

## Summary - The Learning Problem to Neural Networks /MLP

1. **Formalized the learning Problem:** Data Set:  $D = \{\mathbf{x}_i, y_i\}_{i=1}^N$ , Learning Model  $\mathcal{L}$  is a collection of hypotheses, Learning Algorithm  $\mathcal{A}$  takes as input  $D, \mathcal{L}$  and outputs the learned function  $g \in \mathcal{L}$ . The learned function  $g$  is supposed to approximate the target process  $f$ .
2. **Task:** Pick a suitable  $\mathcal{L}$  and  $\mathcal{A}$ .
3. **No Free Lunch (NFL):** If the target function is random, then there is nothing that you can do -  $P_{OTS} = \frac{1}{2}$ . Conversely, given a target function, if your learning model contains every function, then  $P_{OTS} = \frac{1}{2}$  for a learning algorithm that is based on the training set error. Thus, one needs to assume some properties of the target function, *and* one needs to incorporate these properties into  $\mathcal{L}$  and/or  $\mathcal{A}$ .
4. **Bayes Optimal Approach:** Pick action  $\alpha$  to minimize the conditional risk  $R(\alpha|\mathbf{x})$ , given the risk matrix.
5. **0-1 Risk Matrix and Gaussian Class Conditionals:** Assuming that  $\lambda_{ij} = 1$  unless  $i = j$  and that the class conditional  $p(x|w_i)$  is a Gaussian, we get the perceptron decision rule. The parameters can be estimated from the data.
6. **Perceptron Learning Model:** Thus we arrived at the perceptron for our first learning model.

$$\mathcal{L} = \{g(\mathbf{x}; \mathbf{w}) | g(\mathbf{x}; \mathbf{w}) = \text{sign}(\mathbf{w}^T \mathbf{x}), \mathbf{w} \in \mathbf{R}^d\}$$

Forget about Bayes except that we know that we want to minimize the probability of error. Thus the algorithm  $\mathcal{A}$  will be to minimize the training probability of error,  $R_{emp}$  - that is, to output a  $\mathbf{w}$  for which  $R_{emp}(\mathbf{w})$  is a minimum.

**Is this a good thing to do? - i.e., is  $R_{emp} \approx R$  guaranteed to be true?**

**How does one perform the minimization?**

7. **Perceptron Learning Algorithm:** If  $R_{emp} = 0$  is possible with the perceptron, then guaranteed to get there in finite time.
8. **Gradient Descent Algorithm:** First soften the threshold ( $\text{sign}(\cdot) \rightarrow \text{tanh}(\cdot)$ ) to make  $R_{emp}$  a smooth function. Then, minimize the  $R_{emp}$  by iteratively taking small steps in the direction of the negative gradient. The final decision function can be evaluated with the  $\text{sign}(\cdot)$  instead of the  $\text{tanh}(\cdot)$ .
9. **Limitations of the Perceptron Learning Model:** Cannot implement simple functions such as  $f(\mathbf{x}) = \text{sign}(x_1)\text{sign}(x_2)$  or  $f(\mathbf{x}) = \text{sign}(\|\mathbf{x}\|^2 - 1)$ .
10. **Generalization to the MLP:** Many layers possible, arbitrary number of nodes allowed. The function is specified by its weights,  $g(\mathbf{x}, \mathbf{w})$ . The task is to then pick a set of weights.

**Computation of Function:** Given by a recursive algorithm - Forward propagation to compute  $x_i^l$  and  $s_j^l$ . The final output is  $x_1^L$ .

**Algorithm  $\mathcal{A}$ :** Minimize  $R_{emp}(\mathbf{w})$ . To do this we need to be able to compute the gradient of the output  $x_1^L$  with respect to every weight  $w_{\alpha\beta}^m$ .

**Backpropagation:** Computation of the gradients using a recursive backpropagation algorithm that computes the  $\delta_j^l$ 's.

11. **Universal Approximation:** For any given continuous function  $f(\mathbf{x})$ , one can choose a neural network such that for some assignment of the weights, the function implemented is close to the give function  $f$ . The approximation capability clearly increases with more nodes and more layers. Thus, why not start with an arbitrarily large network? Answer, one falls into the NFL problem of including all the functions in your learning mode and then  $P_{OTS} = 1/2$ . Thus one needs to restrict the size of the MLP.

12. **General Approach:** We would like to study many learning models and learning algorithms. Our general requirement will be

- (a) The learning model is expressive. Thus, in principle we could implement any function though on NFL grounds we would like to restrict the learning model.
- (b) There is a good learning algorithm that can be implemented easily from the computational viewpoint.

Our general learning algorithm will be to minimize  $R_{emp}$ . The MLP is our first example that satisfies the criteria using this general learning algorithm.

13. **Questions we might ask:**

- (a) What are other good algorithms?
- (b) How do we restrict the learning models so as not to fall into NFL?
- (c) How do we know that minimizing  $R_{emp}$  is a good thing? - for example does it correspond to minimizing  $R$ ?