Tutorial 9: Neural Networks & Graphs and the Web

Learning Questions

1. We can write a single matrix equation for the outputs of the network, appending the bias to the weights (and making $x_0 = 1$), and transposing the training data matrix, to get one column per training item:

$$\mathbf{Y}^T = \mathbf{W}\mathbf{X}^T$$

$$\mathbf{Y}^T = \begin{bmatrix} 0.05 & 0.1 & 0.2 & 0.1 & 0.15 & 0.05 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} 0.35 & 0.4 & 0.5 \end{bmatrix}$$

Computing the error signals:

$$\begin{aligned} \delta_n &= y_n - t_n \\ \delta_1 &= 0.35 - 1 = -0.65 \\ \delta_2 &= 0.4 - 0 = 0.4 \\ \delta_3 &= 0.5 - 1 = -0.5 \end{aligned}$$

The updated weights after an iteration of gradient descent are:

$$\begin{aligned} w_i^{new} &= w_i - \eta \sum_{n=1}^{N} \delta_n x_{ni} \\ w_0 &= 0.05 - 0.01(-0.65 \cdot 1 + 0.4 \cdot 1 - 0.5 \cdot 1) = 0.0575 \\ w_1 &= 0.1 - 0.01(-0.65 \cdot 1 + 0.4 \cdot 0 - 0.5 \cdot 1) = 0.1115 \\ w_2 &= 0.2 - 0.01(-0.65 \cdot 1 + 0.4 \cdot 1 - 0.5 \cdot 1) = 0.2075 \\ w_3 &= 0.1 - 0.01(-0.65 \cdot 0 + 0.4 \cdot 1 - 0.5 \cdot 0) = 0.096 \\ w_4 &= 0.15 - 0.01(-0.65 \cdot 0 + 0.4 \cdot 0 - 0.5 \cdot 1) = 0.155 \\ w_5 &= 0.05 - 0.01(-0.65 \cdot 0 + 0.4 \cdot 1 - 0.5 \cdot 0) = 0.046 \end{aligned}$$

2. First we look at the single sigmoid output.

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial a} \cdot \frac{\partial a}{\partial w_i} \\ \frac{\partial E}{\partial y} &= -\frac{t}{y} + \frac{1-t}{1-y} \\ &= \frac{-(1-y)t + y(1-t)}{y(1-y)} \\ \frac{\partial y}{\partial a} &= y(1-y) \quad \text{(usual sigmoid derivative)} \\ \frac{\partial a}{\partial w_i} &= x_i \quad \text{(as usual)} \\ \frac{\partial E}{\partial w_i} &= \frac{-(1-y)t + y(1-t)}{y(1-y)} \cdot y(1-y) \cdot x_i \\ &= (-(1-y)t + y(1-t))x_i = (y-t)x_i \end{aligned}$$

So with a logistic sigmoid transfer function and the negative log probability error function, the error signal $\delta = \partial E/\partial a = (y - t)$. The derivative of the sigmoid cancels out. The logistic sigmoid transfer function corresponds to a two class posterior probability estimation, and the negative log probability is the 'natural' or consistent error function: the network estimates a posterior probability, and the error function corresponds to directly optimising the parameters to estimate that (log) probability.

Now the multiclass case. In this case the kth activation a_k—and hence the weight w_{ki}—influences the error function through all the output units, because of the normalising term in the denominator. We have to take this into account when differentiating.

$$\begin{aligned} \frac{\partial E}{\partial w_{ki}} &= \sum_{c=1}^{C} \frac{\partial E}{\partial y_c} \cdot \frac{\partial y_c}{\partial a_k} \cdot \frac{\partial a_k}{\partial w_{ki}} \\ \frac{\partial a_k}{\partial w_{ki}} &= x_i \text{(as usual)} \\ \frac{\partial E}{\partial y_c} &= -\frac{t_c}{y_c} \end{aligned}$$

Now to look at $\partial y_c/\partial a_k$ we look at two cases, when $c = k$, and when $c \neq k$.

First when $c = k$, we apply the quotient rule of differentiation:

$$\begin{aligned} \frac{\partial y_c}{\partial a_c} &= \frac{\sum_j \exp(a_j) \cdot \exp(a_c) - \exp(a_c)\exp(a_c)}{(\sum_j \exp(a_j))^2} \\ &= y_c - y_c^2 = y_c(1 - y_c) \end{aligned}$$

And, when $c \neq k$:

$$\begin{aligned} \frac{\partial y_c}{\partial a_k} &= \frac{-\exp(a_c)\exp(a_k)}{(\sum_j \exp(a_j))^2} \\ &= -y_c y_k \end{aligned}$$

