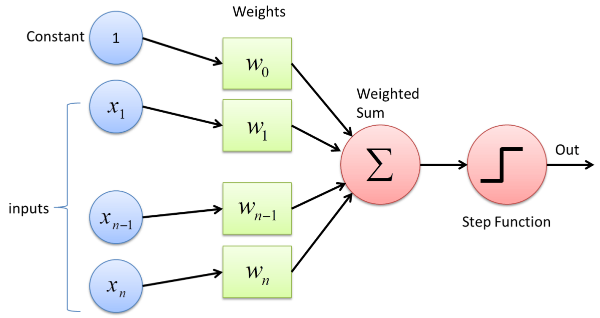
Physics 178/278 - David Kleinfeld - Winter 2020 - Week 10

Revised 9 March 2020 00:58

10 Layered networks

We consider feedforward networks. These may be considered as the "front end" of nervous system, such as retinal ganglia cells that feed into neurons in the thalamus for the case of vision, or touch receptors that feed into trigeminal second-order sensory neurons for the case of orofacial touch. Of course, feedforward circuits are of interest from an engineering perspective for sorting inputs or, more generally, statistical inference.

10.1 Perceptrons



We begin with the case of a single layer, the "Perceptron". We write the calculated output, denoted $\hat{y}$, as

$$\hat{y} = f\left[\sum_{j=1}^{N} W_j x_j \; - \; \theta\right] \; = \; f\left[\vec{W} \cdot \vec{x} - \theta\right] \tag{10.1}$$

where the x's are the inputs, in the form of an N-dimensional vector $\vec{x}$, "θ" is a threshold level, $f[\cdot]$ is a sigmoidal input-output function, and the W_j's are the weights to each of the inputs. We follow the scaling used throughout the feed forward network literature of choosing of $W \propto O\left(\frac{1}{N}\right)$, so that the sum is not scaled by $\frac{1}{N}$. One model for f is

$$f[x] = \frac{1}{1 + e^{-\beta x}} \tag{10.2}$$

where β is the gain. As $\beta \rightarrow \infty$ the form of $f[\cdot]$ takes on that of a step function that transitions rapidly from 0 to 1. In fact, it should remind you of the rapid transition seen in neuronal spike rate with a Hopf bifurcation, such as the squid axon studied by Hodgkin and Huxley. This is a useful approximation, as it allows us to consider the mapping of Boolean functions, a program started in the early 1950's by the prescient paper of McCullock and Pitts.

Consider the case of an AND gate with two inputs and one true output, y. We restrict ourselves to $N = 2$ inputs solely as a means to be able to draw pictures; a model for integration of inputs by a neurons may consist of $N >> 1000$ inputs!

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

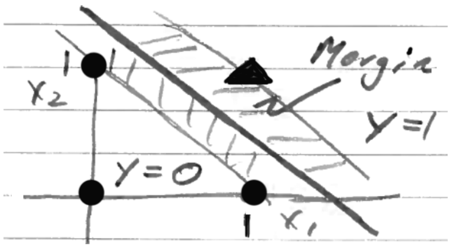
(10.3)

If we look at the input and require that it is positive for $W_1 x_1 + W_2 x_2 - \theta > 0$ and negative for $W_1 x_1 + W_2 x_2 - \theta < 0$, we will have

$$\begin{aligned} 0 &\quad < \theta \\ W_1 &\quad < \theta \\ W_2 &\quad < \theta \\ W_1 + W_2 &\quad > \theta \end{aligned} \tag{10.4}$$

There is a set of values of W_1, W_2, and θ that will work. One that gives the largest margin, *i.e.*, is least susceptibility to variations in the input, is the choice

$$\begin{aligned} W_1 &= 1 \\ W_2 &= 1 \\ \theta &= 3/2 \end{aligned} \tag{10.5}$$



.

