Topic 0 – Intro and Important Concepts

An Unbiased Learner

Intuition: Choose that can express every teachable concept (i.e., is the power set of )

1. Consider = disjunction, conjunctions and negations of out earlier hypotheses in
   * S ← , where and are positive training instances
   * G ← , where and are negative training instances
2. Need training examples for every input instance in to converge to the target concept, eventually

Limitation: cannot classify new unobserved input instances, or cannot generalize beyond observed training examples

Inductive vs Deductive

Inductive is not provably correct, guesses unknown

Deductive is provably correct based on memory.

Active vs passive Learning

Active learning can select input to use, actively selects what to learn

Passive learning just takes in everything

Inductive Bias

Let denote the classification of input instance by some learning algorithm after learning from training examples

**Definition:** The **inductive bias** of is a minimal set of assertions such that for any target concept and training examples ,

Alternatively, knowledge space results in query space.

We put preference for some hypotheses, we do not restrict the hypothesis space

Occam’s Razor

We prefer short /simple hypotheses. We prefer a model with less assumption. Simple means good.

Arguments in favor:

* Fewer short hypotheses than long hypotheses (so a short hypothesis that comes out is very likely true, long hypothesis can be one of many)
  + Short / simple hypothesis that fits data unlikely to be a coincidence
  + Long / complex hypothesis that fits data may be a coincidence

Arguments opposing:

* Many ways to define small sets of hypotheses (e.g., all trees with a prime number of nodes that use attributes beginning with “Z”) {some weird restriction that is very complex}
* Small sets of short/simple hypothesis can be obtained using different hypothesis representations

Just a principal, a bit fuzzy. Used to explain decisions we make. If we do not follow can cause overfitting.

Overfitting

**Definition:** Hypothesis **overfits** the set of training examples iff

Where and denotes the errors of over and set of examples corresponding to instance space .

performs well on training data than however performs worse on data than .

Can be caused by noise or error. Can be caused by limited data

How to avoid overfitting?

* Stop growing DT when expanding a node is not statistically significant
* Allow DT to grow and overfit the data, then post prune it.

Use methods like pruning, k-fold, leave one out

How to select “best” DT?

* Measure performance over training examples / data
* Measure performance over a separate validation dataset
* MDL: minimize size(tree) and size(misclassifications)
  + Use extra variable, regulating value, to minimize size

**Reduced-Error Pruning**

What is pruning? Remove all subtrees and become a leaf node, then get value using PLURALITY-VALUE

Partition data into training and validation sets

Do until further pruning if harmful

1. Evaluate impact on validation set of pruning each possible node
2. Greedily remove the one that most improves the validation set accuracy
   * Produce smallest version of most accurate subtree

**Rule-Post-Pruning**

convert learned DT to an equivalent set of rules by creating one rule for each path form the root to a leaf

* Prune (generalize) each rule by removing any precondition that improves its estimated accuracy
* Can then sort pruned rules by estimated accuracy into desired sequence for use when classifying unobserved instances.

Inductive Learning Assumption:

* Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.
* If we train the model well, it can guess well. This is how we find values for inputs we have not previously seen.

Topic 1 – Concept Learning

**Definition**: An input instance **satisfies** (all constraints of) a hypothesis . In other words, h classifies as a positive example.

**Definition:** A hypothesis is **consistent** with a set of training examples iff for all . In other words, correctly classifies the training examples.

**Definition:** is **more general than or equal to**  (denoted ) iff any input instance that satisfies also satisfies .

relation defines a partial ordering (reflexive, antisymmetric and transitive) over and not a total ordering.

**Definition:**  is **(Strictly) more general than**  (denoted by ) iff and .

**Definition:**  is **more specific than**  iff is more general than .

*Concept*: Boolean valued function over a set of input instances (each comprising input attributes).

*Concept learning:* is a form of supervised learning. Infer an unknown Boolean-valued function from training example. Search for a hypothesis that is consistent with

How to represent a hypothesis

Hypothesis is a conjunction of constraints on input attributes.

