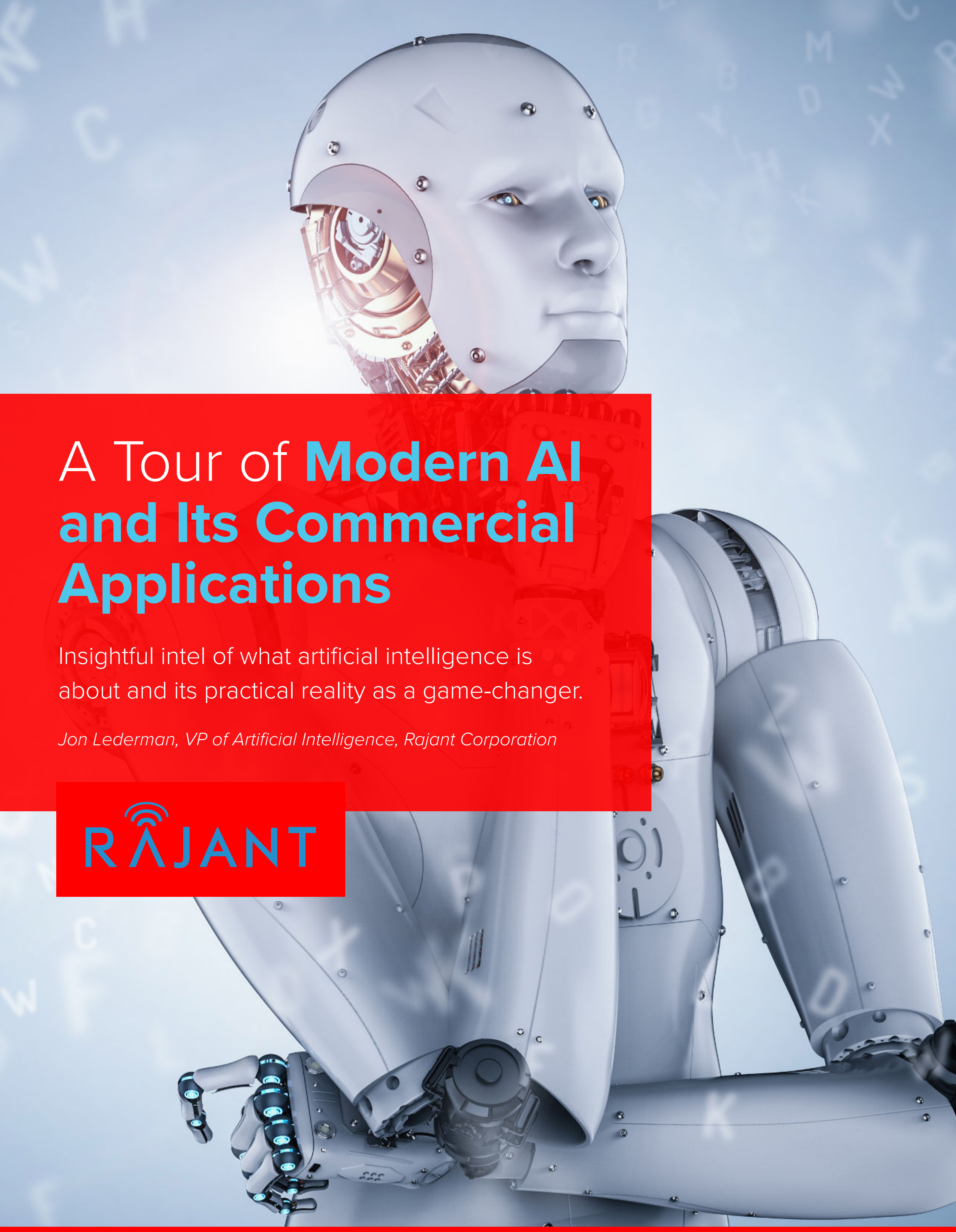


A Tour of Modern AI and Its Commercial Applications

Insightful intel of what artificial intelligence is about and its practical reality as a game-changer.

Jon Lederman, VP of Artificial Intelligence, Rajant Corporation

RAJANT

Table of Contents

Introduction

***Artificial Intelligence (AI) is making huge headlines around the world.** But, what is the practical reality of AI as a game-changer in various industries for solving real-world problems? Before tackling these questions, it's useful to gain some insight into what AI is all about.*

This white paper will begin by embarking on a tour of contemporary AI and how it is accomplished. Although AI and machine learning (ML) can be a highly technical subject involving lots of math and statistics, the aim here is to present these topics in a non-technical way, but at the same time provide a reasonable amount of breadth, depth, and rigor. While there will be some mathematical notation, this can be safely ignored if so desired.[1]

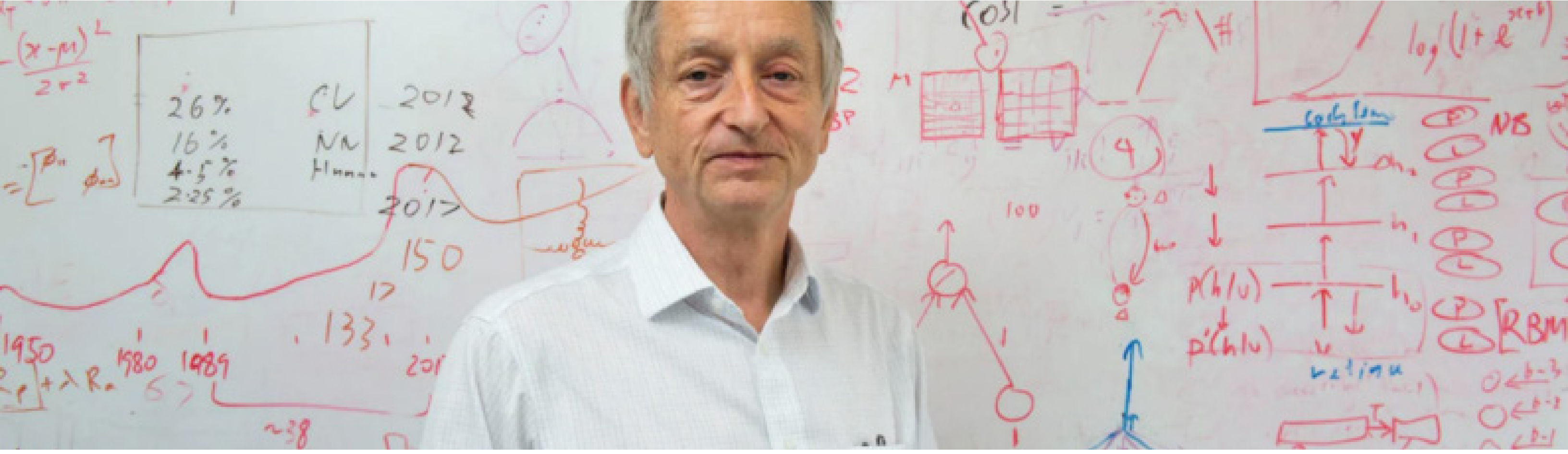
This first installment will present an overview of some of the key ideas of AI and ML at a high level. We will move rapidly through key aspects of the theory and by the end of this installment walk through building a simple deep neural network to recognize handwritten digits. Don't worry if some of the ideas presented here are not entirely clear, as we won't have time to delve into all of the details. In subsequent installments, we will flesh out the key ideas presented here in greater detail. Upon establishing a solid foundation of the mechanics of AI and machine learning, later installments will examine the promise of AI and specifically how the various incarnations of AI can be brought to bear in industry and commercial applications to solve hard problems in the world. More detailed aspects of the content will be presented later in this series of white papers on artificial intelligence.

As we will see, the rudiments of AI and ML arise from probability and statistics. And while there are many ways to interpret how AI works, in fact, the "how" remains an open question that has become a focal area of research. Answers to this question may touch on some advanced topics such as convex function theory, Bayesian inference, information theory, and even aspects of statistical quantum mechanics such as spin glasses. These questions are not merely academic, for they underscore crucial elements of the successful practice of AI. Indeed, it is a vibrant field! But we needn't complicate the discussion for now as we will get to all of that in due course.

We will approach this discussion by building up our understanding of AI from first principles. We will begin with linear regression, which may be thought of as the "Hello World" of AI. From there, we will turn to the classification problem and delve into an area called logistic regression, which is all about teaching a computer to classify an input as belonging to one class or another. If you think about it, classification is a vital part of the foundations of exhibited intelligence. An agent that can recognize and distinguish different entities in the world may be said to present some basic modicum of intelligence.

Logistic regression will serve as a foundation for understanding neural networks and deep learning, which is one of the most popular techniques today in supervised learning. Along the way, we hope to highlight the essential aspects of these techniques. For example, we will learn the two most important aspects of neural networks and deep learning that has enabled this technique to be so successful in areas such as image classification, natural language processing, speech recognition, and speech synthesis.

[1] The key areas of math that are helpful to understand AI are in no particular order, probability and statistics, linear algebra and basic univariate and multivariate calculus. However, reasonable intuition and understanding of AI concepts and the practice of building AI models is certainly achievable without much or any math. From a practitioner's perspective, linear algebra is perhaps the most important area to have some familiarity with in order to make headway.



A Brief Digression on the History of Artificial Intelligence

It's worth taking a brief detour into the history of AI as it is not only interesting but informative as to why modern AI techniques have enjoyed such success. Like many technical fields, it is riddled with dead ends, loose threads, and a trajectory characterized more like an aimless sailing excursion in shifting winds than a rocket launch.

One myth that underlies much of the popular discussion of AI on the Web is that it's new. It's not—at least the basic theoretical knowledge for its execution isn't. The success of AI as of late is a byproduct of decades-old theoretical results and modern cheap computer hardware that has emerged over the last ten years or so. The roots of machine learning and AI can be traced back at least to the 1940s, and its evolution, like many fields, is characterized by a pattern of fits and starts.

