Lecture 25 - Backpropagation Review & Cont.

November 7, 2017

1 Multi-Layer Perceptrons

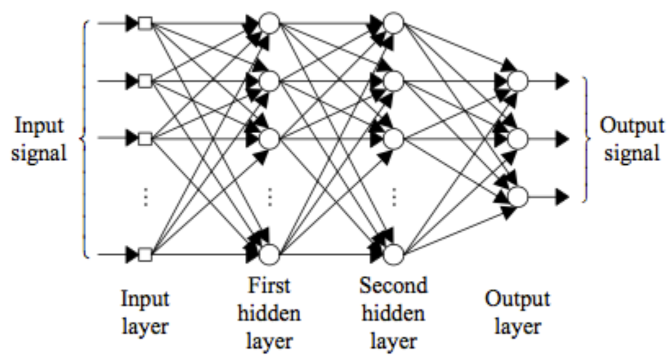


Figure 1: Multi-layer Perceptron with 2 Hidden Layers

- **Universal Approximation Theorem:**
  Let $\phi(\cdot)$ be a non-constant, bounded and monotone-increasing continuous function. Let I_{m_0} denote the m_0-dimensional unit hypercube $[0,1]^{m_0}$. The space of continuous functions on I_{m_0} is denoted by $C(I_{m_0})$. Then, given any function $f \ni C(I_{m_0})$ and $\epsilon > 0$, there exists an integer m_1 and sets of real constants α_i, β_i, and w_{ij}, where $i = 1, \ldots, m_1$ and $j = 1, \ldots, m_0$ such that we may define

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \phi \left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right) \tag{1}$$

  as an approximation realization of the function $f(\cdot)$: that is,

$$|F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0})| < \epsilon \tag{2}$$

for all $x_1, x_2, \ldots, x_{m_0}$ that like in the input space.

Essentially, the Universal Approximation Theorem states that a single hidden layer is sufficient for a multilayer perceptron to compute a uniform ϵ approximation to a given training set - provided you have the *right* number of neurons and the *right* activation function. (However, this does not say that a single hidden layer is optimal with regards to learning time, generalization, etc.)

1.1 Background for Error Back-Propagation

- Error Back-Propagation is based on *gradient descent*. Let's review gradient descent:

  Method of Gradient/Steepest Descent:

  - move in direction opposite to the gradient vector, $g = \bigtriangledown E(\mathbf{w})$

$$\begin{aligned} w(n+1) &= w(n) - \eta g(n) && (3) \\ \Delta w(n) &= w(n+1) - w(n) && (4) \\ \Delta w(n) &= -\eta g(n) \quad \text{Error correction rule} && (5) \end{aligned}$$

  - Show that using steepest descent, $E(\mathbf{w}(n+1)) < E(\mathbf{w}(n))$
  - Recall: Taylor Series Expansion: $f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots.$
  - Approximate $\mathbf{E}(\mathbf{w}(n+1))$ with Taylor Series around $w(n)$

$$\begin{aligned} E(w(n+1)) &\approx E(w(n)) + \Delta E(w(n))(w(n+1) - w(n)) && (6) \\ &\approx E(w(n)) + g^T(n)(\Delta w(n)) && (7) \\ &\approx E(w(n)) - \eta g^T(n) g(n) && (8) \\ &\approx E(w(n)) - \eta \left\| g(n) \right\|^2 && (9) \end{aligned}$$

  - For positive, small η, the cost function is decreased

- *How do we take gradients in matrix/vector notation? What about the Hessian?*

  - The gradient vector is a vector of partial derivatives

$$\nabla f(\bar{x}) = \left(\frac{\partial f(\bar{x})}{\partial x_1}, \frac{\partial f(\bar{x})}{\partial x_2}, \ldots, \frac{\partial f(\bar{x})}{\partial x_n} \right)^T \qquad (10)$$

1.2 Error Back-propagation

- There are *many* approaches to train a neural network. One of the most commonly used is the *Error Back-Propagation Algorithm*.
  - Two kinds of signals:
    1. Function Signals: presumed to perform useful function at the output of the network, also called input signal
    2. Error Signals: propagates backwards, involves an error-dependent function
  - Each hidden or output neuron performs two computations:
    1. Computation of function signal going out of this neuron
    2. Computation of an estimate of the gradient vector

- First let's consider the output layer...
  - Given a training set, $\{\mathbf{x}_n, d_n\}_{n=1}^N$, Let us consider the case where we want to find the parameters of our network that minimizes the squared error:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}(d_n - y_n)^2 \tag{11}$$

  - What is a common optimization approach to estimate the parameters that minimize an objective/error function? *gradient descent*
  - To use gradient descent, what do we need? The analytic form of the gradient.

