Artificial Neural Networks: Lecture 4

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Regularization and Tricks of the Trade in deep networks

Objectives for today:

- Bagging
- Dropout
- What are good units for hidden layers?
- Rectified linear unit (RELU)
- Shifted exponential linear (ELU and SELU)
- BackProp: Initialization
- Linearity problem, vanishing gradient problem, bias problem
- Batch normalization

Reading for this lecture:

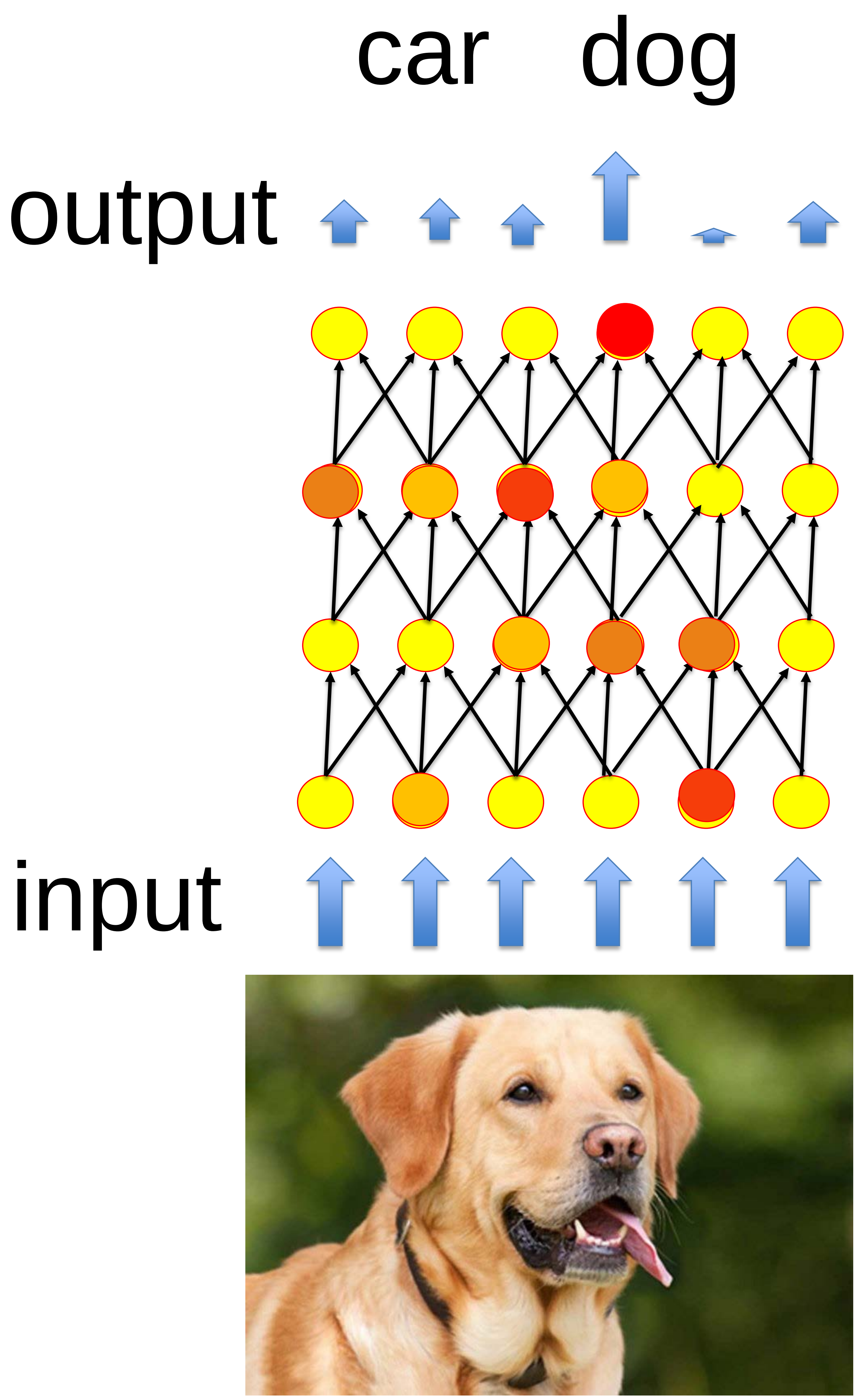
Goodfellow et al.,**2016** *Deep Learning*

- Ch 7.4, 7.8, **7.11 and 7.12,**
- **Ch. 8.4**

Further Reading for this Lecture:

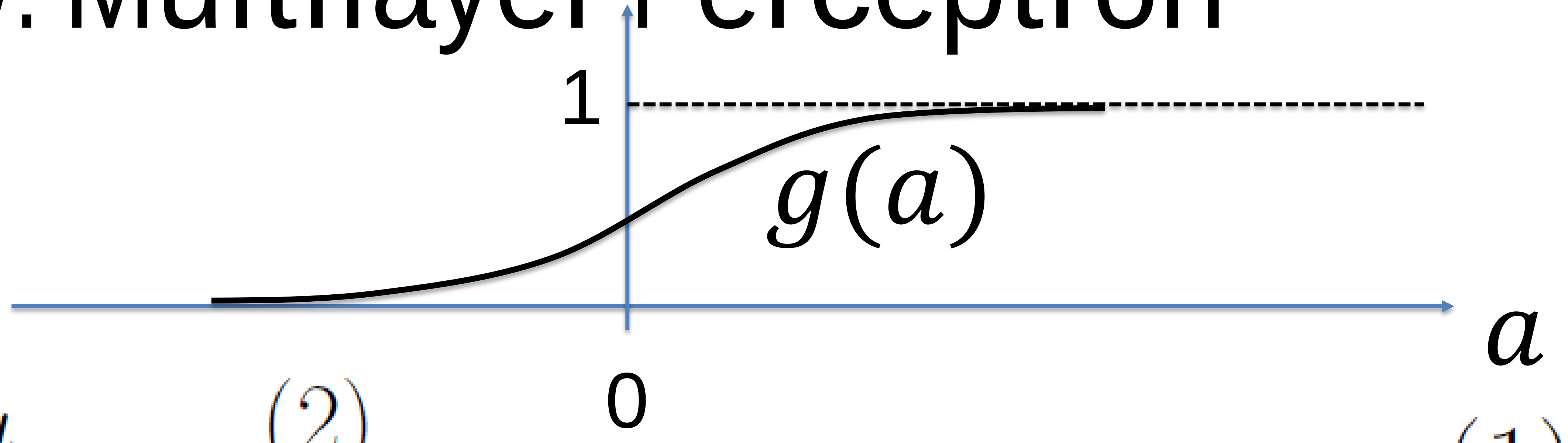
Paper: Klaumbauer, …, Hochreiter (2017)
Self-normalizaing neural networks
https://arxiv.org/pdf/1706.02515.pdf

review: Artificial Neural Networks for classification

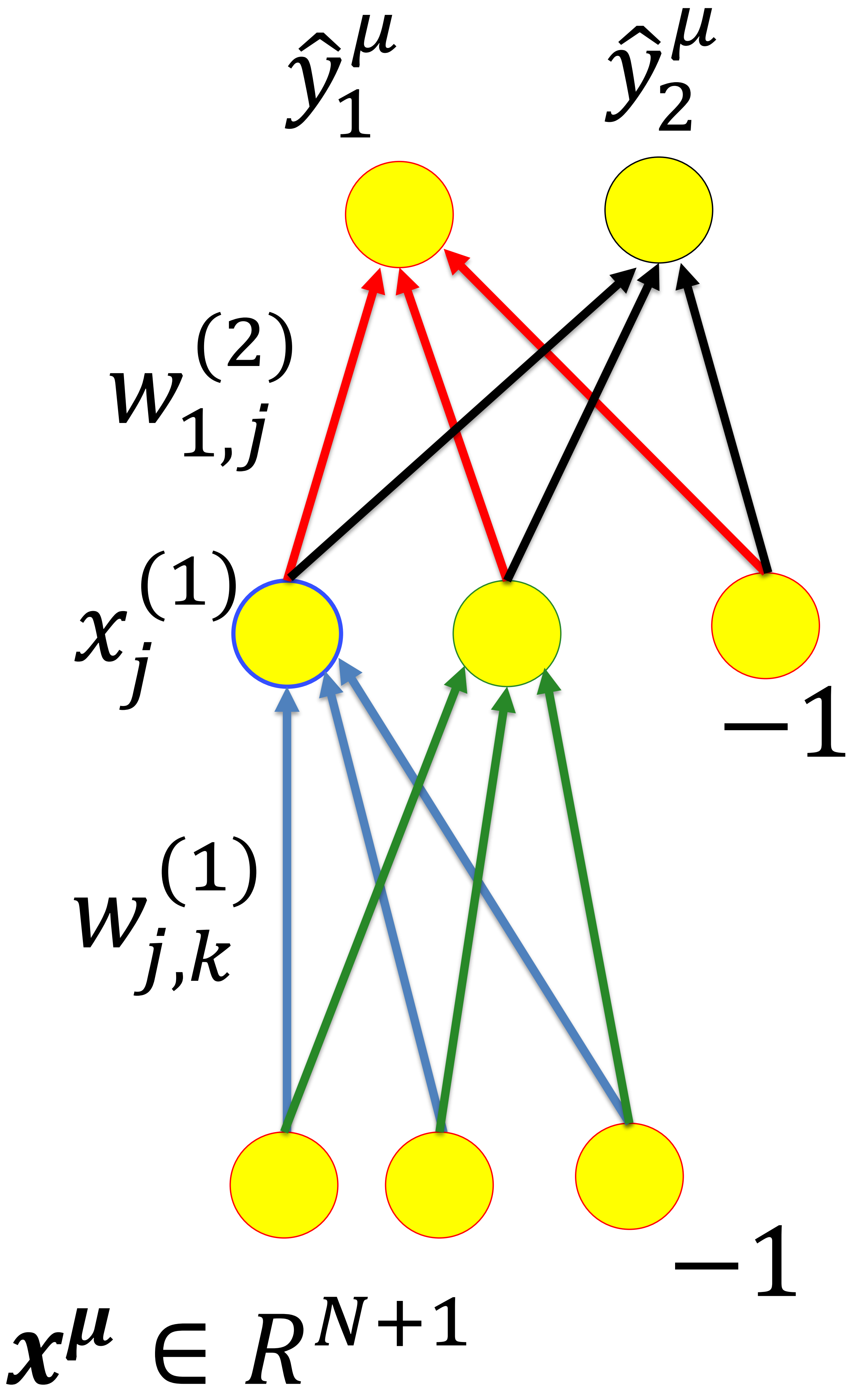


Aim of learning:
Adjust connections such that output is correct (for each input image, **even new ones**)

Review: Multilayer Perceptron



$$
\begin{aligned}
\hat{y}_i^{\mu} &= x_i^{(2)} && (1)\\
&= g^{(2)}[a_i^{(2)}] && (2)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} x_j^{(1)}] && (3)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(a_j^{(1)})] && (4)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(\sum_k w_{jk}^{(1)} x_k^{\mu})] && (5)
\end{aligned}
$$



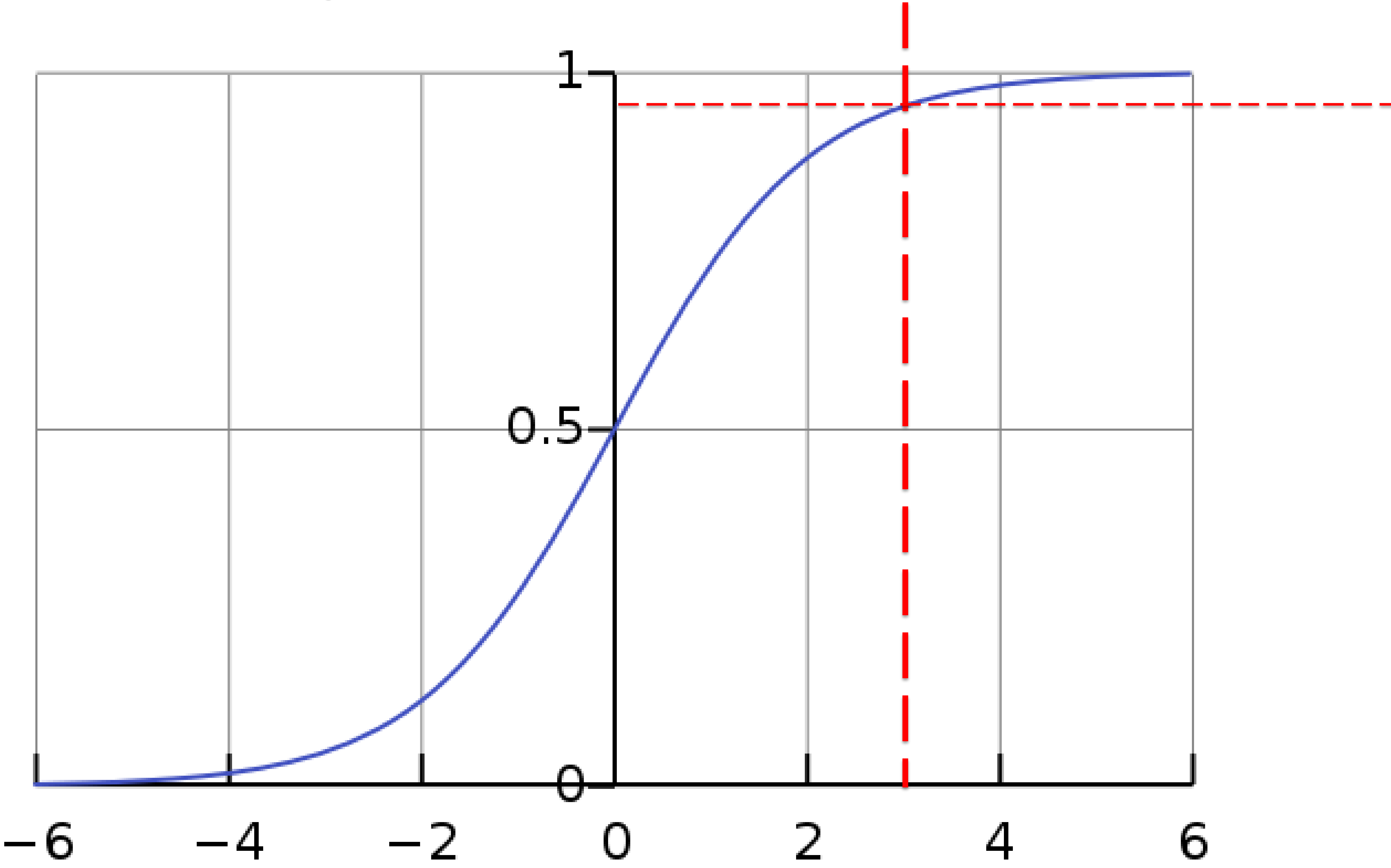
Review. sigmoidal output = **logistic function**

$$g(a) = \frac{1}{1+e^{-a}}$$

Rule of thumb:

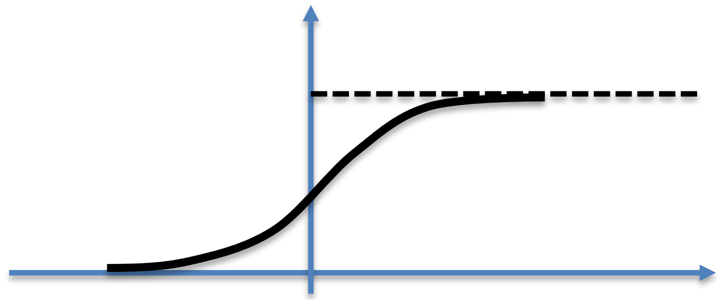
for $a = 3$: $g(3) = 0.95$

for $a = -3$: $g(-3) = 0.05$



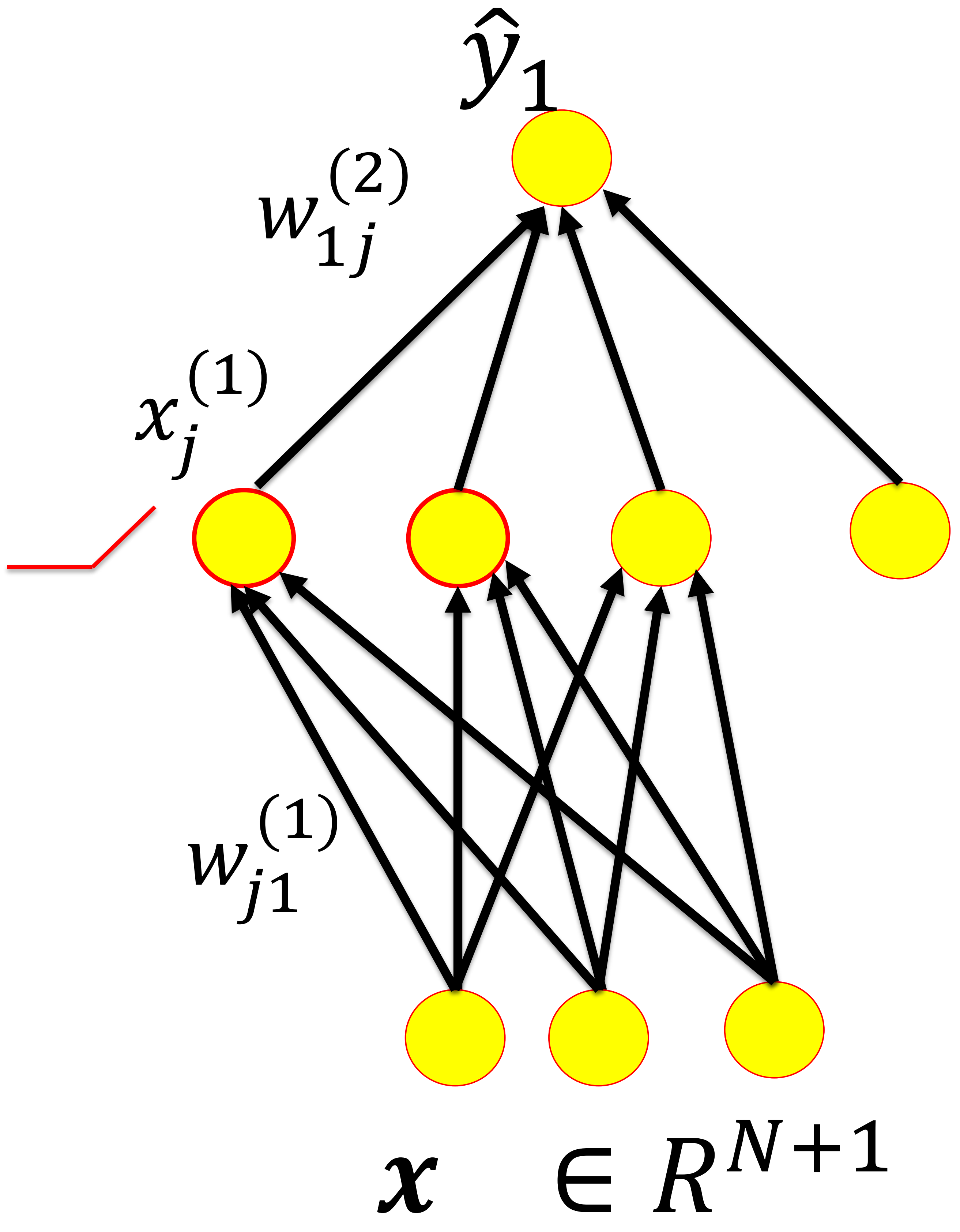
https://en.wikipedia.org/wiki/Logistic_function

Review: Modern Neural Networks



output layer

use sigmoidal unit (single-class)
or softmax (exclusive mutlti-class)



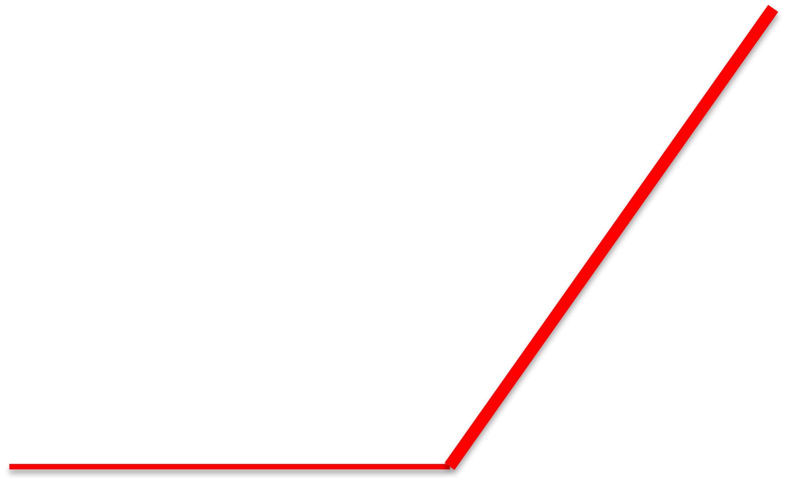
hidden layer

Why?

use rectified linear unit in $N+1$ dim.

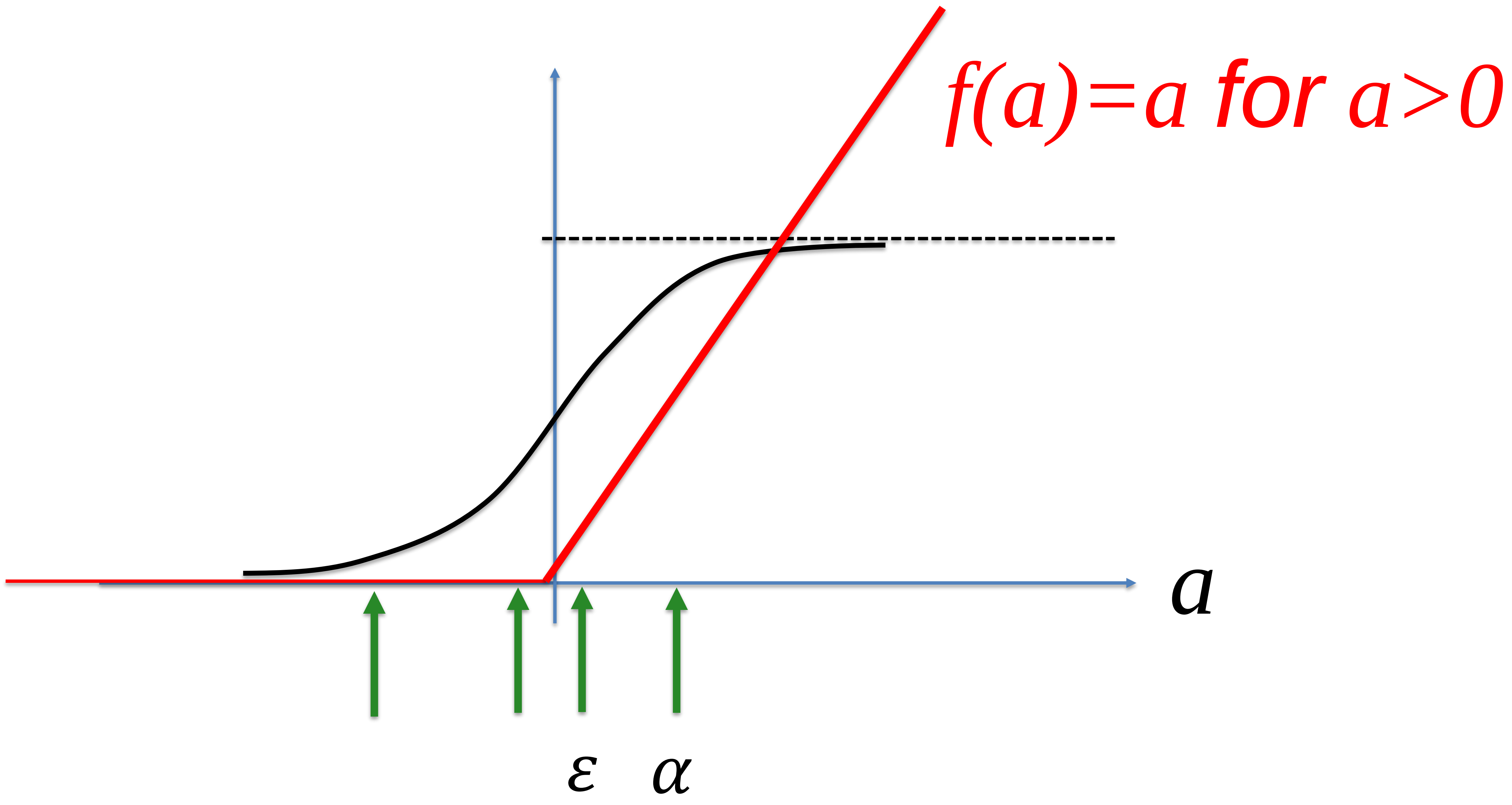
$f(x)=x$ for $x>0$

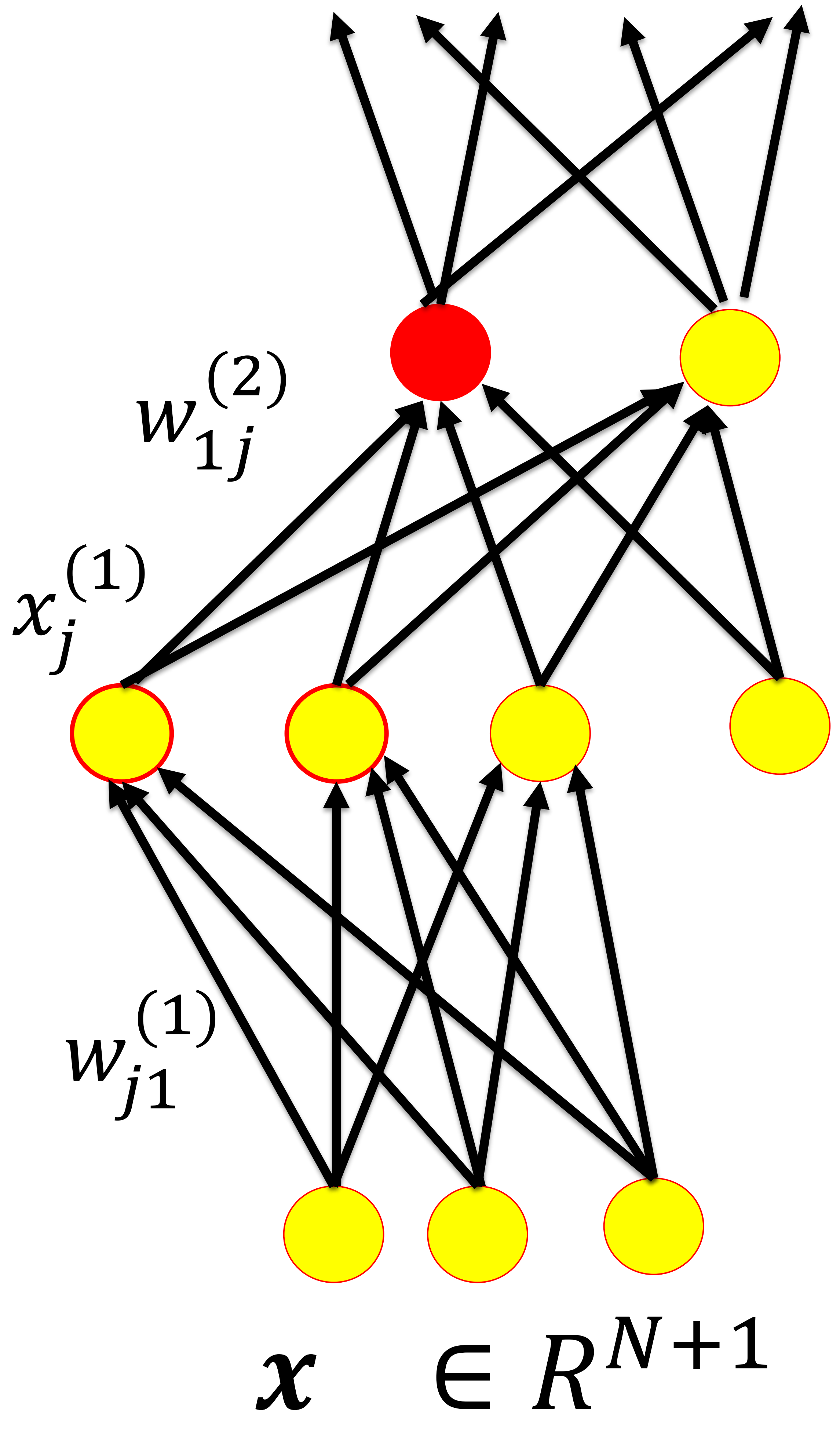
$f(x)=0$ for $x<0$ or $x=0$



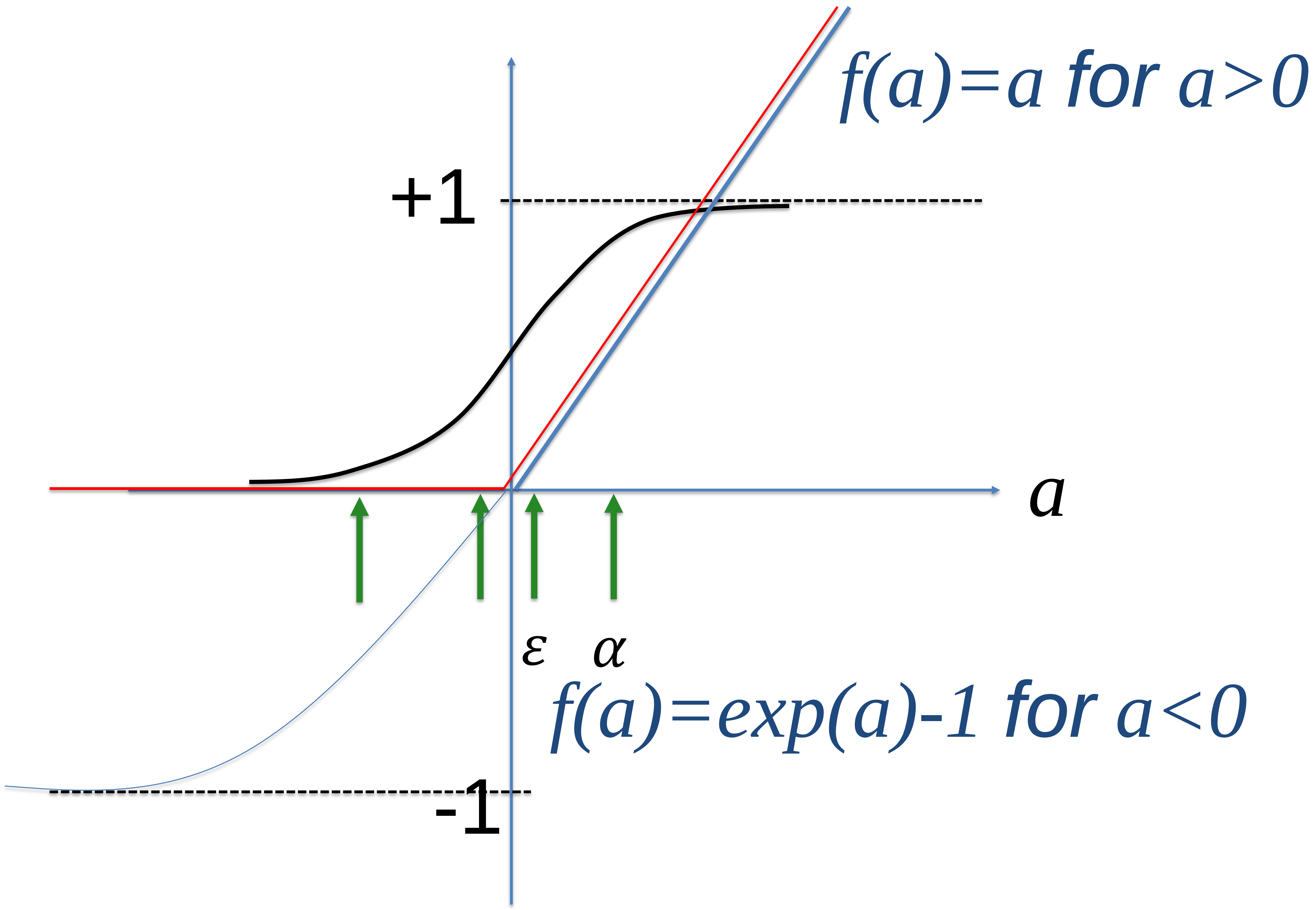
Better choices?

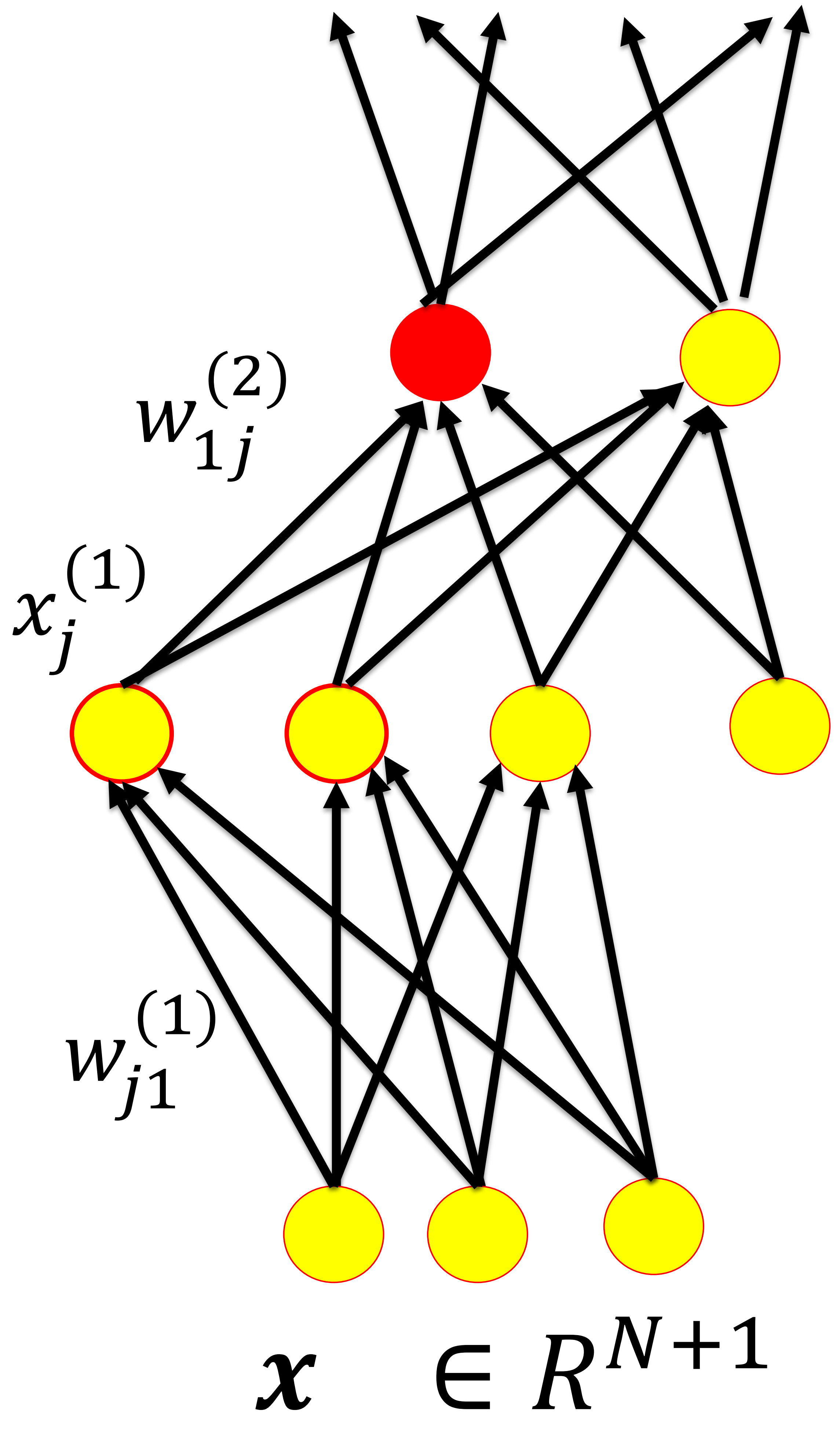
Rectified Linear (RELU) vs. Sigmoidal



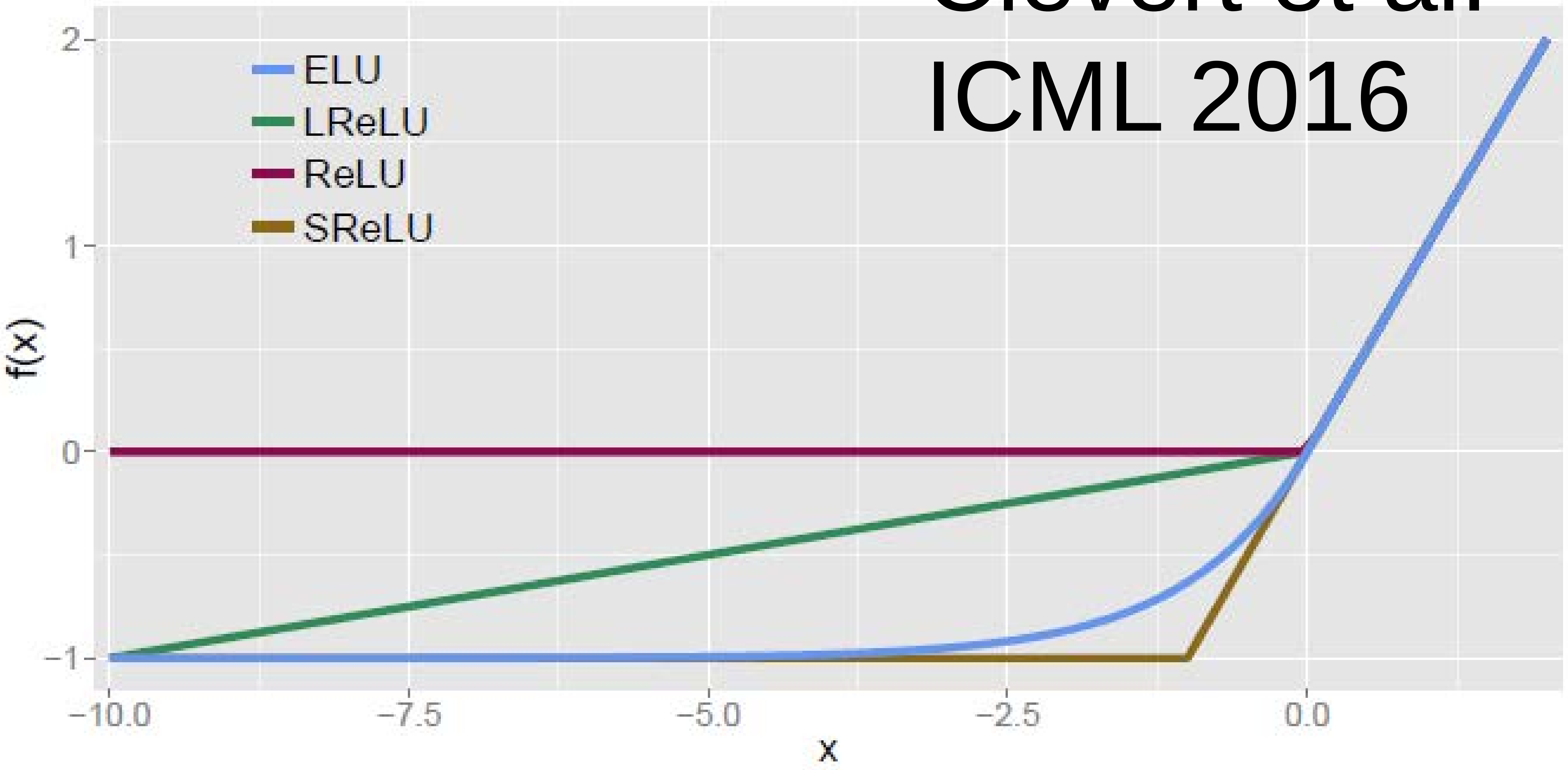


Exponential Linear vs. Sigmoidal

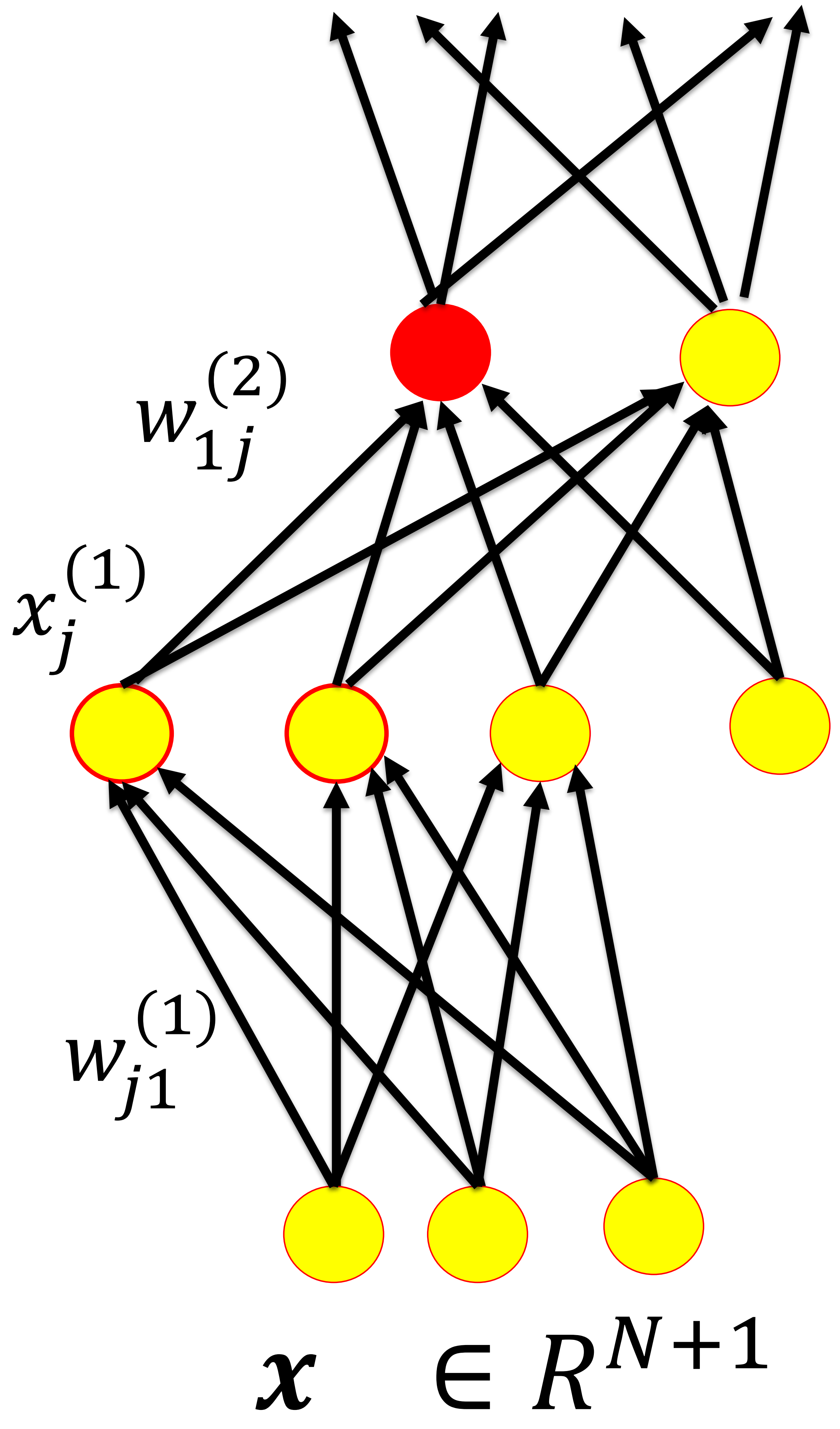




Exponential Linear (ELU) vs. Sigmoidal



Clevert et al.
ICML 2016



Shifted ReLU (SReLU)
Leaky ReLu (LReLU)

Question 1 for this week:

What are good models for hidden neurons?

