**Why Learn Deep Learning?**

Get introduced to deep learning and learn why you should consider studying and/or pursuing it.

**We'll cover the following**

* [What is deep learning?](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/5339617817001984#What-is-deep-learning?)
* [Deep learning applications](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/5339617817001984#Deep-learning-applications)

**What is deep learning?**

Deep learning is considered a subfield of machine learning. Even though there are countless inspirations from real neurons, we will focus on modeling everything with formulas, intuitions, and theories that actually work.

In practice, deep learning is the scaling up of computational structures called neural networks.

Why do we take the time to develop such approaches?

Because it is the optimal solution when working with really large-scale data right now.

It is important to keep in mind that deep learning is all about learning **powerful** representations.

There is a huge shift from extracting features to learning features, and that is what deep learning is all about.

**Deep learning applications**

You will get a general perspective of a huge variety of problems that you can solve with deep learning.

First, you will learn to formulate problems in terms of machine and deep learning. That’s a crucial skill that you will use throughout your career and projects.

Secondly, you will learn the most basic components that tackle some of the following tasks:

* Image classification
* Time series prediction
* Image denoising and compression
* Image generation
* Machine translation
* Graph and node classification

Deep learning has already transformed a variety of **businesses** such as web search, augmented reality, social networks, automobiles, retail, cybersecurity, and manufacturing. But the most exciting thing is the potential novel applications that may appear in the future. These projects can radically transform every industry.

Some experts claim that **AI is the new electricity.** While this may be a disputable idea, what is for certain is that **deep learning is one of the most sought after and well-paid skills.**

So why stay behind?

# Linear Classifiers

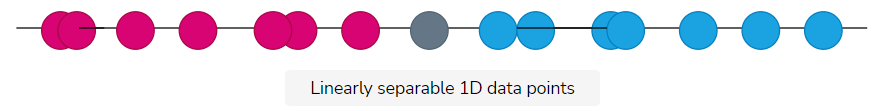
Explore linear classifiers, their principles, and their training process.

**We'll cover the following**

* [What is a linear classifier?](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/5568462289633280#What-is-a-linear-classifier?)
* [Training a classifier](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/5568462289633280#Training-a-classifier)
* [Loss function](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/5568462289633280#Loss-function)
* [Optimization and training process](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/5568462289633280#Optimization-and-training-process)

## What is a linear classifier?

Suppose we want to build a machine learning model to classify the following points into two categories based on their color. It is very easy to see that we can find a single point that separates them perfectly. The goal of our model is to find this point.

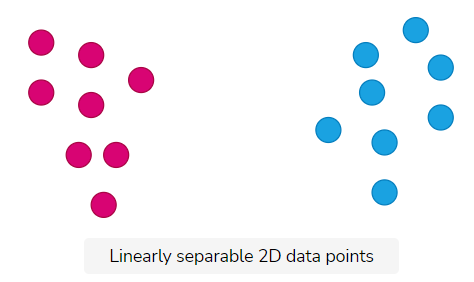


The easiest way to do that is to build a linear classifier. Our classifier has the form  (*x*,*w*)=*w*1​∗*x*1​+*w*2​. The purpose of *f*(*x*,*w*) will be to find the parameters *w*1​ and *w*2​, so that any corresponding scalar point (1D) can be distinguished perfectly. If *f*(*x*,*w*)>0, the point belongs to the blue category. Otherwise, it belongs to the red. Sounds easy?

Let’s extend this idea to 2D data points!

Each point will now be represented as (*x*1​,*x*2​).

For the 2D case, we need to find a line (instead of a point) that separates our 2D points, so our classifier will be *f*(*x*,*w*)=*w*1​∗*x*1​+*w*2​∗*x*2​+*w*3​. Again, the classifier should be trained to find the optimal *w*1​, *w*2​, and *w*3​.

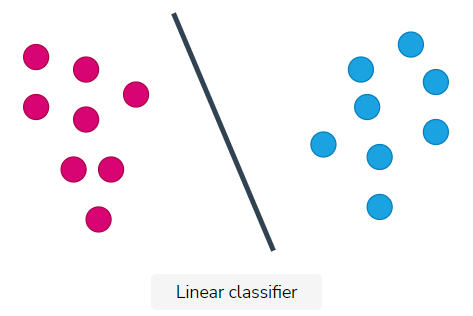


This idea can be naturally extended to arbitrary (N) dimensions. The line in the 2D space will be a plane in the 3D space and N-plane in higher dimensions. We call this the hyperplane of the N-dimensional space that can separate the space into 2 classes (red and blue).

