CSCI 4155/6505 (2018)
Machine Learning

Thomas P. Trappenberg
Dalhousie University

Acknowledgements

These lecture notes have been inspired by several great sources. I would specifically like to acknowledge the influence of Andrew Ng's lecture notes on earlier versions of this manuscript, as well as the teaching material from the NVIDEA Deep Learning Institute.

There are now a variety of excellent books on the theory of machine learning that I would like to recommend for further readings. This includes the by *Introduction to Machine Learning* by Ethem Alpaydin, 2nd edition, MIT Press 2010, and *Pattern Recognition and Machine Learning* by Christopher Bishop, Springer 2006. The standard book on RL is *Reinforcement Learning: An Introduction* by Richard Sutton and Andrew Barto, MIT press, 1998. The standard book for AI, *Artificial Intelligence: A Modern Approach* by Stuart Russell and Peter Norvig, 2nd edition, Prentice Hall, 2003, does also include some chapters on Machine Learning.

The most in-depth book on probabilistic machine learning is likely the book by Kevin Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012. The great book on Deep Learning is the book by Ian Goodfellow, Yoshua Bengio, and Aaron Courville, MIT Press, 2016. These are two books that are invaluable for researchers in machine learning.

Several people have contributed considerably to this lecture notes. In particular I would thank Paul Hollensen, Patrick Connor, and Hossein Parvar. And finally I would like to thank my students who took my classes over the last several years and have challenged me to think deeper about the machine learning.

Preface

Machine learning has recently made a big splash, both in research and in industrial applications. Although many of the machine leaning ideas have been around for many years, the latest breakthrough is based on several advancements. One of them is the availability of large datasets with labeled data, another is the availability of fast specialized processors such as GPUs. I would like to add that the progress is also fueled by a deeper understanding and some new techniques that brought it all together.

There are now several great books on machine learning which include probabilistic approaches, deep learning and Bayesian modeling. So why another one? The main reason is that I believe this book can fill a gap based on the style of books I like. More specifically, I tried to emphasize what seems to be sometimes two opposite ends of machine learning, Bayesian modeling and deep learning. Second, I like to keep things short and on to the point; this is in particular important to keep in mind when using the book. I tried to be brief while still going into some depth. How can this work? The way I tried to approach this is to discuss minimal models and examples that show the essence of the ideas behind machine learning approaches. At the same time I always try to give an outlook to further concepts and tricks. I assume that readers are mature enough to consult other resources for further studies. I do not claim to cover all details of machine learning, but my hope is to provide the fundamentals for a good understanding and further studies.

Another emphasis of this book is to provide a diverse view of machine learning in combining deep learning and probabilistic modeling. These areas are sometimes viewed as opposite poles. However, I would like to argue that both approaches are important, that both approaches have specific strength in specific application areas, and that both approaches can even be combined. Furthermore, we will also mention some older machine learning techniques such as support vector machines and decision trees. While these approaches are considered shallow with respect to deep learning, they have still important practical applications which should not be forgotten. While we will not dive deep into the theory of these traditional methods, studying this book should also give some theoretical background to be a ble to follow more specific literature on these models. s

Contents

1 Introduction 1

1.1 The basic idea behind supervised Machine Learning 1

1.2 Mathematical formulation of the supervised learning problem 5

1.3 Applied Learning: Training, validating and testing 8

1.4 Non-linear regression and high-dimensionality 12

1.5 Deep learning 15

1.6 No free lunch and choice of models 18

2 Scientific programming with Python 20

2.1 Programing environment 20

2.2 Basic constructs 20

2.3 Cross validation example from Intro 25

2.4 Code efficiency and vectorization 27

2.5 Data handling 28

2.6 Programing style 30

3 Machine learning with sklearn 31

3.1 Classification of Iris data with SVM and RF 31

3.2 Performance measures and evaluations 33

3.3 Overlapping Gaussians 35

3.4 Support Vector Machines (SVM) * 37

3.5 Decision Trees and Random Forrest * 48

3.6 Feature Selection, Dimensionality reduction, t-SNE 49

4 Regression and optimization 51

4.1 Linear regression and gradient descent 51

4.2 Error surface and challenges for gradient descent 53

4.3 Advanced gradient optimization (learning) 56

4.4 Regularization: Ridge regression and LASSO 59

4.5 Nonlinear regression 63

5 Basic probability theory 66

5.1 Random numbers and their probability (density) function 67

5.2 Moments: mean, variance, etc. 68

5.3 Examples of probability (density) functions 71
5.4 Some advanced concepts 73
5.5 Density functions of multiple random variables 77

6 Probabilistic regression and classification 82
6.1 Probabilistic models 82
6.2 Learning in probabilistic models: Maximum likelihood estimate 85
6.3 Probabilistic classification 87
6.4 MAP and Regularization with priors 90

7 Generative Models 93
7.1 Modelling classes 93
7.2 Supervised Gaussian model: Discriminant analysis 94
7.3 Unsupervised example: K-means clustering 96
7.4 Mixture of Gaussian and the EM algorithm 98

8 Bayes Nets: Multivariate causal modelling 102
8.1 Causal models 102
8.2 Discrete probabilistic modeling in Python using LEA 105
8.3 Naive Bayes 106

9 MLP 108
9.1 Neurons and the threshold perceptron 108
9.2 Sigmoidal perceptron and the delta learning rule 110
9.3 Multilayer perceptron (MLP) 114
9.4 A letter recognition example 119
9.5 Probabilistic and Stochastic Neural Networks 119

10 DeepLearning 123
10.1 Filters and 1d convolution 123
10.2 2-d convolution and image processing 125
10.3 Convolutions with tensors 127
10.4 CNN 129
10.5 Tensorflow 131
10.6 Representational learning and compression 131
10.7 Denoising autoencoders 133

11 Recurrent Neural Networks 135
11.1 Sequence processing 135
11.2 Gated RNN 138
11.3 RNN with external memory 140
11.4 The Boltzmann machine 140

12 Reinforcement Learning 148

12.1 Learning from reward and the credit assignment problem 148

12.2 Model-based Reinforcement Learning 152

12.3 Model-free Reinforcement Learning 157

12.4 Deep Reinforcement Learning 162

1 Introduction

1.1 The basic idea behind supervised Machine Learning

Machine Learning is literally about building machines that can learn and after learning perform specific tasks. We will encounter several forms of learning, but for the most part we consider **supervised learning**. In supervised learning we show a variety of examples together with the desired response of a specific task to a learning machine from which the machine should learn to perform an appropriate response for new examples. In **unsupervised learning** we show the machine examples without examples of an appropriate response. The reason for this type of learning is usually to find some regularities in the data and to learn to represent data. Unsupervised learning is often combined with some form of supervised learning. A slightly more general form of supervised learning is **reinforcement learning** where a teacher gives some feedback on the goodness of the answers provided by the learner but the learner has to discover appropriate actions itself. While we will encounter all these different type of learning in this book, understanding supervised learning is the basic on which all the other learning methods build. We will hence start with this type of machine learning.