… and why?

BackProp

0. Initialization of weights

1. Choose pattern $\mathbf{x}^\mu$

   input $x_k^{(0)} = x_k^\mu$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \tag{1}$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \tag{2}$$

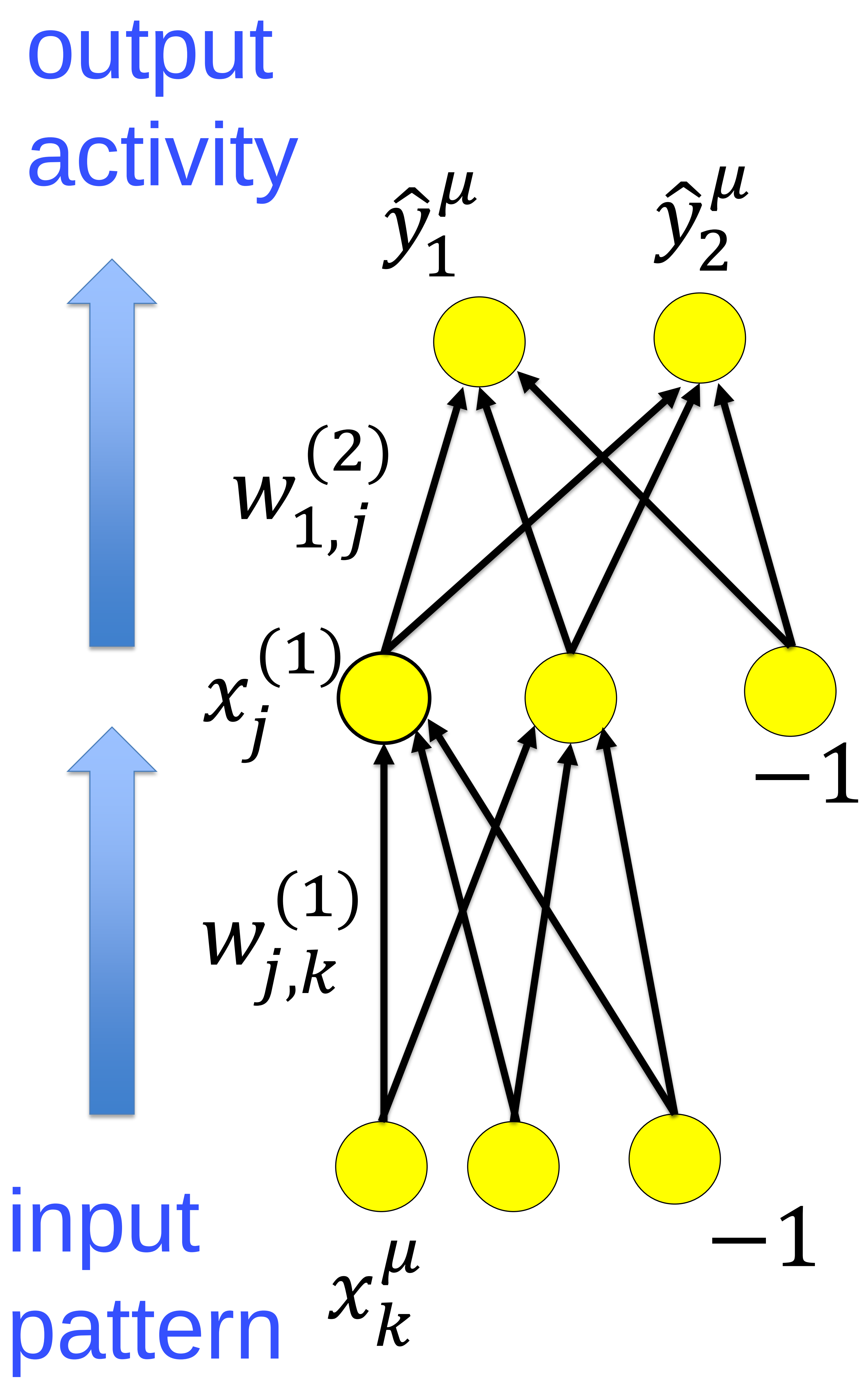
4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \tag{3}$$

5. Update weights (for each (i,j) and all layers (n))

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \tag{4}$$

6. Return to step 1.



BackProp

0. Initialization of weights

1. Choose pattern $\mathbf{x}^{\mu}$

$$\text{input } x_k^{(0)} = x_k^{\mu}$$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \tag{1}$$

$$\text{output } \hat{y}_i^{\mu} = x_i^{(n_{\max})}$$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^{\mu} - \hat{y}^{\mu}] \tag{2}$$

4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

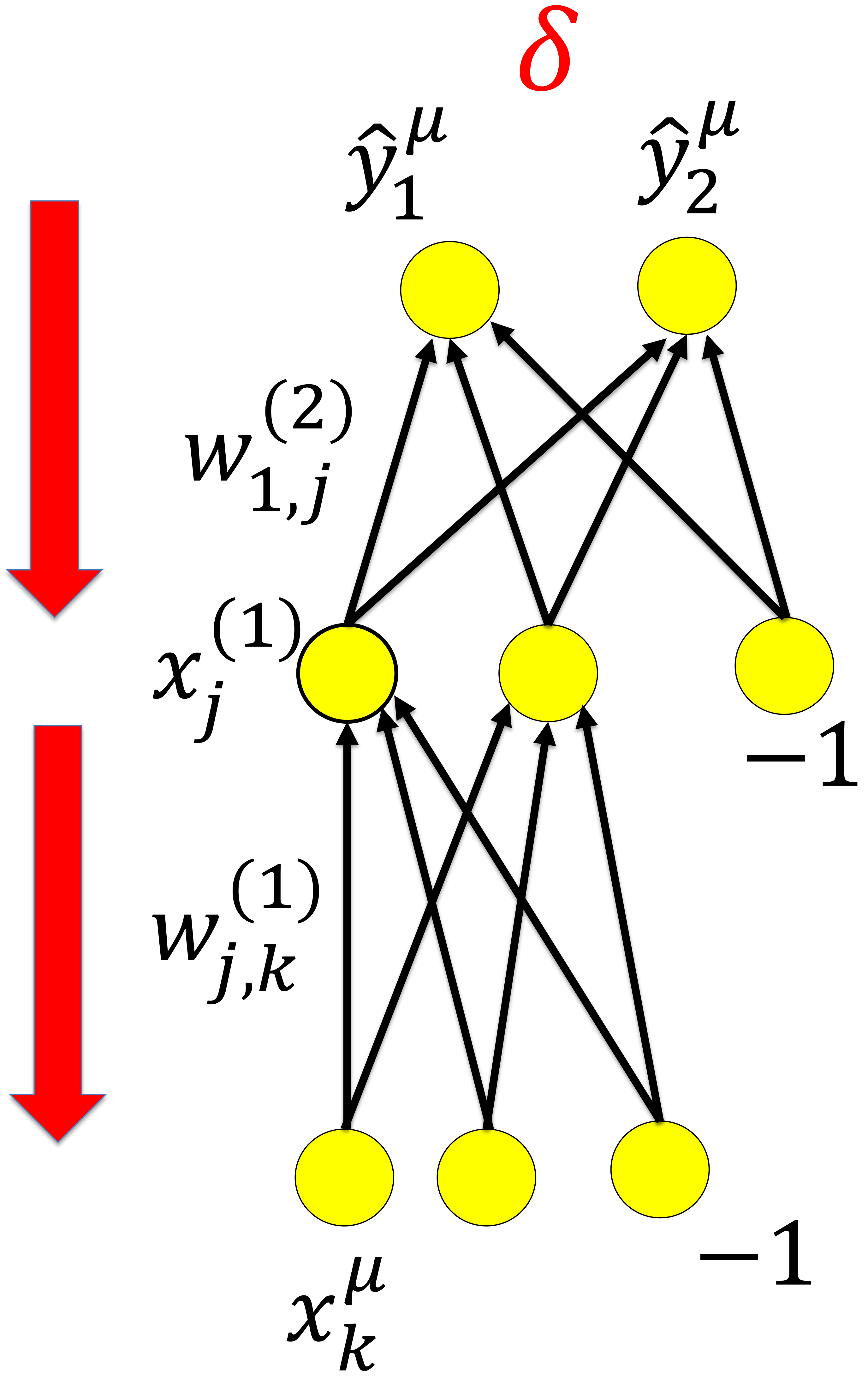
$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \tag{3}$$

5. Update weights (for each (i, j) and all layers (n))

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \tag{4}$$

6. Return to step 1.

Calculate output error



BackProp

0. Initialization of weights

1. Choose pattern $\mathbf{x}^\mu$

input $x_k^{(0)} = x_k^\mu$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \tag{1}$$

output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \tag{2}$$

4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \tag{3}$$

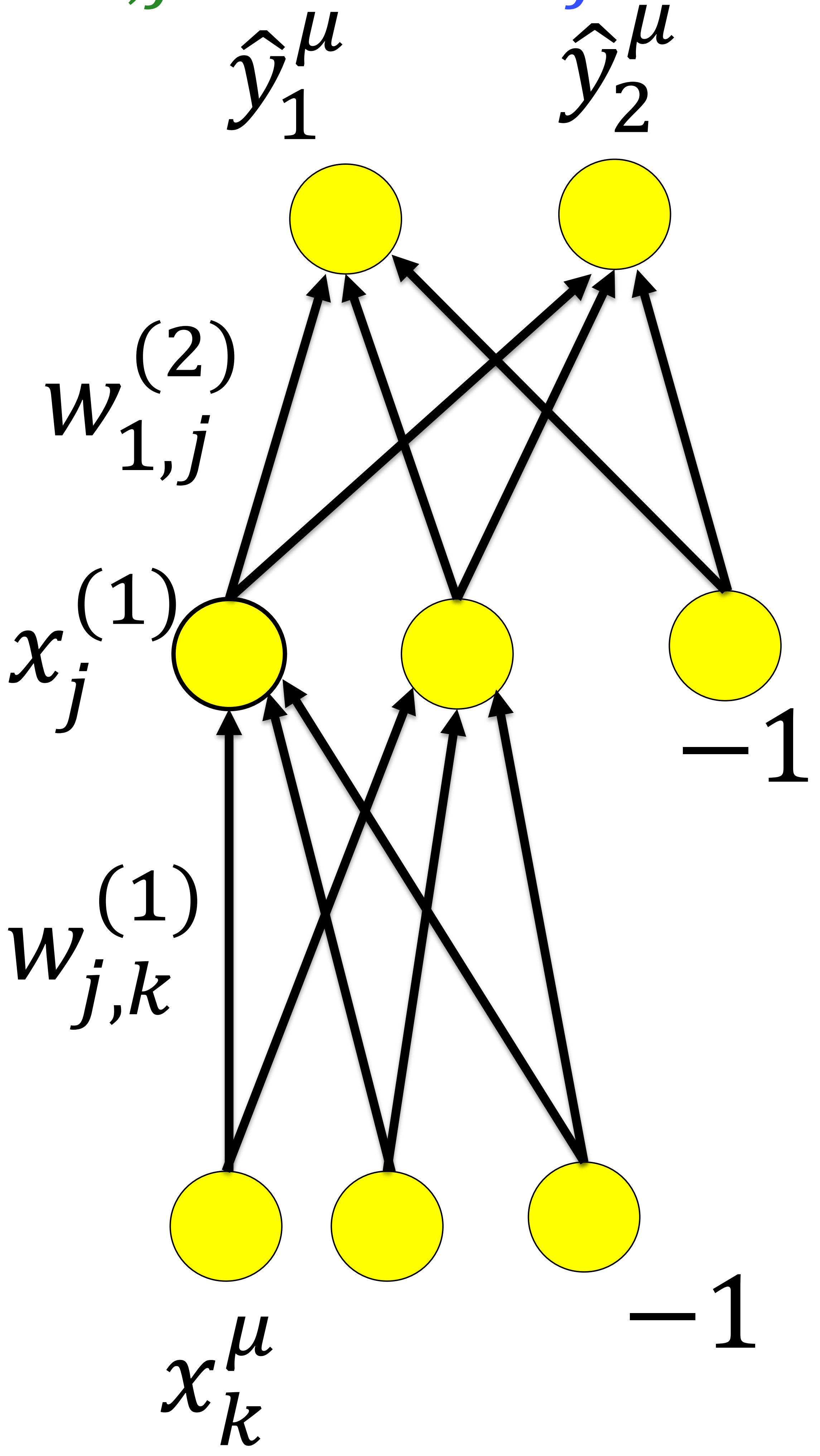
5. Update weights (for each (i,j) and all layers (n))

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \tag{4}$$

6. Return to step 1.

update **all** weights

$$\Delta w_{i,j}^{(n)} = \delta_i^{(n)} x_j^{(n-1)}$$

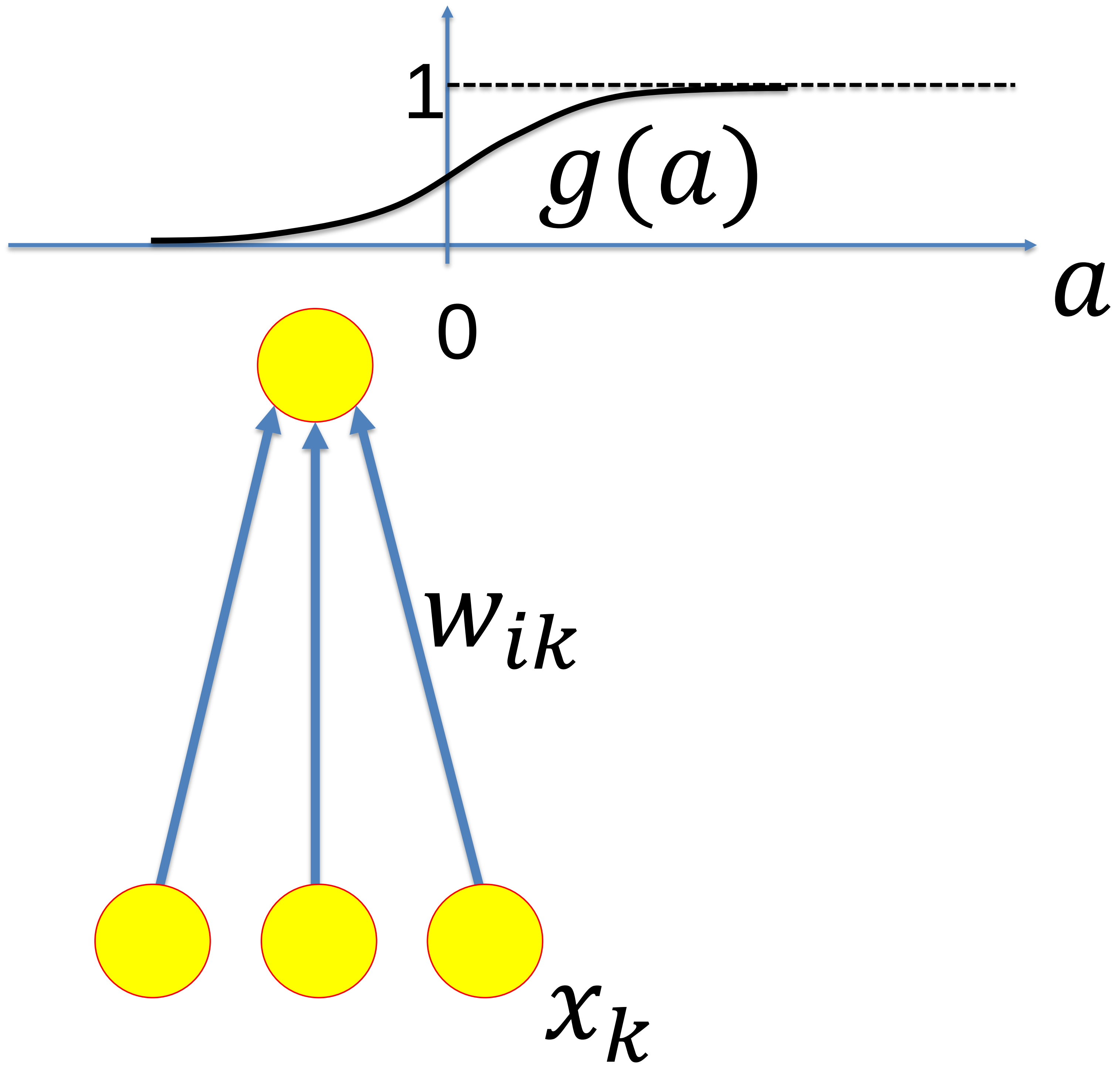


Question 2 for this week:

Why does the initatialization or normalization matter in backprop?

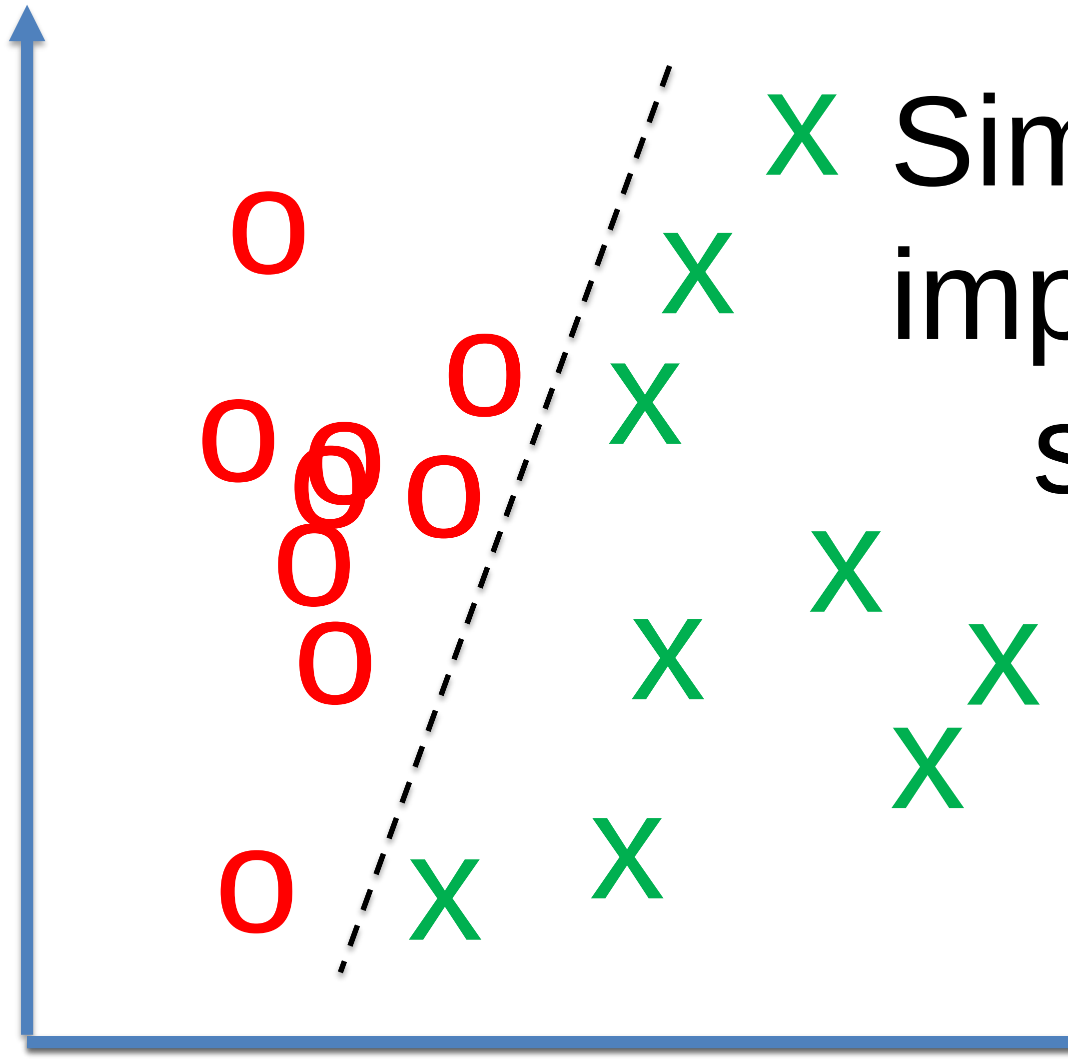
Review: Single-Layer networks/simple perceptron

$$\hat{y} \quad = 0.5[1 + tanh(\sum_k w_k\, x_k - \vartheta)]$$



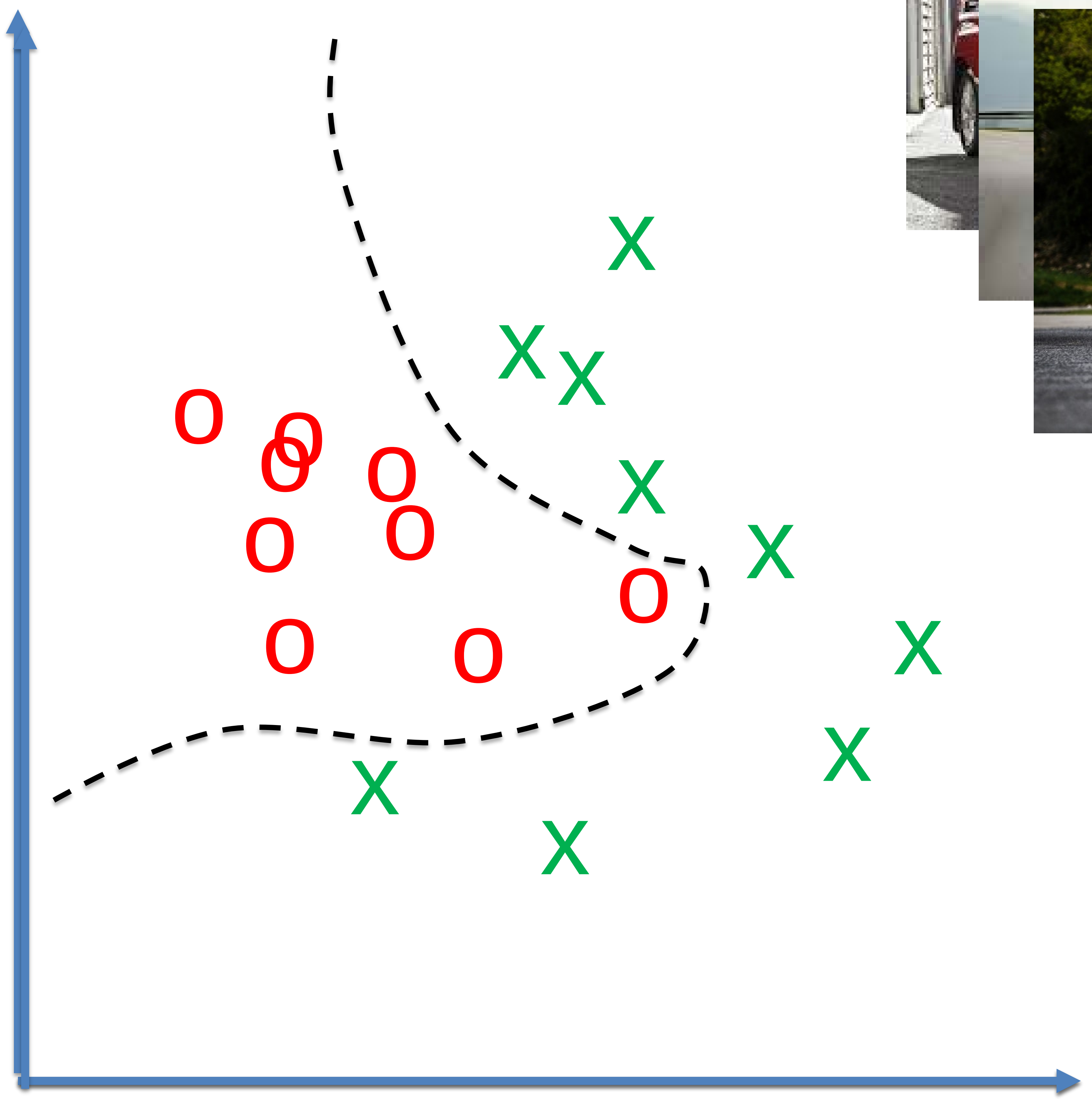
vector $\boldsymbol{x}$

$$d(\boldsymbol{x}) = \sum_k w_k\, x_k - \vartheta = 0$$



Simple perceptron imposes a linear separation

Review: Classification as a geometric problem

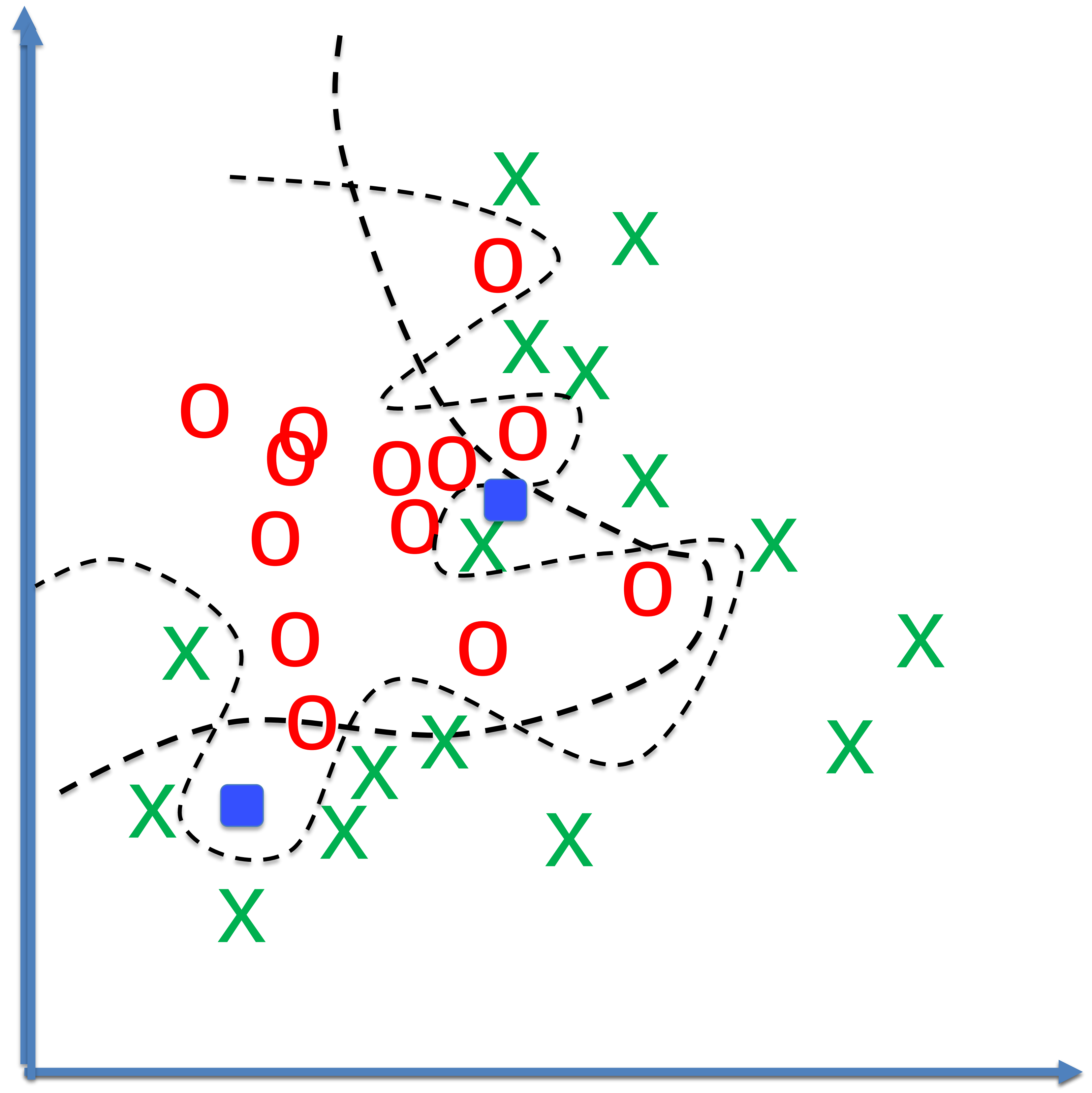


Review: The problem of overfitting

Big Multilayer perceptrons are flexible and can be trained by BackProp to minimize classification error

… but is flexibility always good?

Network has to work on future data:
test data base



Question 3 for this week:

What are good models for regularization?

… and why?

We start with this question!