We will utilize linear algebra and matrices to formulate it. To facilitate the readability, matrices will be denoted in bold.

As a result, we now have *f*(*x*,**W**)=**W**∗*x*+*b* , where *x* and *b* are N-dimensional vectors while *W* is an *N*×*N* matrix. We need to find the correct values of **W** and *b* to define a hyperplane. When we have those, we can receive the category *y* for any data-point *x*.

From now on, we’ll denote our classifier as *f*(*xi*​,**W**)



In Pytorch, we can build a linear classifier with 5 inputs and 10 outputs using just one line of code. The following code will initialize a trainable matrix **W** and a vector *b* and every time we use class instance classifier, it will perform the operation *y*=**W**∗*x*+*b*

## Basic imports

import torch

import torch.nn as nn

## initializes a matrix W and a vector b

classifier = nn.Linear(5, 10)

## Training a classifier

We know that we want to find the matrix **W** and the vector *b* in order to classify our examples.

But how?

First of all, we need training data. Training data are data-points (*x*) whose category (target class *t*) we are aware of. Thus, we can utilize it to “train” our classifier.

“Training” a classifier refers to the notion of trying to find the matrix **W** by feeding to its already known data points.

Because we know the “labels” (category) of the data, these training approaches are called supervised. The data are provided in pairs (*x*,*t*). We use the *x* as an input to the classifier and the labels *t* to compute the loss (distance). Note that *y* refers to the output of the classifier and will be equal with *y*=**W**∗*x*+*b*

Intuitively, we will push the randomly initialized model to learn this mapping from *x*−>*t*.

Before we describe the process of training, we need to describe two more concepts.

## Loss function

**Loss (or cost) is a measure of how good or bad a classification of a data-point is**. Alternatively, it can be defined as how far the classifier’s prediction *y* is, for the data-point *x*, from the actual class *t*. Let’s make that crystal clear:

Given a dataset (*xi*​,*ti*​) of N points where *xi*​ is an N-dimensional point in space and *ti*​ is an integer that defines the point’s category, loss is the distance between *f*(*xi*​,*W*) and *ti*​.*Ci*​(*f*(*xi*​,*W*),*ti*​) is the cost for a single example *xi*​.

The overall loss of the entire training data is simply the average of all the individual losses. However, in practice, we rarely average the loss over all data points.

Note that the choice of the loss function depends on the problem and the form of the data. In our case, from now on, we will use the mean squared error distance defined as:

2*C*=∑(*f*(*xi*​,*W*)−*ti*​)2

Notice that the sum is between the elements of the vector. Here is a code example:

import torch

import torch.nn as nn

model = nn.Linear(10,3)

loss = nn.MSELoss()

## dummy input x

input\_vector = torch.randn(10)

## class number 3, denoted as a vector with the class index to 1

target = torch.tensor([0,0,1])

## y in math

pred = model(input\_vector)

output = loss(pred, target)

print("Prediction: " ,pred)

print("Output: " , output)

It is important to understand that even though the target class is a scalar (3 in the example above), we convert it to a tensor. For three classes, you will have these possible target vectors *t*:

class 1 -> [1,0,0]  
class 2 -> [0,1,0]  
class 3 -> [0,0,1]

**Optimization and training process**

Optimization is the process of finding the weight matrix **W** that minimizes the loss function. In other words, it is the process of selecting the individual weights *wi*​ so that the classifier’s prediction *y* is as close as possible to the point’s real label *t*.

Mathematically this can be written as:

*w*′=*argminw*​(*C*(*w*))

For now, let’s keep in mind that optimization is an abstract concept that describes how we select the matrix. We will dive into it in the next lesson where we will talk about neural networks.

Now, we can describe the training algorithm in its entirety:

Given a set of training examples *xi*​ with their labels *ti*​, we need to:

* Initialize the classifier *f*(*xi*​,**W**) with random weight **W**.
* Feed a training example in the classifier and get the output *yi*​.
* Compute the loss between the prediction *yi*​ and *ti*​.
* Adjust the weights **W** according to the loss *Ci*​ (next lesson).
* Repeat for all training examples.

This is the core idea behind all deep learning models. In the end, we will have a trained classifier that can be generalized in previous unseen examples.

The only step that should be unclear now is how we adjust the weights. We will discuss this in the next lesson.