NAÏVE BAYES AND CONDITIONAL RANDOM FIELDS

REVIEW OF LAST TIME

Review

- Linear regression
  - Model with parameters
  - $h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5$

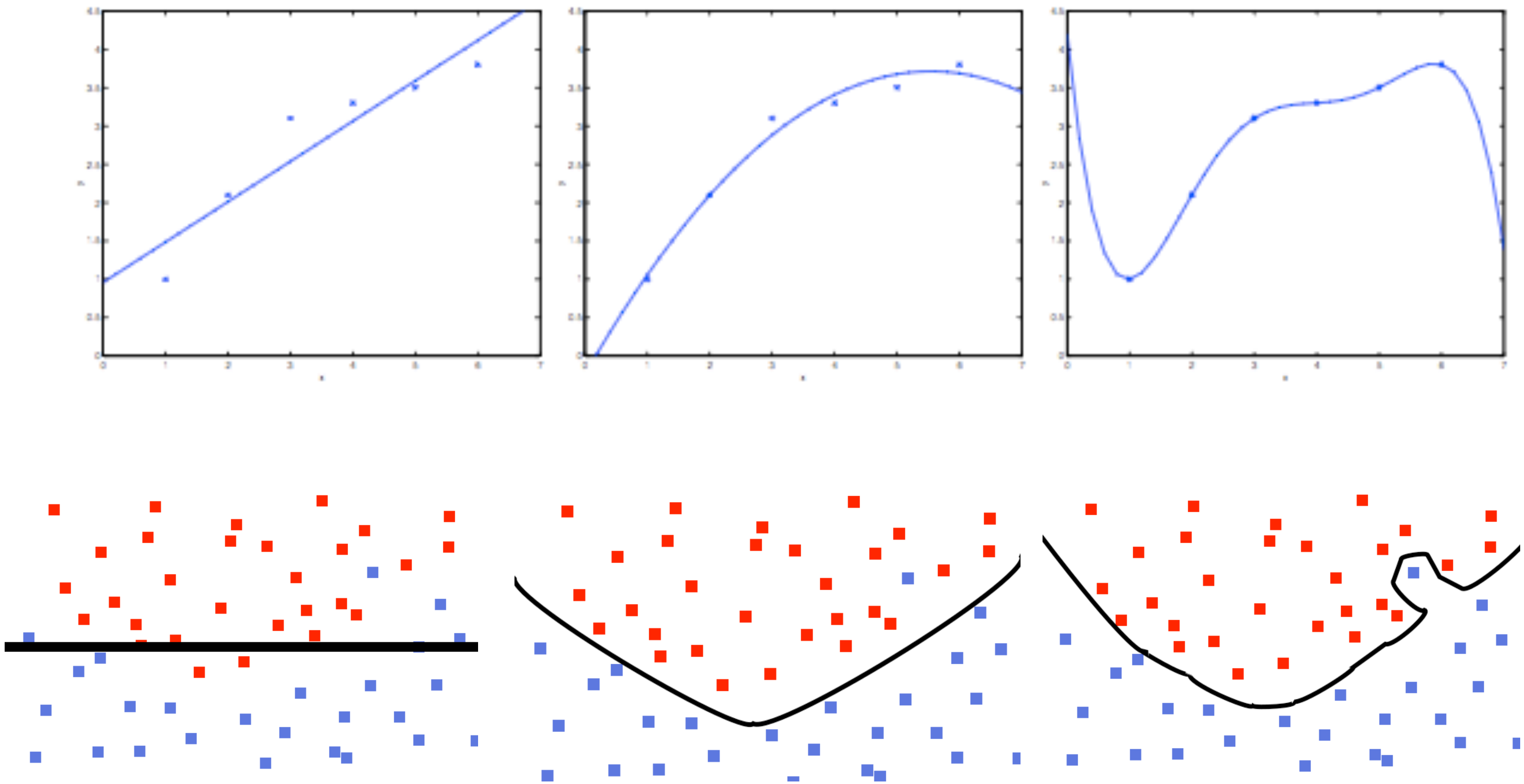
$$h_\theta(x) = \Sigma_{i=0}^{n} \theta_i x_i = \theta^T \mathbf{x}$$

  - Minimizing loss function

$$J(\theta) = \frac{1}{2} \Sigma_{i=1}^{m} (y_i - \theta^T \mathbf{x_i})^2$$

  - Gradient descent
    - Stochastic gradient descent (mini-batch)

Review – overfitting underfitting



Review

- Model selection

Low Number of parameters High

Low Model (decision boundary) complexity High

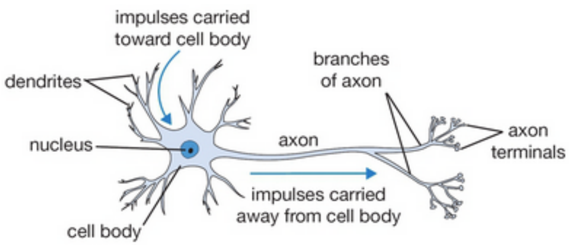
Large Training error Small

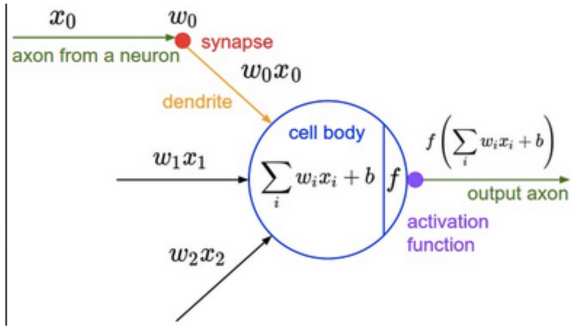
Low Amount of data required High

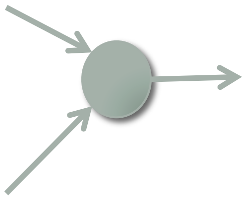
High Generalizability (if not enough data) Low

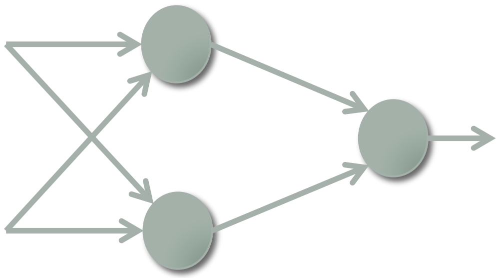
Review

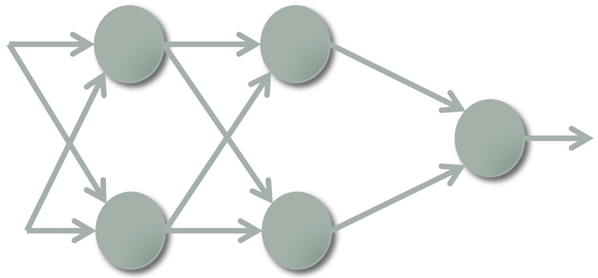
- Neural networks







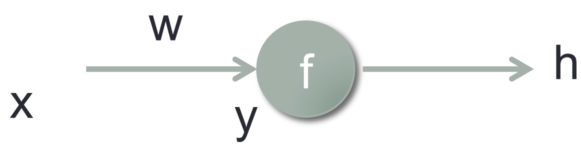




- Computation graph
- Loss function
  - Cross entropy and square loss
- Training using backpropagation (chain rule)

h = f(y)
y = xw

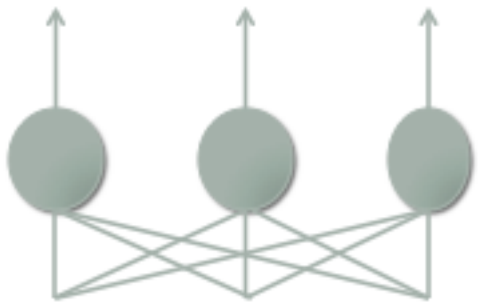
$$\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$$