Artificial Neural Networks: Lecture 4

Tricks of the Trade in deep networks

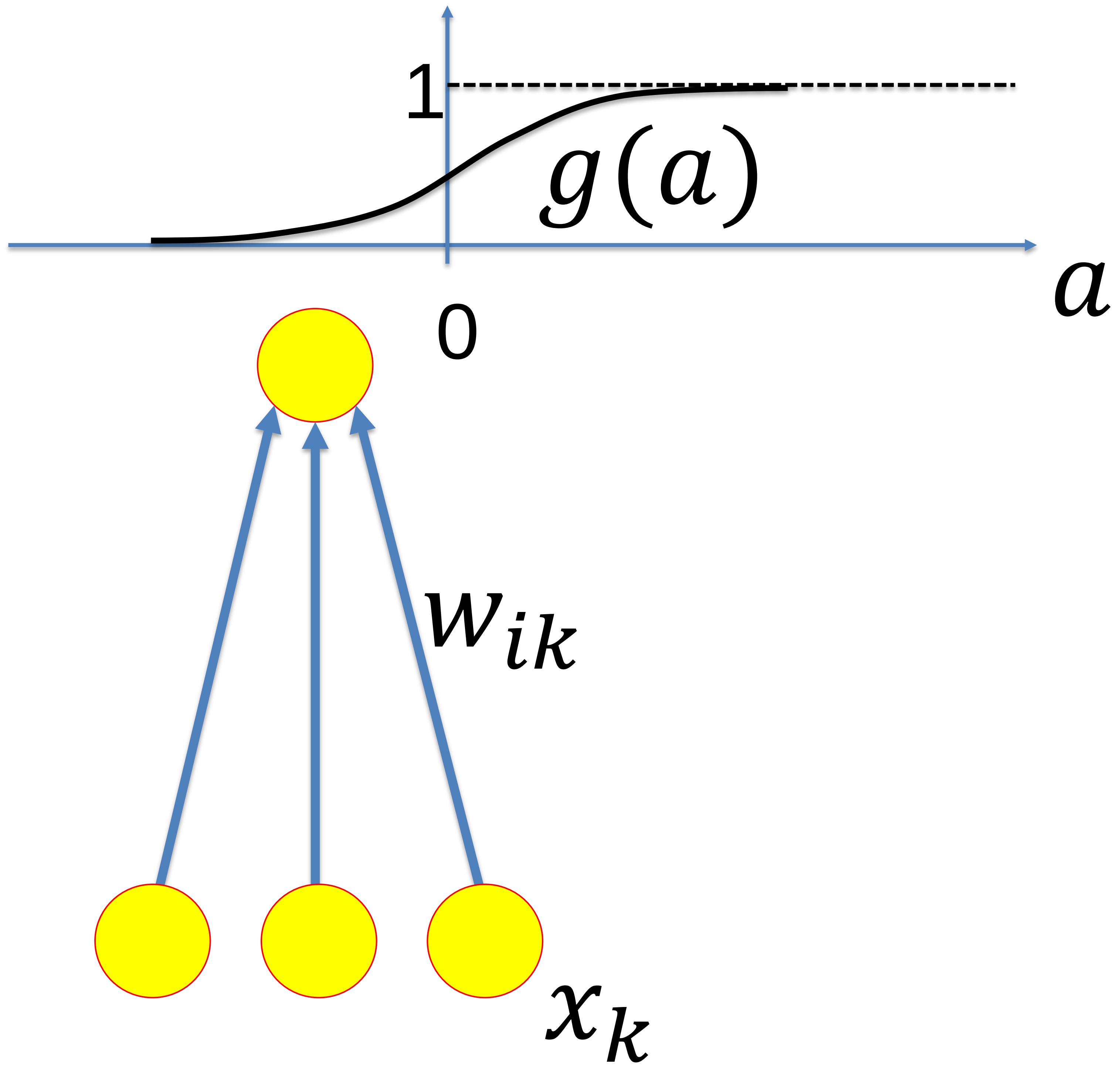
Wulfram Gerstner

EPFL, Lausanne, Switzerland

1. Bagging

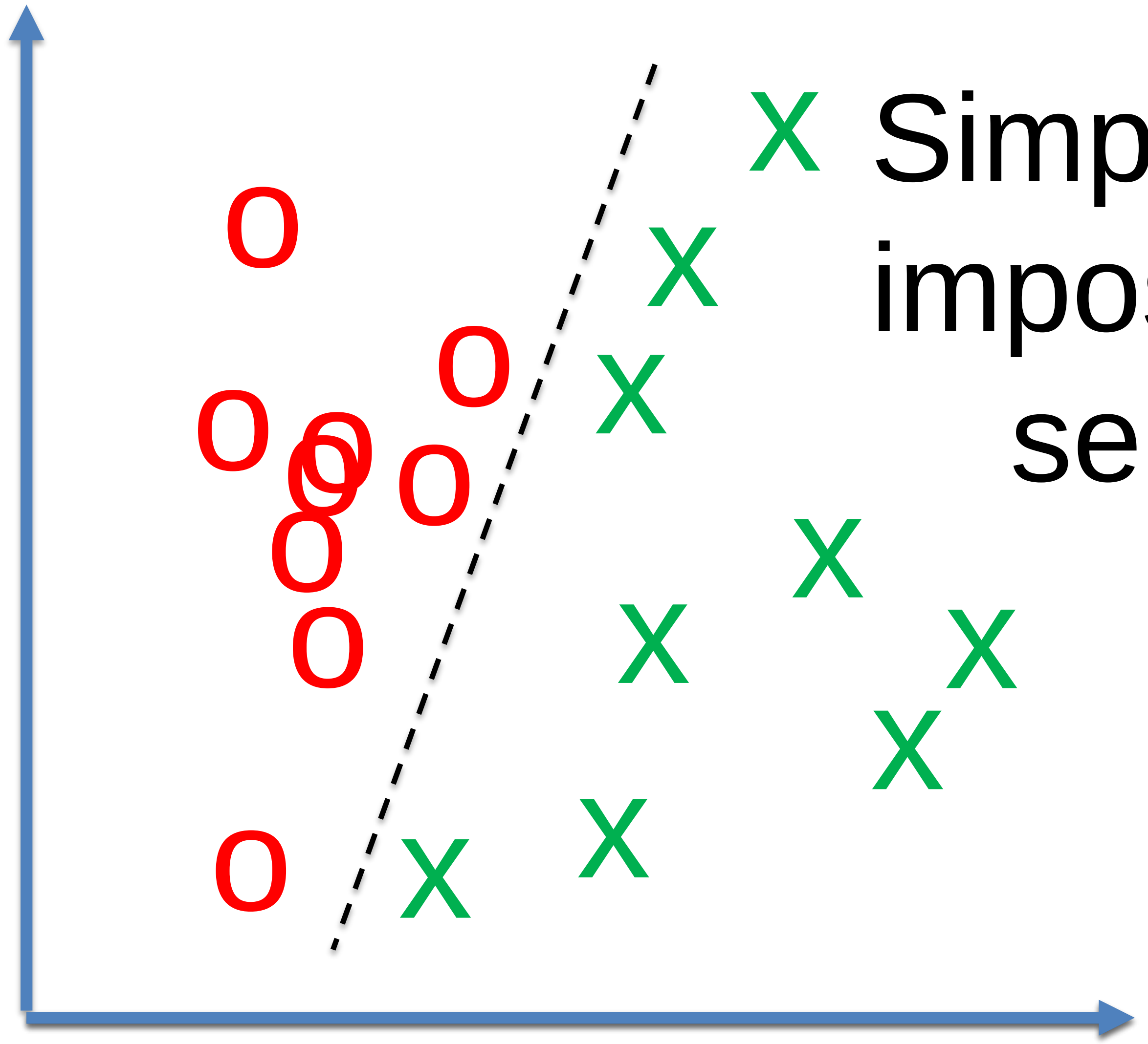
1. Bagging Example: simple perceptron

$$\hat{y} \quad = 0.5[1 + tanh(\textstyle\sum_k w_k\, x_k - \vartheta)]$$



vector $\boldsymbol{x}$

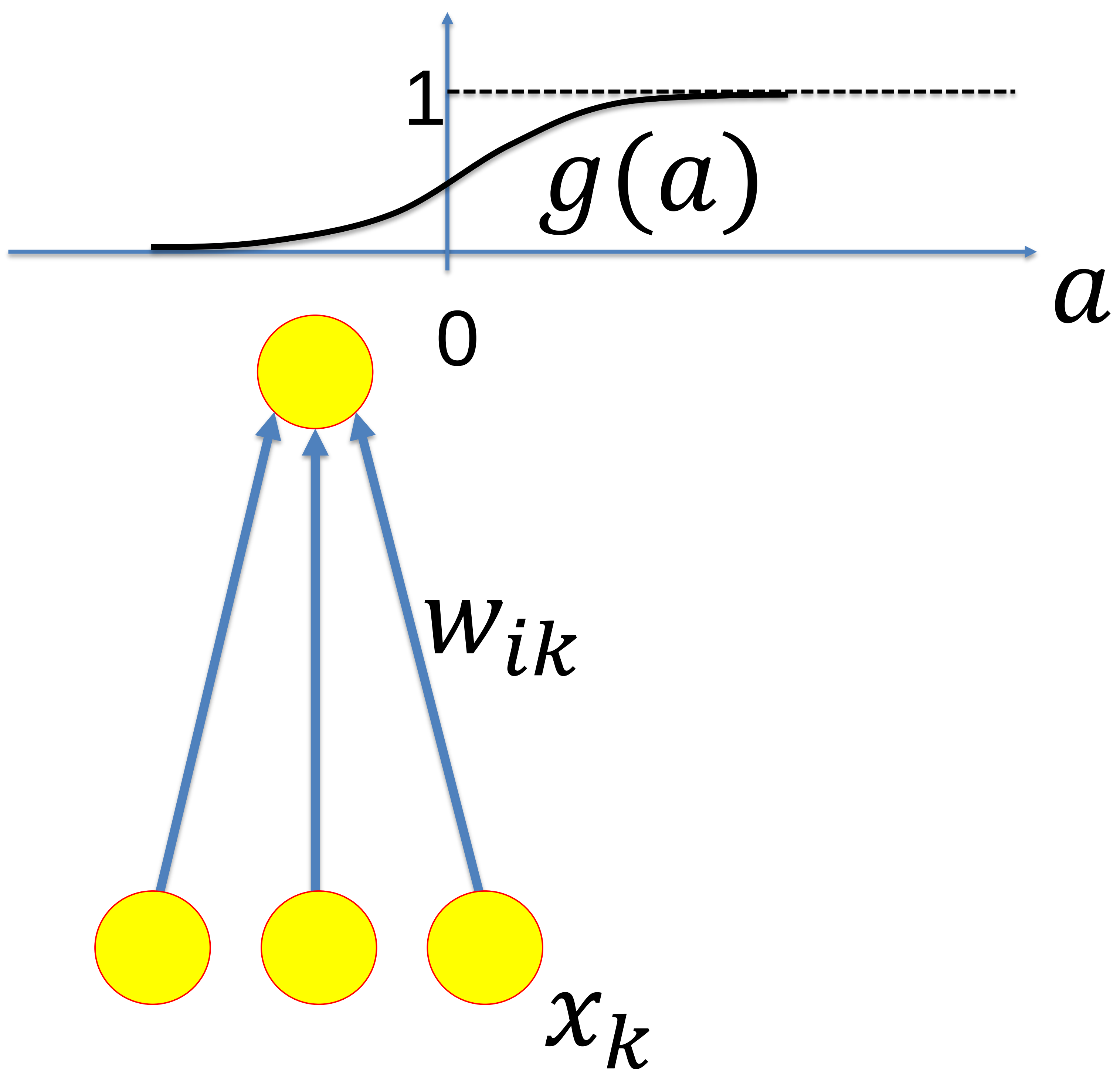
$$d(\boldsymbol{x}) = \sum_k w_k\, x_k - \vartheta = 0$$



Simple perceptron imposes a linear separation

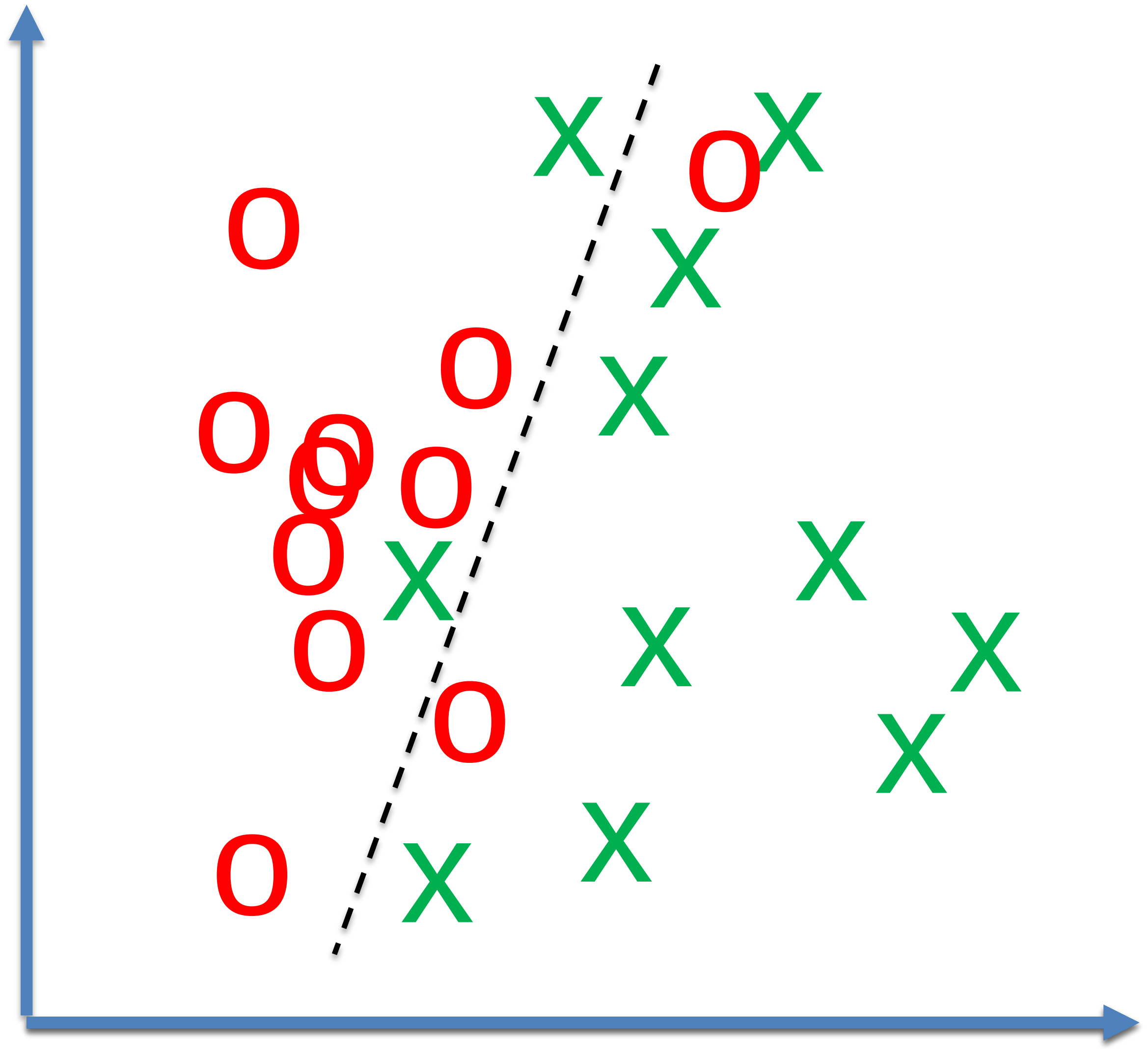
1. Bagging Example: simple perceptron for noisy data

$$\hat{y} \quad = 0.5[1 + tanh(\sum_k w_k \, x_k - \vartheta)]$$



vector $\boldsymbol{x}$

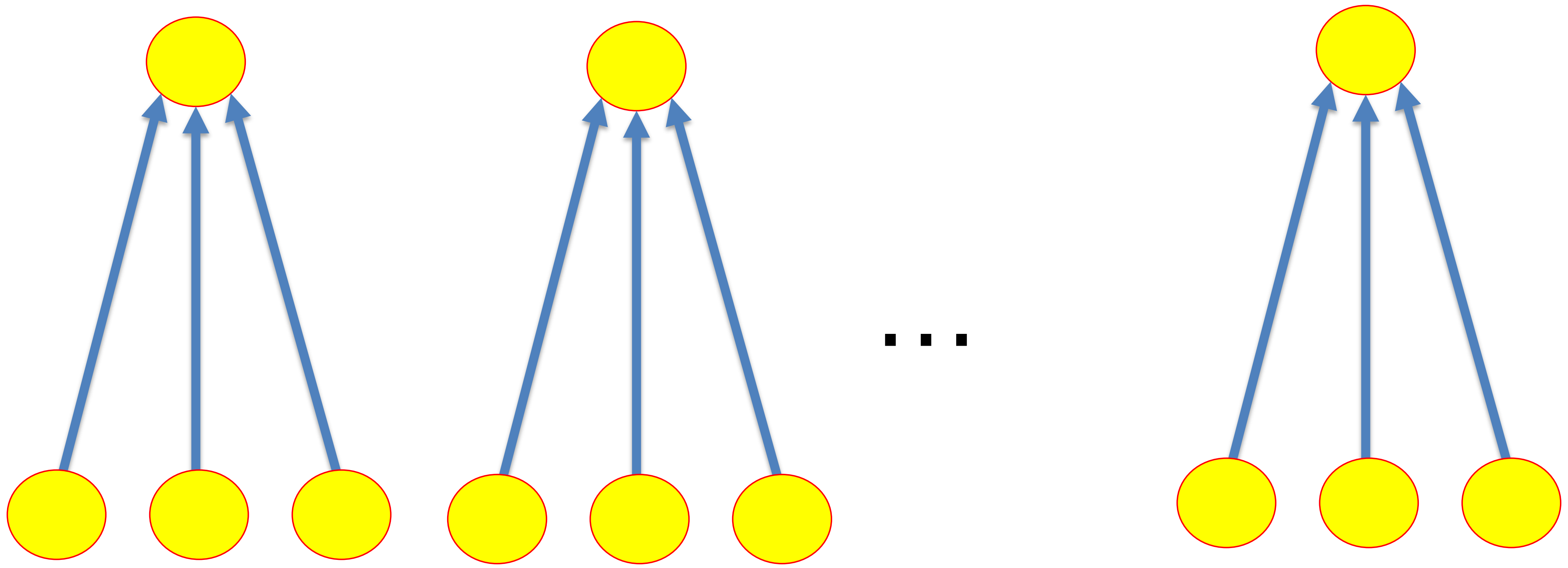
Find best (approximate) linear separation

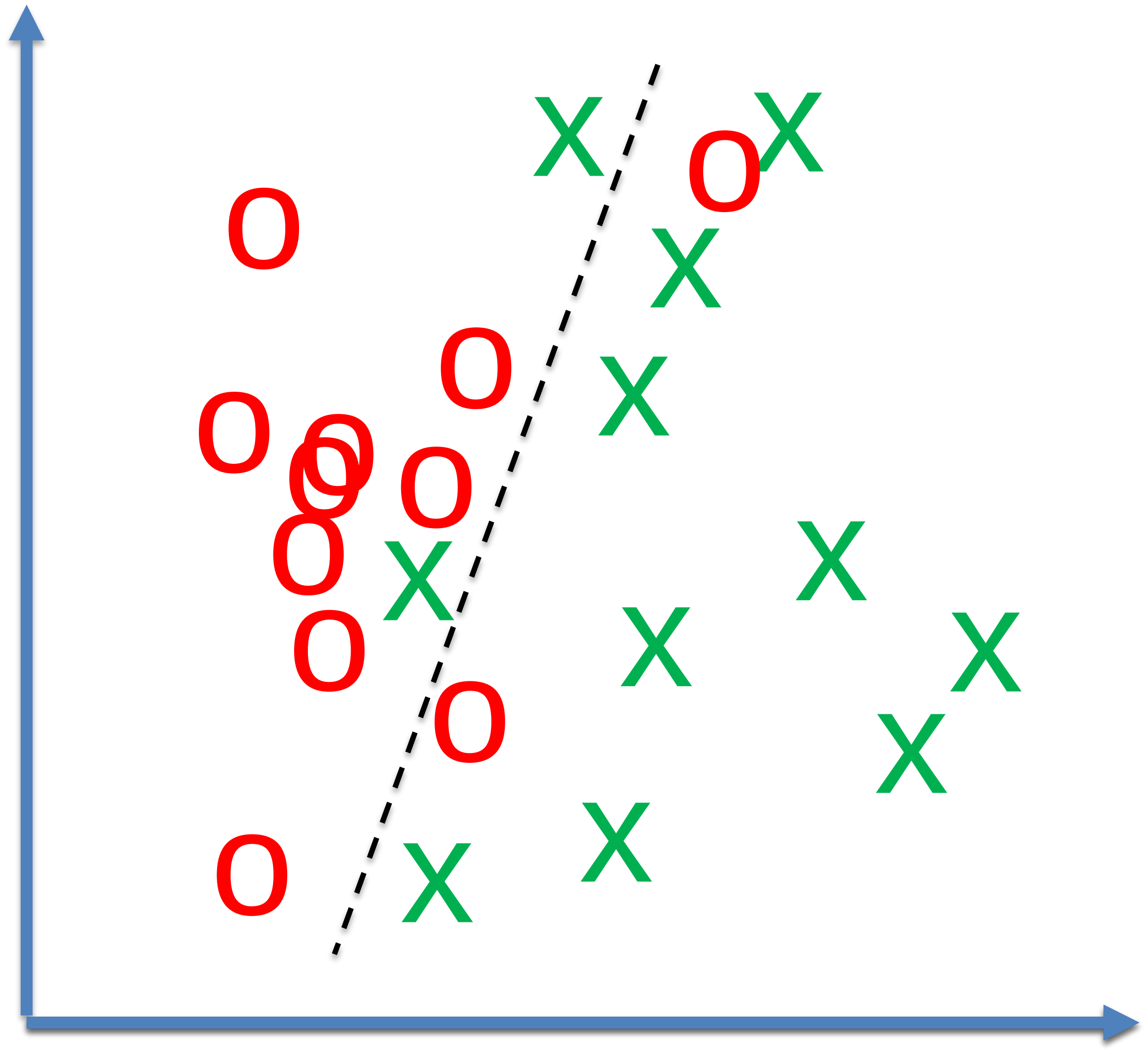


1. Bagging Idea: (i) Repeat variants of your model K times

$$\hat{y} \quad = 0.5[1 + tanh(\sum_k w_k \, x_k - \vartheta)]$$

Find best (approximate) linear separation

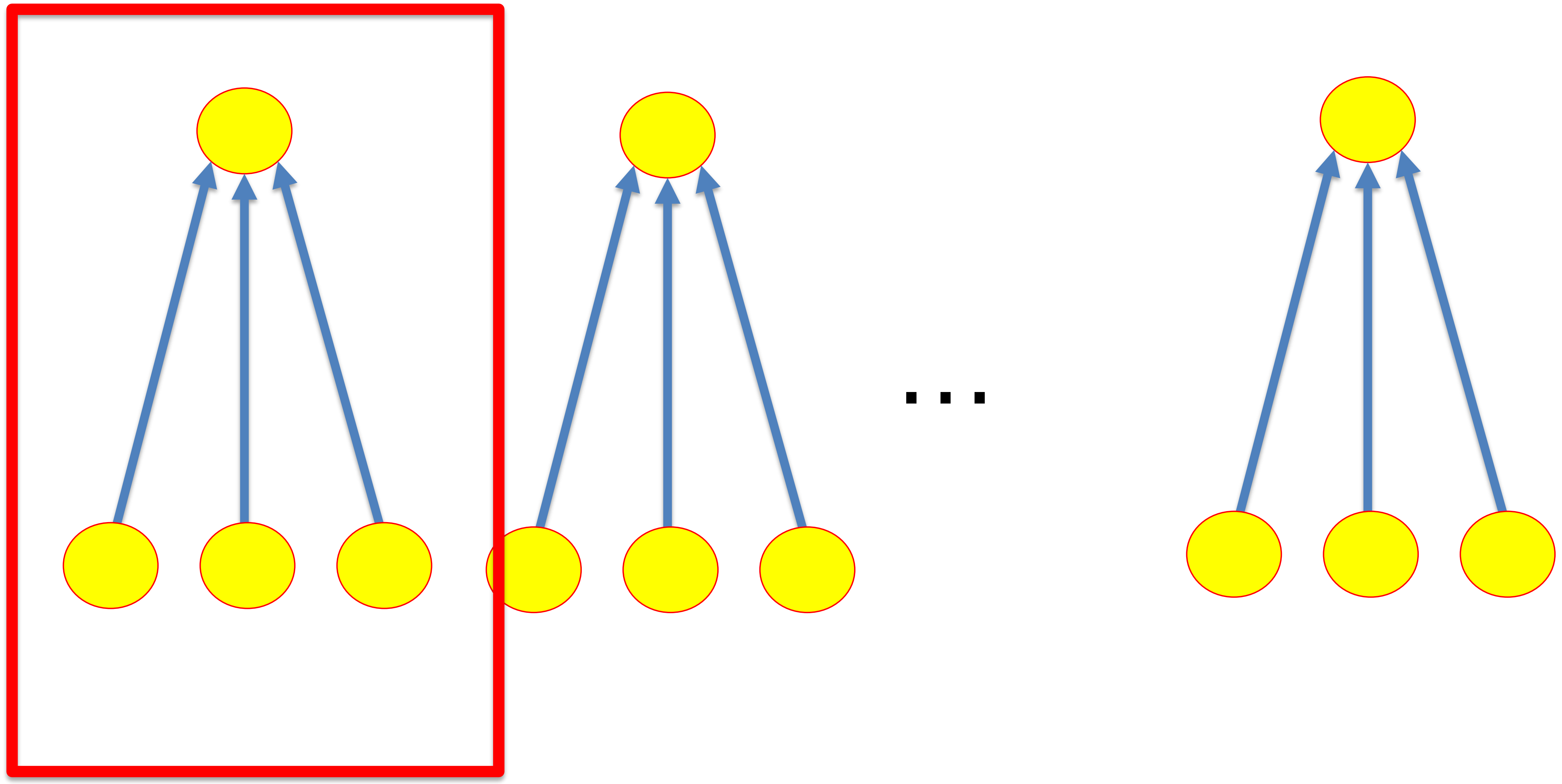


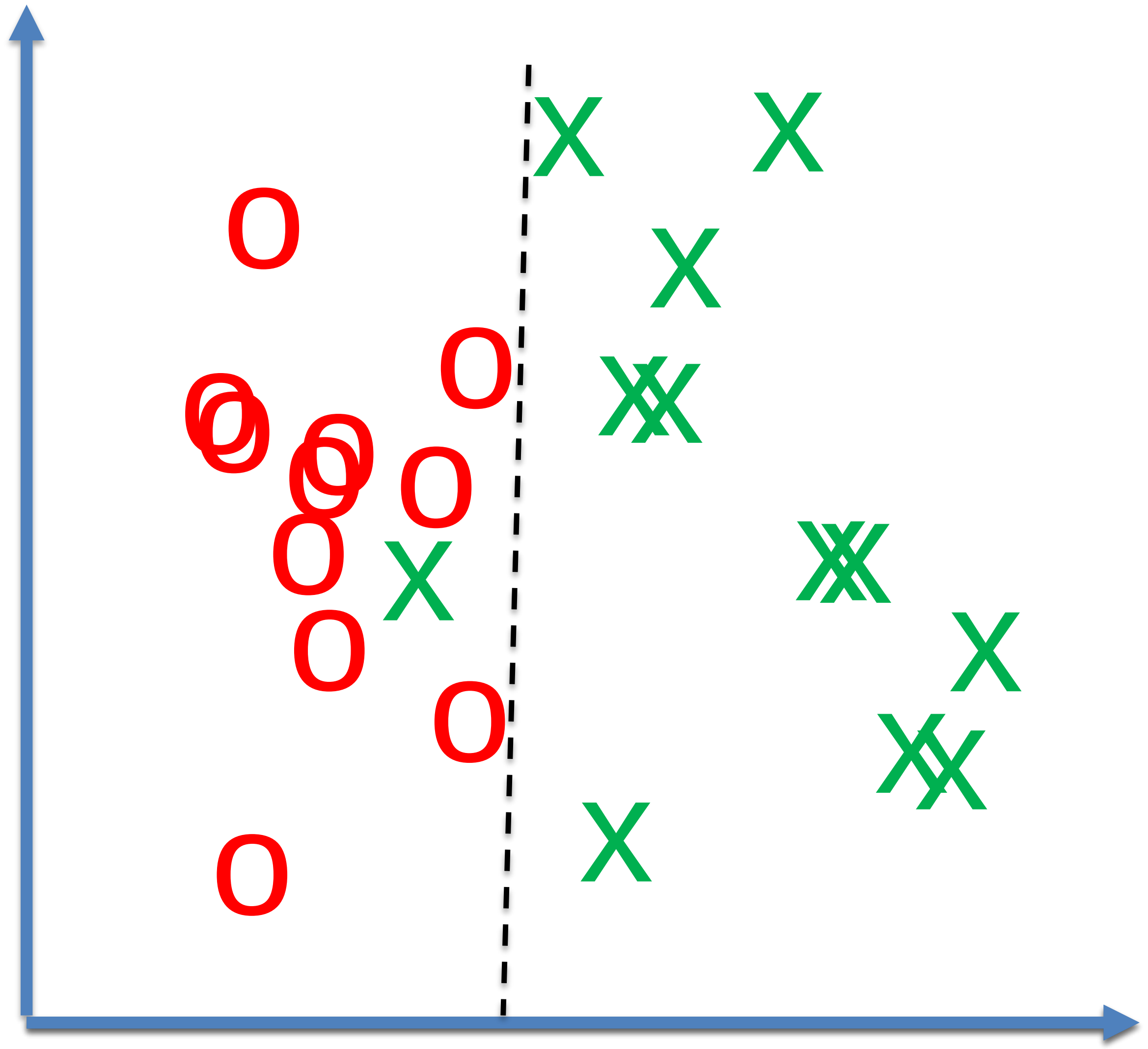


1. Bagging Idea: (ii) Each Variant sees different subsets of data

$$\hat{y}_1 = 0.5[1 + tanh(\sum_k w_k\, x_k - \vartheta)]$$

Find best (approximate) linear separation

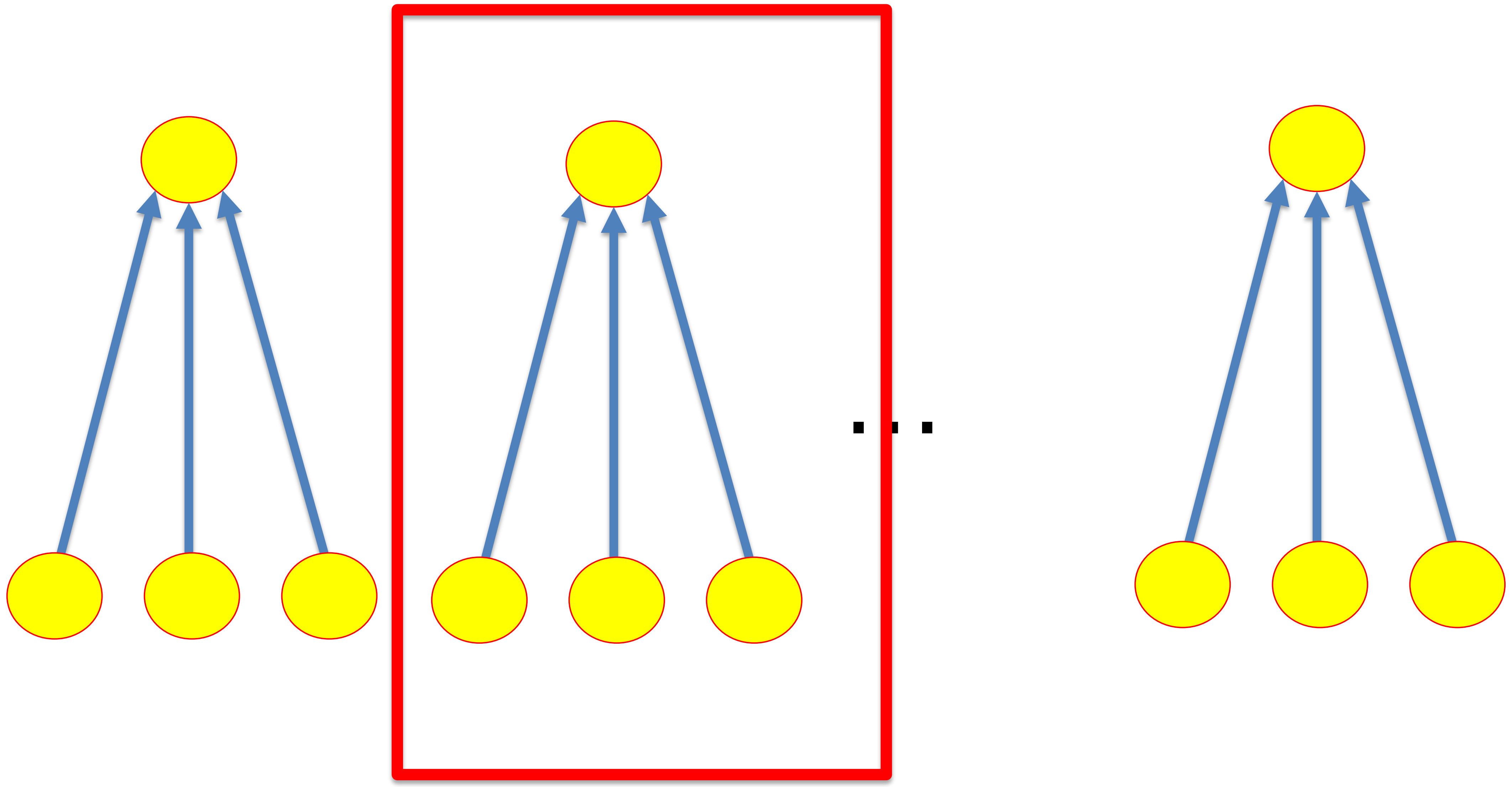


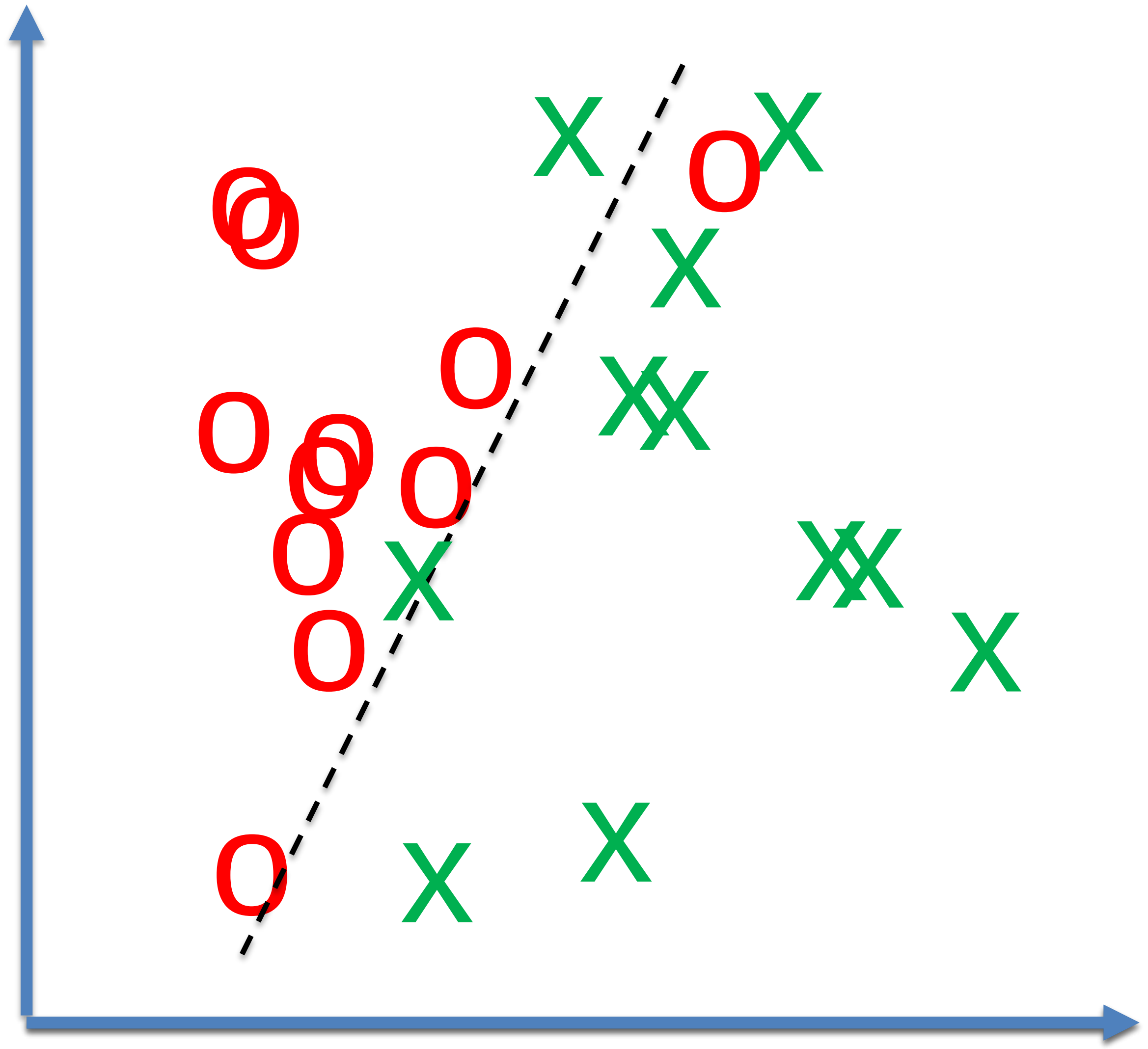


1. Bagging Idea: (ii) Each Variant sees different subsets of data

$$\hat{y}_2 = 0.5[1 + tanh(\sum_k w_k\, x_k - \vartheta)]$$

Find best (approximate) linear separation

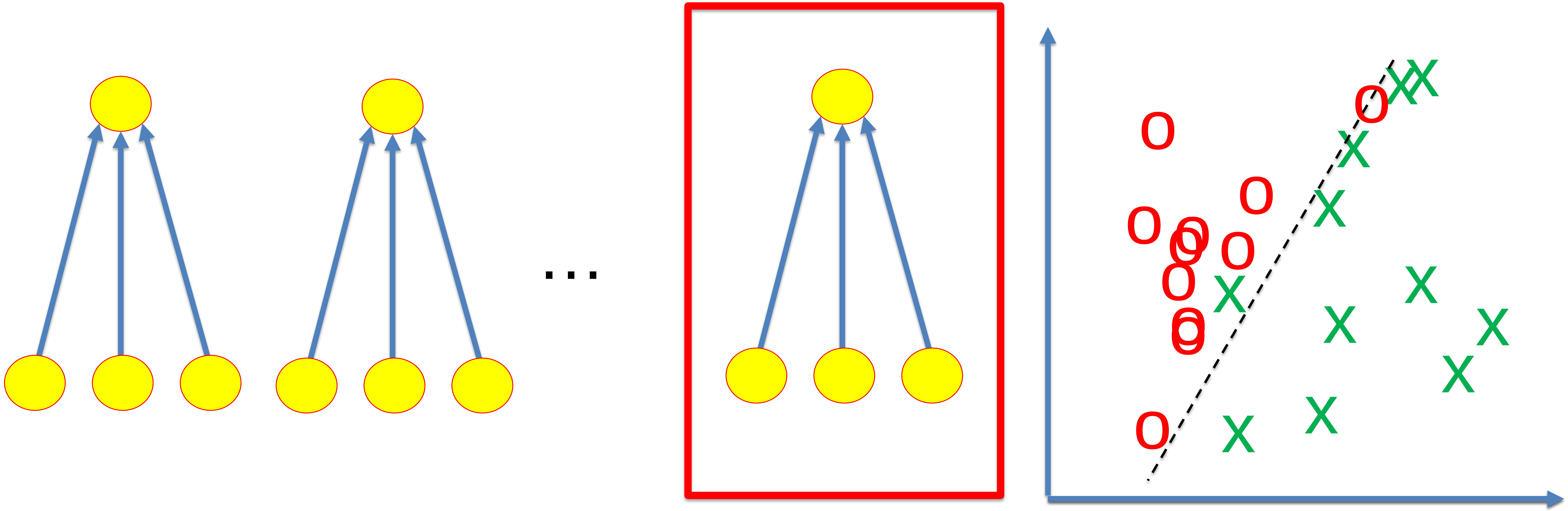




1. Bagging Idea: (ii) Each Variant sees different subsets of data

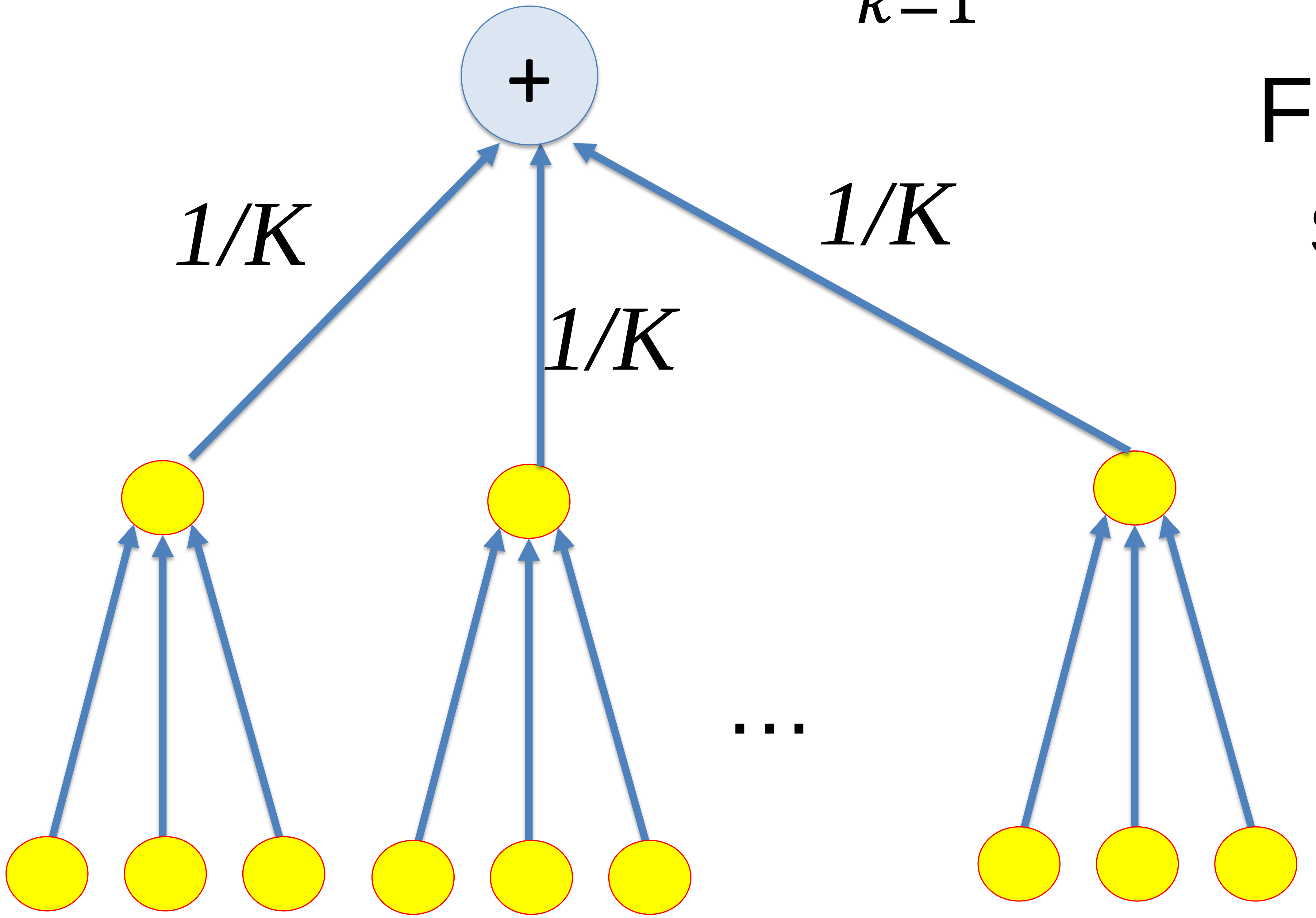
$$\hat{y}_K = 0.5[1 + tanh(\sum_k w_k x_k - \vartheta)]$$

