COMP9444
Neural Networks and Deep Learning
Term 2, 2024

Week 4 Tutorial: Softmax, Hidden Unit Dynamics (Sample Solution)

1. **Softmax**

   Recall the formula of Softmax:
   $Prob(i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$

   Consider a neural netowrk being trained on a classification task with three classes $1, 2, 3$. When the network is presented with a particular input, the output values are:

   $z_1 = 1.0, z_2 = 2.0, z_3 = 3.0$

   Suppose the correct class for this input is Class 2. Compute the following, to two decimal places:

   (a) Prob(i), for $i = 1, 2, 3$

   Prob$(1) = e^1/(e^1 + e^2 + e^3) = 2.718/30.193 = 0.09$

   Prob$(2) = e^2/(e^1 + e^2 + e^3) = 7.389/30.193 = 0.24$

   Prob$(3) = e^3/(e^1 + e^2 + e^3) = 20.086/30.193 = 0.67$

   (b) d(log Prob(2))/dz_j, for $j = 1, 2, 3$

   $$d(log\ Prob(2))/dz_1 = d(z_2 - log\ \textstyle\sum_j exp(z_j))/dz_1$$

   $$= -exp(z_1)/\textstyle\sum_j\ exp(z_j) = -0.09$$

   $$d(log\ Prob(2))/dz_2 = d(z_2 - log\ \textstyle\sum_j exp(z_j))/dz_2$$

   $$= 1 - exp(z_2)/\textstyle\sum_j\ exp(z_j) = 1 - 0.24 = 0.76$$

   $$d(log\ Prob(2))/dz_3 = d(z_2 - log\ \textstyle\sum_j exp(z_j))/dz_3$$

   $$= -exp(z_3)/\textstyle\sum_j\ exp(z_j) = -0.67$$

2. **Identical Inputs**

   Consider a degenerate case where the training set consists of just a single input, repeated 100 times. In 80 of the 100 cases, the target output value is 1; in other 20, it is 0. What will a back-propagation neural network predict for this example, assuming that it has been trained and reaches a global optimum? If the loss function is changed from *Sum Squared Error* to *Cross Entropy*, does it give the same result? (Hint: to find the global optimum, differentiate the loss function and set to zero.)

When Sum Squared Error is minimised, we have:

$E = 80 * \frac{(z-1)^2}{2} + 20 * \frac{(z-0)^2}{2}$

$\frac{dE}{dz} = 80 * (z-1) + 20 * (z-0)$

$\quad = 100 * z - 80$

Setting to zero: $\frac{dE}{dz} = 0$, we get:

$\quad z = 0.8$

When Cross Entropy is minimised, we have:

$E = -80 * log(z) - 20 * log(1-z)$

$\frac{dE}{dz} = \frac{-80}{z} + \frac{20}{(1-z)}$

$\quad = \frac{-80*(1-z)+20*z}{z*(1-z)}$

Setting to zero: $\frac{dE}{dz} = 0$, we get:

$\quad z = 0.8$ as before.

