Informatics 1 Cognitive Science – Tutorial 2 Solutions

Frank Keller, Carina Silberer, Frank Mollica

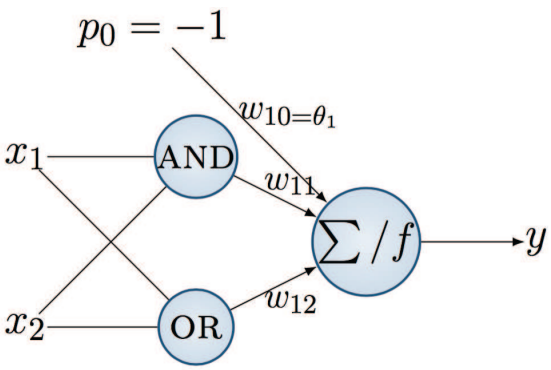
Week 3

The goal of this tutorial is to start building an in-depth understanding of how perceptrons work and the backpropagation algorithm used for training multilayer perceptrons. Before working on the exercises, you should make sure that you have revised the slides for lectures 5 and 6.

1 Perceptrons

In the lecture you learned that the primitive boolean functions AND and OR can be represented by a single perceptron each. You also saw that a single perceptron does not have the representational power to represent the boolean function XOR. A network of perceptrons two levels deep can in fact represent every boolean function.

Exercise 1 Consider the two-level network illustrated below. It is composed of three perceptrons. The two perceptrons of the first level implement the AND and OR functions, respectively.



Determine the weights w_{11}, w_{12} and threshold $\theta_1 (= w_{10})$ such that the network implements the XOR function. The initial weights are set to zero, i.e., $w_{11} = w_{12} = w_{10} = 0$, and the learning rate η is set to 0.1 (Feel free to choose other initial values for $w_{1.}$ and η).

Notes

- The input function for the perceptron on level j is the weighted sum $\sum$ of its input.

- The activation function f for a perceptron is a step function:

$$f = \begin{cases} 1 & if \sum > 0 \\ 0 & otherwise \end{cases}$$

- The threshold θ is considered as a weight, with input $p_0 = -1$.
- Assume that the weights for the perceptrons of the first level are given, meaning that these perceptrons already compute AND and OR correctly and their weights do not change.

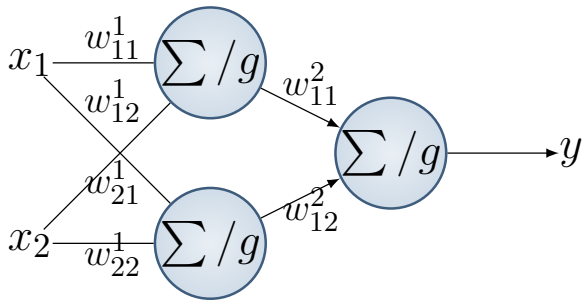
Solution 1

Learning Algorithm for XOR:
for *each training example* x
  $p \leftarrow (p_0,\ f_{\text{AND}}(x),\ f_{\text{OR}}(x))$
  $o \leftarrow f(\sum_{i=0}^{n} w_{1i} p_i)$
  $t \leftarrow target\ of\ x$
  for $i = 0 : n$
    $\Delta w_{1i} \leftarrow \eta(t - o)p_i$
    $w_{ji} \leftarrow w_{1i} + \Delta w_{1i}$

Please see the spreadsheet `xor-with-sols.xlsx` for the detailed computations.

2 Backpropagation in Multilayer Perceptrons

Exercise 2 Consider the following multilayer perceptron.



The network should implement the XOR function. Perform one epoch of backpropagation as introduced in the lecture on multilayer perceptrons. That is, for each training example:

1. Compute the activations of the hidden and output neurons
2. Compute the error of the network; backpropagate the error to determine Δw_{ij} for all weights w_{ij}
3. Update the weight w_{ij}

What was is error before and after updating the weights through one round of training (or epoch)?

We will break this into several steps below. It will be very time consuming to do the computations for backpropatgation by hand. Instead, you can use the Excel spreadsheet provided with the tutorial, or implement backpropagation in a programming language of your choice.

1. Write down the equations for the output of each unit.
2. Write down the equation for the error of the network based on its output using the MSE we discussed in lecture. We don't use this directly until we're reporting the error at the end – in updating the weights we want the gradient of the error.
3. Write down the derivative of the activation function (you don't have to derive this by hand; it's in the slides).
4. Write down the formulas for the changes in the weights (Δw), and the δ terms they rely on.
5. Compute the error for the initial weights. As noted above, use the spreadsheet to do this. Otherwise it's painful.
6. Compute the values for Δw for all weights.
7. Compute the new error after updating the weights.

