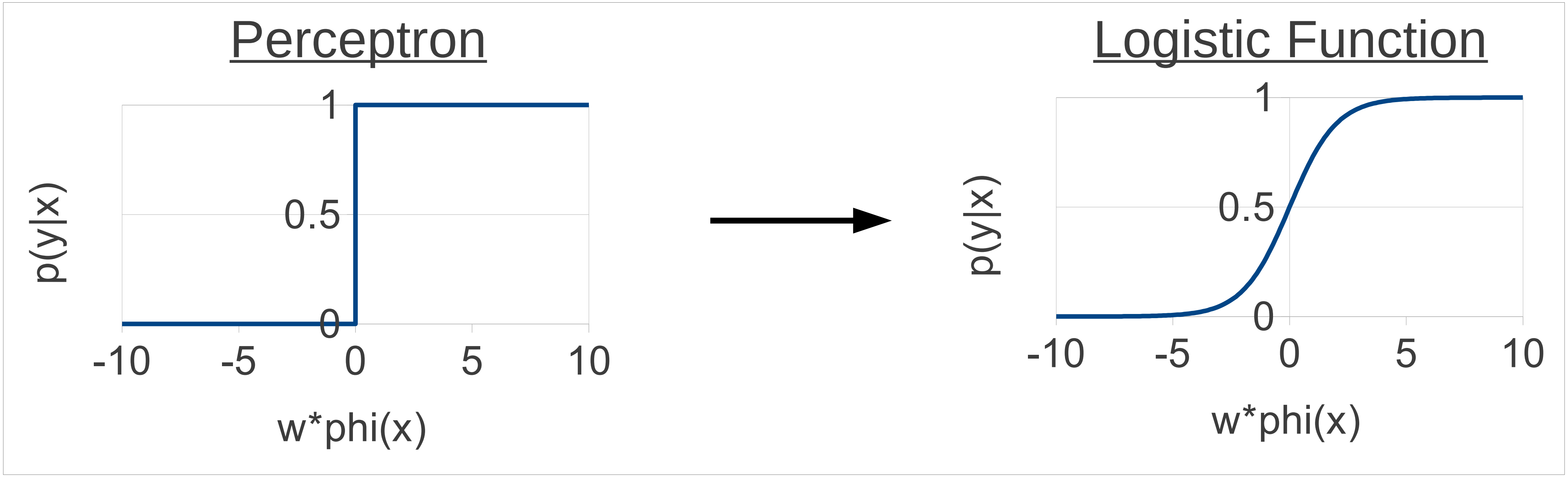
$$P(y=1|x) = 1 \text{ if } \mathbf{w} \cdot \varphi(x) \geq 0$$
$$P(y=1|x) = 0 \text{ if } \mathbf{w} \cdot \varphi(x) < 0$$



The Logistic Function

- The logistic function is a “softened” version of the function used in the perceptron

$$P(y=1|x)=\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$



- Can account for uncertainty
- Differentiable

Logistic Regression

- Train based on conditional likelihood
- Find the parameters **w** that maximize the conditional likelihood of all answers y_i given the example x_i

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \prod_i P(y_i \mid x_i; \boldsymbol{w})$$

- How do we solve this?

Review: Perceptron Training Algorithm

```
create map w
for I iterations
    for each labeled pair x, y in the data
        phi = CREATE_FEATURES(x)
        y' = PREDICT_ONE(w, phi)
        if y' != y
            w += y * phi
```

- In other words
  - Try to classify each training example
  - Every time we make a mistake, update the weights

Stochastic Gradient Descent

- Online training algorithm for probabilistic models (including logistic regression)

```
create map w
for I iterations
    for each labeled pair x, y in the data
        w += α * dP(y|x)/dw
```

