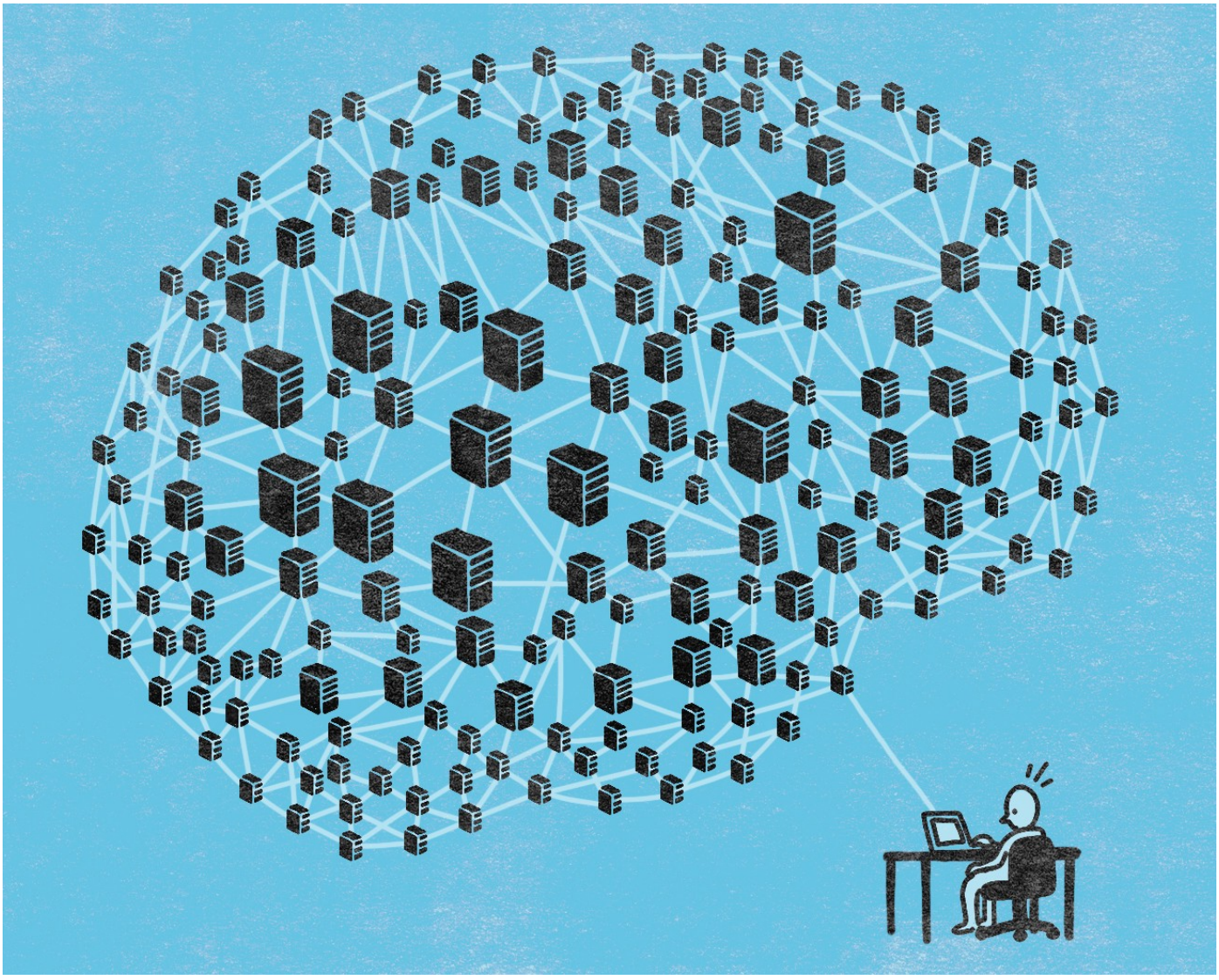
Published in Towards Data Science

Nahua Kang Follow

Jun 18, 2017 · 21 min read · Listen



Source: Allison Linn, Microsoft

Introducing Deep Learning and Neural Networks

Welcome to the first post of my series **Deep Learning for Rookies** by me, a rookie. I'm writing as a *reinforcement learning* strategy to process and digest the knowledge better. But if you are a deep learning rookie, then this is for you as well because **we can learn together as rookies**!

(*You can also read this post on my website, which supports LaTeX with MathJax*)

Source: deepinstinct.com

Deep learning is probably one of the hottest tech topics right now. Large corporations and young startups alike are all gold-rushing this fancy field. If you think big data is important, then you should care about deep learning. The Economist says that data is the new oil in the 21st Century. If data is the crude oil, databases and data warehouses are the drilling rigs that digs and pumps the data on the internet, then think of deep learning as the oil refinery that finally turns crude oil into all the useful and insightful final products. There could be a lot of "fossil fuels" hidden underground, and there are a lot of drills and pumps in the market, but without the right refinery tools, you ain't gonna get anything valuable.

data, unlike oil, is “sustainable” and growing “explosively”. In the meantime, as long as the data isn’t garbage-in, then there’s no garbage-out from deep learning. Hence the more data, the merrier. (Check out Trent McConaghy’s post on blockchains for AI as a solution for data to have reputation!).

Also, this “oil refinery” is improving on both software and hardware. Deep learning algorithms have improved over the past few decades and developers around the world have contributed to open source frameworks like TensorFlow, Theano, Keras, and Torch, all of which make it easy for people to build deep learning algorithms as if playing with LEGO pieces. And thanks to the demand from gamers around the world, GPUs (graphics processing units) make it possible for us to leverage deep learning algorithms to build and train models with impressive results in a time-efficient manner! So to all the parents who don’t like your kids playing games: Gaming has its silver lining…

Deep Learning: The Secret Recipe

You’ve probably read on the news and know that deep learning is the secret recipe behind many exciting developments and has made many of our wildest dreams and perhaps also nightmares come true. Who would have thought that DeepMind’s AlphaGo could defeat Lee Sedol, one of the best Go players, in the deepest board game which boasts more possible moves than there are atoms in the entire universe? A lot of people, including me, never saw it coming. It seemed impossible. But it’s here now. Deep learning is beating us in the most challenging board game. When is the AI awakening? Some think it will be soon.



