Tutorial 9: Neural Networks & Graphs and the Web

The last tutorial is a joint Learning and ADS tutorial. It is not expected that you will be able to tackle all the questions by the tutorial. The suggestion is that you tackle at least three questions, and at least one question per thread.

Learning Questions

1. We have a 5 input, single output single layer network with weights $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} 0.1 & 0.2 & 0.1 & 0.15 & 0.05 \end{bmatrix}$$

and bias = 0.05.

We have 3 training examples, $(\mathbf{X}, \mathbf{T})$, expressed as a matrix, one row per training item:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

What are the weights after one iteration of gradient descent, with learning rate $\eta = 0.01$?

2. Consider a single-layer network with a single output y with a logistic sigmoid transfer function.

$$y = f(a) = \frac{1}{1 + \exp(-a)}$$

This is used for a two-class problem with classes C_1 (denoted by target variable $t = 1$) and C_2 (denoted by $t = 0$). We showed in the notes that in the case of a logistic sigmoid transfer function we can interpret y as the conditional probability $P(C_1 | \mathbf{x})$ and $(1 - y)$ as $P(C_2 | \mathbf{x})$, where $\mathbf{x}$ is the input vector.

The target t is a binary variable. We know that, given $\mathbf{x}$, the probability of $t = 1$ is $P(t = 1 \mid \mathbf{x}) = P(C_1 | \mathbf{x}) = y$; likewise we have $P(t = 0 \mid \mathbf{x}) = P(C_2 | \mathbf{x}) = 1 - y$. We can combine this information and write the distribution of the target t in the form:

$$P(t | x, \mathbf{W}) = y^t (1 - y)^{1-t}$$

This is a Bernoulli distribution, which we met earlier. Note that we have also explicitly shown the dependence on the weights. We can write the log probability:

$$\ln P(t | x, \mathbf{W}) = t \ln y + (1 - t) \ln(1 - y)$$

We can use this to optimise the weights $\mathbf{W}$ to maximise the log probability—or to minimise the negative log probability. We can do this by writing the error function as follows:

$$E(\mathbf{W}) = -(t \ln y + (1 - t) \ln(1 - y)) .$$

This is the error for a single training example, denoted as E_n in the notes. To avoid clutter the superscript n is ignored in this question

If we want to train by gradient descent, we need the derivative $\partial E / \partial w_j$ (where w_j connects the jth input to the single output). What is the expression for this derivative? Compare it with the derivative obtained when the sum square error function is used.

Now consider the C class case in which we have the usual "1-from-C" coding scheme, and in which the kth output y_k is interpreted as $P(C_k | \mathbf{x})$. In this case the negative log probability error function is:

$$E(\mathbf{W}) = -\sum_{k=1}^{C} t_k \ln y_k$$

If the transfer function for the output units is the softmax:

$$y_k = \frac{\exp(a_k)}{\sum_{j=1}^{C} \exp(a_j)}$$

then what is the expression for the derivative $\partial E / \partial w_{kj}$ in this case? Comment on your answer. (This is a slightly trickier derivative, remember that a particular y_k depends on all the a_j.)

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 the set E^T is defined by $E^T = \{(u, v) \in V \times V \mid (v, u) \in E\}$.

   (a) Explain in simple English (in one short sentence) how to obtain a diagram for G^T from one for G.

   (b) Suppose that G is given by an adjacency list representation. How long (in terms of a $\Theta(\cdot)$ expression) does it take to compute an adjacency list representation for G^T?

   What happens for the case when G is given by an adjacency matrix representation? (The hint is in the name!)

   (c) Prove that the strongly connected components of G and for G^T are exactly the same.

   (d) When performing a DFS we often *timestamp* the vertices. To do this we maintain an integer valued counter that is initialised to 0 and is incremented after each use. When we first discover a vertex v we give it its first time stamp which we denote by $d[v]$ (d is of *discovered*). When we finish with a vertex v we give it its second time stamp which we denote by $f[v]$ (f is for *finished*). Consider the following algorithm.

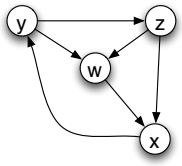
      1. Call dfs(G) to compute $f[v]$ for each vertex v.
      2. Compute G^T.
      3. Call dfs(G^T) but in the main loop of the dfs consider the vertices in decreasing order of $f[v]$ (as computed in line 1).
      4. Output the vertices of each tree in the dfs forest of step 3 as a separate collection for each tree (i.e., each tree gives us a collection of vertices labelled $C_1, C_2, \ldots$ say).

   Simulate this algorithm for three or four small graphs (up to around 8 vertices and with two or three strongly connected components). For example, start with a triangle, orient its edges clockwise so that you can get from any one vertex to any other, simulate the algorithm. Now reverse the direction of one edge and simulate again. Next take two triangles with their edges oriented clockwise. Join the two triangles with two edges going from two corresponding vertices. Direct one edge from the first triangle to the second and the other

edge from the second triangle to the first. Simulate the algorithm. Now direct both edges from the first triangle to the second and simulate the algorithm. Formulate a conjecture on what it finds, if you are not sure then check your simulations and perhaps do some more. The proof of the conjecture is not easy.

2. NOTE: The material for this question might not be covered till after the relevant tutorial, in which case you should to try it in private and check your answer with the sample one that will be placed on the web site. Sample solutions will be placed on the course web pages as usual so you can check your answer.

   Consider the following webgraph.



   (a) Show that the graph satisfies the basic PageRank™ model and its ranking can be expressed as the following linear system.

$$\left(R_w, R_x, R_y, R_z\right) = \left(R_w, R_x, R_y, R_z\right)\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/2 & 0 & 0 & 1/2 \\ 1/2 & 1/2 & 0 & 0 \end{pmatrix}.$$

   (b) Solve the system and display the ranking of the pages in decreasing order.