This defines a line that divides the "1" output from the "0" output. The "OR" function is similarly defined, except that $\theta = 1/2$. So far so good. But we observe that "XOR" cannot be described by this formalism, as there are now two regions with a positive output, not one. So we cannot split the space of inputs with a single line.

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

(10.6)

The "XOR" can be solved by introducing an additional dimension so that we can split the space in 3-dimensions by a plane.

10.2 Perceptron learning - Binary gain function

For a binary input-output function, we can write a construction rule for the values of $\vec{W}$ starting from $\vec{W}$ = (0,0) and using the known pairs of inputs and outputs. We write

$$\vec{W} \leftarrow \vec{W} + y\vec{x}. \tag{10.7}$$

when $\hat{y} \neq y$. Let's use this for the case of AND. Note that the pairing has y = 1 for $\vec{x}$ = (1,1) and that y = 0 for the three other members $\vec{x}$ = (0,1), (1,0), and (0,0), so they will not contribute to learning. Then $\vec{W} \leftarrow (0,0)\ +\ 1(1,1) = (1,1)$.

The learning rule covers the threshold as well, *i.e.*,

$$\theta \leftarrow \theta + y\vec{w} \cdot \vec{x} - \langle y \rangle\,. \tag{10.8}$$

For the case of AND, we start with $\theta = 0$ and have $\theta \leftarrow 0 + 1(1,1)^{\mathbf{T}}(1,1) - 1/2 = 3/2$.

10.3 Convergence of Perceptron learning

We now consider a proof that the Perceptron can aways learn a rule when there is a plane that divides the input space. The analysis becomes a bit easier if we switch to a symmetric notation, with $y = \pm 1$ and elements $x = \pm 1$. Consider "n" sets of Boolean functions with

$$\text{Input(n)} \ \equiv \vec{x}(n) = \{x_1(n), x_2(n), \cdots, x_N(n)\} \tag{10.9}$$

and

$$\text{True output(n)} \ \equiv y(n) = \pm 1. \tag{10.10}$$

Training consists of learning the $\vec{W}$s from the different sets of $\vec{x}(n)$ and $y(n)$, denoted $\{y(k), \vec{x}(k)\}$. We calculate the predicted output above from each $\vec{x}$ using the old values of $\vec{W}$s and compare it with the true of y(n). Specifically

$$\text{Calculated output} \equiv \tilde{y}(n) = f[\vec{W}(n) \cdot \vec{x}(n)]. \tag{10.11}$$

The update rule to the $\vec{W}$s is in terms of the True output and the Calculated output, *i.e.*,

$$\begin{aligned} \vec{W}(m+1) &= \vec{W}(m) + \frac{1}{2}\left[y(n) - \tilde{y}(n)\right]\vec{x}(n) \qquad (10.12) \\ &= \vec{W}(m) + \frac{1}{2}\left[y(n) - f[\vec{W}(n) \cdot \vec{x}(n)]\right]\vec{x}(n) \end{aligned}$$

where we use the notation $\vec{W}(m)$ for the set of weights after m iterations of learning. Clearly we have used $m \geq n$. Correct categorization will lead to

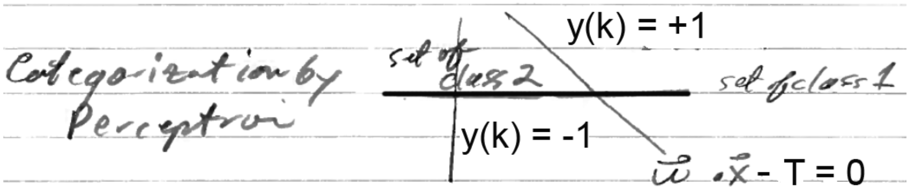
$$\tilde{y}(n) = y(n) \quad \text{implies} \quad \vec{W}(m+1) = \vec{W}(m) \tag{10.13}$$

while incorrect categorization leads to a change in weights

$$\tilde{y}(n) \neq y(n) \quad \text{implies} \quad \vec{W}(m+1) = \{ \begin{array}{ll} \vec{W}(m) + \vec{x}(n) & \text{if } \vec{W}(n) \cdot \vec{x}(n) < 0 \\ \vec{W}(m) - \vec{x}(n) & \text{if } \vec{W}(n) \cdot \vec{x}(n) > 0. \end{array} \tag{10.14}$$

