Lecture 2: Neural network

谢丹
清华大学数学系

2025/08

Linear Model Assumptions

Basic Assumptions

The linear model makes two key assumptions:

1. The target variable follows a Gaussian distribution
2. The mean depends linearly on the input x:

$$\mathbb{E}[y|x] = w^T x$$

Limitations

In many cases, these assumptions don't hold. One solution is to introduce nonlinear basis functions ϕ_i:

$$y = \sum_i w_i \phi_i(x)$$

The choice of ϕ_i functions requires careful consideration.

Neural Networks Approach

An alternative approach is to use Neural Networks, which have proven to be remarkably powerful function approximators.

Neural Network Fundamentals

Definition

A neural network defines a parameterized function $y(x; w)$ where:

- x: Input variables
- y: Output variables
- w: Network parameters

Basic Components

1. **Nodes** (neurons): Computational units
2. **Edges**: Connections between nodes with weights w

Layered Structure

For a network with layer structure, the function at each node is:

$$z_k^{(i)} = h\left(\sum_j w_{kj}^{(i)} z_j^{(i-1)} + b_k^{(i)}\right)$$

Network Implementation

Simplified Notation

By adding a bias node (fixed at 1) to each layer, we get the compact form:

$$\boxed{z_k^{(i)} = h\left(\sum_j w_{kj}^{(i)} z_j^{(i-1)}\right)}$$

Feedforward Process

- Initialize with input layer ($i = 0$) values
- Recursively compute through hidden layers
- Final output at last layer
- Multiple output nodes produce vector outputs

Network Architecture Choices

1. **Structure Design**:
   - Number of layers and nodes per layer
   - Input layer size determined by data dimensionality
   - Deep Neural Network: Many hidden layers
2. **Activation Functions**:

Table: Common Activation Functions

Function	Formula
Logistic Sigmoid	$\sigma(a) = \frac{1}{1+\exp(-a)}$
Tanh	$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
Hard Tanh	$h(a) = \max(-1, \min(1, a))$
Softplus	$h(a) = \ln(1 + \exp(a))$
ReLU	$h(a) = \max(0, a)$
Leaky ReLU	$h(a) = \max(0, a) + \alpha \min(0, a)$