Perhaps the most realistic time frame to consider in tracking the birth of modern AI is the late 1950s and early 1960s and the discovery of the perceptron by Rosenblatt. The perceptron is a learning model, which bears some resemblance to artificial neurons at the heart of deep neural networks today—with some key differences. However, progress in the field was stymied in the late '60s largely due to some theoretical results published by Minksy (MIT) et al. that showed certain fundamental limitations of perceptrons. The Minksy work was misinterpreted as showing that neural networks were limited to linearly separable problems, which unfortunately killed much of the research in the area until the mid-1980s.

The theory of backpropagation, which arguably is the bread and butter of deep neural network training and which we shall explore in great detail later, was derived in the 1960s and implemented to run on a digital computer in 1970 (by Linnainmaa) albeit not for neural network training. In 1974, the idea of applying backpropagation to neural networks was proposed by Paul Werbos in his doctoral dissertation.

The field was resurrected to some extent in the mid-1980s when Geoffrey Hinton (a "Godfather of AI", pictured above) et al. rediscovered backpropagation and, through experimentation, showed that it could be applied to deep neural networks and utilized to automatically generate internal representations of features for learning. However, techniques for carrying out training were limited by the computational horsepower of the day. This remained largely true up through the 2000s.

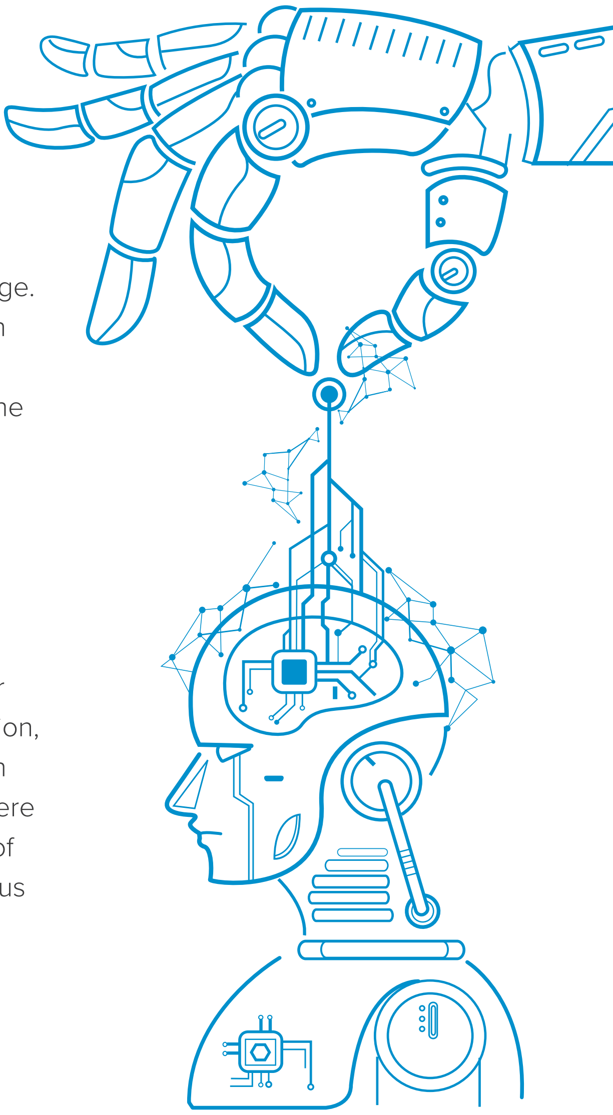
However, if backpropagation provided the kindling, the emergence of affordable and more powerful GPUs in recent years served as the spark for many of the modern advances we've seen in AI and deep learning, especially over the last ten years. Arguably, it was this explosive combination of the theoretical underpinnings of backpropagation and the access to fast parallel computation via GPUs that set the AI field alight.

But, there's more than this. To foreshadow a bit the story of why modern AI has proved so successful, there are two salient features at play. First, rather than relying on hand-engineered features, modern AI and deep learning models learn their features. This, as we shall explore later, is the province of representational learning, a subtle but brilliant kernel of what makes AI tick. Second, deep neural networks are inherently non-linear, but more importantly, can learn non-linear representations without facing the intractable limitations inherent in working with analytic representations of non-linear mathematics. We shall see that these two attributes—representational learning and non-linearity—are the driving forces in the remarkable power and success for which recent AI initiatives have garnered attention.

Before delving into the details, a bit of nomenclature is in order.

A Taxonomy of AI

The terms AI, machine learning, deep learning, neural networks, and many others are weaving their way into public consciousness. Often these terms are used interchangeably. But, there are meaningful differences, which are essential in understanding the potential for commercial AI applications.



Intelligence is in the Eye of the Beholder

The term "artificial intelligence" was coined by computer scientist John McCarthy in 1956 who also developed the famous LISP programming language. Although there is no universal definition of the term, a reasonable description is any machine behavior that appears intelligent or displays what humans would call intelligence. To this end, Alan Turing, subject of the famous film "The Imitation Game", devised a test called the Turing test to determine whether a machine exhibits intelligent behavior.

Machine Learning

The term "machine learning" was coined in 1959 by computer scientist Arthur Samuel. While "machine learning" does not have a precise formalized definition, a working definition is algorithms that perform some type of pattern detection typically accomplished using statistical techniques or statistical inference. There is a divergence of opinion on whether "machine learning" itself is a subfield of AI, and the history of both fields has taken many twists and turns and at various points have intertwined.[2]

But, where is the "learning" in ML? In short, learning refers to a machine's ability to refine or improve its ability to perform a specific action. Learning may be further classified as supervised learning, unsupervised learning and reinforcement learning. Let's explore these in turn.

[2] In general, the AI camp has historically focused more on symbolic vs. statistical techniques. Indeed AI in its purest form focused on symbolic, knowledge representation and logical inference and machine learning is predominantly concerned with pattern recognition primarily using statistical approaches. Indeed, machine learning might be aptly called statistical inference.

Supervised Learning

Human beings often learn from example. For example, they may learn what the word "red" means by observing those objects in the world that have particular visual characteristics are all called "red". After observing many examples, a person may arrive at a general understanding of what "red" means. In short, there is a consensus on what "red" means for which virtually everyone can agree and which is codified in empirical examples in the world.

Supervised machine learning operates superficially in the same way. During a training phase, an algorithm processes a set of labeled training examples to perform a learning algorithm. Each training example comprises an input and an output and represents what is referred to as the "ground truth". The ground truth operates as an instructive example gleaned empirically and represents a correct mapping from one particular input to an associated output. The product of the training phase is called the model and generally comprises a set of abstract numbers (more on this later), which are called the parameters of the model. Once the training is completed, the model may be used to perform inference or prediction (also called the test phase) on examples that were not observed during training. The hope is that the model is sufficiently general that it can make inferences about examples it has never observed previously.

In any case, for all practical purposes, the terms AI and machine learning these days are used almost interchangeably despite their historical differences.

However, supervised learning operates intrinsically at the statistical level. The set of training examples defines some probability distribution from which a statistical model is learned (more on this later).

So, in short, a supervised learning algorithm analyzes the training data (examples) and generates an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

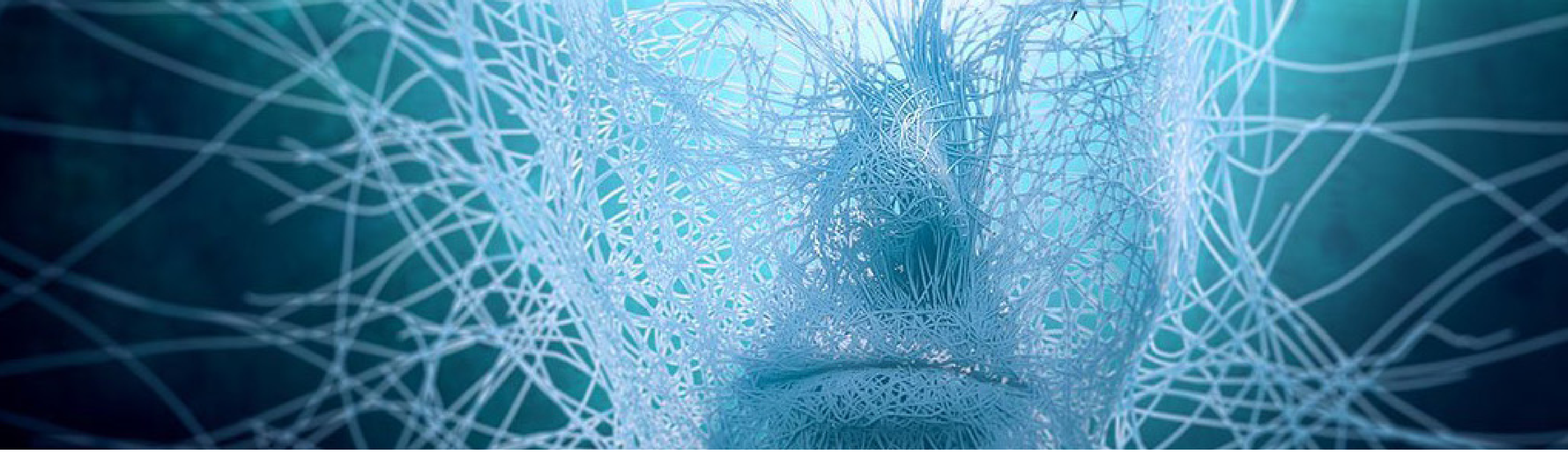
Neural networks and deep learning are two examples of supervised learning techniques that have recently received much attention and fanfare in the media. We will explore neural networks and deep learning in another white paper.

Unsupervised Learning

Although people often learn in a supervised way, in most instances, learning occurs without any reference to specific labels or examples. For example, a person may abstract a general principle that may be applied by analogy to many scenarios. And, this may happen without any specific data indicating such a relationship. In some sense, the brain is wired to learn. Principal component analysis ("PCA") and clustering are examples of unsupervised learning that we will explore later in this series.

[3] The term "test" phase although used heavily in the literature is somewhat misleading. It's really part of the training phase to evaluate how well the model generalizes and in fact comprises. The hope is that the model is sufficiently general that it can reliably perform inference on inputs outside of the training set. We will therefore use the term inference or prediction to distinguish from the training phase.

