Perceptrons and Structured Learning

James Pustejovsky
CS 115B

Brandeis University
January 30, 2024

with slides by Vivek Srikumar

Linear Classifiers

- Input is a n dimensional vector **x**
- Output is a label $y \in \{-1, 1\}$ For now

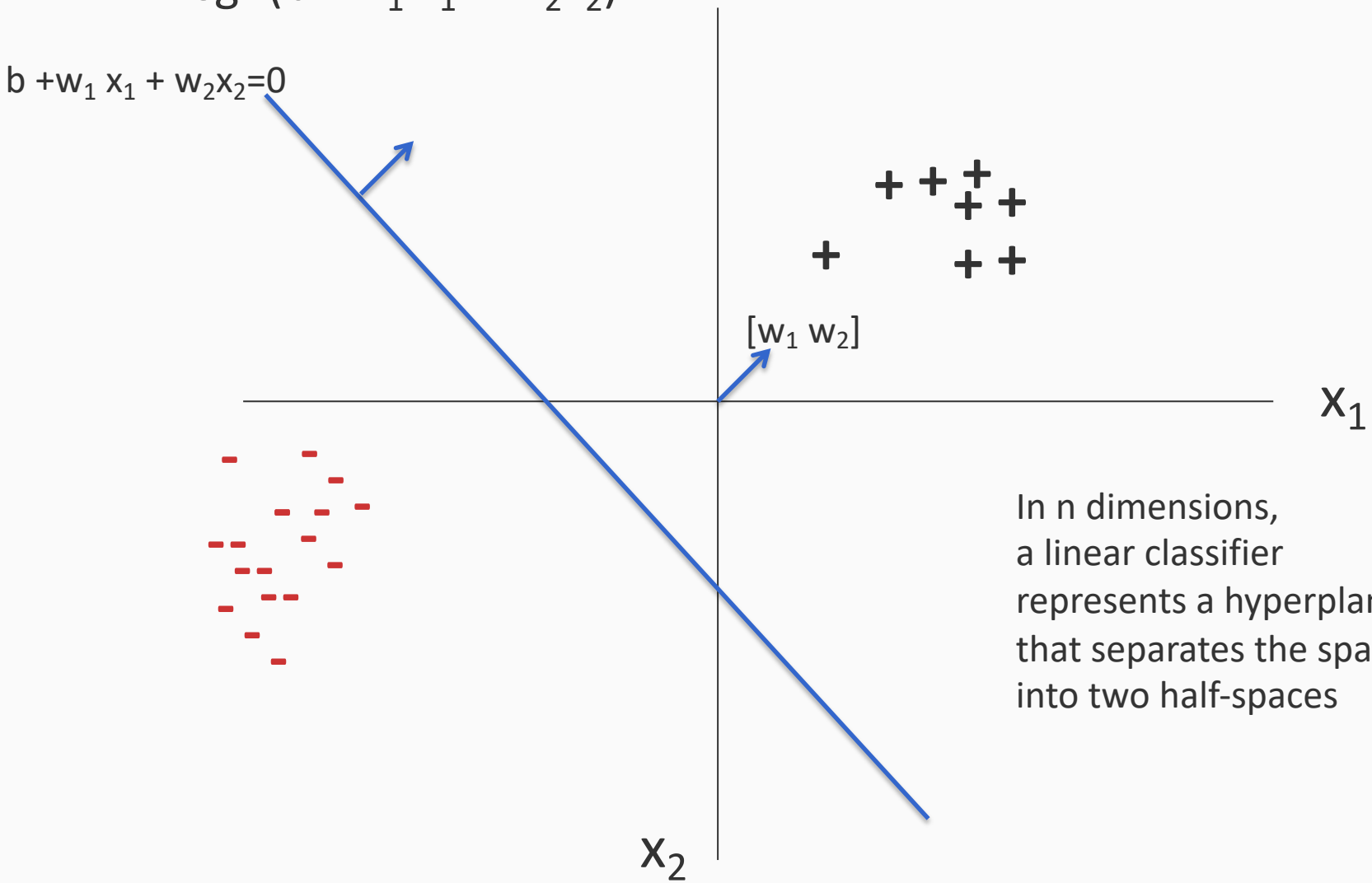
- Linear threshold units classify an example $\boldsymbol{x}$ using the classification rule

$$\mathrm{sgn}(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{sgn}(b + \textstyle\sum_i w_i x_i)$$

  - $b + \boldsymbol{w}^T \boldsymbol{x} \geq 0$) Predict y = 1
  - $b + \boldsymbol{w}^T \boldsymbol{x} < 0$) Predict y = -1

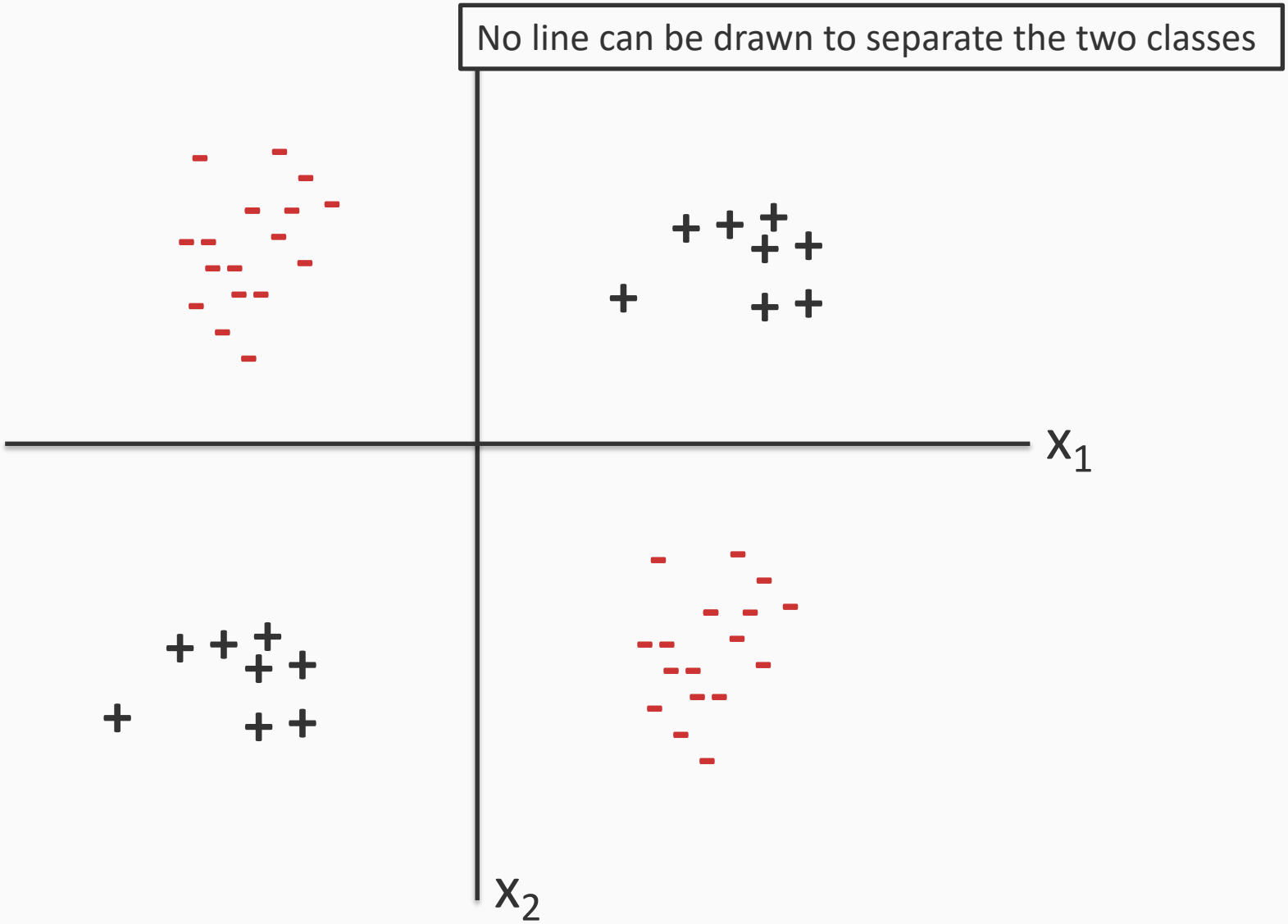
The geometry of a linear classifier

$\text{sgn}(b + w_1 x_1 + w_2 x_2)$



In n dimensions,
a linear classifier
represents a hyperplane
that separates the space
into two half-spaces

XOR is not linearly separable



Even these functions can be *made* linear

These points are not separable in 1-dimension by a line

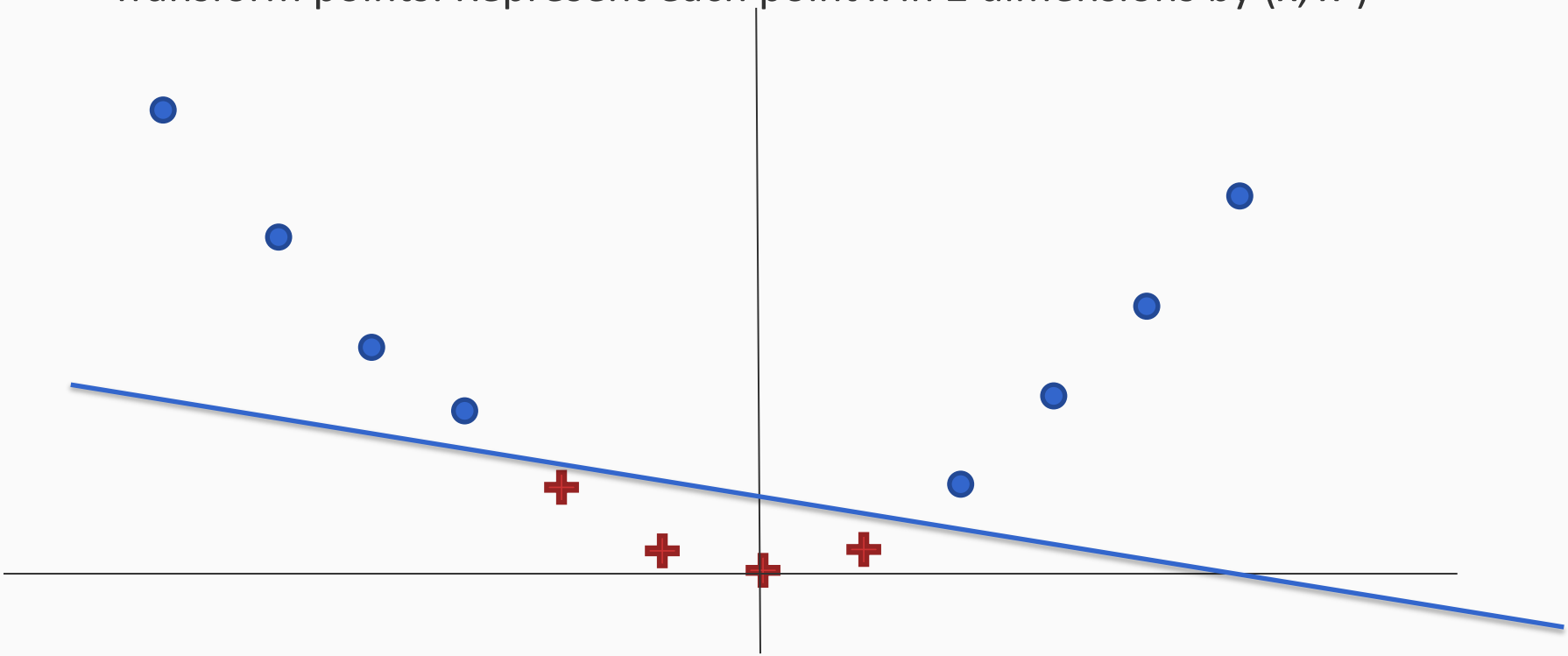
What is a one-dimensional line, by the way?

The trick: Change the representation

Even these functions can be *made* linear

The trick: Use feature *conjunctions*

Transform points: Represent each point x in 2 dimensions by (x, x^2)



Now the data is linearly separable in this space!

Linear classifiers are an expressive hypothesis class

- Many functions are linear
  - Conjunctions, disjunctions
  - At least m-of-n functions
- Often a good guess for a hypothesis space
  - *If we know a good feature representation*
- Some functions are not linear
  - The XOR function
  - Non-trivial Boolean functions

We will see in the coming weeks that structured predictors can also be defined via linear functions.

The Perceptron algorithm

- Rosenblatt 1958
- The goal is to find a separating hyperplane
  - For separable data, guaranteed to find one
- An online algorithm
  - Processes one example at a time
- Several variants exist

The algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^d, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^d$
2. For epoch in $1 \cdots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
            - update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

Prediction on a new example with features $\mathbf{x}$: $\mathrm{sgn}(\mathbf{w}^T \mathbf{x})$

The algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^d, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^d$
2. For epoch in $1 \cdots T$:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
        - update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

T is a hyper-parameter to the algorithm

Another way of writing that there is an error

Prediction on a new example with features $\mathbf{x}$: $\mathrm{sgn}(\mathbf{w}^T \mathbf{x})$

Convergence theorem

If there exist a set of weights that are consistent with the data (i.e. the data is linearly separable), the perceptron algorithm will converge after a finite number of updates.

- [Novikoff 1962]

Beyond the separable case

- The good news
  - Perceptron makes no assumption about data distribution
  - Even adversarial
  - After a fixed number of mistakes, you are done. Don’t even need to see any more data

- The bad news: Real world is not linearly separable
  - Can’t expect to *never* make mistakes again
  - What can we do: more features, try to be linearly separable if you can

Variants of the algorithm

- The original version: Return the final weight vector
- Averaged perceptron
  - Returns the average weight vector from the entire training time (i.e longer surviving weight vectors get more say)
  - Widely used
  - A practical approximation of the Voted Perceptron

Learning as loss minimization

- Collect some annotated data. More is generally better
- Pick a hypothesis class (also called model)
  - Eg: linear classifiers, deep neural networks
  - Also, decide on how to impose a preference over hypotheses
- Choose a loss function
  - Eg: negative log-likelihood, hinge loss
  - Decide on how to penalize incorrect decisions
- Minimize the expected loss
  - Eg: Set derivative to zero and solve on paper, typically a more complex algorithm

Learning as loss minimization

- The setup
  - Examples **x** drawn from a fixed, unknown distribution D
  - Hidden oracle classifier f labels examples
  - We wish to find a hypothesis h that mimics f

Learning as loss minimization

- The setup
  - Examples **x** drawn from a fixed, unknown distribution D
  - Hidden oracle classifier f labels examples
  - We wish to find a hypothesis h that mimics f
- The ideal situation
  - Define a function L that penalizes bad hypotheses
  - **Learning:** Pick a function h to minimize expected loss

$$\min_{h \in H} E_{\mathbf{x} \sim D} \left[L\left(h(\mathbf{x}), f(\mathbf{x})\right) \right]$$

But distribution D is unknown

Learning as loss minimization

- The setup
  - Examples **x** drawn from a fixed, unknown distribution D
  - Hidden oracle classifier f labels examples
  - We wish to find a hypothesis h that mimics f
- The ideal situation
  - Define a function L that penalizes bad hypotheses
  - **Learning:** Pick a function h to minimize expected loss

$$\min_{h \in H} E_{\mathbf{x} \sim D} \left[L\left(h(\mathbf{x}), f(\mathbf{x})\right) \right]$$

But distribution D is unknown

- Instead, minimize *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

Empirical loss minimization

Learning = minimize *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

Is there a problem here?

Overfitting!

We need something that biases the learner towards simpler hypotheses

- Achieved using a regularizer, which penalizes complex hypotheses
- Capacity control for better generalization

Regularized loss minimization

- Learning: $\min_{h \in \mathrm{H}} \mathrm{regularizer(w)} + \mathrm{C} \frac{1}{m} \sum_i L(h(x_i), y_i)$

- With L2 regularization: $\min_{w} \frac{1}{2}\, w^T w + C \sum_i L(F(x_i, w), y_i)$

Regularized loss minimization

- Learning: $\min_{h \in \mathrm{H}} \mathrm{regularizer(w)} + \mathrm{C} \frac{1}{m} \sum_i L(h(x_i), y_i)$
- With L2 regularization: $\min_{w} \frac{1}{2} w^T w + C \sum_i L(F(x_i, w), y_i)$
- What is a loss function?
  - Loss functions should penalize mistakes
  - We are minimizing average loss over the training data

How do we train in such a regime?

- Suppose we have a predictor F that maps inputs x to a score F(x, w) that is thresholded to get a label
  - Here w are the parameters that define the function
  - Say F is a differentiable function

- How do we use a labeled training set to learn the weights i.e. solve this minimization problem?

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

How do we train in such a regime?

- Suppose we have a predictor F that maps inputs x to a score F(x, w) that is thresholded to get a label
  - Here w are the parameters that define the function
  - Say F is a differentiable function

- How do we use a labeled training set to learn the weights i.e. solve this minimization problem?

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

- We could compute the gradient of the loss and descend along that direction to minimize

$$\min_{\boldsymbol{w}} \sum_{i} L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $\mathrm{S} = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:

3. Return w

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $\mathrm{S} = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
    1. Shuffle the training set

3. Return w

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $\mathrm{S} = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in \mathrm{S}$:

3. Return w

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $\mathrm{S} = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in \mathrm{S}$:
      - Treat this example as the entire dataset
        Compute the gradient of the loss $\nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
3. Return w

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $\mathrm{S} = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in \mathrm{S}$:
      - Treat this example as the entire dataset
        Compute the gradient of the loss $\nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $\mathrm{S} = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in \mathrm{S}$:
      - Treat this example as the entire dataset
        Compute the gradient of the loss $\nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

γ_t: learning rate,
many tweaks possible

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $\mathrm{S} = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in \mathrm{S}$:
      - Treat this example as the entire dataset
        Compute the gradient of the loss $\nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

The objective is not convex.
Initialization can be important

γ_t: learning rate,
many tweaks possible

A more general form

Suppose we want to minimize a function that is the sum of other functions

$$f(x) = \sum_{i=1}^{n} f_i(x)$$

- Initialize x
- Loop till convergence:
  - Pick i randomly from $\{1, 2, \cdots, n\}$
  - Update $x \leftarrow x - stepsize \cdot \nabla f_i(x)$
- Return x

In practice…

- There are many variants of this idea
- Several named learning algorithms
  - AdaGrad, AdaDelta, RMSProp, Adam
- But the key components are the same. We need to…
  1. …sample a tiny subset of the data at each step
  2. …compute the gradient of the loss using this subset
  3. …take a step in the negative direction of the gradient

Standard loss functions

We need to think about the problem we have at hand

Is it a…

1. Binary classification problem?
2. Regression problem?
3. Multi-class classification problem?
4. Or something else?

Each case is naturally paired with a different loss function

The ideal case for binary classification: The 0-1 loss

Penalize classification mistakes between true label y and prediction y'

$$L_{0-1}(y, y') = \begin{cases} 1 & \text{if } y \neq y', \\ 0 & \text{if } y = y'. \end{cases}$$

More generally, suppose we have a prediction function of the form $\mathrm{sgn}(F(x, w))$

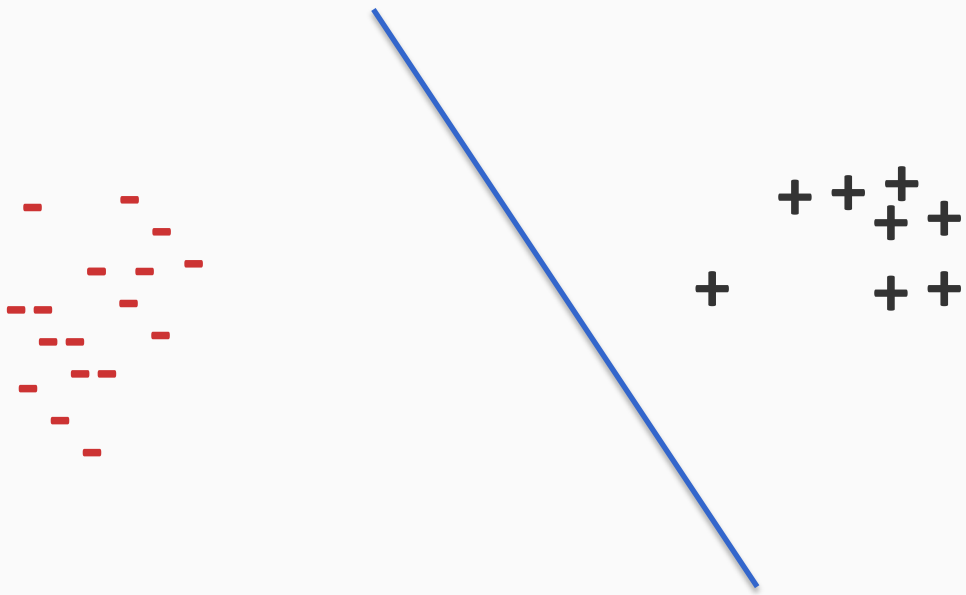
- Note that F need not be linear

$$L_{0-1}(y, y') = \begin{cases} 1 & \text{if } yF(x, w) \leq 0, \\ 0 & \text{if } yF(x, w) > 0. \end{cases}$$

Minimizing 0-1 loss is intractable. Need surrogates

Margin

The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Learning strategy

Find the linear separator that maximizes the margin

Maximizing margin and minimizing loss

Find the linear separator that maximizes the margin

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \boxed{\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)}$$

Maximize margin

Penalty for the prediction:
The Hinge loss

Binary to multiclass

- Can we use an algorithm for training binary classifiers to construct a multiclass classifier?
  - Answer: Decompose the prediction into multiple binary decisions

- How to decompose?
  - One-vs-all
  - All-vs-all
  - Error correcting codes

General setting

- Input $\mathbf{x} \in \Re^n$
  - The inputs are represented by their feature vectors
- Output $\mathbf{y} \in \{1,2,\cdots,K\}$
  - These classes represent domain-specific labels
- Learning: Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$
  - Need a learning algorithm that uses D to construct a function that can predict $\mathbf{x}$ to $\mathbf{y}$
  - Goal: find a predictor that does well on the training data and has low generalization error
- Prediction/Inference: Given an example $\mathbf{x}$ and the learned function, compute the class label for $\mathbf{x}$

1. One-vs-all classification

- Assumption: Each class individually separable from ***all*** the others

1. One-vs-all classification

- Assumption: Each class individually separable from ***all*** the others

- Learning: Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ $\quad \boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Decompose into K binary classification tasks
  - For class k, construct a *binary classification* task as:
    - **Positive examples**: Elements of D with label k
    - **Negative examples**: All other elements of D

1. One-vs-all classification

- Assumption: Each class individually separable from ***all*** the others

- Learning: Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ $\quad \mathbf{x} \in \Re^n$, $\mathbf{y} \in \{1,2,\cdots,K\}$
  - Decompose into K binary classification tasks
  - For class k, construct a *binary classification* task as:
    - **Positive examples**: Elements of D with label k
    - **Negative examples**: All other elements of D
  - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \cdots \mathbf{w}_K$ using any learning algorithm we have seen

1. One-vs-all classification

- Assumption: Each class individually separable from ***all*** the others

- Learning: Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ $\quad \boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \cdots \mathbf{w}_K$ using any learning algorithm we have seen

- Prediction: "*Winner Takes All*"

$$\operatorname{argmax}_i \mathbf{w}_i^T \mathbf{x}$$

1. One-vs-all classification

- Assumption: Each class individually separable from ***all*** the others

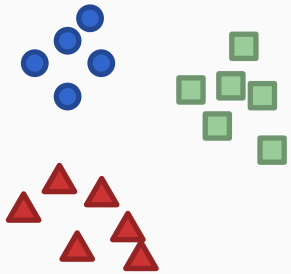
- Learning: Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ $\quad \boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \cdots \mathbf{w}_K$ using any learning algorithm we have seen

Question: What is the dimensionality of each $\mathbf{w}_i$?