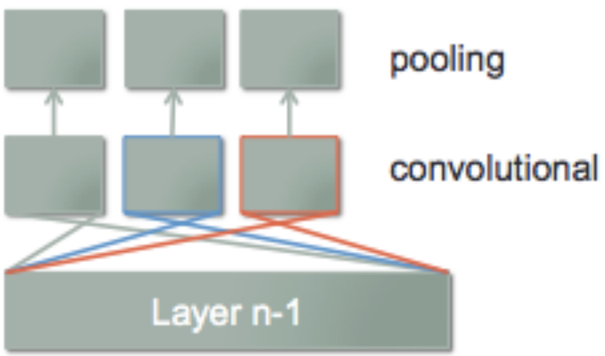
dh/dw = dh/dy * dy/dw
= f'(y) x

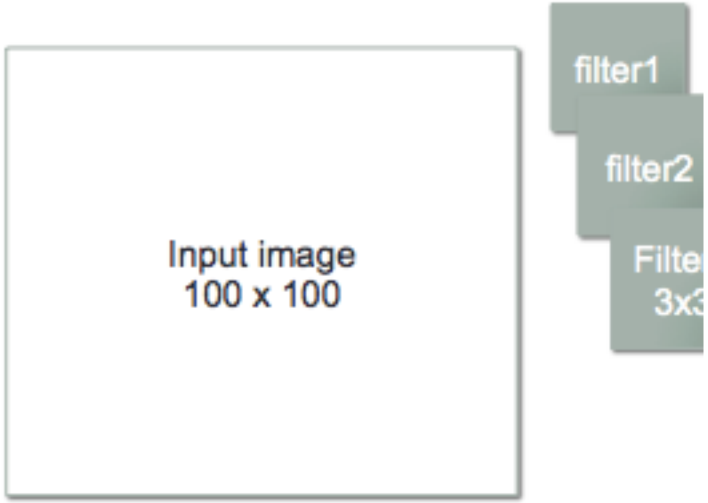
Review

- Reducing overfitting
  - Regularization: L1 L2
  - Dropout



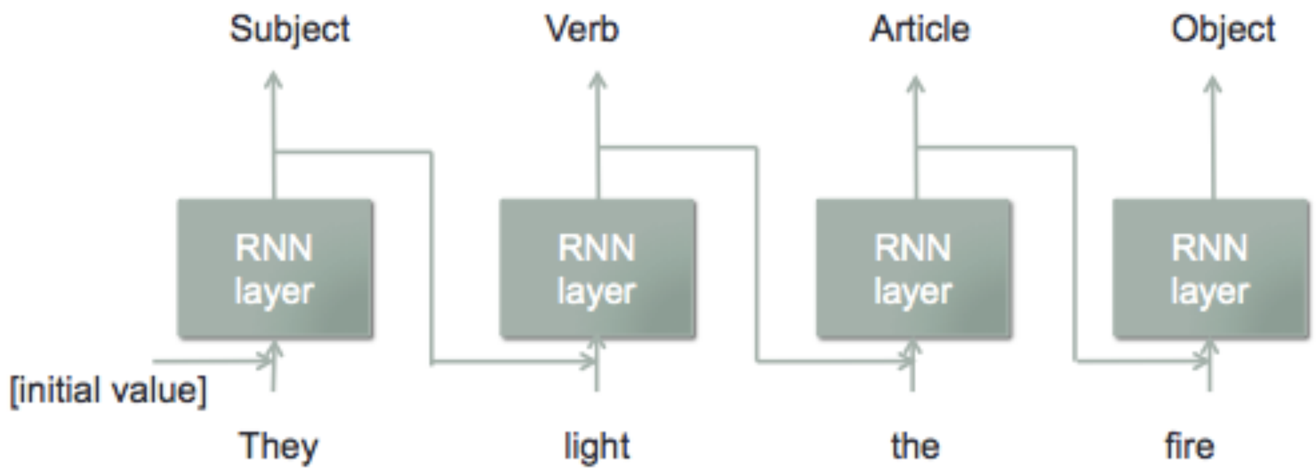
- Convolution neural networks



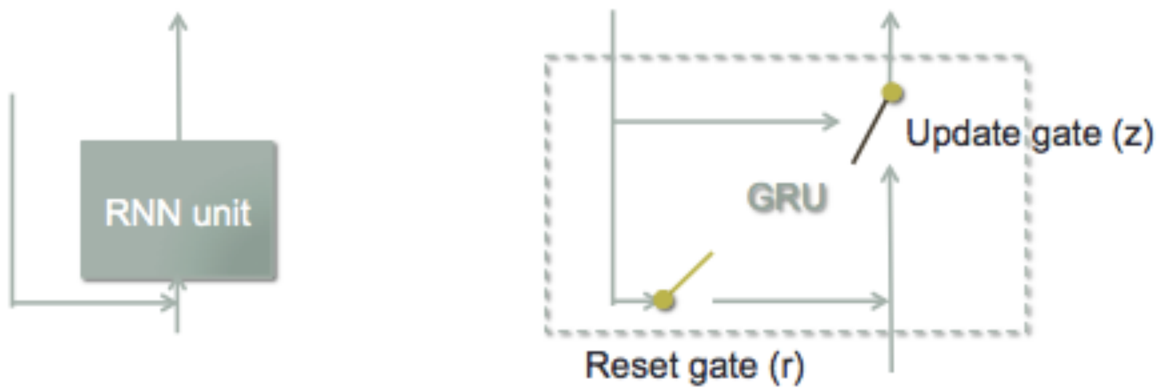


Review

- Recurrent neural networks



- GRU & LSTM

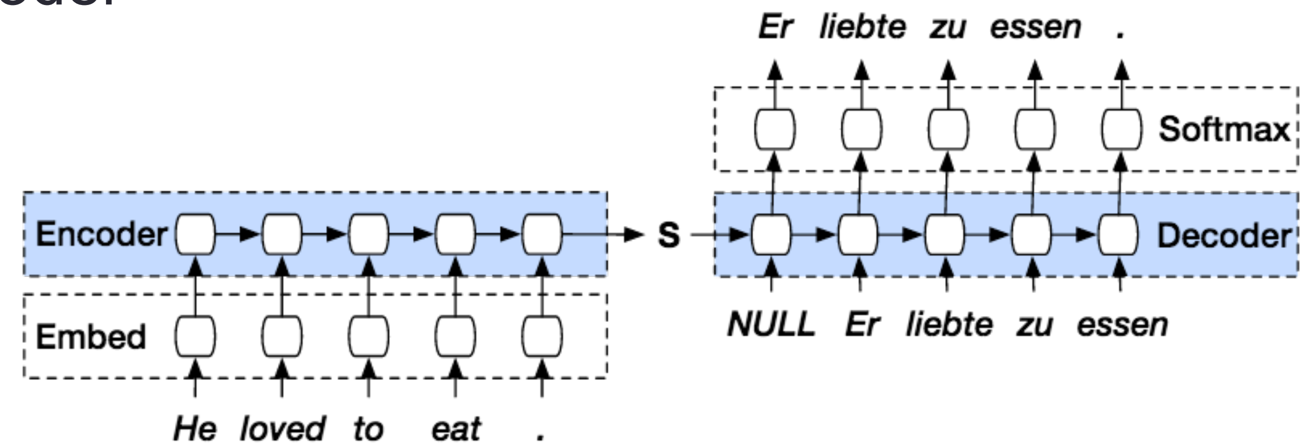


Review

- Embedding
  - Sparse to dense representation

Apple -> 1 -> [1, 0, 0, 0, …] -> [2.3, 1.2]
Bird -> 2 -> [0, 1, 0, 0, …] -> [-1.0, 2.4]
Cat -> 3 -> [0, 0, 1, 0, …] -> [-3.0, 4.0]

  - Word2vec, char2vec, sentence2vec
- Encoder-decoder



Homework

- Feature selection/engineering
  - Non-linear features

- Normalization
  - bug

PROBABILISTIC MODELS

Probabilistic models

- Naïve Bayes (NBs)
- Conditional Random Fields (CRFs)

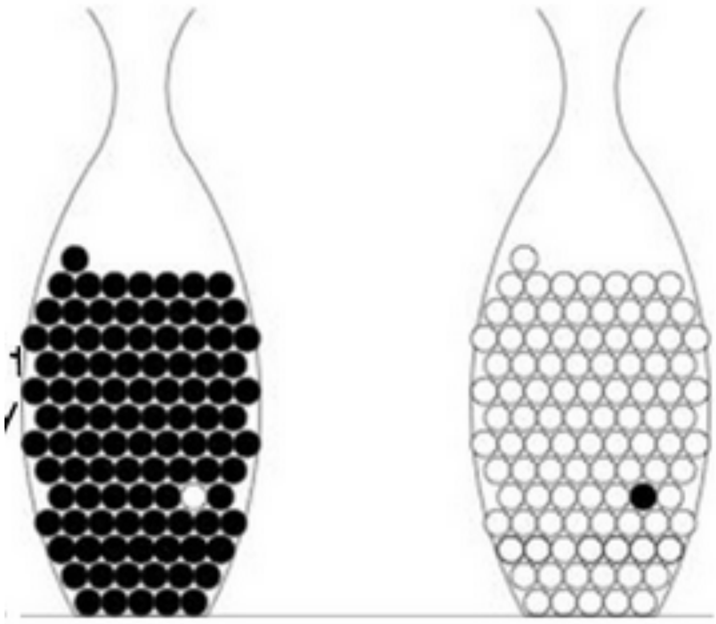
NAÏVE BAYES

Bayes Rule & Learning theory

- x – observed data
- w_i – probability that x comes from class i

$$p(w_i|x) = \frac{p(x|w_i)p(w_i)}{p(x)}$$

$$\text{Posterior} = \frac{\text{likelihood} * \text{prior}}{\text{evidence}}$$

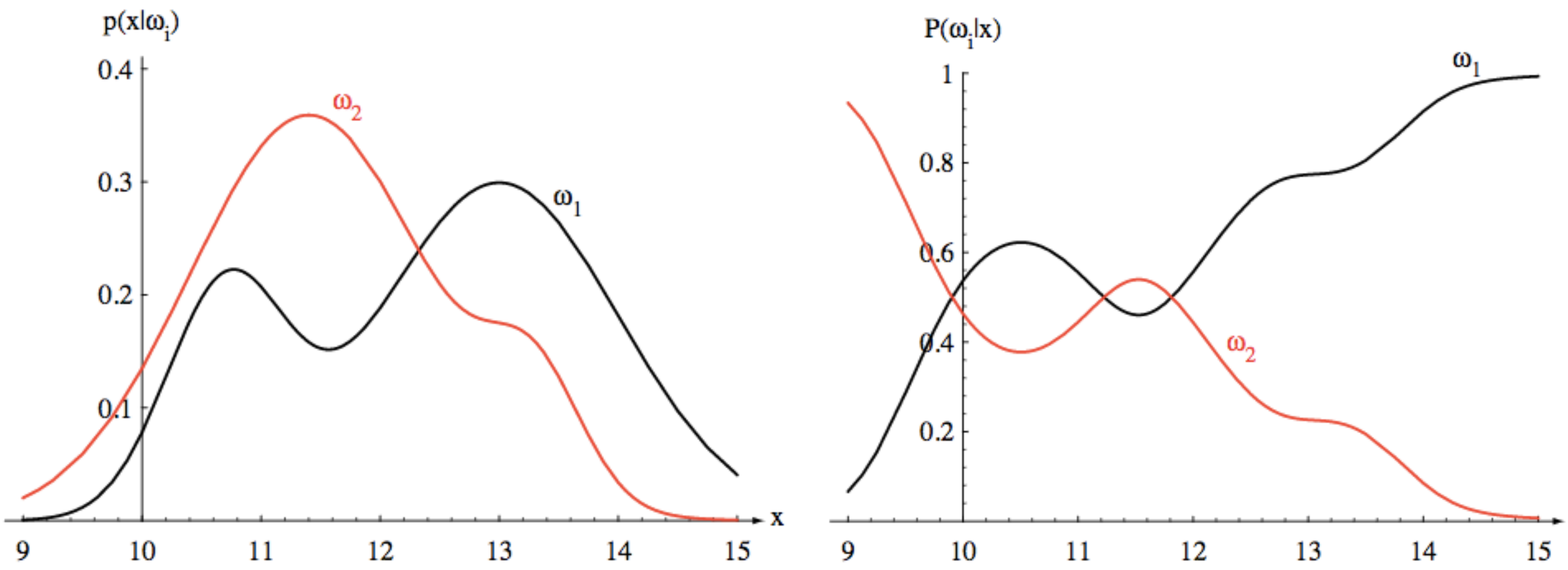


http://slideplayer.com/slide/8845876/

This relationship between the likelihood and the posterior is the basis of many probabilistic models

A simple decision rule

- If we can know either p(x|w) or p(w|x) we can make a classification guess. (how?)



Goal: Find p(x|w) or p(w|x)

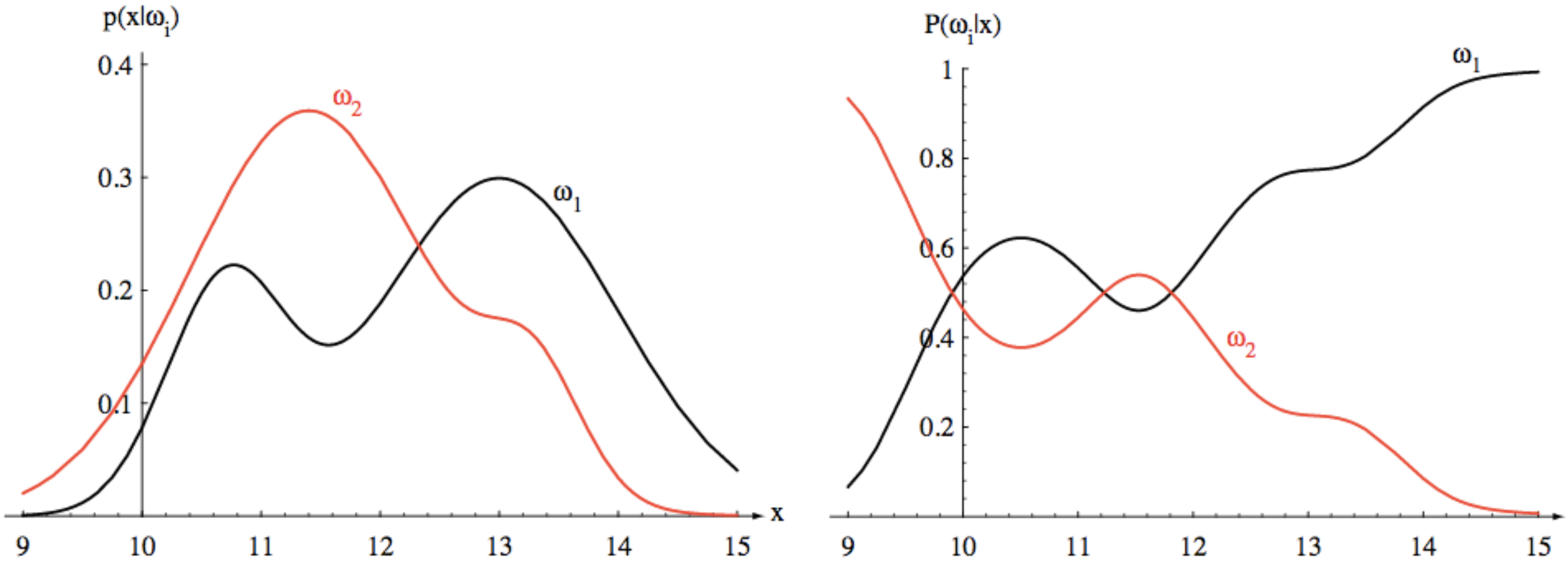
$$p(w_i|x) = \frac{p(x|w_i)p(w_i)}{p(x)}$$

Duda et al. Pattern Classification

A simple decision rule

$$p(w_i|x) = \frac{p(x|w_i)p(w_i)}{p(x)}$$

- If P(w1|x) > P(w2|x), this is class 1.
- Equivalently, If P(x|w1)P(w1)/P(x) > P(w2|x)P(w2)/P(x)