Each constraint can be:

* A specific value (“Water=warm”)
* Don’t care (?)
* No value allowed ()

Every hypothesis containing 1 or more null (∅) symbols represents an empty set of input instance. Hence classifying every instance as a negative example.

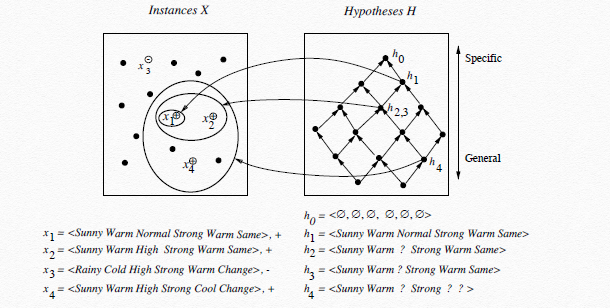
Synthetically distinct hypothesis: (all possible values + 1(?) + 1(∅)) for each input instance

Semantically distinct hypothesis: (all possible values + 1(?)) for each input instance + 1, all empty instance is the same

Find-S

Find the most specific hypothesis (usually all null ). Whenever it wrongly classifies a positive training example as negative, “minimally” generalize it to satisfy its input instance.

1. Initialize to most specific Hypothesis in
2. For each positive training instance
   * For each attribute constraint in :
     1. If satisfies constraint in , do noting
     2. Else replace in by the next more general constraint that is satisfied by
3. Output hypothesis



**Proposition 1:**  if consistent with iff every positive training instance satisfies and every negative instance does not satisfy

**Proposition 2:** Suppose that . Then, is consistent with

Limitations of Find-S

* Can’t tell whether Find-S has learned target concept
* Can’t tell when training examples are inconsistent (i.e., contains error or noise)
* Picks a maximally specific
* Depending on , there might be several (vector space)

Version Spaces

**Definition:** The **Version Space** with respect to hypothesis space and training examples , is the subset of hypotheses from consistent with

* If , then a large enough can reduce to
* If D is insufficient, then represents the uncertainty of what the target concept is
* contains all consistent hypotheses, including maximally specific hypotheses

List-Then-Eliminate Algorithm

Intuition: List all hypotheses in Then, eliminate any hypothesis found inconsistent with any training example.

1. a list containing every hypothesis in
2. For each training example
   * Remove from any hypothesis for which
3. Output the list of hypotheses in

Limitation: Prohibitively expensive to exhaustively enumerate all hypotheses in finite

**Definition:** The **general boundary**  of is the set of maximally general members of that is consistent with .

**Definition:** The **specific boundary**  of is the set of maximally specific members of that is consistent with .

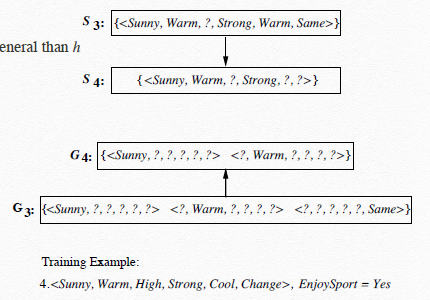
Every member of Version Space lies between these boundaries

Vector Space representation theorem:

Candidate Elimination Algorithm

Intuition: Start with most general and specific hypotheses. Each training example “minimally” generalizes and specializes to remove inconsistent hypotheses from version space.

1. maximally general hypotheses in (all “?”)
2. maximally specific hypotheses in (all “∅”)
3. For each training example
   * If is a positive example
     1. Remove from any hypothesis inconsistent with
     2. For each not consistent with
        + Remove from
        + Add to all minimal generalizations of such that is consistent with , and some members of is more general than
        + Remove from any hypothesis that is more general than another hypothesis in
   * If is a negative example
     1. Remove from any hypothesis inconsistent with
     2. For each not consistent with
        + Remove from
        + Add to all minimal specializations of such that is consistent with , and some members of is more specific than
        + Remove from any hypothesis that is more general than another hypothesis in



Properties of Candidate-Elimination

* If there is Error or Noise in training data
  + and reduced to ∅ with sufficiently large data
* Insufficiently expressive hypothesis representation (input instance not representative, some variables are missing)
  + biased hypothesis space → ? → and also reduced to ∅ with sufficiently large data
* What input instance should an active learner query next for a training example. (actively select training example to use)
  + Query input instance that satisfies exactly half of hypotheses in Version Space (if possible)
  + Version Space reduces by half with each training example, hence requiring at least examples to find target concept

