Artificial Intelligence-BSCE-306L

Module 7

Communicating, Perceiving, and Acting

Dr . Saurabh Agrawal

Faculty Id: 20165

School of Computer Science and Engineering

VIT, Vellore-632014

Tamil Nadu, India

Outline

- Communication
- Fundamentals of Language **(RN_22.1)**
- Probabilistic Language Processing **(RN_22.2)**
- Information Retrieval **(RN_22.3)**
- Information Extraction **(RN_22.4)**
- Perception **(RN_24.2)**
- Image Formation **(RN_24.1)**
- Object Recognition **(RN_24.3)**

Communication

- Homo sapiens is set apart from other species by the capacity for language.
- Somewhere around 100,000 years ago, humans learned how to speak, and about 7,000 years ago learned to write.
- Although chimpanzees, dolphins, and other animals have shown vocabularies of hundreds of signs, only humans can reliably communicate an unbounded number of qualitatively different messages on any topic using discrete signs.
- Of course, there are other attributes that are uniquely human: no other species wears clothes, creates representational art, or watches three hours of television a day.
- But when Alan Turing proposed his Test, he based it on language, not art or TV.
- There are two main reasons why we want our computer agents to be able to process natural languages: first, to communicate with humans, and second, to acquire information from written language.

Communication

- There are over a trillion pages of information on the Web, almost all of it in natural language.
- An agent that wants to do knowledge acquisition needs to understand (at least partially) the ambiguous, messy languages that humans use.
- We examine the problem from the point of view of specific information-seeking tasks: text classification, information retrieval, and information extraction.
- One common factor in addressing these tasks is the use of language models: models that predict the probability distribution of language expressions.

Fundamentals of Language (Language Model)

Formal languages, such as the programming languages Java or Python, have precisely defined language models. A **language** can be defined as a set of strings; "`print(2 + 2)`" is a legal program in the language Python, whereas "`2)+(2 print`" is not. Since there are an infinite number of legal programs, they cannot be enumerated; instead they are specified by a set of rules called a **grammar**. Formal languages also have rules that define the meaning or **semantics** of a program; for example, the rules say that the "meaning" of "`2 + 2`" is 4, and the meaning of "`1/0`" is that an error is signaled.

Natural languages, such as English or Spanish, cannot be characterized as a definitive set of sentences. Everyone agrees that "Not to be invited is sad" is a sentence of English, but people disagree on the grammaticality of "To be not invited is sad." Therefore, it is more fruitful to define a natural language model as a probability distribution over sentences rather than a definitive set. That is, rather than asking if a string of *words* is or is not a member of the set defining the language, we instead ask for $P(S = words)$—what is the probability that a random sentence would be *words*.

Fundamentals of Language (Language Model)

Natural languages are also **ambiguous**. "He saw her duck" can mean either that he saw a waterfowl belonging to her, or that he saw her move to evade something. Thus, again, we cannot speak of a single meaning for a sentence, but rather of a probability distribution over possible meanings.

Finally, natural languages are difficult to deal with because they are very large, and constantly changing. Thus, our language models are, at best, an approximation. We start with the simplest possible approximations and move up from there.

Fundamentals of Language (N-gram Character Model)

Ultimately, a written text is composed of **characters**—letters, digits, punctuation, and spaces in English (and more exotic characters in some other languages). Thus, one of the simplest language models is a probability distribution over sequences of characters. As in Chapter 15, we write $P(c_{1:N})$ for the probability of a sequence of N characters, c_1 through c_N. In one Web collection, $P(\text{"the"}) = 0.027$ and $P(\text{"zgq"}) = 0.000000002$. A sequence of written symbols of length n is called an n-gram (from the Greek root for writing or letters), with special case "unigram" for 1-gram, "bigram" for 2-gram, and "trigram" for 3-gram. A model of the probability distribution of n-letter sequences is thus called an n**-gram model**. (But be careful: we can have n-gram models over sequences of words, syllables, or other units; not just over characters.)

Fundamentals of Language (N-gram Character Model)

An n-gram model is defined as a **Markov chain** of order $n-1$. Recall from page 568 that in a Markov chain the probability of character c_i depends only on the immediately preceding characters, not on any other characters. So in a trigram model (Markov chain of order 2) we have

$$P(c_i \mid c_{1:i-1}) = P(c_i \mid c_{i-2:i-1}) \,.$$

We can define the probability of a sequence of characters $P(c_{1:N})$ under the trigram model by first factoring with the chain rule and then using the Markov assumption:

$$P(c_{1:N}) = \prod_{i=1}^{N} P(c_i \mid c_{1:i-1}) = \prod_{i=1}^{N} P(c_i \mid c_{i-2:i-1}) \,.$$

Fundamentals of Language (N-gram Character Model)

For a trigram character model in a language with 100 characters, $\mathbf{P}(C_i|C_{i-2:i-1})$ has a million entries, and can be accurately estimated by counting character sequences in a body of text of 10 million characters or more. We call a body of text a **corpus** (plural *corpora*), from the Latin word for *body*.

What can we do with n-gram character models? One task for which they are well suited is **language identification**: given a text, determine what natural language it is written in. This is a relatively easy task; even with short texts such as "Hello, world" or "Wie geht es dir," it is easy to identify the first as English and the second as German. Computer systems identify languages with greater than 99% accuracy; occasionally, closely related languages, such as Swedish and Norwegian, are confused.

Fundamentals of Language (N-gram Character Model)

One approach to language identification is to first build a trigram character model of each candidate language, $P(c_i \mid c_{i-2:i-1}, \ell)$, where the variable ℓ ranges over languages. For each ℓ the model is built by counting trigrams in a corpus of that language. (About 100,000 characters of each language are needed.) That gives us a model of $\mathbf{P}(\mathit{Text} \mid \mathit{Language})$, but we want to select the most probable language given the text, so we apply Bayes' rule followed by the Markov assumption to get the most probable language:

$$
\begin{aligned}
\ell^* &= \operatorname*{argmax}_{\ell} P(\ell \mid c_{1:N}) \\
&= \operatorname*{argmax}_{\ell} P(\ell) P(c_{1:N} \mid \ell) \\
&= \operatorname*{argmax}_{\ell} P(\ell) \prod_{i=1}^{N} P(c_i \mid c_{i-2:i-1}, \ell)
\end{aligned}
$$

Fundamentals of Language (N-gram Character Model)

The trigram model can be learned from a corpus, but what about the prior probability $P(\ell)$? We may have some estimate of these values; for example, if we are selecting a random Web page we know that English is the most likely language and that the probability of Macedonian will be less than 1%. The exact number we select for these priors is not critical because the trigram model usually selects one language that is several orders of magnitude more probable than any other.

Fundamentals of Language (N-gram Character Model)

Other tasks for character models include spelling correction, genre classification, and named-entity recognition. Genre classification means deciding if a text is a news story, a legal document, a scientific article, etc. While many features help make this classification, counts of punctuation and other character n-gram features go a long way (Kessler *et al.*, 1997). Named-entity recognition is the task of finding names of things in a document and deciding what class they belong to. For example, in the text "Mr. Sopersteen was prescribed aciphex," we should recognize that "Mr. Sopersteen" is the name of a person and "aciphex" is the name of a drug. Character-level models are good for this task because they can associate the character sequence "ex␣" ("ex" followed by a space) with a drug name and "steen␣" with a person name, and thereby identify words that they have never seen before.