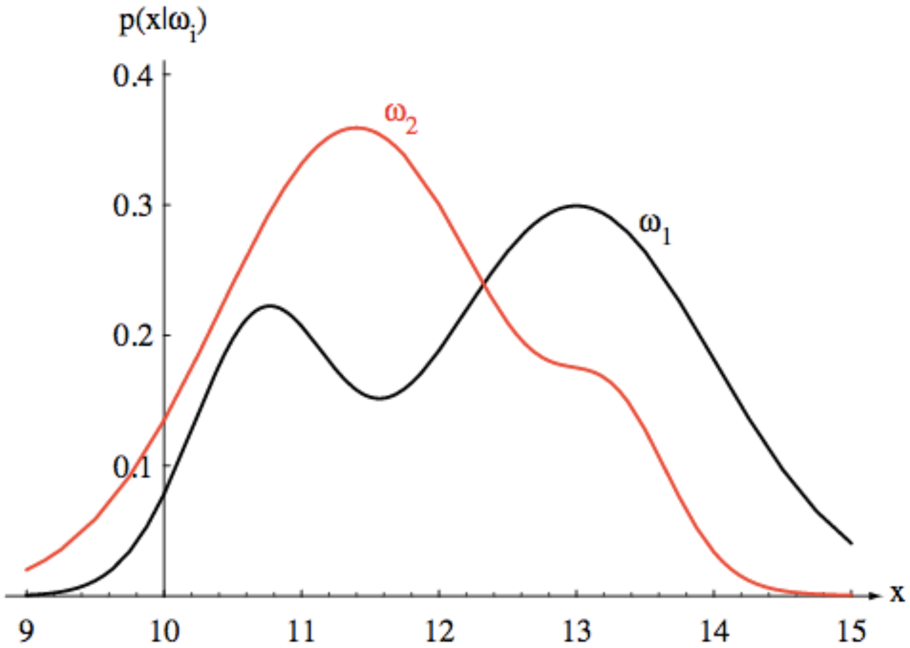


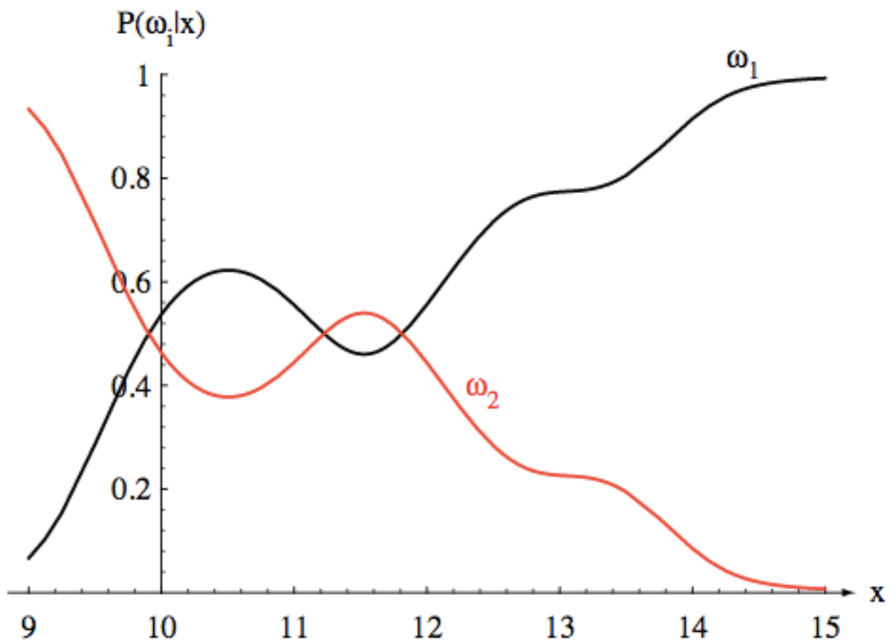
Duda et al. Pattern Classification

A simple decision rule

$$p(w_i|x) = \frac{p(x|w_i)p(w_i)}{p(x)}$$

- If P(w1|x) > P(w2|x), this is class 1.
- Equivalently, If P(x|w1)P(w1) > P(w2|x)P(w2)





Goal: Find p(x|w) or p(w|x) – Easier to find p(x|w)

Duda et al. Pattern Classification

P(class|x) vs P(x|class)

- Predicting rain in the afternoon given sunny or cloudy in the morning

	Rain	No rain
X = cloudy		
X = sunny		

Rain – cloudy
Rain – cloudy
Rain – sunny
No rain – sunny
No rain – sunny
No rain – cloudy
No rain – sunny

P(cloudy| Rain) vs P(Rain | cloudy)

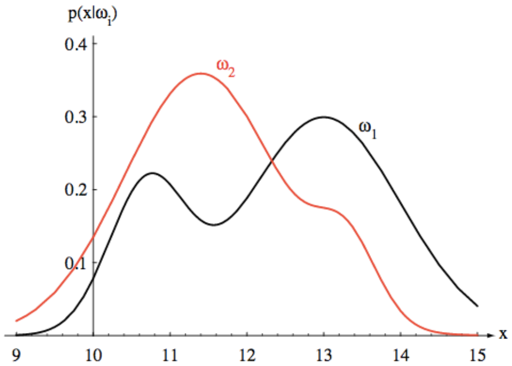
Use counting and ratios

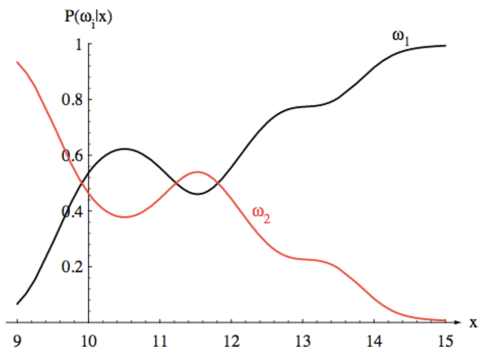
P(class|x) vs P(x|class) – continuous case

- Predicting rain in the afternoon given humidity in the morning

	Rain	No rain
X = value		
X = value		

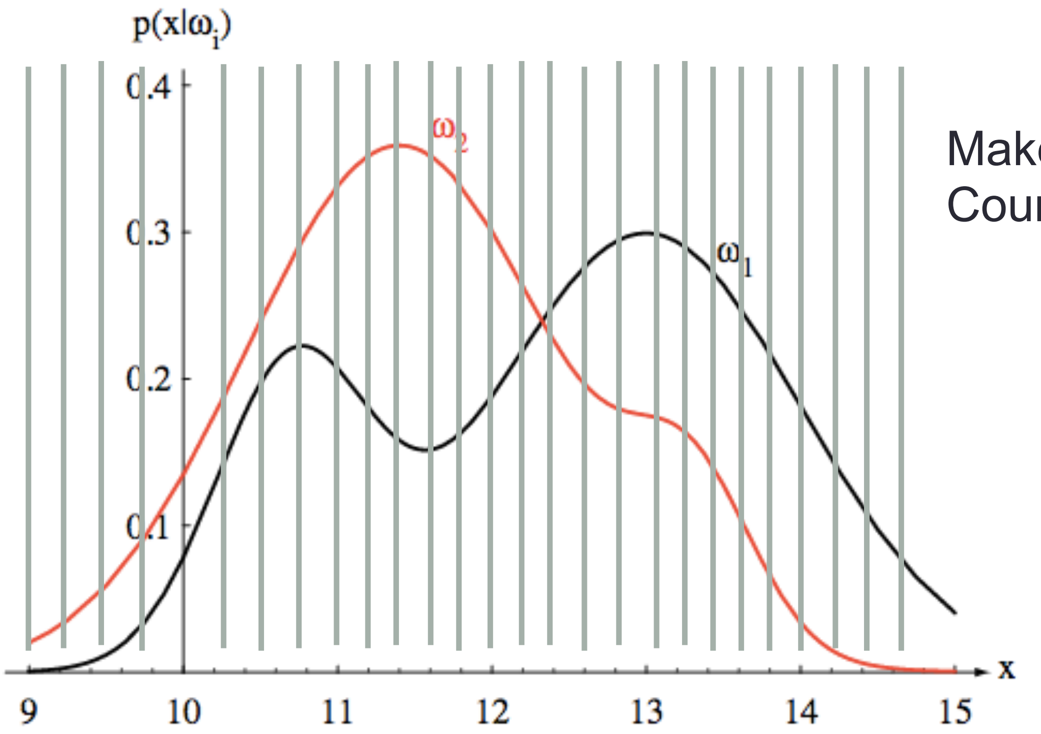
Rain – 68
Rain – 90
Rain – 30
No rain – 30
No rain – 22
No rain – 11
No rain – 25





How to estimate?

Method one, histogram



Make a histogram!
Count how many falls in the bin

Rain – 68
Rain – 90
Rain – 30
No rain – 30
No rain – 22
No rain – 11
No rain – 25

What happens if there is no data in a bin?

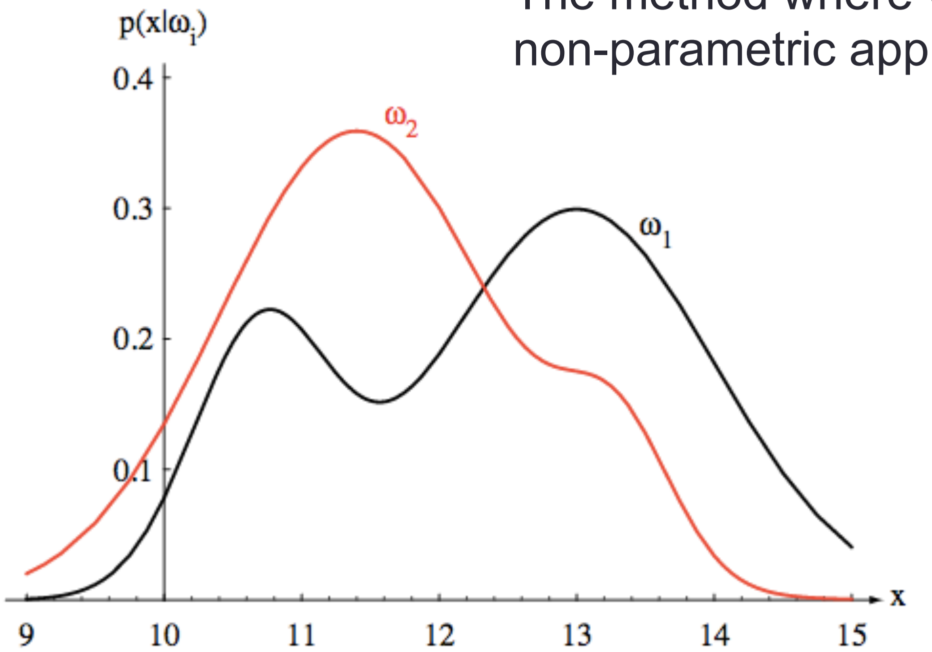
Method two, fit a distribution

- Figure out what distribution describes the data the best

The parametric approach

- We assume p(x|w) or p(w|x) follow some distributions with parameter θ

The method where we find the histogram is the non-parametric approach



Example: Gaussian distribution

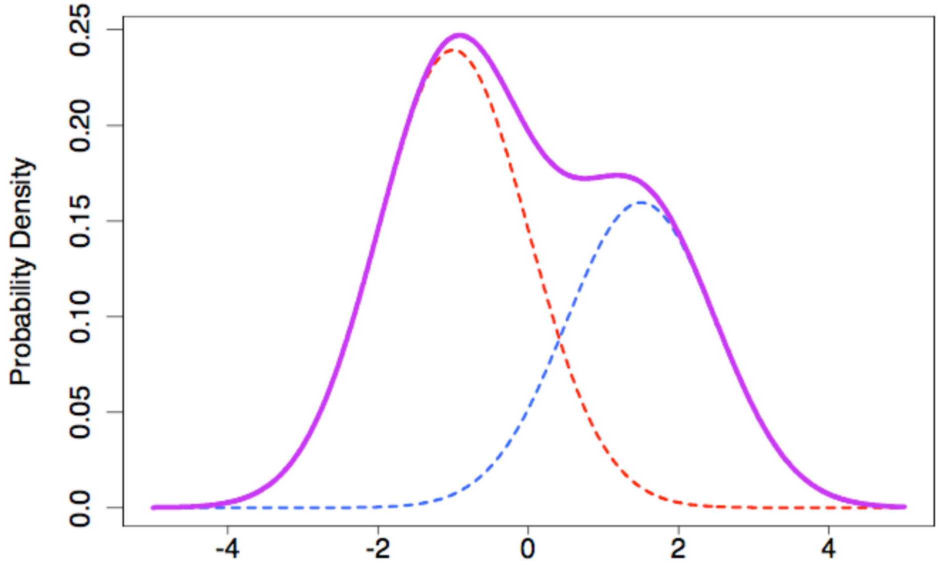
$$\boxed{p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}}$$

Goal: Find θ so that we can estimate p(x|w) or p(w|x)

Gaussian Mixture Models (GMMs)