Find best (approximate) linear separation

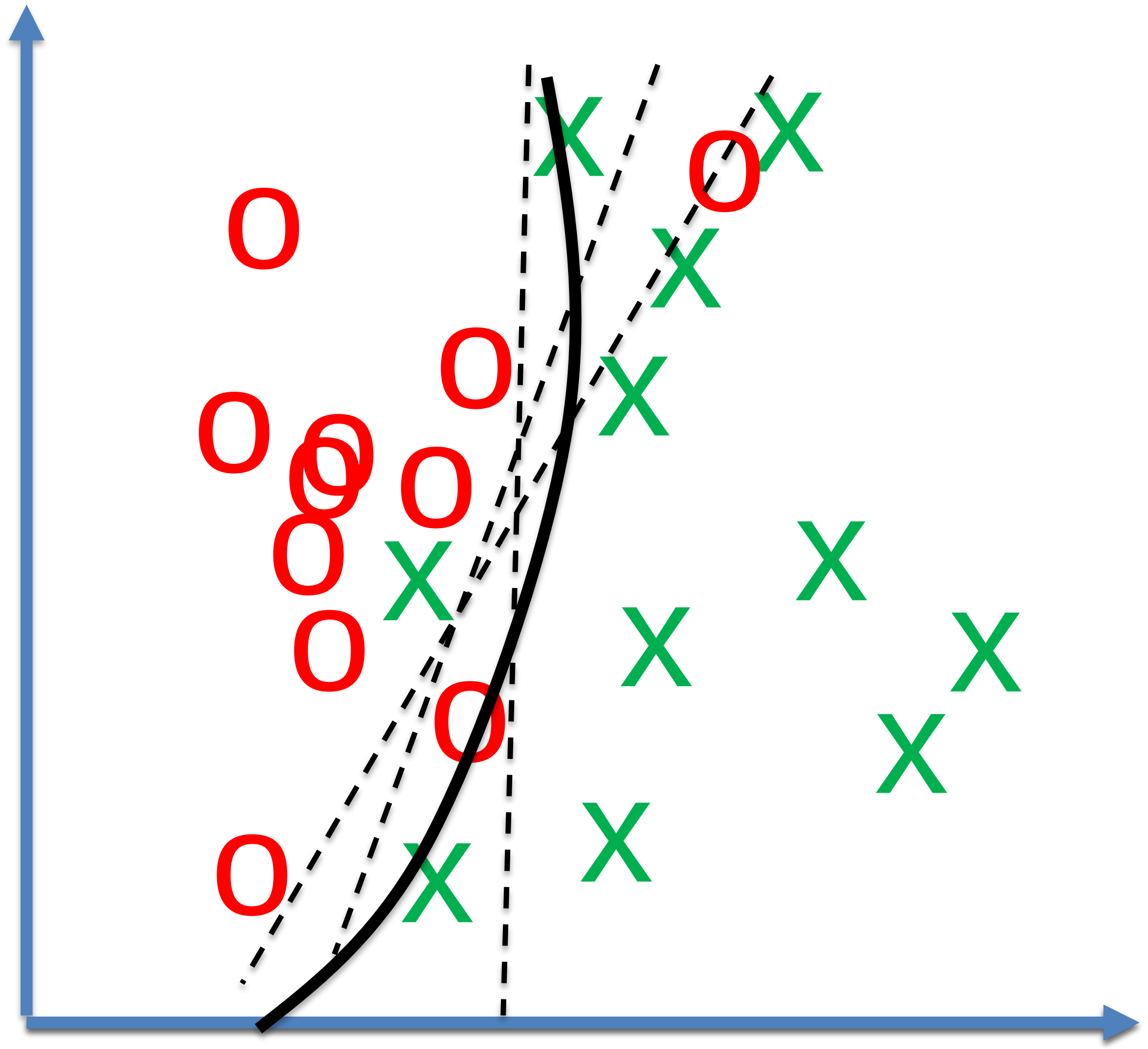


1. Bagging Idea: (iii) Average over all K variants

$$\hat{y}_{bag} = \frac{1}{K}\sum_{k=1}^{K} \widehat{y}_k$$



Find average (nonlinear) separation



1. Bagging : Algorithm

Given: Training data set $\{ \quad (\boldsymbol{x}^{\mu}, t^{\mu}) \quad , \qquad 1 \le \mu \le P1 \quad \}$;

1 Generate K different training sets

for $k=1,\ldots,K$

pick $P1$ times into your data set with replacement

(your can pick the same data point several times)

2 Initialize K different variants of your model

3 Train model k on data set k up to criterion

4 For a future data point (test set)

for $k=1,\ldots,K$

put input $\boldsymbol{x}$ into model k, read out $\hat{y}_k$

5 Report average $\hat{y}_{bag} = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_k$

1. Bagging: Theory

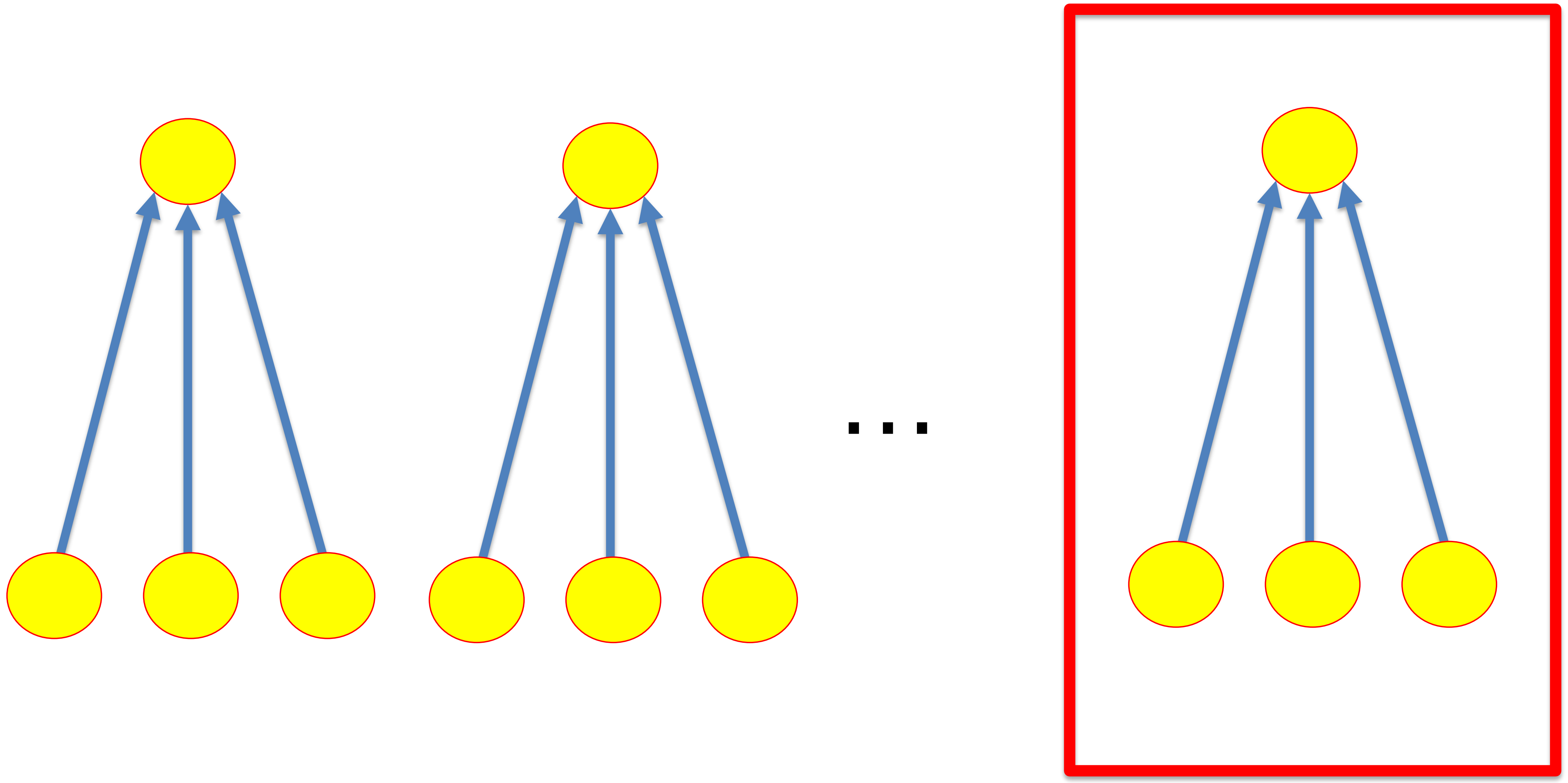
Blackboard: Bagging

Model k

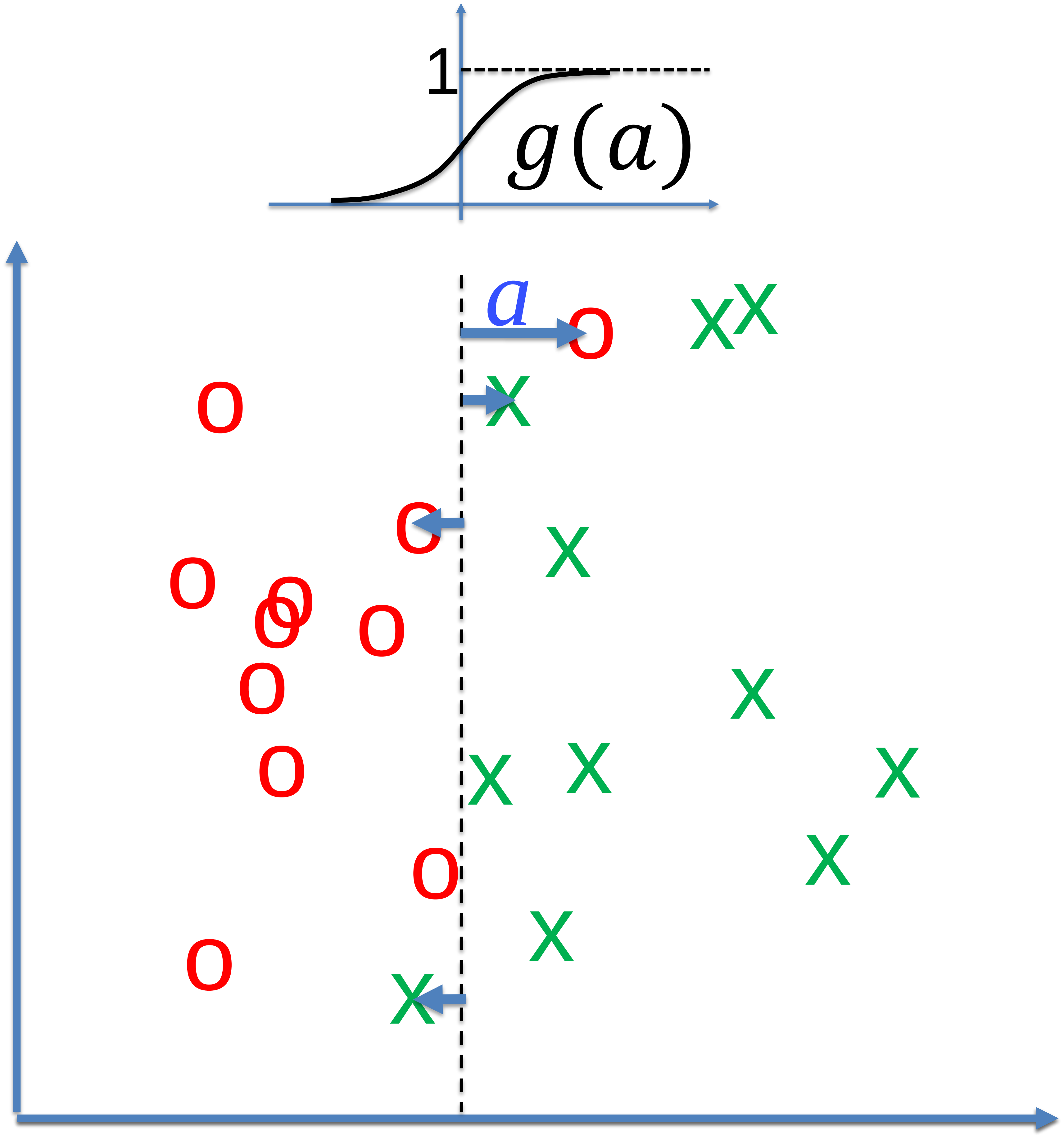
$$\hat{y}_k = 0.5[1 + tanh(\sum_j w_j x_j - \vartheta)]$$

Bagged output

$$\hat{\boldsymbol{y}}_{\mathbf{bag}} = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_k$$



$$\delta_k^\mu = t_k^\mu - \hat{y}_k^\mu = \sigma(a)$$



1. Bagging : Theory

Blackboard: Bagging

Claim: bagged output has smaller quadratic error than a typical individual model

bagged output $\hat{y}_{\mathbf{bag}} = \frac{1}{K}\sum_{k=1}^{K}\hat{y}_k$

1. Bagging : Result

assumption: the average delta-difference, defined as

$$\frac{1}{P}\sum_{\mu=1}^{P}[\delta_k^\mu] = d$$

is the same for all K copies of the model.

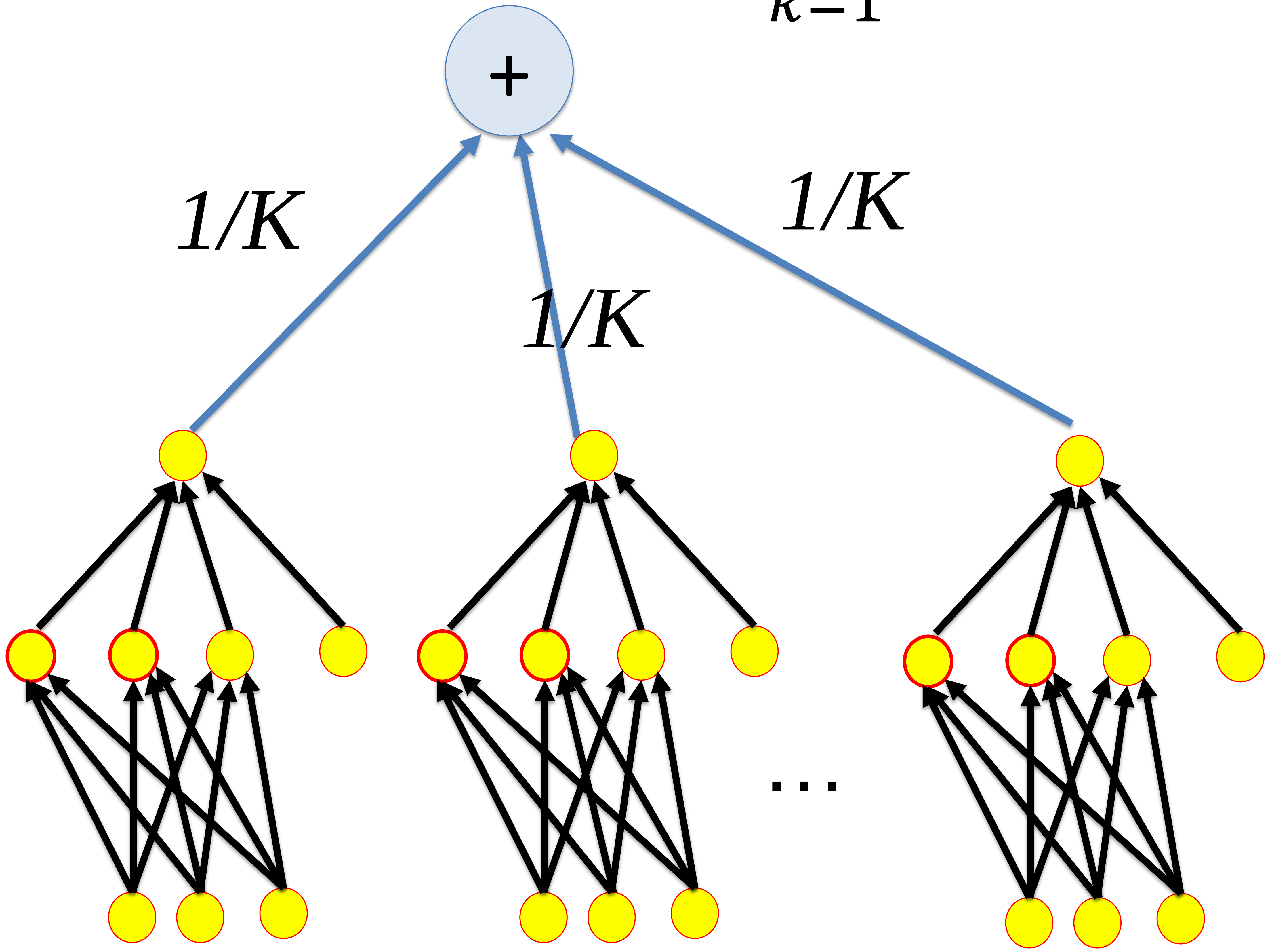
THEN

- bagged output has smaller quadratic error than a typical individual model

- if all K individual models are uncorrelated, the gain in performance scales as 1/K

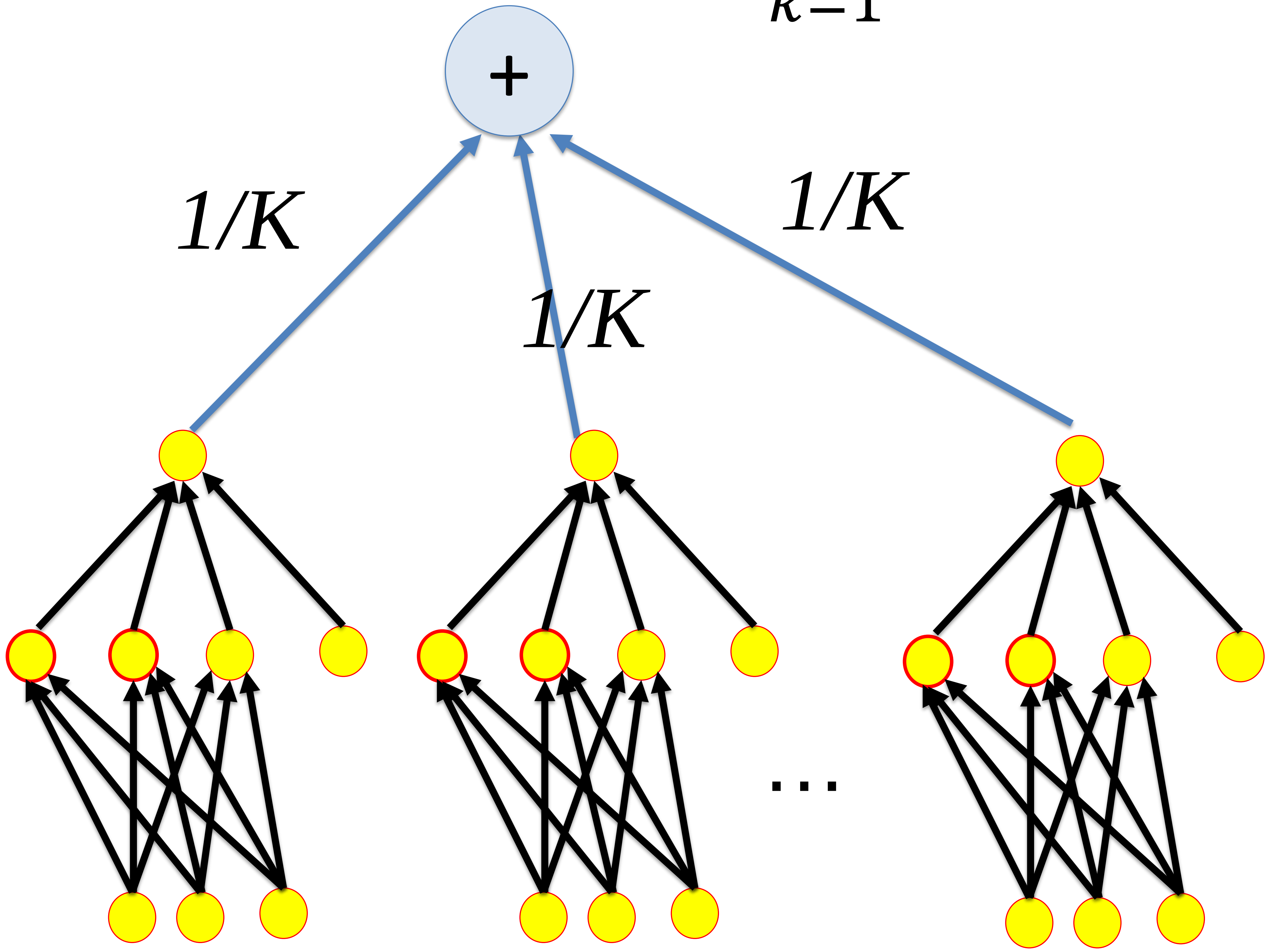
1. Bagging: each of the models can be a deep network

$$\hat{y} = \frac{1}{K}\sum_{k=1}^{K}\widehat{y_k}$$



1. Bagging: each of the models sees a different data set

$$\hat{y} = \frac{1}{K}\sum_{k=1}^{K}\widehat{y}_k$$



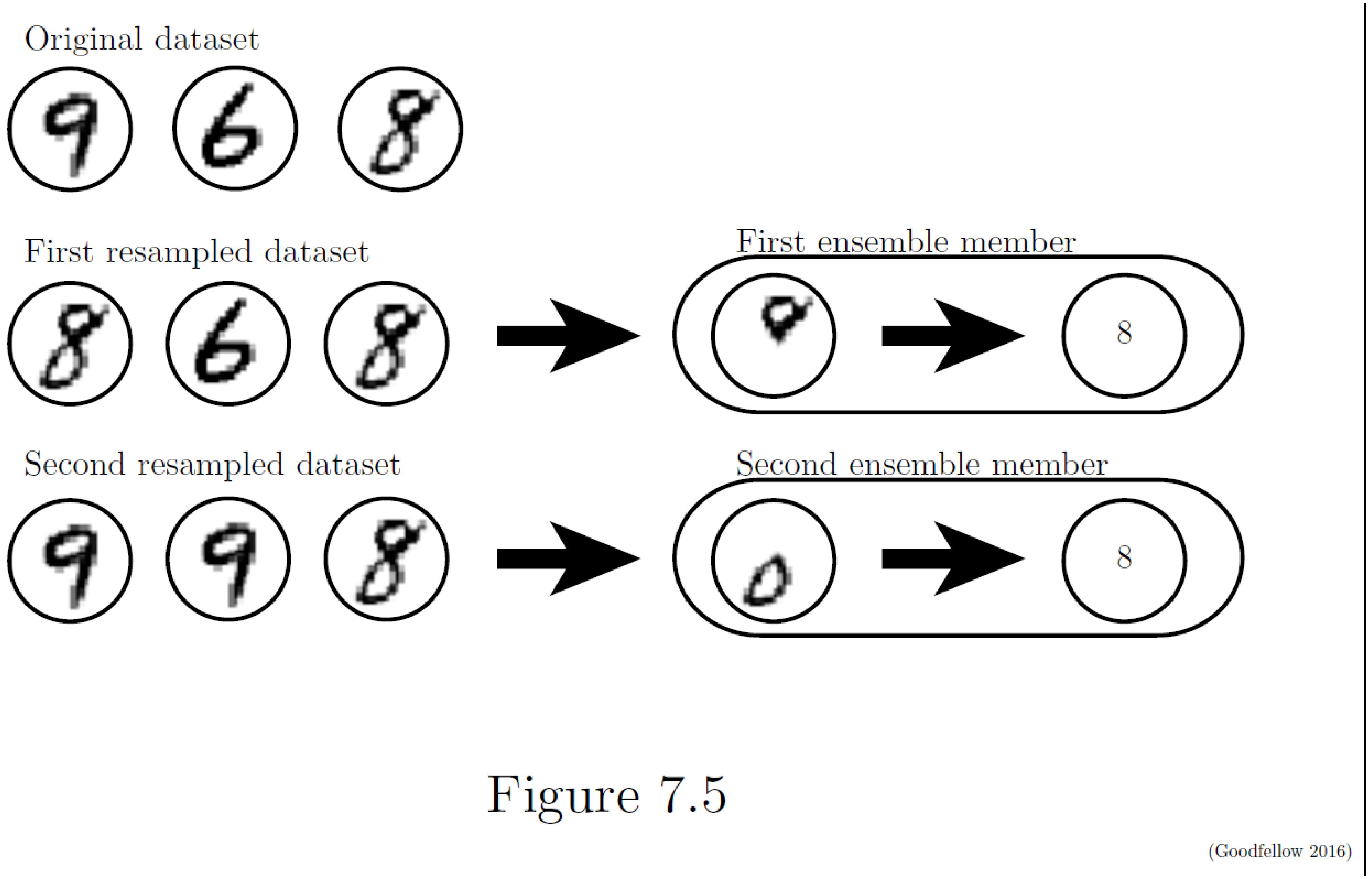


Figure 7.5

Goodfellow et al. 2016

Quiz:

[ ] If you want to win a machine learning competition,
it is better to average the prediction on new data
over ten different models, rather than just using the model
that is best on your validation data.

[ ] If you want to win a machine learning competition,
it is better to hand in 10 contributions (using different author names)
rather than a single contribution

