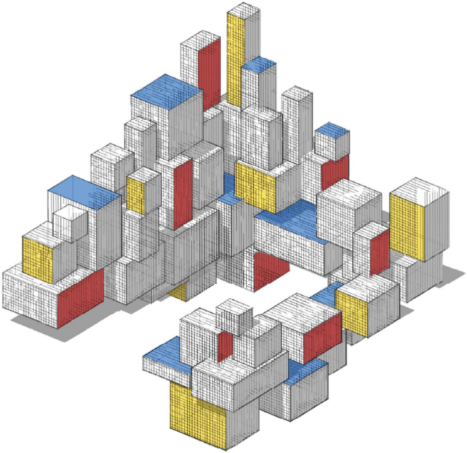
Deep Learning



Purpose

In this lecture we introduce the rich class of approximating functions called neural networks.

The learners belonging to the neural-network class of functions have attractive properties that have made them ubiquitous in modern machine learning applications — their training is computationally feasible and their complexity is easy to control and fine-tune.

Topics include:

- Activation functions
- Feed-forward neural networks
- Example applications:
  - Nonlinear multi-output regression
  - Multi-logit classification
  - Density estimation

Supervised Learning

Recall the basic supervised learning task: predict a random output Y from a random input $\boldsymbol{X}$, using a prediction function $g : \boldsymbol{x} \mapsto y$ that belongs to a suitably chosen class of approximating functions $\mathcal{G}$.

More generally, we may wish to predict a vector-valued output $\boldsymbol{y}$ using a prediction function $\boldsymbol{g} : \boldsymbol{x} \mapsto \boldsymbol{y}$ from class $\mathcal{G}$.

> In what follows, $\boldsymbol{y}$ denotes the vector-valued output for a given input $\boldsymbol{x}$. This differs from our previous use, where $\boldsymbol{y}$ denotes a vector of scalar outputs.

The class $\mathcal{G}$ is sometimes referred to as the hypothesis space or the universe of possible models, and the representational capacity of a hypothesis space $\mathcal{G}$ is simply its complexity.

Approximation–Estimation Tradeoff

Suppose that we have a class of functions $\mathcal{G}_L$, indexed by a parameter L such that $\mathcal{G}_L \subset \mathcal{G}_{L+1} \subset \mathcal{G}_{L+2} \subset \cdots$.

In selecting a suitable class of functions, we have a approximation–estimation tradeoff:

- The class $\mathcal{G}_L$ must be complex (rich) enough to accurately represent the optimal unknown prediction function g^*, which may require a very large L.
- The learners in the class $\mathcal{G}_L$ must be simple enough to train with small estimation error and with minimal demands on computer memory, which may necessitate a small L.

A class of functions that permits such a natural hierarchical construction is the class of neural networks.

Neural Network

Conceptually, a neural network with L layers is a nonlinear parametric regression model whose representational capacity is controlled by L.

Alternatively, we will define the output of a neural network as the repeated composition of linear and (componentwise) nonlinear functions.

This provides a flexible class of nonlinear output functions that can be easily differentiated.

As a result, the training of learners via gradient optimization methods involves mostly standard matrix operations that can be performed very efficiently.

Neural networks were originally intended to mimic the workings of the human brain, with the network nodes modeling neurons and the network links modeling the axons connecting neurons.

Kolmogorov’s Approximation

Many effective machine algorithms have been inspired by mathematical ideas for function approximation.

Theorem: Kolmogorov (1957)

kolmogorov Every continuous function $g^* : [0,1]^p \mapsto \mathbb{R}$ with $p \geqslant 2$ can be written as

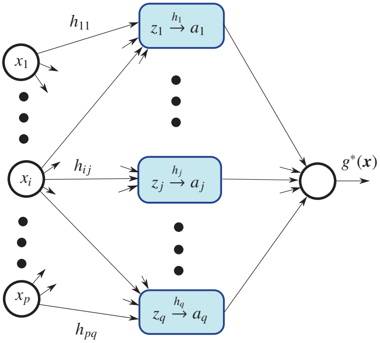
$$g^*(\boldsymbol{x}) = \sum_{j=1}^{2p+1} h_j \left(\sum_{i=1}^{p} h_{ij}(x_i) \right),$$

where $\{h_j, h_{ij}\}$ is a set of univariate continuous functions that depend on g^*.

