AA 274: Principles of Robotic Autonomy Winter 2018

Lecture 10: Object Localization and Detection

Scribes: Junjie Ke, Ryan Wong, Amine Elhafsi, Samuel Sowell, Peter Zachares

1.1 Introduction

This lecture continued our discussion on machine learning techniques used in robotics and led into the topic of object localization. The objective was to describe applications of machine learning within computer vision in robotics and learn basic concepts for Bayesian filtering.

Although object classification is central to computer vision in robotics, it is not sufficient to just know what is in the image. The robot must be able to localize the detected object in the image as well. It must also know where it is relative to the object it detects. Our aim is to begin give an idea of how all of this comes together.

1.2 Machine Learning in Robotic Applications

1.2.1 Object Localization and Detection

Autonomous agents often need to know not only object type (**object classification**) but also object location within an image (**object localization**). **Object detection** is the generalization of object classification with localization to multiple objects. i.e. classifying and localizing multiple objects of interest within an image.

1.2.1.1 Two-headed network

One approach to object classification with localization is the two-headed network (refer to Figure 1.1 below), where two networks are trained on top of a shared feature map of the image (such as the final convolutional layer of a CNN):

- *Classification head*: which performs object classification, outputting the object class.
- *Regression head*: which performs object localization as a regression, outputting the bounding box coordinates (four numbers) describing the rectangle that encloses an object within the image.

1.2.1.2 Sliding window

The sliding window approach [8] involves running the two-headed network multiple times over sub-regions of an image and aggregating the classification scores and predicted bounding boxes.

Figure 1.2 below illustrates the application of the sliding window on an example image. The top two images show the sliding of the window across the image for the first two sub-regions i.e. the two-headed

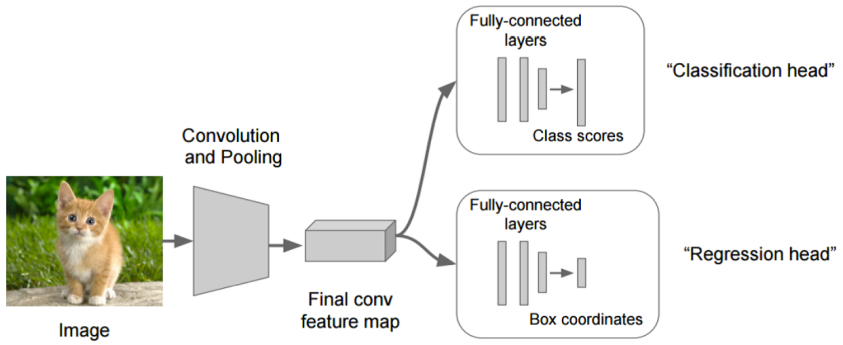


Figure 1.1: Two-headed network for object classification and localization

network applied to the sub-regions of the image (denoted by the black boxes) and the corresponding bounding boxes (red boxes) and classification scores. Bottom left image shows all the bounding box and classification scores from sliding across the entire image. These results are then greedily merged to produce the final result in the bottom right corner.

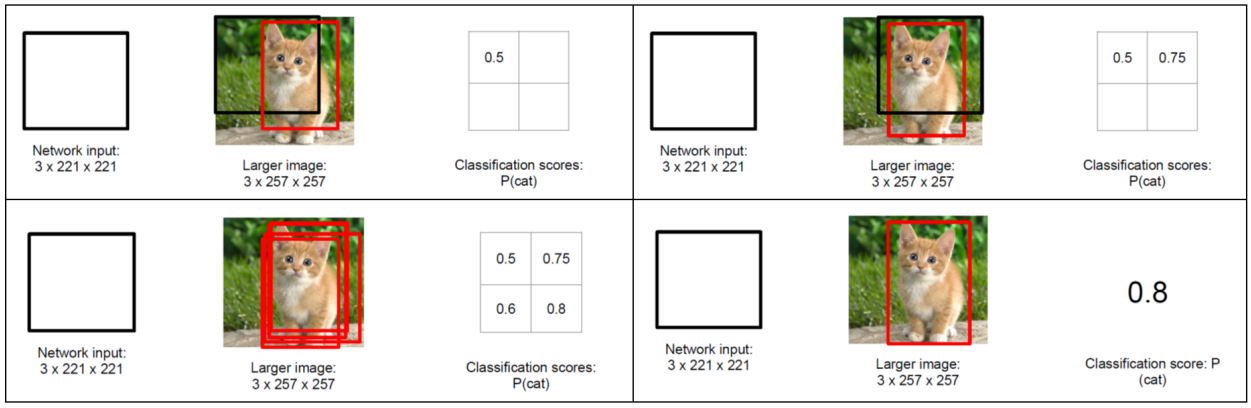


Figure 1.2: Sliding window

Note, it is common practice to use overlapping sliding windows across the image at multiple different scales to ensure objects of varying size will be successfully detected.

