Neural Networks for Signal Processing – I

Prof. Shayan Srinivasa Garani

Department of Electronics Systems Engineering

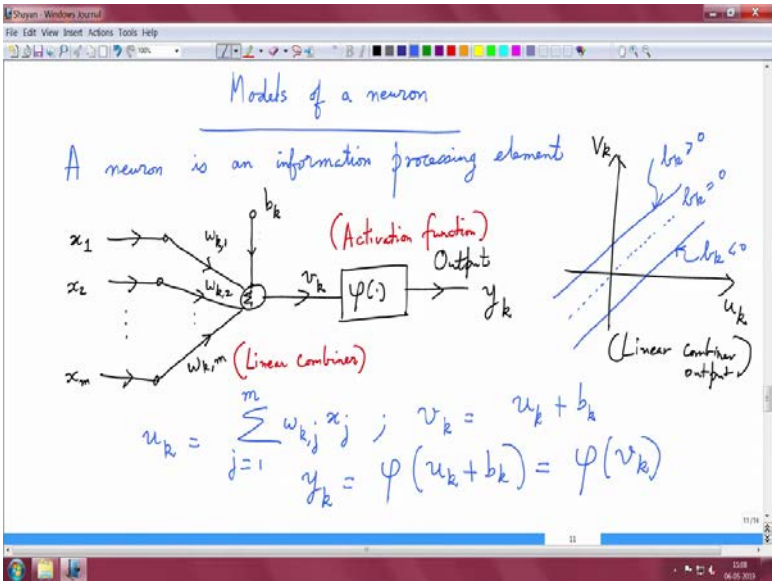
Indian Institute of Science, Bengaluru

Lecture – 03

Models of a Neuron

As we have explored the capabilities of neural networks, we understand that they function as sophisticated information processors. Now, let us delve into the intricacies of these neural networks. The fundamental building block of a neural network is the neuron, also referred to as a processing element. By starting with models for these neurons, we can build a network. Beginning with this basic unit, we aggregate a collection of neurons and integrate specific features, ultimately forming a neural network. This process transforms individual processing elements into a cohesive and intelligent system capable of complex computations and learning.

(Refer Slide Time: 01:01)



Let's delve into the foundational aspects of neural networks. A neuron, the fundamental unit of a neural network, functions as an information processing element. Now, what exactly is a neuron capable of? To describe it more precisely, consider that we have external inputs denoted as x_1, x_2, ..., x_m, each connected to the neuron through corresponding weights, which we'll

denote as $w_{k1}, w_{k2}, \ldots, w_{km}$. Here, k refers to the k^{th} neuron.

The signal x_1 is connected to the k-th neuron via the weight w_{k1}, and similarly for other inputs. These weights essentially scale the inputs, and we then add a bias term, denoted as b_k. At this stage, the neuron functions as a linear combiner, resulting in a signal v_k. This signal v_k is subsequently passed through an activation function, which we will discuss shortly. The output of this process is y_k.

This mechanism of input scaling, bias addition, and activation is fundamental to the functioning of neurons within a neural network.

Let's delve into a more refined description of a neuron. In its simplest form, a neuron acts as a basic unit of information processing. You might wonder why we start with a linear combiner. Could we not introduce some other non-linearity at this stage? The answer lies in the foundational nature of this model. Initially, we gather all stimuli, scale them with the appropriate weights, and add a bias. This bias adjusts the firing threshold of the neuron. The linear signal obtained is then passed through a typically non-linear activation function, resulting in the neuron's output response. This abstract model of a neuron has some biological plausibility.

Consider the points connecting the k-th neuron to the input signals x_j. These connections can be thought of as synapses. While we do not fully understand the exact mechanisms in biological systems, we use mathematical models to approximate and describe these processes as closely as possible.