**Proposition 3:** An input instance satisfies every hypothesis in iff satisfies every member of .

**Proposition 4:** An input instance satisfies none of the hypotheses in iff satisfies none of the members of .

* How to classify unobserved input instance? What degree of confidence?
  + Majority vote what is the most probable classification, assuming all hypothesis in are equally probable *a priori*

Inductive Bias of Candidate-Elimination

Assumption: Candidate-Elimination outputs a classification of input instance if this vote among hypotheses in is unanimously positive or negative, and does not output a classification otherwise.

Topic 2 – Decision Tree

Why study decision tree

|  |  |  |
| --- | --- | --- |
|  | Concept Learning | Decision Tree Learning |
| Target function / concept | Binary Outputs | Discrete outputs |
| Training data | Noise-free | Robust to noise |
| Hypothesis space | Restricted (hard bias) | Complete, expressive |
| Search strategy | Complete: version space  Refine search per example | Incomplete: prefer shorter trees (soft bias)  Refine search using all examples  No backtracking |
| Exploit Structure | General to specific ordering | Simple to complex ordering |

Another possible representation for hypotheses. At each level splits up the data based on some input attribute. Because of this, decision trees can express any function of the input. A leaf it the output (true / false).

Target Concept where each is a conjunction of attribute-value tests required to follow that path leading to a leaf with value true. . This results in substantially simpler than “true” decision tree – a more complex hypothesis isn’t justified by small amount of data.

**AIM**: Find a small tree consistent with the training examples

**IDEA**: greedily choose “most important” attribute as root of tree or subtree

PLURALITY-VALUE(examples):

**Return** majority voting of examples

**function** DECISION-TREE-LEARNING(examples, attributes, parent\_examples) **returns** tree:

**if** examples is empty **then return** PLURALITY-VALUE(parent\_examples)

**else if** all examples have the same classification **then return** the classification

**else if** attributes is empty **then return** PLURALITY-VALUE(examples)

**else**

tree ← a new decision with root test

**for each** value of **do**:

←

← DECISION-TREE-LEARNING(, attributes-, examples)

add branch to tree with label and subtree

**return** tree

Choosing “Most Important” Attribute

Intuition: A good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”

By using Information Theory, to implement the function in DECISION-TREE-LEARNING algorithm, use entropy to measure uncertainty of classification

Entropy measures uncertainty of certain data set

Define as entropy of Boolean r.v. that is true with probability :

Or simply, for a training set containing positive examples and negative examples, entropy of target concept on this set is

* If , then (maximum uncertainty)
* If or , then (no uncertainty)
* If (some uncertainty)

A chosen attribute divides the training set into subsets corresponding to the distinct values of . Each subset has positive and negative examples

Information gain of target concept from the attribute test on is the expected reduction in entropy:

Choose the attribute with the largest .

Hypothesis Space Search

Decision tree learning is guided by function. Information gain heuristic to search through the space of DTs from simplext to increasingly complex

Inductive Bias of DECISION-TREE-LEARNING

* Shorter trees are preferred
* Trees tat place high information gain attributes close to the root are preferred

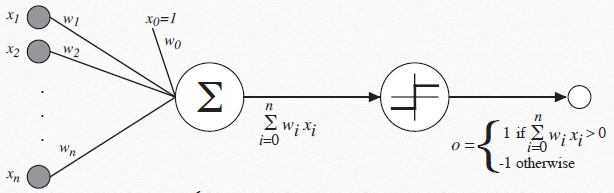
Common Problem Faced

* Continuous-valued attributes
  + Use a discrete valued input attribute to partition the values into discrete intervals.
* Attributes with many values
  + Some values can many possible values (ex: Date, then have one value for each)
  + will most likely pick this attribute (cos most likely each has all positive)
  + Solve by using GainRatio
* Attributes with differing costs
  + Attributes have a cost needed to get the data
  + Usually low-cost attributes tend to have more error / noise
  + Replace with:
    - , where determines importance of cost
* Missing attribute values
  + What if come examples are missing values for some attribute
  + Use training examples anyway and sort through DT
    - If node tests A, then assign most common value of among other examples sorted to node
    - Assign most common value of among other examples sorted to node with same value of output/target concept
    - Assign probability to each possible value
      * Assign fraction of example to each descendant in DT
* Then classify new unobserved input instances with missing attributes values in the same manner