- Gaussians cannot handle multi-modal data well
- Consider a class can be further divided into additional factors
- Mixing weight makes sure the overall probability sums to 1

$$P(x) \sim \sum_{k=1}^{K} w_k N(\mu_k, \sigma_k)$$

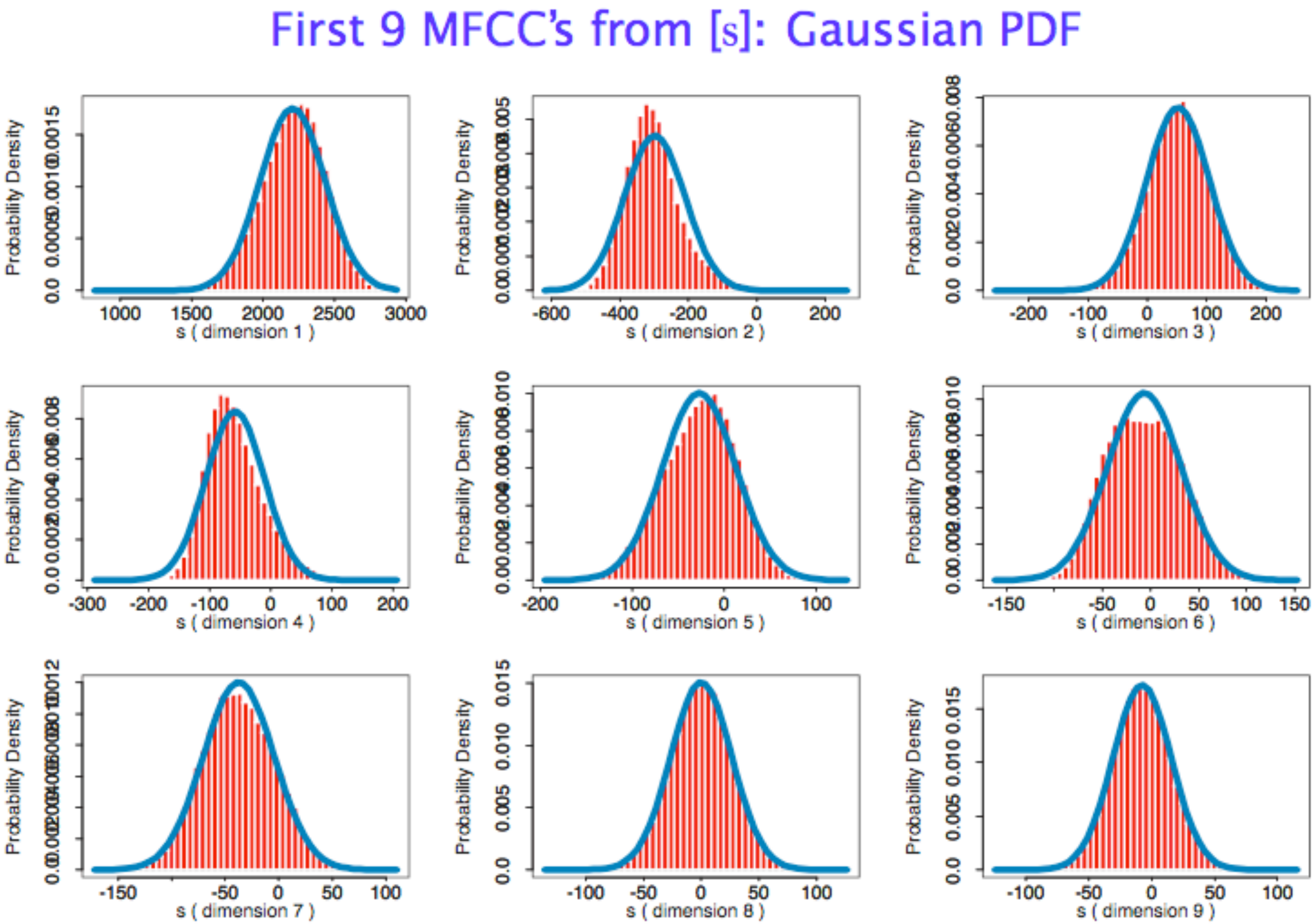


$\frac{x}{\sigma}$

$p(x) = 0.6p_1(x) + 0.4p_2(x)$

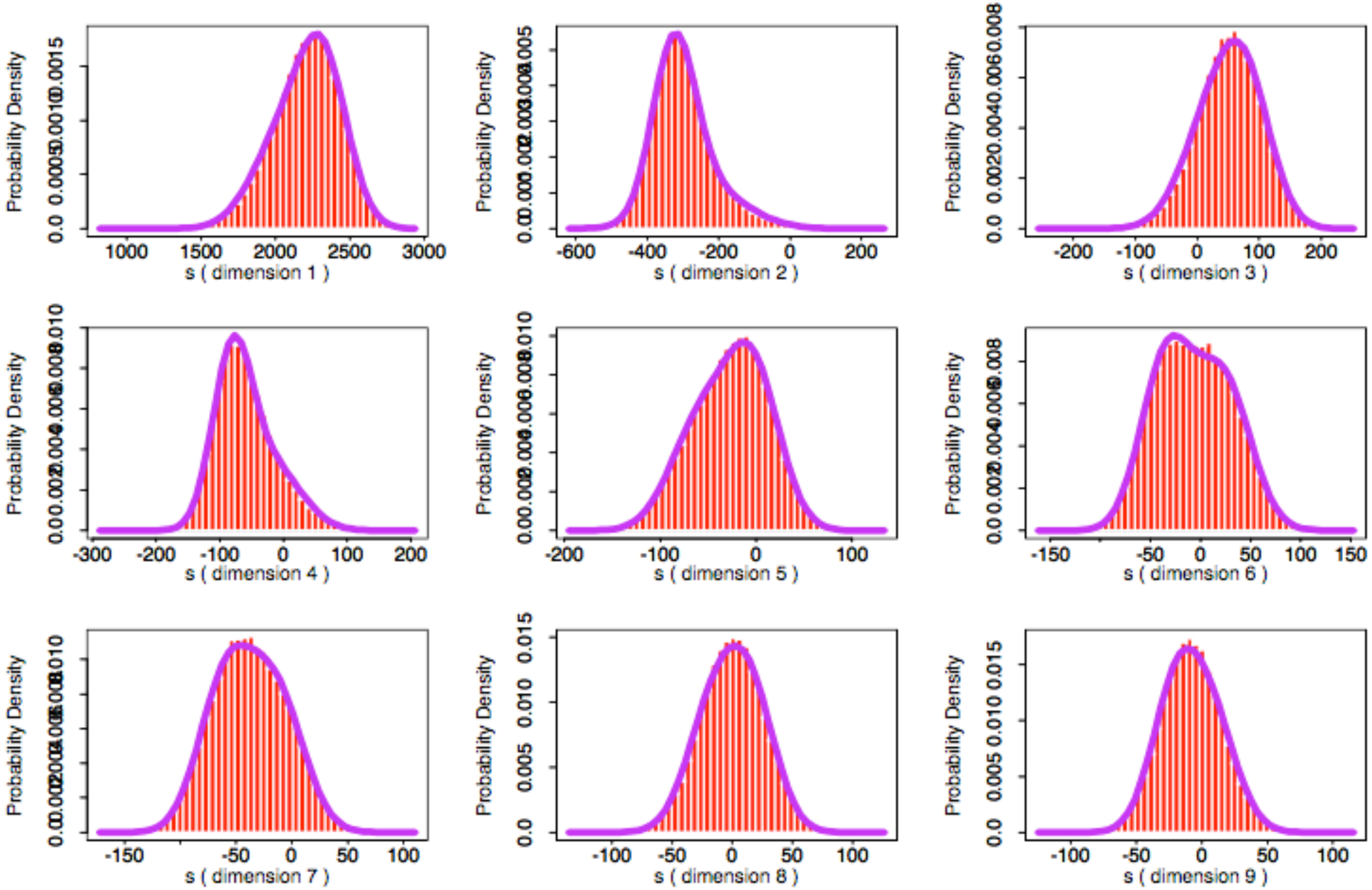
$p_1(x) \sim N(-\sigma, \sigma^2)$ $\quad$ $p_2(x) \sim N(1.5\sigma, \sigma^2)$

Model of one Gaussian



Mixture of two Gaussians

[s]: 2 Gaussian Mixture Components/Dimension



A simple decision rule

$$p(w_i|x) = \frac{p(x|w_i)p(w_i)}{p(x)}$$

- If P(w1|x) > P(w2|x), this is class 1.
- Equivalently, If P(x|w1)P(w1) > P(w2|x)P(w2), this is class 1.

- Usually easier to find P(x|w1), if we fit a distribution
- P(w1), P(w2)?

- This classifier is sometimes known as the Bayes classifier

Naïve Bayes

- Below is the LRT or the Bayes classifier

$$P(x|w_1)P(w_1) \quad ? \quad P(x|w_2)P(w_2)$$

- What about Naïve Bayes?

- Here x is a vector with m features $[x_1, x_2, \ldots x_m]$
- $P(x|w_i)$ is m+1 dimensional
  - Sometimes to hard to model, not enough data, overfit, *curse of dimensionality*, etc.
- Assumes $x_1, x_2, \ldots x_m$ independent given w_i (conditional independence)

Naïve Bayes

- $P(\mathbf{x}|w_i)P(w_i) = P(w_i) \prod_j P(x_j|w_i)$

- This assumption simplifies the calculation
- Note that we do not say anything about what kind of distribution $P(x_j|w_i)$ is.

Naïve Bayes with the log

- $P(\mathbf{x}|w_i)P(w_i) = P(w_i) \prod P(x_j|w_i)$
- Usually give small values
  - Leads to underflow

- Take the log

$P(w_1) \prod P(x_j|w_1)$? $P(w_2) \prod P(x_j|w_2)$

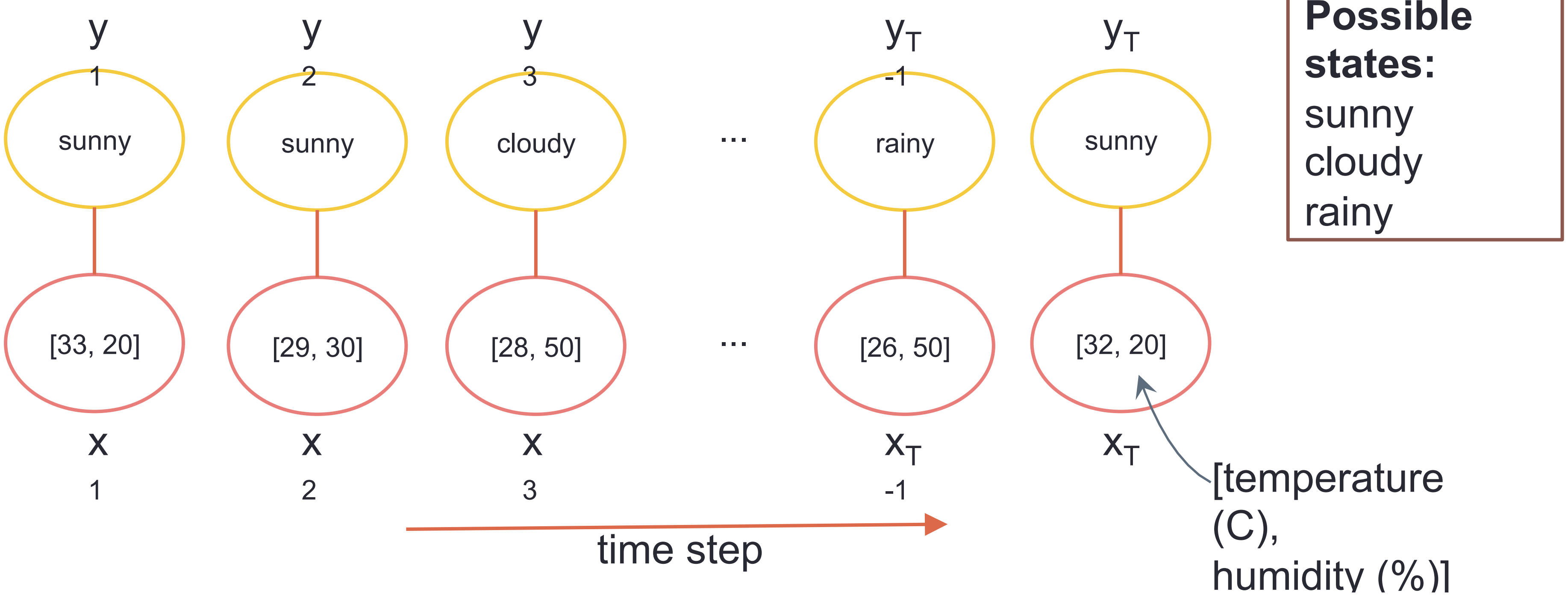
$\log P(w_1) + \sum \log P(x_j|w_1)$? $\log P(w_2) + \sum \log P(x_j|w_2)$

CONDITIONAL RANDOM FIELDS (CRFS)