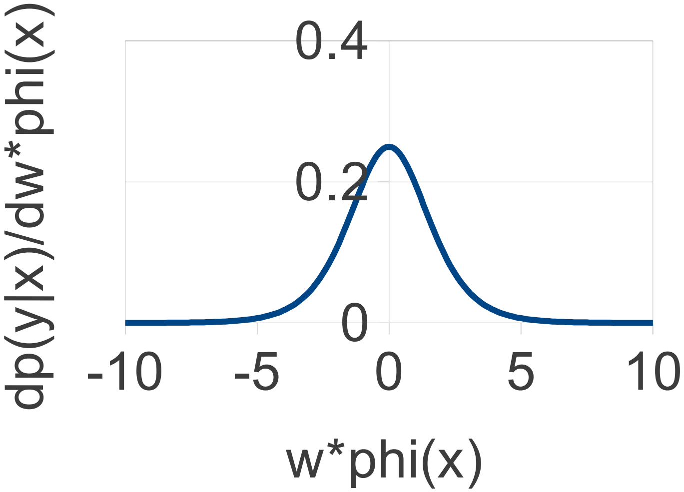
- In other words
  - For every training example, calculate the gradient (the direction that will increase the probability of y)
  - Move in that direction, multiplied by learning rate α

Gradient of the Logistic Function

- Take the derivative of the probability

$$\frac{d}{dw} P(y=1|x) = \frac{d}{dw} \frac{e^{w \cdot \varphi(x)}}{1+e^{w \cdot \varphi(x)}}$$

$$= \varphi(x) \frac{e^{w \cdot \varphi(x)}}{(1+e^{w \cdot \varphi(x)})^2}$$



$$\frac{d}{dw} P(y=-1|x) = \frac{d}{dw} \left(1 - \frac{e^{w \cdot \varphi(x)}}{1+e^{w \cdot \varphi(x)}}\right)$$

$$= -\varphi(x) \frac{e^{w \cdot \varphi(x)}}{(1+e^{w \cdot \varphi(x)})^2}$$

Example: Initial Update

- Set α=1, initialize **w=0**

x = A site , located in Maizuru , Kyoto   y = -1

$$\mathbf{w}\cdot\varphi(x)=0 \qquad \frac{d}{d\,\mathbf{w}}P(y=-1|x) = -\frac{e^0}{(1+e^0)^2}\varphi(x)$$

$$= -0.25\,\varphi(x)$$

$$\mathbf{w} \leftarrow \mathbf{w} + -0.25\,\varphi(x)$$

$w_{\text{unigram “Maizuru”}}$ = -0.25
$w_{\text{unigram “,”}}$ = -0.5
$w_{\text{unigram “in”}}$ = -0.25
$w_{\text{unigram “Kyoto”}}$ = -0.25

$w_{\text{unigram “A”}}$ = -0.25
$w_{\text{unigram “site”}}$ = -0.25
$w_{\text{unigram “located”}}$ = -0.25

Example: Second Update

x = Shoken , monk born in Kyoto y = 1

-0.5 -0.25 -0.25

$$\mathbf{w}\cdot\boldsymbol{\varphi}(x) = -1 \qquad \frac{d}{d\,w}P(y=1|x) = \frac{e^{1}}{(1+e^{1})^{2}}\boldsymbol{\varphi}(x)$$

$$= 0.196\,\boldsymbol{\varphi}(x)$$

$$\mathbf{w} \leftarrow \mathbf{w} + 0.196\,\boldsymbol{\varphi}(x)$$

$w_{\text{unigram “Maizuru”}}$ = -0.25
$w_{\text{unigram “,”}}$ = -0.304
$w_{\text{unigram “in”}}$ = -0.054
$w_{\text{unigram “Kyoto”}}$ = -0.054

$w_{\text{unigram “A”}}$ = -0.25
$w_{\text{unigram “site”}}$ = -0.25
$w_{\text{unigram “located”}}$ = -0.25

$w_{\text{unigram “Shoken”}}$ = 0.196
$w_{\text{unigram “monk”}}$ = 0.196
$w_{\text{unigram “born”}}$ = 0.196