In machine learning we are trying to solve problems with computers without explicitly programming them for a specific tasks. We will still need to program the learning machine, but this is somewhat more general than coding logic for a specific problem. Programming general learning machines instead of specific solutions to a problem is desirable specifically for tasks that would be difficult to program. A classic example that we will discuss in some length is character recognition; writing a program that can translate a visual representation of a character, say the letter A, to the meaning such as representing this letter as the ASCII string 1010101010 is not easy when considering all the shapes and styles that this character can take. Some related tasks form computer vision are shown in Figure 1.1.

Machine Learning might sound like a niche area in science and you might wonder why there is now so much interest in this discipline, both academically and in industry. The reason is that Machine Learning is really about modelling data. Modelling is the basis for advanced object recognition, data mining, and ultimately intelligent systems. Machine Learning is the analytic engine in areas such as data science, big data, data analytics and to some extend to science in general in the sense of building quantitative models.

Machine Learning has a long history with traces far back in time. Alan Turing was probably one of the earliest thinkers in the field of AI. One of the first recognized exciting realizations of the promise of learning machines came in the late 1950s and early 1960s with work like Arthur Samuel's self-learning checkers program and Frank Rosenblatt's perceptron. Arthur Samuel devised a program with some form of reinforcement learning that ultimately learned to outperform its creator. Frank Rosenblatt

Since 1699, when French explorers landed at the great bend of the Mississippi River and celebrated the first Mardi Gras in North America, New Orleans has brewed a fascinating melange of cultures. It was French, then Spanish, then French again, then sold to the United States. Through all these years, and even into the 1900s, others arrived from everywhere: Acadians (Cajuns), Africans, indige-

Fig. 1.1 Typical examples where deep learning has been instrumental in practical applications such as letter and object recognition, and image restorations. Figure from NVIDIA DLI teaching kit.

set much of the foundation of neural networks and even started to build neural network computers, the Mark I Perceptron. Neural networks have been popularized again in the 1980s with a strong influence by David Rumelhart. Many leading figures in Deep learning have started in this era, including Geoffrey Hinton, Yoshua Bengio, Yann LeCun, Jürgen Schmidthuber to name a few. We will discuss how we are now in an era of ‘Deep Learning’ with important recent developments that are mostly the reason for the popularity of Machine Learning today. However, much of the progress of machine learning and their scientific embedding is due to probabilistic methods and statistical learning theories, for which we need to mention some pioneers like Vladislaw Vapnik and Judea Pearl. Indeed, the development of statistical machine learning and Bayesian networks has influenced the field strongly in the last 20 years and has been essential in much of its progress as well as in the deeper understanding of machine learning. This course will hence introduce these more general ideas for a more thorough theoretical underpinning. Finally, we will also discuss the important area of reinforcement learning where Richard Bellman has contributed important work already in the mid 1950s.

We will discuss how basic supervised learning resembles statistical regression. However, there are several aspects of machine learning that go beyond the scope of this traditional statistical approach. In particular, machine learning is usually concerned with high dimensional problems where many factors have to be incorporated into a model. Furthermore, a lot of emphasis is given to methods that can handle nonlinear data. Also, as already mentioned before, there are other forms of Machine Learning such as unsupervised learning in which the machines have to find some structure in the

Fig. 1.2 Some pioneers of Ai and Machine Learning. From top-left to right bottom: Alan Turing, Arthur Samuel, Richard Bellman, Frank Rosenblatt, David Rumelhart, and Judea Pearl.

data. We will discuss later how unsupervised learning has been a central factor in the development of deep networks and which also underlies important application areas of machine learning such as some form of cluster analysis and dimensionality reduction.

Also, the area of reinforcement learning describes situations in which the machine has to find appropriate actions based only on some form of feedback on the value of a state reached by a series of actions. A great example of the recent progress in reinforcement learning is the ability of a computer to learn to play video games. Video games from the old Atari platform have indeed become a useful paradigm for a new class of benchmarks that go beyond classical data sets for machine learning from the UCI machine learning repository that have dominated the benchmarks in the past. Atari games are somewhat slightly simplified worlds but resemble more learning in environments that humans have to figure out. In these benchmarks only visual input is given made up of the computer frames of the video game, and feedback is only provided with how well the player did in the game. Success in this areas was also made very visible when Google DeepMind challenged the best players of the Chinese board game Go. Go was considered to be a real challenge for AI systems as it is considered to rely a lot on ‘gut feelings’ rather quantifiable strategies. It was hence a huge success that computers which only reached levels of an advanced beginner a few years ago would win the world championship in the spring of 2016.

There are several aspects of deep learning that make this area very exciting. One aspect is that it enables representational learning, which can be seen as the foundation of much of the recent progress. A good summary of the evolution of machine learning is shown in Fig. 10.8. While traditional rule based artificial intelligence relied on hand-designed programs such as programming specific rules for inference, classic machine learning tries to find (learn) the mapping between an input and desired output. A smart feature selection and good hand-crafted representations were essential at this time for good results and much of a machine learning course would talk about this. However, we are now seeing increasingly the emphasis on end-to-end solutions where a whole tasks are learned from sensory information which includes the discovery of appropriate hierarchical representations.