Notes

- The activation function g for this perceptron is the sigmoid function: $g(x) = \frac{1}{1+e^{-x}}$.
- The thresholds are not shown in the network. The threshold nodes are set to -1, and their weights are denoted with θ.
- Use the following initial parameter values, where each row corresponds to the unit that each weight/bias feeds into:

$$\begin{array}{lll} w_{11}^1 = 6 & w_{12}^1 = 8 & \theta_1^1 = 2 \\ w_{21}^1 = -6 & w_{22}^1 = -8 & \theta_2^1 = -1 \\ w_{11}^2 = 6 & w_{12}^2 = -6 & \theta^2 = -2 \end{array}$$

- The learning rate is set to $\eta = 0.7$.

Solution 2.1 If the combined inputs for the hidden layer are $u_i^{(1)}$ for $i = 1, 2$, then

$$u_i^{(1)} := \sum_{j=1}^{2} (w_{ij}^1 x_j) - \theta_i^1$$

and the output from these units is $g(u_j^{(1)})$. For the output layer, with only one unit, the combined input is

$$u^{(2)} := \sum_{j=1}^{2} (w_{1j}^2 g(u_j^{(1)})) - \theta^2$$

and the final output is

$$y = g(u^{(2)}).$$

Note that the superscripts on w_{ij}^2, θ^2, and $u^{(2)}$ mean "for units in the second layer" rather than "squared". Parentheses have been added to the superscripts on the u terms to make that more obvious.

Solution 2.2 As mentioned previously, we're dividing the traditional MSE by two, so our formula is

$$E(w) = \frac{1}{2N} \sum_{p=1}^{N} (t^p - y^p)^2$$

where N is the total number of outputs and p denotes the p^{th} pattern. In this case, the superscript 2 does mean "squared". Note that if we're updating our weights incrementally after each different input, and there's only one output, then $N = 1$ and we don't have to sum over anything.

Solution 2.3 The derivative of $g(u)$, which we can express as $g'(u)$, is $g(u)(1-g(u))$. Remember that this is specific to the specific kind of sigmoid function we're using.

Solution 2.4 We want to compute the change Δw_{ij}^l to each weight (we add the change to each weight for the next round), where l is the layer of the weight (or the layer of the unit it feeds into), i is the index of the unit within that layer, and j is the index of the unit in the preceding layer, whose output the weight is multiplying. To compute Δw, we need the change in the MSE as a function of the current weight we're considering. We can factor that change-in-error into two parts: (1) the change in the error as a function of the activation of the unit that the weight feeds into, and (2) the change in activation as a function of the change in the weight. The first is the δ term below, and the second is just the input that the weight is multiplying. Specifically:

$$
\begin{aligned}
\Delta w_{11}^2 &= \eta \delta_3 g(u_1^{(1)}) && \text{(first hidden node to the output node)} \\
\Delta w_{12}^2 &= \eta \delta_3 g(u_2^{(1)}) && \text{(second hidden node to the output node)} \\
\Delta w_{ij}^1 &= \eta \delta_i x_j && \text{(input node } j \text{ to to hidden node } i\text{; there are 4 of these)} \\
\Delta \theta^2 &= -\eta \delta_3 && \text{(bias to output node)} \\
\Delta \theta_1^1 &= -\eta \delta_1 && \text{(bias to output node 1)} \\
\Delta \theta_2^1 &= -\eta \delta_2 && \text{(bias to hidden node 2)}
\end{aligned}
$$

In the case of the output node weights, this is the same as the rule for the a single-layer network (with a sigmoid activation rule). The δ_3 is the decrease in the error as a function of the total activation, or

$$\delta_3 = g'(u^{(2)})(t - g(u^{(2)})) = g(u^{(2)})(1 - g(u^{(2)}))(t - g(u^{(2)})).$$

For the weights that are "earlier" in the network, we can re-use the delta term; an earlier node is contributing to greater error if it increases the activation of node for which higher activation means higher error.

$$
\begin{aligned}
\delta_1 &= g'(u_1^{(1)}) w_{11}^2 \delta_3 \\
\delta_2 &= g'(u_2^{(1)}) w_{12}^2 \delta_3
\end{aligned}
$$

Solution 2.5 We now have expressions for the changes in all of our weights after presenting the network with inputs and a target output ($w_{ij}^l \leftarrow w_{ij}^l + \Delta w_{ij}^l$ after each of the N inputs.) See the spreadsheet `multilayer-sandbox.xlsx` for the actual computations.