Non-linear Models-IV

CS771: Introduction to Machine Learning

Purushottam Kar

Outline of discussion

- Kernel Approximation Methods
- PML with kernels
- Neural networks

Kernel methods can be slow ☹

- Need to work with indirect “dual” representations
- Although finite, these representations blow up with data size
- Prediction requires a full pass over data i.e. $O(dn)$ time
- Will see some techniques to remedy this

The Tale of a Trio of Techniques

- **Post-processing techniques**: learn the kernel SVM (a bit costly), but then make the model cheaper to store and predict

- **Approximate training techniques**: directly learn a kernel SVM model that is cheap to store and predict

- **Kernel approximation techniques**: use a different kernel than the one you wanted to, so that the new kernel mimics the original one but always gives models that are cheap to store and predict

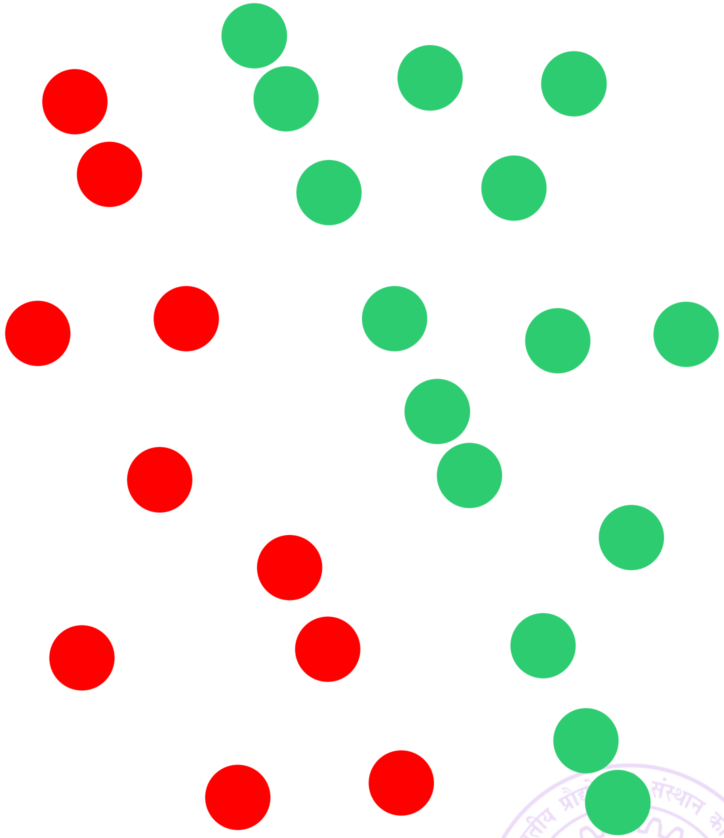
- Kernel approximation is the most successful of the three

Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \dots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \dots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.

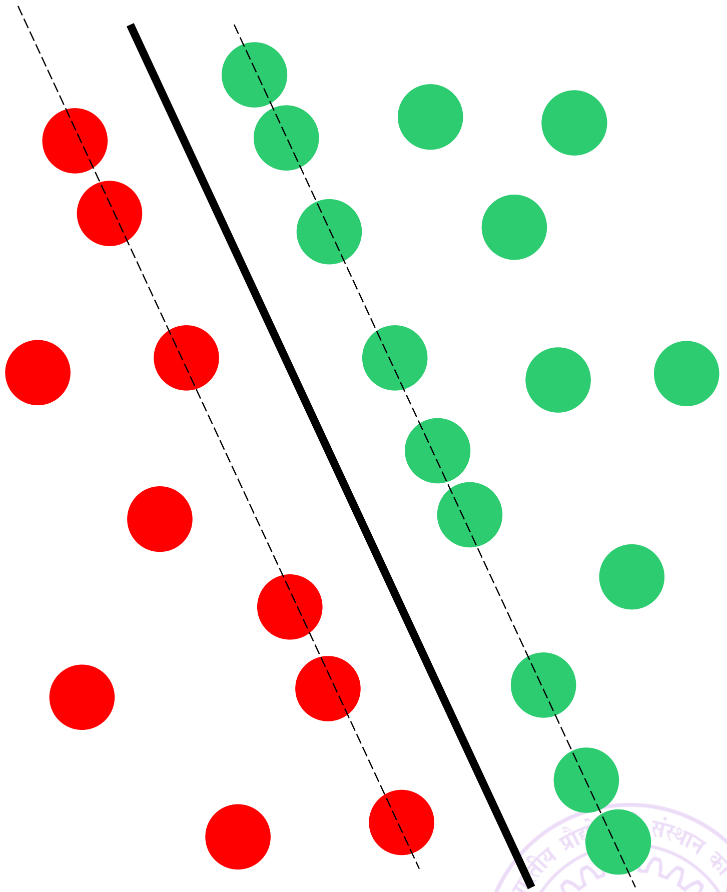
Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \dots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \dots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.



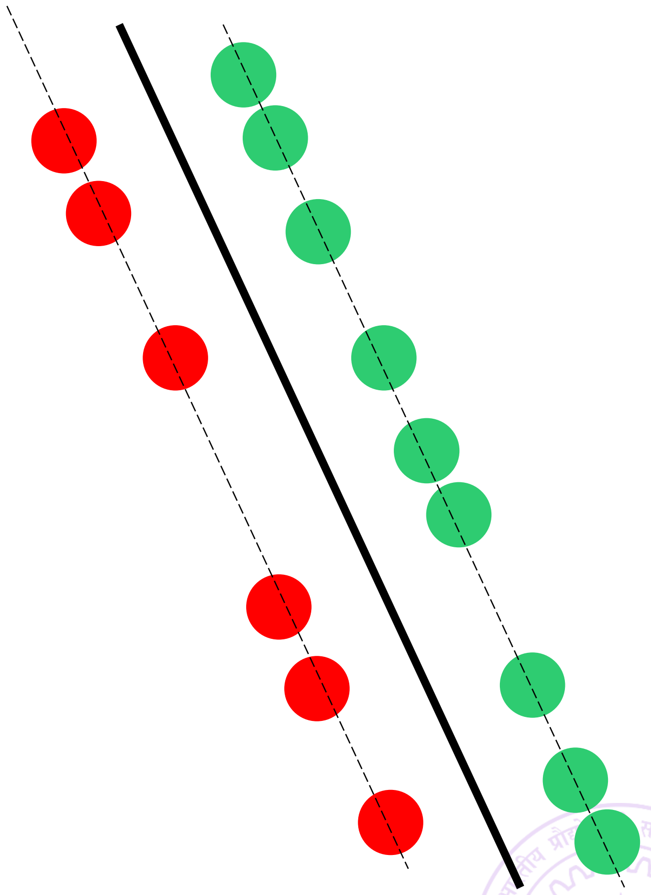
Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.



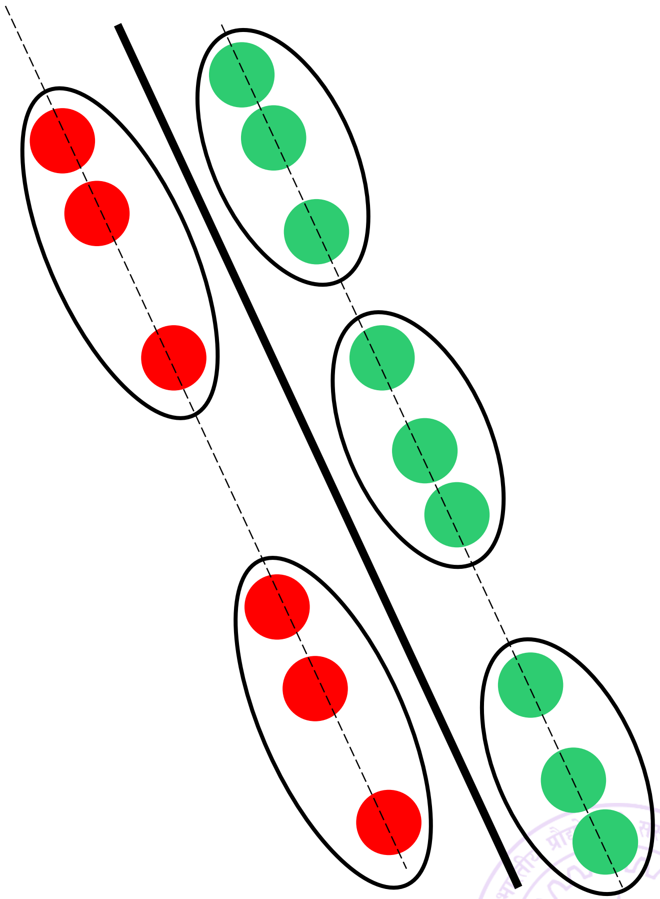
Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.



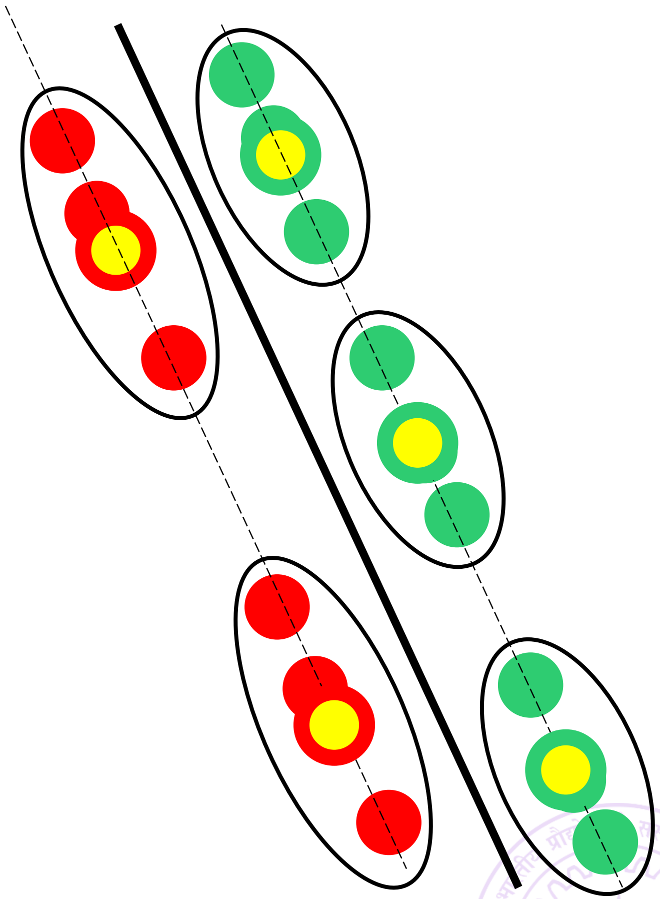
Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \dots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \dots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.



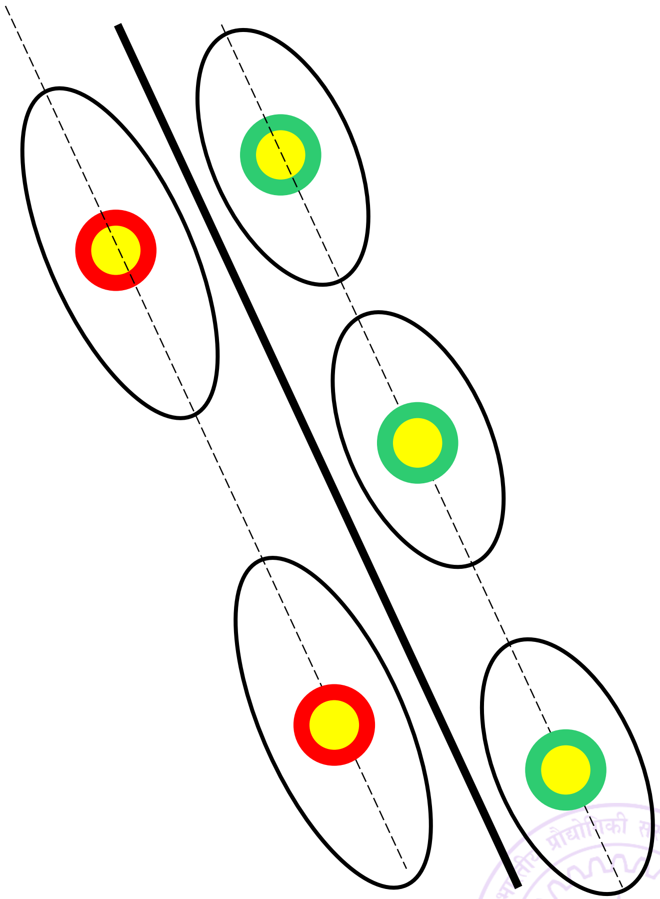
Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \dots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \dots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.



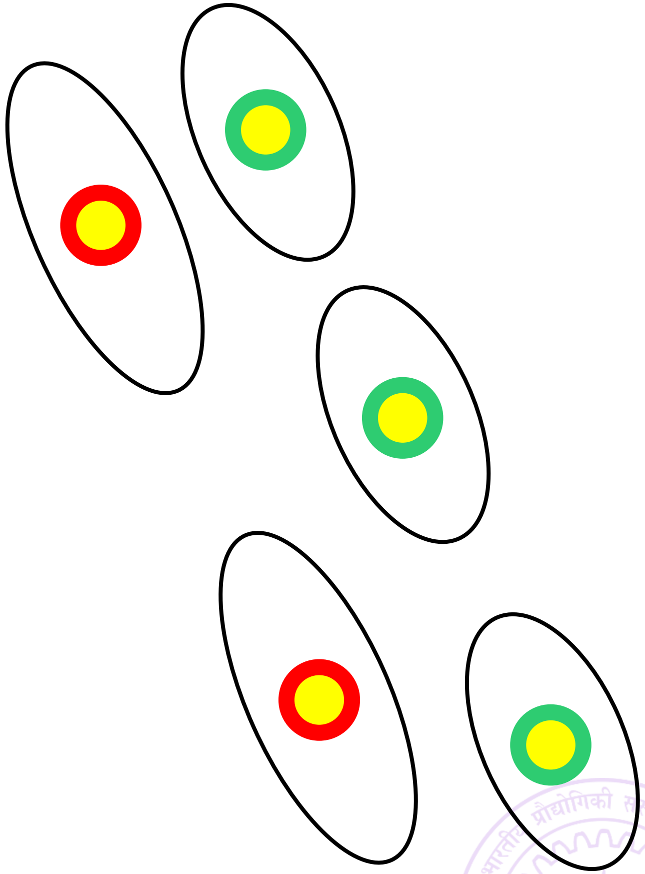
Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.



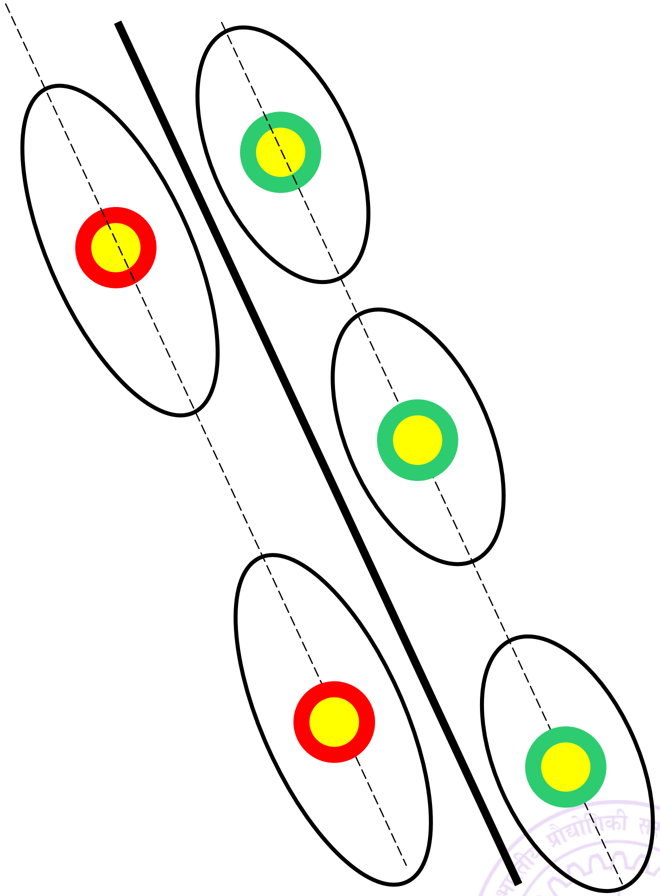
Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.



Post Processing Techniques

- Learn kernel SVM, support vectors $\{x_{i_j}, \alpha_{i_j}\}$
- Find a *reduced set* of $k \ll \tilde{n}$ support vectors $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$ e.g. by using k-means clustering on original support vectors
- Re-compute α values for these reduced set support vectors e.g. by running SVM again on $\{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_k}\}$
- Burges and Scholkopf, Improving the Speed and Accuracy of SVMs, NIPS 1996.
- Cossalter et al. Adaptive Kernel Approximation for Large-Scale Non-Linear SVM Prediction, ICML 2011.



Approximate Training Techniques

- Notice that support vectors are always a subset of training data
- Maybe removing this restriction can reduce their number?
- Learn support vectors as well (not necessarily training points)!
- Learn vectors $\mathbf{z}^1, \ldots, \mathbf{z}^k \in \mathbb{R}^d$ and weights $\alpha_1, \ldots, \alpha_k \in \mathbb{R}$ so that
$$\mathbf{w} = \sum_{i=1}^{k} \alpha_i \cdot \phi_K(\mathbf{z}^i)$$
is a good model (classifier, regressor etc)
- k chosen based on budget (space, time) of application
- $\mathcal{O}(kd)$ storage and $\mathcal{O}(kd)$ time for prediction
- Joachims and Yu. Sparse Kernel SVMs via Cutting-Plane Training, Machine Learning 76(2):179-193, 2009
- Tsang et al. Core Vector Machines, JMLR 6:363-392, 2005

Approximate Training Techniques

- Notice that support vectors are always a subset of training data
- Maybe removing this restriction can reduc r?
- Learn support vectors as well (not necessar training points)!
- Learn vectors $\mathbf{z}^1, \dots, \mathbf{z}^k \in \mathbb{R}^d$ and weights $\alpha_1, \dots, \alpha_k \in \mathbb{R}$ so that

$$\mathbf{w} = \sum_{i=1}^{k} \alpha_i \cdot \phi_K(\mathbf{z}^i)$$

ϕ_K is the map for kernel K

  is a good model (classifier, regressor etc)
- k chosen based on budget (space, time) of application
- $\mathcal{O}(kd)$ storage and $\mathcal{O}(kd)$ time for prediction
- Joachims and Yu. Sparse Kernel SVMs via Cutting-Plane Training, Machine Learning 76(2):179-193, 2009
- Tsang et al. Core Vector Machines, JMLR 6:363-392, 2005

Kernel Approximation

Landmarking

- Given: training set $S = \{x^1, \dots, x^n\}$ and kernel K
- Select $k \ll n$ *landmarks*

$$\hat{S} = \{\hat{x}^1, \dots, \hat{x}^k\} \subset S$$

- Choice may be random or careful (more expensive)
- Use landmarks to create a new k-dim. feature representation

$$\widehat{\phi}(x) = \left[K(x, \hat{x}^1), \dots, K\left(x, \hat{x}^k\right)\right]$$

- Now use $\widehat{\phi}(x)$ to perform classification, regression, etc
- Can be theoretically shown that if K was nice, so will be $\widehat{K}$
- No agony of high dim-feature map with $\widehat{K}$
- Balcan and Blum. On a Theory of Learning with Similarity Functions, ICML 2006.
- K. and Jain. Similarity-based Learning via Data driven Embeddings, NIPS 2011.

Landmarking

- Given training set $S = \{x^1, \ldots, x^n\}$ and kernel K
- $\hat{S} = \{\hat{x}^1, \ldots, \hat{x}^k\} \subset S$
- Choice may be random or careful (more expensi
- Use landmarks to create a new k-dim. feature repr

$$\hat{\phi}(x) = \left[K(x, \hat{x}^1), \ldots, K\left(x, \hat{x}^k\right)\right]$$

- Now use $\hat{\phi}(x)$ to perform classification, regressio
- Can be theoretically shown that if K was nice, so will be
- No agony of high dim-feature map with $\hat{K}$
- Balcan and Blum. On a Theory of Learning with Similarity Functions, IC
- K. and Jain. Similarity-based Learning via Data driven Embeddings, NIPS 2011.

Since k is chosen to be small, so use linear SVM/RR over $\hat{\phi}(x)$ directly

Can think of $\hat{\phi}$ as giving us a new kernel $\hat{K}(x, y) = \langle \hat{\phi}(x), \hat{\phi}(y) \rangle$

$\mathcal{O}(kd)$ model size and $\mathcal{O}(kd)$ prediction time

Work with non-Mercer kernels too!

