

## Lab Class ML:VI

By 2019-01-09 11:59 am, solutions for the following exercises have to be submitted: 1, 4.

## Exercise 1 : Perceptron Learning

Solve the following problems on learning boolean functions with perceptrons. Use the values 0 for *false* and 1 for *true*, and the threshold function  $\varphi(x) = \max(\text{sign}(x), 0)$ .

- (a) Design a single perceptron with two inputs  $x_A$  and  $x_B$ . This perceptron shall implement the boolean formula  $A \wedge \neg B$  with a suitable function  $y(x_A, x_B)$ . Hint: to start, determine the training data and draw it, and a suitable decision boundary, in a coordinate system; then, determine a set of suitable weights  $\mathbf{w} = (w_0, w_1, w_2)$ .
- (b) Train the perceptron with two iterations of the batch gradient descent algorithm, with a learning rate  $\eta$  of 0.1 and the weights initialized with  $w_0 = -0.5$  and  $w_1 = w_2 = 0.5$ . Use the following examples in the given order:

$x_1$	$x_2$	$c(\mathbf{x})$
0	0	0
0	1	0
1	0	1
1	1	0

- (c) Why can the boolean formula  $A \text{ XOR } B$  not be learned by a single perceptron? Justify your answer with a drawing.

## Exercise 2 : Neural Networks

Neural networks correctly classify only the feature vectors of a linearly separable training set  $D$ .

- false
- true
- true, if  $D$  is very large
- depends on the number of output units

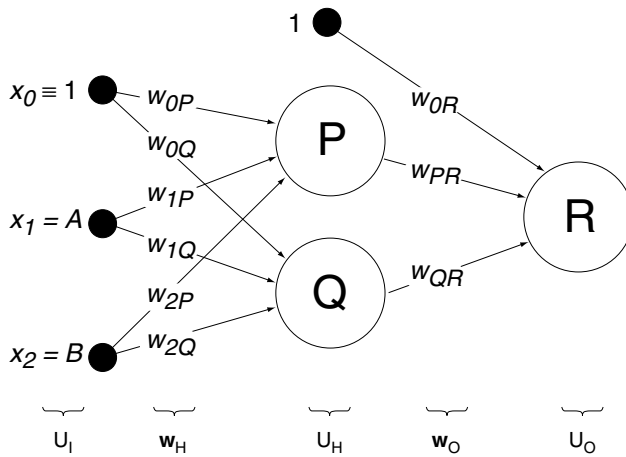
## Exercise 3 : Neural Networks

Which of the statements are true?

- Set-based gradient descent can approximate incremental gradient descent.
- Incremental gradient descent can approximate set-based gradient descent.
- Gradient descent always finds the global optimum.
- The perceptron algorithm converges always on linear separable data.

Exercise 4 : **P** Multilayer Neural Networks

Your task is to approximate the boolean formula  $A \text{ XOR } B$  using a two-layer neural network with the following architecture:



Following the notation used in the lecture:

$$\begin{aligned}
 U_I &= \{x_0, x_1, x_2\} \\
 U_H &= \{x_0, y_P, y_Q\} \\
 U_O &= \{y_R\} \\
 \mathbf{w} &= \mathbf{w}_H \cup \mathbf{w}_O \\
 \mathbf{w}_H &= \{w_{0P}, w_{0Q}, w_{1P}, w_{1Q}, w_{2P}, w_{2Q}\} \\
 \mathbf{w}_O &= \{w_{0R}, w_{PR}, w_{QR}\}
 \end{aligned}$$

For thresholding, we use the sigmoid function  $\sigma(x) = \frac{1}{1+e^{-x}}$ . Hence, for a given input  $x$  and weight vector  $\mathbf{w}$ , the network output  $y_R$  can be written as:

$$\begin{aligned}
 y_R(\mathbf{x}, \mathbf{w}) &= \sigma(w_{0R} + w_{PR} \cdot y_P(\mathbf{x}, \mathbf{w}_H) + w_{QR} \cdot y_Q(\mathbf{x}, \mathbf{w}_H)) \\
 &= \sigma(w_{0R} + w_{PR} \cdot \sigma(w_{0P} + w_{1P}x_1 + w_{2P}x_2) + w_{QR} \cdot \sigma(w_{0Q} + w_{1Q}x_1 + w_{2Q}x_2))
 \end{aligned}$$

- (a) For the 9 elements of  $\mathbf{w}$ , first determine a set of suitable values by hand, so that all examples are classified correctly. Assume the classification rule

$$\hat{c}(x) = \begin{cases} 0 & \text{if } y_R(\mathbf{x}, \mathbf{w}) \leq 0.5 \\ 1 & \text{if } y_R(\mathbf{x}, \mathbf{w}) > 0.5 \end{cases}$$

*Hints: first determine the training set  $D = \{(x, c(x)) \mid x = (1, A, B); c(x) = A \text{ XOR } B\}$  for all possible  $A, B$ . Then, decompose the XOR function into simpler boolean functions, and set the weights  $\mathbf{w}_H$  so that  $y_P$  and  $y_Q$  operate accordingly. Finally, set the weights  $\mathbf{w}_O$  to get the correct  $y_R$ .*

- (b) Implement the weight adaptation via batch gradient descent to find a set of weights for the XOR problem automatically. Employ the error function

$$\text{Err}(\mathbf{w}) = \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - y_R(\mathbf{x}))^2$$

Use the pseudocode for backpropagation with incremental gradient descent given in the [Lecturenotes](#) for guidance, and consider the following hints:

- Represent the training set as a pair of two-dimensional numpy arrays with shapes (4, 3) and (4, 1), e.g.:

```
inputs = array([[1, 0, 0],
                [1, 0, 1],
                ...   ])
outputs = array([[0],
                 [1],
                 ...])
```

- Represent the two weight vectors  $w_H$  and  $w_O$  as two-dimensional numpy arrays  $W_H$  and  $W_O$  with shapes (3, 2) and (3, 1), and initialize them with random values in the range  $[-0.5, 0.5]$  (use an appropriate function from `numpy.random`). Define the `sigmoid` function using numpy array operations.
- Compute the forward pass as in the equation for  $y_R$  given above. Convince yourself that the hidden layer activations  $y_H = (y_P, y_Q)$  can be computed for the entire training set with a single dot product:

```
y_H = sigmoid( np.dot(inputs, W_H) )
```

However, note that  $y_H$  must receive an extra column of ones before  $y_R$  can be computed in a similar fashion.

- The goal of the backpropagation pass then is to compute an adjustment to each element of  $w$  according to  $\frac{\partial Err}{\partial w}$ . Refer to the lecture slides, but keep the difference between incremental and batch gradient descent in mind.
- Run the batch gradient descent for 10000 iterations for different random starting weights, and observe how  $Err$  changes over time. If your implementation fails to converge to a correct solution a lot of the time, adjust the learning rate  $\eta$ , and consider using the *momentum* weight adaptation discussed in the lecture.