Deep Learning

CS771: Introduction to Machine Learning

Purushottam Kar

A New Perspective on Non-linear Learning

Kernels allow us to nicely reuse linear algos to learn nonlinear models

- *Very powerful (and expensive too) – offer non-parametric learning*
- *Not all ML algos can be easily kernelized – e.g. those that use L_1 regularization*
- *In general, algos that use L_2 distances and dot products are kernelizable*

Other techniques exist to perform non-linear learning as well

- *kNN is one such prominent example – also a non-parametric algo*
- *Decision trees offer very good performance and very fast prediction too*
- *Other "parametric" methods such as splines, sinusoids popular in statistics*

Another successful non-linear learning technique is deep learning

- *Predates kernel learning and machine learning itself by decades*
- *Not very popular till recently due to lack of data, computational power*
- *Very successful and well-researched technique currently*

From Kernel Learning to Deep Learning

Notice that kernel learning essentially involves learning a linear model over new features. The new features are given by the kernel map ϕ_K

- *The num of features is large (infinite?) which causes kernel algos to be slow*
- *Several techniques to speed up kernel methods address this exact problem*
- *Landmarking/Nystrom create new features using similarity values*
- *Random feature methods also create new features from rank 1 kernels*

Given a budget k can we directly learn k good features?

- *In some sense, deep learning does exactly this*
- *Several deep networks basically run a linear ML algo on top of learnt features*
- ***Key difference**: instead of first learning good features and then learning a linear model on top of those features, deep learning learns both jointly*
- ***Drawbacks**: non-convex problem, over parameterized, can be bulky/slow*

Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$

Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$

$\mathbf{x}_1$ $\mathbf{x}_2$

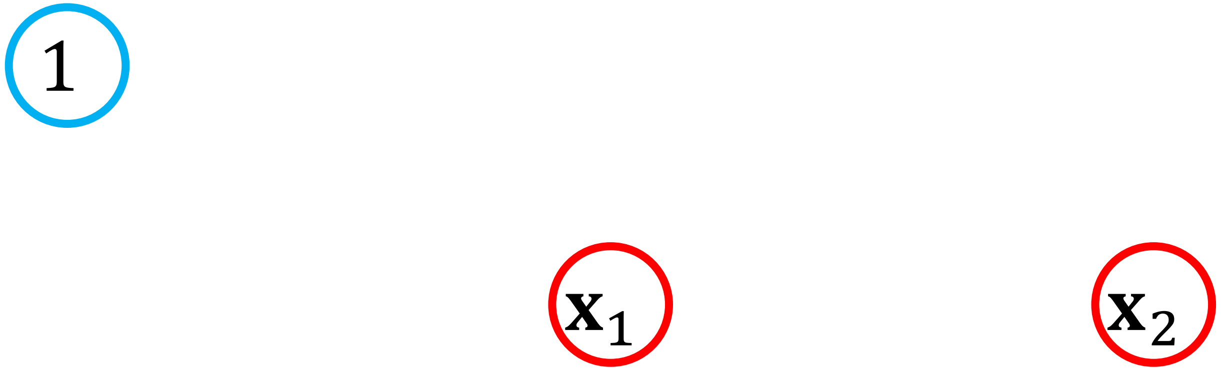
Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



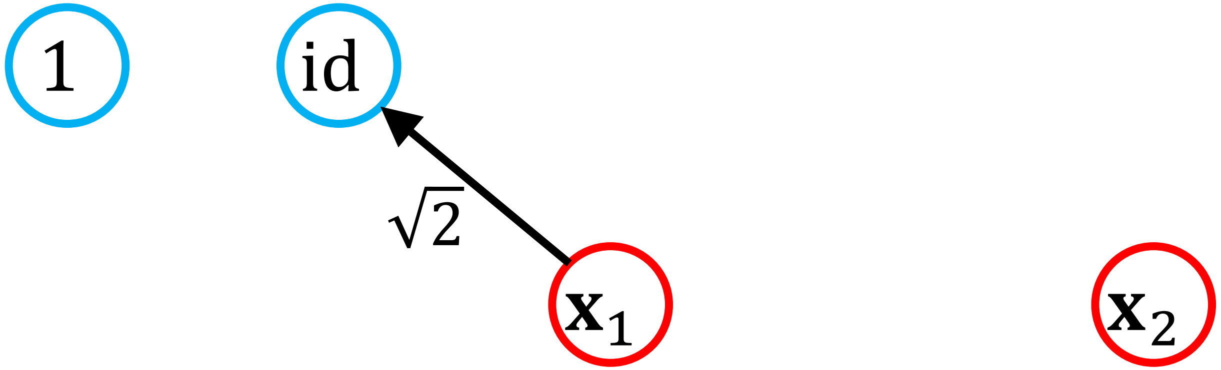
Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



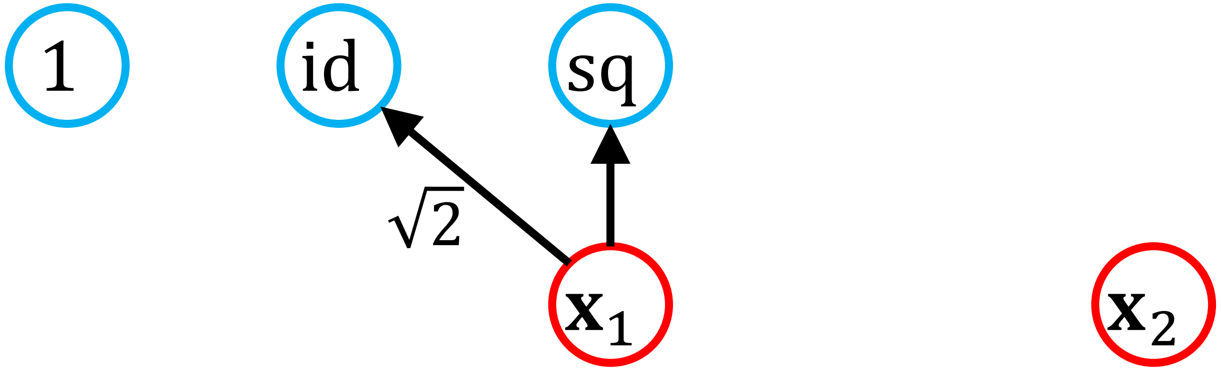
Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



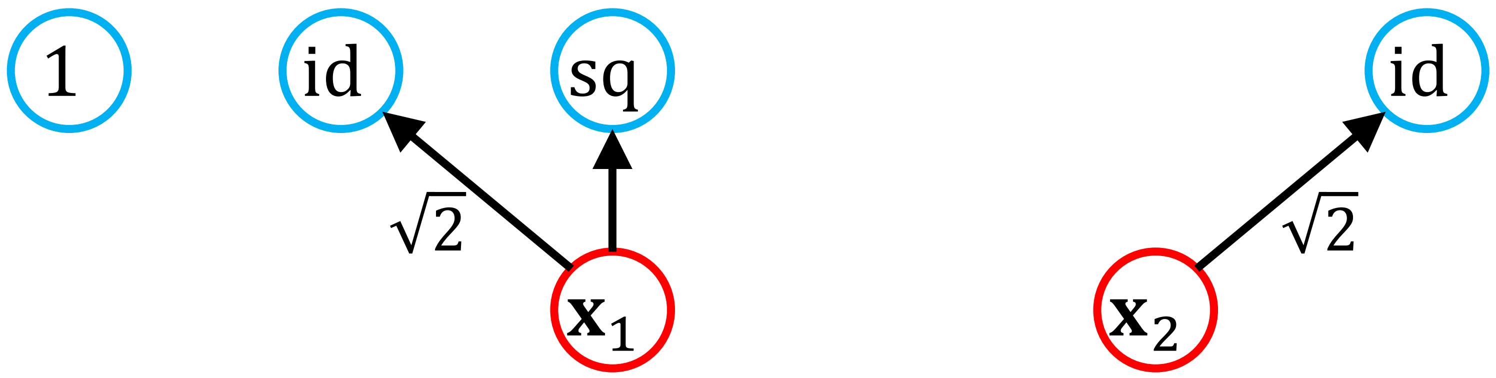
Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



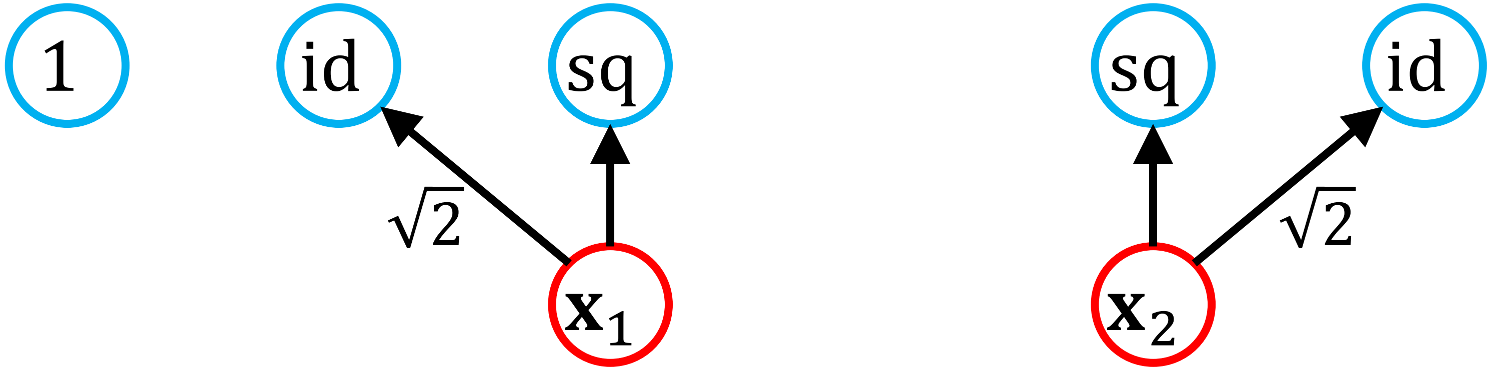
Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\mathrm{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\mathrm{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



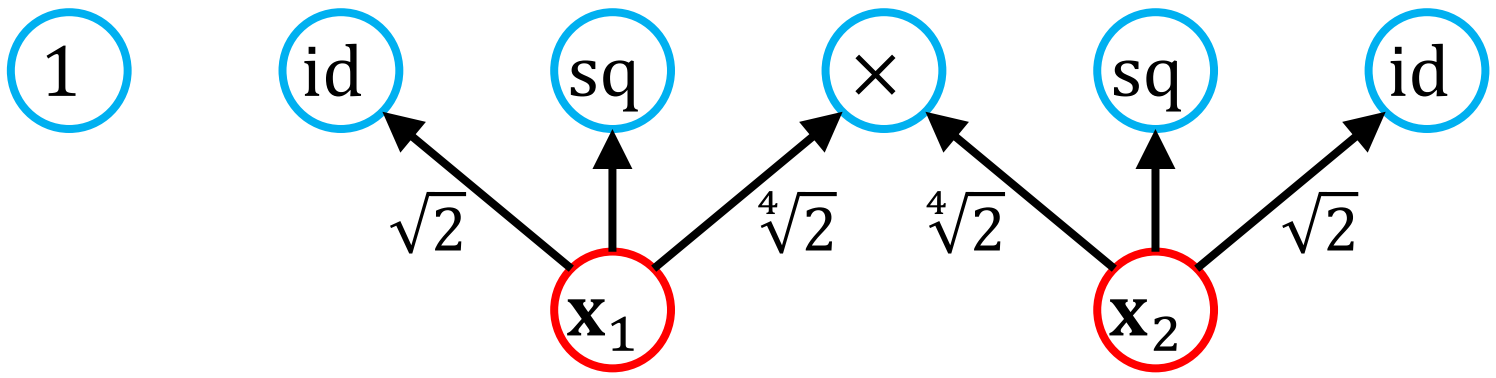
Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



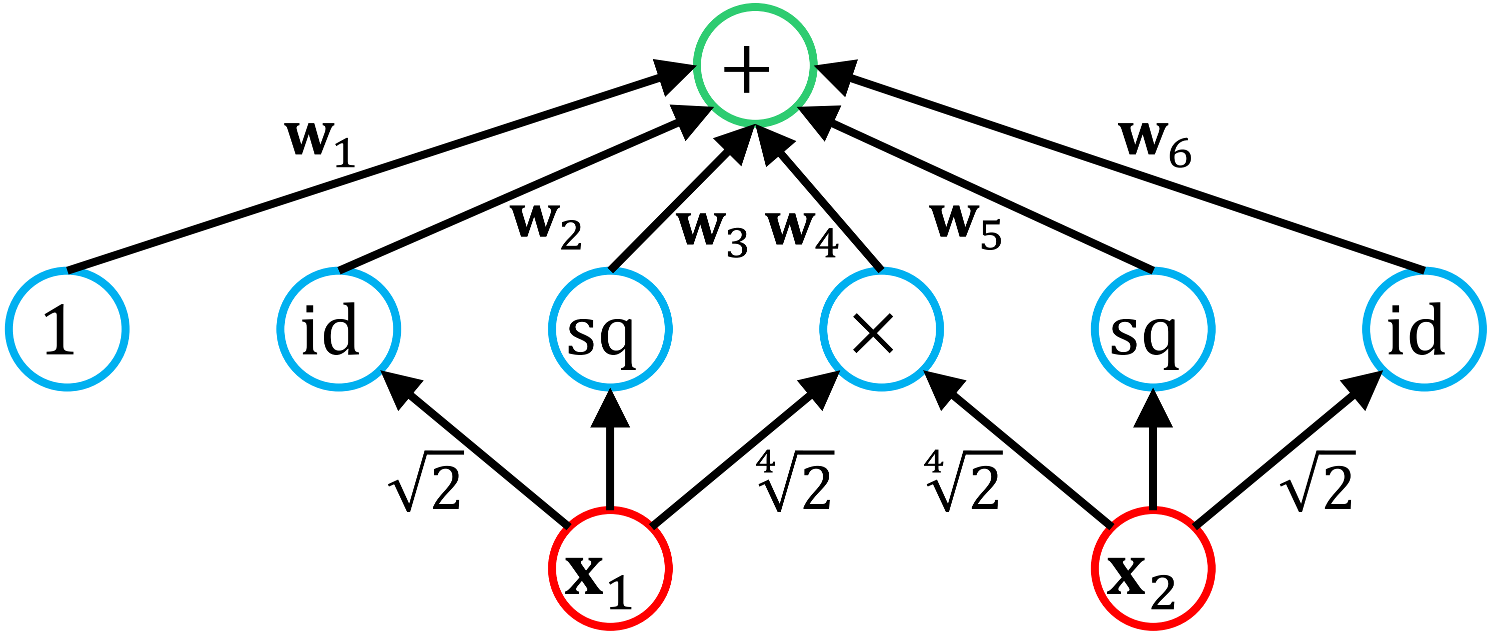
Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



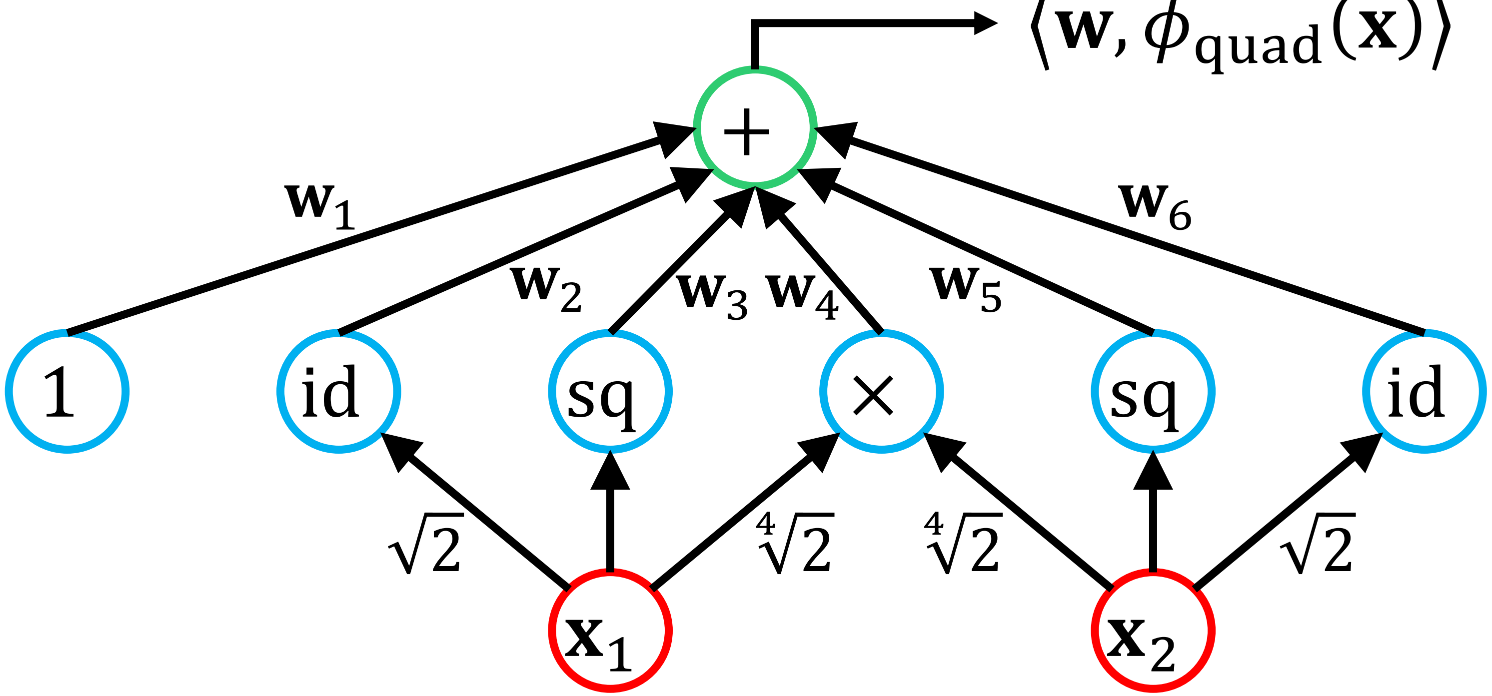
Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

A linear model over ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



Kernel Learning viewed as Deep Learning

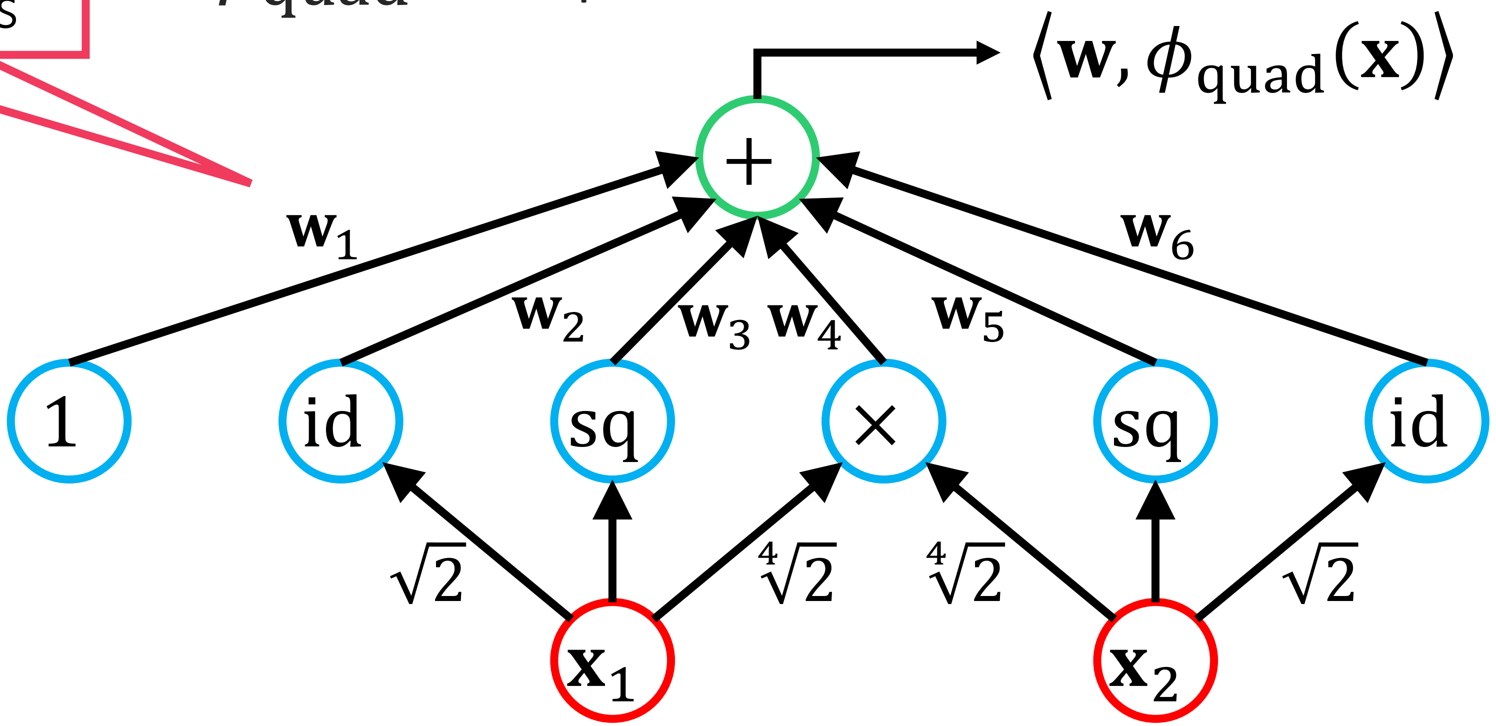
Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = \left[1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2\right] \in \mathbb{R}^6$$

Training kernel SVM/RR tunes these weights

ver ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

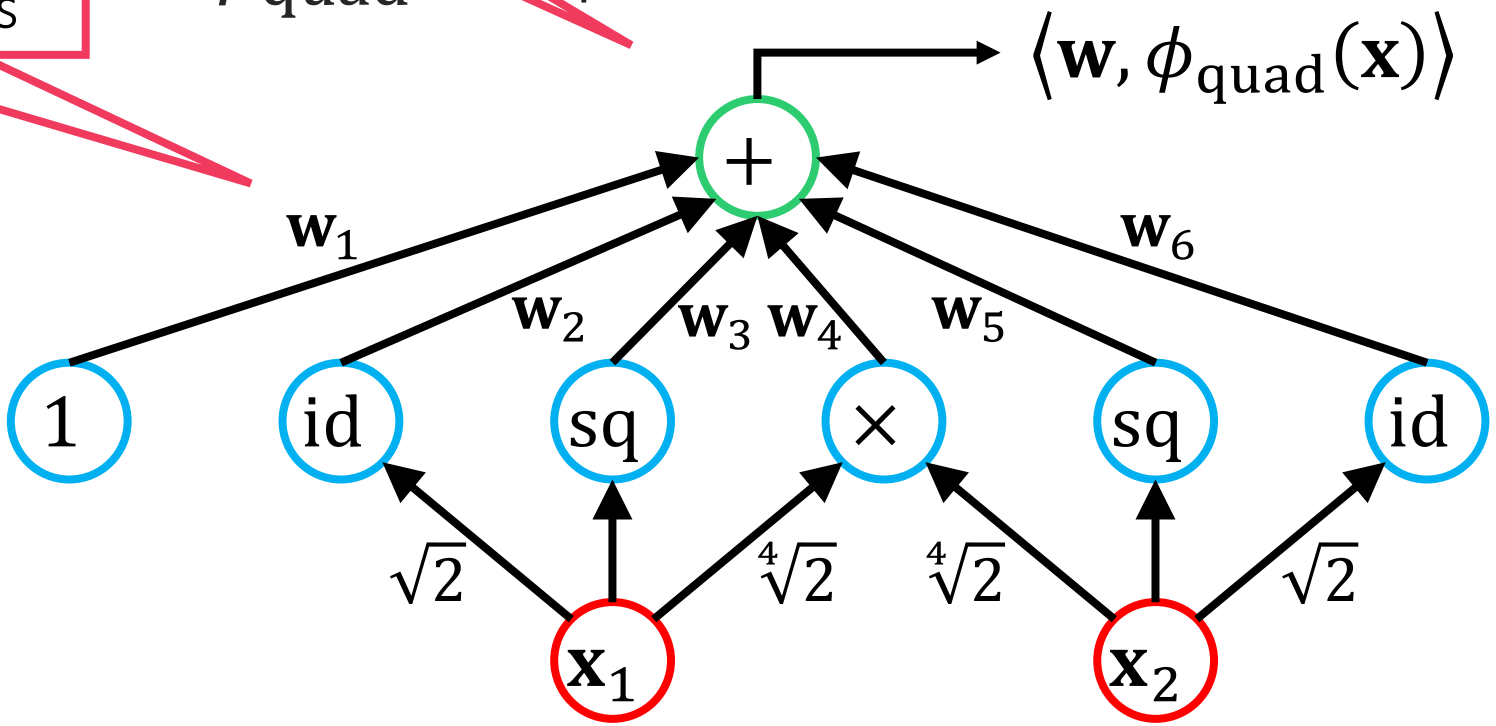
The featu ... uad is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = [1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2] \in \mathbb{R}^6$$