Probabilistic Language Processing (Text Classification)

- We now consider in depth the task of text classification, also known as categorization: given a text of some kind, decide which of a predefined set of classes it belongs to.
- Language identification and genre classification are examples of text classification, as is sentiment analysis (classifying a movie or product review as positive or negative) and spam detection (classifying an email message as spam or not-spam).
- Since “not-spam” is awkward, researchers have coined the term ham for not-spam.
- We can treat spam detection as a problem in supervised learning.
- A training set is readily available: the positive (spam) examples are in my spam folder, the negative (ham) examples are in my inbox.

Probabilistic Language Processing (Text Classification)

- Here is an excerpt:

Spam: Wholesale Fashion Watches -57% today. Designer watches for cheap ...
Spam: You can buy ViagraFr$1.85 All Medications at unbeatable prices! ...
Spam: WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US ...
Spam: Sta.rt earn*ing the salary yo,u d-eserve by o'btaining the prope,r crede'ntials!

Ham: The practical significance of hypertree width in identifying more ...
Ham: Abstract: We will motivate the problem of social identity clustering: ...
Ham: Good to see you my friend. Hey Peter, It was good to hear from you. ...
Ham: PDS implies convexity of the resulting optimization problem (Kernel Ridge ...

Probabilistic Language Processing (Text Classification)

From this excerpt we can start to get an idea of what might be good features to include in the supervised learning model. Word n-grams such as “for cheap” and “You can buy” seem to be indicators of spam (although they would have a nonzero probability in ham as well). Character-level features also seem important: spam is more likely to be all uppercase and to have punctuation embedded in words. Apparently the spammers thought that the word bigram “you deserve” would be too indicative of spam, and thus wrote “yo,u d-eserve” instead. A character model should detect this. We could either create a full character n-gram model of spam and ham, or we could handcraft features such as “number of punctuation marks embedded in words.”

Probabilistic Language Processing (Text Classification)

Note that we have two complementary ways of talking about classification. In the language-modeling approach, we define one n-gram language model for $\mathbf{P}(Message \mid spam)$ by training on the spam folder, and one model for $\mathbf{P}(Message \mid ham)$ by training on the inbox. Then we can classify a new message with an application of Bayes' rule:

$$\operatorname*{argmax}_{c \in \{spam, ham\}} P(c \mid message) = \operatorname*{argmax}_{c \in \{spam, ham\}} P(message \mid c)\, P(c)\ .$$

where $P(c)$ is estimated just by counting the total number of spam and ham messages. This approach works well for spam detection, just as it did for language identification.

Probabilistic Language Processing (Text Classification)

In the machine-learning approach we represent the message as a set of feature/value pairs and apply a classification algorithm h to the feature vector $\mathbf{X}$. We can make the language-modeling and machine-learning approaches compatible by thinking of the n-grams as features. This is easiest to see with a unigram model. The features are the words in the vocabulary: "a," "aardvark," ..., and the values are the number of times each word appears in the message. That makes the feature vector large and sparse. If there are 100,000 words in the language model, then the feature vector has length 100,000, but for a short email message almost all the features will have count zero. This unigram representation has been called the **bag of words** model. You can think of the model as putting the words of the training corpus in a bag and then selecting words one at a time. The notion of order of the words is lost; a unigram model gives the same probability to any permutation of a text. Higher-order n-gram models maintain some local notion of word order.

Probabilistic Language Processing (Text Classification)

With bigrams and trigrams the number of features is squared or cubed, and we can add in other, non-n-gram features: the time the message was sent, whether a URL or an image is part of the message, an ID number for the sender of the message, the sender's number of previous spam and ham messages, and so on. The choice of features is the most important part of creating a good spam detector—more important than the choice of algorithm for processing the features. In part this is because there is a lot of training data, so if we can propose a feature, the data can accurately determine if it is good or not. It is necessary to constantly update features, because spam detection is an **adversarial task**; the spammers modify their spam in response to the spam detector's changes.

Probabilistic Language Processing (Text Classification)

It can be expensive to run algorithms on a very large feature vector, so often a process of **feature selection** is used to keep only the features that best discriminate between spam and ham. For example, the bigram "of the" is frequent in English, and may be equally frequent in spam and ham, so there is no sense in counting it. Often the top hundred or so features do a good job of discriminating between classes.

Once we have chosen a set of features, we can apply any of the supervised learning techniques we have seen; popular ones for text categorization include k-nearest-neighbors, support vector machines, decision trees, naive Bayes, and logistic regression. All of these have been applied to spam detection, usually with accuracy in the 98%–99% range. With a carefully designed feature set, accuracy can exceed 99.9%.

Probabilistic Language Processing (Classification by Data Compression)

Another way to think about classification is as a problem in **data compression**. A lossless compression algorithm takes a sequence of symbols, detects repeated patterns in it, and writes a description of the sequence that is more compact than the original. For example, the text "0.142857142857142857" might be compressed to "0.[142857]*3." Compression algorithms work by building dictionaries of subsequences of the text, and then referring to entries in the dictionary. The example here had only one dictionary entry, "142857."

In effect, compression algorithms are creating a language model. The LZW algorithm in particular directly models a maximum-entropy probability distribution. To do classification by compression, we first lump together all the spam training messages and compress them as a unit. We do the same for the ham. Then when given a new message to classify, we append it to the spam messages and compress the result. We also append it to the ham and compress that. Whichever class compresses better—adds the fewer number of additional bytes for the new message—is the predicted class. The idea is that a spam message will tend to share dictionary entries with other spam messages and thus will compress better when appended to a collection that already contains the spam dictionary.

Probabilistic Language Processing (Classification by Data Compression)

Experiments with compression-based classification on some of the standard corpora for text classification—the 20-Newsgroups data set, the Reuters-10 Corpora, the Industry Sector corpora—indicate that whereas running off-the-shelf compression algorithms like gzip, RAR, and LZW can be quite slow, their accuracy is comparable to traditional classification algorithms. This is interesting in its own right, and also serves to point out that there is promise for algorithms that use character n-grams directly with no preprocessing of the text or feature selection: they seem to be captiring some real patterns.

Information Retrieval

- Information retrieval is the task of finding documents that are relevant to a user's need for information.
- The best-known examples of information retrieval systems are search engines on the World Wide Web.
- A Web user can type a query such as [AI book]2 into a search engine and see a list of relevant pages.