A key disadvantage of the sliding window approach is that it is very computationally inefficient as it requires running a model multiple times over a single image.

1.2.1.3 R-CNN

Regional CNN (R-CNN) involves finding blobby image regions that are likely to contain objects and apply a model (such as the two-headed network) on these regions of interest (RoI) (refer to Figure 1.3 below).

The key advantage of this approach is that you only need to run a model over RoI rather than over every sub-region of the image (as in sliding window). Fast R-CNN and Faster R-CNN are improved, faster variants of R-CNN.

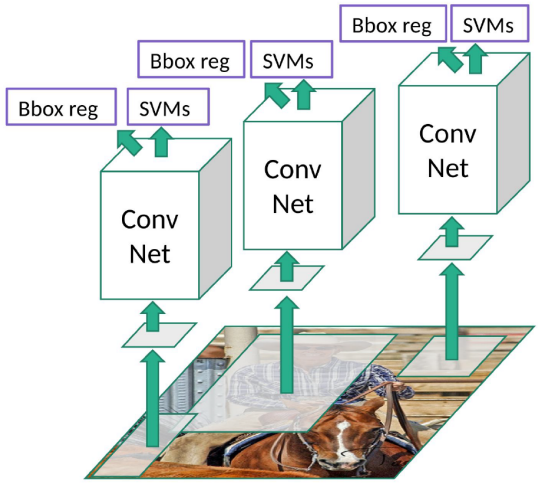


Figure 1.3: R-CNN

1.2.1.4 YOLO

You only look once (YOLO) [7], as the name suggests, involves only applying a single neural network to the full image (as opposed to multiple locations and scales) without RoI proposals (refer to Figure 1.4).

The system divides the input into a S x S grid. Each grid cell predicts B bounding boxes (coordinates and confidence) and class scores. For each bounding box, the confidence score and class scores are combined into a single score that tells us the probability that a bounding box contains a specific object type. From the S x S x B candidate bounding boxes, the final predicted bounding boxes are found based on applying a combined confidence level threshold.

YOLO is computationally more efficient given it is an end-to-end solution that does not require region proposals or multiple applications of a model. In addition, YOLO reasons globally about an image, simultaneously making bounding box predictions for each cell based on the entire image (and not just the pixels within a single grid cell).

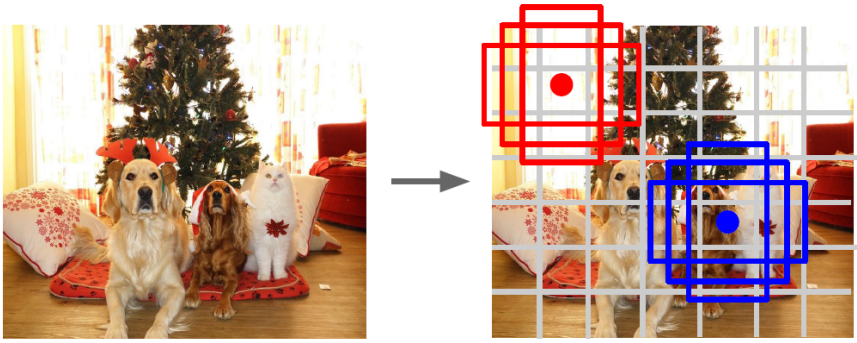


Figure 1.4: YOLO

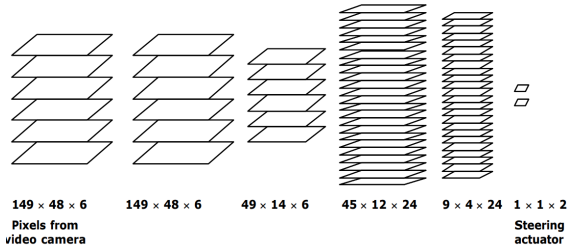
1.2.1.5 Condensed networks for speed

Many robotics applications, such as mobile robotics, require object detection to be performed in resource constrained settings. Hence, there has been work to compress some of these state-of-the-art networks, trading off accuracy for lower latency and size. Two popular condensed networks include Mobilenets [3] (object classification) and Tiny YOLO (object detection).