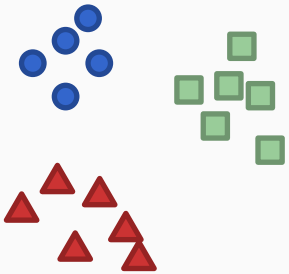
- Prediction: "*Winner Takes All*"

$$\operatorname{argmax}_i \mathbf{w}_i^T \mathbf{x}$$

Visualizing One-vs-all

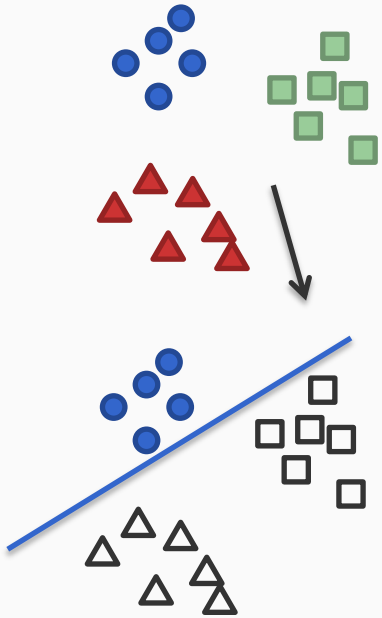


Visualizing One-vs-all



From the full dataset, construct three binary classifiers, one for each class

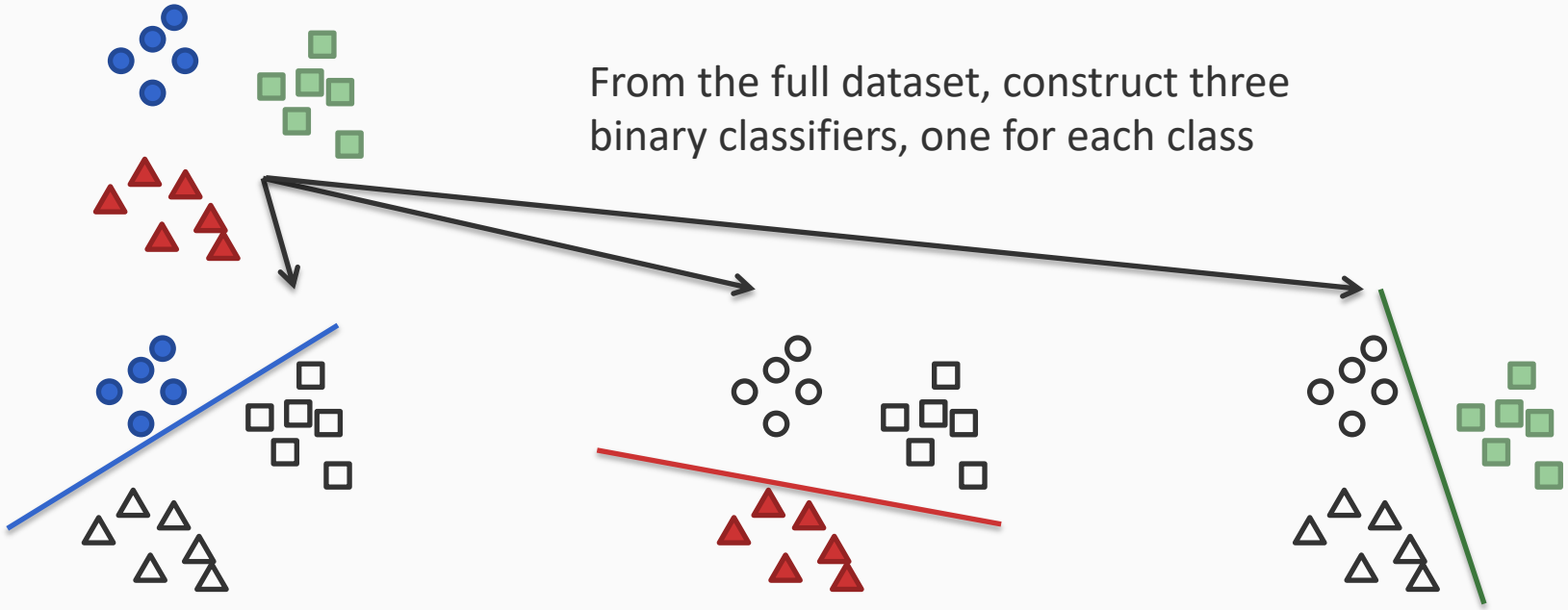
Visualizing One-vs-all



$\mathbf{w}_{\mathbf{blue}}^{T}\mathbf{x} > 0$
for **blue**
circle inputs

From the full dataset, construct three binary classifiers, one for each class

Visualizing One-vs-all

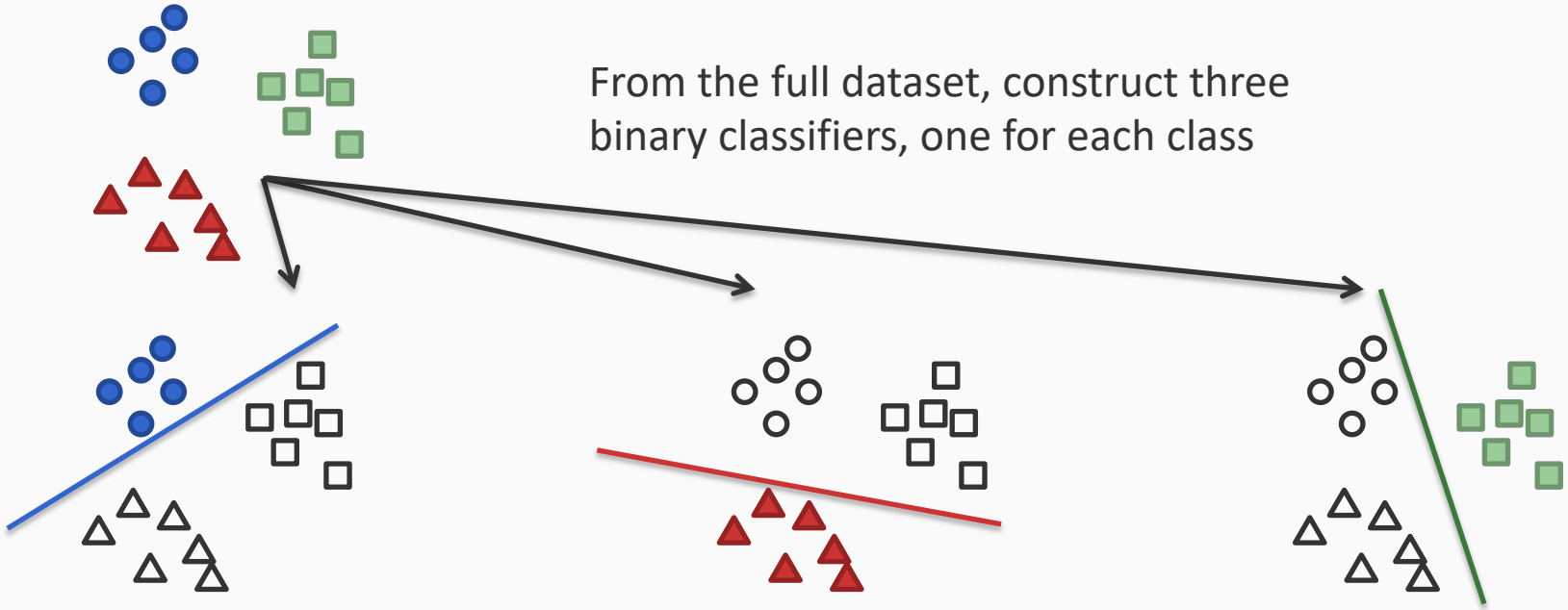


$\mathbf{w}_{\text{blue}}^T\mathbf{x} > 0$ for **blue circle inputs**

$\mathbf{w}_{\text{red}}^T\mathbf{x} > 0$ for **red triangle inputs**

$\mathbf{w}_{\text{green}}^T\mathbf{x} > 0$ for **green square inputs**

Visualizing One-vs-all



From the full dataset, construct three binary classifiers, one for each class

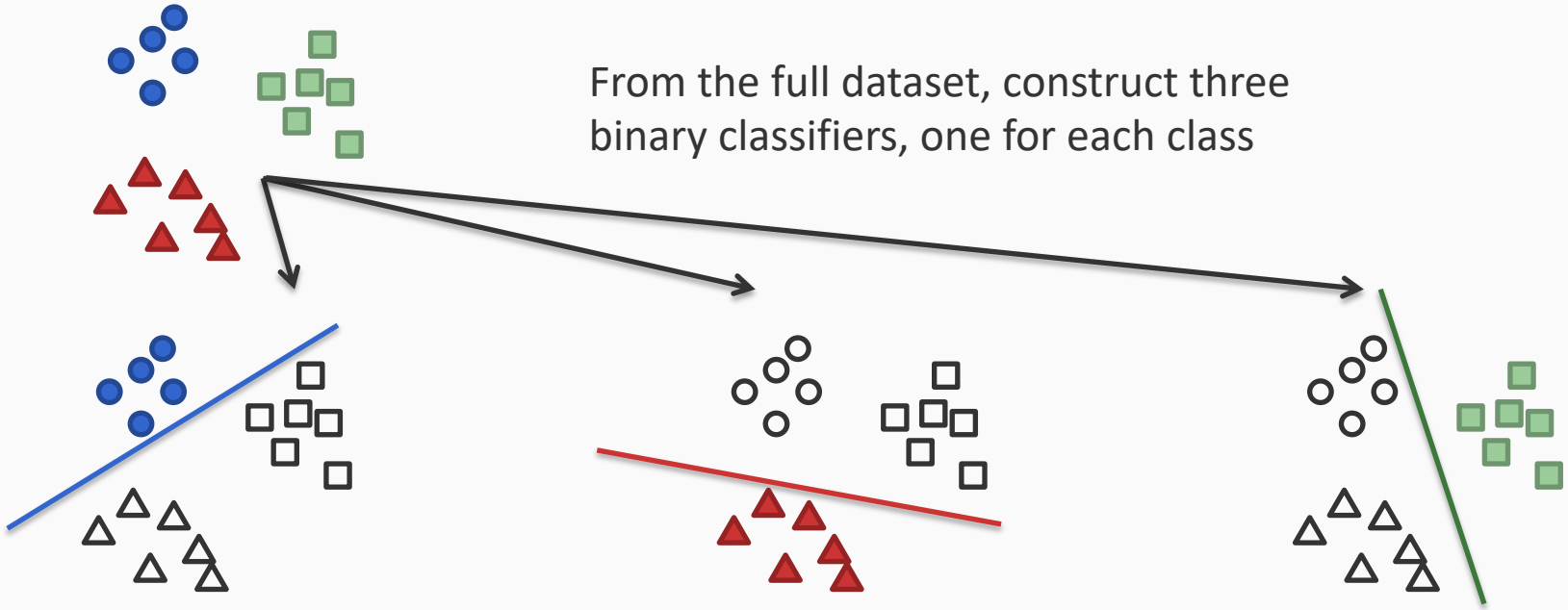
$\mathbf{w}_{\text{blue}}^T\mathbf{x} > 0$ for **blue circle inputs**

$\mathbf{w}_{\text{red}}^T\mathbf{x} > 0$ for **red triangle inputs**

$\mathbf{w}_{\text{green}}^T\mathbf{x} > 0$ for **green square inputs**

Notation: Score for blue label

Visualizing One-vs-all



From the full dataset, construct three binary classifiers, one for each class

$\mathbf{w}_{\text{blue}}^T\mathbf{x} > 0$ for **blue circle inputs**

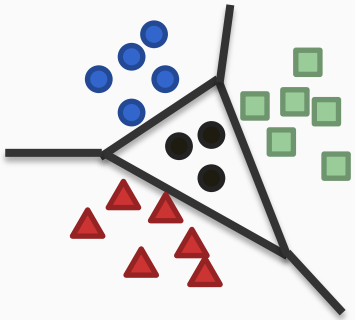
$\mathbf{w}_{\text{red}}^T\mathbf{x} > 0$ for **red triangle inputs**

$\mathbf{w}_{\text{green}}^T\mathbf{x} > 0$ for **green square inputs**

Notation: Score for blue label

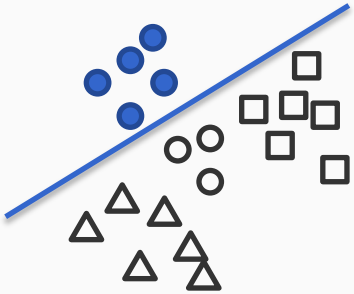
Winner Take All will predict the right answer. Only the correct label will have a positive score

One-vs-all may not always work

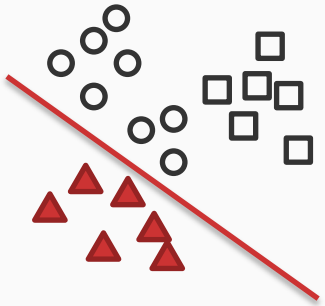


Black points are not separable with a single binary classifier

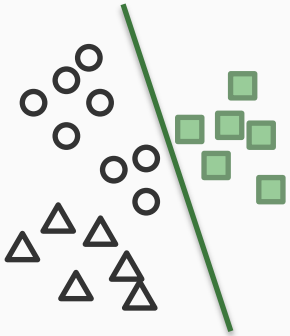
The decomposition will not work for these cases!



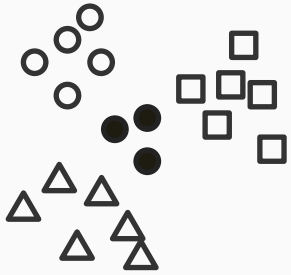
$\mathbf{w}_{\text{blue}}^T\mathbf{x} > 0$ for **blue circle inputs**



$\mathbf{w}_{\text{red}}^T\mathbf{x} > 0$ for **red triangle inputs**



$\mathbf{w}_{\text{green}}^T\mathbf{x} > 0$ for **green square inputs**



???

One-vs-all classification: Summary

- Easy to learn
  - Use any binary classifier learning algorithm

- Problems
  - No theoretical justification
  - Calibration issues
    - We are comparing scores produced by K classifiers trained independently. No reason for the scores to be in the same numerical range!
  - Might not always work
    - Yet, works fairly well in many cases, especially if the underlying binary classifiers are tuned, regularized

2. All-vs-all classification

Sometimes called one-vs-one

- Assumption: *Every* pair of classes is separable

2. All-vs-all classification

Sometimes called one-vs-one

- Assumption: *Every* pair of classes is separable

- Learning: Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, $\mathbf{x} \in \Re^n$, $\mathbf{y} \in \{1,2,\cdots,K\}$
  - For every pair of labels (j, k), create a binary classifier with:
    - Positive examples: All examples with label j
    - Negative examples: All examples with label k
  - Train $\binom{K}{2} = \frac{K(K-1)}{2}$ classifiers to separate every pair of labels from each other

2. All-vs-all classification

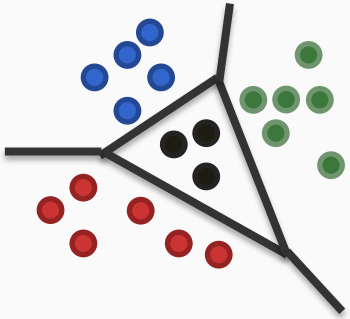
Sometimes called one-vs-one

- Assumption: *Every* pair of classes is separable

- Learning: Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, $\mathbf{x} \in \Re^n$, $\mathbf{y} \in \{1,2,\cdots,K\}$
  - Train $\binom{K}{2} = \frac{K(K-1)}{2}$ classifiers to separate every pair of labels from each other

- Prediction: More complex, each label get K-1 votes
  - How to combine the votes? Many methods
    - Majority: Pick the label with maximum votes
    - Organize a tournament between the labels

All-vs-all classification



- Every pair of labels is linearly separable here
  - When a pair of labels is considered, all others are ignored

- Problems
  1. $O(K^2)$ weight vectors to train and store

  2. Size of training set for a pair of labels could be very small, leading to overfitting of the binary classifiers

  3. Prediction is often ad-hoc and might be unstable

     Eg: What if two classes get the same number of votes? For a tournament, what is the sequence in which the labels compete?

3. Error correcting output codes (ECOC)

- Each binary classifier provides one bit of information
- With K labels, we only need $\log_2 K$ bits to represent the label
  - One-vs-all uses K bits (one per classifier)
  - All-vs-all uses $O(K^2)$ bits
- Can we get by with O(log K) classifiers?
  - Yes! Encode each label as a binary string
  - Or alternatively, if we do train more than O(log K) classifiers, can we use the redundancy to improve classification accuracy?

Using $\log_2 K$ classifiers

label#	Code		
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

8 classes, code-length = 3

- Learning:
  - Represent each label by a bit string (i.e., its code)
  - Train one binary classifier for each bit
- Prediction:
  - Use the predictions from all the classifiers to create a $\log_2 K$ bit string that uniquely decides the output

Example: For some example, if the three classifiers predict 0, 1 and 1, then the label is 3

Using $\log_2 K$ classifiers

label#	Code		
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

8 classes, code-length = 3

- Learning:
  - Represent each label by a bit string (i.e., its code)
  - Train one binary classifier for each bit
- Prediction:
  - Use the predictions from all the classifiers to create a $\log_2 K$ bit string that uniquely decides the output
- What could go wrong here?

Using $\log_2 K$ classifiers

label#	Code		
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

8 classes, code-length = 3

- Learning:
  - Represent each label by a bit string (i.e., its code)
  - Train one binary classifier for each bit
- Prediction:
  - Use the predictions from all the classifiers to create a $\log_2 K$ bit string that uniquely decides the output
- What could go wrong here?
  - Even if one of the classifiers makes a mistake, final prediction is wrong!

Error correcting output coding

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

Answer: Use redundancy

- Assign a binary string with each label
  - Could be random
  - Length of the code word **L** >= $\log_2 K$ is a parameter

- Train one binary classifier for each bit
  - Effectively, split the data into random dichotomies
  - We need only $\log_2 K$ bits
    - Additional bits act as an error correcting code

How to predict?

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

- Prediction
  - Run all **L** binary classifiers on the example
  - Gives us a predicted bit string of length **L**
  - Output = label whose code word is "closest" to the prediction
  - Closest defined using Hamming distance
    - Longer code length is better, better error-correction

- Example
  - Suppose the binary classifiers here predict 11010
  - The closest label to this is 6, with code word 11000

How to predict?

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

- Prediction
  - Run all **L** binary classifiers on the example
  - Gives us a predicted bit string of length **L**
  - Output = label whose code word is "closest" to the prediction
  - Closest defined using Hamming distance
    - Longer code length is better, better error-correction

One-vs-all is a special case of this scheme. How?

- Example
  - Suppose the binary classifiers here predict 11010
  - The closest label to this is 6, with code word 11000

Error correcting codes: Discussion

- Assumes that columns are independent
  - Otherwise, ineffective encoding
- Strong theoretical results that depend on code length
  - If minimal Hamming distance between two rows is d, then the prediction can correct up to $\frac{d-1}{2}$ errors in the binary predictions
- Code assignment could be random, or designed for the dataset or task
- One-vs-all and all-vs-all are special cases
  - All-vs-all needs a ternary code (not binary)

Error correcting codes: Discussion

- Assumes that columns are independent
  - Otherwise, ineffective encoding
- Strong theoretical results that depend on code length
  - If minimal Hamming distance between two rows is d, then the prediction can correct up to $\frac{d-1}{2}$ errors in the binary predictions
- Code assignment could be random, or designed for the dataset or task
- One-vs-all and all-vs-all are special cases
  - All-vs-all needs a ternary code (not binary)

Decomposition methods: Summary

- General idea
  - Decompose the multiclass problem into many binary problems
  - We know how to train binary classifiers
  - Prediction depends on the decomposition
    - Constructs the multiclass label from the output of the binary classifiers

- Learning optimizes ***local correctness***
  - Each binary classifier does not need to be globally correct
    - That is, the classifiers do not have to agree with each other
  - The learning algorithm is not aware of the prediction procedure!

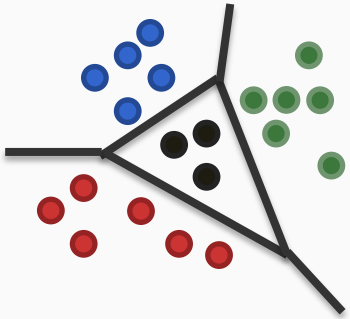
- Poor decomposition gives poor performance
  - Difficult local problems, can be "unnatural"
    - Eg. For ECOC, why should the binary problems be separable?

Modeling Structures

What we have seen: Training multiclass classifiers

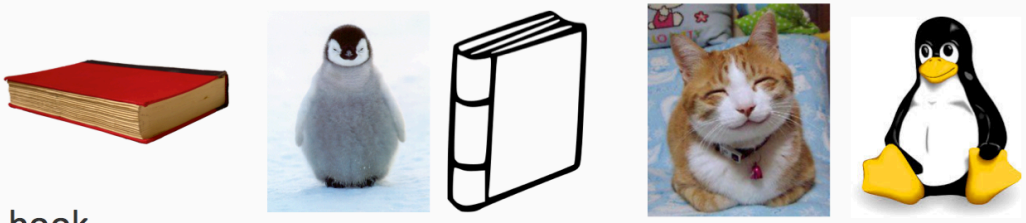
- Label belongs to a set that has more than two elements

- Methods
  - Decomposition into a collection of binary (*local*) decisions
    - One-vs-all
    - All-vs-all
    - Error correcting codes
  - Training a single (*global*) classifier
    - Constraint classification



Recipe for multiclass classification

- Collect a training set (hopefully with correct labels)



book penguin book dog penguin

- Define feature representations for inputs ($\boldsymbol{x} \in \Re^n$)

  And, $\mathbf{y} \in \{book,\ dog,\ penguin\}$

- Define linear functions to score labels

$$\underset{\mathbf{y} \in \{book,\, dog,\, penguin\}}{\operatorname{argmax}} \quad \mathbf{w}_{\mathbf{y}}^T \mathbf{x}$$

Natural extension to non-linear scoring functions too

$$\underset{\mathbf{y} \in \{book,\, dog,\, penguin\}}{\operatorname{argmax}} \quad \mathrm{score}(\mathbf{x}, \mathbf{y})$$

Recipe for multiclass classification

- Train weights so that it scores examples correctly

  e.g., for an input of type $book$, we want

$$\text{score}(\mathbf{x}, book) > \text{score}(\mathbf{x}, penguin)$$
$$\text{score}(\mathbf{x}, book) > \text{score}(\mathbf{x}, dog)$$

- Prediction: $\underset{\mathbf{y} \in \{book, dog, penguin\}}{\text{argmax}} \text{score}(\mathbf{x}, \mathbf{y})$
  - Easy to predict
  - Iterate over the output list, find the highest scoring one

Recipe for multiclass classification

- Train weights so that it scores examples correctly

  e.g., for an input of type $book$, we want

$$\text{score}(\mathbf{x}, book) > \text{score}(\mathbf{x}, penguin)$$
$$\text{score}(\mathbf{x}, book) > \text{score}(\mathbf{x}, dog)$$

- Prediction: $\underset{\mathbf{y} \in \{book, dog, penguin\}}{\text{argmax}} \text{score}(\mathbf{x}, \mathbf{y})$
  - Easy to predict
  - Iterate over the output list, find the highest scoring one

What if the space of outputs is much larger?
Say trees, or in general, graphs. Let's look at examples.

Example 1: Information extraction

Predicting entities and relations from text

Colin went back home to Ordon Village.

Entities can be `Person`, `Location`, `Organization`, `None`

Directed edges between them can be `Kill`, `LiveIn`, `WorkFor`, `LocatedAt`, `OrgBasedIn`, `None`

Example 1: Information extraction

Predicting entities and relations from text

Colin went back home to Ordon Village.

Colin is a `Person`

Ordon Village is a `Location`

Colin → OrdonVillage: `LiveIn`

Ordon Village → Colin `None`

Prediction: a structure

Structured prediction…

… requires an exploration of the combinatorial space of possible outputs to find the best one

Colin is a `Location`
Ordon Village is a `Organization`
Colin → OrdonVillage: `LiveIn`
Ordon Village → Colin `LiveIn`

Colin is a `Person`
Ordon Village is a `Organization`
Colin → OrdonVillage: `WorkFor`
Ordon Village → Colin `LiveIn`

Colin is a `Person`
Ordon Village is a `Location`
Colin → OrdonVillage: `LiveIn`
Ordon Village → Colin `None`

Colin is a `Organization`
Ordon Village is a `Person`
Colin → OrdonVillage: `Kill`
Ordon Village → Colin `WorkFor`

Colin is a `Person`

Colin is a `Person`

Example 2: Semantic Role Labeling

The task: Given a sentence, identify who does what to whom, where and when.

The bus was ***heading*** for Nairobi in Kenya

Example 2: Semantic Role Labeling

The task: Given a sentence, identify who does what to whom, where and when.

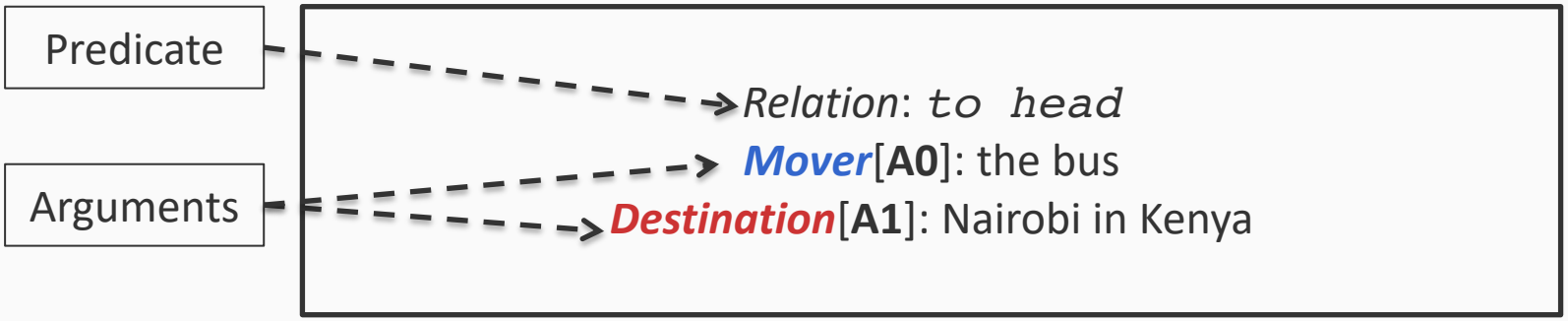
The bus was ***heading*** for Nairobi in Kenya

Relation: `to head`
Mover[**A0**]: the bus
Destination[**A1**]: Nairobi in Kenya

Example 2: Semantic Role Labeling

The task: Given a sentence, identify who does what to whom, where and when.