Nystrom Method

- A more careful implementation of landmarking
- **Basic idea**: landmarks may be correlated – decorrelate them
- Recall landmark set $\hat{S}$ of size k gave us a map $\hat{\phi}$ that maps to $\mathbb{R}^k$
- Let $\hat{G} \in \mathbb{R}^{k \times k}$ be Gram matrix over landmark set $\hat{S}$ and let its eigendecomposition be $\hat{G} = U \Lambda U^\top$ where $U = [u^1, \ldots, u^k] \in \mathbb{R}^{k \times k}$ is the matrix of eigenvectors and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$ be eigenvalues
- Nystrom method defines the similarity between x, y as
$$\hat{\phi}(x)^\top G^\dagger \hat{\phi}(y)$$
- Nystrom features are modified version of landmarking feature $\hat{\phi}$
$$\tilde{\phi}(x) = \sqrt{\Lambda^{-1}} U^\top \hat{\phi}(x)$$
if any $\lambda_i = 0$, remove that eigenvalue from Λ and vector from U

Nystrom Method

- A more careful implementation of landmarking
- **Basic idea**: landmarks may be correlated – decorrelate them
- Recall landmark set $\hat{S}$ of size k gave us a map $\hat{\phi}$ that maps to $\mathbb{R}^k$
- Let $\hat{G} \in \mathbb{R}^{k \times k}$ be Gram matrix over landmark set $\hat{S}$ and let its eigendecomposition be $\hat{G} = U \Lambda U^\top$ where $U = [u^1, \ldots, u^k] \in \mathbb{R}^{k \times k}$ is the matrix of eigenvectors and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_k$ es
- Nystrom method defines the similarity between x, y as

Pseudoinverse

$$\hat{\phi}(x)^\top G^\dagger \hat{\phi}(y)$$

- Nystrom features are modified version of landmarking feature $\hat{\phi}$

$$\tilde{\phi}(x) = \sqrt{\Lambda^{-1}} U^\top \hat{\phi}(x)$$

if any $\lambda_i = 0$, remove that eigenvalue from Λ and vector from U

Nystrom Method

- The Nystrom feature also gives us a new kernel $\widetilde{K}$

$$\widetilde{K}(x, y) = \tilde{\phi}(x)^\top \tilde{\phi}(y) = \hat{\phi}(x)^\top U \Lambda^{-1} U^\top \hat{\phi}(y)$$

- Note that the Gram matrices corresponding to the landmarking kernels $\widehat{K}$ as well as Nystrom kernel $\widetilde{K}$ are always rank atmost k
- Interesting note: suppose actual kernel K has map $\phi_K(x) \in \mathbb{R}^D$
- Let $\Phi_{\hat{S}} = \left[\phi\left(\hat{x}^i\right)\right]_{i=1,\dots,k} \in \mathbb{R}^{D \times k}$ where $\hat{S} = \{\hat{x}^1, \dots, \hat{x}^k\}$ is landmark set
- This means $\hat{\phi}(x) = \Phi_{\hat{S}}^\top \phi_K(x)$ and $\hat{G} = \Phi_{\hat{S}}^\top \Phi_{\hat{S}}$ i.e.

$$\widetilde{K}(x, y) = \phi_K(x)^\top \Phi_{\hat{S}} \left(\Phi_{\hat{S}}^\top \Phi_{\hat{S}}\right)^\dagger \Phi_{\hat{S}}^\top \phi_K(y)$$

- Takes more time $O(k^2 + kd)$ to construct Nystrom feature map
- Williams and Seeger. Using the Nystrom Method to Speed Up Kernel Machines, NIPS 2000
- Yang et al. Nystrom Method vs Random Fourier Features, NIPS 2012

Nystrom Method

- The Nystrom feature also gives us a new kernel $\widetilde{K}$

$$\widetilde{K}(x,y) = \tilde{\phi}(x)^\top \tilde{\phi}(y) = \hat{\phi}(x)^\top U \Lambda^{-1} U^\top \hat{\phi}(y)$$

- Note that the Gram matrices corresponding to the landmarking kernels $\widehat{K}$ as well as Nystrom kernel $\widetilde{K}$ are always rank atmost k
- Interesting note: suppose actual kernel K has map $\phi_K(x) \in \mathbb{R}^D$
- Let $\Phi_{\hat{S}} = \left[\phi\left(\hat{x}^i\right)\right]_{i=1,\ldots,k} \in \mathbb{R}^{D\times k}$ where $\hat{S} = \{\hat{x}^1, \ldots, \hat{x}^k\}$ is landmark set
- This means $\hat{\phi}(x) = \Phi_{\hat{S}}^\top \phi_K(x)$ and $\hat{G} = \Phi_{\hat{S}}^\top \Phi_{\hat{S}}$ i.e.

$$\widetilde{K}(x,y) = \phi_K(x)^\top \Phi_{\hat{S}} \left(\Phi_{\hat{S}}^\top \Phi_{\hat{S}}\right)^\dagger \Phi_{\hat{S}}^\top \phi_K(y)$$

Decorrelation

- Takes more time $O(k^2 + kd)$ to construct Nystrom feature map
- Williams and Seeger. Using the Nystrom Method to Speed Up Kernel Machines, NIPS 2000
- Yang et al. Nystrom Method vs Random Fourier Features, NIPS 2012

Explicit Feature Constructions

- Realize that high dim. of feature map ϕ_K is root of all problems
- If ϕ_K were small dim. then training, storage, testing much easier
- Given a Mercer kernel $K: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ can we construct $\bar{\phi} : \mathcal{X} \to \mathbb{R}^k$
  - k should not be too large so we can use $\bar{\phi}$ explicitly
  - It should be easy to map $x \mapsto \bar{\phi}(x)$
  - $\bar{\phi}$ should act as an approx feature map for K i.e. for all $x, y \in \mathcal{X}$
$$\langle \bar{\phi}(x), \bar{\phi}(y) \rangle =: \bar{K}(x, y) \approx \mathrm{K}(\mathrm{x}, \mathrm{y}) = \langle \phi_K(x), \phi_K(y) \rangle$$
- Note that landmarking and Nystrom do not seek to ensure that $\hat{K}$ or $\tilde{K}$ values approximate K but $\bar{K}$ should approximate K values
- Why should such $\bar{\phi}$ even exist?

Random Feature Constructions

- Several popular Mercer kernels have a peculiar form
$$K(x,y) = \mathbb{E}_{\omega\sim\mathcal{D}_K}[K_\omega(x,y)]$$
    - ω is an auxiliary variable (depending on the kernel, $\omega \in \mathbb{N}, \mathbb{R}, \mathbb{R}^d$)
    - $\mathcal{D}_K$ is a distribution that depends on kernel K and known to us
    - K_ω is a very "simple" Mercer kernel, has a one-dim. feature map
$$K_\omega(x,y) = \langle\phi_\omega(x), \phi_\omega(y)\rangle$$
$$\phi_\omega : \mathcal{X} \to \mathbb{R}$$
- Sample several $\omega_1, \dots, \omega_k$ and define the map
$$\bar{\phi} : x \mapsto \left[\phi_{\omega_1}(x), \dots, \phi_{\omega_k}(x)\right] \in \mathbb{R}^k$$
- Can theoretically prove that with high probability
$$\left\langle\bar{\phi}(x), \bar{\phi}(y)\right\rangle \approx K(\mathrm{x},\mathrm{y})$$

Random Feature Constructions

- Gaussian/Laplacian kernels

$$K(\mathbf{x},\mathbf{y}) = \mathbb{E}_{\boldsymbol{\omega}\sim\mathcal{D}_K}[\cos(\boldsymbol{\omega}^\top\mathbf{x})\cos(\boldsymbol{\omega}^\top\mathbf{y})]$$

$$\phi_{\boldsymbol{\omega}}: \mathbf{x} \mapsto \cos(\boldsymbol{\omega}^\top\mathbf{x})$$

  Note that auxiliary variable is a vector here $\boldsymbol{\omega} \in \mathbb{R}^d$

  Rahimi and Recht, Random Features for Large Scale Kernel Machines, NIPS 2007

- Intersection kernel

  Maji and Berg, Max-margin Additive Classifiers for Detect, ICCV 2009.

- Homogeneous kernels

  Vedaldi and Zisserman. Efficient Additive Kernels via Explicit Feature Maps, CVPR 2010

- Polynomial kernels

  K. and Karnick. Random Feature Maps for Dot Product Kernels. AISTATS 2012

Other kernel approximation approaches

- Use decision trees to compute similarity between two points and use that as kernel – extremely fast prediction

  Jose et al. Local Deep Kernel Learning, ICML 2013.

- Learn these kernel approximations in a task-dependent manner

  Perronnin et al. d Yan Liu. Large-scale Image Categorization with Explicit Data embedding, CVPR 2010.

PML with Kernels

Gaussian Processes

Priors and Posteriors

- How can we argue about priors and posteriors in an RKHS $\mathcal{H}_K$?
- Details too advanced (covered in CS772, CS775, CS698X)
- Basic idea: argue about distributions over functions $f: \mathcal{X} \to \mathbb{R}$
- Gaussian processes is one such family of distributions

$$f \sim \mathrm{GP}(\mu, K)$$

  μ: $\mathcal{X} \to \mathbb{R}$ is the *mean* function and κ is the covariance kernel
- What does it mean to sample a function?
- Think of sampling a very very long vector (imprecise way though)
- Let $|\mathcal{X}| = N < \infty$ with $\mathcal{X} = \{x^1, x^2, \ldots, x^N\}$
- Then can think of $f: \mathcal{X} \to \mathbb{R}$ as a vector in $\mathbb{R}^N$

$$f = [f(x^1), \ldots, f(x^N)]$$

Gaussian Processes

- For $|\mathcal{X}| = N < \infty$, we say a function f is sampled from $\mathrm{GP}(\mu, K)$ if
$$f \sim \mathcal{N}(\mu, G)$$
where $\mu \in \mathbb{R}^N$ is mean fn. and $G \in \mathbb{R}^{N \times N}$ with $G_{ij} = K\left(x^i, x^j\right)$
- Note that f need not be linear etc, can be very complex
- Gaussian processes popularly use a Gaussian kernel for K
- Note that the Gaussian kernel K forces f to be *smooth* i.e. if two points $x^i, x^j \in \mathcal{X}$ are close i.e. $\left\|x^i - x^j\right\|_2$ is small then functions f that take very different values on these points get low prob.
- **Exercise**: verify this yourself
- Mean function is taken to be zero (unless we have other reasons)

Gaussian Process Regression

- Solve a regression problem $\{x^i, y^i\}_{i=1,\ldots,n}$,$x^i \in \mathcal{X}, y^i \in \mathbb{R}$ and $n \ll N$
- Prior dist. (GP) $f \sim \mathrm{GP}(0, K)$
- Likelihood dist. (Gaussian) $y^i | f \sim \mathcal{N}\left(f\left(x^i\right), \sigma^2\right)$
- Note: GP makes sense even if $\mathcal{X}$ is set of vectors, images, text etc
- Can do regression over vectors, images as we did in kernel RR
- Let $\mathbf{y} = [y^1, \ldots, y^n]^\top \in \mathbb{R}^n$
- Using a very special property of Gaussians we can show

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, G_n + \sigma^2 \cdot I_n)$$

where $\mathbf{0} \in \mathbb{R}^n$ and $G_n \in \mathbb{R}^{n \times n}$ is the Gram matrix of training data

Gaussian Process Regression

- Solve $^i, y^i\}$ $,x^i \in \mathcal{X}, y^i \in \mathbb{R}$ and $n \ll N$
- Prior (
- Likelih

If a vector $\mathbf{v} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ then $\mathbf{v}_S \sim \mathcal{N}\left(\mu_S, \Sigma_{S,S}\right)$ $\Sigma_{S,S} \in \mathbb{R}^{|S|\times|S|}$

If a vector $\mathbf{v} \in \mathbb{R}^n$ is distributed according to a Gaussian, then for every subset $S \subset [n]$, the sub-vector $\mathbf{v}_S = [\mathbf{v}_i]_{i\in S} \in \mathbb{R}^{|S|}$ is also a Gaussian vector!

- es sense even if $\mathcal{X}$
- sion over ve s, in

$\mathbf{y}$ is just a subvector of f

- Let $\mathbf{y} = [y$ $y^n]^\top \in \mathbb{R}^n$
- Using a very special property of Gaussians we can show

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, G_n + \sigma^2 \cdot I_n)$$

where $\mathbf{0} \in \mathbb{R}^n$ and $G_n \in \mathbb{R}^{n\times n}$ is the Gram matrix of training data

Gaussian Process Regression

- So we have $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, G_n + \sigma^2 \cdot I_n)$
- Now a new test point comes along $\tilde{x} \in \mathcal{X}$. Can we predict $\tilde{y}$?
- Let $\tilde{\mathbf{y}} = [\mathbf{y}, \tilde{y}] \in \mathbb{R}^{n+1}$, G_{n+1} be the Gram matrix over $\{x^i\}_{i=1,\ldots,n} \cup \tilde{x}$
- Previous slide gives us $\tilde{\mathbf{y}} \sim \mathcal{N}(\mathbf{0}, G_{n+1} + \sigma^2 \cdot I_{n+1})$
- Let $\tilde{\mathbf{g}} = [K(x^1, \tilde{x}), \ldots, K(x^n, \tilde{x})]^\top \in \mathbb{R}^n$
- Then we can show that

$$\mathbb{P}\left[\tilde{y} \,\middle|\, \tilde{x}, \{x^i, y^i\}_{i=1,\ldots,n}\right] = \mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2)$$
$$\tilde{\mu} = \tilde{\mathbf{g}}^\top (G_n + \sigma^2 \cdot I_n)^{-1} \mathbf{y}$$
$$\tilde{\sigma}^2 = K(\tilde{x}, \tilde{x}) + \sigma^2 - \tilde{\mathbf{g}}^\top (G_n + \sigma^2 \cdot I_n)^{-1} \tilde{\mathbf{g}}$$

Gaussian Process Regression

- So we have $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, G_n + \sigma^2 \cdot I_n)$
- Now a new test point comes along $\tilde{x} \in \mathcal{X}$. Can we predict $\tilde{y}$?
- Let $\tilde{\mathbf{y}} = [\mathbf{y}, \tilde{y}] \in \mathbb{R}^{n+1}$, G_{n+1} be the Gram matrix over
- Previous slide gives us $\tilde{\mathbf{y}} \sim \mathcal{N}(\mathbf{0}, G_{n+}$
- Let $\tilde{\mathbf{g}} = [K(x^1, \tilde{x}), \ldots, K(x^n, \tilde{x})]^\top \in \mathbb{R}^n$
- Then we can show that

$$\mathbb{P}\left[\tilde{y} \,\middle|\, \tilde{x}, \{x^i, y^i\}_{i=1,\ldots,n}\right] = \mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2)$$

$$\tilde{\mu} = \tilde{\mathbf{g}}^\top (G_n + \sigma^2 \cdot I_n)^{-1} \mathbf{y}$$

$$\tilde{\sigma}^2 = K(\tilde{x}, \tilde{x}) + \sigma^2 - \tilde{\mathbf{g}}^\top (G_n + \sigma^2 \cdot I_n)^{-1} \tilde{\mathbf{g}}$$

Predictive posterior

Verify that the mean $\tilde{\mu}$ is nothing but the kernel RR solution!

A few thoughts

- GP regression is a Bayesian counterpart to kernel RR
- Similar cost for storing model, making predictions
- GP gives additional information about variance in prediction just as Bayesian models usually do (ref. Bayesian linear regression)
- Can apply accelerated learning techniques to GPs as well
- Can use GPs to perform kernel dim-redn as well
- Just as we did online MAP, can do online GP as well
- Btw, can do online kernel SVM, online kernel RR as well ☺
- Kernel perceptron is already online

Neural Networks

Disclaimers

- Field is progressing rapidly – newer methods being proposed
- Some of the mentors, even some course students, more experienced with neural networks than the instructor
- Will cover very basics and essentials

Back to Kernels first

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$

Back to Kernels first

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$
- The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

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

Can represent any quadratic fn over $\mathbf{x}$

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

Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$

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

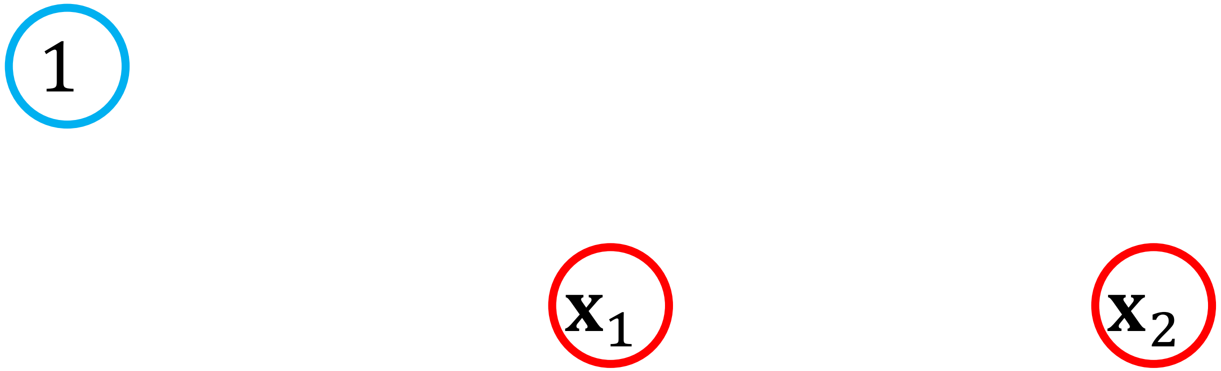
Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$



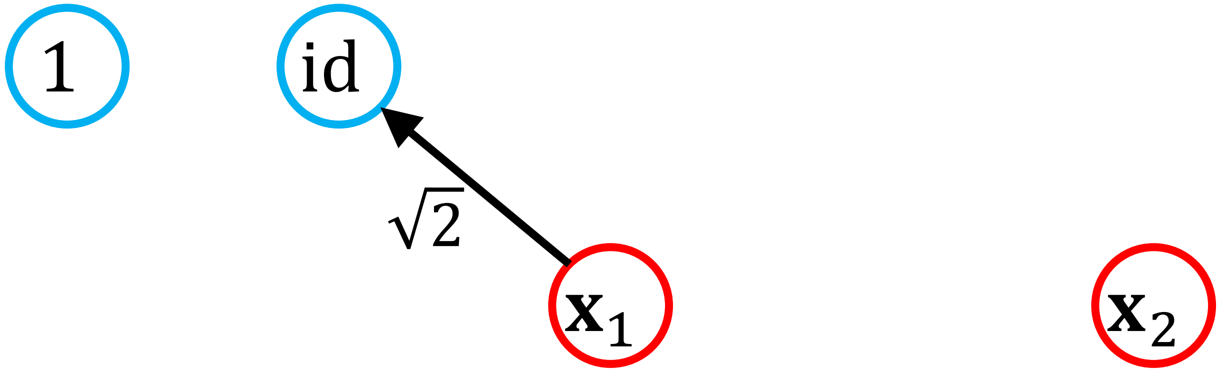
Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$



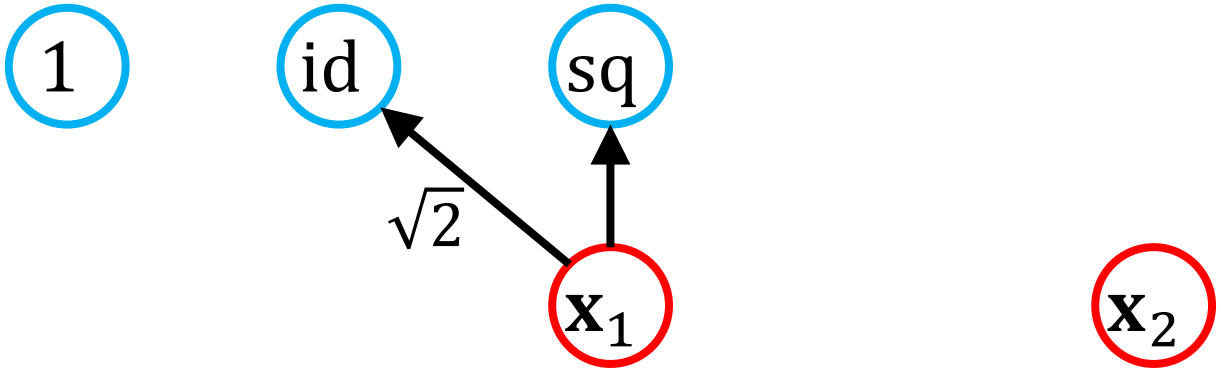
Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$



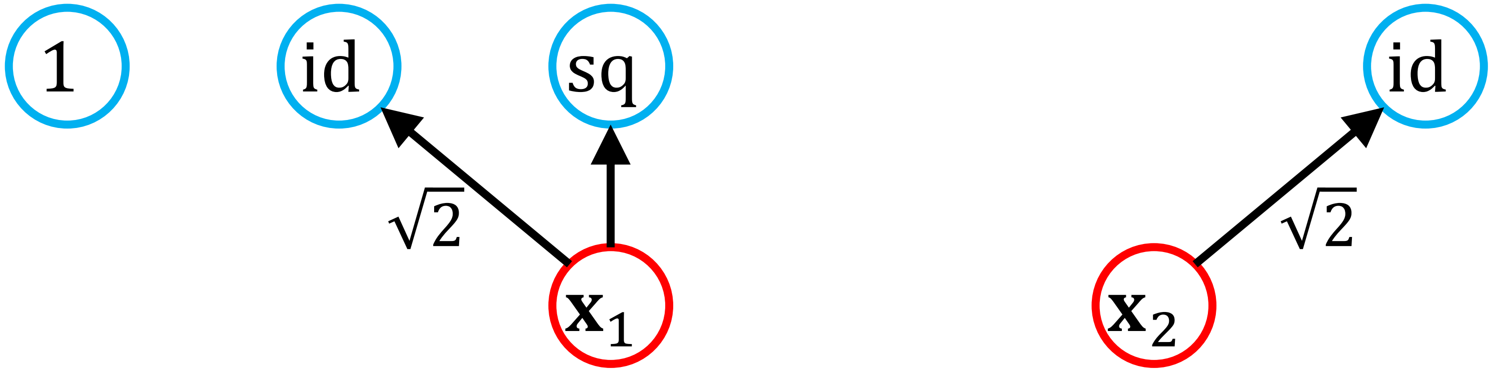
Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$



