Automatic Speech Recognition (CS753)

Lecture 9: Brief Introduction to Neural Networks

Instructor: Preethi Jyothi

Feb 2, 2017

Final Project Landscape

Tabla bol transcription

Automatic Tongue Twister Generator

Sanskrit Synthesis and Recognition

Voice-based music player

InfoGAN for music

Music Genre Classification

Automatic authorised ASR

Audio Synthes Using LSTMs

Singer Identification

Speech synthesis & ASR for Indic languages

Keyword spotting for continuous speech

Speaker Verification

Transcribing TED Talks

Swapping instruments in recordings

Ad detection in live radio streams

Emotion Recognition fro speech

Programming with speech-based commands

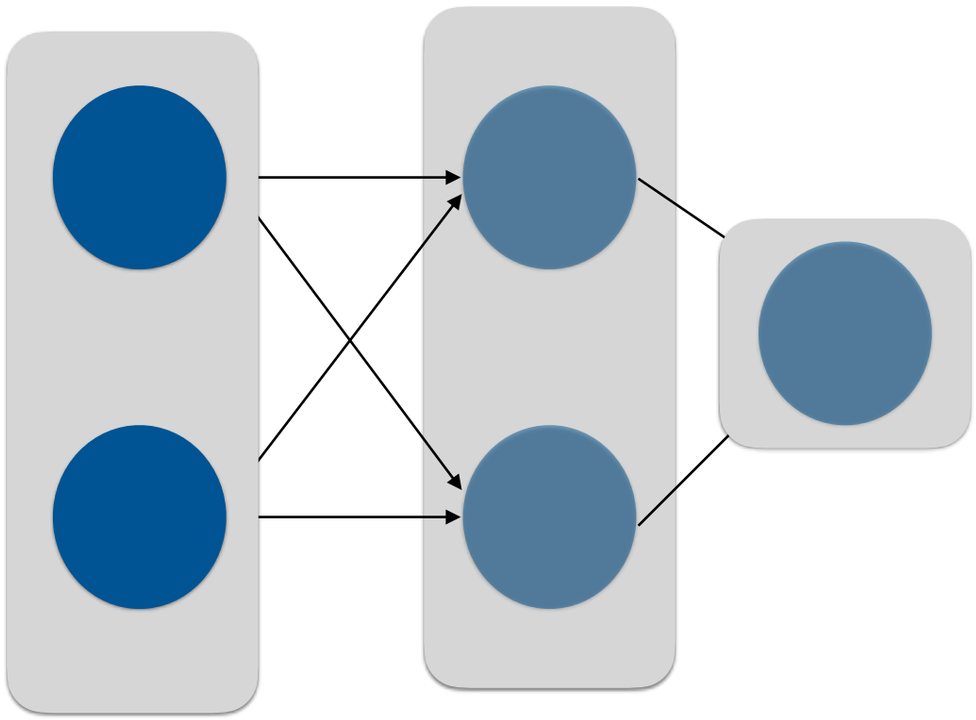
Speaker Adaptatior

End-to-end Audio-Visual Speech Recognition

Nationality detection from speech accents

Bird call Recognition

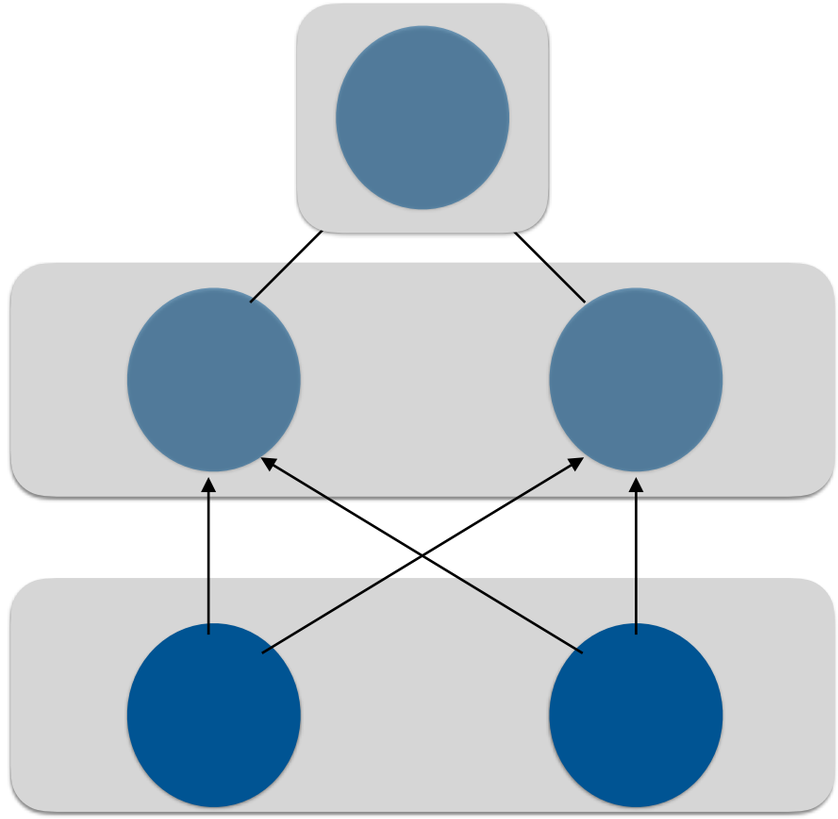
Feed-forward Neural Network



Output Layer

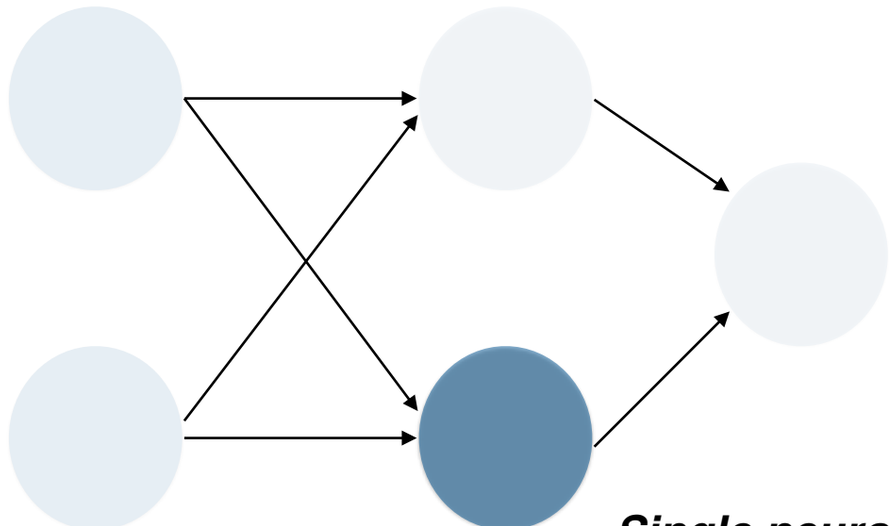
Input Layer

Hidden Layer

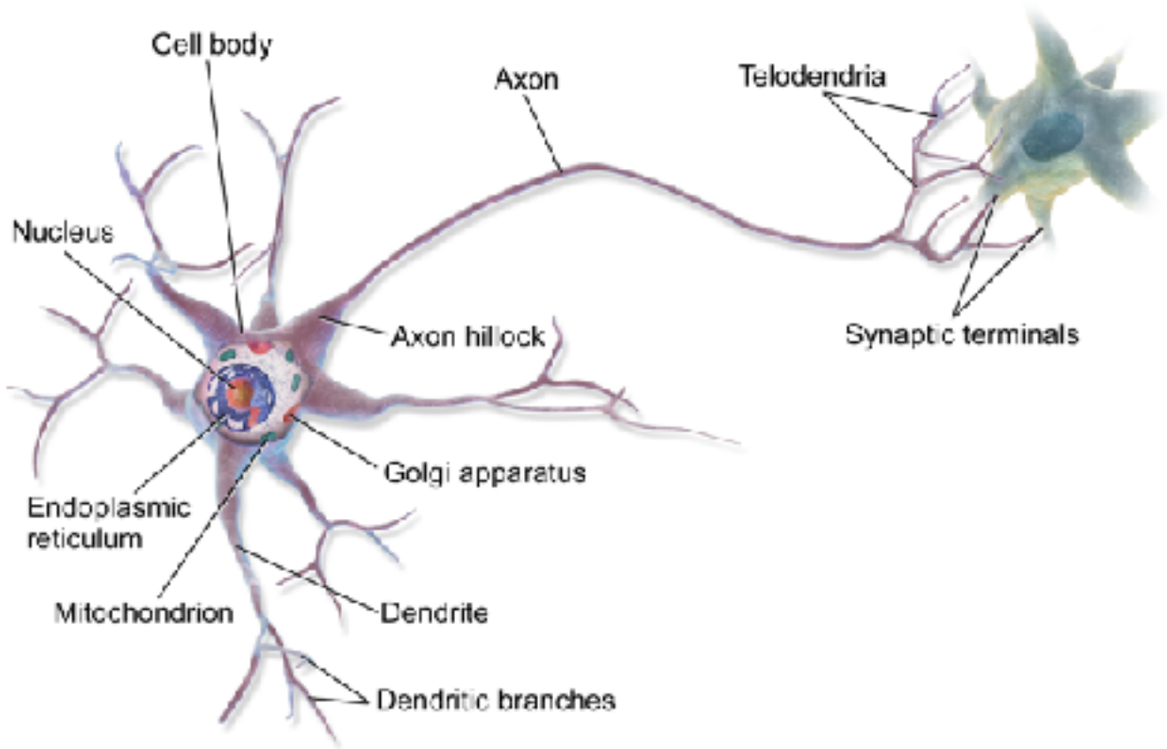


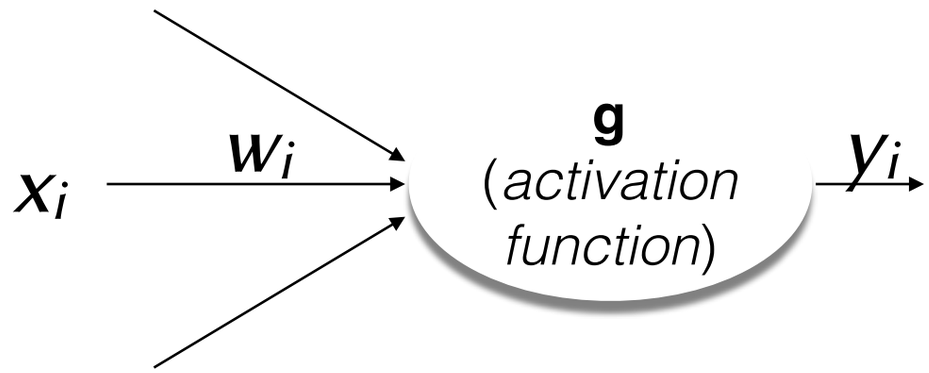
Feed-forward Neural Network

Brain Metaphor



Single neuron



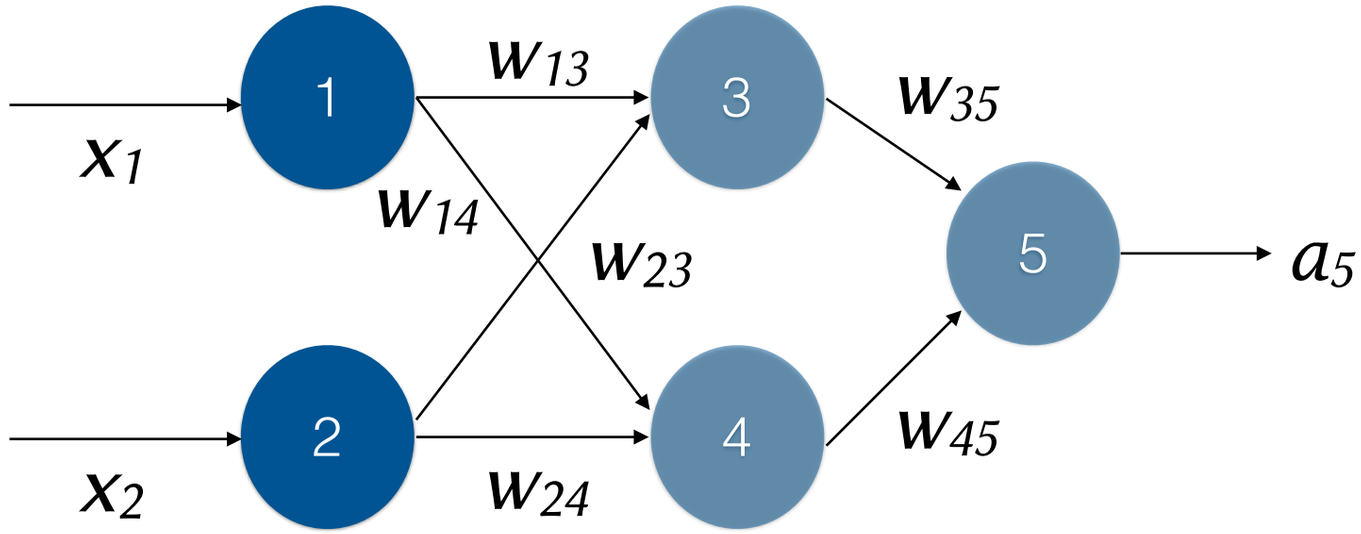


$$y_i = g(\Sigma_i\, w_i \cdot x_i)$$

Image from: https://upload.wikimedia.org/wikipedia/commons/1/10/Blausen_0657_MultipolarNeuron.png