Let's formalize this with basic equations. The input to the neuron is denoted as:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

Here, we are summing over all sensory inputs j from 1 to m, scaled by their corresponding weights w_{kj}. The signal v_k is then:

$$v_k = u_k + b_k$$

Finally, the output y_k is given by:

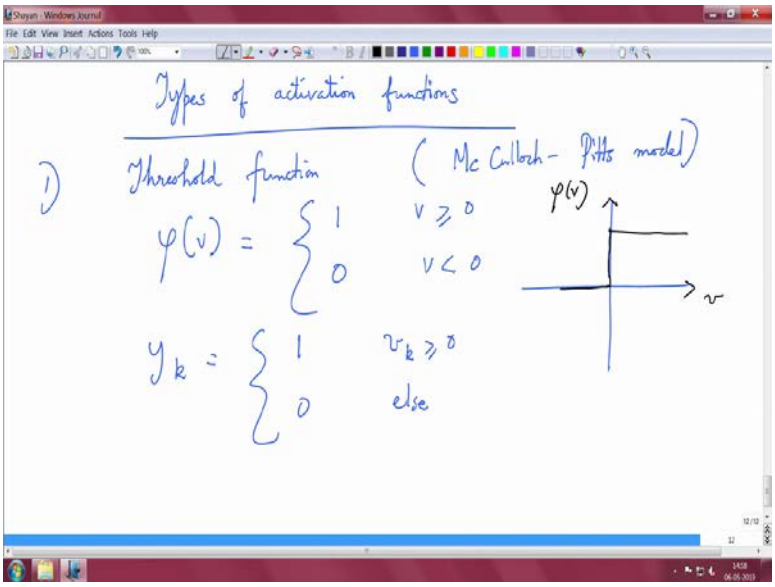
$$y_k = \phi(u_k + b_k) = \phi(v_k)$$

where \phi represents the activation function.

To visualize this, imagine plotting u_k and v_k. The relationship is linear, so the plot of u_k against v_k is a straight line. If b_k is positive, you have a positive y-intercept. If b_k is zero, the line passes through the origin. If b_k is negative, the y-intercept is negative. This represents an affine transformation.

We have established a fundamental structure for the neuron, which will remain consistent throughout our discussion. The primary variation lies in the activation function and the network configuration, but the basic processing element retains this form. The bias term can be interpreted as an input of essentially '1', scaled by b_k, and then fed into the linear combiner. This interpretation provides a clearer understanding of the role of the bias in the neuron's operation.

(Refer Slide Time: 08:43)



Let's delve into the types of activation functions, starting with the simplest one: the threshold function, also known as the McCulloch-Pitts model. This is characterized by a simple Heaviside function $\phi(v)$. Here, v is our local induced field, and $\phi(v)$ is defined as:

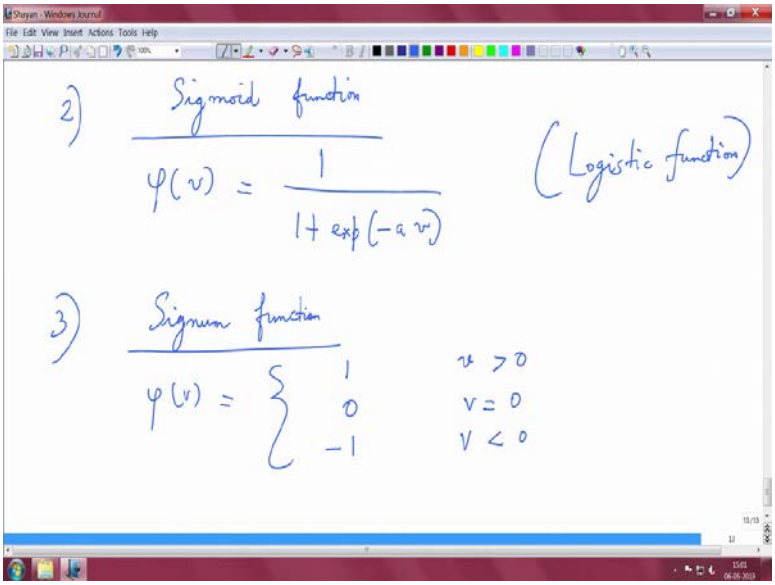
$$\phi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Thus, y_k, which represents the output, is essentially $\phi(v)$. Therefore, y_k is 1 if the local induced field v is zero or positive, and 0 otherwise. A sketch of this would simply illustrate a step function at v = 0.

This is a very basic activation function, often visualized as a step function. We can relate this to how we form binary decisions. For example, if it's raining heavily, you are certain you won't step out of the house; if it's not raining, you decide to go out. These are clear-cut 1 or 0 decisions, similar to the Heaviside function.