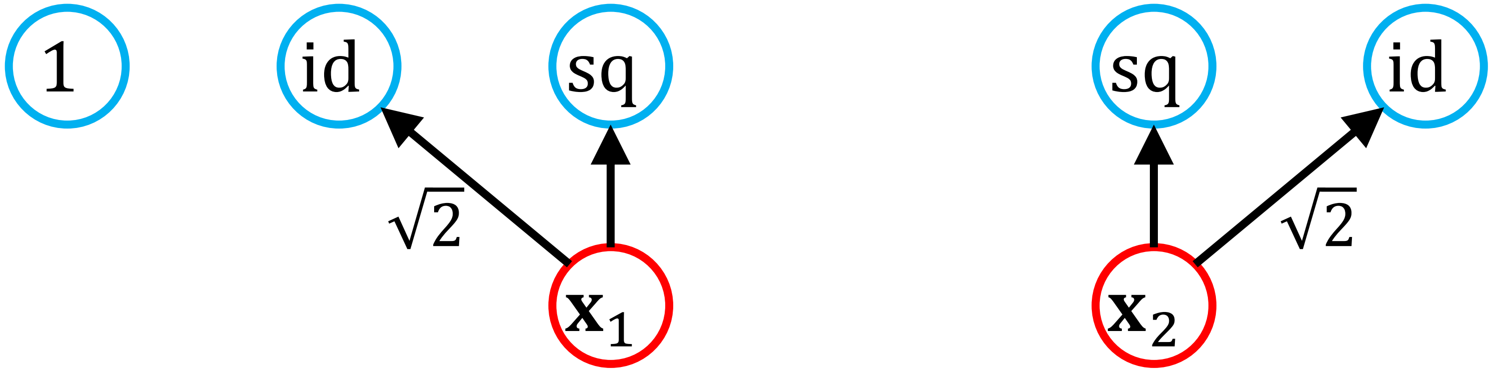
Back to Kernels first

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$

Can represent any quadratic fn over $\mathbf{x}$



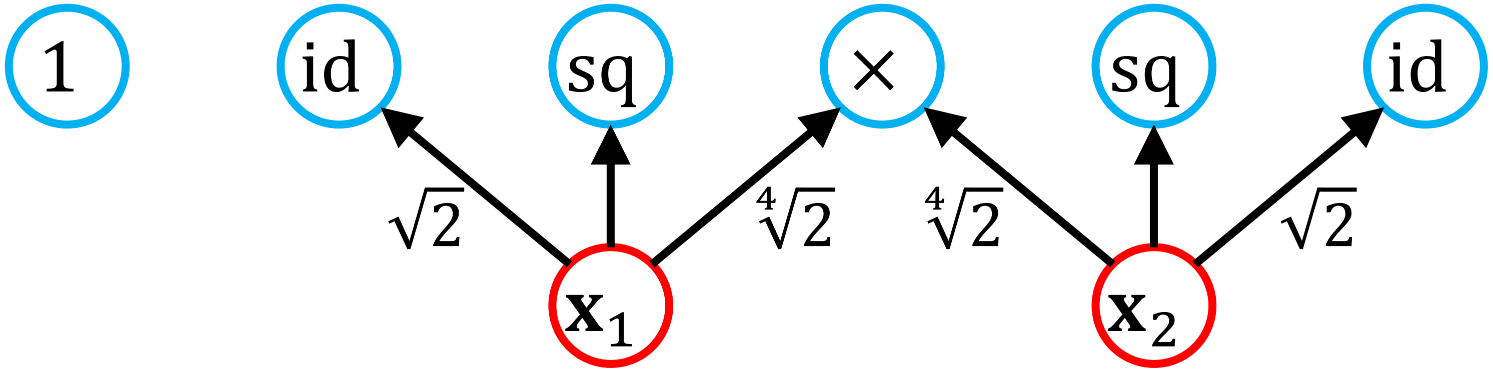
Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$



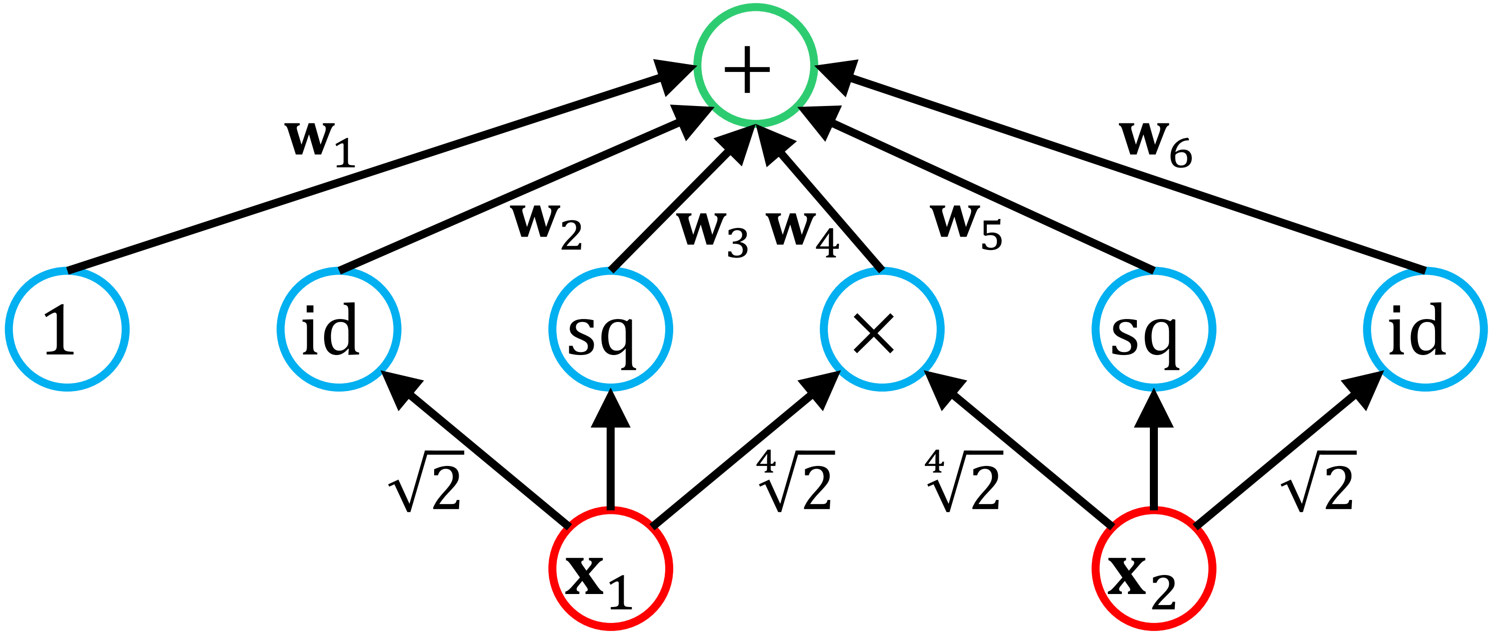
Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$



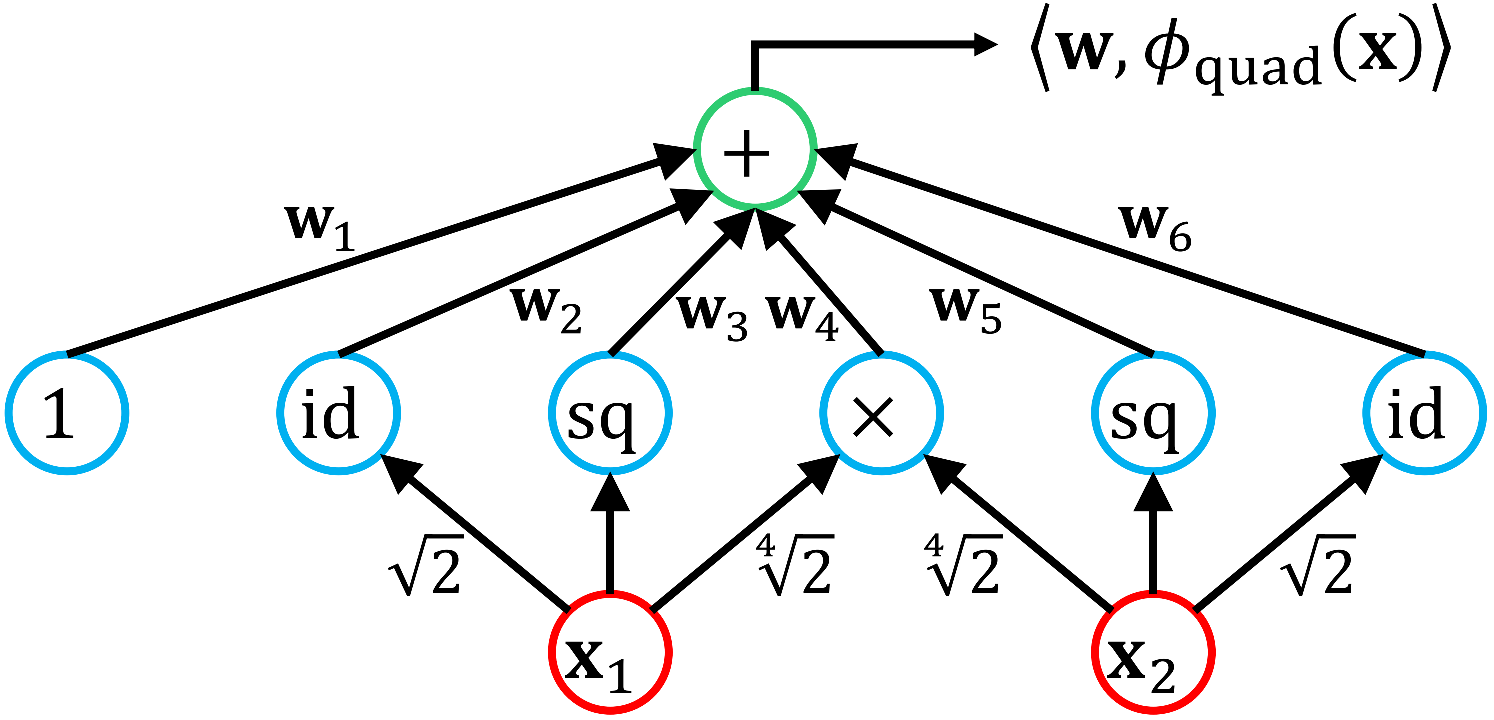
Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ 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 $\mathbf{w} \in \mathbb{R}^6$



Back to Kernels first

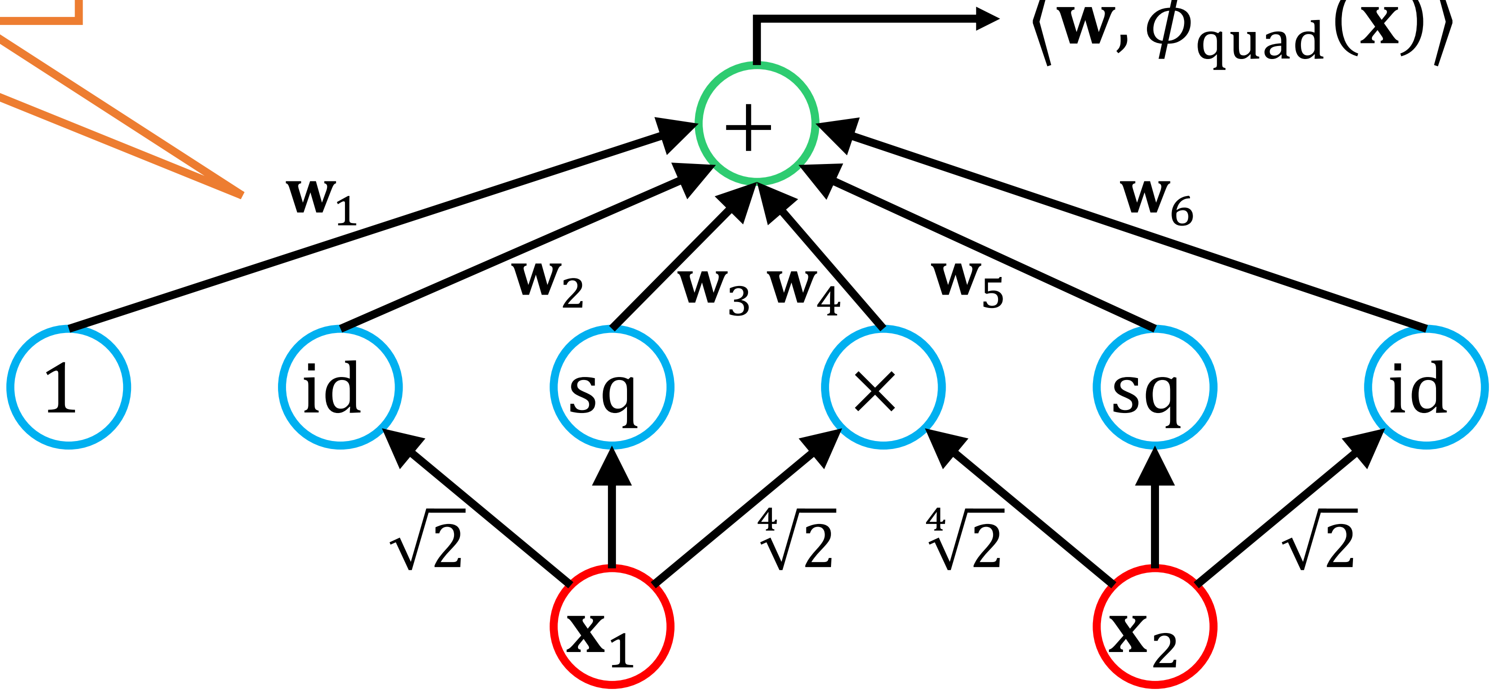
Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$
- The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

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

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

Training an SVM using GD/CD tunes these weights



Back to Kernels first

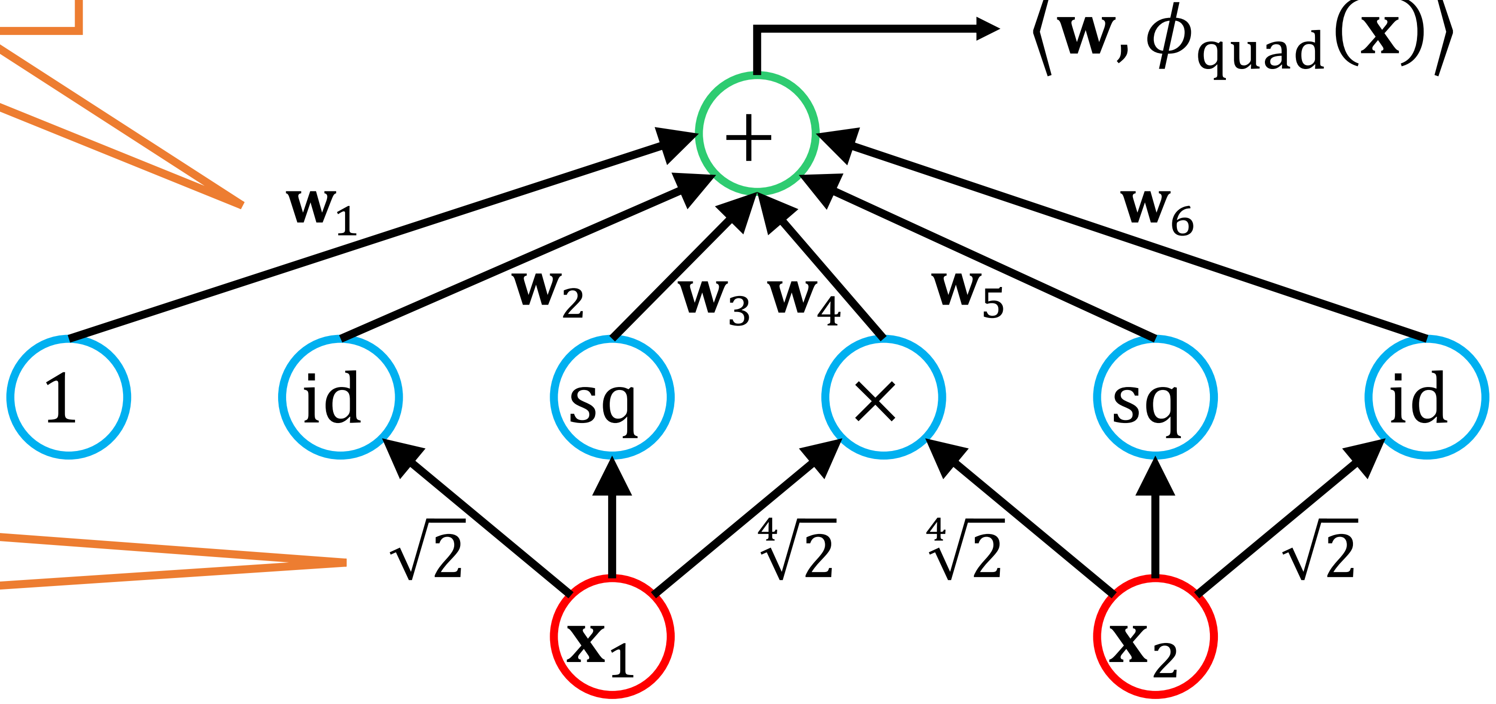
Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$
- The feature map for K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

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

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

Training an SVM using GD/CD tunes these weights



But not these weights

Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

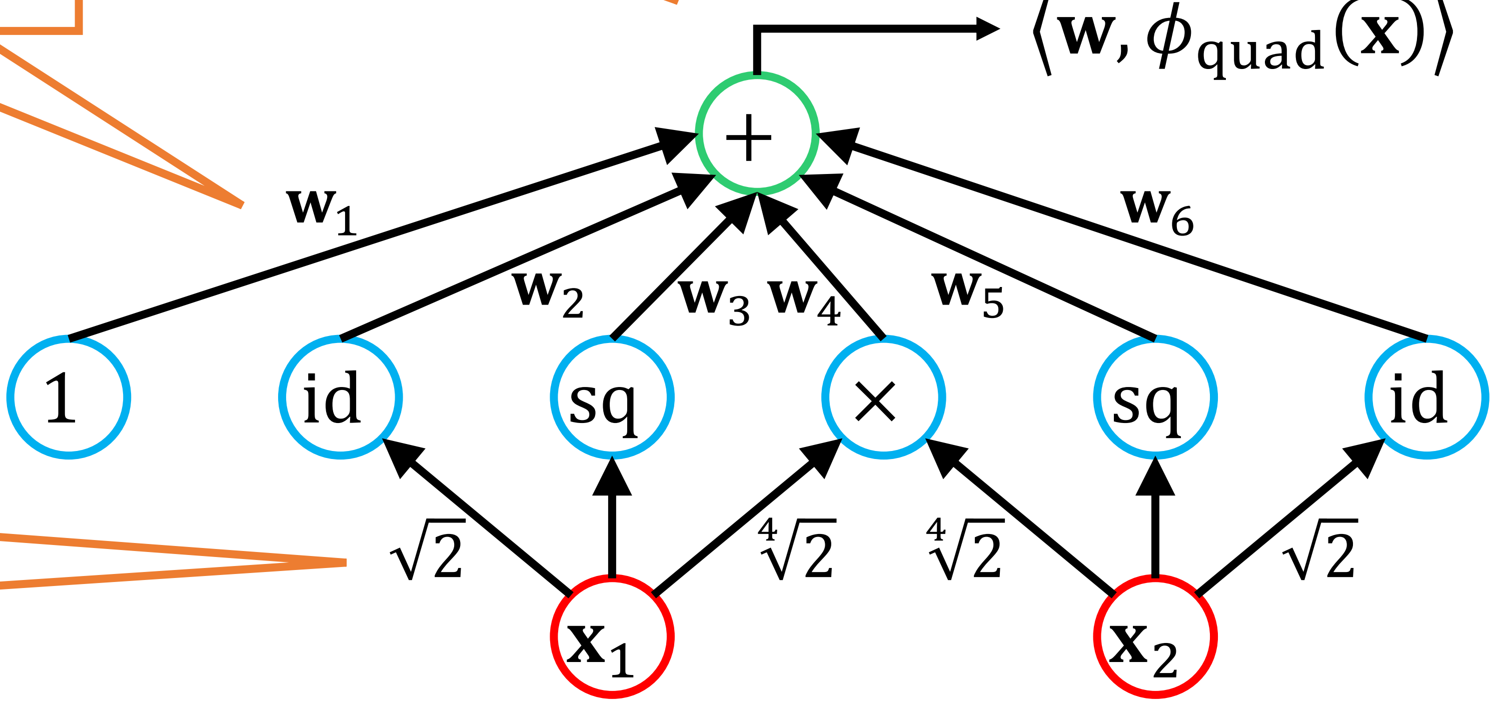
- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$
- Th K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

