CS 481

Artificial Intelligence Language Understanding

April 4, 2023

Announcements / Reminders

- Please follow the Week 20 To Do List instructions
- Programming Assignment #02 is due on ~~Sunday 04/02/23~~ Thursday 04/06/23 11:59 PM CST

- Final Exam date:

  Thursday 04/27/2023 (last week of classes!)

  - Ignore the date provided by the Registrar
  - Section 02 [Online]: contact Mr. Charles Scott (scott@iit.edu) to arrange your exam

Plan for Today

- Word Embeddings
- Word2Vec
- NLP and Neural Networks

Word2Vec: the Approach

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get **negative examples**
3. Use **logistic regression** to train a classifier to distinguish those two cases
4. Use the **learned classifier weights** as the **embeddings**

Word2Vec: the Approach

Given the set of **positive** and **negative** training instances, and an **initial set of embedding vectors**

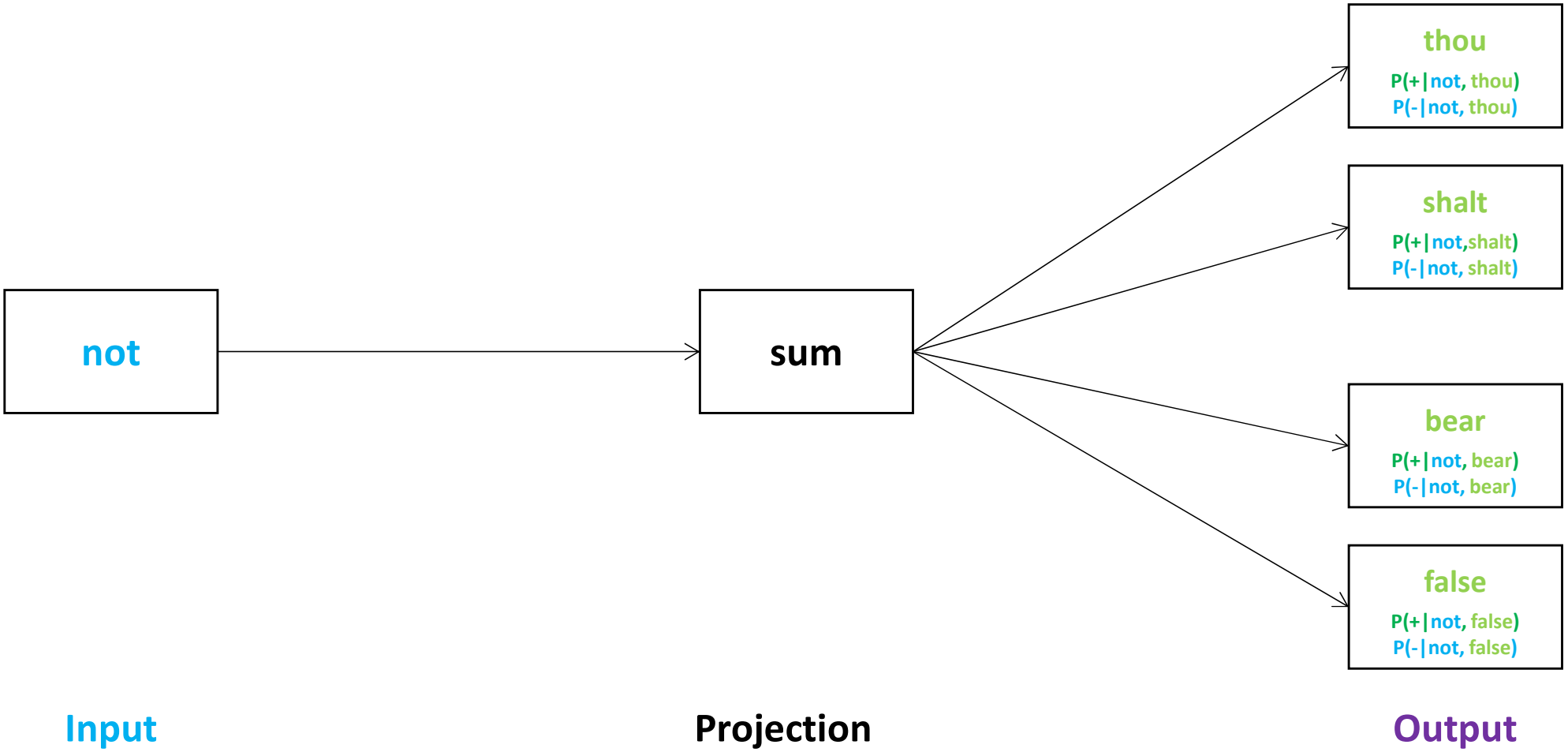
The goal of learning is to adjust those word vectors such that we:

- **maximize** the similarity of the target word, context word pairs (w, c_{pos}) drawn from the **positive** data
- **minimize** the similarity of the (w, c_{neg}) pairs drawn from the **negative** data.