Artificial Neural Networks: Lecture 4

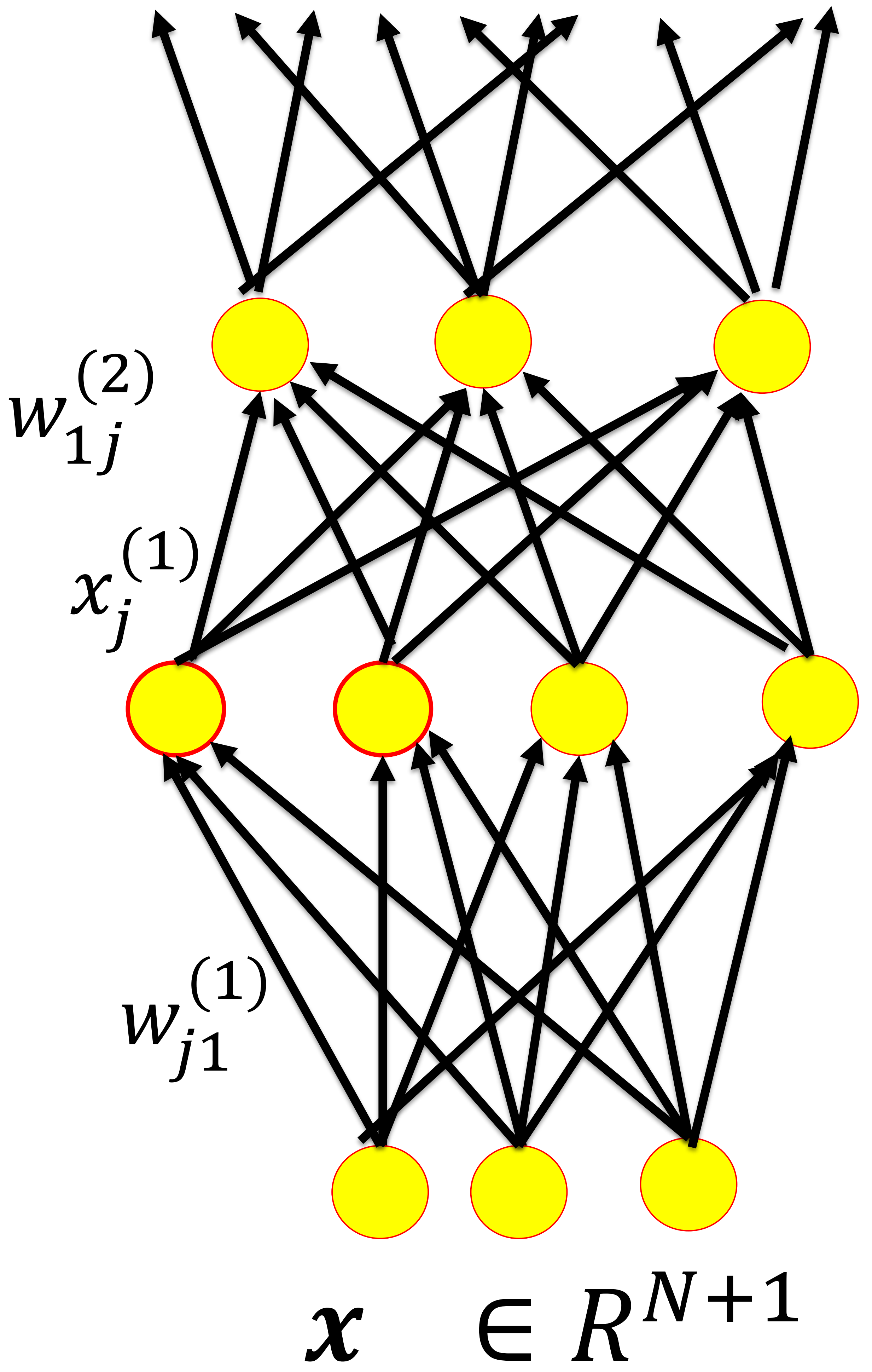
Tricks of the Trade in deep networks

Wulfram Gerstner

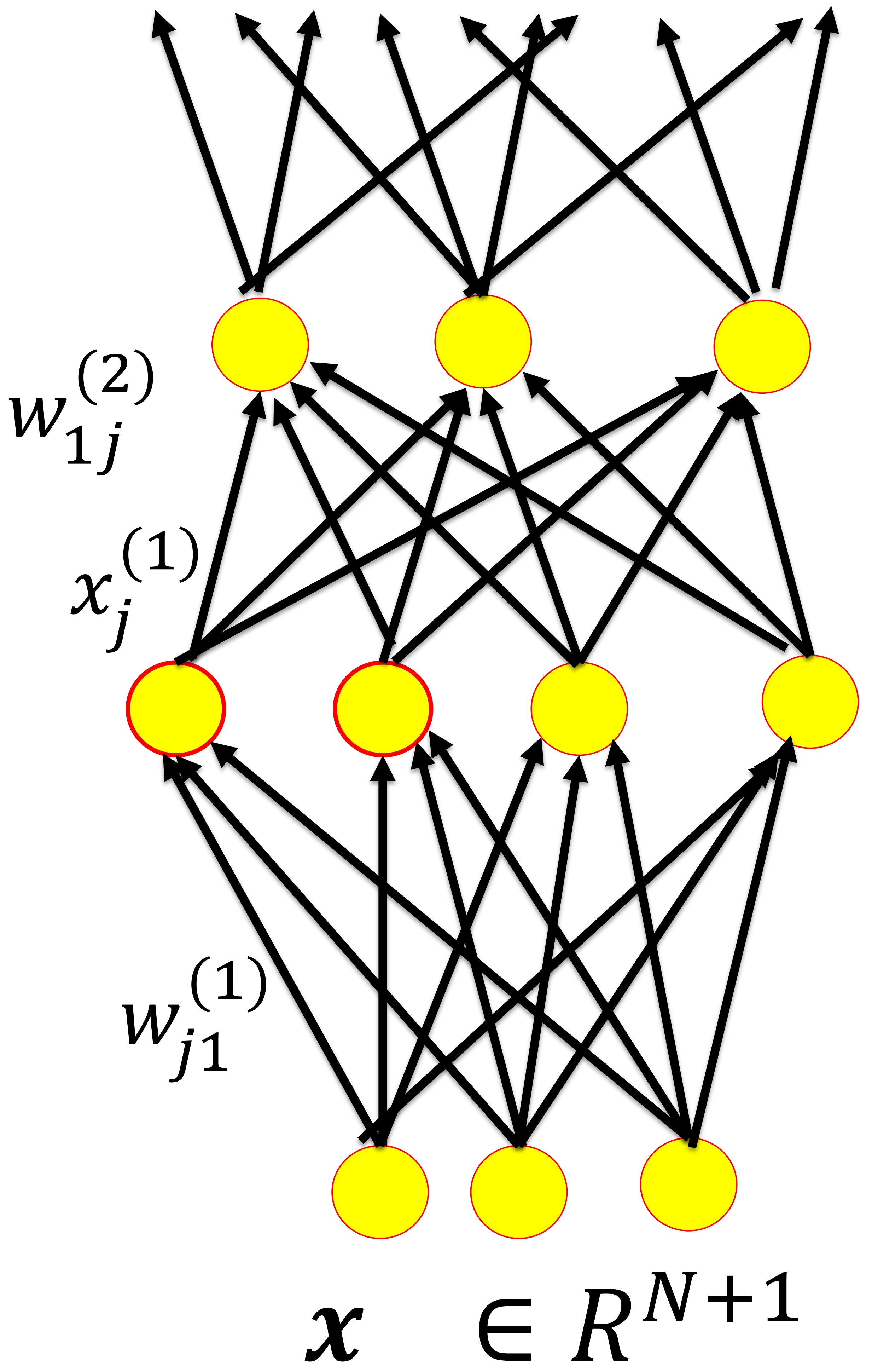
EPFL, Lausanne, Switzerland

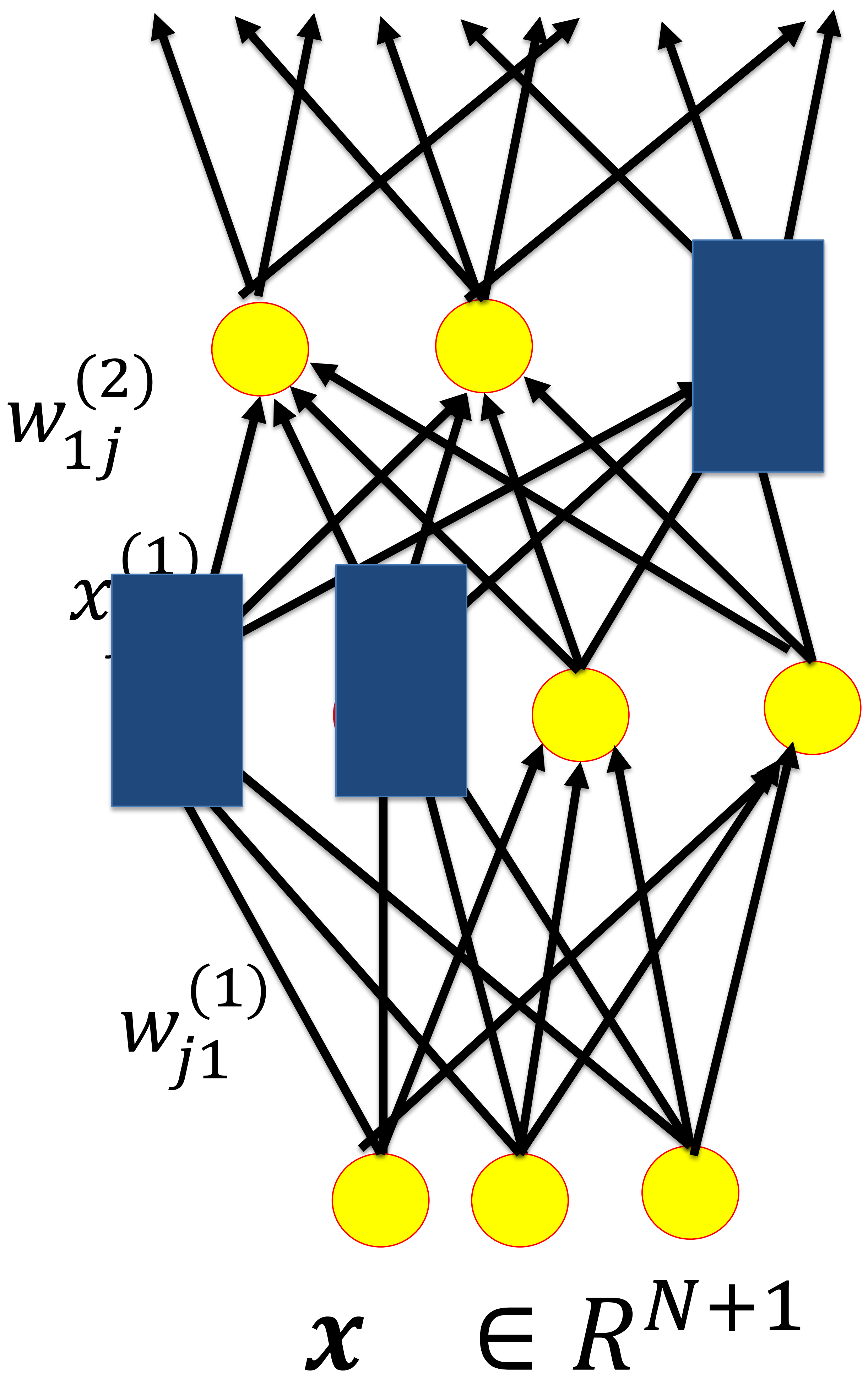
1. Bagging
2. Dropout

2. Dropout: suppress 50 percent of hidden units during training

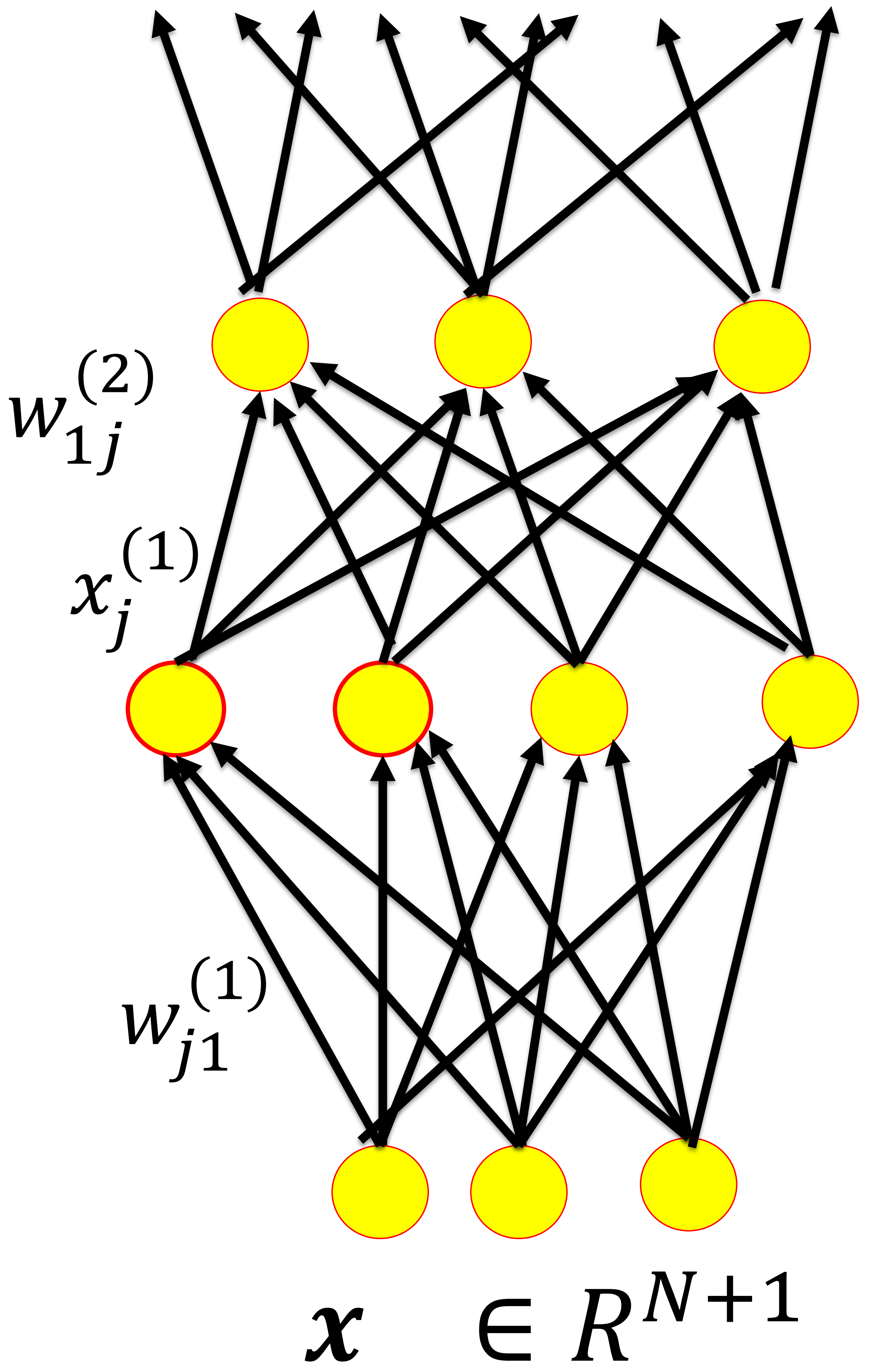


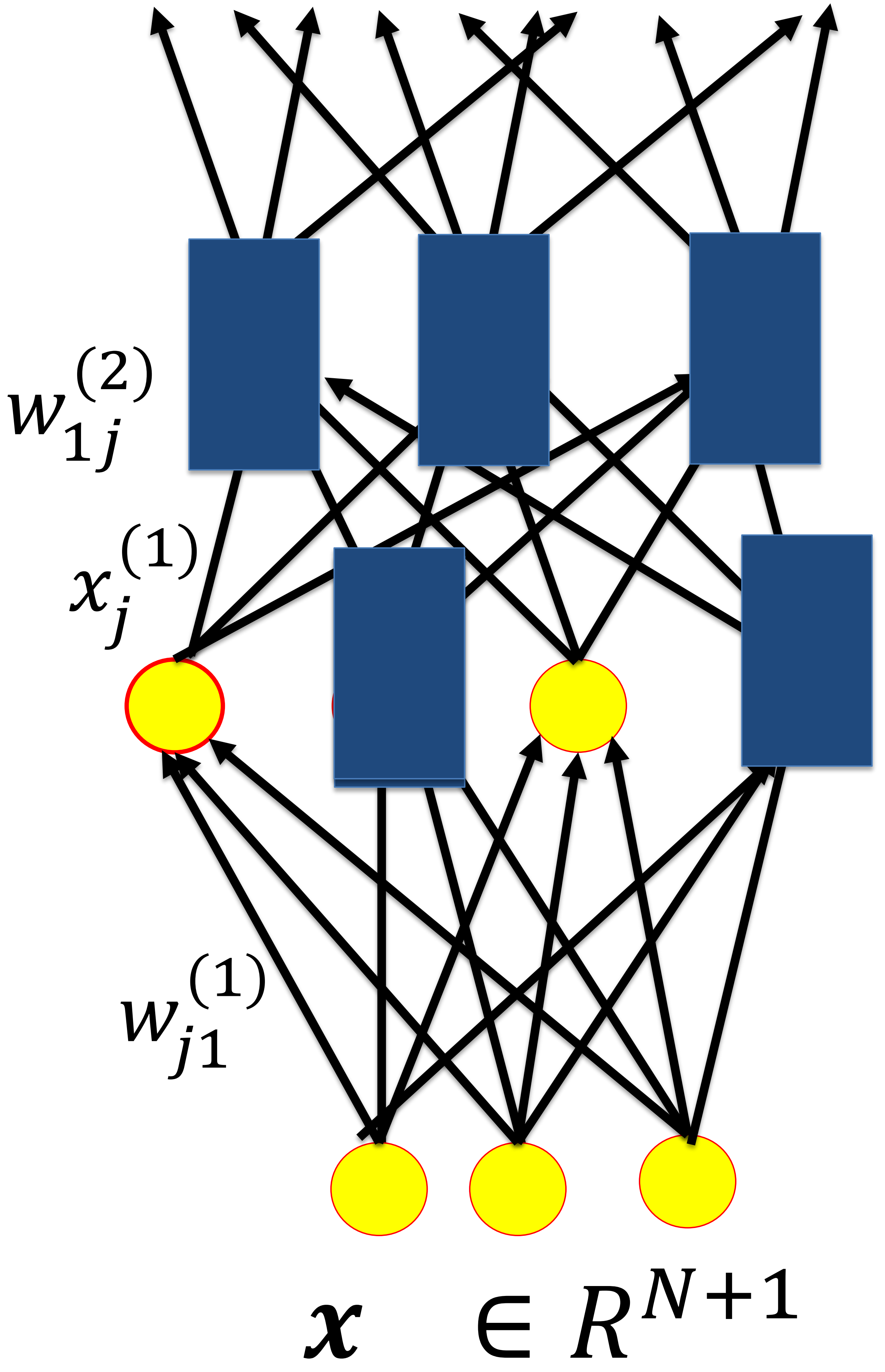
2. Dropout: suppress 50 percent of hidden units during training



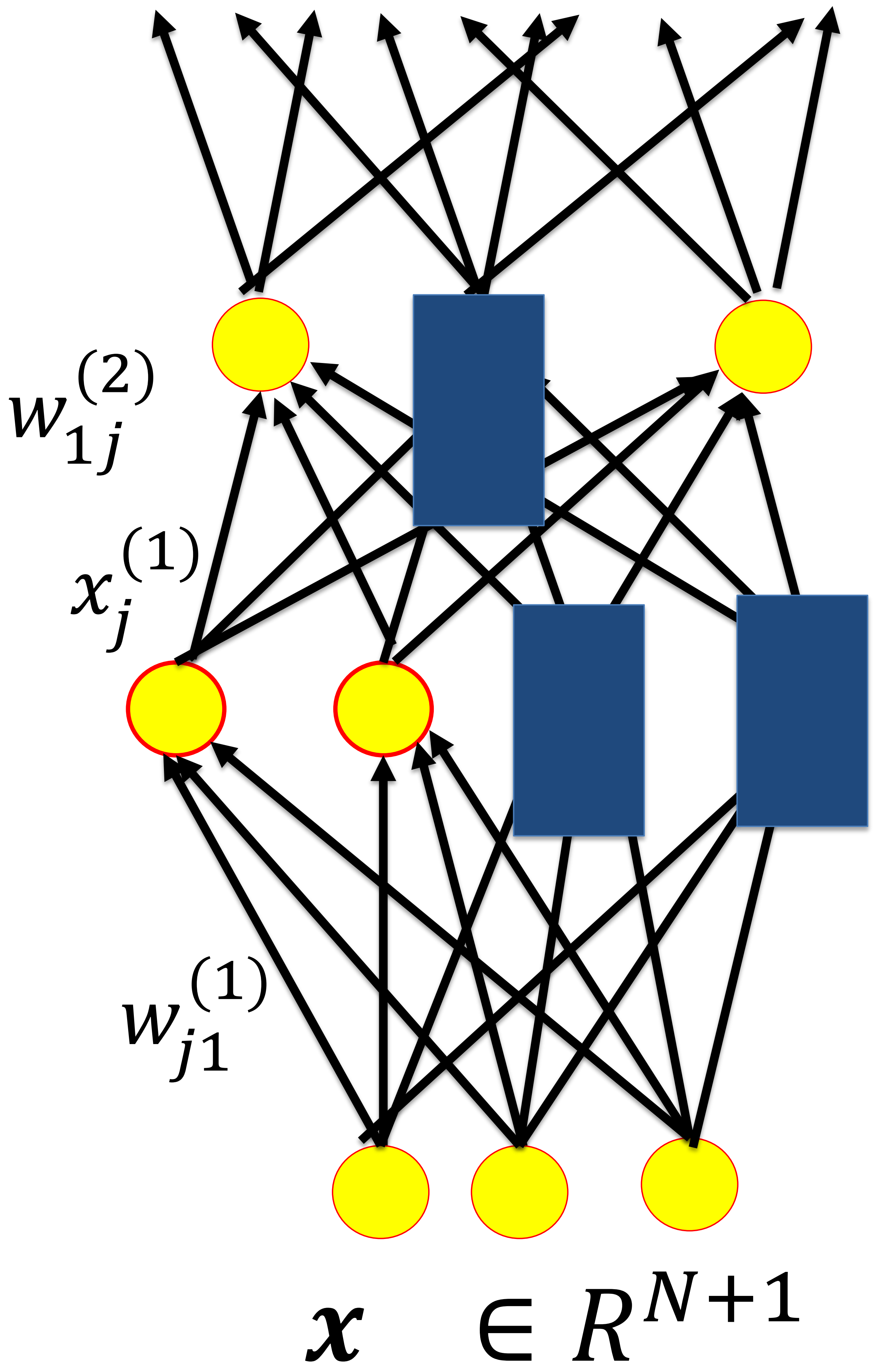


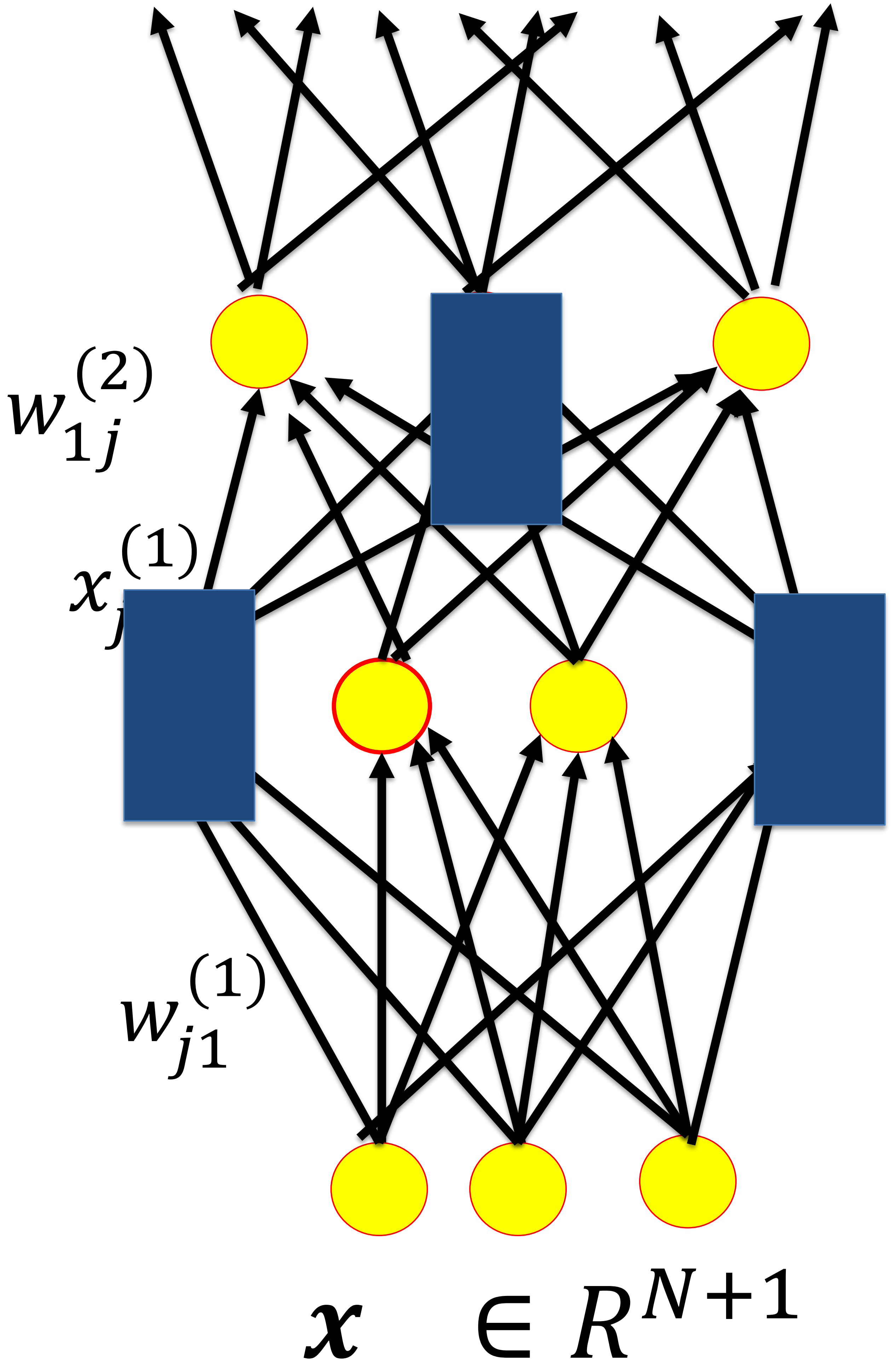
2. Dropout: suppress 50 percent of hidden units during training



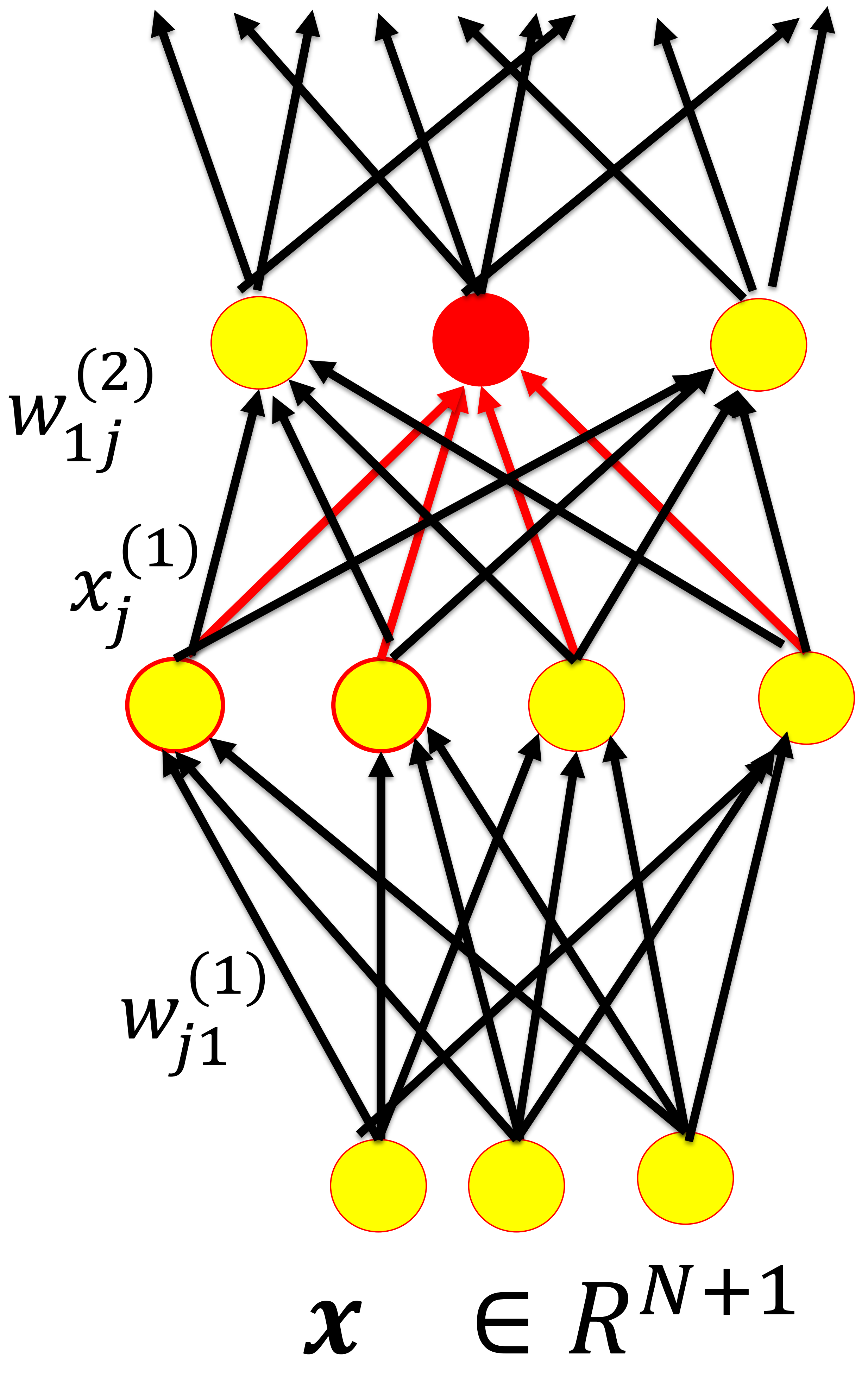


2. Dropout: suppress 50 percent of hidden units during training





2. Dropout: use full network for validation and test



For test:

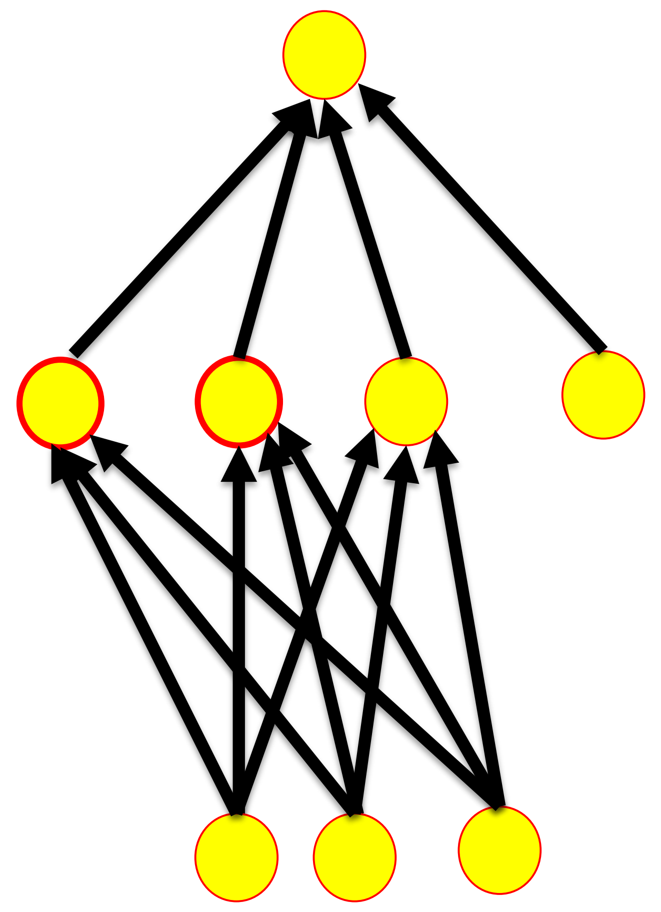
- full network
- but multiply output weights from hidden units by 1/2

→ Total input to each unit is roughly same as during training

2. Dropout: two different interpretations

1. An approximate, but practical implementation of bagging

2. A tool to enforce representation sharing in the hidden neurons



2. Dropout as approximate bagging

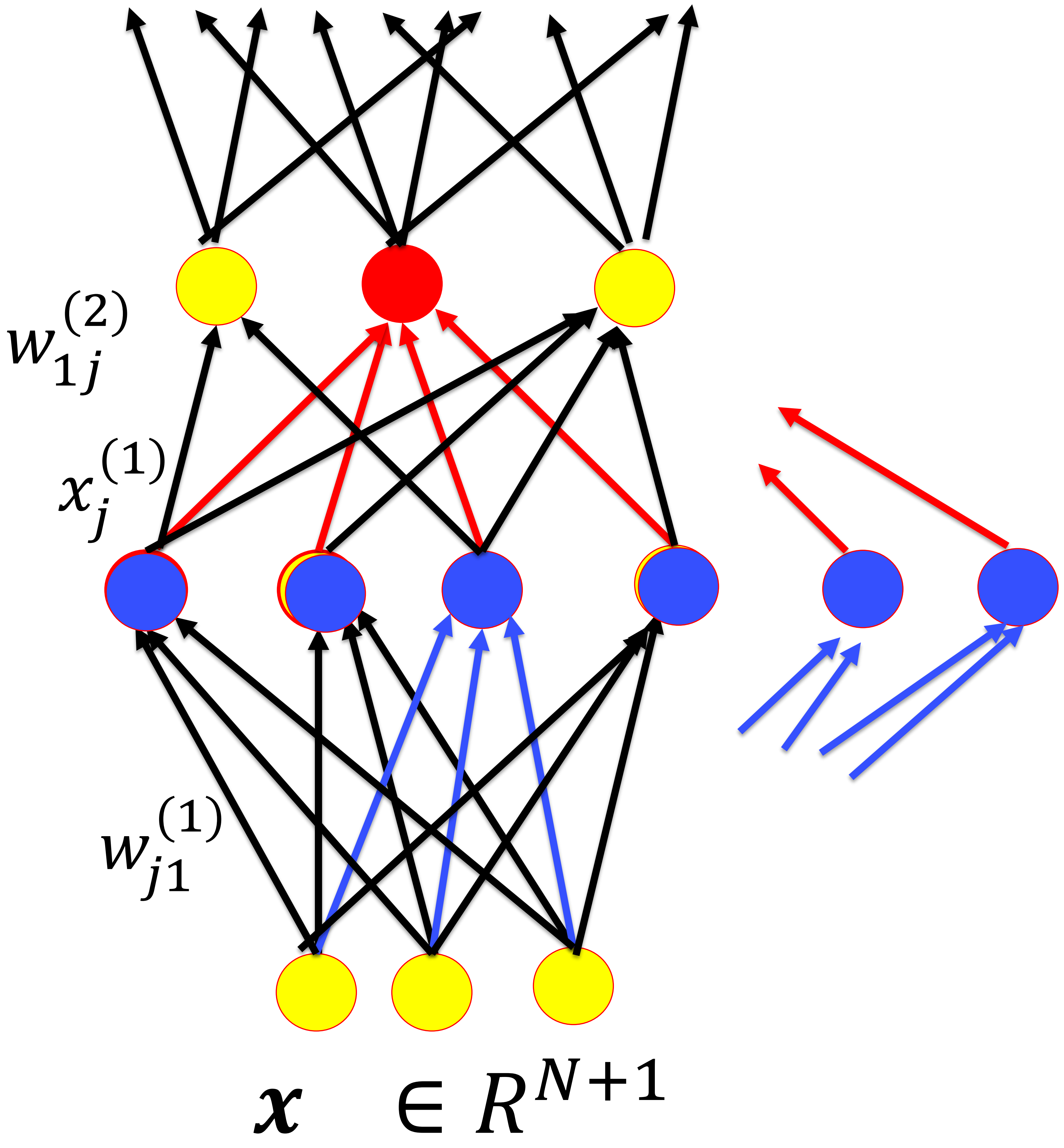
Dropout can be seen as a practical application of the ideas of bagging to deep networks

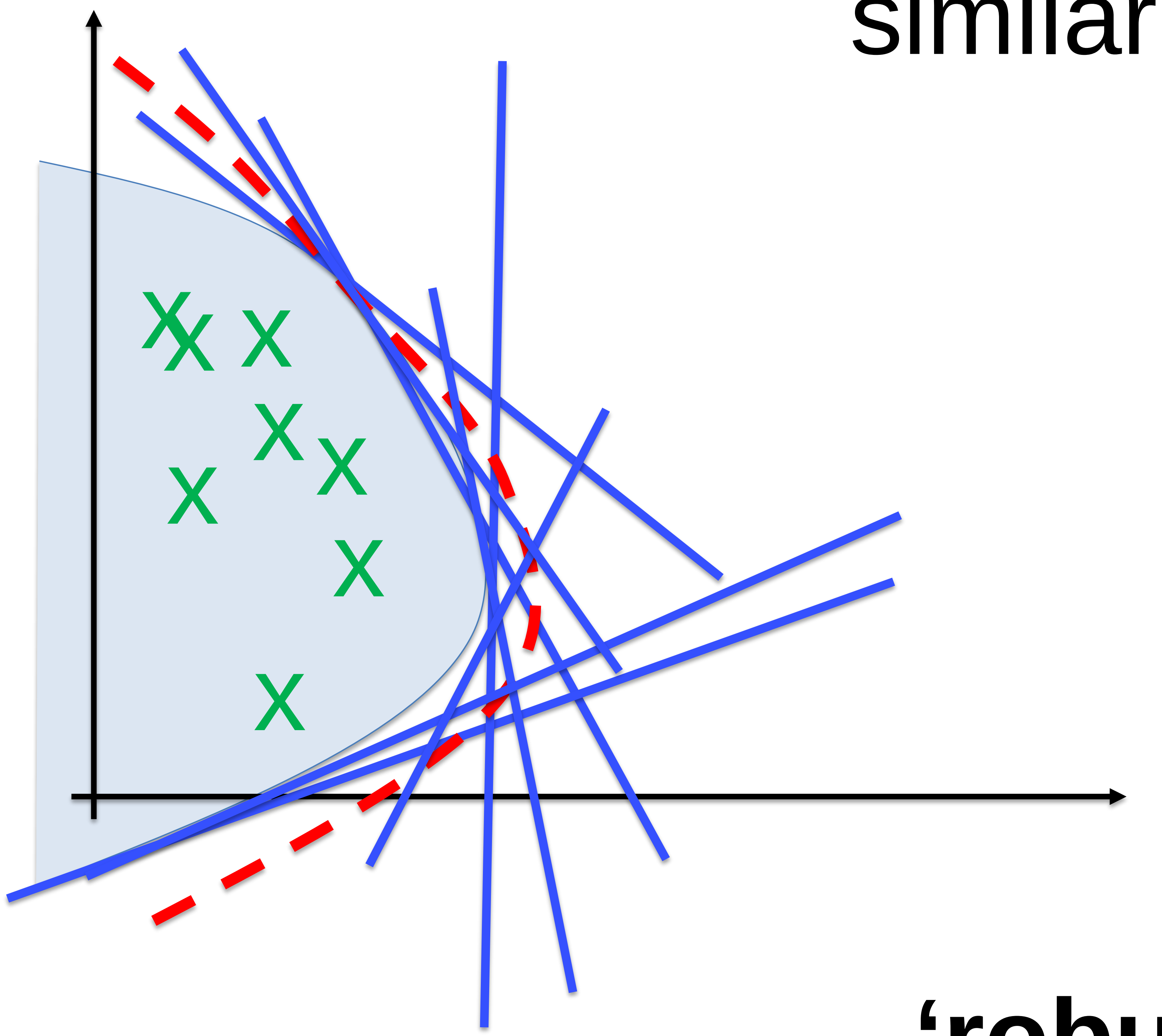
Differences to standard bagging:

- not a fixed data base for each ‘dropout’ configuration
- models not independent: share weights
- output not a ‘sum over model outputs’

2. Dropout as forced feature sharing

Feature sharing:
Take 2 times as many neurons,
But make sure they all solve
similar tasks



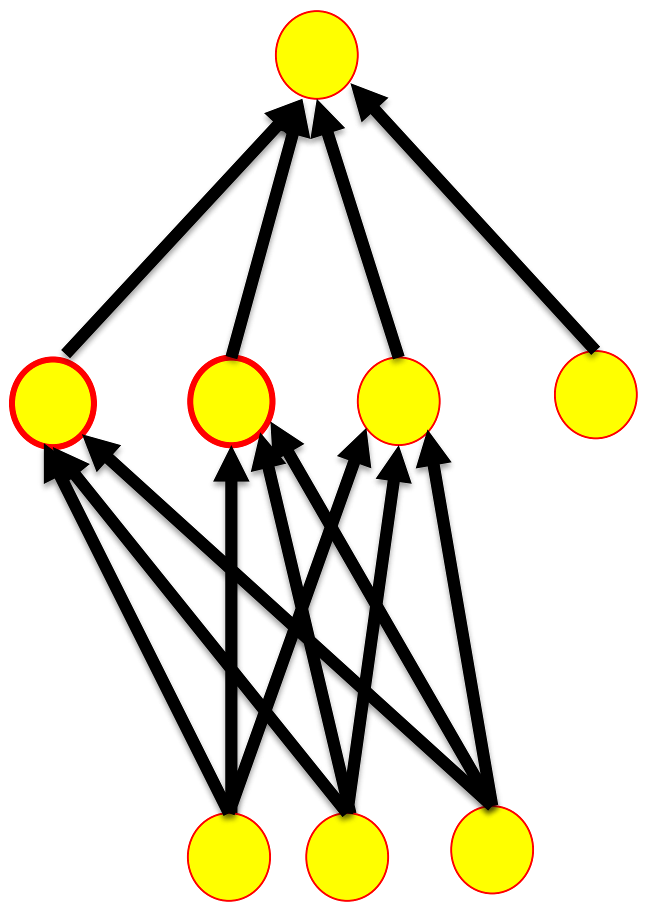


'robust'