One way to look at learning is that the examples can be divided into two training sets:

$$\begin{array}{llll} \text{Set of class 1 examples} & \{y(k), \vec{x}(k)\} & \text{with} & y(k) = +1 \quad \forall \quad k \\ \text{Set of class 2 examples} & \{y(k), \vec{x}(k)\} & \text{with} & y(k) = -1 \quad \forall \quad k. \end{array} \tag{10.15}$$



An important point about the use of Perceptrons is the existence of a learning rule that can be proved to converge. The idea in the proof is to show that with consideration of more and more examples, *i.e.*, with increasing n, the corrections to the $\vec{W}(n)$ grow faster than the number of errors.

10.3.1 Growth of corrections to the $\vec{W}(n)$ as a function of iteration

Suppose we make m errors, which leads to m updates, among our set of n input-output pairs. That is, $\vec{W}(m) \cdot \vec{x}(m) < 0 \, \forall m$ for the set of class 1 examples, yet $y(m) = +1$. Let us estimate how the corrections to the $\vec{W}(m)$s grow as a function of the number of learning steps, *e.g.*, as m, m^2, m^3, *etc.* The update rule is

$$\begin{aligned} \vec{W}(m+1) &= \vec{W}(m) + \vec{x}(m) \qquad (10.16) \\ &= \vec{W}(m-1) + \vec{x}(m-1) + \vec{x}(m) \\ &= \vec{W}(m-2) + \vec{x}(m-2) + \vec{x}(m-1) + \vec{x}(m) \\ &= \cdots \\ &= \vec{W}(0) + \sum_{k=0}^{m} \vec{x}(k). \end{aligned}$$

In the above, all of the m corrections made use of the first m entries of the set of class 1 examples. With no loss of generality, we take the initial value of the weight vector as $\vec{W}(0) = 0$, so that

$$\vec{W}(m+1) = \sum_{k=0}^{m} \vec{x}(k). \tag{10.17}$$

Now consider a solution to the Perceptron, denoted $\vec{W}_1$, that is based on the set of class 1 examples; by definition there is no index to this set of weights. Further, this satisfies $\vec{W}_1 \cdot \vec{x}(m) > 0 \, \forall m$ for any set of $\vec{x}$. We use the overlap of $\vec{W}_1$ as a means to form bounds on the corrections with increasing iterations of learning. We have

$$\begin{aligned} \vec{W}_1 \cdot \vec{W}(m+1) &= \sum_{k=1}^{m} \vec{W}_1 \cdot \vec{x}(k) \qquad (10.18) \\ &\geq m \times \text{minimum} \left\{ \vec{W}_1 \cdot \vec{x}(k) \right\} \end{aligned}$$

where $\vec{x}(k) \in$ set 1 examples. Then

$$\|\vec{W}_1 \cdot \vec{W}(m+1)\| \geq m \times \min \left\{ \vec{W}_1 \cdot \vec{x}(k) \right\}. \qquad (10.19)$$

But by the Cauchy-Schwartz inequality,

$$\|\vec{W}_1\| \|\vec{W}(m+1)\| \; \geq \; \|\vec{W}_1 \cdot \vec{W}(m+1)\| \qquad (10.20)$$

so

$$\|\vec{W}_1\| \|\vec{W}(m+1)\| \; \geq \; m \times \min \left\{ \vec{W}_1 \cdot \vec{x}(k) \right\} \qquad (10.21)$$

or

$$\|\vec{W}(m+1)\| \; \geq \; m \times \frac{\min \left\{ \vec{W}_1 \cdot \vec{x}(k) \right\}}{\|\vec{W}_1\|} \qquad (10.22)$$

and we find that the correction to the weight vector $\vec{W}$ after m steps of learning scales as m.