A different way to decompose $P(\mathbf{x}|w_i)$

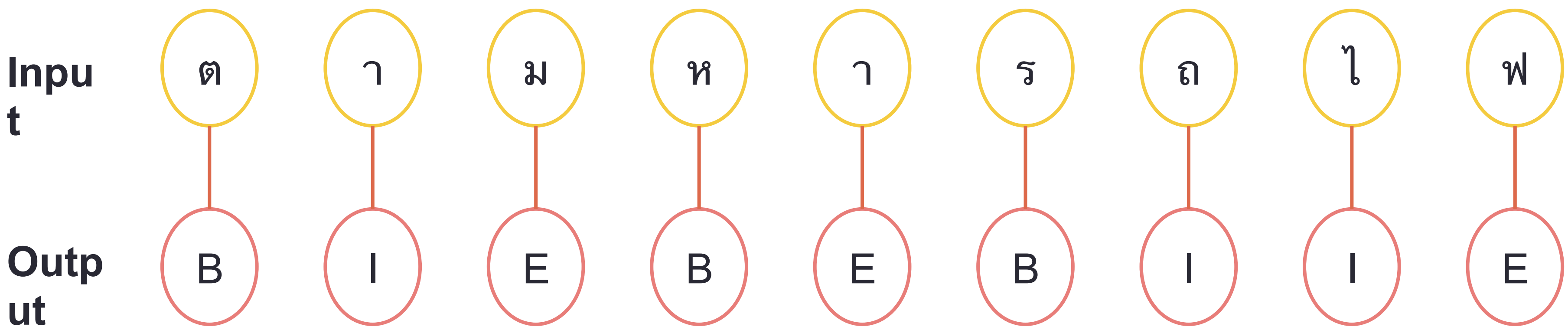
Motivation: Sequence Labeling

- Input is a sequence of data **X** = $x_1, x_2, \ldots, x_T$
- Output is a sequence of **categorical labels (or states) Y** = $y_1, y_2, \ldots, y_T$ corresponding to each input $x_1, x_2, \ldots, x_T$ respectively.
- Each y_i is a member of a given set of states/labels



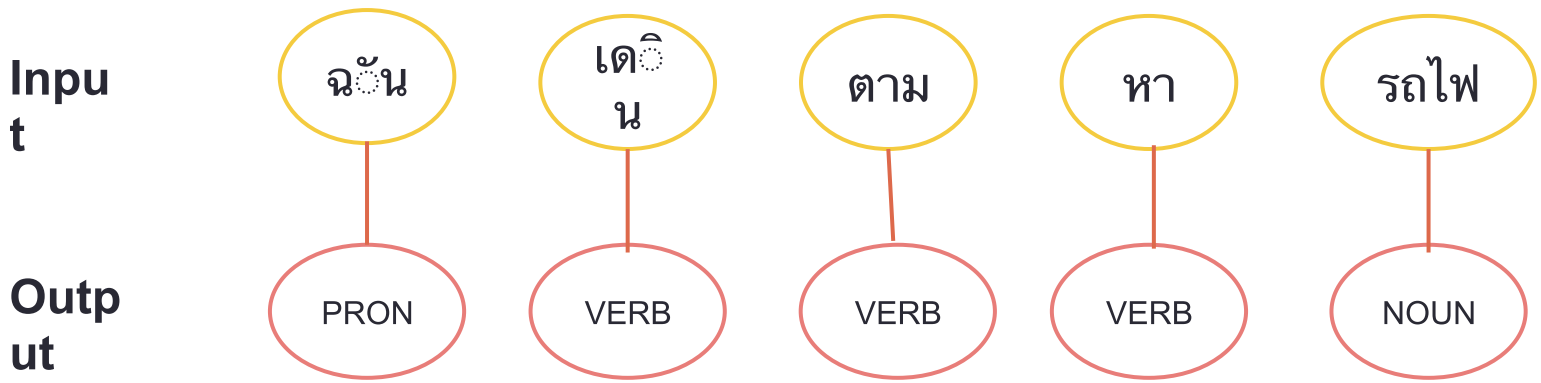
Sequence Labeling in NLP

Word segmentation
Label a character with one of character tags
(B - Beginning, I - Intermediate , E - Ending, S - Single)



Sequence Labeling in NLP

Part-of-speech tagging



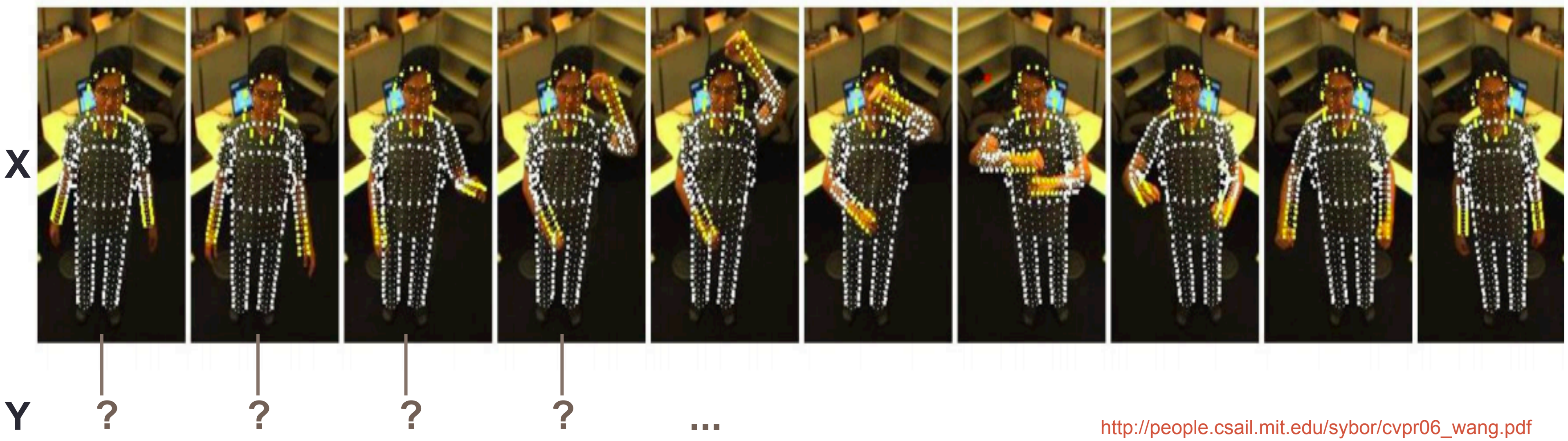
Sequence Labeling in non-NLP task

Gesture recognition

possible gestures

Flip back | Shrink Vertically | Expand Vertically | Double Back | Point & Back | Expand Horizontally

sequence of photos



http://people.csail.mit.edu/sybor/cvpr06_wang.pdf

Sequence labeling

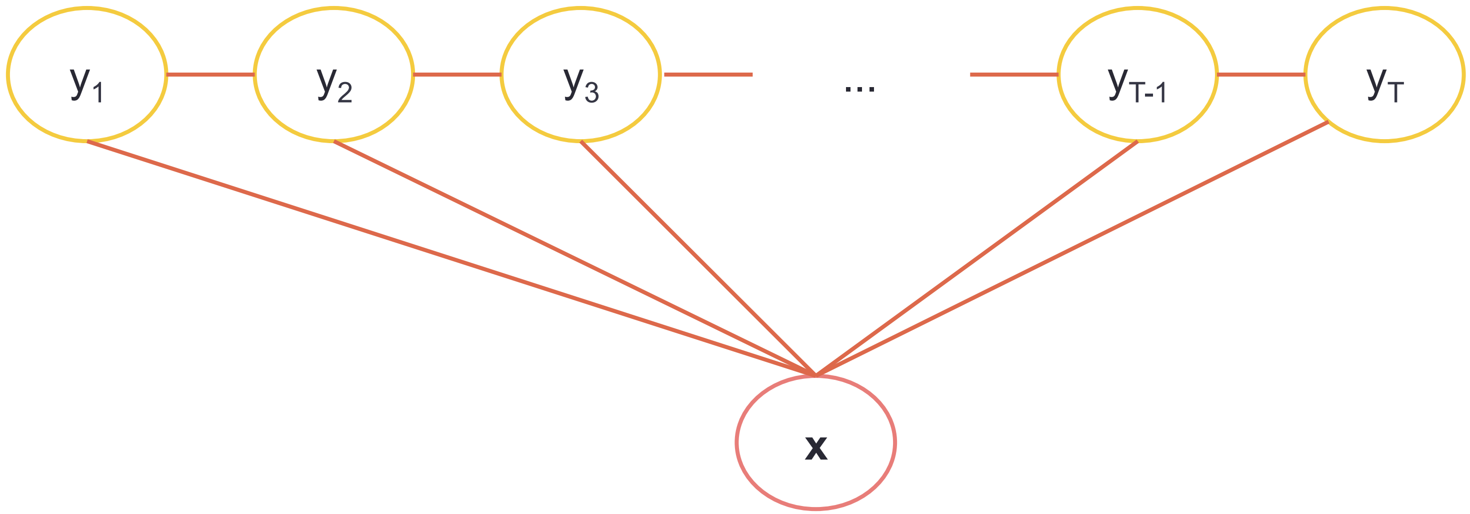
- Input x_{it} (time and feature number)
- Output y_t
- Goal, find P(**Y**|**X**) and compare.
- Hard to find P(**Y**|**X**).
- Need a way to decompose (just like how we decompose using the independence assumption in Naïve Bayes)

Linear chain CRF

Given a sequence of input $\mathbf{x} = x_1, x_2, ..., x_T$ and a sequence of output labe, $\mathbf{y} = y_1, y_2, ..., y_T$, linear-chain CRF models $p(\mathbf{y}|\mathbf{x})$ with these independency assumption:

(1) each label y_i only depends on previous label y_{i-1}

(2) each label y_i globally depends on **x**



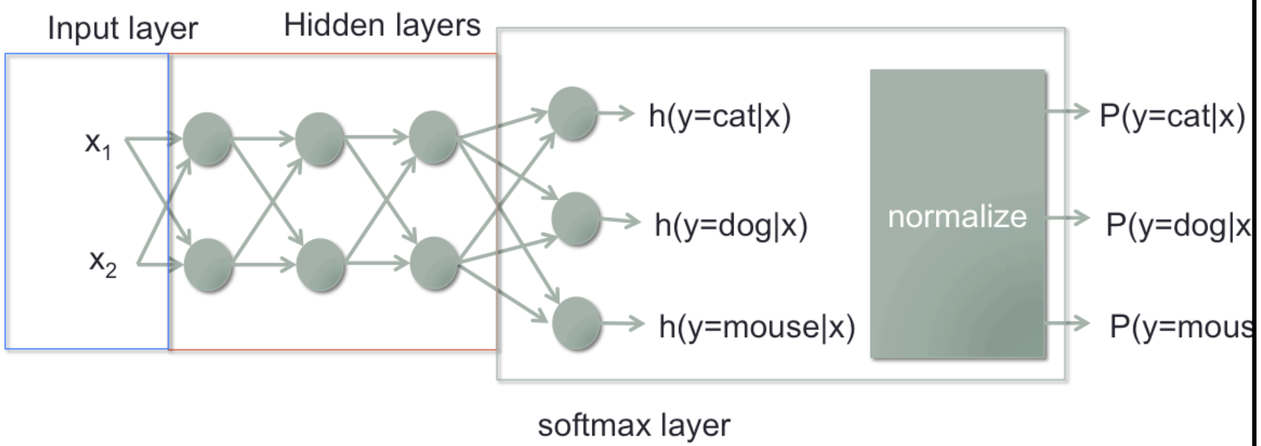
$$\mathbf{x} = x_1, x_2, x_3, ..., x_T$$

Goal

- Find a function that will represent “probabilities”

- We can turn functions into probabilities easily.
  - Softmax function normalization
  - We just need to have function that give higher to more likely inputs

$$P(y=j|x)=\frac{e^{h(y=j|x)}}{\Sigma_y e^{h(y|x)}}$$



Goal

- Find a function that will represent “probabilities”

- We can turn functions into probabilities easily.
  - Softmax function normalization
  - We just need to have function that give higher to more likely inputs

- Building functions that represent the whole sequence is hard
  - We’ll build by combining pieces
  - Let’s look at the pieces
  - But each piece should have the form $f(y_t, y_{t-1}, \mathbf{x}, t) \to \mathbb{R}$
    - This is from our independence assumption.
  - We call these functions, feature functions

Feature function

At each time step, a feature function $f(y_t, y_{t-1}, \mathbf{x}, t) \rightarrow \mathbb{R}$ is used to capture some characteristics of current label and the observation.