... ver ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$

A 3 layer network

Training kernel SVM/RR tunes these weights



Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

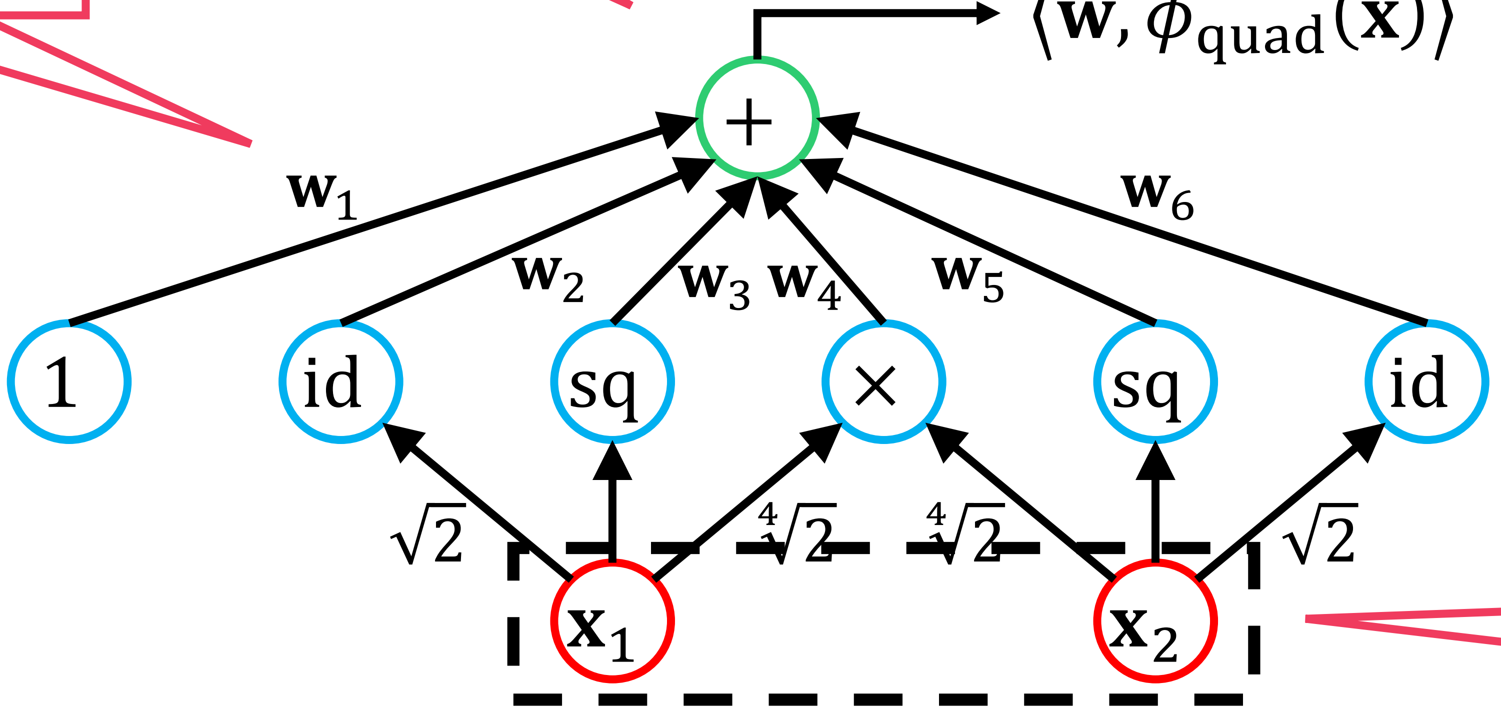
The featu… …uad is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = [1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2] \in \mathbb{R}^6$$

…ver ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$

A 3 layer network

Training kernel SVM/RR tunes these weights



Input layer

Kernel Learning viewed as Deep Learning

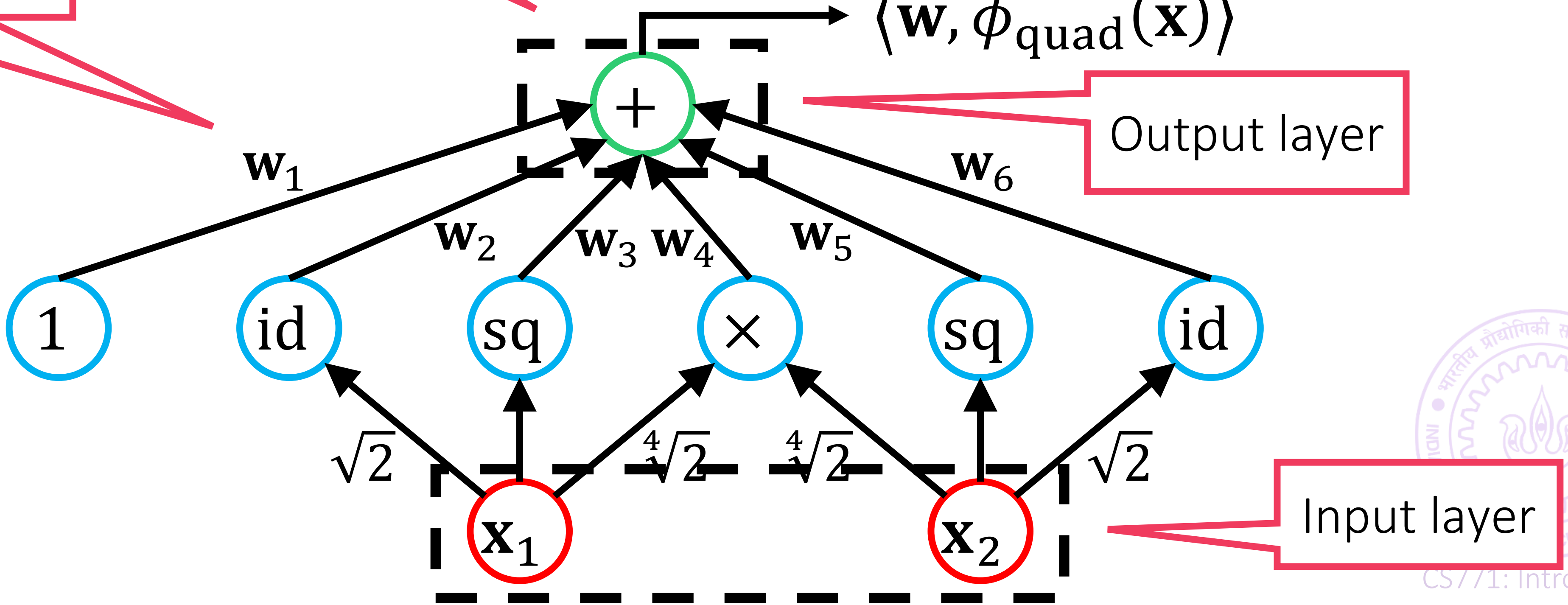
Consider the quadratic kernel $K_{\mathbf{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The featu A 3 layer network uad is $\phi_{\mathbf{quad}}$ where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\mathbf{quad}}(\mathbf{x}) = [1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2] \in \mathbb{R}^6$$

Training kernel SVM/RR tunes these weights

ver $\phi_{\mathbf{quad}}$ is represented as a vector $\mathbf{w} \in \mathbb{R}^6$



Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The featu… uad is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

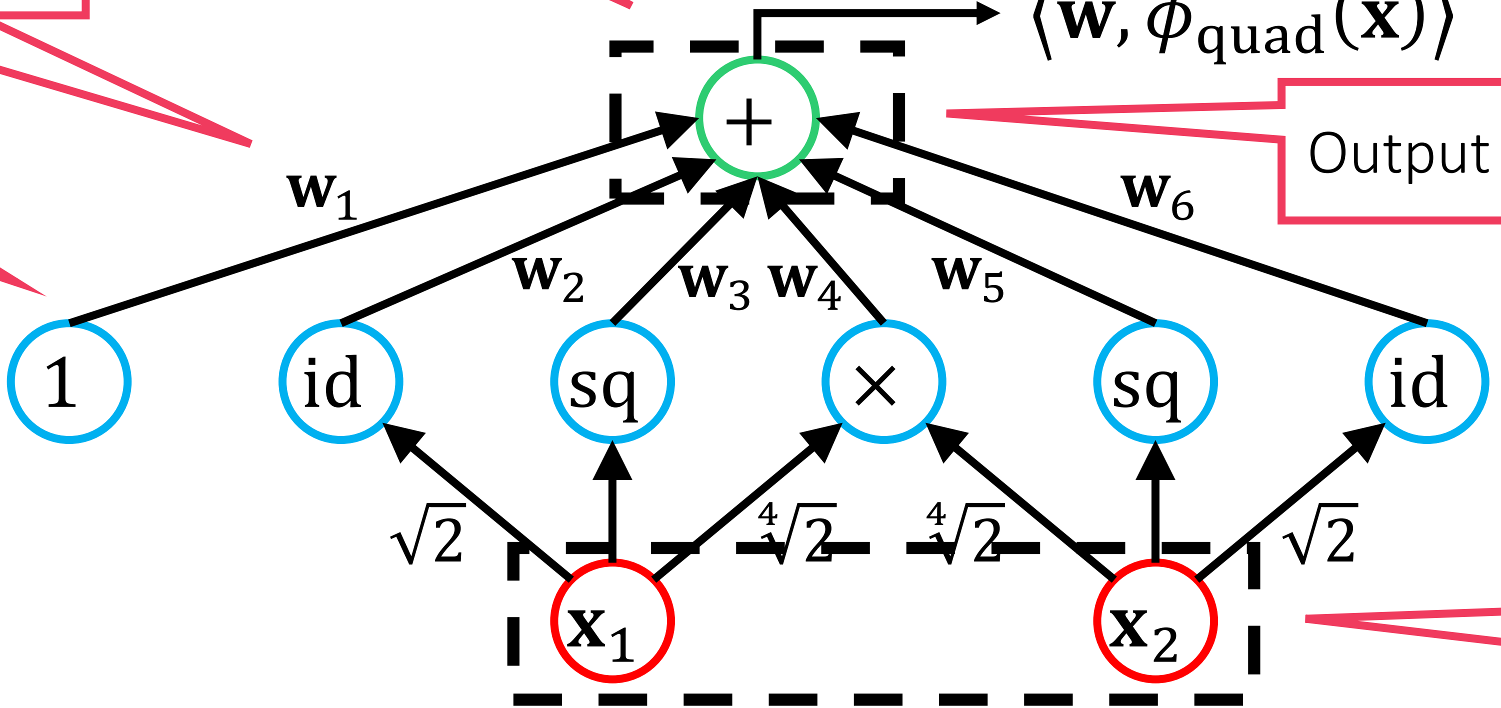
$$\phi_{\text{quad}}(\mathbf{x}) = [1, \sqrt{2}\cdot\mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2}\cdot\mathbf{x}_1\cdot\mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2}\cdot\mathbf{x}_2] \in \mathbb{R}^6$$

…ver ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$

A 3 layer network

Training kernel SVM/RR tunes these weights

I/O layers are called "visible"



Output layer

Input layer

Kernel Learning viewed as Deep Learning

Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

The featu[...]uad is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

$$\phi_{\text{quad}}(\mathbf{x}) = [\ldots 2 \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2] \in \mathbb{R}^6$$

[...]ver ϕ_{quad} is represented as a vector $\mathbf{w} \in \mathbb{R}^6$

A 3 layer network

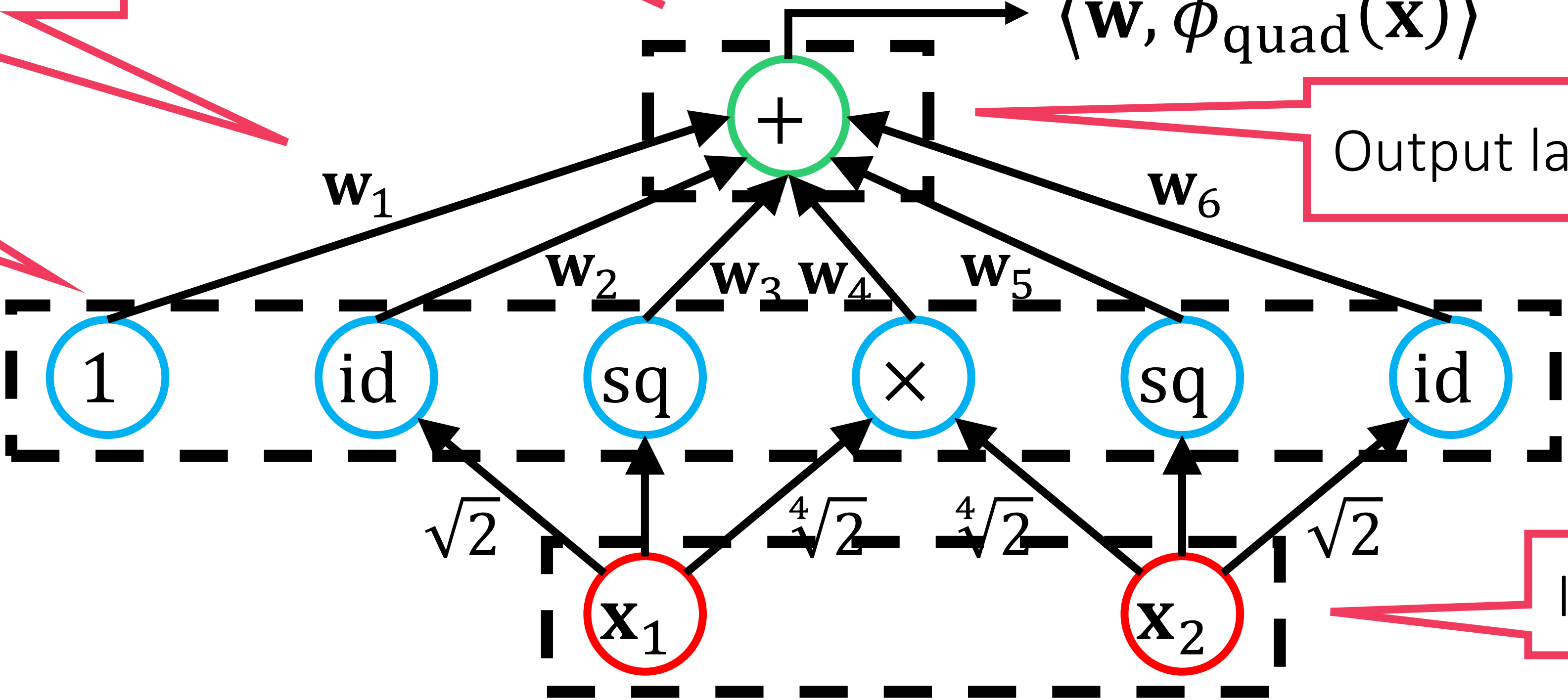
Training kernel SVM/RR tunes these weights

I/O layers are called “visible”