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

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

A 3 layer network

Training an SVM using GD/CD tunes these weights



But not these weights

Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

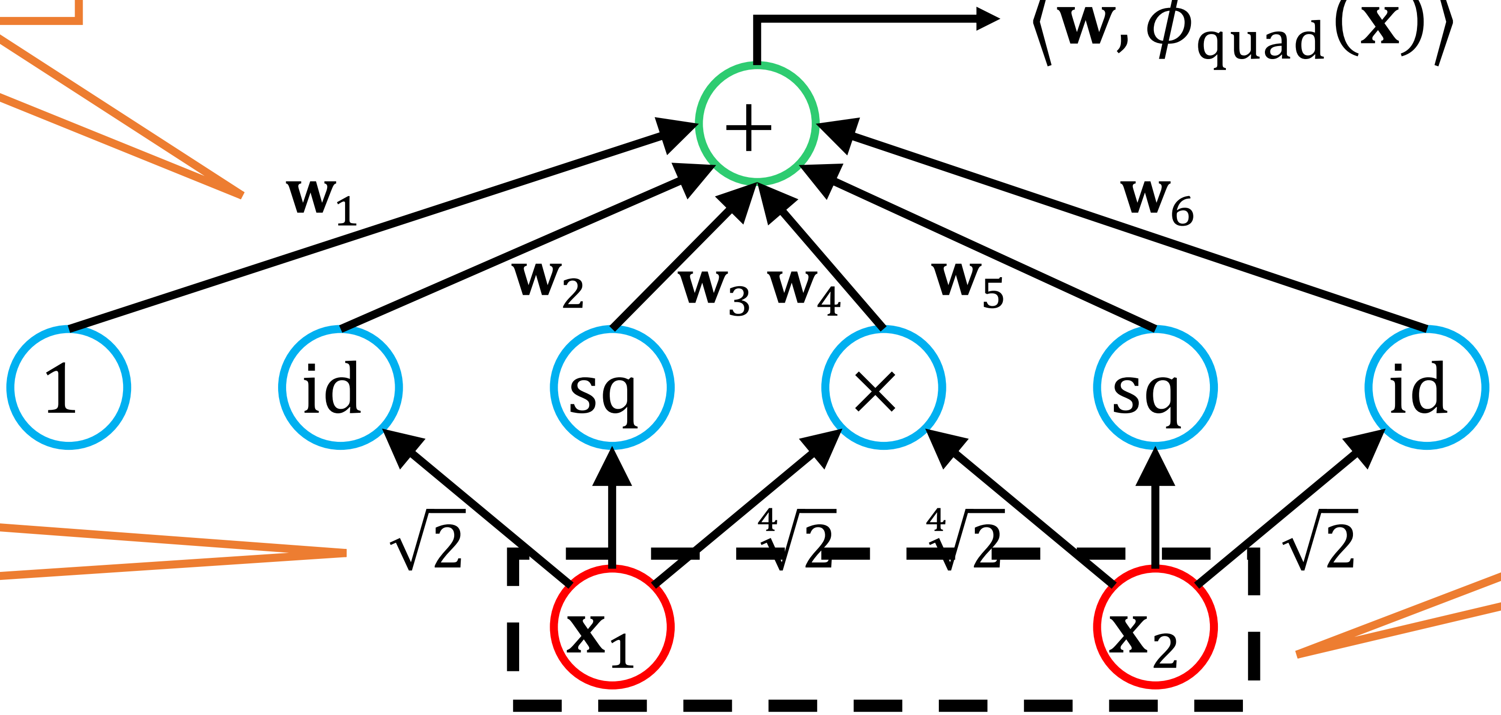
- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$
- Th K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

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

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

A 3 layer network

Training an SVM using GD/CD tunes these weights



But not these weights

Input layer

Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$
- Th K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

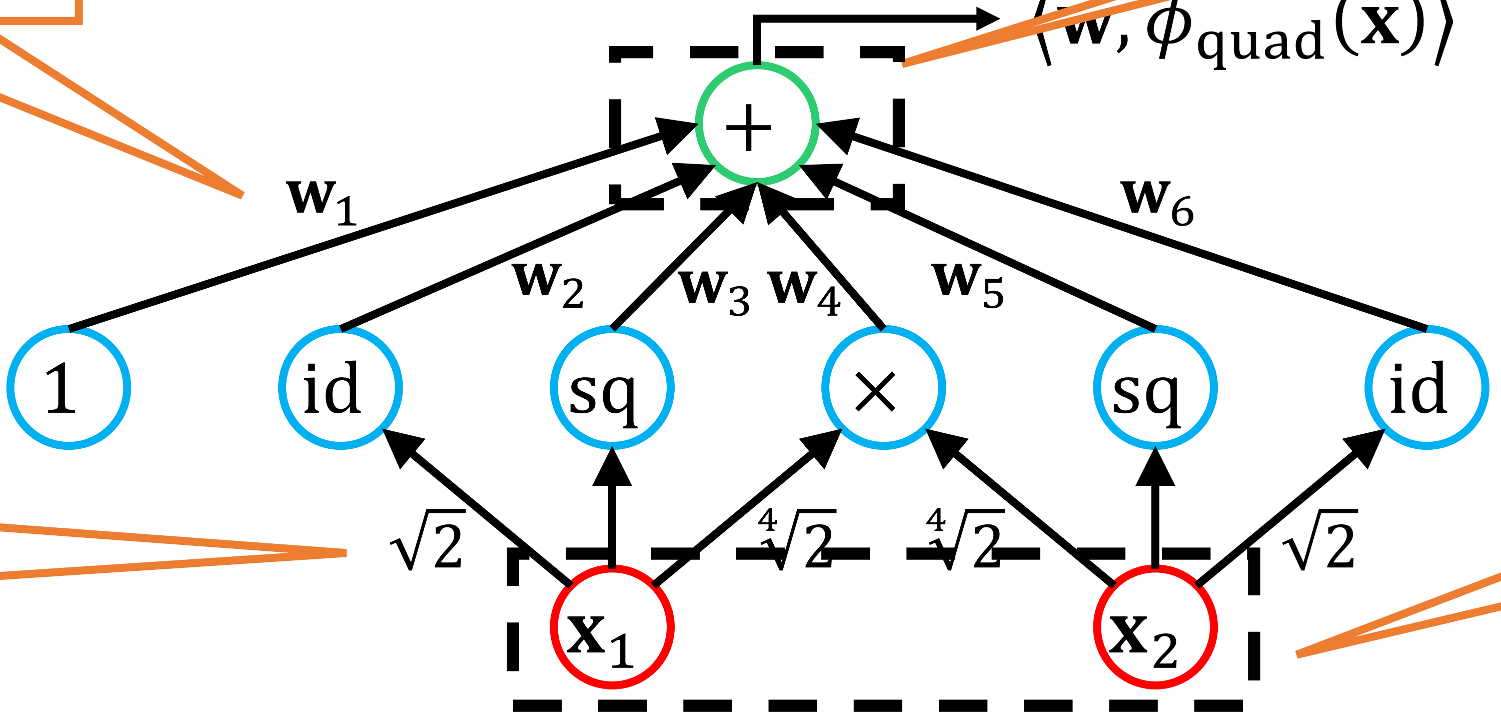
$) = [1\ \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2] \in \mathbb{R}^6$

over ϕ_{quad} is represented as a vect $\mathbf{w} \in \mathbb{R}$

A 3 layer network

Training an SVM using GD/CD tunes these weights

Output layer



But not these weights

Input layer

Back to Kernels first

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$
- Th K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

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

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

Can represent any quadratic fn over $\mathbf{x}$

A 3 layer network

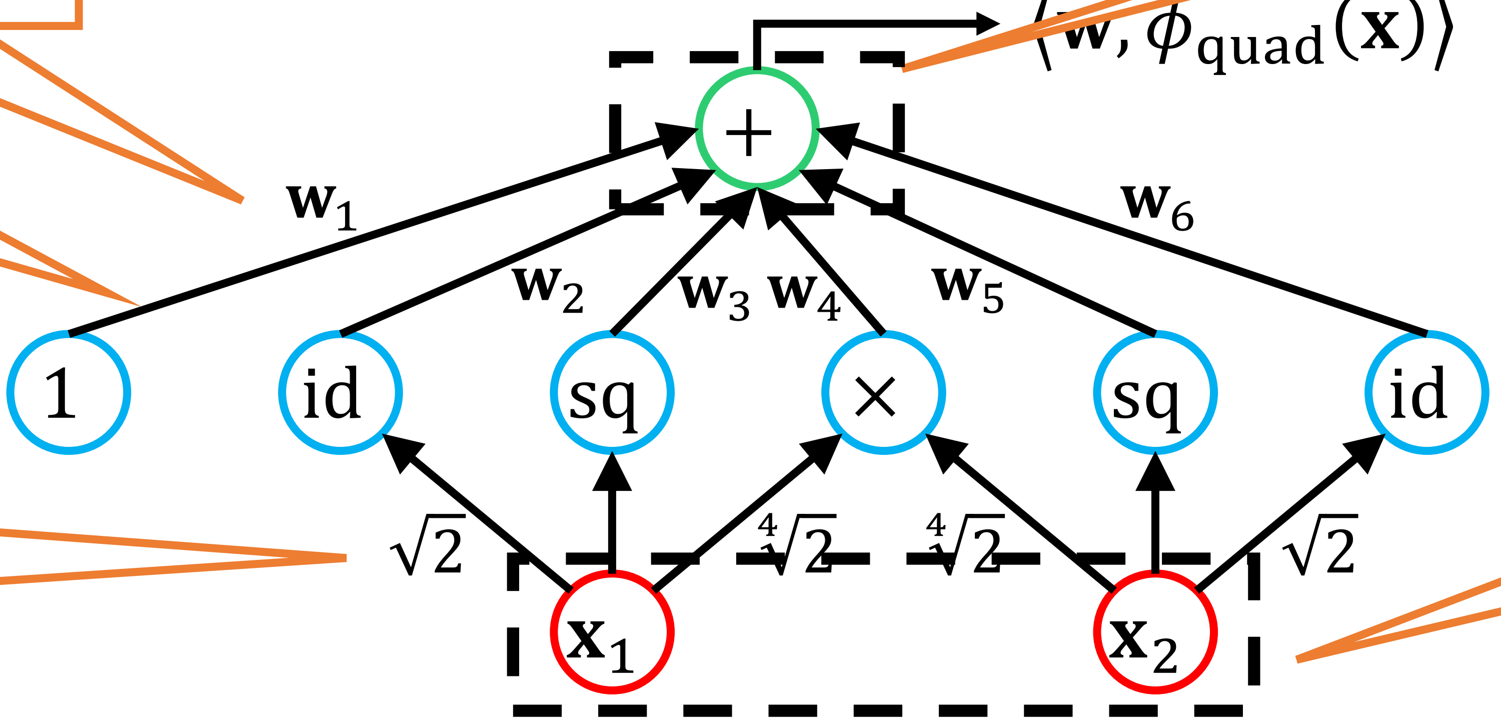
Training an SVM using GD/CD tunes these weights

Output layer

I/O layers are called "visible"

But not these weights

Input layer



Back to Kernels first

Can represent any quadratic fn over $\mathbf{x}$

- Consider the quadratic kernel $K_{\text{quad}} = (\langle \mathbf{x}^1, \mathbf{x}^2 \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$
- Th K_{quad} is ϕ_{quad} where for $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$

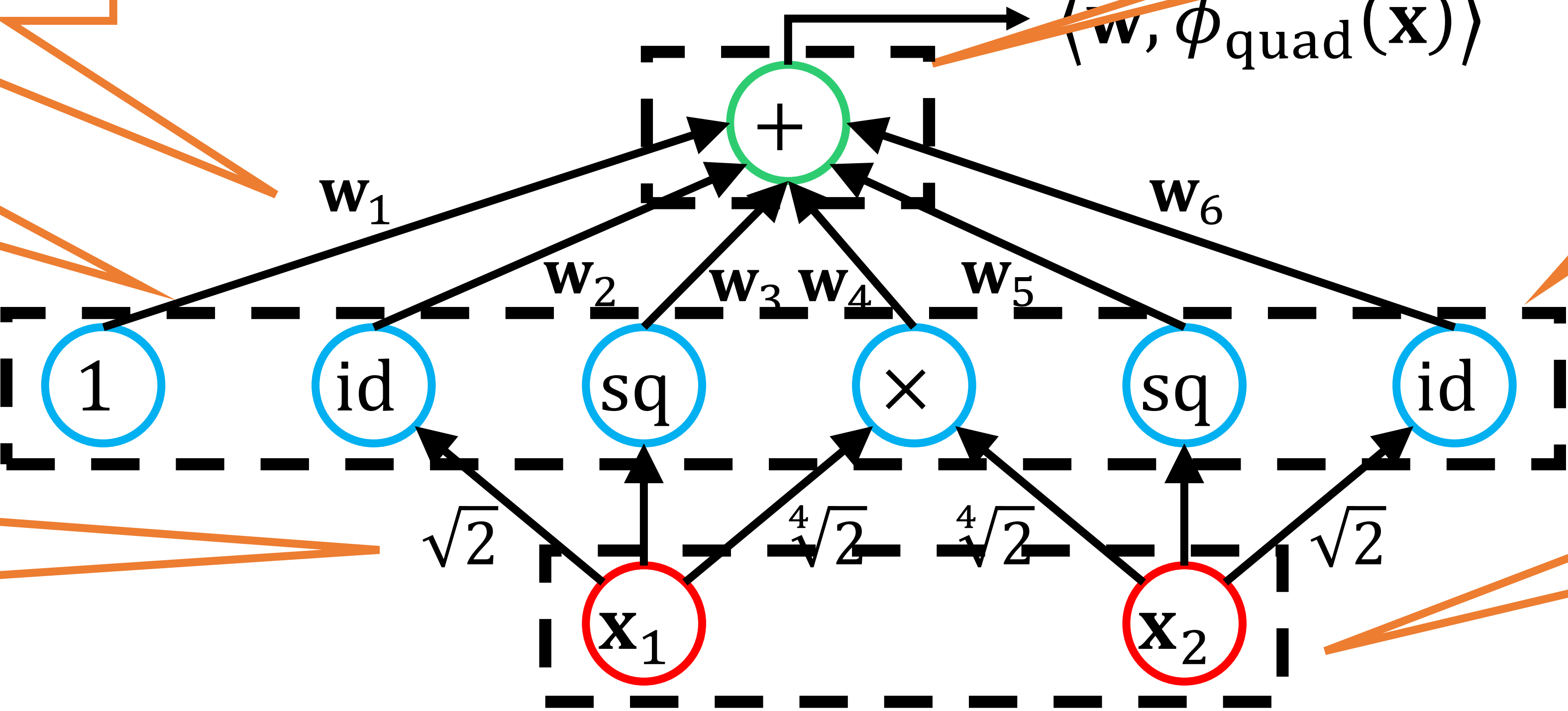
$$) = [1, \sqrt{2} \cdot \mathbf{x}_1, \mathbf{x}_1^2, \sqrt{2} \cdot \mathbf{x}_1 \cdot \mathbf{x}_2, \mathbf{x}_2^2, \sqrt{2} \cdot \mathbf{x}_2] \in \mathbb{R}^6$$

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

A 3 layer network

Training an SVM using GD/CD tunes these weights

Output layer



I/O layers are called "visible"