Topic 3 – Neural Network

|  |  |  |
| --- | --- | --- |
|  | DT Learning | Neural Networks |
| Target function / concept | Discrete outputs | Discrete or real vector |
| Input instance | Discrete | Discrete or real high dimension |
| Training data | Robust to noise | Robust to noise |
| Hypothesis space | Complete, expressive | Restricted: #hidden unit  (hard bias), expressive |
| Search strategy | Incomplete: prefer shorter trees (soft bias)  Refine search using all examples  No backtracking | Incomplete: prefer smaller weights (soft bias)  Gradient ascent  Batch node: all examples  Stochastic: min-batches |
| Training time | Short | Long |
| Prediction time | Fast | Fast |
| Interpretability | White-box | Black-box |

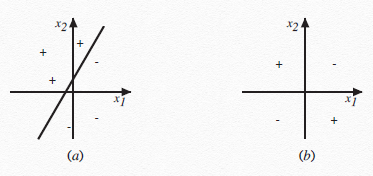
Perceptron Unit



Notice is all the input attributes. We have an extra and to act as biased input and biased weights. To make it more expressive

Using vector notation:

We want to search for a hypothesis



For graph (a), the line is .

The weight vector, is the orthogonal vector point towards the positive examples.

For graph(a), it points to the negative direction of axis, then we know that is negative. Similarly, is positive.

We need and so that the line does not necessarily cross the origin.

If line is above origin, is negative. If line is below origin, is positive

Figure (a) is **linearly separable,** there exists a horizontal line that can separate the examples. The line is not unique.

Figure (b) is **linearly non-separable,** no horizontal line can separate the examples

Perceptron Training Rule

**Idea:** Initialize randomly, apply perceptron training rule to every training example, and iterate thru all training examples till is consistent. Update weights if it is not consistent with a training example. We iterate through because we can stop at any time, so that we can stop the learning at any moment, if needed.

For where:

* is target output for training example
* is perceptron output
* η is a small positive constant (eg.1) called learning rate.
  + This value is usually small and decreases over time.

This algorithm is guaranteed to converge if training examples are linearly separable and η is sufficiently small.

Gradient Descent

Can be used even if linearly non seperable

**Idea:** Search to find weight vector that “best fits” the (possibly linearly non-separable) training examples.

Usually need to search a very large space, possibly infinite . Gradient Descent can do gradient climbing to find

Learn that minimizes squared error/loss. This function needs to be differentiable.

Where is the set of training examples, and are target outputs and output of linear unit for training example respectively.

**Idea:** Find that minimizes by first initializing it randomly and then repeatedly updating it in the direction of steepest descent

Gradient:

Training rule:

|  |  |
| --- | --- |
|  |  |

That is

|  |  |
| --- | --- |
|  |  |

We put negative because we want to find the steepest descent, gets us the greatest increase

**Gradient Descent Algorithm**

Idea: Initialize randomly, apply linear unit training rule to all training examples and repeat

Gradient-Descent(, η )

Initialize each to some small random value

Until termination condition is met, do

Initialize each to zero.

For each , do:

Input instance to linear unit and compute output

For each linear unit weight , do

For each linear unit weight , do

Here, is the output of a linear unit.

