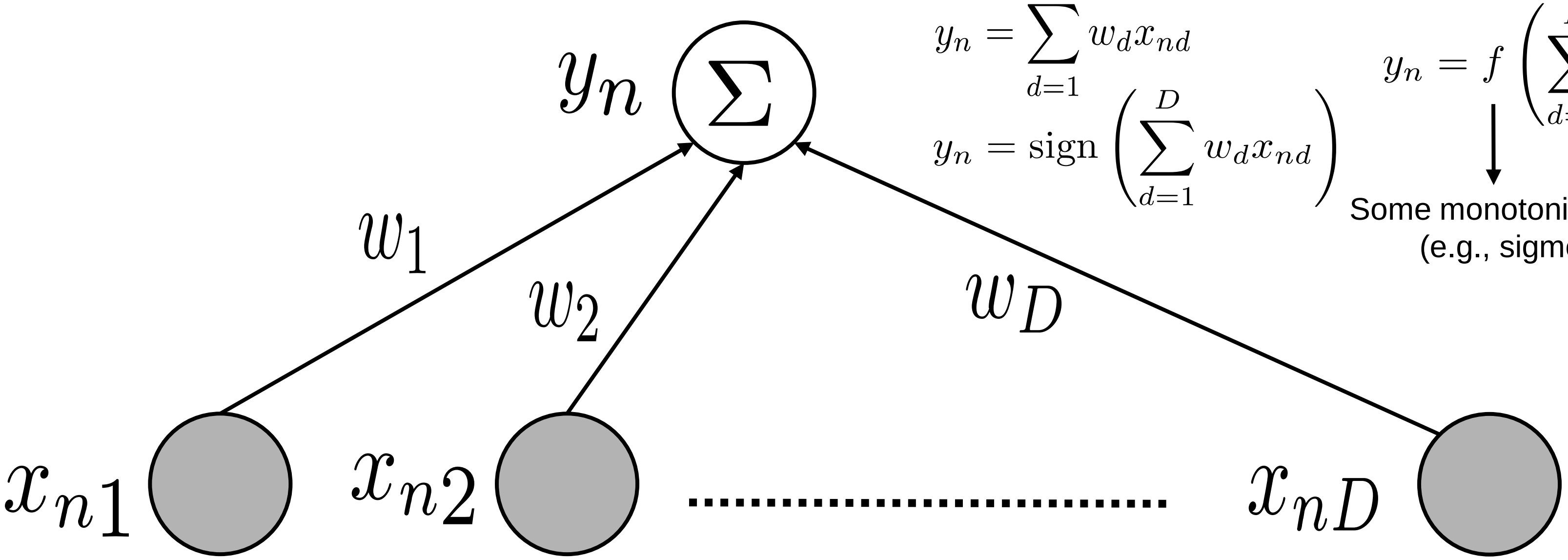
“Deep” Learning

CS771: Introduction to Machine Learning

Nisheeth

Limitations of Linear Models

- Linear models: Output produced by taking a linear combination of input features



$$y_n = \sum_{d=1}^{D} w_d x_{nd}$$

$$y_n = \text{sign}\left(\sum_{d=1}^{D} w_d x_{nd}\right)$$

$$y_n = f\left(\sum_{d=1}^{D} w_d x_{nd}\right)$$

Some monotonic function (e.g., sigmoid)

- This basic architecture is classically also known as the "Perceptron" (not to be confused with the Perceptron "algorithm", which learns a linear classification model)

- This can't however learn nonlinear functions or nonlinear decision boundaries

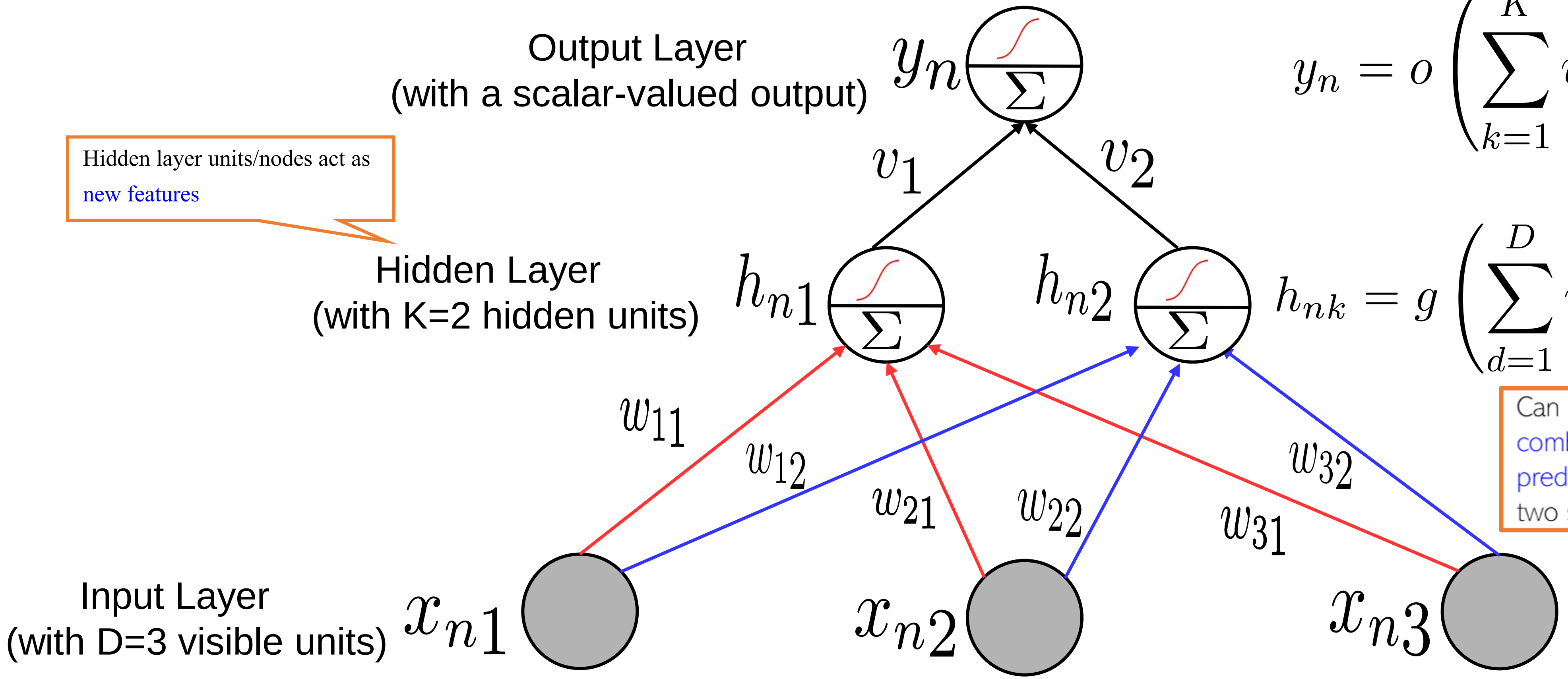
Limitations of Classic Non-Linear Models