Skip Gram Word2Vec

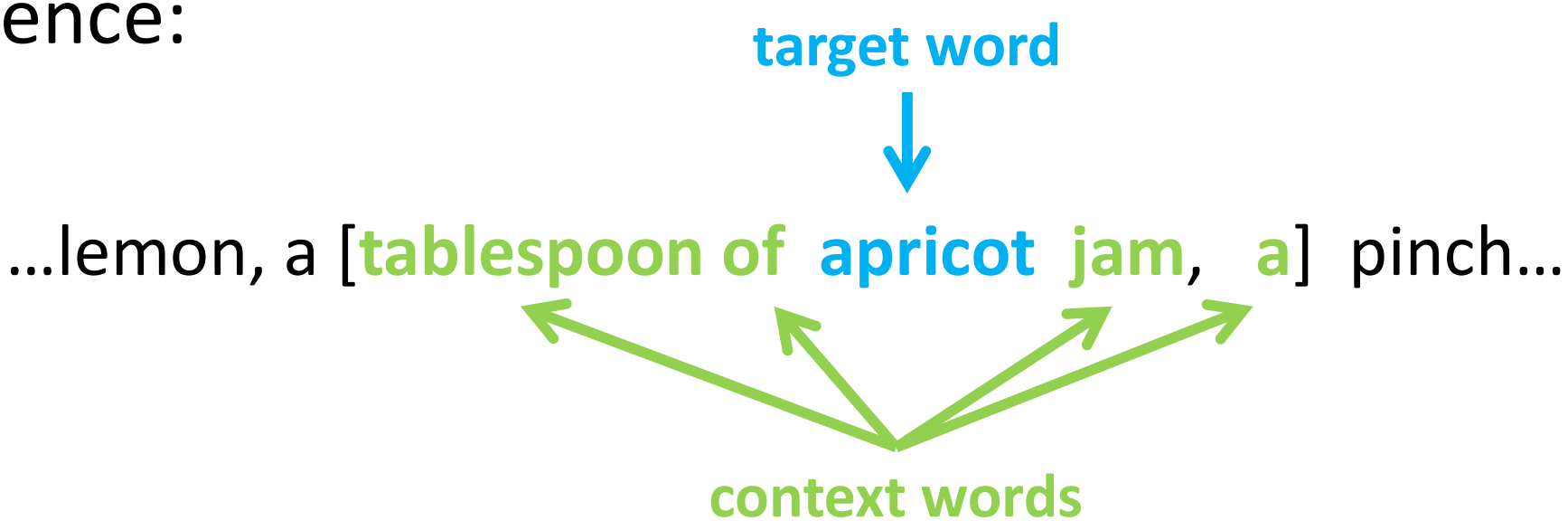
Predict context given target word:

thou shalt _not_ bear false **witness**



Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:



Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

w

c_1 c_2 c_3 c_4

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 c_3 c_4

Goal 1: **train a classifier** that is given a candidate (**word**, **context word**) pair: (**apricot**, **jam**), (**apricot**, **aardvark**), etc.

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [tablespoon of apricot jam, a] pinch…