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\left[\frac{1}{2}\sum_{n=1}^{N}(d_n - y_n)^2\right] \tag{12}$$

$$= \frac{1}{2}\sum_{n=1}^{N}\frac{\partial}{\partial w_i}(d_n - y_n)^2 \tag{13}$$

$$= \frac{1}{2}\sum_{n=1}^{N}2(d_n - y_n)\frac{\partial}{\partial w_i}(d_n - y_n) \tag{14}$$

$$= \sum_{n=1}^{N}(d_n - y_n)\left(\frac{\partial}{\partial w_i}d_n - \frac{\partial}{\partial w_i}y_n\right) \tag{15}$$

$$= \sum_{n=1}^{N}(d_n - y_n)\left(-\frac{\partial}{\partial w_i}y_n\right) \tag{16}$$

- What is y_n in terms of w_i? (At first let's assume we have no hidden layers, only the output layer to deal with)

$$y_n = \phi(v_n) = \phi(\mathbf{w}^T\mathbf{x}_n) \tag{17}$$

- Going back to computing our gradient...

$$= \sum_{n=1}^{N}(d_n - y_n)\left(-\frac{\partial}{\partial w_i}y_n\right) \tag{18}$$

$$= \sum_{n=1}^{N}-(d_n - y_n)\frac{\partial y_n}{\partial v_n}\frac{\partial v_n}{\partial w_i} \tag{19}$$

- So, $\frac{\partial y_n}{\partial v_n}$ will depend what form of an activation function we use. If we use the sigmoid: $y_n = \frac{1}{1+\exp(-\alpha v_n)}$, then *what is* $\frac{\partial y_n}{\partial v_n}$?

$$
\begin{aligned}
\frac{\partial y_n}{\partial v_n} &= \frac{\partial \phi(v_n)}{\partial v_n} && (20)\\
&= \frac{\partial}{\partial v_n}\frac{1}{1+\exp(-\alpha v_n)} && (21)\\
&= \frac{(1+\exp(-\alpha v_n))\left(\frac{\partial}{\partial v_n}1\right) - (1)\left(\frac{\partial}{\partial v_n}1+\exp(-\alpha v_n)\right)}{(1+\exp(-\alpha v_n))^2} && (22)\\
&= \frac{-\frac{\partial}{\partial v_n}(1+\exp(-\alpha v_n))}{(1+\exp(-\alpha v_n))^2} && (23)\\
&= \frac{-1}{(1+\exp(-\alpha v_n))^2}\exp(-\alpha v_n)(-\alpha) && (24)\\
&= \alpha\frac{1}{1+\exp(-\alpha v_n)}\frac{1}{1+\exp(-\alpha v_n)}\exp(-\alpha v_n) && (25)\\
&= \alpha\frac{1}{1+\exp(-\alpha v_n)}\frac{\exp(-\alpha v_n)}{1+\exp(-\alpha v_n)} && (26)\\
&= \alpha y_n(1-y_n) && (27)
\end{aligned}
$$

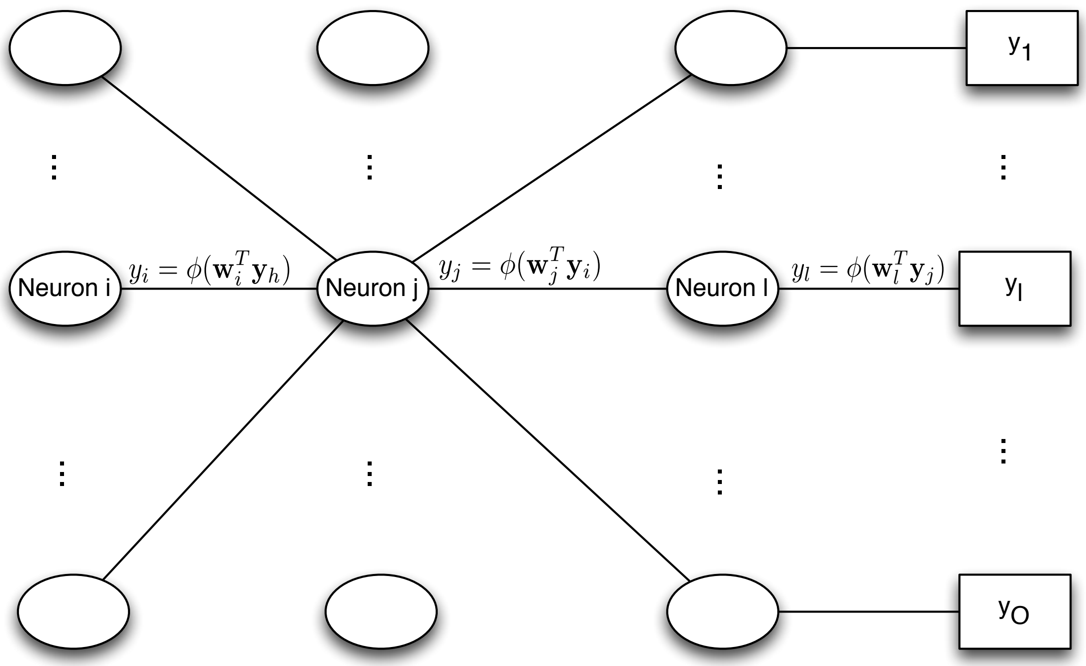
– Going back to computing our gradient...