- Non-linear models: kNN, kernel methods, generative classification, decision trees etc.
- All have their own disadvantages
- kNN and kernel methods are expensive to generate predictions from
- Kernel based and generative models particularize the decision boundary to a particular class of functions, e.g. quadratic polynomials, gaussian functions etc.
- Decision trees require optimization over many arbitrary hyperparameters to generate good results, and are (somewhat) expensive to generate predictions from
  - Not a deal-breaker, most common competitor for deep learning over large datasets tends to be some decision-tree derivative
- In general, non-linear ML models are complicated beasts

Neural Networks: Multi-layer Perceptron (MLP)

- An MLP consists of an input layer, an output layer, and one or more hidden layers

Hidden layer units/nodes act as new features



$$y_n = o\left(\sum_{k=1}^{K} v_k h_{nk}\right)$$

Learnable weights

$$h_{nk} = g\left(\sum_{d=1}^{D} w_{dk} x_{nd}\right)$$

Can think of this model as a combination of two predictions h_{n1} and h_{n2} of two simpler models

Illustration: Neural Net with One Hidden Layer

- Each input $\boldsymbol{x}_n$ transformed into several pre-activations using linear models

$$a_{nk} = \mathbf{w}_k^\top \mathbf{x}_n = \sum_{d=1}^{D} w_{dk} x_{nd}$$

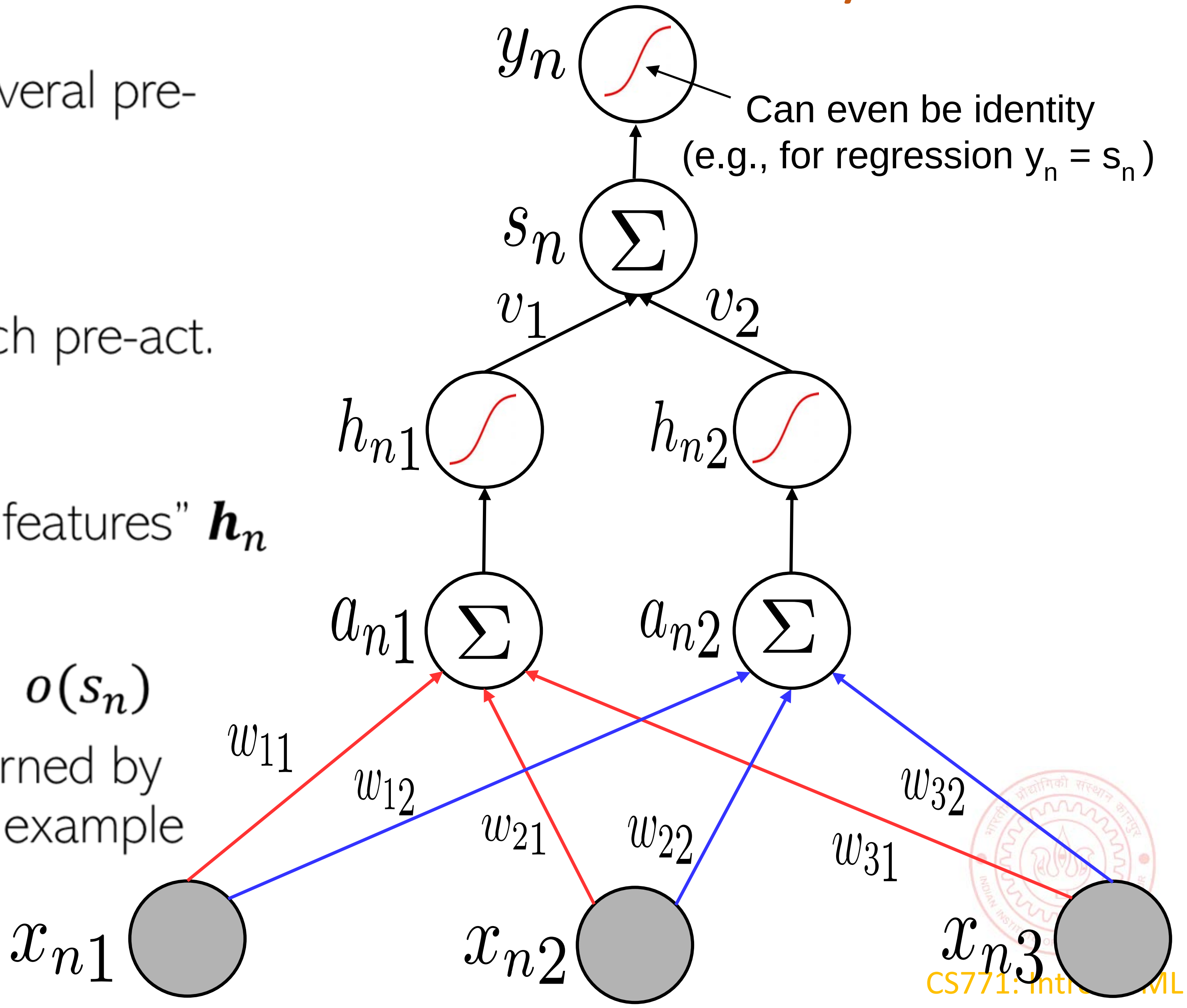
- Nonlinear activation applied on each pre-act.