10.3.2 Growth of errors in the $\vec{W}(n)$ as a function of learning

We now estimate how the error to the weight vector $\vec{W}(m)$ grows as a function as the number of learning steps. The error can grow as each learning step can add noise as well as corrects for errors in the output $\hat{y}(m)$. The key for convergence is that the error grows more slowly than the correction, *i.e.*, as most as $m^{2-\epsilon}$. We start with the change in the weight vector as as function of the update step. After m updates, we have

$$\vec{W}(m+1) = \vec{W}(m) \, + \, \vec{x}(m). \qquad (10.23)$$

But

$$\begin{aligned} \|\vec{W}(m+1)\|^2 &= \|\vec{W}(m) \, + \, \vec{x}(m))\|^2 \qquad (10.24) \\ &= \|\vec{W}(m)\|^2 \, + \, \|\vec{x}(m)\|^2 \, + \, 2\vec{W}(m) \cdot \vec{x}(m) \\ &\leq \|\vec{W}(m)\|^2 \, + \, \|\vec{x}(m)\|^2 \end{aligned}$$

so

$$\|\vec{W}(m+1)\|^2 \, - \, \|\vec{W}(m)\|^2 \leq \|\vec{x}(m)\|^2. \qquad (10.25)$$

Now we can iterate:

$$\begin{aligned} \|\vec{W}(m+1)\|^2 \, - \, \|\vec{W}(m)\|^2 &\leq \|\vec{x}(m)\|^2 \qquad (10.26) \\ \|\vec{W}(m)\|^2 \, - \, \|\vec{W}(m-1)\|^2 &\leq \|\vec{x}(m-1)\|^2 \\ &\cdots \\ \|\vec{W}(1)\|^2 \, - \, \|\vec{W}(0)\|^2 &\leq \|\vec{x}(0)\|^2. \end{aligned}$$

We sum the right and left sides separately, and again take $\vec{W}(0) = 0$, to get

$$\begin{aligned} \|\vec{W}(m+1)\|^2 &\leq \sum_{k=0}^{m} \|\vec{x}(k)\|^2 \\ &\leq (m+1) \times \text{maximum} \left\{ \|\vec{x}(k)\|^2 \right\}. \end{aligned} \tag{10.27}$$

Thus we find that the errors to the weight vector $\vec{W}$ after m corrections scale as $\sqrt{m+1} \approx \sqrt{m}$, i.e.,

$$\|\vec{W}(m+1)\| \leq \sqrt{m \times \max\{\|\vec{x}(k)\|^2\}}. \tag{10.28}$$

or

$$\sqrt{m \times \max\{\|\vec{x}(k)\|^2\}} \geq \|\vec{W}(m+1)\|. \tag{10.29}$$

10.3.3 Proof of convergence

We now have two independent constraints on $\|\vec{W}(m+1)\|$:

$$\sqrt{m \times \max\{\|\vec{x}(k)\|^2\}} \geq \|\vec{W}(m+1)\| \geq m \times \frac{\min\left\{\vec{W}_1 \cdot \vec{x}(k)\right\}}{\|\vec{W}_1\|} \tag{10.30}$$

and thus

$$m \geq \frac{\max\{\|\vec{x}(k)\|^2\}}{\left(\min\left\{\vec{W}_1 \cdot \vec{x}(k)\right\}\right)^2} \|\vec{W}_1\|^2. \tag{10.31}$$

Fini!