Information Retrieval

- An information retrieval (henceforth IR) system can be characterized by:

1. **A corpus of documents.** Each system must decide what it wants to treat as a document: a paragraph, a page, or a multipage text.
2. **Queries posed in a query language**. A query specifies what the user wants to know. The query language can be just a list of words, such as [AI book]; or it can specify a phrase of words that must be adjacent, as in ["AI book"]; it can contain Boolean operators as in [AI AND book]; it can include non-Boolean operators such as [AI NEAR book] or [AI book site:www.aaai.org].
3. **A result set.** This is the subset of documents that the IR system judges to be **relevant** to the query. By *relevant*, we mean likely to be of use to the person who posed the query, for the particular information need expressed in the query.
4. **A presentation of the result set.** This can be as simple as a ranked list of document titles or as complex as a rotating color map of the result set projected onto a three-dimensional space, rendered as a two-dimensional display.

Information Retrieval Scoring Function

Most IR systems have abandoned the Boolean model and use models based on the statistics of word counts. We describe the **BM25 scoring function**, which comes from the Okapi project of Stephen Robertson and Karen Sparck Jones at London's City College, and has been used in search engines such as the open-source Lucene project.

A scoring function takes a document and a query and returns a numeric score; the most relevant documents have the highest scores. In the BM25 function, the score is a linear weighted combination of scores for each of the words that make up the query. Three factors affect the weight of a query term: First, the frequency with which a query term appears in a document (also known as TF for term frequency). For the query [farming in Kansas], documents that mention "farming" frequently will have higher scores. Second, the inverse document frequency of the term, or IDF. The word "in" appears in almost every document, so it has a high document frequency, and thus a low inverse document frequency, and thus it is not as important to the query as "farming" or "Kansas." Third, the length of the document. A million-word document will probably mention all the query words, but may not actually be about the query. A short document that mentions all the words is a much better candidate.

Information Retrieval Scoring Function

The BM25 function takes all three of these into account. We assume we have created an index of the N documents in the corpus so that we can look up $TF(q_i, d_j)$, the count of the number of times word q_i appears in document d_j. We also assume a table of document frequency counts, $DF(q_i)$, that gives the number of documents that contain the word q_i. Then, given a document d_j and a query consisting of the words $q_{1:N}$, we have

$$BM25(d_j, q_{1:N}) = \sum_{i=1}^{N} IDF(q_i) \cdot \frac{TF(q_i, d_j) \cdot (k+1)}{TF(q_i, d_j) + k \cdot (1 - b + b \cdot \frac{|d_j|}{L})},$$

where $|d_j|$ is the length of document d_j in words, and L is the average document length in the corpus: $L = \sum_i |d_i|/N$. We have two parameters, k and b, that can be tuned by cross-validation; typical values are $k = 2.0$ and $b = 0.75$. $IDF(q_i)$ is the inverse document

Information Retrieval Scoring Function

frequency of word q_i, given by

$$IDF(q_i) = \log \frac{N - DF(q_i) + 0.5}{DF(q_i) + 0.5} .$$

Of course, it would be impractical to apply the BM25 scoring function to every document in the corpus. Instead, systems create an **index** ahead of time that lists, for each vocabulary word, the documents that contain the word. This is called the **hit list** for the word. Then when given a query, we intersect the hit lists of the query words and only score the documents in the intersection.

Information Retrieval System Evaluation

How do we know whether an IR system is performing well? We undertake an experiment in which the system is given a set of queries and the result sets are scored with respect to human relevance judgments. Traditionally, there have been two measures used in the scoring: recall and precision. We explain them with the help of an example. Imagine that an IR system has returned a result set for a single query, for which we know which documents are and are not relevant, out of a corpus of 100 documents. The document counts in each category are given in the following table:

	In result set	Not in result set
Relevant	30	20
Not relevant	10	40

Information Retrieval System Evaluation

Precision measures the proportion of documents in the result set that are actually relevant. In our example, the precision is $30/(30+10) = .75$. The false positive rate is $1 - .75 = .25$. **Recall** measures the proportion of all the relevant documents in the collection that are in the result set. In our example, recall is $30/(30+20) = .60$. The false negative rate is $1 - .60 = .40$. In a very large document collection, such as the World Wide Web, recall is difficult to compute, because there is no easy way to examine every page on the Web for relevance. All we can do is either estimate recall by sampling or ignore recall completely and just judge precision. In the case of a Web search engine, there may be thousands of documents in the result set, so it makes more sense to measure precision for several different sizes, such as "P@10" (precision in the top 10 results) or "P@50," rather than to estimate precision in the entire result set.

It is possible to trade off precision against recall by varying the size of the result set returned. In the extreme, a system that returns every document in the document collection is guaranteed a recall of 100%, but will have low precision. Alternately, a system could return a single document and have low recall, but a decent chance at 100% precision. A summary of both measures is the F_1 score, a single number that is the harmonic mean of precision and recall, $2PR/(P+R)$.

Information Extraction

- Information extraction is the process of acquiring knowledge by skimming a text and looking for occurrences of a particular class of object and for relationships among objects.
- A typical task is to extract instances of addresses from Web pages, with database fields for street, city, state, and zip code; or instances of storms from weather reports, with fields for temperature, wind speed, and precipitation.
- In a limited domain, this can be done with high accuracy.
- As the domain gets more general, more complex linguistic models and more complex learning techniques are necessary.
- We will see how to define complex language models of the phrase structure (noun phrases and verb phrases) of English.
- But so far there are no complete models of this kind, so for the limited needs of information extraction, we define limited models that approximate the full English model, and concentrate on just the parts that are needed for the task at hand.

Information Extraction

- The models we describe are approximations in the same way that the simple 1-CNF logical model is an approximations of the full, wiggly, logical model.

Information Extraction

The simplest type of information extraction system is an **attribute-based extraction** system that assumes that the entire text refers to a single object and the task is to extract attributes of that object. For example, we mentioned in Section 12.7 the problem of extracting from the text "IBM ThinkBook 970. Our price: $399.00" the set of attributes {Manufacturer=IBM, Model=ThinkBook970, Price=$399.00}. We can address this problem by defining a **template** (also known as a pattern) for each attribute we would like to extract. The template is defined by a finite state automaton, the simplest example of which is the **regular expression**, or regex. Regular expressions are used in Unix commands such as grep, in programming languages such as Perl, and in word processors such as Microsoft Word. The details vary slightly from one tool to another and so are best learned from the appropriate manual, but here we show how to build up a regular expression template for prices in dollars:

Information Extraction

`[0-9]`	matches any digit from 0 to 9
`[0-9]+`	matches one or more digits
`[.][0-9][0-9]`	matches a period followed by two digits
`([.][0-9][0-9])?`	matches a period followed by two digits, or nothing
`[$][0-9]+([.][0-9][0-9])?`	matches $249.99 or $1.23 or $1000000 or …

Templates are often defined with three parts: a prefix regex, a target regex, and a postfix regex. For prices, the target regex is as above, the prefix would look for strings such as "price:" and the postfix could be empty. The idea is that some clues about an attribute come from the attribute value itself and some come from the surrounding text.