The bus was ***heading*** for Nairobi in Kenya



Predicting verb arguments

The bus was **heading** for Nairobi in Kenya.

Predicting verb arguments

The bus was **heading** for Nairobi in Kenya.

1. Identify candidate arguments for verb using parse tree
   - Filtered using a binary classifier

Predicting verb arguments

1. Identify candidate arguments for verb using parse tree
   - Filtered using a binary classifier
2. Classify argument candidates
   - Multi-class classifier (one of multiple labels per candidate)

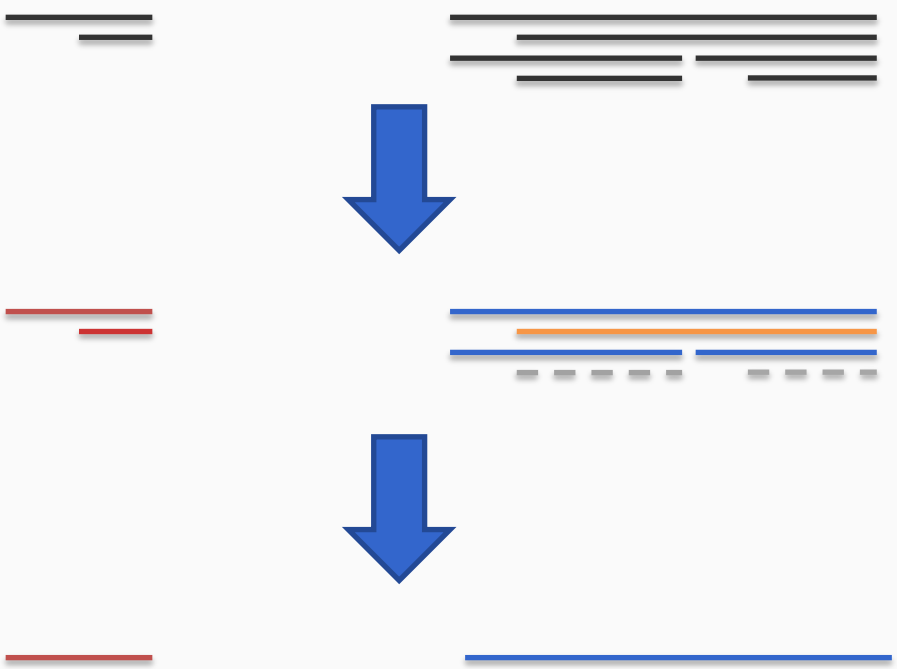
The bus was **heading** for Nairobi in Kenya.

Each color is a different label here

Predicting verb arguments

1. Identify candidate arguments for verb using parse tree
   - Filtered using a binary classifier
2. Classify argument candidates
   - Multi-class classifier (one of multiple labels per candidate)
3. Inference
   - Using probability estimates from argument classifier
   - Must respect structural and linguistic constraints
     - Eg: The same word can not be part of two arguments

The bus was **heading** for Nairobi in Kenya.



Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.

Suppose we are assigning colors to each span

Special label, meaning "Not an argument"

Inference: verb arguments

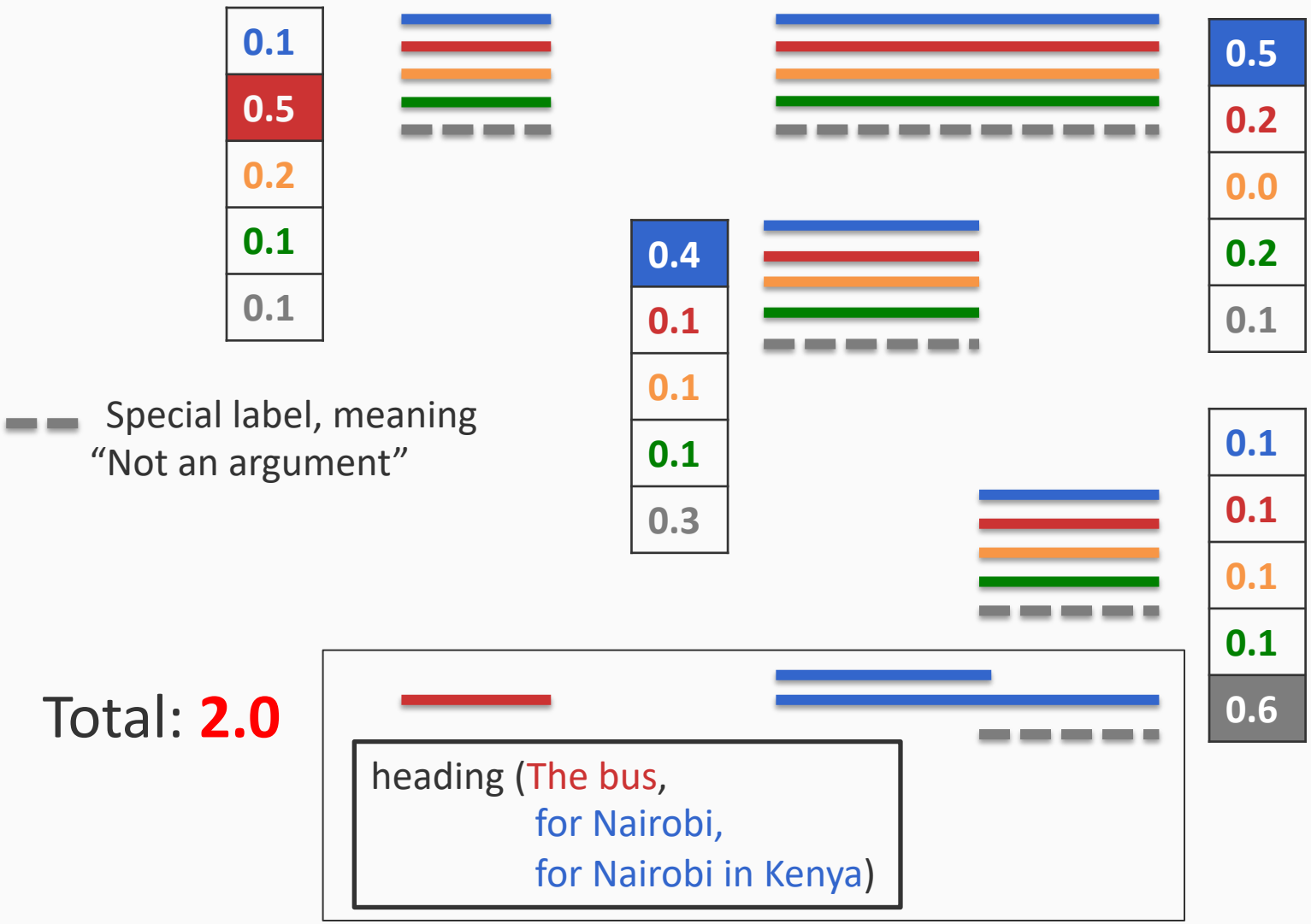
The bus was **heading** for Nairobi in Kenya.

Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.

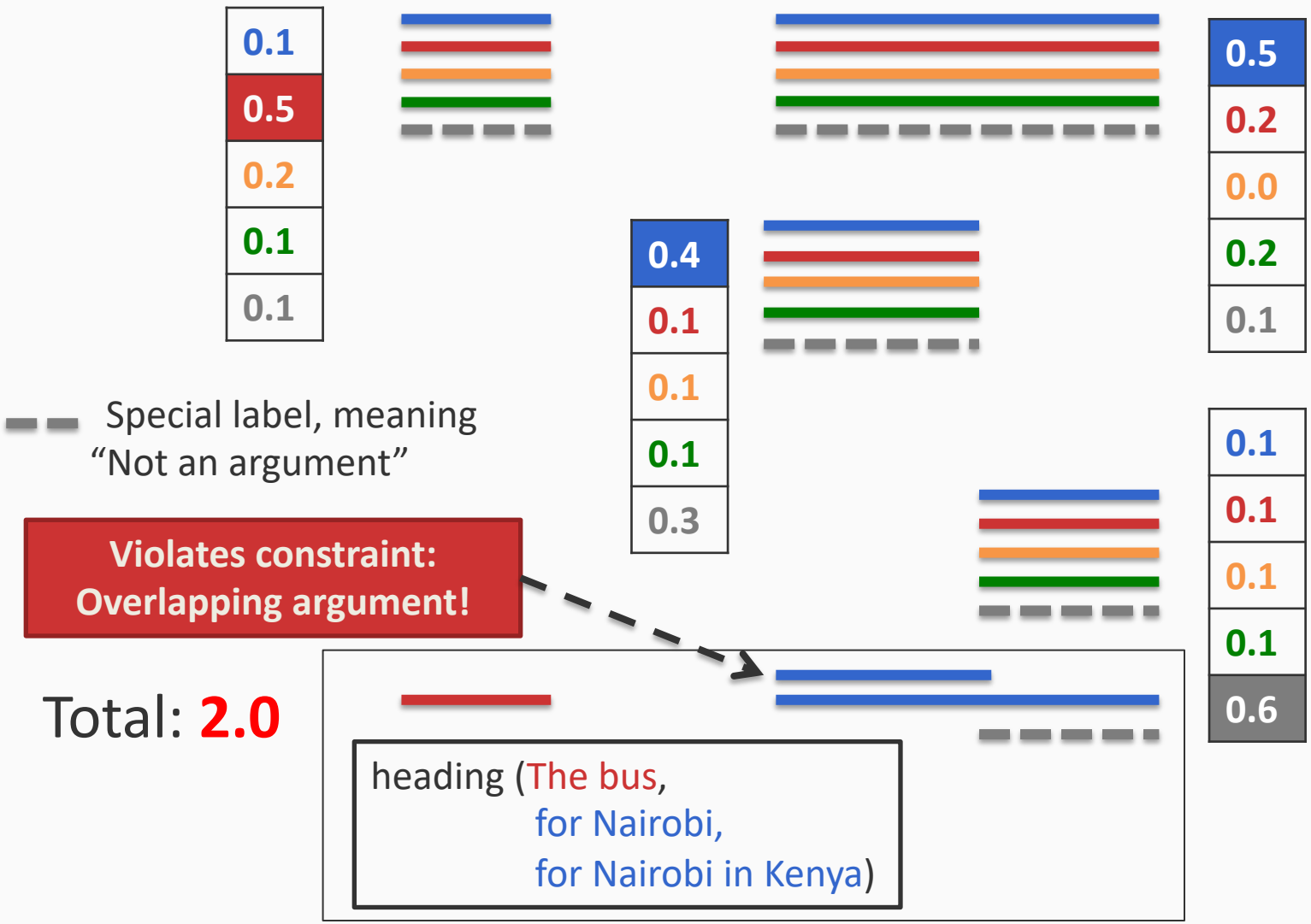
Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.



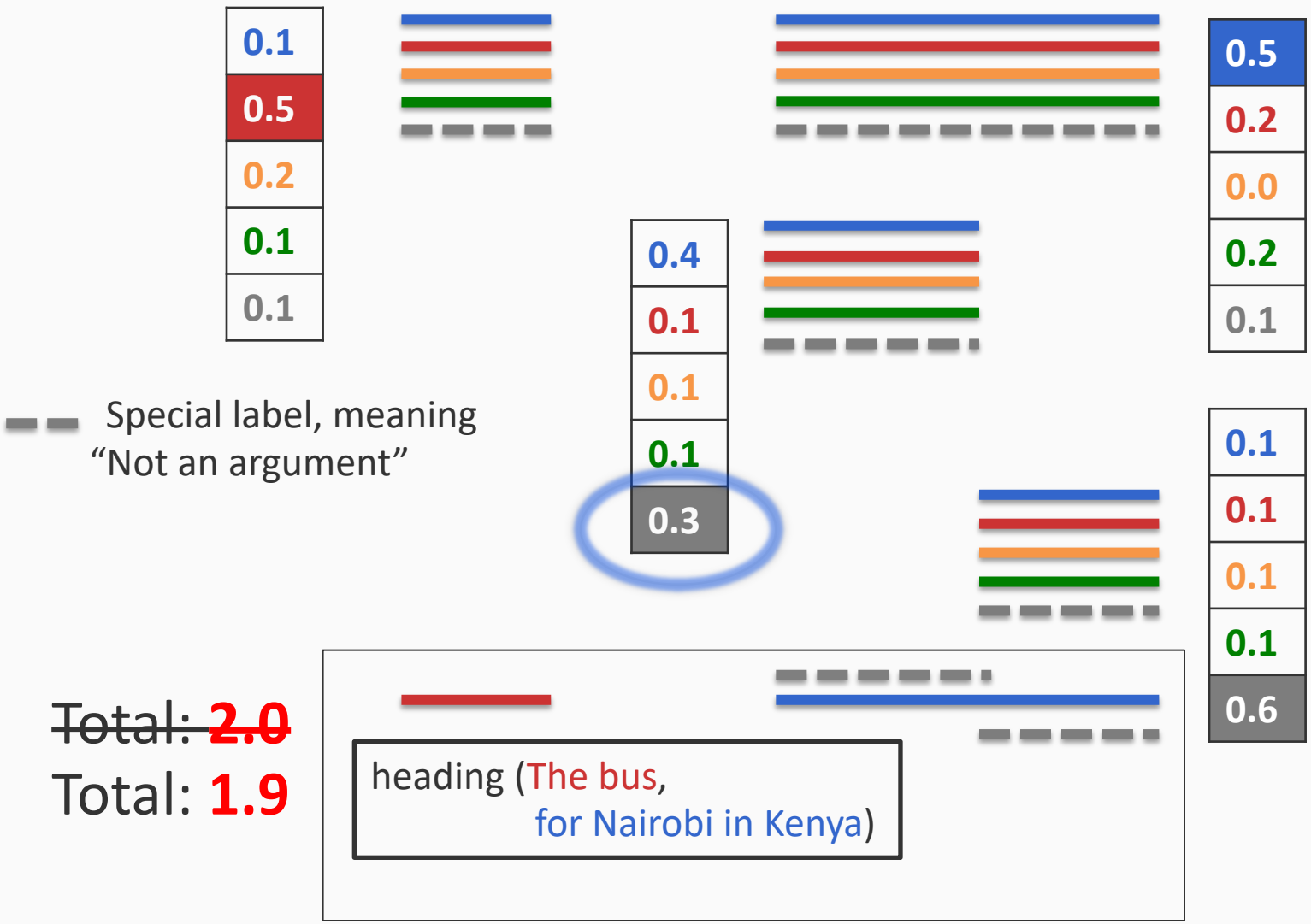
Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.



Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.



Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.

Input $\mathbf{x}$ Text with pre-processing

Output Five possible decisions for each candidate (●●●●●)
Create a binary variable for each decision, only one of which is **true** for each candidate. Collectively, a "structure"

$$\mathbf{y} = \{y^1, y^2, \cdots, y^n\} \in \{0,1\}^n$$

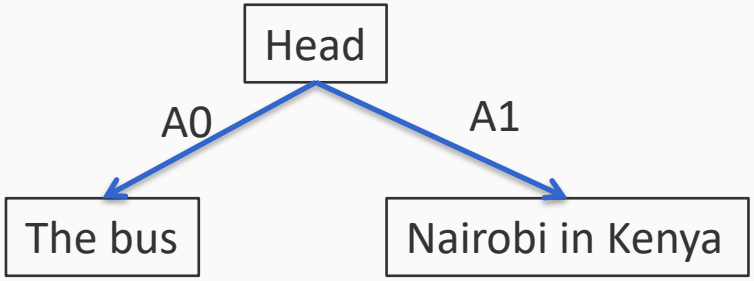
heading (The bus,
for Nairobi in Kenya)

Structured output is…

- A *data structure* with a pre-defined schema
  - Eg: SRL converts raw text into a record in a database

Predicate	A0	A1	Location
Head	The bus	Nairobi in Kenya	-

- Equivalently, a *graph*
  - Often restricted to be a specific family of graphs: chains, trees, etc



Questions/comments?

Example 3: Object detection

Photo by Andrew Dressel - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0

Example 3: Object detection

Photo by Andrew Dressel - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0

Example 3: Object detection

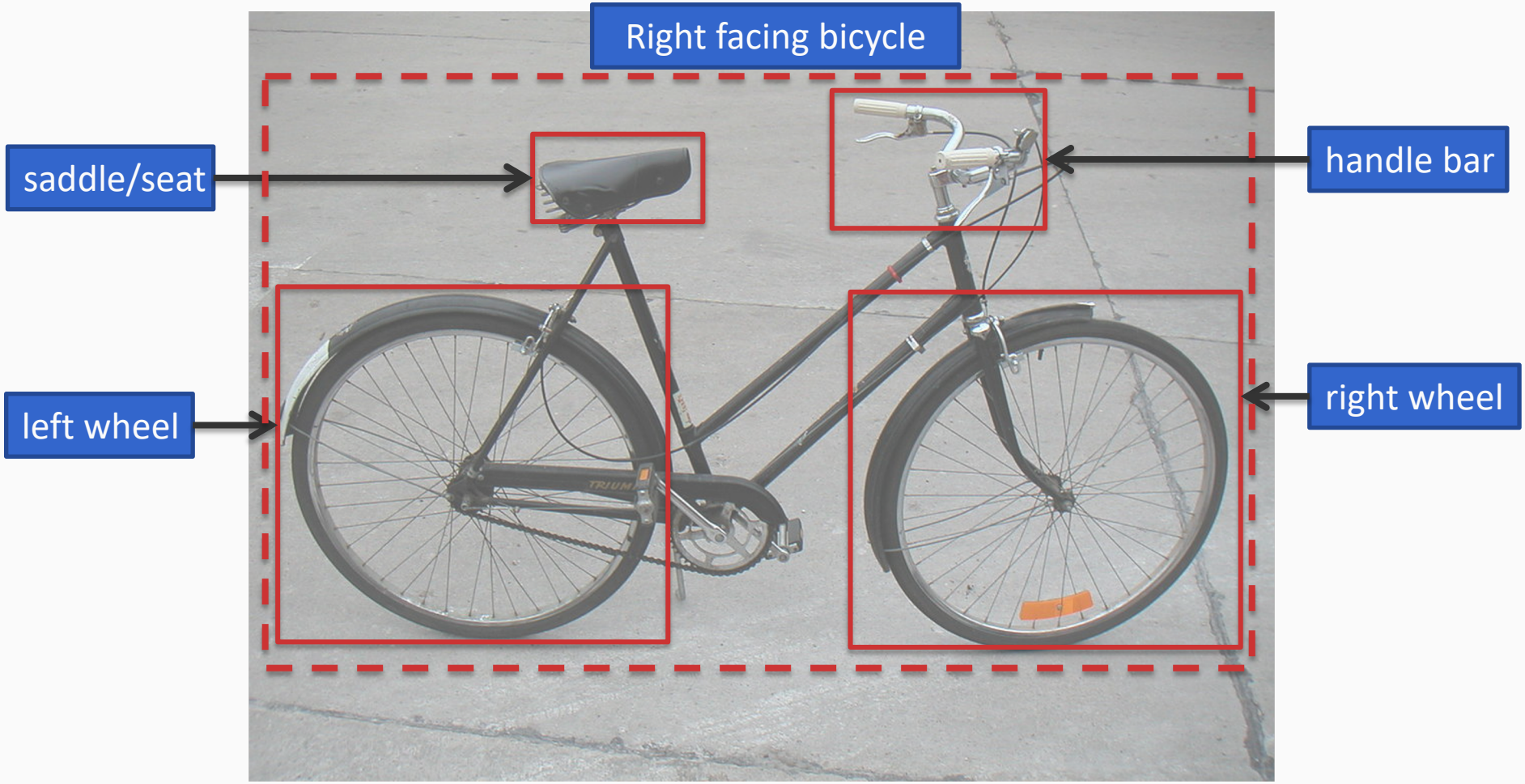
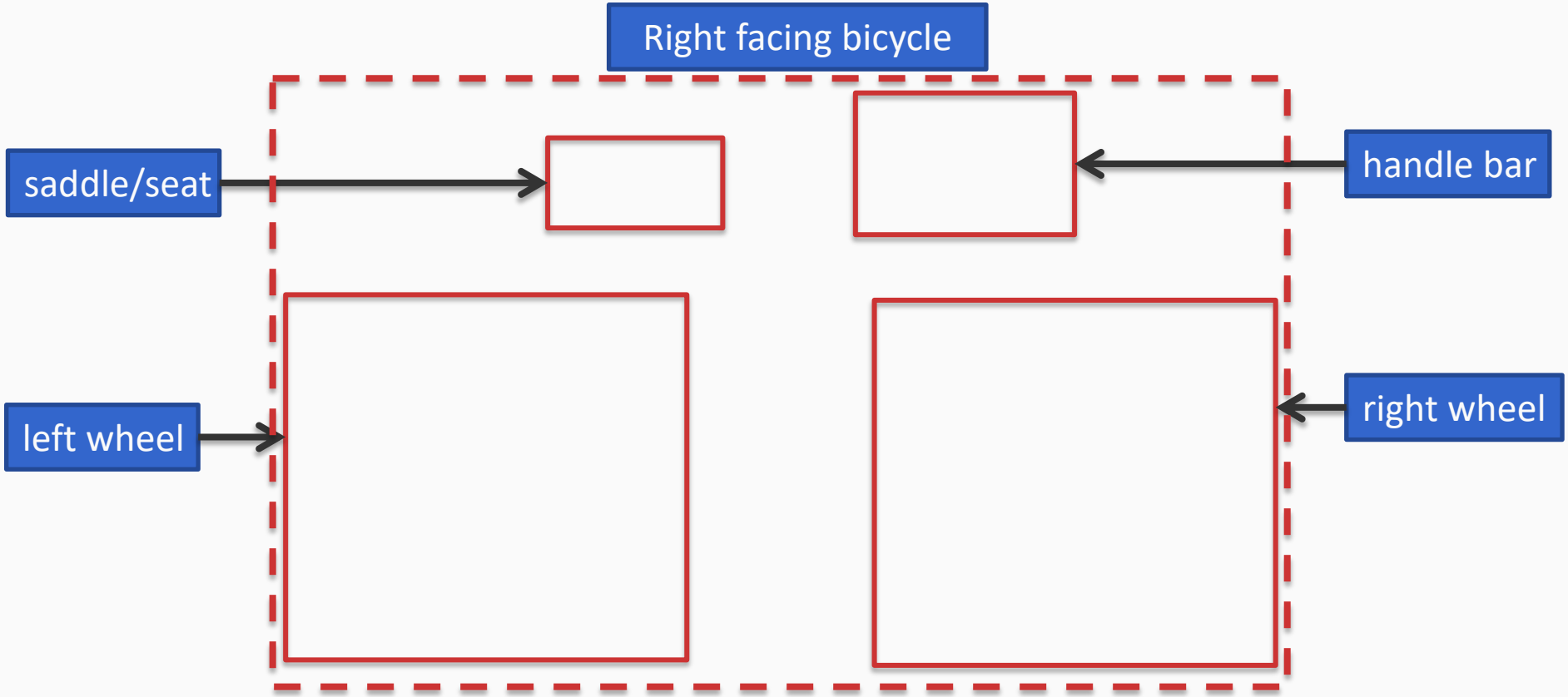


Photo by Andrew Dressel - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0

The output: A schematic showing the parts and their relative layout



Once again, a structure

Object detection

How would you design a predictor that labels all the parts using the tools we have seen so far?

