The Final Exam

- Tuesday, December 17, 12:25 – 2:25 p.m.
  rooms 1111 and 3650 Humanities Bldg
- Covers topics since Midterm (i.e., constraint satisfaction through face detection) only
- Closed book
- 8.5" x 11" sheet with notes on both sides (typed or handwritten)
- **Bring a calculator (not on a phone)!**

1

- Covers lecture notes (see updated notes!), readings in textbook, and 2 papers, one on deep learning and one on HMMs
- True/False and multiple choice questions (with calculations required)

2

Constraint Satisfaction

Problem formulation in terms of variables, domains and constraints, constraint graph, depth-first search, backtracking with consistency checking, most constrained variable heuristic, most constraining variable heuristic, least constraining value heuristic, min-conflicts heuristic, min-conflicts algorithm, forward checking algorithm, arc consistency algorithm (AC-3), combining search with CSP inference

3

CSP

Algorithms
- Min-Conflicts
- Backtracking (DFS) with consistency checking
- Forward checking
- Arc consistency (AC-3)

4

Min-Conflicts Algorithm

Assign to each variable a random value, defining the initial state
while state not consistent **do**
Pick a variable, *var*, that has constraint(s) violated
Find value, *v*, for *var* that minimizes the *total* number of violated constraints (over all variables)
var = *v*

Value selection by **min-conflicts heuristic**:
choose value that *violates the fewest constraints*, i.e., hill-climb by minimizing *f*(*n*) = total number of violated constraints

5

Backtracking (DFS) w/ Consistency Checking

Start with empty state
while not all vars in state assigned a value **do**
Pick a variable (randomly or with a heuristic)
if it has a value that does not violate any constraints
then Assign that value
else
Go back to previous variable and assign it another value

6

Backtracking (DFS) w/ Consistency Checking

- Don't generate a successor that creates an inconsistency with any *existing* assignment, i.e., perform **consistency checking** *when node is generated*
- Successor function assigns a value to an unassigned variable that does ***not*** conflict with *all* current assignments
  - "backward checking"
  - Deadend if no legal assignments (i.e., no successors)

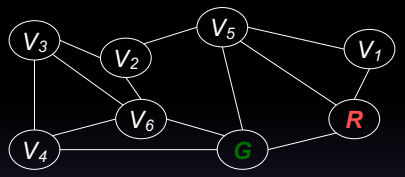
7

Heuristics used with DFS

1. Most-Constrained Variable (minimum remaining values heuristic)
   - Choose the variable with the *fewest* number of consistent values
2. Most-Constraining Variable (degree heuristic)
   - Choose the variable with the *most* constraints on the *remaining variables*
3. Least-Constraining Value
   - Pick the value that rules out the *fewest values* in the remaining variables

8

Forward Checking Algorithm



- Initially, for each variable, record the **set of *all* possible legal values for it**
- When you assign a value to a variable in the search, *update the set of legal values for **all** unassigned variables. Backtrack immediately if you **empty** a variable's set of possible values.*

9

Neural Networks

Perceptron, LTU, activation functions, bias input, input units, output units, Perceptron learning rule, Perceptron learning algorithm, Perceptron convergence theorem, epoch, weight space, input space, linearly separable, credit assignment problem, multi-layer feed-forward networks, hidden units, sigmoid function, ReLU, softmax, back-propagation algorithm, gradient descent search in weight space, stochastic gradient descent, parameter setting using a tuning set, deep learning, convolutional neural networks, pooling

10

Back-Propagation Algorithm

Initialize the weights in the network (usually random values)
Repeat until stopping criterion is met **{**
  forall *p*,*q* in network, $\Delta \boldsymbol{W}_{p,q} = 0$
  foreach example ***e*** in training set **do** **{**
    $\boldsymbol{O}$ = neural_net_output(***network***, ***e***) // forward pass
    Calculate error ($\boldsymbol{T}$ - $\boldsymbol{O}$) at the output units // $\boldsymbol{T}$ = teacher output
    Compute $\Delta w_{j,k}$ for all weights from hidden unit j to output unit k
    Compute $\Delta w_{i,j}$ for all weights from input unit i to hidden unit j
    forall *p*,*q* in network $\Delta \boldsymbol{W}_{p,q} = \Delta \boldsymbol{W}_{p,q} + \Delta w_{\boldsymbol{p,q}}$
    }
  for all *p*,*q* in network $\Delta \boldsymbol{W}_{p,q} = \Delta \boldsymbol{W}_{p,q}$ / num_training_examples
  network = update_weights(***network***, $\Delta \boldsymbol{W}_{p,q}$)
}

backward pass

Note: Uses average gradient for all training examples when updating weights

11

Back-Prop using Stochastic Gradient Descent (SGD)

- Most practitioners use SGD to update weights using the *average gradient computed using a small batch of examples*, and repeating this process for many small batches from the training set
- In extreme case, update after *each* example
- Called *stochastic* because each small set of examples gives a noisy estimate of the average gradient over *all* training examples

12

CNNs are an extension of traditional multi-layer, feed-forward networks that incorporate **4 key ideas**:

- Use of **many layers**
  - Learn a hierarchy of features
- **Local "receptive fields"/filters and local connections**
  - Layers are *not* completely connected
  - Want translation-invariant and distortion-invariant local features
- **Shared weights**
- **Pooling**

13

Why are they called "Convolutional" NNs?