In the brain, there is ample evidence of such binary decision-making processes. The clarity and decisiveness of these decisions are the motivations for using Heaviside functions in neural networks.

(Refer Slide Time: 11:06)



So, this is the hard limiter. However, one may also use a function that is not a hard limiter, such as the sigmoid function. The sigmoid function is defined as:

$$\phi(v) = \frac{1}{1 + e^{-av}}$$

This is also called the logistic function. To understand how this function behaves, you can plot it. The exponential term e^{-av} represents an exponentially decaying function when a is positive. By adding one and taking the inverse, you can sketch the sigmoid function. I encourage you to plot this function as an exercise.

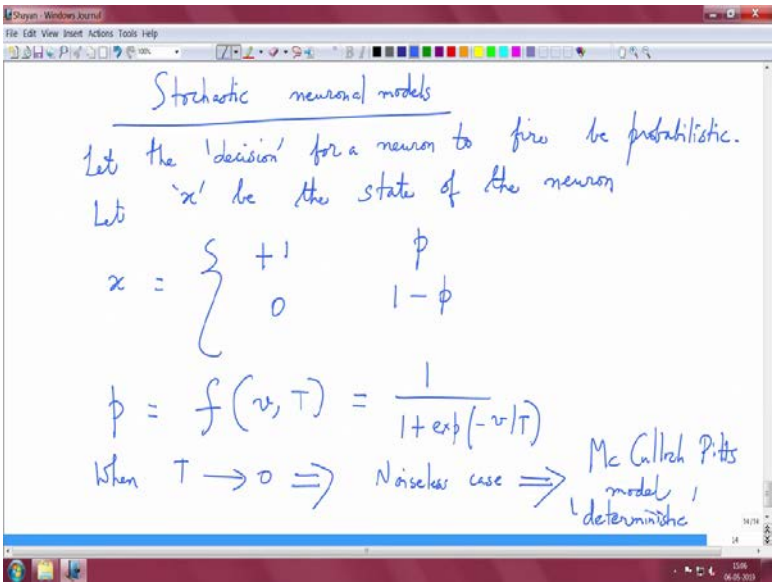
The sigmoid function is preferred over the threshold function for a few reasons. The threshold function is not differentiable at the origin and is discontinuous, whereas the sigmoid function is smooth and differentiable at all points, including the origin. When deriving learning rules, you often need to take the derivative of the activation function. The differentiable nature of the sigmoid function makes it particularly suitable for such purposes.

To delve deeper into activation functions, let's consider the behavior of the sigmoid function as v varies. When v is very large and a is fixed, e^{-av} approaches zero, causing the sigmoid function to saturate at 1. Conversely, when v is strongly negative, e^{-av} becomes a large positive value, and the inverse of a large quantity approaches zero. This behavior illustrates the smooth transition of the sigmoid function between its asymptotic limits.

In addition to the sigmoid function, other activation functions can be used. For instance, consider the sign function (signum function):

$$\text{sign}(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases}$$

(Refer Slide Time: 14:13)



The sign function returns 1 for positive v, 0 when v is zero, and -1 for negative v. While it provides a simple activation mechanism, similar to the Heaviside function, it remains a hard limiter, offering discrete output levels (+1, 0, -1) rather than a smooth, continuous output like

the sigmoid function. This discontinuous nature can be a drawback when differentiability is required for learning algorithms, making the smooth sigmoid function a more favorable choice in many scenarios.

We also have stochastic neuronal models. When describing these models, we will link them to the deterministic models we previously discussed. The motivation for stochastic neuronal models arises from the stochastic resonance effect observed in neurobiology.

So, what is this stochastic resonance effect? Essentially, even in the presence of noisy neurons, intelligent information processing can still occur because the noise can be effectively canceled out. This means that faulty gates or components generating inherent noise can self-cancel, thus providing noise immunity and enhancing error correction capabilities beyond what is possible without such noise.

This stochastic resonance effect provides the basis for engineering neuromorphic systems using stochastic neuronal models. In these models, the firing of neurons is stochastic, leading to interesting non-linear dynamics. These dynamics can potentially offer significant noise immunity. With this motivation in mind, we will now derive the stochastic neuronal model.

Let's delve into the description of a stochastic neuronal model. In this model, the decision for a neuron to fire is probabilistic. Let x represent the state of the neuron, where the state is +1 with a probability p and 0 with a probability 1 - p. This is analogous to a Bernoulli process.