$$h_{nk} = g(a_{nk})$$

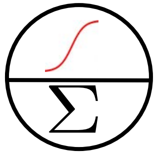
- Linear model learned on the new "features" $\boldsymbol{h}_n$

$$s_n = \mathbf{v}^\top \mathbf{h}_n = \sum_{k=1}^{K} v_k h_{nk}$$

- Finally, output is produced as $\boldsymbol{y} = o(s_n)$
- Unknowns $(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K, \boldsymbol{v})$ learned by minimizing some loss function, for example $\mathcal{L}(\boldsymbol{W}, \boldsymbol{v}) = \sum_{n=1}^{N} \ell(y_n, o(s_n))$ (squared, logistic, softmax, etc)

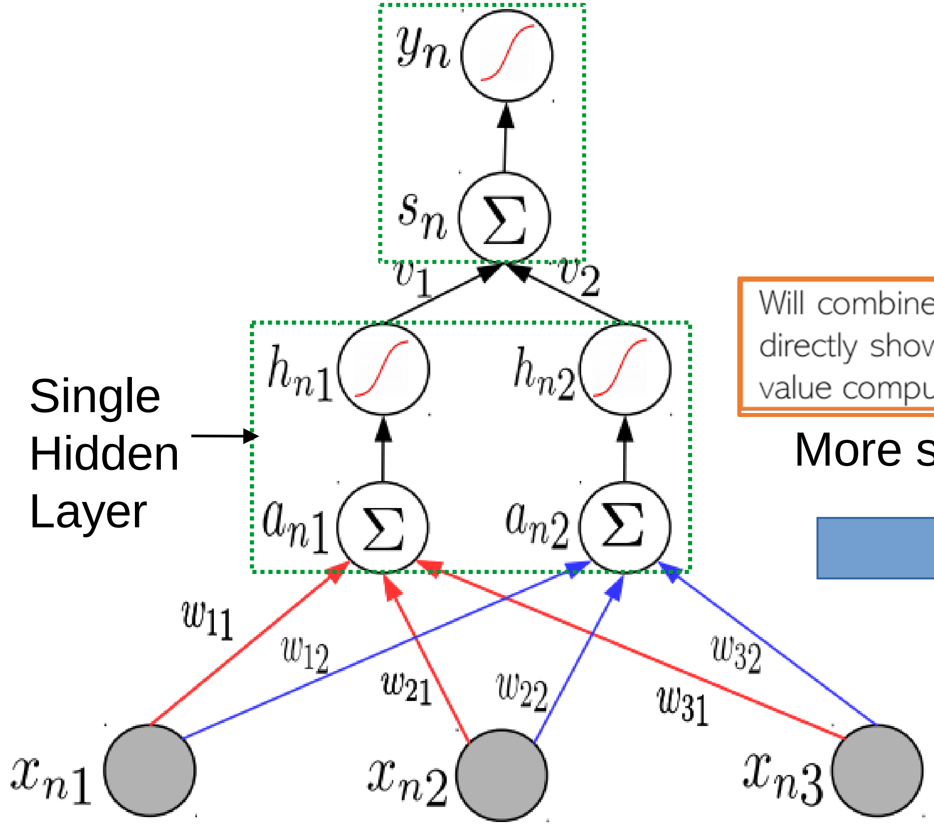


Neural Nets: A Compact Illustration



Will denote a linear combination of inputs followed by a nonlinear operation on the result

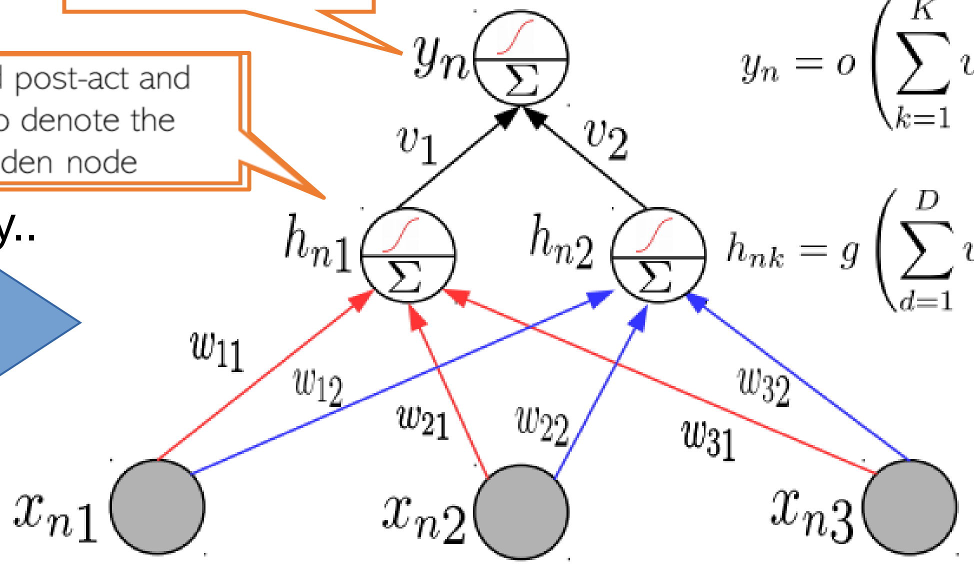
- Note: Hidden layer pre-act a_{nk} and post-act h_{nk} will be shown together for brevity



More succinctly..