The image filtering operation defined as

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

is very similar to the **Convolution** operation defined as

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m-k, n-l]$$

- **In CNN's, *f* corresponds to the inputs from the layer below and *g* corresponds to the weights**
- So, CNN's will learn a set of filters!

Credit: K. Grauman

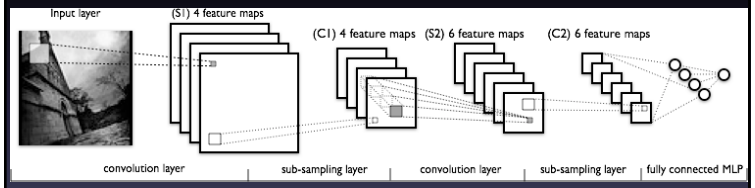
14

Convolution Layers

- Learn "filters" (i.e., weights) that process small regions of the layer below it and compute "features" at many spatial positions
- Example: 32 x 32 x 3 input RGB image, and receptive field (filter size): 5 x 5
  - Each unit in the Conv layer will have weights connected to a 5 x 5 x 3 region in the input layer, with 5*5*3 = 75 weights
  - Can have multiple units associated with a given receptive field in order to compute multiple features at each position
  - "Stride" parameter defines shift amount

15

CNN Architecture



Convolutions and ReLU Convolutions and ReLU

Max pooling Max pooling

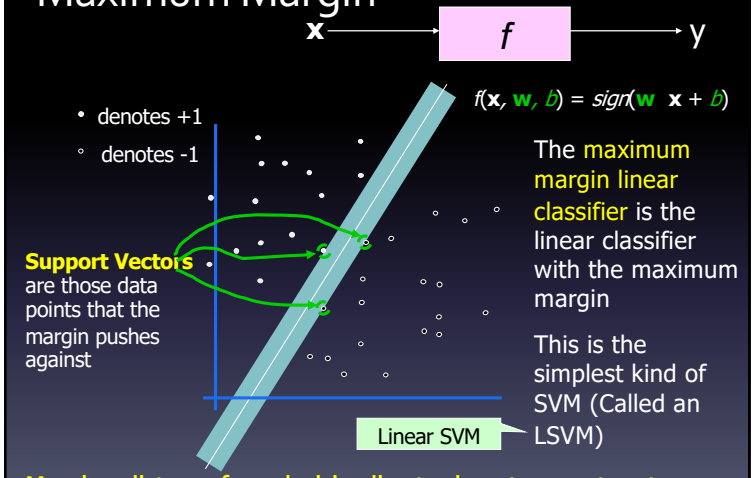
16

Support Vector Machines

Maximum margin, definition of margin, kernel trick, support vectors, slack variables

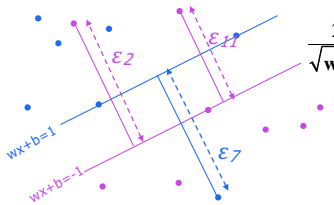
17

Maximum Margin



18

Learning Maximum Margin with Non-Linearly-Separable Data



"slack variables"

$\frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}}$

1. Given guess of **w** , *b* we can
   1. Compute sum of distances of points to their correct zones
   2. Compute the margin width

Assume *N* examples, each ($\mathbf{x}_k$, y_k) where y_k = +1 / -1

What should our optimization criterion be?

Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{k=1}^{N}\varepsilon_k$

How many constraints will we have? **2*N***

What should they be?