2. Dropout: two different interpretations

1. An approximate, but practical
   implementation of bagging

2. A tool to enforce representation sharing
   in the hidden neurons



→ useful regularization method,
→ simple to implement

Wulfram Gerstner
EPFL, Lausanne, Switzerland

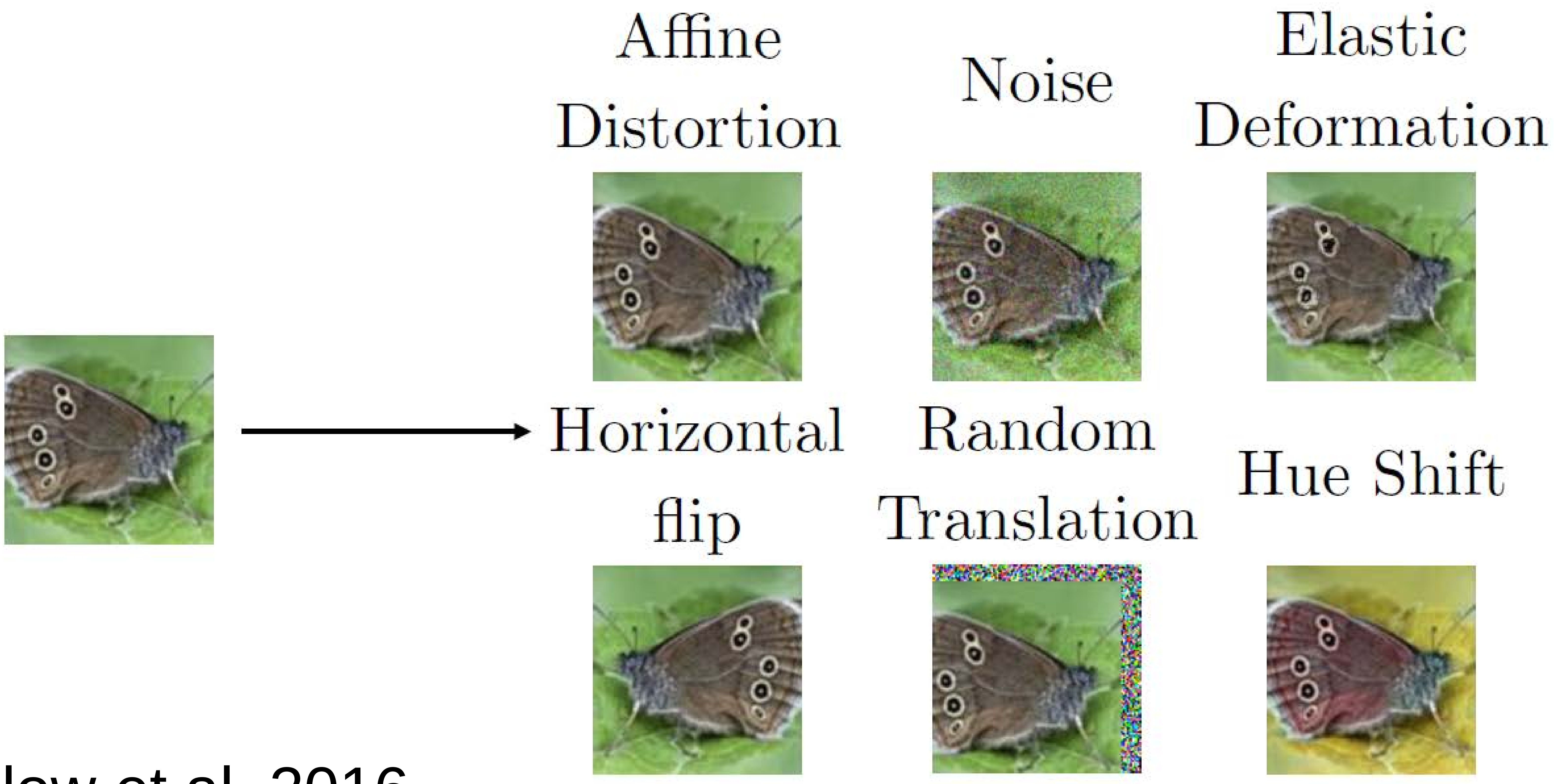
Artificial Neural Networks: Lecture 4

Tricks of the Trade in deep networks

1. Bagging
2. Dropout
3. Other simple regularization methods

3. Other easy regularization methods: dataset augmentation

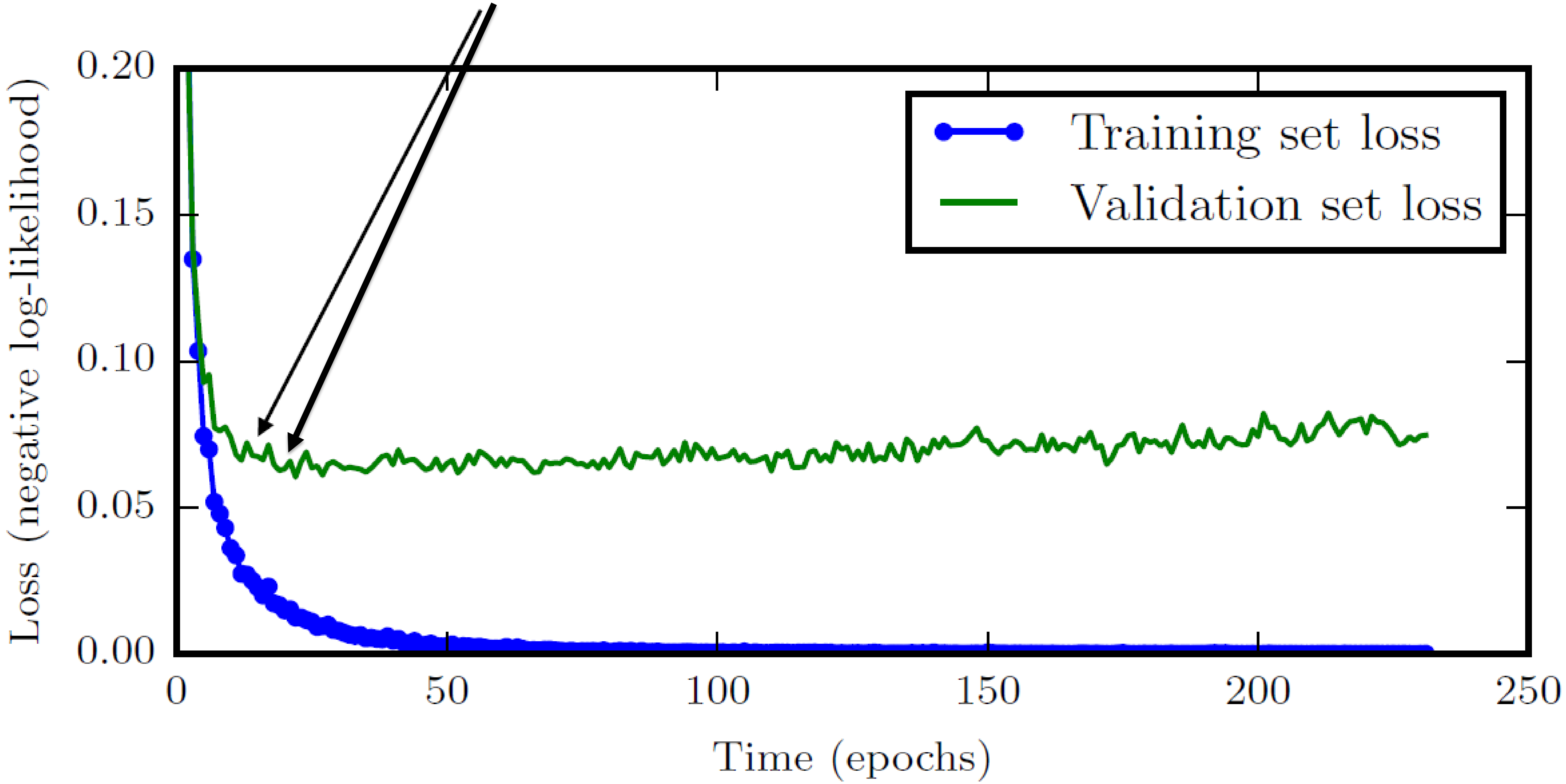
Dataset Augmentation



Goodfellow et al. 2016

3. Other easy regularization methods: early stopping

Go back to **weights** where validation error was minimal



Example: MNIST data base, see Goodfellow et al. 2016

Artificial Neural Networks: Lecture 4

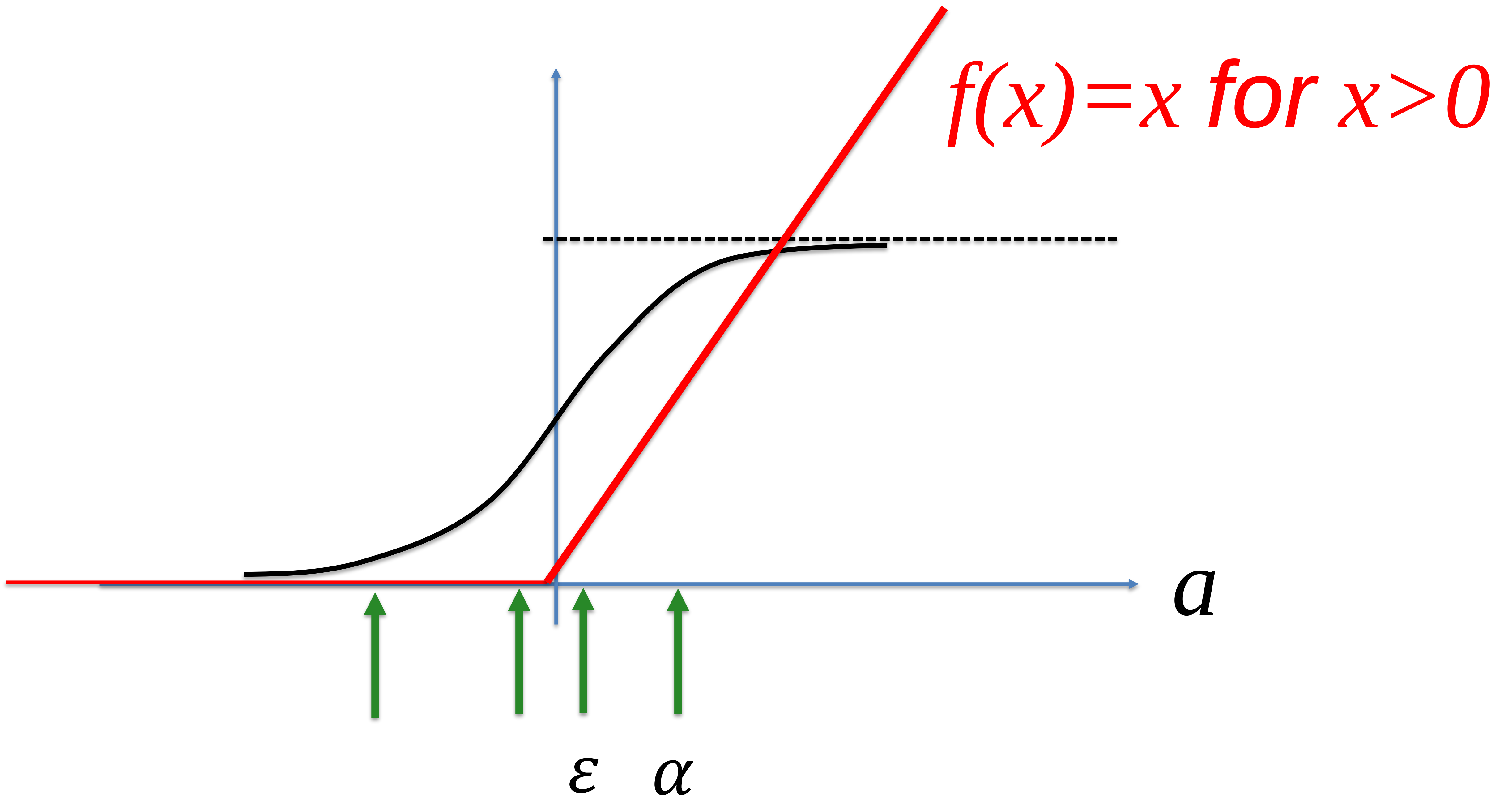
Wulfram Gerstner

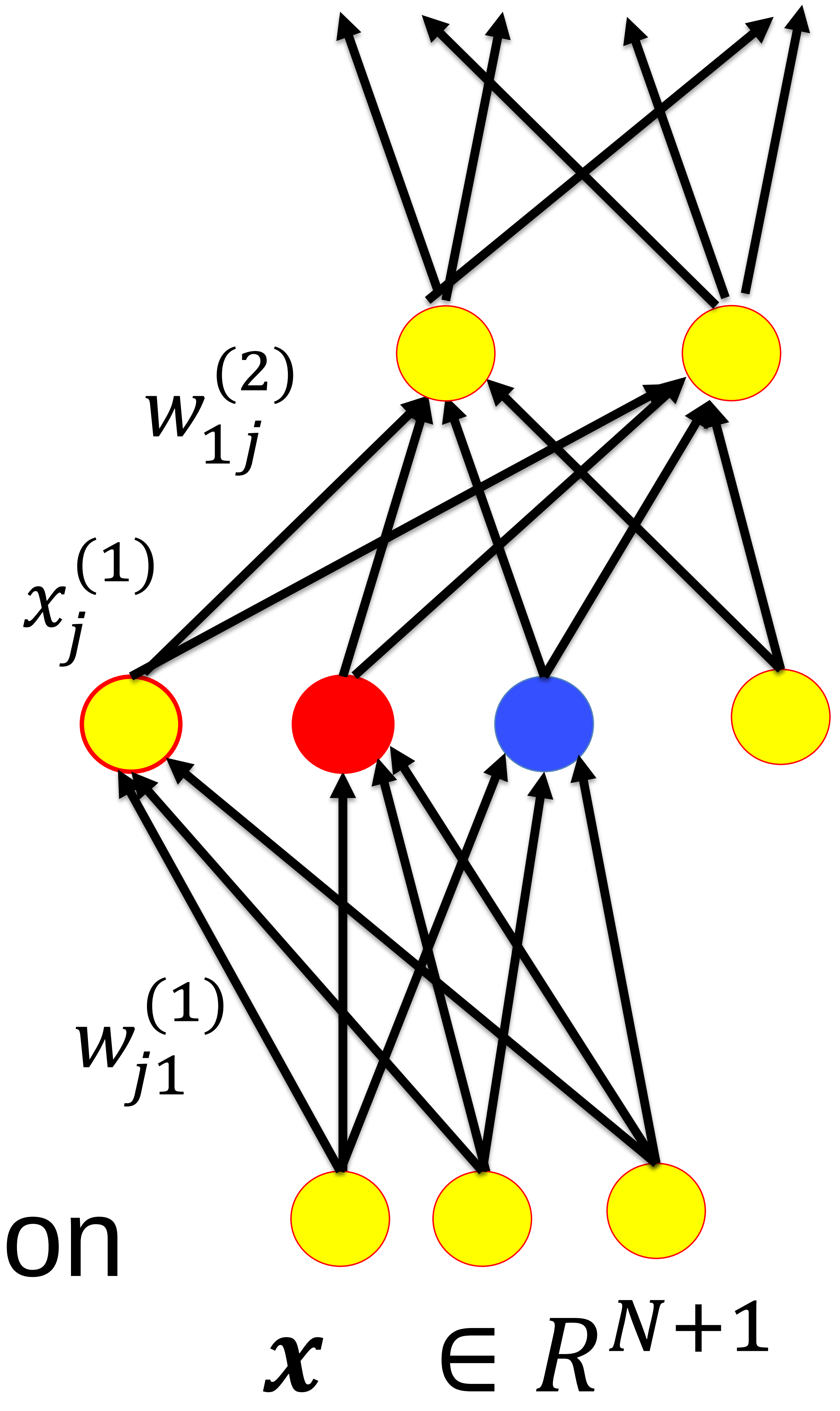
EPFL, Lausanne, Switzerland

Tricks of the Trade in deep networks

1. Bagging
2. Dropout
3. Other simple regularization methods
4. Weight initialization and choice of hidden units

4. Choice of units





-different patterns give different activation of same neuron (red)

-same input pattern gives different activation of different neurons (red, blue)

4. Initialization (input layer)   Blackboard

Normalization of data base:

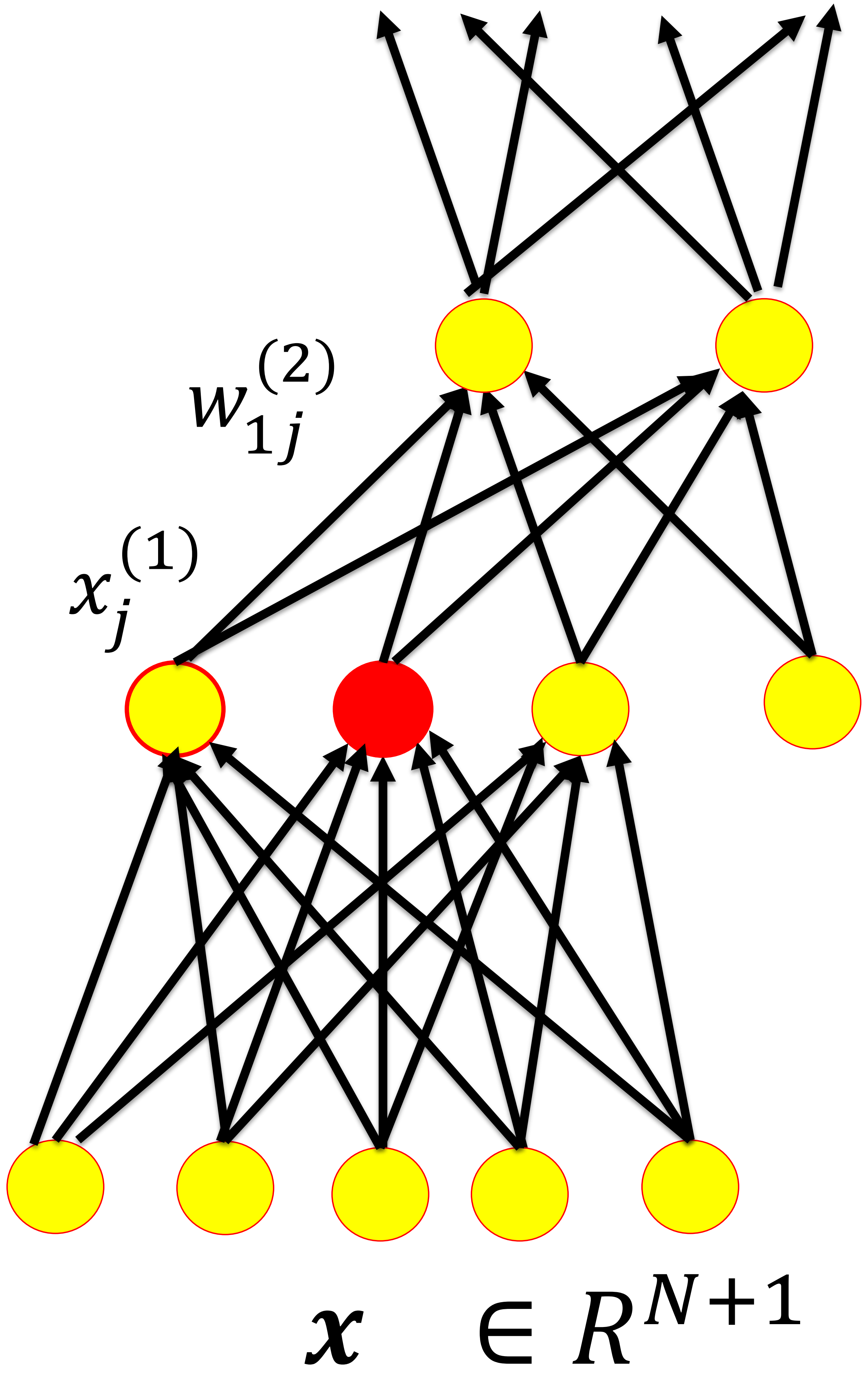
(1) $< x_j > = \frac{1}{P}\sum_{\mu=1}^{P} x_j^{\mu} = 0$

Random initialization of weights:

(2) $< w_{ij}^{(n)} > = 0$

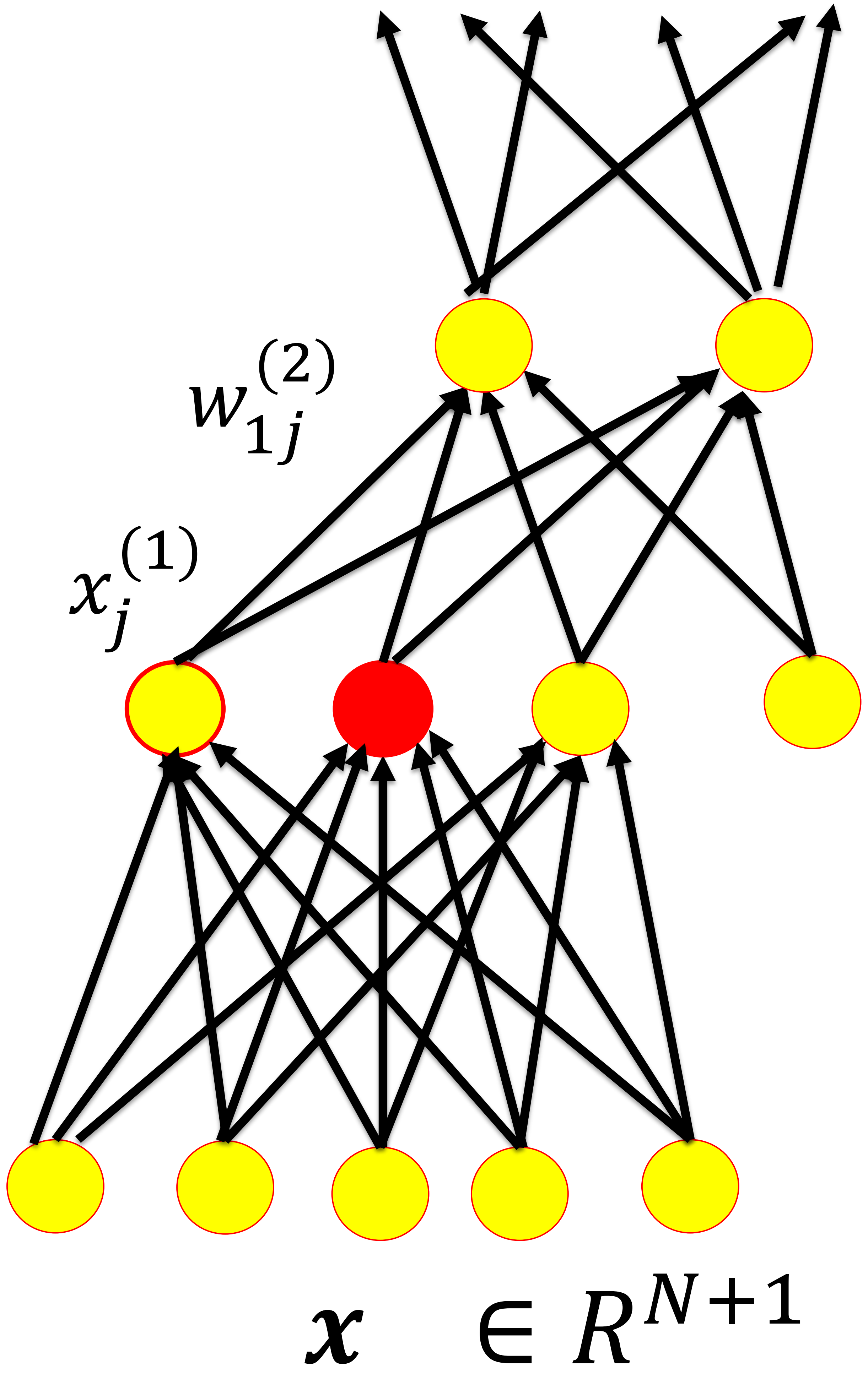
How should you choose the variance?

Claim: square root of *N* is important



Blackboard: Initialization

Claim: square root of *N* is important



4. Initialization (input layer)

Normalization of data base:

(1) $< x_j >= \frac{1}{P} \sum_{\mu=1}^{P} x_j^{\mu}$ =0

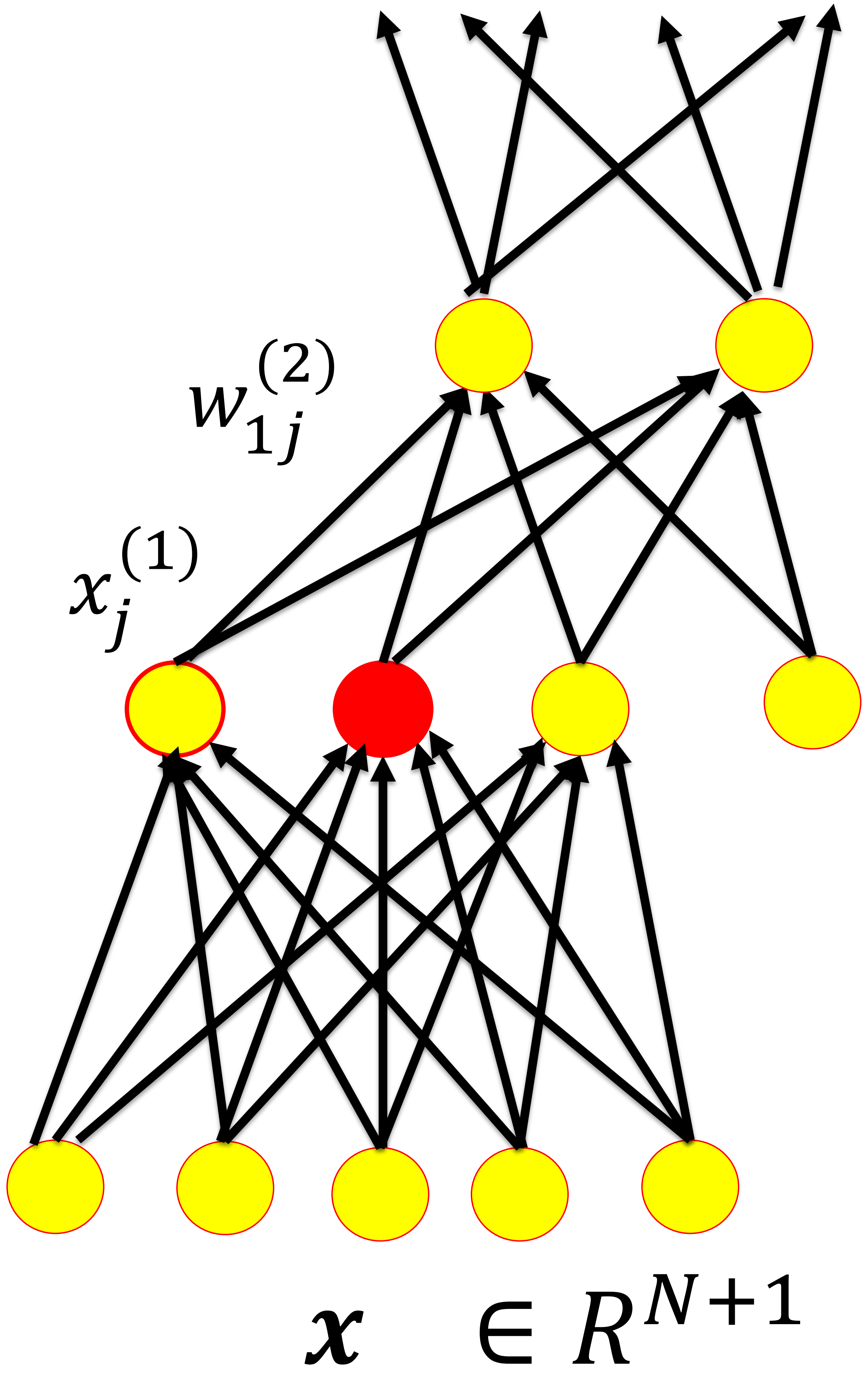
Random initialization of weights:

(2) $< w_{ij}^{(1)} > = 0$

And standard deviation propto $1/\sqrt{N}$

→ **Distribution of $x_j^{(1)}$ in layer 1**

→ **Distribution of $x_j^{(k)}$ in layer k**



BackProp

0. Initialization of weights

1. Choose pattern $\mathbf{x}^\mu$

   input $x_k^{(0)} = x_k^\mu$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \tag{1}$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \tag{2}$$

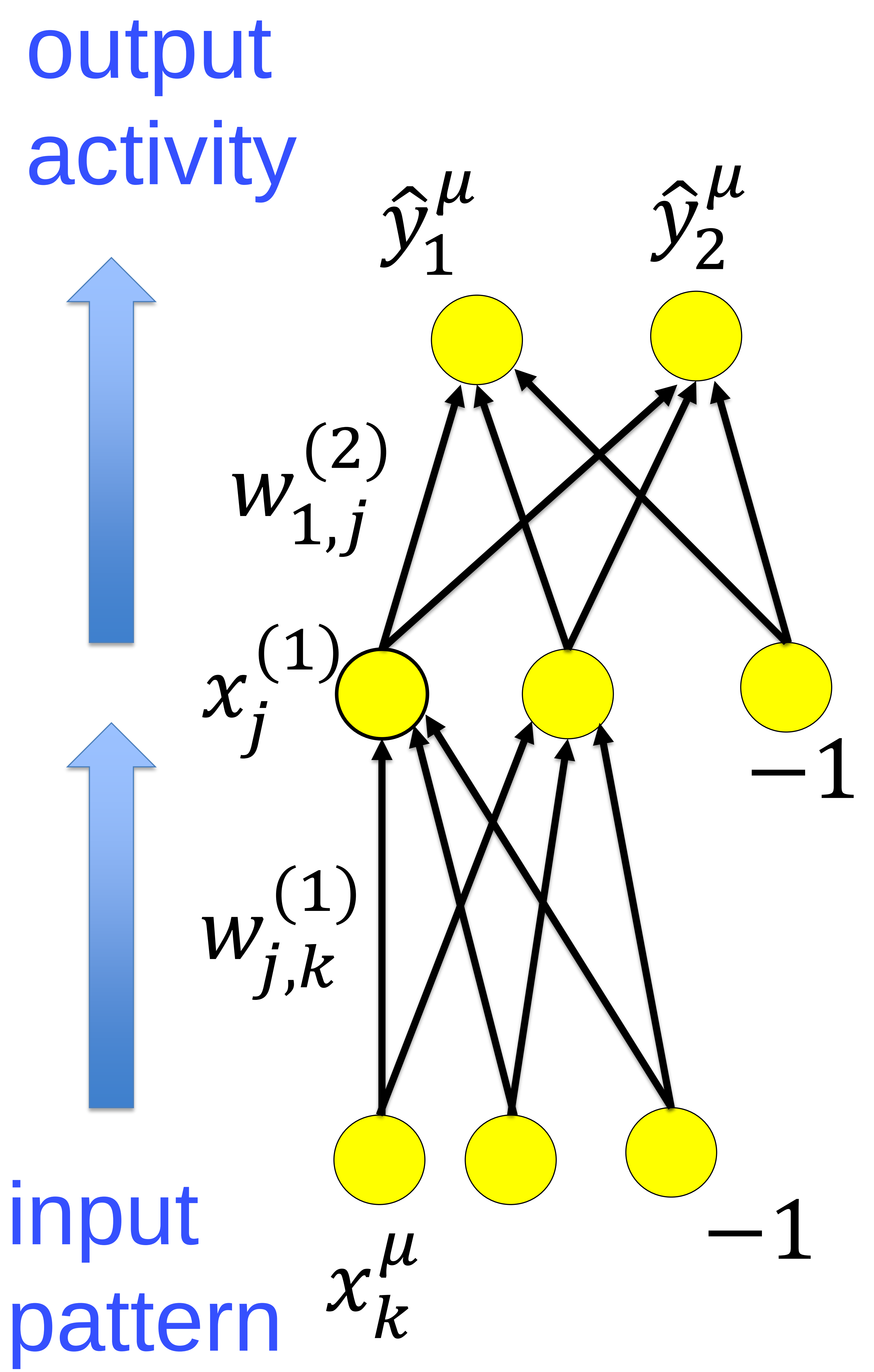
4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \tag{3}$$

5. Update weights (for each (i,j) and all layers (n))

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \tag{4}$$

6. Return to step 1.



BackProp

0. Initialization of weights

1. Choose pattern $\mathbf{x}^\mu$

input $x_k^{(0)} = x_k^\mu$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \quad (1)$$

output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \quad (2)$$

4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

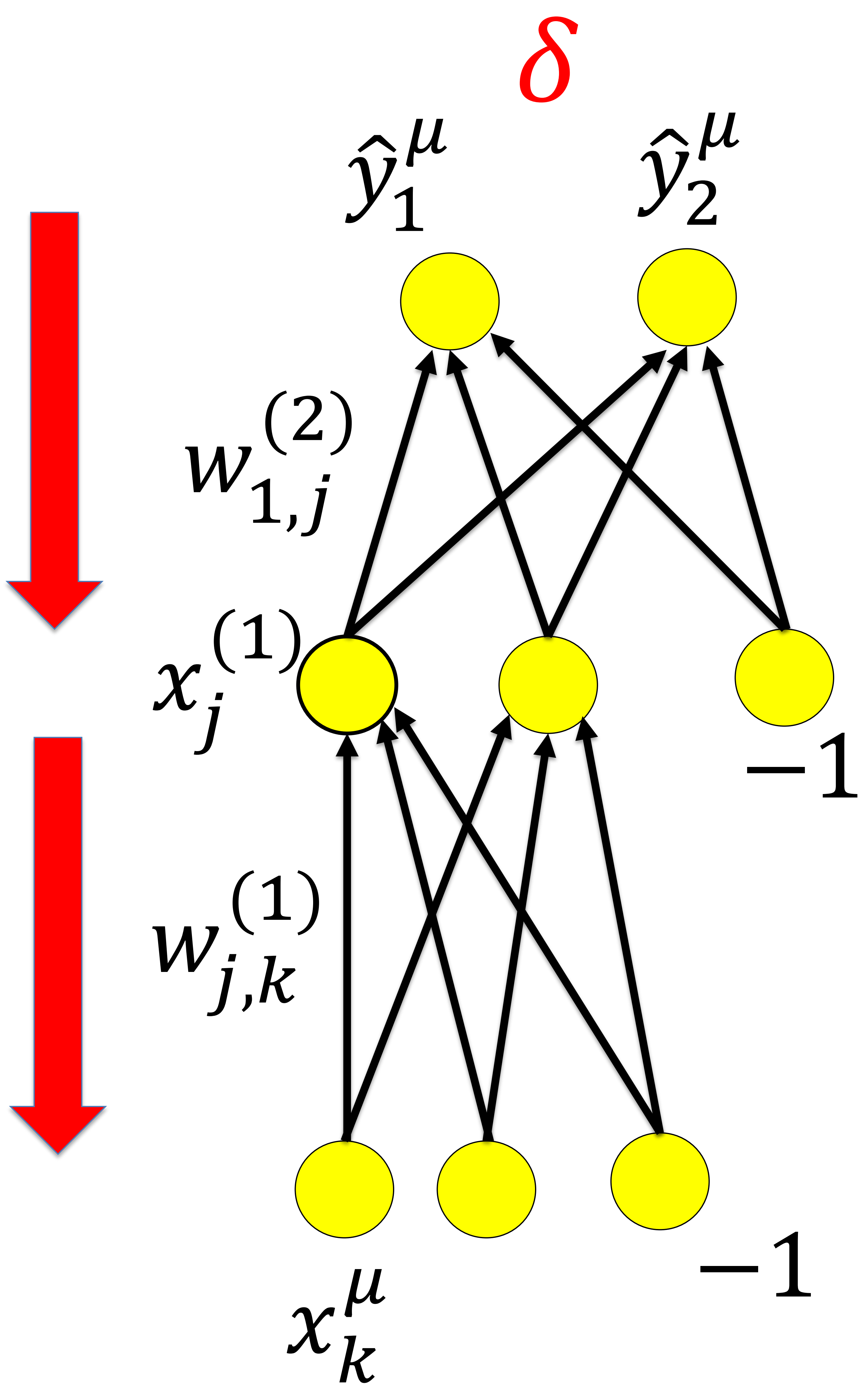
$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \quad (3)$$

5. Update weights (for each (i,j) and all layers (n))

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \quad (4)$$

6. Return to step 1.

Calculate output error



BackProp

0. Initialization of weights

1. Choose pattern $\mathbf{x}^\mu$

   input $x_k^{(0)} = x_k^\mu$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \tag{1}$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})}) \; [t_i^\mu - \hat{y}^\mu] \tag{2}$$

4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij} \, \delta_i^{(n)} \tag{3}$$

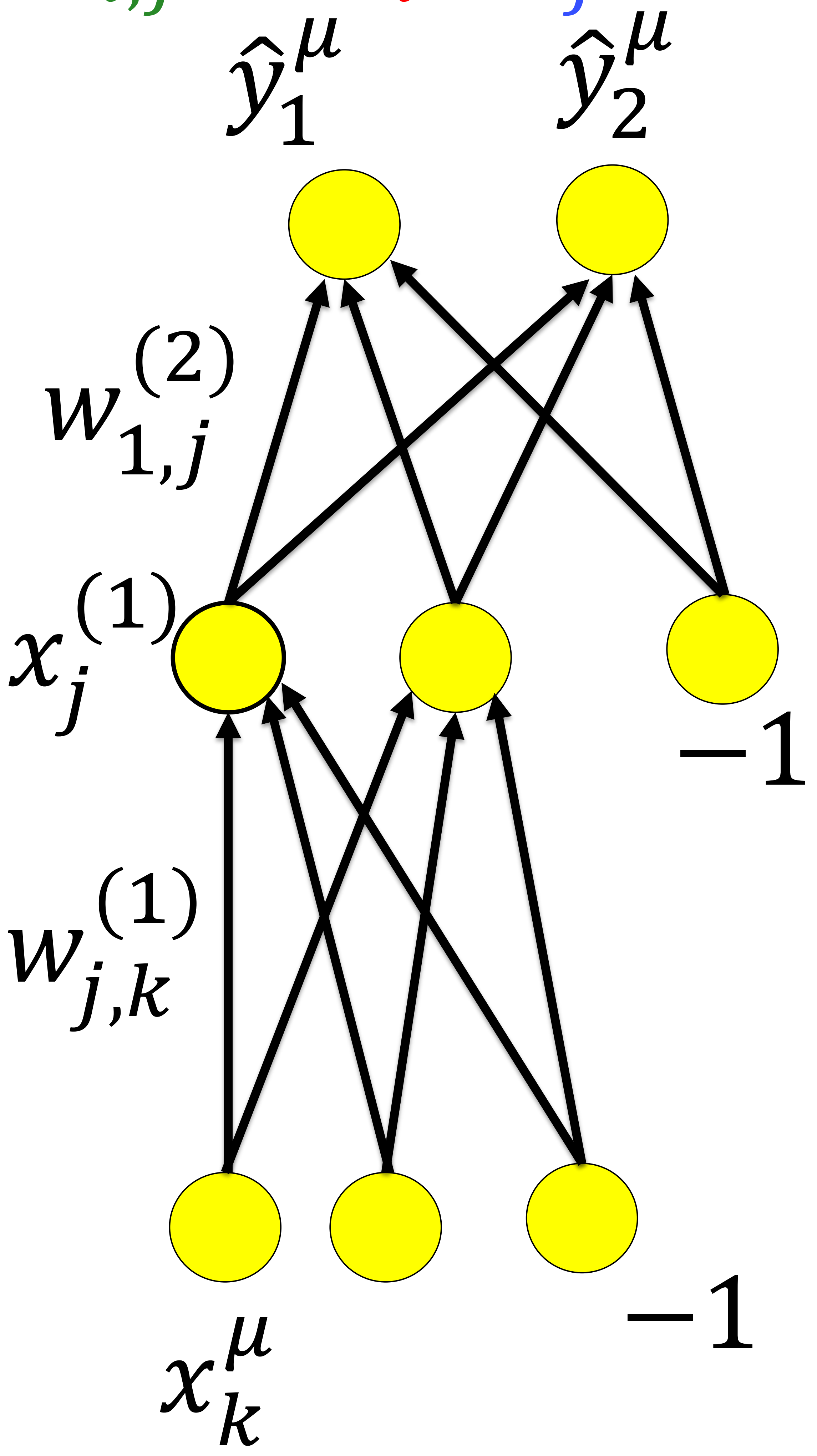
5. Update weights (for each (i,j) and all layers (n))

$$\Delta w_{ij}^{(n)} = \eta \, \delta_i^{(n)} \, x_j^{(n-1)} \tag{4}$$

6. Return to step 1.

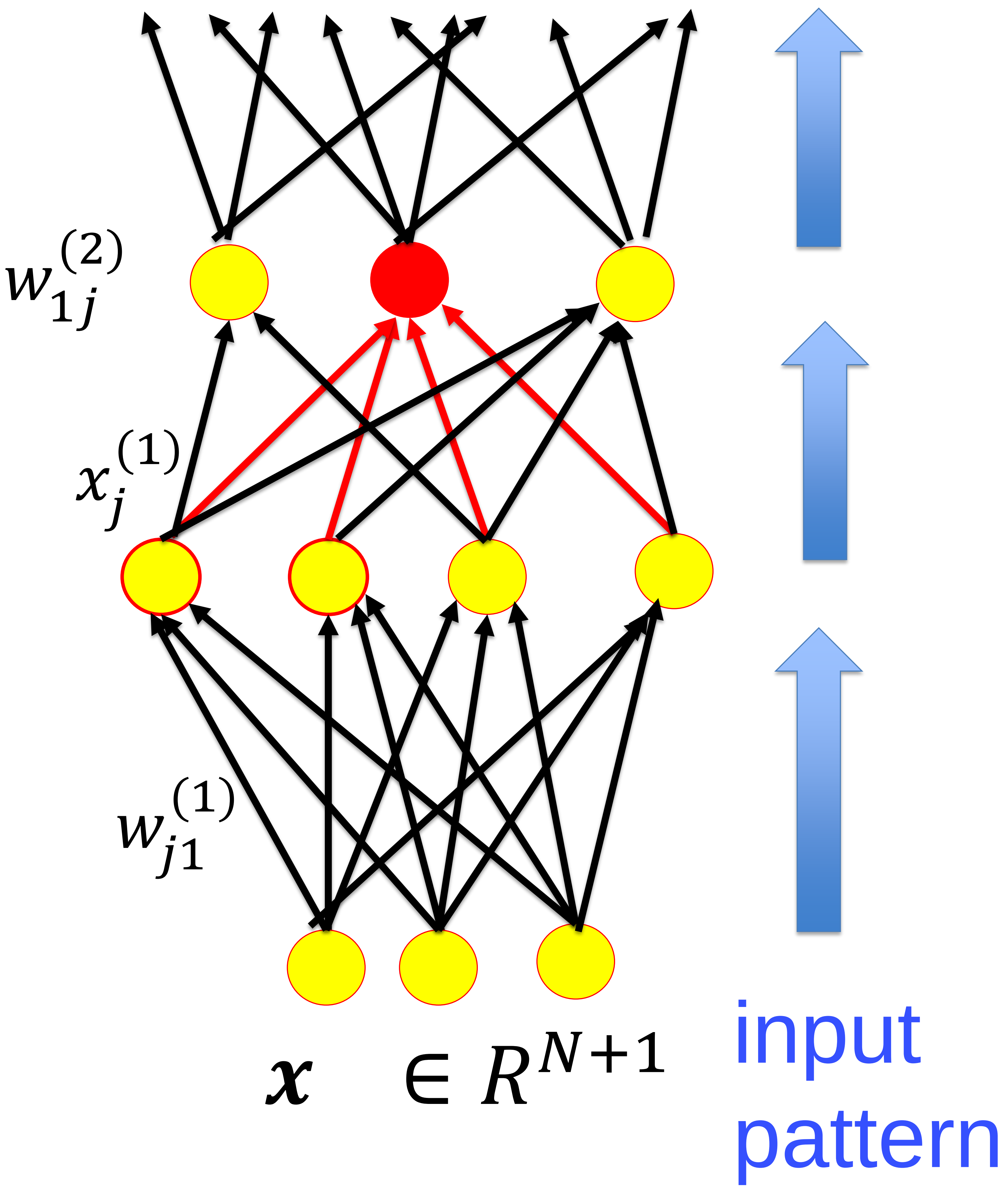
update **all** weights

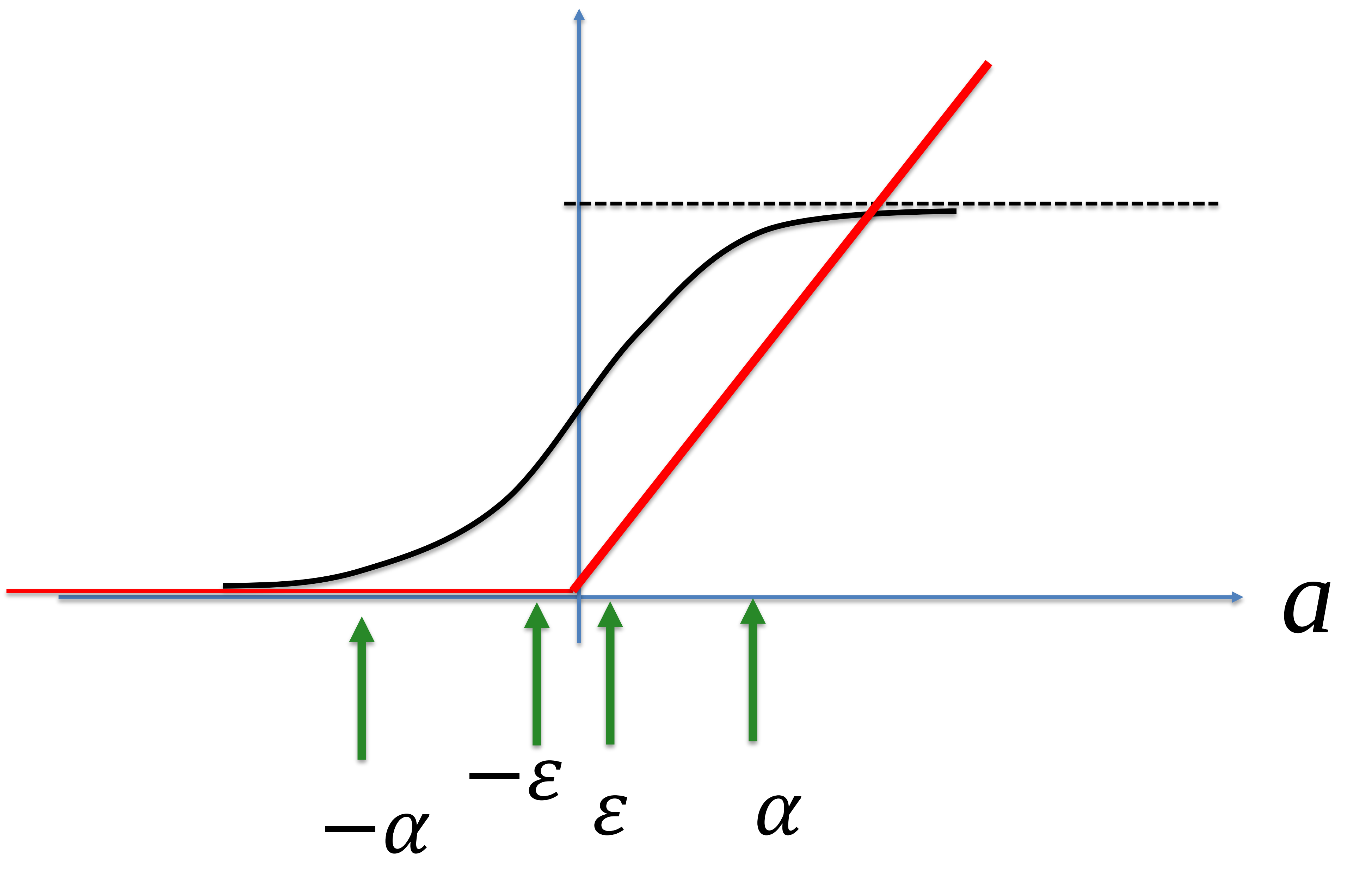
$$\Delta w_{i,j}^{(n)} = \delta_i^{(n)} x_j^{(n-1)}$$



Why does the initatialization or normalization matter in backprop?