This result tells us that any continuous high-dimensional map can be represented as the function composition of much simpler (1D) maps.

Representation of Kolmogorov's Approximation

A neural network representation of the composition of the maps needed to compute the output $g^*(\boldsymbol{x})$ for a given input $\boldsymbol{x} \in \mathbb{R}^p$.



h_{11}
$z_1 \xrightarrow{h_1} a_1$
x_1
x_i
h_{ij}
$z_j \xrightarrow{h_j} a_j$
$g^*(\boldsymbol{x})$
x_p
$z_q \xrightarrow{h_q} a_q$
h_{pq}

Figure: Every continuous function $g^* : [0,1]^p \mapsto \mathbb{R}$ can be represented by a neural network with one hidden layer ($l = 1$), an input layer ($l = 0$), and an output layer ($l = 2$).

In/Hidden/Output Layers

- Each of the p components of the input $\boldsymbol{x}$ is represented as a node in the input layer ($l = 0$).
- In the hidden layer ($l = 1$) there are $q := 2p + 1$ nodes, each of which is associated with a pair of variables (z, a) with values

$$z_j := \sum_{i=1}^{p} h_{ij}(x_i) \quad \text{and} \quad a_j := h_j(z_j).$$

  A link between nodes (z_j, a_j) and x_i with weight h_{ij} signifies that the value of z_j depends on the value of x_i via function h_{ij}.
- The output layer ($l = 2$) represents the value $g^*(\boldsymbol{x}) = \sum_{j=1}^{q} a_j$.

Activation Functions

In practice, we do not know the collection of (generally nonlinear) functions $\{h_i, h_{ij}\}$, because they depend on the unknown g^*.

Kolmogorov's theorem only asserts the existence of $\{h_j, h_{ij}\}$, and does not tell us how to construct these nonlinear functions.

One way out of this predicament is to replace these $(2p+1)(p+1)$ unknown functions with a much larger number of *known* nonlinear functions called activation functions.

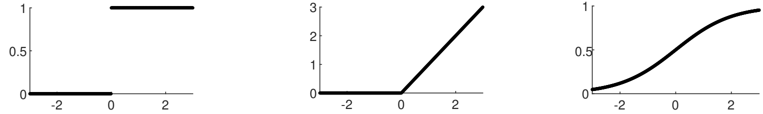
An example is the logistic activation function:

$$S(z) = (1 + \exp(-z))^{-1}.$$

We hope that a network built from a sufficiently large number of activation functions will have similar representational capacity as Kolmogorov's approximation network with $(2p+1)(p+1)$ functions.

Activation Functions

In general, we wish to use the simplest activation functions that will allow us to build a learner with large representational capacity and low training cost. There are many choices of activation functions.



Heaviside or unit step — $\mathbb{I}\{z \geqslant 0\}$

rectified linear unit (ReLU) — $z \times \mathbb{I}\{z \geqslant 0\}$

logistic — $(1 + \exp(-z))^{-1}$

Another way to improve the representational capacity of the network is to introduce more hidden layers.

Feed-Forward Neural Networks

In a neural network with $L + 1$ layers, the input layer ($l = 0$) encodes the input feature vector $\boldsymbol{x}$, and the output layer ($l = L$) encodes the output function $\boldsymbol{g}(\boldsymbol{x})$. The remaining layers are hidden layers.

Each layer l has p_l nodes.

All nodes in the hidden layers are associated with a pair of variables (z, a), gathered into p_l-dimensional column vectors $\boldsymbol{z}_l$ and $\boldsymbol{a}_l$.

In feed-forward networks, the variables in any layer l are simple functions of the variables in the preceding layer $l - 1$. In particular, $\boldsymbol{z}_l$ and $\boldsymbol{a}_{l-1}$ are related via the linear relation

$$\boldsymbol{z}_l = \mathbf{W}_l \, \boldsymbol{a}_{l-1} + \boldsymbol{b}_l,$$

for some weight matrix $\mathbf{W}_l$ and bias vector $\boldsymbol{b}_l$.

