MACHINE LEARNING

Jaideep Ganguly, Sc.D.

Contents

Preface

Machine Learning perceived as an esoteric domain of *data scientists*. In reality, it is an intersection of *statistics* and *computer science*. In this book, I have attempted to present the subject of machine learning in a lucid yet comprehensive way.

Topics are explained by starting with the physical explanation of the phenomenon, followed by a rigorous mathematical treatment of the algorithm and finally with python code samples.

CHAPTER 1

History of Machine Learning

The excitement around Machine Learning and Deep Learning prompted me to trace the history of Artificial Intelligence at the Massachusetts Institute of Technology (MIT) and elsewhere and take stock of the current state of Machine Learning. Arthur Samuel, who is best known for the checkers playing program, is credited with coining the name "Machine Learning". Before we get started, a quick overview of some terms; Learning is acquisition of knowledge, discovery is the observation of a new phenomena and invention is the process of making something new. Learning is necessary for invention but is not a sufficient condition for innovation. Machine Learning as it stands today, does not invent but it does discover patterns in large quantities of data. In particular, Deep Neural Networks, have attracted the imagination of many because of some interesting solutions it offers in all three channels - text, speech and images. Incidentally, most deep neural nets are rather wide and are usually not more than ten layers deep. So the name should have really been "wide neural nets" but the word "deep" has stuck.

The question of whether a computer can think is no more interesting than the question of whether a submarine can swim, said Dijkstra. It is more interesting to understand the evolution of Machine Learning - how did it start, where are we today and where do we go from here. The human brain is a remarkable thing; it has enabled us understand science and advance mankind. The idea of mimicking the human brain or even improving the human cognitive functions is an alluring one and is an objective of Artificial Intelligence research. But we are not even close in-spite of a century of research. However, it continues to have a major hold on our imagination given the potential of the rewards.

It seems that about 50,000 years ago, after we have been around for about a hundred thousand years or so, palaeontologists believe that some of us, possibly just a few thousand, were able to deal with symbols. This is a major step in evolution. Noam Chomsky thinks we were then able to create a new concept from 2 existing ideas or concepts without damaging or limiting the existing concepts.

Around 350 BC, Aristotle devised syllogistic logic, the first formal deductive reasoning system to model the way humans think about their world and reason with it. 2,000 years later, Bertrand Russel and Alfred Whitehead published Principia Mathematica that laid down the foundations for a formal representation of Mathematics. John McCarthy, who championed the cause of mathematical logic in AI, was Aristotle of his day.

In 1942, Alan Turing showed that any form of mathematical reasoning could be processed by a machine. By 1967, Marvin Minsky declared that "within a generation, the problem of creating Artificial Intelligence would substantially be solved". Clearly we are not there yet, attempts to build systems with first order logic as described by early philosophers failed because of lack of computing power, inability to deal with uncertainty and lack of large amounts of data.

In 1961, Minsky published "Steps towards Artificial Intelligence" where he talked about search, matching, probability, learning and it was quite visionary. Turing told us that it was possible to make a machine intelligent and Minsky told us how. In 1986, Minsky wrote the highly influential book "The Society of Mind", 24 centuries after Plato wrote "Politeia"; Minsky was Plato of his days. Minsky taught us to think about heuristic programming, McCarthy wanted us to use logic to the extreme, Newel wanted to build cognitive models of problem solving and

Simon believed that when we see something that is complicated in behavior, it is more of a consequence of a complex environment rather than because of a complex thinker. Thereafter, a number of model backed systems were built. Terry Winograd built a model backed system for dialog understanding, Patrick Winston built another model backed system for learning and Gerald Sussman built a model backed system for understanding blocks. During the same era Roger Schank believed that understanding stories is the key to modeling human intelligence. David Marr, who is best known for his work on vision, treated vision as an information processing system. Marr's tri-level hypothesis in cognitive science comprised of a computational level - what does the system do, an algorithmic level - how does the system do and a physical level - how is the system physically realized, e.g., in the case of biological vision, what neural structures and neuronal activities implement the visual system.

In the 1980s, the expert systems were of great interest and focused on knowledge and inference mechanisms. While these systems did a pretty good job in their domains, they were narrow in specialization and were difficult to scale. The field of AI was defined as computers performing tasks that were specifically thought of as something only humans can do. However, once these systems worked, they were no longer considered to be AI! For example, today the best chess players are routinely defeated by computers but chess playing is no longer really considered as AI! McCarthy referred to as the "AI effect". IBM's Watson is a program at a level such as that of a human expert but it is not certainly not the first one. Fifty years ago Jim Slagle's symbolic integration program at MIT was a tremendous achievement. Nevertheless, it is very hard to build a program that has "common sense" and not just narrow domains of knowledge.

Today, at the core is the debate between logic inspired and neural network inspired paradigms for cognition. LeCun, Bengio and Hinton state that succinctly in a review paper in Nature, dated 28th May 2015, as "The issue of representation lies at the heart of the debate between the logic-inspired and the neural-network-inspired paradigms for cognition. In the logic-inspired paradigm, an instance of a symbol is something for which the only property is that it is either identical or non-identical to other symbol instances. It has no internal structure that is relevant to its use; and to reason with symbols, they must be bound to the variables in judiciously chosen rules of inference. By contrast, neural networks just use big activity vectors, big weight matrices and scalar non-linearities to perform the type of fast 'intuitive' inference that underpins effortless commonsense reasoning".

Rosenblatt is credited with the concept of Perceptrons, "a machine which senses, recognizes, remembers, and responds like the human mind" as early as in 1957 but in a critical book written in 1969 by Marvin Minsky and Seymour Papert showed that Rosenblatt's original system was painfully limited, literally blind to some simple logical functions like XOR. In the book they said: "... our intuitive judgment that the extension (to multilayer systems) is sterile". This intuition was incorrect and the field of "Neural Networks" pretty much disappeared! Geoff Hinton built more complex networks of virtual neurons that allowed a new generation of networks to learn more complicated functions (like the exclusive-or that had bedeviled the original Perceptron). Even the new models had serious problems though. They learned slowly and inefficiently and couldn't master even some of the basic things that children do. By the late 1990s, neural networks had again begun to fall out of favor.

In 2006, Hinton developed a new technique that he dubbed deep learning, which extends earlier important work by Yann LeCun. Deep learning's important innovation is to have models learn categories incrementally, attempting to nail down lower-level categories (like letters) before attempting to acquire higher-level categories (like words).

In April 2000, in a seminal work published in Nature by Mriganka Sur, et. al, at MIT's laboratory for brain and cognitive sciences, the authors were able to successfully "rewire" brains in very young mammals, inputs from the eye were directed to brain structures that normally process

hearing. The animal's auditory cortex successfully interpreted input from its eyes. But it didn't do the job as well as the primary visual cortex would have, suggesting that while the brain's plasticity, or ability to adapt, is enormous, it is limited by genetic preprogramming. Environmental input, while key to the development of brain function, does not "write on a blank slate". This addresses an age-old question – is the brain is genetically programmed or shaped by environment? It is a dramatic evidence of the ability of the developing brain to adapt to changes in the external environment, and speaks to the enormous potential and plasticity of the cerebral cortex – the seat of our highest abilities. This provided some theoretical underpinning to the neural net computation theory.

Deep neural nets spawned a subset known as Recurrent Neural Nets which were an attempt to model sequential events. Support Vector Machines, logistic regression, feedforward networks have proved very useful without explicitly modeling time. But the assumption of independence precludes modeling long range dependencies. DNNs were also helped by the emergence of GPUs which enabled parallelism as much of computation in DNN is intrinsically parallel in nature. RNNs are connectionist models with the ability to selectively pass information across sequence steps while processing sequential data one element at a time. They can model input and/or output consisting of sequences of elements that are not independent. However, learning with recurrent networks is difficult. For standard feedforward networks, the optimization task is NP-complete. Learning with recurrent networks is challenging due to the difficulty of learning long-range dependencies. Problems of vanishing and exploding gradients occur when back propagating errors across many time steps. In 1997, Hochreiter and Schmidhuber introduced the Long Short Term Memory (LSTM) model to overcome vanishing gradients. LSTMs have been proven to be remarkable in speech and handwriting recognition. Similarly, another variation of the Deep Net Model is the Convolution Neural Network (CNN) that has been very successful in classifying images.

We have come a long way, the name "Machine Learning" is indicative of the potentials that it can possibly achieve in the future. Deep Nets appear to be very promising in some areas although they are computationally very expensive. However, deep learning is only part of the larger challenge of building intelligent machines. It lacks ways of representing causal relationships, have no obvious ways of performing logical inferences, and is still a long way from integrating abstract knowledge, such as information about what objects are, what they are for, and how they are typically used. The most powerful A.I. systems, like Watson use techniques like deep learning as just one element in a very complicated ensemble of techniques, ranging from the statistical technique of Bayesian inference to deductive reasoning.

CHAPTER 2

Introduction

Machine Learning (ML) is a subset of Artificial Intelligence (AI). In 1959, Arthur Samuel, one of the pioneers of machine learning, defined machine learning as a *"field of study that gives computers the ability to learn without being explicitly programmed"*. Machine Learning algorithms discover hidden patterns in data and make predictions about future data.

In mathematical terms, machine learning algorithms establishes a function, i.e., a relationship, between input X to output Y. In machine learning, a feature is an individual measurable attribute of a phenomenon that is being observed. This means given training examples (X_i, Y_i) where X_i is the feature vector and Y_i the target variable, learn a function F to best fit the training data, i.e., $Y_i \approx F(X_i)$ for all i. For a new sample X with unknown Y, predict Y using $F(X)$.

2.1 Why Machine Learning?

For complex tasks where deterministic solution don't suffice, e.g., speech recognition, handwriting recognition, etc. Other examples include repetitive tasks needing human-like expertise such as recommendations, spam and fraud detection, personalization which cannot be solved manually in a large scale.