Gradient Descent vs Stochastic (incremental) Gradient Descent

|  |  |
| --- | --- |
| Batch Gradient Descent | Stochastic Gradient Descent |
| 1. Compute gradient 2. where | For each training example   1. Compute gradient 2. where |

Batch gradient is defined over all training examples, Stochastic Gradient Descent is defined for each example

SGD uses sampling, thus it is an unbiased estimator of true gradient

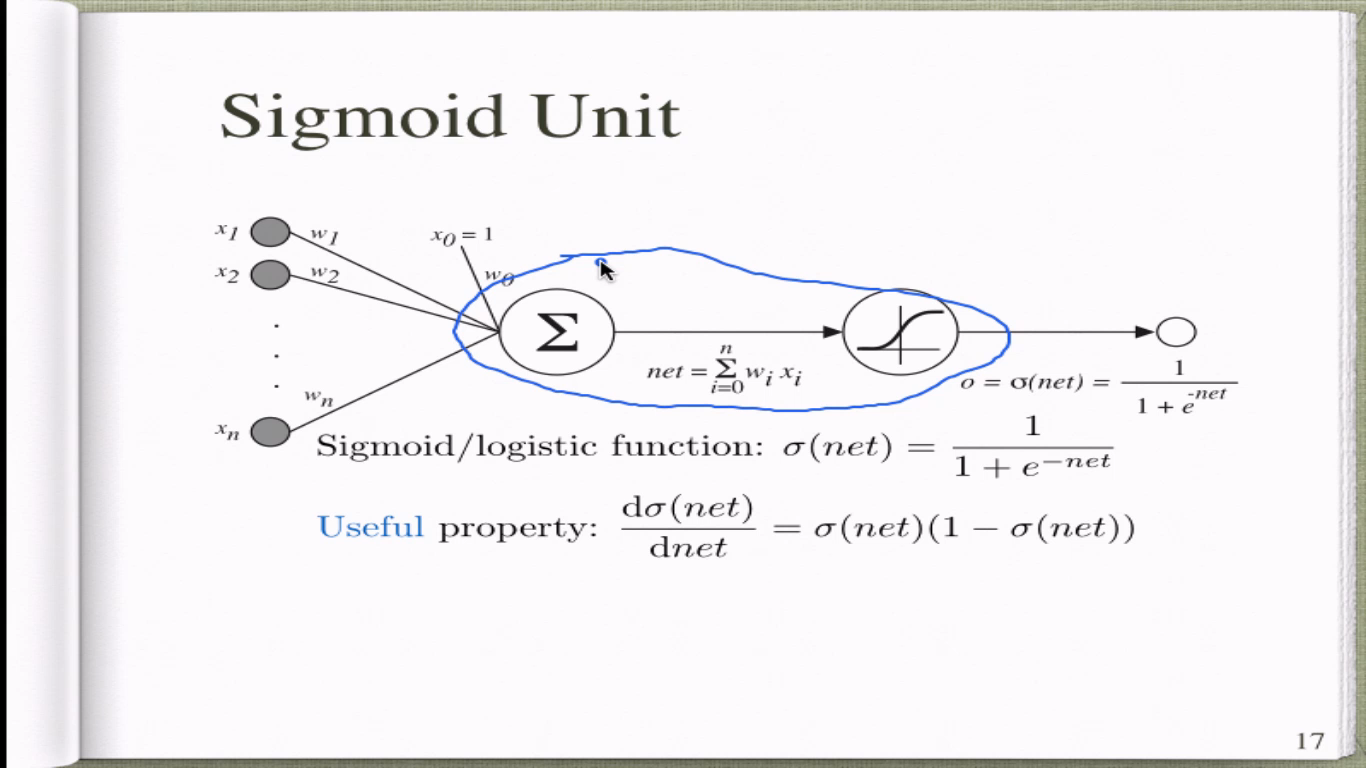
SGD can approximate batch GD arbitrarily close if learning rate η is sufficiently small

SGD usefulness:

* Computational cost on big data: Although it is linear time. Using SGD, we can decide how many data points to process at each iteration, resulting in constant time at each iteration.
* Any time performance: can get some performance at any time. Even though haven’t process all data.
* Economical cost: can buy data based on how much we have or need.
* Can escape local minima: each iteration advises you to another direction to adjust to. Batch will point to one direction

**General idea:** Objective function (differentiable wrt model parameters w) can be decomposed into a sum of terms, each depending on a subset of training examples. So that we can use Stochastic Gradient Descent. Can use other formula to find expected loss.