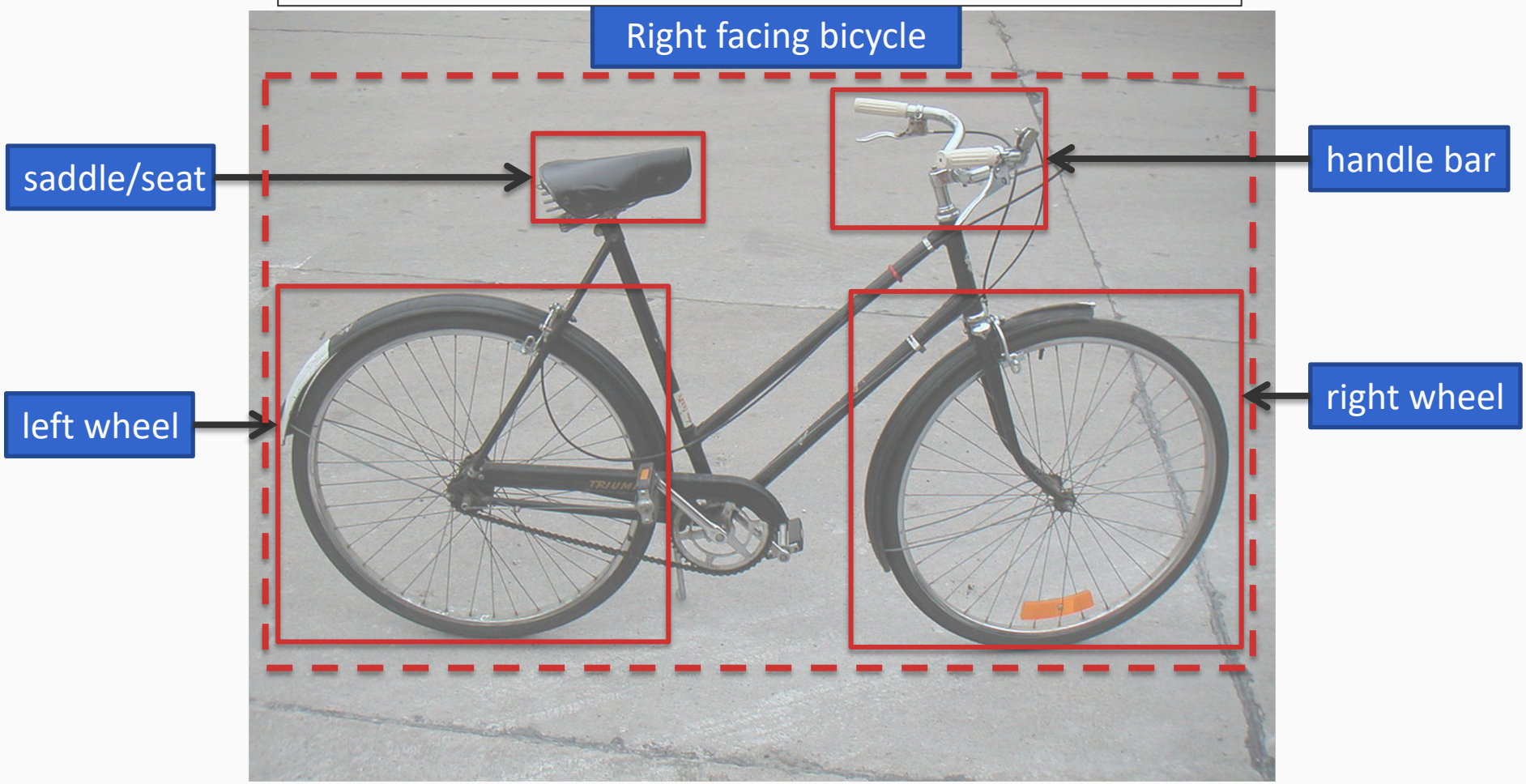


Photo by Andrew Dressel - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0

One approach to build this structure

Left wheel detector: Is there a wheel in this box? Binary classifier

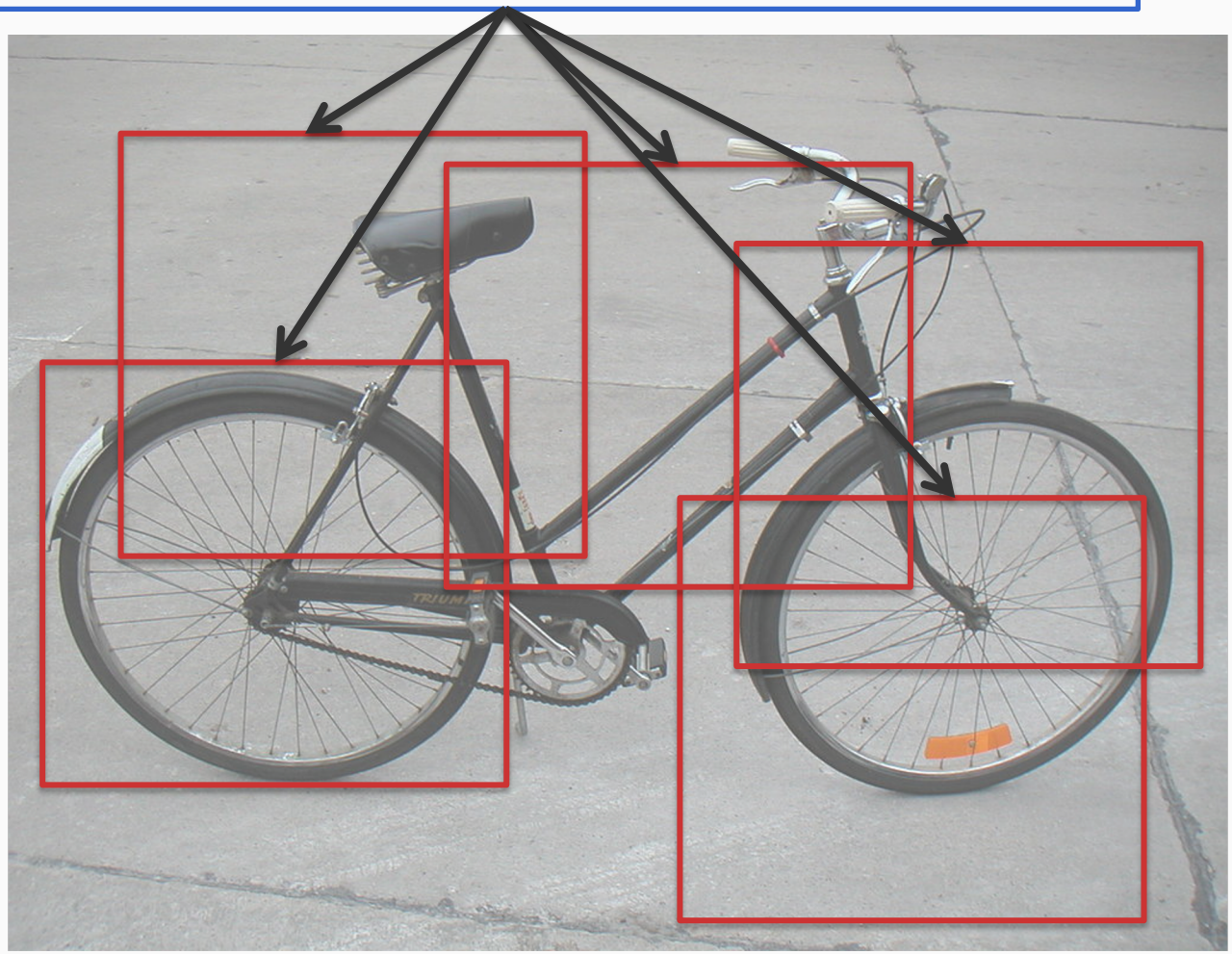


Photo by Andrew Dressel - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0

One approach to build this structure

Handle bar detector: Is there a handle bar in this box? Binary classifier

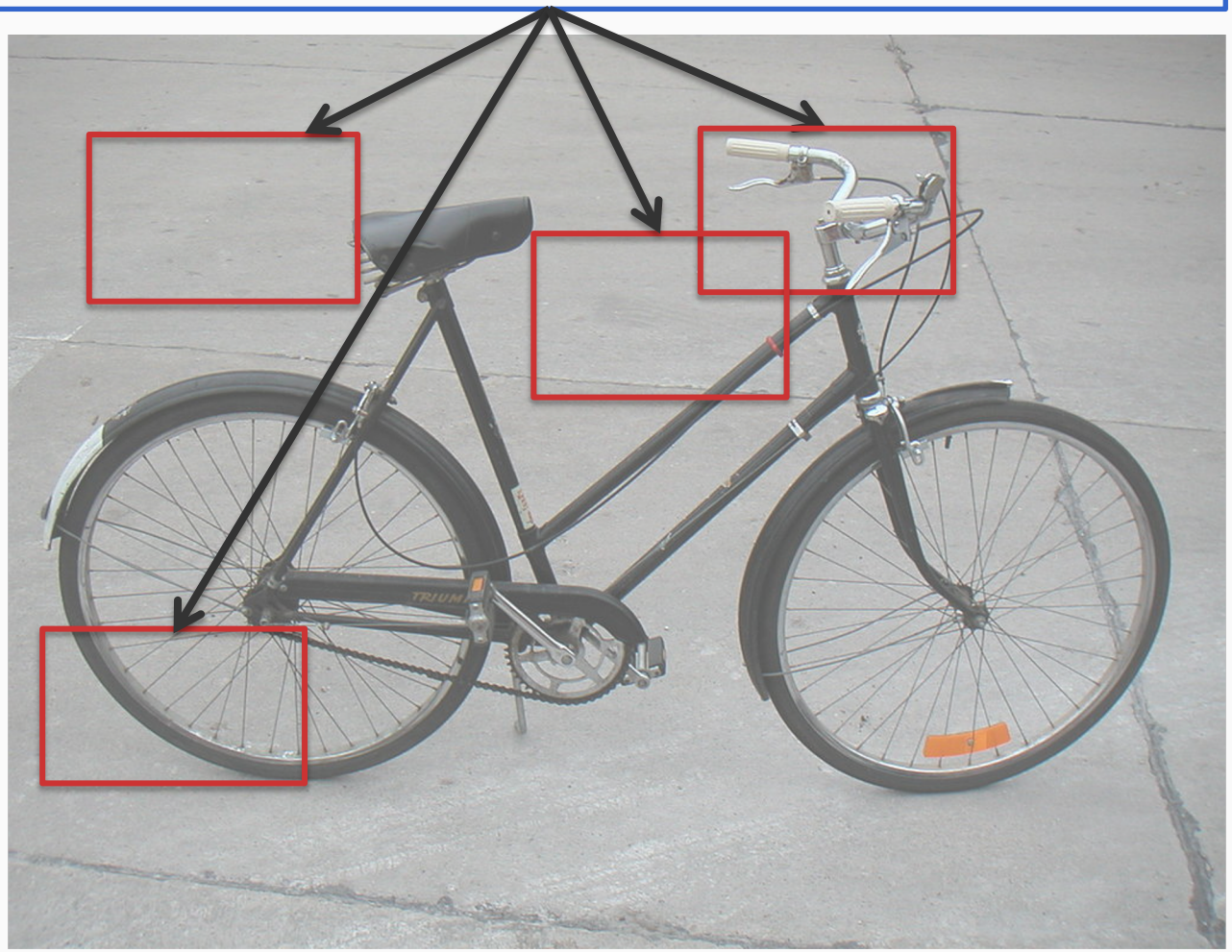


Photo by Andrew Dressel - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0

One approach to build this structure

1. Left wheel detector

2. Right wheel detector

3. Handle bar detector

4. Seat detector

Photo by Andrew Dressel - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0

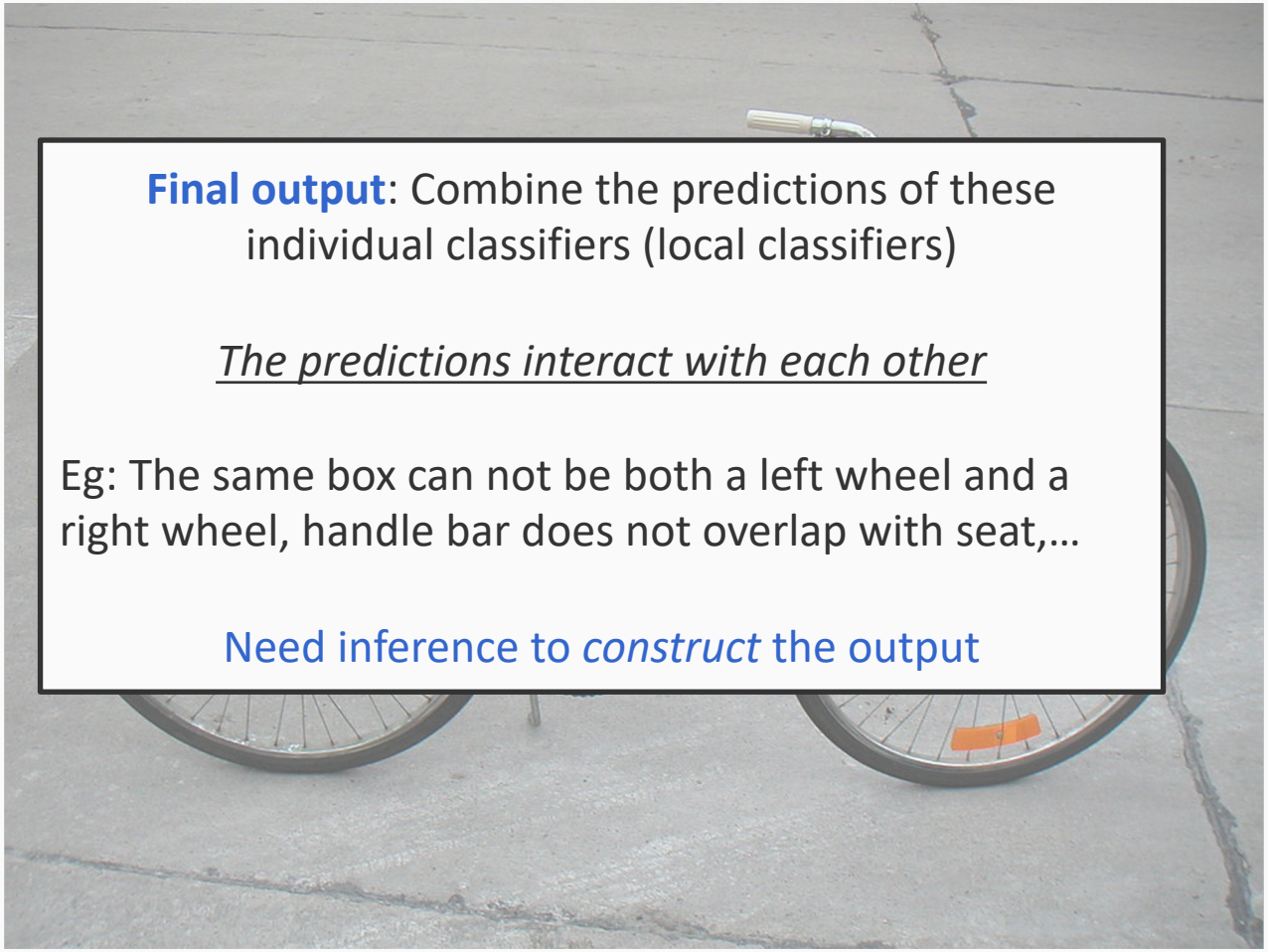
One approach to build this structure

1. Left wheel detector

2. Right wheel detector

3. Handle bar detector

4. Seat detector



Final output: Combine the predictions of these individual classifiers (local classifiers)

The predictions interact with each other

Eg: The same box can not be both a left wheel and a right wheel, handle bar does not overlap with seat,…

Need inference to *construct* the output

Photo by Andrew Dressel - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0

Example 4: Sequence labeling

More on this in next lecture

- Input: A sequence of *tokens* (like words)
- Output: A sequence of labels of same length as input

Eg: Part-of-speech tagging:

Given a sentence, find parts-of-speech of all the words

Example 4: Sequence labeling

- Input: A sequence of *tokens* (like words)
- Output: A sequence of labels of same length as input

Eg: Part-of-speech tagging:

Given a sentence, find parts-of-speech of all the words

The	Fed	raises	interest	rates

Example 4: Sequence labeling

- Input: A sequence of *tokens* (like words)
- Output: A sequence of labels of same length as input

Eg: Part-of-speech tagging:

Given a sentence, find parts-of-speech of all the words

The	Fed	raises	interest	rates
`Determiner`	`Noun`	`Verb`	`Noun`	`Noun`

Example 4: Sequence labeling

- Input: A sequence of *tokens* (like words)
- Output: A sequence of labels of same length as input

Eg: Part-of-speech tagging:

Given a sentence, find parts-of-speech of all the words

The	Fed	raises	interest	rates
`Determiner`	`Noun`	`Verb`	`Noun`	`Noun`

Other labels are possible in different contexts

`Verb` (I *fed* the dog)

`Verb` (Poems don't *interest* me)

`Verb` (He *rates* movies online)

Part-of-speech tagging

Given a word, its label depends on :

- The identity and characteristics of the word

  E.g. Raises is a `Verb` because it ends in –es (among other reasons)

- Its grammatical context

  - Fed in “The Fed” is a `Noun` because it follows a `Determiner`
  - Fed in “I fed the..” is a `Verb` because it follows a `Pronoun`

Part-of-speech tagging

Given a word, its label depends on :

- The identity and characteristics of the word
- Its grammatical context

One possible modeling assumption:

Each output label is dependent on its neighbors in addition to the input word

Part-of-speech tagging

Given a word, its label depends on :

- The identity and characteristics of the word
- Its grammatical context

One possible modeling assumption:

Each output label is dependent on its neighbors in addition to the input word

Two kinds of scoring functions for labels

1. Score for label associating with a particular word in context
2. Score for a pair of labels following each other

Part-of-speech tagging

Given a word, its label depends on :

- The identity and characteristics of the word
- Its grammatical context

One possible modeling assumption:

Each output label is dependent on its neighbors in addition to the input word

Two kinds of scoring functions for labels

1. Score for label associating with a particular word in context
2. Score for a pair of labels following each other

What we want: Find a sequence of labels that *maximizes* the sum/product of these scores

More examples

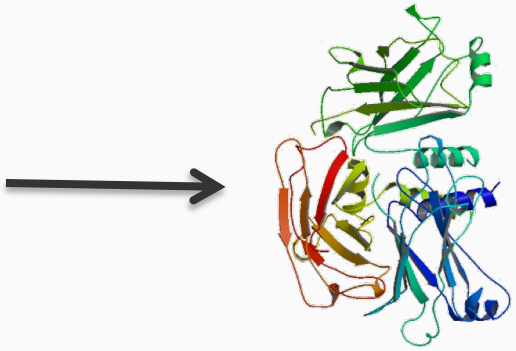
Protein 3D structure prediction

```
AMADLEQKVL EMEASTYDGV FIWKISDFAR KRQEAVAGRI PAIFSPAFYT
 HHHHHHHHH HHHH  SSSE EEEEEESHHH HHHHHHTTSS  EEE   EES

SRYGYKMCLR IYLNGDGTGR GTHLSLFFVV MKGPNDALLR WPFNQKVTLM
STTS EEEEE EETT  GGGT TTEEEEEEEE E  TTGGGS  SS    EEEE

LLDQNNREHV IDAFRPDVTS SSFQRPVNDM NIASGCPLFC PVSKMEAKNS
E  TTSS  E EEEE   TTS GGGS  SSSB    EEEEEEE ETTTTTTSTT

YVRDDAIFIK AIVDLTGL
T  SSEEEEE EEE  TT
```



Inferring layout of a room

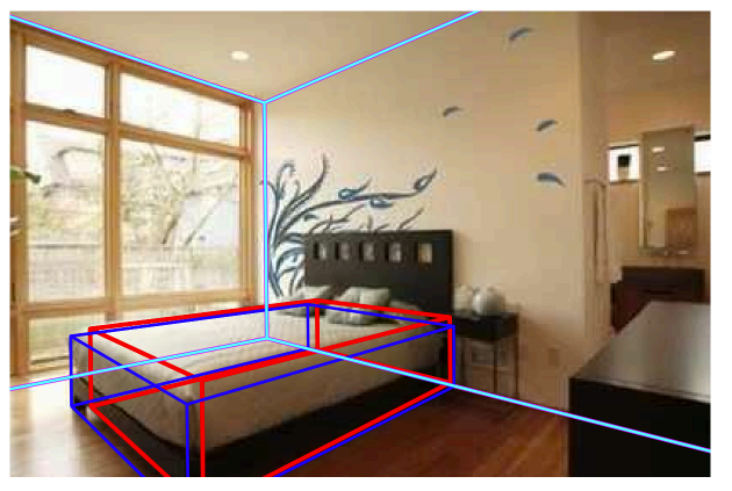


Image from [Schwing et al 2013]

Structured output is…

- A graph, possibly labeled and/or directed
  - Possibly from a restricted family, such as chains, trees, etc.
  - A discrete representation of input
  - Eg. A table, the SRL frame output, a sequence of labels etc

- A collection of inter-dependent decisions
  - Eg: The sequence of decisions used to construct the output
- The result of a combinatorial optimization problem

$$\underset{\mathbf{y} \in \, all \; outputs}{\operatorname{argmax}} \quad score(\mathbf{x}, \mathbf{y})$$

Structured output is…

Representation

- A graph, possibly labeled and/or directed
  - Possibly from a restricted family, such as chains, trees, etc.
  - A discrete representation of input
  - Eg. A table, the SRL frame output, a sequence of labels etc

Procedural

- A collection of inter-dependent decisions
  - Eg: The sequence of decisions used to construct the output
- The result of a combinatorial optimization problem

$$\underset{\mathbf{y} \in \, all \; outputs}{\operatorname{argmax}} \quad score(\mathbf{x}, \mathbf{y})$$

We have seen something similar before in the context of multiclass

Structured output is…

Representation

- A graph, possibly labeled and/or directed
  - Possibly from a restricted family, such as chains, trees, etc.
  - A discrete representation of input
  - Eg. A table, the SRL frame output, a sequence of labels etc

Procedural

- A collection of inter-dependent decisions
  - Eg: The sequence of decisions used to construct the output
- The result of a combinatorial optimization problem

$$\underset{\mathbf{y} \in \, all \; outputs}{\operatorname{argmax}} \quad score(\mathbf{x}, \mathbf{y})$$

We have seen something similar before in the context of multiclass

There are a countable number of graphs
Question: Why can't we treat each output as a label and train/predict as multiclass?

Challenges with structured output

Two challenges

1. We cannot train a separate weight vector for each possible inference outcome
   - For multiclass, we could train one weight vector for each label
2. We cannot enumerate all possible structures for inference
   - Inference for multiclass was easy

Challenges with structured output

Two challenges

1. We cannot train a separate weight vector for each possible inference outcome
   - For multiclass, we could train one weight vector for each label
2. We cannot enumerate all possible structures for inference
   - Inference for multiclass was easy

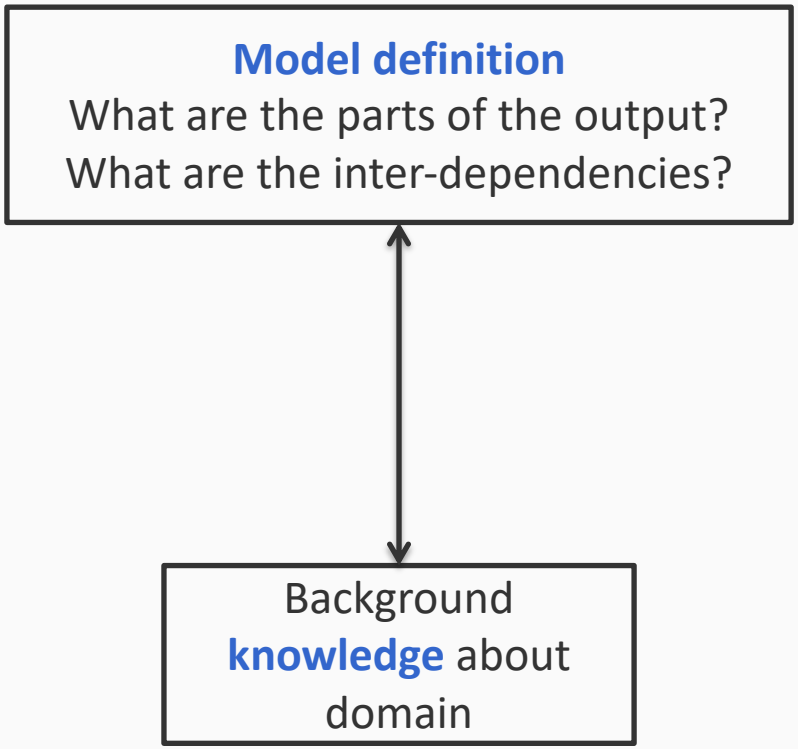
Solution

- Decompose the output into parts that are labeled
- Define
  - how the parts interact with each other
  - how these labeled interacting parts are scored
  - an inference algorithm to assign labels to all the parts so that the whole is meaningful