We can combine these using the Kronecker delta δ_{ck} ($\delta_{ck} = 1$ if $c = k$, $\delta_{ck} = 0$ if $c \neq k$):

$$\frac{\partial y_c}{\partial a_k} = y_c(\delta_{ck} - y_k)\,.$$

Putting these derivatives together in the chain rule we have:

$$\begin{aligned}\frac{\partial E}{\partial w_{ki}} &= \sum_{c=1}^{C}\left(-\frac{t_c}{y_c}\right)(y_c(\delta_{ck} - y_k))\,x_i\\ &= \sum_{c=1}^{C} t_c(y_k - \delta_{kc})x_i\,.\end{aligned}$$

Now, since we have a '1-from-C' output coding, we know that $\sum_c t_c y_k = y_k$ (in fact this holds for the weaker condition $\sum_c t_c = 1$), thus:

$$\frac{\partial E}{\partial w_{ki}} = (y_k - t_k)x_i\,.$$

Beautiful! Once again the derivative of the transfer function cancels, and we have the error signal $\delta_k = y_k - t_k$. The softmax is the multiclass counterpart of the logistic sigmoid and is the natural partner of the negative log probability error function.

Lots of subscripts in this question, so need to take care.

Some other comments:

- Sum-squared error function is not invalidated and is a good 'general-purpose error function'. But if you are doing classification, and interpret the outputs as posterior probability estimates, then it is consistent to maximise the probability (or, in practice, minimise the negative log probability). And you are rewarded by a nice derivative.
- Don't be confused by the Kronecker delta which is unrelated to the error signal δ. Sorry for overloaded notation, but both are standard.
- logs to base e since we have exp in the transfer function.
- Not having the transfer function derivative ($y(1 - y)$) results in larger derivative values, and experiments consistently indicate that this leads to faster gradient descent training.
- Remember all the derivatives in the question are for a single training example; you would use these directly in stochastic gradient descent; otherwise you would sum over the training set.

ADS questions

1. Let $G = (V, E)$ be a directed graph with n vertices and m edges. We define the *transpose* of G to be the graph $G^T = (V, E^T)$ where $E^T = \{(u, v) \in V \times V \mid (v, u) \in E\}$.

   (a) Reverse the direction of each edge.

   (b) Create an array A for the adjacency list representation of G^T, initially all the lists are empty. For each vertex v of G travel along the list of edges for v and for each vertex u there insert an entry v at the head of the list under $A[u]$. We take time $\Theta(n)$ to create and initialise A.

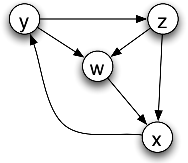
   Each insertion is $\Theta(1)$ time and we do $\Theta(m)$ of these so the total cost here is $\Theta(m)$. Thus the time to create the adjacency list representation of G^T is $\Theta(n) + \Theta(m) = \Theta(n + m)$.

   For the adjacency matrix representation we just compute the transpose of the matrix. This follows since the (i, j)th entry of the transposed matrix is the (j, i)th entry of the input matrix. Thus the time is $\Theta(n2)$.

   (c) Since G^T simply reverses directions of edges, a path from v to u in G becomes a path from u to v in G^T and vice versa. The claim now follows since two vertices u, v of G are in the same strongly connected component if and only if there is a directed path from v to u and a directed path form u to v.

   (d) The algorithm outputs the strongly connected components of G, see [CLRS] for details.

2. Here is the webgraph for reference.



   (a) The closed walk x, y, z, w, x shows that the graph is strongly connected. It is aperiodic since the cycle x, y, w has length 3 and z, w, x, y has length 4.

   The linear system follows from the fact that $|\mathrm{Out}(x)| = 1$, $|\mathrm{Out}(y)| = 2$, $|\mathrm{Out}(z)| = 2$ and $|\mathrm{Out}(w)| = 1$. So we get the table

	w	x	y	z
w	0	1/1	0	0
x	0	0	1/1	0
y	1/2	0	0	1/2
z	1/2	1/2	0	0

   Recall that we if there is an edge from u to v then we put $1/|\mathrm{Out}(u)|$ in entry (u, v) otherwise we put 0.

   (b) We see immediately from the system that $R_y = R_x$ and $R_z = R_y/2$. Thus we have $R_y = 2R_z$ and $R_x = 2R_z$ (this way we avoid fractions for the moment). Now the first two conditions are

$$\begin{aligned}R_w &= \frac{1}{2}R_y + \frac{1}{2}R_z,\\ R_x &= R_w + \frac{1}{2}R_z.\end{aligned}$$

   From the second one we have $R_w = R_x - R_z/2 = 2R_z - R_z/2 = 3R_z/2$. The first one is redundant (but it would be a good idea to substitute the values found to see that they do cross check). Thus the eigenvector is any non-zero multiple of $(3/2, 2, 2, 1)$. So our ranking is

$$R_x = R_y > R_w > R_z.$$