4. Forward pass: Linear and nonlinear processing



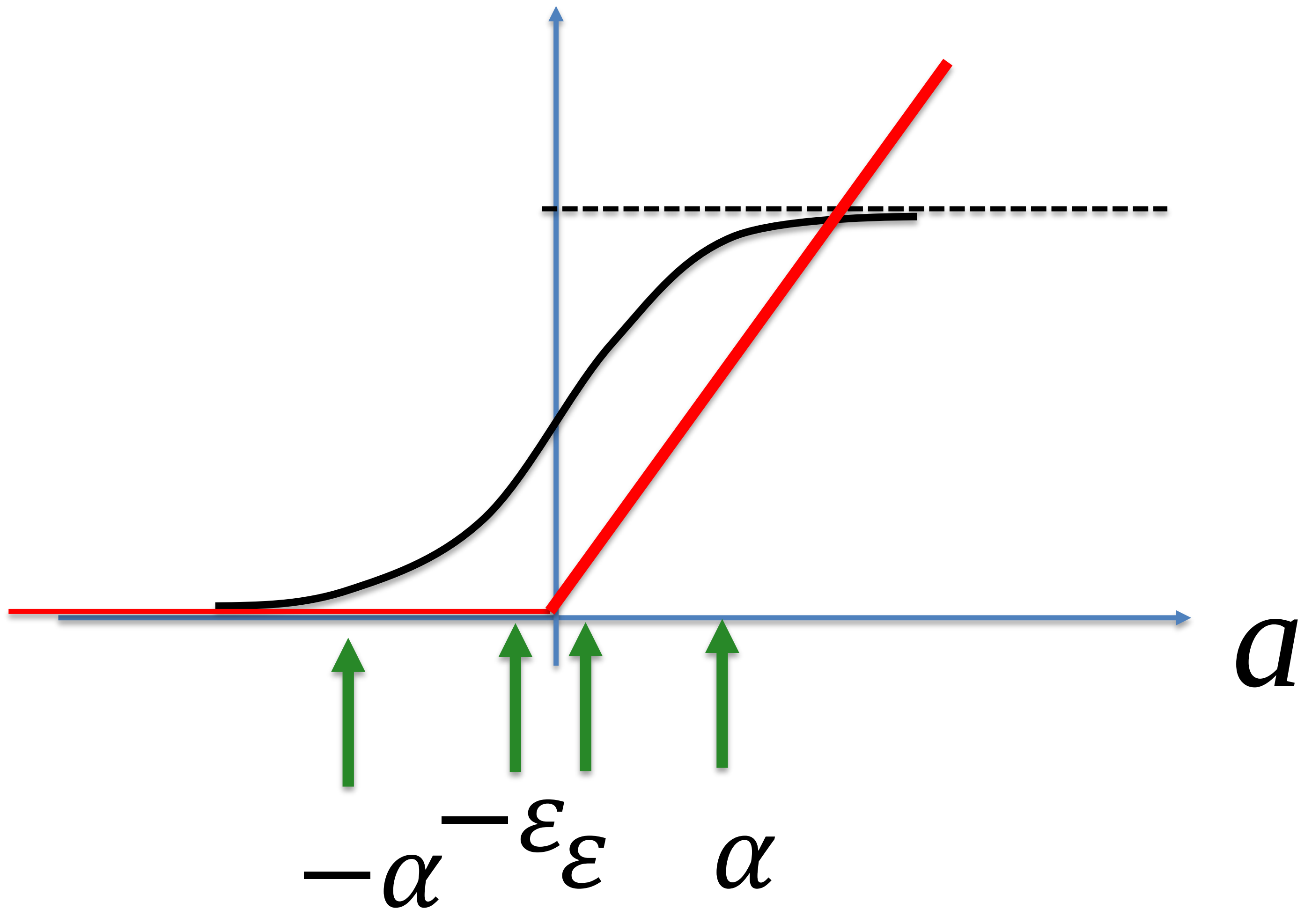


4. Forward pass: Linear and nonlinear processing

Observations:

if all patterns in all layers touch the linear regime of $g(a)$, then the whole network is linear

→ different patterns should touch different regions of $g(a)$.



- this is automatically true for ReLu, if the mean (across patterns) is $a=0$
- this is automatically true for sigmoidals, if the variance (across patterns) is > 2

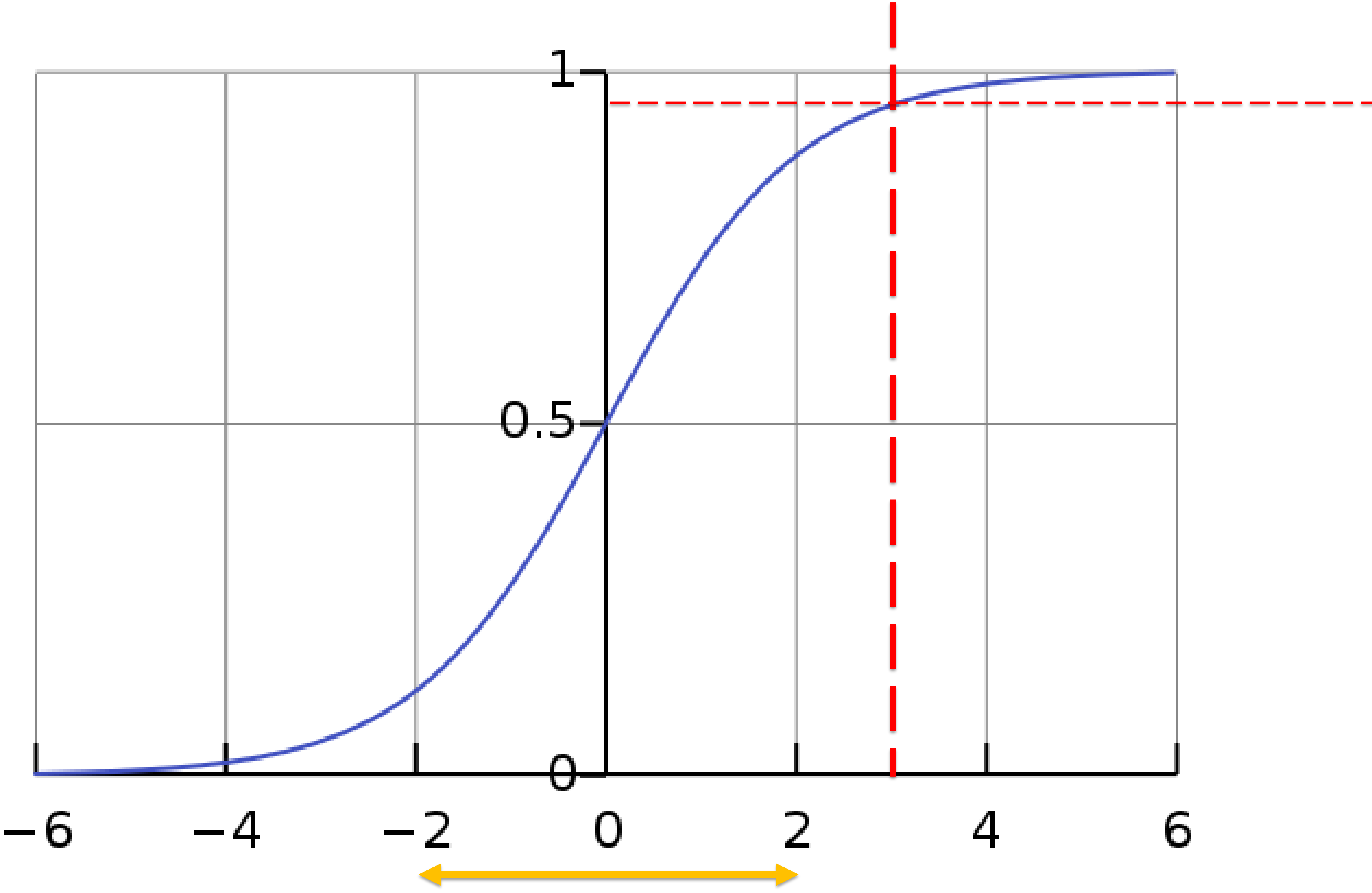
Review. sigmoidal output = **logistic function**

$$g(a) = \frac{1}{1+e^{-a}}$$

Rule of thumb:

for $a = 3$: $g(3) = 0.95$

for $a = -3$: $g(-3) = 0.05$



https://en.wikipedia.org/wiki/Logistic_function

4. Forward pass: exploit nonlinearities ('linearity problem')

To **exploit nonlinearities** of all units in the network, we must

1. Make sure that the **initialization** of weights is well chosen
   → expectation (across patterns) of the activation variable

$$0 = < a_j^{(n)} >; \; a_j^{(n)} = \sum_k w_{j,k}^{(n)} x_k^{(n-1)}$$

   → standard deviation of the activation variable

$a_j^{(n)}$ of order 1.

2. Make sure that **weight updates** do not shift mean (and standard deviation) of distribution too much

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Artificial Neural Networks: Lecture 4

Tricks of the Trade in deep networks

1. Bagging
2. Dropout
3. Other simple regularization methods
4. Choice of hidden units and initialization: ‘linearity problem’
5. Vanishing gradient problem

0. **Initialization of weights**

BackProp

1. **Choose pattern** $\mathbf{x}^{\mu}$

input $x_k^{(0)} = x_k^{\mu}$

2. **Forward propagation of signals** $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \quad (1)$$

output $\hat{y}_i^{\mu} = x_i^{(n_{\max})}$

3. **Computation of errors in output**

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})}) \; [t_i^{\mu} - \hat{y}^{\mu}] \quad (2)$$

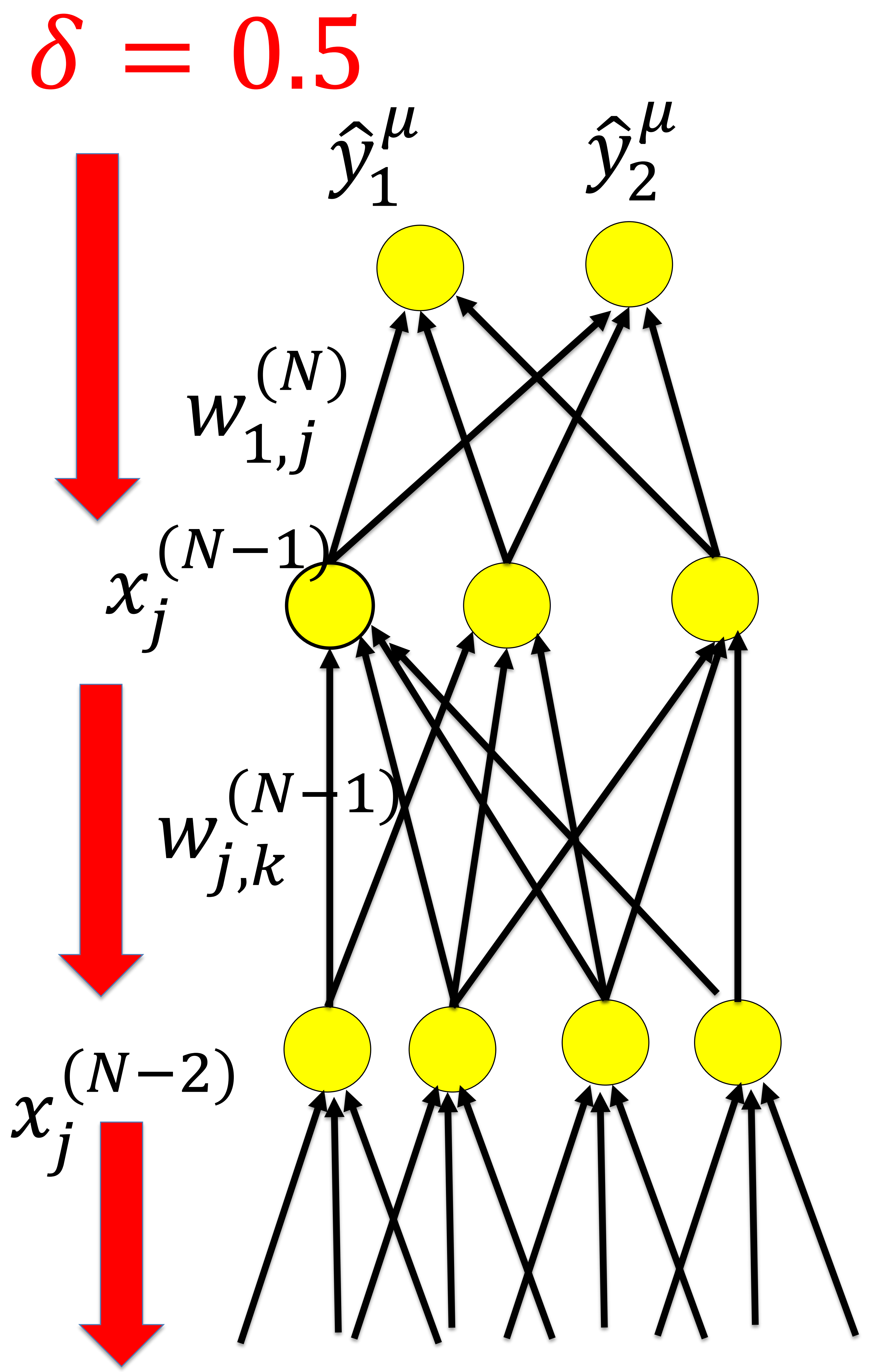
4. **Backward propagation of errors** $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij} \, \delta_i^{(n)} \quad (3)$$

5. **Update weights (for each** (i, j) **and all layers** (n)**)**

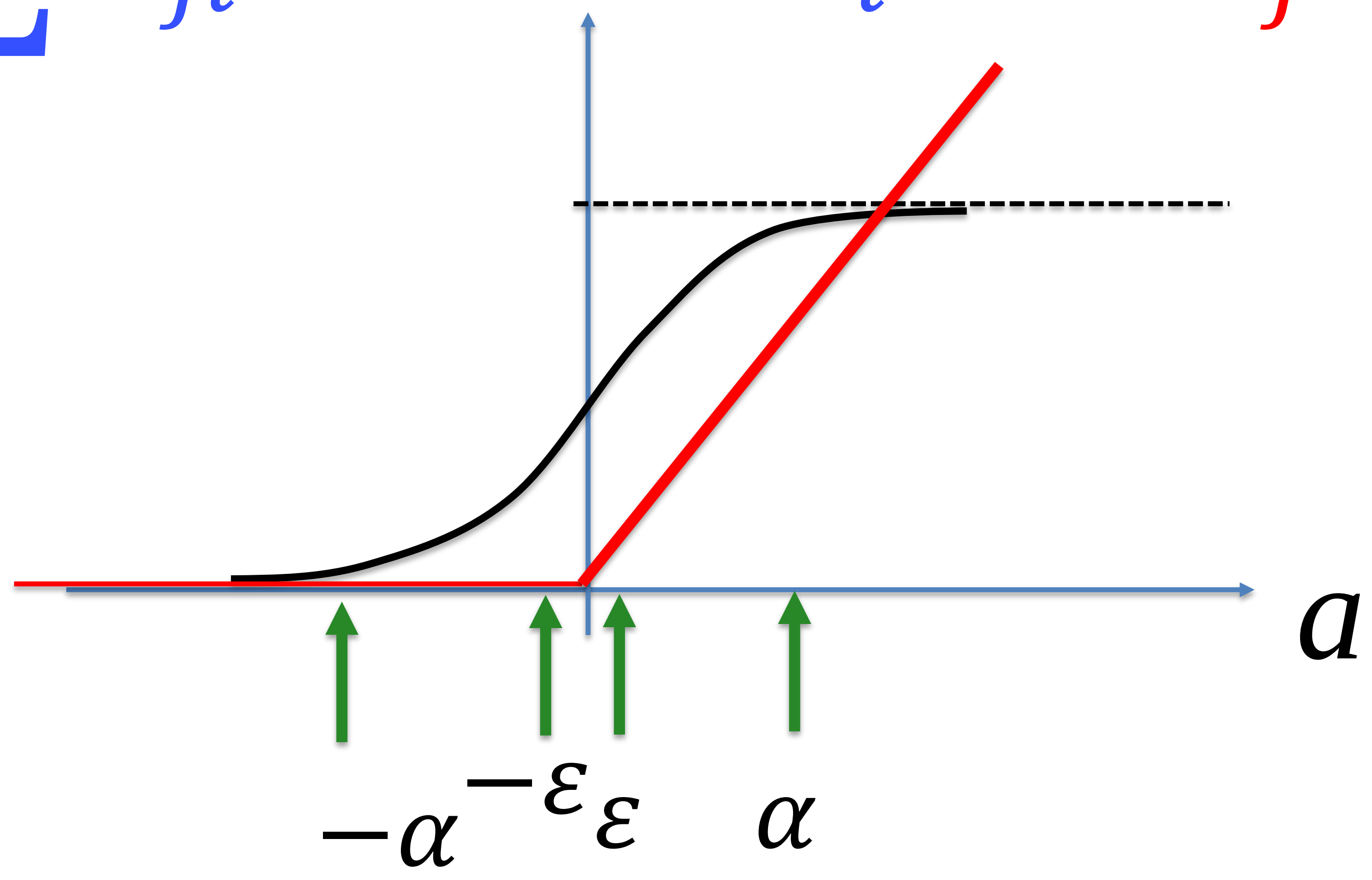
$$\Delta w_{ij}^{(n)} = \eta \, \delta_i^{(n)} \, x_j^{(n-1)} \quad (4)$$

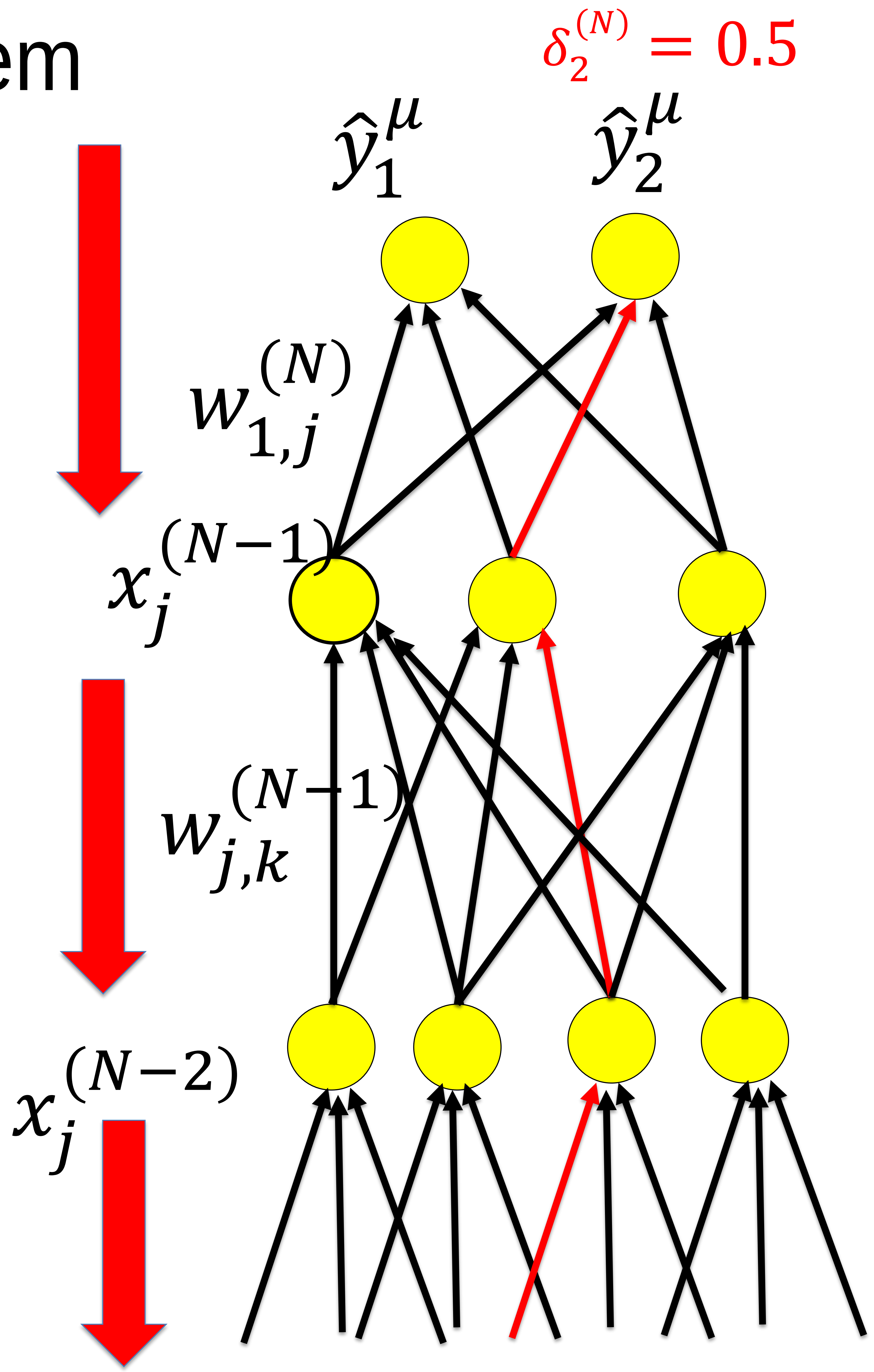
6. **Return to step 1.**



5. Backward pass: Vanishing gradient problem

$$\delta_i^{(n-1)} = \sum_j w_{ji}^{(n)} g'^{(n-1)}(a_i^{(n-1)}) \delta_j^{(n)}$$



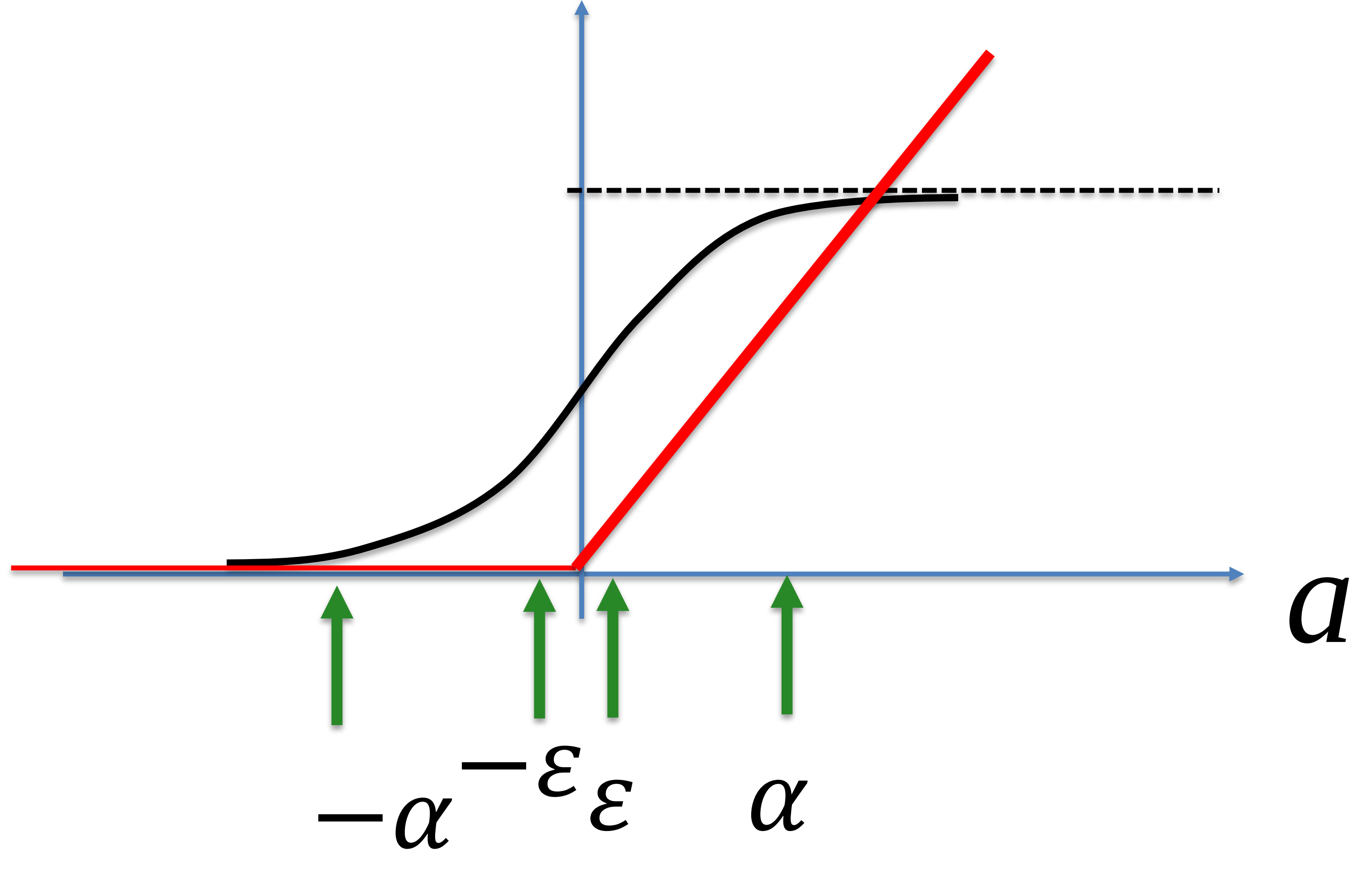


After N layers: each path contributes

$$\delta_i^{(1)} \sim g'^{(1)} g'^{(2)} \ldots g'^{(N-1)} \delta_j^{(N)}$$

Many terms to be summed,
but most terms are tiny if N large

5. Vanishing gradient problem

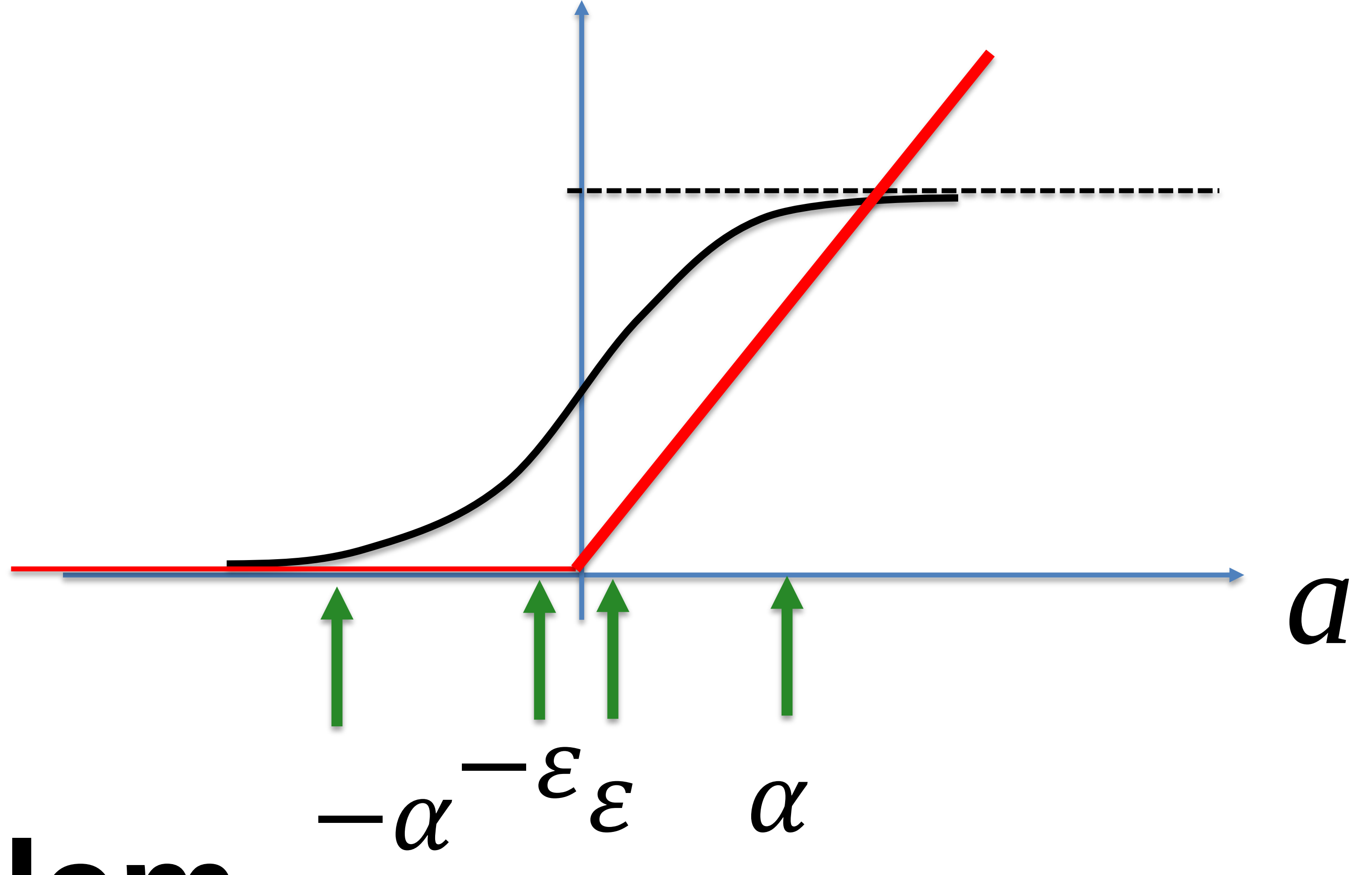


Observations:

- for each single path many terms g'
- g' is small for sigmoidal at $-\alpha$ or $+\alpha$ ($|a|=4$)
- g' vanishes for ReLu if one inactive unit sits in path
- g'=1 for all ReLu on 'active paths'

→ for ReLu highly active forward paths coincide with good gradient transmission on backward path

5. Vanishing gradient problem



Conclusion:

Sucessful forward pass

→ needs to avoid the linearity problem.

('exploit nonlinearities')

Successful backward pass

→ needs to avoid the vanishing gradient problem.

A good hidden units must be good for forward and backward pass!

Artificial Neural Networks: Lecture 4
Tricks of the Trade in deep networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

1. Bagging
2. Dropout
3. Other simple regularization methods
4. Initialization and choice of hidden units are important.
5. Vanishing gradient problem
6. Weight update: mean input and bias problem

6. Weight update step

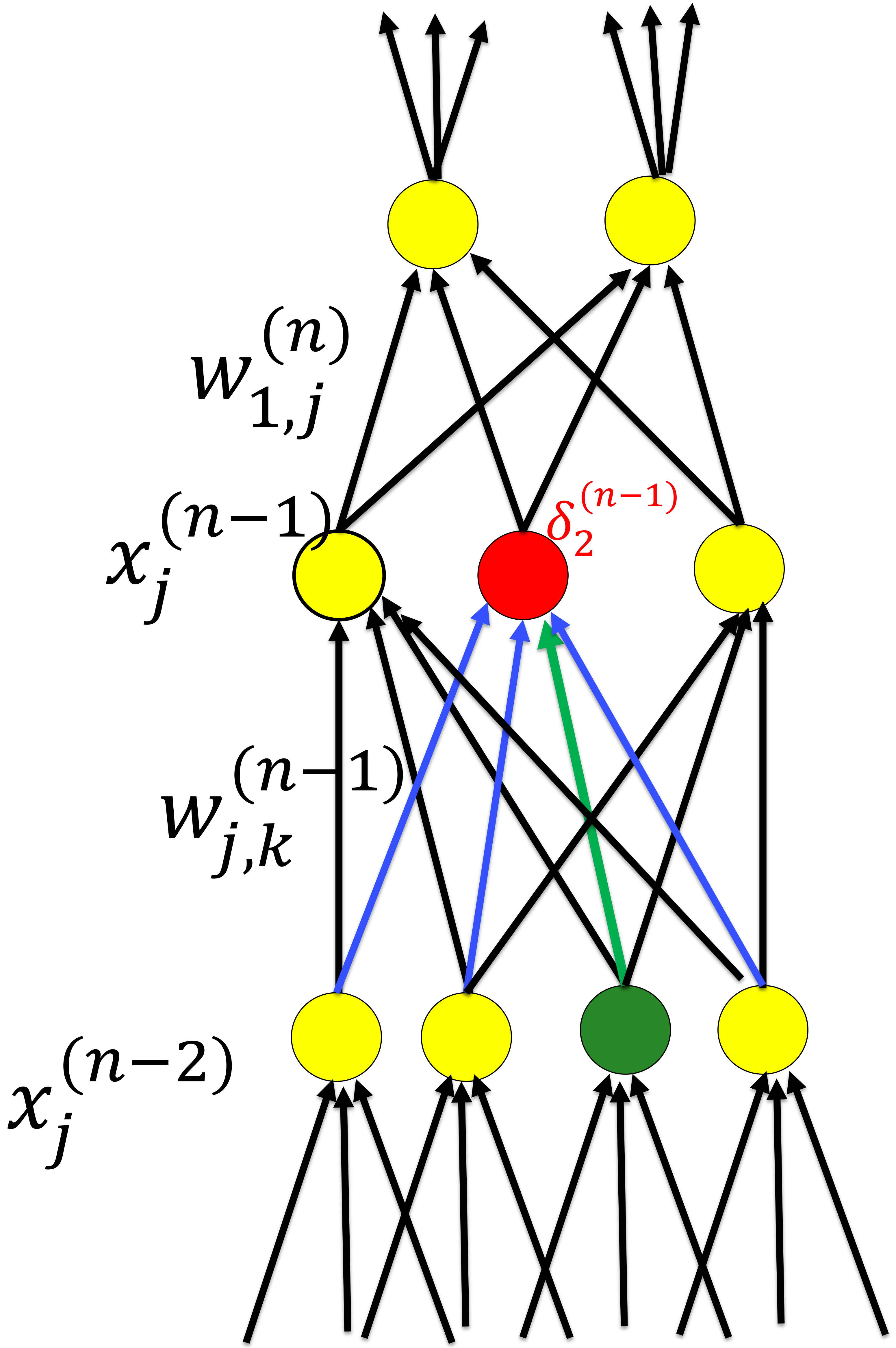
update **all** weights

$$\Delta w_{i,j}^{(n-1)} = \delta_i^{(n-1)} x_j^{(n-2)}$$

Weights onto the same neuron (red) are all updated with same delta

→ if $x_j^{(n-2)}$ are all positive,

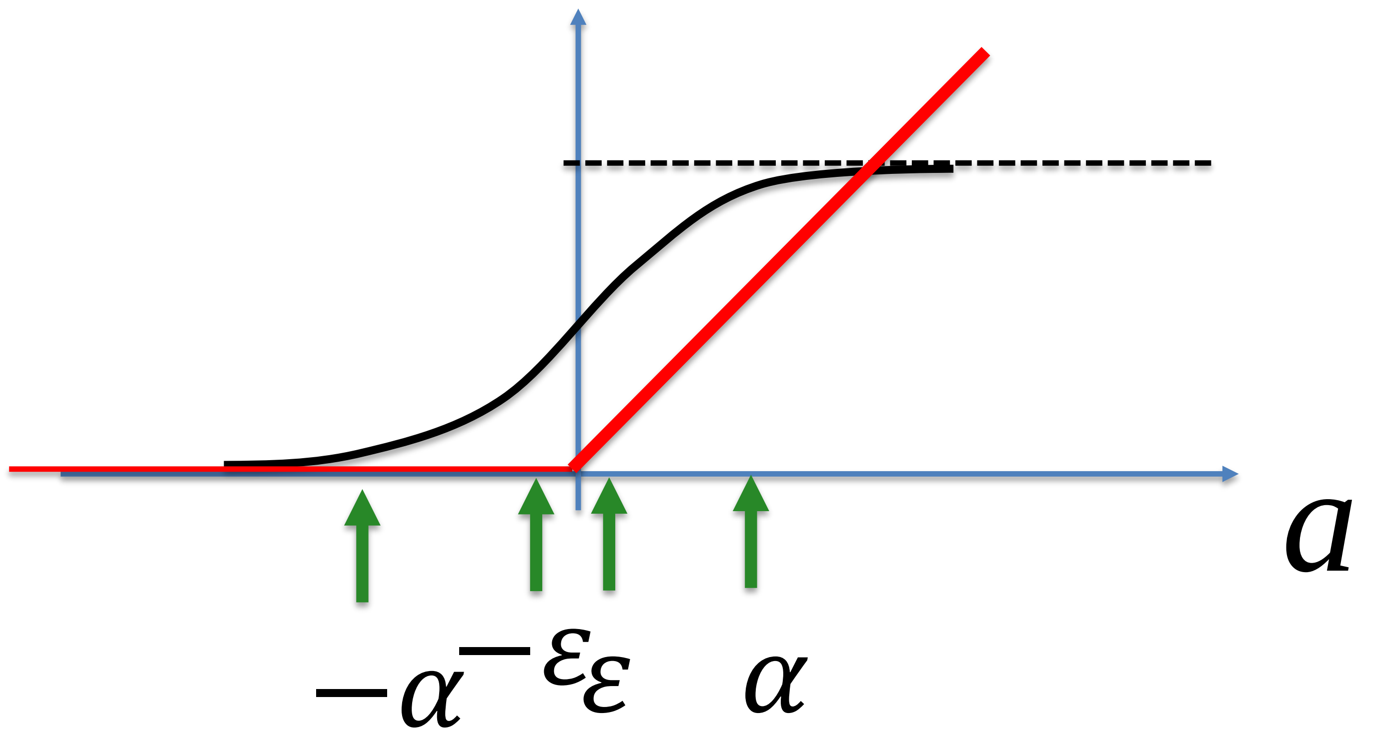
all the weights onto red neuron increase or decrease together