Feed-forward Neural Network

Parameterized Model



$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$
$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

Parameters of the network: all w_{ij} (and biases not shown here)

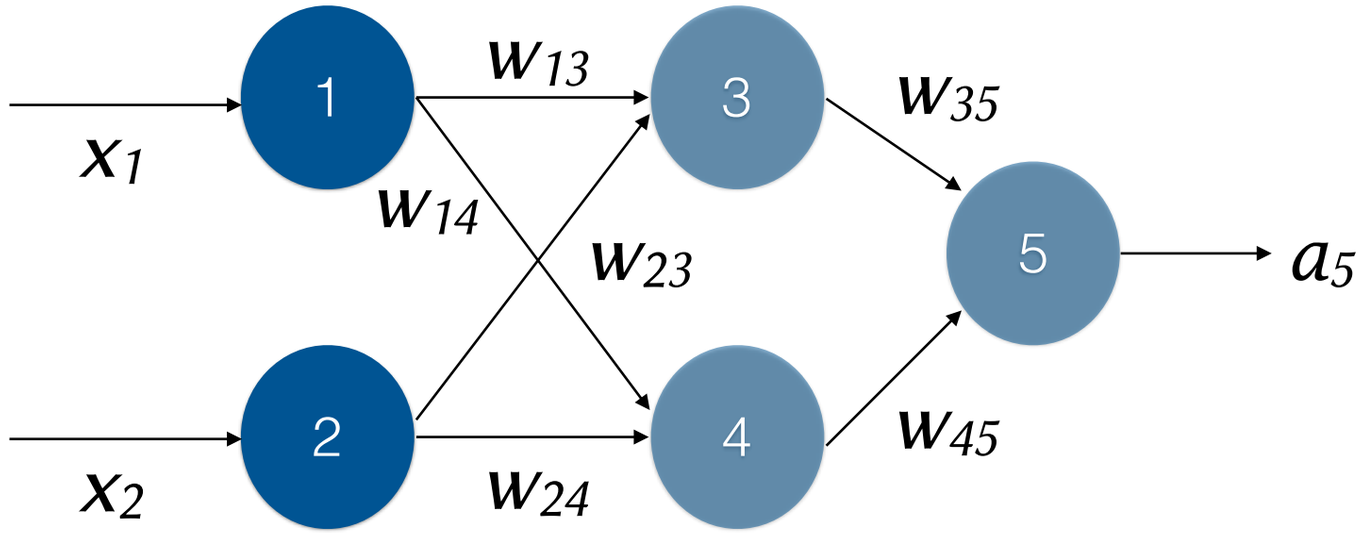
If **x** is a 2-dimensional vector and the layer above it is a 2-dimensional vector **h**, a fully-connected layer is associated with:

$$\mathbf{h} = \mathbf{xW} + \mathbf{b}$$

where w_{ij} in **W** is the weight of the connection between i^{th} neuron in the input row and j^{th} neuron in the first hidden layer and **b** is the bias vector