2.2 Types of Machine Learning

ML algorithms are classified as Supervised or Unsupervised depending on how it has been trained.

2.3 Supervised Learning

In supervised learning, the machine learns from data that has already been tagged with the correct answer by an expert. This data is typically termed as the training data or training set. Thereafter, when the machine is provided with a new set of data, it generates the answer based on its learning. The majority of practical machine learning uses supervised learning. An example of supervised learning is separating spam from legitimate email.

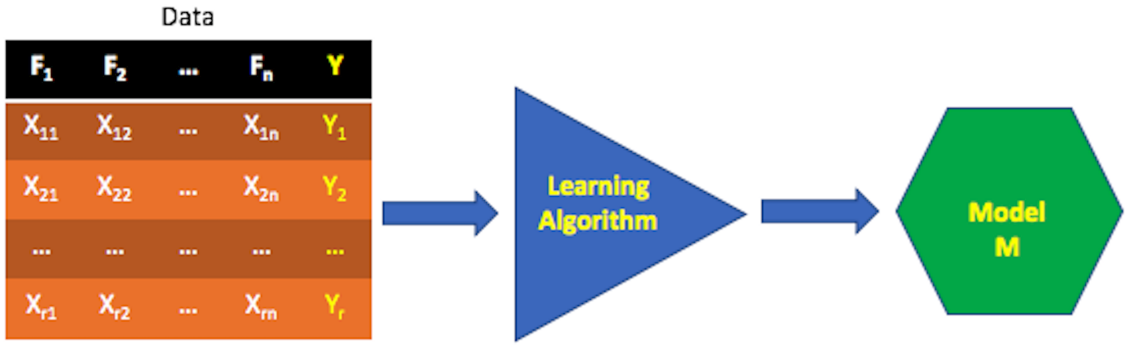


Figure 2.1 – Machine Learning.

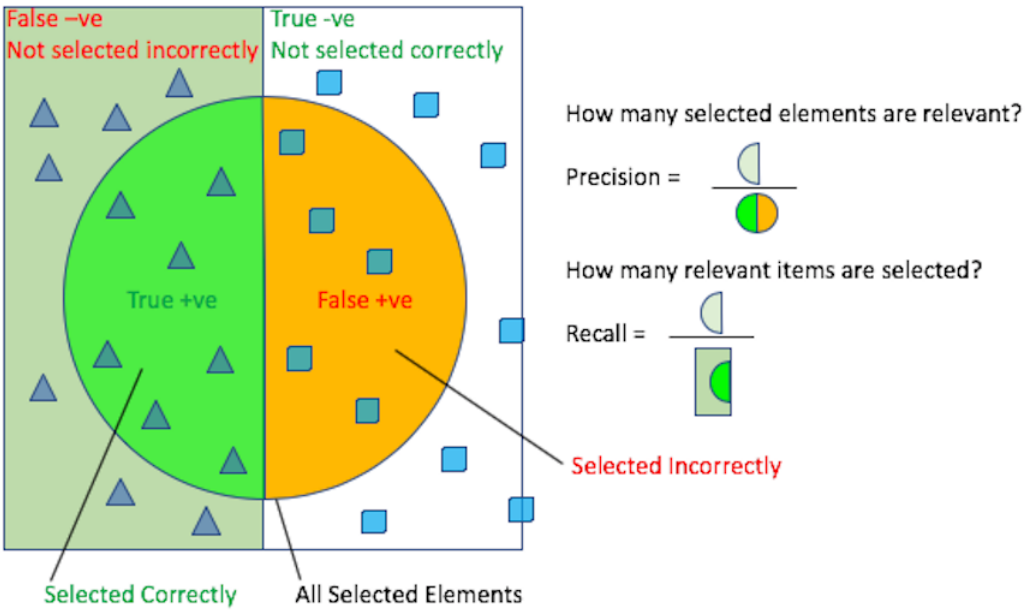


FIGURE 2.2 – Precision & Recall.

2.4 Unsupervised Learning

In unsupervised learning, the machine groups unsorted information according to similarities, patterns and differences without any prior training of data. The machine automatically figures out the patterns on its own. An example of unsupervised learning is playing the game of GO.

2.5 Definitions

Features are vectors and are represented as X. X_i represents the data corresponding to the ith data set and n is the number of features.

$$X_i = [x_{i1}\ x_{i2}\ x_{i3}\ \cdots\ x_{in}] \tag{2.5.1}$$

$$Precision = \frac{t_p}{t_p + f_p} \tag{2.5.2}$$

$$Recall = \frac{t_p}{t_p + f_n} \tag{2.5.3}$$

$$Accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{2.5.4}$$

$F1$ is the harmonic mean of precision and recall and is given by:

$$F1 = \frac{1}{2} \times \left(\frac{1}{Precision} + \frac{1}{Recall} \right) \tag{2.5.5}$$

2.6 Model

2.6.1 Regression

In machine learning, we often use Regression to establish a model. Regression is a measure of the relation between the mean value of one variable (e.g. output) and corresponding values of other variables. In machine learning, we use regression to determine the relation between X_i and Y_i.

2.6.2 Linear Regression

A common function used to model training data is a linear regression model. The model is linear, i.e., the relationship does not involve any quadratic or other higher order terms. The model or the hypothesis is given by:

$$y_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} \cdots + \theta_n x_{in} \tag{2.6.1}$$

where x_{ij} is the observed value of the feature x_j, y_i is the predicted value of the outcome and θ_i are constants the values of which need to be determined. For now, we limit the the values of the features x_{ij} to numbers, positive or negative. Later on, we will study techniques on how to deal with the situation where the feature value is a string.

2.6.3 Logistic Regression

In many cases the value of y_i need to be bounded between 0 and 1. In such cases, we use the logistic regression model as given below:

$$y_i = \frac{1}{1 + e^{-z}} \tag{2.6.2}$$

where z is given by:

$$z_i = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \cdots + \theta_n x_n \tag{2.6.3}$$

Equation 1.6.2 is known as sigmoid and its shape is shown in 2.3. Note that:

- for z is large +ve number, $\frac{1}{e^z} = 0$; $y_i = 1$
- for z = 0, $y_i = 0.5$
- for z is large -ve number, $y_i = 0$

Hence, the value of y_i is bounded between 0 and 1 for z between -∞ and ∞.

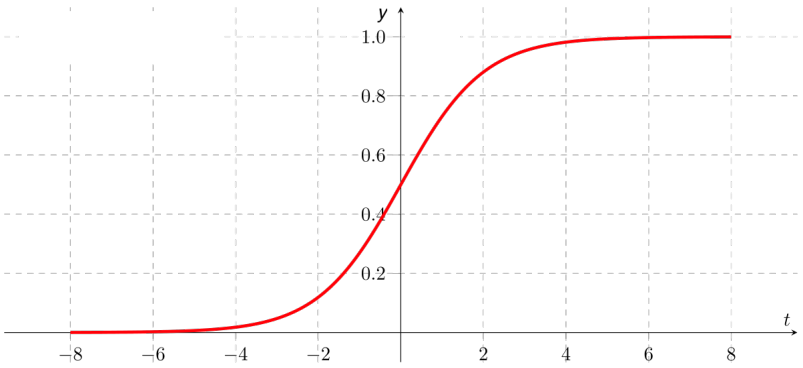


FIGURE 2.3 – Sigmoid.

2.7 Loss Function

Our objective is to find a good model that fits the training data. For that, we need to minimize some loss function over the training data D. Sometimes the word cost function is used in lieu of loss function. The total loss L corresponding to r set of data is given by:

$$L = Squared\ Error\ Loss = \frac{1}{2} \times \sum_{i=1}^{r} (\hat{y}_i - y_i)^2 \tag{2.7.1}$$

where $\hat{y}_i$ is the observed value of the ith outcome and n is the number of features. The factor 1/2 is introduced to make subsequent computations simpler as will see in the next chapter. Note that the square of each error is considered so that positive and negative errors do not cancel out.

Our objective is to find a set of θ_j in 2.6.1 and 2.6.3 such that the loss is minimum. But before we do that, let us first understand the nature of potential errors.

2.8 Errors

As you can see, predictions from the model, i.e., the hypothesis will have differences or errors. For example, consider the following diagram that depicts a correct fit.

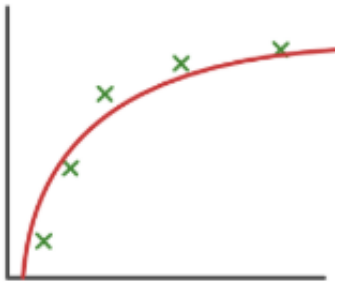


FIGURE 2.4 – Correct Fit.

There can be two types of errors.

2.8.1 Overfitting

Overfitting occurs when the model fits the training data well but does not generalize well to unseen data. In linear regression, overfitting will occur when an excessive number of features are used than required.

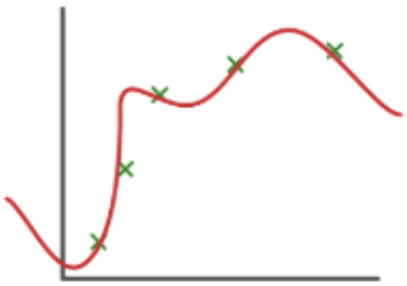


FIGURE 2.5 – Overfit.

2.9 Underfitting

Underfitting occurs when the model is simplistic and lacks expressive power to capture the entire phenomenon. The model fits the training data well but does not generalize well to unseen data. In linear regression, underfitting will occur when an insufficient number of features are used than required.

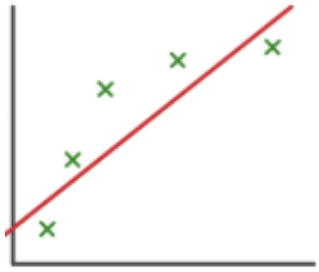


FIGURE 2.6 – Underfit.

