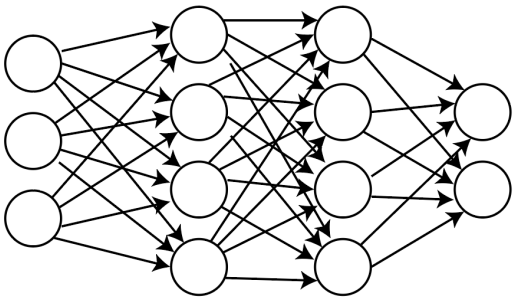
Project T Team Ferda: MultiLayer Perceptrons

Alec Zhou, Conor O'Shea, Jordan Byck

Figure 1:



1 What is a Multilayer Perceptron?

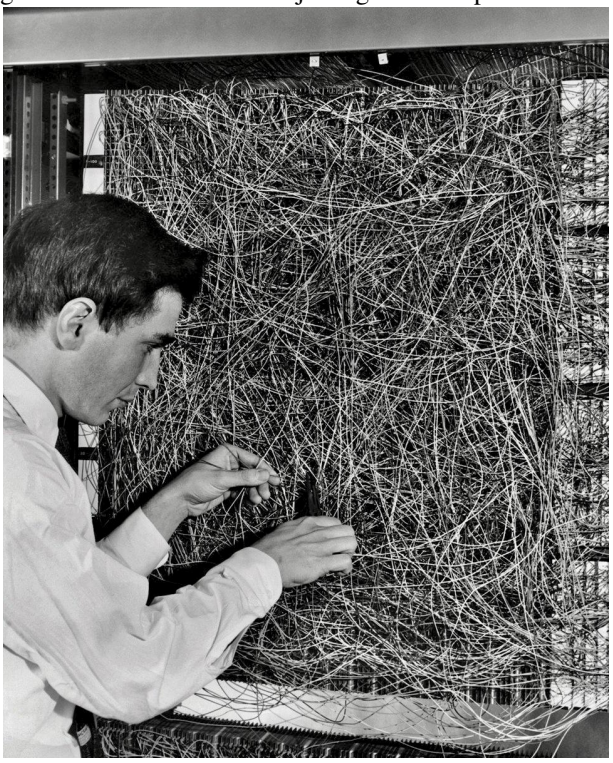
A Multilayer Perceptron is a form of model used for machine learning tasks. When first learning about the structure of MLPS, it may seem overwhelming; however, once you understand how the various parts come together, you'll find it stunningly simple and maybe even a little beautiful. Multilayer Perceptrons were one of the first machine learning models created and now have many different forms used for various tasks. The form the basis of the much more complicated neural networks used in practice today, but for now let's just figure out how the simplest form works. The underlying intuition of an MLP is that it is an *artificial neural network*. This just means that it's a digital attempt at mimicking the structure and function of the neurons in animal brains (though, be careful with this metaphor - it is only a guiding model). It comes with no surprise then that MLP and its derivatives are exceptional at tasks such as image and speech recognition and machine translation. MLPs are also useful since they can easily learn non-linear models since they belong to a class of of models called universal function approximators.

2 History of Multilayer Perceptrons (MLPs)

The Multilayer Perceptron algorithm was developed at the Cornell Aeronautical Laboratory by **Frank Rosenblatt**, initially designed to be a physical machine (see Figure 2). The first manifestation was created for image recognition, where the neurons are randomly connected to 400 photocells. *Weights* were stored in potentiometers and updated using electric motors. We will explain what these *weights* are later. Its development spurred the first AI hype cycle around the 1960s, though following the discovery of its limitations, the proverbial "AI winter" commenced. Only in the past decade when our computational capacity had grown enough to feasibly run "deep neural networks" did another AI hype cycle begin.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

Figure 2: Frank Rosenblatt adjusting the Perceptron machine.



3 MLP Inputs and Outputs

A multilayer perceptron is a *feedforward artificial network*, meaning that there are no cyclical connections between neurons. Before we dive into the various components of a MLP, let's first look at it as a black box and examine which outputs and inputs work best with them.

3.1 Inputs

The input to an MLP should be a dataset that is numerical. If the data are categorical (eg. dog vs. cat), it can be **one hot encoded**. The MLP takes in a single data point at a time and gives an output value or vector. Ideally, the dataset should also have underlying patterns that we are trying to train our MLP to pick up on.

3.1.1 One Hot Encoding

One hot encoding is the task of turning categorical data into numerical data, which can subsequently be used by in machine learning models. This is because numerical functions and matrix operations used by an MLP cannot be performed on categorical data.