Will combine pre-act and post-act and directly show only h_{nk} to denote the value computed by a hidden node

Will directly show the final output

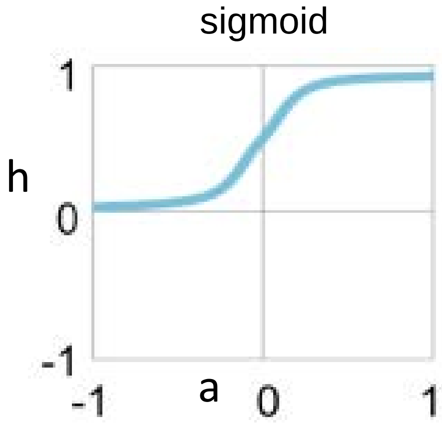


$$y_n = o\left(\sum_{k=1}^{K} v_k h_{nk}\right)$$

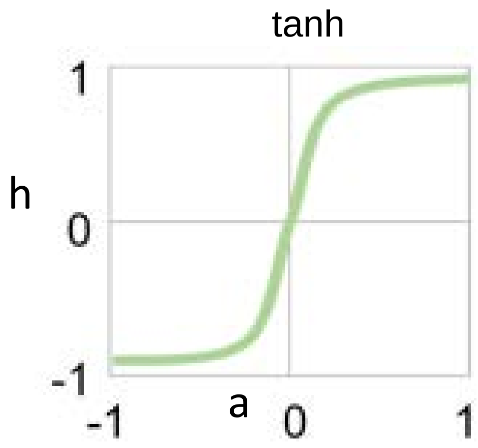
$$h_{nk} = g\left(\sum_{d=1}^{D} w_{dk} x_{nd}\right)$$

- Different layers may use different non-linear activations. Output layer may have none.

Activation Functions: Some Common Choices



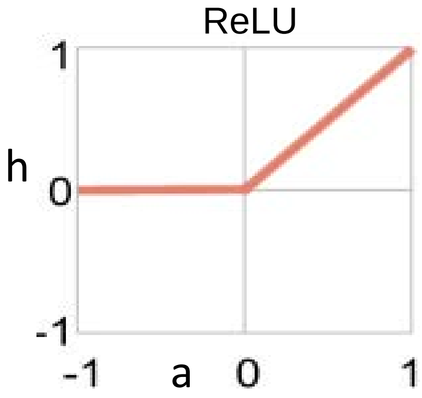
For sigmoid as well as tanh, gradients saturate (become close to zero as the function tends to its extreme values)



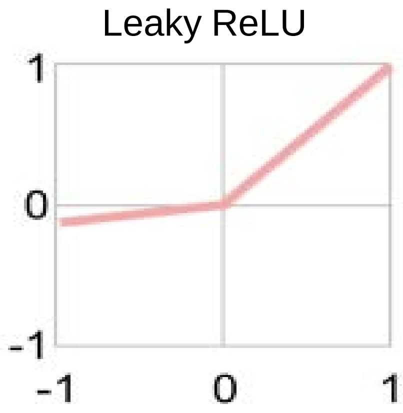
Preferred more than sigmoid. Helps keep the mean of the next layer's inputs close to zero (with sigmoid, it is close to 0.5)

Sigmoid: $h = \sigma(a) = \frac{1}{1+\exp(-a)}$

tanh (tan hyperbolic): $h = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = 2\sigma(2a) - 1$



Helps fix the dead neuron problem of ReLU when a is a negative number



ReLU and Leaky ReLU are among the most popular ones

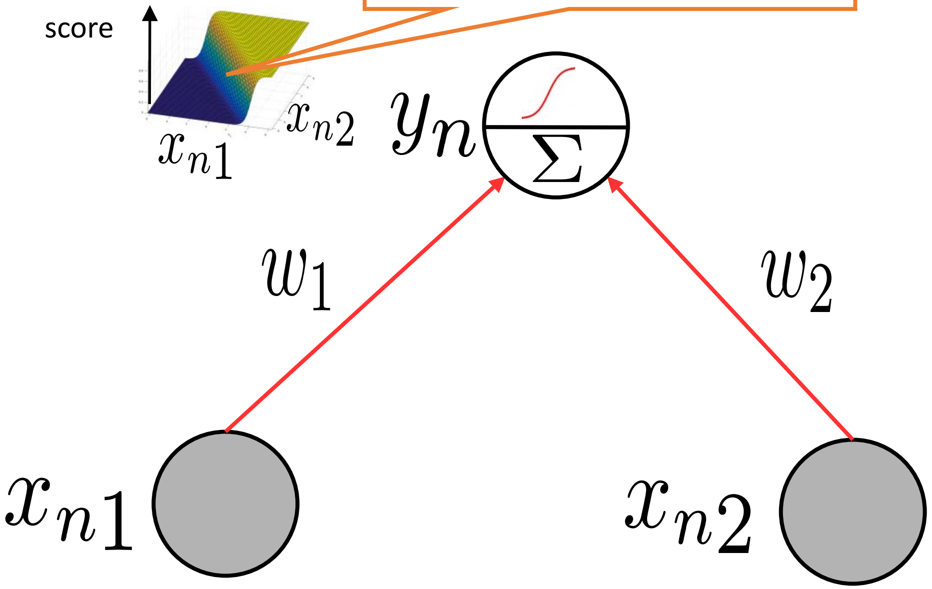
ReLU (Rectified Linear Unit): $h = \max(0, a)$

Leaky ReLU: $h = \max(\beta a, a)$
where β is a small postive number

MLP Can Learn Nonlin. Fn: A Brief Justification

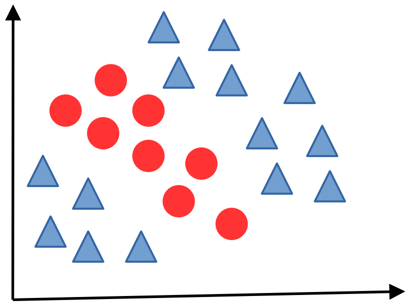
- An MLP can be seen as a composition of multiple linear models combined nonlinearly

Score monotonically increases. One-sided increase (not ideal for learning nonlinear decision boundaries)