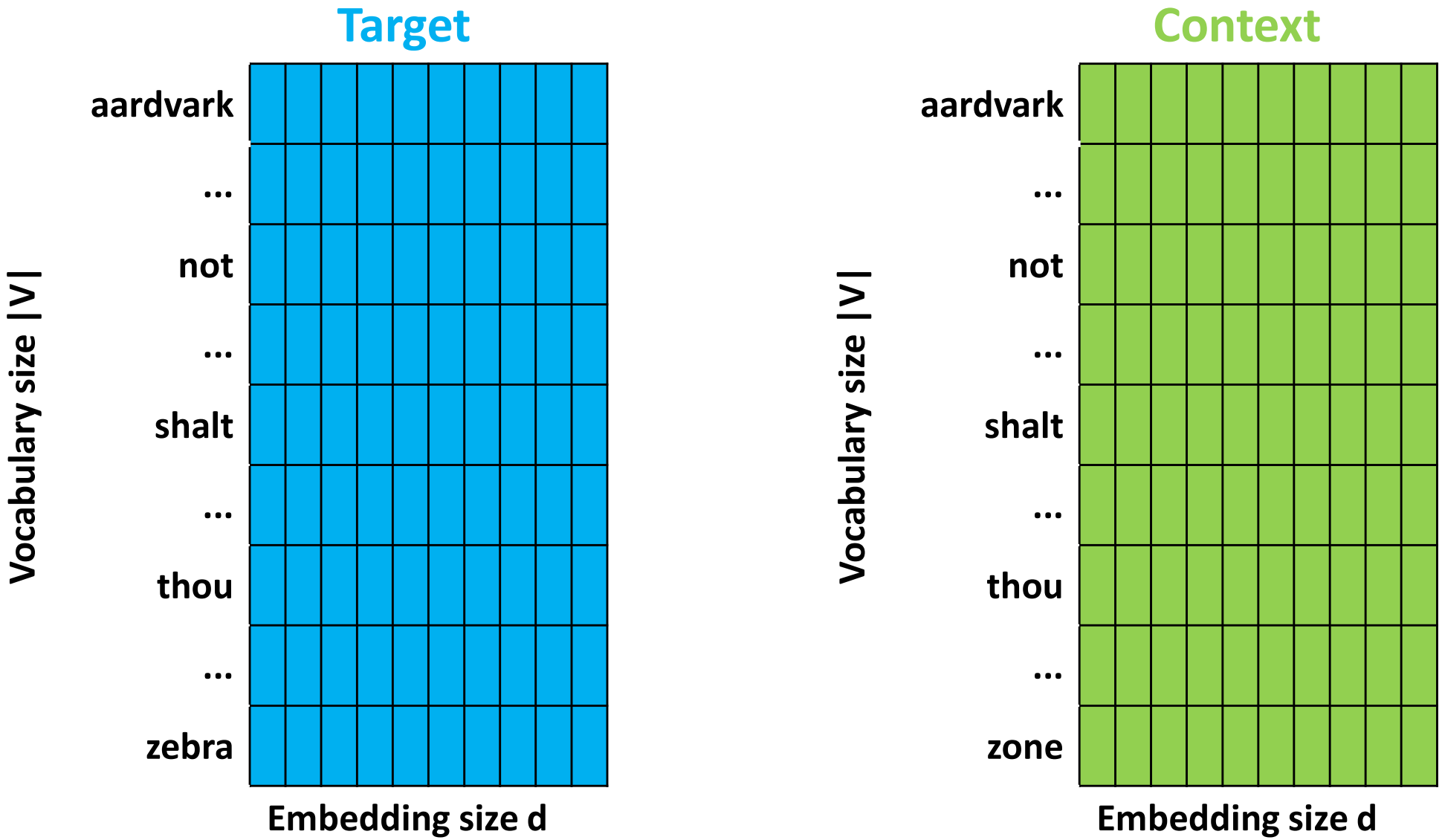
c_1 c_2 c_3 c_4

Goal 2: **assign probabilities** to every (**word**, **context word**) pair:

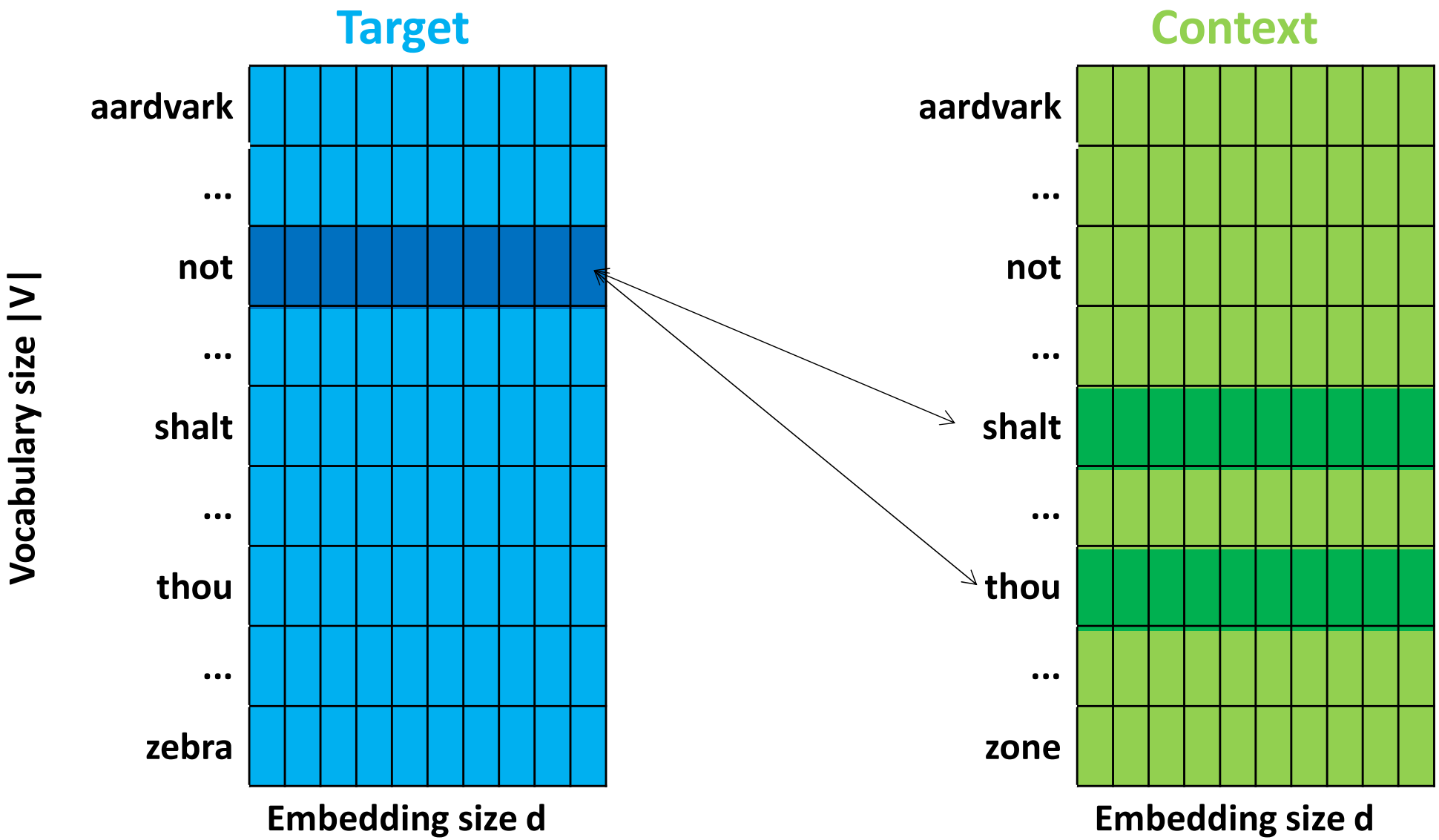
$$P(+ \mid w, c_i) \textbf{ and}$$

$$P(- \mid w, c_i) = 1 - P(+ \mid w, c_i)$$

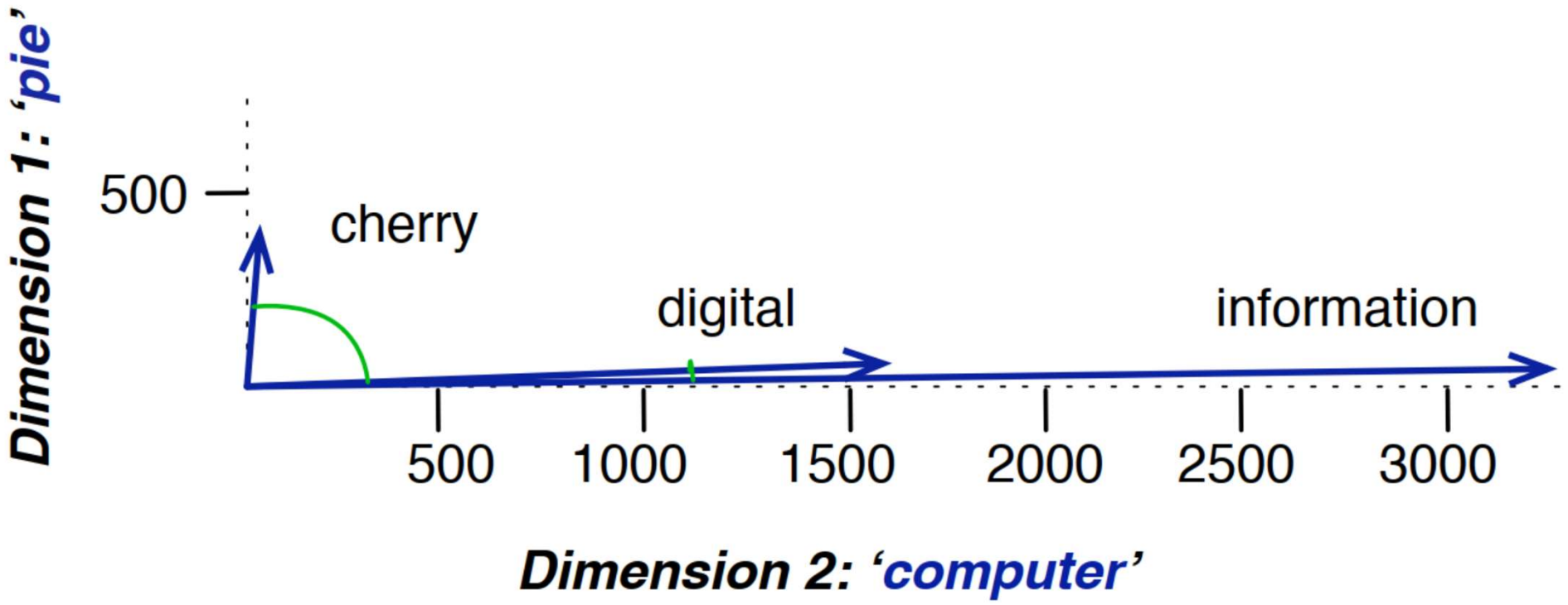
Target and Context Embeddings



Intuition: Target & Context Similar

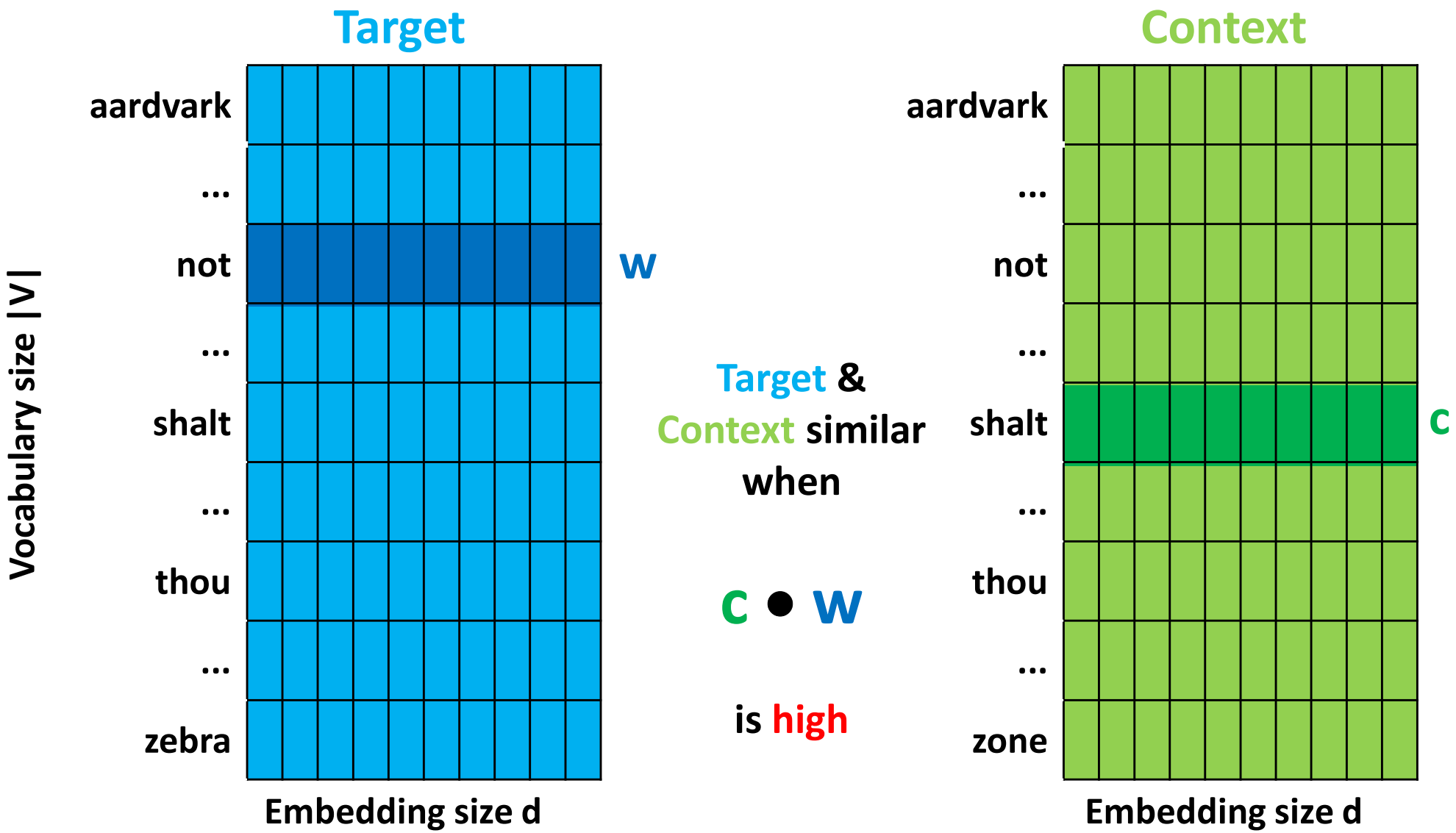


Cosine Similarity Visualization

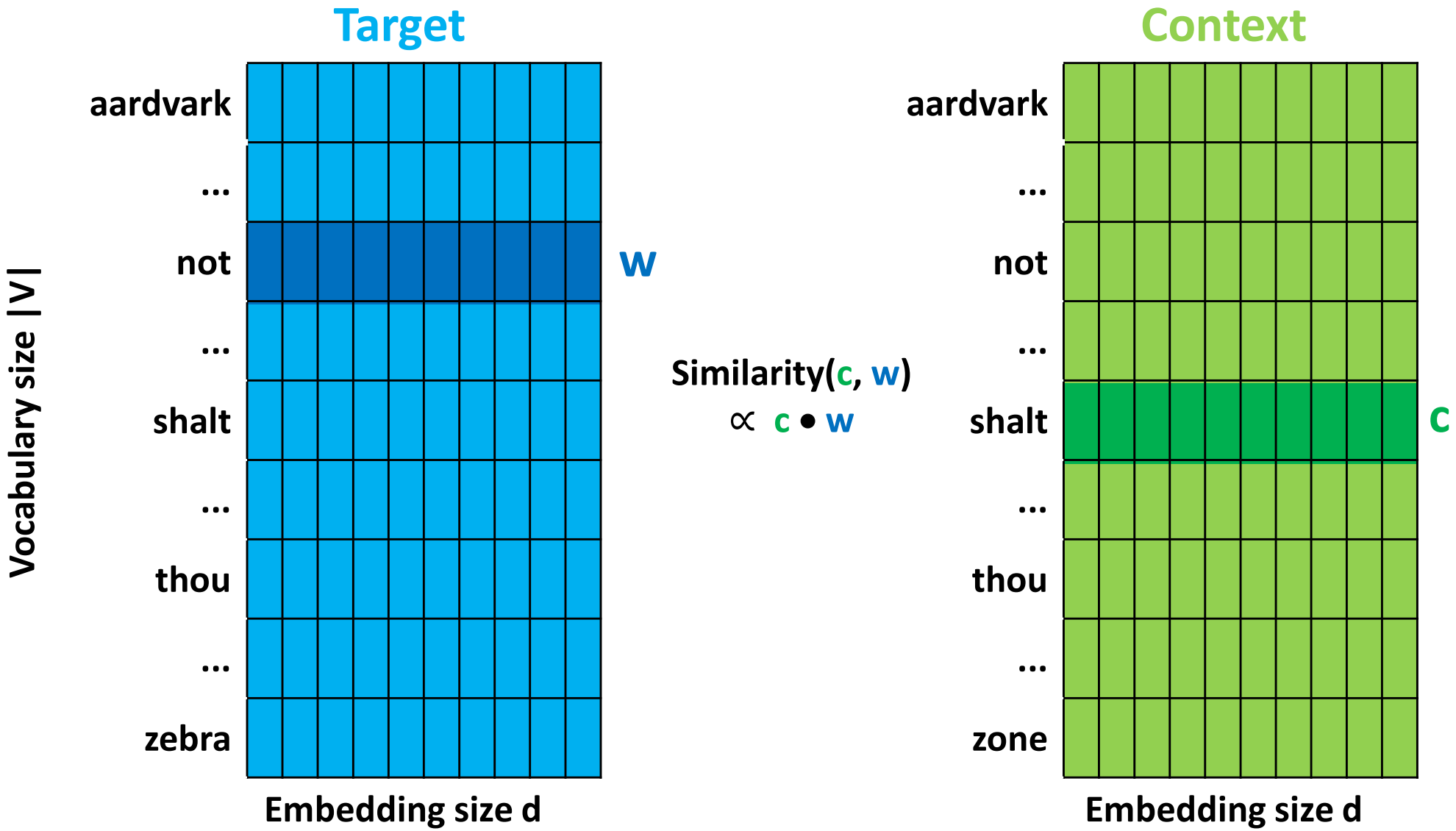


Two vectors are similar if they have a high dot product | cosine similarity

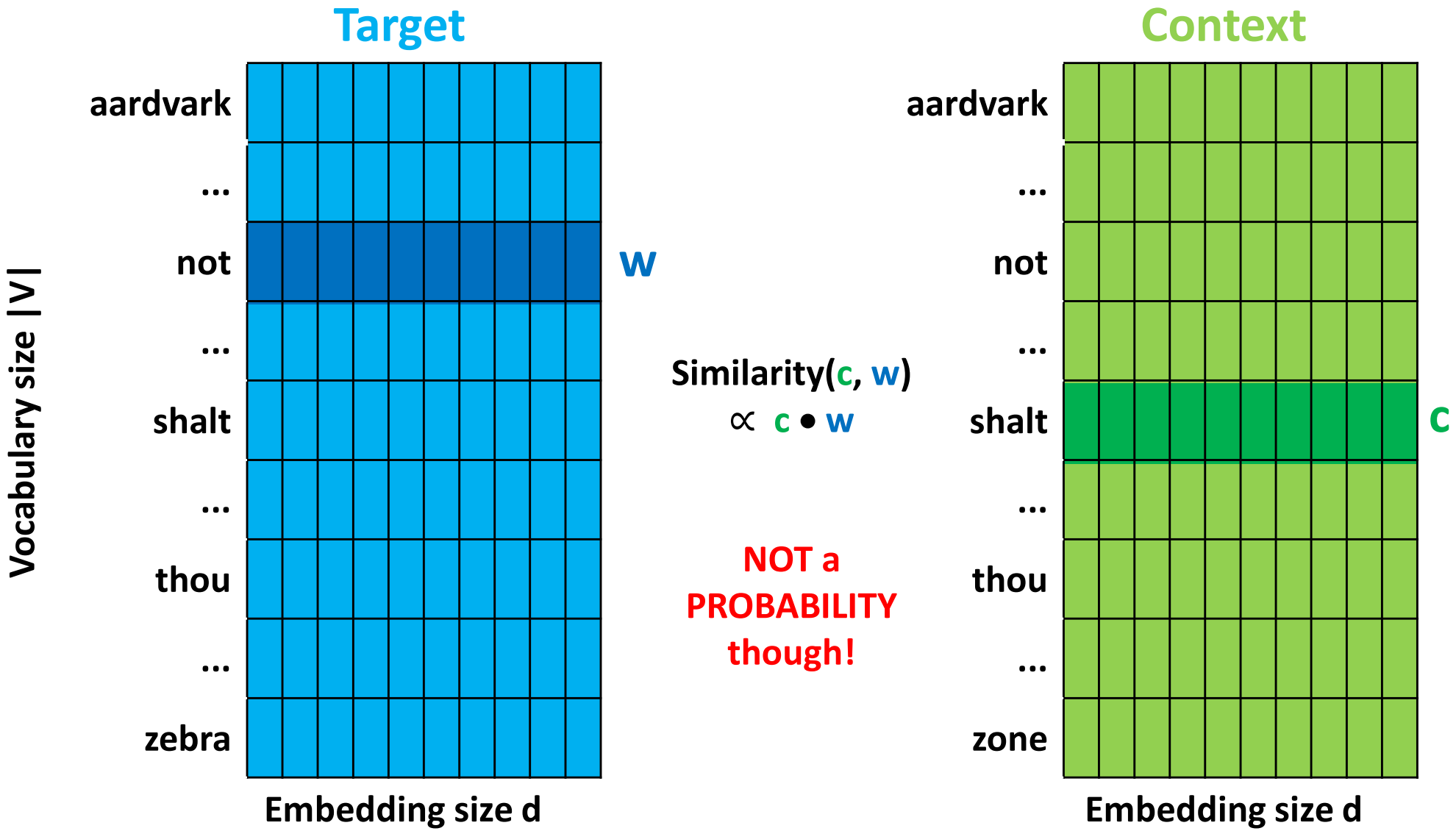
Intuition: Target & Context Similar