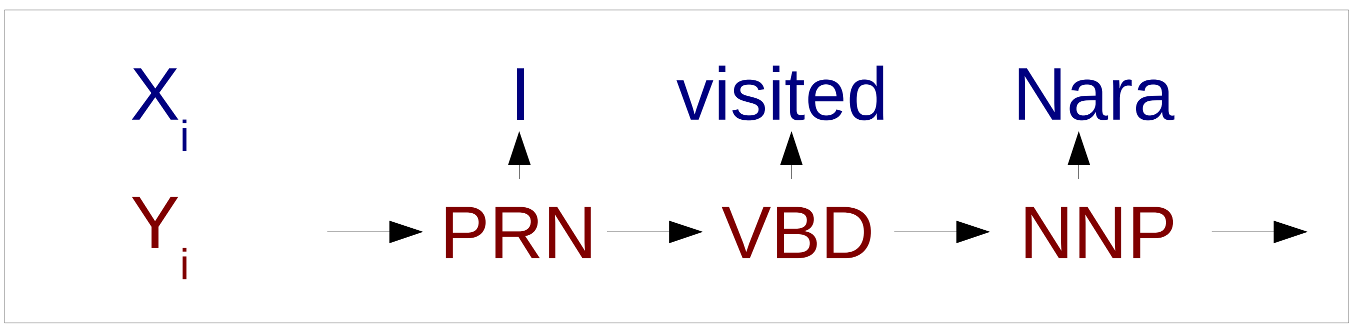
Calculating Optimal Sequences, Probabilities

Sequence Likelihood

- Logistic regression considered probability of $y \in \{-1, +1\}$

$$P(y|x)$$

- What if we want to consider probability of a sequence?



$$P(Y|X)$$

Calculating Multi-class Probabilities

- Each sequence has it's own feature vector

$$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ N & V\end{pmatrix} \quad \varphi_{T,\text{<S>},N}=1 \;\; \varphi_{T,N,V}=1 \;\; \varphi_{T,V,\text{</S>}}=1 \;\; \varphi_{E,N,\text{time}}=1 \;\; \varphi_{E,V,\text{flies}}=1$$

$$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ V & N\end{pmatrix} \quad \varphi_{T,\text{<S>},V}=1 \;\; \varphi_{T,V,N}=1 \;\; \varphi_{T,N,\text{</S>}}=1 \;\; \varphi_{E,V,\text{time}}=1 \;\; \varphi_{E,N,\text{flies}}=1$$

$$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ N & N\end{pmatrix} \quad \varphi_{T,\text{<S>},N}=1 \;\; \varphi_{T,N,N}=1 \;\; \varphi_{T,N,\text{</S>}}=1 \;\; \varphi_{E,N,\text{time}}=1 \;\; \varphi_{E,N,\text{flies}}=1$$

$$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ V & V\end{pmatrix} \quad \varphi_{T,\text{<S>},V}=1 \;\; \varphi_{T,V,V}=1 \;\; \varphi_{T,V,\text{</S>}}=1 \;\; \varphi_{E,V,\text{time}}=1 \;\; \varphi_{E,V,\text{flies}}=1$$

- Use weights for each feature to calculate scores

$$w_{T,\text{<S>},N}=1 \;\; w_{T,V,\text{</S>}}=1 \;\; w_{E,N,\text{time}}=1$$

$$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ N & V\end{pmatrix} * \mathbf{w}=3 \qquad \varphi\begin{pmatrix}\text{time} & \text{flies}\\ V & N\end{pmatrix} * \mathbf{w}=0$$

$$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ N & N\end{pmatrix} * \mathbf{w}=2 \qquad \varphi\begin{pmatrix}\text{time} & \text{flies}\\ V & V\end{pmatrix} * \mathbf{w}=1$$

The Softmax Function

- Turn into probabilities by taking exponent and normalizing (the Softmax function)

$$P(Y|X) = \frac{e^{\mathbf{w} \cdot \boldsymbol{\varphi}(Y,X)}}{\sum_{\tilde{Y}} e^{\mathbf{w} \cdot \boldsymbol{\varphi}(\tilde{Y},X)}}$$

- Take the exponent and normalize

$\exp(\boldsymbol{\varphi}(\substack{\text{time flies}\\ \text{N V}})*\mathbf{w})=20.08$ $\exp(\boldsymbol{\varphi}(\substack{\text{time flies}\\ \text{V N}})*\mathbf{w})=1.00$