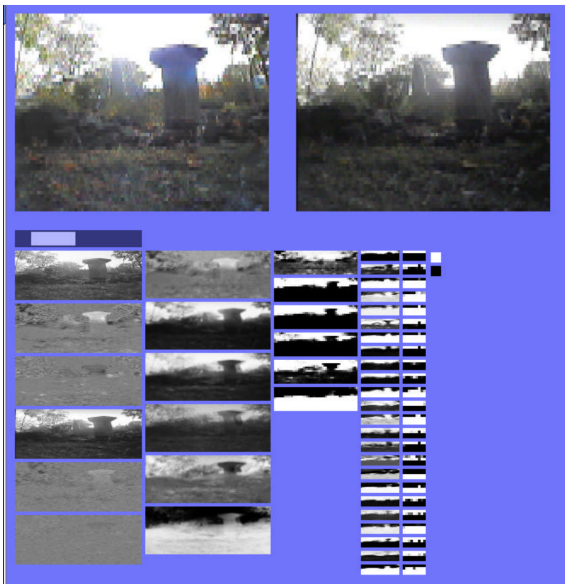
1.2.2 End-to-end: From Pixels to Motor Commands

One of most exciting applications of deep neural network in robotics is training an "end-to-end" controlling system. This powerful end-to-end approach means that with minimum training data from humans, the system can learn to perform tasks under complicated circumstances.

For example, we can use convolutional neural networks (CNNs) to map the raw pixels from a front-facing camera to the steering commands for a self-driving car. With the power of CNNs, features are now learned automatically from training examples.



(a) Network architecture for DAVE.

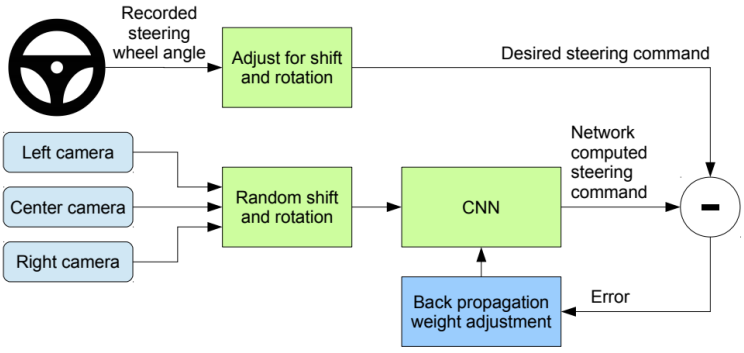


(b) A snap shot of the trained network for DAVE. The camera input (YUV maps of the two cameras) is on the left and steering output is on the right.

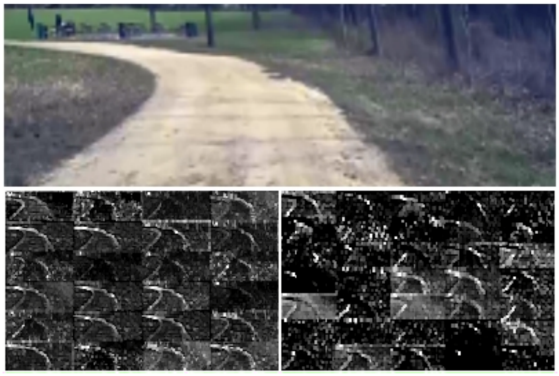
Figure 1.5: Network architecture and training snapshot for DAVE[5] autonomous driving system.

One of the earliest attempt to build an end-to-end self-driving system was actually done over 10 years ago in a Defense Advanced Research Projects Agency (DARPA) seedling project known as DARPA Autonomous Vehicle (DAVE) [5], in which a sub-scale radio control (RC) car drove through a junk-filled alley way. DAVE was trained on hours of human driving in similar, but not identical, environments. The training data included video from two cameras and the steering commands sent by a human operator, as illustrated in Figure 1.6.

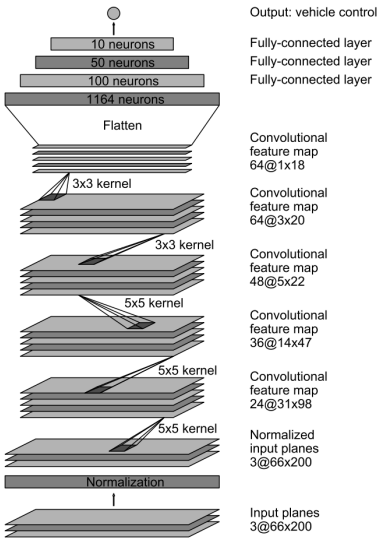
In 2016, NVIDIA started a new effort to improve on the original DAVE, and create a robust system (DAVE-2 [1]) for driving on public roads. Figure 1.6a shows the diagram of its training system. Images are fed



(a) End-to-end training procedure for DAVE-2.



(b) How the DAVE-2 CNN sees an unpaved road.



(c) CNN architecture for DAVE-2. The network has about 27 million connections and 250 thousand parameters

Figure 1.6: Network and system architecture for DAVE-2[1] autonomous driving system.

into a CNN that then computes a proposed steering command. The proposed command is compared to the desired command for that image and the error is back propagated to adjust the weights. Once trained, the network is able to generate steering commands from the video images of a single center camera.