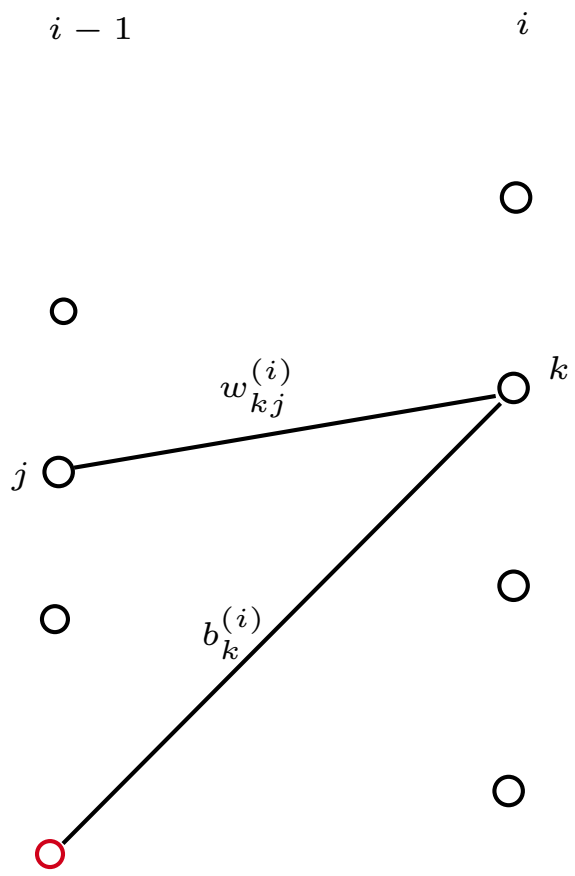
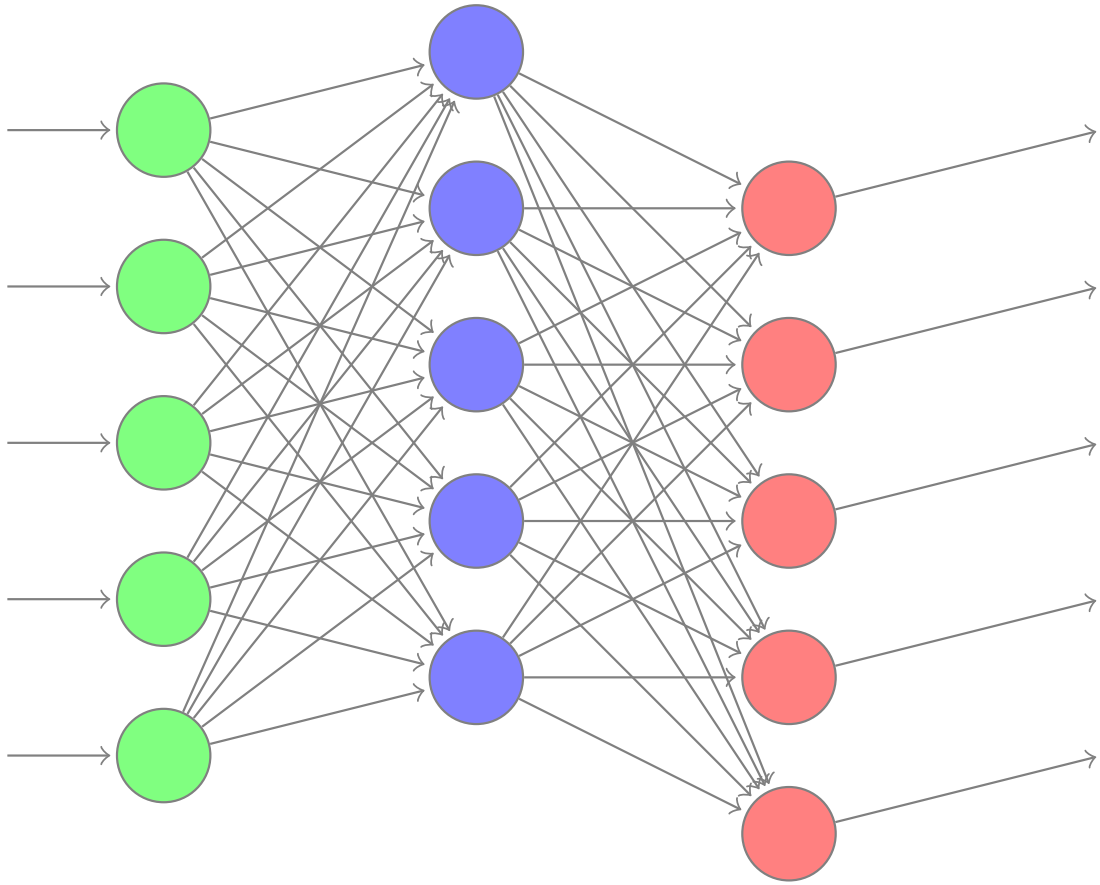


Figure: The building block of neural network.



Here is the probability model for the neural network:

$$P(y|\mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(y|y(x, w), \sigma^2)$$

where $y(x, w)$ is defined by using neural network. Similarly, one can get the probability model for the classification.

Neural Network Loss Function

General Formulation

For both regression and classification problems, the loss function generalizes naturally:

$$E(\mathbf{w}) = \sum_{n=1}^{N} (t_n - y(\mathbf{x}_n, \mathbf{w}))^2 + \lambda \sum_i w_i^2$$

where:

- $y(\mathbf{x}_n, \mathbf{w})$: Neural network output
- λ: Regularization parameter

Per-observation Contribution

The loss decomposes as $E = \sum E_n$ where:

$$E_n = \frac{1}{2}[t_n - y(\mathbf{x}_n, \mathbf{w})]^2 = \frac{1}{2}\left(t_n - h\left(\sum_j w_{kj}^{(f)} z_j^{(f-1)}\right)\right)^2$$

Optimization

The gradient descent method can be applied to minimize $E(\mathbf{w})$, with gradients computed efficiently via:

Backpropagation Algorithm

(based on chain rule differentiation)

对某一个参数 $w_{ji}^{(l)}$ 求导，利用微积分中的链式法则，有

$$\boxed{\frac{\partial E_n}{w_{kj}^{(i)}} = \frac{\partial E_n}{\partial a_k^{(i)}} \frac{\partial a_k^{(i)}}{\partial w_{kj}^{(i)}} = \frac{\partial E_n}{\partial a_k^{(i)}} z_j^{(i-1)} = \delta_k^{(i)} z_j^{(i-1)}}$$

这里我们利用了 $a_k^{(i)} = \sum_j w_{kj}^{(i)} z_j^{(i-1)}$. 这样偏导数就有一个局域的表达形式， 只需要在每一个节点上定义一个新的error项

$$\delta_k^{(i)} = \frac{\partial E_n}{\partial a_k^{(i)}} = \sum_l \frac{\partial E_n}{\partial a_l^{(i+1)}} \frac{\partial a_l^{(i+1)}}{\partial a_k^{(i)}} =$$

$$\sum_l \frac{\partial E_n}{\partial a_l^{(i+1)}} \frac{\partial a_l^{(i+1)}}{\partial a_k^{(i)}} = \sum_l \delta_l^{(i+1)} w_{lk}^{(i+1)} h'(a_k^{(i)})$$