“Hidden” layer Usually many of these

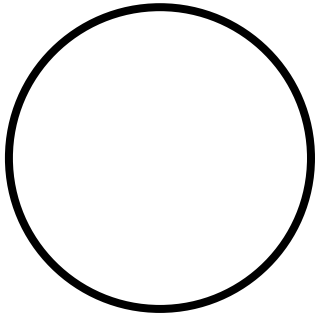
Output layer

Input layer

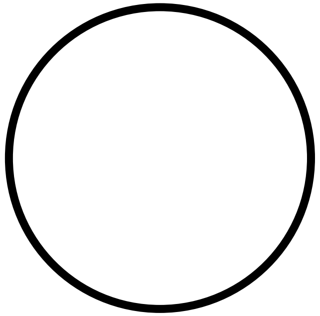


The “Neuron” in Deep/Neural Networks

The “Neuron” in Deep/Neural Networks

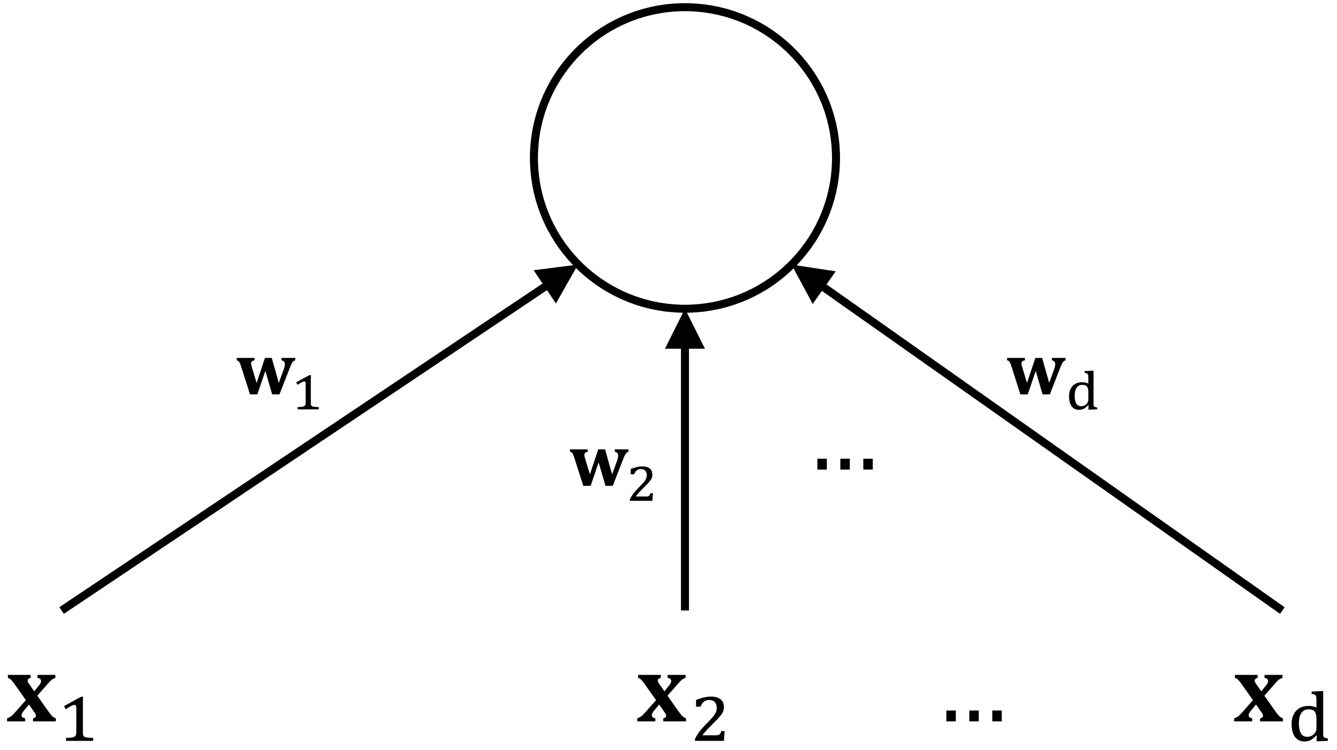


The “Neuron” in Deep/Neural Networks

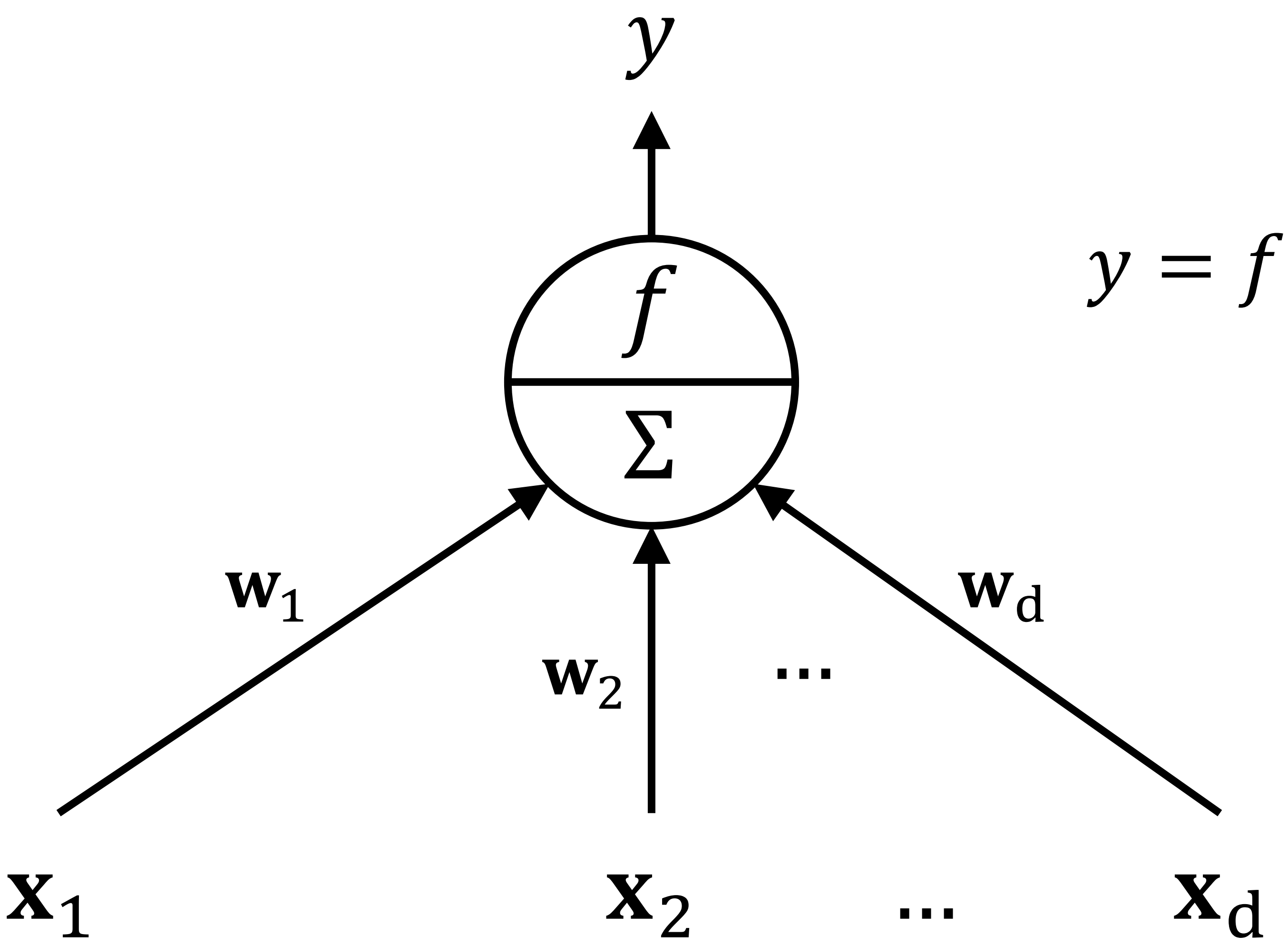


$\mathbf{x}_1$ $\mathbf{x}_2$ … $\mathbf{x}_\mathrm{d}$

The “Neuron” in Deep/Neural Networks

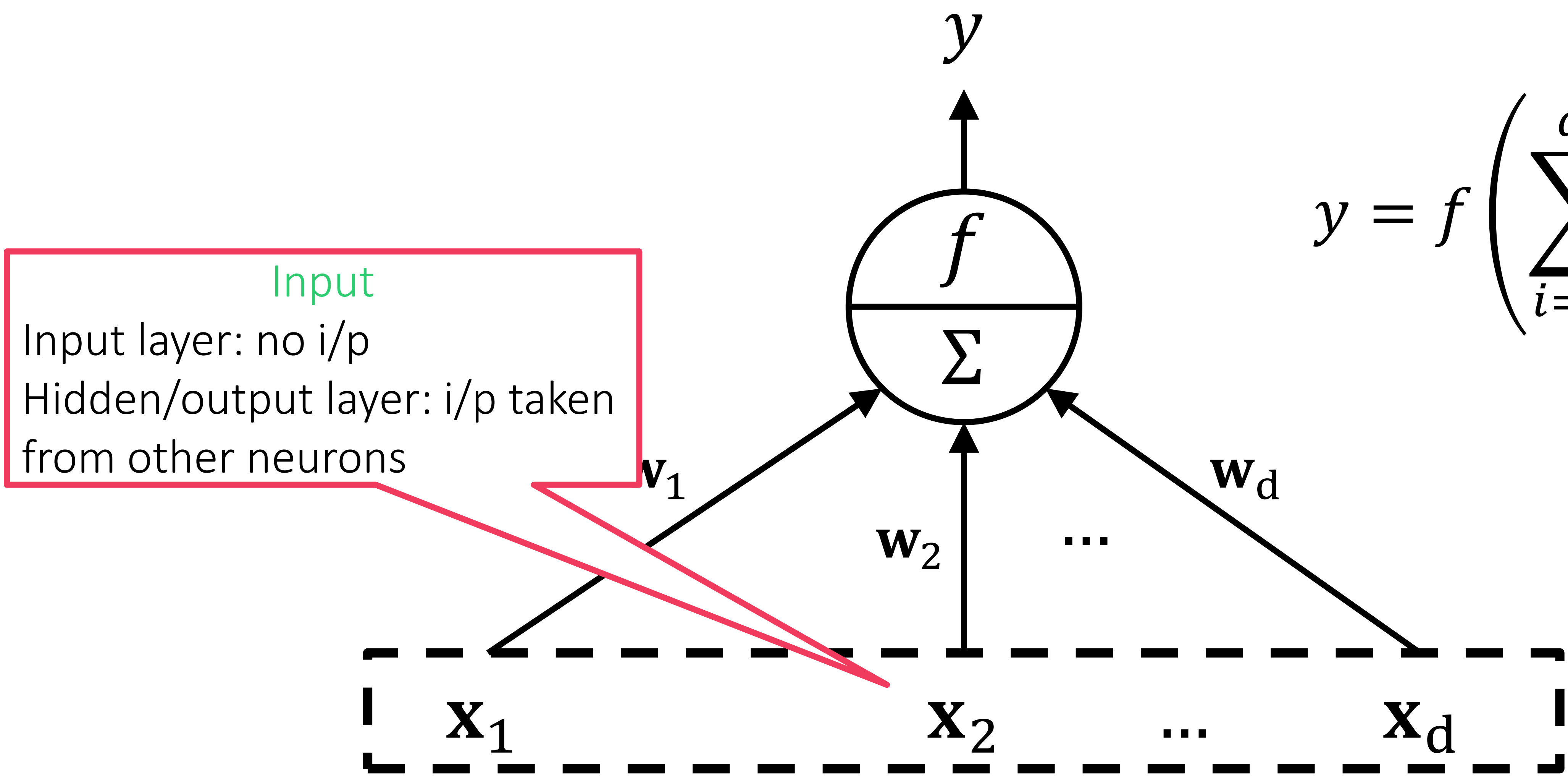


The "Neuron" in Deep/Neural Networks



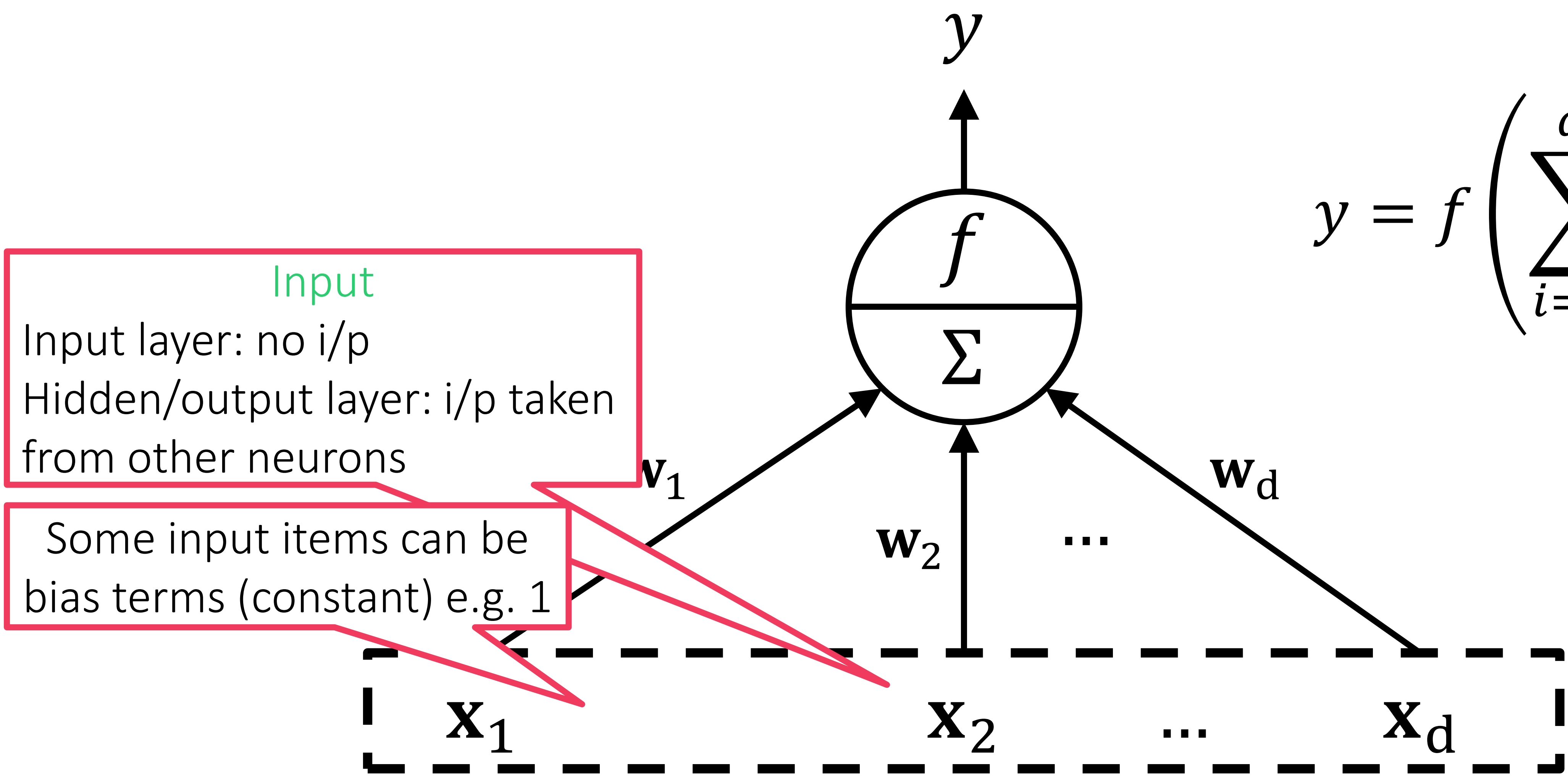
$$y = f\left(\sum_{i=1}^{d} \mathbf{w}_i \cdot \mathbf{x}_i\right)$$

The "Neuron" in Deep/Neural Networks



$$y = f\left(\sum_{i=1}^{d} \mathbf{w}_i \cdot \mathbf{x}_i\right)$$

The “Neuron” in Deep/Neural Networks



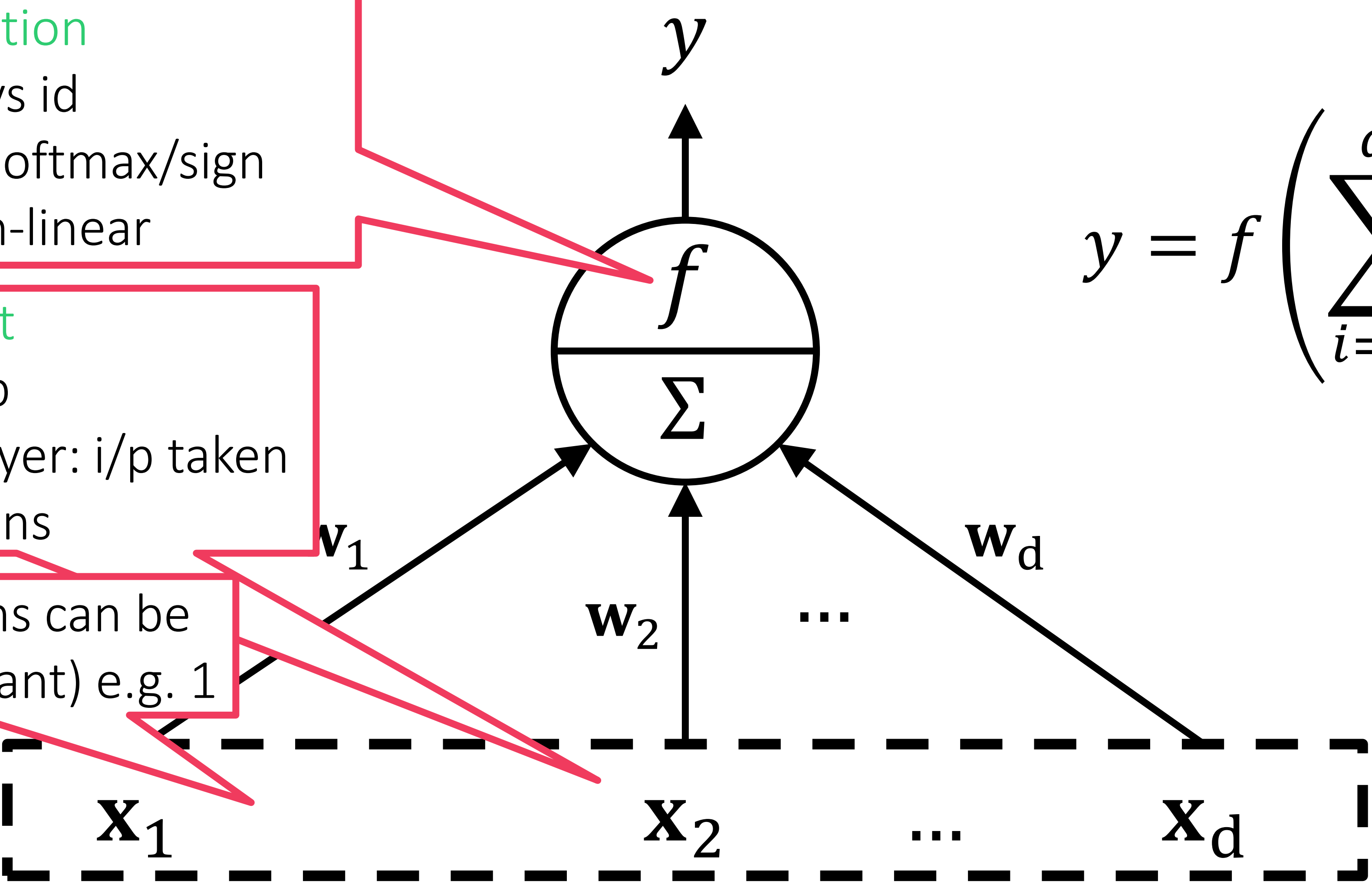
$$y = f\left(\sum_{i=1}^{d} \mathbf{w}_i \cdot \mathbf{x}_i\right)$$

The “Neuron” in Deep/Neural Networks

Activation
Input layer: always id
Output layer: id/softmax/sign
Hidden layer: non-linear

Input
Input layer: no i/p
Hidden/output layer: i/p taken from other neurons

Some input items can be bias terms (constant) e.g. 1



$$y = f\left(\sum_{i=1}^{d} \mathbf{w}_i \cdot \mathbf{x}_i\right)$$

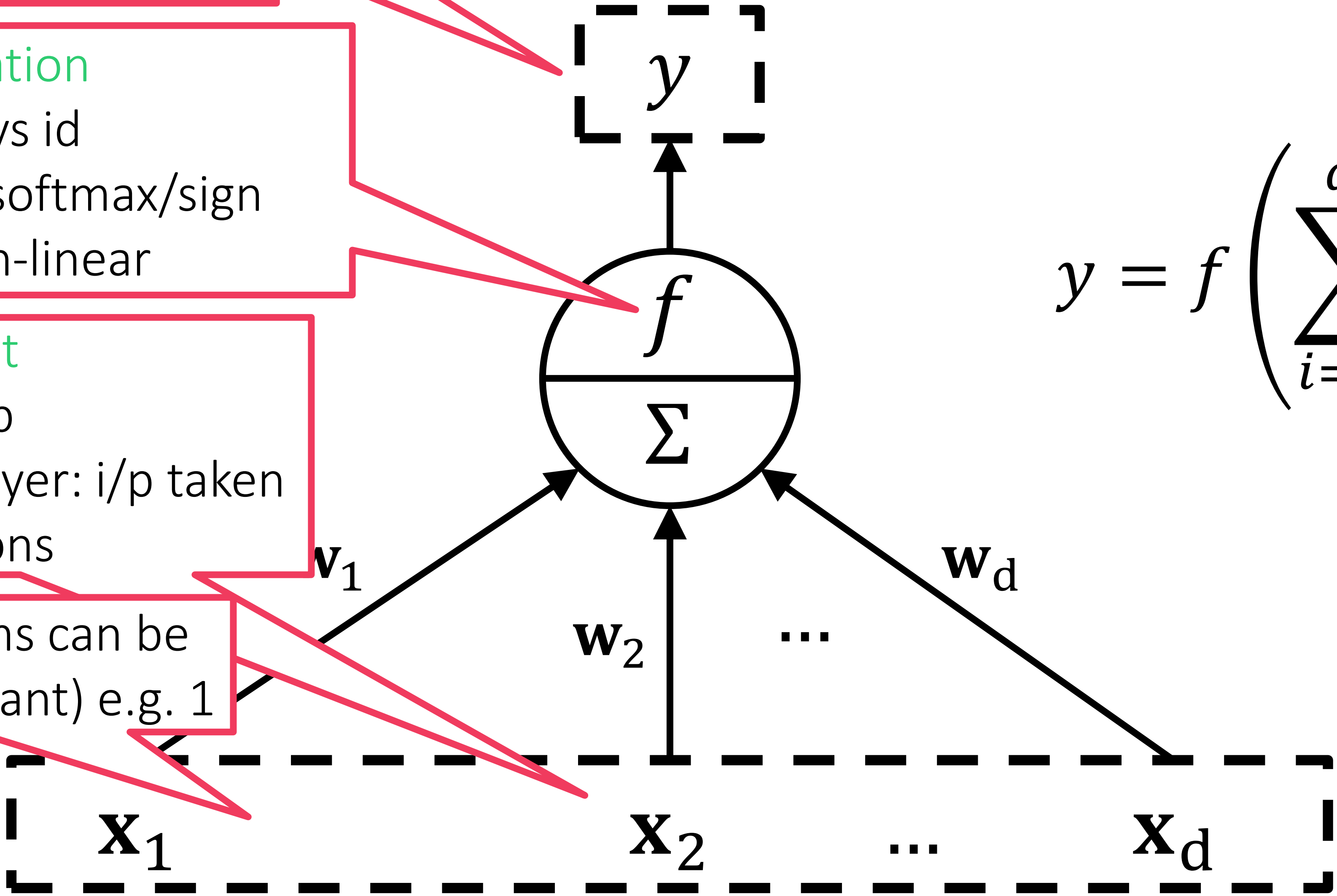
in Deep/Neural Networks

Output
Output layer: final o/p
Input/hidden layer: o/p to other neurons

Activation
Input layer: always id
Output layer: id/softmax/sign
Hidden layer: non-linear

Input
Input layer: no i/p
Hidden/output layer: i/p taken from other neurons

Some input items can be bias terms (constant) e.g. 1



$$y = f\left(\sum_{i=1}^{d} \mathbf{w}_i \cdot \mathbf{x}_i\right)$$

Output

Output layer: final o/p

Input/hidden layer: o/p to other neurons

If no hidden layer then the network is just a generalized linear model, also called a **perceptron**

Activation

Input layer: always id

Output layer: id/softmax/sign

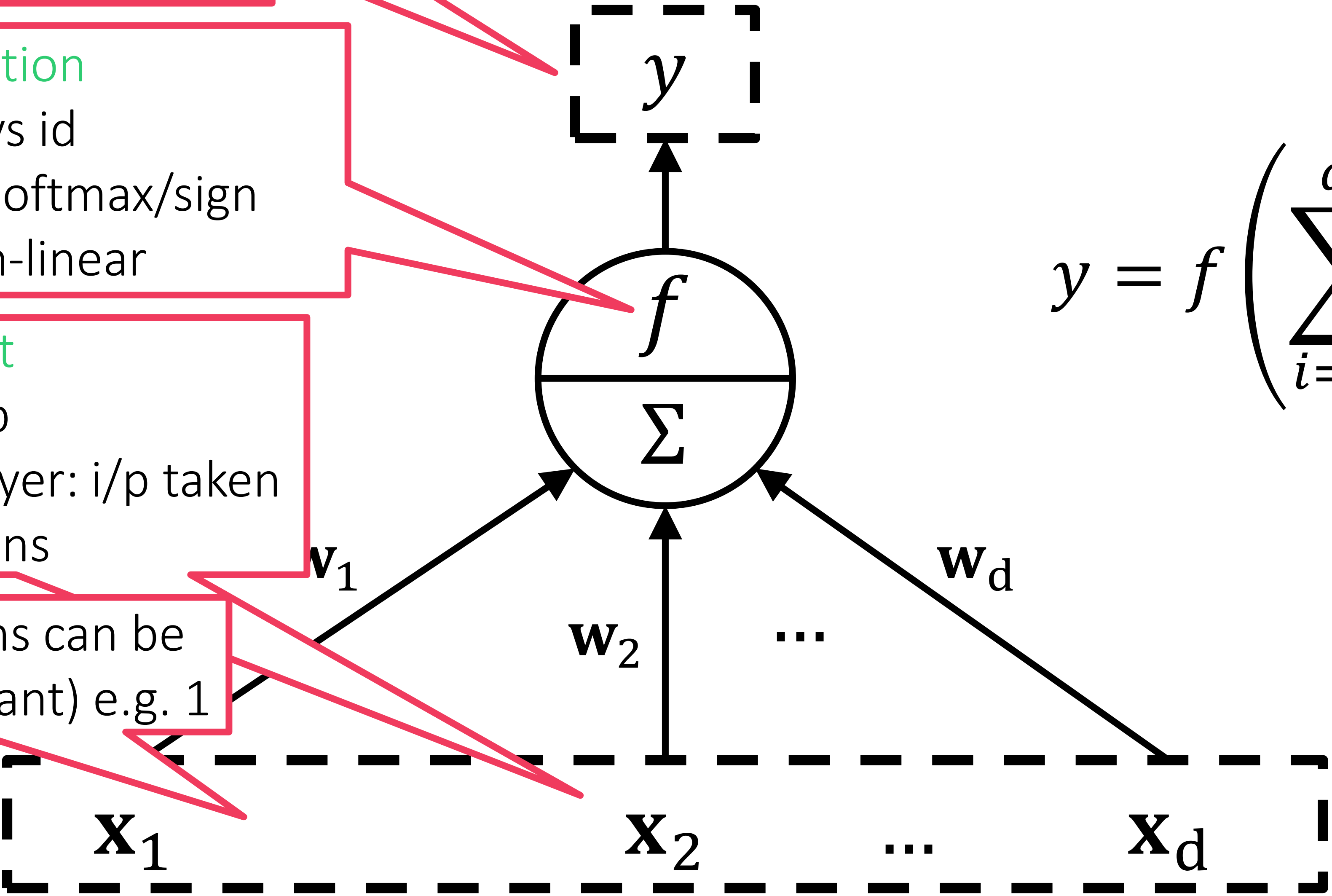
Hidden layer: non-linear

Input

Input layer: no i/p

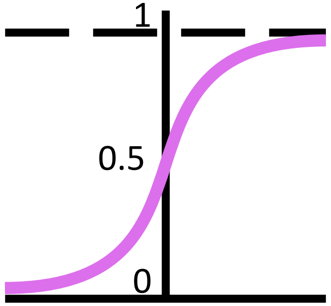
Hidden/output layer: i/p taken from other neurons

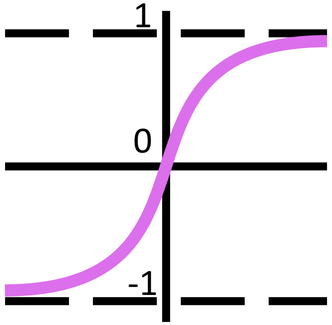
Some input items can be bias terms (constant) e.g. 1



$$y = f\left(\sum_{i=1}^{d} \mathbf{w}_i \cdot \mathbf{x}_i\right)$$

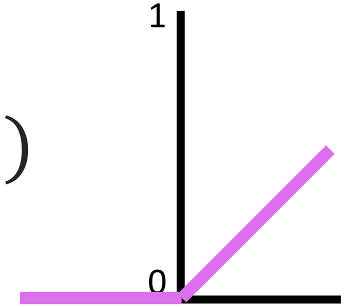
Common Activation Functions



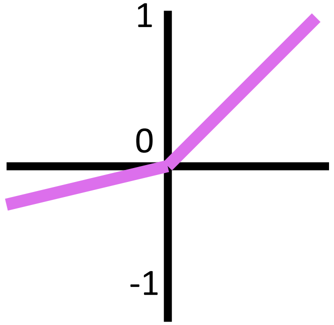


Sigmoid $\sigma(t) = \frac{\exp(t)}{\exp(t)+1}$

Tan Hyperbolic $\tanh(t) = \frac{\exp(2t)-1}{\exp(2t)+1} = 2\sigma(2t) - 1$



Rectified Linear Unit (ReLU) $r(t) = [t]_+ = \max(t, 0)$



Leaky ReLU $r(t) = \max(\beta \cdot t, t)\,, \beta \in [0,1]$

Others e.g. smoothed (exponential, softplus) ReLU, maxout

Sigmoid/tanh have a problem of vanishing gradients

Notice that the curves of these functions flatten out in both directions

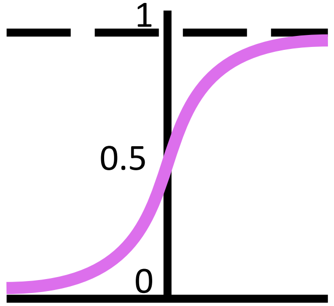
(Leaky) ReLU does not have this problem – very popular