2.10 Bias & Variance

For a given model, bias is the difference between average model prediction and the true target value and variance is the variation in predictions across different training data samples. Simple models with small number of features have high bias and low variance whereas complex models with large number of features have low bias and high variance.

2.11 Regularizaton

Regularization is a technique used to avoid problem of overfitting. It prevents overfitting in linear models by a penalty term that penalizes large weight values. A regression model that uses L1 regularization (or L1 Norm) technique is called Lasso Regression and model that uses L2 regularization (or L2 Norm) is called Ridge Regression. The key difference between these two is the penalty term.

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds the absolute value of magnitude of coefficient as a penalty term to the loss function. Here the highlighted part represents L1 regularization element where λ is a constant.

$$L + \boxed{\lambda \sum_{j=1}^{n} |\theta_j|} \tag{2.11.1}$$

Ridge regression adds the squared magnitude of coefficient as a penalty term to the loss function. Here the highlighted part represents L2 regularization element.

$$L + \boxed{\lambda \sum_{j=1}^{n} \theta_j^2} \tag{2.11.2}$$

Note that if λ is zero, the loss is the same as the squared error. The key difference between these techniques is that Lasso shrinks the less important feature's coefficient to zero thus removing some features altogether. Hence, this works well for feature selection where we have a large number of features. Mathematically, L1 regularization produces sparse coefficients while L2 regularization produces non-sparse coefficients. However, the L1-norm does not have an analytical solution, whereas the L2-norm does. This allows the L2-norm solutions to be computed efficiently.

We reduce bias by increasing model complexity, i.e., by adding more features and decreasing regularization.

We reduce variance by increasing training data and decreasing model complexity, i.e., by better feature selection and aggressive regularization.

CHAPTER 3

Gradient Descent

In this chapter we will study the Gradient Descent method to solve the linear regression as well as the logistic regression models. By solving we mean determining the values of coefficients θ_j such that the loss function is minimized.

3.1 Linear Regression with Gradient Descent

In the previous chapter, we introduced linear regression and the squared error loss function.

$$y_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} \cdots + \theta_n x_{in} \tag{3.1.1}$$

$$\text{Squared Error Loss (L)} = \frac{1}{2} \times \sum_{i=1}^{r} (\hat{y}_i - y_i)^2 \tag{3.1.2}$$

Notice that the loss function is always positive, there is a minimum and is monotonically increasing from that minimum value in both positive and negative directions along the x-axis. Such a function is called a convex function. Mathematically, A convex function is a continuous function whose value at the midpoint of every interval in its domain does not exceed the arithmetic mean of its values at the ends of the interval.

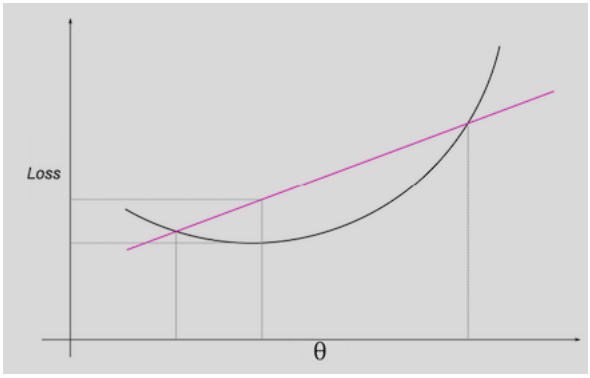


Figure 3.1 – Convex Function.

Such a loss function can be minimized by setting the partial derivative of the loss with respect to θ_j to zero. Using the chain rule in calculus, we have:

$$\frac{\partial L}{\theta_0} = \sum_{i=1}^{r} (\hat{y}_i - y_i) = 0 \tag{3.1.3}$$

$$\frac{\partial L}{\theta_j} = \sum_{i=1}^{r} (\hat{y}_i - y_i) x_{ij} = 0 \tag{3.1.4}$$

Notice that the factor 1/2 disappears. It was for this reason the factor 1/2 was introduced in the earlier chapter to simplify the calculation. We will use these equations for determining θ_j.

We begin by randomly assigning values to θ_j. For a convex function it does not matter what the initial weights are as it will always converge to their correct values. Notice that in the above

equations, $\hat{y}$ and x_{ij} are observed and y_i is computed. This means the slope of the loss function can be readily computed.

Now for a given θ_j, and keeping all other θ_i where $i \neq j$ constant, we change the value of θ_j slightly and compute a new value of θ_j

$$\triangle\theta_j = \alpha\theta_j$$
$$\theta_j^{'} = \theta_j + \triangle\theta_j$$

where α is small constant and is called the learning rate. Since the slope of $\frac{\partial L}{\theta_j}$ is already calculated from the previous equations, we can compute the new value of the loss as follows:

$$L^{'} = L - \frac{\partial L}{\theta_j}\alpha\theta_j$$

If $L^{'} < L$, we continue incrementing θ_j as long as the loss continues to decrease. When the direction changes and the loss increases, we have found the θ_j for which the loss is minimum. If $L^{'} > L$, we travel in the reverse direction and follow the same process. Like this, we compute the value of each and every θ_j.

3.1.1 Stochastic Gradient Descent (SGD)

The word stochastic means a system or a process that is linked with a random probability. In SGD, a single sample data is used to compute the θ_j. The sample is randomly shuffled and selected for performing the iteration.

3.1.2 Batch Gradient Descent

In batch gradient descent, the the gradient is calculated from the whole training set and hence the word "batch". If we have a huge dataset with millions of data points, running the batch gradient descent can be quite costly since we need to reevaluate the whole training dataset each time we take one step towards the global minimum.

3.1.3 Mini Batch Gradient Descent

A compromise between computing the true gradient for the entire batch and the gradient at a single example is to compute the gradient for a mini batch at each step. This can perform significantly better than stochastic gradient descent because the code can make use of vectorization libraries rather than computing each step separately. It may also result in smoother convergence, as the gradient computed at each step uses more training examples. It is the algorithm of choice for neural networks with the batch sizes are usually between 50 and 256. Mini Batch Gradient Descent runs much faster than the Batch Gradient Descent. In Batch Gradient Descent, the Batch Size refers to the total number of training examples present in a single batch and Epoch means that each sample in the training dataset has had an opportunity to update all the model parameters θ_j.

3.1.4 Optimization

Optimizing techniques such as the momentum method helps accelerate gradients vectors in the right directions, thus leading to faster convergence. Instead of using only the gradient of the current step to guide the search, momentum also accumulates the gradient of the past steps to

determine the direction to go. An update term is added as follows:

$$\begin{aligned} \text{update}_t &= \gamma \times \text{update}_{t-1} + \eta \nabla L \\ L_{t+1} &= L_t - \gamma \times \text{update}_t \end{aligned}$$

$$\begin{aligned} \text{update}_0 &= 0 \\ \text{update}_1 &= \gamma \times \text{update}_0 + \eta \nabla L_1 = \eta \nabla L_1 \\ \text{update}_2 &= \gamma \times \text{update}_1 + \eta \nabla L_2 = \gamma \eta \nabla L_1 + \eta \nabla L_2 \\ &\vdots \\ \text{update}_t &= \gamma \times \text{update}_{t-1} + \eta \nabla L_t = \gamma^{t-1} \eta \nabla L_1 + \gamma^{t-2} \eta \nabla L_2 + \cdots \eta \nabla L_t \end{aligned}$$

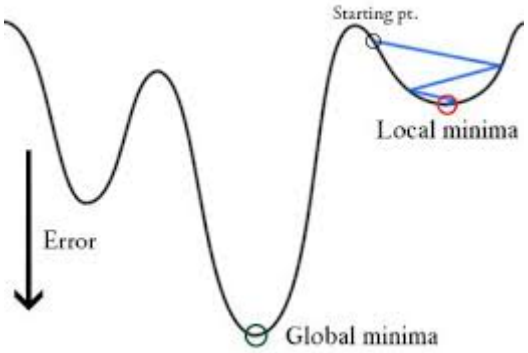
where γ is slightly less than 1. This implies that we have the maximum faith in the latest gradient and an exponentially decaying faith in the previous gradients.

3.2 Logistic Regression

In the previous chapter, we used a sigmoid function to model logistic regression as shown below:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T X}} \tag{3.2.1}$$

For logistic regression, Least Squared Error will result in a *non-convex* graph with local minimums and hence is not feasible.



Logistic regression is modeled as follows:

$$P(y|x,\theta) = \begin{cases} h_\theta(x) & \text{if y = 1} \\ (1 - h_\theta(x)) & \text{if y = 0} \end{cases} \tag{3.2.2}$$

The above function can be written compactly as follows:

$$P(y|x,\theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y} \tag{3.2.3}$$

Assuming that the r training examples were generated independently, we can then write down the likelihood L of the parameters as:

$$L(\theta) = p(\vec{y}|x,\theta) = \prod_{i=1}^{r} P(y^{(i)}|x^{(i)},\theta) = \prod_{i=1}^{r} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \tag{3.2.4}$$

The log likelihood is given by:

$$l(\theta) = log\, L(\theta) = \prod_{i=1}^{r} y^{(i)} log(h(x^{(i)}) + (1 - y^{(i)}) log(1 - h(x^{(i)})) \tag{3.2.5}$$

Intuitively, we want to assign more punishment when the prediction is 1 while the actual is 0 and similarly when the prediction is 0 while the actual is 1. The loss function in logistic regression is doing exactly this and hence is called Logistic Loss.