[4] Whether models can generalize is an involved topic that relates to two types of conditions, underfitting and overfitting. This in turn relates to what is called the bias variance tradeoff. We won't explore these aspects in this white paper, but they are very important and will be examined in great detail later.



Reinforcement Learning

Reinforcement Learning ("RL") and deep reinforcement learning are very hot fields that are making their way into commercial applications. It is particularly applicable to situations where an agent must interact with its environment such as in robotics or autonomous driving. As a fun illustration of RL, it has been applied to teach machines how to play classic arcade video games that can decimate human opponents.

For those who are familiar with the astounding success of AlphaGo and AlphaGoZero, deep reinforcement learning is the machine learning technique behind that technology. AlphaGo and AlphaGoZero have received quite a bit of publicity after defeating top-ranked professional Go players in the world; a feat considered the holy grail for employing AI to strategic gaming.[5]

A hallmark of reinforcement learning that distinguishes it from other types of learning is that it uses training information that evaluates actions taken rather than instructs by giving correct actions. Evaluative feedback depends on the action taken typically in terms of some reward based on the action. Supervised learning, on the other hand, relies on instructive feedback, which is characterized by a correct action to take independent of the action actually taken. In short, evaluative feedback, which is used to train reinforcement learning systems, depends on the action taken. In contrast, instructive feedback, which is used to train supervised learning systems, is independent of the action taken.

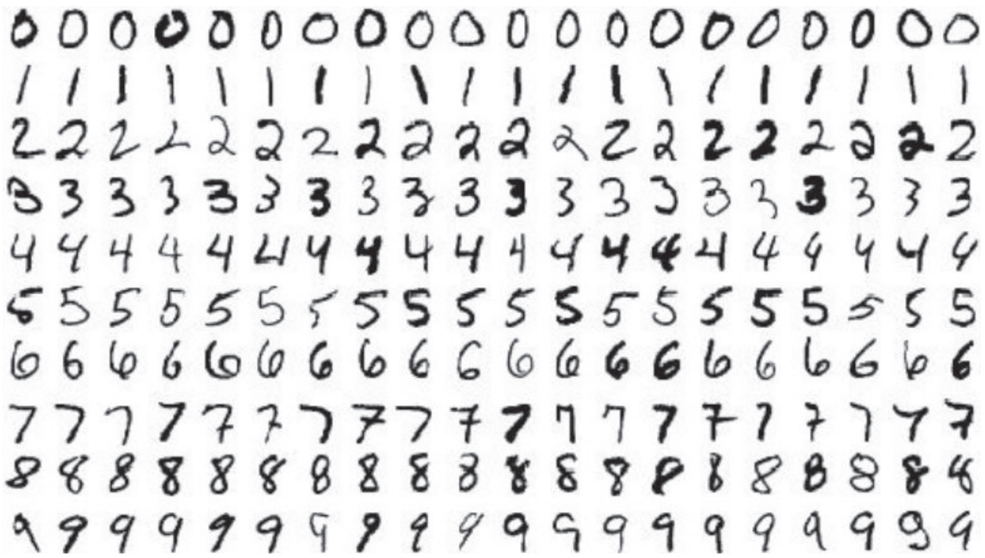
[5] The estimated number of possible board configuration is 10^{120} in chess which is astronomical but Go has around 10^{174}. After the first two moves of a Chess game, there are 400 possible next moves. In Go, there are close to 130,000.

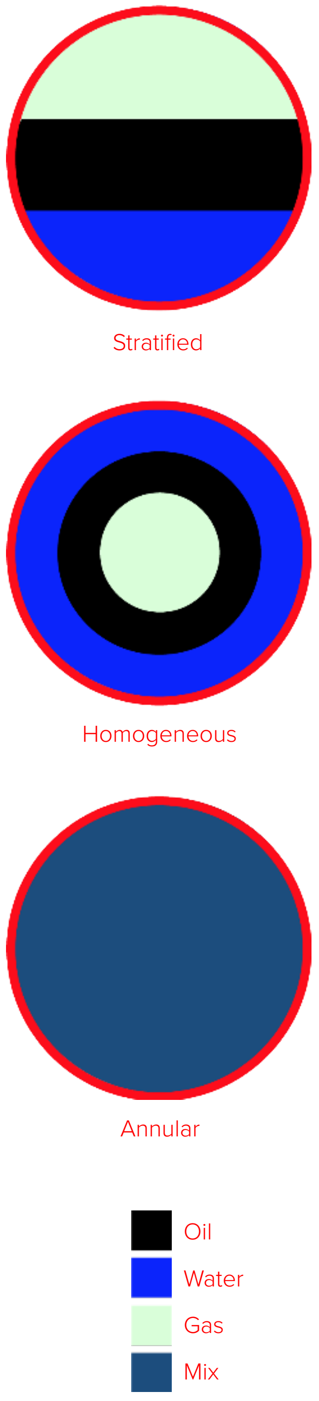
Data...Data...Data

Before diving into our first high-level topic of supervised learning, we will briefly discuss the fuel of ML and AI - data. With machine learning, there is no such thing as too much data. However, no data, too little data, or lack of good data can be a non-starter for any machine learning initiative.

As food for thought, we highlight two toy datasets that will be useful to keep in mind when reading this material. The first by Bishop and James relates to the non-invasive measurement of the proportions of oil, water, and gas in the North Sea oil transfer pipelines.[6]

The second is the MNIST dataset of handwritten digits. This dataset consists of handwritten digits. We seek to train a machine to recognize these handwritten digits. Here is an example drawn from the dataset:





[6] This image was taken from Pattern Recognition and Machine Learning by Christopher M. Bishop (1993).

Linear Regression—the Hello World of Supervised Machine Learning

Anyone who has taken a basic statistics class has already had a taste of the foundations of machine learning in the form of linear regression. In some sense, linear regression can be viewed as the most basic form of supervised learning. Linear regression is a technique for inferring a linear relationship or mapping between two sets of data, an independent variable (input) and a dependent variable (output). The input variables are often described as the features, while the output variable is often called the target.

A good toy problem to which linear regression might be applied is predicting housing prices. Suppose we desire to predict the price (target) of a particular house based on various features, including such things as location, square footage, number of bedrooms, etc. A reasonable model is to assume that the price of the house is a function of some linear combination of the features. In other words, let's assume our target (the housing price) to be modeled as a linear function of three features—location, square footage, and number of bedrooms:

$$\text{price} = w_1 * \text{location} + w_2 * \text{footage} + w_1 * \text{bedrooms}$$

The variables w_1 w_2 and w_3 are called weights. These weights express the relative importance of each of the features in predicting a housing price.

But how does machine learning fit into this picture? In short, our first example of machine learning is learning these weights based upon some observations. Let's take the housing prices example, generalize it a bit, and gain a glimpse of what this learning process looks like. Let's generalize our problem from three features to an arbitrary number of input features. The simplest model for linear[7] regression is simply a linear combination of input features as follows:

$$f(x,w) = w_0 + x_1w_1 + x_2w_2 + \ldots + x_nw_n$$ [8]

The relationship can be expressed compactly in vector form as:

$$f(x) = w^Tx$$

where **w** is a vector of parameters and **x** is a vector of input features[9]. Technically, the parameters are a set of weights and a bias parameter. The meaning of these terms will be explored in subsequent papers.

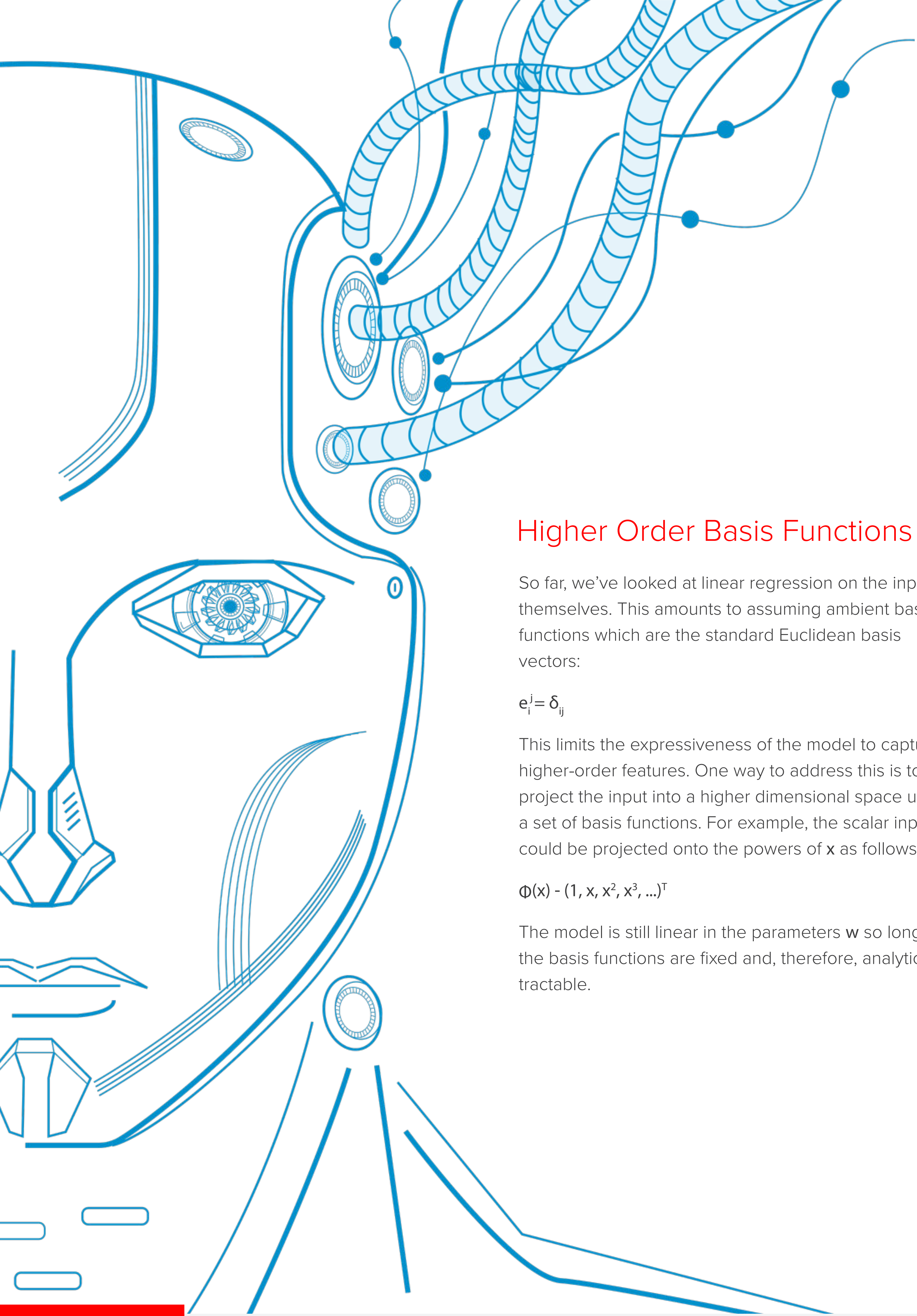
Let's unpack these relationships:

- x is a vector holding the input features to be processed by the linear regression model.
- f (x) is the output of the linear regression model. It is the number we are trying to predict.
- w is a vector of parameters comprising weights and a bias.

