Vectors in Neural Networks

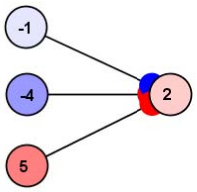
Any list of numbers can be thought of as a vector. List vectors can be represented using parenthesized, comma-separated lists. Here are some examples:

(0, 0); **(0, 1);** **(1, 0);** **(1, 1)**

The numbers inside a vector are called its "**components**". The number of components a vector has corresponds to the dimension of the vector space it is part of. So the vectors above are members of (or points in) a 2-dimensional vector space. Here are some vectors from a 5-dimensional vector space:

(0,-1, 1, .4, 9); **(-1, 2, 4, -3, 9);** **(0, 0, 0, -1, -1);** **(0, -1, 0, -1, 0)**

Typically in neural networks, lists of activation values, weight strengths and input values are treated as vectors. A subset of a neural network's nodes can be described by an activation vector, each component of which corresponds to the activation of one of those nodes. For example, we can describe this network's activation by **(-1,-4, 5, 2)**.

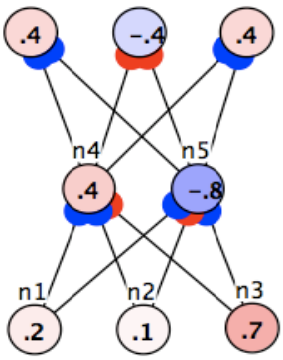


In addition to describing the state of all of a network's nodes by an activation vector, we can describe a subset of its nodes using an activation vector. Often these correspond to "**layers**." Though there is no formal definition of a layer that I know of, they are usually a subset of a network's nodes that are **functionally significant** as a group. For example, in a feed-forward network each layer is a set of neurons that collectively receive and/ or send connections to other layers, and that are not connected to each other.

In the network above there is an "**input layer**" with three neurons, whose state is described by an input vector **(-1, -4, 5).** The "**output layer**" on the right is a trival layer, with one neuron, represented by a vector with one component: **(2).**

Let us consider the following network. You can load it, performing the following steps:

- Go to **File --> Open network** in the **"labs"**-folder
- Select "**VectorNN.xml**".



The bottom row of nodes **n1 - n3** is an input layer, the middle row of nodes **n4, n5** is a "**hidden layer**", and the upper row is an output layer. The weights are already instantiated. Try to enter the input activations by double clicking nodes 1 to 3 and changing their activation from **(0, 0, 0)** to **(0.2, 0.1, 0.7)** each**.** Now iterate the network to see the resulting activity in the hidden and the output layer. What vectors do they constitute?

The set of weights fanning in to and out of a node comprise **weight vectors**. For instance, in the three layer network above, all the weights are either **1** or **-1**, indicated by the red or blue disks. Try now to "extract" the weight values by visual inspection and write them down as vectors $\mathbf{n4_{in} = (w_{14}, w_{24}, w_{34},)}$, $\mathbf{n4_{out} = (w_{46}, w_{47}, w_{48})}$, $\mathbf{n5_{in} = (...)}$, and $\mathbf{n5_{out} = (...)}$ respectively.

Vectors can also be classified according to other properties, which will be important when we consider pattern recognition and learning later. Here are some common types.

- A **binary vector** is a vector, all of whose components are **0** or **1**. Some examples:
  (0, 0, 1, 0); **(1, 1, 0, 0);** **(1, 1, 1, 1).**
- A **bipolar vector** is a vector, all of whose components are **-1** or **1.** Here are some examples:
  (-1, -1, 1, -1); **(1, 1, -1, -1);** **(1,1,1,1)**.

The dot product

One simple and important operation that can be performed on vectors is that of taking their **dot product** (or "scalar product"; the concept of an "inner product" is related but more general). We will represent this by a dot symbol, "·". The dot product is obtained by multiplying the pairwise components of two vectors and adding the results. Of course, this requires that the two vectors have the same number of components. For example

$$\mathbf{(1,2,3) \cdot (4,5,6) = (1 * 4) + (2 * 5) + (3 * 6) = 32}$$

Exercise: Compute the dot products of these vectors by instantiating them in an appropriate network (**3-to-1** and **5-to-1** respectively):

$$\mathbf{(0,0,0) \cdot (4,5,6) =}$$

$$\mathbf{(1,1,-1) \cdot (4,5,6) =}$$

$$\mathbf{(1,1,1,1,1) \cdot (1,1,1,1,1) =}$$

$$\mathbf{(0,1,0,1,0) \cdot (1,0,1,0,1) =}$$

Vector-valued Functions

One way of looking at the internal processing of a neural network is as computing a series of vector to vector transformations based on the intervening weights (which, we will see, can be represented by "weight matrices"). In fact, the operation of the brain as a whole can be thought of as a massive series of vector to vector transformations mediated in part by weight matrices.

These transformations can be thought of as in terms of **vector valued functions**. Recall from algebra that a function is a rule which associates objects in a domain with unique objects in a range. For example **$f(x) = x^2$** associates real numbers with positive real numbers**: f(2) = 4, f(-1) = 1**, and **f(0) = 0**. We can think of feed-forward neural networks as computing functions, which associate input vectors with output vectors.

As the weights in this network change, the way it associates input vectors with output vectors changes. That is, as the weights change, so too does the vector valued function the network as a whole computes. Learning (in the sense of weight change) can therefore be thought of as a way of producing specific vector-valued functions by modifying weight matrices. In **LAB 4** we will consider automatic procedures for updating weights to approximate desired functions. For now, we consider some very simple vector-valued functions.

BOOLEAN functions

A simple form of function to compute with a neural network is one which implements a **boolean function** which associates **truth values** (**1** for TRUE and **0** for FALSE) with other truth values. These functions can be represented by **truth tables**. Here is a truth table which represents three separate boolean functions: **AND**, **OR**, and **XOR**.

input		output		
P	**Q**	**P AND Q**	**P OR Q**	**P XOR Q**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

P could stand for "I ate pizza" and **Q** could stand for "I drank soda." The four rows of the table show all possible combinations of truth values for these two statements.

Let us now try to build a network capable of performing the logical **AND**. Do this by:

- Open up a fresh network window by pressing the network button.
- Create three nodes and arrange them in a way that two input nodes are feeding one output node.

- Connect the input nodes with the output node by selecting the input nodes and pressing **1** and then selecting the output node and pressing **2.**
- Double click on the output node and set the update rule to be "**binary**" with lower bound **0** and threshold **0.9**.
- Select all weights by pressing **W** and clear them by pressing **C**.
- Now try to find a combination of weights which reflect the AND-operation in the truth table above.

With this experience the following exercise should be quite easy.

- Try the same for the logical **OR**.
- What weights have you chosen?

From the course you already know, that the XOR-operation is a bit nastier. To spare you the efforts, it is actually unsolvable with such a network architecture.

- Insert an additional layer with two nodes and connect them as depicted.
- Set the new nodes to be binary with bounds **0** and **1** and a threshold of **0.5.**
- Try to figure out possible weights for this type of architecture so that the **XOR**-operation works.