$\mathbf{w}^T\mathbf{x}_k + b \geq +1 - \varepsilon_k$ if $y_k = +1$

$\mathbf{w}^T\mathbf{x}_k + b \leq -1 + \varepsilon_k$ if $y_k = -1$

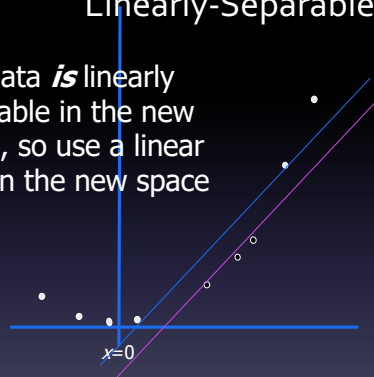
$\varepsilon_k \geq 0$ for all k

19

The Kernel Trick for Dealing with Non-Linearly-Separable Data

The data ***is*** linearly separable in the new space, so use a linear SVM in the new space

The **Kernel Trick**: Preprocess the data, mapping **x** into a higher dimensional space, **z** = Φ(**x**)



$\Phi(x) = (x\ , x^2)$

20

- Dual formulation of the optimization problem depends on the input data only in dot products of the form:
  $\Phi(\mathbf{x}_i)^T \cdot \Phi(\mathbf{x}_j)$ where $\mathbf{x}_i$ and $\mathbf{x}_j$ are two examples
- We can compute these dot products efficiently for certain types of Φ's where $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^T \cdot \Phi(\mathbf{x}_j)$
- Example:

$$\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\Phi(\mathbf{x}_i)^T \cdot \Phi(\mathbf{x}_j) = (\mathbf{x}_i^T \cdot \mathbf{x}_j)^2 = K(\mathbf{x}_i, \mathbf{x}_j)$$

- Since the data *only* appears as dot products, we do *not* need to map the data to higher dimensional space (using $\Phi(\mathbf{x})$) because we can use the kernel function K instead

21

Reasoning under Uncertainty

Random variable, mutually exclusive, prior probability, 3 axioms of probability, joint probability, conditional probability, posterior probability, full joint probability distribution, degrees of freedom, summing out, marginalization, normalization, product rule, chain rule, conditionalized version of chain rule, Bayes's rule, conditionalized version of Bayes's rule, addition/conditioning rule, independence, conditional independence, naïve Bayes classifier as a Bayesian network, Add-1 smoothing, Laplace smoothing

22

Summary of Important Rules

- **Conditional Probability**: $P(A|B) = P(A,B)/P(B)$
- **Product rule**: $P(A,B) = P(A|B)P(B)$
- **Chain rule**: $P(A,B,C,D) = P(A|B,C,D)P(B|C,D)P(C|D)P(D)$
- **Conditionalized version of Chain rule**:
  $P(A,B|C) = P(A|B,C)P(B|C)$
- **Bayes's rule**: $P(A|B) = P(B|A)P(A)/P(B)$
- **Conditionalized version of Bayes's rule**:
  $P(A|B,C) = P(B|A,C)P(A|C)/P(B|C)$
- **Addition / Conditioning rule**: $P(A) = P(A,B) + P(A,\neg B)$
  $P(A) = P(A|B)P(B) + P(A|\neg B)P(\neg B)$

23

Naive Bayes Classifier Testing Phase

- For a given test instance defined by $X_1=v_1, \ldots, X_n=v_n$, compute

$$argmax_c P(Y=c) \prod_{i=1}^{n} P(X_i=v_i \mid Y=c)$$

Class variable

Evidence variable

- Assumes all evidence variables are conditionally independent of each other given the class variable
- Robust because it gives the right answer as long as the correct class is more likely than all others

24

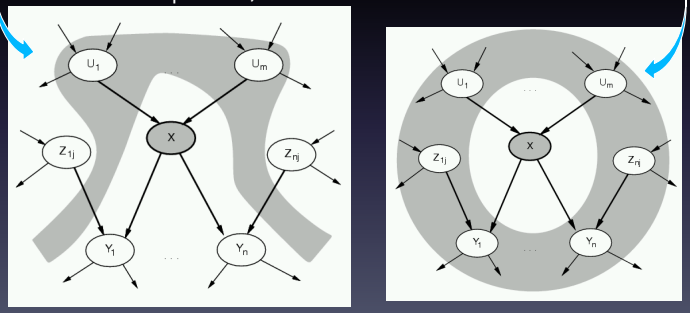
Bayesian Networks

Bayesian network DAG, conditional probability tables, space saving compared to full joint probability distribution table, conditional independence property defined by a Bayesian network, inference by enumeration from a Bayesian network, naïve Bayes classifier as a Bayesian network

25

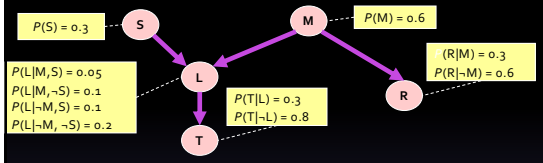
Conditional Independence in Bayes Nets

- **A node is conditionally independent of its non-descendants, given its parents**
- A node is conditionally independent of all other nodes, given its "Markov blanket" (i.e., its parents, children, and children's parents)



26

Computing with Bayes Net



Apply the Chain Rule + conditional independence!