1.2.3 More Advanced Topics in Machine Learning

Training and Evaluation Techniques

The common practice of training a model is splitting the dataset into 3 parts: training, validation and testing. Models are trained and optimized on training set but evaluated on the held-out validation set to give an unbiased estimate of model skill. Hyperparameters are chosen based on the score on validation set. The model's final performance is determined by the test set.

Learning rate and hyperparameter tuning are usually a tedious task. There are different variants of gradient descent that automatically change the learning rate to improve the performance of training. Most commonly used adaptive learning rate methods includes RMSprop [10] and Adam [4].

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are popular models that have shown great promise in processing sequential or time-series data, such as video streams and speech. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations, as illustrated in Figure 1.7.

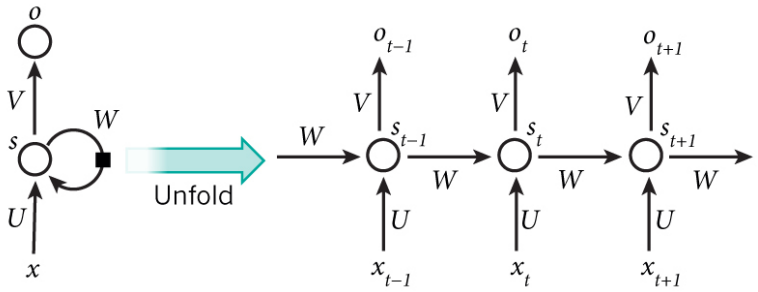


Figure 1.7: A recurrent neural network and the unfolding in time of the computation involved in its forward computation.

Deep Reinforcement Learning

Reinforcement learning (RL) is about an agent interacting with the environment, learning an optimal policy, by trial and error, for sequential decision making problems. With recent exciting achievements in deep learning, the combination of deep neural networks and reinforcement learning has achieved great success, like deep Q-network [6] and AlphaGo [9].

1.2.4 Useful Packages for Building Deep Learning Models

Building deep learning model is naturally a big challenge for data scientists and engineers. Luckily, theres a growing range of frameworks that make it much easier.

Tensorflow

Tensorflow[1], developed by the Google Brain team, is one of the most popular Deep Learning libraries. This is the go-to framework for learning deep learning. It's Python-based, has a very good documentation, and there are tons of tutorials and videos available on the Internet.

Keras

Creating models using pure Tensorflow can be quite complex for a beginners. Keras[2] was built as a simplified interface for building efficient neural networks in just a few lines of code and it can be configured to work on top of TensorFlow. Written in Python, Keras is very lightweight, easy to use, and straightforward to learn.

Torch

Torch[3] is a Lua-based deep learning framework and has been used and developed by Facebook. Torch was built with an aim to achieve maximum flexibility and make the process of building your models extremely simple. More recently, the Python implementation of Torch, called PyTorch, has found popularity and is gaining rapid adoption.

1.3 Localization

1.3.1 Behavioral Approach

Behavioral approach attempts to design a set of behaviors that result in desired robot motion and trajectory. This method avoids mapping the robot's environment for localization. For example, consider Fig. 1.8, where a robot wants to navigate from room A to room B. Using behavioral approach, a robot may find its way to room B by following the left wall, incorporating other procedures such as collision avoidance and identification of room B. However, compared to map-based approach, this method of navigation has difficulty scaling to different problems or environments. Depending on situations, techniques like following the left wall can also be suboptimal.

1.3.2 Map-based Approach

Map-based localization uses information about the robot's surrounding to identify its position *with respect to* the environment. To gain knowledge of its environment, the robot must actively gather and process

[1] https://www.tensorflow.org/
[2] https://keras.io/
[3] http://torch.ch/

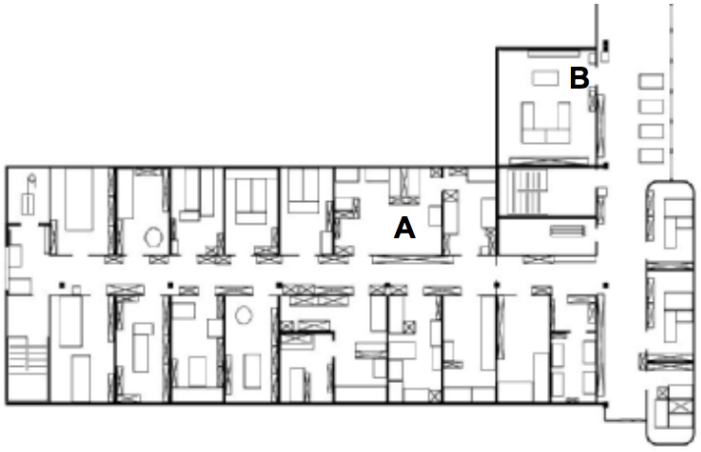


Figure 1.8: Siegwart Figure 5.6. An environment with rooms A and B

sensor data of the surrounding. The most popular sensor is LIDAR, which can provide 360° coverage of the robot's surrounding. This approach can scale to any kind of environment, because a robot can use same techniques and sensors to map its surrounding.