$\exp(\boldsymbol{\varphi}(\substack{\text{time flies}\\ \text{N N}})*\mathbf{w})=7.39$ $\exp(\boldsymbol{\varphi}(\substack{\text{time flies}\\ \text{V V}})*\mathbf{w})=2.72$

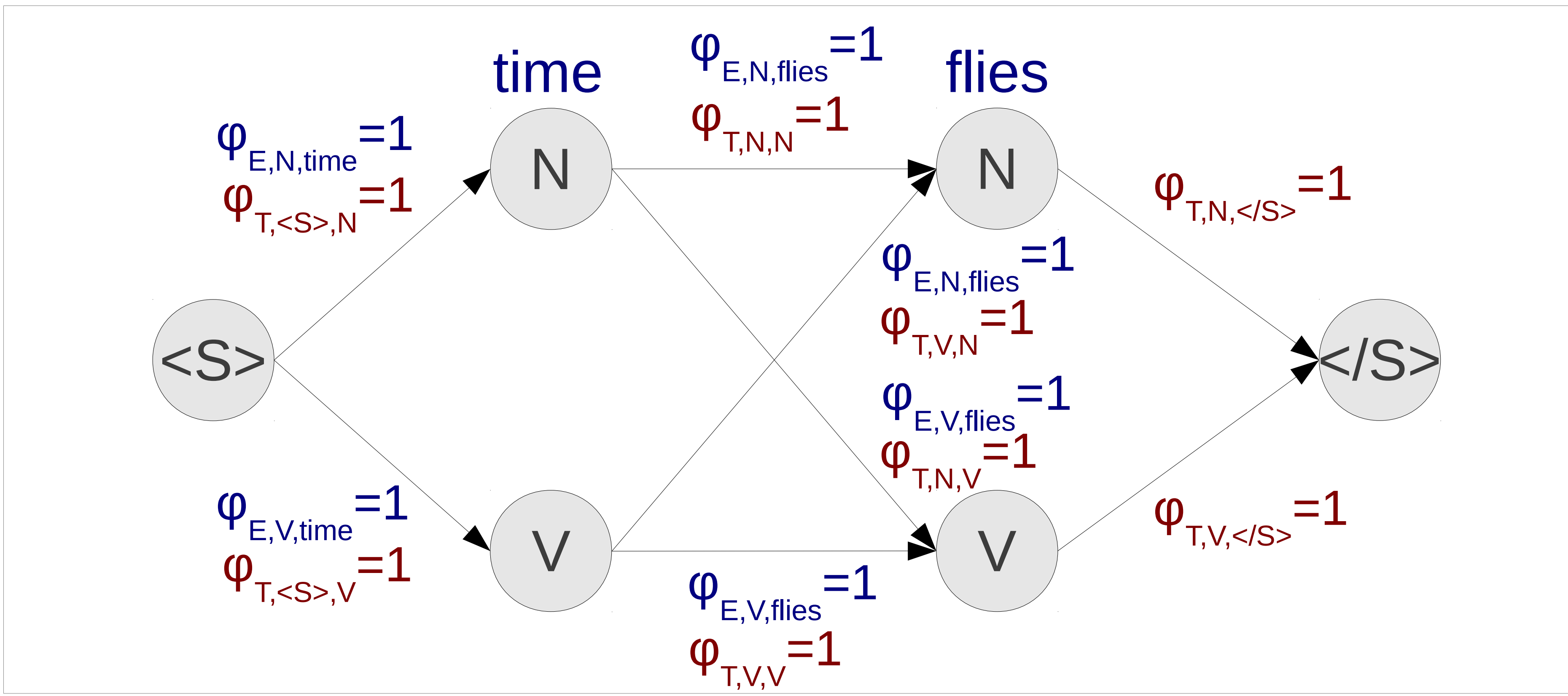
↓

P(N V | time flies)=.6437 P(V N | time flies)=0.0320

P(N N | time flies)=.2369 P(V V | time flies)=0.0872

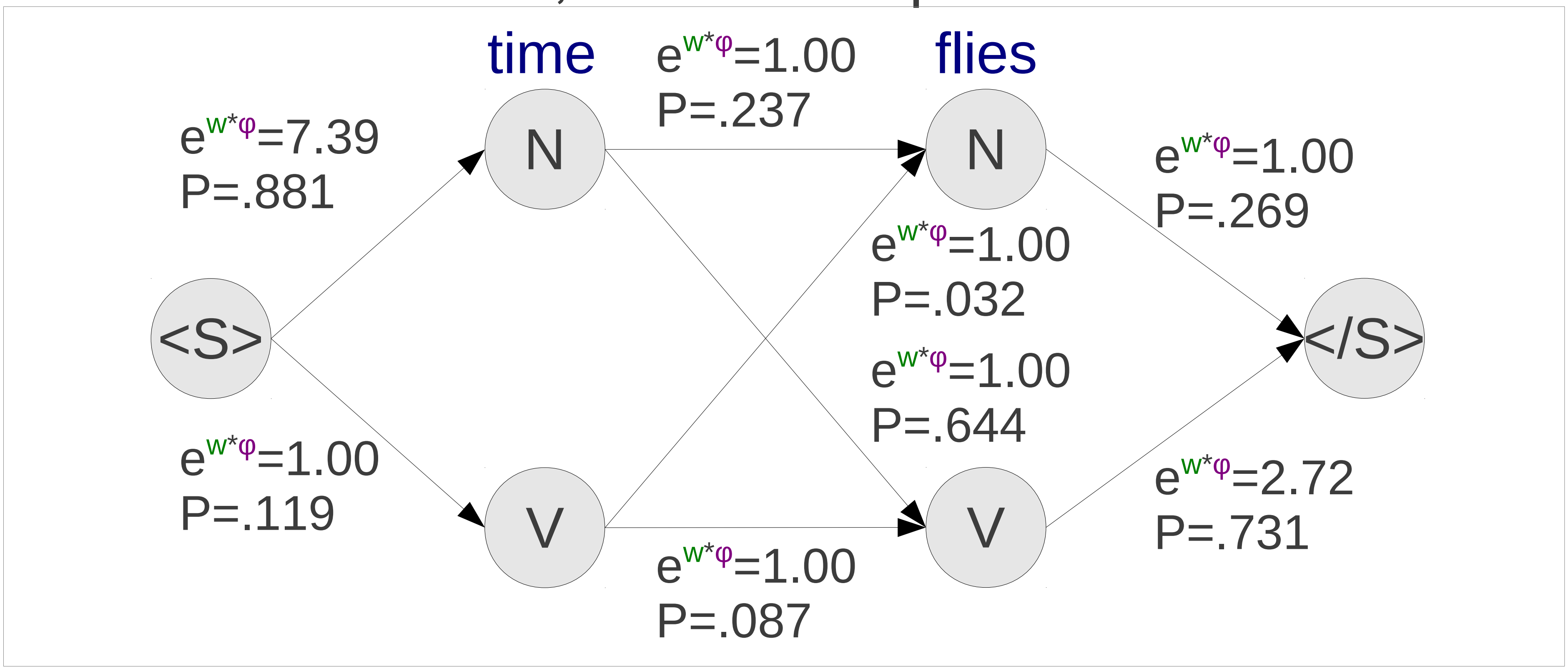
Calculating Edge Features

- Like perceptron, can calculate features for each edge



Calculating Edge Probabilities

- Calculate scores, and take exponent



- This is now the same form as the HMM
  - Can use the Viterbi algorithm
  - Calculate probabilities using forward-backward

Conditional Random Fields

Maximizing CRF Likelihood

- Want to maximize the likelihood for sequences

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \prod_i P(Y_i | X_i ; \mathbf{w})$$

$$P(Y|X) = \frac{e^{\mathbf{w} \cdot \varphi(Y,X)}}{\sum_{\tilde{Y}} e^{\mathbf{w} \cdot \varphi(\tilde{Y},X)}}$$

- For convenience, we consider the log likelihood

$$\log P(Y|X) = \mathbf{w} \cdot \varphi(Y,X) - \log \sum_{\tilde{Y}} e^{\mathbf{w} \cdot \varphi(\tilde{Y},X)}$$