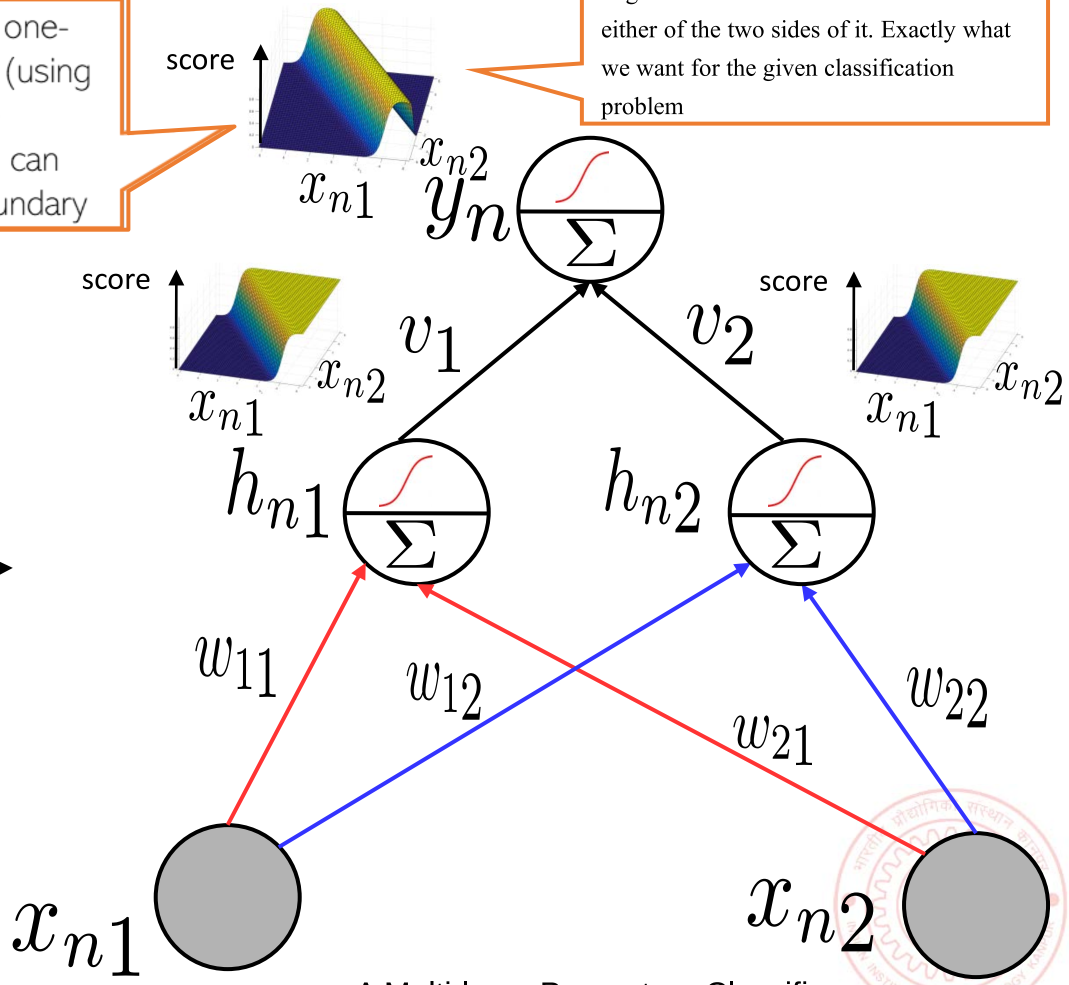
Standard Single "Perceptron" Classifier (no hidden units)

Obtained by composing the two one-sided increasing score functions (using $v_1 = 1$, and $v_2 = -1$ to "flip" the second one before adding). This can now learn nonlinear decision boundary



A nonlinear classification problem

High-score in the middle and low-score on either of the two sides of it. Exactly what we want for the given classification problem



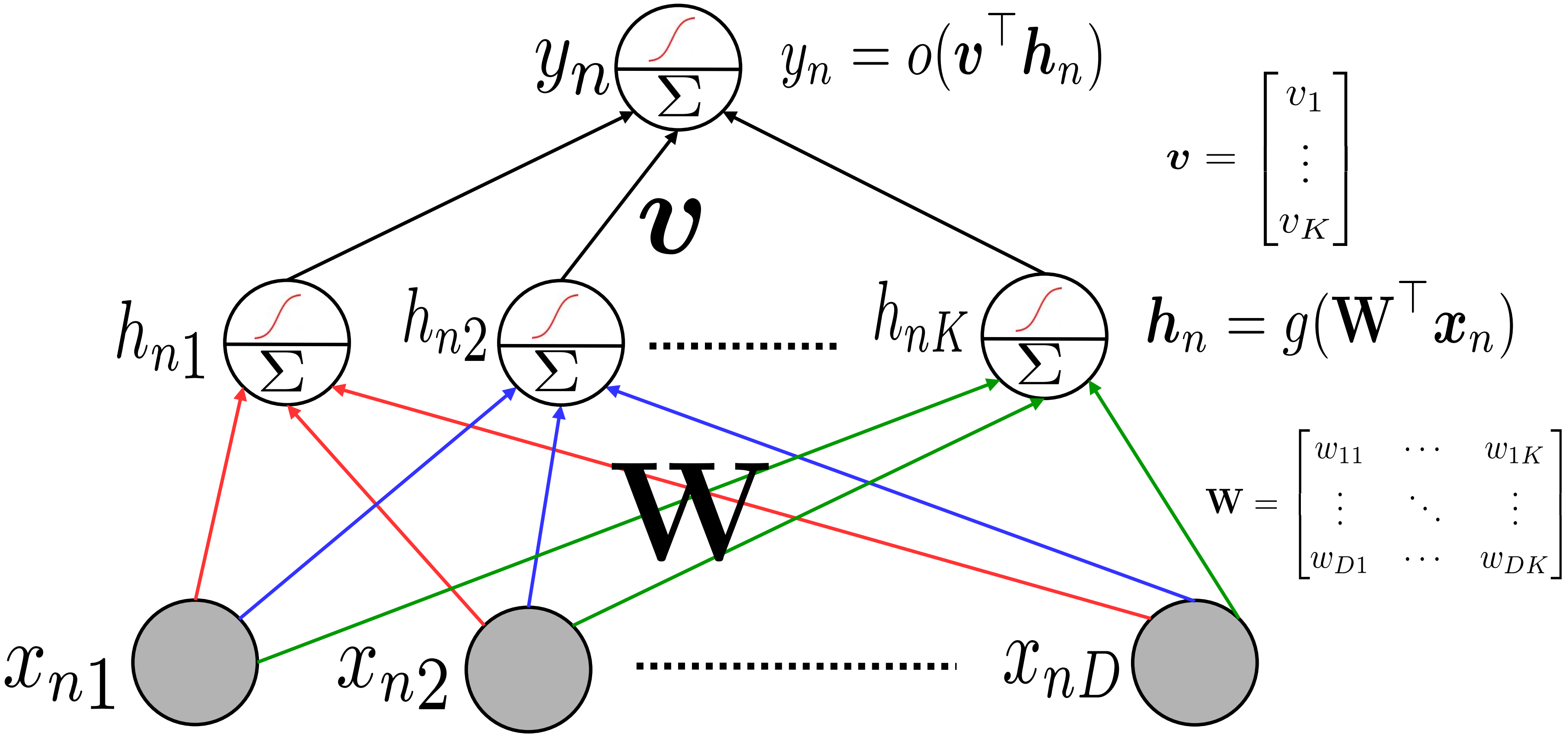
A Multi-layer Perceptron Classifier
(one hidden layer with 2 units)
Capable of learning nonlinear boundaries