Feed-forward Neural Network

Parameterized Model



$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$

$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

The simplest neural network is the perceptron:

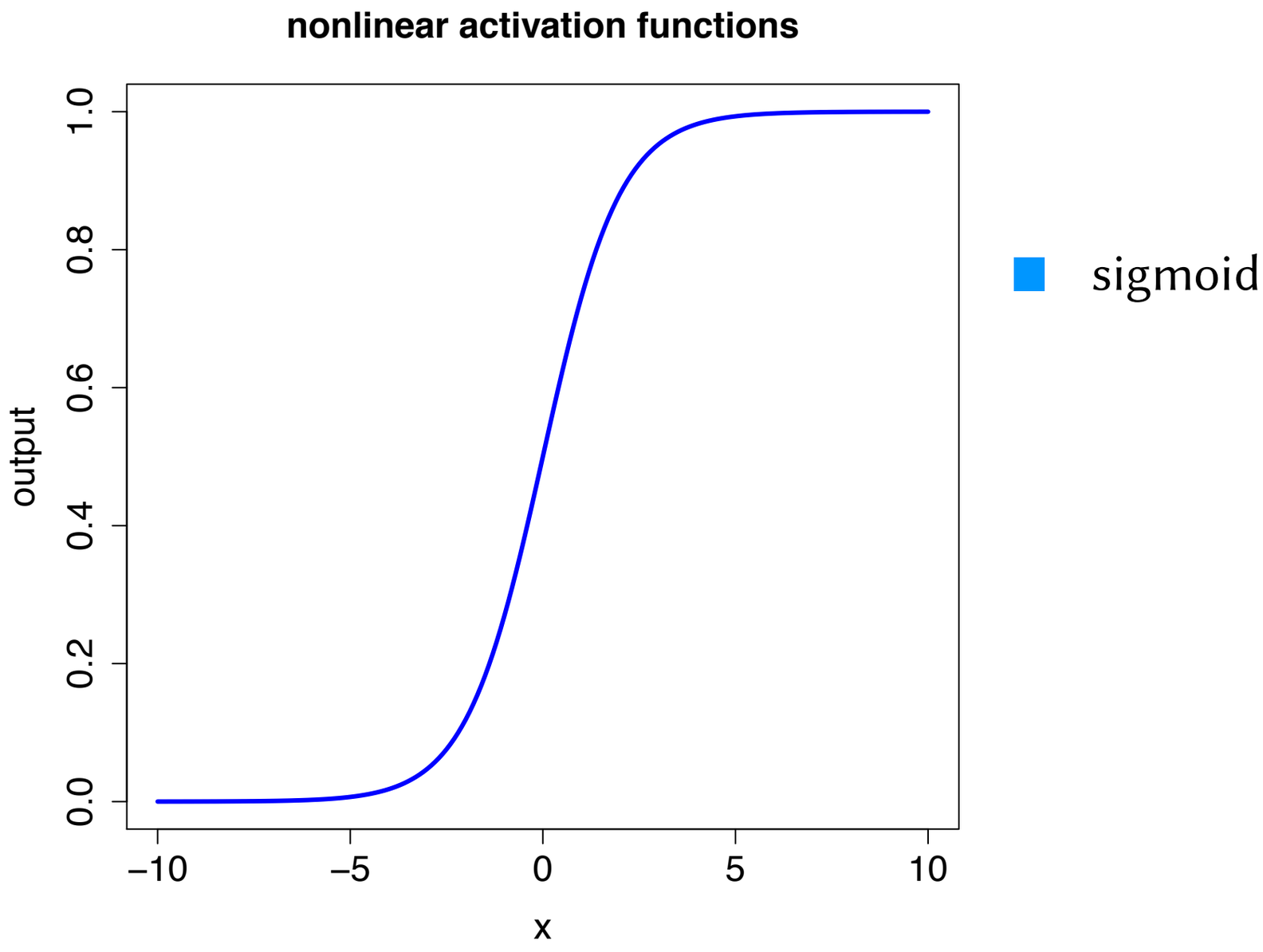
$$\text{Perceptron}(x) = \mathbf{xW} + \mathbf{b}$$

A 1-layer feedforward neural network has the form:

$$\text{MLP}(x) = g(\mathbf{xW}_1 + \mathbf{b}_1)\,\mathbf{W}_2 + \mathbf{b}_2$$

Common Activation Functions (g)