[7] The term "linear" here can be intuitively thought of in this context as lying on a line. Of course, in this example, we are in n dimensions and thus the points lie on a hyperplane of dimension N-1 in N dimensions. The term linear though has a more formal mathematical definition that makes linearity very powerful, which we won't discuss here, but perhaps will do so later.

[8] Note a key point is that the model is linear in the parameters **w**. Later, we will find we can model quadratic or higher terms by choosing a set of basis functions. However, these models will still be linear in the parameters.

[9] Note, we can set the input feature x_0 = 1 so that w_0 can be conveniently included in this vector.



Higher Order Basis Functions

So far, we've looked at linear regression on the inputs themselves. This amounts to assuming ambient basis functions which are the standard Euclidean basis vectors:

$$e_i^j = \delta_{ij}$$

This limits the expressiveness of the model to capture higher-order features. One way to address this is to project the input into a higher dimensional space using a set of basis functions. For example, the scalar input **x** could be projected onto the powers of **x** as follows:

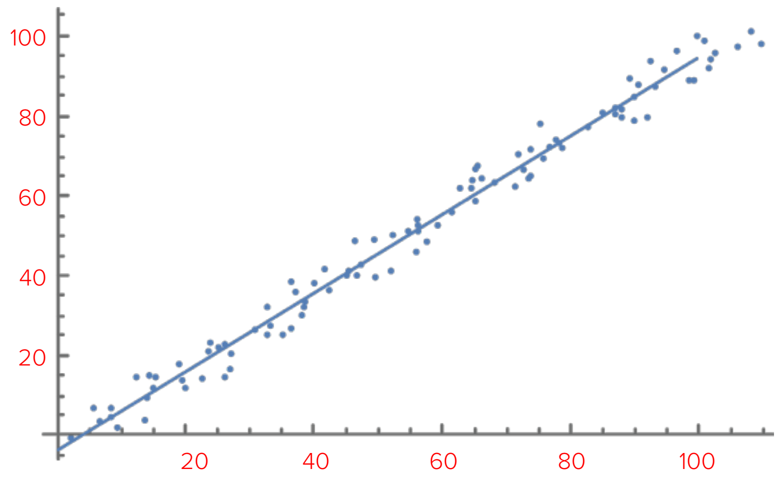
$$\Phi(x) - (1, x, x^2, x^3, \ldots)^T$$

The model is still linear in the parameters **w** so long as the basis functions are fixed and, therefore, analytically tractable.

Model and Observation

Let's assume now that we have some data (our first dataset) of housing prices and a corresponding vector of features associated with that housing price. We can call this set of data our observations. But, more specifically, it is labeled training data in that we know the actual or correct or instructive answer for the housing price based on the input features. This is often referred to as the ground truth. We will see how these training examples serve as instructive feedback to train our model (in our case the determination of the parameters w_0 w_1 w_2 ... w_n).

However, before we consider learning, let's consider the reality of how we obtain these observations in the first place. The plot below shows an example set of observations for a linear regression model in which we have a single input feature along the x-axis. The target value (output) is along the y-axis. You could imagine this as our housing price predictor in which we considered only a single input feature, which in this case might be (for example) the income of housing buyers (in \$1000s) along the x-axis and target price (in \$1000s) along the y-axis. This isn't a particularly realistic or useful example, but it's fine for instruction purposes.



Note something interesting about the plot above. The data points don't all lie on a straight line! Why is this? The answer is that each observation will be corrupted and thereby perturbed by some noise. We use the term noise to generically capture any error in our observations. It may stem from measurement error, physical noise in a measurement apparatus, human error, and a host of other sources that introduce some randomness into the mix. Thereby, we can model this noise as a random variable **ε**. As we will learn later, the noise term **ε** will be characterized by a probability density function (typically Gaussian) and temporal correlation with noise at other time instants. Usually, we assume the noise is uncorrelated across time, and we typically model this noise as a Gaussian random variable characterized by its mean and variance. With this new insight, our observations **y** can be expressed as follows:

$$y = f(x) + \varepsilon$$

However, we are interested in **f (x)**, and specifically, the parameters **w**. In essence, the linear regression problem boils down to the problem of finding the optimal parameters **w** for a linear model **f (x)** based upon observations that have been perturbed by some random noise **ε**.

How do we find these parameters **w**? This process is what we call learning. And we shall see that for linear regression, there is a myriad of techniques for finding the parameters, including a closed-form solution called maximum likelihood least squares, via Bayesian methods and optimization methods such as gradient descent. While we shall explore all of these at some level, gradient descent is the most relevant for our discussion as it is more or less the only game in town when considering non-linear models such as deep neural networks. We will, therefore spend considerable time exploring how it works (but not in this first paper).

For now, let's simply show the results for the closed-form maximum likelihood least squares closed solution, which is possible because the problem is linear (this result will be derived in a later paper).

Maximum Likelihood, Least Squares, and Loss Functions

Suppose we have a linear regression model and wish to evaluate how well it performs. One straightforward way to do this is to compare its predictions with the ground truth—our observations. Of course, our observations are corrupted by noise, but that's the whole point. We wish to find the model that minimizes these errors in some way.

We can define what is called the expected loss as follows:

$$E[L] = \int\int L(t, \hat{f}(x))p(x, t)dx\, dt$$

Here, $t = y(x)$ for notational convenience. The function **L** is called the loss function. A first-order function won't do as we might expect 0 if the error is distributed uniformly around 0. So, instead, typically a second order loss function is defined:

$$L(t, \hat{f}(x)) = (t - \hat{f}(x))^2$$

$\hat{f}(x)$ is the estimate for the model, i.e., the set of parameters that best estimate the true model. In other words, the best-fit line in the plot above represents the parameters for **f (x)** that minimize the total loss function overall observations.

Starting with the likelihood probability distribution:

$$p(y|X, w, \beta)$$

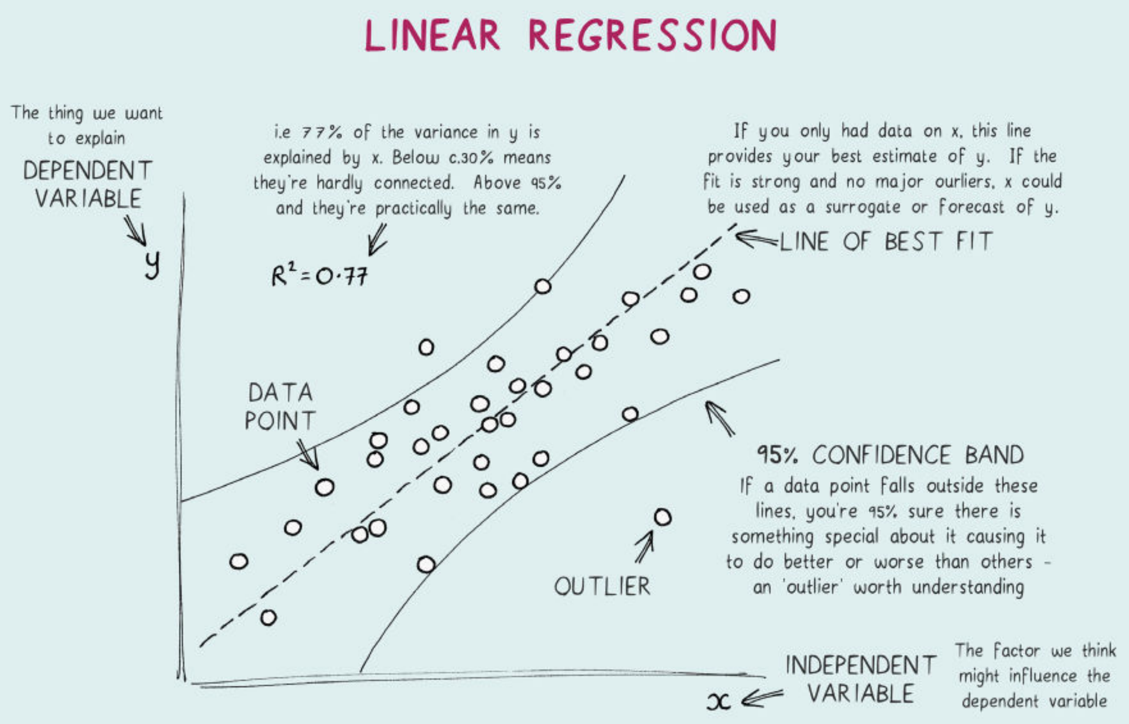
We can then find the extremum (maximum values) for this probability distribution by taking the gradient with respect to the parameters **w**. We will see later that this maximization is equivalent to minimizing an error or cost function, which is typically a quadratic function of the estimated and target values for the model. This method is also referred to as the least-squares solution. After some manipulation, we will find:

$$w = (\Phi^T\Phi)^{-1}\Phi^T y$$

Where **Φ** is a matrix of the data set vectors, which may be mapped to a higher dimensional space using a set of non-linear basis vectors (more on this later).