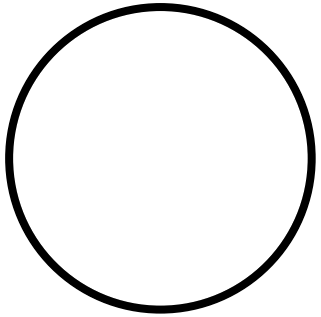
Hidden layer

But not these weights

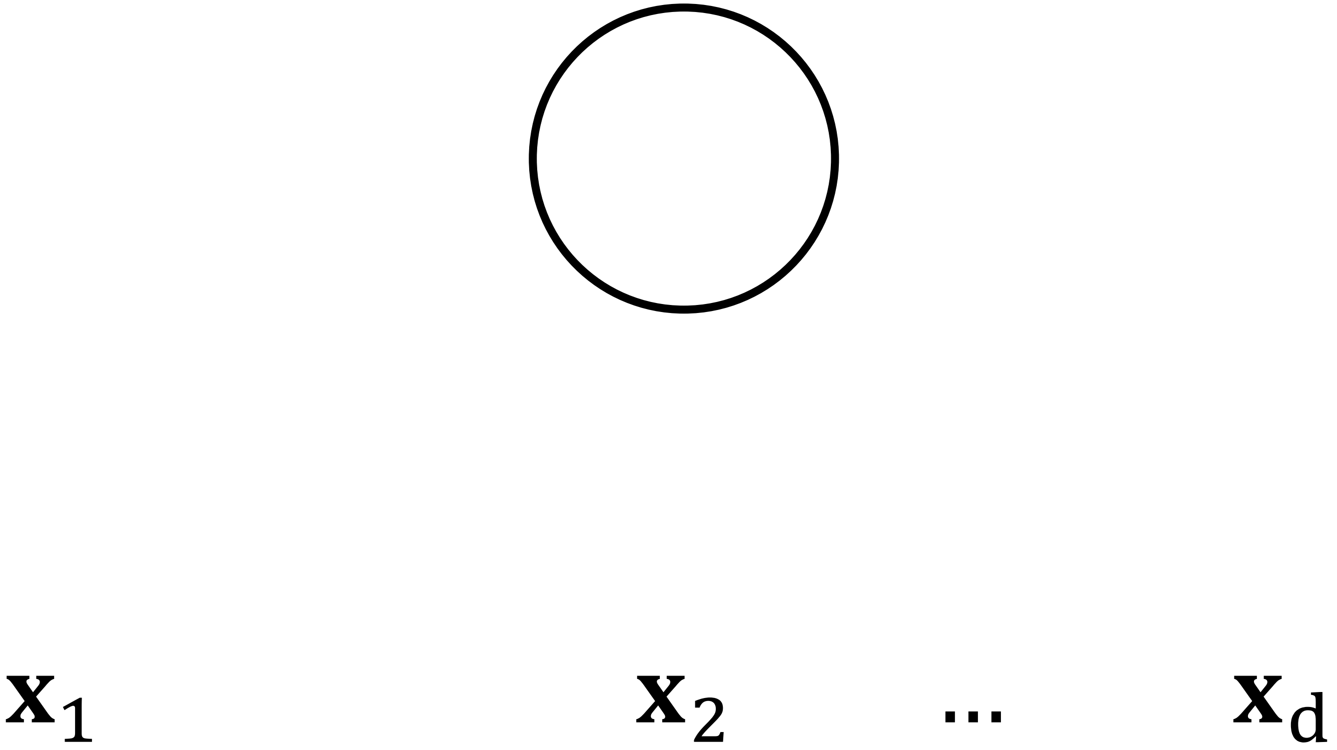
Input layer

The “neuron” in Neural Networks

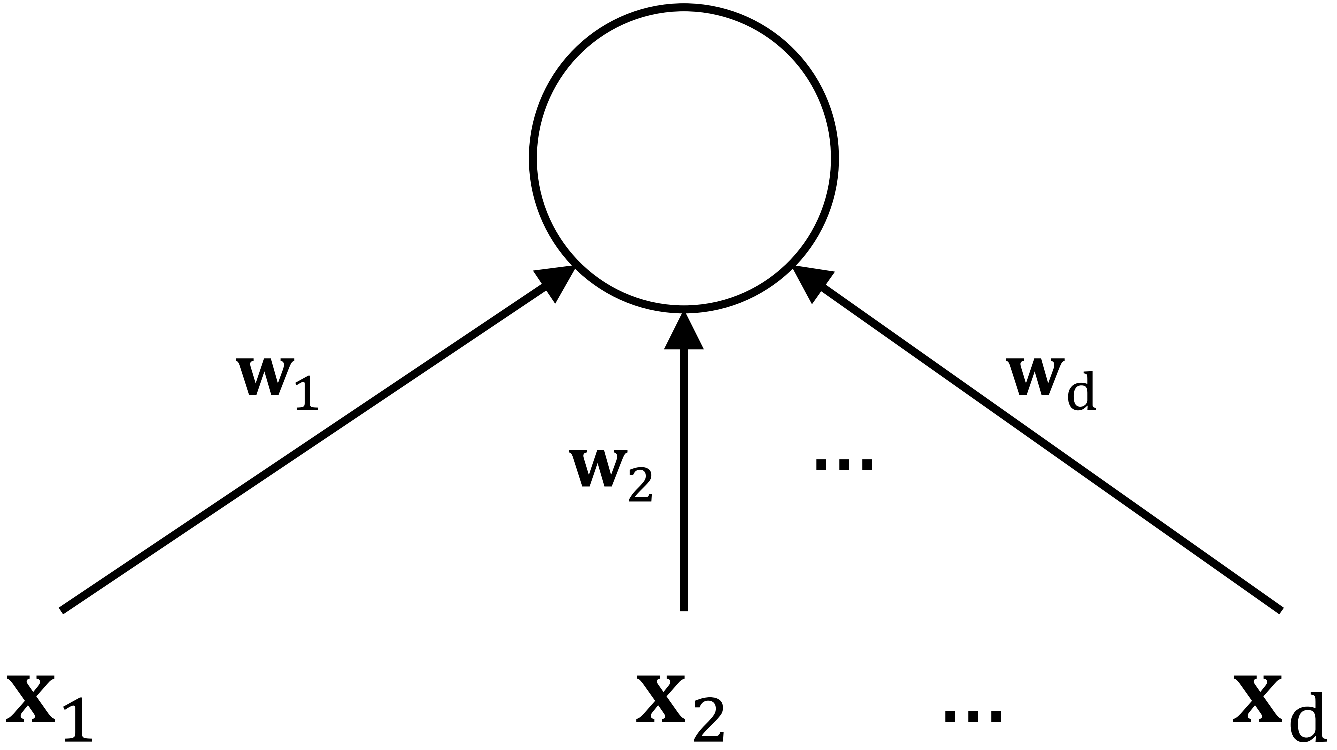
The “neuron” in Neural Networks



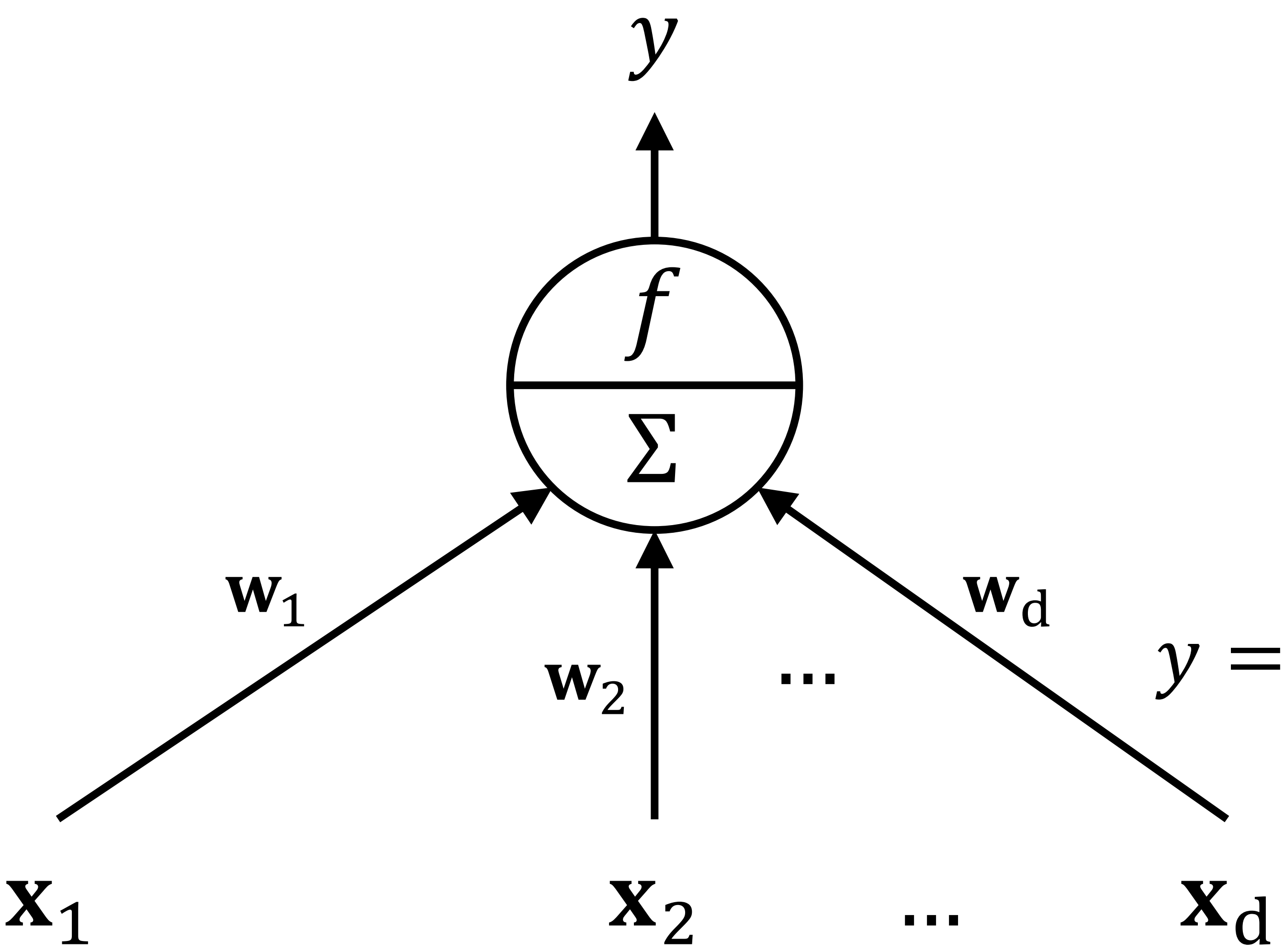
The “neuron” in Neural Networks



The “neuron” in Neural Networks

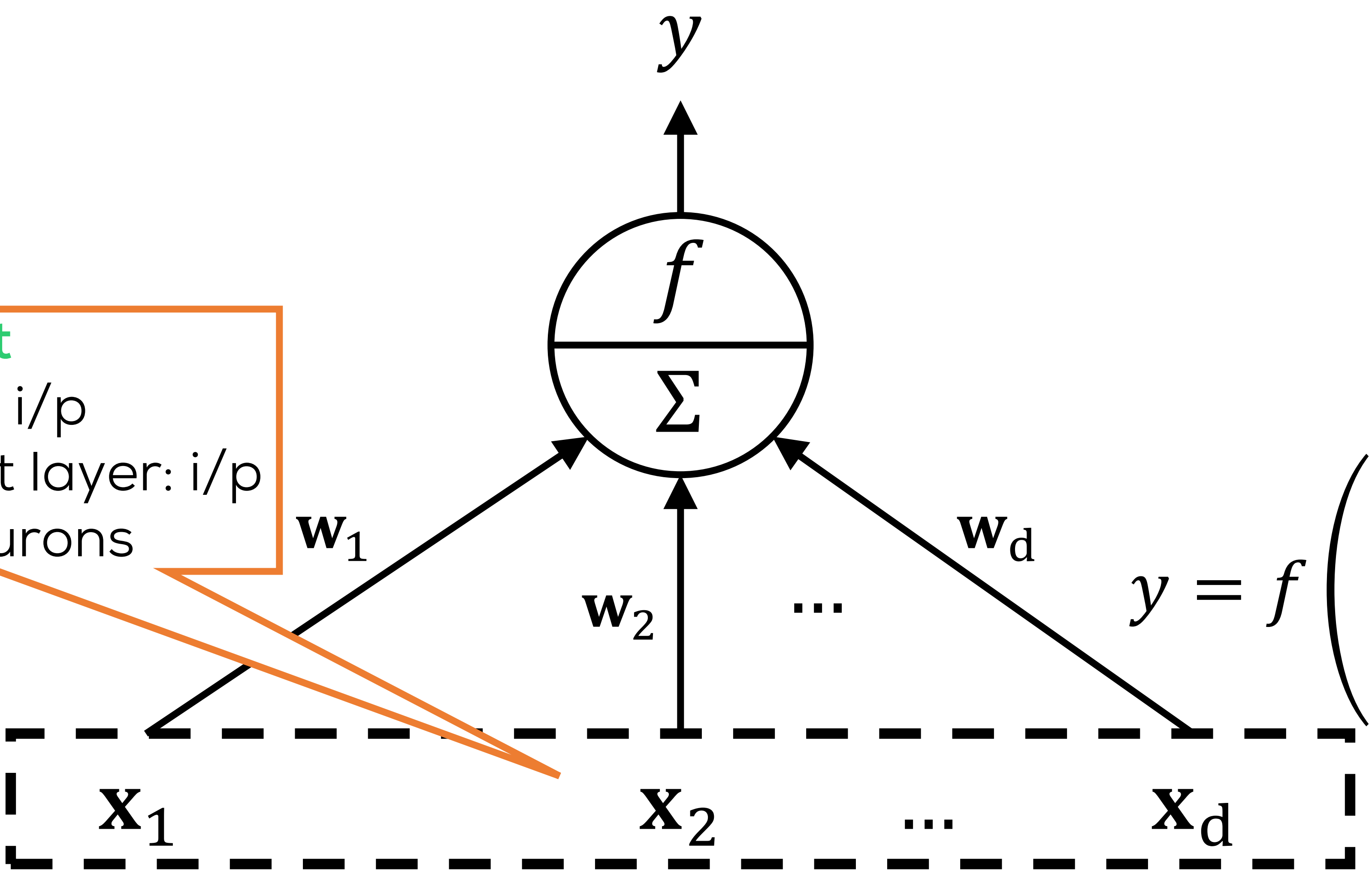


The “neuron” in Neural Networks



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

The "neuron" in Neural Networks



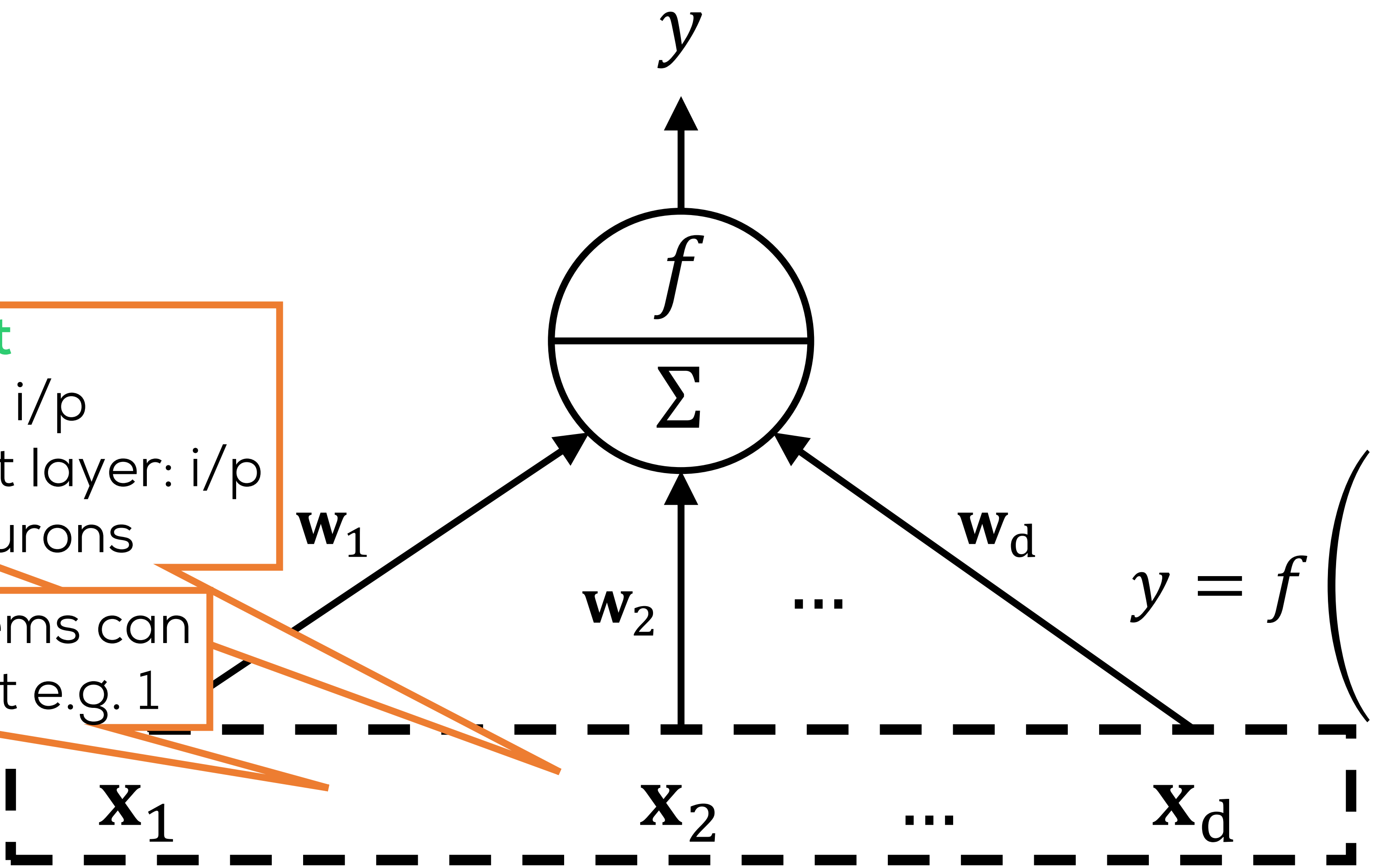
Input

Input layer: no i/p

Hidden/output layer: i/p from other neurons

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

The "neuron" in Neural Networks



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

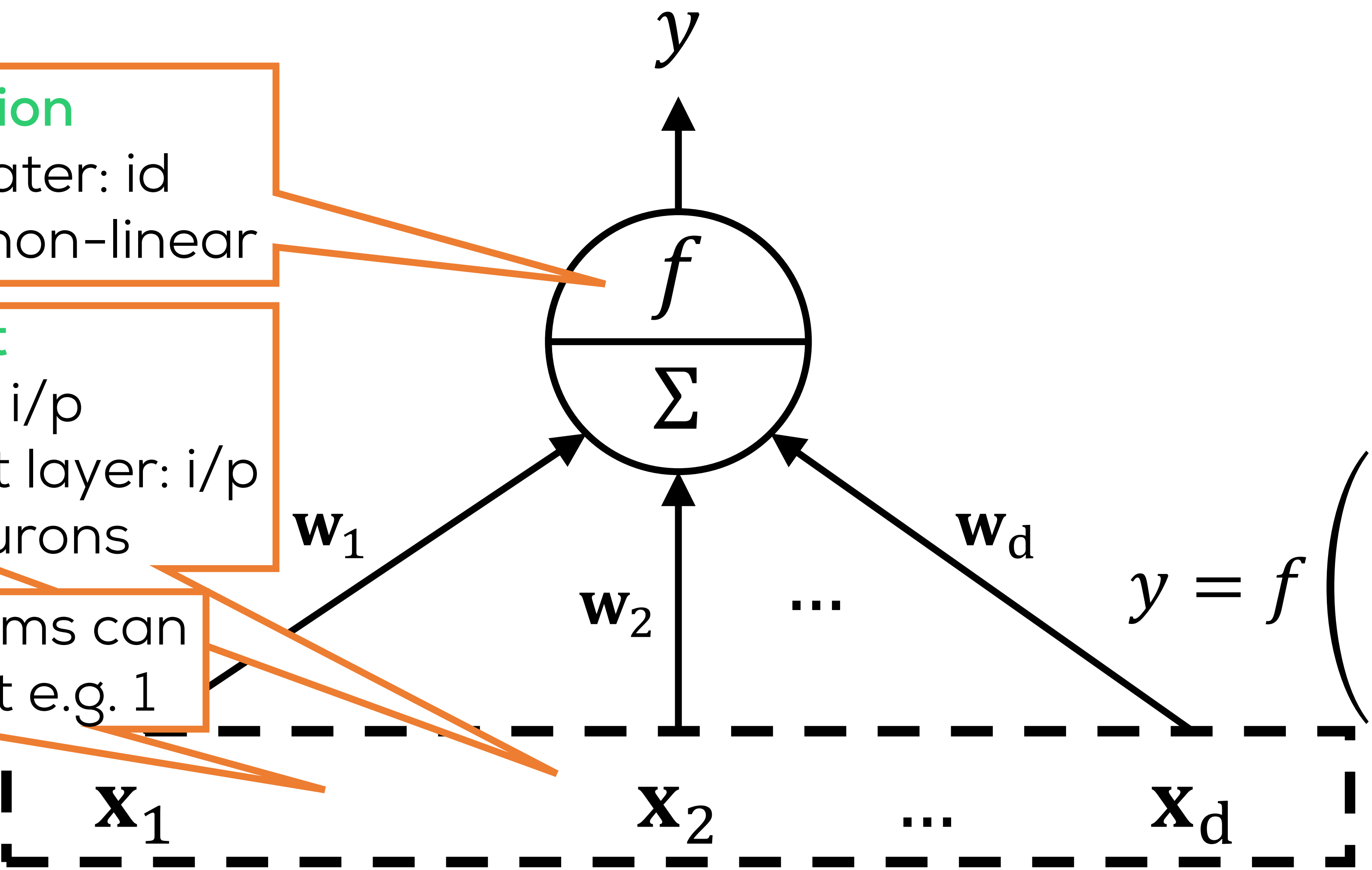
Input

Input layer: no i/p

Hidden/output layer: i/p from other neurons

Some input items can be a constant e.g. 1

The “neuron” in Neural Networks



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

Activation

Input/output later: id

Hidden layer: non-linear

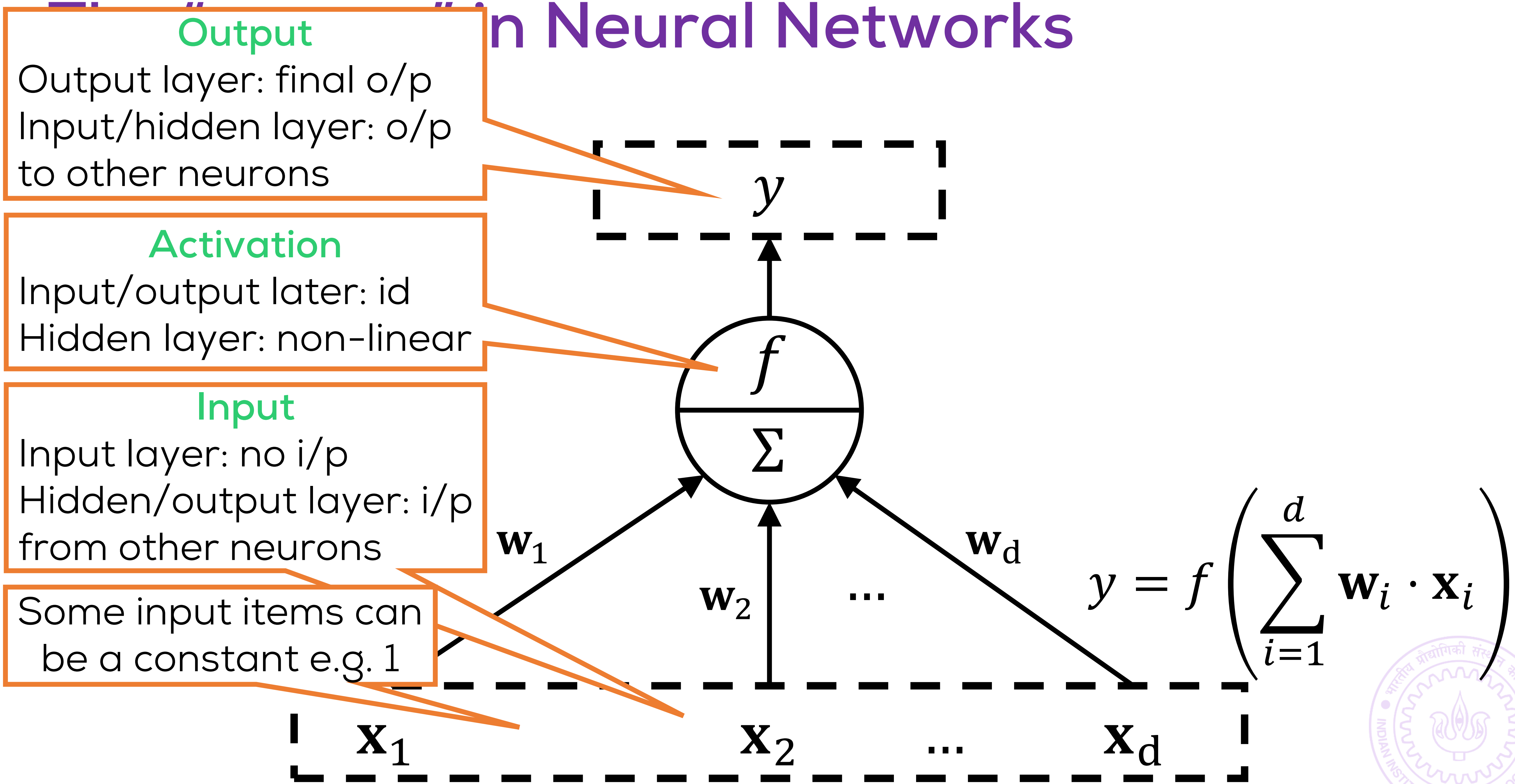
Input

Input layer: no i/p

Hidden/output layer: i/p from other neurons

Some input items can be a constant e.g. 1

Th“ ” n Neural Networks



Output

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

Activation

Input/output later: id
Hidden layer: non-linear

Input

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

Some input items can
be a constant e.g. 1

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

n Neural Networks

Output

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

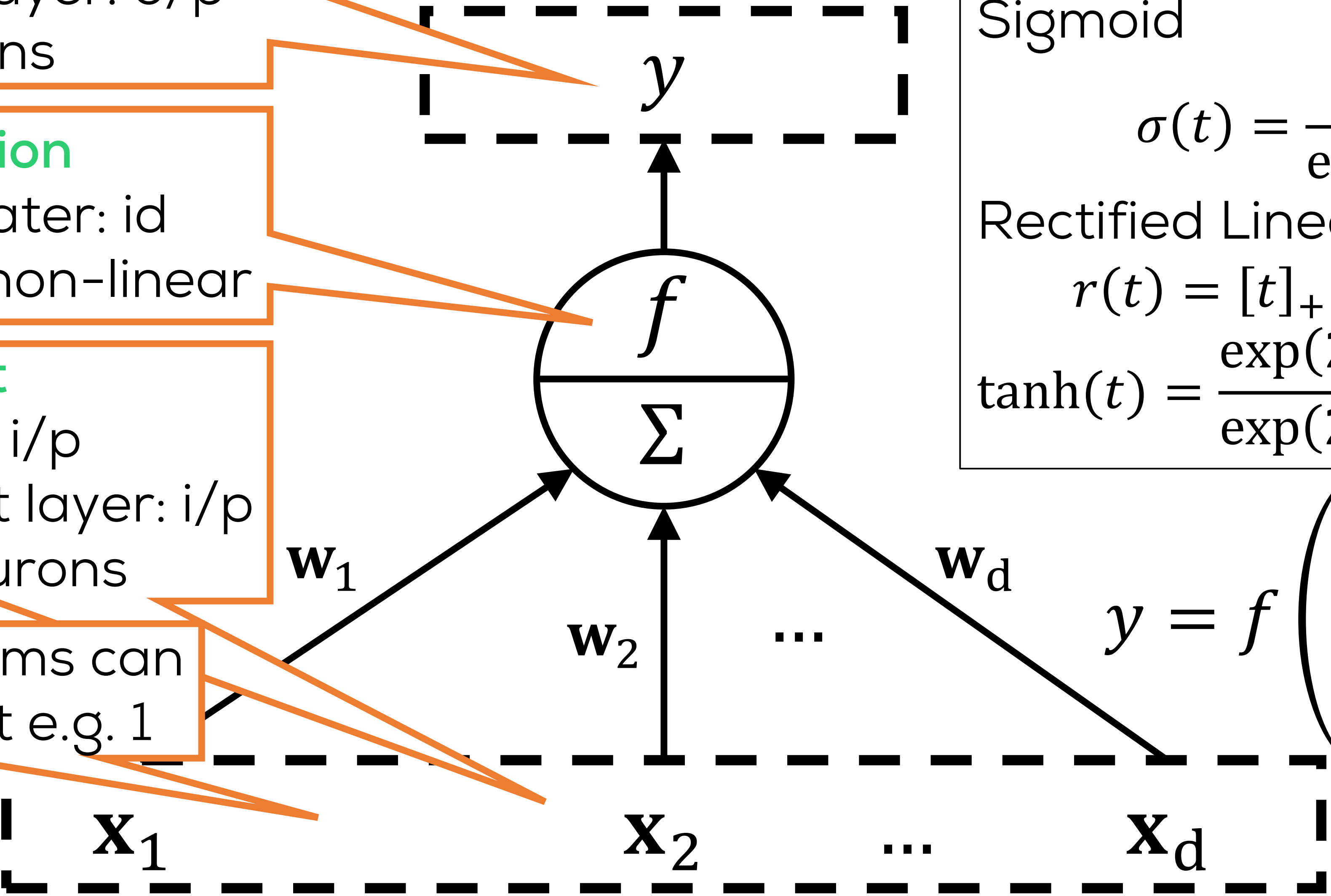
Activation

Input/output later: id
Hidden layer: non-linear

Input

