Contents

1 Fundamental Ideas

1.1 Introduction

We assume familiarity with traditional machine learning, basic probability and statistics. We look at why a different perspective is needed and why deep learning is a suitable alternative.

1.2 Traditional Machine Learning

The older/traditional way of applying machine learning consisted of the following steps:

1. **Feature Extraction:** Features which can be used to discriminate between classes is identified. This step usually requires in-field knowledge about the problem.

2. **Model Selection:** A model is selected which trains on the extracted features. Ensable methods can be used to boost performance.

3. **Cross-validation/Testing:** The model is tested/cross-validated on withheld data to check accuracy and tune hyperparameters.

The drawbacks of this approach are:

1. **Feature Extraction:** This step requires in-field knowledge. It is very difficult to study a whole new branch of knowledge for a single problem.

2. **Amount of Data:** In the current era, the amount of data sometimes is simply so large that it is hard to extract features manually.

3. **Unorganized Data:** Feature extraction is hard in unorganized data (such as a text corpus or media inputs like images, audio and videos).

1.3 The idea behind a neural network: An intuitive perspective

The idea is to let the machine learn the important features by itself. For example consider the problem of recognizing handwritten digits like in figure 1.1. The machine learns to recognize easy features like say a straight line(highlighted in blue), curved arc(highlighted in red) and circles(highlighted in green) and how those features combine(Like how two circles form an 8).

To make an algorithm that can do this we take inspiration from one of the best pattern learning devices in the world: The human brain*. We construct an artificial neuron called a perceptron. Our idea is each perceptron is responsible for recognizing a single feature: It gives a high output whenever a feature is present and a low output when it is absent. So if we have multiple neurons combined, we will be able to recognize complex features that contains many simpler features that the other neurons have identified.

Figure 1.1: Simple features present in handwritten digits

*Taking inspiration from the brain is a repeated theme in deep learning. Those inspirations helped us come up with CNNs and attention mechanisms

1.4 Ideas behind a neural network: A mathematical perspective

From a mathematical point of view, there exists a latent space from where the dataset is sampled from. We model the decision boundary in this space using a parametric equation. Then we use already existing data to tune the parameter so that our modelled decision boundary is an estimate of the actual decision boundary.

1.5 Comparison between traditional ML and deep learning

Traditional ML models show better prediction when the amount of features involved is small. Features can be individually engineered and interpreted. Moreover, such models often provide more transparency on ow each feature is used and should be preferred when the question of how the machine a particular conclusion becomes important. Examples include medical domains or when there is a question of ethics involved.

Deep learning models are better when data is unstructured or there are a lot of features which need to be considered. With proper construction and training almost any decision boundaries can be learned.

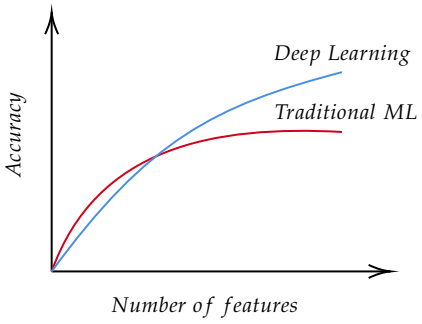


Figure 1.2: Comparison of accuracy between deep learning and traditional ML methods.

1.6 The perceptron

As mentioned before, a perceptron can be thought to be an artificial neuron. We make a simplification and assume that each perceptron

is responsible for identifying some pattern P. A scheme of what a perceptron looks like is given in 1.3

We assume the perceptron returns a high value when it detects P. The inputs to a perceptron can be features from known observation or outputs of other neuron. Let the inputs be $x_1, x_2 \dots x_n$. We arrange them neatly in a vector $X = [x_1, x_2, x_3 \dots x_n]$. Each of those x_is can be thought to be the presence and absence of a simpler feature. We take a weighted sum of those inputs to get $s = \sum_{1 \leq i \leq n} w_i x_i + w_0$. The intuition is the magnitude of w_i is a measure of the importance of feature x_i and the sign is the direction in which x_i affects the feature which the perceptron is detecting. For example, if the perceptron is detecting if the input is 8 and x_i is the output from another perceptron that detects if a straight line is present then w_i will be negative: there is no straight line in 8. On the other hand, x_j is the output from another perceptron that detects if a circle is present then w_i will be positive: there are two of them in 8. w_0 is just a centering constant. The output of the perceptron will be $y = \sigma(s)$, where σ is known as the activation function.