Lee Sedol vs. AlphaGo in 2016, Source: The New Yorker

And we haven't talked about the other impressive applications powered by deep learning, such as Google Translate and Mobileye's autonomous driving. You name it. Did I forget to mention that deep learning is also beating physicians at diagnosing cancer? Deep learning excels at many tasks with lower error rates than humans! It's not just automating the boring stuff, but also the fun stuff. Sigh, we mundane humans…

> *Dear Mr. President, it's not foreigners. It's automation.*

Here's a short list of general tasks that deep learning can perform in real situations:

1. Identify faces (or more generally image categorization)
2. Read handwritten digits and texts
3. Recognize speech (no more transcribing interviews yourself)

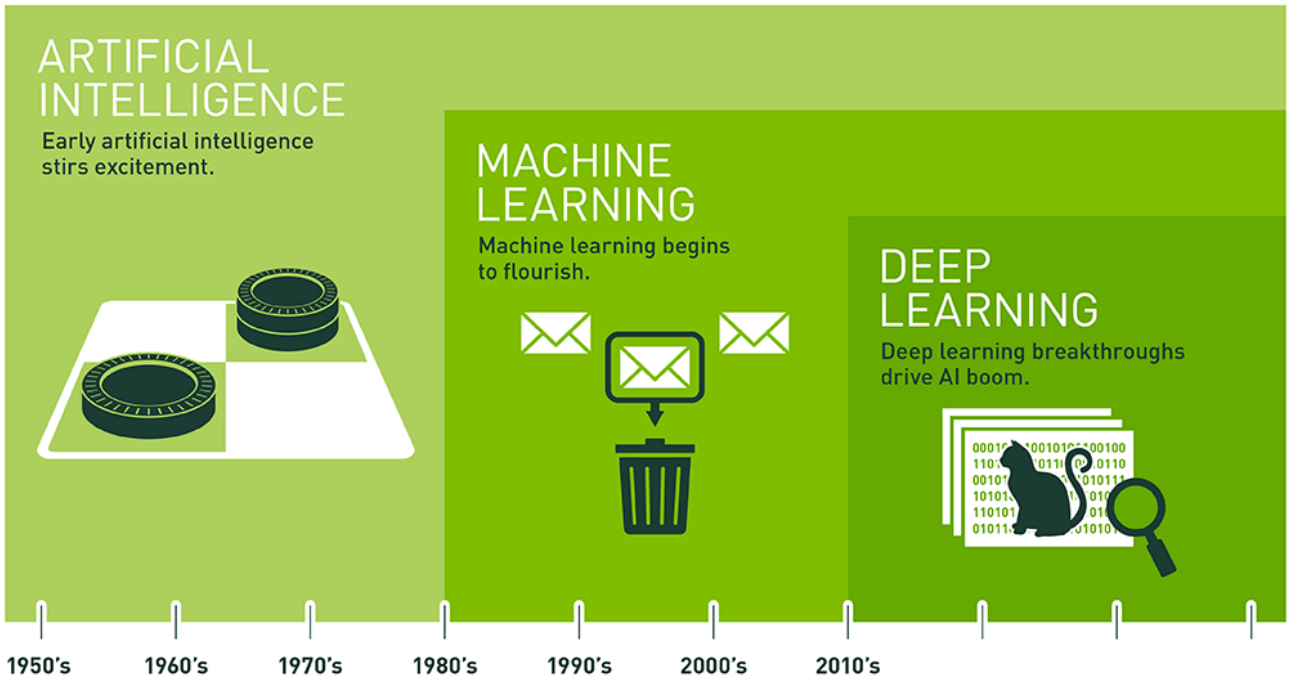
6. Control self-driving cars (and other types of robots)

And there's more. Just pause for a second and imagine all the things that deep learning could achieve. It's amazing and perhaps a bit scary!

Distinctions: AI, Machine Learning, and Deep Learning

Okay, wait a second. You've probably seen terms like Artificial Intelligence (**AI**), Machine Learning (**ML**), and Deep Learning (**DL**) flying all over the place in your social media newsfeed. What's the difference between all of them? Do they mean the same thing or what? Great question. Before we go deeper into deep learning, it is important to gradually build a conceptual framework of these jargons.

Roughly speaking, the graph below demonstrates the relationship for these 3 concepts. Deep learning is a subfield of Machine Learning, and Machine Learning is a subfield of Artificial Intelligence. Both Ophir Samson and Carlos E. Perez have written good stories about them. Check here and here.



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then

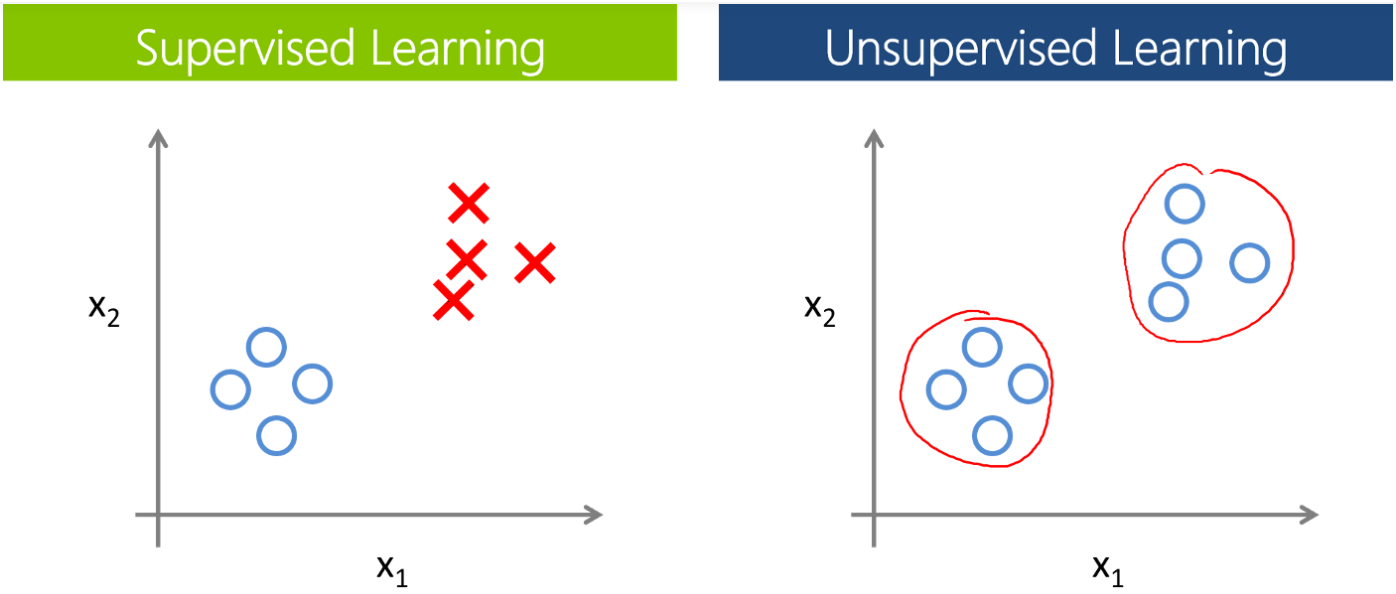
Let's discuss the all encompassing AI first. You probably know the **Turing Test** already. A computer passes the Turing Test if a human, after posing some written questions to the computer, cannot tell whether the written responses come from another human or the computer. According to *Artificial Intelligence: A Modern Approach*, Peter Norvig and Stuart Russell define the 4 capabilities a computer must command in order to pass the Turing Test:

- **Natural Language Processing:** to communicate successfully in English
- **Knowledge Representation:** to store what the computer reads
- **Automated Reasoning:** to use the stored knowledge to answer questions and draw new conclusions
- **Machine Learning:** to adapt to new circumstances and to identify new patterns

Ah, there's the term "machine learning"! ML is about training the learning algorithms like Linear Regression, KNN, K-Means, Decision Trees, Random Forest, and SVM with datasets, so that the algorithms could learn to adapt to a new situation and find patterns that might be interesting and important. Again, ML is data-driven. A lot of strange terms for the learning algorithms? No worries, I don't know all of them either. So we'll learn together in the future.