Thus, the line in the above plot for a simple 1-D problem, represents the maximum likelihood least squares for the dataset shown—i.e., it is the best fit for the data set shown.

The image below sums up much of what we've discussed so far re: linear regression.


LINEAR REGRESSION
The thing we want to explain
DEPENDENT VARIABLE
y
i.e 77% of the variance in y is explained by x. Below c.30% means they're hardly connected. Above 95% and they're practically the same.
R² = 0·77
If you only had data on x, this line provides your best estimate of y. If the fit is strong and no major ourliers, x could be used as a surrogate or forecast of y.
LINE OF BEST FIT
DATA POINT
95% CONFIDENCE BAND
If a data point falls outside these lines, you're 95% sure there is something special about it causing it to do better or worse than others - an 'outlier' worth understanding
OUTLIER
INDEPENDENT VARIABLE
x
The factor we think might influence the dependent variable

Bias and Variance

Before moving on, it is essential to discuss a very important topic that will continue to play a major theme in our AI and ML investigations. The topic is *overfitting* and *underfitting*. The capacity of a model refers to its ability to fit a wide variety of functions. Models with low capacity may fail to fit the training set resulting in a situation called underfitting. Models with high capacity can overfit by learning properties of the test set that are undesirable such as noise.

The capacity of a model is determined by the number of parameters in relation to the dataset to be modeled. An overfitted model comprises more parameters than are justified by the data. Conversely, underfitting occurs when the number of parameters is insufficient to capture the data being modeled. An example would be attempting to fit a linear model to non-linear data.

We will learn later that overfitting is an artifact of maximum likelihood estimation and does not arise in Bayesian approaches. From a frequentist perspective, a relation called the bias-variance tradeoff arises. We won't explore it in any depth here. However, we will write down the equations that define this relationship and discuss their qualitative meaning. Later, we will derive this relation. Here is the bias/variance tradeoff:

$$E_D[(y - \hat{f}(x; D))^2] = (\mathrm{Bias}_D[\hat{f}(x; D)])^2 + \mathrm{Var}_D[\hat{f}(x; D)] + \sigma^2$$

where

$$\mathrm{Bias}_D[\hat{f}(x; D)] = E_D[\hat{f}(x; D)] - f(x)$$

$$\mathrm{Var}_D[\hat{f}(x; D)] = E_D[\hat{f}(x; D)^2] - E_D[\hat{f}(x; D)]^2$$

σ^2 is the squared variance of the noise $\boldsymbol{\varepsilon}$.

The squared bias, represents the extent to which the average prediction over all datasets differs from the desired regression function.

The variance measures the extent to which the solutions for individual data sets vary around their average. Hence, this measures the extent to which the function $y(x; D)$ is sensitive to the particular choice of data set.

The "tradeoff" relates to the fact that models with a lower bias have a higher variance and vice versa.

Logistic Regression—the Art of Classification

We've seen that linear regression is a very powerful technique for modeling relationships between input features and inference such as, for example, the dependence of housing prices upon various variables such as location, square footage, number of bedrooms, etc. While linear regression is great for predicting the value of a function based upon a set of input features, what if we wish to predict whether some entity in the world belongs to a particular class or type?

For example, suppose we have a set of photos of dogs and wish to build a model to predict the breed of dog from a photo. In this case, we may have a range of classes such as, "German Shepard", "Labrador", "Poodle", etc. The input to our model might be a digital image comprising individual pixels, and the output would be a variable indicating the particular dog breed in the photo. For example, perhaps using our example "German Shepard" is assigned to the class 1, "Labrador" is assigned to the class 2 and "Poodle" is assigned to the class 3. In other words, we wish to build a model that can learn how to distinguish these different breeds of dogs based only upon an input photograph.

Logistic regression is a statistical technique to model the probability of an input belonging to a particular class. While in linear regression, the model prediction $f(x)$ was a linear function of the input parameters w:

$$f(x) = w^T x$$

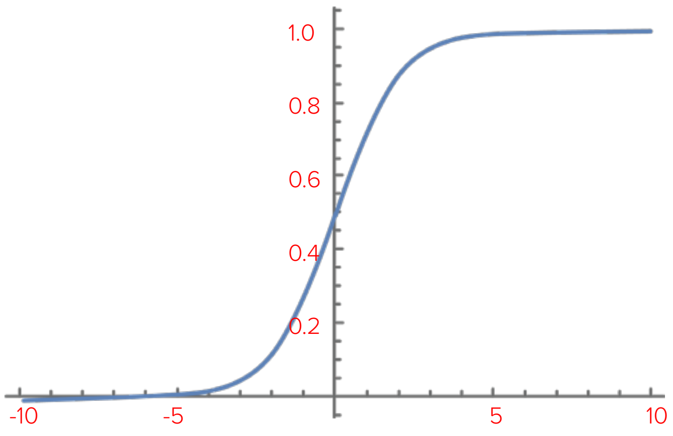
In logistic regression, we seek to predict the probability of an input belonging to a particular class. Therefore, our output values range from 0-1. In other words, the output of our model is in fact a probability. In order to achieve this, we can build off the linear regression model by simply wrapping that model in a non-linear function with range 0-1 like so[10]:

$$f(x) = g(w^T x)$$

This non-linear function g is often referred to as an activation function of which there are several commonly used variants, including the sigmoid function, hyperbolic tangent, and ReLu ("Rectified Linear Unit"). We will explore these different activation functions later. For now, the sigmoid function is expressed as:

$$g(x) = \frac{1}{1 + e^{-x}}$$

Here's what it looks like:

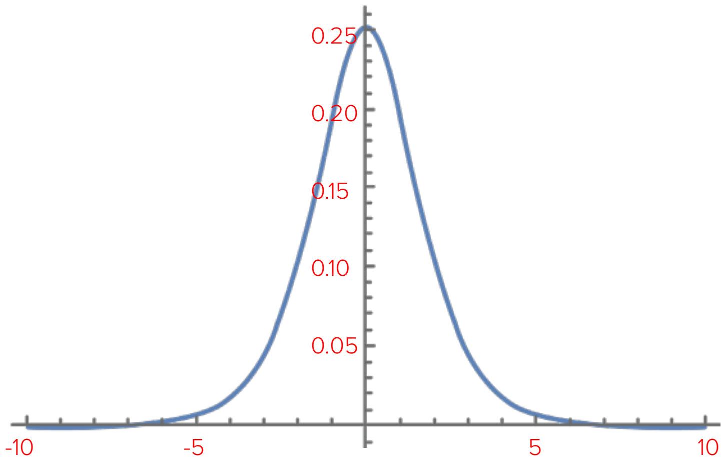


Then, we will see later that the activation function is essential to the non-linear modeling characteristic of

[10] Sometimes people refer to this activation as a "squashing function" as it does just that. It squashes the range to be between 0 and 1.

deep neural networks, which among other things, makes them so powerful.

The sigmoid function is a cumulative distribution function ("CDF"). The associated probability distribution function ("PDF") may be found by taking its derivative and looks like this:



with analytic form:

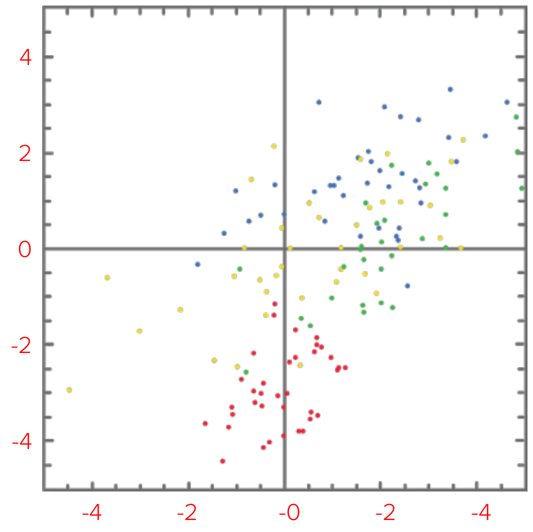
$$f(x) = \frac{e^{-x}}{(e^{x/2} + e^{-x/2})^2}$$

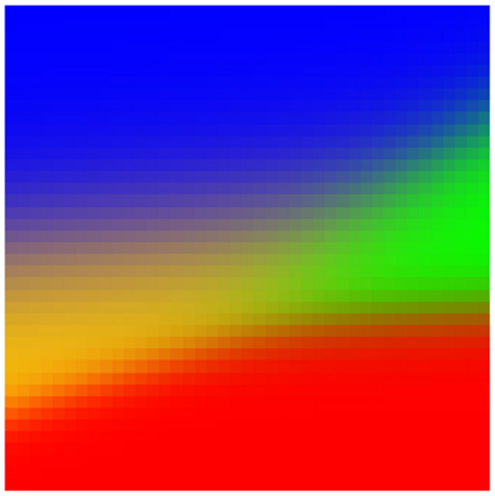
Since we now know how to model our classification problem as a probability, we would like our machine learning algorithm to generate a prediction:

$p(C_k|x)$

This is the conditional probability of class **k** given **x**. In other words, if the input is **k**, what is the probability that it belongs to class **k**.

Here is an example of a data set drawn from a probability distribution to which a classifier has been applied.