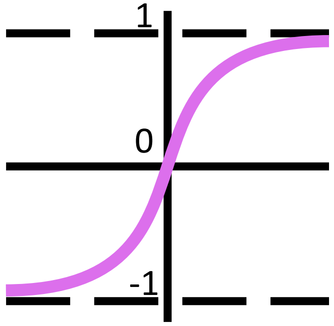
Also offer cheap gradient computations since they are piecewise linear

(Leaky) ReLU networks always learn piecewise linear functions

***Proof**: by induction on number of hidden layers (base case – no hidden layer)*

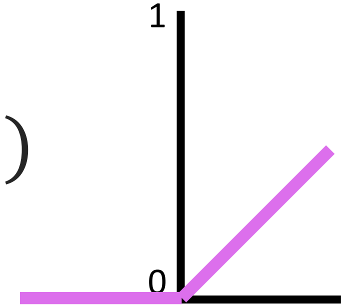
Common Activation Functions





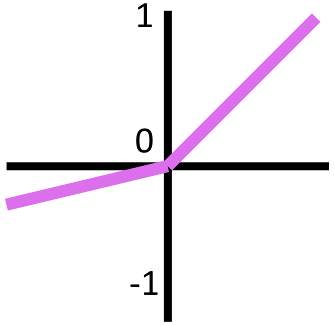
Sigmoid $\sigma(t) = \frac{\exp(t)}{\exp(t)+1}$

Tan Hyperbolic $\tanh(t) = \frac{\exp(2t)-1}{\exp(2t)+1} = 2\sigma(2t) - 1$



Rectified Linear Unit (ReLU) $r(t) = [t]_+ = \max(t, 0)$

Leaky ReLU $r(t) = \max(\beta \cdot t, t), \beta \in [0,1]$



Others e.g. smoothed (exponential, softplus) ReLU, maxout

Sigmoid/tanh have a problem of vanishing gradients

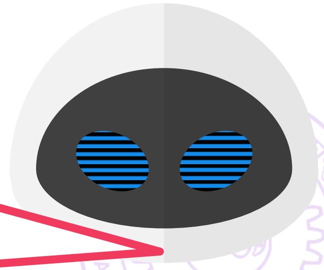
Notice that the curves of these functions flatten out in both directions

(Leaky) R

Also offe

(Leaky) R

Output layer neurons are usually given a task-specific activation e.g. identity for regression problems, sign for binary/multilabel classification problems, softmax for multiclassification problems



Proof*: by induction on number of hidden layers (base case – no hidden layer)*

Toy Example

Input sent to first hidden layer with two nodes

Each node first computes its pre-activation *value*

$$a_i = \left(\mathbf{w}^i\right)^\top \mathbf{x}, i = 1,2, \mathbf{w}^i, \mathbf{x} \in \mathbb{R}^d$$

Non-linear activation applied to each preactivation

$$h_i = f(a_i), i = 1,2$$

The vector $\mathbf{h} \in \mathbb{R}^2$ sent to next layer as input

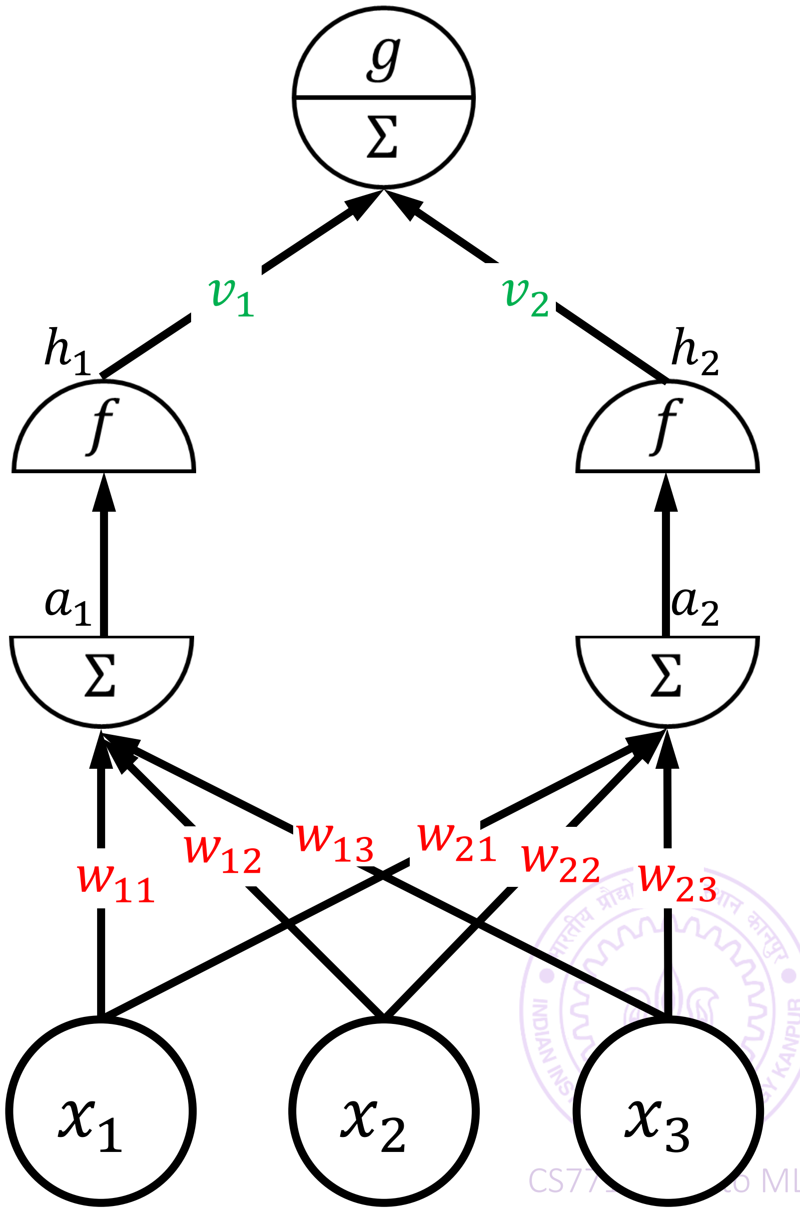
Process of preactivation, activation etc repeats

Shorthand: let $W = [\mathbf{w}^1, \mathbf{w}^2] \in \mathbb{R}^{d \times 2}$

Function computed by this network is

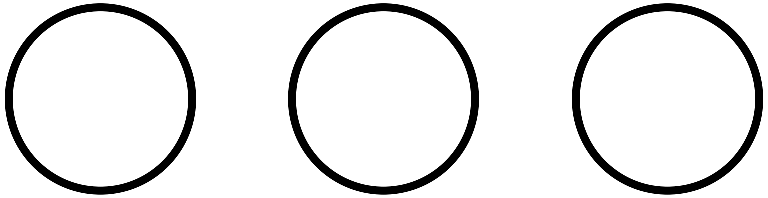
$$g\left(\mathbf{v}^\top f(W^\top \mathbf{x})\right)$$