A single hidden layer MLP with sufficiently large number of hidden units can approximate any function (Hornik, 1991)

Examples of some basic NN/MLP architectures

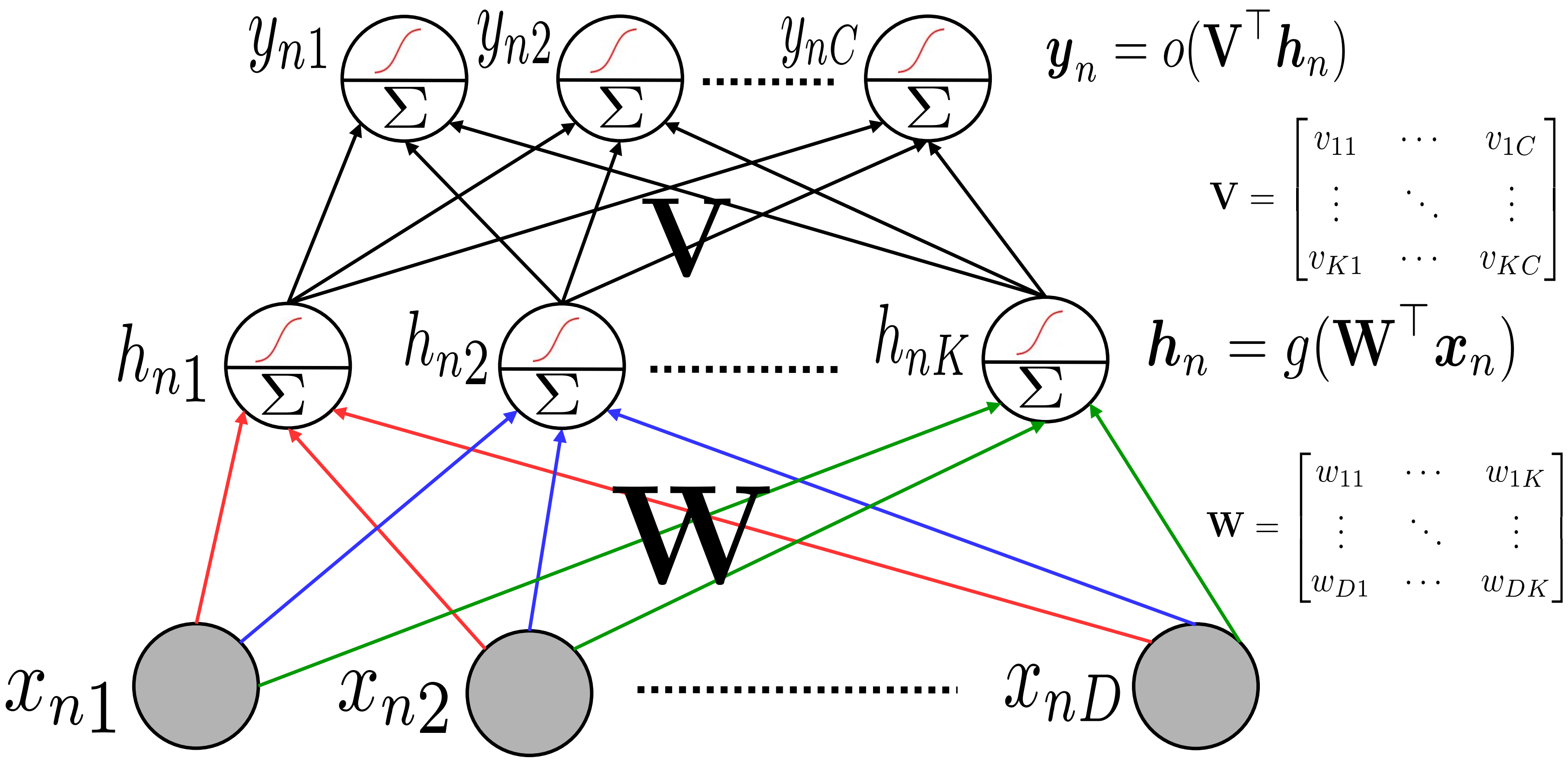
Single Hidden Layer and Single Outputs

- One hidden layer with K nodes and a single output (e.g., scalar-valued regression or binary classification)