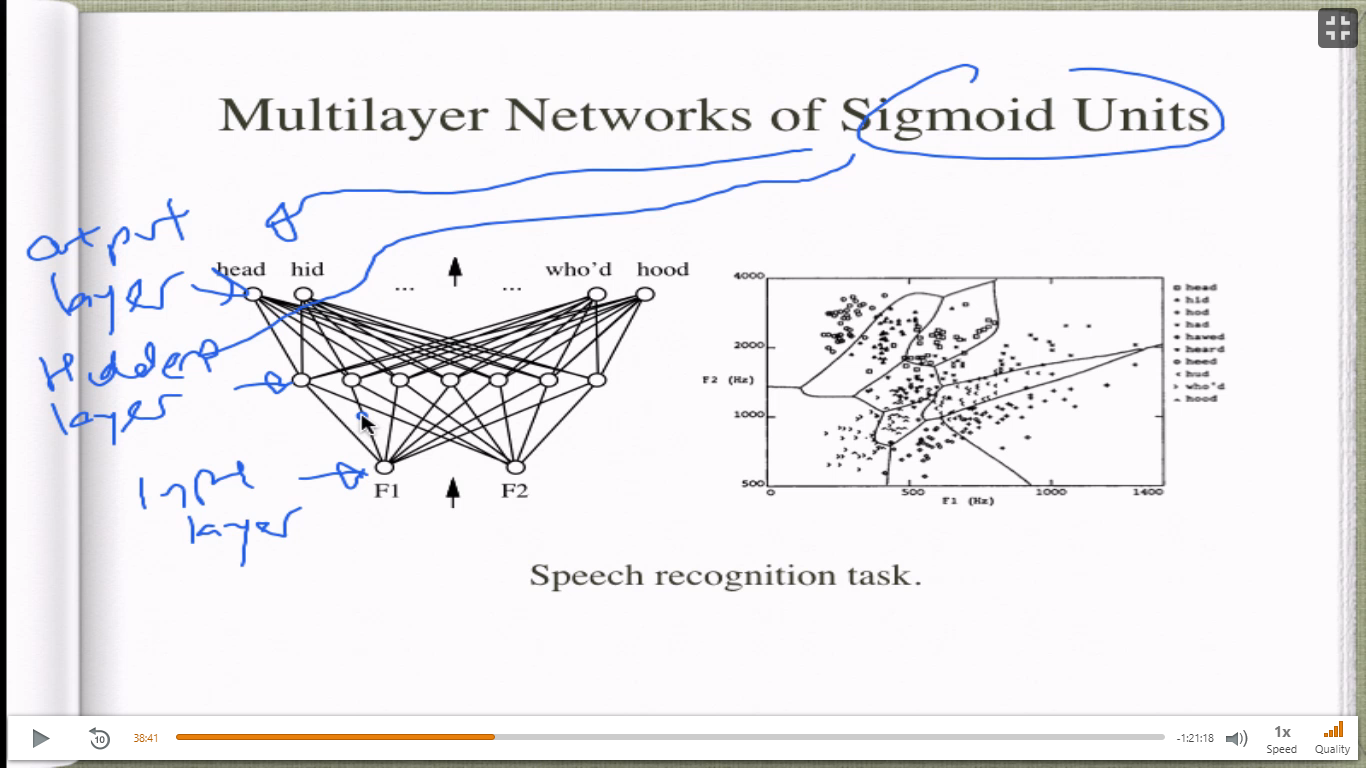
Sigmoid Unit



Input to activation function is the net/sum. Output ranges from 0 to 1.

Gradient descent can be derived to train it

Multilayer Networks of Sigmoid Units



Outputs for one hidden layer will be the input for next layer.

Sigmoid in all layers except input layer.

We choose the output with the highest value.

Gradient descent can be derived to train multilayer using backpropagation

Glossary

* denotes the Hypotheses space, which is the set of all semantically distinct hypotheses
* D denotes the set of training data
* *A priori:* A given proposition is knowable *a priori* if it can be known independent of any experience other than the experience of learning the language in which the proposition is expressed
* *A Posteriori:* a proposition that is knowable *a posteriori* is known on the basis of experience