Visualizing Small Neural Nets

Jake Barnwell, Jessica Noss, et al

October 2016

1 Background

Here we will review the concepts behind visualizing neurons as lines and shaded regions in a plane. This should mostly be a review from recitation. If you want to get straight to the explanations for the lab questions, you can skip this entire section.

1.1 A Single Neuron as a Geometric Line

Suppose we have a single neuron N with two inputs x and y with weights a and b, respectively. To determine the N's output, we compare its total *input*, in this case $ax + by$, against some `threshold_fn`.

For simplicity's sake, we can consider our threshold function to be the stairstep function, which outputs a 1 if the input is $\geq T$, or a 0 if the input is $< T$. So, our neuron N is just comparing $ax + by$ to T: if $ax + by \geq T$, the output of N is 1, otherwise, the output is 0. Now recall that x and y are input variables, and a, b, and T are defined constants. This means that the boundary $ax + by = T$ corresponds exactly to a *line in two dimensions*!

More generally, **any single neuron of two inputs draws some line on a plane**. To be a bit more technically accurate, we should say that a single neuron of two inputs corresponds to a *linear inequality*, that is, two *regions* (or zones) on a plane, separated by a line. One region represents all input data points that cause the neuron to output a 1, while the other region represents all input data points that cause the neuron to output a 0.

We typically refer to the region representing output = 1 as "the shaded region," since in diagrams we will shade the side of the line corresponding to an output of 1.

1.2 Combining Planar Regions with Logic

Once we know how to form one line on a plane, it's easy to do more complicated things. For example, we can draw *two* lines (which are just the boundaries of two linear inequalities), with each line separating the plane into two halves. Each of these linear inequalities shades one half of the plane denoting that the

neuron output is 1, while the other half of the plane denotes that the neuron output is 0.

But what makes this interesting is that

- two non-parallel lines in a plane will intersect once, separating the plane into four distinct regions; and,
- two parallel (but distinct) lines in a plane separate the plane into three distinct regions.

Let's consider the first case, where two neurons, call them NA and NB, draw two non-parallel lines A and B on a plane, splitting the plane into four distinct regions (zones). Then, we can ask the question, "What does this particular region represent?"

Below, we give a couple of simple examples. In each example, imagine that the zone in question is *shaded*, and that the other zones are *not* shaded:

- One of the four zones represents all data points (x, y) that would yield an output of 1 from *both* NA and NB: this is conceptualized as a logical `AND`.
- One of the four zones represents data points (x, y) that would yield an output of 0 from *both* NA and NB: this is conceptualized as a logical `NOR`.

There are, of course, more complicated zones. For example, one of the four zones contains all points that would yield an output of 1 from NA and an output of 0 from NB. Assuming that this zone was "shaded" and the others were not, this would technically represent the logical "Material Nonimpliciation function" for NA and NB.

In addition to asking about individual zones, we can ask about *subsets* of zones, i.e. combinations of individual zones. Here are some examples:

- Consider the union of the `AND` region with the `NOR` region, as explained above. This new combined region represents the logical `XNOR` of NA and NB. Note that these two regions are "opposite" each other in the plane.
- Consider the shaded complement of the `XNOR` region described above. By definition, this corresponds to the logical `XOR` function of NA and NB.
- Consider the union of the `XNOR` region with the `AND` region. This combined region represents the logical `OR` function of NA and NB.

This is by no means a comprehensive list, but these examples should give you an idea of how we can intuit the combination of linear inequalities.

1.3 A Neuron as a Logic Gate

We showed that a neuron can represent a line on a plane. But can we make a neuron to represent a logic gate? The answer is yes, sometimes!

In particular, some logical functions can be modeled by a single neuron; other logical functions will need more than one neuron. But all logical functions can be modelled by at most a couple of interacting neurons.

Let's start by considering an `AND` gate. We are going to construct a neuron N that takes in two *binary* inputs (0 or 1) and outputs the logical `AND` of the inputs (hence, N will *emulate* or *model* an `AND` gate). Let's call the inputs x and y, and assign a weight of 1 to each of the input wires. Then, let's define a threshold $T = 1.5$ for our neuron.

This completes the construction of the neuron N which *models* the logical `AND` gate. As proof, let's consider the universe of possible inputs and outputs. If $x = y = 1$, then the sum is 2 which is greater than T, so N outputs a 1. If either x or y is 0 (or both are 0), the sum of the inputs is either 1 or 0, neither of which is larger than T, so N outputs a 0. This aligns with the definition of an `AND` gate.

Similarly, an `OR` gate can be encoded into a single neuron by setting both input weights to 1, and having a threshold of $T = 0.5$. Check it out yourself!

We have just showed you how to construct *logical gates* using something as primitive as a neuron. Incredible!