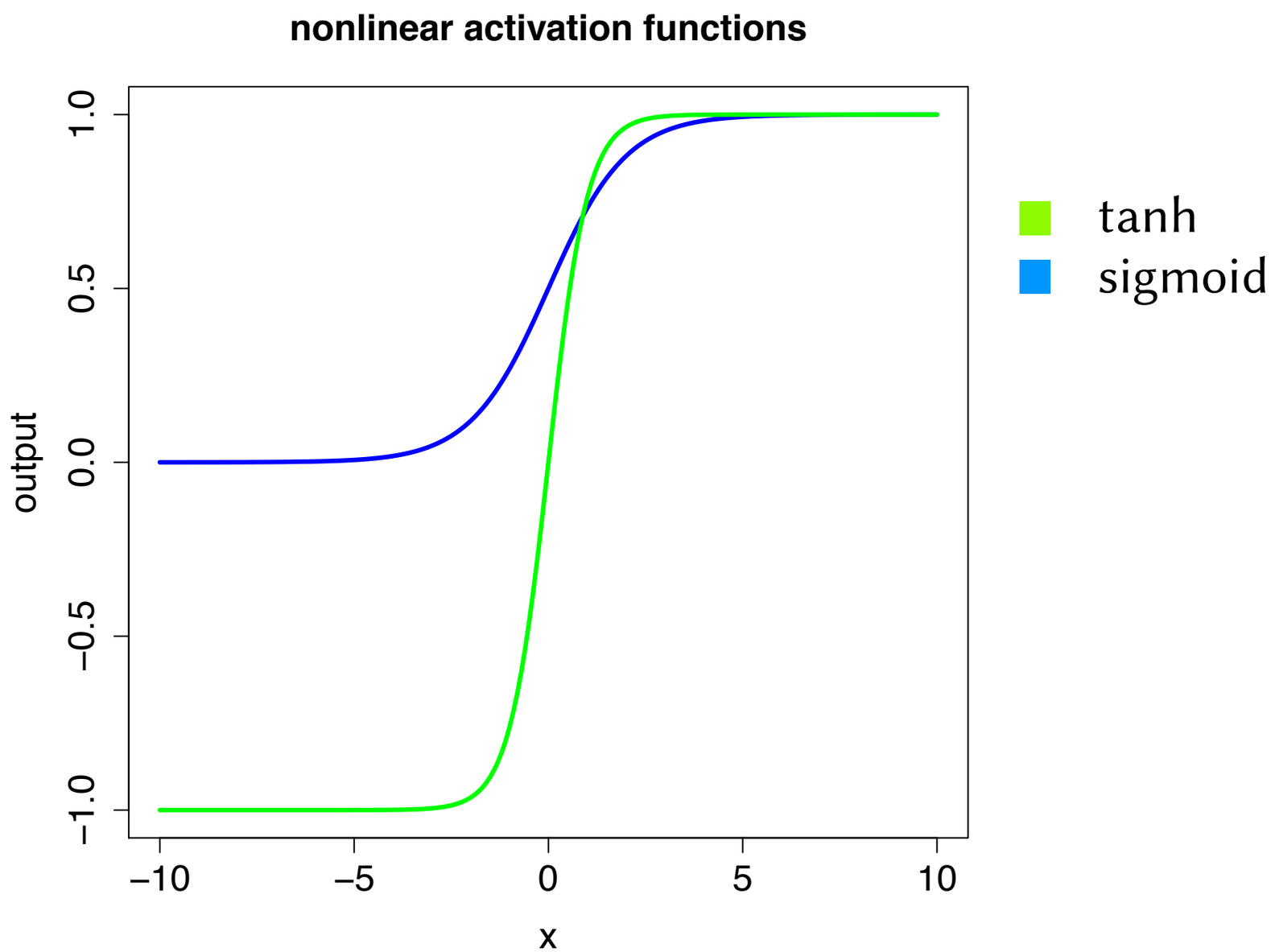
Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$



Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

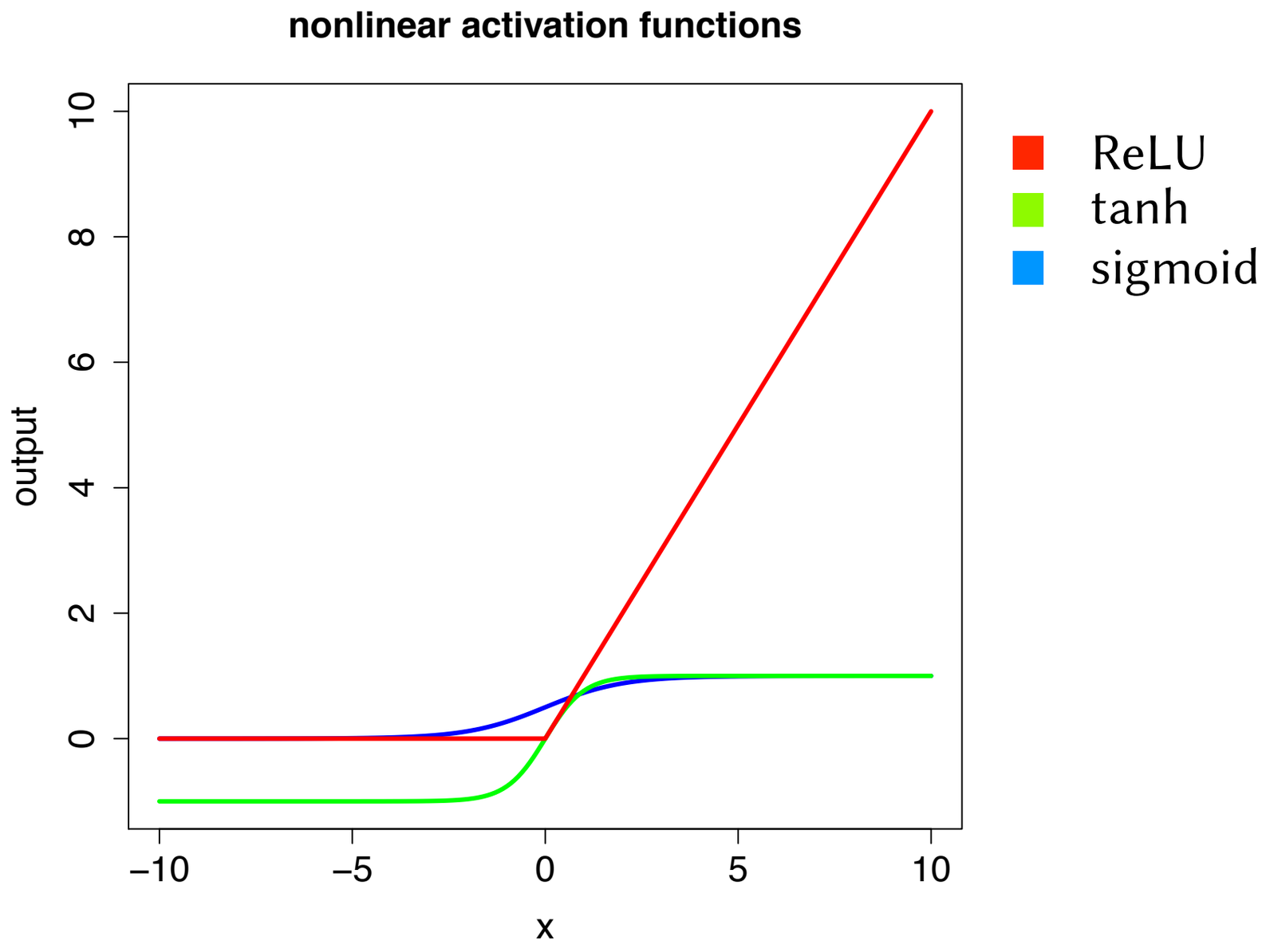


Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

Rectified Linear Unit (ReLU): $\text{RELU}(x) = \max(0, x)$



Optimization Problem

- To train a neural network, define a loss function $L(y,\tilde{y})$: a function of the true output y and the predicted output $\tilde{y}$

- $L(y,\tilde{y})$ assigns a non-negative numerical score to the neural network's output, $\tilde{y}$

- The parameters of the network are set to minimise L over the training examples (i.e. a sum of losses over different training samples)

- L is typically minimised using a *gradient-based method*

Stochastic Gradient Descent (SGD)

SGD Algorithm

Inputs:

Function NN(x; θ), Training examples, $x_1 \ldots x_n$ and outputs, $y_1 \ldots y_n$ and Loss function L.

do until stopping criterion

Pick a training example x_i, y_i

Compute the loss $L(NN(x_i; \theta), y_i)$

Compute gradient of L, ∇L with respect to θ

$\theta \leftarrow \theta - \eta \nabla L$

done

Return: θ

Training a Neural Network

Define the **Loss function** to be minimised as a node L

Goal: Learn weights for the neural network which minimise L

Gradient Descent: Find $\partial L/\partial w$ for every weight w, and update it as $w \leftarrow w - \eta\, \partial L/\partial w$

How do we efficiently compute $\partial L/\partial w$ for all w?

Will compute $\partial L/\partial u$ for every node u in the network!

$\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$ where u is the node which uses w

Training a Neural Network

New goal: compute $\partial L/\partial u$ for every node u in the network

Simple algorithm: Backpropagation

Key fact: Chain rule of differentiation

If L can be written as a function of variables $v_1,\ldots, v_n$, which in turn depend (partially) on another variable u, then

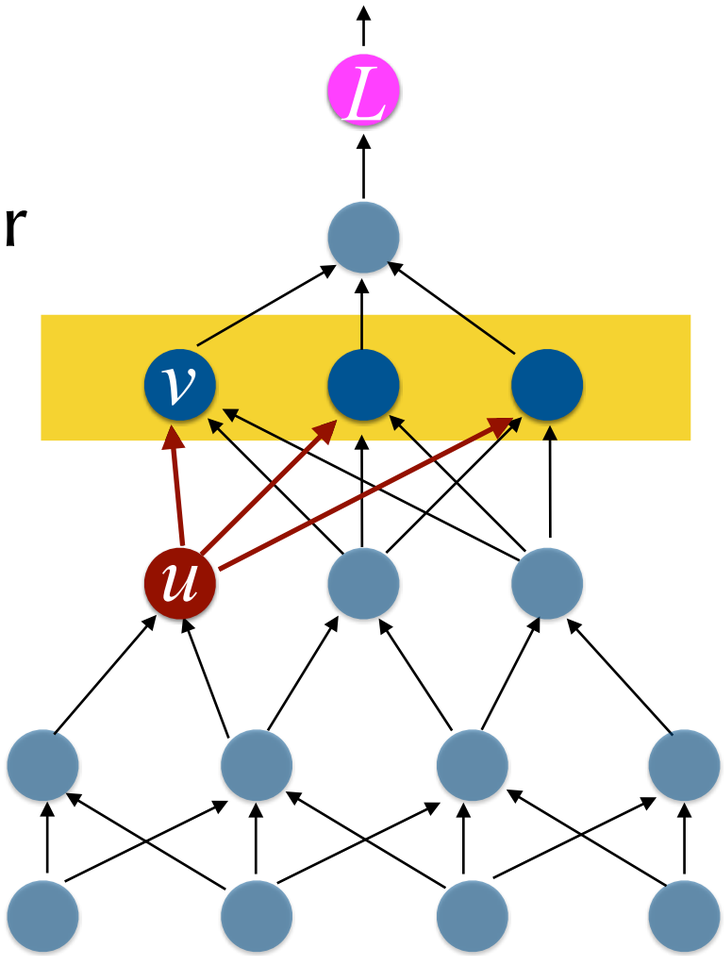
$$\partial L/\partial u = \Sigma_i \, \partial L/\partial v_i \cdot \partial v_i/\partial u$$

Backpropagation

If L can be written as a function of variables $v_1,\ldots, v_n$, which in turn depend (partially) on another variable u, then

$$\partial L/\partial u = \Sigma_i \, \partial L/\partial v_i \cdot \partial v_i/\partial u$$