1.3.3 Belief representation

Due to sensor noise, a robot cannot localize itself with absolute certainty. To compensate for the error, we can represent the estimate of robot's location as a probability distribution across the environment. There are different ways of representation. For example, the belief of robot's position can be *continuous*, which can be expressed by continuous variables (x, y, θ) Fig. 1.9(b).

On the other hand, the belief can be *discrete*. For instance, the map of the environment may be discretized into $N \times N$ cells as in Fig. 1.9(c), and the belief can assign probabilities of being in different cells. The other possible discrete representation is a topological map in Fig. 1.9(d), which maps the environment using nodes and edges. We can then assign probabilities of robot being in certain nodes.

The belief representation can also be classified as either *single-hypothesis belief* or *multiple-hypothesis belief*. *Single-hypothesis belief*, as in Fig. 1.10(a), is useful when describing a unique estimate of a robot's position. On the other hand, *multiple-hypothesis belief* in Fig. 1.10(b, c, d) can be used to describe multiple possible locations of a robot.

Continuous belief representation typically incorporates single-hypothesis belief (ex. state vector $[x, y, \theta]$) that is compact and easy to compute. However, discrete belief representation typically incorporates multiple-hypothesis belief. Because discrete method assigns the probabilities of robot's presence across different cells or nodes in the map, we see that the accuracy of estimate is limited by the resolution of discretization. It may also involve heavy computation, because a robot must keep track of probabilities associated each cell [2].

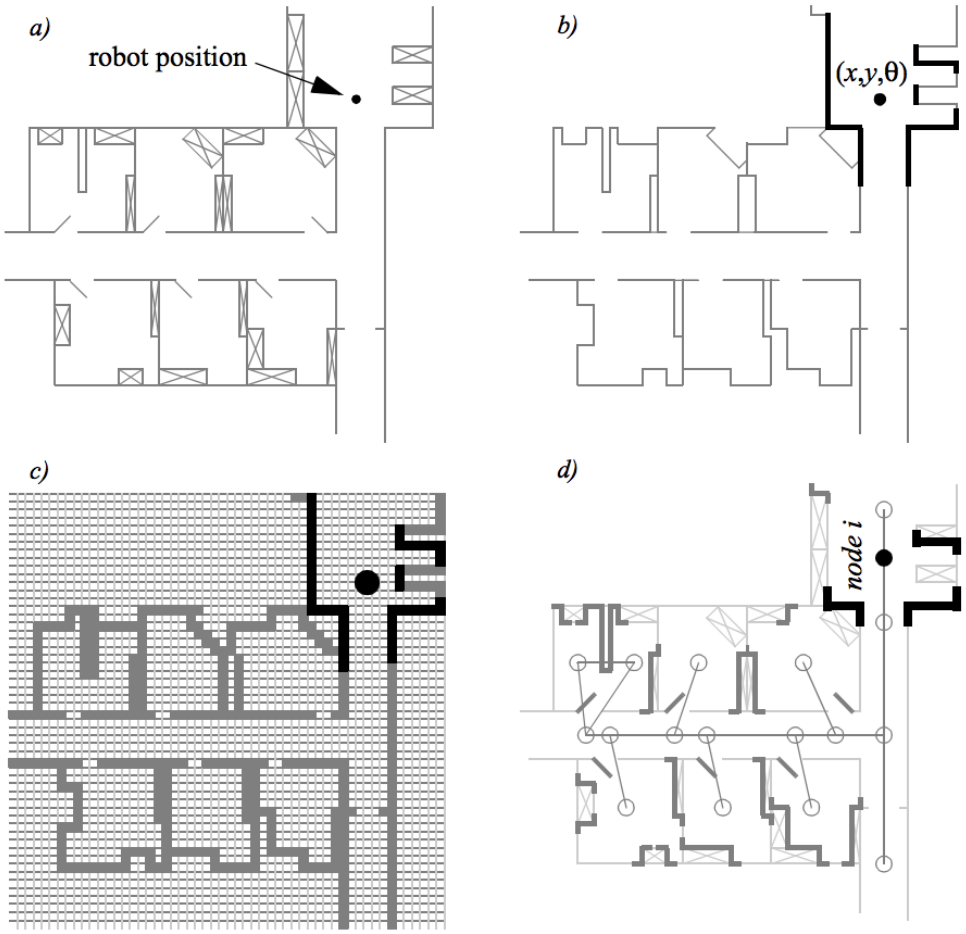


Figure 1.9: Siegwart Figure 5.10. (a) Real map. (b) Map where continuous lines represent walls. (c) Same map discretized into 3000 grid cells. (d) Topological map with nodes and edges.