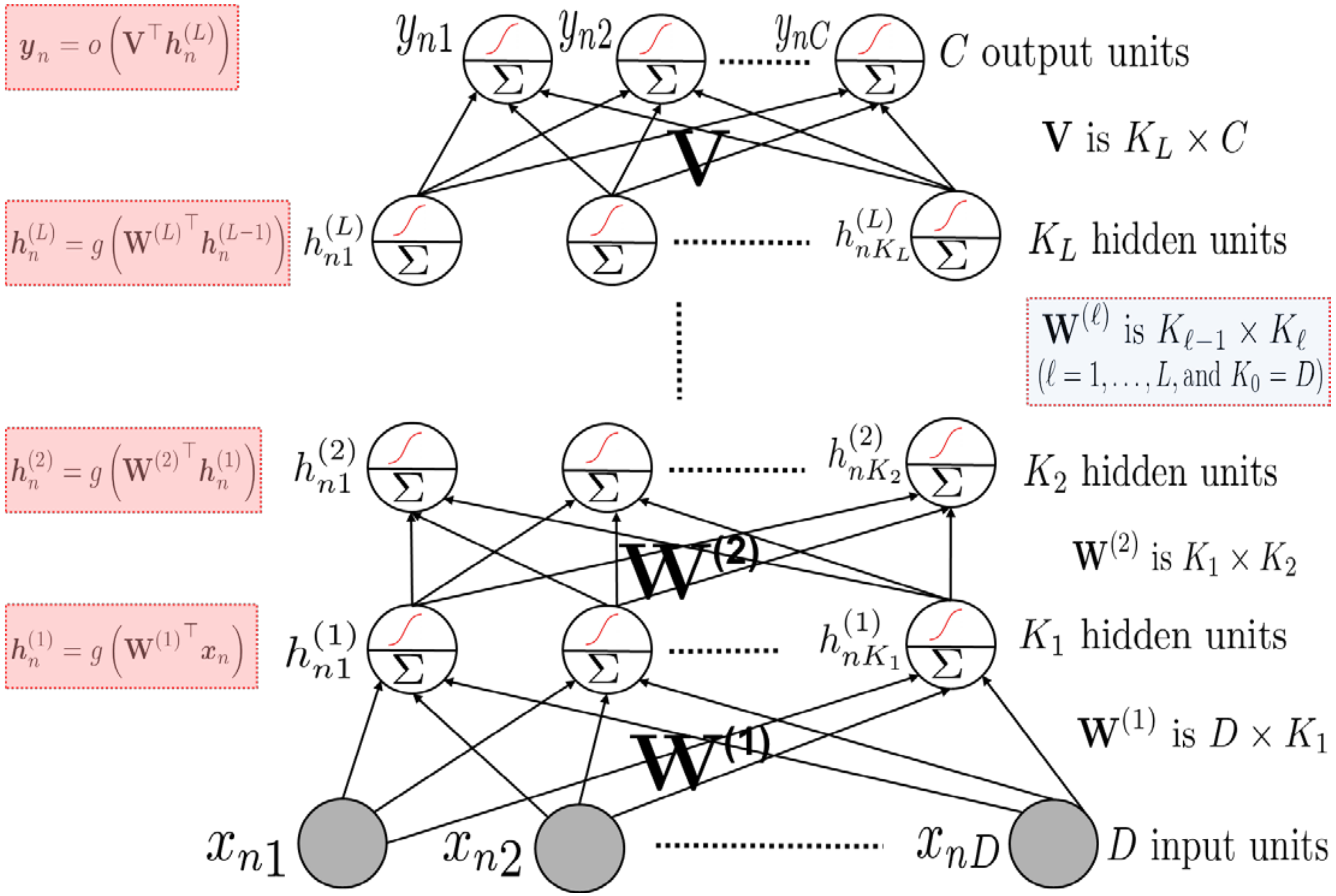
Single Hidden Layer and Multiple Outputs

- One hidden layer with $\boldsymbol{K}$ nodes and a vector of $\boldsymbol{C}$ output (e.g., vector-valued regression or multi-class classification or multi-label classification)



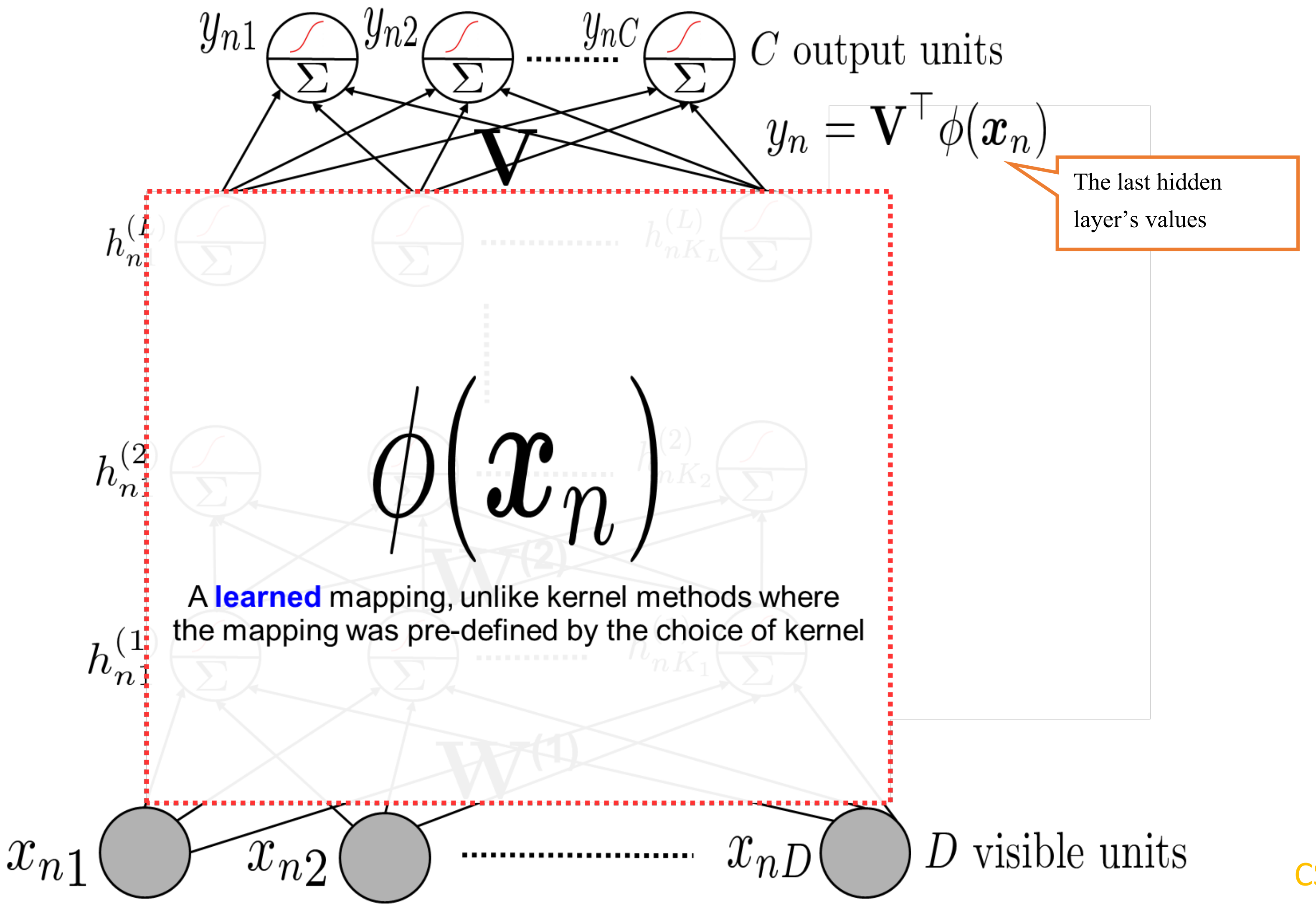
Multiple Hidden Layers (One/Multiple Outputs)

