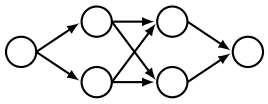
pgfplots

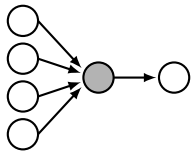
Chapter 13

Neural nets

Neural nets are functions loosely modeled on the brain. In the brain, we have billions of neurons that connect to one another. Each neuron can be thought of as a node in a graph, and the edges are the connections from one neuron to the next. The edges are directed because electrical signals propagate in just one direction along the wires in the brain:



Outgoing edges are called axons and incoming edges are called dendrites. A neuron "fires", sending a pulse down its axon, when the incoming pulses, from the dendrites, exceed a threshold. Let's consider a neuron, shaded in gray, that has four inputs and one output:



A simple model for this neuron is the **perceptron**. A perceptron is a neuron with N inputs $\{x_i\}_{i=1}^N$ and one output y, that maps inputs to outputs according to the following equations:

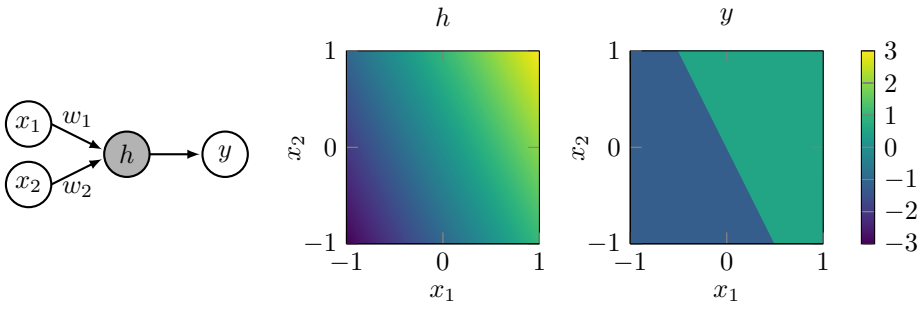
$$h = f(\mathbf{x}) = \sum_{i=1}^{N} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b \qquad \triangleleft \text{ linear layer} \qquad (13.1)$$

$$g(h) = \begin{cases} 1, & \text{if} \quad h > 0 \\ 0, & \text{otherwise} \end{cases} \qquad \triangleleft \text{ activation layer} \qquad (13.2)$$

$$y = g(f(\mathbf{x})) \qquad \triangleleft \text{ perceptron} \qquad (13.3)$$

In words, we take a weighted sum of the inputs and, if that sum exceeds a threshold, the neuron fires (outputs a 1).

People got excited about perceptrons in the late 1950s because it was shown that they can learn to classify data [Rosenblatt, 1957]. Let's see how that works. We will consider a perceptron with two inputs, x_1 and x_2, and one output, y. Let the incoming connection weights be $w_1 = 2$, $w_2 = 1$, and $b = 0$. h and y, as a function of x_1 and x_2, look like this:

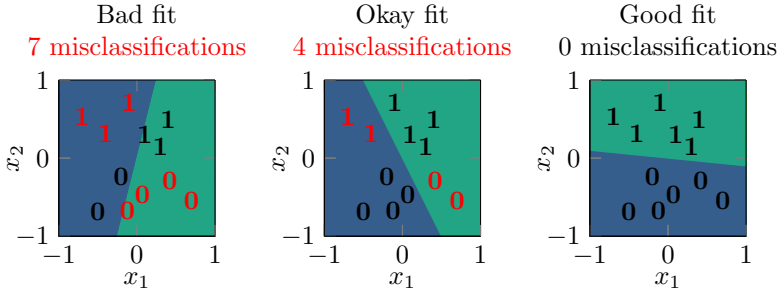


y takes on values 0 or 1, so you can think of this as a classifier that assigns a class label of 1 to the upper-right half of the plotted region.

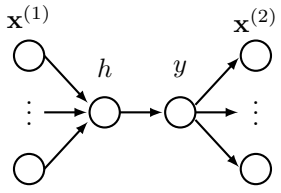
So a perceptron acts like a classifier, but how can we use it to learn? The idea is that given data, $\{\mathbf{x}^i, y^i\}_{i=1}^N$, we will adjust the weights $\mathbf{w}$ and the bias b, in order to minimize a classification loss, $\mathcal{L}$:

$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w},b} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\mathbf{w}^T \mathbf{x}^i + b, y^i) \tag{13.4}$$

In the picture above, this optimization process corresponds to shifting and rotating the classification boundary, until you find a line that separates data labeled as $y_i = 0$ from data labeled as $y_i = 1$:

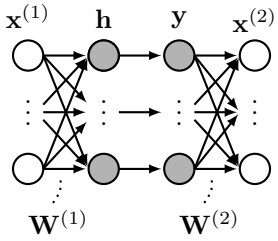


Perceptrons can solve linearly separable binary classification problems, but they are otherwise rather limited. For one, they only produce a single output. What if we want multiple outputs? We can achieve this by adding a edges that "fan out" after the perceptron:



This network maps an input **layer** of activations $\mathbf{x}$ to an output layer of activations $\mathbf{y}$. The neurons in between inputs and outputs are called **hidden units**.

More commonly we might have many hidden units in stack, which we call a **hidden layer**:



How many layers does this net have? Some texts will say two [$\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$], others three [$\mathbf{x}^{(1)}$, $\{\mathbf{h}, \mathbf{y}\}$, $\mathbf{x}^{(2)}$], others four [$\mathbf{x}^{(1)}$, $\mathbf{h}$, $\mathbf{y}$, $\mathbf{x}^{(2)}$]. We must get comfortable with the ambiguity.