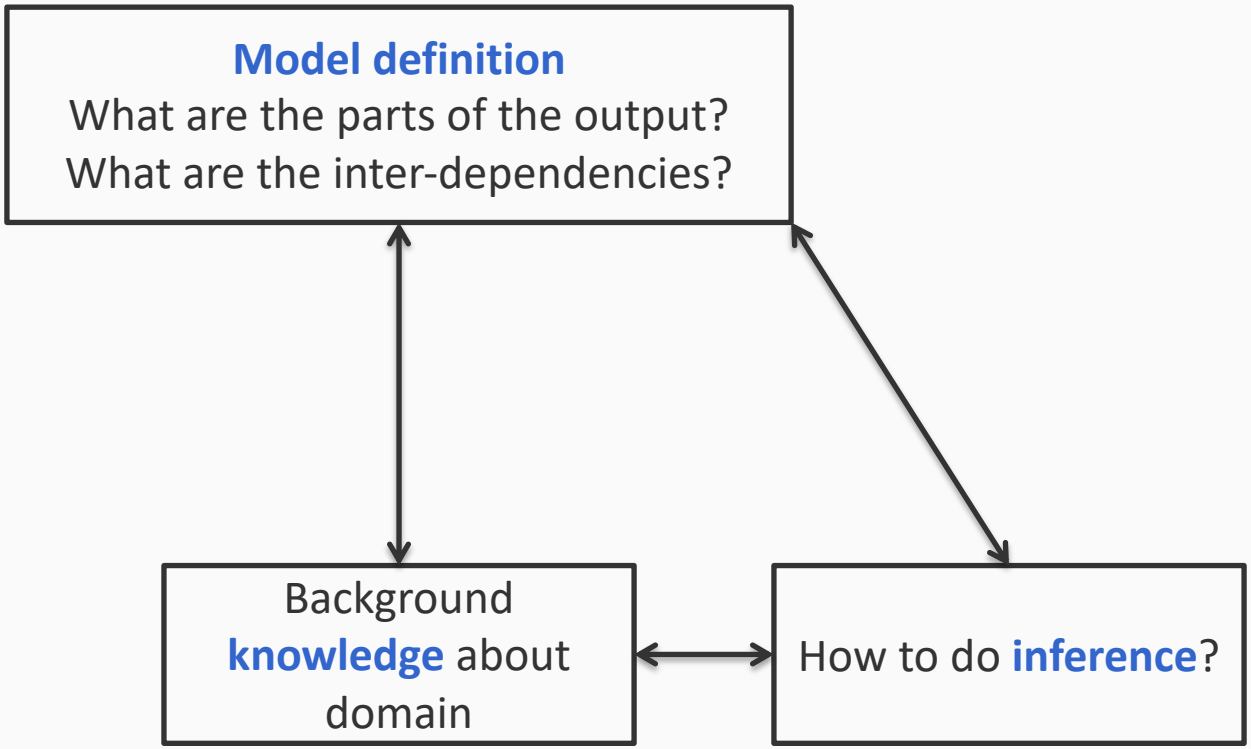
Computational issues

Background
knowledge about
domain

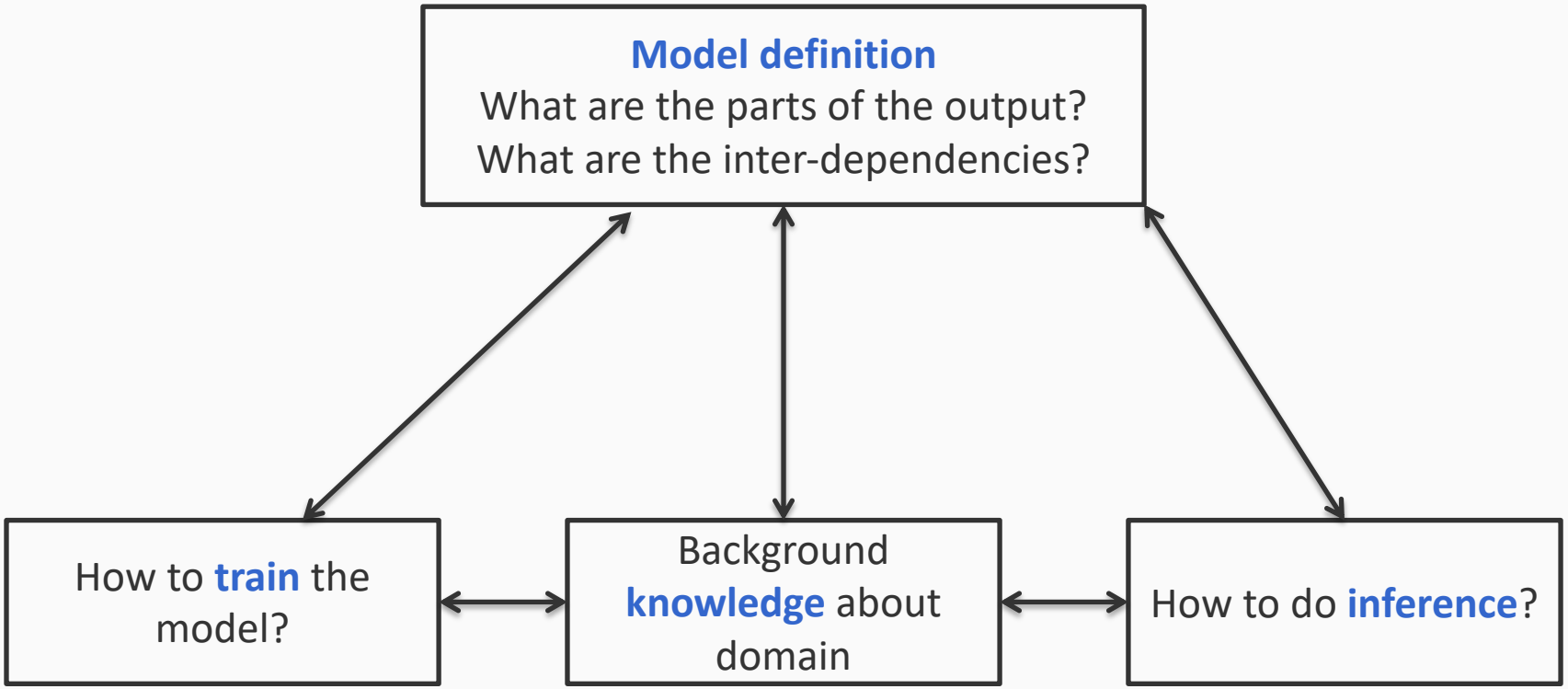
Computational issues



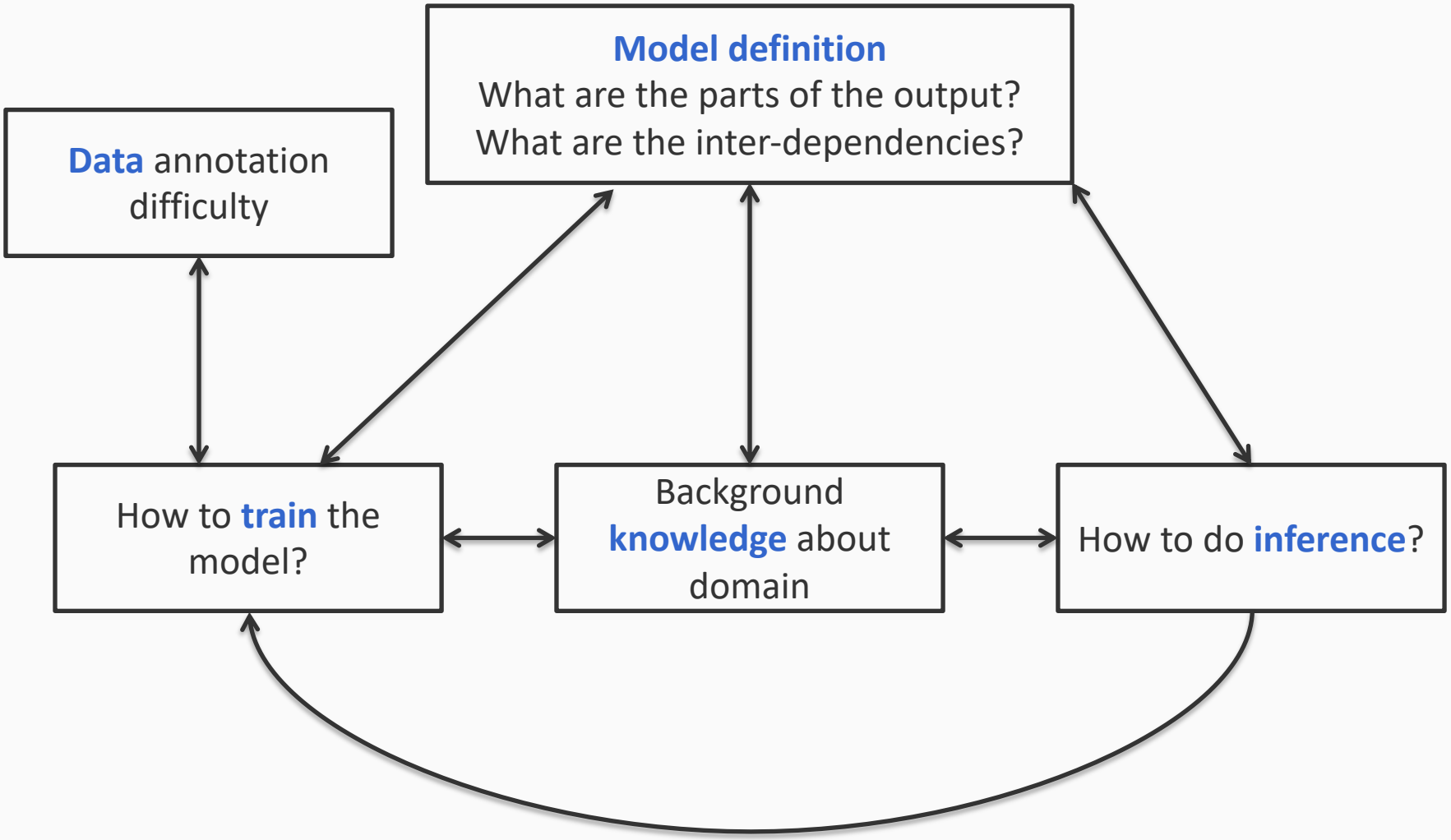
Computational issues



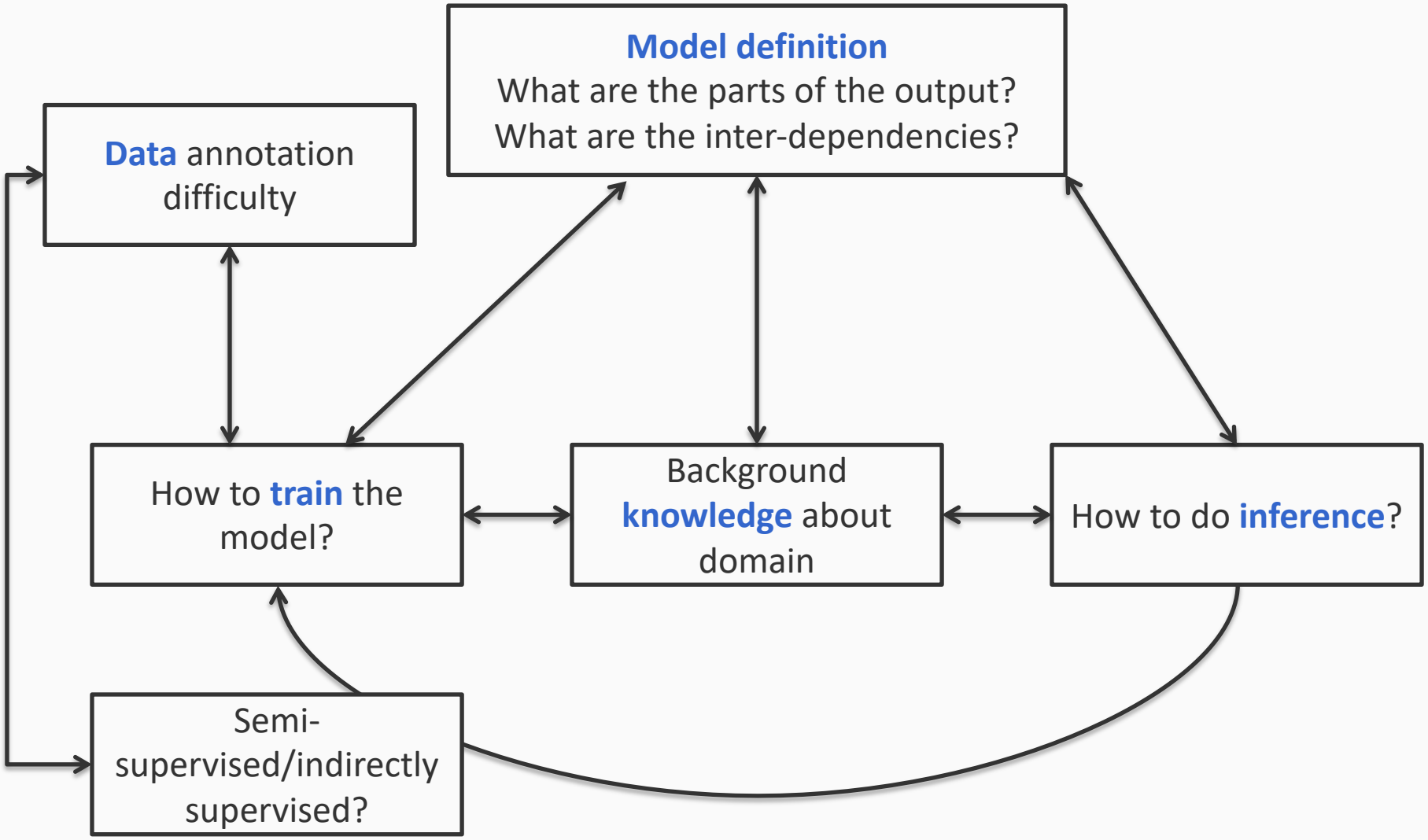
Computational issues



Computational issues



Computational issues



Summary

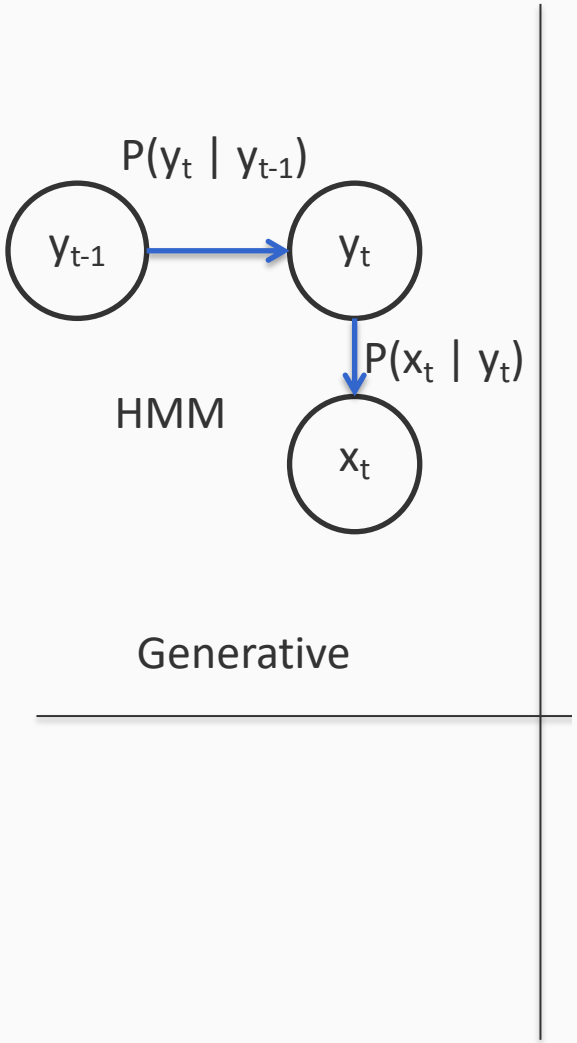
- We saw several examples of structured output
  - Structures are graphs
    - Sometimes useful to think of them as a sequence of decisions
    - Also useful to think of them as data structures
- Multiclass is the simplest type of structure
  - Lessons from multiclass are useful
- Modeling outputs as structures
  - Decomposition of the output, inference, training

Predicting Sequences: Global Models

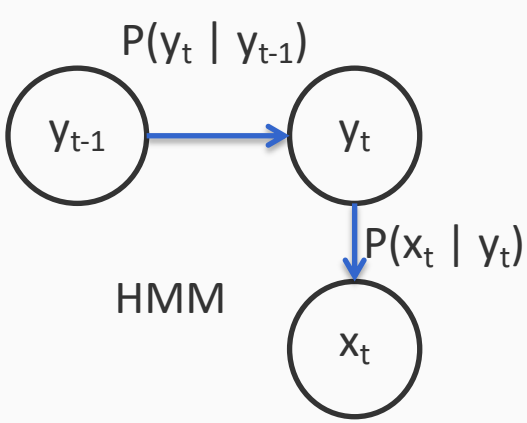
Global models

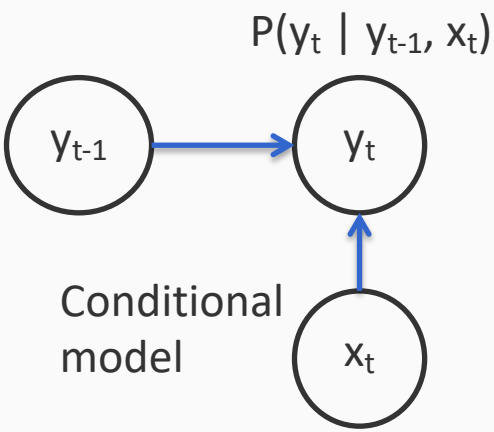
- Train the predictor globally
  - Instead of training local decisions independently
- *Normalize globally*
  - Make each edge in the model undirected
  - Not associated with a probability, but just a “score”
- Recall the difference between local vs. global for multiclass

HMM vs. A local model vs. A global model



HMM vs. *A local model* vs. A global model



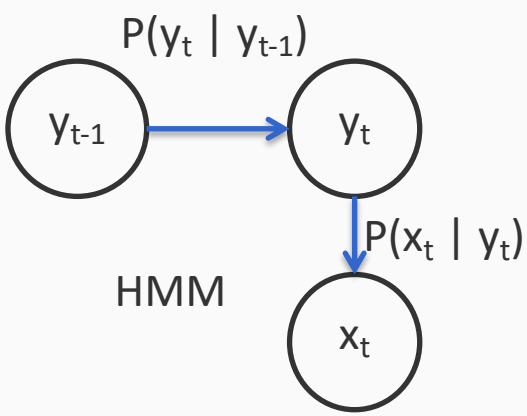


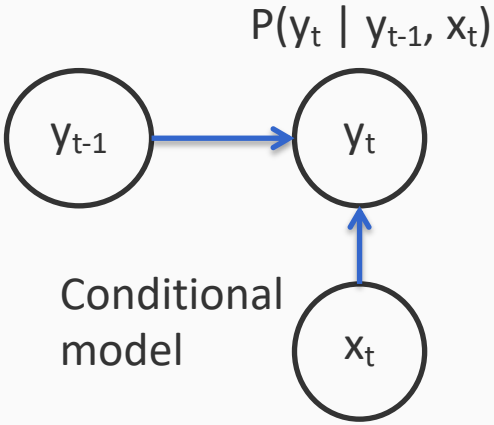
Generative

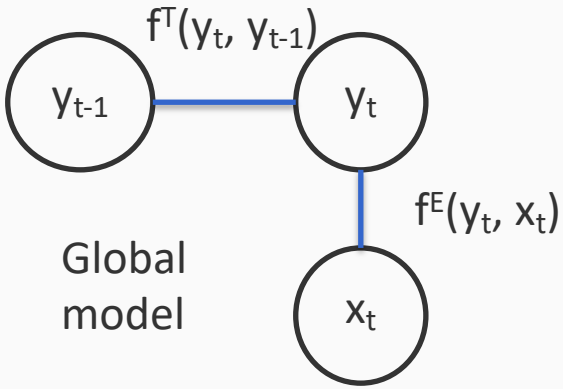
Discriminative

Local: P is locally normalized to add up to one for each step

HMM vs. A local model vs. *A global model*







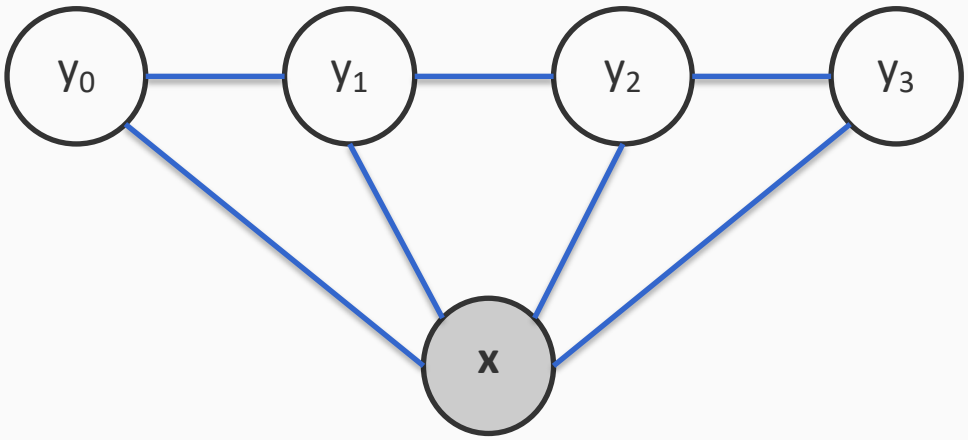
Generative

Discriminative

Local: P is locally normalized to add up to one for each step

Global: The functions f^T and f^E are scores that are not normalized

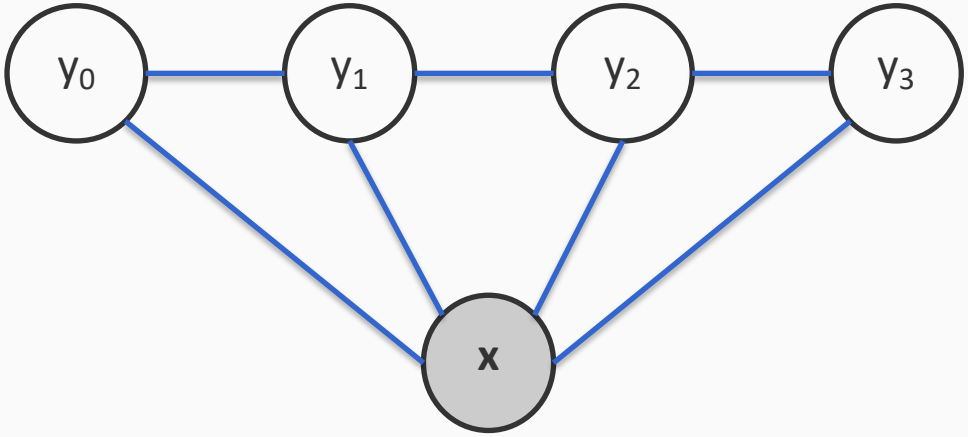
Conditional Random Field



Each node is a random variable

We observe some nodes and the rest are unobserved

Conditional Random Field



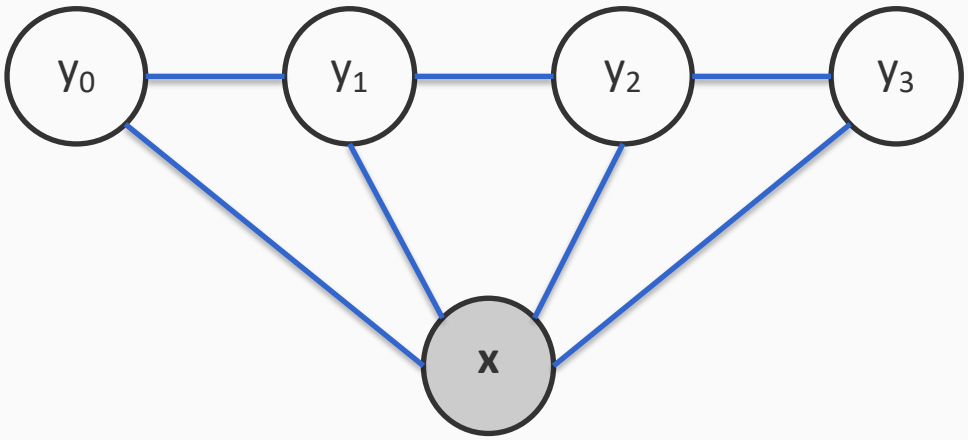
Each node is a random variable

We observe some nodes and the rest are unobserved

For example:

- x could be a random variable representing an input video,
- the y's could represent whether the corresponding time step is at the start, end, or within a scene.

Conditional Random Field



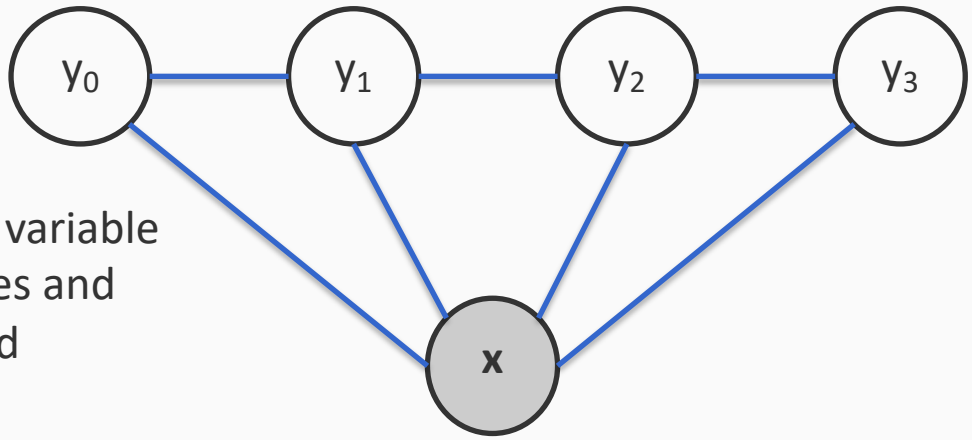
Each node is a random variable

We observe some nodes and the rest are unobserved

The goal: To characterize a probability distribution over the unobserved variables, conditioned on the observed ones.

That is, to characterize $P(y_0, y_1, \cdots \mid \mathbf{x})$.