For training ML, the dataset can be labeled, e.g. it comes with an "answer sheet", telling the computer what the right answer is, like which emails are spams and which are not. This is called a **supervised learning** and algorithms like Linear Regression and KNN are used for such supervised **regression** or **classification**. Other datasets might not be labeled, and you are literally telling the algorithm such as K-Means to **associate** or **cluster** patterns that it finds without any answer sheet. This is called **unsupervised learning**. Here's a great answer to supervised vs. unsupervised learning on Stack Overflow and here's also a post on supervised vs unsupervised from Olivia Klose's blog.





Source: http://oliviaklose.com/

Norvig and Russell also mention another test called the **Total Turing Test**, which further examines the computer's perceptual abilities with physical simulation. To pass this one, the computer needs:

- **Computer Vision:** to perceive objects in the surroundings
- **Robotics:** to manipulate objects and move around

So what about DL now? Remember the general tasks that deep learning is good at from the section above? Things like face or handwritten text recognition have to do with computer vision because you are feeding in graphics to the computer for analysis. Other tasks like language translation or speech recognition have to do with natural language processing (NLP). So DL is a sub-branch of ML in that it also has a set of learning algorithms that can train on and learn from data, and more specifically **DL is powered by neural networks**. Moreover, DL can perform outside the machine learning area and comes to assist other areas like computer vision and NLP so that hopefully AI could pass the Turing Test and Total Turing Test one day!

But what the heck is a neural network? Is it imitating the behaviors of actual neuron cells?

Half-way Bonus: Valuable Resources

Neural networks and DL are often hidden behind a mysterious veil. All the technical jargons related to the topic might make beginners deeply confused. Since DL will automate many tasks and replace many workers in the future, I personally believe it is important that we all keep an open mind and curiosity to learn new technology. DL can replace a worker who works on manual, repetitive tasks. But DL cannot replace the scientist or the engineer building and maintaining a DL application.

Currently there are already many great courses, tutorials, and books on the internet covering this topic, such as (not exhaustive or in specific order):

1. Michael Nielsen's Neural Networks and Deep Learning
2. Geoffrey Hinton's Neural Networks for Machine Learning
3. Goodfellow, Bengio, & Courville's Deep Learning
4. Ian Trask's Grokking Deep Learning,
5. Francois Chollet's Deep Learning with Python
6. Udacity's Deep Learning Nanodegree (not free but high quality)
7. Udemy's Deep Learning A-Z ($10–$15)
8. Stanford's CS231n and CS224n
9. Siraj Raval's YouTube channel

The list goes on and on. David Venturi has a post for freeCodeCamp that lists many more resources. Check it out here.

We're truly blessed in the age of self-education. By the way, have you heard about the high school student, Abu Qader, from Chicago? This kid taught himself machine learning and Tensorflow, a deep learning framework, and helped improve the diagnosis of breast cancer to 93%-99% real-time accuracy! He was featured on Google I/O 2017. Below is an

Source: Google

Perceptron: The Prelude of DL

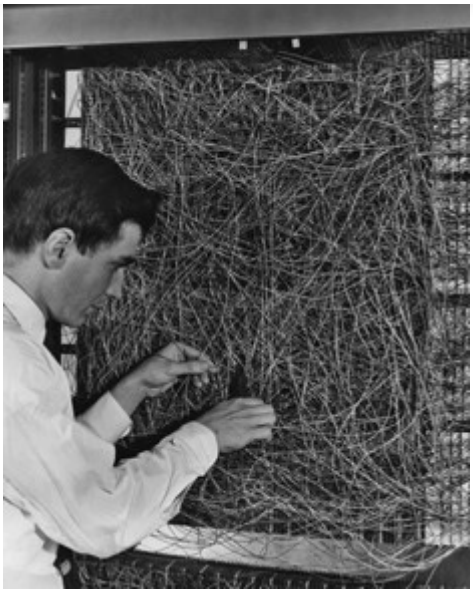
Alright, I hope Abu Qader's story makes you excited about learning! Let's get to the second main topic of this post: an introduction to neural networks. The ancient Chinese philosopher Lao Tzu once said:

> *"The journey of a thousand miles begins with one step."*

So we will begin and end this post with a very simple neural network. Sounds cool? Wunderbar.

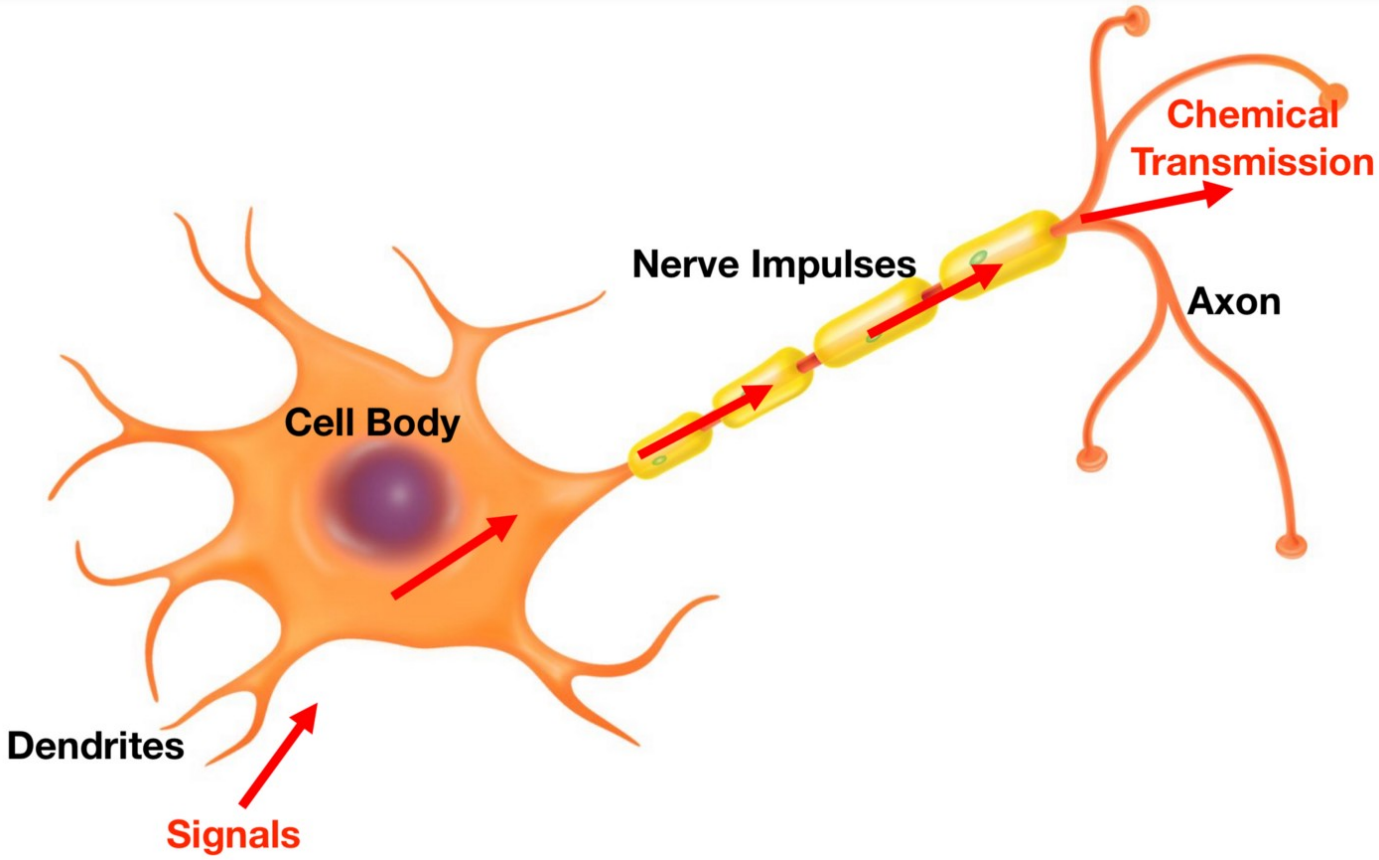
Despite its new-found fame, the field of neural networks isn't new at all. In 1958, Frank Rosenblatt, an American psychologist, attempted to build "a machine which senses, recognizes, remembers, and responds like the human mind" and called the machine a **Perceptron**. But Rosenblatt didn't invent perceptrons out of thin air. Actually, he stood on the shoulders of giants and was inspired by previous works from Warren McCulloch and Walter Pitts in the 1940s. Gosh, that makes neural networks or, more specifically perceptron, a dinosaur in this fast-changing world of technology.