3. **Hidden Unit Dynamics**

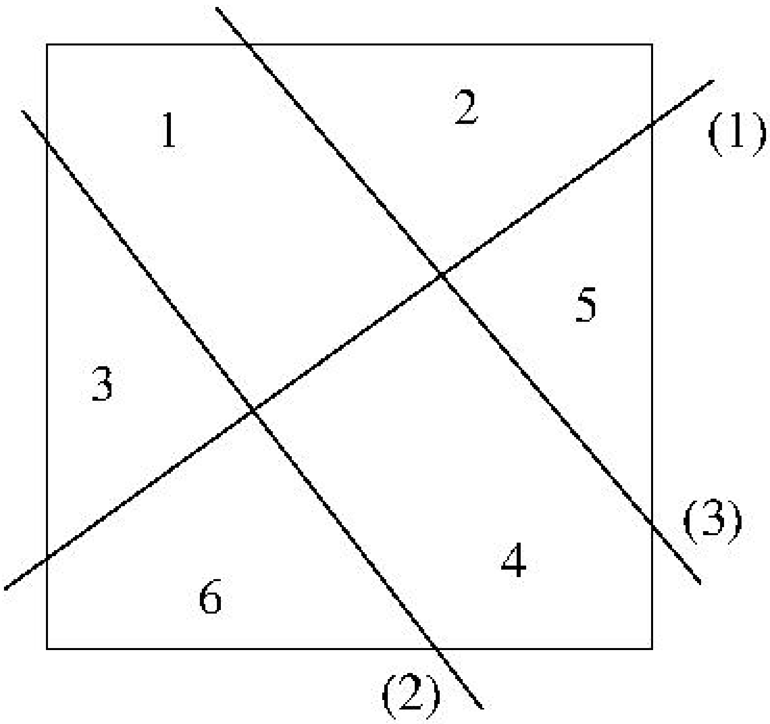
   Consider a fully connected feedforward neural network with 6 inputs, 2 hidden units and 3 outputs, using a tanh activation at the hidden units and sigmoid at the outputs. Suppose this network is trained on the following data, and that the training is successful.

   Draw a diagram showing:

Item	Inputs	Outputs
	123456	**123**
1.	100000	000
2.	010000	001
3.	001000	010
4.	000100	100
5.	000010	101
6.	000001	110

(a) for each input, a point in hidden unit space corresponding to that input, and

(b) for each output, a line dividing the hidden unit space into regions for which the value of that output is greater/less than one half.



4. **Linear Tranfer Functions**

Suppose you had a neural network with linear transfer functions. That is, for each unit the activation is some constant c times the weighted sum of the inputs.

(a) Assume that the network has one hidden layer. we can write the weights from the input to the hidden layer as a matrix $\mathbf{W^{HI}}$, the weights from the hidden to output layer as $\mathbf{W^{OH}}$, and the bias at the hidden and output layer as vector $\mathrm{b}^{\mathbf{H}}$ and $\mathrm{b}^{\mathbf{O}}$. Using matrix notation, write down equations for the value $\mathbf{O}$ of the units in the output layer as a function of these weights and biases, and the input $\mathbf{I}$. Show that, for any given assignment of values to these weights and biases, there is a simpler network with no hidden layer that computes the same function.

Using vector and matrix multiplication, the hidden activations can be written as:

$$\mathbf{H} = c * (\mathbf{b}^H + \mathbf{W}^{HI} * \mathbf{I})$$

The output activations can be written as:

$$\mathbf{O} = c * [\mathbf{b}^O + \mathbf{W}^{OH} * \mathbf{H}]$$

$$= c * [\mathbf{b}^O + \mathbf{W}^{OH} * c * (\mathbf{b}^H + \mathbf{W}^{HI} * \mathbf{I})]$$

$$= c * [(\mathbf{b}^O + \mathbf{W}^{OH} * c * \mathbf{b}^H) + (\mathbf{W}^{OH} * c * \mathbf{W}^{HI}) * \mathbf{I}]$$

Deu to the associativity of matrix multiplication, this can be written as :

$$\mathbf{O} = c * (\mathbf{b}^{OI} + \mathbf{W}^{OI} * \mathbf{I})$$

where:
$\mathbf{b}^{OI} = \mathbf{b}^O + \mathbf{W}^{OH} * c * \mathbf{b}^H$
$\mathbf{W}^O I = \mathbf{W}^{OH} * c * \mathbf{W}^{HI}$

Therefore, the same function can be computed by a simpler network, with no hidden layer, using the weights $\mathbf{W}^{OI}$ and bias $\mathbf{b}^{OI}$.

(b) Repeat the calculation in part(a), this time for a network with any number of hidden layers. What can you say about the usefulness of linear transfer functions?

By removing the layers one at a time as above, a simpler network with no hidden layer can be constructed which computes exactly the same function as the original multi-layer network. In other words, with linear activation functions, you don't get any benefit from having more than one layer.