This nesting of functions gives DN their power

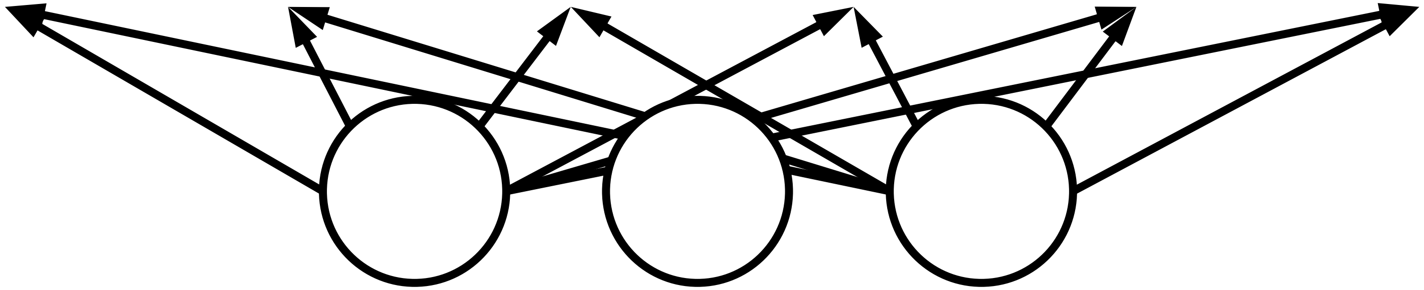


Feedforward Deep Networks

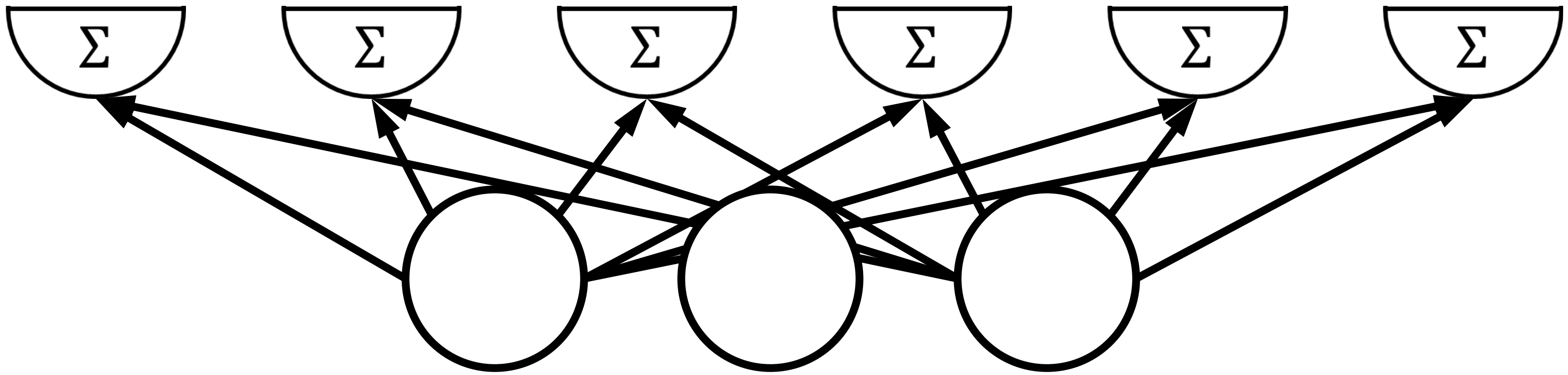
Feedforward Deep Networks



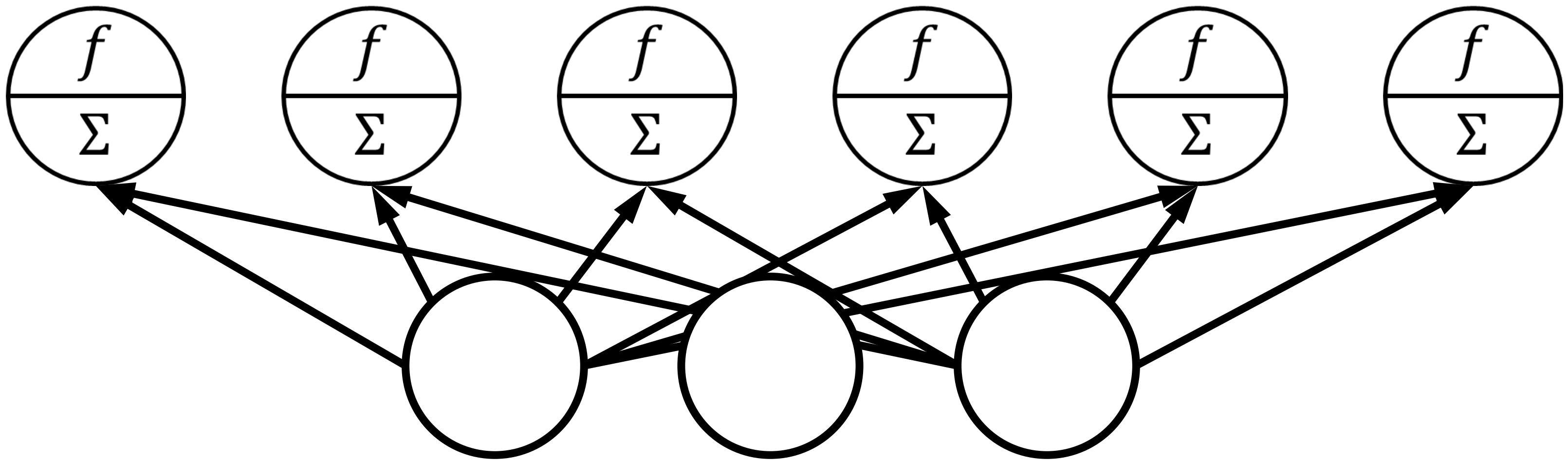
Feedforward Deep Networks



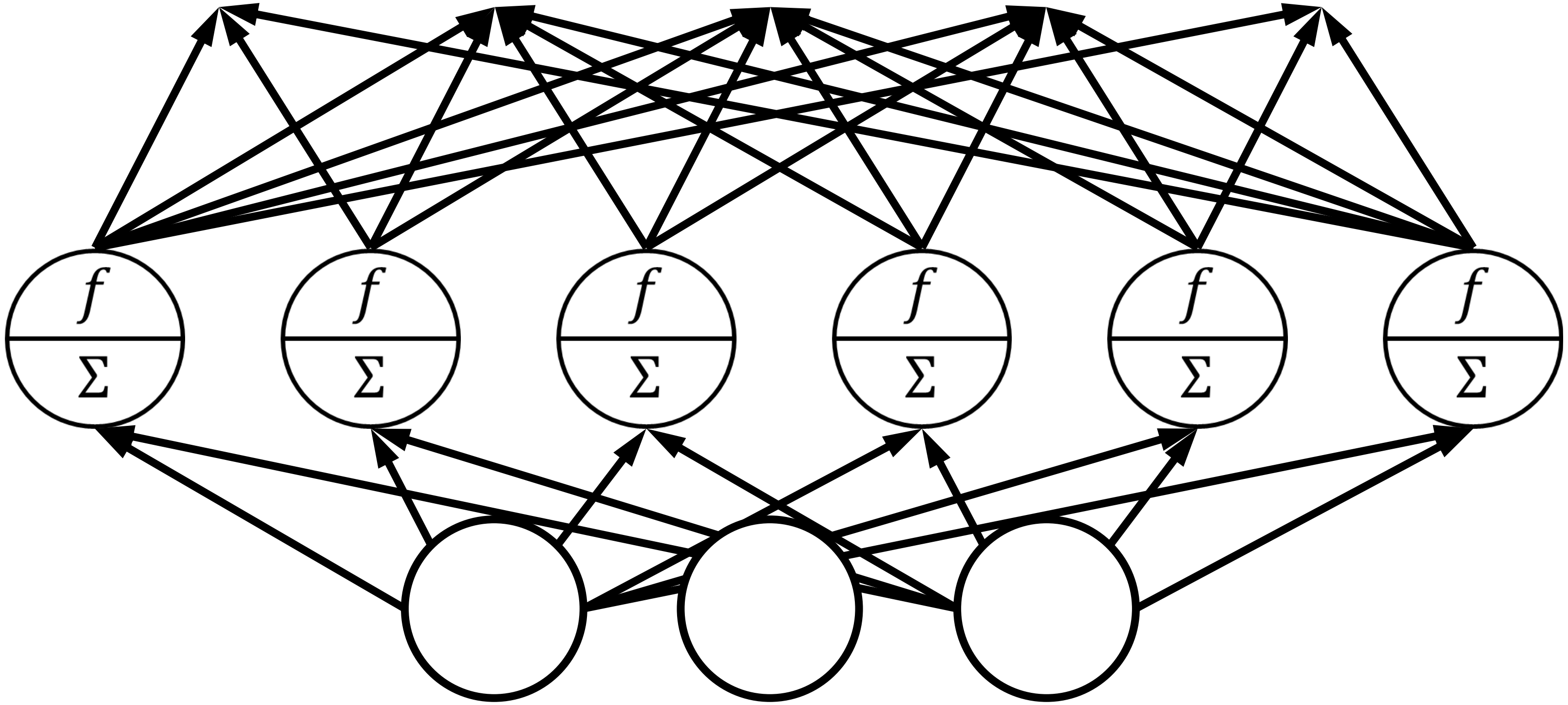
Feedforward Deep Networks



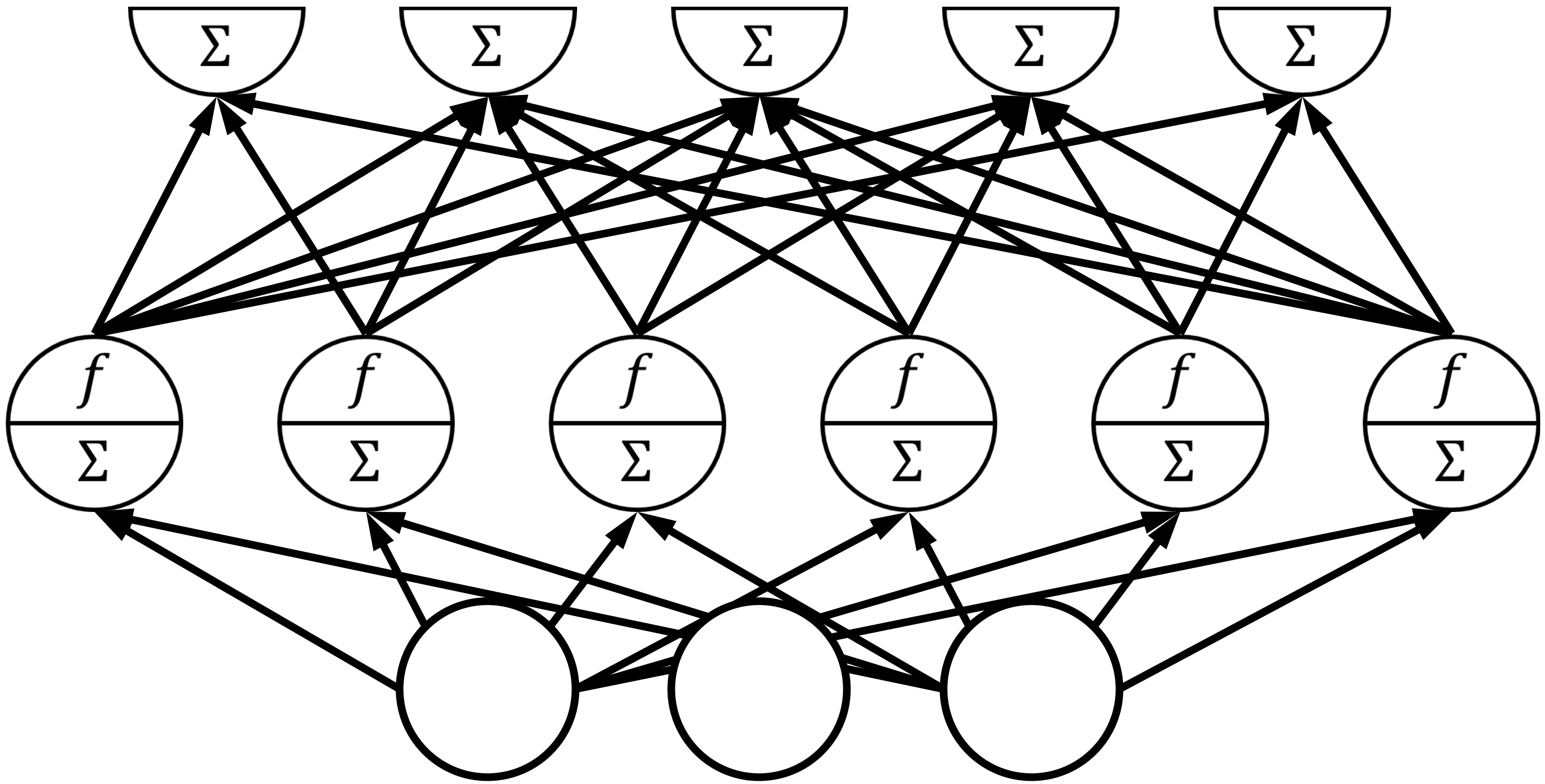
Feedforward Deep Networks



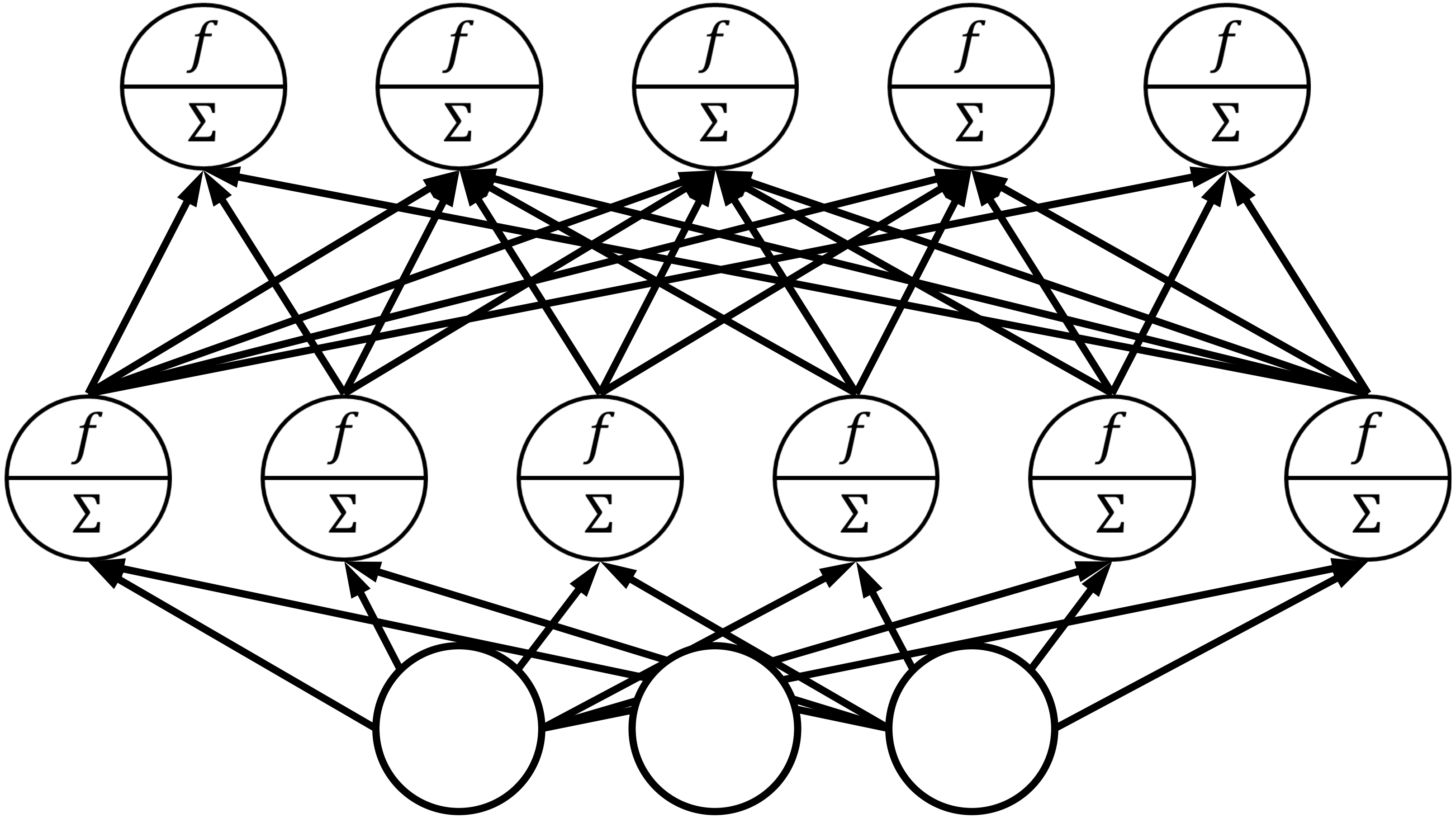
Feedforward Deep Networks



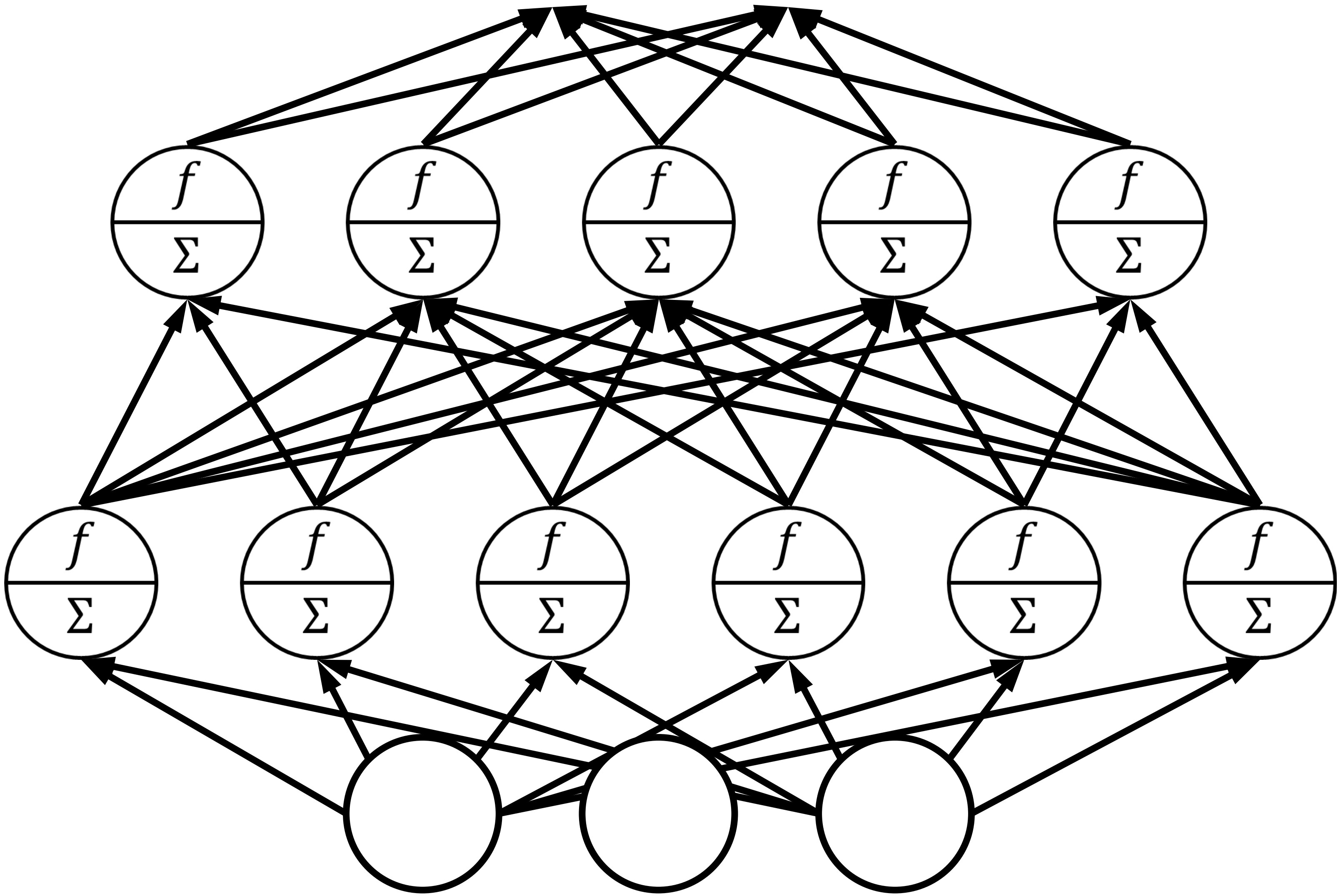
Feedforward Deep Networks



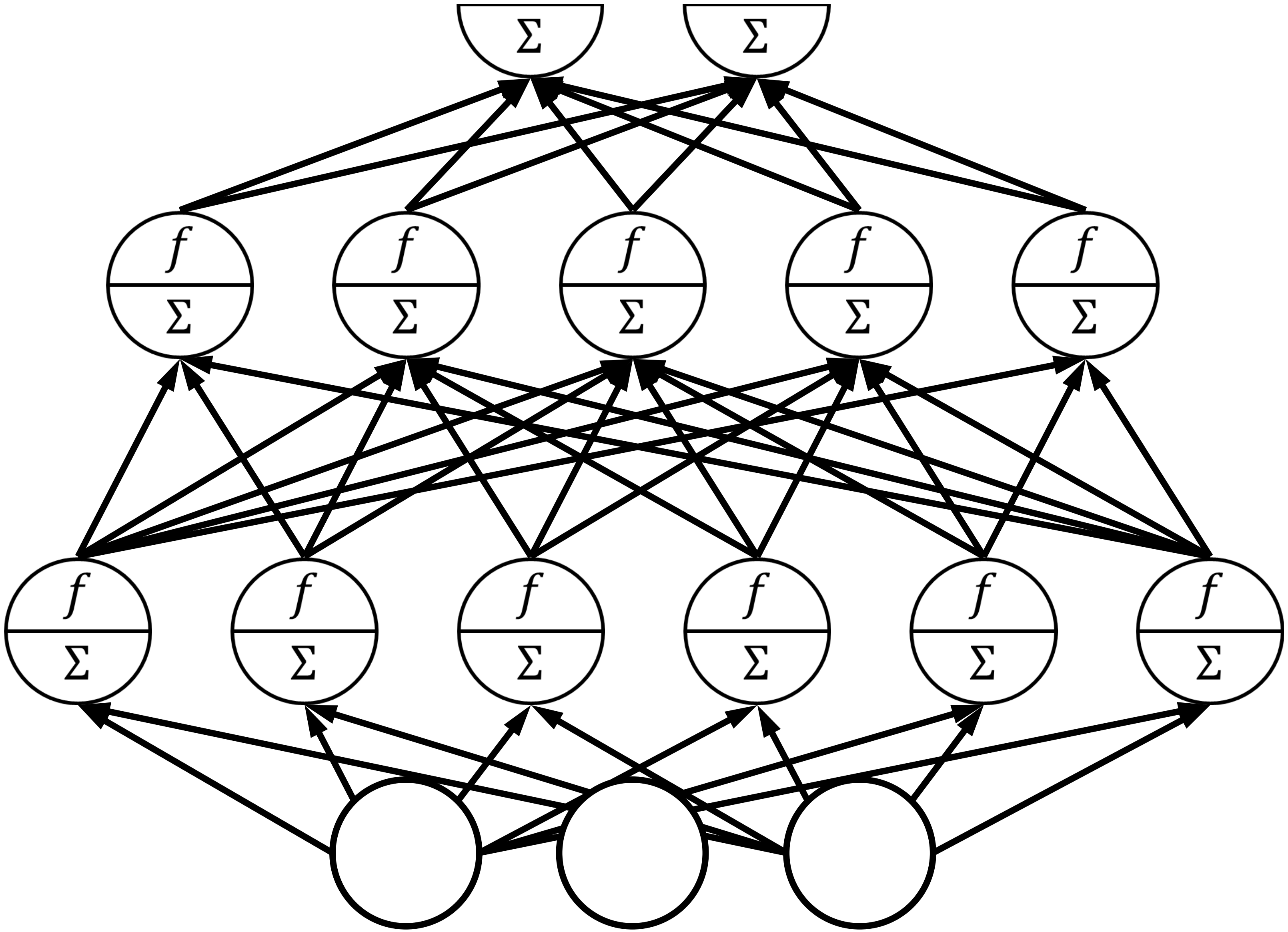
Feedforward Deep Networks



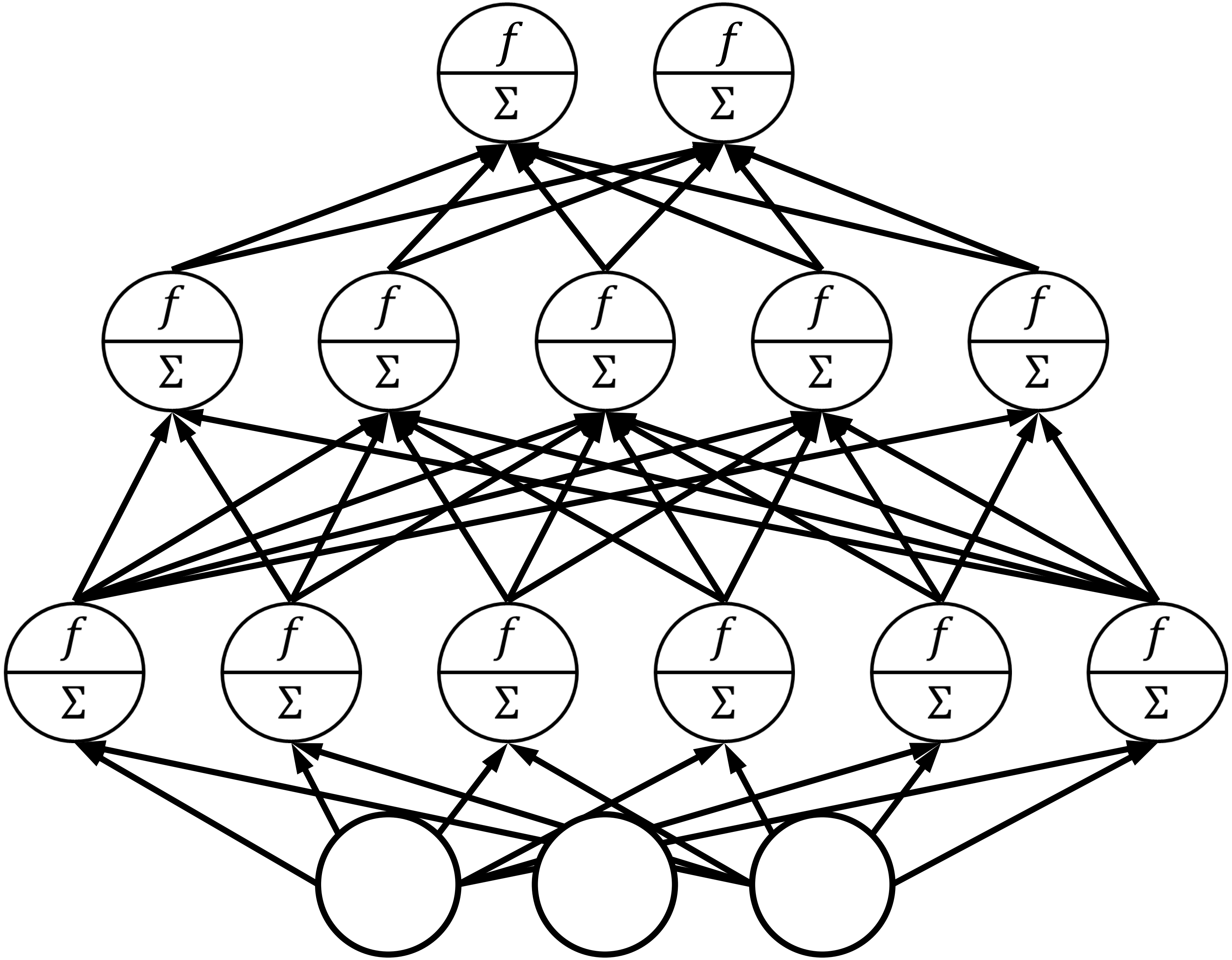
Feedforward Deep Networks



Feedforward Deep Networks

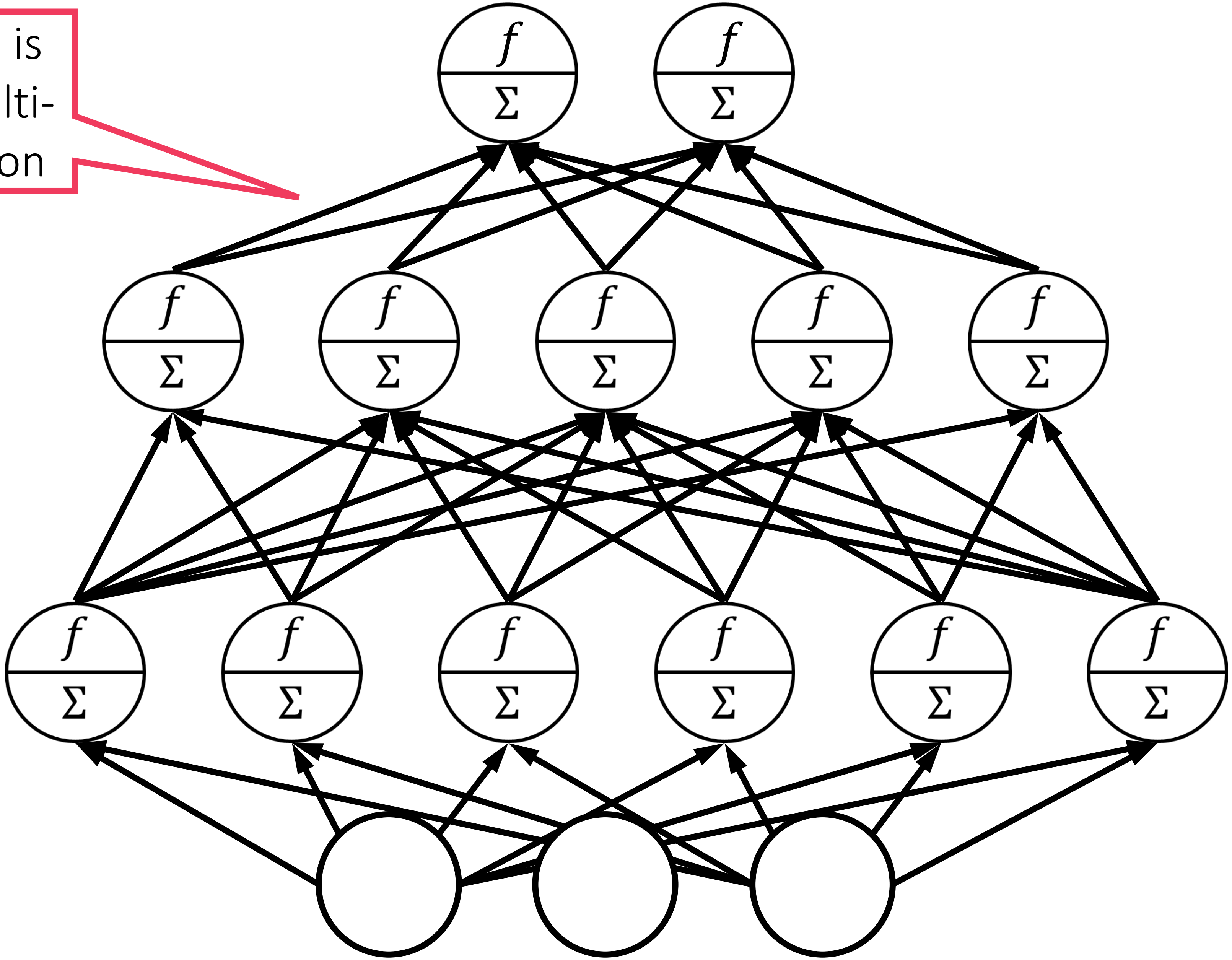


Feedforward Deep Networks



Feedforward Deep Networks

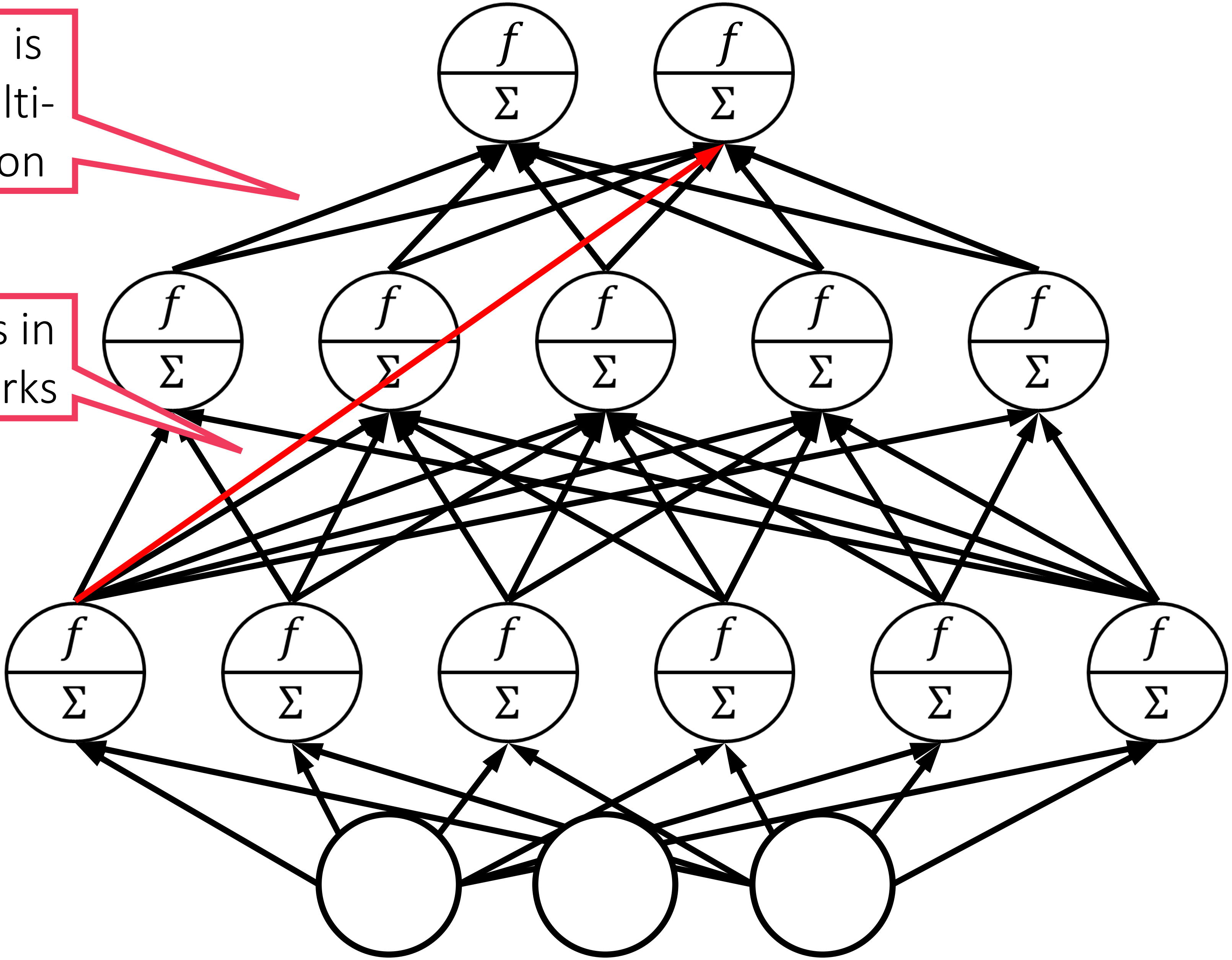
This architecture is often called a Multi-layered perceptron



Feedforward Deep Networks

This architecture is often called a Multi-layered perceptron

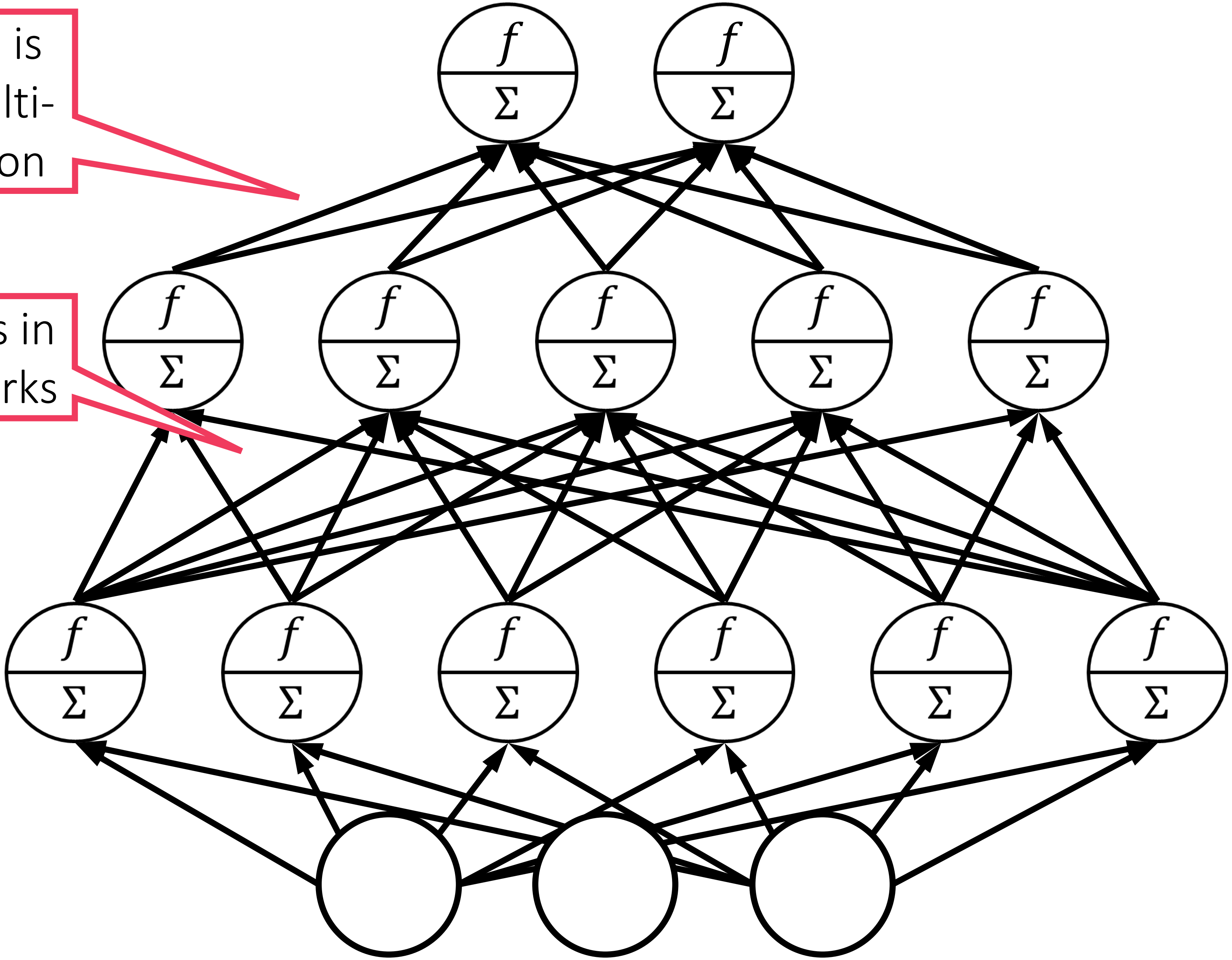
Cannot skip layers in classical FF networks