10.4 Perceptron learning - Analog gain function

We return to the Perceptron but this time will focus on the use of a gain function that is not binary, i.e., β infinite. We consider changes to the wights and write the estimated output as $\hat{y}(n)$, where n refers to the n-th round of learning. Then

$$\hat{y}(n) = f\left[\sum_{j=1}^{N} W_j(n) x_j(n)\right] \tag{10.32}$$

where the $\vec{W}(n)$ are the weights after n rounds of learning and the $(\vec{x}(n), y(n)$ are the n-th input-output pair in the training set. The mean-square error between the calculated and the actual output is

$$E(n) = \frac{1}{2}\left[y(n) - \hat{y}(n)\right]^2 \tag{10.33}$$

The learning rule for the weights on the n-th iteration, $W_j(n) = W_j(n-1) + \Delta W_j(n)$ is now written it terms of minimizing the error, with

$$\Delta W_j(n) = -\eta \frac{\partial E(n)}{\partial W_j(n)} \tag{10.34}$$

where η is a small number. We then insert the form of $E(n)$ and compute

$$\begin{aligned}
\Delta W_j(n) &= -\eta \left[y(n) - \hat{y}(n)\right](-1)\frac{\partial \hat{y}(n)}{\partial W_j(n)} \qquad (10.35)\\
&= \eta \left[y(n) - \hat{y}(n)\right] \frac{\partial f[\cdot]}{\partial[\cdot]} \frac{\partial\left[\sum_{j=1}^{N} W_j(n)x_j(n)\right]}{\partial W_j(n)}\\
&= \eta \left[y(n) - \hat{y}(n)\right] \frac{\partial f[\cdot]}{\partial[\cdot]} x_j(n)
\end{aligned}$$

where we used the above form of the gain function,

$$f[z] = \frac{1}{1+e^{-\beta z}} \qquad (10.36)$$

for $\hat{y}(n)$. We further recall that

$$\frac{\partial f[z]}{\partial[z]} = \beta f[z]\left[1 - f[z]\right] \qquad (10.37)$$

so

$$\Delta W_j(n) = \eta\ \beta\ \hat{y}(n)\ \ [1 - \hat{y}(n)]\ \ [y(n) - \hat{y}(n)]\ \ x_j(n). \qquad (10.38)$$

The first group of terms, $\eta\beta$, are just constants. The product $\hat{y}(n)\,[1 - \hat{y}(n)]$ compresses the fractional change in the weight away from the mean value $\hat{y}(n) = 1/2$. The term $[y(n) - \hat{y}(n)]$ is proportional to the error. The final term $x_j(n)$ is just the value of the input. The process of learning is repeated over the entire training set $(\vec{x}(n), y(n))$.

10.5 Two layered network - Analog gain function

We consider a two layer network with a single output neuron. The input layer corresponds to the $\vec{x}(n)$, as above. The middle layer is refereed to as the hidden layer, $\vec{h}(n)$. Using the same form of the nonlinearity as above,

$$\hat{h}_j(n) = f_h\left[\sum_{i=1}^{N} W_{ji}(n)x_i(n)\right] \qquad (10.39)$$

where the W_{ji} are the connections from each of the input units x_i to the hidden unit h_j and

$$f_h[z] = \frac{1}{1+e^{-\beta_{\hat{h}} z}}. \qquad (10.40)$$

The output unit is driven by the output from the hidden units, i.e.,

$$\hat{y}(n) = f_o\left[\sum_{j=1}^{M} W_j(n)\hat{h}_j(n)\right] \qquad (10.41)$$

where the W_j are the connections from each of the hidden units h_j to the output unit y sand

$$f_o[z] = \frac{1}{1 + e^{-\beta_o z}}. \tag{10.42}$$

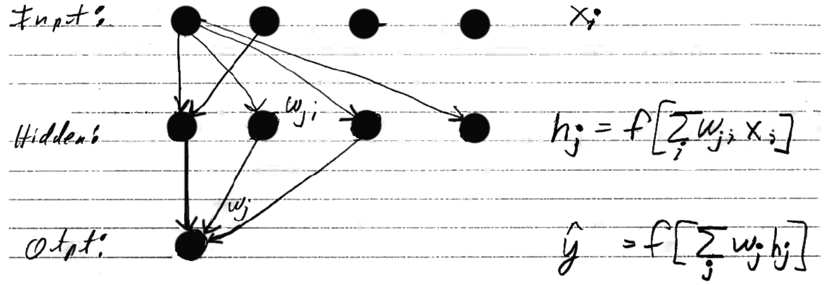
The error after n iterations of learning can only depend on the output, and as above,