In this context, the state of the neuron +1 occurs with probability p, and 0 otherwise. The probability p is a function of the local induced field v and a parameter T, which controls the noise level and influences the uncertainty in firing. Typically, this probability is given by:

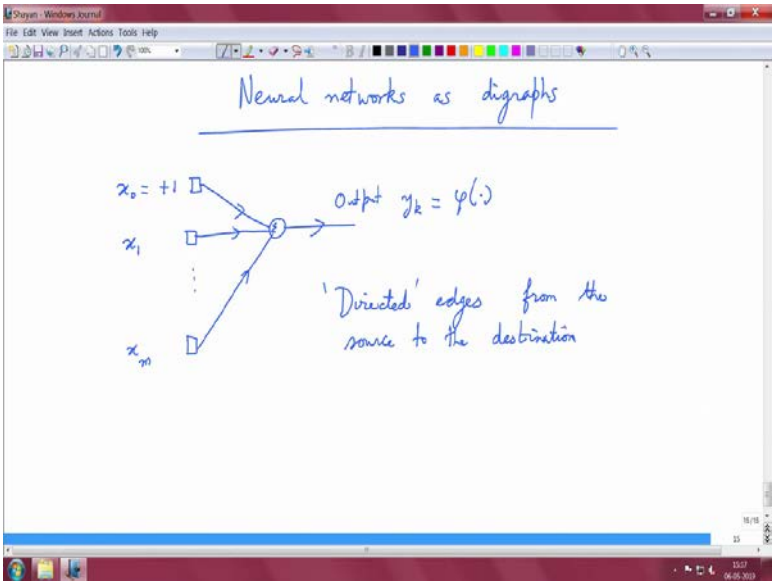
$$p = \frac{1}{1 + \exp\left(-\frac{v}{T}\right)}$$

When T approaches 0, implying a noiseless scenario, this model converges to the McCulloch-Pitts model. As T goes to 0, $\exp\left(-\frac{v}{T}\right)$ approaches 0, making p approach 1, indicating a deterministic state where the neuron will always fire if v is positive.

This parameter T can be thought of as an annealing parameter, controlling the thermal fluctuations or synaptic noise, thereby adjusting the firing threshold.

As previously discussed, the role of stochastic neuronal models is crucial in engineering artificial systems that are noise-immune. This concept stems from the stochastic resonance effect observed in biological systems, which are naturally immune to noise. Incorporating stochastic neuronal models into engineered systems and neuromorphic designs can help achieve this noise immunity, leveraging the benefits of the stochastic resonance effect.

(Refer Slide Time: 19:28)



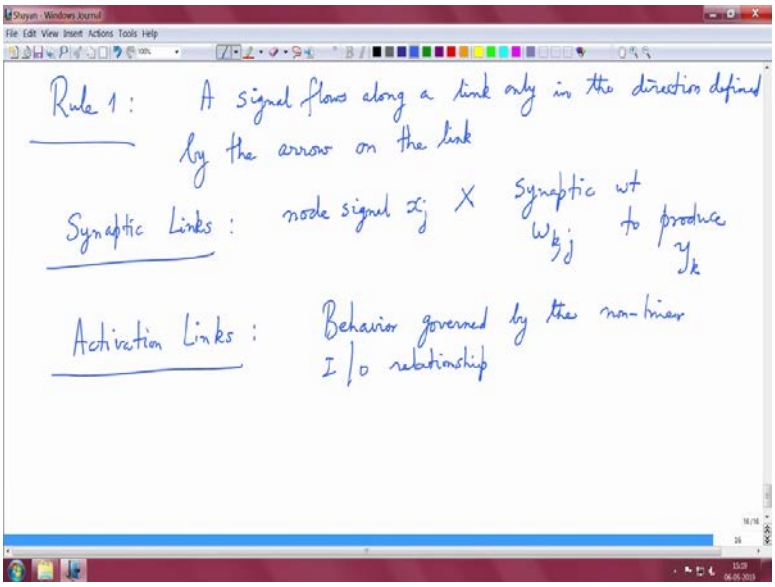
Let's begin with the interpretation of neural networks as digraphs, which are directed graphs. Let's sketch this concept. Assume $x_0 = 1$ to account for the bias component, and then we have x_1, x_2, ..., x_m, all of which connect through edges to the linear combiner, producing an output y_k through the activation function.