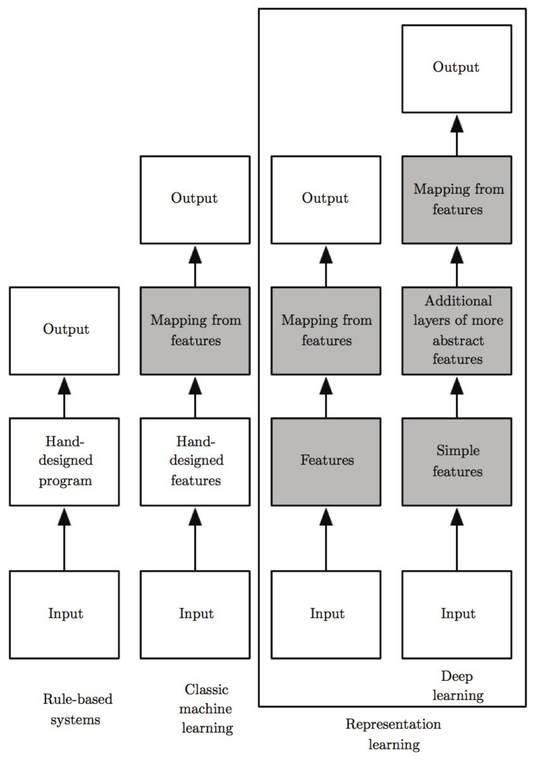
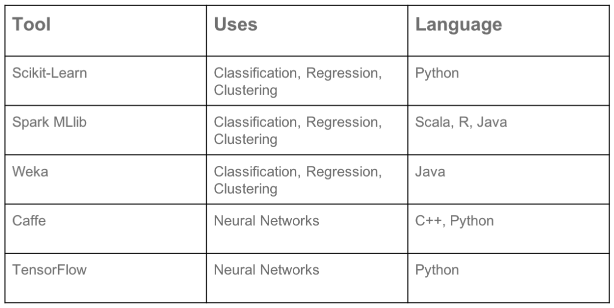


Fig. 1.3 Evolution of machine learning systems (from Goodfellow, Bengio, Courville 2015).

Deep learning is now a very important part of machine learning which we will focus on after introducing the basis of machine learning in more general terms. A deeper understanding of machine learning requires, to some extend, an understanding of data modelling in a wider context. This includes the importance of a probabilistic framework to formulate the problems and solutions. A course on machine learning also needs to include an overview of some traditional methods such as support vector machines, classification trees, and clustering methods. Applying machine learning methods is often easy in principle and difficult in practice. That is, there are now many tools available with which Machine Learning implementation can be programmed in a few lines of code. However, the correct application of these methods in practice requires some care, experience, and a deeper understanding of the underlying issues. Therefore, our approach will be to explain some of the basic ideas of supervised learning first. This includes the introduction of some basic concepts such as knowing the difference between a training set, a validation set is, and a testing set, and how cross-validation can be used for hyperparameters optimization.



Tool	Uses	Language
Scikit-Learn	Classification, Regression, Clustering	Python
Spark MLlib	Classification, Regression, Clustering	Scala, R, Java
Weka	Classification, Regression, Clustering	Java
Caffe	Neural Networks	C++, Python
TensorFlow	Neural Networks	Python

Fig. 1.4 Supervised Learning Frameworks.

In the second chapter we learn how to apply such methods with some programming frameworks. Fig. 1.4 list some of the common Machine Learning frameworks. We will be using the Python programing language together with some machine learning libraries, in particular sklearn and tensorflow. The next several chapters explore the principle behind supervised learning in the form of regression and classifications. We thereby switch frequently between a functional and a probabilistic framework. A refresher on the basic probability formalism is included in the third chapter. The following chapters outline some of the fundamental ideas behind probabilistic machine learning and some important Machine Learning algorithm including supervised and unsupervised learning, and issues beyond regression and classification such as dimensionality reduction and variable selection. The second half of the course is dedicated to neural networks and deep learning. This includes convolutional networks, autoencoders, and various recurrent networks. These subjects will be discussed with the tensorflow framework. The last section will be dedicated to (deep) reinforcement learning.

1.2 Mathematical formulation of the supervised learning problem

Much of what is currently most associated with the success of machine learning is supervised learning, sometimes also called predictive learning. The basic task of supervised learning is that of taking a collection of input data $\mathbf{x}$, such as the pixel values of an image, some measured medical data, or robotic sensor data, and predicting an output value $\mathbf{y}$ such as the name of an object in an image, the state of a patient's health, or the location of obstacles. It is common that each input has many components, such as many millions of pixel values in an image, and it is useful to collect these values in a mathematical structure such as a vectors in one dimension, a matrix in two dimensions, or generally in a tensor for higher dimensions. We often refer to Machine Learning problems as high-dimensional which refers in this context to the large number of components and not the dimension of the input tensor.

At this time we use the mathematical terms of a vector, matrix and tensor mainly to signify a data structure. In a programming context these are more commonly described as 1, 2 or 3-dimensional arrays. The difference between arrays and tensors (a vector and matrix is in some sense a special form of a tensor) is, however, that the mathematical definitions also include rules how to calculate with these data structures. This manuscript is not a course on mathematics; we are only users of mathematical notations and methods. Mathematical notation help us enormously to keep the text short while being precise. We follow here a common notation of denoting a vector, matrix or tensor with bold faced letters, whereas we use regular fonts for scalars. We usually call the input vector a **feature vector** as the components of this are typically a set feature values of an object. The output could also be a multi-dimensional object such as a vector or tensor itself. Mathematically we can denote the relations between the input and the output as a function

$$y = f(\mathbf{x}). \tag{1.1}$$

We consider the function above as a description of the **true underlying world**, and our task in science or engineering is to find this relation. In the above formula we considered a single output value and several input values for illustration purposes, although we see later that we can extend this readily to multiple output values.

Before proceeding it is useful to clarify our use of some term feature. These values, which often represent measured data in machine learning are sometimes called attributes. The term feature is then usually used in a lightly more general context as it could also include derived values such as a function of attributes or learned representations. This strict distinction is usually not necessary for our purpose so that our use of the term feature includes attributes.

The challenge for machine learning is to find this function or at least to approximate it sufficiently. Machine learning has several approaches to deal with this. One approach that we will predominantly follow for much of the course is to define a general parameterized function

$$\hat{y} = \hat{f}(\mathbf{x}; \mathbf{w}). \tag{1.2}$$

This formula describes that we make a parameterized hypothesis in which we specified a function $\hat{f}$ that depends on parameters $\mathbf{w}$ to approximate the desired input-output relation. This function is called a **model**:

A model is an approximation of a system to study specific aspects of the system and to predict novel behaviour

This often means that not all of the underlying world has to be captured in depth. For example, a building engineer might make a model of a bridge to tests its static without including the ascetic aspects that an architect might emphasize in a model. In our context the word model is synonymous with approximation. Note that we have indicated that this model is an approximation of the desired relation by using a hat symbol above the y and the f. However, we frequently drop the hat symbol when the relation is clear from the context.

