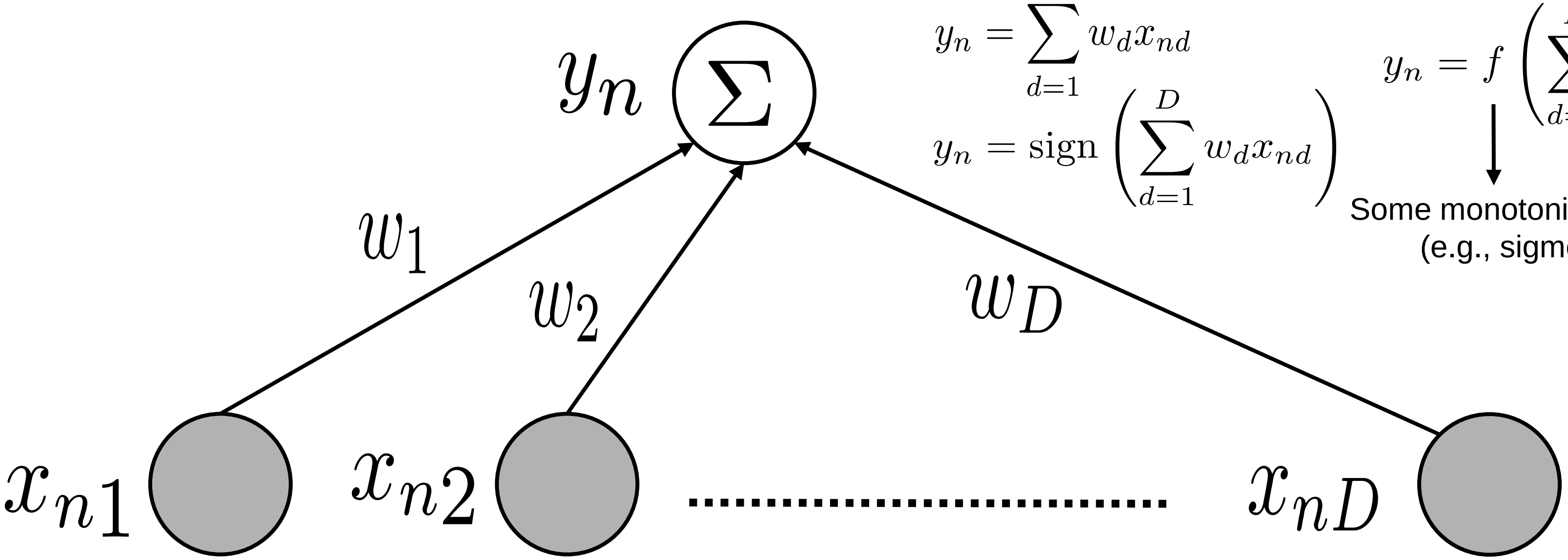
CMSC 471

Neural Network (Additional slides)