Consider $v_1,\ldots, v_n$ as the layer above u, $\Gamma(u)$



Then, the chain rule gives

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)} \; \partial L/\partial v \cdot \partial v/\partial u$$

Backpropagation

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)} \; \partial L/\partial v \cdot \partial v/\partial u$$

Backpropagation

Base case: $\partial L/\partial L = 1$

For each u (top to bottom):

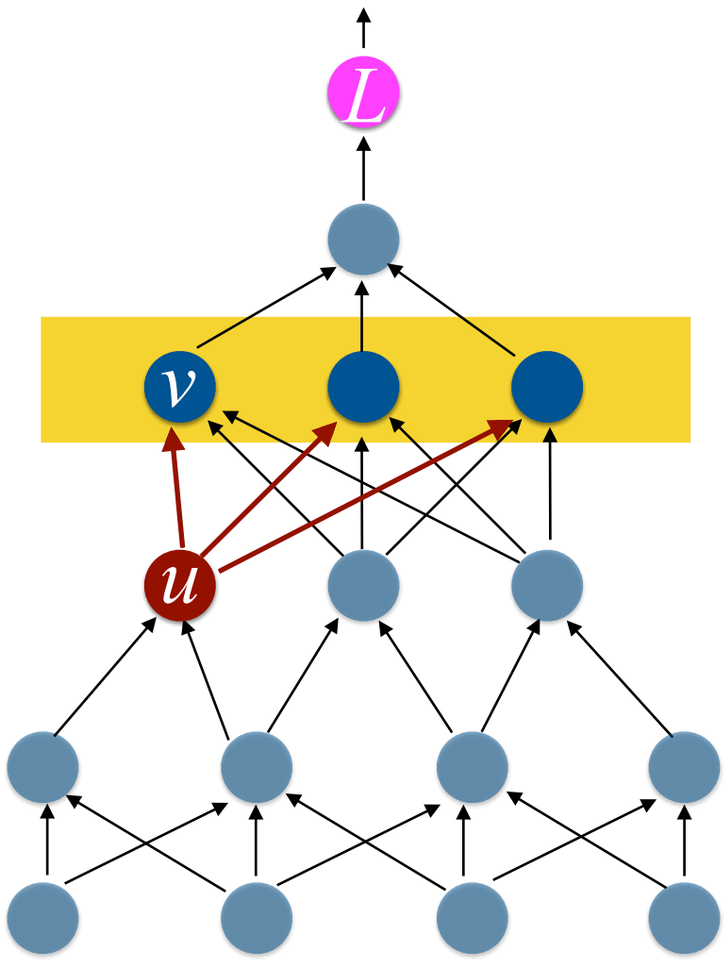
For each $v \in \Gamma(u)$:

Inductively, have computed $\partial L/\partial v$

Directly compute $\partial v/\partial u$

Compute $\partial L/\partial u$

Compute $\partial L/\partial w$ where $\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$



Forward Pass

First compute all values of u given an input, in a forward pass
(The values of each node will be needed during backprop)

Where values computed in the forward pass may be needed

Neural Network Acoustic Models

- Input layer takes a window of acoustic feature vectors
- Output layer corresponds to classes (e.g. monophone labels, triphone states, etc.)