Feed-Forward Neural Networks

Within any hidden layer $l = 1, \dots, L-1$, the components of the vectors $\boldsymbol{z}_l$ and $\boldsymbol{a}_l$ are related via

$$\boldsymbol{a}_l = \boldsymbol{S}_l(\boldsymbol{z}_l),$$

where $\boldsymbol{S}_l : \mathbb{R}^{p_l} \mapsto \mathbb{R}^{p_l}$ is a nonlinear multivalued function.

All of these multivalued functions are typically of the form

$$\boldsymbol{S}_l(\boldsymbol{z}) = [S(z_1), \dots, S(z_{\dim(\boldsymbol{z})})]^\top, \quad l = 1, \dots, L-1, \tag{1}$$

where S is an activation function common to all hidden layers.

The function $\boldsymbol{S}_L : \mathbb{R}^{p_{L-1}} \mapsto \mathbb{R}^{p_L}$ in the output layer is more general and its specification depends, for example, on whether the network is used for classification or for the prediction of a continuous output Y.

Feed-Forward Neural Networks

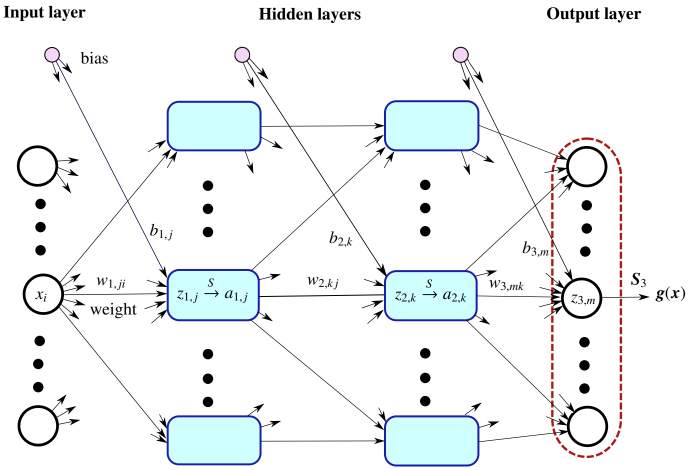


Figure: A neural network with $L = 3$: the $l = 0$ layer is the input layer, followed by two hidden layers, and the output layer. Hidden layers may have different numbers of nodes.

Output of a Feed-Forward Neural Network

The output of a feed-forward neural network is thus determined by the input vector $\boldsymbol{x}$, (nonlinear) functions $\{\boldsymbol{S}_l\}$, as well as weight matrices $\mathbf{W}_l = [w_{l,ij}]$ and bias vectors $\boldsymbol{b}_l = [b_{l,j}]$ for $l = 1, 2, 3$.

The (i, j)-th element of the weight matrix $\mathbf{W}_l = [w_{l,ij}]$ is the weight that connects the j-th node in the l-th layer with the i-th node in the $(l + 1)$-st layer.

The name given to L (the number of layers without the input layer) is the network depth and $\max_l p_l$ is called the network width.

While we mostly study networks that have an equal number of nodes in the hidden layers ($p_1 = \cdots = p_{L-1}$), in general there can be different numbers of nodes in each hidden layer.

Function Composition

The output $\boldsymbol{g}(\boldsymbol{x})$ of a multiple-layer neural network is obtained from the input $\boldsymbol{x}$ via the following sequence of computations:

$$\begin{aligned} &\underbrace{\boldsymbol{x}}_{\boldsymbol{a}_0} \to \underbrace{\mathbf{W}_1\,\boldsymbol{a}_0 + \boldsymbol{b}_1}_{\boldsymbol{z}_1} \to \underbrace{\boldsymbol{S}_1(\boldsymbol{z}_1)}_{\boldsymbol{a}_1} \to \underbrace{\mathbf{W}_2\,\boldsymbol{a}_1 + \boldsymbol{b}_2}_{\boldsymbol{z}_2} \to \underbrace{\boldsymbol{S}_2(\boldsymbol{z}_2)}_{\boldsymbol{a}_2} \to \cdots \\ &\quad \to \underbrace{\mathbf{W}_L\,\boldsymbol{a}_{L-1} + \boldsymbol{b}_L}_{\boldsymbol{z}_L} \to \underbrace{\boldsymbol{S}_L(\boldsymbol{z}_L)}_{\boldsymbol{a}_L} = \boldsymbol{g}(\boldsymbol{x}). \end{aligned} \tag{2}$$