Slide courtesy: 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

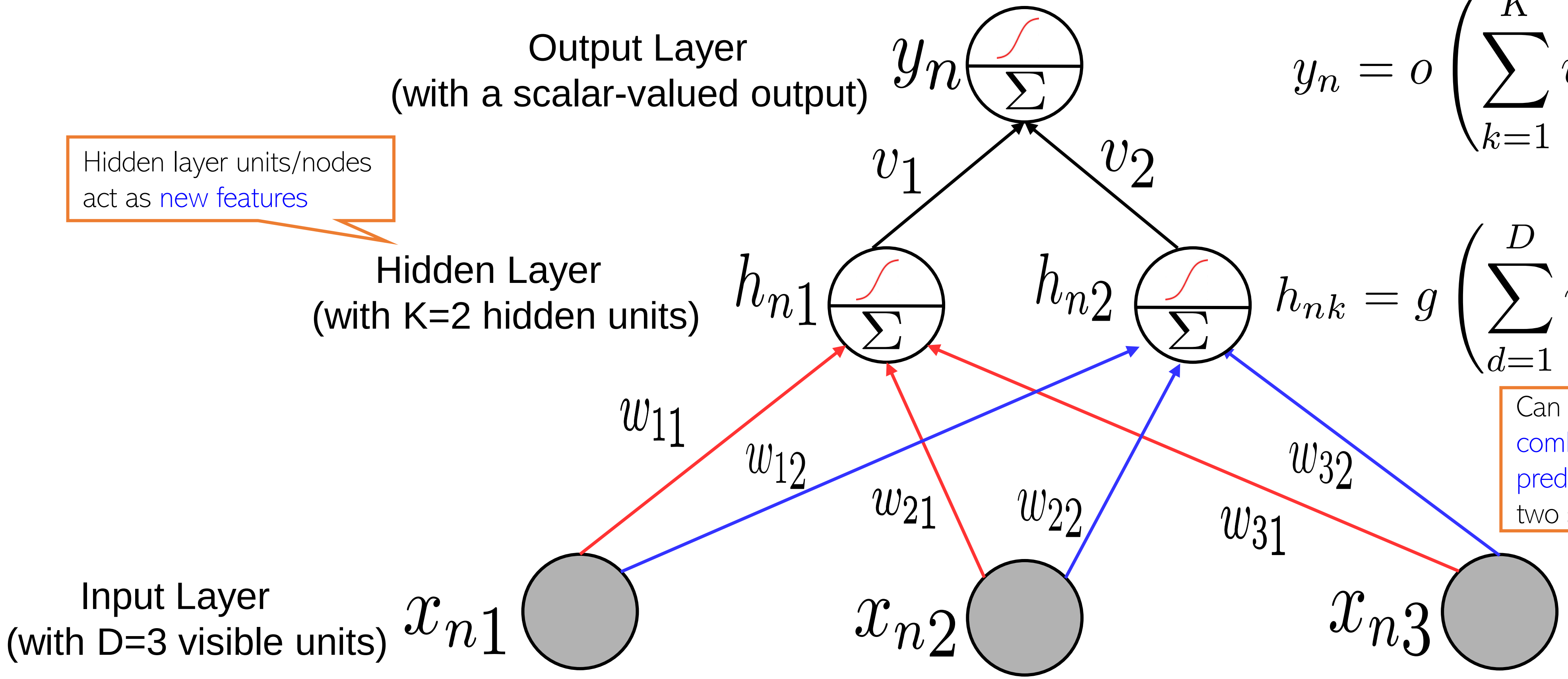
- Non-linear models: kNN, kernel methods, generative classification, decision trees etc.
- All have their own disadvantages
- kNN and kernel methods are expensive to generate predictions from
- Kernel based and generative models particularize the decision boundary to a particular class of functions, 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