In the process of one hot encoding, the various categories for a variable are spread out into binary variables which correspond to that class appearing in that data. As a toy example (visualized in *Figure 2*), let us one hot encode 4 cars that have some of the colors red, blue, or yellow. We can see that each *car* data point corresponds to three binary variables that indicate whether any of *Red, Blue, or Yellow* are present. When the categorical variable contains the value that matches the binary variable, the binary variable is 1.

	Color
My car	red
Sister's car	yellow
Brother's car	blue
Clown car	red, blue, yellow

	Red	Blue	Yellow
My car	1	0	0
Sister's car	0	0	1
Brother's car	0	1	0
Clown car	1	1	1

Example of one hot encoding. The left table corresponds to the data in categorical form while the right is one hot encoded. Each row is a data point.

3.2 Outputs

An MLP ultimately outputs the values or vector of values required by the problem. MLPs are often used in:

- **Regression Problem** – our model predicts a continuous value (eg. birth weight, restaurant rating, AQI level, housing price)
- **Binary Classification Problem** – our model predicts between two classes (eg. malignant or benign tumor, hot dog or not, Democrat or Republican)
- **Multi-Class Classification** – a generalization of the Binary Classification Problem, our model predicts among multiple classes (eg. colors, brands, countries, handwritten digits - think this one through!)

4 The Neuron

The **neuron** is the fundamental building block of a Multilayer Perceptron, which are represented as circular nodes in Figure . Within the functioning of a perceptron, there are a few key players which are essential to both understand alone and as a part of the whole.

- **Weight** (w) – connection between two neurons which holds a value, gives a "weight" to an input by multiplication
- **Bias** (b) – a constant offset independent from the data features for which we also learn the weight
- **Activation function** $\phi(z)$ – a nonlinear transformation that determines how intensely the neuron "fires" (we'll cover how this summation works more later).

Now let's see them in action. Below we have the output for a single neuron.

$$\phi(b + \sum_{i=1}^{n} x_i \cdot w_i) = Z \tag{1}$$

For each input x_i, we multiply it by its weight. These weighted inputs are summed up to create a *weighted sum*, which we can also calculate by using a dot product between a vector of weights **w** and a vector of inputs **x**. **MLPs 'learn' by iteratively adjusting weights and seeing how these adjustments match out desired output down the line.**

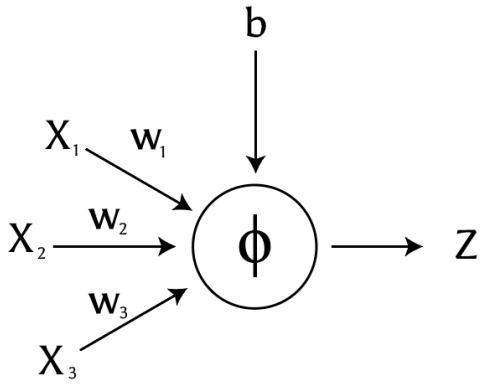
In case our weighted sum is not enough, the *bias term* b is then an extra positive or negative nudge we can give to our weighted sum (before passing it into the activation function $\phi(z)$). **MLPs also learn by iteratively adjusting and testing bias terms.**

Figure 3 illustrates the visual intuition behind this formula acting like a neuron. The dot product of each input x_i and its corresponding weight w_i are computed, summed and transformed by an activation function $\phi(x)$ to produce an output Z. Let's examine two activation functions, which will be covered in more detail in later lectures.

5 Activation Functions

Activation Functions are applied to each perceptron to transform the weighted sum and bias term into values that are easier to train subsequent weights on. The simplest activation function is a linear activation function; however, networks made from solely linear activation functions are easy to computationally train, but cannot be trained to properly map complex functions.

Figure 3: The structure of a single neuron. The neuron below has three inputs X_1, X_2, and X_3, however, there could be an n number of inputs and their corresponding weights.



5.1 ReLu Function

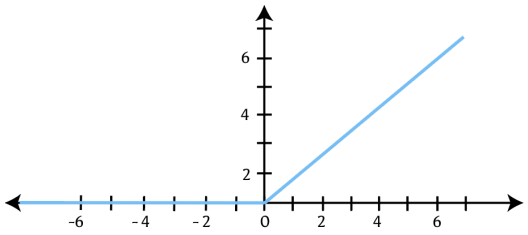
ReLu stands for Rectified Linear Unit. Networks that use ReLus for hidden layers are referred to as rectified networks. If the function value is less than 0, it will output 0, otherwise if the function value is greater than 0, it will simply pass the function input. The ReLu activation function can be defined as a piecewise linear function:

$$ReLu(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

While, in code, it is much simpler. This makes it easier to compute on many nodes too.

$$ReLu(x) = max(0, x) \tag{2}$$

Figure 4: The ReLu function graphed.