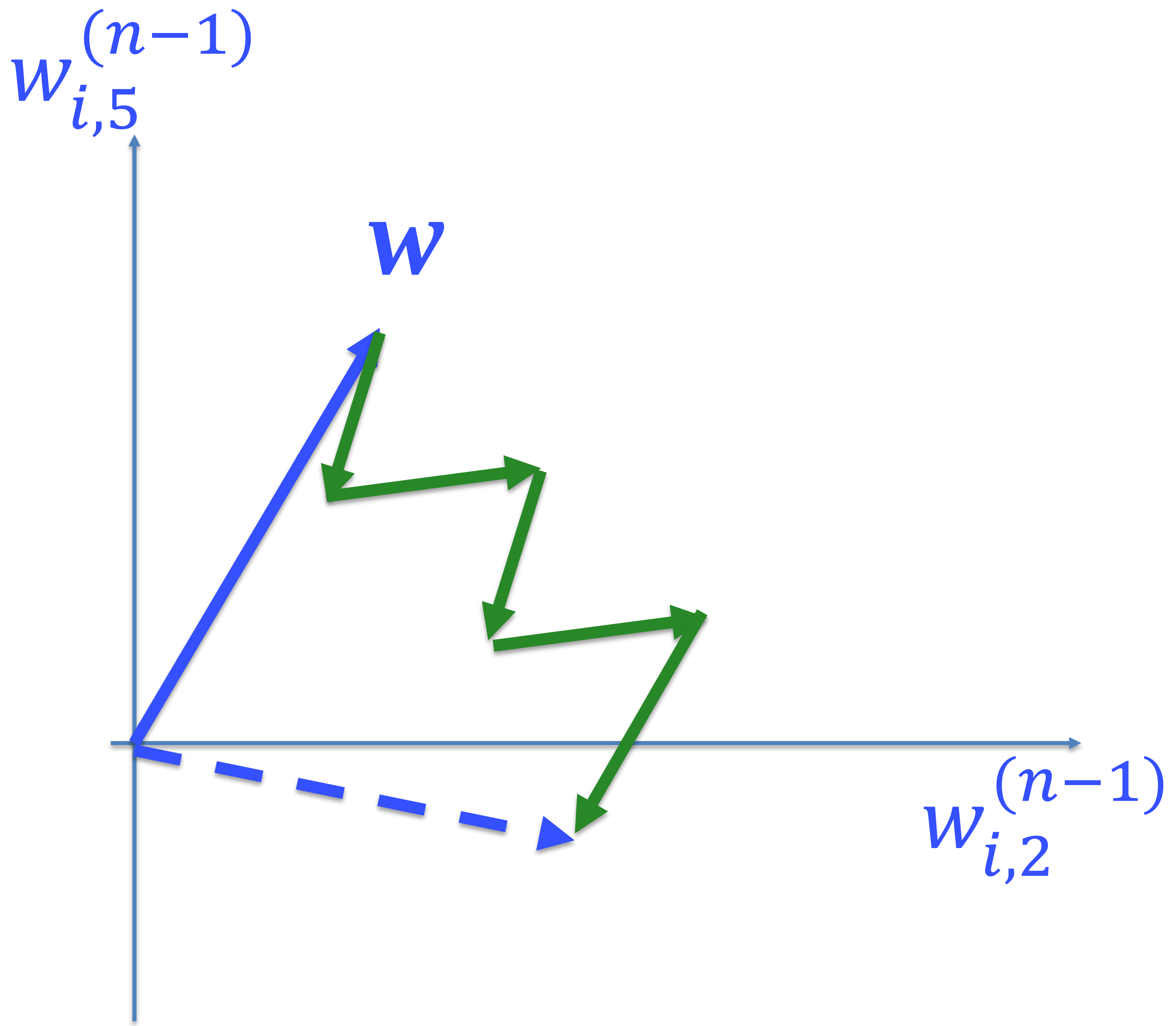


6. Weight update step

update **all** weights

$$\Delta w_{i,j}^{(n-1)} = \delta_i^{(n-1)} x_j^{(n-2)}$$





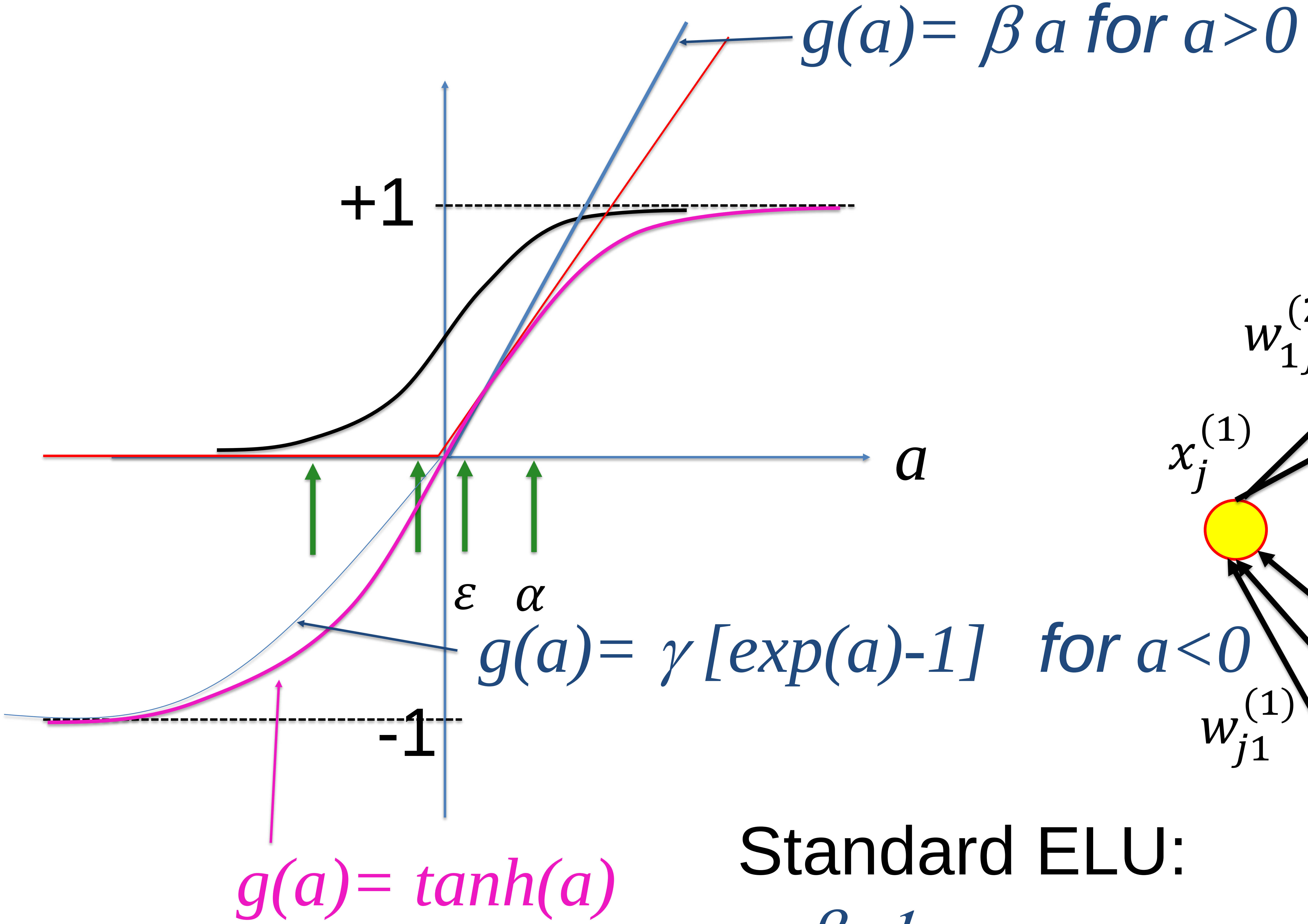
Weights onto the same neuron are all updated with same delta

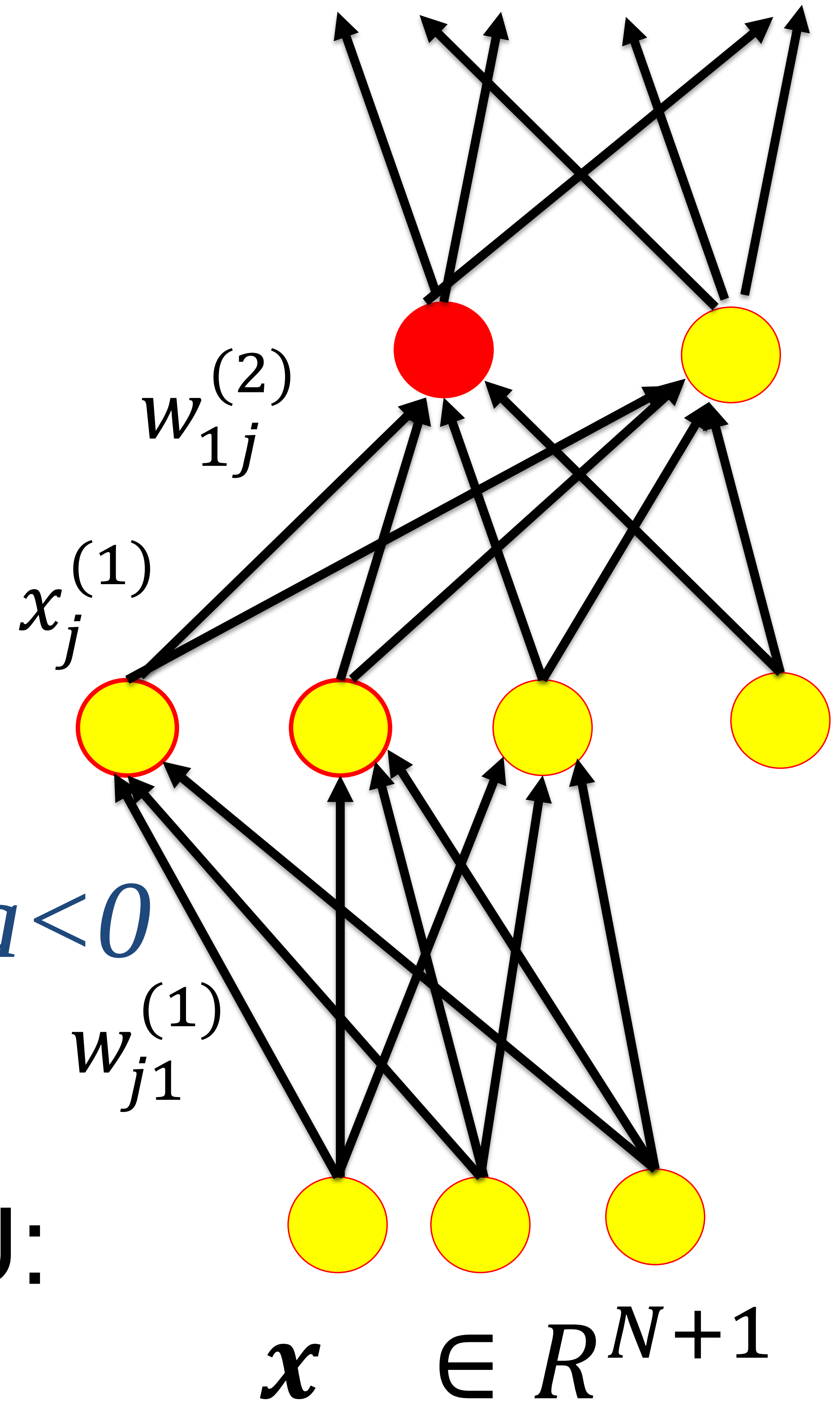
→ Problem for ReLu and other units with non-negative x

→ No problem for tanh

→ No problem for shifted exponential linear Selu

Shifted Exponential Linear (SELU) vs. tanh





Standard ELU:

γ=β=1

6. Bias problem

update **all** weights

$$\Delta w_{i,j}^{(n)} = \delta_i^{(n)} x_j^{(n-1)}$$

Before update

$$a_i^{(n)} = \sum_j w_{ij}^{(n)} x_j^{(n-1)} - \vartheta$$

after update

$$a_i^{(n)} = \sum_j [w_{ij}^{(n)} + \Delta w_{i,j}^{(n)}] x_j^{(n-1)} - \vartheta$$

same sign for all j

non-negative
(for ReLu etc)

Weights onto the same neuron
are all updated with same delta

→ Problem for ReLu and other units with non-negative x

→ The mean changes! ('bias problem')

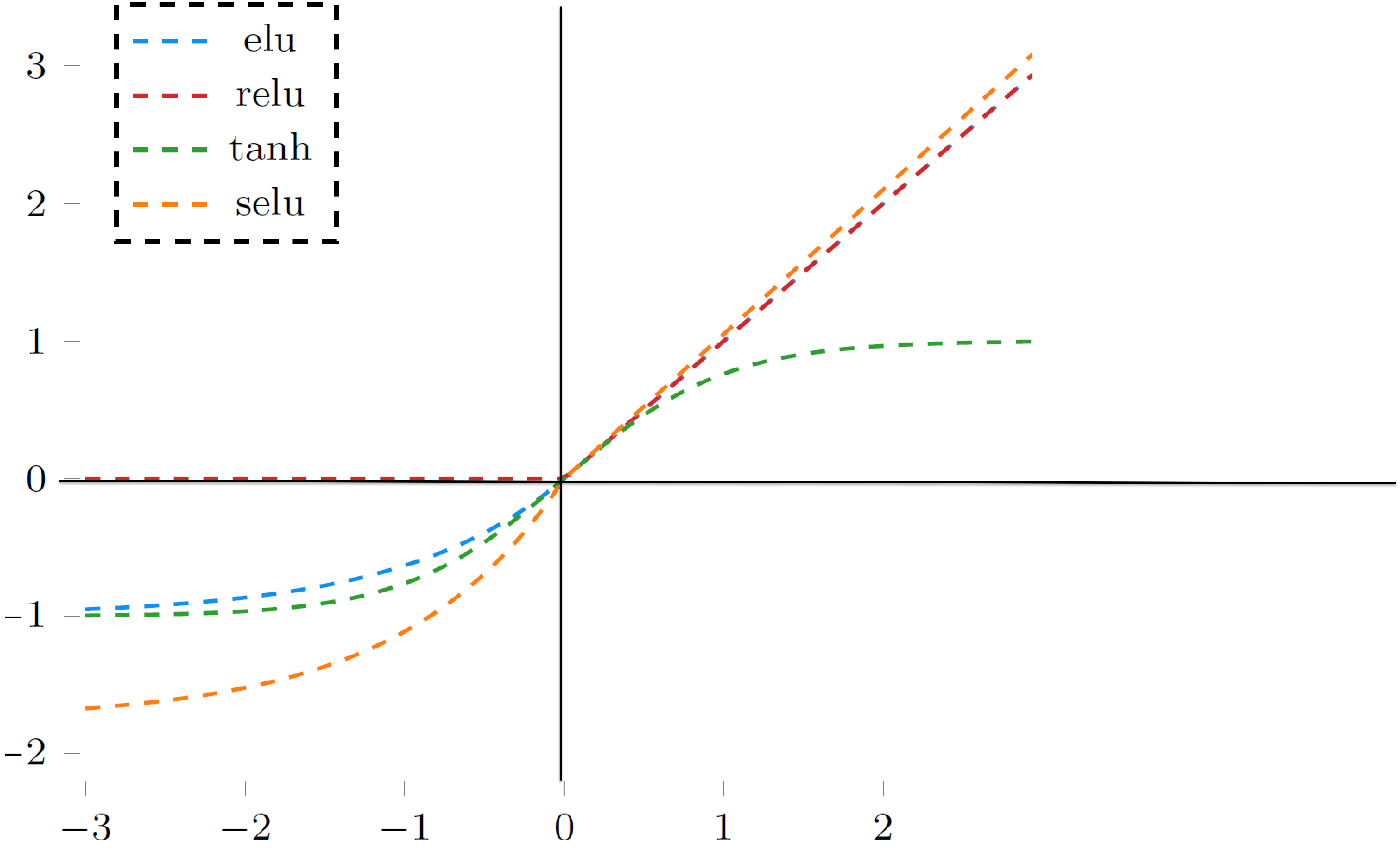
→But controlling the mean was important for correct initialization!

→ Return of vanishing gradient and linearity problem!

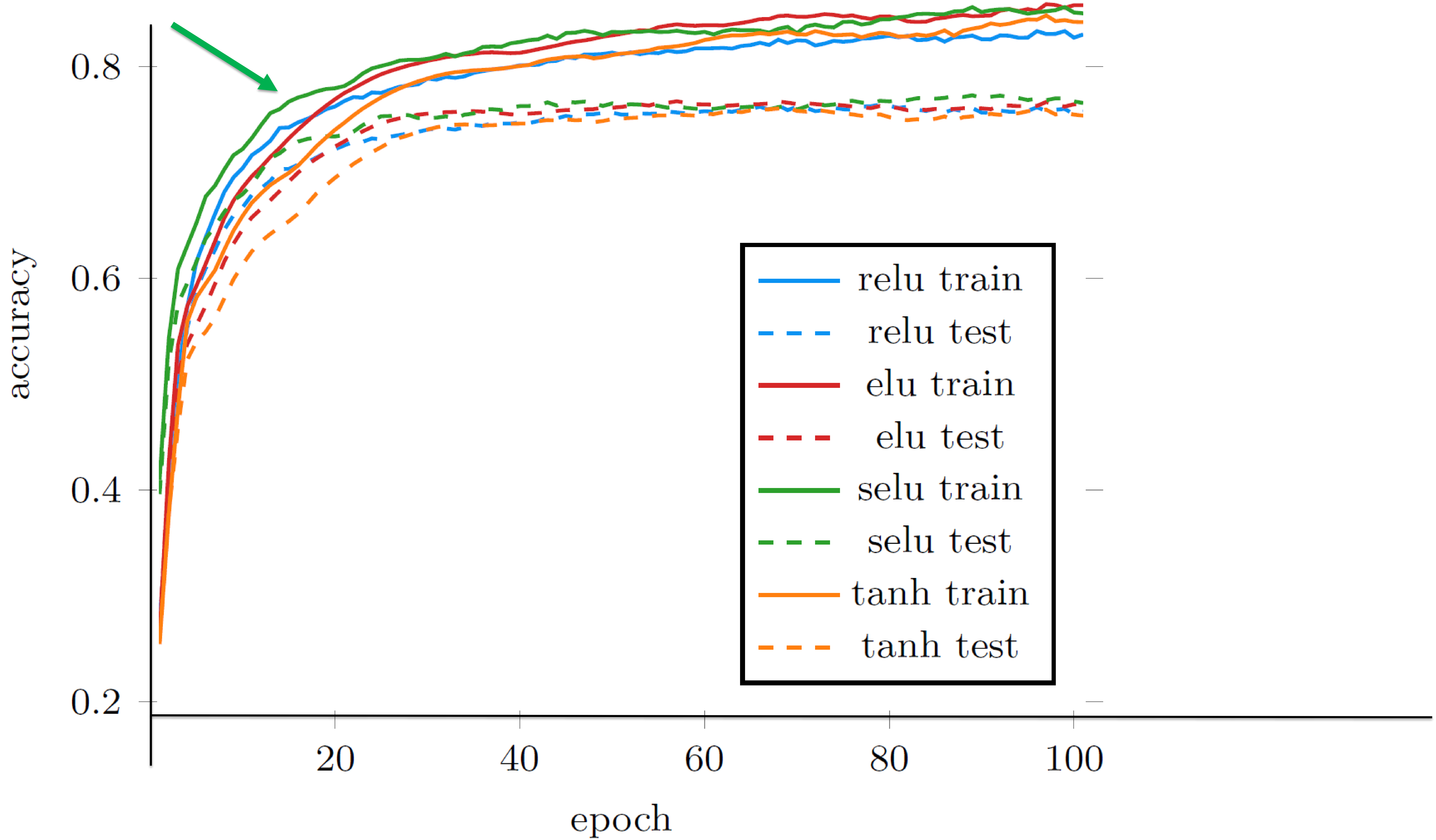
Quiz:

[ ] forward propagation with ReLu leaves only a few active paths
[ ] back propagation with ReLu leaves only a few active paths
[ ] a non-zero weight update step of ReLu shifts most often the mean
[ ] forward propagation with ReLu is always linear on the active paths
[ ] in a ReLu network all patterns are processed with the same linear filter
[ ] in a sigmoidal network with small weights (and normalized inputs) all patterns are processed with the same linear filter
[ ] in a sigmoidal network with big weights, there are active units in the forward pass that contribute a vanishing gradient in the backward path
[ ] in a network with SELU, there are active units in the forward path which contribute a vanishing gradient in the backward path
[ ] a non-zero the weight update step of SELU shifts the mean

Shifted Exponential Linear vs. tanh



Shifted Exponential Linear (SELU)



6. Conclusion

- initialization is important so as to exploit nonlinearities
- choice of hidden unit is important in **initial phase** of training
- ReLu has disadvantages in keeping the mean
  → batch normalization
- Tanh has problems with vanishing gradient
- Sigmoidal has problems with vanishing gradient **and** mean
- SELU solves all problems and is currently best choice

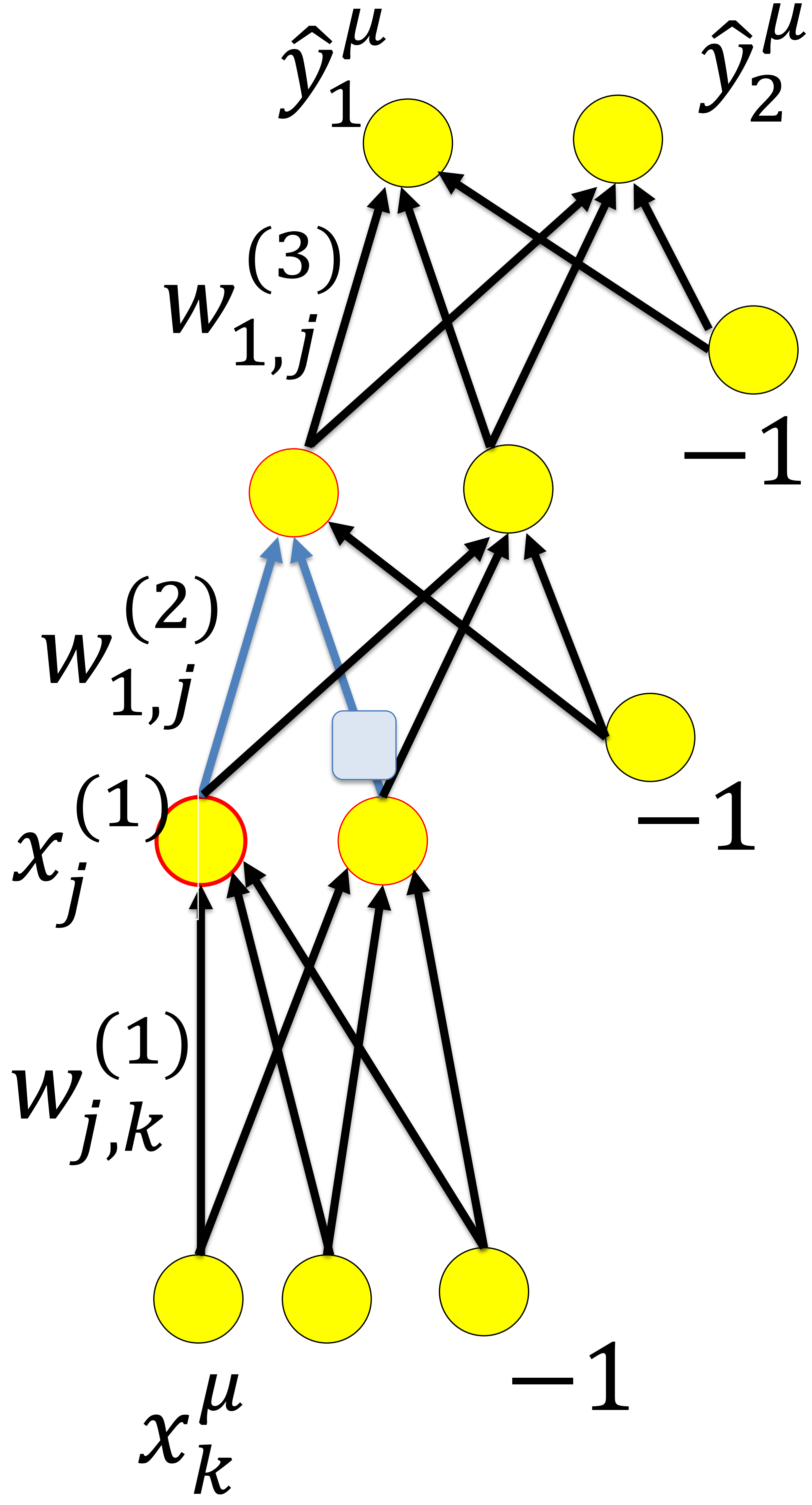
Paper: Klaumbauer, …, Hochreiter (2017)
Self-normalizaing neural networks
https://arxiv.org/pdf/1706.02515.pdf

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Artificial Neural Networks: Lecture 4
Tricks of the Trade in deep networks

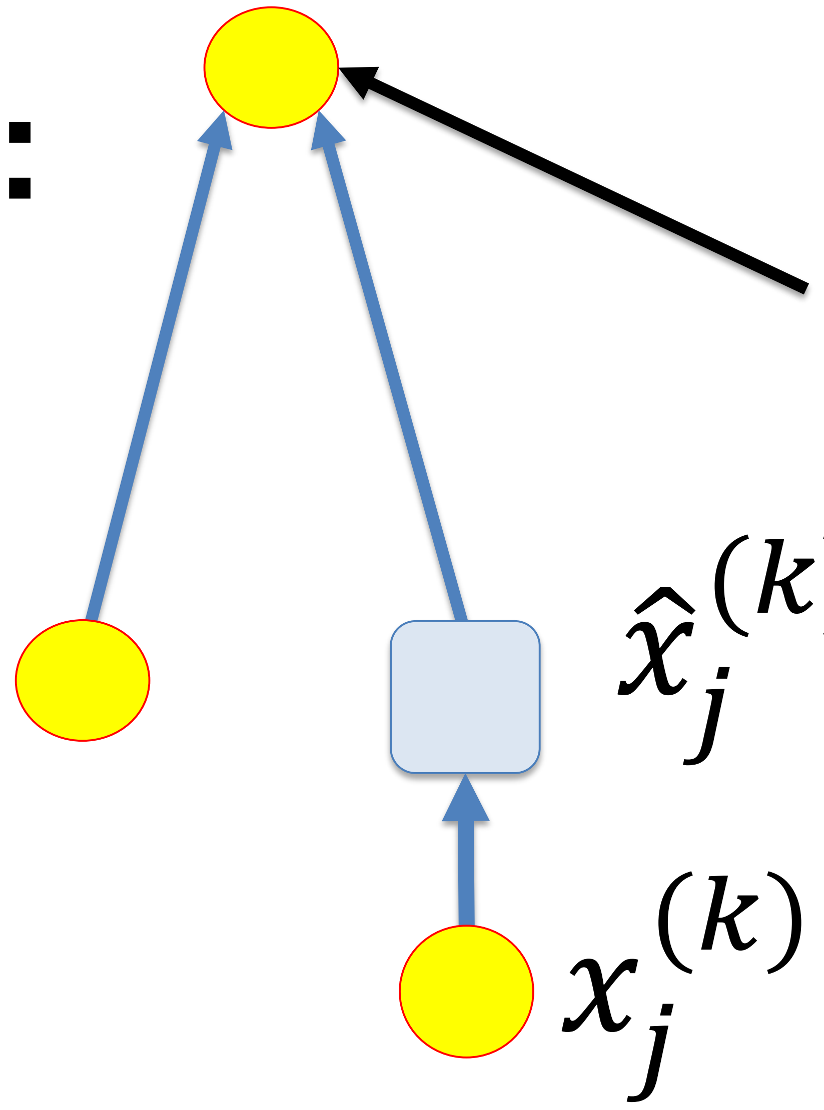
1. Bagging
2. Dropout
3. Other simple regularization methods
4. Hidden units: linearity problem (exploit nonlinearities)
5. Hidden units: Vanishing gradient problem
6. Weight update: bias problem
7. Batch normalization

7. Batch normalization: Idea



Normalize input on each input line

Zoom:



$$\hat{x}_j^{(k)} = \frac{x_j^{(k)} - E[x_j^{(k)}]}{\sqrt{Var[x_j^{(k)}]}}$$

7. Batch normalization

Ioffe&Szegedi, 2015

Work with minibatch:
Normalize per minibatch

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

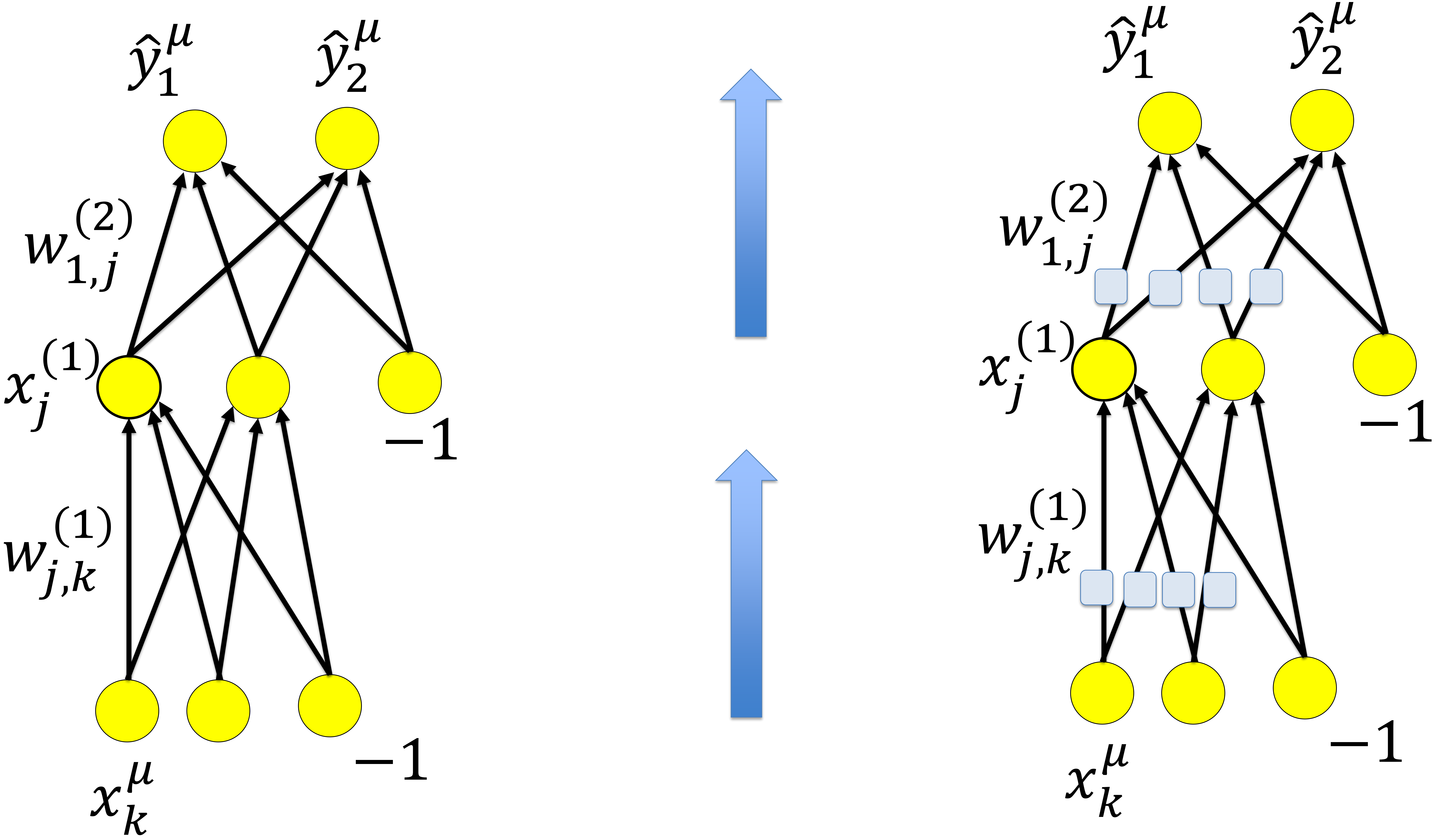
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

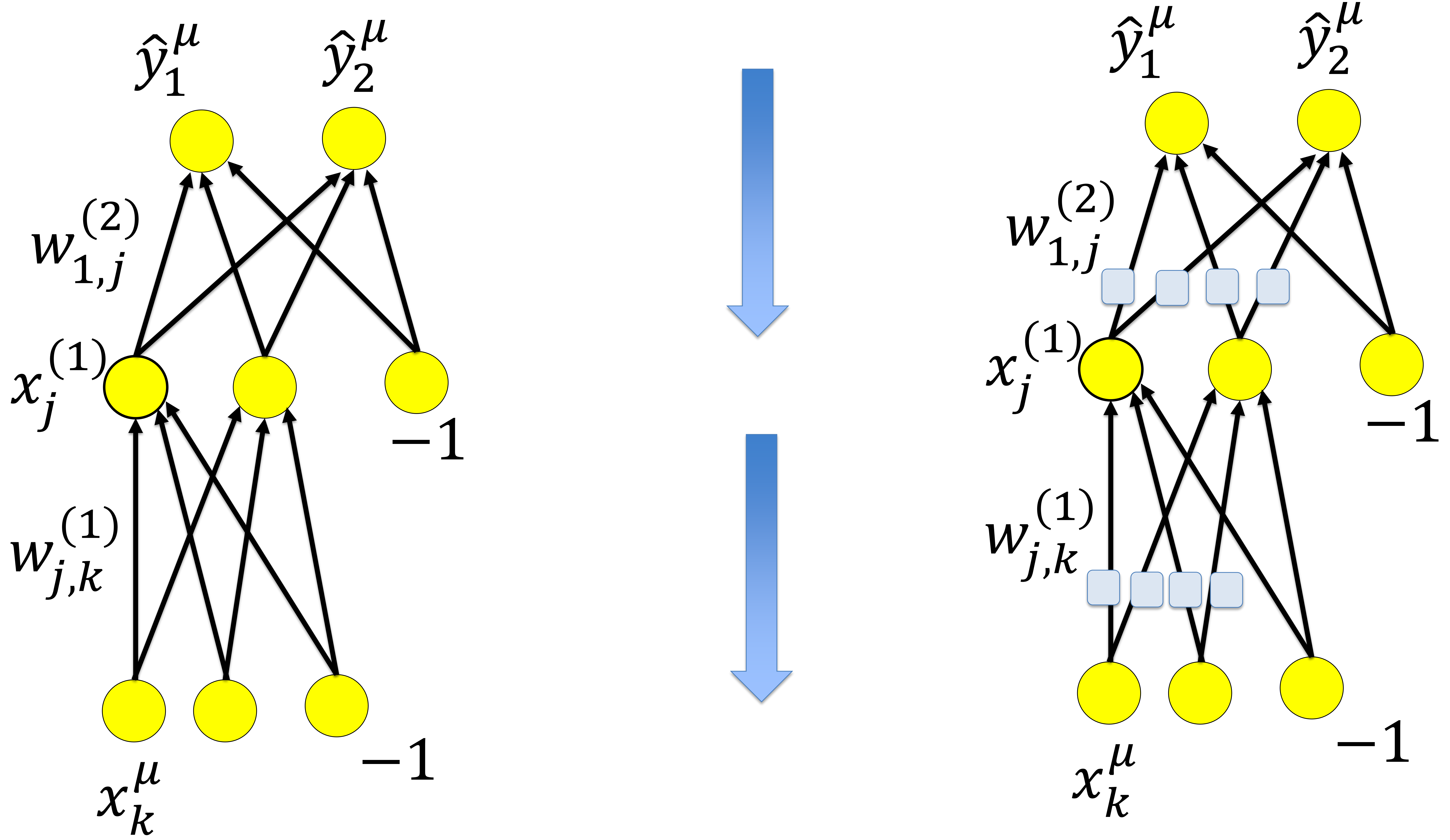
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

7. Batch normalization Ioffe&Szegedi, 2015



7. Batch normalization Ioffe&Szegedi, 2015



7. Batch normalization

Ioffe&Szegedi, 2015

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

$N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network

for $k = 1 \ldots K$ **do**

  Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)

  Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead

end for

end for

Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters Θ $\{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

$N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with froze // parameters

for $k = 1 \ldots K$ **do**

  // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc

  Process multiple training mini-batches $\mathcal{B}$, each size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

  In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ wi

$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$$

end for

Algorithm 2: Training a Batch-Normalized Network

7. Batch normalization

Ioffe&Szegedi, 2015

Necessary for ReLu and other unbalanced hidden units

Normalization step in forward pass is also taken care of during backward pass

Objectives for today:

- Bagging: multiple models help always to improve results!
- Dropout: two interpretations
  - (i) a practical implementation of bagging
  - (ii) forced feature sharing
- BackProp: Initialization, nonlinearity, and symmetry
- What are good units for hidden layers?
  - problems of vanishing gradient and shift of mean
  - → solved by Shifted exponential linear (SELU)
- Batch normalization → necessary for ReLu

The end