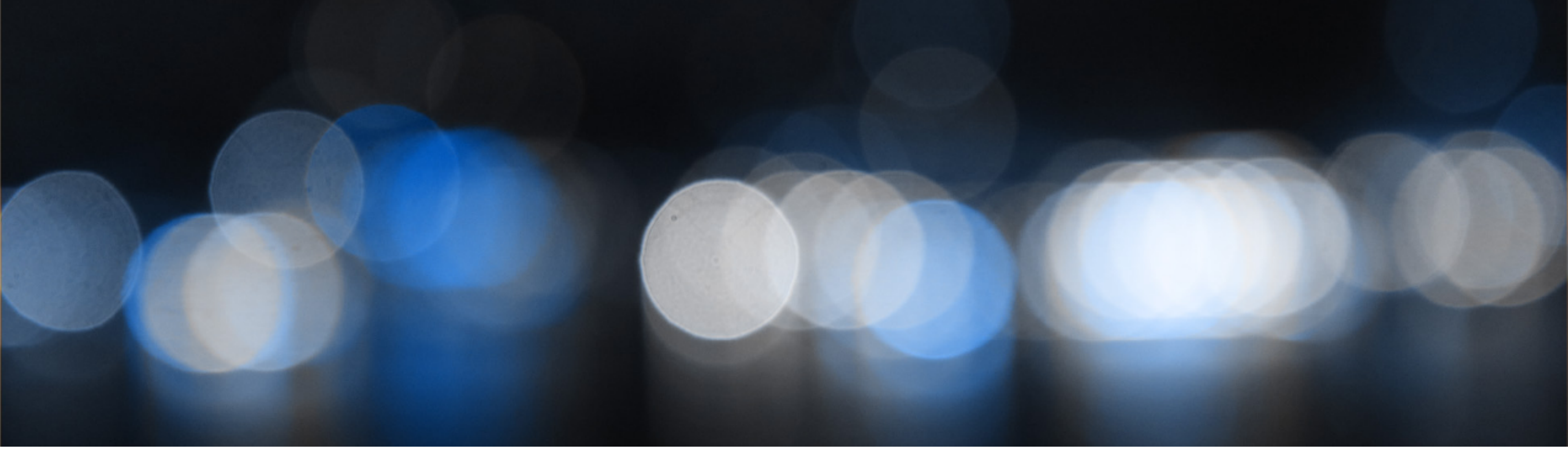


Logistic Regression

You may wonder what type of loss function we might employ for logistic regression. Typically, classifier networks utilize something called cross-entropy loss. The cross-entropy effectively measures the "distance" between two probability distributions. This makes sense, as our output is indeed a probability distribution. In other words, cross-entropy loss measures the distance between the probability distribution of our training data and the probability distribution generated by our network predictions. For a binary classifier, the cross-entropy loss can be expressed as:

$$H(p,q) = -\sum_i p_i \log q_i = -y \log \hat{y} - (1 - y) \log (1 - \hat{y})$$

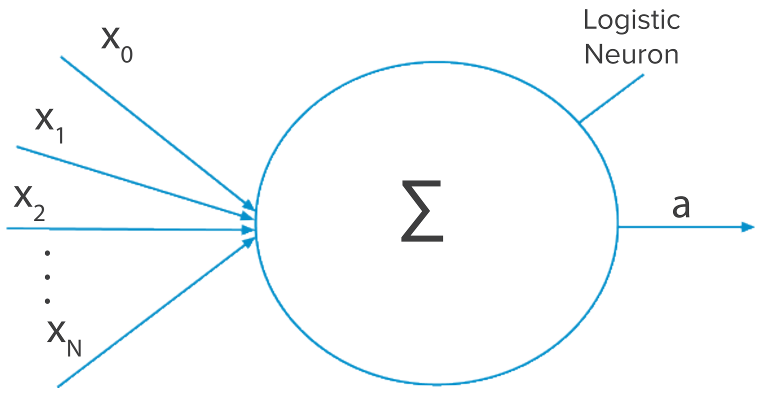
Finally, we wish our output to in fact resemble a probability distribution. How is this accomplished? For this, an activation function called the softmax activation function is employed. It effectively squashes our arbitrary output into a card-carrying PDF.



From Logistic Regression to Deep Neural Networks

We will soon explore the deep learning “zoo” housing many species including feedforward neural networks, convolutional neural networks, recurrent neural networks, LSTMs, GRUs, autoencoders and many others. Each species plays a unique role in attacking particular problems and applications, but the development of a model architecture is at least as much art as it is science.

A logistic neuron is the building block of deep neural networks. Here it is in diagrammatic form:



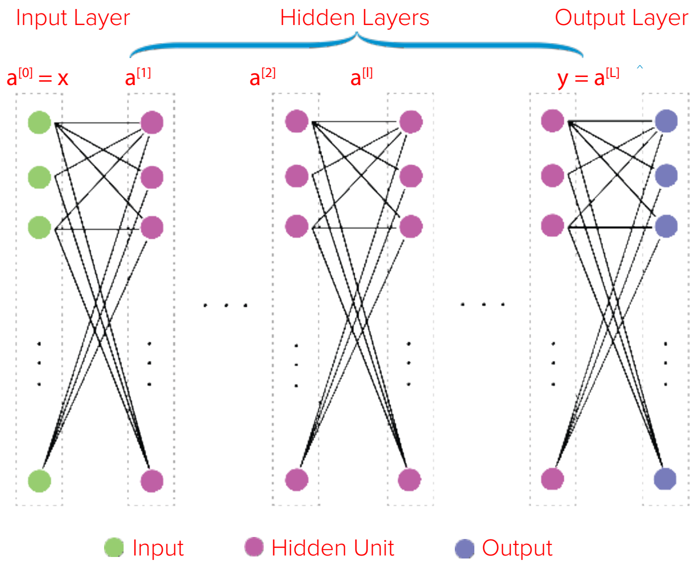
$z = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$

$a = \sigma(z)$

$\sigma(z) = \dfrac{1}{1 + e^{-x}}$

There’s nothing new here. The logistic neuron simply encapsulates logistic regression in a single unit. And, in some sense, a deep neural network is constructed by weaving together many layers of these logistic neurons.

Here’s the basic architecture of a deep neural network called a feedforward neural network, which couples together many logistic neurons arranged in layers. With this approach, we end up with an architecture called a feedforward deep neural network.



The feedforward network consists of an input layer, an arbitrary number of hidden layers, and an output layer. Note that each layer is typically followed by a non-linear activation function, such as the sigmoid we discussed earlier. These activation functions are what imbues non-linearity into our network, which is so highly desired. Indeed, the real world, which we seek to model is highly non-linear.

The hidden layers are hidden in the sense that they and the representations they learn are not directly accessible from the outside. In this sense, we arrive at a key aspect of neural networks and how they learn called representational learning.

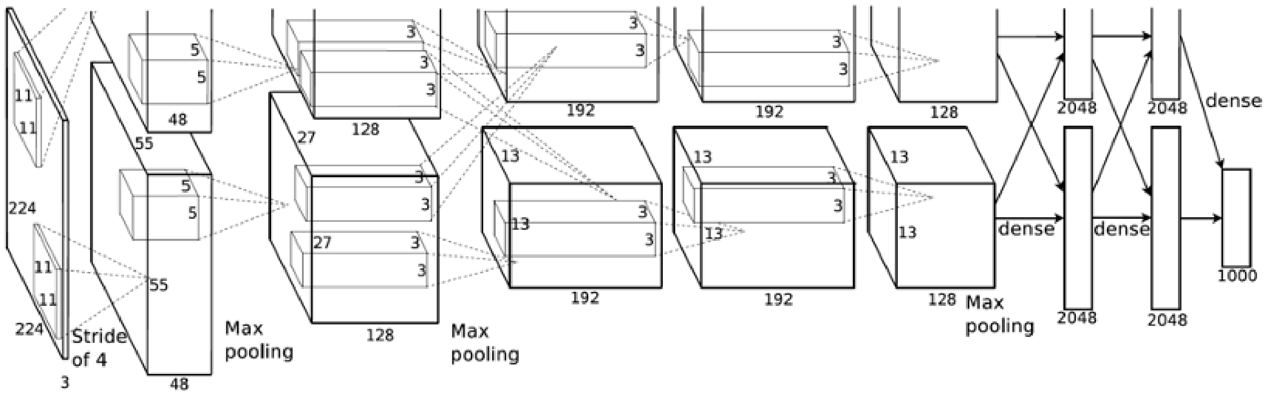
Representational Learning

Deep neural networks learn using a technique called representational learning. This is a very important concept along with non-linearity enabling the power of deep learning. Early excursions into pattern recognition depended upon hand-engineering features. Hand-engineered features mean that a human decides a priori, perhaps based upon some heuristic, which features mattered in distinguishing one class from another.

For example, in distinguishing dog breeds, perhaps poodles might be distinguished from other breeds based upon the salient qualities of their fur. Alternatively, perhaps the eye placement or spacing might provide a significant clue to classifying these breeds. With hand engineered features, the algorithm author—a human—manually codes for the features that seem relevant. The problem is that generally hand-engineering of features just doesn't work that well and regularly falls apart when attempting to generalize. As humans, we simply don't have enough insight into building robust algorithmic representations of classifiers. Our intuition alone is not sufficient.