- Want to find gradient for stochastic gradient descent

$$\frac{d}{d\,\mathbf{w}} \log P(Y|X)$$

Deriving a CRF Gradient:

$$\begin{aligned}\log P(Y|X) &= \mathbf{w}\cdot\boldsymbol{\varphi}(Y,X) - \log\sum_{\tilde{Y}} e^{\mathbf{w}\cdot\boldsymbol{\varphi}(\tilde{Y},X)} \\ &= \mathbf{w}\cdot\boldsymbol{\varphi}(Y,X) - \log Z\end{aligned}$$

$$\begin{aligned}\frac{d}{d\mathbf{w}}\log P(Y|X) &= \boldsymbol{\varphi}(Y,X) - \frac{d}{d\mathbf{w}}\log\sum_{\tilde{Y}} e^{\mathbf{w}\cdot\boldsymbol{\varphi}(\tilde{Y},X)} \\ &= \boldsymbol{\varphi}(Y,X) - \frac{1}{Z}\sum_{\tilde{Y}}\frac{d}{d\mathbf{w}} e^{\mathbf{w}\cdot\boldsymbol{\varphi}(\tilde{Y},X)} \\ &= \boldsymbol{\varphi}(Y,X) - \sum_{\tilde{Y}}\frac{e^{\mathbf{w}\cdot\boldsymbol{\varphi}(\tilde{Y},X)}}{Z}\boldsymbol{\varphi}(\tilde{Y},X) \\ &= \boldsymbol{\varphi}(Y,X) - \sum_{\tilde{Y}} P(\tilde{Y}|X)\,\boldsymbol{\varphi}(\tilde{Y},X)\end{aligned}$$

In Other Words...

- To get the gradient we:

$$\frac{d}{d\,\boldsymbol{w}} \log P(Y \mid X) = \boldsymbol{\varphi}(Y, X) - \sum_{\tilde{Y}} P(\tilde{Y} \mid X)\,\boldsymbol{\varphi}(\tilde{Y}, X)$$

add the correct feature vector

subtract the expectation of the features

Example

$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ N & V\end{pmatrix}$ $\varphi_{T,<S>,N}=1$ $\varphi_{T,N,V}=1$ $\varphi_{T,V,</S>}=1$ $\varphi_{E,N,time}=1$ $\varphi_{E,V,flies}=1$ P=.644

$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ V & N\end{pmatrix}$ $\varphi_{T,<S>,V}=1$ $\varphi_{T,V,N}=1$ $\varphi_{T,N,</S>}=1$ $\varphi_{E,V,time}=1$ $\varphi_{E,N,flies}=1$ P=.032

$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ N & N\end{pmatrix}$ $\varphi_{T,<S>,N}=1$ $\varphi_{T,N,N}=1$ $\varphi_{T,N,</S>}=1$ $\varphi_{E,N,time}=1$ $\varphi_{E,N,flies}=1$ P=.237

$\varphi\begin{pmatrix}\text{time} & \text{flies}\\ V & V\end{pmatrix}$ $\varphi_{T,<S>,V}=1$ $\varphi_{T,V,V}=1$ $\varphi_{T,V,</S>}=1$ $\varphi_{E,V,time}=1$ $\varphi_{E,V,flies}=1$ P=.087

$\varphi_{T,<S>,N}$, $\varphi_{E,N,time}$ = 1-.644-.237 = .119

$\varphi_{T,<S>,V}$, $\varphi_{E,V,time}$ = 0-.032-.087 = -.119

$\varphi_{T,V,</S>}$, $\varphi_{E,V,flies}$ = 1-.644-.087 = .269

$\varphi_{T,N,</S>}$, $\varphi_{E,V,flies}$ = 0-.032-.237 = -.269

$\varphi_{T,N,V}$ = 1-.644 = .356

$\varphi_{T,V,N}$ = 0-.032 = -.032

$\varphi_{T,N,N}$ = 0-.237 = -.237

$\varphi_{T,V,V}$ = 0-.087 = -.087

Combinatorial Explosion

- Problem!: The number of hypotheses is exponential.