$$
\begin{aligned}
&= \sum_{n=1}^{N} -(d_n - y_n)\frac{\partial y_n}{\partial v_n}\frac{\partial v_n}{\partial w_i} && (28)\\
&= \sum_{n=1}^{N} -(d_n - y_n)\alpha y_n(1-y_n)\frac{\partial v_n}{\partial w_i} && (29)\\
&= \sum_{n=1}^{N} -(d_n - y_n)\alpha y_n(1-y_n)\frac{\partial}{\partial w_i}\mathbf{w}^T\mathbf{x}_n && (30)\\
&= \sum_{n=1}^{N} -(d_n - y_n)\alpha y_n(1-y_n)x_{ni} && (31)
\end{aligned}
$$

– *Now that we have the gradient, how do we use this to update the output layer weights in our MLP?*

– *How will this update equation (for the output layer) change if the network is a multilayer perceptron with hidden units?*

– *Can you write this in vector form to update all weights simultaneously?*



- Now to address hidden layers... We have to deal with the credit assignment problem.
- Suppose we want to update w_{jn} where j is a hidden layer.
- The error objective function (i.e., the instantaneous sum of squared errors of the network at time k) is

$$E(k) = \frac{1}{2}\sum_{l=1}^{M} e_l^2(k) = \sum_{l=1}^{M}(d_l(k) - y_l(k))^2 = \sum_{l=1}^{M}(d_l(k) - \phi_l(v_l(k)))^2 \tag{32}$$

- We want to take the derivative of $E(k)$ with respect to w_{jn}

$$
\begin{aligned}
\frac{\partial E(k)}{\partial w_{jn}} &= \sum_{l=1}^{M} e_l \frac{\partial e_l(k)}{\partial y_j(k)} \frac{\partial y_j(k)}{\partial w_{jn}} && (33) \\
&= \sum_{l=1}^{M} e_l \left[\frac{\partial e_l(k)}{\partial v_l(k)} \frac{\partial v_l(k)}{\partial y_j(k)} \right] \frac{\partial y_j(k)}{\partial w_{jn}} && (34) \\
&= \sum_{l=1}^{M} e_l \left[\big(-\phi'(v_l(k))\big) \frac{\partial v_l(k)}{\partial y_j(k)} \right] \frac{\partial y_j(k)}{\partial w_{jn}} && (35) \\
&= \sum_{l=1}^{M} e_l \left[\big(-\phi'(v_l(k))\big) (w_{lj}(k)) \right] \frac{\partial y_j(k)}{\partial w_{jn}} && (36) \\
&= \sum_{l=1}^{M} e_l \left[\big(-\phi'(v_l(k))\big) (w_{lj}(k)) \right] \frac{\partial y_j(k)}{\partial v_j} \frac{\partial v_j(k)}{\partial w_{jn}} && (37) \\
&= \sum_{l=1}^{M} e_l \left[\big(-\phi'(v_l(k))\big) (w_{lj}(k)) \right] \left[\phi_j'(v_j(k)) y_i(k) \right] && (38)
\end{aligned}
$$

- Let's define a *local gradient* $\delta_l(k)$

$$
\begin{aligned}
\delta_l(k) &= -\frac{\partial E(k)}{\partial v_l(k)} && (39) \\
&= e_l \phi'(v_l(k)) && (40)
\end{aligned}
$$

- Similarly,

$$
\begin{aligned}
\delta_j(k) &= -\frac{\partial E(k)}{\partial v_j(k)} && (41)\\
&= -\sum_{l=1}^{M} e_l \frac{\partial e_l(k)}{\partial v_l(k)} \frac{\partial v_l(k)}{\partial y_j(k)} \frac{\partial y_j(k)}{\partial v_j} && (42)\\
&= \sum_{l=1}^{M} \delta_l(k) \frac{\partial v_l(k)}{\partial y_j(k)} \frac{\partial y_j(k)}{\partial v_j} && (43)\\
&= \sum_{l=1}^{M} \delta_l(k)\left(w_{lj}(k)\right) \frac{\partial y_j(k)}{\partial v_j} && (44)\\
&= \phi_j'(v_j(k)) \sum_{l=1}^{M} \delta_l(k)\left(w_{lj}(k)\right) && (45)\\
& && (46)
\end{aligned}
$$

- So, you can write the gradient at a hidden neuron in terms of the local gradient and the connected neurons in the next layer