The equation for this neural net is:

$$\mathbf{h} = \mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)} \tag{13.5}$$

$$\mathbf{y} = g(\mathbf{h}) \tag{13.6}$$

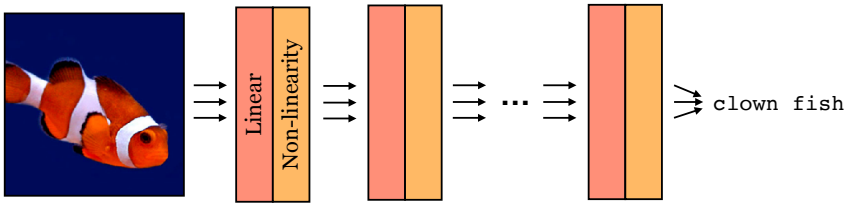
$$\mathbf{x}^{(2)} = \mathbf{W}^{(2)}\mathbf{y} + \mathbf{b}^{(2)} \tag{13.7}$$

where g is a pointwise nonlinearity, that is, $g(\mathbf{h}) = [g(h_1), \ldots, g(h_N)]$.

This kind of sequence – linear layer, pointwise nonlinearity, linear layer, pointwise nonlinearity, and so on – is the prototpyical motif in deep neural networks.

13.1 Deep nets

Deep nets are neural nets that stack the above motif many times:



Each layer is a function. Therefore, a deep net is a composition of many functions:

$$f(\mathbf{x}) = f^L(f^{L-1}(\ldots f^2(f^1(\mathbf{x})))) \tag{13.8}$$

These functions are parameterized by weights $[\mathbf{W}^1, \ldots, \mathbf{W}^L]$ and biases $[\mathbf{b}^1, \ldots, \mathbf{b}^L]$. Some layers we will see later have other parameters. Collectively, we will refer to the concatenation of all the parameters in a deep net as θ.

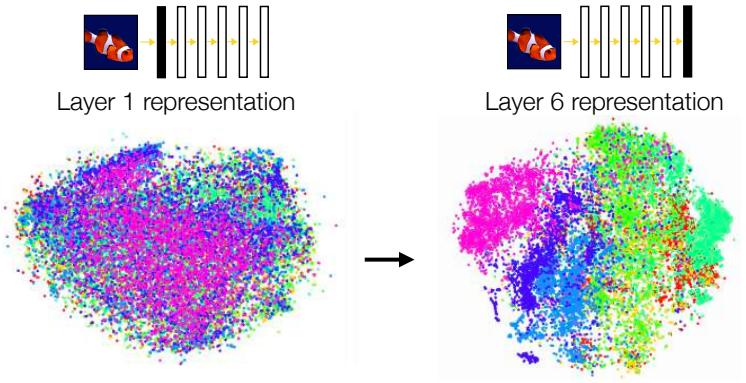
Deep learning refers to learning with deep nets. Using the formalism we defined in Chapter 12, learning consists of using an *optimizer* to find a function in a *hypothesis space*, that maximizes an *objective*. From this perspective, neural nets are simply a special kind of hypothesis space.

Deep learning also usually involves using a particular optimization algorithm called backpropagation, which we will see in the next chapter. However, it is certainly possible to optimize neural nets with other methods. Indeed, it not clear if biological neural nets actually use backprogation. Alternative optimizers include genetic algorithms and brute force search. There is also a broad literature on "bottom up learning rules" that do not explicitly optimize any objective (though they may implicitly). One such rule, called **Hebbian learning**, is "fire together, wire together", that is, we increase the weight of the connection between two neurons whenever the two neurons are active at the same time.

13.1.1 Representations in deep nets

Each layer in a deep net is a mapping from one representation of the data to another: $f : \mathbf{x}_{\texttt{in}} \rightarrow \mathbf{x}_{\texttt{out}}$. The data can be thought of as itself a representation. The target output, e.g., class labels, can be thought of as another representation. Then a classifier network, layer by layer, maps representations starting with the raw data to a representation that matches the labels. The sequence of representations for an input datapoint $\mathbf{x}^i_{\texttt{in}}$ is $\mathbf{x}^i_{\texttt{in}} \rightarrow \mathbf{h}^{(1)i} \rightarrow \mathbf{h}^{(2)i} \rightarrow \cdots \rightarrow \mathbf{x}^i_{\texttt{out}}$.

One way to characterize this sequence of representations is by looking at how each subsequent layer represents an entire *dataset*. For example, if the representation $\mathbf{h}^{(1)i}$ were a two-dimensional vector, then we could visualize this representation for all datapoints i in a dataset as a 2D scatter plot. If the next layer $\mathbf{h}^{(2)i}$ were also two-dimensional, then it would be a different scatter plot, with each datapoint shifted to a new position. Typically, each hidden layer is a high-dimensional vector, so it's non-trivial to visualize datapoints in this space. However, we may apply tools from **dimensionality reduction** to show a 2D projection of the data, such that the distance between two datapoints in the 2D projection is roughly proportional to their actual distance in the high-dimensional space. The below plot, reproduced from [Donahue et al.2014], uses a dimensionality reduction called t-SNE [Maaten and Hinton2008] to visualize how different layers of a deep net represent a dataset of images of different semantic classes:



Notice that on the first layer, semantic classes are not well separated but by layer 6 the representation has **disentangled** the semantic classes so that each class occupies a different part of representational space. This is expected because layer 6 is near the output of the network, and the output is being trained to be a one-hot representation of semantics – i.e. a completely disentangled representation.

Bibliography

[Donahue et al.2014] Donahue, Jeff, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, 647–655.

[Maaten and Hinton2008] Maaten, Laurens van der, and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research* 9 (Nov): 2579–2605.