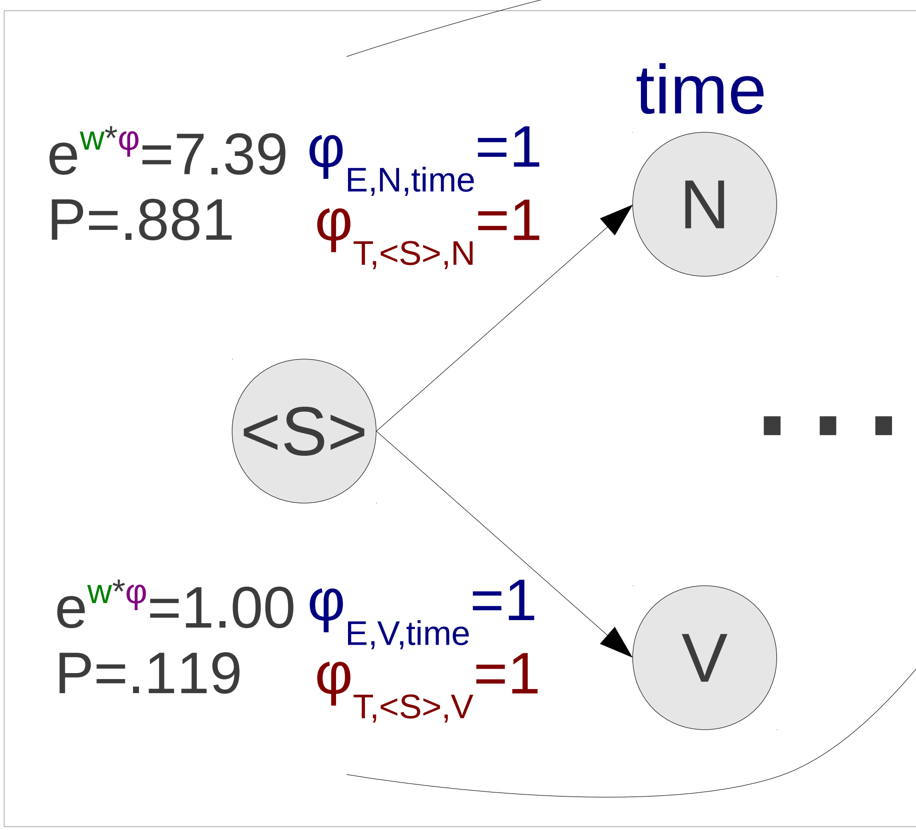
$$\frac{d}{d\boldsymbol{w}} \log P(Y|X) = \boldsymbol{\varphi}(Y,X) - \sum_{\tilde{Y}} P(\tilde{Y}|X)\,\boldsymbol{\varphi}(\tilde{Y},X)$$

$O(T^{|X|})$

T = number of tags

Calculate Feature Expectations using Edge Probabilities!

- If we know the edge probabilities, just multiply them!



$\varphi_{T,<S>,N}, \varphi_{E,N,time} = 1-.881 = .119$

$\varphi_{T,<S>,V}, \varphi_{E,V,time} = 0-.119 = -.119$

Same answer as when we explicitly expand all Y!

$\varphi_{T,<S>,N}, \varphi_{E,N,time} = 1-.644-.237 = .119$

$\varphi_{T,<S>,V}, \varphi_{E,V,time} = 0-.032-.087 = -.119$

CRF Training Procedure

- Can perform stochastic gradient descent, like logistic regression

```
create map w
for I iterations
    for each labeled pair X, Y in the data
        gradient = φ(Y,X)
        calculate e^(φ(y,x)*w) for each edge
        run forward-backward algorithm to get P(edge)
        for each edge
            gradient -= P(edge)*φ(edge)
        w += α * gradient
```

- Only major difference is gradient calculation
- Learning rate α

Learning Algorithms

Batch Learning

- Online Learning: Update after each example

Online Stochastic Gradient Descent

create map *w*
for *I* iterations
    for each labeled pair *x*, *y* in the data
        w += α * dP(y|x)/dw

- Batch Learning: Update after all examples

Batch Stochastic Gradient Descent

create map *w*
for *I* iterations
    for each labeled pair *x*, *y* in the data
        gradient += α * dP(y|x)/dw
    w += *gradient*