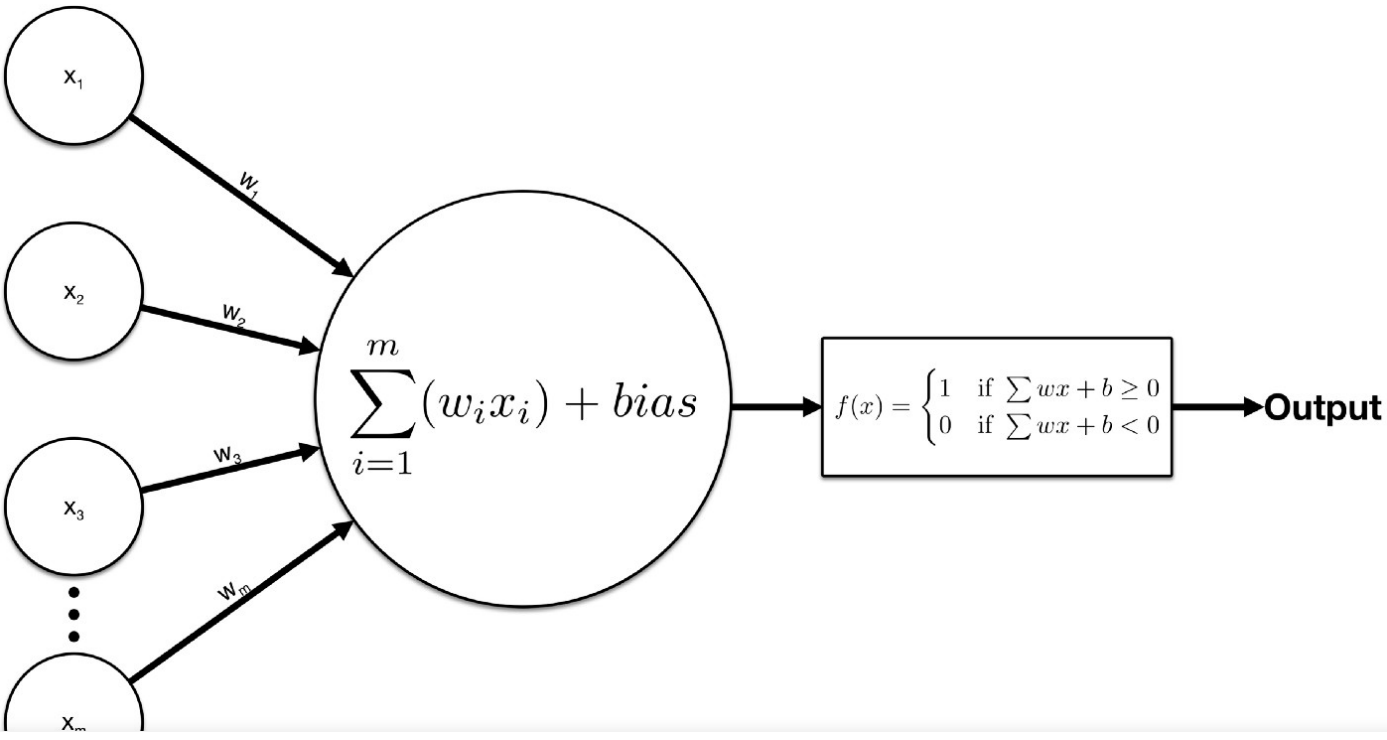


Rosenblatt and Perceptron, Source: The New Yorker

So let's see what a perceptron is. First, have a look at the neuron cell below: the **dendrites** are extensions of the nerve cell (in the left lower corner of the graph). They receive signals and then transmit the signals to the cell body, which processes the stimulus and decide whether to trigger signals to other neuron cells. In case this cell decides to trigger signals, the extension on the cell body called **axon** will triggers chemical transmission at the end of the axon to other cells. You don't have to memorize anything here. We are not studying neuroscience, so a vague impression of how it works will be enough.



Original Source: thinglink.com



Now, above is a graph of how a perceptron looks like. Pretty similar to the nerve cell graph above, right? Indeed. Perceptrons and other neural networks are inspired by real neurons in our brain. Note it's only *inspired* and does not work exactly like real neurons. The procedure of a perceptron processing data is as follows:

1. On the left side you have neurons (small circles) of *x* with subscripts *1, 2, … , m* carrying **data input.**
2. We multiply each of the input by a **weight** w, also labeled with subscripts *1, 2, …, m*, along the arrow (also called a **synapse**) to the big circle in the middle. So *w1 * x1*, *w2 * x2*, *w3 * x3* and so on.
3. Once all the the inputs are multiplied by a weight, we sum all of them up and add another pre-determined number called **bias.**
4. Then, we push the result further to the right. Now, we have this **step function** in the rectangle. What it means is that if the result from step 3 is any number equal or larger than 0, then we get 1 as output, otherwise if the result is smaller than 0, we get0 as output.
5. The **output** is either 1 or 0.