Denoting the function $\boldsymbol{z} \mapsto \mathbf{W}_l\,\boldsymbol{z} + \boldsymbol{b}_l$ by $\boldsymbol{M}_l$, the output $\boldsymbol{g}(\boldsymbol{x})$ can thus be written as the function composition

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{S}_L \circ \boldsymbol{M}_L \circ \cdots \circ \boldsymbol{S}_2 \circ \boldsymbol{M}_2 \circ \boldsymbol{S}_1 \circ \boldsymbol{M}_1(\boldsymbol{x}). \tag{3}$$

The algorithm for computing the output $\boldsymbol{g}(\boldsymbol{x})$ for an input $\boldsymbol{x}$ is called feed-forward propagation.

Algorithm 1: Feed-Forward Propagation for a Neural Network

```
Input: Feature vector $\boldsymbol{x}$; weights $\{w_{l,ij}\}$, biases $\{b_{l,i}\}$ for each
       layer $l = 1, \ldots, L$.
Output: The value of the prediction function $\boldsymbol{g}(\boldsymbol{x})$.
$\boldsymbol{a}_0 \leftarrow \boldsymbol{x}$                // the zero or input layer
for $l = 1$ to $L$ do
    Compute the hidden variable $z_{l,i}$ for each node $i$ in layer $l$:
        $\boldsymbol{z}_l \leftarrow \mathbf{W}_l\, \boldsymbol{a}_{l-1} + \boldsymbol{b}_l$
    Compute the activation function $a_{l,i}$ for each node $i$ in layer $l$:
        $\boldsymbol{a}_l \leftarrow \boldsymbol{S}_l(\boldsymbol{z}_l)$
return $\boldsymbol{g}(\boldsymbol{x}) \leftarrow \boldsymbol{a}_L$                // the output layer
```

Example: Nonlinear Multi-Output Regression

Given the input $\boldsymbol{x} \in \mathbb{R}^{p_0}$ and an activation function $S : \mathbb{R} \mapsto \mathbb{R}$, the output $\boldsymbol{g}(\boldsymbol{x}) := [g_1(\boldsymbol{x}), \ldots, g_{p_2}(\boldsymbol{x})]^\top$ of a nonlinear multi-output regression model can be computed via a neural network with:

$$\begin{aligned} \boldsymbol{z}_1 &= \mathbf{W}_1 \boldsymbol{x} + \boldsymbol{b}_1, \quad \text{where } \mathbf{W}_1 \in \mathbb{R}^{p_1 \times p_0}, \boldsymbol{b}_1 \in \mathbb{R}^{p_1}, \\ a_{1,k} &= S(z_{1,k}), \quad k = 1, \ldots, p_1, \\ \boldsymbol{g}(\boldsymbol{x}) &= \mathbf{W}_2 \boldsymbol{a}_1 + \boldsymbol{b}_2, \quad \text{where } \mathbf{W}_2 \in \mathbb{R}^{p_2 \times p_1}, \boldsymbol{b}_2 \in \mathbb{R}^{p_2}, \end{aligned}$$

which is a neural network with one hidden layer and output function $\boldsymbol{S}_2(\boldsymbol{z}) = \boldsymbol{z}$.

In the special case where $p_1 = p_2 = 1$, $\boldsymbol{b}_2 = 0$, $\mathbf{W}_2 = 1$, and we collect all parameters into the vector $\boldsymbol{\theta}^\top = [\boldsymbol{b}_1, \mathbf{W}_1] \in \mathbb{R}^{p_0+1}$, the neural network can be interpreted as a generalized linear modelwith $\mathbb{E}[Y \mid \boldsymbol{X} = \boldsymbol{x}] = h([1, \boldsymbol{x}^\top]\, \boldsymbol{\theta})$ for some activation function h.