The sigmoid function has an interesting property. It's derivative is given by:

$$g^{'}(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = \frac{e^{-x}}{(1+e^{-x})^2} \quad = \frac{1}{1+e^{-x}} \times \left(1 - \frac{1}{1+e^{-x}}\right) \quad = \boxed{g(x)(1-g(x))} \tag{3.2.6}$$

Now, consider just one training example (x,y), and take derivatives to derive the stochastic gradient ascent rule:

$$\begin{aligned}
\frac{\partial(l(\theta))}{\partial\theta_j} &= y\frac{1}{g(\theta^T x))} - (1-y)\frac{1}{1-g(\theta^T x)}\frac{\partial}{\partial\theta_j} g(\theta^T x) \\
&= y\frac{1}{g(\theta^T x))} - (1-y)\frac{1}{1-g(\theta^T x)} g(\theta^T x)(1-g(\theta^T x))x_j \\
&= y(1-g(\theta_T x) - (1-y)g(\theta^T x)x_j \\
&= y - h_\theta(x))x_j
\end{aligned}$$

With the above equation, we are now in a position to compute the gradient and follow the same steps to determine θ_j corresponding to minimum loss as in the case of linear regression.

3.3 Softmax

In the training data the values corresponding to the features x_j can have numerical values of different magnitudes. For example, one feature may have values ranging between $0 \ldots 10$, while another feature may have values ranging between $10,000 \cdots 100,000$. Some of these values can also be negative and the components will obviously not add up to 1.

The softmax function is a simple function as given below that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities in the interval 0...1 with the components adding up to 1. The larger input components will correspond to larger probabilities.

$$\sigma(z)_i = \frac{e_i^z}{\sum_{j=1}^{K} e^{z_j}} \tag{3.3.1}$$

The feature values are normalized with the softmax function prior to inputing in a regression model.

3.4 One Hot Encoding

In the linear regression and the logistic regression models that we studied so far can only deal with numerical data. It cannot handle features that have text as values. In one-hot encoding, the string encoded variable is replaced with new variables of boolean type. For example, a particular feature, say color, can have "red", "green" and "blue" as possible values. This feature color, will be replaced by 3 features, red, blue and green which hold the values 1 or 0. We can then encode color, in terms of red, blue and green as follows:

red	green	blue
1	0	0
0	1	1
0	0	1

Note that if the boolean value of the feature is 1, it must be 0 for all the other features

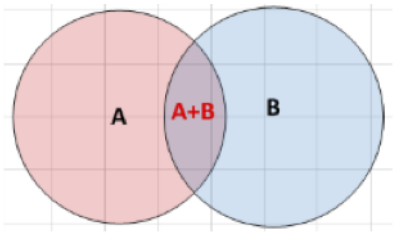
CHAPTER 4

NAIVE BAYES

Naive Bayes is a supervised machine learning algorithm.

4.1 Conditional Probability

Consider two *independent* events A and B. A *Venn diagram* representation is given below:



$$\mathbf{P(A|B) = \frac{P(A \cap B)}{P(B)}}$$

We can write:

$$P(A|B) \times P(B) = P(B|A) \times P(A)$$

$$\boxed{P(A|B) = P(B|A) \times \frac{P(A)}{P(B)}} \tag{4.1.1}$$

$$\boxed{P(B|A) = P(A|B) \times \frac{P(B)}{P(A)}}$$

Consider the following example.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$P(Play = Yes) = 9/14$$
$$P(Play = No) = 5/14$$

$$\begin{aligned} P(Outlook = Sunny \mid Play = Yes) &= 2/9 \\ P(Temperature = Cool \mid Play = Yes) &= 3/9 \\ P(Humidity = High \mid Play = Yes) &= 3/9 \\ P(Wind = Strong \mid Play = Yes) &= 3/9 \end{aligned}$$

$$\begin{aligned} P(Outlook = Sunny \mid Play = No) &= 3/5 \\ P(Temperature = Cool \mid Play = No) &= 1/5 \\ P(Humidity = High \mid Play = No) &= 4/5 \\ P(Wind = Strong \mid Play = No) &= 3/5 \end{aligned}$$

Given, X = (Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong), do we play? Yes or No?

$$\begin{aligned} P(Yes \mid X) = &\quad P(Sunny \mid Yes) \times P(Cool \mid Yes) \times P(High \mid Yes) \times P(Strong \mid Yes) \times P(Play = Yes) &= 0.0053 \\ P(No \mid x) = &\quad P(Sunny \mid No) \times P(Cool \mid No) \times P(High \mid No) \times P(Strong \mid No) \times P(Play = No) &= 0.0206 \end{aligned}$$

Since $P(Yes \mid X) < P(No \mid X)$, we label X to be No.

CHAPTER 5

DECISION TREE

Decision Tree is a supervised machine learning algorithm. In decision trees, the goal is to tidy the data. You try to separate your data and group the samples together in the classes they belong to. You know their label since you construct the trees from the training set. You maximize the purity of the groups as much as possible each time you create a new node of the tree (meaning you cut your set in two). Of course, at the end of the tree, you want to have a clear answer.

At each step, each branching, you want to decrease the entropy, so this quantity is computed before the cut and after the cut. If it decreases, the split is validated and we can proceed to the next step, otherwise, we must try to split with another feature or stop this branch.

Before and after the decision, the sets are different and have different sizes. Still, entropy can be compared between these sets, using a weighted sum, as we will see in the next section.

5.1 Information & Uncertainty

In Clause Shannon's information theory, one bit of information reduces the uncertainty by 2. Similarly, if 3 bits of information are sent, then the reduction in uncertainly by 2^3, i.e., 8. This is intuitive. With 3 bits, there could be 8 possible values and so if a particular set of bits are transmitted, 8 possibilities are eliminated with 1 certainty.

5.1.1 Information Content

When the information is probabilistic, the self-information I_x, or Information Content of *measuring a random variable X as outcome x is defined as:*

$$I_x = log\left(\frac{1}{p(x)}\right) = -log(p(x)) \tag{5.1.1}$$

where $p(x)$ is probability mass function.

5.2 Shannon Entropy

The Shannon Entropy of the random variable X is defined as:

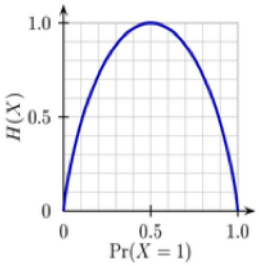
$$H(X) = \sum_x -p(x)log(p(x)) = E(I_x) \tag{5.2.1}$$

It is the *expected information content* of the measurement of X.

Suppose S has 25 examples, 15 positive and 10 negatives [15+, 10-]. Then the entropy of S relative to this classification is computed as:

$$E(S) = -(15/25)log_2(15/25) - (10/25)log_2(10/25) = 0.67$$

In a 2 class system, The entropy is 0 if the outcome is *certain*. The entropy is maximum if we have no knowledge of the system, i.e., whemn any outcome is equally possible.



5.3 Cross Entropy - Kullback–Leibler (KL) divegence

Cross-Entropy is defined as:

$$H(p,q) = -\sum_i p(i) log_2 q(i) \tag{5.3.1}$$

where p is the true distribution and q is the predicted distribution. If the predictions are perfect, then the cross-entropy is same as the entropy. If the prediction differs, then there is a divergence which is known as *Kullback–Leibler (KL) divergence*. Hence,

$$Cross\ Entropy = Entropy + KL\ Divergence$$

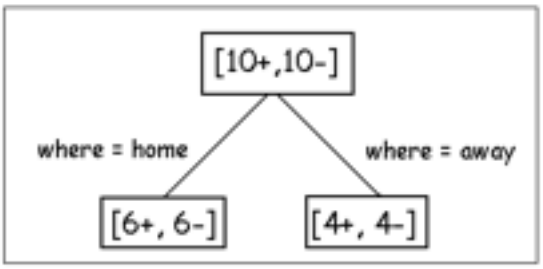
or,

$$KL\ Divergence = H(p,q) - H(p)$$

Hence, if the predicted distribution is closer to true distribution when KL divergence is low.

5.4 Minimizing Entropy

Consider the following table for win/loss of soccer games at home and away.



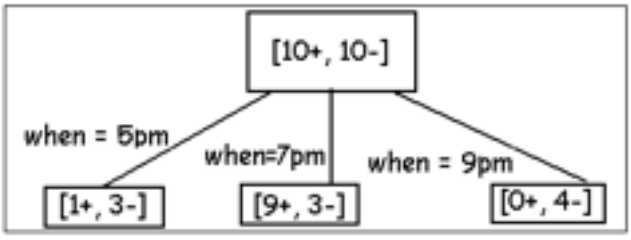
The entropy is:

$$H\left(\frac{6}{12}, \frac{6}{12}\right) = -\frac{6}{12} log_2\left(\frac{6}{12}\right) - \frac{6}{12} log_2\left(\frac{6}{12}\right) = 0.69$$

$$H\left(\frac{4}{8}, \frac{4}{8}\right) = -\frac{4}{8} log_2\left(\frac{4}{8}\right) - \frac{4}{8} log_2\left(\frac{4}{8}\right) = 0.69$$

$$H = -\frac{12}{20} \times H\left(\frac{6}{12}, \frac{6}{12}\right) - \frac{8}{20} H\left(\frac{4}{8}, \frac{4}{8}\right) = 0.69$$

Partitioning by when, we have:



$$H(5pm) = H\left(\frac{1}{4}, \frac{3}{4}\right) = -\frac{1}{4}log_2\left(\frac{1}{4}\right) - \frac{3}{4}log_2\left(\frac{3}{4}\right) = 0.56$$

$$H(7pm) = H\left(\frac{9}{12}, \frac{3}{12}\right) = -\frac{9}{12}log_2\left(\frac{9}{12}\right) - \frac{3}{12}log_2\left(\frac{3}{12}\right) = 0.56$$

$$H(9pm) = H\left(\frac{0}{4}, \frac{4}{4}\right) = -\frac{0}{4}log_2\left(\frac{0}{4}\right) - \frac{4}{4}log_2\left(\frac{4}{4}\right) = 0.00$$

Entropy after partition is:

$$H = -\frac{4}{20}H\left(\frac{1}{4}, \frac{3}{4}\right) - \frac{12}{20}H\left(\frac{12}{20}, \frac{3}{4}\right) - \frac{4}{20}H\left(\frac{4}{20}, \frac{3}{4}\right) = 0.45$$

Hence, the Information gain is 0.69 - 0.45 = 0.24.