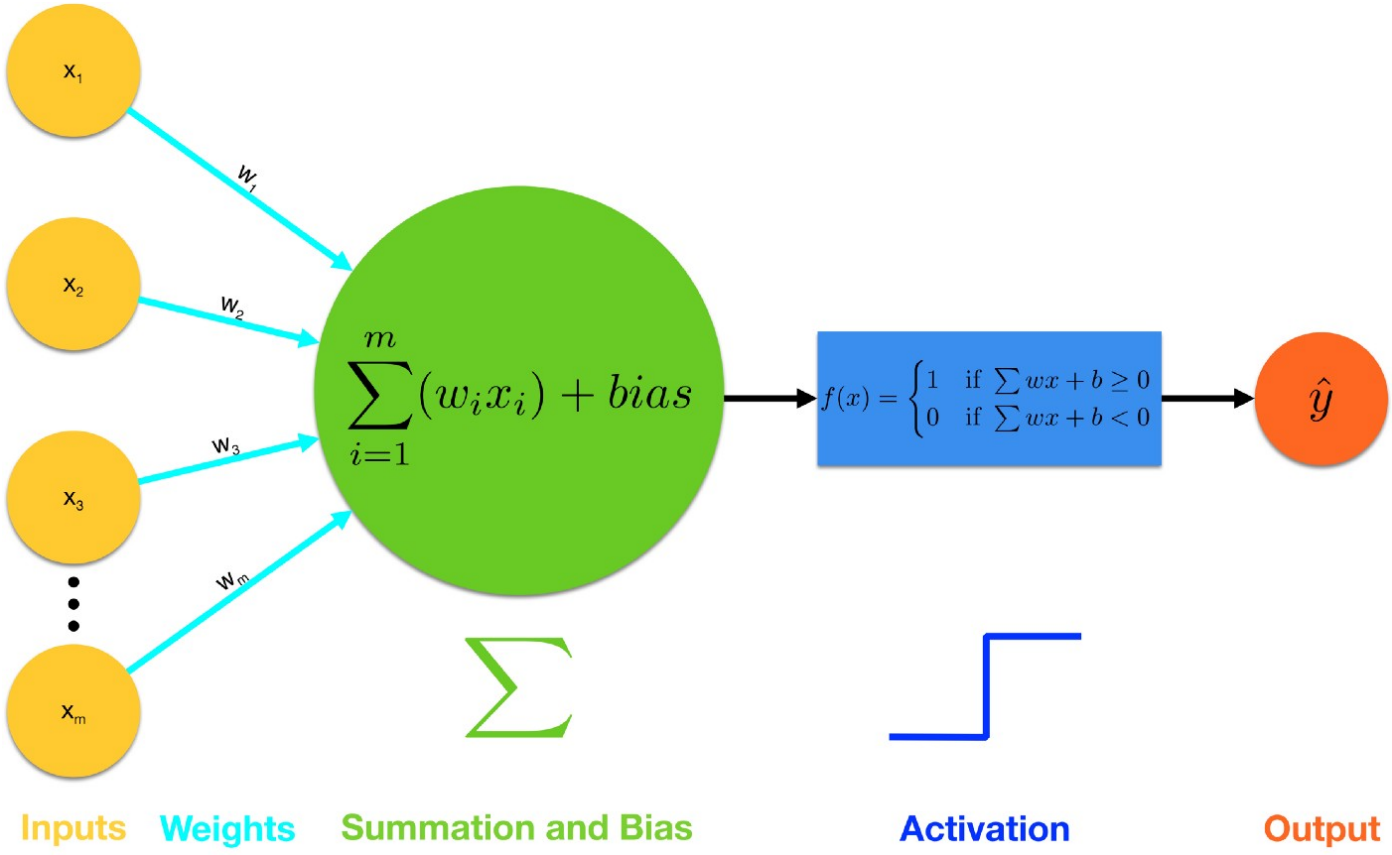
Note that alternatively, if you move bias to the right side of the equation in the activation function like *sum(wx) ≥ -b* then this *-b* is called a **threshold value**. So if the sum of the inputs and weights is greater than or equal to the threshold, then the activation triggers an 1. Otherwise, the activation outcome is 0. Choose whichever that helps you to understand better as these two ways of representation are interchangeable.

So instead of $\sum_{i=1}^{m} w_i x_i + bias$ with the activation function written as

$$output = f(x) = \begin{cases} 1 & \text{if } \sum w_i x_i + b \geq 0 \\ 0 & \text{if } \sum w_i x_i + b < 0 \end{cases}$$

We could also write $\sum_{i=1}^{m} w_i x_i = -bias = threshold$ with the activation function written as

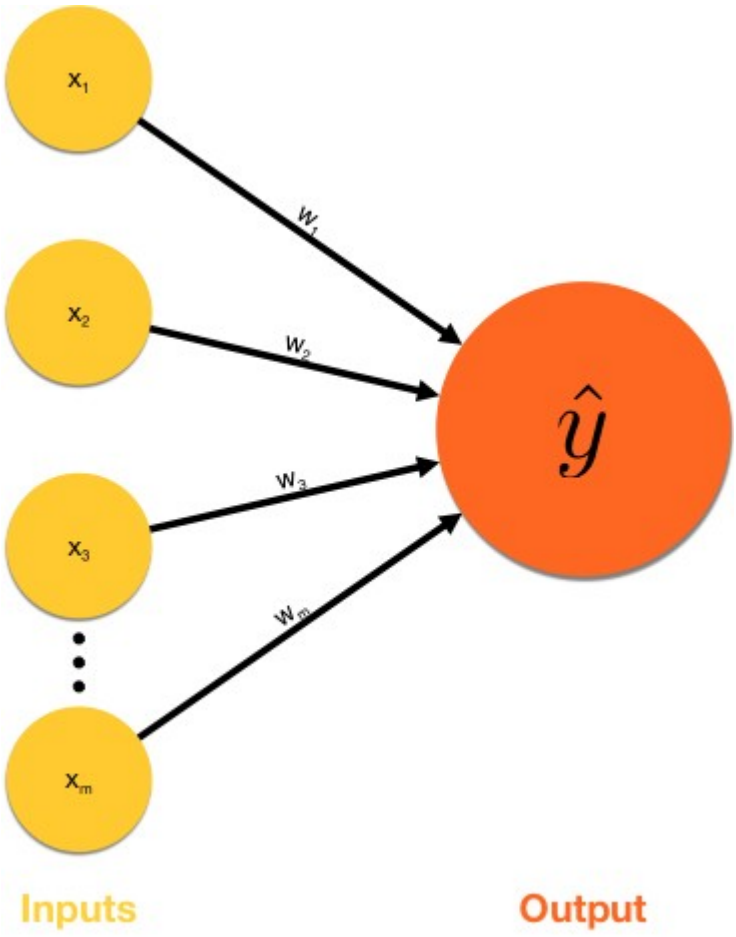
I add another perceptron graph below, this time with each step colored. It's very important that you fully understand it and remember what is happening at each step because we will ignore the middle steps in future graphs when we talk about more complicated neural network structures:



Procedures of a Perceptron labeled in colors

1. **Inputs** are fed into the perceptron
2. **Weights** are multiplied to each input
3. **Summation** and then add **bias**
4. **Activation function** is applied. Note that here we use a step function, but there are other more sophisticated activation functions like **sigmoid, hyperbolic tangent (*tanh*), rectifier (*relu*) and more**. No worries, we will cover many of them in the future!

In the future, we might sometimes simplify our perceptron as the following without mentioning steps 3 and 4. This type of perceptron we just talked about is also a **single-layer perceptron** since we process the inputs directly into an output without any more layers of neurons in the middle:



Simplified Representation of Perceptron

Perceptron: Intuition

Okay, you know how a perceptron works now. It's just some mechanical multiplications followed by summation and then some activation…voilà you get an output. Yeah, how on earth is this anything close to neurons in human brain?