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

Some input items can be a constant e.g. 1



Common "activation" fns f

Sigmoid

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

Rectified Linear Unit (ReLU)

$$r(t) = [t]_+ = \max(t,0)$$

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

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

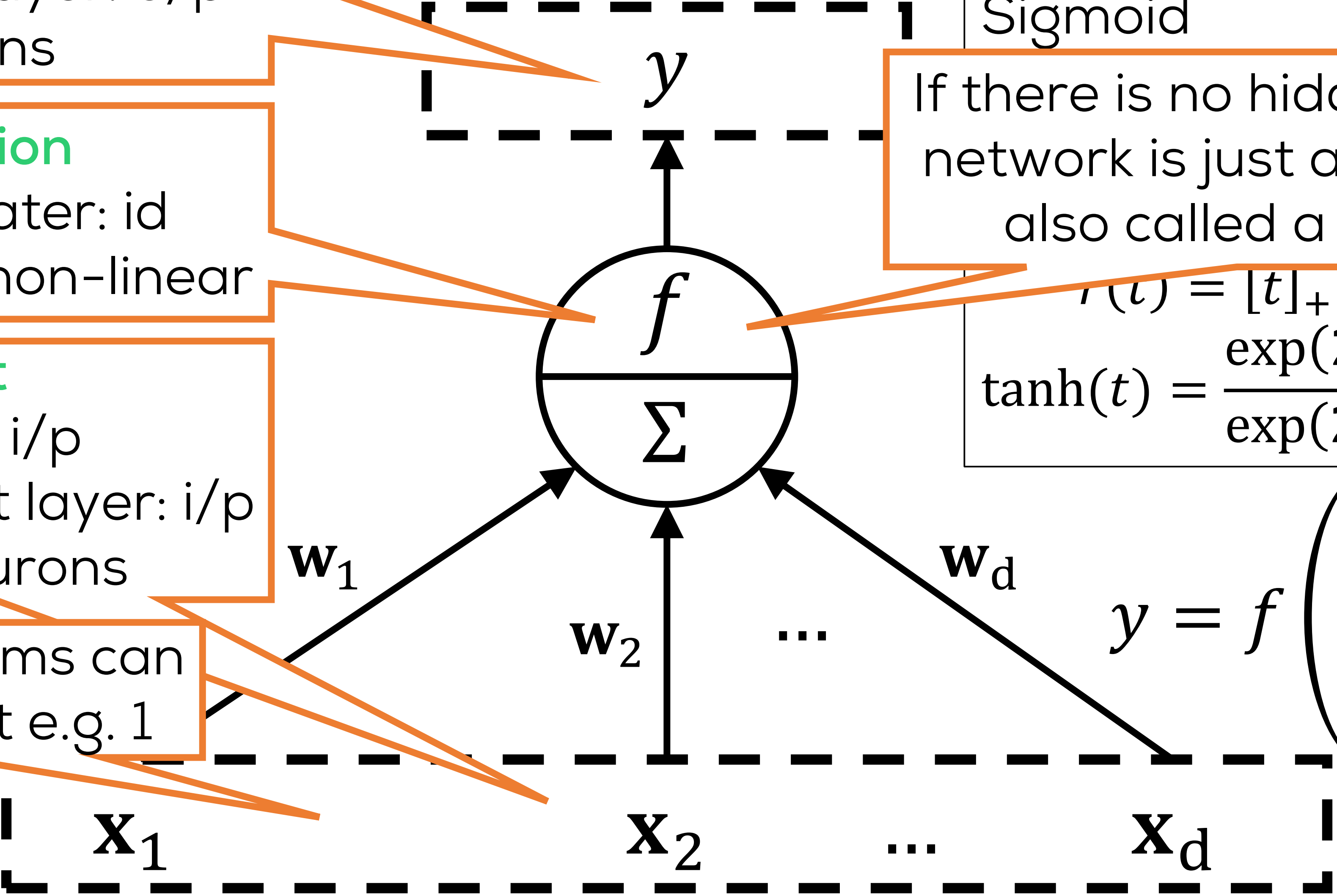
in Neural Networks

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

Activation
Input/output later: id
Hidden layer: non-linear

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

Some input items can be a constant e.g. 1



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

Common "activation" fns f

Sigmoid

$r(t) = [t]_+ = \max(t, 0)$

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

If there is no hidden layer then network is just a linear model, also called a **percepton**

Th “ ” i orks

Output

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

Sometimes output layer is given a non-id activation. Matter of convention

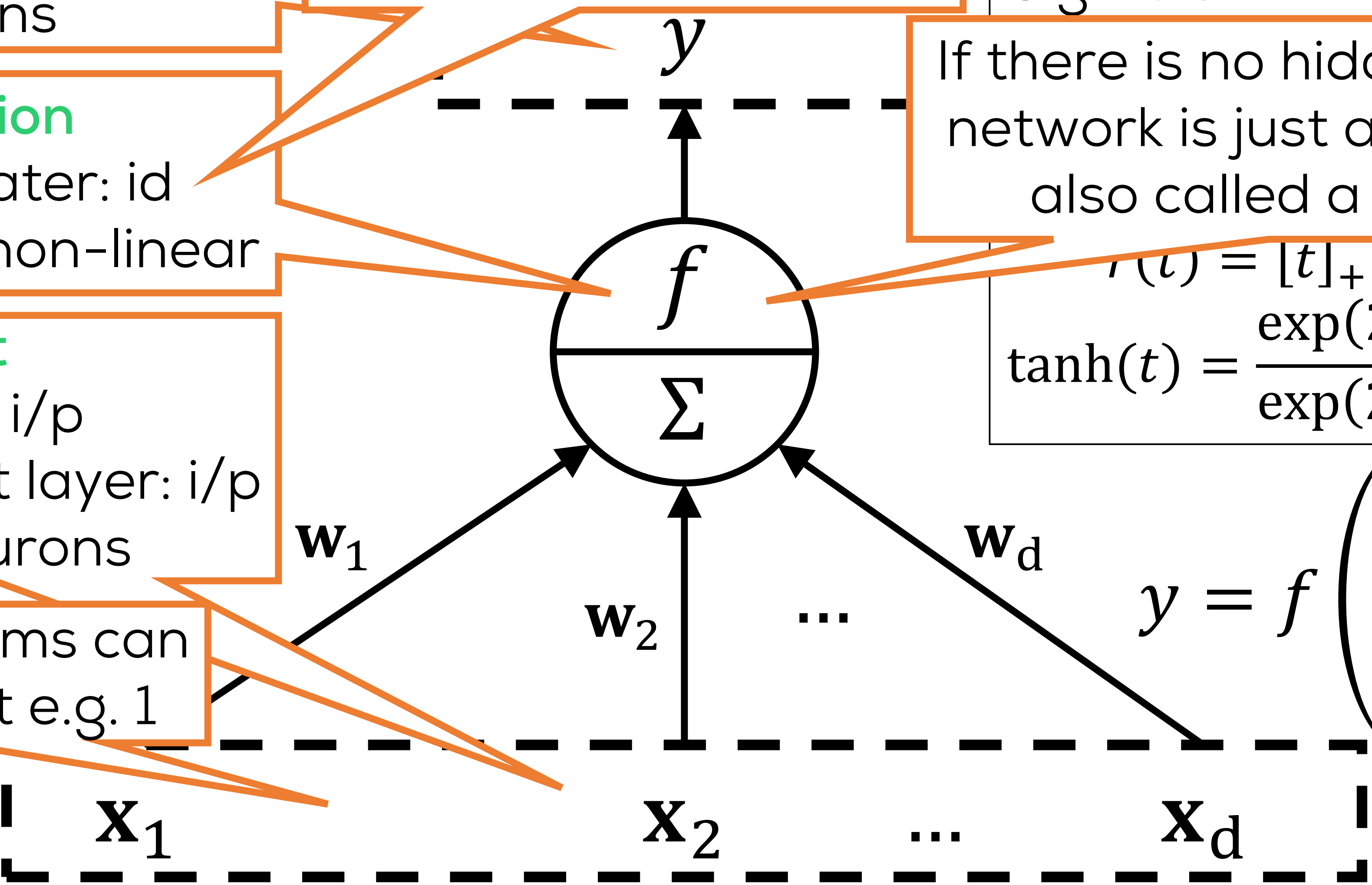
Activation

Input/output later: id
Hidden layer: non-linear

Input

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

Some input items can be a constant e.g. 1



Common “activation” fns f

Sigmoid

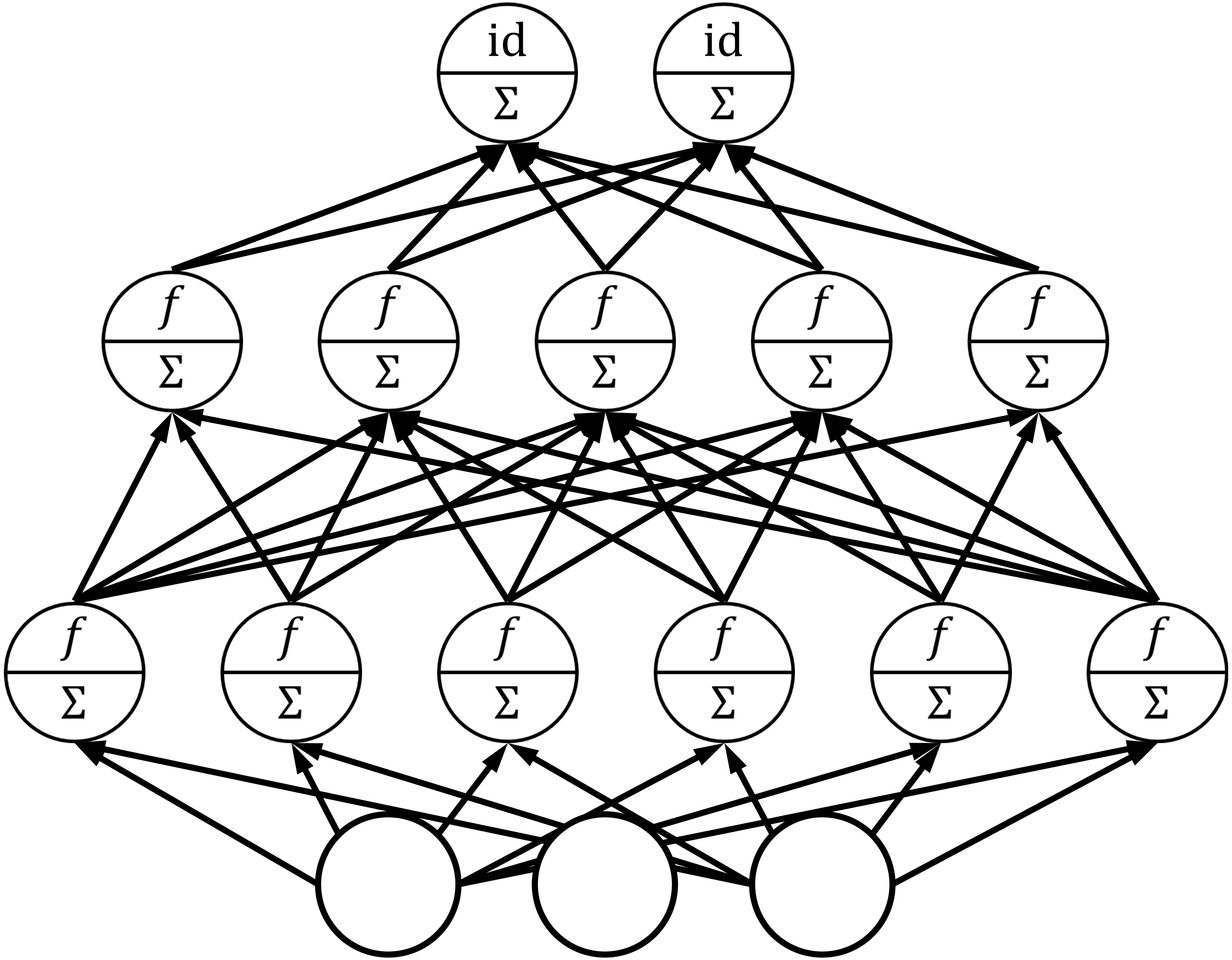
$f(t) = [t]_+ = \max(t, 0)$

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

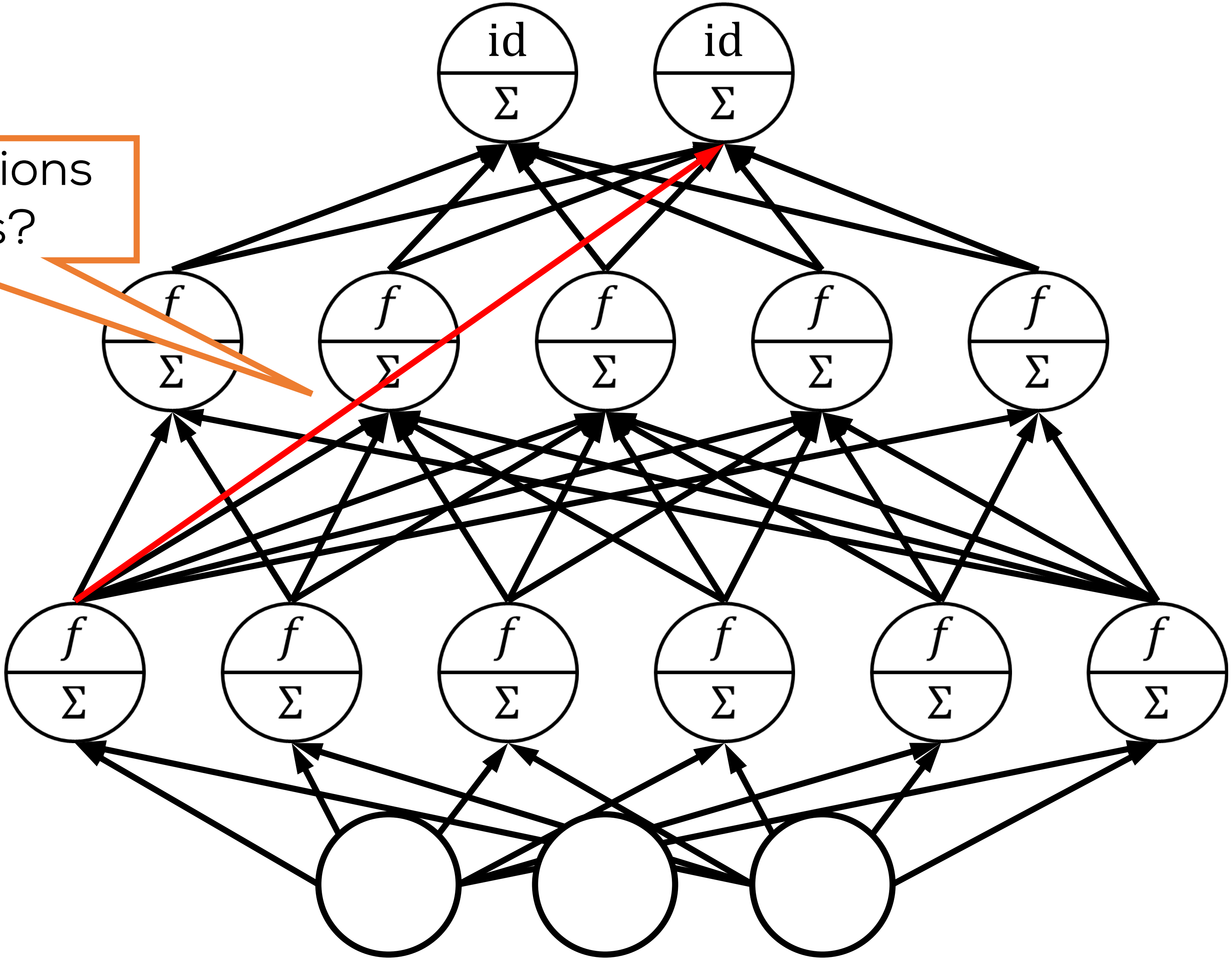
If there is no hidden layer then network is just a linear model, also called a **percepton**

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

A Feedforward Network

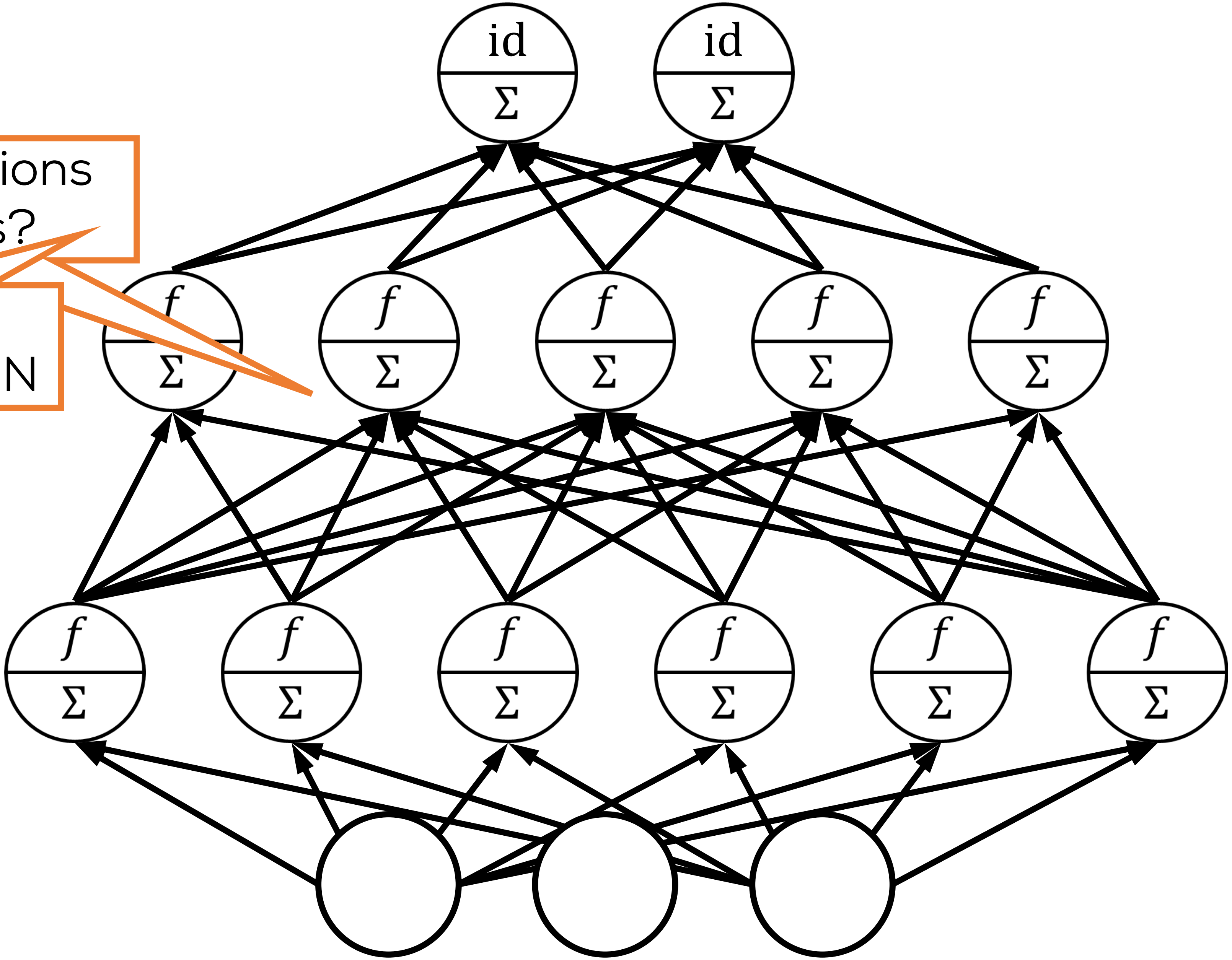


A Feedforward Network



Can I connections jump layers?