In the context of machine learning, a model typically includes parameters so that their presence is synonymous with a model. The parameters are specified in this function by including the parameters, which we often specify as vector $\mathbf{w}$, behind a semicolon in the function arguments. A more appropriate mathematical statement would be that the formula defines a set of functions in the parameter space. **Learning** is the challenge to find the values for the parameters that best describe the data. Finding these parameters is usually done with a learning algorithm. A common way of such a learning algorithm is to define a function that describes our goal of learning, such as minimizing the number of wrong classifications. We will call this function the **loss function** L, although other terms are sometimes used in the literature such as objective function, error function, or risk. A common algorithm to minimize such a loss function is to use an algorithm called **gradient descent** that is an iterative method over the training data and that changes the parameters along the gradient $\triangledown\mathcal{L}$ of the loss function,

$$w_i \leftarrow w_i - \epsilon \nabla \mathcal{L} \tag{1.3}$$

where ϵ is called the learning rate and ∇ is the Nabla operator which signifies the gradient. This is a typical learning algorithm to find the parameters of a model based an example data. We elaborate on this algorithm later.

While the gradient descent can find parameters to minimize the loss of the training data, our real goal is to find the values of $\mathbf{w}$ that best predicts data that has not been seen before. Just describing the training data is somewhat more like a memory, but being able to **generalize** is the main goal of machine learning. Hence, a good solution of the machine (model) learning problem is represented by a point in the parameter space that approximates best the true underlying world. However, since we usually don't know the true underlying world we estimate how good this model is by evaluating how good new predictions are.

In some applications of supervised learning we want to predict a continuous output variable. For example, we might want to predict the price of a house from the size information. This is called **regression**. In contrast, sometimes we want to predict discrete values such as the categories of object in a picture. This is called **classification**. The output variable y in classification is called a **label**. It is now common to generally refer to the output of the supervised learner as label, even in the case of regression where we have a continuous "label". We will see later that regression and classification are anyhow closely related; for example, binary classification can be seen as a regression problems with a discrete function $f(x)$ such as a sign function which would give us two labels, positive and negative.

An important part if our treatment of machine learning is to consider cases when we might not be able to predict an exact label or output value. We thus consider usually the probability that a certain value will occur. This is very important for several reasons. For example, it is quite common that the process under investigation includes stochastic (random) factors or unknown factors that can be treated with probabilistic methods. Thus, we conjecture the the true underlying world model is better described by a probability density function

$$p(Y = y|\mathbf{x}). \tag{1.4}$$

This give the probability density, or probability mass in the case of categorical data, of the label Y having a value y given that we have an input vector $\mathbf{x}$. The arguments of density functions are provided after the horizontal line $|$, and we write random variables as upper case letters and specific values a with lowercase letters.

Formulating Machine Learning in a probabilistic (stochastic) context has been most useful and provides us with the formalization that created the most insight into this field. In the probabilistic framework we are then modelling a density function

$$p(\hat{Y} = \hat{y}|\mathbf{x};\mathbf{w}). \tag{1.5}$$

Density function approximation is in some sense a special case of function approximation as the density function is atill a function albeit with some constraint such as a normalization $\int p(y)\mathrm{d}y = 1$. However, modeling density function is also the more general case for modeling functions in the sense that they proivide the probabilistic information of how of how likely label values are given a certain input.

Also, a probabilistic framework can be used for a more general formulation of learning. Given a parameterized model as written in equation 6.47, we want to know which paraneters describe data well. This is not necessarily a single solution, and ideally we would like to know the probability density function of the parameters given data,

$$p(\mathbf{w}|y,\mathbf{x}). \tag{1.6}$$

However, most applications of machine learning use a learning principle where learning can be described as finding the most likely parameters given the data,

$$\mathbf{w}^* = \mathrm{argmax}_w p(\mathbf{w}|y,\mathbf{x}) \tag{1.7}$$

Such a maximum aposteriori choice, or some point estimates derived from related principles, are currently the dominating forms of machine learning. Of course, we still need to find the specific form of the probability function $p(\mathbf{w}|y,\mathbf{x})$, but we can then derive learning rules from this principle. The point here is to illustrate how important and useful a probabilistic formalism is in machine learning. It is therefore important to consider uncertainty in machine learning right from the outset.

Formulating specific probabilistic models for problems with many stochastic factors is demanding. An important and useful way to formulate multifariate probabilistic models is the area of **causal model**. Such models provide specific probabilistic models of the components that provide the necessary foundations of the inference engine. Inference here means that the system can be used to 'argue' about a solution in a probabilistic sense. Such systems are the domain of Bayesian networks, and we will include some introduction to this important domain in this book as it provides an important aspect of machine learning. To some extend we will argue that probabilistic models and deep neural networks represent somewhat a diverse spectrum of machine learning models, but we will also argue that they can be viewed in a unified way. Fig. 1.5 shows a famous example form Judea Pearl, one of the inventors of this important modelling framework.

1.3 Applied Learning: Training, validating and testing

A linear model in low dimensions is excellent way to demonstrate the principle mechanisms of machine learning. In the following example we consider at first only one input feature x. Supervised machine learning in this example is equivalent to linear

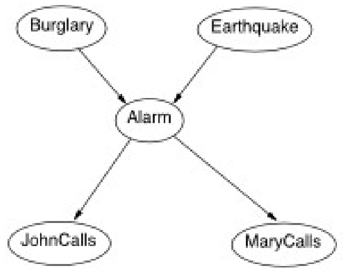


Fig. 1.5 An example of a graphical representation of a causal model.

regression, and area that has been studied for centuries. What makes machine learning different today is that we are usually considering high-dimensional non-linear problems, and that we have computers (machines) to help us with this task. For now we will follow the function approximation formalization, but we will return to the probabilistic framework later.

Coming up with the right parameterized approximation function is the hard problem in machine learning, and we will later discuss several choices. There are also methods to systematically develop the approximation function from the data, generally called non-parametric methods. At this point we assume that we have a parameterized approximation function. To illustrate this with we chose here an example where we assume we have a single input feature, x, and we hypotheses that the output y is linearly related to x. Mathematically we write this linear model as

$$\hat{y} = ax + b, \tag{1.8}$$

where a is the slope of the linear function and b is the y-axis intercept or bias of this function. Using the training data to determine good parameters is called linear regression.