Information Extraction

If a regular expression for an attribute matches the text exactly once, then we can pull out the portion of the text that is the value of the attribute. If there is no match, all we can do is give a default value or leave the attribute missing; but if there are several matches, we need a process to choose among them. One strategy is to have several templates for each attribute, ordered by priority. So, for example, the top-priority template for price might look for the prefix "our price:"; if that is not found, we look for the prefix "price:" and if that is not found, the empty prefix. Another strategy is to take all the matches and find some way to choose among them. For example, we could take the lowest price that is within 50% of the highest price. That will select $78.00 as the target from the text "List price $99.00, special sale price $78.00, shipping $3.00."

Information Extraction

If a regular expression for an attribute matches the text exactly once, then we can pull out the portion of the text that is the value of the attribute. If there is no match, all we can do is give a default value or leave the attribute missing; but if there are several matches, we need a process to choose among them. One strategy is to have several templates for each attribute, ordered by priority. So, for example, the top-priority template for price might look for the prefix "our price:"; if that is not found, we look for the prefix "price:" and if that is not found, the empty prefix. Another strategy is to take all the matches and find some way to choose among them. For example, we could take the lowest price that is within 50% of the highest price. That will select $78.00 as the target from the text "List price $99.00, special sale price $78.00, shipping $3.00."

One step up from attribute-based extraction systems are **relational extraction** systems, which deal with multiple objects and the relations among them. Thus, when these systems see the text "$249.99," they need to determine not just that it is a price, but also which object has that price. A typical relational-based extraction system is FASTUS, which handles news stories about corporate mergers and acquisitions. It can read the story

Information Extraction

Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.

and extract the relations:

$$e \in JointVentures \wedge Product(e, \text{“}golf\ clubs\text{”}) \wedge Date(e, \text{“}Friday\text{”})$$
$$\wedge\ Member(e, \text{“}Bridgestone\ Sports\ Co\text{”}) \wedge Member(e, \text{“}a\ local\ concern\text{”})$$
$$\wedge\ Member(e, \text{“}a\ Japanese\ trading\ house\text{”})\ .$$

A relational extraction system can be built as a series of **cascaded finite-state transducers.** That is, the system consists of a series of small, efficient finite-state automata (FSAs), where each automaton receives text as input, transduces the text into a different format, and passes it along to the next automaton. FASTUS consists of five stages:

1. Tokenization
2. Complex-word handling
3. Basic-group handling
4. Complex-phrase handling
5. Structure merging

Information Extraction

FASTUS's first stage is **tokenization**, which segments the stream of characters into tokens (words, numbers, and punctuation). For English, tokenization can be fairly simple; just separating characters at white space or punctuation does a fairly good job. Some tokenizers also deal with markup languages such as HTML, SGML, and XML.

The second stage handles **complex words**, including collocations such as "set up" and "joint venture," as well as proper names such as "Bridgestone Sports Co." These are recognized by a combination of lexical entries and finite-state grammar rules. For example, a company name might be recognized by the rule

CapitalizedWord+ ("Company" | "Co" | "Inc" | "Ltd")

Information Extraction

CapitalizedWord+ ("Company" | "Co" | "Inc" | "Ltd")

The third stage handles **basic groups**, meaning noun groups and verb groups. The idea is to chunk these into units that will be managed by the later stages. We will see how to write a complex description of noun and verb phrases in Chapter 23, but here we have simple rules that only approximate the complexity of English, but have the advantage of being representable by finite state automata. The example sentence would emerge from this stage as the following sequence of tagged groups:

```
1 NG: Bridgestone Sports Co.   10 NG: a local concern
2 VG: said                     11 CJ: and
3 NG: Friday                   12 NG: a Japanese trading house
4 NG: it                       13 VG: to produce
5 VG: had set up               14 NG: golf clubs
6 NG: a joint venture          15 VG: to be shipped
7 PR: in                       16 PR: to
8 NG: Taiwan                   17 NG: Japan
9 PR: with
```

Here NG means noun group, VG is verb group, PR is preposition, and CJ is conjunction.

Information Extraction

The fourth stage combines the basic groups into **complex phrases**. Again, the aim is to have rules that are finite-state and thus can be processed quickly, and that result in unambiguous (or nearly unambiguous) output phrases. One type of combination rule deals with domain-specific events. For example, the rule

Company+ SetUp JointVenture ("with" Company+)?

captures one way to describe the formation of a joint venture. This stage is the first one in the cascade where the output is placed into a database template as well as being placed in the output stream. The final stage **merges structures** that were built up in the previous step. If the next sentence says "The joint venture will start production in January," then this step will notice that there are two references to a joint venture, and that they should be merged into one. This is an instance of the **identity uncertainty problem** discussed in Section 14.6.3.

Information Extraction

In general, finite-state template-based information extraction works well for a restricted domain in which it is possible to predetermine what subjects will be discussed, and how they will be mentioned. The cascaded transducer model helps modularize the necessary knowledge, easing construction of the system. These systems work especially well when they are reverse-engineering text that has been generated by a program. For example, a shopping site on the Web is generated by a program that takes database entries and formats them into Web pages; a template-based extractor then recovers the original database. Finite-state information extraction is less successful at recovering information in highly variable format, such as text written by humans on a variety of subjects.

Perception

- Perception provides agents with information about the world they inhabit by interpreting the response of sensors.
- A sensor measures some aspect of the environment in a form that can be used as input by an agent program.
- The sensor could be as simple as a switch, which gives one bit telling whether it is on or off, or as complex as the eye.
- A variety of sensory modalities are available to artificial agents. Those they share with humans include vision, hearing, and touch.
- Modalities that are not available to the unaided human include radio, infrared, GPS, and wireless signals. Some robots do active sensing, meaning they send out a signal, such as radar or ultrasound, and sense the reflection of this signal off of the environment.
- Rather than trying to cover all of these, this chapter will cover one modality in depth: vision.

Perception

- We saw in our description of POMDPs that a model-based decision-theoretic agent in a partially observable environment has a sensor model: a probability distribution P(E | S) over the evidence that its sensors provide, given a state of the world.
- Bayes' rule can then be used to update the estimation of the state.
- For vision, the sensor model can be broken into two components: An object model describes the objects that inhabit the visual world—people, buildings, trees, cars, etc.
- The object model could include a precise 3D geometric model taken from a computer-aided design (CAD) system, or it could be vague constraints, such as the fact that human eyes are usually 5 to 7 cm apart.
- A rendering model describes the physical, geometric, and statistical processes that produce the stimulus from the world.

Perception

- Rendering models are quite accurate, but they are ambiguous.
- For example, a white object under low light may appear as the same color as a black object under intense light.
- A small nearby object may look the same as a large distant object.
- Without additional evidence, we cannot tell if the image that fills the frame is a toy Godzilla or a real monster.
- Ambiguity can be managed with prior knowledge—we know Godzilla is not real, so the image must be a toy—or by selectively choosing to ignore the ambiguity.
- For example, the vision system for an autonomous car may not be able to interpret objects that are far in the distance, but the agent can choose to ignore the problem, because it is unlikely to crash into an object that is miles away.