For the sake of understanding perceptrons, let's look at a very easy example that isn't necessarily realistic. Suppose you are very motivated after reading my post and **you need**

2. Is the relevant mathematics and programming easy (Yes: 1, No: 0)
3. You can work on DL immediately without the need for an expensive GPU (Yes: 1, No: 0)

We use *x1, x2,* and *x3* as input variables for each of these factors and assign a binary value (1 or 0) to each of them since the answer is simply yes or no. Suppose you really like DL so far and you are willing to overcome your life-long fear of math and programming. And you also have some savings invest now in a pricey Nvidia GPU to train your DL models. Assume these two factors are equally less important as you can compromise on them. However, you really want to earn more money after spending so much time and energy in learning DL. So with a high expectation of return on investment, if you can't earn more $$$ afterwards, you won't waste your precious time on DL.

With an understanding of your decision-making preference, let's assume you have a 100% probability of earning more money after learning DL because there's a lot of demand for very little supply in the market. So $x1 = 1$. Let's say the math and programming is super hard. So $x2 = 0$. Finally, let's say you must have a powerful GPU like Titan X. So $x3 = 0$. Ok, we have the inputs ready and can also initialize the weights. Let's go for $w1 = 6$, $w2 = 2$, $w3 = 2$. **The larger the weight, the more influential the corresponding input is**. So since you value money the most for your decision to learn DL, $w1 > w2$ and $w1 > w3$.

We'll assume the threshold value $threshold = 5$, which is equivalent to say that the bias term $bias = -5$. We add it all up and plus the bias term. Check the following for the process of determining whether you will learn DL or not by using a perceptron.



So the summation is:

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$
$$= 6 * 1 + 0 * 2 + 0 * 2 - 5$$
$$= 6 - 5$$
$$= 1$$

According to our activation function below:

$$output = f(x) = \begin{cases} 1 & \text{if } \sum w_i x_i + b \geq 0 \\ 0 & \text{if } \sum w_i x_i + b < 0 \end{cases}$$

since $\sum w_i x_i + b = 1 > 0$, our output is 1 and the perceptron determines the desired decision that you will learn DL.

Note with a threshold value of 5, we will only learn deep learning if we earn more money. Even if both the math is easy (*x2 = 1*) and you don't need to spend money buying a GPU (*x3 = 1*), you will still not study DL if you can't earn more money later. See the illustration below:

The summation for $x_1 = 0, x_2 = 1, x_3 = 1$ is:

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$
$$= 6 * 0 + 1 * 2 + 1 * 2 - 5$$
$$= 4 - 5$$
$$= -1$$

Since $\sum = -1 < 0$, the output from activation is 0 and you won't learn DL.

Here's the fun part: varying the weights and threshold/bias will result in different possible decision-making models. So if, for instance, we lower the threshold from *threshold = 5* to *threshold = 3,* then we have more possible scenarios for the output to be 1. Now, the *bare minimum requirement* for *output = 1* is:

1. You will earn more money afterwards, so *x1 = 1* guarantees your decision to learn DL regardless of the values to *x2* and *x3*
2. Or, the math is easy and you don't need to buy GPU, so *x2 = x3 = 1* also guarantees that you decide to learn DL regardless of the value to *x1*

How so? You probably know already ;) Below is the explanation:

This is because with $x_1 = 0,$ and $x_2 = x_3 = 1$, our summation with bias would be:

$$\begin{aligned}\sum_{i=1}^{m} w_i x_i + bias &= w_1 x_1 + w_2 x_2 + w_3 x_3 + bias \\ &= 6 * 0 + 2 * 1 + 2 * 1 - 3 \\ &= 4 - 3 \\ &= 1\end{aligned}$$

So given that $\sum wx + b > 0$, the output is 1 and you will learn DL despite that you won't earn more money.

Yup. The threshold is lower now so the other 2 factors could motivate you to learn DL even if your prospect of earning more money was gone. Now, I encourage you to play around with the weights *w1, w2, and w3* and see how your decision on learning DL will change accordingly!

Perceptron: Learning in Action

Why should you play around with the weights on this example? Because it helps you to understand how perceptron learns. Now, we'll use this example together with the inputs and the weights to illustrate the single layer perceptron and see what it can achieve

In a real DL model, we are given input data, which we can't change. Meanwhile, the bias term is initialized before you train your neural networks model. So suppose we assume the bias is 7. Now, let's assume the following input data so that (1). you will earn more money, (2). math and programming for DL will be hard, and (3). yes you have to spend $1400 for a GPU to work on DL, and *most importantly,* ***we assume that you actually want to learn deep learning,*** *which we'll name as the* ***desired output*** *for how ideally the perceptron should correctly predict or determine*:

$$x_1 = 1, x_2 = 0, x_3 = 0, bias = 7$$

Let's further assume that our weights are initialized at the following:

$$w_1 = 6, w_2 = 2, w_3 = 2$$

So with input data, bias, and the output label (desired output):

$x_1 = 1, x_2 = 0, x_3 = 0$
$w_1 = 6, w_2 = 2, w_3 = 2$
desired output $= 1$ since you know you want to learn DL
Now we run our perceptron network once below:

$$\begin{aligned}\sum_{i=1}^{m} w_i x_i + bias &= w_1 x_1 + w_2 x_2 + w_3 x_3 + bias \\ &= 6 * 1 + 2 * 0 + 2 * 0 - 7 \\ &= 6 - 7 \\ &= -1\end{aligned}$$

So given that $\sum w_i x_i + b = -1 < 0$, our *actual output* $= 0$ while it should

and improve, given this difference between actual and desired output? Yup, we can't change input data, and we have initialized our bias now. So the only thing we can do is that we can tell the perceptron to adjust the weights! If we tell the perceptron to increase *w1* to 7, without changing *w2* and *w3*, then:

We have a new value for $w_1 : w_1^{'} = 7$
$w_1^{'} = 7, w_2 = 2, w_3 = 2$
Now we run our perceptron network again below:

$$\begin{aligned}\sum_{i=1}^{m} w_i x_i + bias &= w_1^{'} x_1 + w_2 x_2 + w_3 x_3 + bias \\ &= 7 * 1 + 2 * 0 + 2 * 0 - 7 \\ &= 7 - 7 \\ &= 0\end{aligned}$$

So given that $\sum w_i x_i + b = -1 \geq 0$, our *actual output* $= 1$ as our desired output of 1. So with this weight adjustment, the perceptron successfully determines that you will learn DL.

Adjusting the weights is the key to the learning process of our perceptron. And single-layer perceptron with step function can utilize the learning algorithm listed below to tune the weights after processing each set of input data. Try updating the weights with this algorithm yourself :) This is pretty much how a single-layer perceptron learns.

$$f(x) = \begin{cases} 1 & \text{if } \sum w_i x_i + b \geq 0 \\ 0 & \text{if } \sum w_i x_i + b < 0 \end{cases}$$

$$\text{Learning algorithm: } \quad w_i = w_i + (y - \hat{y}) x_i$$

need a GPU. However, you can use cloud services like AWS, Floyd, and possibly Google TPU) instead of a real GPU when you start training larger datasets with a lot of image files.