1.4 Probability Basics

1.4.1 Random Variable

Quantities such as sensor measurements, states of a robot and its environment are modeled as random variables. A random variable is a variable whose possible values are outcomes of a random phenomenon. In the case of sensor measurements, the source of randomness could be sensor noise. Random variables fall under one of two categories: discrete or continuous.

1.4.1.1 Discrete Random Variable

Discrete random variables can take on only a countable number of values. For example, if we flip a coin twice and the random variable X represents the number of coin flips resulting in "heads", X can only take on values of 0, 1 or 2 (and similarly for the case of "tails"). A discrete random variable is characterized by

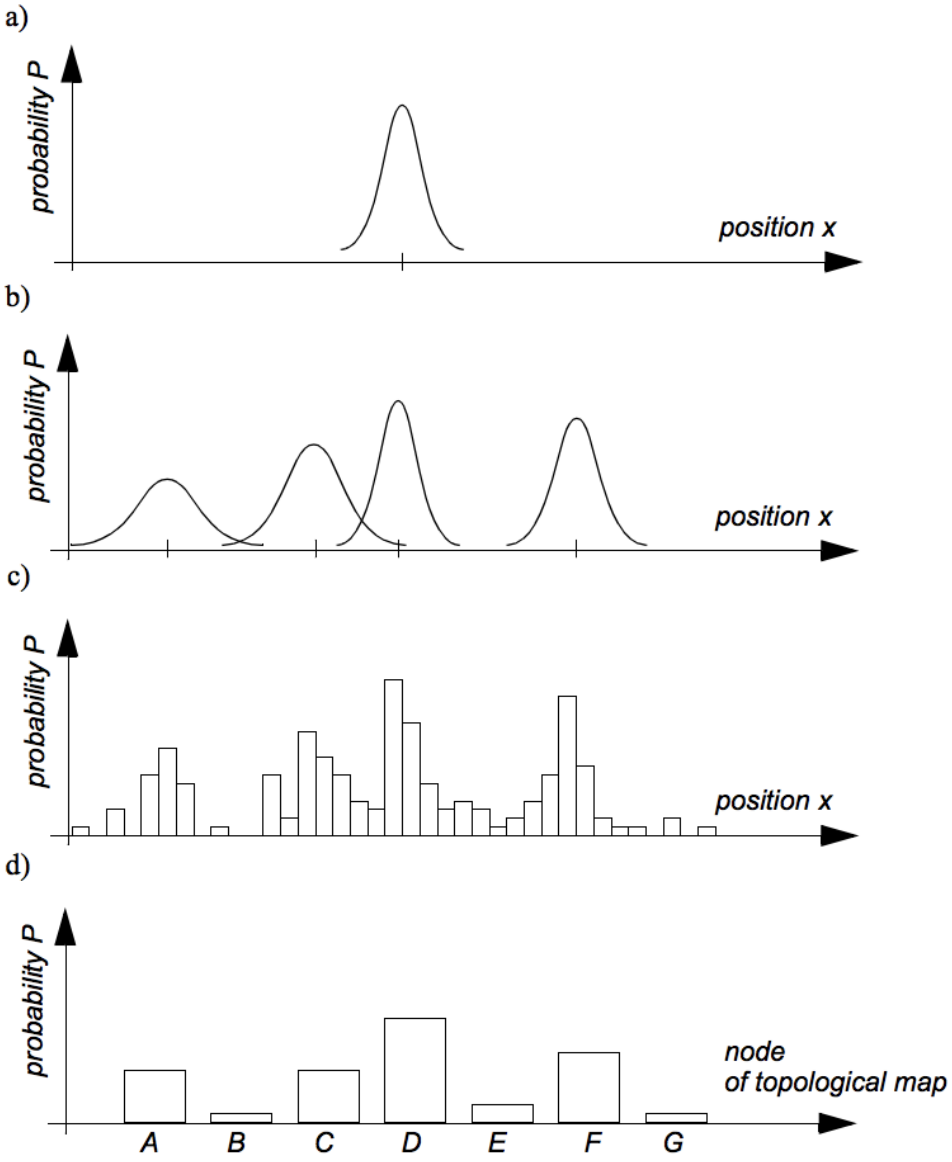


Figure 1.10: Siegwart Figure 5.9. (a) Continuous single-hypothesis belief. (b) Continuous multiple-hypothesis belief. (c) Discrete probabilities for grid-based map. (d) Discrete probabilities for topological map.