We can apply the same approach for other columns to find the most dominant factor on decision. This way we construct the decision tree that has the least entropy. Decision trees can generate understandable rules and classify without much computation. It can handle both continuous and categorical variables. Also, it provides a clear indication of which fields are most important for prediction or classification. However, it is not suitable for prediction of continuous attributes. It performs poorly with many class and small data. It is computationally expensive to train.

5.5 Random Forest

Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. The fundamental idea behind a random forest is to combine many decision trees into a single model. Individually, predictions made by decision trees (or humans) may not be accurate, but combined together, the predictions will be closer to the mark on average.

Why the name 'random forest?' Well, much as people might rely on different sources to make a prediction, each decision tree in the forest considers a random subset of features when forming questions and only has access to a random set of the training data points. This increases diversity in the forest leading to more robust overall predictions and the name 'random forest.' When it comes time to make a prediction, the random forest takes an average of all the individual decision tree estimates. (This is the case for a regression task, such as our problem where we are predicting a continuous value of temperature. The other class of problems is known as classification, where the targets are a discrete class label such as cloudy or sunny. In that case, the random forest will take a majority vote for the predicted class).

CHAPTER 6

CLUSTERING

Clustering is an unsupervised machine learning algorithm.

6.1 Algorithm

1. Create the first K initial clusters from the dataset by choosing K rows of data randomly from the dataset.
2. Each record is assigned to the nearest cluster (the cluster which it is most similar to) using a measure of distance or similarity.
3. Re-calculate the arithmetic mean for each cluster.
4. Repeat step 2.
5. Stop when arithmetic mean for the clusters stabilize.

CHAPTER 7

PRINCIPAL COMPONENT ANALYSIS

A set of Attributes X_1 through X_M where M is the number of attributes under consideration. For each of the attributes, there are N observations available.

7.1 Observation Matrix

The observation Matrix M consists of observation vectors which is represented as:

$$X_1 = \begin{bmatrix} x_{11} \\ x_{21} \\ \dots \\ x_{n1} \end{bmatrix}$$

7.2 Correlation Matrix

$$\begin{bmatrix} \rho_{11} & \rho_{11} & \cdots & \rho_{1M} \\ \rho_{21} & \rho_{22} & \cdots & \rho_{2M} \\ \vdots & \vdots & \dots & \vdots \\ \rho_{M1} & \rho_{M1} & \cdots & \rho_{MM} \end{bmatrix}$$

where,

$$\mu_j = \frac{\sum_{1=1}^{N} x_{ij}}{N}$$

$$\sigma_j = \sqrt{\frac{\sum_{1=1}^{N} (x_{ij} - \mu_j)^2}{N-1}}$$

$$\rho_{pq} = \frac{\sum_{1=1}^{N} (x_{rp} - \mu_p)(x_{rq} - \mu_q)}{\sigma_p \sigma_q}$$

Compute the Eigenvalues of the Correlation Matrix as:

$$|(\rho - \lambda I)| = 0$$

Eigenvectors are computed from:

$$\rho E = \lambda E$$

resulting in ordered eigenvalues:

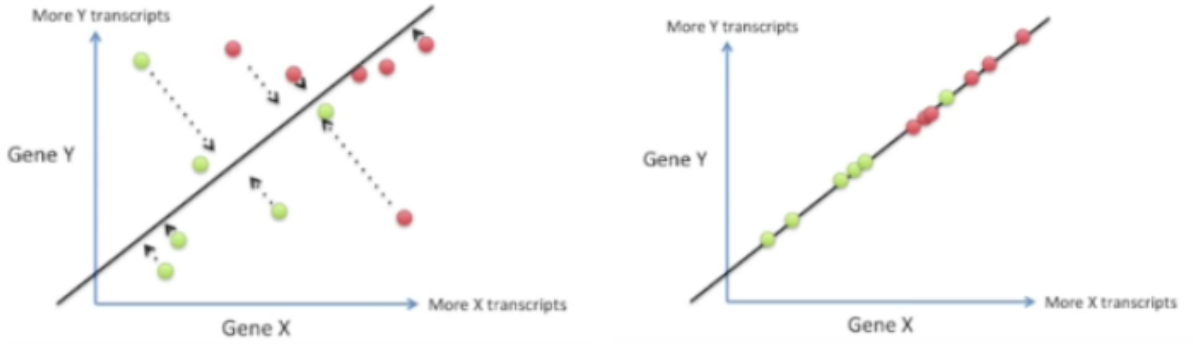
$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \cdots \geq \lambda_k$$

From these Eigenvectors it is now possible to determine the percentage contributions of the various attributes.

CHAPTER 8

LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant Analysis is unsupervised machine learning algorithm that works when data is normally distributed, i.e., satisfies Gaussian distribution. In this method, we find a line on which the data is projected such that "separation" between means is maximum.



Our goal is to maximimize the distance between means and minimise the variation which is know as "scatter". Fisher linear function $w^T x$ that maximizes:

$$J(w) = \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2} \tag{8.0.1}$$

where μ is mean and s is standard deviation.

To maximize, we set the derivative to zero.

$$\frac{dJ(w)}{dw} = 0 \tag{8.0.2}$$

$$S_W^{-1} S_B w = Jw \tag{8.0.3}$$

where,

$$S_W = S_1 + S_2 \tag{8.0.4}$$

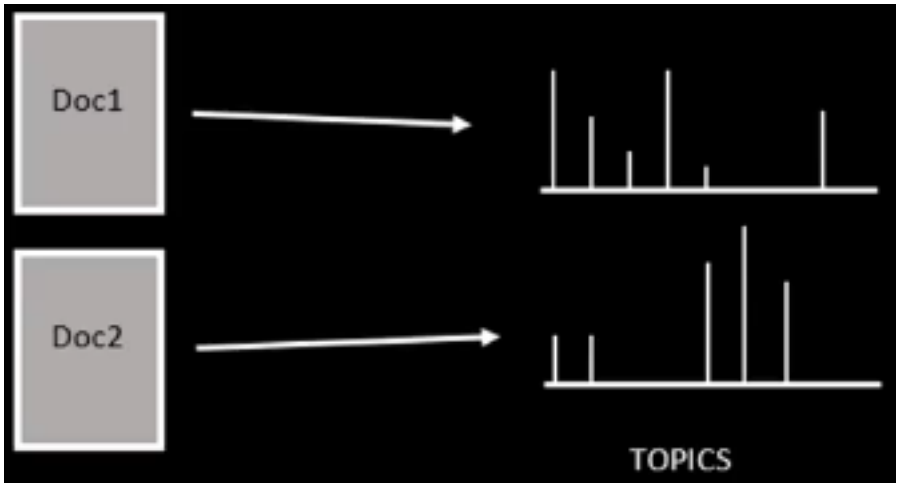
$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \tag{8.0.5}$$

Solve the Eigenvalue problem and compute Eigenvector w.

CHAPTER 9

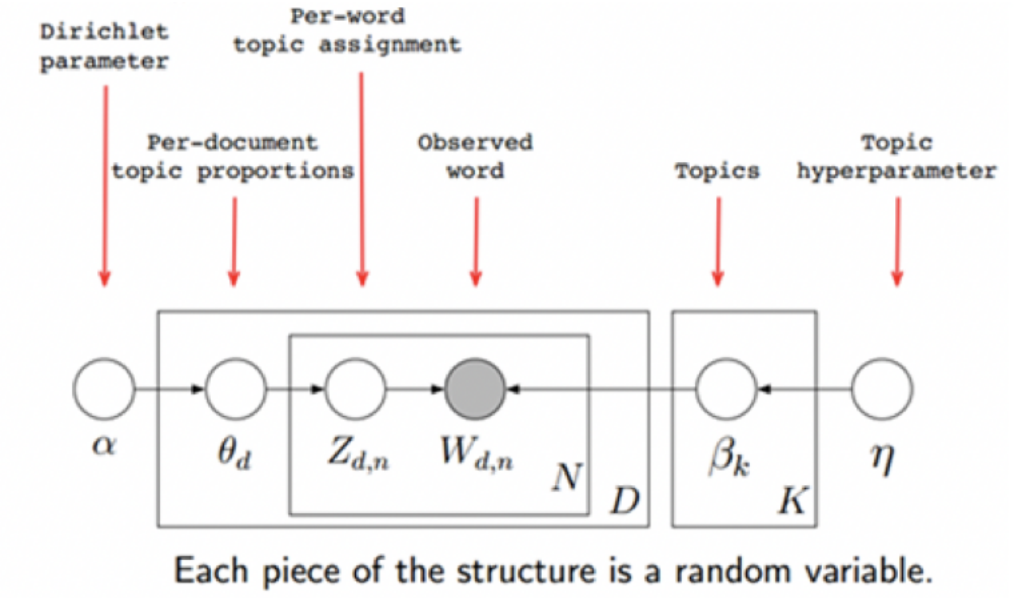
LATENT DIRICHLET ALLOCATION

LDA is an unsupervised machine learning algorithm. LDA uses a generative process that treats a document as bag of words and learns the topics in the document. The words are observable and the topics in the document are hidden and hence the term latent. The Latent part of LDA comes into play because in statistics, a variable we have to infer rather than directly observe is a "latent variable". We are directly observing the words and not the topics, so the topics themselves are latent variables (along with the distributions themselves). Documents are probability distribution over latent topics. and topics are probability distribution over words.