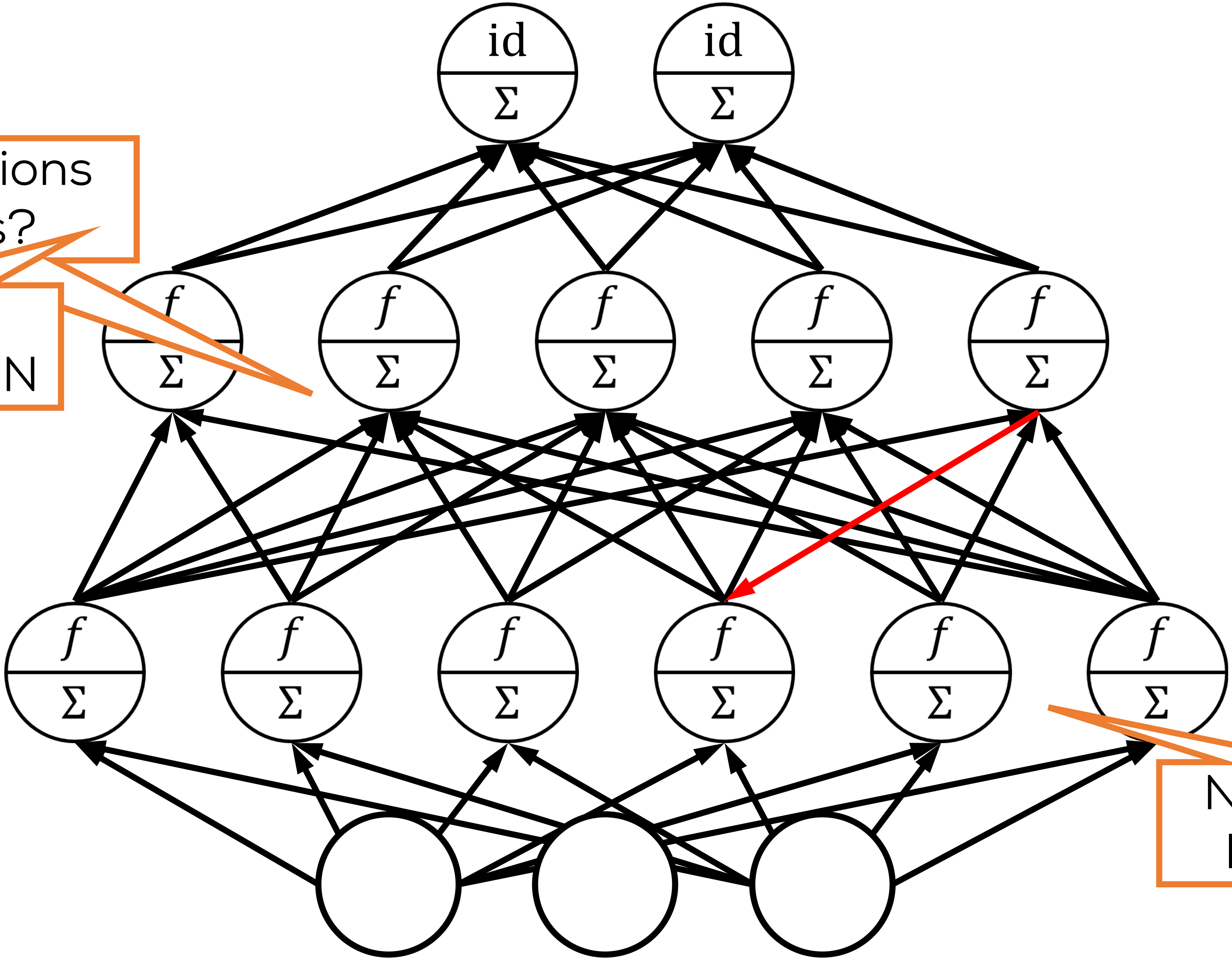
A Feedforward Network



Can I connections jump layers?

Not in a feedforward NN

A Feedforward Network

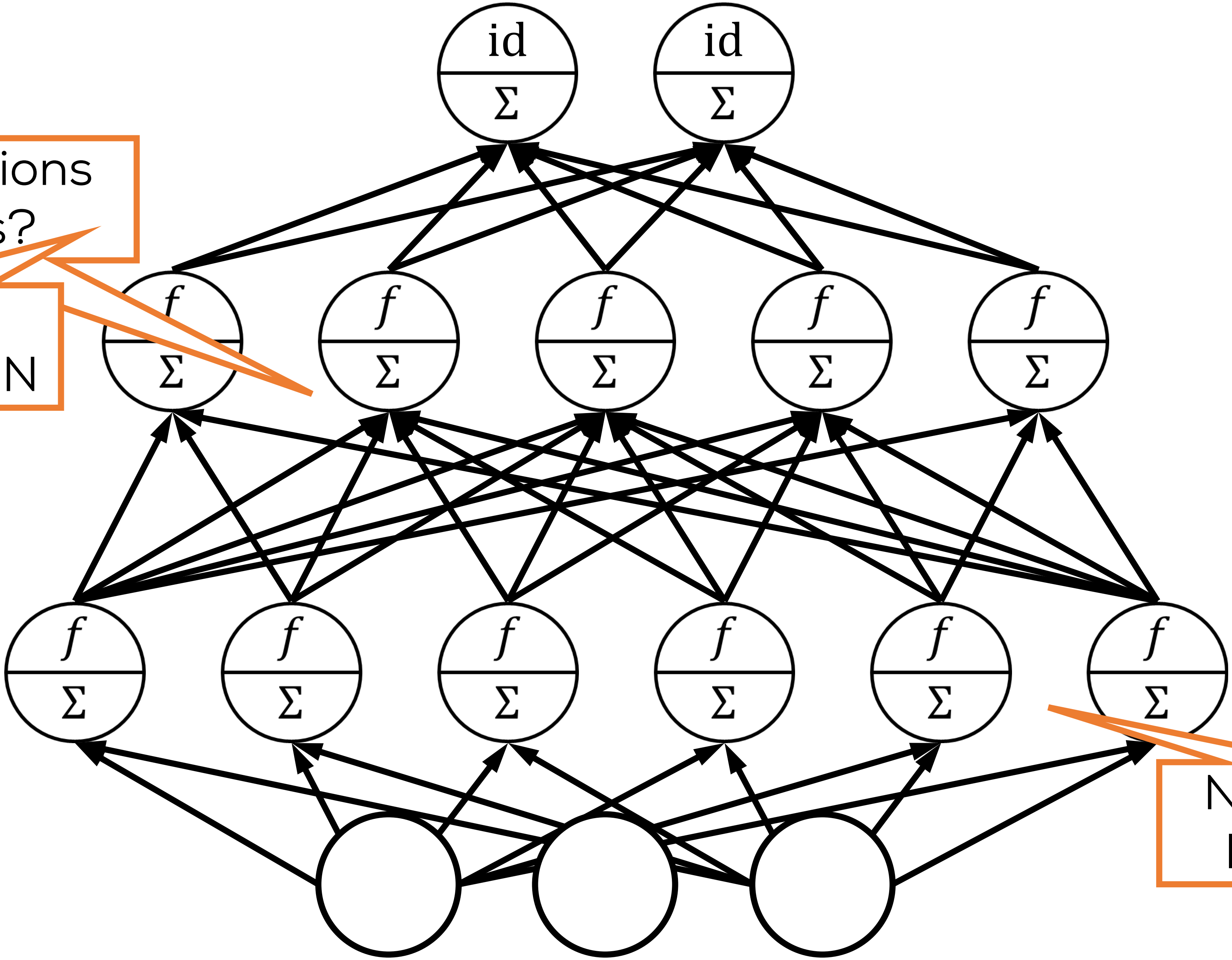


Can I connections jump layers?

Not in a feedforward NN

No "reverse" links either

A Feedforward Network

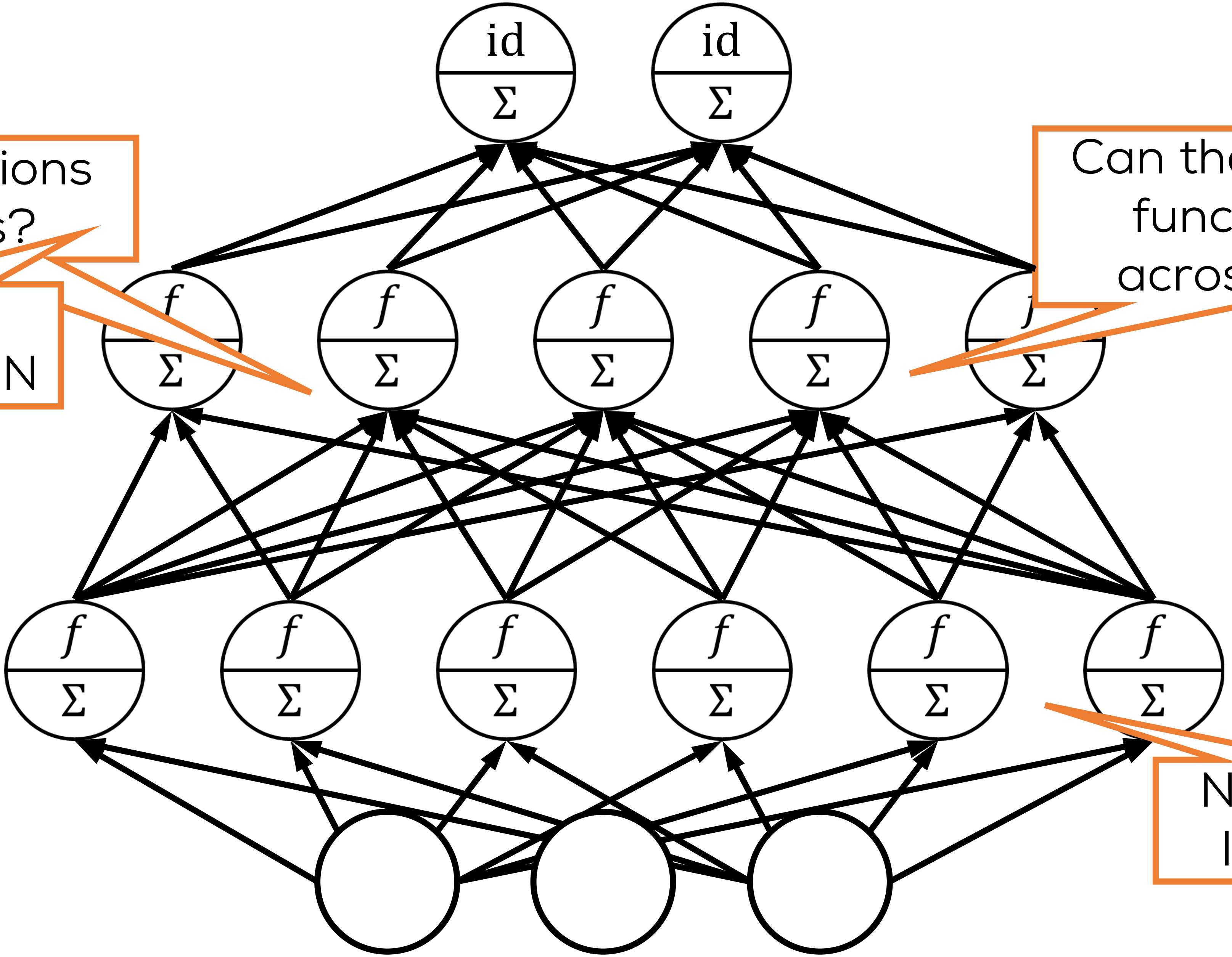


Can I connections jump layers?

Not in a feedforward NN

No “reverse” links either

A Feedforward Network



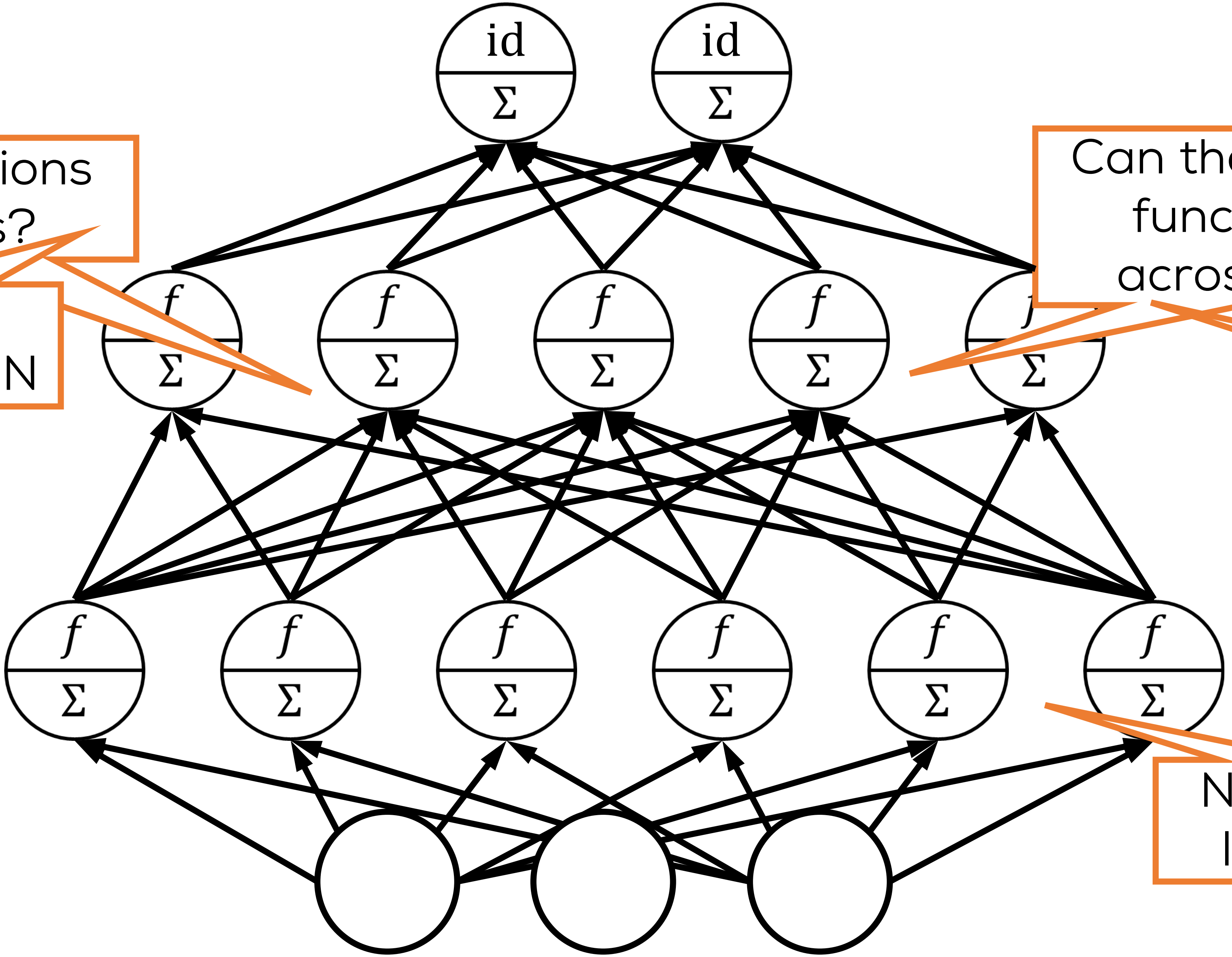
Can I connections jump layers?

Not in a feedforward NN

Can the activation function vary across layers?

No “reverse” links either

A Feedforward Network



Can I connections jump layers?

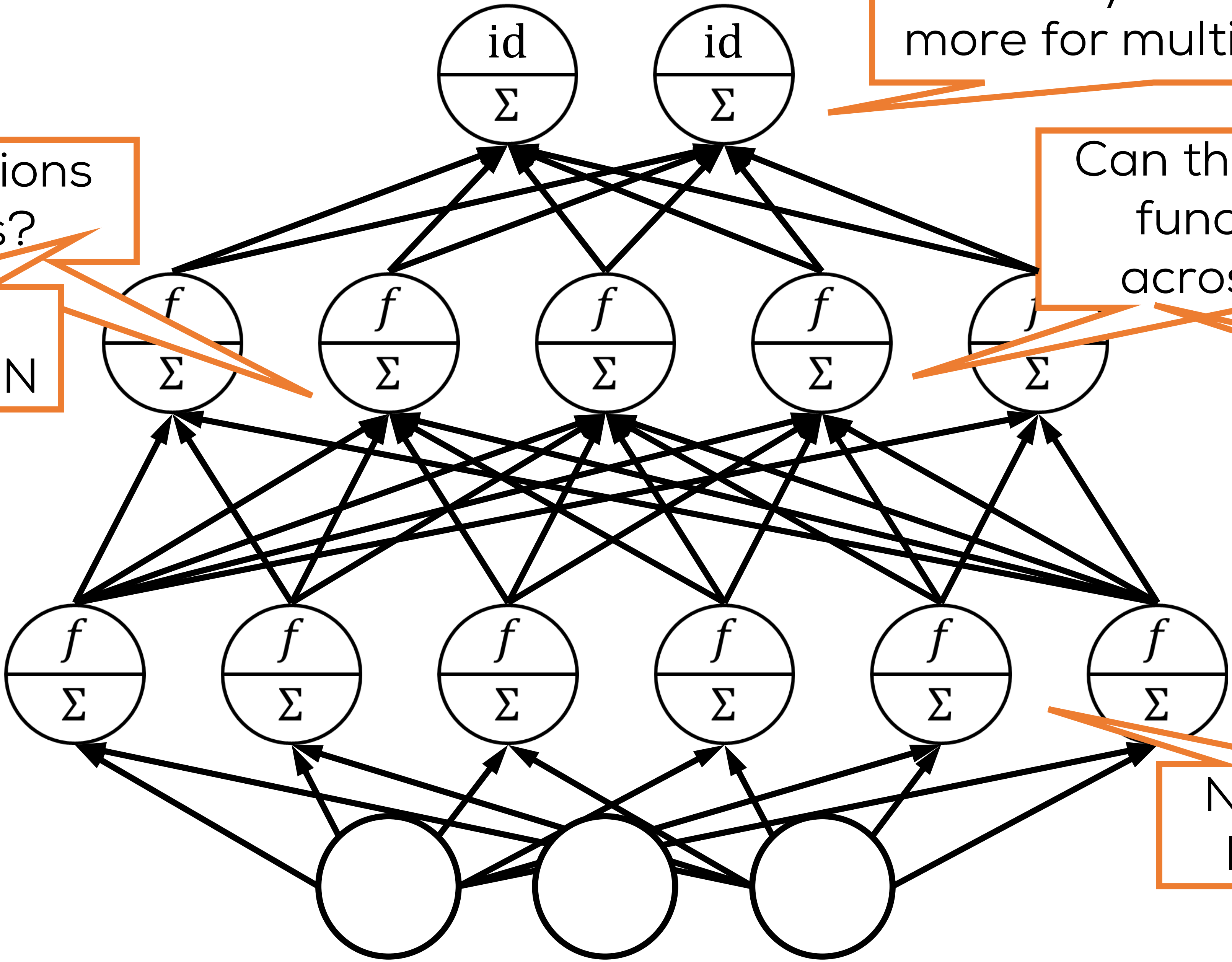
Not in a feedforward NN

Can the activation function vary across layers?

Yes. Wait for CNNs

No "reverse" links either

A Feedforward Network



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

Can I connections jump layers?