Perception

- A decision-theoretic agent is not the only architecture that can make use of vision sensors.
- For example, fruit flies (Drosophila) are in part reflex agents: they have cervical giant fibers that form a direct pathway from their visual system to the wing muscles that initiate an escape response—an immediate reaction, without deliberation.
- Flies and many other flying animals make used of a closed-loop control architecture to land on an object.
- The visual system extracts an estimate of the distance to the object, and the control system adjusts the wing muscles accordingly, allowing very fast changes of direction, with no need for a detailed model of the object.
- Compared to the data from other sensors (such as the single bit that tells the vacuum robot that it has bumped into a wall), visual observations are extraordinarily rich, both in the detail they can reveal and in the sheer amount of data they produce.

Perception

- A video camera for robotic applications might produce a million 24-bit pixels at 60 Hz; a rate of 10 GB per minute.
- The problem for a vision-capable agent then is: Which aspects of the rich visual stimulus should be considered to help the agent make good action choices, and which aspects should be ignored? Vision—and all perception—serves to further the agent's goals, not as an end to itself.
- We can characterize three broad approaches to the problem.
- The feature extraction approach, as exhibited by Drosophila, emphasizes simple computations applied directly to the sensor observations.
- In the recognition approach an agent draws distinctions among the objects it encounters based on visual and other information.
- Recognition could mean labeling each image with a yes or no as to whether it contains food that we should forage, or contains Grandma's face.

Perception

- Recognition could mean labeling each image with a yes or no as to whether it contains food that we should forage, or contains Grandma's face.
- Finally, in the reconstruction approach an agent builds a geometric model of the world from an image or a set of images.
- The last thirty years of research have produced powerful tools and methods for addressing these approaches.
- Understanding these methods requires an understanding of the processes by which images are formed.
- Therefore, we now cover the physical and statistical phenomena that occur in the production of an image.

Image Formation

- **Experiential Learning (Image Formation)**
- **Complete the topic on or before 01 May 2024.**
- **If you have any doubt then ask me in the class of 02 May 2024.**

Object Recognition

- We will study three useful image-processing operations: edge detection, texture analysis, and computation of optical flow.
- These are called “early” or “low-level” operations because they are the first in a pipeline of operations.
- Early vision operations are characterized by their local nature (they can be carried out in one part of the image without regard for anything more than a few pixels away) and by their lack of knowledge: we can perform these operations without consideration of the objects that might be present in the scene.
- This makes the low-level operations good candidates for implementation in parallel hardware—either in a graphics processor unit (GPU) or an eye.
- We will then look at one mid-level operation: segmenting the image into regions.

24.2.1 Edge detection

Edges are straight lines or curves in the image plane across which there is a "significant" change in image brightness. The goal of edge detection is to abstract away from the messy, multimegabyte image and toward a more compact, abstract representation, as in Figure 24.6. The motivation is that edge contours in the image correspond to important scene contours. In the figure we have three examples of depth discontinuity, labeled 1; two surface-normal discontinuities, labeled 2; a reflectance discontinuity, labeled 3; and an illumination discontinuity (shadow), labeled 4. Edge detection is concerned only with the image, and thus does not distinguish between these different types of scene discontinuities; later processing will.

Figure 24.7(a) shows an image of a scene containing a stapler resting on a desk, and (b) shows the output of an edge-detection algorithm on this image. As you can see, there is a difference between the output and an ideal line drawing. There are gaps where no edge appears, and there are "noise" edges that do not correspond to anything of significance in the scene. Later stages of processing will have to correct for these errors.

How do we detect edges in an image? Consider the profile of image brightness along a one-dimensional cross-section perpendicular to an edge—for example, the one between the left edge of the desk and the wall. It looks something like what is shown in Figure 24.8 (top).

Object Recognition

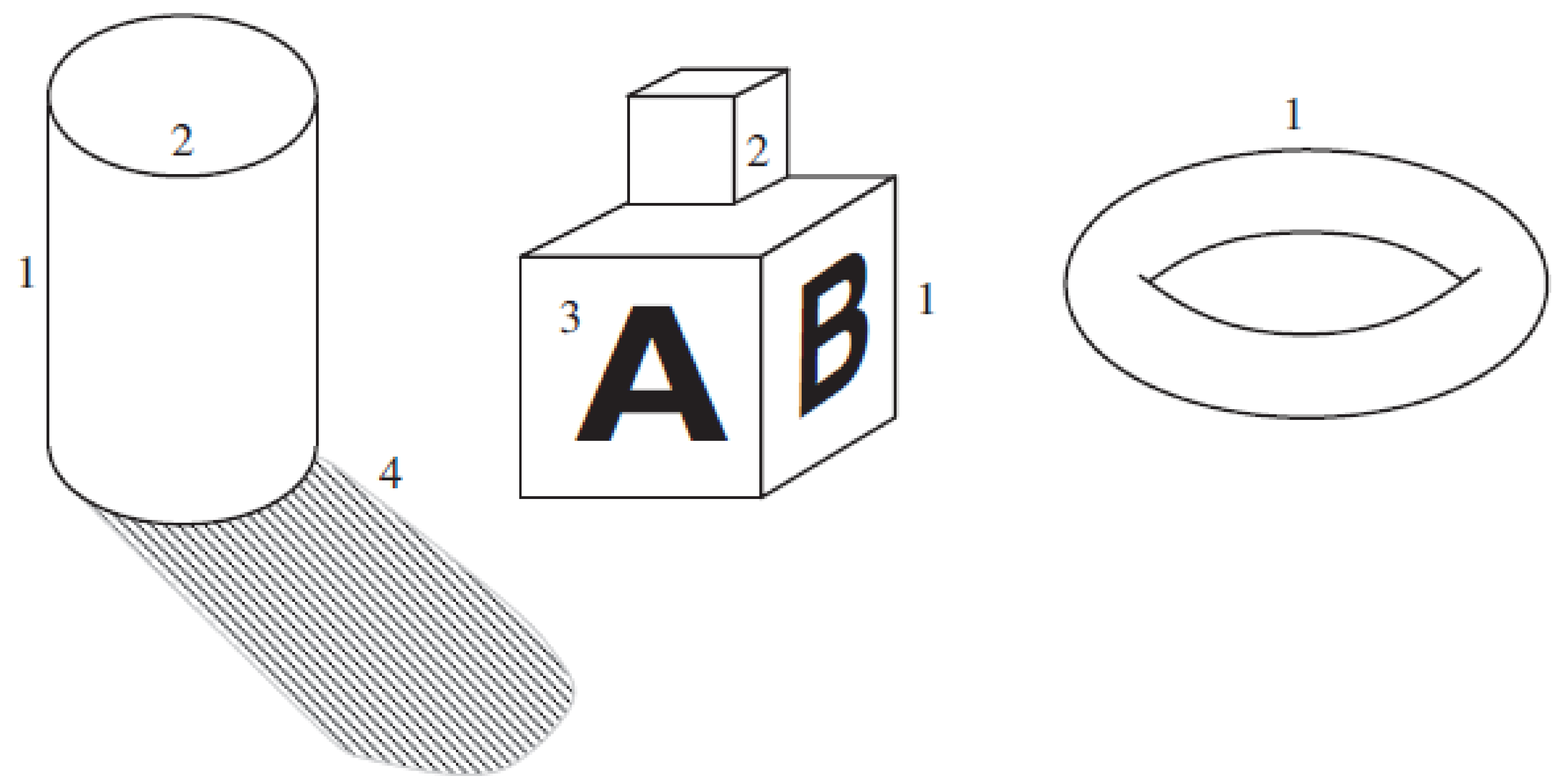
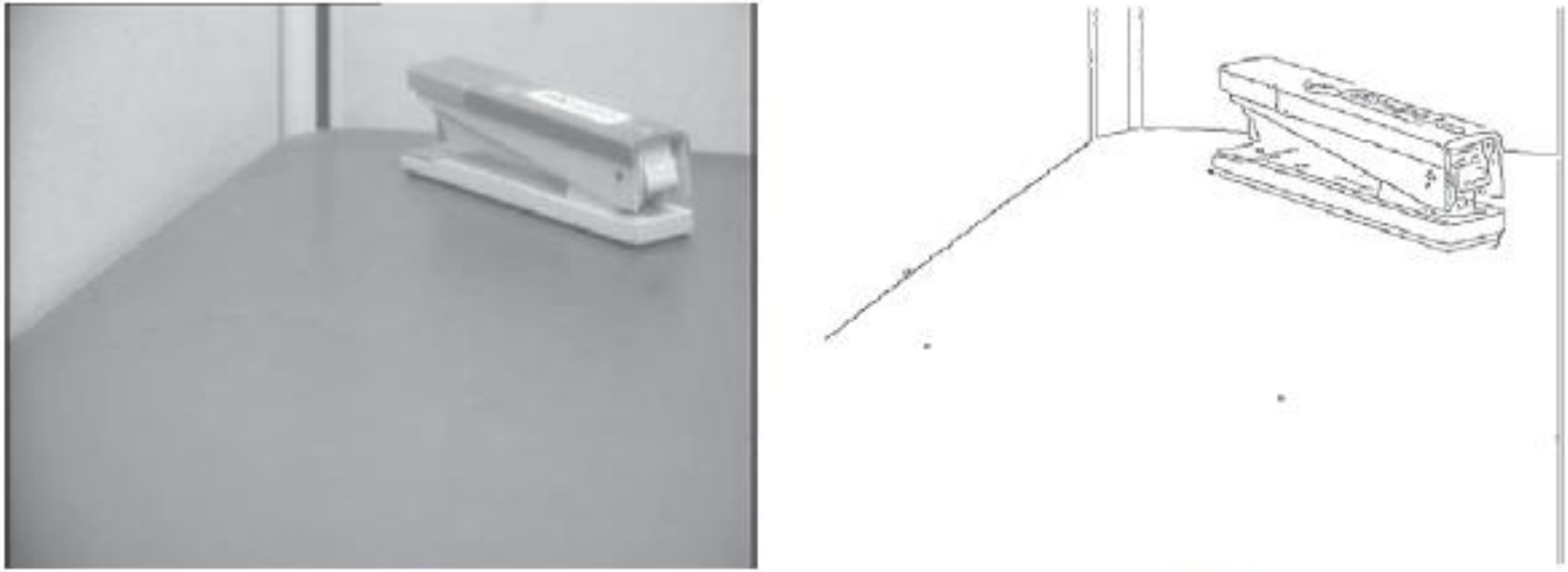