9.1 Plate Notation

The plate notation is a concise way of visually representing the dependencies among the model parameters. α is the parameter in the Dirichlet prior on the per document distribution. A high α implies that a document is likely to contain most of the topics. β is the parameter in the Dirichlet prior on the per topic word distribution. A high β implies that a topic is likely to contain most of the words.



where,
w represents a word
W represents a document (a vector of words); $w = w_1, w_2, \cdots w_N$
α is the parameter of the Dirichlet distribution; $\alpha = (\alpha_1, \alpha_2, \cdots \alpha_n)$
θ is per document topic proportions
Z is a vector of topics, where if the i th element of z is 1 then w draws from the i th topic
D is the total set of documents

LDA assumes that new documents are generated by choosing a topic mixture for the document, (eg., 20% topic-1, 30%, topic-2 and 50% topic-3)) and generate words by first pick a topic based on the documnent's multinomial distribution and then pick a word based on the topics topic's multinomial distribution.

Following are the steps:

1. The first step is to decide that there are k topics.
2. Next, from a given document, assign each word to of the k topics randomly.
3. Assume that all topic assignments except for the current word are correct. Calculate two ratios.
   a. Proportion of words in the document d that are currently assigned to topic t.
   b. Proportion of assignments to topic t over all documents that come from this word.
   c. Assign the current word to the topic for which the probability p is maximum.

We continue repeating this process of re-assigning words until we reach a steady state.

Mathematically, the LDA is expressed as follows:

$$\boxed{p(\theta, z|w, \alpha, \beta) \propto p(\theta, z, w|\alpha, \beta) = p(w|z, \beta)p(z|\theta)p(\theta|\alpha)} \tag{9.1.1}$$

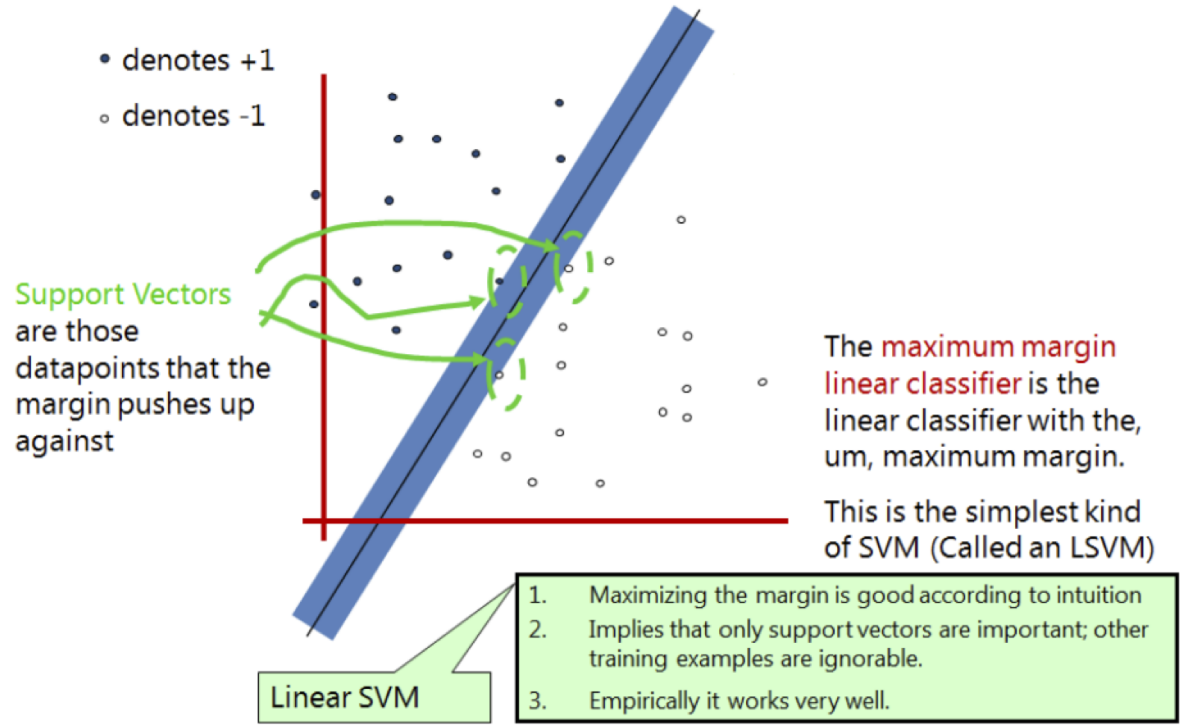
Now, given a new document, how do we determine what topics does it consist of? We can score potential topics using the cross-entropy, i.e., KL distance.

CHAPTER 10

SUPPORT VECTOR MACHINES

Support Vector Machines (SVM) is a supervised machine learning algorithm. SVM performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. To understand this, consider the following 2 dimensional data points.

10.1 Algorithm



The equation of a plane is given by:

$$w^T x + b = 0 \tag{10.1.1}$$

For two points x' and x'' on the plane, we have:

$$w^T x^{'} + b = 0$$
$$w^T x^{''} + b = 0$$
$$w^T (x^{'} - x^{''}) = 0$$

The normal distance d of a point x_n from x is given by:

$$d = \|\hat{w}(x^n - x)\| = \frac{1}{\|w\|}(w^T x^n + b - w^T x - b) = \frac{2}{\|w\|} \tag{10.1.2}$$

Our goal is to maximize the distance which is the same as minimizing $\frac{1}{2}w^T w$ subject to the condition:

$$wx_i + b \geq 1 \text{ for } y_i = +1 \tag{10.1.3}$$
$$wx_i + b \leq 1 \text{ for } y_i = -1 \tag{10.1.4}$$

i.e., for all i:

$$y_i(wx_i + b) \geq 1 \tag{10.1.5}$$

We will now use the Lagrange formulation to minimize:

$$\boxed{\mathcal{L}(w, b, \alpha) = \frac{1}{2}w^T w - \sum^{N} \alpha_n(y_n(w^T x_n + b) - 1)} \tag{10.1.6}$$

with respect to w and b and maximize with respect to each $\alpha_n > 0$

$$\nabla\mathcal{L} = w - \sum_{n=1}^{N} \alpha_n y_n x_n = 0 \tag{10.1.7}$$
$$\frac{\partial\mathcal{L}}{\partial b} = -\sum_{n=1}^{N} \alpha_n y_n = 0 \tag{10.1.8}$$
$$w = \sum_{n=1}^{N} \alpha_n y_n x_n \tag{10.1.9}$$
$$\boxed{\sum_{n=1}^{N} \alpha_n y_n = 0} \tag{10.1.10}$$
$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} - \sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m \tag{10.1.11}$$

maximize the above equation with respect to α subject to $\alpha_x > 0 for n = 1, 2, \cdots N$.

$$max\ \alpha\ \sum_{n=1}^{N} \alpha_n - \frac{1}{2} - \sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m \tag{10.1.12}$$
$$min\ \alpha\ \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m - \sum_{n=1}^{N} \alpha_n \tag{10.1.13}$$
$$min\ \alpha\ \frac{1}{2}\alpha^T] \begin{vmatrix} y_1 y_1 x_1^T x_1 & y_1 y_2 x_1^T x_2 & \cdots & y_1 y_N x_1^T x_N \\ y_2 y_1 x_2^T x_1 & y_2 y_2 x_2^T x_2 & \cdots & y_2 y_N x_2^T x_N \\ \cdots & \cdots & \cdots & \cdots \\ y_N y_1 x_N^T x_1 & y_N y_2 x_N^T x_2 & \cdots & y_N y_N x_N^T x_N \end{vmatrix} \alpha + (-1^T)\alpha \tag{10.1.14}$$

subject to $y^T \alpha = 0; 0 < \alpha \leq \infty$.

$$min\ \alpha\ \boxed{\frac{1}{2}\alpha^T Q\alpha - 1^T \alpha} \tag{10.1.15}$$

subject to Karush Kuhn Tucker (KKT) condition: $y^T \alpha = 0; \alpha \geq 0$. We then compute w:

$$\boxed{w = \sum_{n=1}^{N} \alpha_n y_n x_n} \tag{10.1.16}$$

and solve for b:

$$\boxed{y_n(wx_n + b) = 1} \tag{10.1.17}$$

For non-linear systems, we transform to z space:

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} - \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m z_n^T z_m \tag{10.1.18}$$

and we will have support vectors in the transformed space.

CHAPTER 11

DEEP LEARNING

*The issue of representation lies at the heart of the debate between the logic-inspired and the neural-network-inspired paradigms for cognition. In the logic-inspired paradigm, an instance of a symbol is something for which the only property is that it is either identical or non-identical to other symbol instances. It has no internal structure that is relevant to its use; and to reason with symbols, they must be bound to the variables in judiciously chosen rules of inference**. By contrast, *neural networks just use big activity vectors, big weight matrices and scalar non-linearities to perform the type of fast 'intuitive' inference that underpins effortless commonsense reasoning." - Review Paper, Nature, 28-May-2015 | Vol 521 | 437, Deep learning, Yann LeCun, Yoshua Bengio & Geoffrey Hinton.*

Frank Rosenblatt - *Perceptron: a machine which senses, recognizes, remembers, and responds like the human mind"*. The system was capable of categorizing some basic shapes. But in the critical book written in 1969 by Marvin Minsky and Seymour Papert showed that Rosenblatt's original system was painfully limited, literally blind to some simple logical functions like XOR. In the book they said: *...our intuitive judgment that the extension (to multilayer systems) is sterile*. With that, the field of "Neural Networks" pretty much disappeared.

Geoff Hinton built more complex networks of virtual neurons that allowed a new generation of networks to learn more complicated functions (like the exclusive-or that had bedeviled the original Perceptron). Even the new models had serious problems though. They learned slowly and inefficiently and couldn't master even some of the basic things that children do. By the late 1990s, neural networks had again begun to fall out of favor.

In 2006, Hinton developed a new technique that he dubbed deep learning, which extends earlier important work by Yann LeCun. Deep learning's important innovation is to have models learn categories incrementally, attempting to nail down lower-level categories (like letters) before attempting to acquire higher-level categories (like words).