A feature function in linear-CRF:

returns a real value

$$f(y_t, y_{t-1}, \mathbf{x}, t) \rightarrow \mathbb{R}$$

current label

previous label

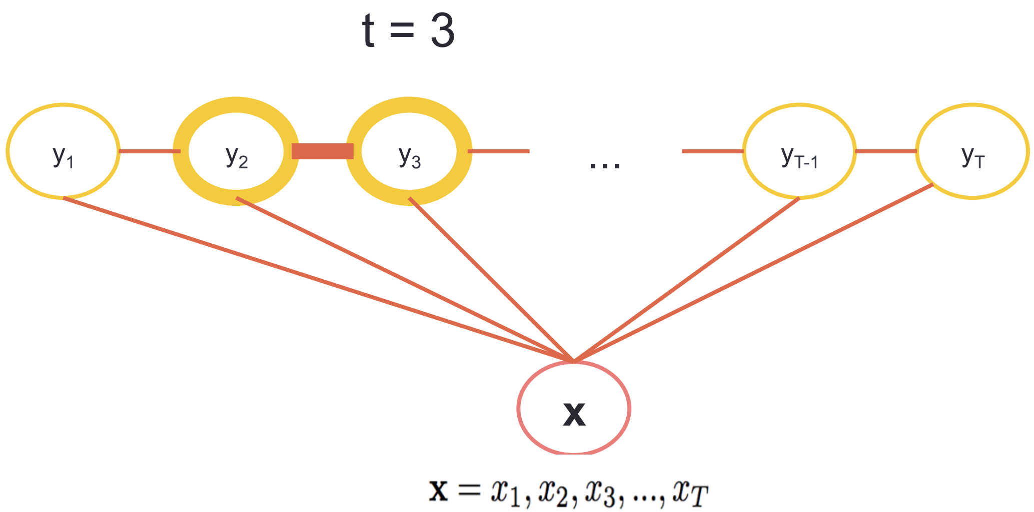
input sequence

current index

Example features

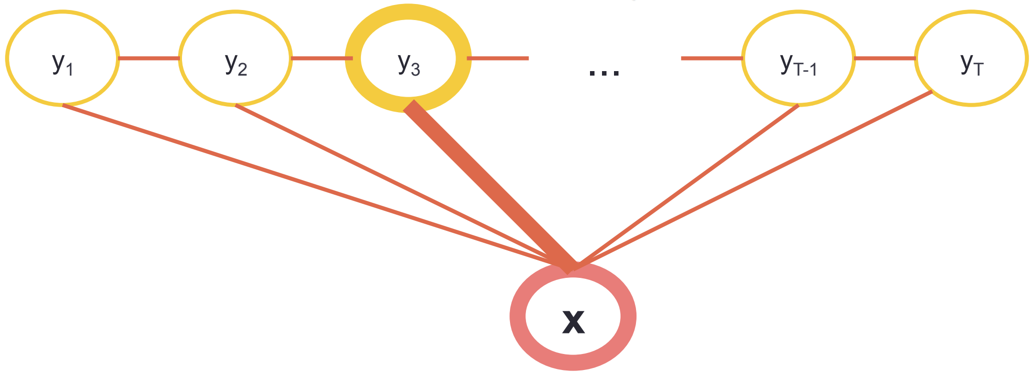
In general, we often define a feature function as a binary function, taking current label and its dependent variable into account. For example:

- transition function $$f_1(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } y_{t-1}\text{=ADJ} \\ 0 & \text{otherwise} \end{cases}$$



Feature function: More example

- state function $$f_2(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_t\text{=fox} \\ 0 & \text{otherwise} \end{cases}$$



$\mathbf{x} = x_1, x_2, \boxed{x_3}, ..., x_T$

- The whole input sequences can be used in a feature function.

$$f_3(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_{t-1}\text{=an} \\ 0 & \text{otherwise} \end{cases}$$

Feature function: More example

Other features other than word form can be used too.

$$f_4(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=PROPER NOUN and } x_t \text{ is capitalized} \\ 0 & \text{otherwise} \end{cases}$$

$$f_5(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_{t-1} \text{ ends with ``est''} \\ 0 & \text{otherwise} \end{cases}$$

Potential

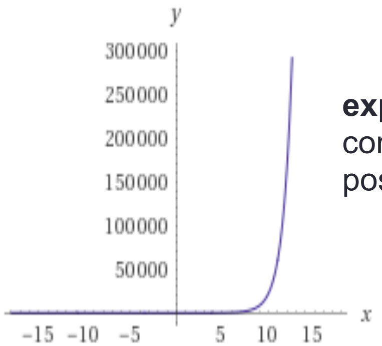
At each time step, a potential $\Psi_t(y_t, y_{t-1}, \mathbf{x}) \to \mathbb{R}^+$ is a function that takes all feature functions into account, by summing their products with the associated weight

number of all feature functions

factors at time t

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}) = \exp\{\sum_k^K \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}$$

k^th feature function



exponential function: used to convert the summation to the positive range

weight associated with each feature function, estimated within training process

Potential: Example

t	t=1	t=2	t=3	t=4	...
y*	NOUN	VERB	NOUN	VERB	...
x	The	fastest	fox	jumps	...

$$f_1(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } y_{t-1}\text{=ADJ} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_t\text{=fox} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_{t-1}\text{=an} \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=PROPER NOUN and } x_t \text{ is capitalized} \\ 0 & \text{otherwise} \end{cases}$$

$$f_5(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_{t-1} \text{ ends with ``est''} \\ 0 & \text{otherwise} \end{cases}$$

$$\theta_1 = 2.54, \theta_2 = 0.13, \theta_3 = 1.12, \theta_4 = 2.01, \theta_5 = 0.97$$

From feature functions and trained weights on the right, we can compute potentials for the predicted label sequence **y*** at time step t=3 as following:

$$\begin{aligned}\Psi_3(y_3^*, y_2^*, \mathbf{x}) &= \exp\{\sum_{k=1}^{5} \theta_k f_k(y_3^*, y_2^*, \mathbf{x})\} \\ &= \exp\{(0 \times 2.54) + (1 \times 0.13) + (0 \times 1.12) + (0 \times 2.01) + (1 \times 0.97)\} \\ &= 3.00\end{aligned}$$

Probability of the whole sequence

Joint probability distribution of input and output sequence $p(\mathbf{y}, \mathbf{x})$ can be represented as:

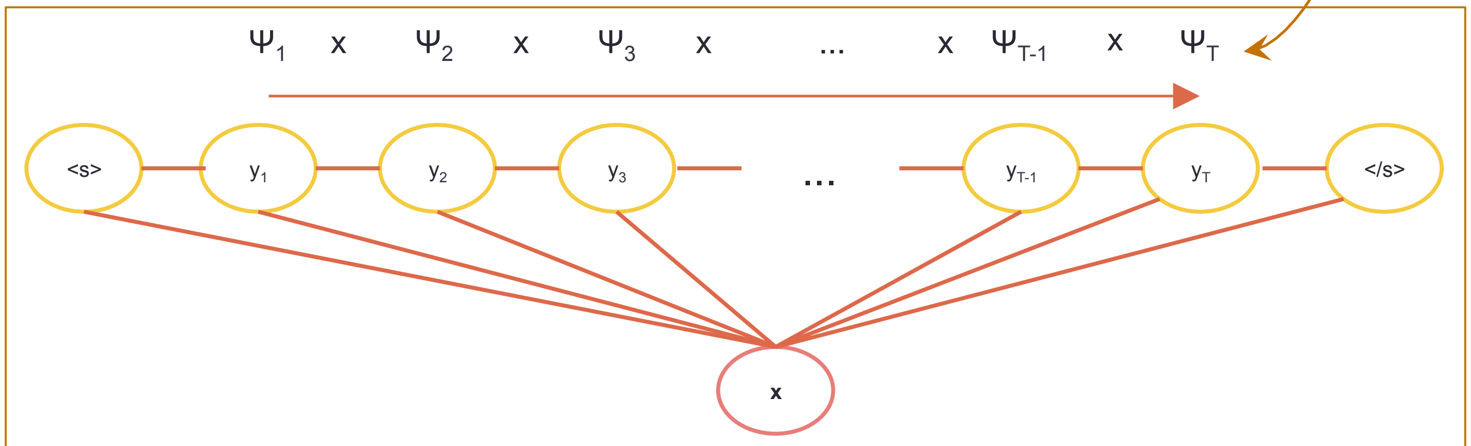
$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x})$$

With p(**y**, **x**) we can compare and pick the best **y**

Sum of scores for all possible labels with all possible input sequences

$$Z = \sum_{\mathbf{x},\mathbf{y}} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x})$$

Score of a sequence label y for a given sequence input x



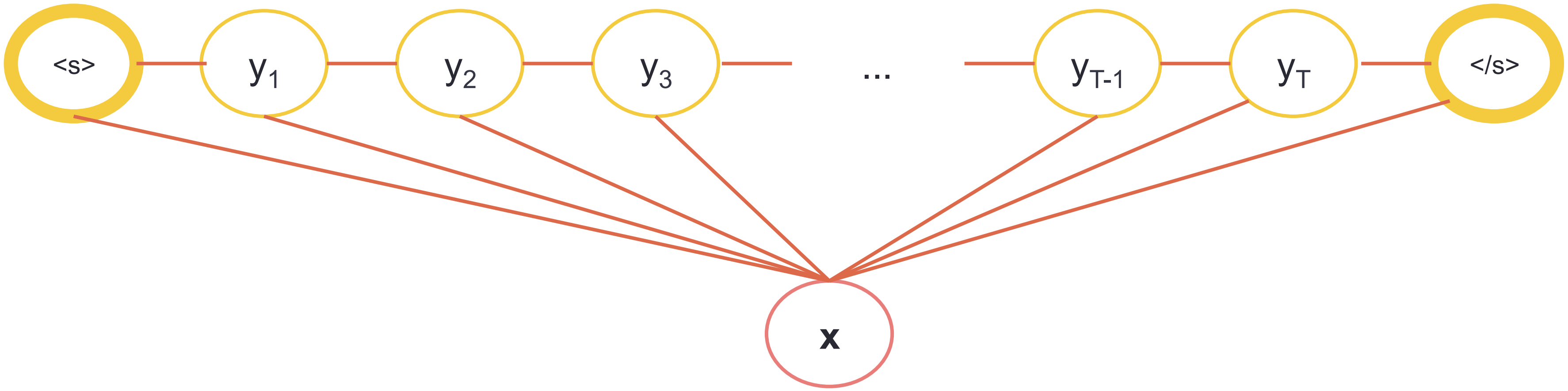
$$\mathbf{x} = \text{<s>}, x_1, x_2, x_3, ..., x_T, \text{</s>}$$

Special states and characters

To simplify modeling, we add two new special states and characters:

<s> indicates the beginning of the sequence

</s> indicates the end of the sequence



$$\mathbf{x} = \text{<s>}, x_1, x_2, x_3, ..., x_T, \text{</s>}$$

Product of sum over feature functions

From the definition of factors, the joint distribution can be represented by a number of feature functions

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x})$$

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}$$

$$Z = \sum_{\mathbf{x},\mathbf{y}} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x})$$

Computing **Z** is intractable:

Imagine a sentence of 20 words with vocabulary size of 100,000

we have to consider all $(100000)^{20}$ possible input sequences!

Linear-chain CRF

Modeling conditional probability distribution $p(\mathbf{y}|\mathbf{x})$ is enough for classification tasks.

So, in linear-chain CRF, we model the conditional distribution by using these two equations:

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y},\mathbf{x})}{p(\mathbf{x})} = \frac{p(\mathbf{y},\mathbf{x})}{\sum_{\mathbf{y}'} p(\mathbf{y}',\mathbf{x})}$$

$$p(\mathbf{y},\mathbf{x}) = \frac{1}{\mathbf{Z}} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}$$

Linear-chain CRF

A linear-chain CRF is a conditional distribution

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}$$

, where Z(**x**) is an instance-specific normalization function

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}$$

$$Z = \sum_{\mathbf{x},\mathbf{y}} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x})$$

Linear-chain CRF

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \prod_{t=1}^{T} \underbrace{\exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}}_{}$$

normalization function
the sum of products of all possible output sequences

not the same as **Z** in the joint distribution

multiply over all time steps

sum of weighted feature functions at one time step, then taken to the **exponential** function

Linear-chain CRF big picture

- Wants P(y|x)
- Assumes independence, where we only consider $P(y_{t-1}, y_t, \mathbf{x})$
- How to model $P(y_{t-1}, y_t, \mathbf{x})$?
  - Still too hard, let's make it into a function where high value means high probability – potential functions $\Psi_t(y_t, y_{t-1}, \mathbf{x})$
  - Still too hard, let's build it from pieces – feature functions
- We can get $P(\mathbf{y},\mathbf{x})$ by multiplying all $\Psi_t(y_t, y_{t-1}, \mathbf{x})$
  - This is not a probability, need a normalization
- We can also get P(y|x) from multiplying all $\Psi_t(y_t, y_{t-1}, \mathbf{x})$ and use chain rule.
  - Still need a normalization, but easier.

- This is our model, but!
  - How to use? How to estimate potential functions? What features functions?

How to use?

- If we are given the model, and **x**

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}$$

- find **y**
  - Not so straight forward, many possible **y**
    - Noun, adjective, verb
    - Noun, noun, verb
    - Verb, noun, noun
    - Too many possibilities to compare
- Solution, dynamic programing (Viterbi).