By the way, most logic gates aren't this easy. Some, like `XOR` and `XNOR`, will need more than one neuron to emulate, as mentioned earlier.

2 Explanations for 6.034 2016 Lab 6 Visualizations of Neural Nets

2.1 `nn_half`: [1]

This one is pretty simple. A single line separates the plane into two regions. This can be accomplished, by definition, with a single neuron.

2.2 `nn_angle`: [2, 1]

We see that it takes two lines to form a zone like this. In particular, this can be considered the `AND` region between two lines (as long as you define the neurons so that the "direction" of the output = 1 zone is correct). An `AND` neuron, as explained earlier, can be constructed from a single neuron. Hence, we need one (input) neuron per line, and then the `AND` neuron in the next layer to compute the final result.

2.3 `nn_cross`: [2, 2, 1]

As usual, we'll need two neurons in the input layer to form the two lines. Then, we're going to need an `XOR` gate (or an `XNOR` gate, depending on which way your

line neurons are defined).

Constructing an `XOR` gate is fun. Recall that this particular `XOR` gate should take in two binary inputs, and output a 0 or 1 depending on if they represent an XOR or not. First of all, there are several ways to implement this gate, and in fact there are several ways to implement this gate using a [2, 1] structure (which is the simplest possible). Here we will describe two of these ways.

The first way:
We can conceptually define the `XOR` as follows: "If both inputs are 1, or if both inputs are 0, then this should output a 0." So let's make an `AND` node called $N1$ that outputs 1 if both inputs are 1, and let's make an `NOR` neuron called $N2$ that outputs a 1 if both inputs are 0. (By the way, a `NOR` neuron is just as simple to make as an `OR` neuron. Can you figure out how? You'll have to flip some signs.) Then, we will combine the outputs of $N1$ and $N2$ into another `NOR` neuron called $N3$ which will output a 0 if *either* input is a 1, or output a 1 if both inputs are 0.

The second way:
We can conceptually define the `XOR` as follows: "If either input is 1, *and* if it's the case that *both* inputs are *not* 1, then this should output a 1." So, we'll define a neuron $N1$ which is a standard `OR` neuron, and a neuron $N2$ which is a `NAND` neuron. Combine the outputs of $N1$ and $N2$ into a third neuron $N3$, which should be a an `AND` neuron, and we have our result.

2.4 `nn_stripe`: [3, 1]

Here we have three non-intersecting lines. Let's call the uppermost line A, the middle line B, and the lower-most line C. The graphical region presented can be translated into English as "the points (x, y) that are under A and above B; *or* that are under C."

Hopefully you can see where this is going. Define an `AND` neuron $N1$ (to define the region between A and B) in the second layer, and then define an `OR` neuron $N2$ in the third layer which takes $N1$ and the C-neuron as inputs, outputting a final result. And we're done...

...but wait! We just constructed a net of the form [2, 1, 1]. Is there a simpler one? Yes! It requires a bit of creativity though. Imagine that the weights of A and B outputs were 1 each, and the weight of the C output was 2. Can you think of a way to use these *three* outputs as inputs into a single neuron N in layer 2? Hint: The new neuron N is going to be similar to a 3-input `OR` neuron.

2.5 `nn_hexagon`: [6, 1]

This one should be pretty straight-forward. Six lines are required to draw this hexagon, and we just AND them all together with a seventh neuron.

2.6 `nn_grid`: [4, 2, 1]

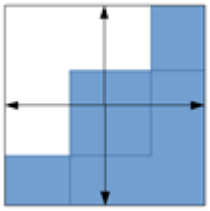
Again, there are multiple ways to do this one. We will just show one way.

First note that we will need four distinct lines to split the grid into nine regions. Call the two vertical line neurons VL and VR (for the *left* and *right* vertical lines, respectively). Similarly, call the two horizontal line neurons HU and HL (for *upper* and *lower*).

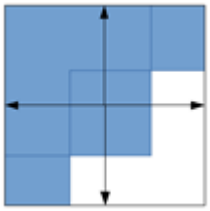
Set weights so that VL and VR each shade to their right (i.e. for more positive x), and that HU and HL shade below (i.e. for more negative y).

Define a four-input neuron SE (for *south-east*) in the second layer, taking in the outputs from all four line neurons. Additionally, define another four-input neuron NW (for *north-west*) in the second layer, which also takes in the outputs from all four line neurons.

- For SE, set all input weights to 1, and set the threshold to 1.5. This is testing that *at least* two inputs are shaded. This neuron should draw the following:



- For NW, set all input weights to -1, and set the threshold to -2.5. This is testing that *at most* two inputs are shaded. This neuron should draw the following:



Lastly, we combine NW and SE with an `AND` neuron in the third layer.