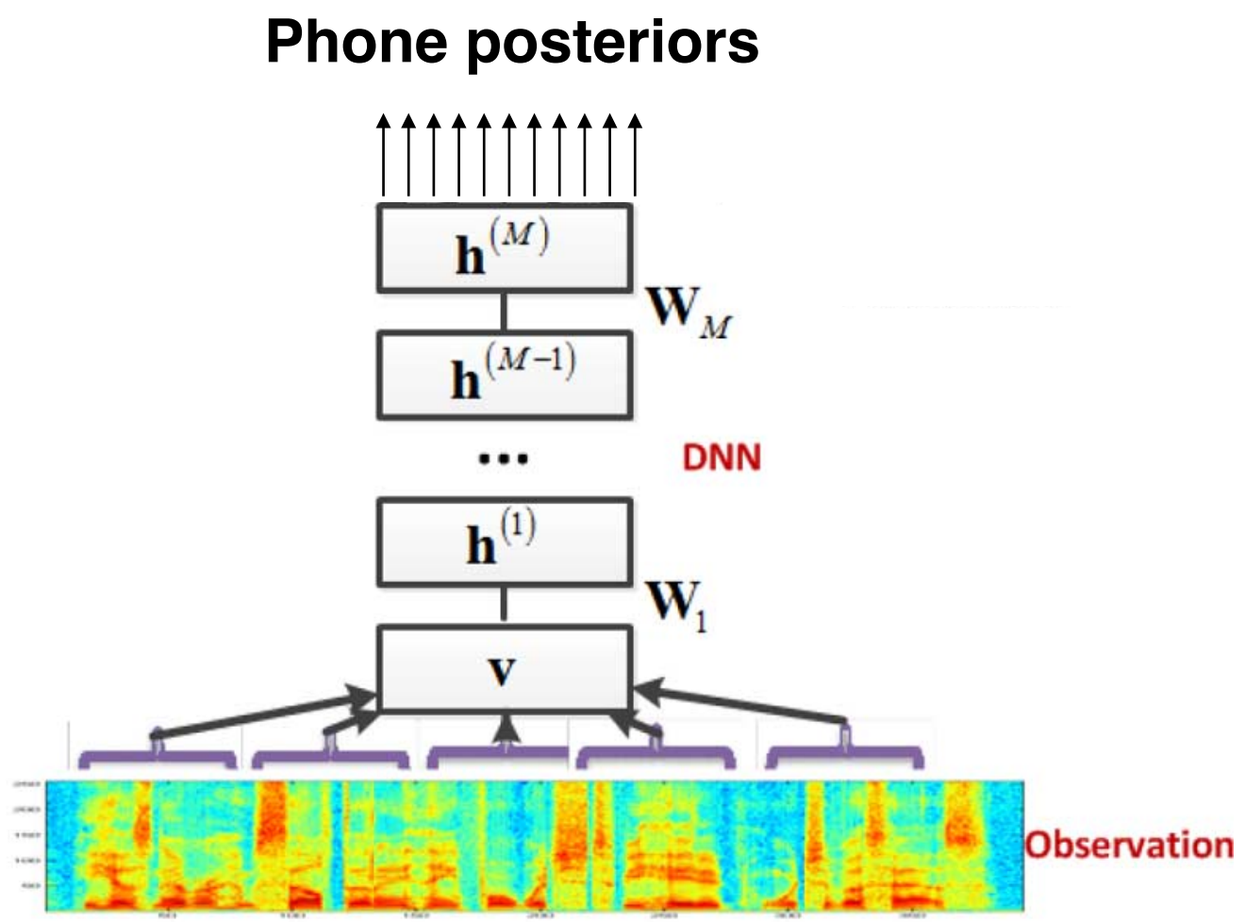


Image adapted from: Dahl et al., "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition", TASL'12

Neural Network Acoustic Models

- Input layer takes a window of acoustic feature vectors
- Hybrid NN/HMM systems: replace GMMs with outputs of NNs

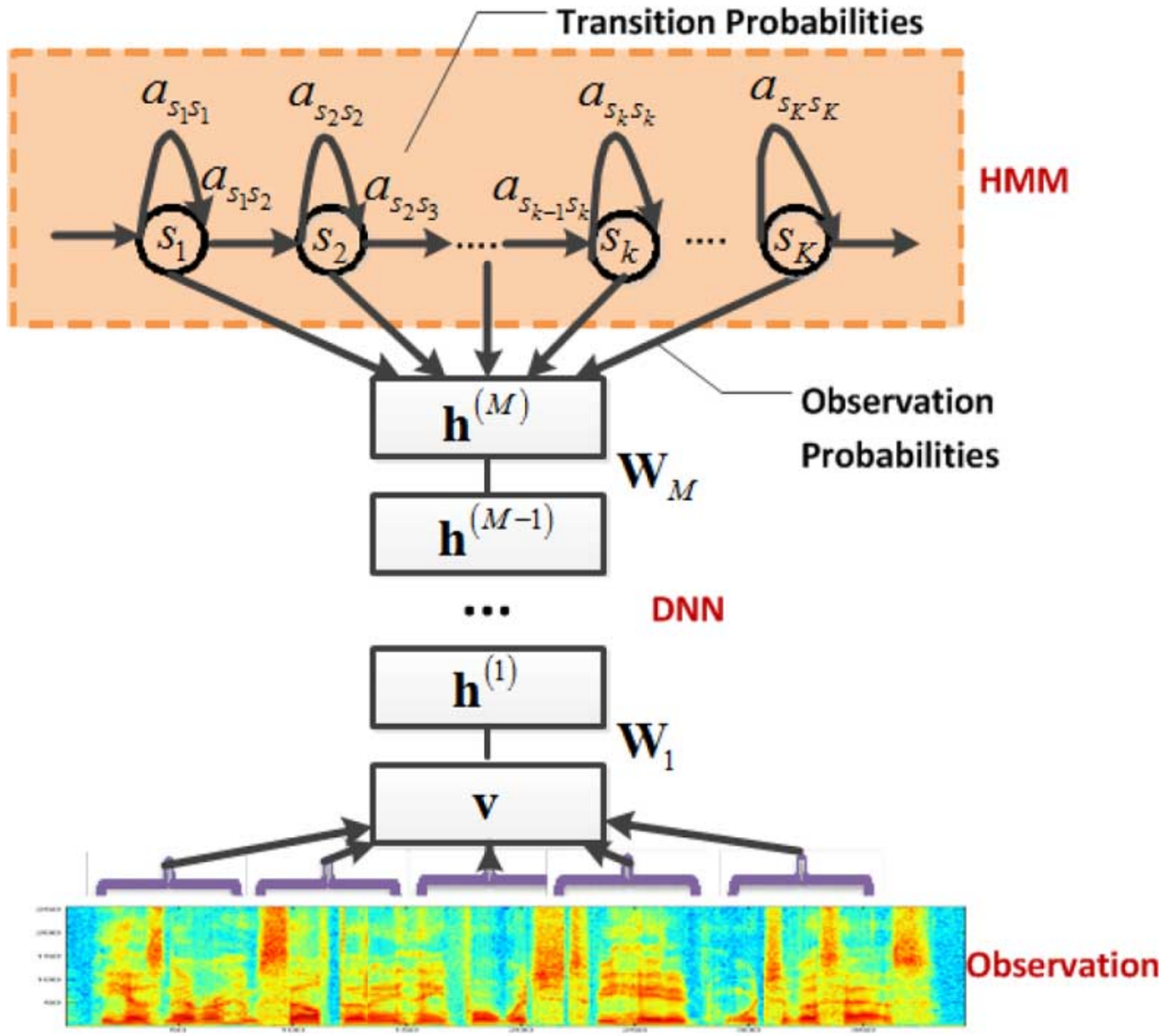


Image from: Dahl et al., "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition", TASL'12