The key point to understand here is that there are directed edges from the source nodes to the destination node, which is why we conceptualize this as a digraph. There is a signal flow occurring through the system, and when considering signal flow, we need to describe it using signal flow graphs and the associated rules. Let's begin with the interpretation of neural networks as digraphs, which are directed graphs. Let's sketch this concept. Assume $x_0 = 1$ to account for the bias component, and then we have x_1, x_2, ..., x_m, all of which connect through edges to the linear combiner, producing an output y_k through the activation function.

The key point to understand here is that there are directed edges from the source nodes to the destination node, which is why we conceptualize this as a digraph. There is a signal flow occurring through the system, and when considering signal flow, we need to describe it using

signal flow graphs and the associated rules.

(Refer Slide Time: 21:05)



Let's outline these rules coherently and explore their connections within the context of neural networks.

Rule 1: A signal flows along a link only in the direction defined by the arrow on the link. We have two types of links: synaptic links and activation links. Let's describe them carefully.

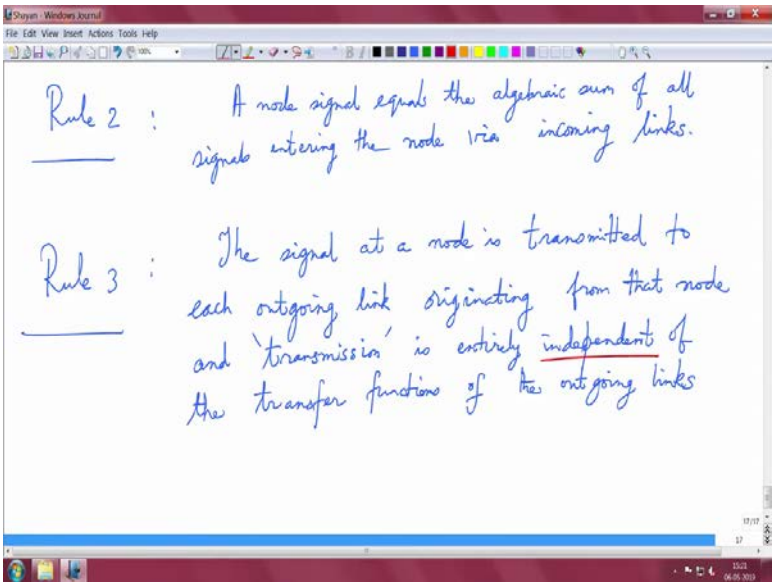
- **Synaptic Links:** Consider a node signal x_j that is scaled by a synaptic weight w_{kj} (where kkk is the neuron index and j is the input index) to produce y_k. Here, w_{kj} represents the synaptic weight connecting signal x_j to neuron k.

- **Activation Links:** The behavior of these links is governed by the activation function, which establishes the non-linear input-output relationship.

Rule 2: A node signal equals the algebraic sum of all signals entering the node via incoming links. This rule is intuitive; it means we take the stimuli, scale them by the corresponding synaptic weights, linearly combine them, and then apply an activation function.

Rule 3: The signal at a node is transmitted to each outgoing link originating from that node, and this transmission is entirely independent of the transfer functions of the outgoing links. This transmission independence is crucial because it allows the signal to propagate through the network regardless of the specific transfer functions applied to the outgoing links.

(Refer Slide Time: 23:15)



In this context, you can think about signal flow graphs similar to those used in control systems, as pioneered by Mason. These concepts have been successfully applied in signal processing, coding, and various other applications.

For example, when considering error correction coding, the concept of a weight enumeration function can be approached using signal flow graphs. Similarly, in control systems, if you want to simplify transfer functions, signal flow graphs are employed. This same concept can be applied to neural networks, albeit with nonlinearities and different types of mappings rather than traditional transfer functions.

Representing neural networks through digraphs offers a clear visualization of input sources and signal flow. This method allows us to distinguish between different configurations, such as feedforward structures, feedback loops, or the absence of feedback. Thus, using directed graphs to represent neural networks provides a valuable and insightful framework.

With this foundational understanding, we are now prepared to delve into the concept of feedback mechanisms and explore their roles within neural networks. Let's stop here for now.