The question is then how we determine good parameters. This is where the learning process comes in. In supervised learning we must be given some examples of input-output relation from which we learn. We can think about these examples as given by a teacher. The teacher data called the **training set** are used to directly determine the parameters of the model. We can denote this training set as

$$\{\mathbf{x}^{(i)}, y^{(i)}\}, \tag{1.9}$$

where the superscript i labels the specific training example. These indices are enclosed in brackets to not confuse them with exponents. An example of a training set with 4 example points are shown in the left table in Fir. 2.1.

There are several possible training algorithm to determine the parameters of a model. One way to determine the two parameters in this example is to use some training data to analytically calculate the two unknowns. As we have a linear equation with only two unknowns, we only need two data points to determine their values. Using the two first data points as **training set**, we get

$$a = \frac{y^{(2)} - y^{(1)}}{x^{(2)} - x^{(1)}} = 4.9 \tag{1.10}$$

x	y
0	1.80
1	6.72
2	7.56
3	10.15

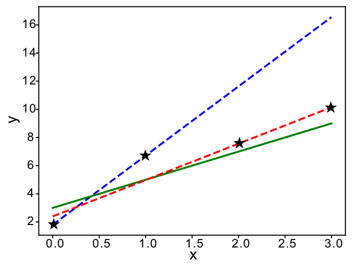


Fig. 1.6 A form of linear regression of data and cross-validation.

$$b = y^{(1)} - ax^{(1)} = 1.8 \tag{1.11}$$

This regression line is shown as blue dotted line in Fig. 2.1. To quantify the goodness of fit we need to define an evaluation function that is commonly called the **loss function** or **error function** E. We chose here to evaluate the goodness of the model with the mean square error (MSE) function,

$$E = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y} - y)^2, \tag{1.12}$$

where N is the number of data points used to calculate the loss. If we use the training data themselves to calculate the **training error** results in a zero loss, $E_{\text{training}} = 0$. We will later use more general applicable learning rules that do not always lead to zero training error with limited training, but we need to keep in mind that the training error can typically be made small and even zero and that this does not mean that the performance of the model prediction is necessarily good. In this specific example, a small or even zero training error occurs when the number of parameters in the model is large compared to the number of training data. What is more telling than the training error, and our principle aim in supervised learning, is how the models performs in predicting labels of new data points. Data that has not been used in any way in the learning process is called the **testing set**. There are sometimes competition for machine learning algorithms, and testing data for these competitions are usually withheld from the participants so that they cannot be used to develop the model. Also, it is common that some data from the training set are withheld by the model developer to 'test' the model. The test on data that have not been used in the learning process can give us an estimate about the expected performance on unseen data. If we calculate the error function from the test data we call this the **generalization error** E_{g} since this measure gives us an indication of how the model prediction performs to unseen data. The generalization error of the model with parameters calculated from the first two points in the training set is

$$E_{\text{g}}(x^{(1)}, y^{(1)}, x^{(2)}, y^{(2)}) = 21.8. \tag{1.13}$$

Of course, it would be much better to have more test data so that we can not only give a better estimate on the mean, but even provide an estimate of the range such as a variance, or in general the distribution of the generalization error. We will discuss this point further on examples later in this course.

At this point we want to ask if the choice of using the first two points was good or if we should have used to other points to build the model. Indeed, if we use the last two data points for training we get the following parameter values,

$$a = \frac{y^{(4)} - y^{(3)}}{x^{(4)} - x^{(3)}} = 2.6 \tag{1.14}$$

$$b = y^{(3)} - ax^{(3)} = 2.4, \tag{1.15}$$

which is shown as red dotted line in Fig. 2.1. So which one is better, the blue dotted line or the red dotted line? It seems that the second fits better all the points, though it also becomes now handy that we quantified the goodness of the fit with the loss function on the test data, which gives in this case the value

$$E_{\mathrm{g}}(x^{(3)}, y^{(3)}, x^{(4)}, y^{(4)}) = 1.7 \tag{1.16}$$

We can even check how well different pairs of training data can predict the other data not used in training. Using different data from the original data set for training and testing is called **cross-validation**. If we calculate the generalization error of the different training/test sets we can choose as our predictive model the parameters with the smallest cross-validation error. What we did here is that we used the originally withheld data to ultimately tune a parameter of an algorithm, here the algorithm which tells us which data points to use in the training set. This is again a form of training, we determine some parameters from example data. We will call these parameters **hyperparameters** and denote them here with the symbol θ. We will later encounter different algorithm parameters such as learning rates, momentum terms or maximum number of iterations, or which data points to use in the training set to determine w as in the above example.

If we are using the original test data to tune or train the hyperparameters, than this data that we originally called the test set is really the training set for the hyperparameter. To distinguish them we call this specific training set that is used to validate the w training in order to train the hyperparameters the **validation set**. This can be sometimes confusing and some literature even uses the term validation set and cross validation in situations of testing. In principle we could view the complete procedure, our mathematical model together with the training procedures as the hypermodel $f(\mathbf{x}; \mathbf{w}, \theta)$. If we want to test this hypermodel then we need to hold out data that are neither used in the determination of $\mathbf{w}$ nor θ. Hence in this case we need really three data sets, the training set to determine w, the validation set to determine θ, and the test set to evaluate the resulting model. It is of utmost importance not to use data in any learning process if we want to estimate the performance of the model on unseen data. Using test data in training, or even any derived information of test data in training can lead to a drastic underestimation of the generalization error. This is sometimes called **information contamination**, and information contamination can completely invalidate the results.

Note that the generalization error is still only an estimate of the true error which we usually never really know. However, since I was the one who generated the example data I can tell you how I chose them. I actually derived them from the world model

$$y = 2x + 3 + \eta, \tag{1.17}$$

where η is a normal distributed random variable. I added this random number to the perfect linear model to include a typical challenge in machine learning, that of having imprecise measurements and hence noisy training data. The other way to interpret the model is actually to accept the 'world' as stochastic and hence we are considering a stochastic model. In any case, from this model we chose some data points by sampling, though the true parameters of the world model are $a = 2$ and $b = 3$.

