

Neural Networks

The most standard, and straightforward, type of network model is the two-layer feedforward network. Here, inputs are completely connected to K -many internal (or hidden) units by weights; let w_{ij} denote the weight from input x_i to hidden unit j , so \mathbf{W} is a matrix of weights. Then each hidden unit is connected to the output with some weight u_j , put in a vector \mathbf{u} . A non-linearity is placed on the hidden units, often a logistic, though sometimes a tanh or even just a threshold. The final prediction is:

$$\begin{aligned} f(\mathbf{x}) &= \sum_j u_j \sigma\left(\sum_i w_{ij} x_i\right) \\ &= \mathbf{u}^\top \sigma(\mathbf{W}\mathbf{x}) \end{aligned}$$

We can then place an error on the output, exactly as before. This can be logistic or hinge (leading to classification) or squared (leading to regression).

Important properties of this model:

- Because of the σ , this model is *non-linear*
- It is also not convex in either \mathbf{W} or \mathbf{u}
- There are lots of symmetries in the model
- The more hidden units, the more non-linearity
- You can approximate *any* function with a sufficiently large (either wide or deep) network

The easiest way to train this beast is gradient descent; this leads to the backpropagation algorithm (which we'll see next time).

Many more architectures are possible, however. Deep networks have recently become popular (there will be a brain teaser about them), and undirected models known as Boltzmann machines (or Helmholtz machines) are very common in unsupervised learning (they also have nice connections to statistical physics).

Controlling complexity in neural networks can be done in many ways:

- Explicit regularization of the weights, eg with an ℓ_1 or ℓ_2 penalty
- Early stopping of the optimization procedure (a poor-man's approximation to regularization)
- Trimming the number of hidden units