Example: Multi-Logit Classification

Suppose that, for a classification problem, an input $\boldsymbol{x}$ has to be classified into one of c classes, labeled $0, \ldots, c-1$. We can perform the classification via a neural network with one hidden layer, with $p_1 = c$ nodes. In particular, we have

$$\boldsymbol{z}_1 = \mathbf{W}_1 \boldsymbol{x} + \boldsymbol{b}_1, \quad \boldsymbol{a}_1 = \boldsymbol{S}_1(\boldsymbol{z}_1),$$

where $\boldsymbol{S}_1$ is the softmax function:

$$\text{softmax} : \boldsymbol{z} \mapsto \frac{\exp(\boldsymbol{z})}{\sum_k \exp(z_k)}.$$

For the output, we take $\boldsymbol{g}(\boldsymbol{x}) = [g_1(\boldsymbol{x}), \ldots, g_c(\boldsymbol{x})]^\top = \boldsymbol{a}_1$, which can then be used as a pre-classifier of $\boldsymbol{x}$. The actual classifier of $\boldsymbol{x}$ into one of the categories $0, 1, \ldots, c-1$ is then

$$\underset{k \in \{0,\ldots,c-1\}}{\operatorname{argmax}} \; g_{k+1}(\boldsymbol{x}).$$

This is equivalent to the multi-logit classifier.

Example: Density Estimation

A Gaussian mixture density with p_1 components and a common scale parameter $\sigma > 0$ can be viewed as an NN with two hidden layers.

Let the activation function in the first hidden layer, $\boldsymbol{S}_1$, be of the form (1) with $S(z) := \exp(-z^2/(2\sigma^2))/\sqrt{2\pi\sigma^2}$. Then the density value $g(x)$ is computed via:

$$\begin{aligned} \boldsymbol{z}_1 &= \mathbf{W}_1\, x + \boldsymbol{b}_1, \qquad \boldsymbol{a}_1 = \boldsymbol{S}_1(\boldsymbol{z}_1), \\ \boldsymbol{z}_2 &= \mathbf{W}_2\, \boldsymbol{a}_1 + \boldsymbol{b}_2, \quad \boldsymbol{a}_2 = \boldsymbol{S}_2(\boldsymbol{z}_2), \\ g(x) &= \boldsymbol{a}_1^\top \boldsymbol{a}_2, \end{aligned}$$

where $\mathbf{W}_1 = \mathbf{1}$ is a $p_1 \times 1$ column vector of ones, $\mathbf{W}_2 = \mathbf{O}$ is a $p_1 \times p_1$ matrix of zeros, and $\boldsymbol{S}_2$ is the softmax function.

We identify the column vector $\boldsymbol{b}_1$ with the p_1 location parameters, $[\mu_1, \ldots, \mu_{p_1}]^\top$ of the Gaussian mixture and $\boldsymbol{b}_2 \in \mathbb{R}^{p_1}$ with the p_1 weights of the mixture.

Network Architecture

The network architecture is an important design consideration in neural networks.

For example, if the connectivity from one layer to the next is sparse and the links share the same weight values $\{w_{l,ij}\}$ (called parameter sharing) for all $\{(i,j) : |i-j| = 0, 1, \ldots\}$, then the weight matrices will be sparse and Toeplitz, allowing fast matrix-vector product calculations, e.g., via the fast Fourier transform.

An important example is the convolution neural network (CNN), in which the network layers encode the convolution operation:

$$\mathbf{W}_l \, \boldsymbol{a}_{l-1} = \boldsymbol{w}_l * \boldsymbol{a}_{l-1},$$

where $[\boldsymbol{x} * \boldsymbol{y}]_i := \sum_k x_k y_{i-k+1}$.

Convolution Neural Networks

CNNs are particularly suited to image processing problems.

Suppose that the input image is given by an $m_1 \times m_2$ matrix of pixels.

Define a $k \times k$ matrix (sometimes called a kernel, where k is generally taken to be 3 or 5).

Then, the convolution layer output can be calculated using the discrete convolution of all possible $k \times k$ input matrix regions and the kernel matrix; the convolution layer output size is $(m_1 - k + 1) \times (m_2 - k + 1)$.

The figure shows a 5×5 input image and a 2×2 kernel with a 4×4 output matrix.