Decoding

Decoding is the process to find the best output (label sequences) from the given input (input sequences)

For linear-chain CRF which takes the form:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}$$

Decoding is the process to find the label sequence $\mathbf{y} = y_1, y_2, ..., y_T$ which yields the maximum $p(\mathbf{y}|\mathbf{x})$

Decoding: Brute force approach

If we have a set of states S with size |S| and an input sequence of size T, there are $|S|^T$ possible label sequences, each sequence uses O(T) decoding time.

The	quick	…	dog	.
ADJ	ADJ	…	ADJ	ADJ
ADJ	ADJ	…	ADJ	ADP
		…		
DET	ADJ	…	ADJ	PUNCT
		…		
X	X	…	X	VERB
X	X	…	X	X

Consider an input sequence of 10 words, a set of part-of-speech with size of 17:
all possible label sequences $= 17^{10} \approx 2\text{x}10^{12}$ (trillion) sequences

Viterbi algorithm

Viterbi algorithm is an algorithm for decoding based on dynamic programming.

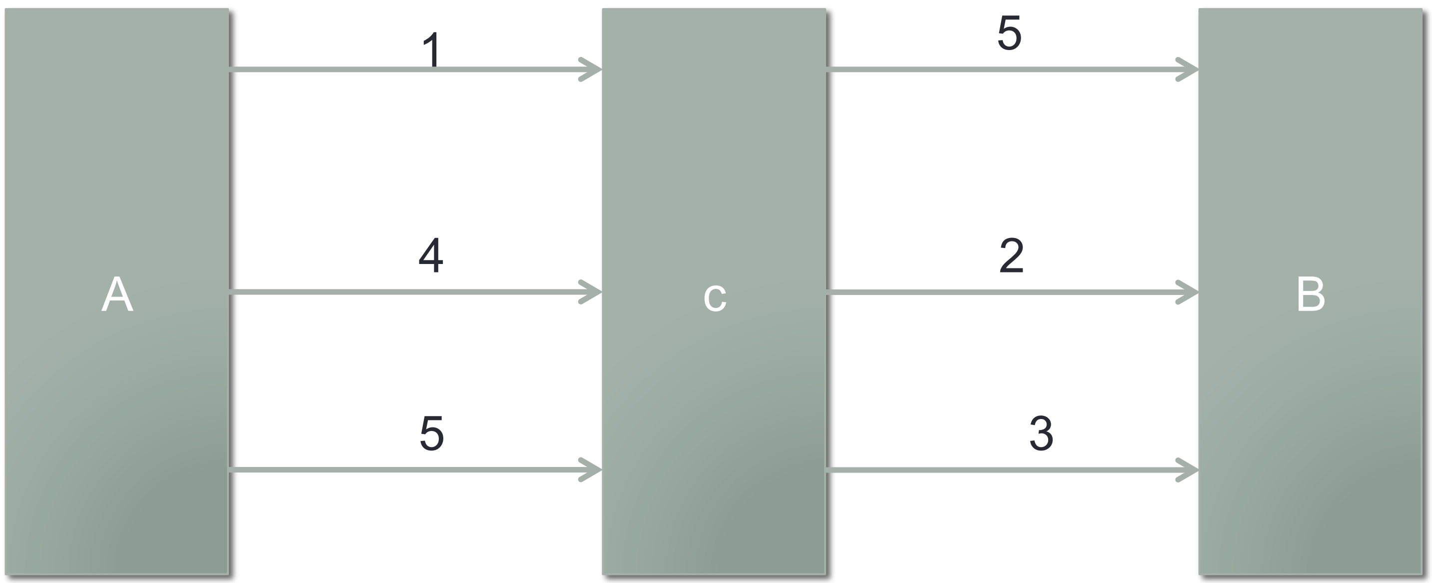
From the equation, we can see the Z(x) is the same for all possible label sequences, so we can consider only the part in the rectangle

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \boxed{\prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}}$$

Find the label sequence **y** that maximize this value

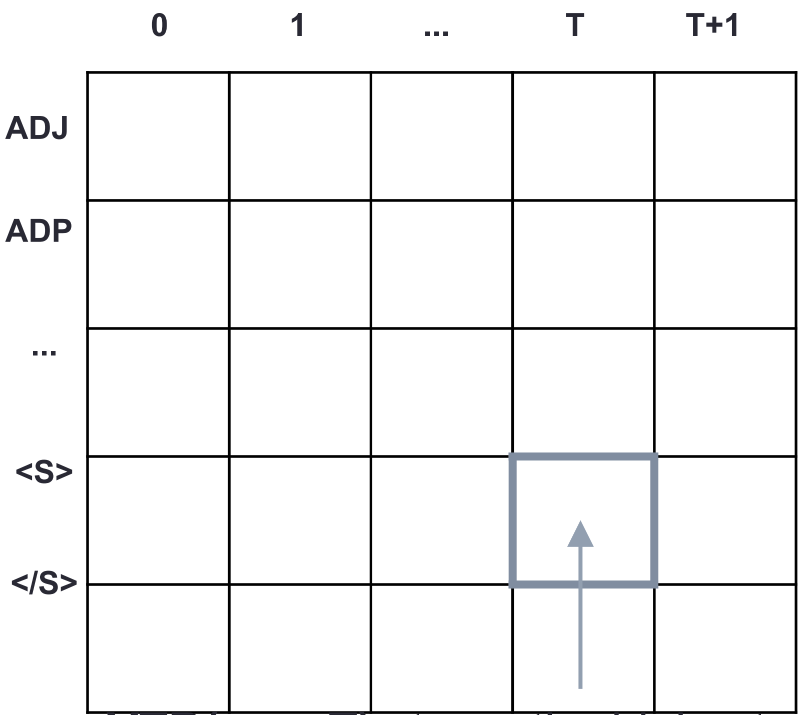
Dynamic programing

- Saving computation for future use. How?
- Example: Find best route from A to B

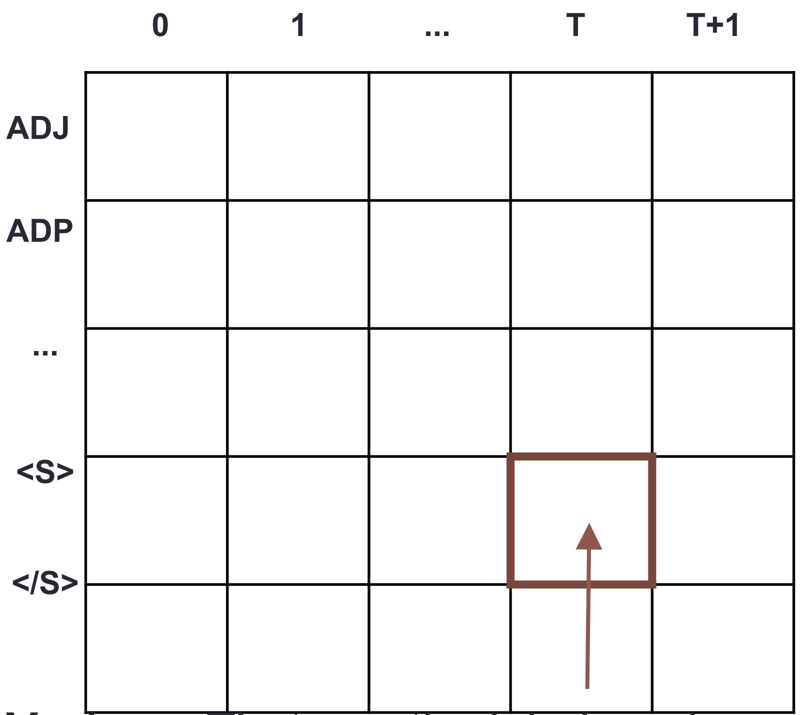


Viterbi: Structure

Create two 2D arrays: VTB and Var



VTB(<s>, T) stores the highest value a label sequence that y_T is <s> can take



Var(<s>, T) stores the label y_{T-1} in the sequence that yields the value in VTB(<s>, T)

Viterbi: Initialization

VTB

	0	1	...	T	T+1
ADJ	0	0.12			
ADP	0	0.03			
...	...	...			
<S>	0	10e-9			
</S>	1	10e-3			

For all other labels, apply the same way as ADJ

$$\mathrm{VTB}(\mathrm{ADJ},1) = \underbrace{\Psi_1(\mathrm{ADJ}, <s>, \mathbf{x})}_{\text{Factors at time t=1, for current label=ADJ, previous label=<s>}}\underbrace{\mathrm{VTB}(<s>,0)}_{1}$$

Var

	0	1	...	T	T+1
ADJ	-	<s>			
ADP	-	<s>			
...	...	...			
<S>	-	<s>			
</S>	-	<s>			

The first label of the output sequence must be <s>

Viterbi: Iteration

Iterate from t=2 to t=T+1

VTB

	0	1	...	T	T+1
ADJ	0	0.12	...	2.32	0.22
ADP	0	0.03	...	0.02	0.10
...	...	...	...	...	...
<S>	0	10e-9	...	0.03	0.02
</S>	1	10e-3	...	1.12	3.09

Var

	0	1	...	T	T+1
ADJ	-	<s>	...	NOUN	PREPO
ADP	-	<s>	...	NOUN	ADJ
...	...	...	...	...	...
<S>	-	<s>	...	X	X
</S>	-	<s>	...	VERB	ADJ

$$\mathrm{VTB}(\mathrm{ADJ},t) = \max_{i \in S} \underbrace{\Psi_t(\mathrm{ADJ}, i, \mathbf{x})}_{\text{Factors at time t}} \underbrace{\mathrm{VTB}(i, t-1)}_{\text{Maximum value that label i can take at time t-1}}$$

Find the max among values from all previous label i

Factors at time t

Maximum value that label i can take at time t-1

Var(y, t) Stores the value i that maximize the value of VTB(y,t)

Viterbi: Finalize

Backtrack from Var(</s>, T+1) to get the label seque $p(\mathbf{y}|\mathbf{x})$ that maximize

Var

	0	**1**	**...**	**T**	**T+1**
ADJ	-	<s>	...	NOUN	PREPO
ADP	-	<s>	...	NOUN	ADJ
...	...	...	...	...	...
<S>	-	<s>	...	X	X
</S>	-	<s>	...	VERB	ADJ

For example:
output sequence =
<s>, NOUN, …, NOUN, ADJ,
</s>

Linear-chain CRF big picture

- Wants P(y|x)
- Assumes independence, where we only consider $P(y_{t-1}, y_t, \mathbf{x})$
- How to model $P(y_{t-1}, y_t, \mathbf{x})$?
  - Still too hard, let's make it into a function where high value means high probability – potential functions $\Psi_t(y_t, y_{t-1}, \mathbf{x})$
  - Still too hard, let's build it from pieces – feature functions
- We can get $P(\mathbf{y}, \mathbf{x})$ by multiplying all $\Psi_t(y_t, y_{t-1}, \mathbf{x})$
  - This is not a probability, need a normalization
- We can also get P(y|x) from multiplying all $\Psi_t(y_t, y_{t-1}, \mathbf{x})$ and use chain rule.
  - Still need a normalization, but easier.

- This is our model, but!
  - How to use? Viterbi
  - How to estimate potential functions? What features functions?

Parameters?

Parameters to be learned are weights associated to each feature functions. So, number of parameters equals number of feature functions.

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\}$$

parameters

Training loss

For linear-chain CRF, parameters are trained by maximum likelihood.

$$\ell(\theta) = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})$$

To clarify, parameters θ are trained to maximize the log probability of all pairs of label $y^{(i)}$ and input $x^{(i)}$ in the training set.

Learning algorithm

To learn parameters from the loss function $\ell(\theta)$, several learning algorithm can be used. Some popular learning algorithms for linear-chain CRFs are

- Limited-memory BFGS
- Stochastic Gradient Descent

Feature functions?