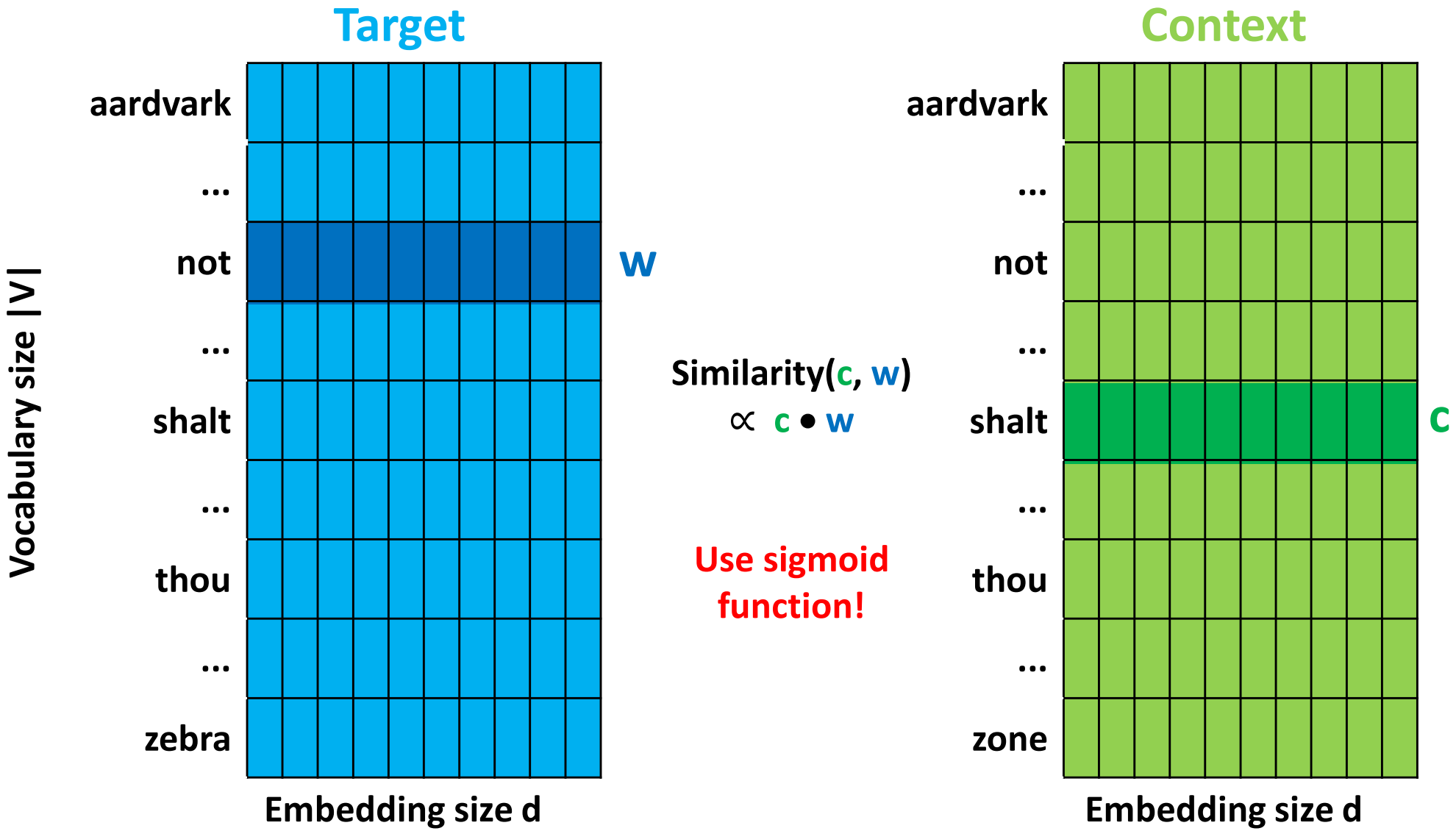
Intuition: Target & Context Similar



Intuition: Target & Context Similar



Intuition: Target & Context Similar



Similarity → Probability

$$P(+ \mid \mathbf{w}, \mathbf{c}) = \sigma(\mathbf{c} \bullet \mathbf{w}) = \frac{1}{1+exp(-\mathbf{c} \bullet \mathbf{w})}$$

$$P(- \mid \mathbf{w}, \mathbf{c}) = 1 - P(+ \mid \mathbf{w}, \mathbf{c}) =$$

$$= \sigma(-\mathbf{c} \bullet \mathbf{w}) = \frac{1}{1+exp(\mathbf{c} \bullet \mathbf{w})}$$