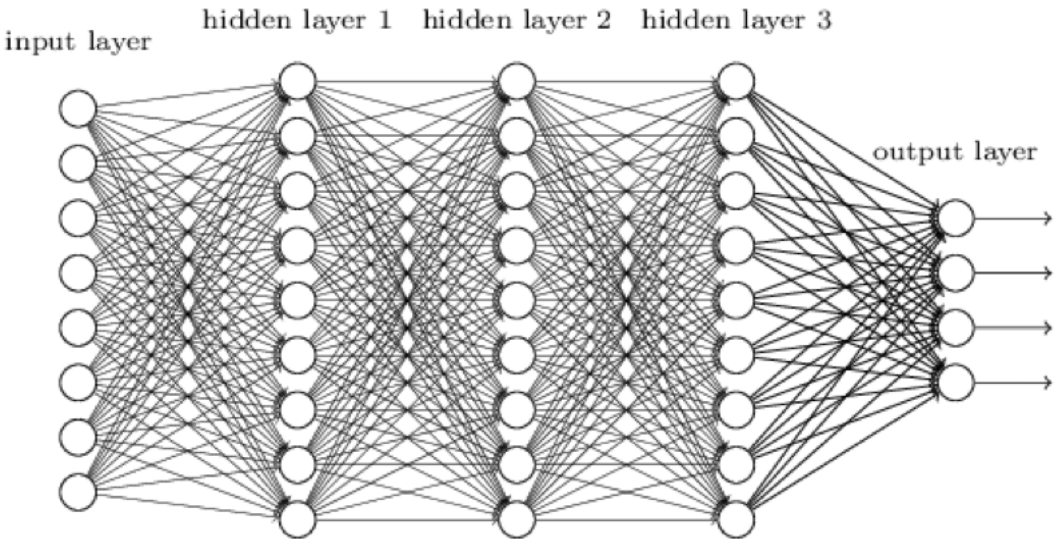
11.1 DEEP NEURAL NETWORKS

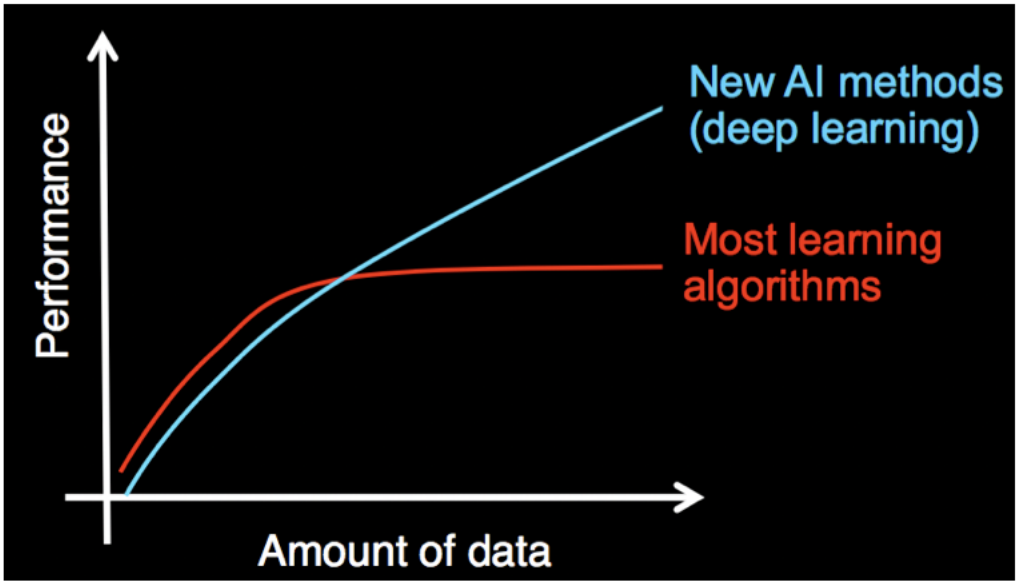
DNNs are suitable where the raw underlying features are not individually interpretable. This success is attributed to their ability to learn hierarchical representations, unlike traditional methods that rely upon hand-engineered features.

DNNs are greedily built by stacking restricted Boltzmann machines, and Convolutional Neural Networks, which exploit the local dependency of visual information, and have demonstrated record-setting results on many important applications.

Neural networks are powerful learning models that achieve state-of-the-art results in a wide range of supervised and unsupervised machine learning tasks for classification & regression.

Simple linear models tend to under-fit, and often under utilize computing resources with 8-GPU Hydra.





RNNs have roots in both cognitive modeling and supervised machine learning. Owing to two different perspectives, many published papers have divergent aims and priorities. In many foundational papers, generally published in cognitive science and computational neuroscience journals, such as [Hopfield, 1982, Jordan, 1986, Elman, 1990], biologically plausible mechanisms are emphasized. In other papers [Schuster and Paliwal, 1997, Socher et al., 2014, Karpathy and Fei-Fei, 2014], biological inspiration is downplayed in favor of achieving empirical results on important tasks and datasets.

Since the earliest conception of artificial intelligence, researchers have sought to build systems that interact with humans in time. In Alan Turing's groundbreaking paper Computing Machinery and Intelligence, he proposes an "imitation game" which judges a machine's intelligence by its ability to convincingly engage in dialogue [Turing, 1950].

Dialogue systems, self-driving cars, robotic surgery, speech, etc. among others require an explicit model of sequentiality or time – a combination of classifiers or regressors cannot provide these functionalities.

SVM, logistic regression, feedforward networks have proved very useful without explicitly modeling time. But the assumption of independence precludes modeling long range dependencies.

Frames from video, snippets of audio, and words pulled from sentences – the independence assumption fails.

RNNs are connectionist models with the ability to selectively pass information across sequence steps while processing sequential data one element at a time. They can model input and/or output consisting of sequences of elements that are not independent.

Given any fixed architecture, a set of nodes, edges, and activation functions, an RNN with this architecture are differentiable end to end. The derivative of the loss function can be calculated with respect to each of the weights in the model and are amenable to gradient-based training.

RNNs are Turing complete. But given a fixed-size RNN with a specific architecture, it is not actually possible to reproduce any arbitrary program. Capability of the model to perform any calculation, given a sufficient amount of resources (i.e. infinite), not to show whether a specific implementation of a model does have those resources.

Hidden Markov models (HMMs) model an observed sequence as probabilistically dependent on a sequence of unobserved states were described in the 1950s and have been widely studied since the 1960s [Stratonovich, 1960].

Traditional Markov model approaches are limited because their states must be drawn from a modestly sized discrete state space S. The dynamic programming algorithm that is used to perform efficient inference with hidden Markov models scales in time [Viterbi, 1967]. Further, the transition table capturing the probability of moving between any two time adjacent states is of size. HMM is infeasible when the set of possible hidden states grows large. Further, each hidden state can depend only on the immediately previous state.

Recurrent neural networks can capture long range time dependencies, overcoming the chief limitation of Markov models. A state in an RNN depends only on the current input as well as on the state of the network at the previous time step. The hidden state at any time step can contain information from a nearly arbitrarily long context window.

The number of distinct states that can be represented in a hidden layer of nodes grows exponentially with the number of nodes in the layer. With the value of a node as a real number, a network can represent a large number of distinct states. The complexity of both inference and training grows at most quadratically.

Finite sized RNNs with nonlinear activations are a rich family of models, capable of nearly arbitrary computation. A well-known result is that a finite-sized recurrent neural network with sigmoidal activation functions can simulate a universal Turing machine [Siegelmann and Sontag, 1991]. The capability of RNNs to perform arbitrary computation demonstrates their expressive power.

11.2 Activation Functions

Sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{11.2.1}$$

Tanh:

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{11.2.2}$$

ReLU:

$$f(x) = max(0, x) \tag{11.2.3}$$

For multiclass classification with K alternative classes, use a a softmax nonlinearity in an output layer of K nodes.

With Sigmoidal or Tanh activation functions, the nodes in each layer never take value exactly zero. So, even if the inputs are sparse, the nodes at each hidden layer are not. Rectified linear units (ReLUs) introduce sparsity to hidden layers [Glorot et al., 2011]. Store the sparsity pattern when computing each layer's values and use it to speed up computation of the next layer in the network. Convolution Neural Networks use ReLU.

11.3 Backpropagation

After Input X, each higher layer is then subsequently computed until output is generated at the topmost layer . Learning is accomplished by iteratively updating each of the weights to minimize a loss function, , which penalizes distance between output and the target .

Algorithm for training neural networks is backpropagation Rumelhart et al. [1985]. Uses chain rule to calculate the derivative of the loss function with respect to each parameter in the network. The weights are then adjusted by gradient descent.

Because the loss surface is non-convex, there is no assurance that backpropagation will reach a global minimum. Exact optimization is known to be an NP-hard problem.

However, heuristic pre-training and optimization techniques have led to impressive empirical success on many supervised learning tasks. In particular, convolutional neural networks, popularized by LeCun et al. [1990], are a variant of feedforward neural network, holds records since 2012 in many computer vision tasks such as object detection [Krizhevsky et al., 2012].

A recurrent neural network can be regularized via techniques that help prevent overfitting, such as weight decay, dropout, and limiting degrees of freedom.

11.4 Vanishing / Exploding Gradients

Learning with recurrent networks is challenging due to the difficulty of learning long-range dependencies, [Bengio et al. 1994], [Hochreiter et al, 2001].

Problems of vanishing and exploding gradients occur when backpropagating errors across many time steps.

Which of the two phenomena occurs depends on whether the weight of the recurrent edge $|wjj| > 1$ or $|wjj| < 1$ and on the activation function in the hidden node. For a sigmoid activation function, the vanishing gradient problem is more pressing, but with a rectified linear unit $max(0, x)$, it is easier to imagine the exploding gradient.