- Anything you can think of, the more the better.
  - The model will learn what is important.

$$f_1(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } y_{t-1}\text{=ADJ} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_t\text{=fox} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_{t-1}\text{=an} \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=PROPER NOUN and } x_t \text{ is capitalized} \\ 0 & \text{otherwise} \end{cases}$$

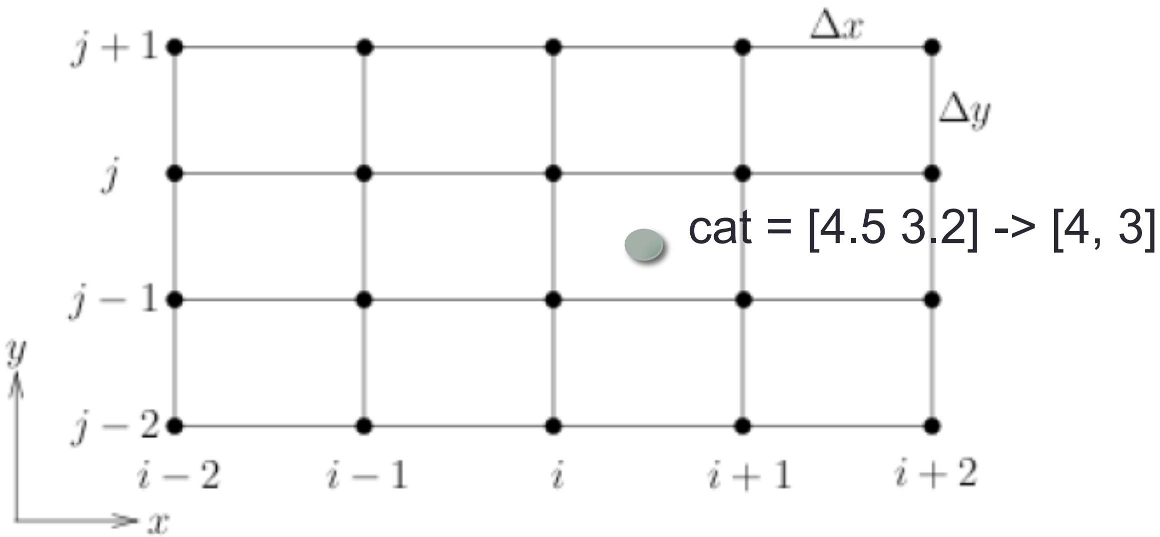
$$f_5(y_t, y_{t-1}, \mathbf{x}, t) = \begin{cases} 1 & \text{if } y_t\text{=NOUN and } x_{t-1} \text{ ends with ``est''} \\ 0 & \text{otherwise} \end{cases}$$

CRFsuite

- An implementation of CRFs for labeling sequential data in C++
  SWIG API is provided to be an interface for various languages
  http://www.chokkan.org/software/crfsuite/
- python-crfsuite: Python binding for crfsuite
  https://github.com/scrapinghub/python-crfsuite
- An example use of python-crfsuite can be found at
  https://github.com/scrapinghub/python-crfsuite/blob/master/examples/CoNLL%202002.ipynb

CRF with neural networks

- Many ways to use neural networks with CRFs
  - Caveats: most CRF takes discrete features.
- Neural networks features (embeddings) are continuous.
- Solution1: discretize the embeddings



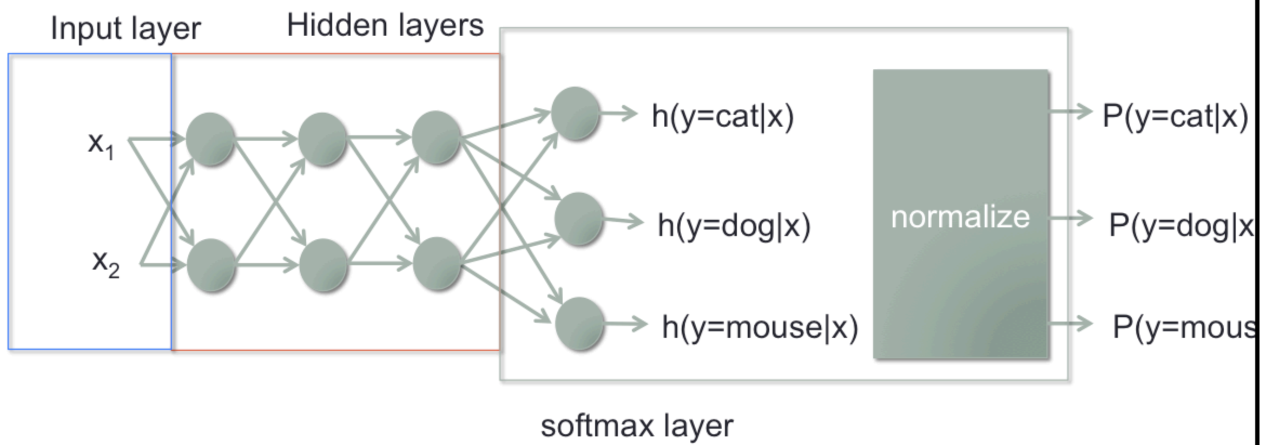
CRF with neural networks

- Many ways to use neural networks with CRFs
  - Caveats: most CRF takes discrete features.
- Neural networks features (embeddings) are continuous.
- Solution2: Use continuous version of CRF

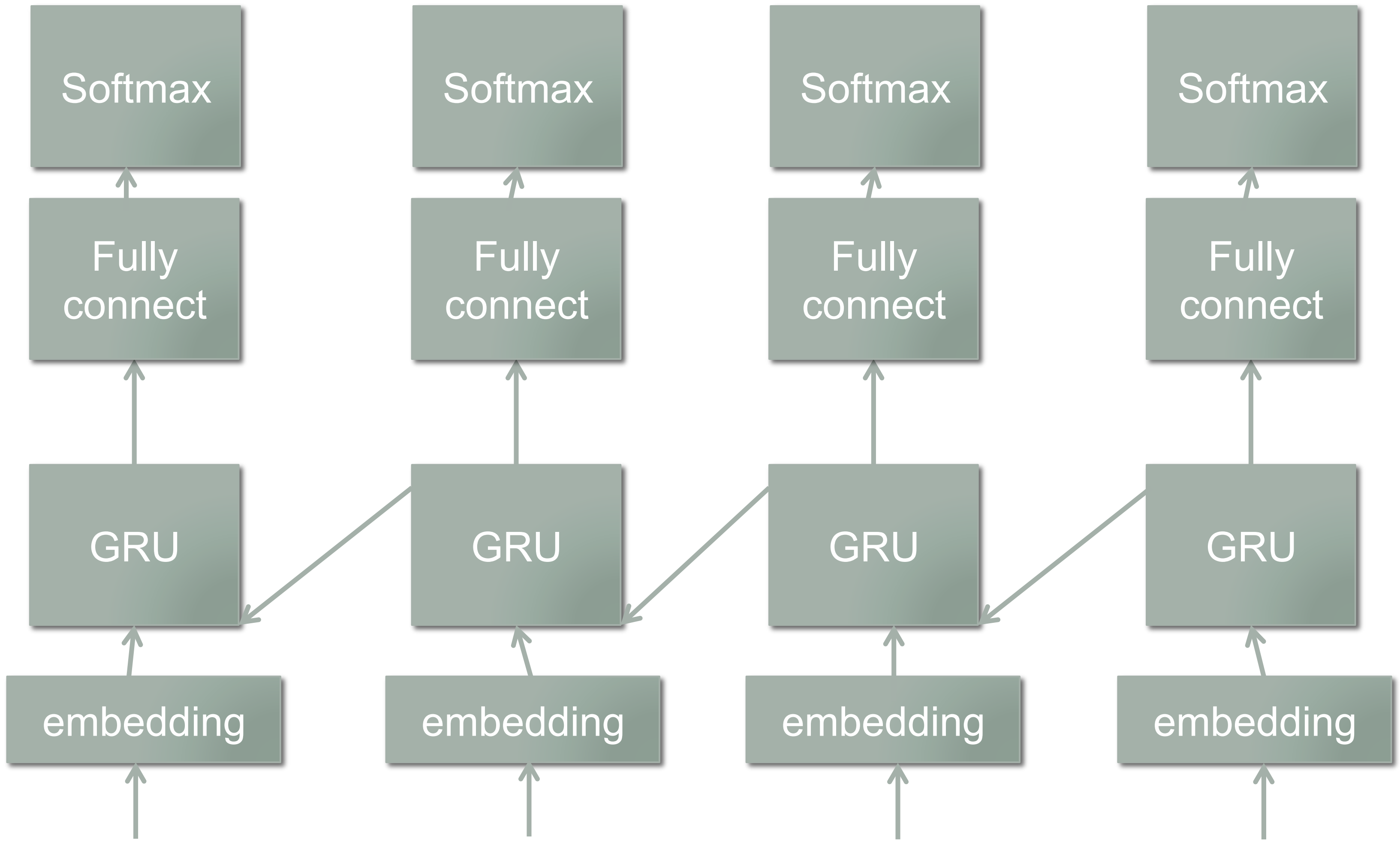
CRF with neural networks

- Solution3: change the softmax layer and loss function
- CRFlayer: P(**y**|**x**) not just P(y_t|x_t)

$$P(y = j|x) = \frac{e^{h(y=j|x)}}{\Sigma_y e^{h(y|x)}}$$

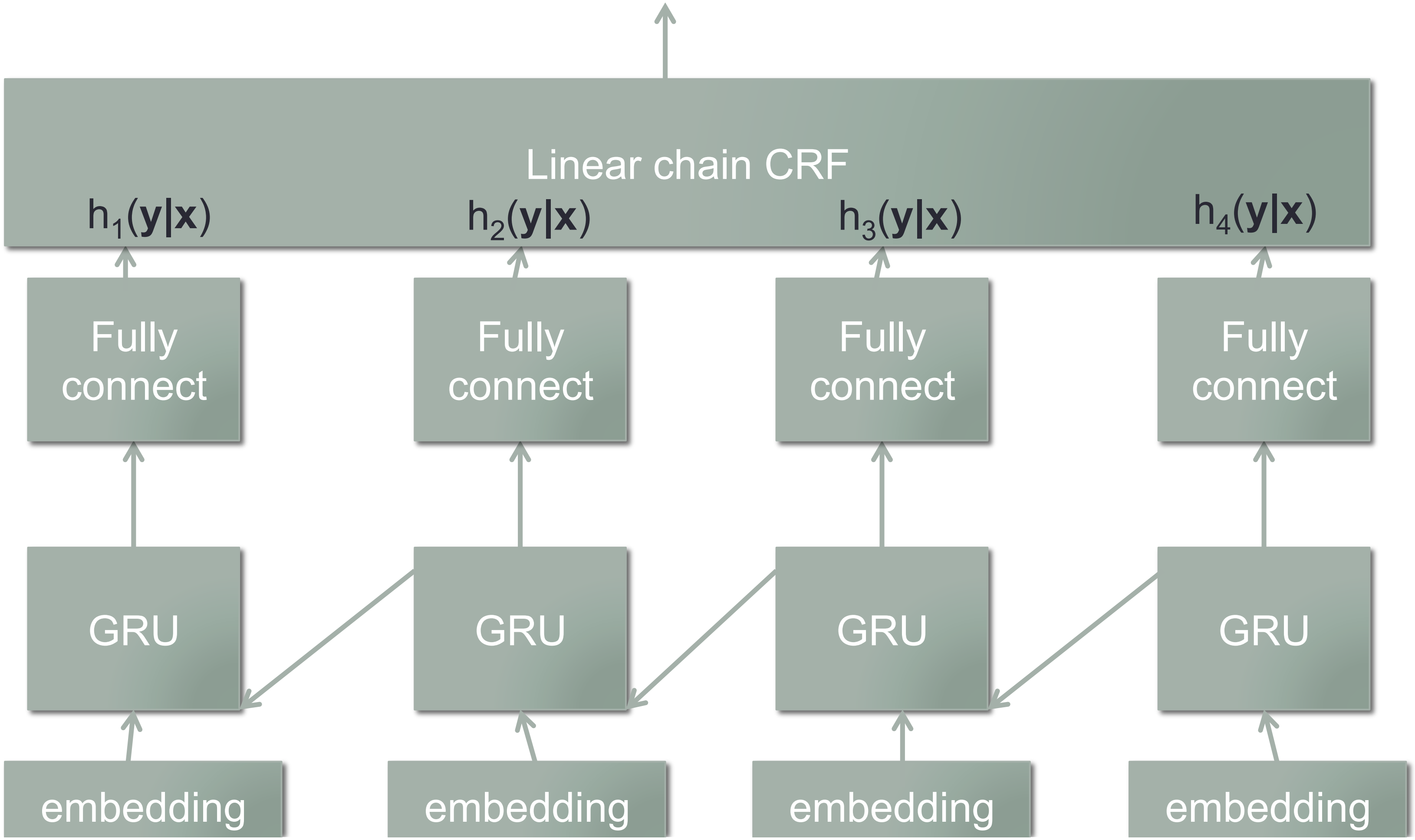


Neural network for POS



Neural network for POS with CRF output

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp \cdot h_t(\mathbf{y}|\mathbf{x})$$



Neural network for POS with CRF output

- Need to use Viterbi still for finding the best sequence

- Loss function: still cross-entropy
  - Loss = -log(P(**y'**)) where **y'** is the true output

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp \cdot h_t(\mathbf{y}|\mathbf{x})$$

Example code: Tensorflow
https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html
Example code: Keras
https://github.com/Hironsan/keras-crf-layer

Homework/workshop