- Most general case: Multiple hidden layers with (with same or different number of hidden nodes in each) and a scalar or vector-valued output



Neural Nets are Feature Learners

- Hidden layers can be seen as learning a feature rep. $\boldsymbol{\phi}(\boldsymbol{x}_n)$ for each input $\boldsymbol{x}_n$



Kernel Methods vs Neural Nets

- Recall the prediction rule for a kernel method (e.g., kernel SVM)

$$y = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- This is analogous to a single hidden layer NN with fixed/pre-defined hidden nodes $\{k(\boldsymbol{x}_n, \boldsymbol{x})\}_{n=1}^{N}$ and output weights $\{\alpha_n\}_{n=1}^{N}$
- The prediction rule for a deep neural network

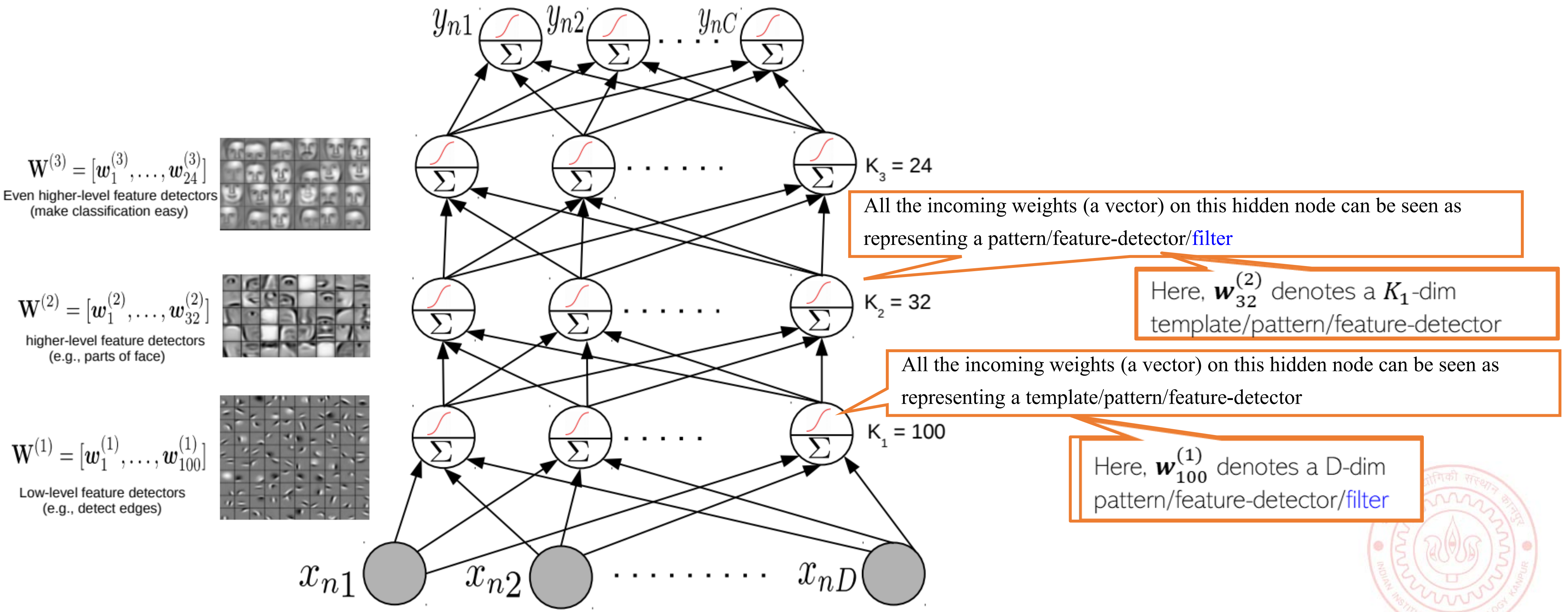
$$y = \sum_{k=1}^{K} v_k h_k$$

Also note that neural nets are faster than kernel methods at test time since kernel methods need to store the training examples at test time whereas neural nets do not

- Here, the $\boldsymbol{h}_k$'s are learned from data (possibly after multiple layers of nonlinear transformations)
- Both kernel methods and deep NNs be seen as using nonlinear basis functions for making predictions. Kernel methods use fixed basis functions (defined by the kernel) whereas NN learns the basis functions adaptively from data

Feature Learned by a Neural Network

- Node values in each hidden layer tell us how much a "learned" feature is active in $\boldsymbol{x}_n$
- Hidden layer weights are like pattern/feature-detector/filter



Why Neural Networks Work Better: Another View

- Linear models tend to only learn the "average" pattern
- Deep models can learn multiple patterns (each hidden node can learn one pattern)
  - Thus deep models can learn to capture more subtle variations that a simpler linear model