Figure 24.6 Different kinds of edges: (1) depth discontinuities; (2) surface orientation discontinuities; (3) reflectance discontinuities; (4) illumination discontinuities (shadows).

Object Recognition



(a) (b)

Figure 24.7 (a) Photograph of a stapler. (b) Edges computed from (a).

Object Recognition

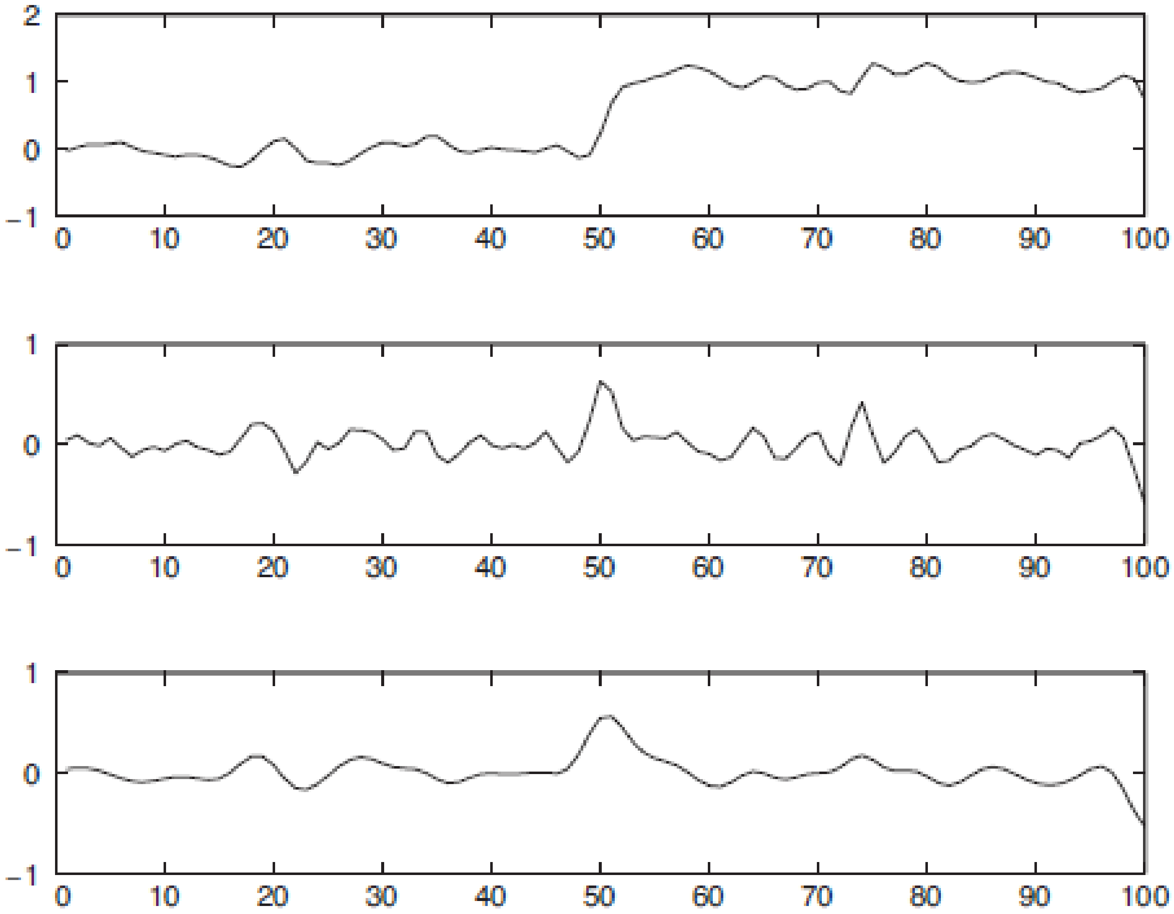


Figure 24.8 Top: Intensity profile $I(x)$ along a one-dimensional section across an edge at $x = 50$. Middle: The derivative of intensity, $I'(x)$. Large values of this function correspond to edges, but the function is noisy. Bottom: The derivative of a smoothed version of the intensity, $(I * G_\sigma)'$, which can be computed in one step as the convolution $I * G'_\sigma$. The noisy candidate edge at $x = 75$ has disappeared.