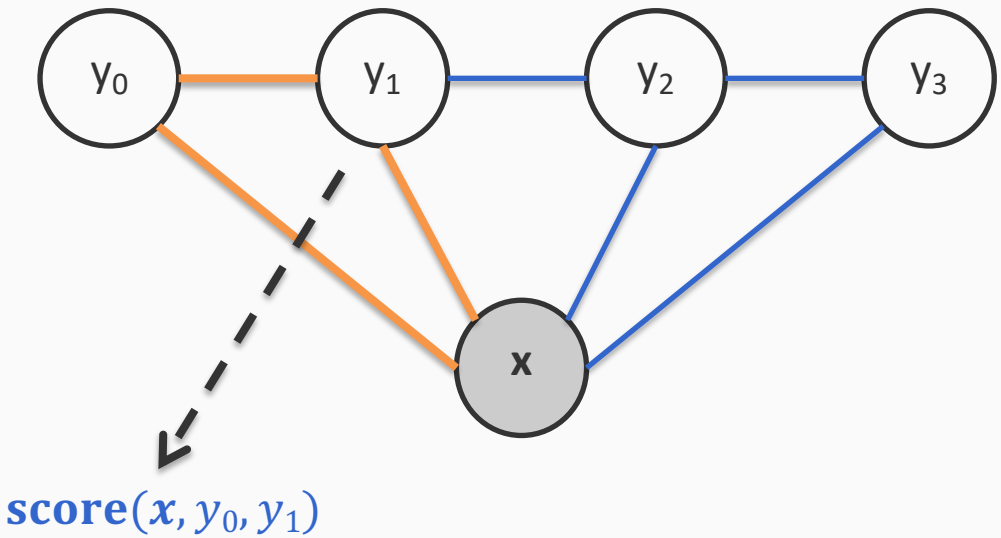
Scoring assignments to outputs



Each node is a random variable
We observe some nodes and
the rest are unobserved

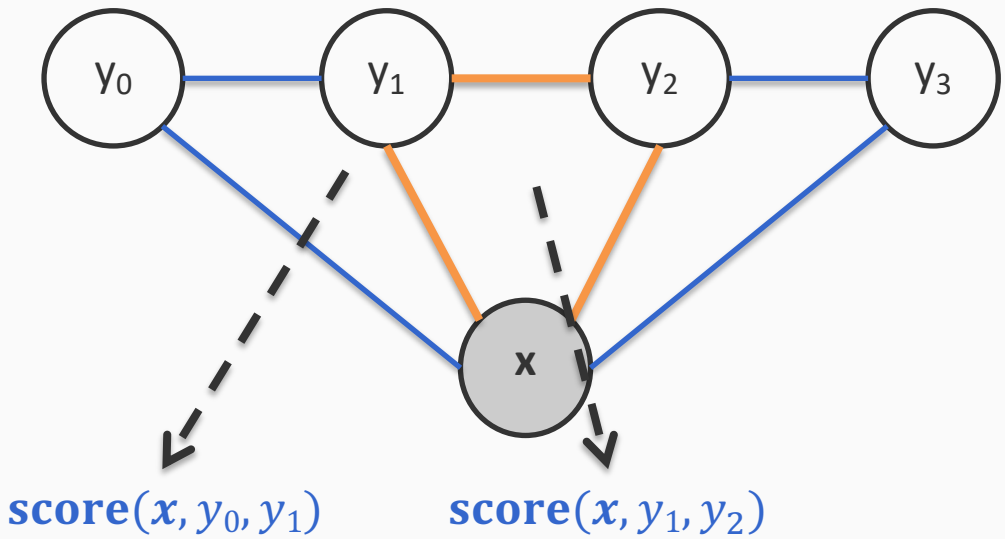
The strategy: Each ***clique*** is associated with a score

Scoring assignments to outputs



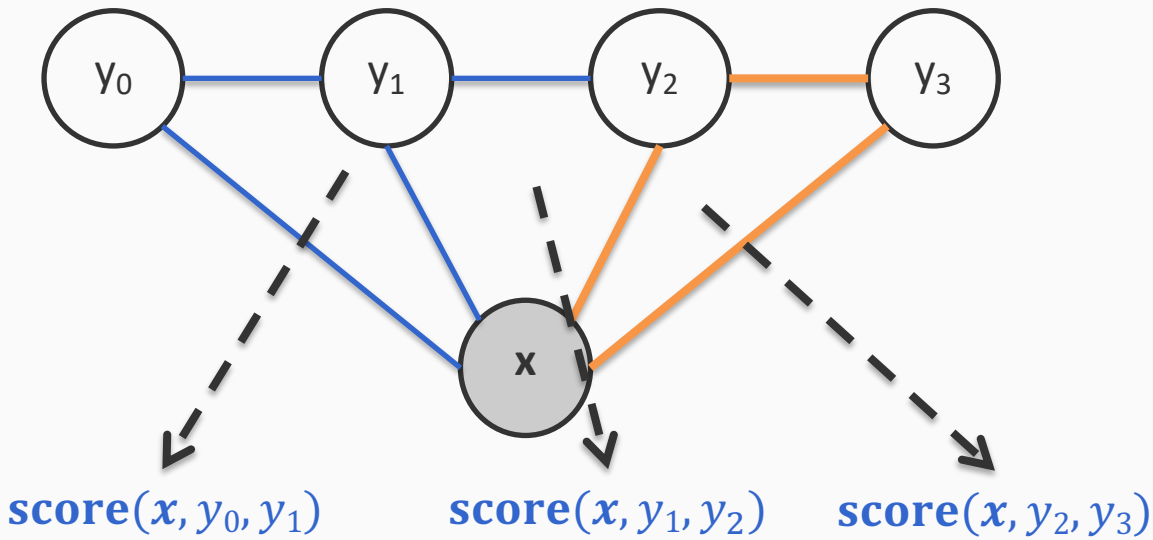
The strategy: Each ***clique*** is associated with a score

Scoring assignments to outputs



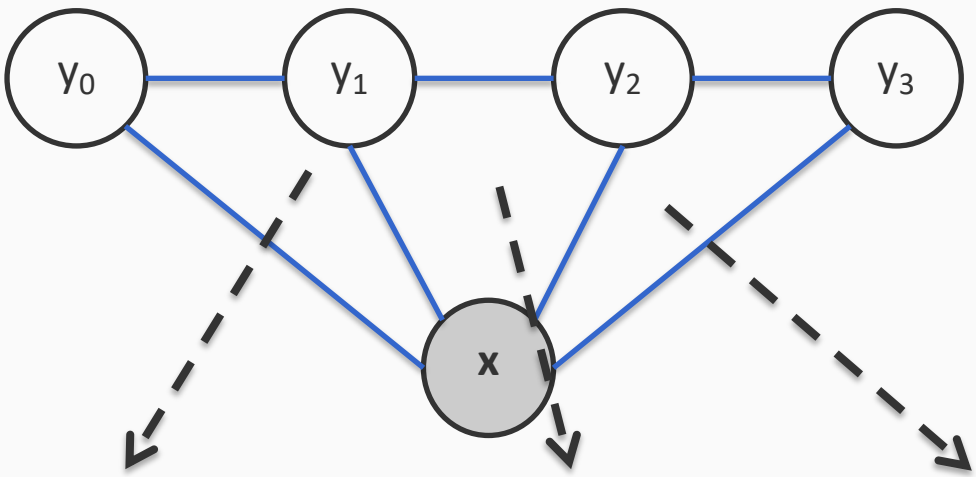
The strategy: Each ***clique*** is associated with a score

Scoring assignments to outputs



The strategy: Each ***clique*** is associated with a score

Scoring assignments to outputs



$\mathbf{score}(\boldsymbol{x}, y_0, y_1) = \mathbf{w}^T \phi(x, y_0, y_1)$

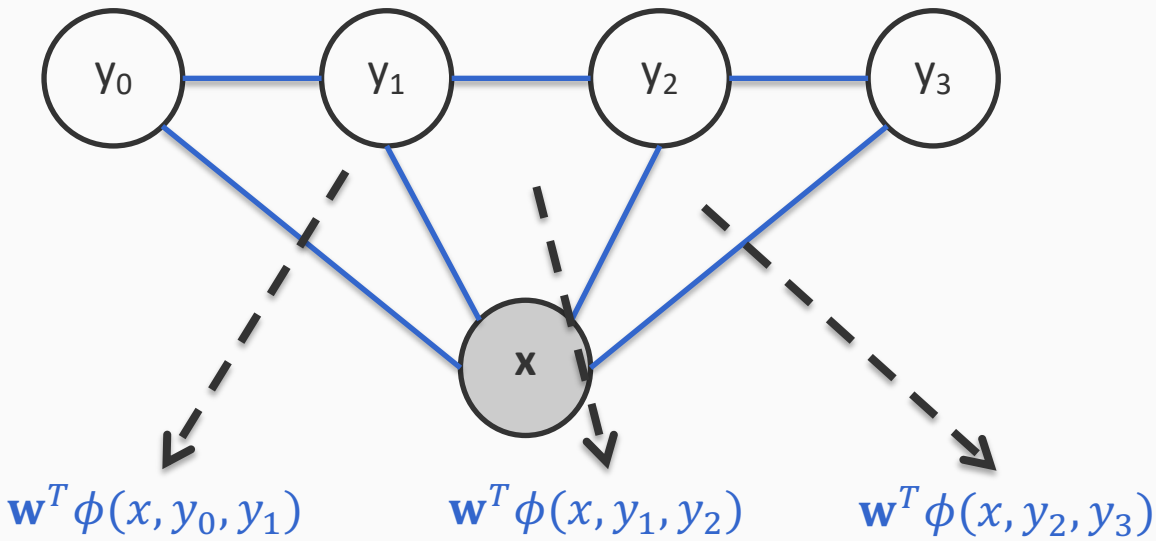
$\mathbf{score}(\boldsymbol{x}, y_1, y_2) = \mathbf{w}^T \phi(x, y_1, y_2)$

$\mathbf{score}(\boldsymbol{x}, y_2, y_3) = \mathbf{w}^T \phi(x, y_2, y_3)$

The strategy: Each ***clique*** is associated with a score

The usual scoring function: A linear function of weights and features of the associated nodes

Scoring assignments to outputs



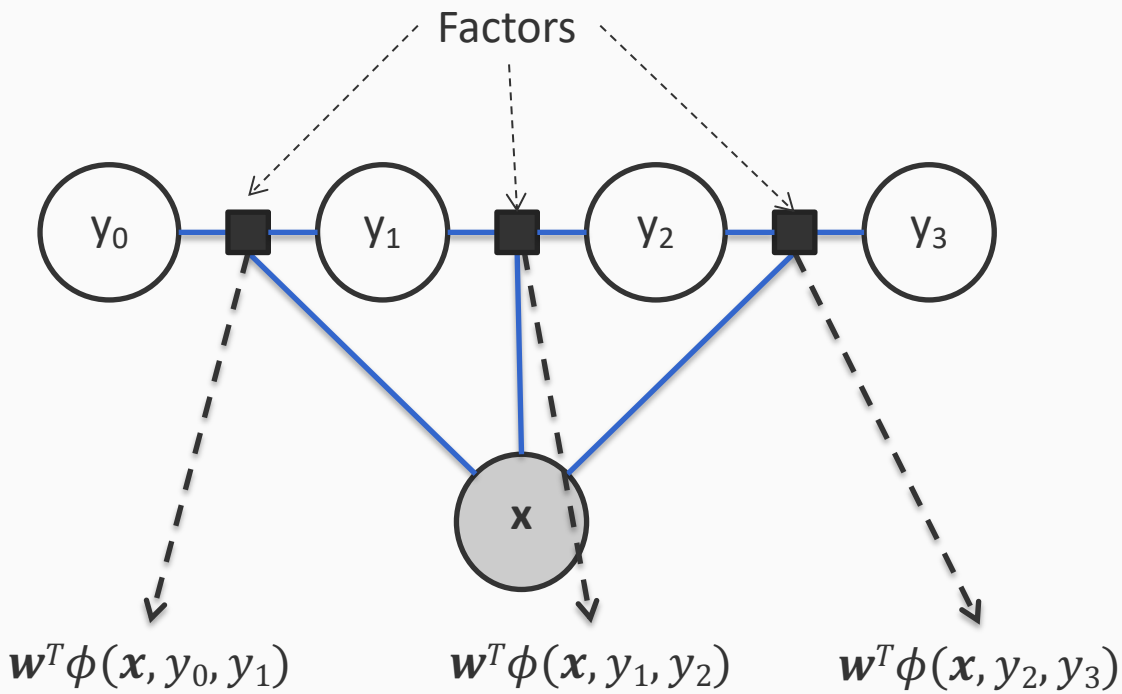
Each node is a random variable

We observe some nodes and need to assign the rest

Each ***clique*** is associated with a score, typically linear

Arbitrary features, as with local conditional models

Another notation: A *factor graph*



Each node is a random variable

We observe some nodes and need to assign the rest

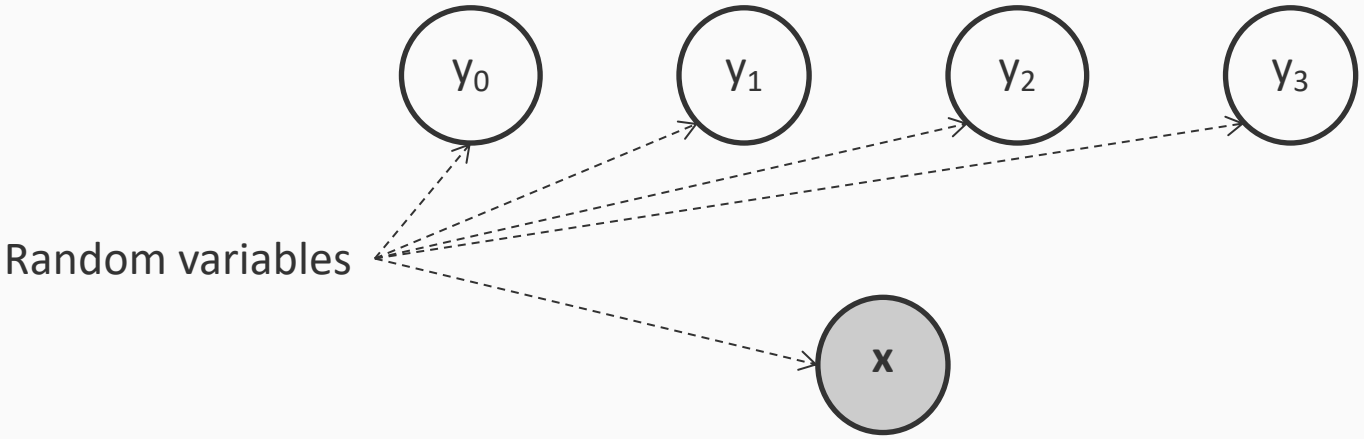
Each ~~clique~~ factor is associated with a score

Notation: A factor graph

- A bipartite graph consisting of two kinds of nodes

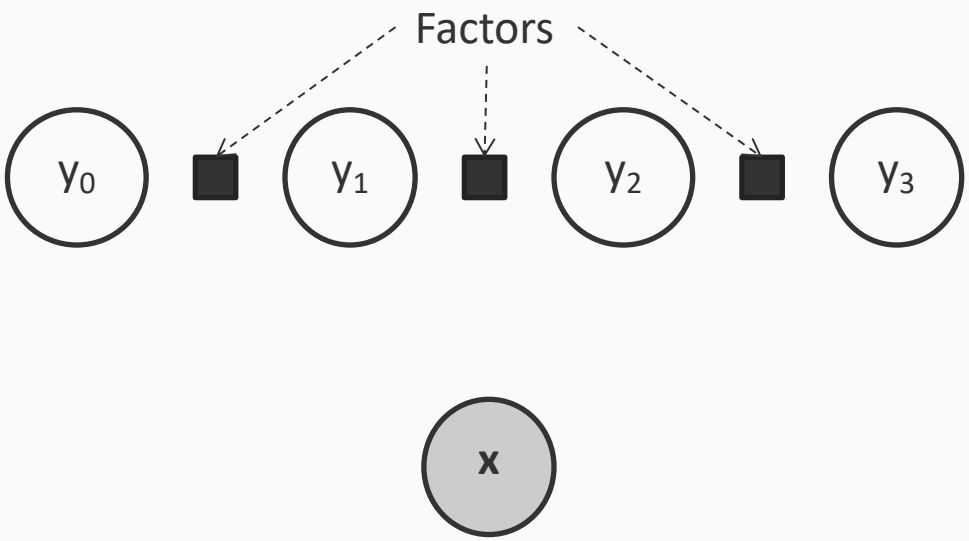
Notation: A factor graph

- A bipartite graph consisting of two kinds of nodes
  - Random variables (usually circles) represent decisions



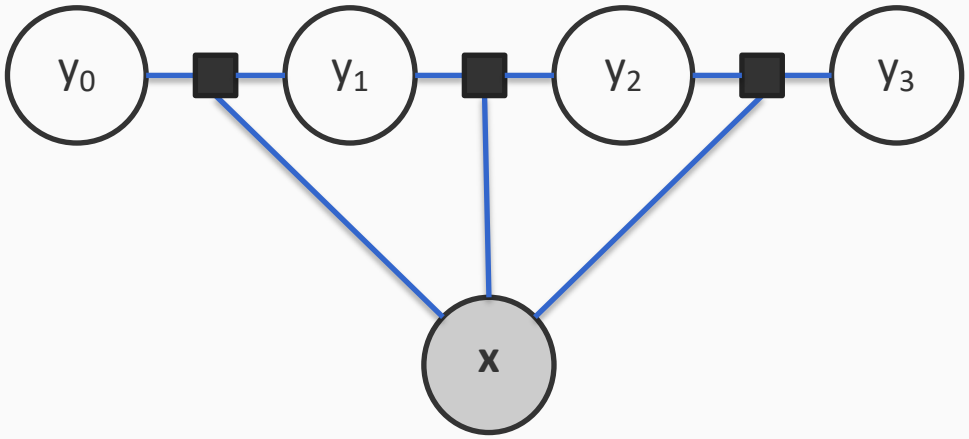
Notation: A factor graph

- A bipartite graph consisting of two kinds of nodes
  - Random variables (usually circles) represent decisions
  - Factors (usually squares) represent interactions



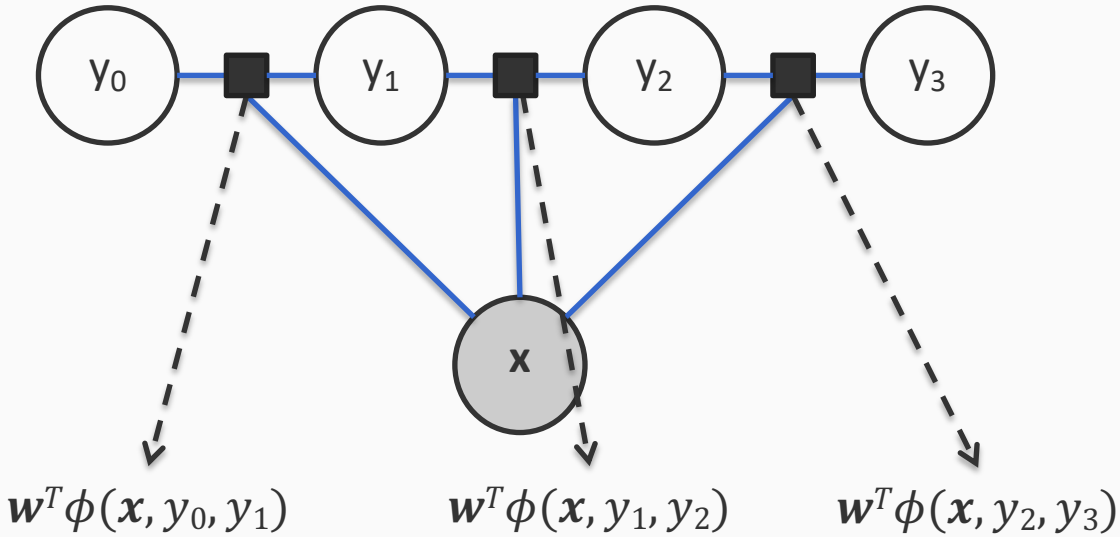
Notation: A factor graph

- A bipartite graph consisting of two kinds of nodes
  - Random variables (usually circles) represent decisions
  - Factors (usually squares) represent interactions
  - Edges: Random variables that interact with each other (think parts)

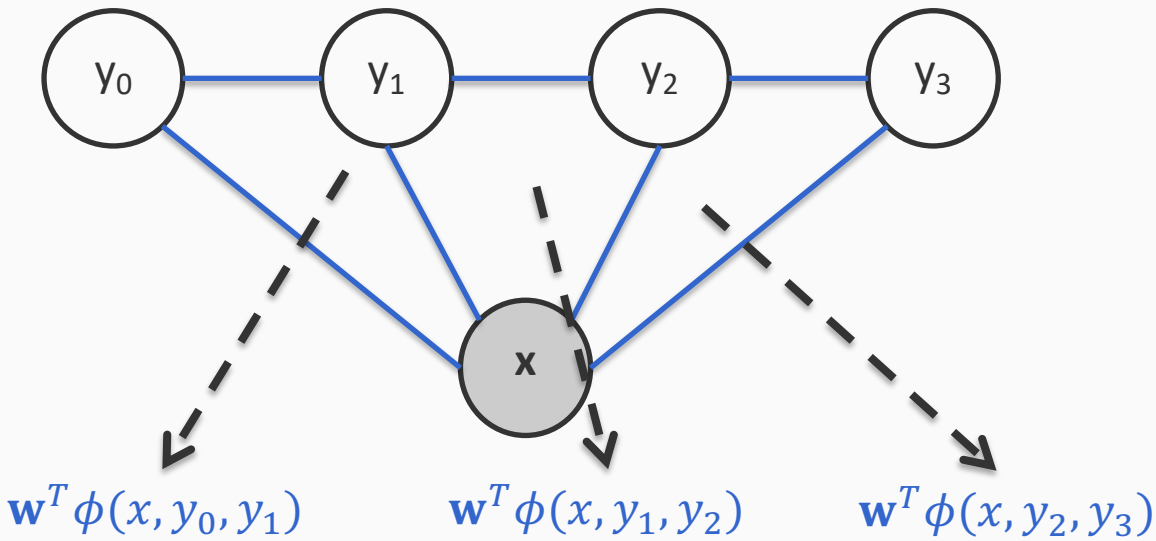


Notation: A factor graph

- A bipartite graph consisting of two kinds of nodes
  - Random variables (usually circles) represent decisions
  - Factors (usually squares) represent interactions
- Semantics: All random variables that are connected to a factor are scored together. *That is, each factor corresponds to a score.*



Scoring assignments to outputs



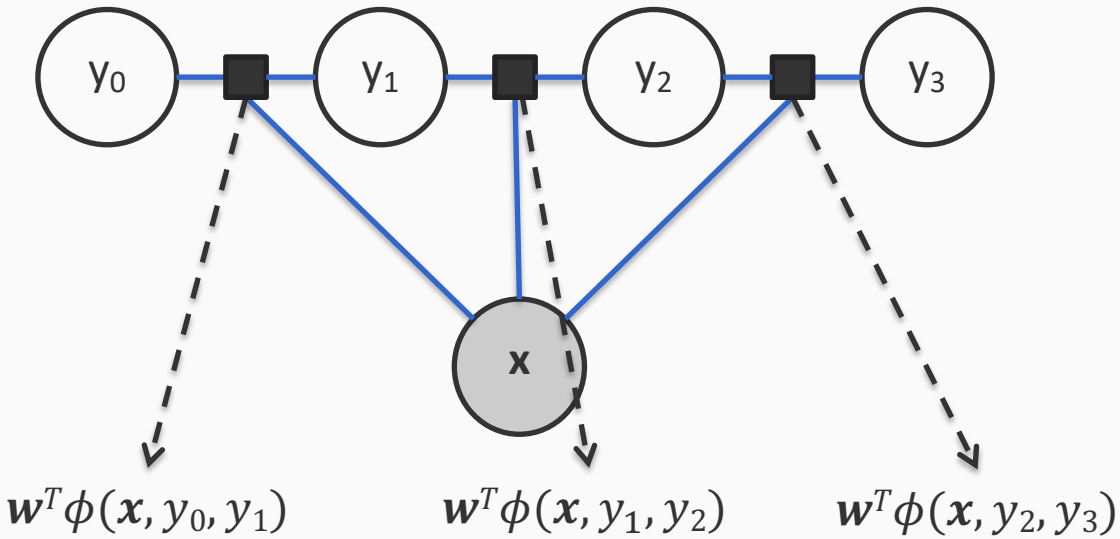
Each node is a random variable

We observe some nodes and need to assign the rest

Each ***clique*** is associated with a score, typically linear

Arbitrary features, as with local conditional models

Scoring with factor graphs



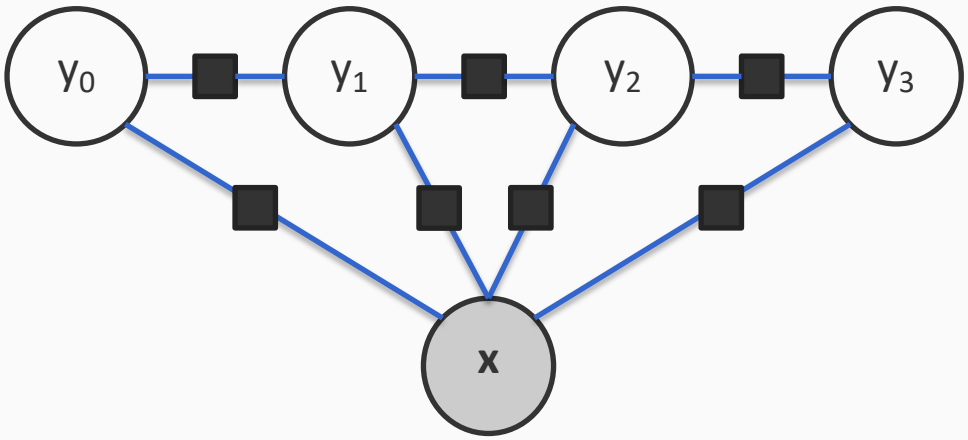
Each node is a random variable

We observe some nodes and need to assign the rest

Each ~~clique~~ factor is associated with a score

Scoring with factor graphs

A different factorization: Recall decomposition of structures into parts. Same idea



