Neural Networks and Deep Learning

Regression and Classification with Linear Models

In the beginning of the semester, we looked at linear models of the form

\[ f(x) = w^T x + b \]

We also looked at threshold classifiers of the form

\[ f(x) = \begin{cases} 1 & \text{if } w^T x \geq b \\ 0 & \text{if } w^T x < b \end{cases} \]

The neural network operates with units which look like neurons, that is, each “neuron” takes as input a set of input features \( x = (x_1, \ldots, x_D) \) and weights \( w = (w_1, \ldots, w_D) \) and computes an output

\[ y = \sigma(w^T x + b) \]

where \( \sigma \) is usually a nonlinear function. We may omit the \( b \) term by introducing a feature mapping \( \phi \).

For regression, our \( \sigma \) function is \( \sigma(a) = a \). For classification, our \( \sigma \) function is

\[ \sigma(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases} \]

Usually we use \( f(x) = \sigma(w^T \phi(x)) \) where \( \phi : \mathbb{R}^D \to \mathbb{R}^M \). The function \( \phi \) is fixed in advance.

**Example:** (2 Layer Neural Network). There are \( D \) inputs, \( x_1, \ldots, x_D \). The first layer has \( M \) neurons, where the value at each neuron is computed using the \( \phi \) function (evaluated at the \( D \) inputs). Since \( \phi(x) \in \mathbb{R}^M \), each neuron in this first layer outputs one of the \( M \) elements of \( \phi(x) \). The outputs of these \( M \) neurons then get sent to a final neuron (second layer), which outputs \( y = \sigma(w^T \phi(x)) \).
The Model

Feed Forward

Let \( \xi_i \) be the output of neuron \( i \) and let \( I(j) \) denote the set of input neurons to neuron \( j \). Then our model is

\[
a_j = \sum_{i \in I(j)} w_{j,i} \xi_i \\
\xi_j = h(a_j)
\]

where \( w_{j,i} \) is the weight from neuron \( i \) to neuron \( j \) and \( h : \mathbb{R} \rightarrow \mathbb{R} \) is called the transfer function.

This network defines a function \( f : \mathbb{R}^D \rightarrow \mathbb{R} \). Evaluate the output of each neuron by “forward propogation”. The output neuron defines \( f(x) = \xi_O \) where \( O \) is the output neuron.

Minimize the error of the network on training data (Empirical risk minimization). We can define an error function that gets evaluated at network parameters \( \theta \) (weights)

\[
E(\theta) = \sum_{i=1}^{N} (f(x_i) - y_i)^2
\]

Gradient Descent

We have the usual procedure,

1. \( \theta_{t+1} = \theta_t - r \nabla E(\theta_t) \)

2. \( \nabla E(\theta) = \sum_{i=1}^{n} \nabla E_i(\theta) \)

where \( E_i(\theta) = (f(x_i) - y_i)^2 \) implies we can compute \( \nabla E(\theta) \) via the chain rule (back-propogation). Note that by the chain rule,

\[
\frac{\partial E}{\partial w_{j,i}} = \frac{\partial a_j}{\partial w_{j,i}} \frac{\partial E}{\partial a_j} \tag{1}
\]

We directly compute

\[
\frac{\partial a_j}{\partial w_{j,i}} = \xi_i \tag{2}
\]
from our first boxed expression. Denote $\delta_j = \frac{\partial E}{\partial a_j}$. Then

$$\delta_j = \frac{\partial E}{\partial a_j} = \sum_{k \in O(j)} \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad \text{(chain rule)} \quad (3)$$

where $O(j)$ is the set of neurons that have neuron $j$ as input. We have

$$a_k = \sum_{i \in I(k)} w_{k,i} h(a_i) \quad (4)$$

Since we can represent $a_k$ in this way, we differentiate (4) to obtain

$$\frac{\partial a_k}{\partial a_j} = w_{k,j} h'(a_j) \quad (5)$$

Plugging (5) back into (3), we get

$$\delta_j = \sum_{k \in O(j)} \frac{\partial E}{\partial a_k} w_{k,j} h'(a_j)$$

Combining this with (2), we can compute the gradient of the error using expression (1).

**Back Propogation**

We have

$$\delta_j = \sum_{k \in O(j)} \delta_k w_{k,j} h'(a_j)$$

Note that the above is a recursive expression building up from the right side of our neural net (the output).

For the base case,

$$\delta_O = \frac{\partial E}{\partial a_O}$$

Our error is $E = (h(a_O) - y)^2$ and thus

$$\delta_O = 2(h(a_0) - y)h'(a_O)$$

Note that for this scheme to work, we need to pick a transfer function $h$ which is differentiable. In practice, we typically use the **sigmoid** function defined below

$$h(a) = \frac{1}{1 + e^{-a}}$$
Note that $h : \mathbb{R} \to (0, 1)$ and $h'$ is nonzero everywhere. Another common choice of $h$ is the inverse tan function. The sigmoid function $h$ satisfies the following properties

$$h(a) = 1 - h(-a)$$
$$h'(a) = h(a)(1 - h(a))$$