Feedforward Deep Networks

This architecture is often called a Multi-layered perceptron

Cannot skip layers in classical FF networks

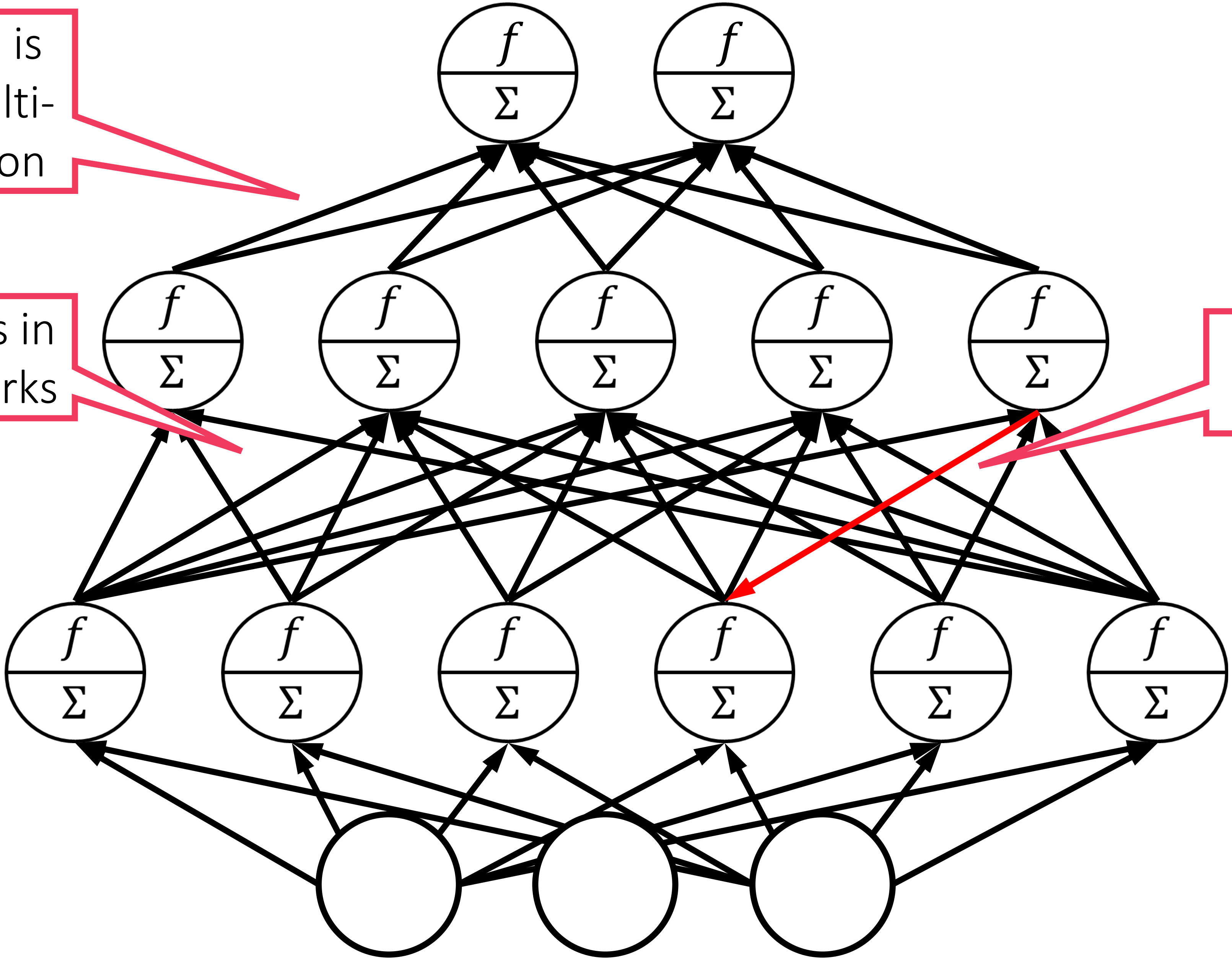


Feedforward Deep Networks

This architecture is often called a Multi-layered perceptron

Cannot skip layers in classical FF networks

No "reverse" links allowed

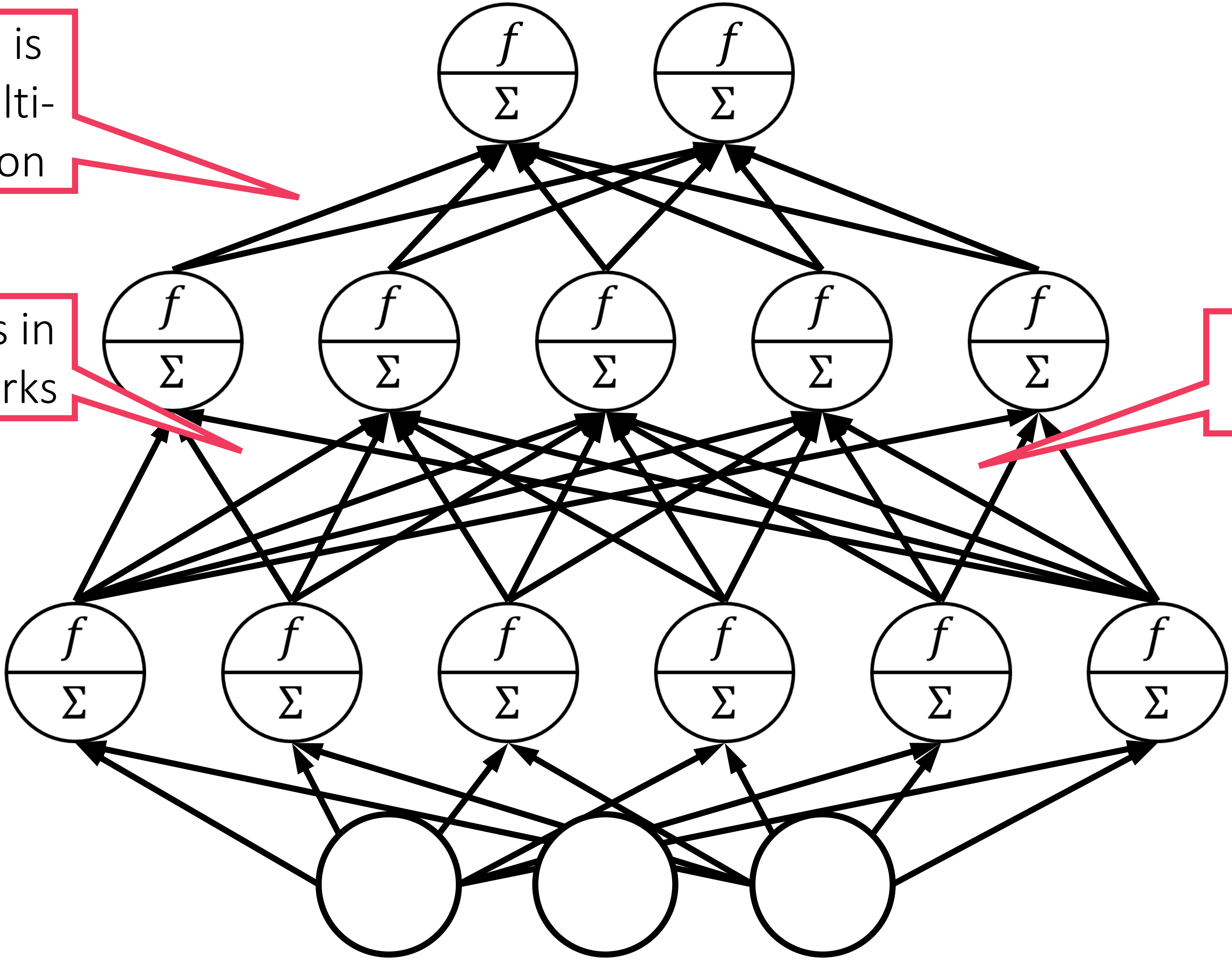


Feedforward Deep Networks

This architecture is often called a Multi-layered perceptron

Cannot skip layers in classical FF networks

No "reverse" links allowed



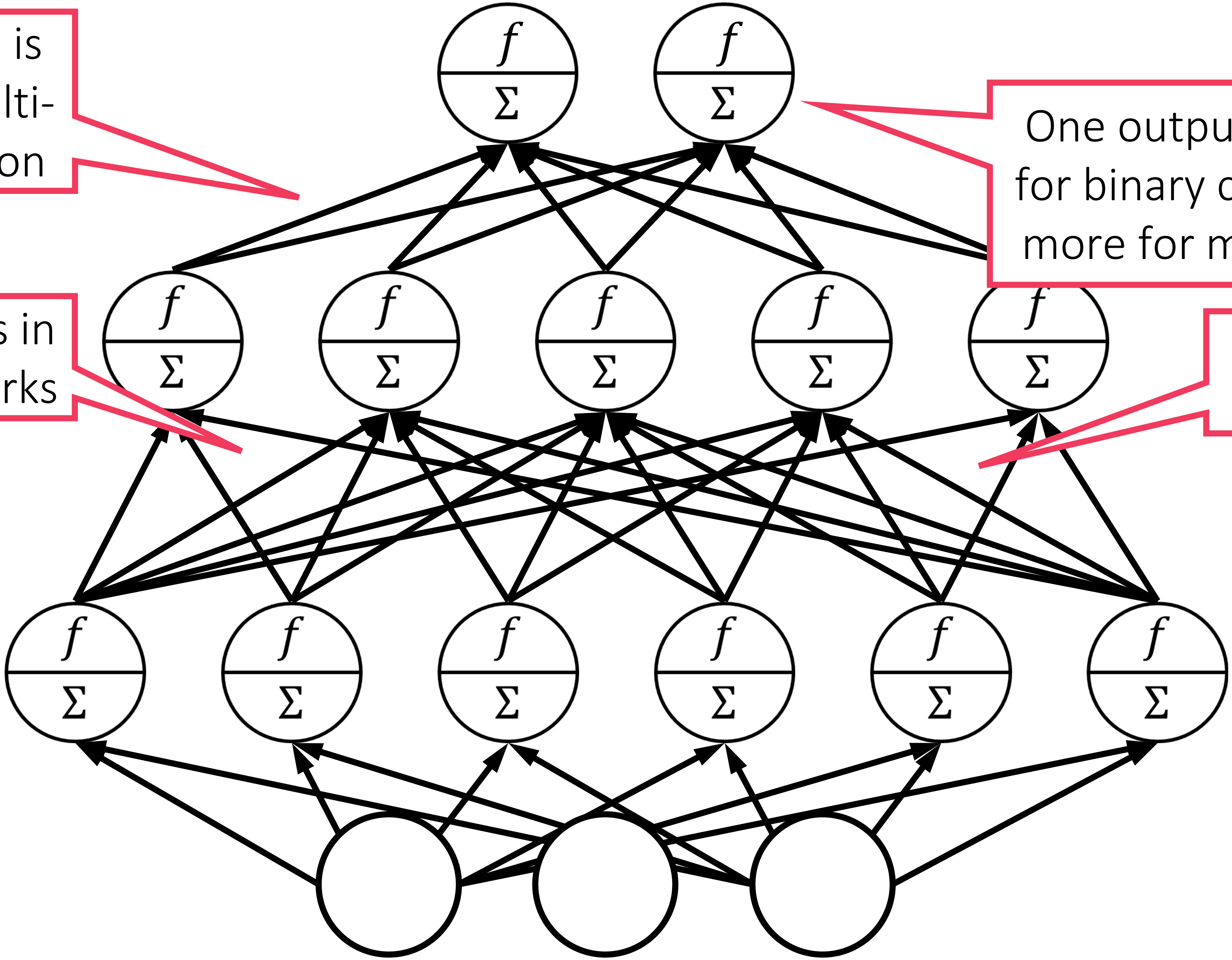
Feedforward Deep Networks

This architecture is often called a Multi-layered perceptron

One output node needed for binary classfn, regresn, more for multi-label/class

Cannot skip layers in classical FF networks

No "reverse" links allowed



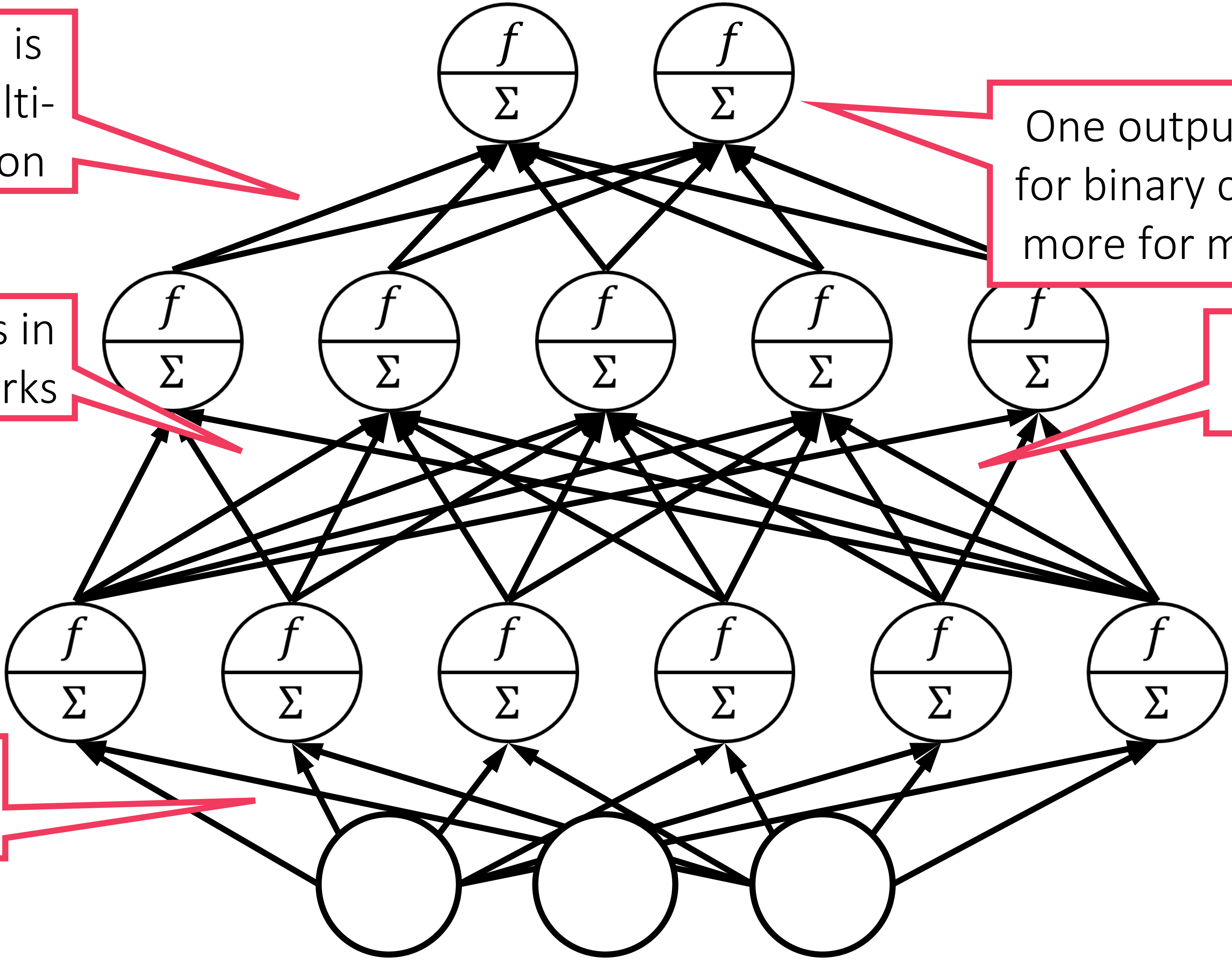
Feedforward Deep Networks

This architecture is often called a Multi-layered perceptron

One output node needed for binary classfn, regresn, more for multi-label/class

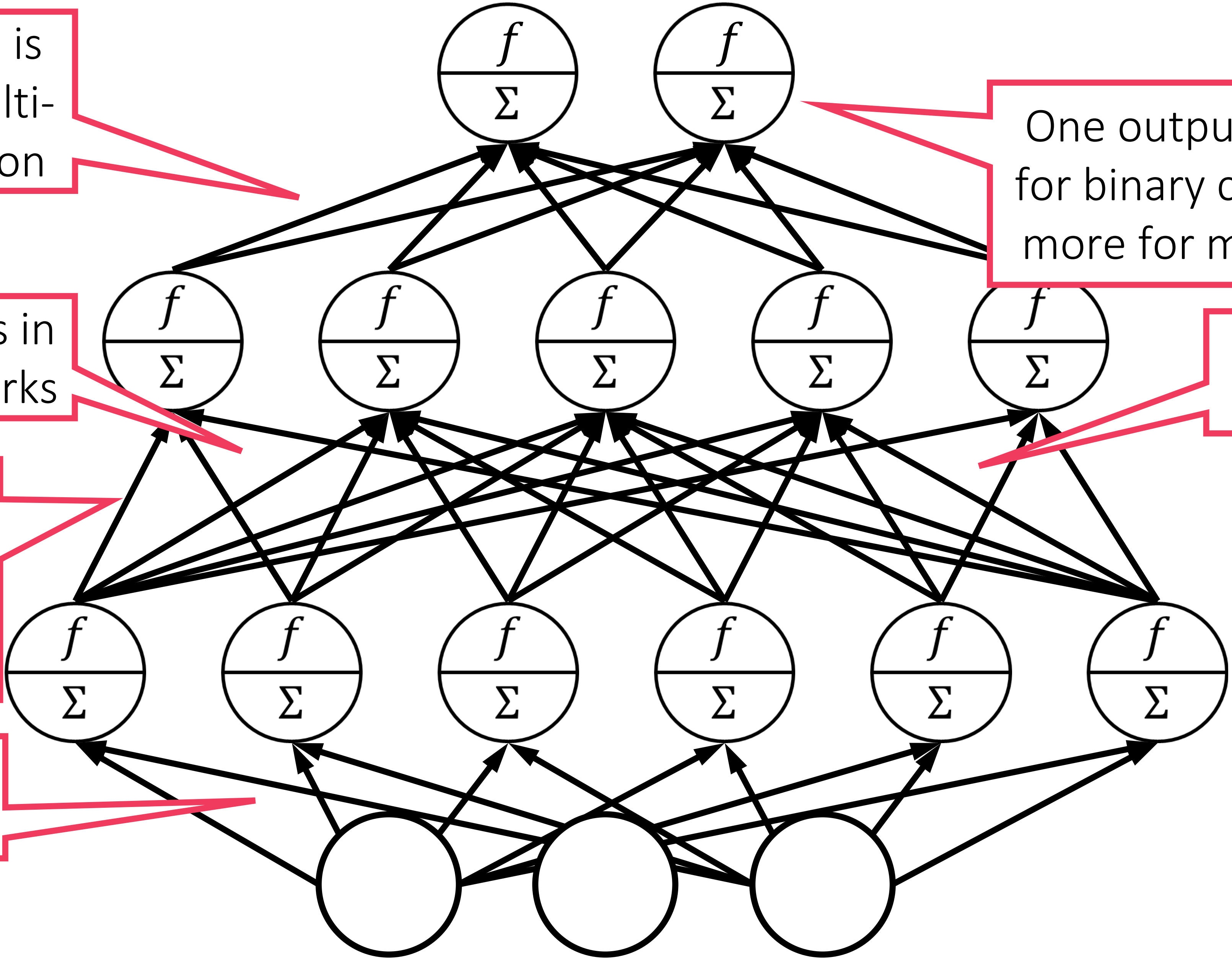
Cannot skip layers in classical FF networks

No “reverse” links allowed



All weights are learnt

Feedforward Deep Networks



This architecture is often called a Multi-layered perceptron

One output node needed for binary classfn, regresn, more for multi-label/class

Cannot skip layers in classical FF networks

No "reverse" links allowed

Connections b/w layers usually "dense" – all pairs connected

All weights are learnt

Feedforwa

Some networks do allow "skip", "feedback" connections, others allow activation fn f to change from layer to layer

This architecture is often called a Multi-layered perceptron

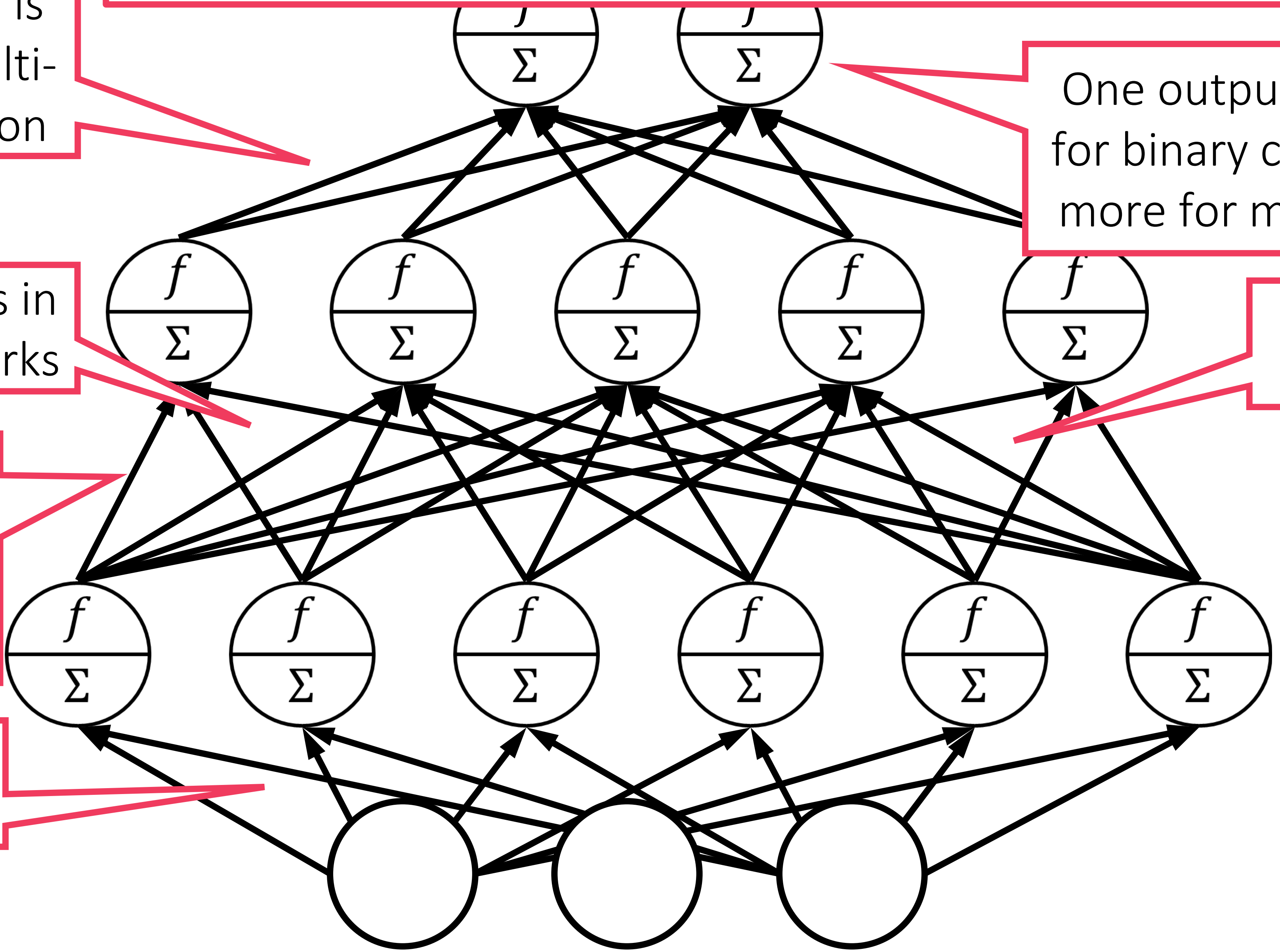
One output node needed for binary classfn, regresn, more for multi-label/class

Cannot skip layers in classical FF networks

No "reverse" links allowed

Connections b/w layers usually "dense" – all pairs connected

All weights are learnt



Feedforwa

Some networks do allow “skip”, “feedback” connections, others allow activation fn f to change from layer to layer