11.5 Long Short Term Memory (LSTM)

Hochreiter and Schmidhuber [1997] introduced the LSTM model to overcome vanishing gradients.

Each node in the hidden layer is replaced by a memory cell. Each memory cell contains a node with a self-connected recurrent edge of fixed weight one, ensuring that the gradient can pass across many time steps without vanishing or exploding.

Simple recurrent neural networks have long-term memory in the form of weights. The weights change slowly during training, encoding general knowledge about the data. They also have short-term memory in the form of ephemeral activations, which pass from each node to successive nodes.

The LSTM model introduces an intermediate type of storage via the memory cell. A memory cell is a composite unit, built from simpler nodes in a specific connectivity pattern, with the

novel inclusion of multiplicative nodes, represented in diagrams by the letter . s is a vector containing the value of s**c at each memory cell c in a layer.

Input Node g**c – takes activation from input layer x**(t) at current time step and along recurrent edges from the hidden layer at the previous time step h(t-1). The summed weighted input is run through a Tanh activation function, although in the original LSTM paper, the activation function is a sigmoid.

Input Gate – Gates are a distinctive feature of the LSTM approach. A gate is a sigmoidal unit that, like the input node, takes activation from the current data point x(t) as well as from the hidden layer at the previous time step. A gate is so-called because its value is used to multiply the value of another node – if it is 0, then flow from the other node is cut off, if it is 1, all flow is passes through. The value of the input gate ic multiplies the value of the input node.

Internal State – node sc with linear activation. The internal state sc has a self-connected recurrent edge with fixed unit weight. This edge (constant error carousel) spans adjacent time steps with constant weight, error can flow across time steps without vanishing or exploding.

Forget Gate – Introduced by Gers et al. [2000]. They provide a method by which the network can learn to flush the contents of the internal state. This is especially useful in continuously running networks. With forget gates, the equation to calculate the internal state on the forward pass is:

Output gate: The value vc ultimately produced by a memory cell is the value of the internal state sc multiplied by the value of the output gate oc. It is customary that the internal state first be run through a Tanh activation function, as this gives the output of each cell the same dynamic range as an ordinary tanh hidden unit.

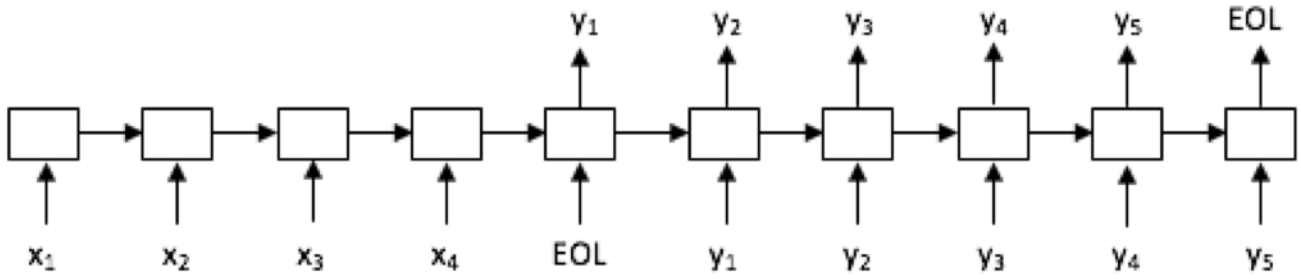
In terms of the forward pass, the LSTM can learn when to let activation into the internal state. As long as the input gate takes value zero, no activation can get in. Similarly, the output gate learns when to let the value out. When both gates are closed, the activation is trapped in the memory cell, neither growing nor shrinking, nor affecting the output at intermediate time steps.

In terms of the backwards pass, the constant error carousel enables the gradient to propagate back across many time steps, neither exploding nor vanishing. In this sense, the gates are learning when to let error in, and when to let it out. In practice, the LSTM has shown a superior ability to learn long range dependencies as compared to simple RNNs.

Output from each memory cell flows in the subsequent time step to the input node and all gates of each memory cell. Weights change slowly during training, encoding general knowledge about the data. Short-term memory in the form of ephemeral activations, which pass from each node to successive nodes.

11.6 Dialog Systems

Each sentence ends with a special end of line symbol <EOL> which enables the model to define a distribution over sequences of all possible lengths. distribution is represented with a Softmax over all words in vocab. [Graves, Sutskever]



$$p(y_1, y_2, \cdots, y_q | x_1, x_2, \cdots, x_p) = \prod_{n=1}^{q} p(y_t | v, y_1, y_2, \cdots, t_{t-1}) \tag{11.6.1}$$

LSTM computes the representation of $x_1, x_2, x_3, x_4, < EOL >$ and then uses this representation to compute the probability of $y_1, y_2, y_3, y_4, y_5, < EOL >$

Learning by Softmax activation, maximize Cross-Entropy, Back Propagation.

Multiple candidates at a given step, beam search to predict sequence based on maximum joint probability.

11.7 Convolution Neural Network (CNN)

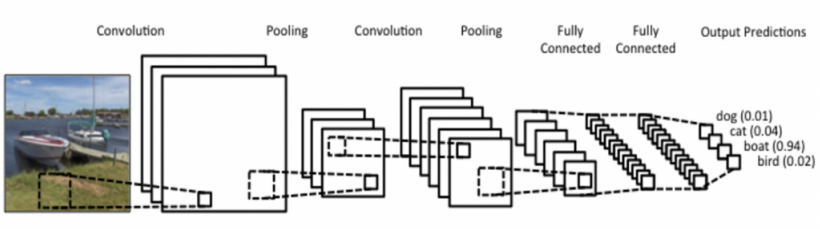
Convolution – think of it as a sliding window function applied to a matrix.

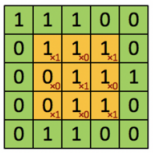
The sliding window is called a kernel, filter, or feature detector. Eg**. Sobel filter.

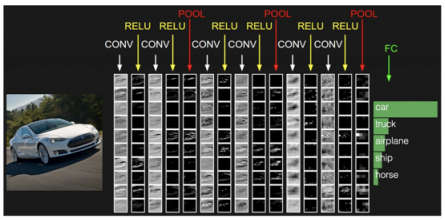
CNNs are several layers of convolutions with nonlinear activation functions like ReLU or tanh applied to the results.

During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform.

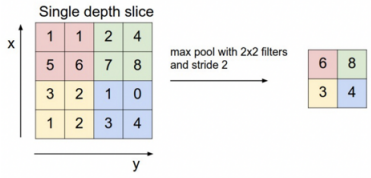
Pooling layers subsample their input. The most common way to do pooling it to apply a operation to the result of each filter. Pooling also reduces the output dimensionality but (hopefully) keeps the most salient information.

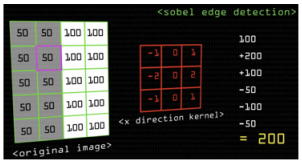




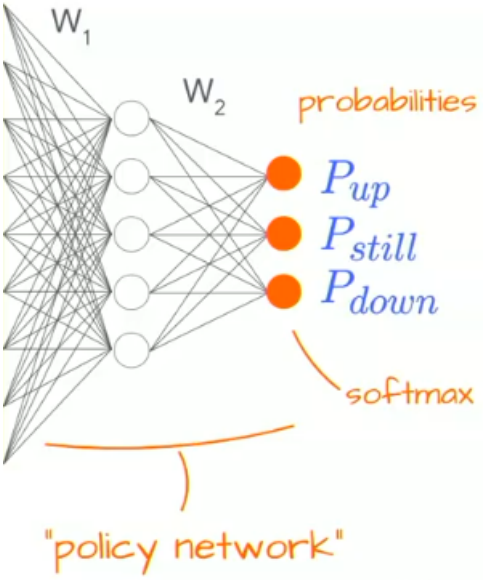


- Rectifier is an activation function
- f(x) = max(0,x), x is input
- A unit employing the rectifier is called a rectifier liner unit (ReLU)





11.8 Policy Engine



cross-entropy

$$loss = -(P'_{up}.log(P_{up}) + P'_{still}.log(P_{still}) + P'_{down}.log(P_{down}))$$

"correct move"
Ex: 1 0 0

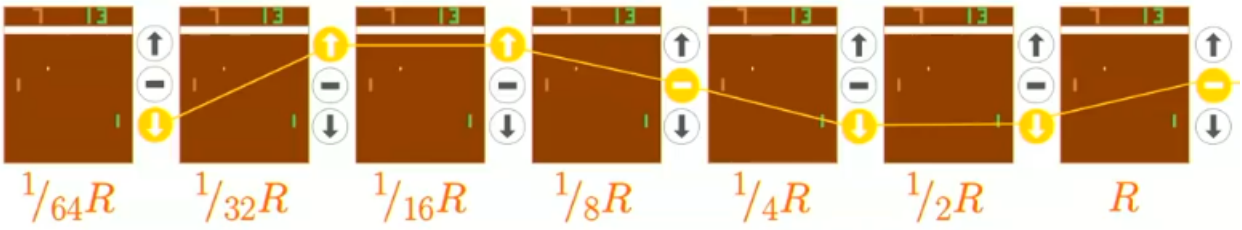
$$CrossEntropy = Loss = -\left(P'_up.log(P_{up} + P'_{still}.log(P_{still}) + P'_{down}.log(P_{down})\right) \tag{11.8.1}$$

$$loss = -R\,(P'_{up}.log(P_{up}) + P'_{still}.log(P_{still}) + P'_{down}.log(P_{down}))$$

for each move

"sampled move"
Ex: 1 0 0

predicted



① discounted rewards

whichever was chosen

$$loss_i = -R_i log(P_i^{up/still/down})$$

② normalized rewards

$$R_i = \frac{R_i - \overline{R}}{stdev(R)}$$

across a "batch" of moves

move #i