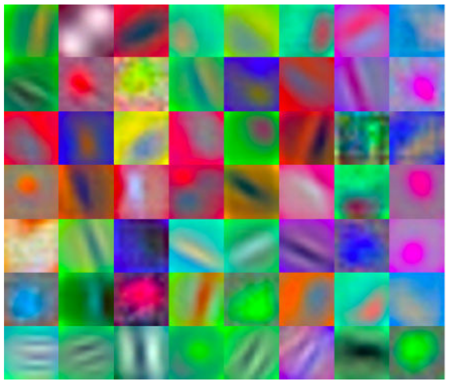
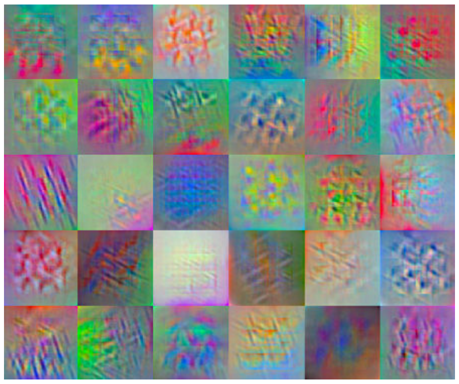
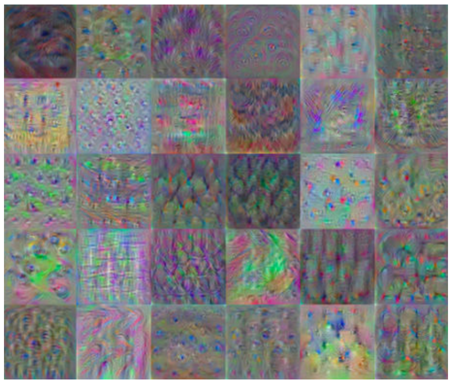
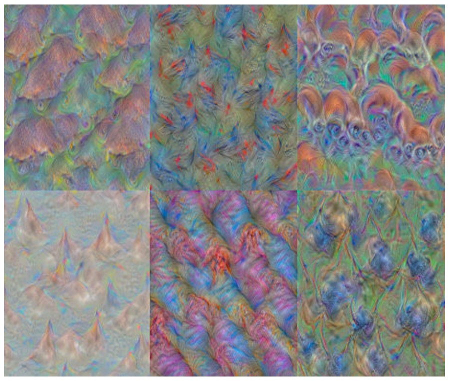
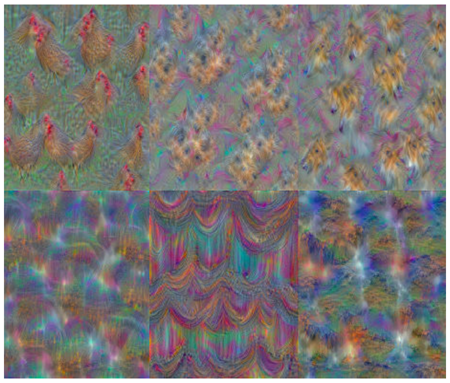
Representational or feature learning, on the other hand, takes a much more clever approach: Let the AI model itself learn the features that matter to it. This is a key insight that cannot be overestimated in its importance in the success of AI. We will explore some of the examples of representational features that AlexNet (an image classification network) learned by itself. In most cases, the features that the AI model cares about have little to no intuitive meaning. In some cases, there are hints of human-level intuition in the features learned. We don't and shouldn't care (except for theoretical studies) what features our AI model cares about so long as it can achieve high accuracy in its inference task.

AlexNet is a deep neural network using a particular variant of neural networks called convolutional neural networks for performing image classification. It competed and won in an international competition in 2012 called the ImageNet Large Scale Visual Recognition Challenge. Here's the high-level architecture of AlexNet.



Don't worry about what these blocks and interconnections mean. They will all become clear by the end of this series. Since we haven't yet formally defined what a neural network is or, for that matter, a deep neural network, this may seem opaque, but it's somewhat intuitive. Imagine a deep neural net as comprising a series of abstract layers. Each layer may learn different features relevant to the task at hand. The images below show the features learned by AlexNet at various layers of the network. With this in mind, consider the images below, which show a graphical representation of features that various layers of AlexNet learned.

The earlier layers are presented first. Note, that the shallowest layer appears to be learning abstract shapes and edges. As you delve into deeper layers, these patterns become more and more complex. However, notice by the deeper layers it is possible to make out the archetypes of image primitives that we in fact, can recognize.

Whether our human visual perception system operates similarly is a fascinating question. However, for our purposes in exploring AI, what matters most is an appreciation of the power of representational learning - the idea that the model itself learns the features that matter.

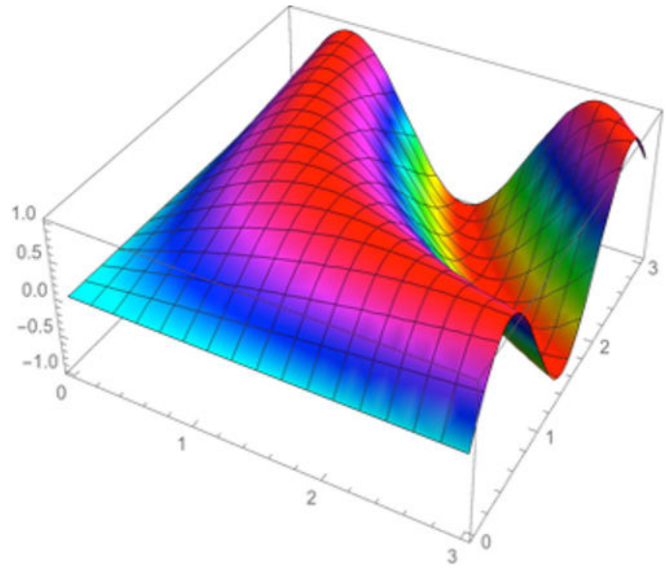
You may be asking yourself how representational learning can happen. How can the model know what features to learn without having a human explicitly tell it? The answer is somewhat subtle but lies in the process of training using an optimization method called gradient descent that we shall explore in some depth in later papers.

Gradient Descent, Backpropagation, Convexity and All That

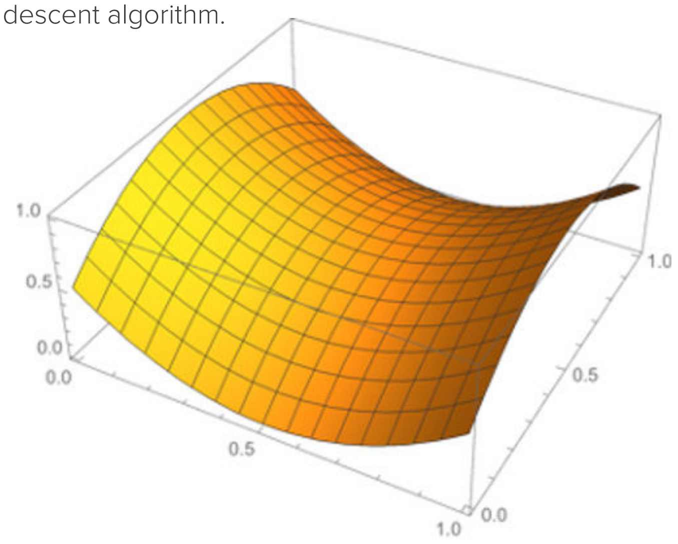
Because with neural networks and deep learning we are in a highly non-linear space, the closed-form solution for finding the optimal parameters we saw with linear regression won't work anymore. Instead, neural networks rely on numerical methods, specifically an algorithm called gradient descent. Gradient descent is best imagined as to how one might navigate in mountainous terrain to find a valley. If the ground is sufficiently rugged, a navigator would not be able to see globally in which direction the valley lies. Instead, the navigator following a gradient descent algorithm finds the local direction of steepest descent and follows that for a small distance. The direction of steepest descent can be applied by using the gradient (derivative) operator in our high-dimensional parameter space. After proceeding some distance, the navigator retests for the local direction of steepest descent and then proceeds in that direction. The process is repeated indefinitely until some convergence criteria is obtained.

You may wonder how these gradients are computed in practice in the high dimensional parameter space. This is the province of the famous back propagation algorithm, which we will discuss in depth in later installments.

With the great power that deep neural networks offer in addressing non-linear models, they also bring with them some potential pitfalls, among which is the issue of the topology of the parameter space over which we seek to find a minima for the loss function. The loss function itself may be non-convex. While there is a precise mathematical definition for convexity, what it means in this context is that the weight space may be riddled with many local minima. Therefore, instead of finding the global minimum, which we seek, our gradient descent algorithm may, in fact, find a local minima.



We will find that under certain reasonable assumptions, the probability of non-convex functions decreases with the dimensionality of the parameter space we work in. Instead of local minima, we would expect to find saddle points, which, although are not fatal, may significantly slow the speed of convergence of the gradient descent algorithm.

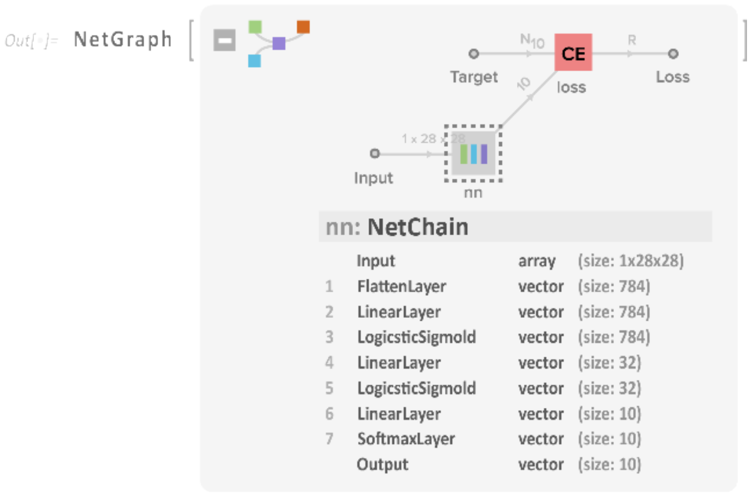


Our First Neural Network

After all this theory, let's do something practical with it! The goal here is to briefly illustrate building a deep neural network to make predictions based upon the MNIST data. While typically in production we might utilize a framework such as TensorFlow or PyTorch for visualization purposes, we are using Mathematica, which is a beautiful environment for doing machine learning.