这里我们用到了 $a_l^{(i+1)} = \sum_t w_{lt}^{(i+1)} h(a_t^{(i)})$. 上面的公式给了一个递归的方式来计算error, 但是和原来量不一样的是这个时候 初始的量为输出端。初始条件为

$$\delta_j^{(f)} = \frac{\partial E_n}{\partial a_j^{(f)}} = (t_n - y_n)$$

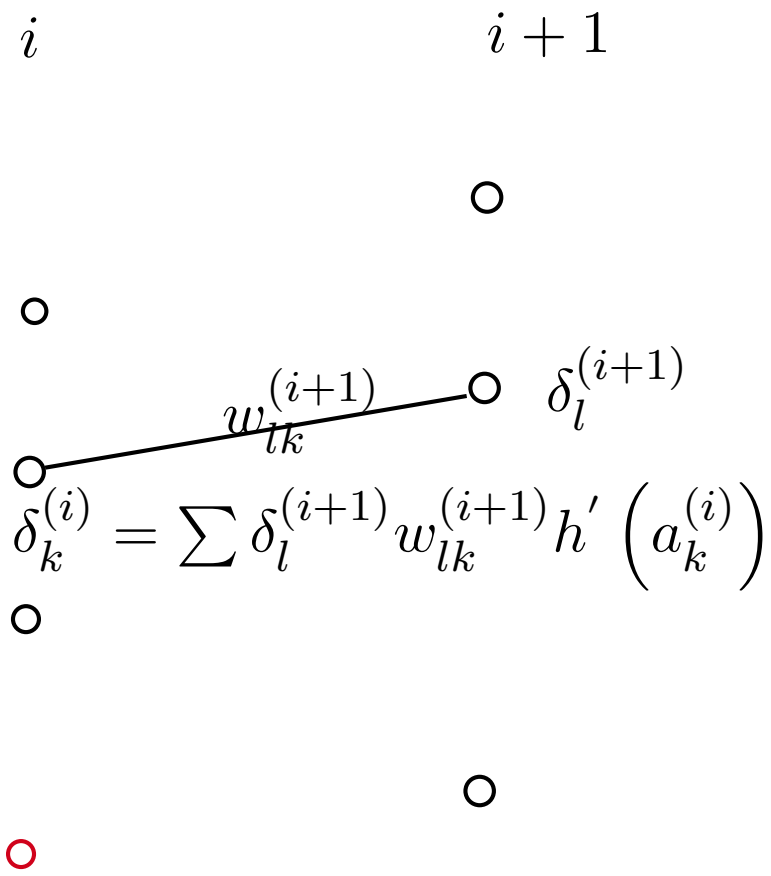


Figure: Backpropagation of gradient

Classification with Neural Networks

Probability Output

For a network with K output nodes $(y_i, i = 1, \ldots, K)$, we can obtain classification probabilities via softmax:

$$p_i = \frac{\exp(y_i)}{\sum_{j=1}^{K} \exp(y_j)}$$

Temperature Parameter

We can introduce temperature T to control output distribution:

$$p_i(T) = \frac{\exp(y_i/T)}{\sum_{j=1}^{K} \exp(y_j/T)}$$

- Higher T: Softer probabilities
- Lower T: More peaked distribution

Generative Applications

The network can parameterize many probability distribution, enabling powerful generative AI models: Text generation, diffusion model for image generation.

Deep Neural Networks

Function Representation

Neural networks provide a geometric framework for representing complex nonlinear functions: universal function approximator.

Depth Advantage

Empirically, deeper networks (more hidden layers) yield better performance for many tasks.

Training Challenges

1. **Gradient Issues**:
   - Vanishing gradients (too small)
   - Exploding gradients (too large)
2. **Computational Complexity**:
   - Millions to billions, trillions of parameters
   - Enormous computational requirements

Solutions

- Architectural innovations address gradient problems
- GPU acceleration enables large-scale training

Computational Considerations

Matrix Operations

- Neural networks primarily perform matrix multiplications
- These operations are highly parallelizable
- GPUs excel at parallel computation of these operations

Efficiency

- Modern GPUs can perform thousands of operations simultaneously
- Specialized tensor cores further accelerate training

Practical Example

Coming Next:

Hands-on Demonstration

of Neural Network Implementation