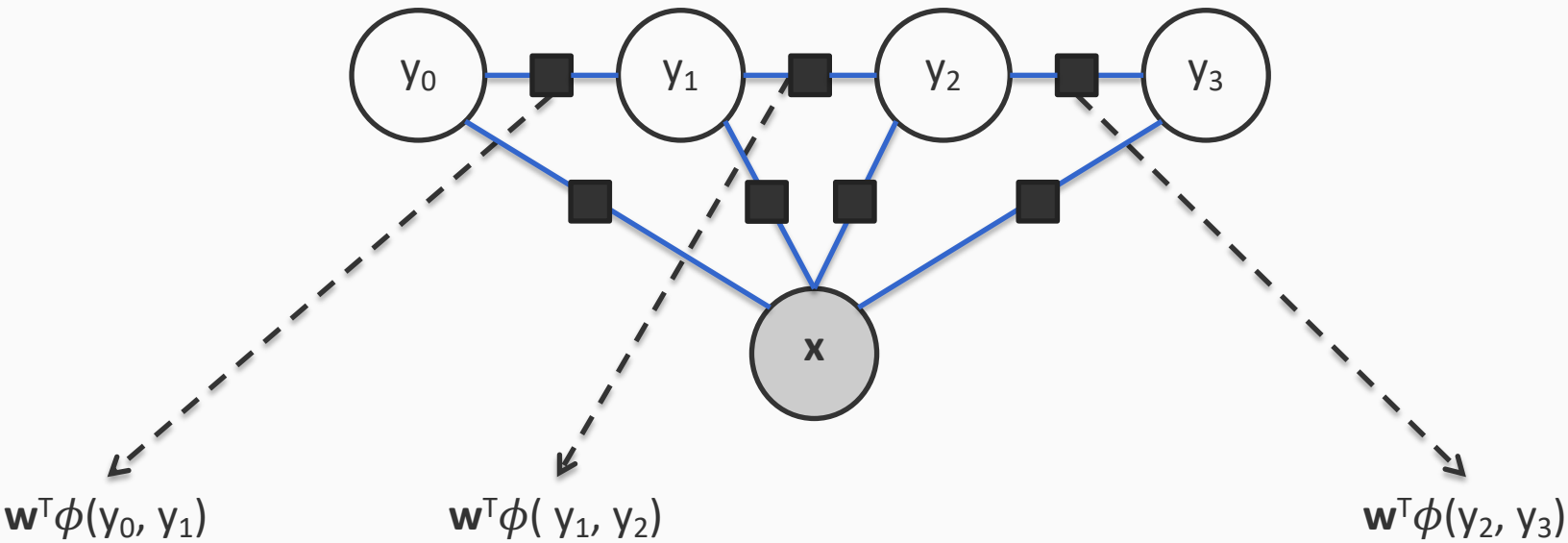
Each node is a random variable

We observe some nodes and need to assign the rest

Each factor is associated with a score

Scoring with factor graphs

A different factorization: Recall decomposition of structures into parts. Same idea



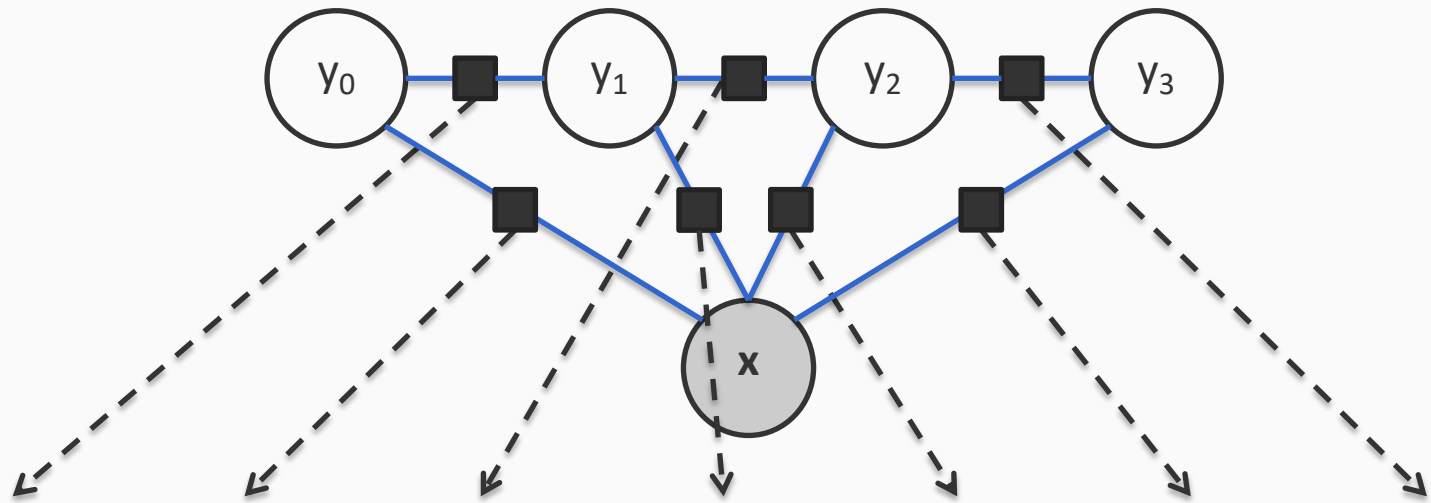
Each node is a random variable

We observe some nodes and need to assign the rest

Each factor is associated with a score

Scoring with factor graphs

A different factorization: Recall decomposition of structures into parts. Same idea



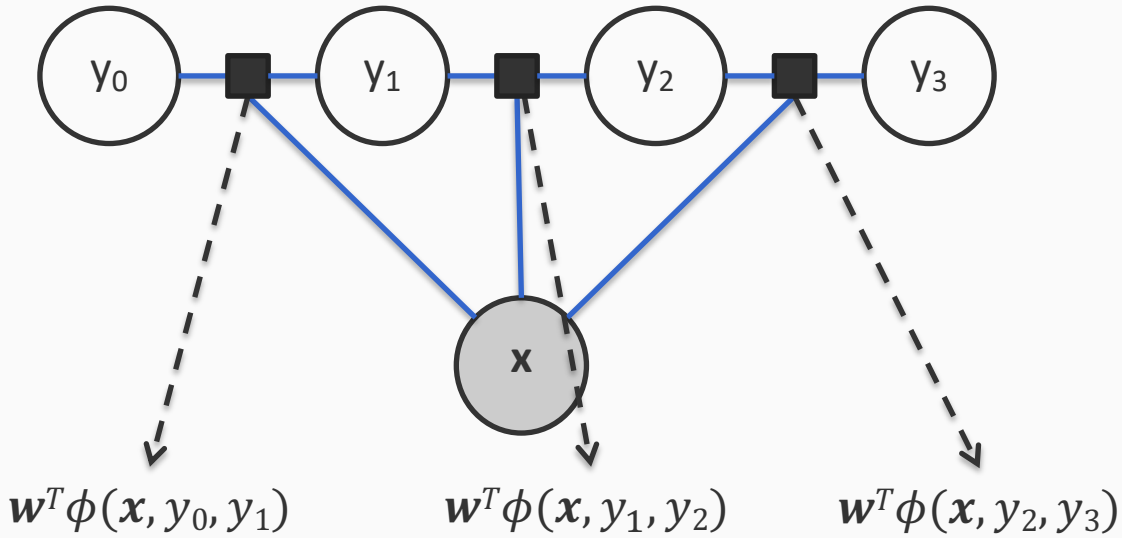
$\mathbf{w}^T\phi(y_0, y_1)$ $\mathbf{w}^T\phi(y_0, \mathbf{x})$ $\mathbf{w}^T\phi(y_1, y_2)$ $\mathbf{w}^T\phi(y_1, \mathbf{x})$ $\mathbf{w}^T\phi(y_2, \mathbf{x})$ $\mathbf{w}^T\phi(y_3, \mathbf{x})$ $\mathbf{w}^T\phi(y_2, y_3)$

Each node is a random variable

We observe some nodes and need to assign the rest

Each factor is associated with a score

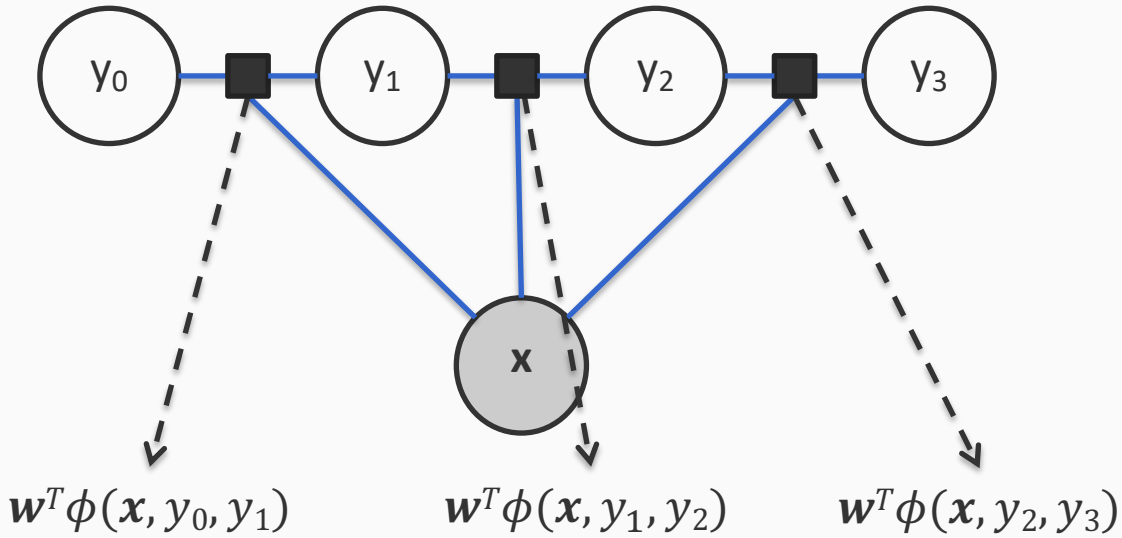
From scores to a probability



Recall our goal: To characterize a probability distribution over the unobserved variables, conditioned on the observed ones.

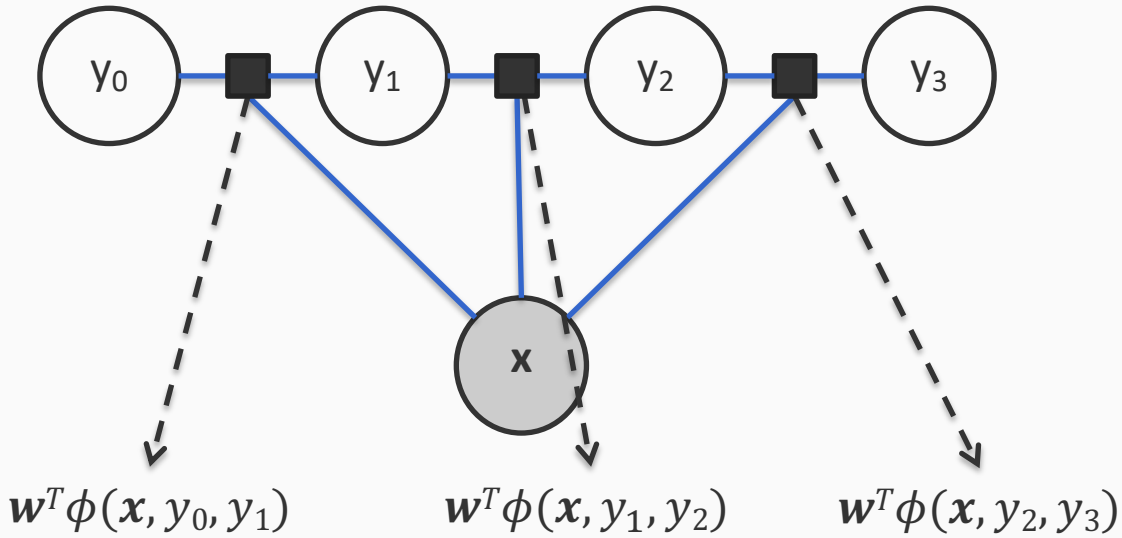
That is, to characterize $P(\, y_0, y_1, \cdots \mid \mathbf{x}\,)$.

From scores to a probability



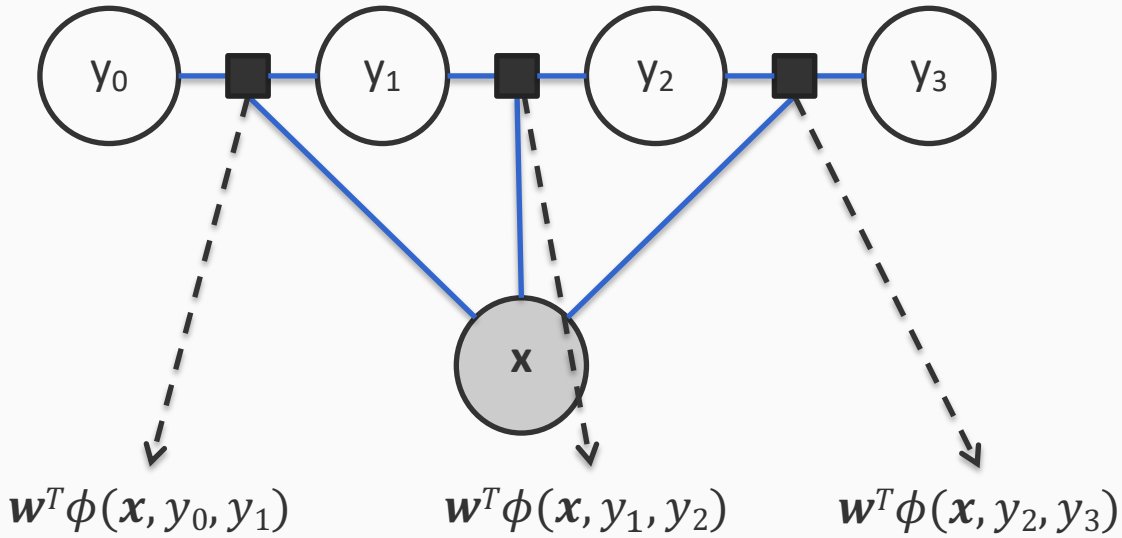
$$P(\mathbf{y} \mid \mathbf{x}) \propto \prod_{f \in \text{factors}} \exp\big(\text{score}(factor)\big)$$

From scores to a probability



$$P(\mathbf{y} \mid \mathbf{x}) \propto \prod_i \exp\left(w^T \phi(\mathbf{x}, y_{i-1}, y_i)\right)$$

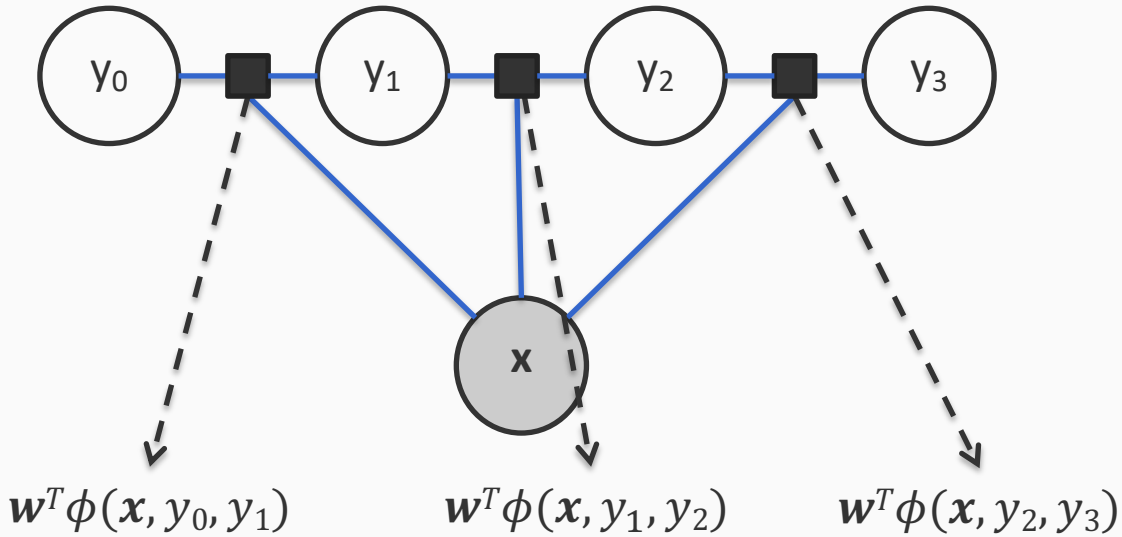
From scores to a probability



$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_i \exp\left(w^T \phi(\mathbf{x}, y_{i-1}, y_i)\right)$$

To get a probability, we need to normalize this using a term $Z(\mathbf{x})$ that ensures that the probabilities add up to one.

From scores to a probability



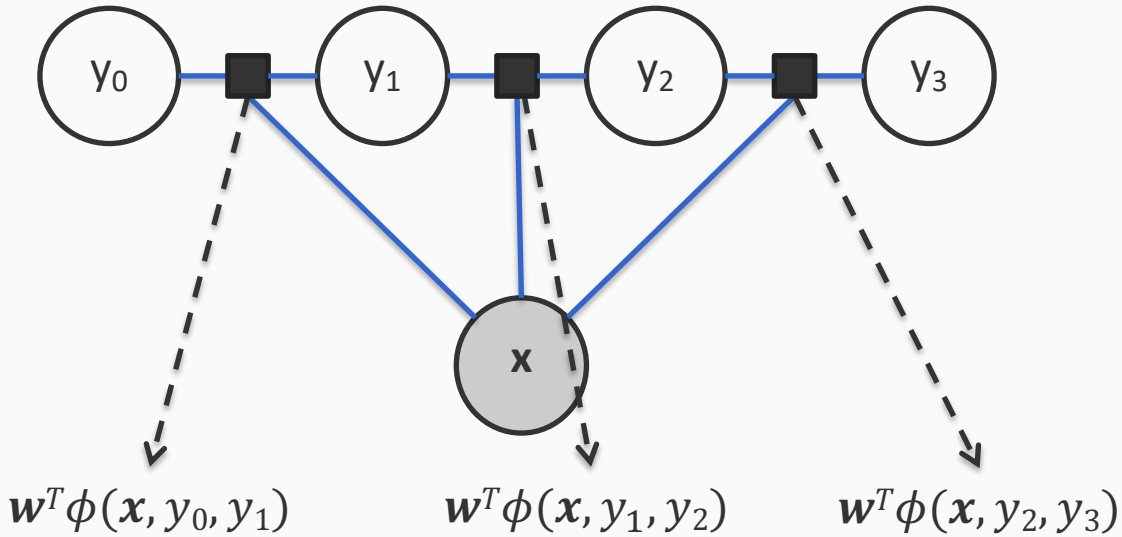
$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_i \exp\left(w^T \phi(\mathbf{x}, y_{i-1}, y_i)\right)$$

To get a probability, we need to normalize this using a term $Z(\mathbf{x})$ that ensures that the probabilities add up to one.

$$Z(\mathbf{x}) = \sum_{\hat{y}} \left(\prod_i \exp(w^T \phi(x, y_{i-1}, y_i) \right)$$

Called the *partition function*

Conditional Random Fields



$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_i \exp\left(w^T \phi(\mathbf{x}, y_{i-1}, y_i)\right)$$

The conditional probability of the labels given the input is a product of normalized factor scores.

Such models are called ***conditional random fields***.

CRF: A different view

- Input: **x**, Output: **y**, sequence (for now)
- Define a feature vector for the **entire** input and output sequence: $\Phi(\mathbf{x}, \mathbf{y})$

CRF: A different view

- Input: **x**, Output: **y**, sequence (for now)
- Define a feature vector for the **entire** input and output sequence: $\Phi(\mathbf{x}, \mathbf{y})$
- Define a giant log-linear model, P(**y** | **x**) parameterized by **w**

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z} \prod_i \exp\left(\mathbf{w}^T \phi(\boldsymbol{x}, y_i, y_{i-1})\right) \propto \exp\left(\mathbf{w}^T \sum_i \phi(\boldsymbol{x}, y_i, y_{i-1})\right)$$

CRF: A different view

- Input: **x**, Output: **y**, sequence (for now)
- Define a feature vector for the **entire** input and output sequence: $\Phi(\mathbf{x}, \mathbf{y})$
- Define a giant log-linear model, P(**y** | **x**) parameterized by **w**

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z} \prod_i \exp\left(\mathbf{w}^T \phi(\boldsymbol{x}, y_i, y_{i-1})\right) \propto \exp\left(\mathbf{w}^T \sum_i \phi(\boldsymbol{x}, y_i, y_{i-1})\right)$$

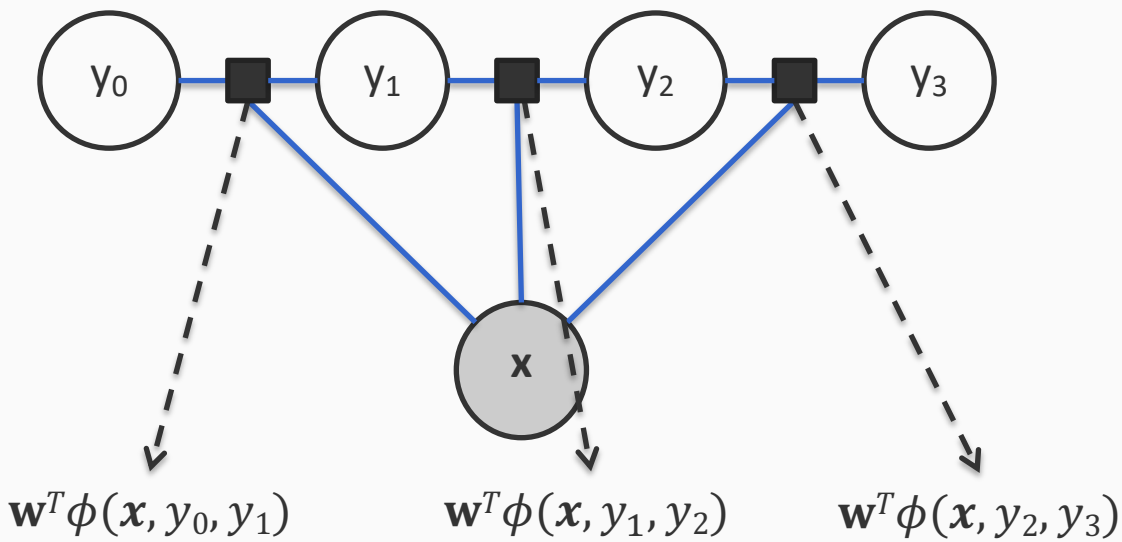
  - Just like any other log-linear model, except
    - Space of **y** is the set of all possible sequences of the correct length
    - Normalization constant sums over all sequences

In an MEMM, probabilities were locally normalized

Global features

The feature function decomposes over the factors in sequence (that is, the factor graph)

$$\Phi(\mathbf{x}, \mathbf{y}) = \sum_i \phi(x, y_{i-1}, y_i)$$



Where are we?