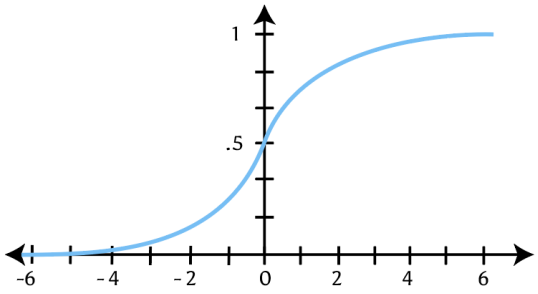
5.2 Sigmoid Functions

Sigmoid Functions are function which have a sigmoid or S shape. Large input values of a sigmoid function become larger, map to 1, while small input values map to 0. The sigmoid function essentially squeezes the inputs $0 \leq x \leq 1$, placing less emphasis on large negative or positive outlier values. This specific sigmoid function (the logistic function) is mathematically defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

We now

Figure 5: The sigmoid or logistic function graphed



6 Layers

A layer is a group of neurons which are bundled together to perform computation on previous inputs - these layers can then be stacked. A Multilayer Perceptron is **fully connected**, which means that every neuron is connected to every other neuron in the previous layer and following layer. Though each layer is conceptually the same, it is helpful to regard the first layer as the **input layer** whose outputs are the data features, the middle layers as **hidden layers** that all take the outputs of previous layers as input, and finally the **output layer** that gives us our prediction.

6.1 Input Layer

An input layer contains all of the inputs being passed into the Multilayer Perceptron, which **are required to be in numerical form** – if the data is categorical, we can transform it into numerical values as shown before through *one hot encoding.*

6.2 Hidden Layers

Hidden layers are the part of the MLP which we modify. We can alter the number of hidden layers d or the size of each layer n to have our MLP best learn the function to map the inputs onto the outputs.

6.3 Output Layer

The output layer's form is dictated by the nature of the output we desire. Let's go back over the problems we listed before and see how their output layers would look.

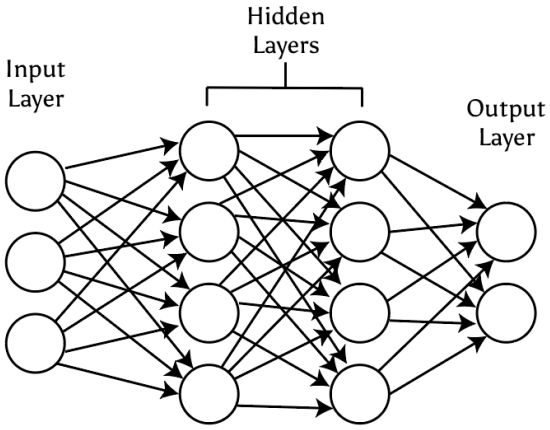
- **Regression Problem**
  - **Single Neuron** – Desire to regress on a single dimension; the estimated miles per gallon of a vehicle.
  - **Multiple Neurons** – Desire to regress on multiple dimensions; the estimated spatial coordinates of a satellite.
- **Binary Classification Problem**
  - **Single Neuron** – Desire to classify a single variable to one of two classes; is this movie going to win an academy award?
- **Multi-Class Classification**
  - **Multiple Neurons** – Desire to find the probabilities that an output is in different classes; what is the probability the person in this picture is smiling, crying, laughing?

6.4 Feedforward

For an input to traverse the layers of a neural network and to become an output it must go through the Feedforward mechanism. If you draw the connections between neurons as a directed graph, in

a Multilayer Perceptron there will be no self-connections or backwards connections in that graph (ie the graph will be acyclic). This is the "forward" part of the feedforward mechanism: nodes in the graph cannot effect nodes that are further up the chain than themselves. Feedforward networks differ from their more modern alternative, recurrent neural networks, in that recurrent networks have self dependencies and can feed information back into themselves allowing for computations to be performed on inputs of variable length.

Figure 6:



In later lectures, we will cover how to train an MLP in more detail, particularly the concept of *backpropogation*.