1.4 Non-linear regression and high-dimensionality

Above we have discussed a case where we assumed a linear function, but regression with more general non-linear functions brings another level of challenges. An example of data that do not follow a linear trend is shown in Fig.1.7A. There, the number of transistors of microprocessors is plotted against the year each processor was introduced. This plot includes a line showing a linear regression, which is of course not very good. It is however interesting to note that this linear approximation shows some systematic deviation in some regional under and over estimation of the data. This systematic deviation or **bias** suggest that we have to take more complex functions into account. Finding the right function is one of the most difficult tasks, and there is not a simple algorithm that can give us the answer. This task is therefore an important area where experience, a good understanding of the problem domain, and a good understanding of scientific methods are required.

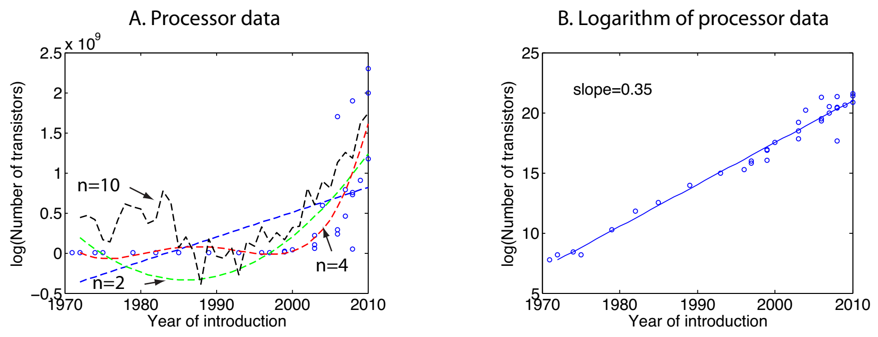


Fig. 1.7 Data showing the number of transistors in microprocessors plotted against the year they were introduced. (A) Data and some linear and polynomial fits of the data. (B) Logarithm of the data and linear fit of these data.

It is often a good idea to visualize data in various ways since the human mind seems good in 'seeing' trends and patterns. Domain-knowledge can also be valuable as specialists in the area from which the data are collected can give important advice or they might have specific hypothesis that can be investigated. It is often helpful to know common mechanisms that might influence processes. For example, the rate of change in basic growth processes is often proportional to the size of the system itself.

Such an situation leads to exponential growth. (Think about why this is the case). Such situations can be revealed by plotting the functions on a logarithmic scale or the logarithm of the function as shown in Fig.1.7B. A linear fit of the logarithmic values is also shown, confirming that the average growth of the number of transistors in microprocessors is exponential, which is known as Moore's law.

But how about more general functions. For example, we can consider a polynomial of order n, that can be written as

$$y = \mathbf{w}_0 x^0 + \mathbf{w}_1 x^1 + \mathbf{w}_2 x^2 + ... + \mathbf{w}_n x^n. \tag{1.18}$$

We would usually even consider different offsets for each term which we neglected for simplicity. Given this new hypothesis in the form of a non-linear parametric function, we can again use a regression method to determine the parameters from the data by minimizing the LMS error function between the hypothesis and the data. The LMS regression of the transistor data to polynomials for orders $n = 2, 4, 10$ are shown in Fig.1.7A as dashed lines.

Using a polynomial as a nonlinear function is only one possible choice of many. We will later consider mainly functions that that have been termed Artificial Neural Networks. These functions can be represented graphically as shown in Fig.1.8. Each node in such a graph is also called a neuron as it resembles to some extend the basic functionality of a biological neuron in the brain. Such an artificial neuron weights each individual input with an adjustable parameter, sums this weighted input, and then applies a nonlinear function such as a $\tanh$ function on this summed input. The output of each node is hence,

$$y_j = \tanh(\sum_i w_{ji} x_i). \tag{1.19}$$

This output can be the input to another node, and we can in such a way build elaborate functions with graphs of such nodes as shown in Fig.1.8B. We will later elaborate on specific network architectures that represent specific classes of non-linear functions that will be useful for specific applications. We will specifically explore how networks with many layers of neurons have advanced the capabilities of learning machines considerably which is now known as **deep learning**.

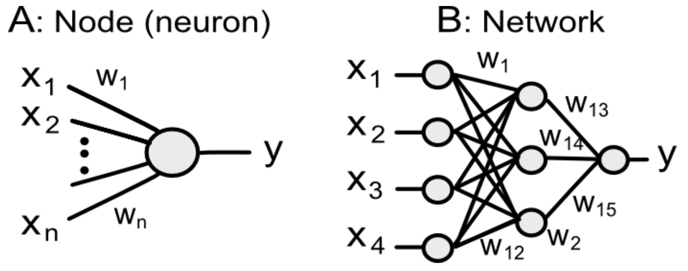


Fig. 1.8 Basic elements of an Artificial Neural Network (ANN). Each node represents an operation of summing weighted inputs and applying an nonlinear transfer functions f to this net input. The output of each node can become the input of another node or represent the output of the networks.

A major question when fitting data with fairly general non-linear functions is the complexity of the function in terms of the number of parameters such as the order of the polynomial or the number of nodes. The polynomial of order $n = 4$ seem to somewhat fit the transistor data also shown in Fig.1.7B. However, notice there are systematic deviations between the curve and the data points. For example, all the data between years 1980 and 1995 are below the fitted curve, while earlier data are all above the fitted curve. Such a systematic **bias** is typical when the order of the model is too low. However when we increase the order, then we usually get large fluctuations, or **variance**, in the curves. This fact is also called **overfitting** the data since we have typically too many parameters compared to the number of data points so that our model starts describing individual data points with their fluctuations that we earlier assumed to be due to some noise in the system. This difficulty to find the right balance between these two effects is also called the **bias-variance trade-off**.

The bias-variance trade-off is quite important in practical applications of machine learning because the complexity of the underlying problem is often not know. It then becomes quite important to study the performance of the learned solutions in some detail. A schematic figure showing the bias-variance trade-off is shown in Fig.1.9. The plot shows the error rate as evaluated by the training data (dashed line) and validation curve (solid line) when considering models with different complexities. When the model complexity is lower than the true complexity of the problem, then it is common to have a large error both in the training set and in the evaluation due to some systematic bias. In the case when the complexity of the model is larger than the generative model of the data, then it is common to have a small error on the training data but a large error on the generalization data since the predictions are becoming too much focused on the individual examples. Thus varying the complexity of the data, and performing experiments such as training the system on different number of data sets or for different training parameters or iterations can reveal some of the problems of the models.