- We have seen how a CRF assigns probabilities to sequences
  - Global normalization instead of local normalization
  - Avoid the label bias problem because of this

- Next:
  - How to predict the most probable sequence
  - How to train the scoring functions

Prediction

Goal: To predict most probable sequence **y** for an input **x**

$$\begin{aligned}\underset{\mathbf{y}}{\operatorname{argmax}}\, P(\,\mathbf{y} \mid \mathbf{x}\,) &= \underset{\mathbf{y}}{\operatorname{argmax}} \exp(\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})) \\ &= \underset{\mathrm{y}}{\operatorname{argmax}}\, \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})\end{aligned}$$

Prediction

Goal: To predict most probable sequence **y** for an input **x**

$$\begin{aligned}\underset{\mathbf{y}}{\operatorname{argmax}}\, P(\,\mathbf{y} \mid \mathbf{x}\,) &= \underset{\mathbf{y}}{\operatorname{argmax}} \exp(\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})) \\ &= \underset{\mathrm{y}}{\operatorname{argmax}}\, \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})\end{aligned}$$

But the score decomposes as $\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y}) = \sum_i \mathbf{w}^T \phi(\mathrm{x}, y_{i-1}, y_i)$

Prediction

Goal: To predict most probable sequence **y** for an input **x**

$$\begin{aligned}\underset{\mathbf{y}}{\operatorname{argmax}}\, P(\,\mathbf{y} \mid \mathbf{x}\,) &= \underset{\mathbf{y}}{\operatorname{argmax}} \exp(\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})) \\ &= \underset{\mathrm{y}}{\operatorname{argmax}}\, \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})\end{aligned}$$

But the score decomposes as $\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y}) = \sum_i \mathbf{w}^T \phi(\mathrm{x}, y_{i-1}, y_i)$

Prediction via Viterbi (with sum instead of product)

Prediction

Goal: To predict most probable sequence **y** for an input **x**

$$\begin{aligned}\underset{\mathbf{y}}{\operatorname{argmax}}\, P(\,\mathbf{y} \mid \mathbf{x}\,) &= \underset{\mathbf{y}}{\operatorname{argmax}} \exp(\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})) \\ &= \underset{\mathrm{y}}{\operatorname{argmax}}\, \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})\end{aligned}$$

But the score decomposes as $\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y}) = \sum_i \mathbf{w}^T \phi(\mathrm{x}, y_{i-1}, y_i)$

Prediction via Viterbi (with sum instead of product)

1. Base case: $\mathrm{score}_0(\mathrm{s}) = \mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}, \mathrm{start}, y_0)$

Prediction

Goal: To predict most probable sequence **y** for an input **x**

$$\underset{\mathbf{y}}{\operatorname{argmax}}\, P(\mathbf{y} \mid \mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmax}} \exp(\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y}))$$
$$= \underset{\mathrm{y}}{\operatorname{argmax}}\, \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y})$$

But the score decomposes as $\mathbf{w}^T \Phi(\mathbf{x}, \mathbf{y}) = \sum_i \mathbf{w}^T \phi(\mathrm{x}, y_{i-1}, y_i)$

Prediction via Viterbi (with sum instead of product)

1. Base case: $\mathrm{score}_0(\mathrm{s}) = \mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}, \mathrm{start}, y_0)$
2. Recursive case:

$$\mathrm{score}_{\mathrm{i}}(\mathrm{s}) = \max_{y_{i-1}} \left(\mathbf{w}^T \phi(\mathbf{x}, y_{i-1}, y_i) + \mathrm{score}_{i-1}(y_{i-1})\right)$$

Training a chain CRF

- Input:
  - Dataset with labeled sequences, D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>}
  - A definition of the feature function

- How do we train?
  - Maximize the (regularized) log-likelihood

$$\max_{\mathbf{w}} -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$$

Recall: Empirical loss minimization

Training with inference

$$\max_{\mathbf{w}} -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$$

- Many methods for training
  - Numerical optimization
  - Can use a gradient or hessian based method

- Simple gradient ascent

$$\mathbf{w} \leftarrow \mathbf{w} + \sum_i \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - \sum_{\hat{\mathbf{y}}} P(\hat{\mathbf{y}}|\mathbf{x}_i, \mathbf{w})\phi(\mathbf{x}_i, \hat{\mathbf{y}}) \right)$$

Training with inference

$$\max_{\mathbf{w}} -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$$

- Many methods for training
  - Numerical optimization
  - Can use a gradient or hessian based method

- Simple gradient ascent

$$\mathbf{w} \leftarrow \mathbf{w} + \sum_i \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - \sum_{\hat{\mathbf{y}}} P(\hat{\mathbf{y}}|\mathbf{x}_i, \mathbf{w})\phi(\mathbf{x}_i, \hat{\mathbf{y}}) \right)$$

- Training involves inference!
  - A different kind than what we have seen so far
  - Summing over all sequences is just like Viterbi
    - With summation instead of maximization

CRF (for sequences): Summary

- An undirected graphical model
  - Decompose the score over the structure into a collection of factors
  - Each factor assigns a score to assignment of the random variables it is connected to

- Training and prediction
  - Final prediction via argmax $w^T \phi(\mathbf{x}, \mathbf{y})$
  - Train by maximum (regularized) likelihood

- Relation to other models
  - Effectively a linear classifier
  - A generalization of logistic regression to structures
  - An instance of Markov Random Field, with some random variables observed
    - We will see this soon

Predicting Sequences: Structured Perceptron

Conditional Random Fields summary

- An undirected graphical model
  - Decompose the score over the structure into a collection of factors
  - Each factor assigns a score to assignment of the random variables it is connected to

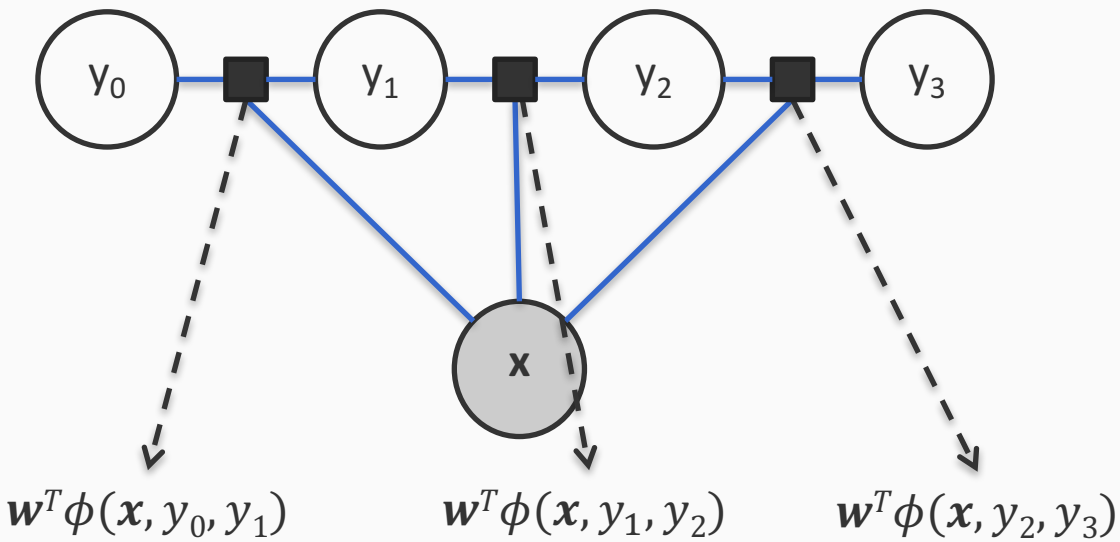
- Training and prediction
  - Final prediction via argmax $w^T \phi(\mathbf{x}, \mathbf{y})$
  - Train by maximum (regularized) likelihood

- Connections to other models
  - Effectively a linear classifier
  - A generalization of logistic regression to structures
  - An conditional variant of a Markov Random Field
    - We will see this soon

Global features

The feature function decomposes over the sequence

$$\phi(\mathbf{x}, \mathbf{y}) = \sum_i \phi(\mathbf{x}, y_i, y_{i-1})$$



Outline

- Sequence models
- Hidden Markov models
  - Inference with HMM
  - Learning
- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields
  - *Structured Perceptron for sequences*

HMM is also a linear classifier

Consider the HMM:

$$P(\mathbf{x}, \mathbf{y}) = \prod_i P(\, y_i \mid y_{i-1}\,) P(\, x_i \mid y_i\,)$$

HMM is also a linear classifier

Consider the HMM:

$$P(\mathbf{x}, \mathbf{y}) = \prod_i \underbrace{P(y_i \mid y_{i-1})}_{\text{Transitions}} \underbrace{P(x_i \mid y_i)}_{\text{Emissions}}$$

HMM is also a linear classifier

Consider the HMM:

$$P(\mathbf{x}, \mathbf{y}) = \prod_i P(\, y_i \mid y_{i-1}\,) P(\, x_i \mid y_i\,)$$

Or equivalently

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}\,) + \log P(x_i \mid y_i\,)$$

Log joint probability = transition scores + emission scores

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Log joint probability = transition scores + emission scores

Let us examine this expression using a carefully defined set of *indicator functions*

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Log joint probability = transition scores + emission scores

Let us examine this expression using a carefully defined set of *indicator functions*

$$I_{[z]} = \begin{cases} 1, & z \text{ is true,} \\ 0, & z \text{ is false.} \end{cases}$$

Indicators are functions that map Booleans to 0 or 1

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \boxed{\log P(y_i \mid y_{i-1})} + \log P(x_i \mid y_i)$$

Log joint probability = transition scores + emission scores

Let us examine this expression using a carefully defined set of *indicator functions*

Equivalent to

$$\boxed{\sum_s \sum_{s'} \log P(s \mid s') \cdot I_{[y_i = s]} \cdot I_{[y_{i-1} = s']}}$$

The indicators ensure that only one of the elements of the double summation is non-zero

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \boxed{\log P(x_i \mid y_i)}$$

Log joint probability = transition scores + emission scores

Let us examine this expression using a carefully defined set of *indicator functions*

Equivalent to

$$\boxed{\sum_s \log P(x_i \mid s) \cdot I_{[y_i = s]}}$$

The indicators ensure that only one of the elements of the summation is non-zero

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \sum_s \sum_{s'} \log P(s \mid s') \cdot I_{[y_i = s]} \cdot I_{[y_{i-1} = s']}$$
$$+ \sum_i \sum_s \log P(x_i \mid s) \cdot I_{[y_i = s]}$$

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \sum_s \sum_{s'} \log P(s \mid s') \cdot I_{[y_i = s]} \cdot I_{[y_{i-1} = s']} + \sum_i \sum_s \log P(x_i \mid s) \cdot I_{[y_i = s]}$$

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \sum_i I_{[y_i = s]} I_{[y_{i-1} = s']} + \sum_s \log P(x_i \mid s) \sum_i I_{[y_i = s]}$$

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \sum_i I_{[y_i = s]} I_{[y_{i-1} = s']}$$

$$+ \sum_s \log P(x_i \mid s) \sum_i I_{[y_i = s]}$$

Number of times there is a transition in the sequence from state s' to state s

$\mathrm{Count}(s' \to s)$

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \cdot \mathrm{Count}(s' \to s)$$
$$+ \sum_s \log P(x_i \mid s) \sum_i I_{[y_i = s]}$$

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \cdot \mathrm{Count}(s' \to s)$$
$$+ \sum_s \log P(x_i \mid s) \sum_i I_{[y_i = s]}$$

Number of times state s occurs in the sequence: $\mathrm{Count}(s)$

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Let us examine this expression using a carefully defined set of *indicator functions*

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \cdot \text{Count}(s' \to s) + \sum_s \log P(x_i \mid s) \cdot \text{Count}(s)$$

HMM is also a linear classifier

$$\log P(\mathbf{x}, \mathbf{y}) = \sum_i \log P(y_i \mid y_{i-1}) + \log P(x_i \mid y_i)$$

Let us examine this expression using a carefully defined set of *indicator functions*

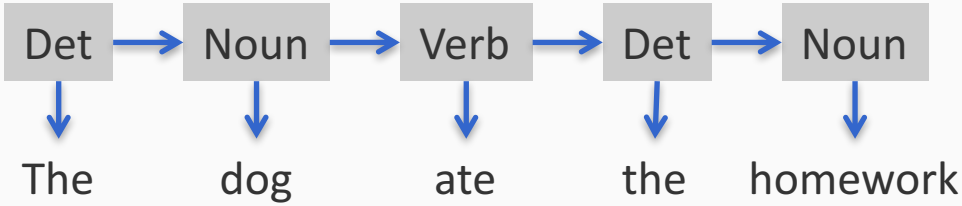
$$\log P(\mathbf{x}, \mathbf{y}) = \sum_s \sum_{s'} \log P(s \mid s') \cdot \mathrm{Count}(s' \to s) + \sum_s \log P(x_i \mid s) \cdot \mathrm{Count}(s)$$

This is a linear function

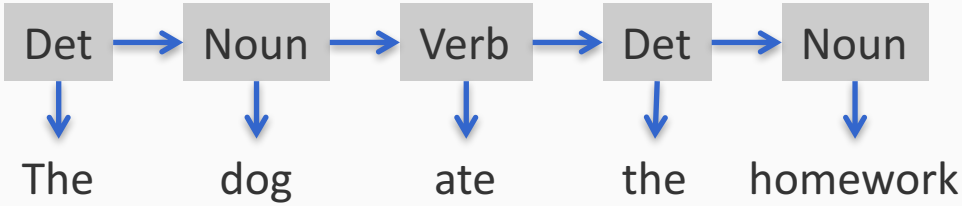
log P terms are the weights; counts via indicators are features

Can be written as $\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x}, \mathbf{y})$ and add more features

HMM is a linear classifier: An example

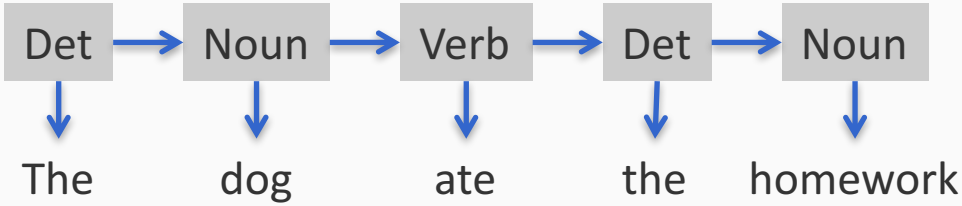


HMM is a linear classifier: An example



Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

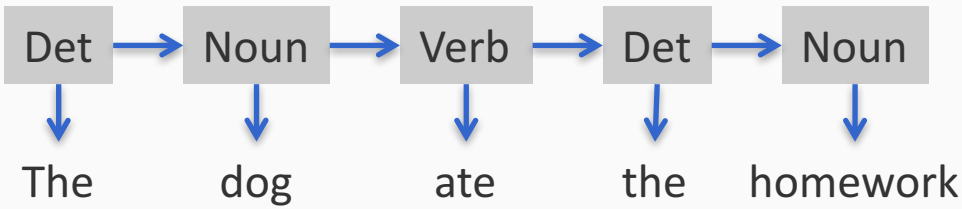
HMM is a linear classifier: An example



Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

Transition scores + Emission scores

HMM is a linear classifier: An example



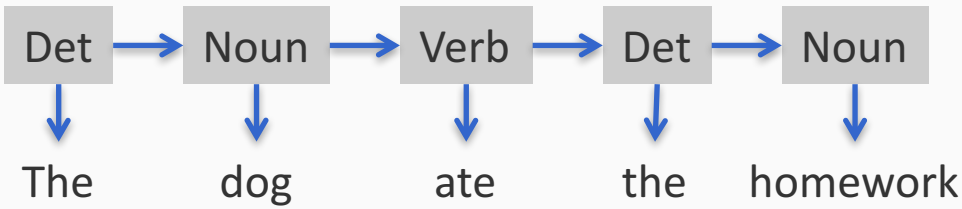
Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

log P(Det $\rightarrow$ Noun) $\times$ 2

+ log P(Noun $\rightarrow$ Verb) $\times$ 1 + Emission scores

+ log P(Verb $\rightarrow$ Det) $\times$ 1

HMM is a linear classifier: An example



Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

log P(Det $\rightarrow$ Noun) $\times$ 2

+ log P(Noun $\rightarrow$ Verb) $\times$ 1

+ log P(Verb $\rightarrow$ Det) $\times$ 1

+

log P(The | Det) $\times$ 1

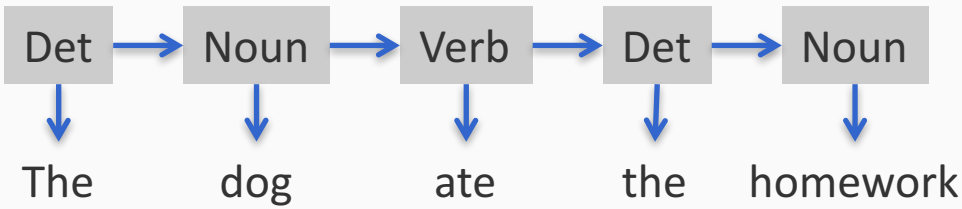
+ log P(dog| Noun) $\times$ 1

+ log P(ate| Verb) $\times$ 1

+ log P(the| Det) $\times$ 1

+ log P(homework| Noun) $\times$ 1

HMM is a linear classifier: An example



Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

log P(Det → Noun) × 2

+ log P(Noun → Verb) × 1

+ log P(Verb → Det) × 1

+

log P(The | Det) × 1

+ log P(dog| Noun) × 1

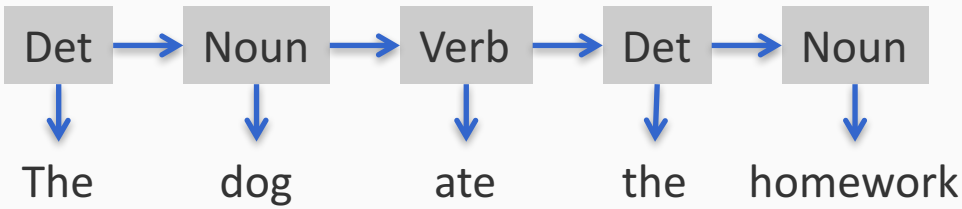
+ log P(ate| Verb) × 1

+ log P(the| Det) × 1

+ log P(homework| Noun) × 1

w: Parameters of the model

HMM is a linear classifier: An example



Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

log P(Det → Noun) × 2

+ log P(Noun → Verb) × 1

+ log P(Verb → Det) × 1

+

log P(The | Det) × 1

+ log P(dog| Noun) × 1

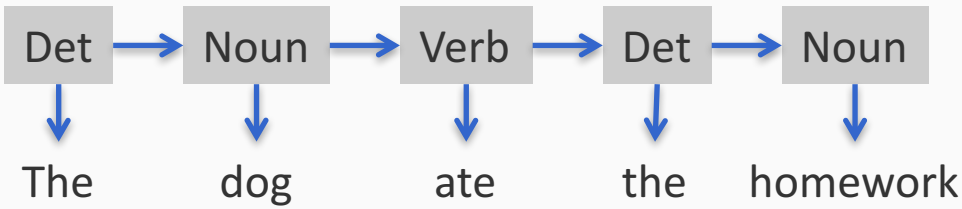
+ log P(ate| Verb) × 1

+ log P(the| Det) × 1

+ log P(homework| Noun) × 1

ϕ(**x**, **y**): Properties of this output and the input

HMM is a linear classifier: An example



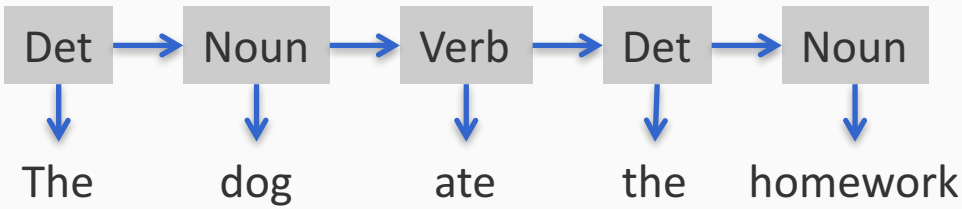
Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

$$\begin{bmatrix} \log P(Det \rightarrow Noun) \\ \log P(Noun \rightarrow Verb) \\ \log P(Verb \rightarrow Det) \\ \log P(The \mid Det) \\ \log P(dog \mid Noun) \\ \log P(ate \mid Verb) \\ \log P(the \mid Det \\ \log P(homework \mid Noun) \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

w: Parameters of the model

ϕ(**x**, **y**): Properties of this output and the input

HMM is a linear classifier: An example



Consider $\log P(\mathbf{y}, \mathbf{x}) = \sum_i \log P(y_i | y_{i-1}) + \log P(x_i | y_i)$

$$\begin{bmatrix} \log P(Det \rightarrow Noun) \\ \log P(Noun \rightarrow Verb) \\ \log P(Verb \rightarrow Det) \\ \log P(The \mid Det) \\ \log P(dog \mid Noun) \\ \log P(ate \mid Verb) \\ \log P(the \mid Det \\ \log P(homework \mid Noun) \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

log P(**x**, **y**) = A linear scoring function = $\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x},\mathbf{y})$

w: Parameters of the model

ϕ(**x**, **y**): Properties of this output and the input

Towards structured Perceptron

1. HMM is a linear classifier
   - Can we treat it as any linear classifier for training?
   - If so, we could add additional features that are global properties
     - As long as the output can be decomposed for easy inference

Towards structured Perceptron

1. HMM is a linear classifier
   - Can we treat it as any linear classifier for training?
   - If so, we could add additional features that are global properties
     - As long as the output can be decomposed for easy inference

2. The Viterbi algorithm calculates max $w^T \phi(\mathbf{x}, \mathbf{y})$

   Viterbi only cares about scores to structures (not necessarily normalized)

Towards structured Perceptron

1. HMM is a linear classifier
   - Can we treat it as any linear classifier for training?
   - If so, we could add additional features that are global properties
     - As long as the output can be decomposed for easy inference

2. The Viterbi algorithm calculates max $w^T\phi(\mathbf{x}, \mathbf{y})$

   Viterbi only cares about scores to structures (not necessarily normalized)

3. We could push the learning algorithm to train for un-normalized scores
   - If we need normalization, we could always normalize by exponentiating and dividing by Z (the partition function)
   - That is, the learning algorithm can effectively just focus on the score of **y** for a particular **x**
   - *Train a discriminative model instead of the generative one!*

Structured Perceptron algorithm

Given a training set D = {(**x**,**y**)}

1. Initialize **w** = 0 $\in \Re^n$
2. For epoch = 1 … T:
   1. For each training example (**x**, **y**) $\in$ D:

      Structured Perceptron update

3. Return **w**

Prediction: $\text{argmax}_{\mathbf{y}}\, \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})$

Structured Perceptron algorithm

Given a training set D = {(**x**,**y**)}

1. Initialize **w** = 0 $\in \Re^n$
2. For epoch = 1 … T:
   1. For each training example (**x**, **y**) $\in$ D:
      1. Predict **y'** = $\text{argmax}_{y'}$ $\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
      2. If **y** ≠ **y'**, update **w** $\leftarrow$ **w** + r ($\phi(\mathbf{x}, \mathbf{y})$ - $\phi(\mathbf{x}, \mathbf{y'})$)
3. Return **w**

Prediction: $\text{argmax}_{\mathbf{y}}\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$

Structured Perceptron algorithm

Given a training set D = {(**x**,**y**)}

1. Initialize **w** = 0 $\in \Re^n$
2. For epoch = 1 … T:
   1. For each training example (**x**, **y**) $\in$ D:
      1. Predict **y'** = $\text{argmax}_{y'}$ $\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
      2. If **y** ≠ **y'**, update **w** $\leftarrow$ **w** + r (ϕ(**x**, **y**) - ϕ(**x**, **y'**))
3. Return **w**

Update only on an error.
Perceptron is an mistake-driven algorithm.
If there is a mistake, promote **y** and demote **y'**

Prediction: $\text{argmax}_y\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$

Structured Perceptron algorithm

Given a training set D = {(**x**,**y**)}

1. Initialize **w** = 0 $\in \Re^n$
2. For epoch = 1 … T:
   1. For each training example (**x**, **y**) $\in$ D:
      1. Predict **y'** = $\text{argmax}_{y'}$ $\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
      2. If **y** ≠ **y'**, update **w** $\leftarrow$ **w** + r ($\phi(\mathbf{x}, \mathbf{y})$ - $\phi(\mathbf{x}, \mathbf{y'})$)
3. Return **w**

T is a hyperparameter to the algorithm

Prediction: argmax_y $\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$

Structured Perceptron algorithm

In practice, good to shuffle D before the inner loop

Given a training set D = {(**x**,**y**)}

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. For each training example $(\mathbf{x}, \mathbf{y}) \in D$:
      1. Predict $\mathbf{y'} = \text{argmax}_{\mathbf{y'}}\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
      2. If $\mathbf{y} \neq \mathbf{y'}$, update $\mathbf{w} \leftarrow \mathbf{w} + r\,(\phi(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y'}))$
3. Return **w**

Prediction: $\text{argmax}_{\mathbf{y}}\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$

Structured Perceptron algorithm

Given a training set D = {(**x**,**y**)}

1. Initialize **w** = 0 $\in \Re^n$
2. For epoch = 1 … T:
   1. For each training example (**x**, **y**) $\in$ D:
      1. Predict **y'** = $\text{argmax}_{y'}$ $\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
      2. If **y** ≠ **y'**, update **w** $\leftarrow$ **w** + r (ϕ(**x**, **y**) - ϕ(**x**, **y'**))
3. Return **w**

Inference in training loop!

Prediction: argmax_y $\mathbf{w}^T\phi(\mathbf{x}, \mathbf{y})$

Notes on structured perceptron

- Mistake bound for separable data, just like perceptron
- In practice, use averaging for better generalization
  - Initialize **a** = 0
  - After each step, whether there is an update or not, **a** $\leftarrow$ **w** + **a**
    - Note, we still check for mistake using **w** not **a**
  - Return **a** at the end instead of **w**

    Exercise: Optimize this for performance – modify **a** *only* on errors
- Global update
  - One weight vector for entire sequence
    - Not for each position
  - Same algorithm can be derived via constraint classification
    - Create a binary classification data set and run perceptron

Structured Perceptron with averaging

Given a training set D = {(**x**,**y**)}

1. Initialize $\mathbf{w} = 0 \in \Re^n$, $\mathbf{a} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. For each training example $(\mathbf{x}, \mathbf{y}) \in D$:
      1. Predict $\mathbf{y'} = \text{argmax}_{\mathbf{y'}}\ \mathbf{w}^T\phi(\mathbf{x}, \mathbf{y'})$
      2. If $\mathbf{y} \neq \mathbf{y'}$, update $\mathbf{w} \leftarrow \mathbf{w} + r\,(\phi(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y'}))$
      3. Set $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$
3. Return **a**

CRF vs. structured perceptron

Stochastic gradient descent update for CRF

– For a training example ($\mathbf{x}_i$, $\mathbf{y}_i$)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - E_{\mathbf{y}}[\phi(\mathbf{x}_i, \mathbf{y})]\right)$$

Structured perceptron

– For a training example ($\mathbf{x}_i$, $\mathbf{y}_i$)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - \phi(\mathbf{x}_i, \hat{\mathbf{y}})\right)$$

Expectation vs max

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})$$

Caveat: Adding regularization will change the CRF update, averaging changes the perceptron update

Sequence models: Summary

- Goal: Predict an output sequence given input sequence
- Hidden Markov Model
- Inference
  - Predict via Viterbi algorithm
- Conditional models/discriminative models
  - Local approaches (no inference during training)
    - MEMM, conditional Markov model
  - Global approaches (inference during training)
    - CRF, structured perceptron
- To think
  - What are the *parts* in a sequence model?
  - How is each model scoring these parts?

Prediction is not always tractable for general structures

Same dichotomy for more general structures