$P(T, \neg R, L, \neg M, S)$
$= P(T \mid \neg R, L, \neg M, S) * P(\neg R, L, \neg M, S)$
$= P(T \mid L) * P(\neg R, L, \neg M, S)$
$= P(T \mid L) * P(\neg R \mid L, \neg M, S) * P(L, \neg M, S)$
$= P(T \mid L) * P(\neg R \mid \neg M) * P(L, \neg M, S)$
$= P(T \mid L) * P(\neg R \mid \neg M) * P(L \mid \neg M, S) * P(\neg M, S)$
$= P(T \mid L) * P(\neg R \mid \neg M) * P(L \mid \neg M, S) * P(\neg M \mid S) * P(S)$
$= P(T \mid L) * P(\neg R \mid \neg M) * P(L \mid \neg M, S) * P(\neg M) * P(S)$

27

"Inference by Enumeration" Algorithm

Computing ***any*** conditional probability:
P(Some variables | Some other variable values)

$$P(E_1 \mid E_2) = \frac{P(E_1 \wedge E_2)}{P(E_2)} = \frac{\sum_{\text{joint entries matching } E_1 \text{ and } E_2} P(\text{joint entry})}{\sum_{\text{joint entries matching } E_2} P(\text{joint entry})}$$

28

Speech Recognition

Phones, phonemes, speech recognition using Bayes's rule, language model, acoustic model, bigram model, trigram model, first-order Markov assumption, probabilistic finite state machine, first-order Markov model, state transition matrix, π vector, computing conditional probabilities from a Markov model, hidden Markov model, observation likelihood matrix, computing joint probabilities and conditional probabilities from an HMM by enumeration

29

- *Signal* = observation sequence
- *Words* = sequence of words $W = w_1, w_2, w_3, \ldots, w_n$
- Best match metric: $\hat{W} = \arg\max_{W \in L} P(W \mid O)$
- Bayes's rule: $\hat{W} = \arg\max_{W \in L} \frac{P(O \mid W)P(W)}{P(O)} \propto \arg\max_{W \in L} P(O \mid W)P(W)$

observation likelihood (acoustic model) — prior (language model)

30

1st-Order Markov Model

- Markov Model M = (A, π) consists of
  - Discrete set of states, $s_1, s_2, \ldots, s_N$
  - $\boldsymbol{\pi}$ vector, where $\boldsymbol{\pi}_i = P(q_1 = s_i)$
  - State transition matrix, **A** = $\{a_{ij}\}$ where $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$
- The state transition matrix is fixed a priori and describes probabilities associated with a (completely-connected) graph of the states

31

HMM Summary

- An HMM contains 2 types of information:
  - Hidden states: $s_1, s_2, s_3, \ldots$
  - Observable states
    - In speech recognition, the vector quantization values in the input sequence, $\boldsymbol{O} = o_1, o_2, o_3, \ldots$
- An HMM, $\lambda = (A, B, \pi)$, contains 3 sets of probabilities:
  - $\boldsymbol{\pi}$ vector, $\boldsymbol{\pi} = (\pi_i)$
  - State transition matrix, **A** = (a_{ij}) where $a_{ij} = P(q_t = s_i \mid q_{t-1} = s_j)$
  - Observation likelihood matrix, **B** = $b_j(o_k) = P(y_t = o_k \mid q_t = s_j)$

32

Face Detection

Viola-Jones face detection algorithm, boosting ensemble learning, weak classifier, decision stump, weighted-majority classifier, Adaboost algorithm

33

Viola-Jones Algorithm

- Compute *lots* of very simple features
- Efficiently choose the best features
- Each feature is used to define a "**weak classifier**" (aka "**decision stump**")
- Combine weak classifiers into an **ensemble classifier** based on **boosting (Adaboost)**
- Learn multiple ensemble classifiers and "cascade" them together to improve classification accuracy and speed

34

AdaBoost Algorithm

Given a set of training windows labelled +1 or -1, initially give equal weight to each training example
Repeat *T* times

1. Select *best* weak classifier (decision stump) (i.e., one with minimum total weighted error on all training examples)
2. Increase weights of the examples misclassified by current weak classifier

- Each round greedily selects the best feature (i.e., decision stump) given all previously selected features and weighted training examples
- Final classifier combines the weak classifiers by their weighted-majority class

35

Viola-Jones Cascaded Classifier

- A *T*=1 feature classifier achieves 100% detection rate and about 50% false positive rate
- A *T*=5 feature classifier achieves 100% detection rate and 40% false positive rate (20% cumulative)
  - using data from previous stage
- A *T*=20 feature classifier achieve 100% detection rate with 10% false positive rate (2% cumulative)

36

38