Batch Learning Algorithms: Newton/Quasi-Newton Methods

- Newton-Raphson Method:
  - Choose how far to update using the second-order derivatives (the Hessian matrix)
  - Faster convergence, but |**w**|*|**w**| time and memory
- Limited Memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS):
  - Guesses second-order derivatives from first-order
  - Most widely used?
  - Library: http://www.chokkan.org/software/liblbfgs/
- More information: http://homes.cs.washington.edu/~galen/files/quasi-newton-notes.pdf

Online Learning vs. Batch Learning

- Online:
  - In general, simpler mathematical derivation
  - Often converges faster
- Batch:
  - More stable (does not change based on order)
  - Trivially parallelizable

Regularization

Cannot Distinguish Between Large and Small Classifiers

- For these examples:

-1	he saw a bird in the park
+1	he saw a robbery in the park

- Which classifier is better?

Classifier 1	Classifier 2
he +3	bird -1
saw -5	robbery +1
a +0.5	
bird -1	
robbery +1	
in +5	
the -3	
park -2	

Cannot Distinguish Between Large and Small Classifiers

- For these examples:

-1	he saw a bird in the park
+1	he saw a robbery in the park

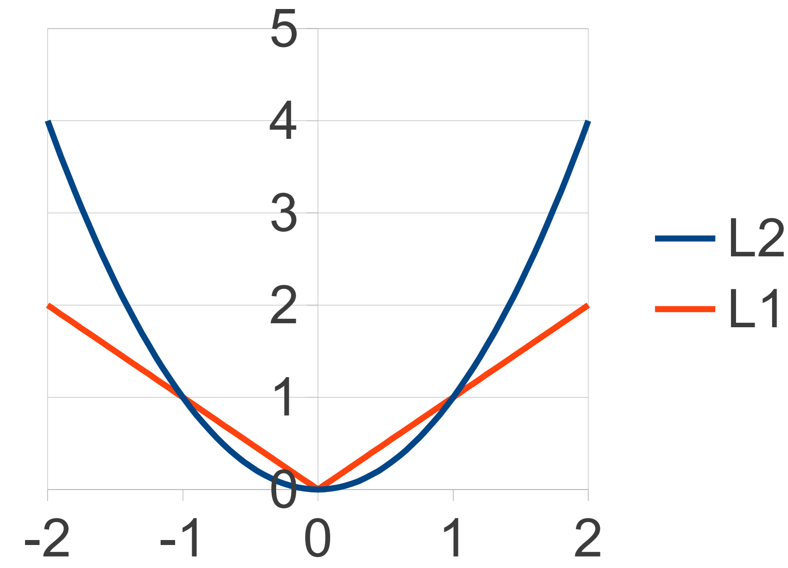
- Which classifier is better?

Classifier 1	Classifier 2
he +3	bird -1
saw -5	robbery +1
a +0.5	
bird -1	
robbery +1	
in +5	
the -3	
park -2	

Probably classifier 2!
It doesn't use
irrelevant information.

Regularization

- A penalty on adding extra weights
- L2 regularization:
  - Big penalty on large weights, small penalty on small weights
  - High accuracy
- L1 regularization:
  - Uniform increase whether large or small
  - Will cause many weights to become zero → small model



Regularization in Logistic Regression/CRF

- To do so in logistic regression/CRF, we add the penalty to the log likelihood (for the whole corpus)

L2 Regularization

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \left(\prod_i P(Y_i \mid X_i; \mathbf{w}) \right) - c \sum_{w \in \mathbf{w}} w^2$$

- c adjusts the strength of the regularization
  - smaller: more freedom to fit the data
  - larger: less freedom to fit the data, better generalization
- L1 also used, slightly more difficult to optimize

Conclusion

Conclusion

- Logistic regression is a probabilistic classifier
- Conditional random fields are probabilistic structured discriminative prediction models
- Can be trained using
  - Online stochastic gradient descent (like peceptron)
  - Batch learning using a method such as L-BFGS
- Regularization can help solve problems of overfitting

Thank You!