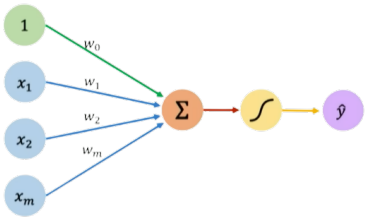


Figure 1.3: Schematic diagram of a perceptron,*Src: MIT Introduction to Deep Learning,6.S191,Lec-1*

1.7 *Chaining Neurons: A simple neural network*

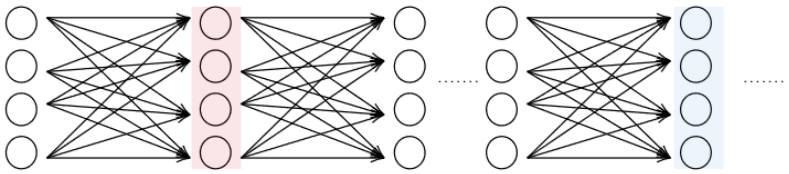


Figure 1.4: A simple neural network

A very simple neural network is show in 1.4. Each circle represents the output of a perceptron while the arrow pointing to it are the inputs. Each row of neuron is(for example, the blue or the red box) called a layer and the layers between the input and output layer are called hidden layers. The layers on the left(For example, the red layer) learns the simpler features while the layers on the right(For example, the blue layer) learns more complex features, which is composed of multiple simpler features.

1.8 *Non-linearity of the activation function*

Ideally, we want our activation function to be non-linear. A simple calculation shows if the activation function for all perceptrons are linear then even if you use multiple layers the output will be a linear combination of the input. Therefore, we use non-linear activation

functions to capture non-linear dependencies between the input features/ simpler patterns. From a more mathematical perspective, we wish to transform the latent space in a way so that the decision boundaries are linear. This is a common idea when we deal with the question of on which "side" of a hypersurface does a point lie in. For example, if our latent surface is $\mathbb{R}^2$ and the decision boundary is given by some nicely parameterized curve γ (Some examples of a nice γ are $x^2 + y^2 - r^2 = 0, mx + c - y = 0, p(x) - y = 0$ where p is polynomial) then the two sides of gamma are given by $\gamma^{-1}(-\infty, 0)$ and $\gamma^{-1}(0, \infty)$. An example showcasing this is given in 1.5

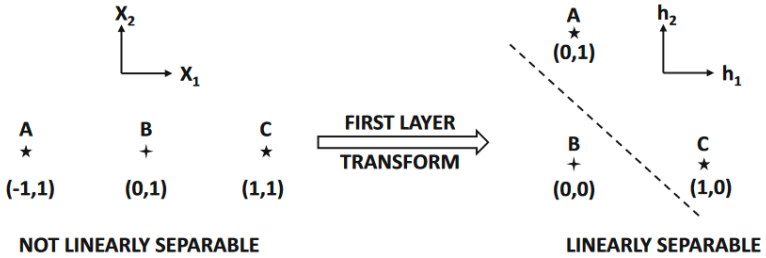
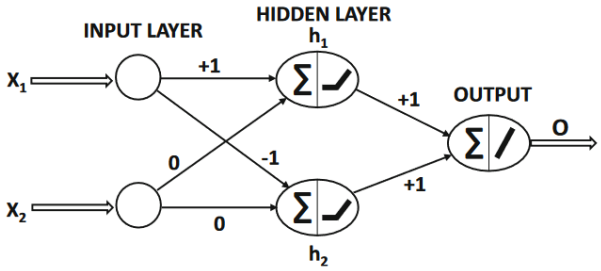


Figure 1.5: A representation of how a non-linear activation function transforms the latent space to give linear boundaries(Aggarwal et al., 2018)



1.9 *Inputs and outputs to a neural network*

1.9.1 *Classification problem with tabular dataset*

Suppose you have an observation where given features $X = [x_1, x_2 \dots]$ you need to predict the label Y. First we replace all categorical x_i and Y with their one hot representation*. If X can take one l labels (number of possible choices of Y), then the second to last layer and the last layer both have l nodes. If the output from the second to last layer is

*Given categories $c_1, c_2, c_3 \dots c_m$, if x_i belongs to category c_k then in a one hot representation, we represent x_i by e_k where e_k is the k^{th} vector in the standard basis

y, at the last layer we take a softmax defined by:

$$\hat{Y}_i = \frac{\exp(y_i)}{\sum_{1 \leq i \leq l} \exp(y_i)}$$

The softmax function maps $\mathbb{R}^n \to [0,1]^n$ and the interpretation is $\hat{Y}_i = P(X \text{ has the label } i)$[Note, by definition $\hat{Y}_i \in [0,1]$ and $\sum \hat{Y}_i = 1$]. Softmax is quite expensive to compute. Depending on the problem, we might replace it with other scoring system where a higher $\hat{Y}_i$ in the output layer means a higher probability that X is in class i.

1.9.2 *Problems where the input is unstructured data*

If the data is unstructured, we use embedding mechanisms to represent them as points in some $\mathbb{R}^n$ in such a way that similar entries are closer together. Then we proceed as usual.

1.9.3 *Regression tasks*

In regression problems, the output $\hat{Y}$ is simply an estimate of Y.

Bibliography

Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10(978):3, 2018.