Deep neural networks are a form of high dimensional non-linear fitting function, and preventing overfitting is therefore a very important component in deep learning. Deep networks have many free parameters, and large data sets (big data) has therefor been important for the recent progress in this area in combination with other techniques to prevent such as a technique called **dropout** that we will discuss later. In general one can think about techniques to prevent overfitting by restricting the possible range of the parameters. Indeed, learning from data already represents providing information about the values of the parameters, and restricting such ranges further is a key element in machine learning. This area is generally discussed under the heading of **regularization**. In a probabilistic framework this can be incorporated with a **prior**, a probability density function of our prior belief or restrictions in the parameters.

In the next section we will see that basic implementation of Machine learning methods are not difficult when using application programs that implement these techniques. This is good news. However, a deeper understanding of the methods is necessary to make these applications and their conclusion appropriate. The machine learning algorithms will come up with some predictions, but if these predictions are sensible is important to comprehend and evaluate. Machine learning education needs therefore to go beyond learning how to run an application program, and this course aims to find a balance between practical applications and their theoretical foundation.

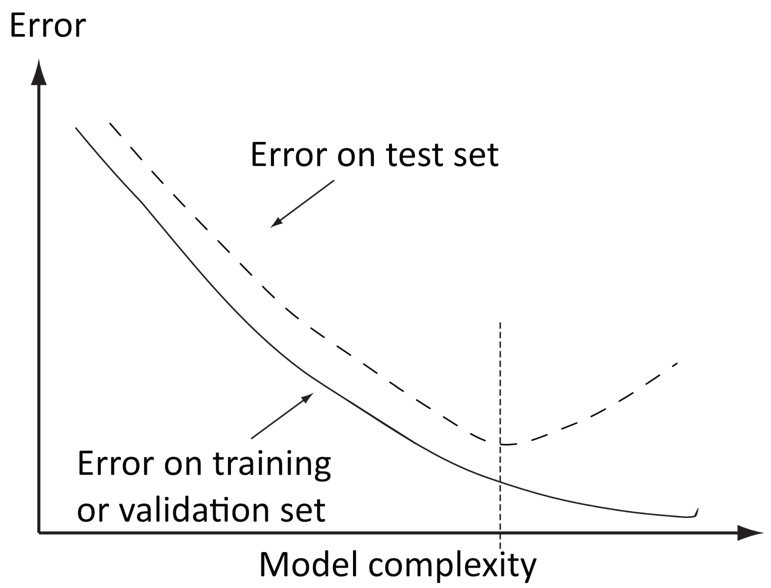


Fig. 1.9 Illustration of bias-variance trade-off.

1.5 Deep learning

We mentioned above that deep learning is basically about neural networks with many layers. However, this statement alone does not pay justice to the enormous advancement and impact deep learning had in recent years. This section outlines this progress and the reason for new exciting advancements in applications such as image processing and natural language processing.

It can be shown that a multilayer perceptron with one hidden layer is a universal function approximator. This means that the error between the network approximation and the function to be modeled can be made arbitrarily small given enough hidden nodes. However, this does not mean that the representation with only one hidden layer is always the most efficient and even appropriate at all. Indeed, it is likely that even for reasonably complex functions many hidden neurons must be used. Moreover, if we think of these functions as ways to describe objects, then it does not seem very useful to base this decision on a large number of low level features. For example, a table or a chair can be appropriately described by components, such as the legs and the armrests, and these components can in turn made up of separate components such as screws and wood. This is where hierarchical structures of describing objects seem much more appropriate and efficient.

While the idea of a deep network is straightforward, realizing them in computers has proven tricky. Several factors have been contributing to the difficulty of getting neural networks with many layers learning sufficiently with gradient-based optimization. One is that the number of parameters in the model gets much bigger so that overfitting ought to be a problem. The availability of large datasets with crowed-sourced labels and the generation of synthetic training data have this been important. Another problem has been that training of deep networks itself is often very slow, as the gradients underlying the training algorithms can become small. This vanishing gradient problem becomes more pronounced the deeper the network get. This has particularly been the problem when using a sigmoidal transfer function, and using piecewise linear functions such as the rectified linear unit (RELU), $f(x) = x$ for $x > 0$ and zero otherwise, as considerable helped with training.

Increasing the number of layers in a neural network will drastically increase the number of parameters (weights) of the model. Also, the all-to-all connections of multilayer perceptrons do not seem the right approach for computer vision for the following reason. Say we have a neuron that recognizes an edge at a specific location in a visual scene; then this neuron would only specialize to this scene location and we would have to learn an edge detector for other locations. This seems a waste of resources; not only do we need many edge detectors, but we have to train them from examples of edges at all possible locations in a visual scene since edges could be everywhere. Hence, it seems much better to have one edge detector and to apply this specific filter to all possible locations in the visual scene. Such an operation is called a convolution, and convolutional neural networks such as the one shown in Fig. 1.10 are now the workhorse in computer vision.

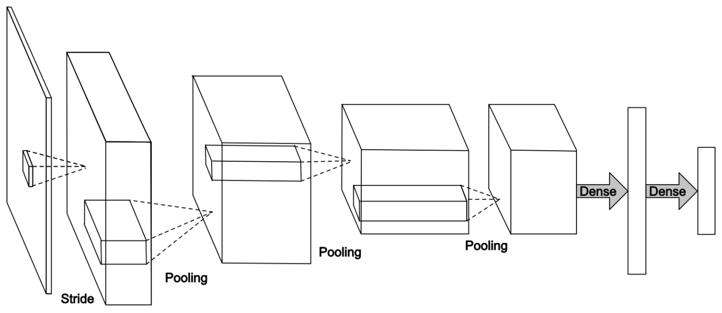


Fig. 1.10 A typical design of a deep convolutional network for image processing. A convolutional layer is followed by some pooling layer which compresses the images fopr example by averaging over some pixels. At the end some fully connected layers as in MLPs are commonly used (adapted from Krizhevsky et al., 2012).

