My Vector Learning AI Model

1. Why vectors?
2. Data as vectors
3. My model

1. Why vectors?

When ChatGPT came out I understood that even **language**, something I thought was not mathematically “solvable”, **had a mathematical nature** that could be predicted and **presented**.

So where can this mathematical nature be found? ChatGPT uses a neural network -model to find it. The working principle of neural networks isn’t complicated. It looks for **regularities in data**, by **adjusting weights** between data and nodes in a way, that the algorithm finds out the wanted result. More data means more adjusting, more adjusting means more accuracy.

Neural networks -models aren’t useful for just language based models. They can be trained to find symmetries from all kinds of data such as visual and sound input.

Three Blue One Brown has a great video on neural networks, in which he explains that neural networks can be presented using vectors and matrices. This is where I got my idea.

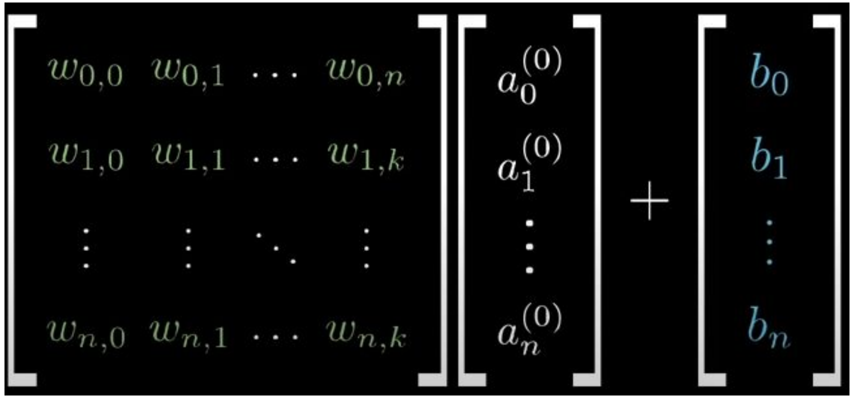
2. Data as vectors

On the right you see the two formulas that Thee Blue One Brown showed on his video. The top one contains from the left weight, input and bias.

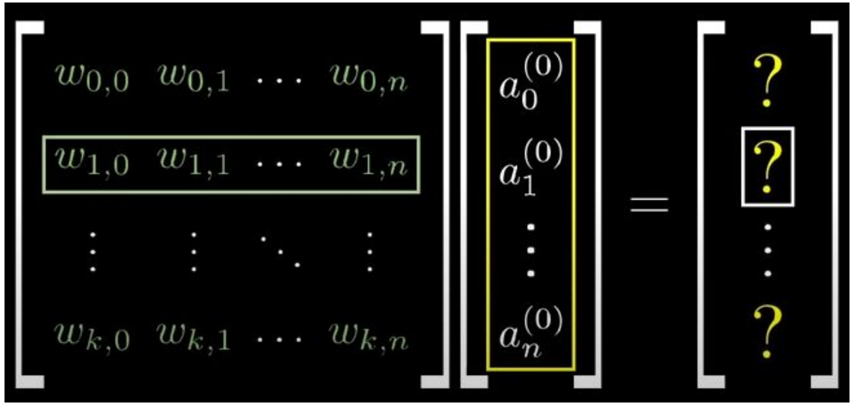
- Weight calculates how much a certain data “matters” for the result.
- The input is self explanatory. It is the data.
- Bias is what we add to the formula to guide the algorithm to the result we want.

When the model was presented to me like this, I started thinking about these algorithms from the point of view of geometry. Any input is just a vector in space and the weight matrix is a matrix transformation, which changes the vector in a way that gives us the wanted result, as is presented in the bottom image. The bias in this example would be like a starting nudge for the input vector into a certain direction.

Of course imaging these vectors in truth is not realistic, as we are talking about multi dimensional vectors that can’t possible be understood. But still, the logic doesn’t change.



$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,0} & w_{n,1} & \cdots & w_{n,k} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$



$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ \boxed{w_{1,0} \quad w_{1,1} \quad \cdots \quad w_{1,n}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \boxed{\begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix}} = \begin{bmatrix} ? \\ \boxed{?} \\ \vdots \\ ? \end{bmatrix}$$

3. My model

Now that I had the image in my head I started thinking: “What are the axis of these vectors?”

Data. Any set of information can be put into a vector, and as long as each axis of data is from the same system, the relations can be found, if they exist. The hard part was figuring out when a data can be considered to be from the same system.

To explain the logic I’ll use Neural Numbers as an example. In Neural Numbers a neural network model is trained to identify handwritten numbers between 0-9 from a 28-by-28 pixel grid. Each of the 676 pixels is an input of a number between 0 and 1 which shows how white a pixel is. The model is to be trained to guess correctly which number the pixels represent. So the output should be a single number between 0-9.

Here we can think each input pixel as an axis on a vector. We get a 676-dimensional vector that points to a point in a 676-dimensional space. The idea for my model is to make it figure out which points inside an area in that 676-dimensional space qualify for which numbers.

My model for this task should have a transformation matrix that turns 676-dimensional vectors to digits. So the matrix should be of shape 676x1. So really the matrix would just be a vector. Each axis in this transformation matrix would be a weight that has been adjusted during learning. The math would go down in a way that we end up with a close approximation to an integer, which is then round up to the closest integer.