a probability mass function (PMF) $p(X = x)$ (or $p(x)$) where

$$\sum_x p(x) = 1.$$

1.4.1.2 Continuous Random Variable

In contrast with a discrete random variable, a continuous random variable can take on an infinite number of values. A continuous random variable is characterized by a probability density function (PDF), $p(x)$, where the probability of a random variable being contained within the interval $[a, b]$ is

$$P(a \leq X \leq b) = \int_a^b p(x)dx$$

where

$$\int_{-\infty}^{+\infty} p(x)dx = 1.$$

1.4.2 Joint Distributions, Independence and Conditional Probability

1.4.2.1 Joint Distributions

The joint distribution of two random variables X and Y is denoted as

$$p(x, y) := p(X = x \text{ and } Y = y).$$

1.4.2.2 Independence

If X and Y are random variables such that,

$$p(x, y) = p(x)p(y)$$

then X and Y are said to be independent.

1.4.2.3 Conditional Probability

Conditional probability is the probability of an event happening given that another event has occurred. That is, suppose that we know that the event $Y = y$ occurs with probability $p(y) > 0$. The probability of the event $X = x$ occurring given that $Y = y$ has occurred, or alternatively stated as the conditional probability of X given Y, is given by

$$p(x|y) := \frac{p(x,y)}{p(y)}.$$

Thus if X and Y are independent, $p(x|y) = p(x)$. Also, note that the above definition of conditional probability is indeed a definition and not theoretically derived.

In order to better understand the concept of conditional probability, consider Fig. 1.11İn this figure, the sample space Ω contain the set of all possible outcomes for a random variable. Consider the case where there are two possible outcomes, $B1$ and $B2$ which may occur with an equal probability of 0.5. These events correspond to the orange and blue regions of Ω, respectively. An event A that occurs with probability 0.5 can also be defined, which corresponds to the centered shaded region in Fig. 1.11.

Suppose that we wish to determine the probability of A occurring and that a sample from Ω is observed to result in the event $B1$. Given that $B1$ has occurred, we restrict our attention to the region $B1$. In this restricted region, the area corresponding to A occurring is the intersection of A and $B1$. Since region A overlaps with half of the region $B1$, we have the result that $p(A|B1) = 0.5$. A slightly different but equivalent interpretation of this figure, as given during lecture, is that we rescale the probability of event A occurring by the probability of the event that has happened. Mathematically, this is expressed as

$$p(A|B1) = \frac{p(A \cap B1)}{p(B1)} = \frac{0.25}{0.5} = 0.5.$$

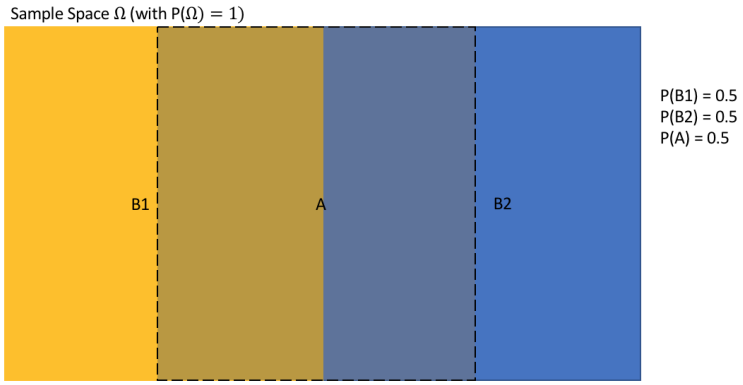


Figure 1.11: Sample space representation. Note that region A is centered an overlaps $B1$ and $B2$ equally.

Finally, note that if X and Y are independent $p(x|y) := p(x)$.

1.4.2.4 Conditional Independence

If X, Y and Z are random variables such that,

$$p(x, y \mid z) = p(x \mid z)p(y \mid z)$$

then X and Y are said to be conditionally independent given Z.
However:

$$p(x, y \mid z) = p(x \mid z)p(y \mid z) \not\Rightarrow p(x, y) = p(x)p(y)$$

Nor:

$$p(x, y) = p(x)p(y) \not\Rightarrow p(x, y \mid z) = p(x \mid z)p(y \mid z)$$

1.4.3 Theorem of Total Probability

Once again consider Fig. 1.11. If we wished to find the probability of A without any assumptions regarding conditioning, we can do so by calculating

$$p(A) = p(A \cap B1) + p(A \cap B2).$$

This idea can be generalized for discrete random variables into what is known as the theorem (or law) of total probability as