Another important domain of deep learning are recurrent networks. Such networks are important for modeling sequences where a value of a sequence at a certain position should be predicted from previous data points. Many applications are sequences at discrete times, like the value of the stock market at each hour, and we hence speak commonly about time instead of sequence position. To do this case we can pass back the state of the hidden state with a modulation value (weight) of less then one. In this way we can implement a form of exponentially decaying short term memory. This way of representing a sequence in neural networks does therefore include feedback connections, usually called recurrent connections with neural networks. An example is shown in Figure 8a.

What is interesting in this case compared to the tapped delay line input to a purely feedforward network is that we introduced a form of weight sharing in the sense that

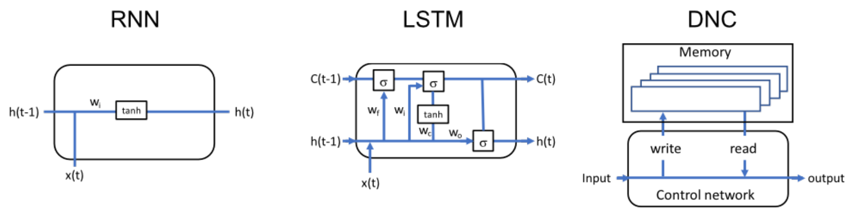


Fig. 1.11 A) Illustration of a hidden layer of the recurrent network (adapted from Chris Olahâ^s blog). B) Representation of a gated RNN called LSTM in a similar style to A). C) The more genral network with an external memory such as the Differential Neural Computer (DNC).

only the relative times of the sequence are important. This assumption is similar to the position invariant assumption in convolutional networks, and such assumptions enable much simpler yet larger models through some form of weight sharing. As stated above, the assumption of a diminishing influence from previous time steps has the form of an exponential decay which is not always appropriate such as in language processing, but we will later see how we can amend this architecture to allow more flexible memory structures. Of course, in order to build a deeper network we have to include non-linearities. The term 'recurrent' comes from the fact that such networks were often viewed with the analogy of an electric flow in a circuit. Sometimes such an information flow has been termed 're-entry'. Another way to visualize this simple RNN is shown on the right side of Figure 8 which is an adaptation from the excellent description of recurrent networks in Chris Olah's blog at http://colah.github.io. The inputs are now vectors and the connections represent weight matrices. To train these networks we have to unfold these networks in time. This is demonstrated for the simplest version in Figure 9.

Figure 9: Unfolding of a recurrent network in time so that it can be trained with the error-backpropagation learning algorithm.

So, to this end, what have we gained? We simplified the graph of a feed-forward network and replaced it with a recursive version, only to unfold it in time again in order to train it. However, note that we have to do the unfolding only during training (which is of course bad enough), and that we still have a form of weight sharing that makes these models much easier than the general ones with which we started.

4b Gated RNNs As mentioned above, the basic recurrent network has a form of memory that takes earlier states into account. However, the influence of these states is exponential fading which is not always appropriate. For example, in language processing it is necessary to take some context into account that might be remotely relative to words at the current time. Or in other word, some memories should only 'kick in' at some appropriate time. It is hence important to gate some of this information until they are useful at a later state of processing. The first network which has taken this into consideration is called LSTM which stands for Long Short Term Memory (Hochreiter and Schmidhuber 1997). This network is illustrated in Figure 10.

Figure 10: A gated recurrent network called Long Short Term memory (LSTM) that has an internal memory state and corresponding gates to erase, write and read from this state.

The gated network introduces an explicit cell state C(t), or intrinsic memory state, that can be forwarded to the next time step. This cell state can be modified with two separate operations, a forgetting gate and a write gate that are indicated with sigmoid units. The new memory state is updated with these factors and of course the new input. The interesting thing is that the gating functions are also leaned with corresponding weight values and a sigmoidal gain function of the logistic variety to scale these terms with a value between 0 and 1. Finally, the output for this hidden node is calculated from this internal memory state. A simplified and popular variant of LSTM is the called the Gated Recurrent Unit (GRU) (Chung et al. 2014).

Figure 12: Differential Neural Computer (DNC) generalizes the architecture of LSTM to a more general external memory with read, write, and erase functions (called heads). The architecture is still fully trainable with gradient descent methods as all operations are differentiable. Gated recurrent networks like LSTM and GRU are important networks which have gained increasing popularity for temporal processing. Each processing unit in these networks has some form of gated memory. A further step is then to take the idea further in the form of an external memory. The first version of such a model was the neural Turing machine (NTM), which was later refined to the Differentiable Neural Computer (DNC). An graphical outline of such a model is illustrated in Figure 12.

1.6 No free lunch and choice of models

Neural networks and other models such as support vector machines and decision trees are fairly general models in contrast to Bayesian models that seem to specify much better a causal structure. Such specific models, as long as they faithfully represent the underlying structure of the world model, should always outperform more general model. This fact is captured by David Wolpert's "No free lunch" theorem which basically states that there is not a single algorithms that covers all applications better than some other algorithms. Applying machine learning algorithms is therefore somewhat of an art and requires experience and knowledge of the constraints of the algorithms.

We will see in Chapter 3 writing a program that applies ML algorithms to data is usually not too difficult. It is common that new algorithms will find their way to graphical data mining tools, which makes them available to an even larger application community. However, applying such algorithms correctly in different application domains can be challenging and it is well known that some experience is required. We therefore concentrate in the following in explaining what is behind these algorithms and how different theoretical concepts are explored by the different algorithms. Some understanding of the algorithms is absolutely necessary to avoid pitfalls in their application.

The basic first step for the application of ML methods is how to represent the data. We discussed already above how to convert different type of inputs to numerical vectors or tensors. However, there are usually many different possibilities to represent the data, such as a fine grain representation or using some summary statistics. In the past it has been crucial to work out an appropriate high level data representation. However, the recent progress in deep learning made it possible to treat this representation itself as part of the learning problem. Representational learning has thus become an important part of machine learning.

Once the problem has been defined by representing the data and possible goals in an appropriate way, and once the appropriate ML algorithm has been chosen, it is then the main challenge to chose good parameters of the algorithms such as the number of neurons or layers of neurons in neural networks, which kernel to use in support vector machines, how many training steps to take in gradient descent learning, or how many data to use for learning versus validation. We call these parameters of the algorithms the **hyperparameters**. Choosing the right hyperparameters is commonly a major question and to make it clear from the start, there is no simple answer. Thinking about how to approach this question with appropriate experiments and to understand the options and possible approaches is hence a mayor part of machine learning applications.