$$E(n) = \frac{1}{2}\left[y(n) - \hat{y}(n)\right]^2 \tag{10.43}$$

10.5.1 Learning at the output layer - Analog gain function

$$\begin{aligned}
\Delta W_j(n) &= -\eta \frac{\partial E(n)}{\partial W_j(n)} &(10.44)\\
&= -\eta\left[y(n) - \hat{y}(n)\right](-1)\frac{\partial \hat{y}(n)}{\partial W_j(n)}\\
&= \eta\left[y(n) - \hat{y}(n)\right]\frac{\partial f_o[\cdot]}{\partial[\cdot]}\frac{\partial\left[\sum_{j=1}^{N} W_j(n)\hat{h}_j(n)\right]}{\partial W_j(n)}\\
&= \eta\left[y(n) - \hat{y}(n)\right]\frac{\partial f_o[\cdot]}{\partial[\cdot]}\hat{h}_j(n)\\
&= \eta\beta_o\ \left[y(n) - \hat{y}(n)\right]\ \hat{y}(n)\left[1 - \hat{y}(n)\right]\ \hat{h}_j(n)
\end{aligned}$$

which is the same as the result for a Perceptron with the difference that the input is from the hidden layer.



10.5.2 Learning at the hidden layer - Analog gain function

We begin by substitution in the form of the input from the hidden layer into the equation of the estimated output to get

$$\hat{y}(n) = f_o\left[\sum_{j=1}^{M} W_j(n)\ f_h\left[\sum_{i=1}^{N} W_{ji}(n)x_i(n)\right]\right] \tag{10.45}$$

Then

$$\begin{aligned}
\Delta W_{ji}(n) &= \eta\frac{\partial E(n)}{\partial W_{ji}(n)} &(10.46)\\
&= -\eta\left[y(n) - \hat{y}(n)\right](-1)\frac{\partial \hat{y}(n)}{\partial W_{ji}(n)}
\end{aligned}$$

$$
\begin{aligned}
&= \eta\,[y(n) - \hat{y}(n)]\,\frac{\partial f_o[\cdot]}{\partial[\cdot]}\frac{\partial\left[\sum_{j=1}^{M} W_j(n)\; f_h\left[\sum_{i=1}^{N} W_{ji}(n)x_i(n)\right]\right]}{\partial W_{ji}(n)} \\
&= \eta\,[y(n) - \hat{y}(n)]\,\frac{\partial f_o[\cdot]}{\partial[\cdot]}\;\sum_{j=1}^{M} W_j(n)\;\frac{\partial f_h[\cdot]}{\partial[\cdot]}\frac{\partial\left[\sum_{i=1}^{N} W_{ji}(n)x_i(n)\right]}{\partial W_{ji}(n)} \\
&= \eta\,[y(n) - \hat{y}(n)]\,\frac{\partial f_o[\cdot]}{\partial[\cdot]}\frac{\partial f_h[\cdot]}{\partial[\cdot]}\;\sum_{j=1}^{M} W_j(n)\; x_i(n).
\end{aligned}
$$

We see that the update to the weight $\Delta W_{ji}(n)$ contains terms, $\sum_{j=1}^{M} W_j(n)$, that are non-local. Thus the value of the nonlocal terms must arrive in a retrograde manner from the output. In the slang of layer networks they are ”back propagated”. This overall process could be continued for any number of hidden layers. It also contain products of the derivative of gain functions. An attempt to gain biological relevance suggest the form

$$\Delta W_{ji}(n)\;\propto < W_j(n) >\; x_i(n). \tag{10.47}$$