This architecture is often called a Multi-layered perceptron

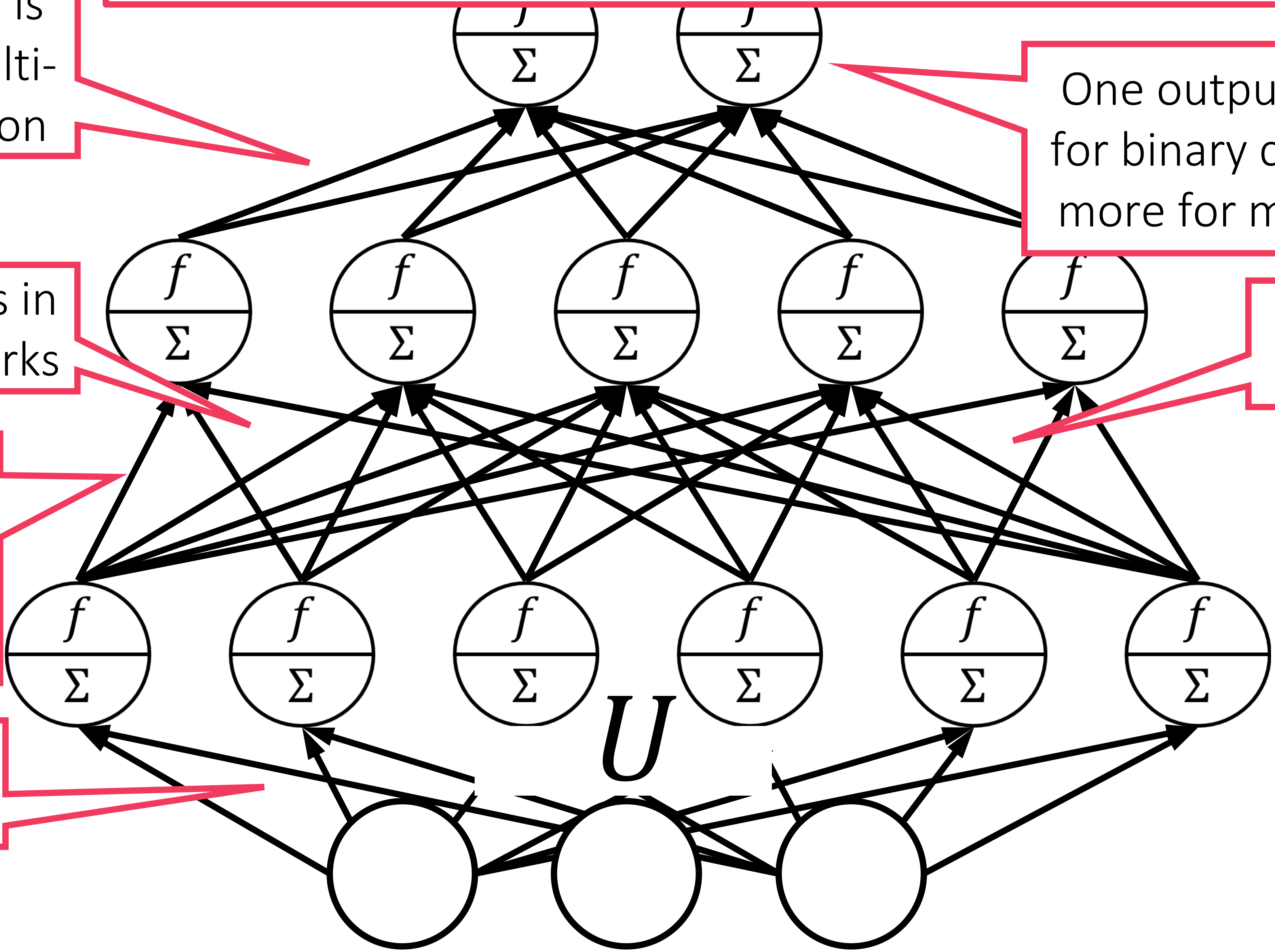
One output node needed for binary classfn, regresn, more for multi-label/class

Cannot skip layers in classical FF networks

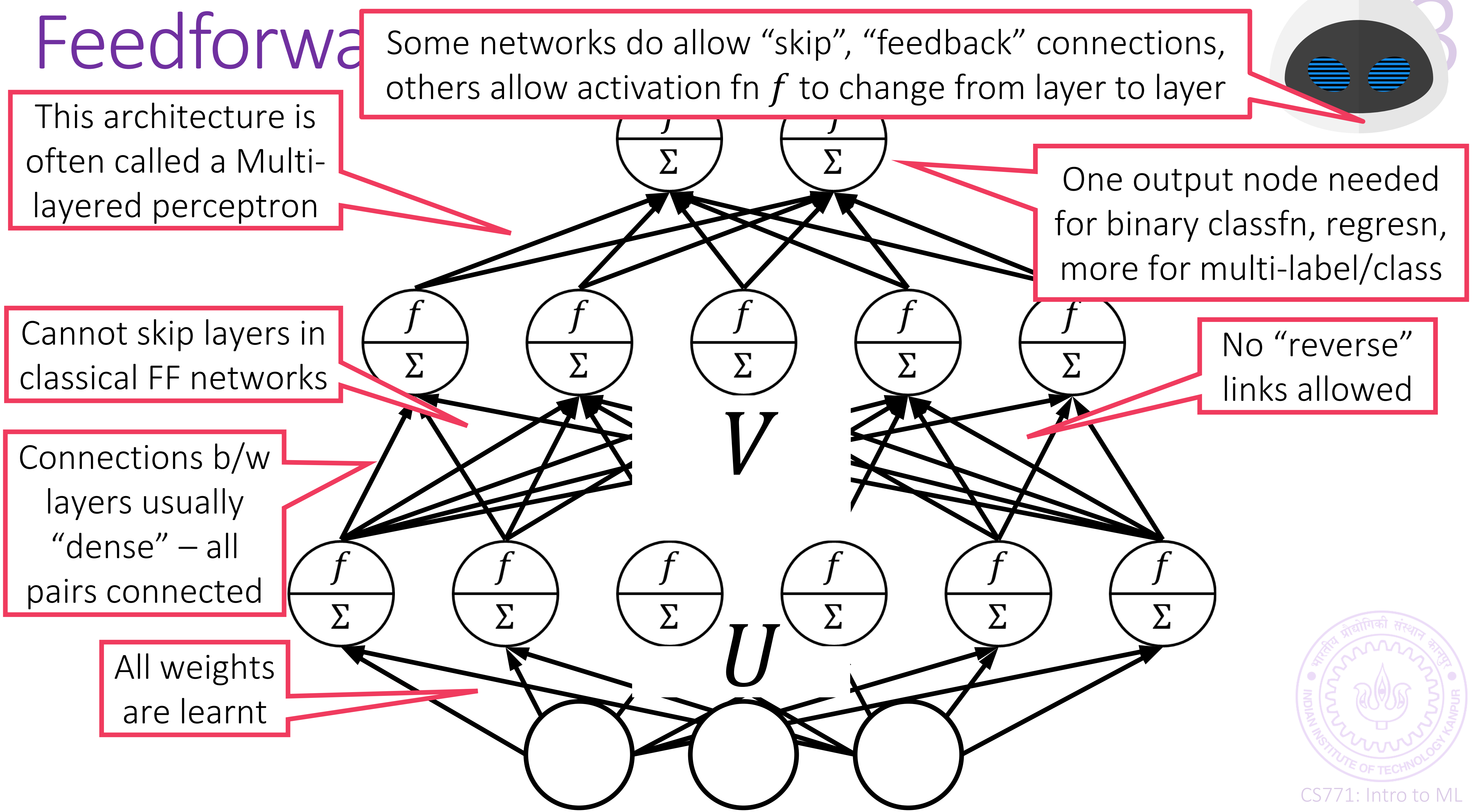
No “reverse” links allowed

Connections b/w layers usually “dense” – all pairs connected

All weights are learnt



Feedforwa[rd]



Feedforwa

Some networks do allow “skip”, “feedback” connections, others allow activation fn f to change from layer to layer

This architecture is often called a Multi-layered perceptron

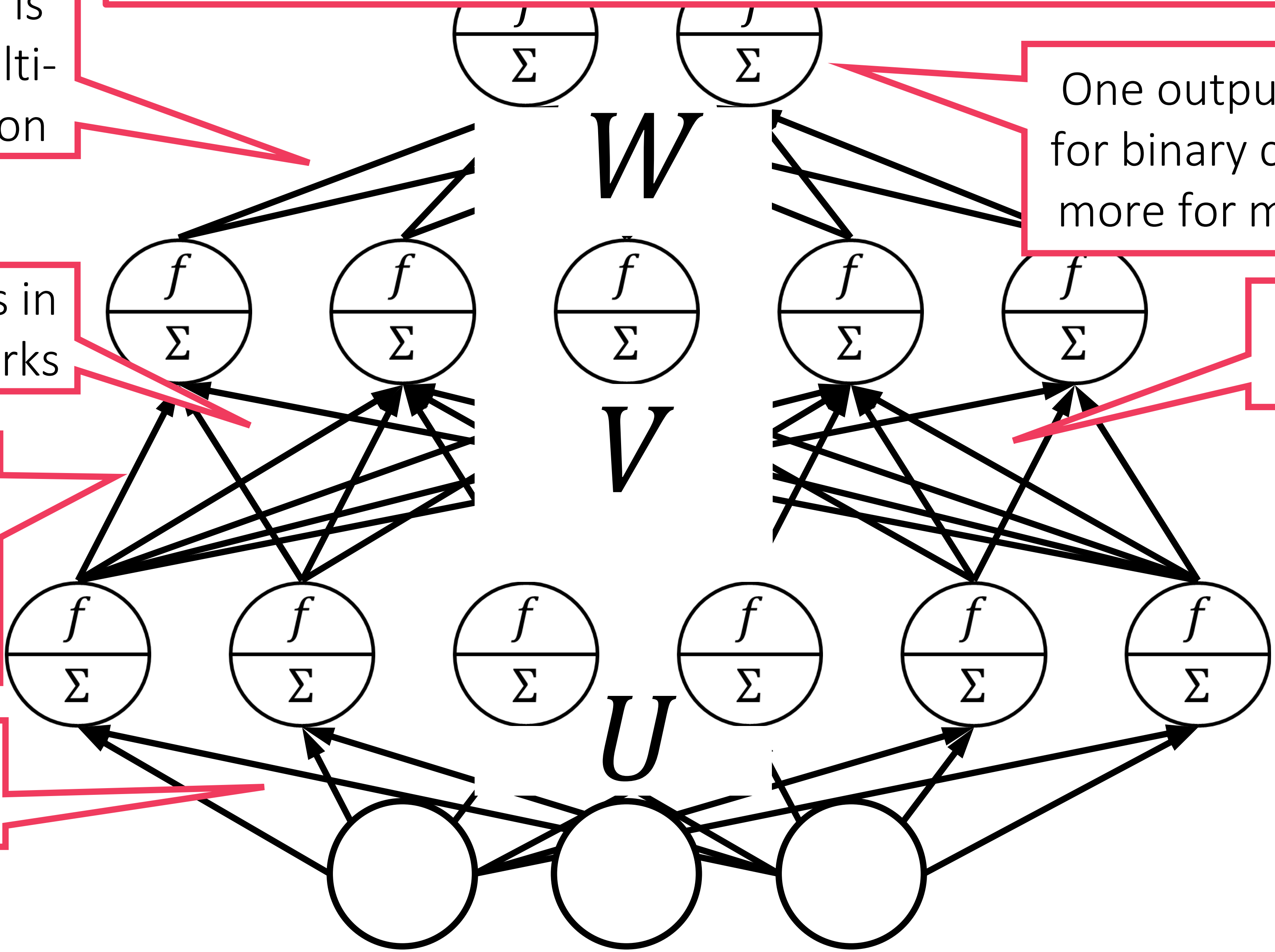
One output node needed for binary classfn, regresn, more for multi-label/class

Cannot skip layers in classical FF networks

No “reverse” links allowed

Connections b/w layers usually “dense” – all pairs connected

All weights are learnt



Feedforwa

Some networks do allow "skip", "feedback" connections, others allow activation fn f to change from layer to layer

This architecture is often called a Multi-layered perceptron

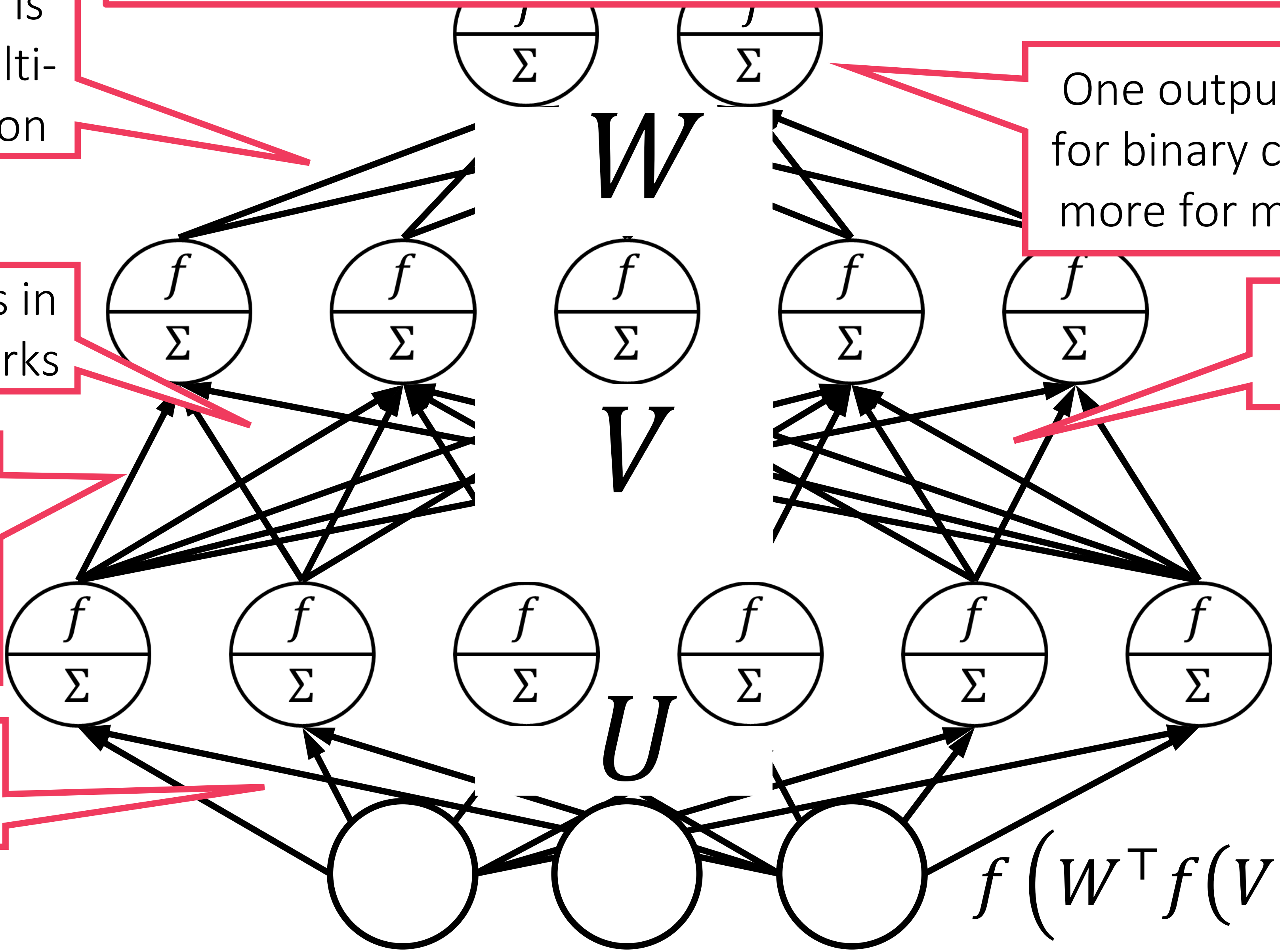
One output node needed for binary classfn, regresn, more for multi-label/class

Cannot skip layers in classical FF networks

No "reverse" links allowed

Connections b/w layers usually "dense" – all pairs connected

All weights are learnt



$$f\left(W^\top f(V^\top f(U^\top \mathbf{x}))\right)$$

Feedforwa

Some networks do allow “skip”, “feedback” connections, others allow activation fn f to change from layer to layer

This architecture is often called a Multi-layered perceptron

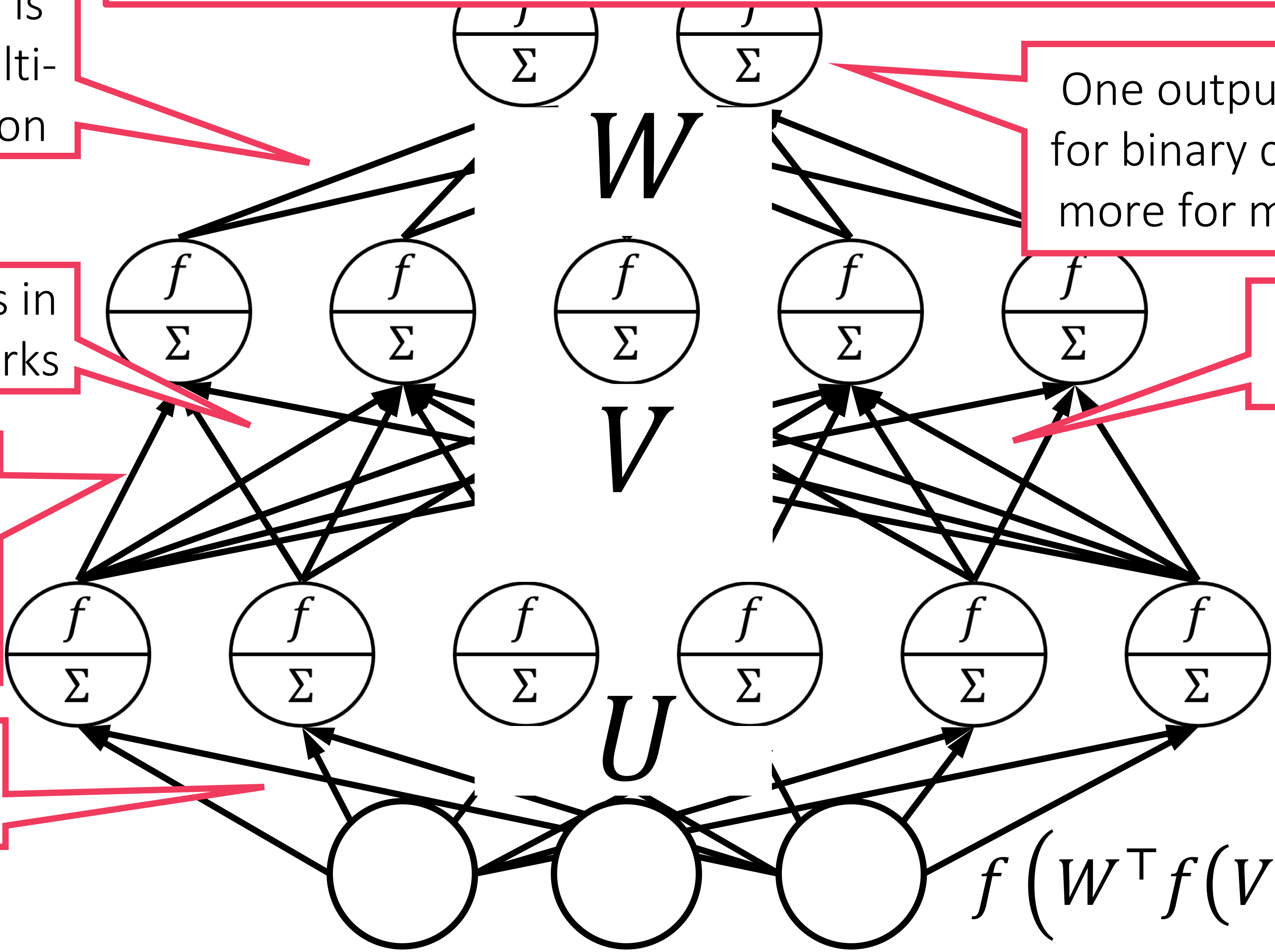
One output node needed for binary classfn, regresn, more for multi-label/class

Cannot skip layers in classical FF networks

No “reverse” links allowed

Connections b/w layers usually “dense” – all pairs connected

All weights are learnt



$U \in \mathbb{R}^{3\times 6}$

$V \in \mathbb{R}^{6\times 5}$

$W \in \mathbb{R}^{5\times 2}$

$$f\left(W^\top f(V^\top f(U^\top \mathbf{x}))\right)$$

Deep Networks

Deep Networks

The layered architecture is what makes deep networks "deep"

Deep Networks

The layered architecture is what makes deep networks "deep"

Lower layers can be interpreted as computing useful features

Deep Networks

The layered architecture is what makes deep networks "deep"

- *Lower layers can be interpreted as computing useful features*
  - Networks that learn polynomial etc features exist too – *Sum Product Networks (SPN)*

Deep Networks

The layered architecture is what makes deep networks "deep"

Lower layers can be interpreted as computing useful features

Networks that learn polynomial etc features exist too – *Sum Product Networks (SPN)*

Last layer exploits all this hard work to learn a good linear model over them – can have any no. of layers, any no. of nodes in each layer

Deep Networks

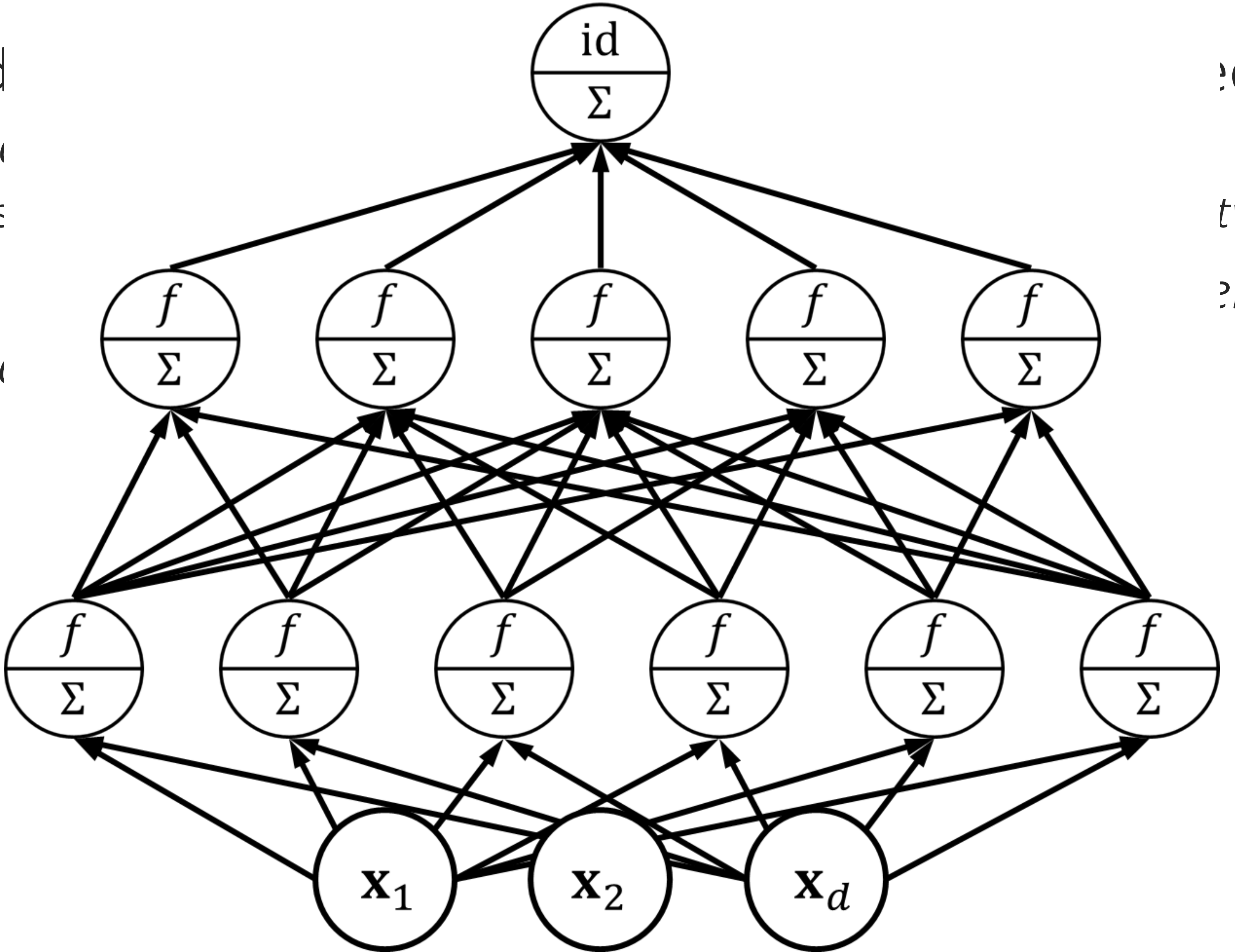
The layered ... eep"

Lower lay...