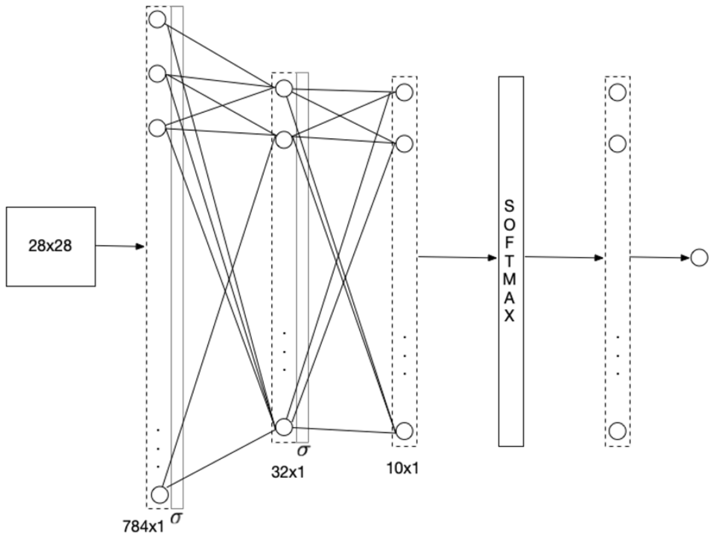
Here is a random sample from our MNIST training data:

Let's build a simple architecture to perform this inference based upon what we know so far. We will use a feedforward neural network with a single hidden layer for illustrative purposes. Here's a block diagram of the architecture of the proposed neural network:



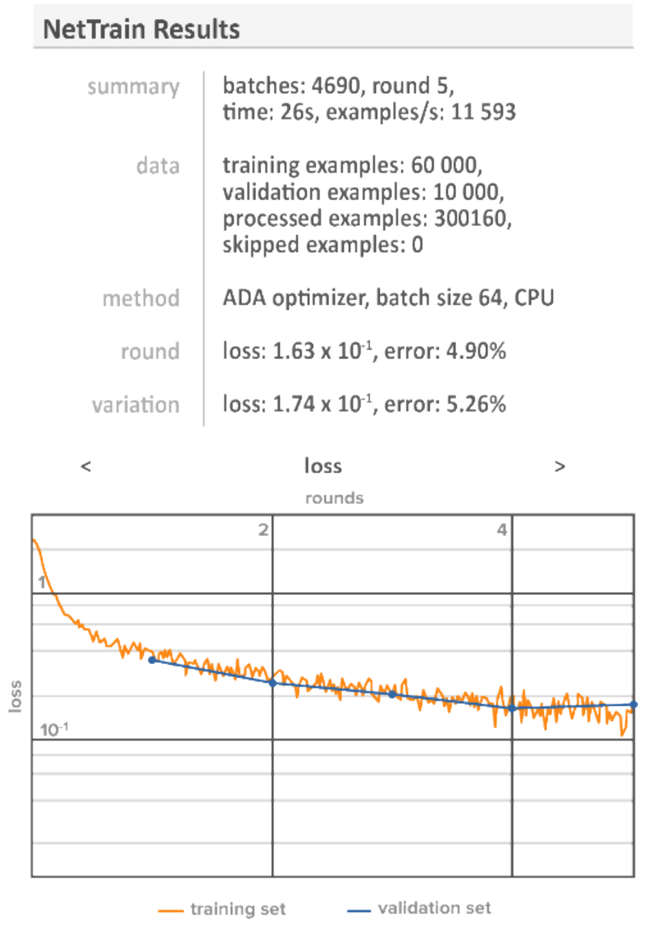
nn: NetChain

	Input	array	(size: 1x28x28)
1	FlattenLayer	vector	(size: 784)
2	LinearLayer	vector	(size: 784)
3	LogicsticSigmoId	vector	(size: 784)
4	LinearLayer	vector	(size: 32)
5	LogicsticSigmoId	vector	(size: 32)
6	LinearLayer	vector	(size: 10)
7	SoftmaxLayer	vector	(size: 10)
	Output	vector	(size: 10)



Our simple architecture comprises a 784x1 input layer followed by a sigmoid activation function. The 784x1 layer size is dictated by the input images for each digit, which are of size 28x28 (pixels) and then flattened to a one-dimensional vector of size 784x1. The input layer is followed by a hidden layer of size 32x1, followed by another sigmoid layer. Finally, our output is of size 10x1 (representing one of the ten digits we seek to infer). The output layer is followed by something called a softmax activation function, which operates to transform our output into a probability distribution (we will discuss the softmax in much more detail later). Finally, based upon the probability distribution from the softmax layer, the network produces a single value, which is the predicted digit.

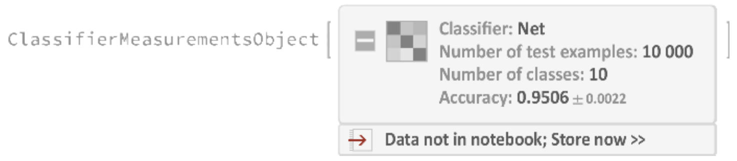
Upon training with the MNIST data set, we find our loss function evolving as follows (the downward trend indicates convergence).



NetTrain Results

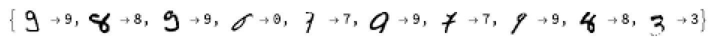
summary	batches: 4690, round 5, time: 26s, examples/s: 11 593
data	training examples: 60 000, validation examples: 10 000, processed examples: 300160, skipped examples: 0
method	ADA optimizer, batch size 64, CPU
round	loss: 1.63 x 10^{-1}, error: 4.90%
variation	loss: 1.74 x 10^{-1}, error: 5.26%

Upon applying our trained model to our test set we end up with 95% accuracy.

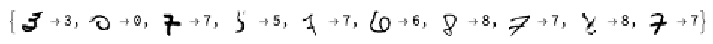


```
ClassifierMeasurementsObject[ Classifier: Net
                              Number of test examples: 10 000
                              Number of classes: 10
                              Accuracy: 0.9506 ± 0.0022
                              Data not in notebook; Store now >> ]
```

And, here were the worst classified examples:

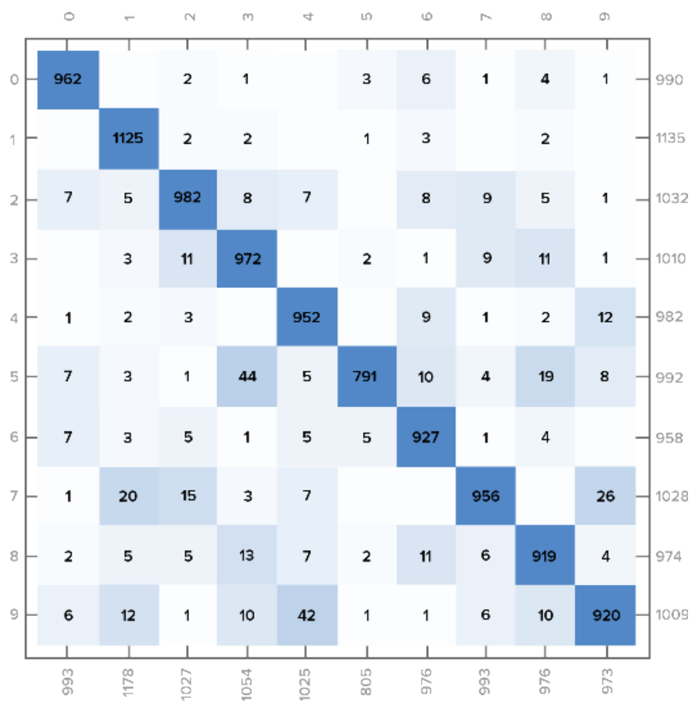


The least certain examples were:



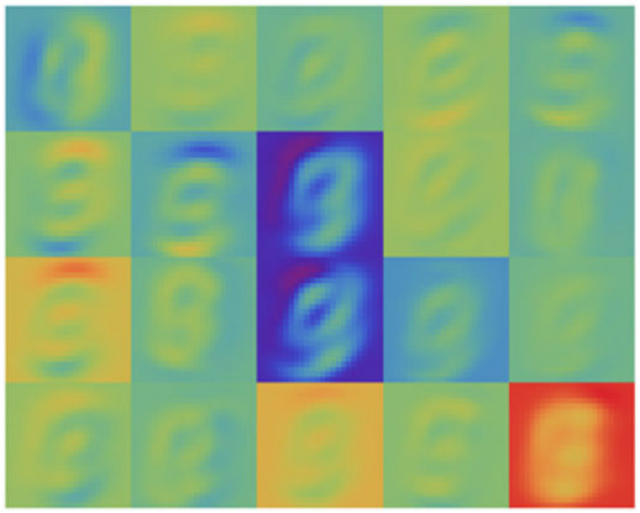
The top confusions were:
7 -> 9, 5 -> 8, 4 -> 9, 3 -> 8, 5 -> 6, 2 -> 7, 3 -> 7, 4 -> 6, 2 -> 3, 2 -> 6

The confusion matrix below shows correct and misclassified examples:



	0	1	2	3	4	5	6	7	8	9	
0	962		2	1		3	6	1	4	1	990
1		1125	2	2		1	3		2		1135
2	7	5	982	8	7		8	9	5	1	1032
3		3	11	972		2	1	9	11	1	1010
4	1	2	3		952		9	1	2	12	982
5	7	3	1	44	5	791	10	4	19	8	992
6	7	3	5	1	5	5	927	1	4		958
7	1	20	15	3	7			956		26	1028
8	2	5	5	13	7	2	11	6	919	4	974
9	6	12	1	10	42	1	1	6	10	920	1009
	993	1178	1027	1054	1025	805	976	993	976	973	

At the end of the day, we would most likely use a convolutional neural network for this task. We haven't discussed those yet but will do in later installments. For now, here is a visualization of what the kernel of one layer of a convolutional neural network applied to the same problem learned:



Finally....

This has been a whirlwind high-level tour of some of the fundamental building blocks of AI and ML. We've covered a lot of ground, albeit at a high level for now. We started with some history and quickly moved to define some of the key terms in the field, including AI itself, machine learning, and its various incarnations as supervised, unsupervised, or reinforcement. Then, we delved into a toy example of machine learning—linear regression. From there, we learned about logistic regression and logistic neurons, the building blocks of deep neural networks and deep learning. Finally, we started to explore at a high level what neural networks are all about and illustrated building a simple feedforward deep neural network in code.

Moving forward in subsequent white papers, we will explore in much more detail the topics outlined here as well as many others, including reinforcement learning. And, of course, once these building blocks are in place, we will examine how AI and ML are currently being applied to solve problems in the commercial sphere. Equally important, we will explore emerging opportunities to apply ML/AI in new ways to tackle many new challenges facing commercial enterprise in the 21st century. We hope you'll stay onboard for what will be an exciting journey.

RAJANT

*If it's **moving**, it's Rajant.*
*Industrial Wireless Networks **Unleashed.***

20+ Years OF GAME-CHANGING WIRELESS

Tel: 484.595.0233 | www.rajant.com

Discover how you can transform your industrial environment into an autonomous operation. Visit www.rajant.com or contact a representative to get started today.