$$a_{nk} = \sum_{d=1}^{D} w_{dk} x_{nd}$$

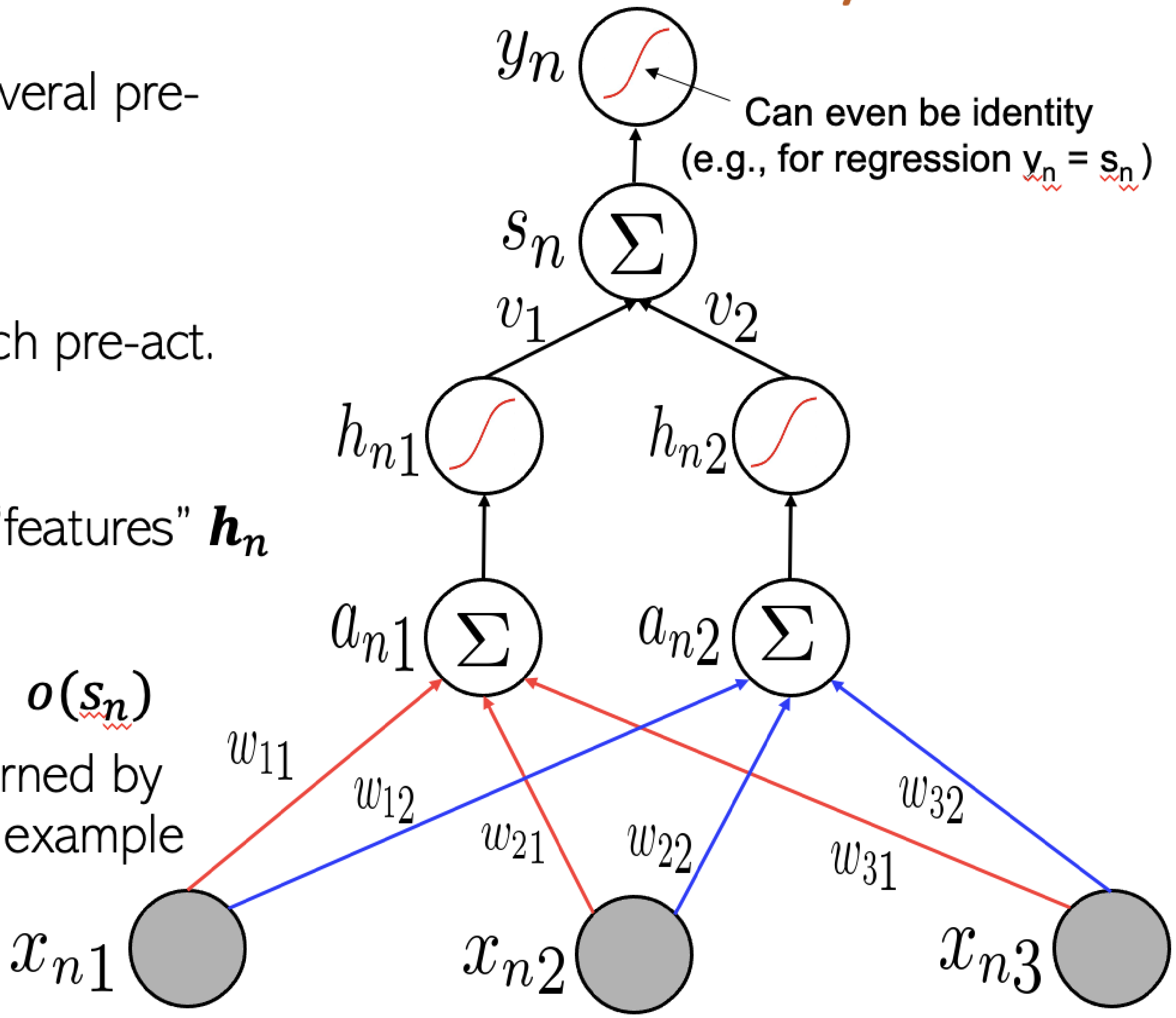
- Nonlinear activation applied on each pre-act.

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

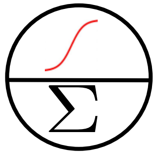
- Linear model learned on the new "features" $\boldsymbol{h}_n$

$$s_n = \sum_{k=1}^{K} v_k h_{nk}$$

- Finally, output is produced as $\boldsymbol{y} = \boldsymbol{o}(\boldsymbol{s_n})$
- 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)