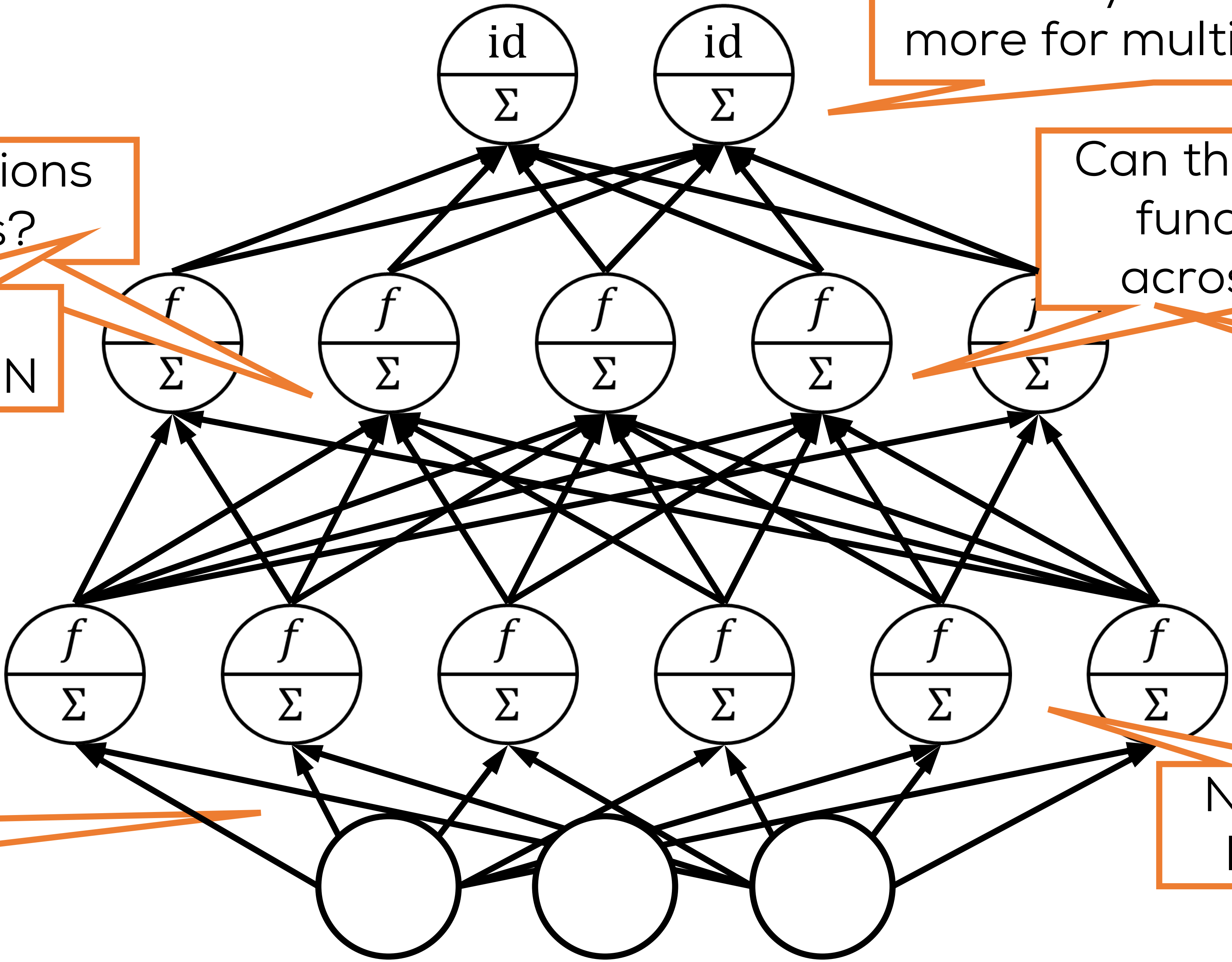
Not in a feedforward NN

Can the activation function vary across layers?

Yes. Wait for CNNs

No "reverse" links either

A Feedforward Network



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

Can I connections jump layers?

Not in a feedforward NN

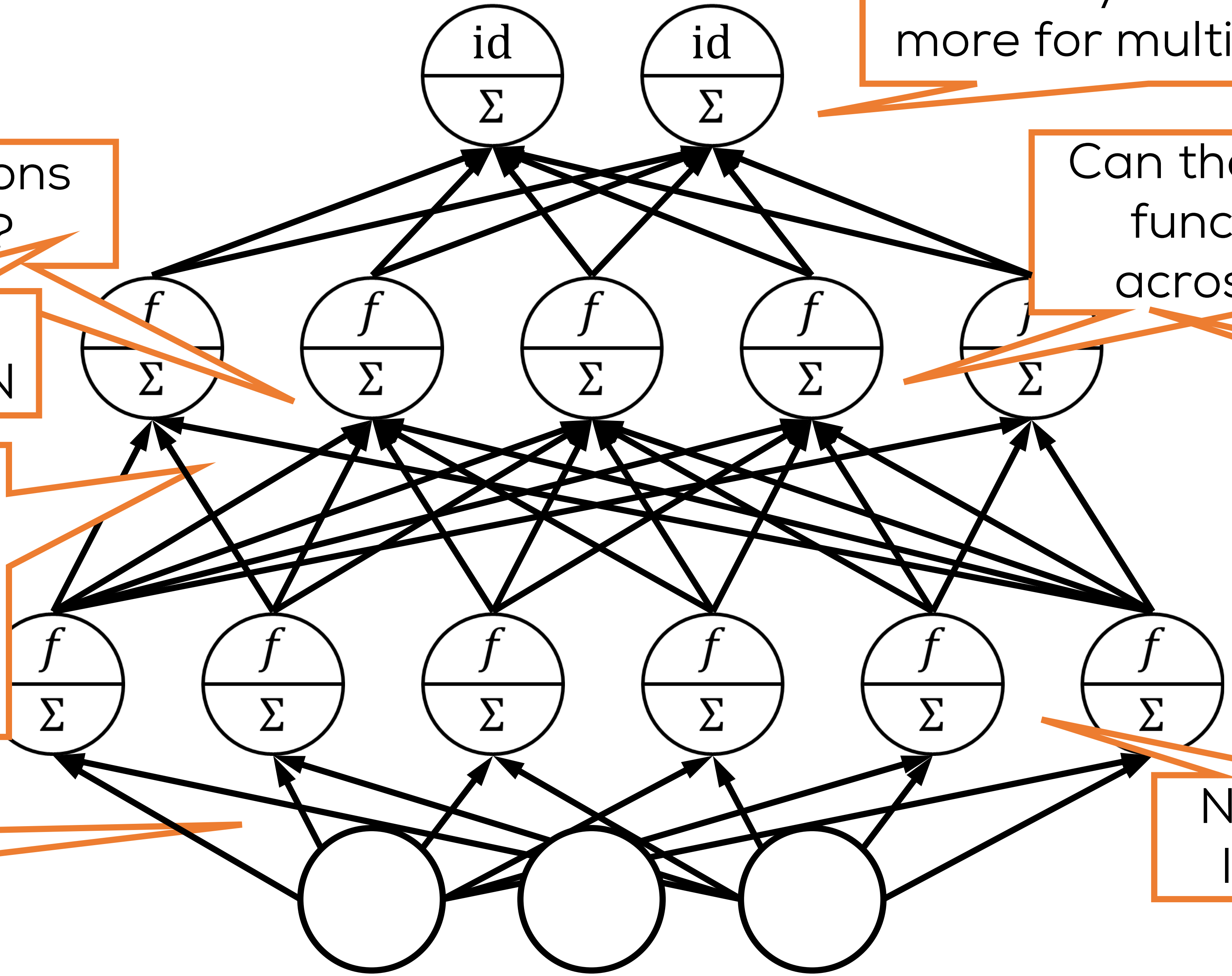
Can the activation function vary across layers?

Yes. Wait for CNNs

All weights are learnt

No "reverse" links either

A Feedforward Network



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

Can I connections jump layers?

Not in a feedforward NN

Can the activation function vary across layers?

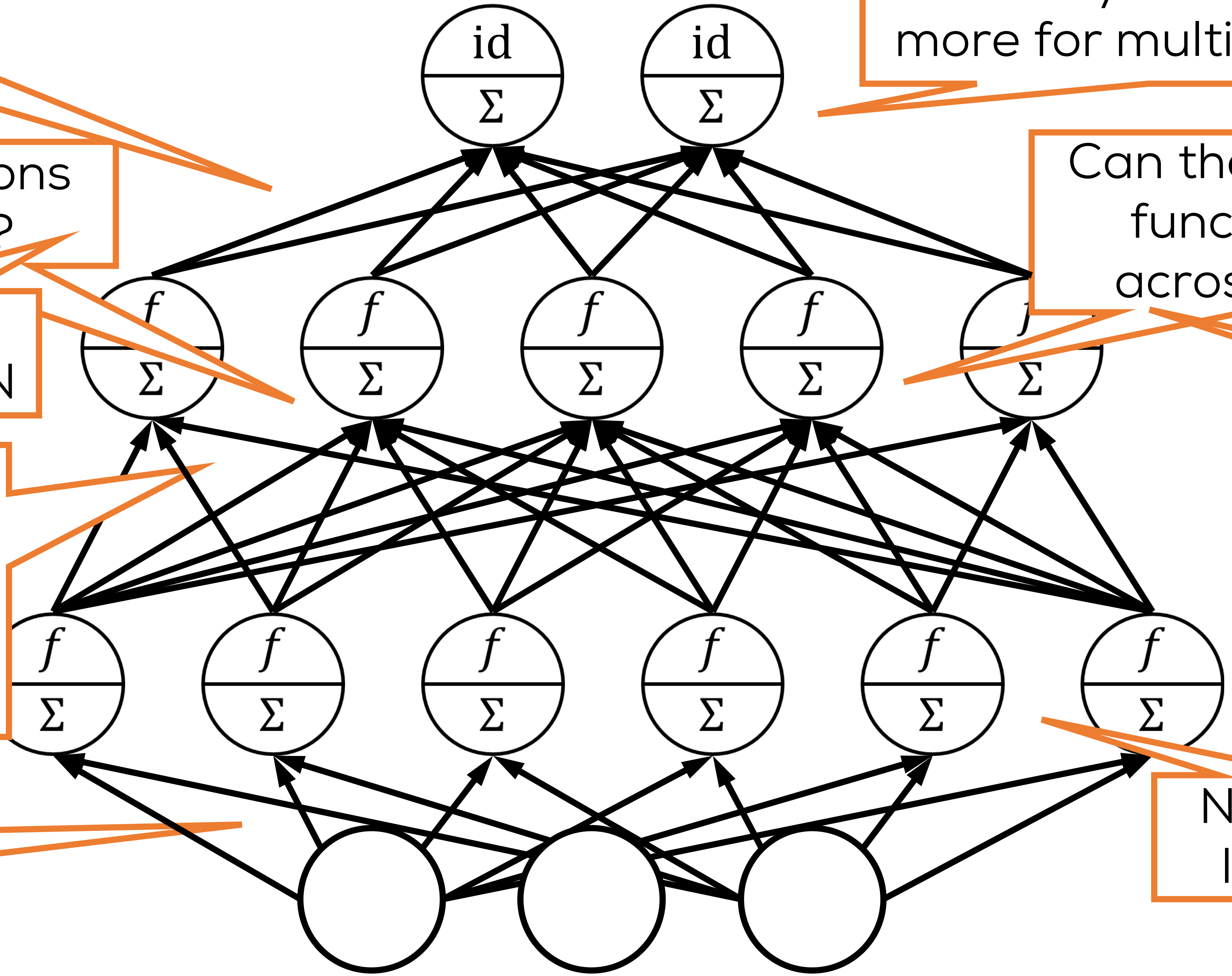
Yes. Wait for CNNs

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

All weights are learnt

No “reverse” links either

A Feedforward Network



Multi-layered **perceptron**

Can I connections jump layers?

Not in a feedforward NN

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

All weights are learnt

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

Can the activation function vary across layers?

Yes. Wait for CNNs

No “reverse” links either

A few thoughts

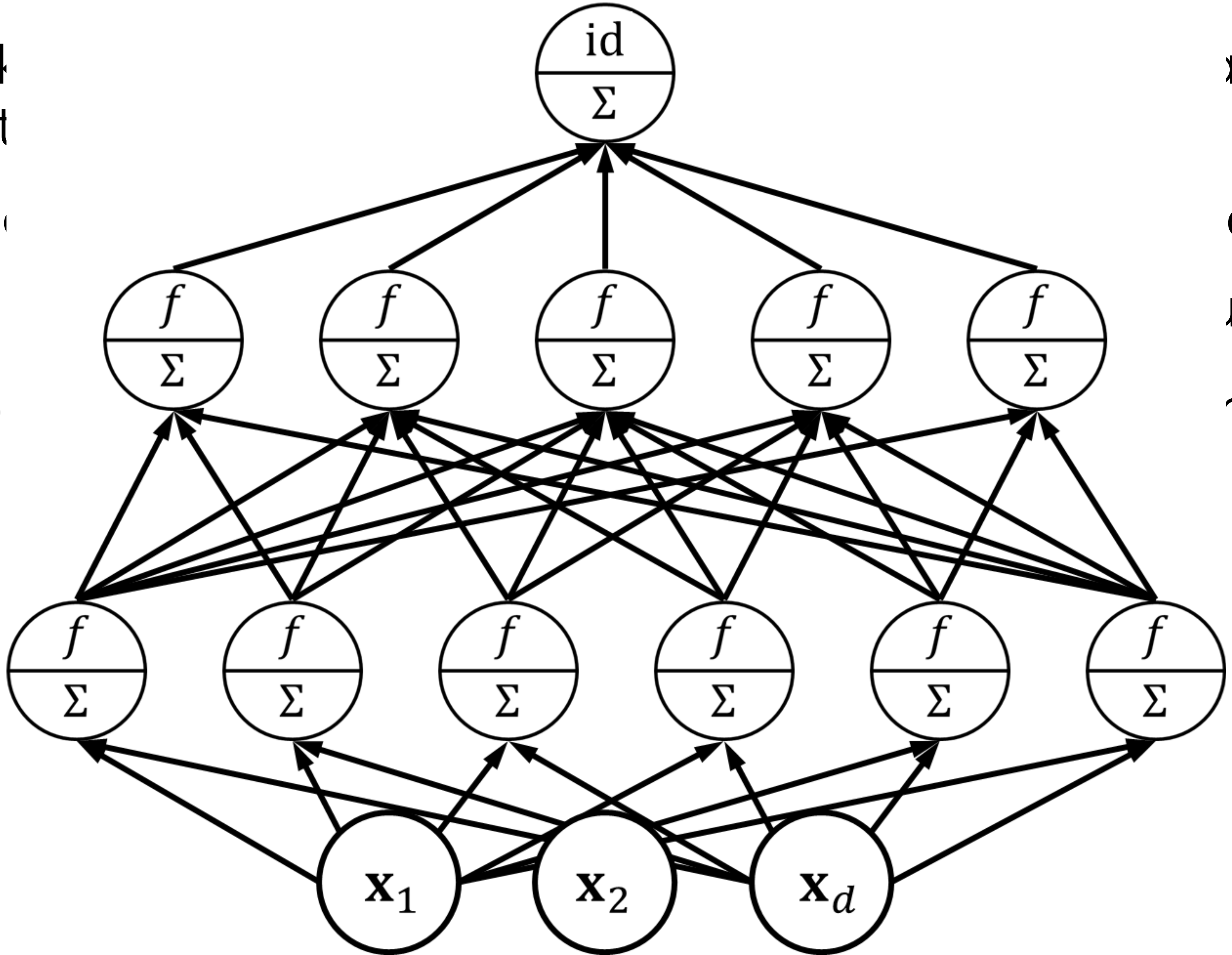
- Just as in kernel slide, lower layers can be interpreted as working very hard to compute useful and informative features
- Last layer exploits all this hard work to learn a good model

  $y = \sum_{i=1}^{k_l} \mathbf{w}_i \cdot \phi(\mathbf{x}_i)$, k_l = #nodes in layer previous to output layer
- Note: output is linear in the features computed by lower layers

A few thoughts

- Just as in k as working
  very hard t
- Last layer del

  $y = \sum_{i=1}^{k_l} \mathbf{w}_i$ ut layer
- Note: outp layers

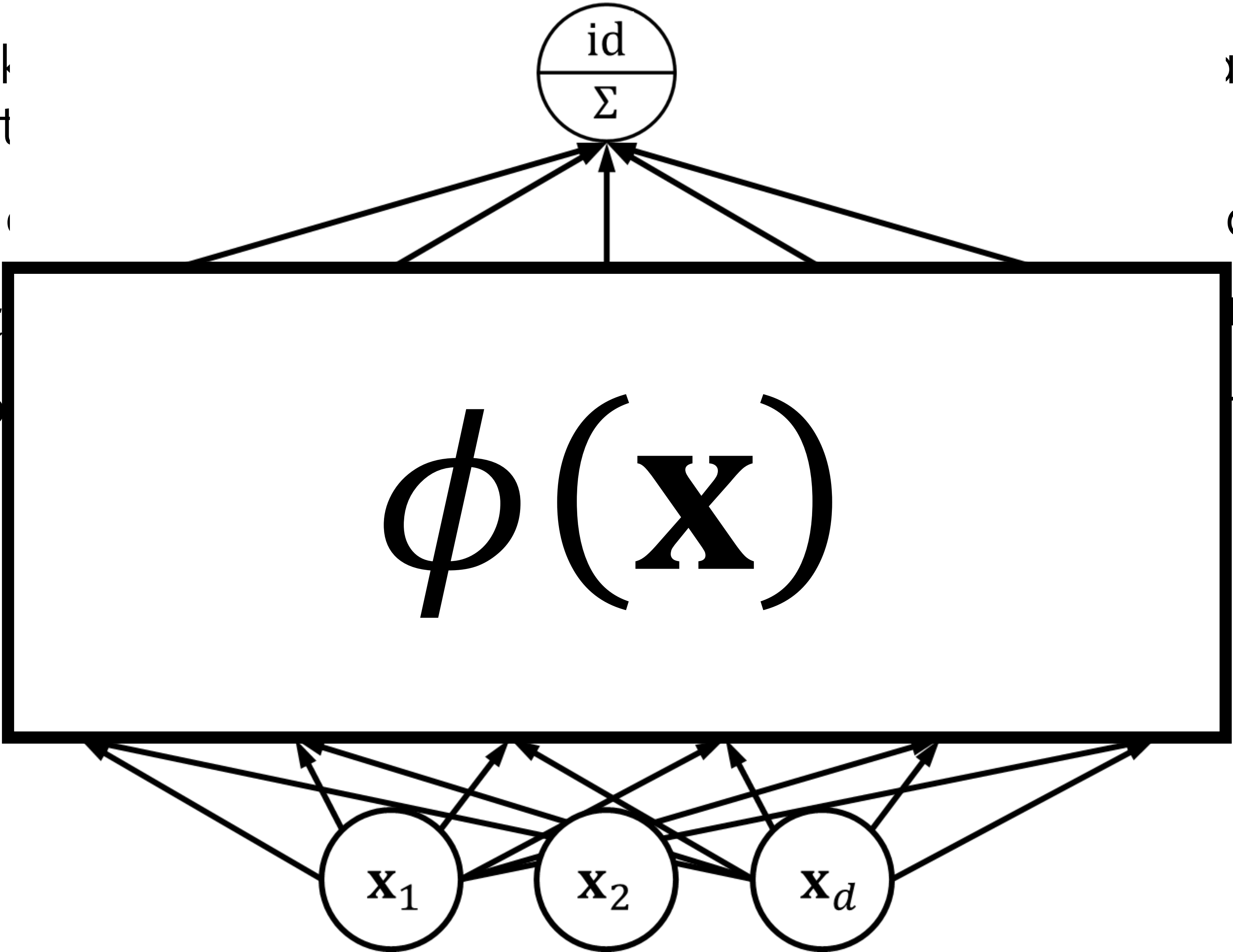


A few thoughts

- Just as in k
  very hard t

- Last layer
  $y = \sum_{i=1}^{k_l} \mathbf{w}_i$

- Note: outp



as working

del

ut layer

r layers

A few thoughts

- Just as in kernel slide, lower layers can be interpreted as working very hard to compute useful and informative features
- Last layer exploits all this hard work to learn a good model

  $y = \sum_{i=1}^{k_l} \mathbf{w}_i \cdot \phi(\mathbf{x}_i)$, k_l = #nodes in layer previous to output layer
- Note: output is linear in the features computed by lower layers
- Can have any no. of layers, any no. of nodes in each layer
- Having a linear activation function is useless since the entire network will then just learn a linear function in input
- ReLU networks always learn piecewise linear functions
- Try proving the above two results by induction on number of hidden layers (base case – no hidden layer) as an exercise

A few thoughts

- Kernel models work with a vast (often infinite) set of features. NN methods try to learn a small set of features from data itself
- Features are non-linear in kernels as well as NNs
- Why can't I have the nice operations of product, squaring, identity as "activation functions" as in the kernel slide?
- A variant called *Sum-product Networks* (SPN) does exactly this
- Neural networks are also *universal*
- A neural network with a single hidden layer with infinitely many nodes or else infinitely many layers each with finitely many nodes can learn any function of the input (details technical)
- Next class: how NNs are trained

Please give your Feedback

http://tinyurl.com/ml17-18afb