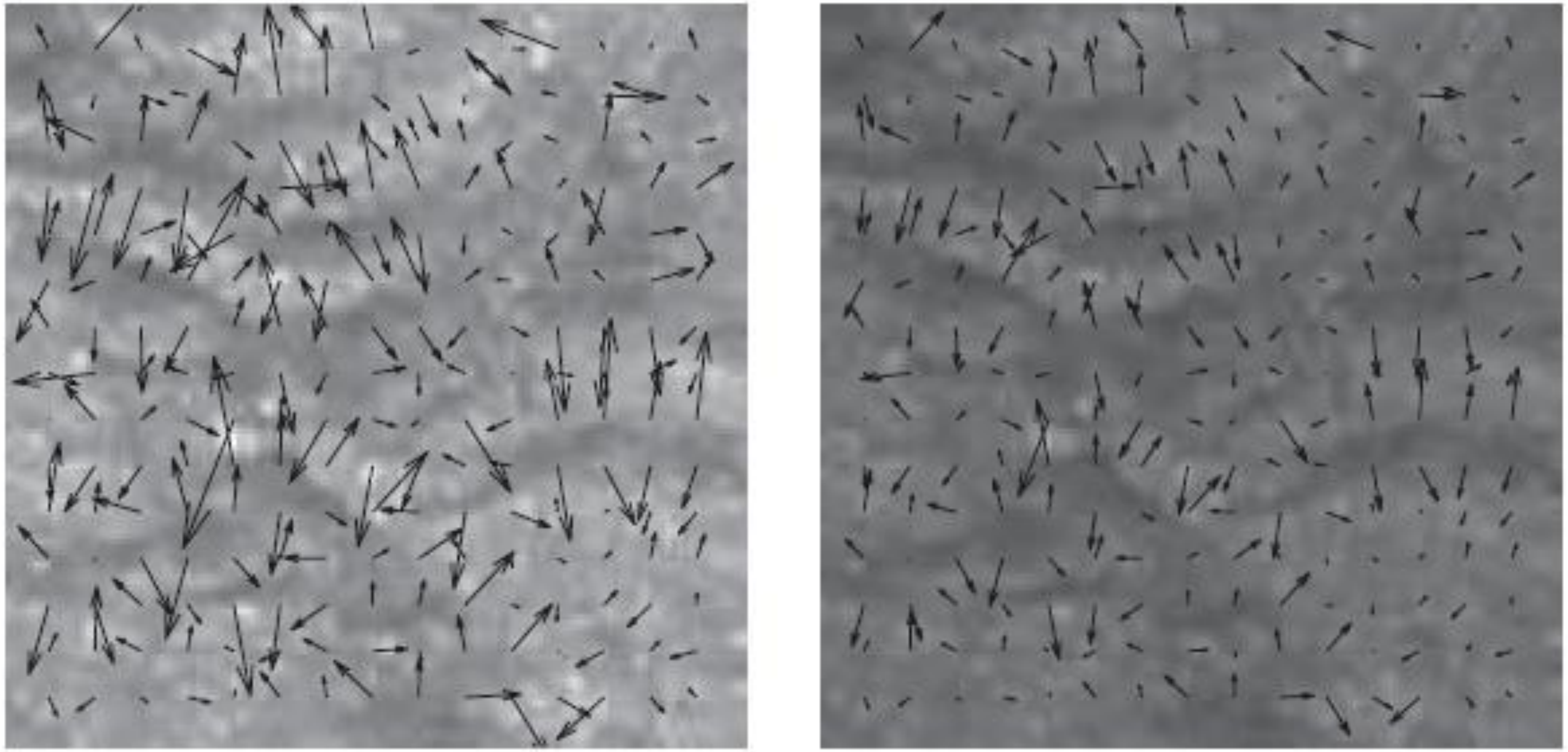
24.2.2 Texture

In everyday language, **texture** is the visual feel of a surface—what you see evokes what the surface might feel like if you touched it ("texture" has the same root as "textile"). In computational vision, texture refers to a spatially repeating pattern on a surface that can be sensed visually. Examples include the pattern of windows on a building, stitches on a sweater, spots on a leopard, blades of grass on a lawn, pebbles on a beach, and people in a stadium. Sometimes the arrangement is quite periodic, as in the stitches on a sweater; in other cases, such as pebbles on a beach, the regularity is only statistical.

Whereas brightness is a property of individual pixels, the concept of texture makes sense only for a multipixel patch. Given such a patch, we could compute the orientation at each pixel, and then characterize the patch by a histogram of orientations. The texture of bricks in a wall would have two peaks in the histogram (one vertical and one horizontal), whereas the texture of spots on a leopard's skin would have a more uniform distribution of orientations.

Figure 24.9 shows that orientations are largely invariant to changes in illumination. This makes texture an important clue for object recognition, because other clues, such as edges, can yield different results in different lighting conditions.

Object Recognition



(a) (b)

Figure 24.9 Two images of the same texture of crumpled rice paper, with different illumination levels. The gradient vector field (at every eighth pixel) is plotted on top of each one. Notice that, as the light gets darker, all the gradient vectors get shorter. The vectors do not rotate, so the gradient orientations do not change.

24.2.3 Optical flow

Next, let us consider what happens when we have a video sequence, instead of just a single static image. When an object in the video is moving, or when the camera is moving relative to an object, the resulting apparent motion in the image is called **optical flow**. Optical flow describes the direction and speed of motion of features *in the image*—the optical flow of a video of a race car would be measured in pixels per second, not miles per hour. The optical flow encodes useful information about scene structure. For example, in a video of scenery taken from a moving train, distant objects have slower apparent motion than close objects; thus, the rate of apparent motion can tell us something about distance. Optical flow also enables us to recognize actions. In Figure 24.10(a) and (b), we show two frames from a video of a tennis player. In (c) we display the optical flow vectors computed from these images, showing that the racket and front leg are moving fastest.