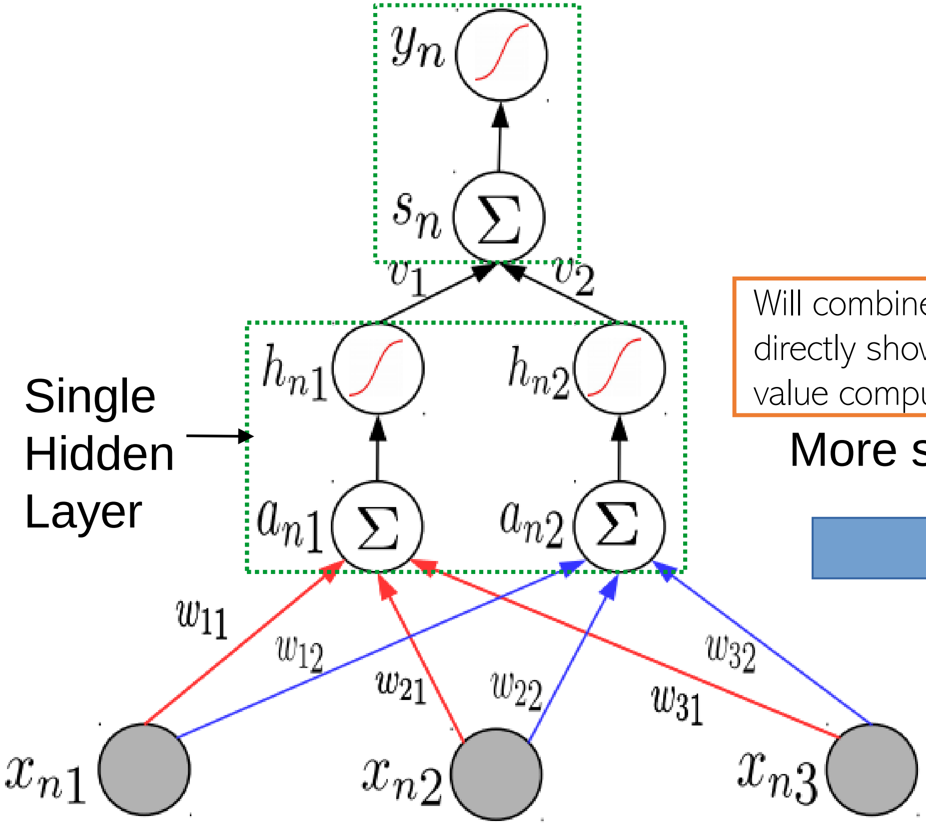


Neural Nets: A Compact Illustration



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

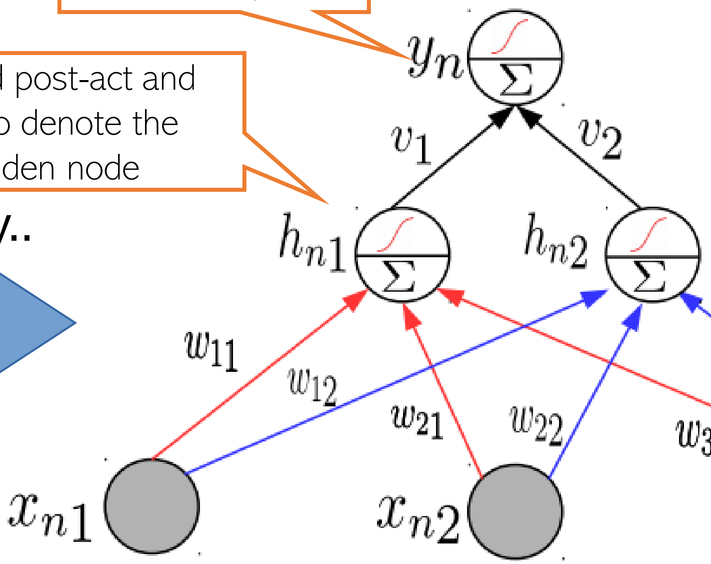
- Note: Hidden layer pre-act a_{nk} and post-act h_{nk} will be shown together for brevity



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

More succinctly..

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.

Gradient Descent

- Unknowns (w1, w2, …, wk, v) learned by minimizing loss function.
- In order to estimate how much you should increase or decrease your weights based on a loss, we can take the gradient of the loss function w.r.t to our unknowns (w1, w2, …, wk, v) and move our unknowns in the direction of the gradient, i.e.

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \frac{\partial loss}{\partial W}_{\text{old}}$$

Let's recalculate v1 using gradient descent

- Let the loss be: Y_{actual} - y_n
- To re-evaluate v1, we will do:
  - $V1_{new} = v1_{old} - \frac{dLoss}{dv1}$
- Let's calculate $\frac{dLoss}{dv1}$

- $\frac{dLoss}{dv1} = \frac{d(Y_{actual} - y_n)}{dv1} = \frac{d(Y_{actual})}{dv1} - \frac{d(y_n)}{dv1} = 0 - \frac{d(o(sn))}{dv1}$

Y_{actual} is independent of v1

Let's recalculate v1 using gradient descent

- Let us take the simplest case where there is no activation function i.e. $o(sn)) = s_n$

Therefore,

$$\frac{dLoss}{d\mathrm{v1}} = - \frac{d(\mathrm{s_n})}{d\mathrm{v1}} = - \frac{d(\sum_{k=1}^{K} v_k h_{nk})}{d\mathrm{v1}} = \mathrm{h_{n1}} = \mathrm{w_{11}x_{n1}} + \mathrm{w_{21}x_{n2}} + \mathrm{w_{31}x_{n3}}$$