Also, the math for DL isn't easy but also not insurmountable. Mostly, we will just encounter some matrix operations and basic calculus. But remember, nothing that makes you stand out is easy to learn. Here's a quote from *Talent is Overated*:

> *"Doing things we know how to do well is enjoyable, and that's exactly the opposite of what deliberate practice demands. Instead of doing what we're good at, we should insistently seek out what we're not good at. Then we identify the painful, difficult activities that will make us better and do those things over and over. If the activities that lead to greatness were easy and fun, then everyone would do them and they would not distinguish the best from the rest."*

So to those of you who are scared of math and programming like I used to be, I hope this quote gives you some courage to keep learning and practicing :)

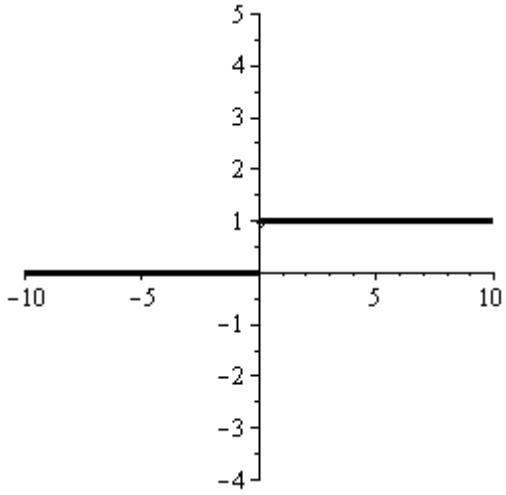
Perceptron: Limitations

Despite some early sensations from the public, perceptron's popularity faded away quietly because of its limitations. In 1969, Marvin Minsky and Seymour Papert discussed these limitations, including perceptron's inability to learn an XOR (exclusive-or) gate (so basically a ***single-layer* perceptron** with step function cannot understand the logic that the weather has to be either hot or cold, but not both). And these logic gates like AND, OR, NOT, XOR are very important concepts that are powering your computer ;) TutorialsPoint has a list of logic gates if you want to learn more. Check here.

Of course, later people realized that ***multi-layer* perceptrons** are capable of learning the logic of an XOR gate, but they require something called ***backpropagation*** for the network to learn from trials and errors. After all, remember that deep learning neural networks are data-driven. If we have a model and its actual output is different from the **desired output,** we need a way to back-propagate the error information along the neural network to tell weights to adjust and correct themselves by a certain value so that gradually the actual output from the model gets closer to the desired output after rounds and rounds of testing.

separable), step function won't work because it doesn't support backpropagation, which require the chosen activation function to have meaningful derivative.



Source: mathnotes.org

Some calculus talk: Step function is a **linear activation function** whose derivative is zero for all input points except the point zero. At point zero, the derivative is undefined since the function is discontinuous at point zero. So although it is a very simple and easy activation function, it cannot handle more complicated tasks. Keep reading and you'll find out more.

Linear vs. Non-linear

What!? So what is a **linear combination**? And why can't a perceptron learn an XOR gate? This is getting confusing now. No problem, here's an explanation:

So a linear combination of inputs x_1, x_2, x_3 up to x_m is just to multiply each of these inputs with a constant w_i:

$$w_1x_1 + w_2x_2 + w_3x_3 + ... + w_mx_m$$

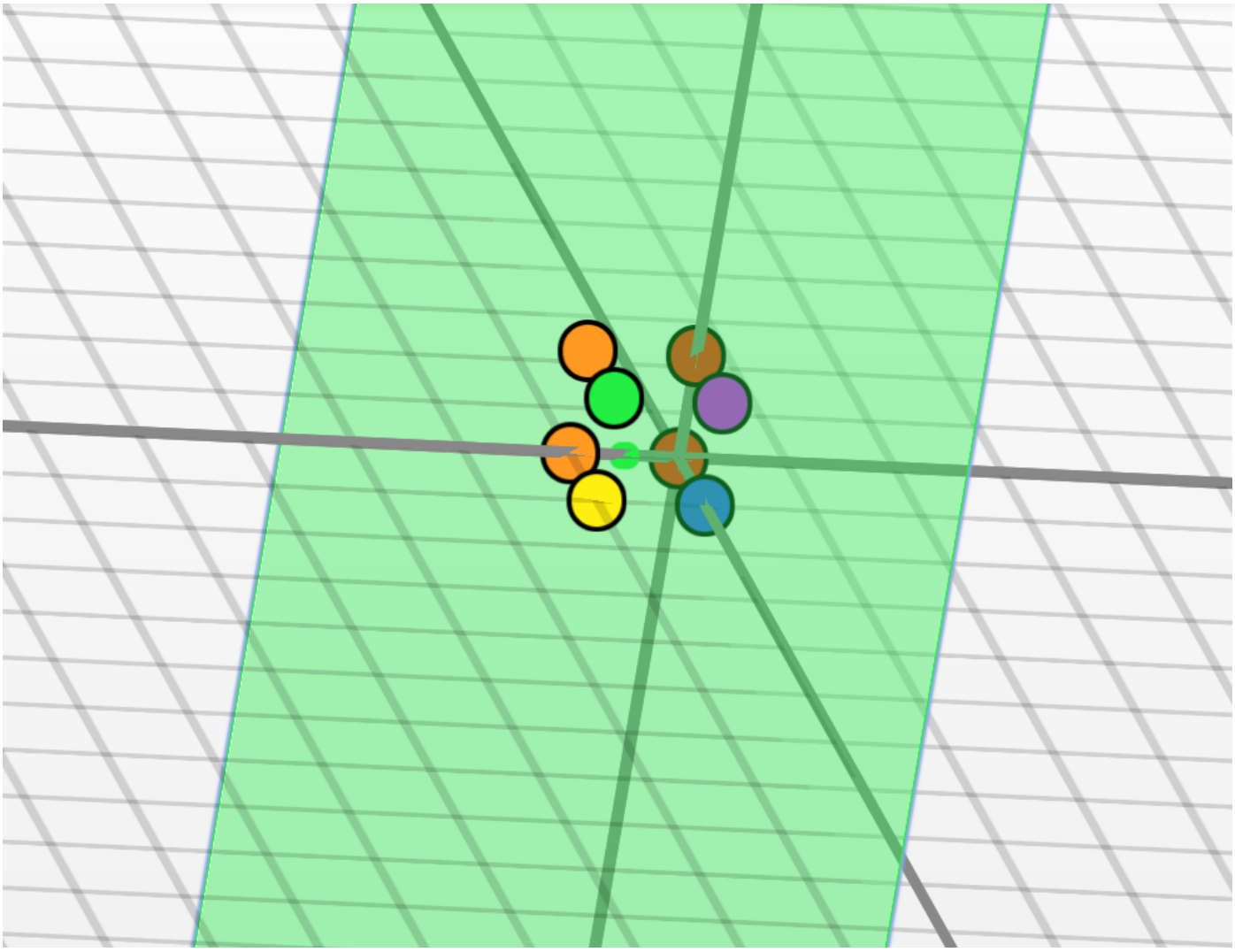
Think for a moment about our previous example. With 3 binary inputs for whether you