$$p(x) = \sum_y p(x, y) = \sum_y p(x|y)p(y)$$

where the definition of conditional probability was used to obtain the last expression. For continuous random variables, the summations simply turn into integrals:

$$p(x) = \int p(x, y)dy = \int p(x|y)p(y)dy$$

1.4.4 Bayes' Rule

To facilitate the explanation of Bayes' Rule please note: a sample space is a set containing all the possible outcomes of an experiment and an event is a subset of elements in a sample space. Imagine that there is an event A whose set of outcomes is contained in the sets of events B_i where $i = 0, 1, ..., n$ of a sample space. For example in Fig. 1.11 event A's set is contained in the sets of events B_1 and B_2. Then,

$$p(B_i|A) = \frac{p(B_i \cap A)}{p(A)}.$$

Using the formula for conditional probability $p(B_i \cap A)$ and $p(A)$ can be expressed as:

$$p(B_i \cap A) = p(A|B_i)p(B_i).$$
$$p(A) = \sum_{i=0}^{n} p(B_i \cap A) = \sum_{i=0}^{n} p(A|B_i)p(B_i).$$

Combining these two expressions to derive Bayes' Rule for discrete sample spaces:

$$p(B_i|A) = \frac{p(A|B_i)p(B_i)}{\sum_{i=0}^{n} p(A|B_i)p(B_i)}.$$

For a continuous sample space Bayes' Rule is:

$$p(B|A) = \frac{p(A|B)p(B)}{\int p(A|B=b')p(B=b')db'}.$$

Note that $p(A)$ is independent of which event Bi is selected. So the denominator of Bayes' formula can be considered a constant for all events Bi with event A. Furthermore, by knowing the probabilities $p(A|Bi)$ and $p(Bi)$, we can determine $p(Bi|A)$. As an example, a brewery buys equal amounts of hops from two farmers, $B1$ and $B2$. The brewery has a strict policy of using only one type of hops in each batch of beer it brews. Furthermore, the brewery uses half of all its hops from both sources to produce Pilsner beer. Bayes' Rule can be applied to determine, given that a batch of Pilsner is produced, what is the probability that the batch was brewed using hops from farmer $B1$. Note this example uses the sample space described in Fig. 1.11

$$p(B_1|A) = \frac{p(A|B_1)p(B_1)}{\sum_{i=0}^{1} p(A|B_i)p(B_i)} = \frac{p(A|B_1)p(B_1)}{p(A|B_1)p(B_1)+p(A|B_2)p(B_2)} = \frac{0.5*0.5}{(0.5*0.5+0.5*0.5)} = \frac{0.25}{0.5} = 0.5.$$

Bayes' Rule can be extended with the use of additional random variables. In the three variable case, given $X = x$, $Y = y$ Bayes' Rule can be conditioned on an additional variable $Z = z$ as follows:

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)}.$$

This is the probability that event $Z = z$, and event $X = x|Y = y$ occur, which describes $Z = z$ influence on joint event $X = x|Y = y$.

1.4.5 Expectation and Covariance

The expectation value of a random variable $E(X)$ is the average result of an experiment over an infinite number of trials. It is also known as the first moment of the distribution. For the discrete case:

$$E(X) = \sum_{i=0}^{n} x_i p(X = x_i).$$

For the continuous case:

$$E(X) = \int_{-\infty}^{\infty} x' p(X = x')dx'.$$

The expectation value of a constant is a constant and its calculation is a linear operation.

$$E(aX + b) = aE(X) + b.$$

In the case of a vector of random variables. The expectation of the vector is the vector of each variables expectation value.

Covariance is a calculation used to measure the relationship between random variables. For two random variable vectors $X = x, Y = y$, the covariance is:

$$cov(X, Y) = E[(X - E(X))(Y - E(Y))] = E(XY^T) - E(X)E(Y^T)$$

If the covariance is positive this implies that X tends to increase at the same time as Y. If the covariance is negative then this implies that X tends to decrease as Y increases.If changes in X and Y have a random relationship, then the covariance will be close to zero. If two variables have a covariance of zero they are called uncorrelated. If two variables are independent they will be uncorrelated. However, two variables which are uncorrelated are not always independent.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] C. Clark. Lecture 9—Markov Localization. E160:Autonomous Robot Navigation, Harvey Mudd College, 2016.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Y. LeCun, E. Cosatto, J. Ben, U. Muller, and B. Flepp. Dave: Autonomous off-road vehicle control using end-to-end learning. Technical report, Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, http://www. cs. nyu. edu/yann/research/dave/index. html, 2004.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[8] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[9] D. Silver and D. Hassabis. Alphago: Mastering the ancient game of go with machine learning. *Research Blog*, 2016.

[10] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.