Object Recognition

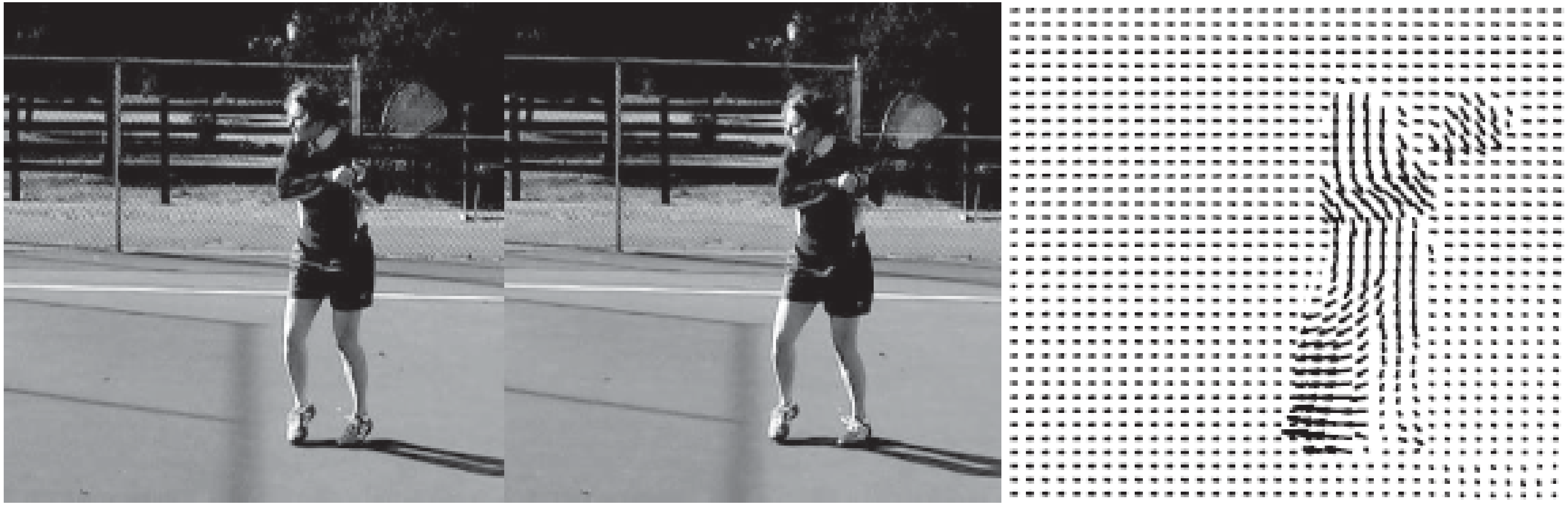


Figure 24.10 Two frames of a video sequence. On the right is the optical flow field corresponding to the displacement from one frame to the other. Note how the movement of the tennis racket and the front leg is captured by the directions of the arrows. (Courtesy of Thomas Brox.)

24.2.4 Segmentation of images

Segmentation is the process of breaking an image into **regions** of similar pixels. Each image pixel can be associated with certain visual properties, such as brightness, color, and texture. Within an object, or a single part of an object, these attributes vary relatively little, whereas across an inter-object boundary there is typically a large change in one or more of these attributes. There are two approaches to segmentation, one focusing on detecting the boundaries of these regions, and the other on detecting the regions themselves (Figure 24.11).

A boundary curve passing through a pixel (x, y) will have an orientation θ, so one way to formalize the problem of detecting boundary curves is as a machine learning classification problem. Based on features from a local neighborhood, we want to compute the probability $P_b(x, y, \theta)$ that indeed there is a boundary curve at that pixel along that orientation. Consider a circular disk centered at (x, y), subdivided into two half disks by a diameter oriented at θ. If there is a boundary at (x, y, θ) the two half disks might be expected to differ significantly in their brightness, color, and texture. Martin, Fowlkes, and Malik (2004) used features based on differences in histograms of brightness, color, and texture values measured in these two half disks, and then trained a classifier. For this they used a data set of natural images where humans had marked the "ground truth" boundaries, and the goal of the classifier was to mark exactly those boundaries marked by humans and no others.

Object Recognition

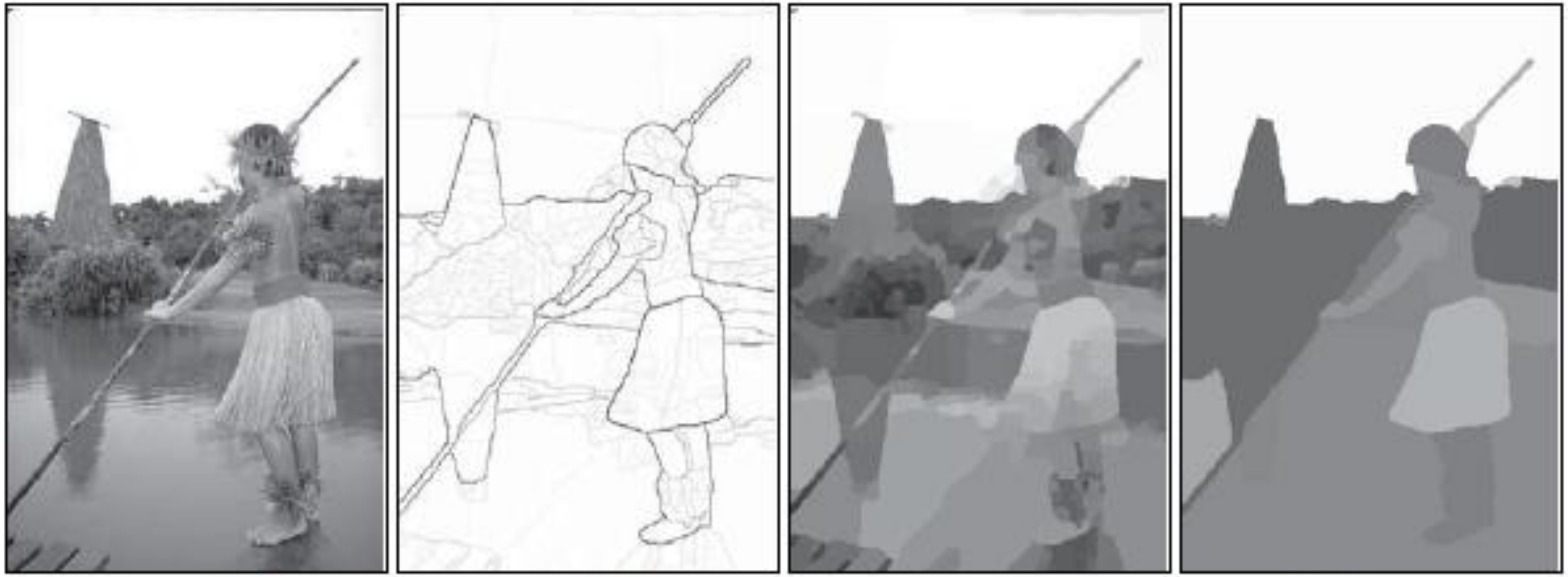

(a) (b) (c) (d)

Figure 24.11 (a) Original image. (b) Boundary contours, where the higher the P_b value, the darker the contour. (c) Segmentation into regions, corresponding to a fine partition of the image. Regions are rendered in their mean colors. (d) Segmentation into regions, corresponding to a coarser partition of the image, resulting in fewer regions. (Courtesy of Pablo Arbelaez, Michael Maire, Charles Fowlkes, and Jitendra Malik)

Note for Students

- ❑**This power point presentation is for lecture, therefore it is suggested that also utilize the text books and lecture notes.**
- ❑**Also Refer the solved and unsolved examples of Text and Reference Books.**