1. (1, 0, 0) -> sum + bias = 6–5 = 1, desired output = 1
2. (1, 1, 0) -> sum + bias = 8–5 = 3, desired output = 1
3. (1, 1, 1) -> sum + bias = 10–5 = 5, desired output = 1
4. (1, 0, 1) -> sum + bias = 8–5 = 3, desired output = 1
5. (0, 1, 1) -> sum + bias = 4–5 = -1, desired output = 0
6. (0, 1, 0) -> sum + bias = 2–5 = -3, desired output = 0
7. (0, 0, 1) -> sum + bias = 2–5 = -3, desired output = 0
8. (0, 0, 0) -> sum + bias = 0–5 = -5, desired output = 0

So literally, if we start with random weights than the ones of (*w1 = 6, w2 = 2, w3 = 2*), our perceptron will try to learn to adjust and find the ideal weights (*w1 = 6, w2 = 2, w3 = 2*), which would correctly match the actual output of each set of (*x1, x2, x3*) to the desired output. In fact, we can separate these 8 possible sets in a 3D space with a plane, such as the plane of *x1 = 0.5*. This type of classification problem where you can draw a plane to separate different outputs (or draw a line for outputs in 2D) is a problem that our single-layer perceptron can solve.

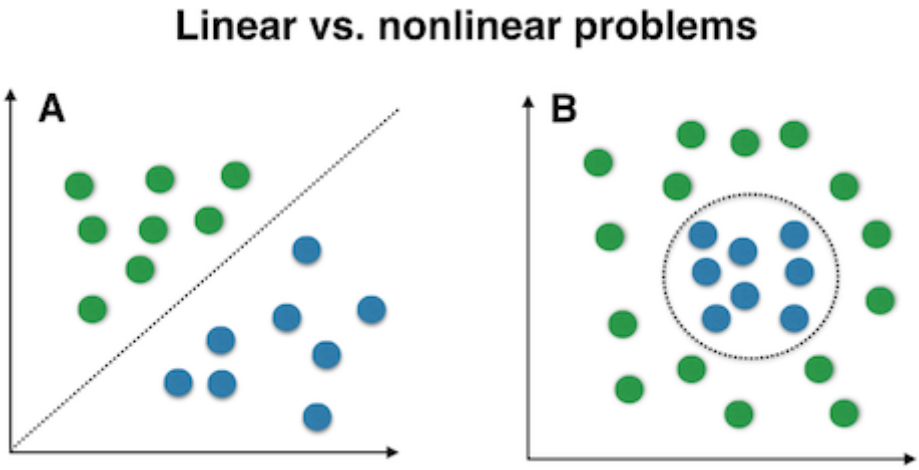


A plane separating the 4 sets to the left from the 4 to the right



A Plane separating the 4 sets to the left from the 4 to the right

I hope you can imagine the plane separating the 8 sets of inputs above. Since the sets of (*x1, x2, x3*) involves a 3-dimensional space, it can be a bit challenging. But in general, a single-layer perceptron with a linear activation function can learn to separate a dataset like the chart A in the graph below, which is separable by a line of $y = ax + b$. But if the dataset is only separable non-linearly by a circle like in chart B, our perceptron will perform horribly.

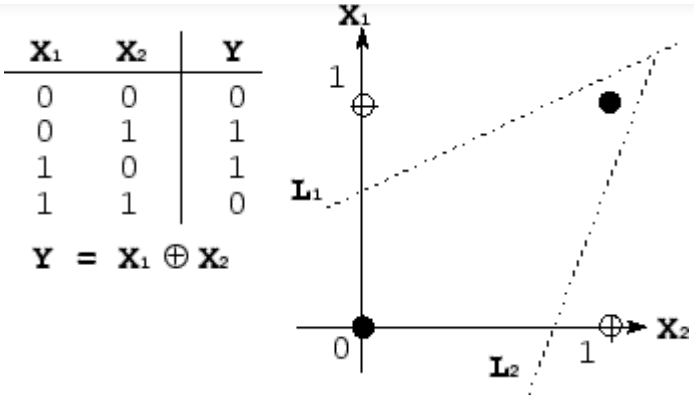


Source: Sebastian Raschka

Why is it so? I copy paste a quote from Stack Overflow for you. You can check out the answer here as well.

> *"Activation functions cannot be linear because neural networks with a linear activation function are effective only one layer deep, regardless of how complex their architecture are. Input to networks are usually linear transformation (input * weight), but real world and problems are non-linear. To make the incoming data nonlinear, we use nonlinear mapping called activation function. An activation function is a decision making function that determines the presence of particular neural feature. It is mapped between 0 and 1, where zero mean the feature is not there, while one means the feature is present. Unfortunately, the small changes occurring in the weights cannot be reflected in the activation value because it can only take either 0 or 1. Therefore, nonlinear functions must be continuous and differentiable between this range. A neural network must be able to take any input from -infinity to +infinite, but it should be able to map it to an output that ranges between {0,1} or between {-1,1} in some cases — thus the need for activation function. Non-linearity is needed in activation functions because its aim in a neural network is to produce a nonlinear decision boundary via non-linear combinations of the weight and inputs." -user7479*

So why can't a single-layer perceptron with step function learn the XOR gate? Check out the graph below. On the left is an XOR logic chart, and on the right is a cartesian representation of the 2 different outcomes (1 and 0) from an XOR gate.



Source: https://stackoverflow.com/a/35919708/6297414

Indeed, we cannot possibly separate the white dots from the black dots with a single, linear line. We need to have something more powerful to let a neural network learn the XOR gate logic. I will write more on this topic soon.

Recap

Deep learning is an exciting field that is rapidly changing our society. We should care about deep learning and it is fun to understand at least the basics of it. We also introduced a very basic neural network called (single-layer) perceptron and learned about how the decision-making model of perceptron works.

Our single-layer perceptron plus a step function works fine for simple linearly-separable binary classification. But that's about it…as it won't work very well for more complicated problems involving non-linear outputs. I know this sounds a bit disappointing, but we will cover them in detail next!

Coming next will be a new post on neural networks with **hidden layers** (sounds fancy right?) and a new activation function called the **sigmoid function**. If space allows, we will touch on **gradient descent** and **backpropagation,** which are the key concepts for a smart neural network to learn. Stay tuned (here's the link to the second post) :)

For now, congratulations on following the post so far and you've already learned a lot now! Stay motivated and I hope you are having fun with deep learning! If you are impatient to wait for my writing, check out the following free resources to learn more

We're focusing on handwriting recognition because it's an excellent prototype problem for learning about neural...

neuralnetworksanddeeplearning.com

CS231n Convolutional Neural Networks for Visual Recognition

Course materials and notes for Stanford class CS231n: Convolutional Neural Networks for Visual Recognition.

cs231n.github.io

Enjoy learning!

Did you enjoy this reading? Don't forget to follow me on Twitter!