Networks ... tworks (SPN)

Last layer ... el over them –

can have ...



Deep Networks

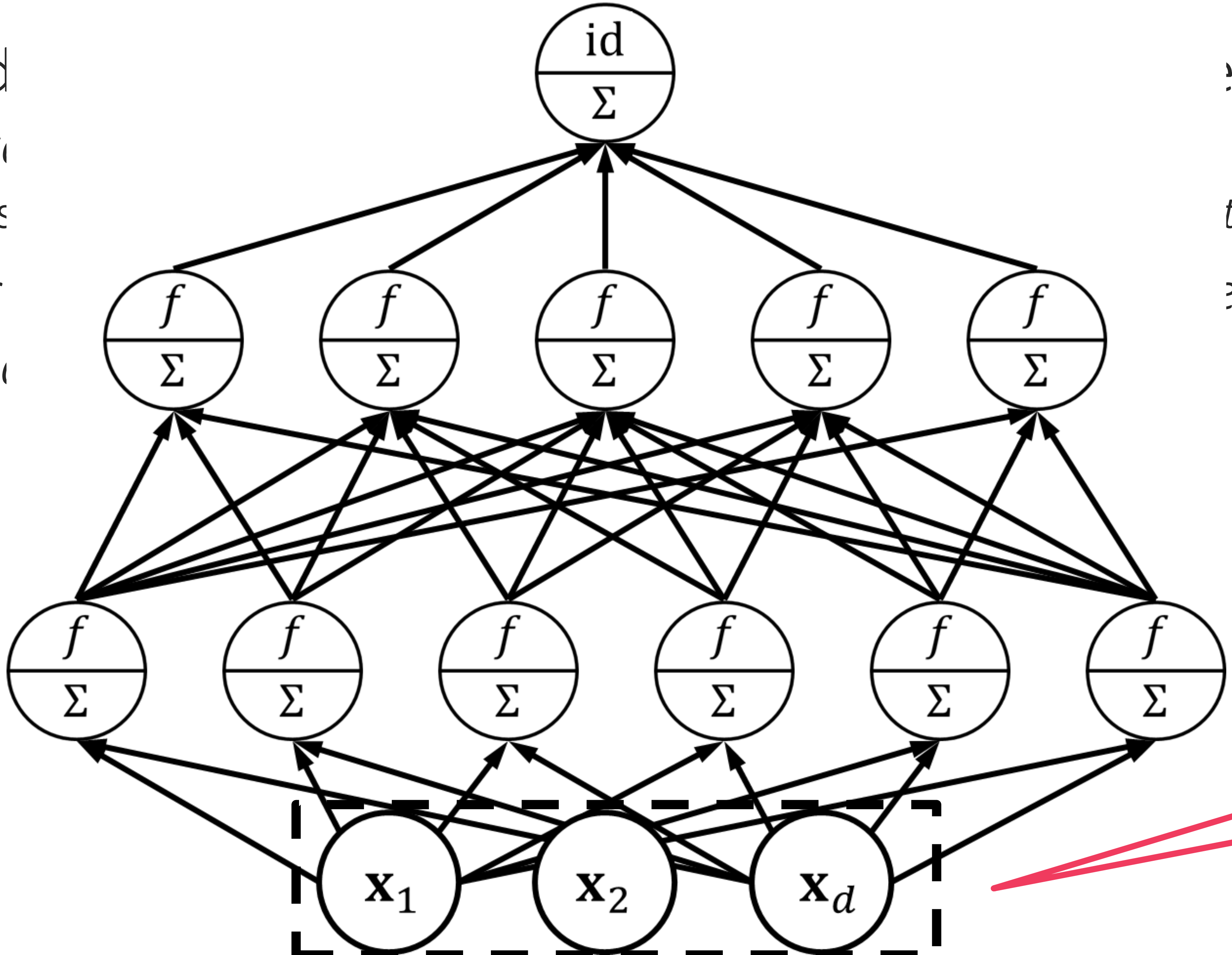
The layered ... ep”

Lower lay...

Networks ... *tworks (SPN)*

Last layer ... el over them –

can have ...



Deep Networks

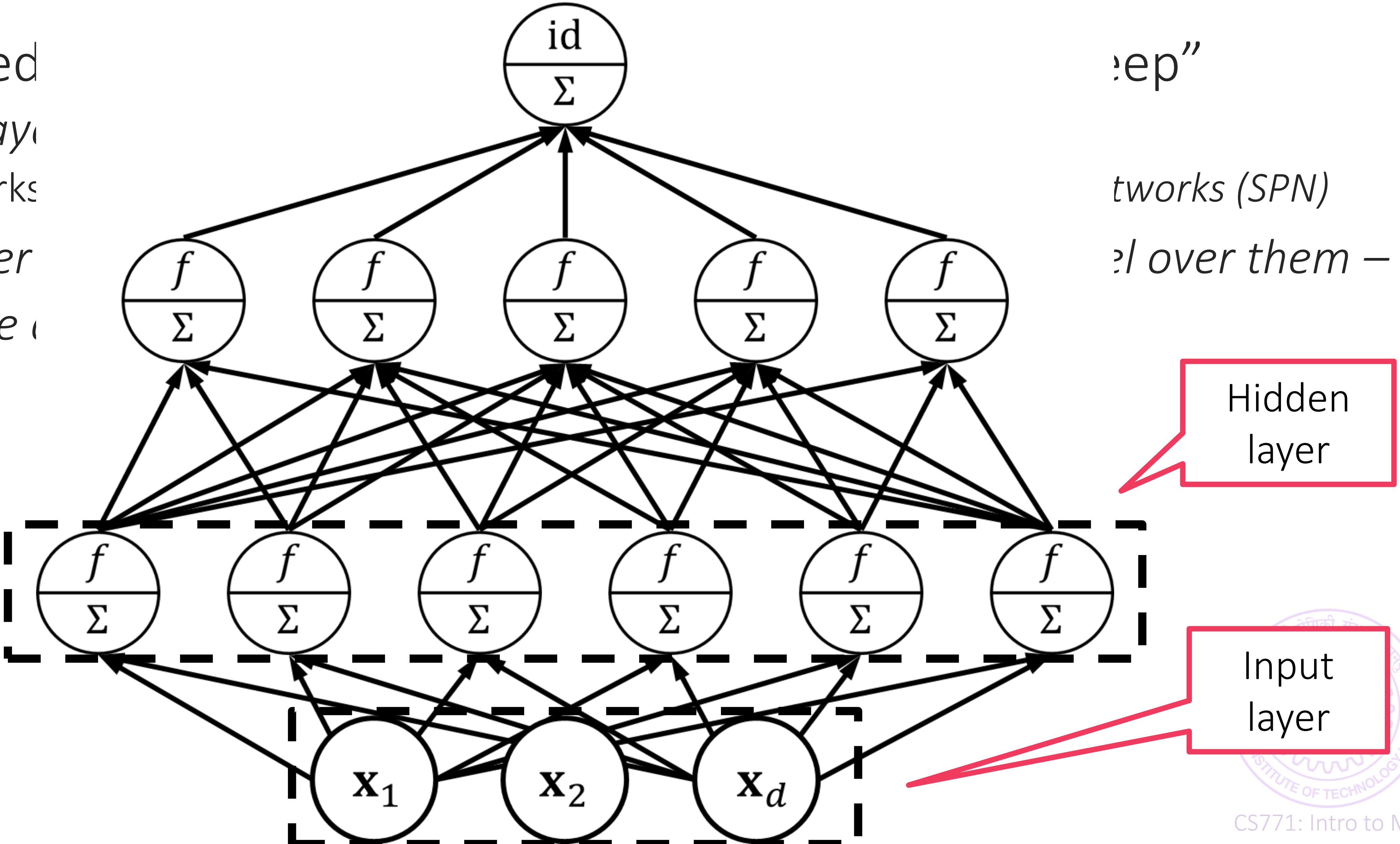
The layered … ep"

Lower lay…

Networks … *tworks (SPN)*

Last layer … l over them –

can have …



Deep Networks

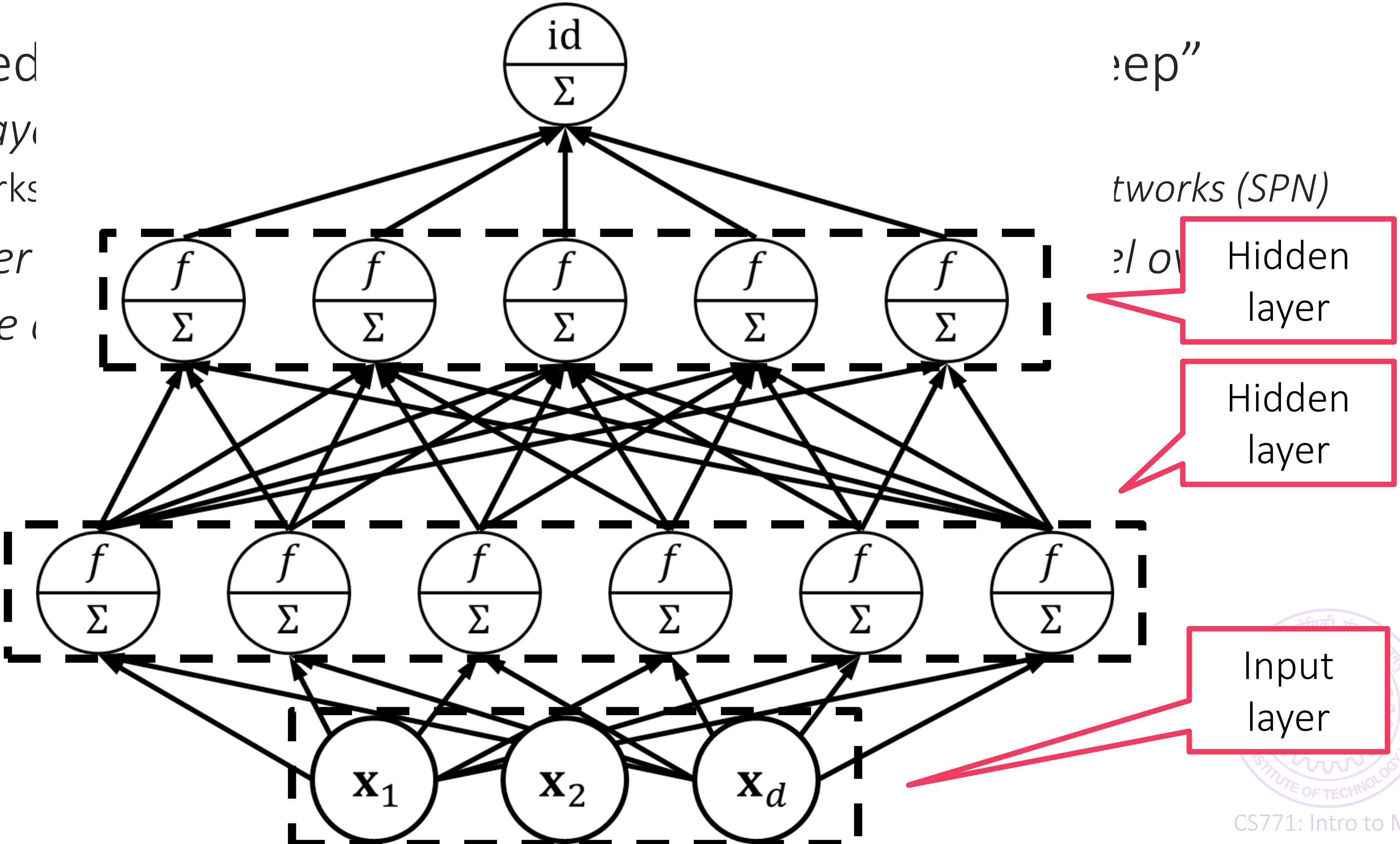
The layered ... ep"

Lower lay ...

Networks ... *tworks (SPN)*

Last layer ... *el o*

can have ...



Deep Networks

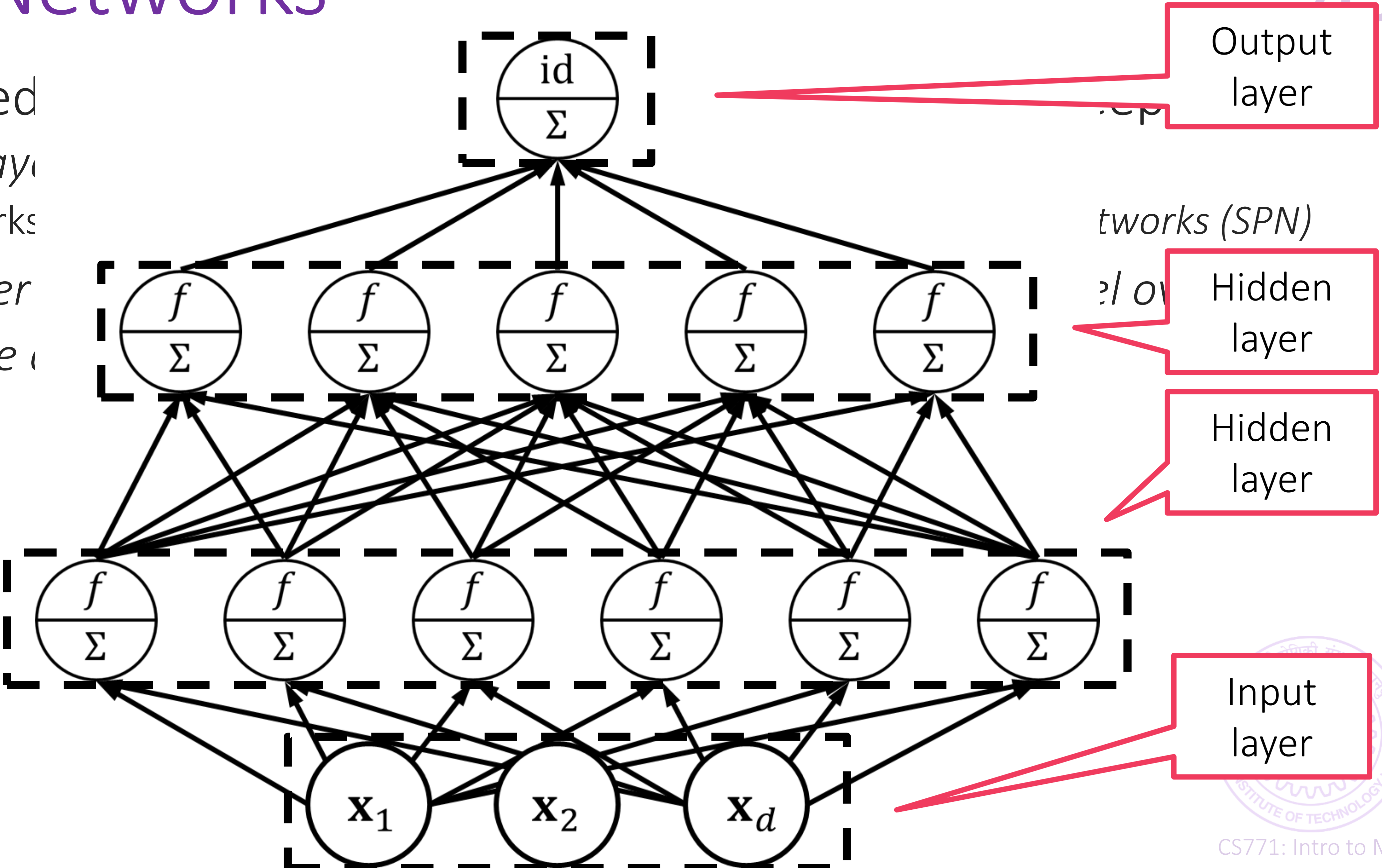
The layered

Lower lay

Networks

Last layer

can have



tworks (SPN)

el ov

Deep Networks

The layered

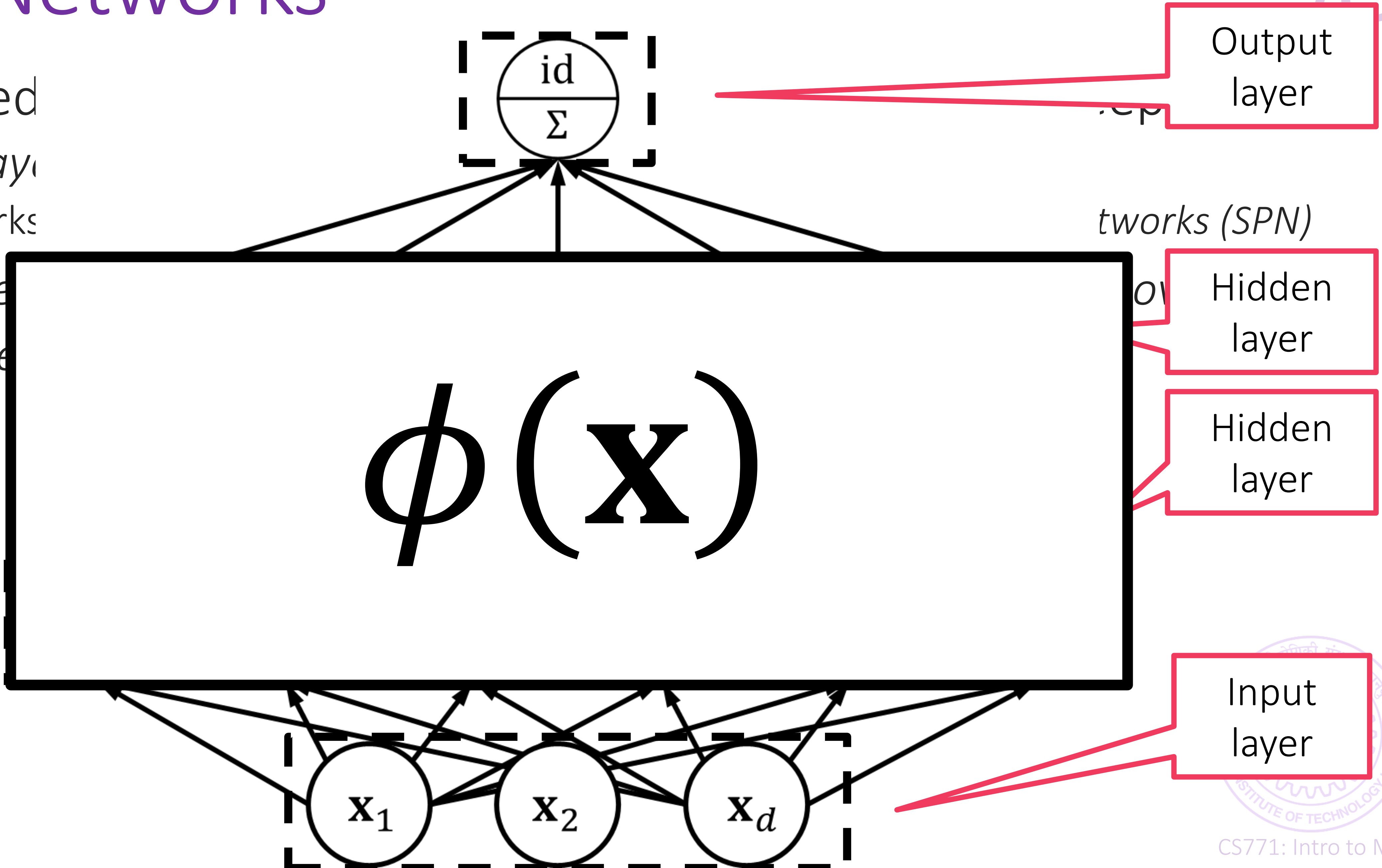
Lower lay

Networks

tworks (SPN)

Last laye

can have



Deep Networks

The layered architecture is what makes deep networks "deep"

Lower layers can be interpreted as computing useful features

Networks that learn polynomial etc features exist too – *Sum Product Networks (SPN)*

Last layer exploits all this hard work to learn a good linear model over them – can have any no. of layers, any no. of nodes in each layer

Deep Networks

The layered architecture is what makes deep networks "deep"

- *Lower layers can be interpreted as computing useful features*
  - Networks that learn polynomial etc features exist too – *Sum Product Networks (SPN)*
- *Last layer exploits all this hard work to learn a good linear model over them – can have any no. of layers, any no. of nodes in each layer*
- *Kernels throw a large (often infinite) number of features at a problem, deep networks instead try to learn a small number of problem specific features*

Deep Networks

The layered architecture is what makes deep networks “deep”

- *Lower layers can be interpreted as computing useful features*
  - Networks that learn polynomial etc features exist too – *Sum Product Networks (SPN)*
- *Last layer exploits all this hard work to learn a good linear model over them – can have any no. of layers, any no. of nodes in each layer*
- *Kernels throw a large (often infinite) number of features at a problem, deep networks instead try to learn a small number of problem specific features*
  - Easier said that done though ☺

Deep Networks

The layered architecture is what makes deep networks “deep”

- *Lower layers can be interpreted as computing useful features*
  - Networks that learn polynomial etc features exist too – *Sum Product Networks (SPN)*
- *Last layer exploits all this hard work to learn a good linear model over them – can have any no. of layers, any no. of nodes in each layer*
- *Kernels throw a large (often infinite) number of features at a problem, deep networks instead try to learn a small number of problem specific features*
  - Easier said that done though ☺

A network having a linear activation function in hidden layers is useless since such a network can only learn a linear function in input

Deep Networks

The layered architecture is what makes deep networks "deep"

Lower layers can be interpreted as computing useful features

Networks that learn polynomial etc features exist too – *Sum Product Networks (SPN)*

Last layer exploits all this hard work to learn a good linear model over them – can have any no. of layers, any no. of nodes in each layer

Kernels throw a large (often infinite) number of features at a problem, deep networks instead try to learn a small number of problem specific features

Easier said that done though ☺

A network having a linear activation function in hidden layers is useless since such a network can only learn a linear function in input

Can collapse, replace all hidden layers with a single connection from i/p to o/p

Deep Networks

The layered architecture is what makes deep networks "deep"

Lower layers can be interpreted as computing useful features

Networks that learn polynomial etc features exist too – *Sum Product Networks (SPN)*

Last layer exploits all this hard work to learn a good linear model over them – can have any no. of layers, any no. of nodes in each layer

Kernels throw a large (often infinite) number of features at a problem, deep networks instead try to learn a small number of problem specific features

Easier said that done though ☺

A network having a linear activation function in hidden layers is useless since such a network can only learn a linear function in input

Can collapse, replace all hidden layers with a single connection from i/p to o/p

Even (leaky) ReLU are piecewise linear, not linear – non-linearity is essential