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

w

c_1 c_2 c_3 c_4

$P(+ \mid w, c_1)$ $P(+ \mid w, c_2)$ $P(+ \mid w, c_3)$ $P(+ \mid w, c_4)$

$P(- \mid w, c_1)$ $P(- \mid w, c_2)$ $P(- \mid w, c_3)$ $P(- \mid w, c_4)$

OK, but we have lots of possible context words!

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

w

c_1 c_2 c_3 c_4

$P(+ \mid w, c_1)$ $P(+ \mid w, c_2)$ $P(+ \mid w, c_3)$ $P(+ \mid w, c_4)$

$P(- \mid w, c_1)$ $P(- \mid w, c_2)$ $P(- \mid w, c_3)$ $P(- \mid w, c_4)$

Assuming word independence, calculate:

$$P(+ \mid w, c_1, c_2, c_3, c_4) = \prod_{i=1}^{4} \sigma(c_i \bullet w)$$

Skip Gram Classifier

Assume a +/- 2 (L = 4) word window, given training sentence:

w

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 c_3 c_4

$P(+ \mid w, c_1)$ $P(+ \mid w, c_2)$ $P(+ \mid w, c_3)$ $P(+ \mid w, c_4)$

$P(- \mid w, c_1)$ $P(- \mid w, c_2)$ $P(- \mid w, c_3)$ $P(- \mid w, c_4)$

In general:

$$P(+ \mid w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \bullet w)$$

Skip Gram Classifier

Assume a +/- 2 (**L = 4**) word window, given training sentence:

…lemon, a [tablespoon of **apricot** jam, a] pinch…

w

c_1 c_2 c_3 c_4

$P(+ \mid w, c_1)$ $P(+ \mid w, c_2)$ $P(+ \mid w, c_3)$ $P(+ \mid w, c_4)$

$P(- \mid w, c_1)$ $P(- \mid w, c_2)$ $P(- \mid w, c_3)$ $P(- \mid w, c_4)$

In general [with sums instead of products]:

$$\log P(+ \mid w, c_{1:L}) = \sum_{i=1}^{L} log\ \sigma(c_i \bullet w)$$

Skip Gram Classifier: Summary

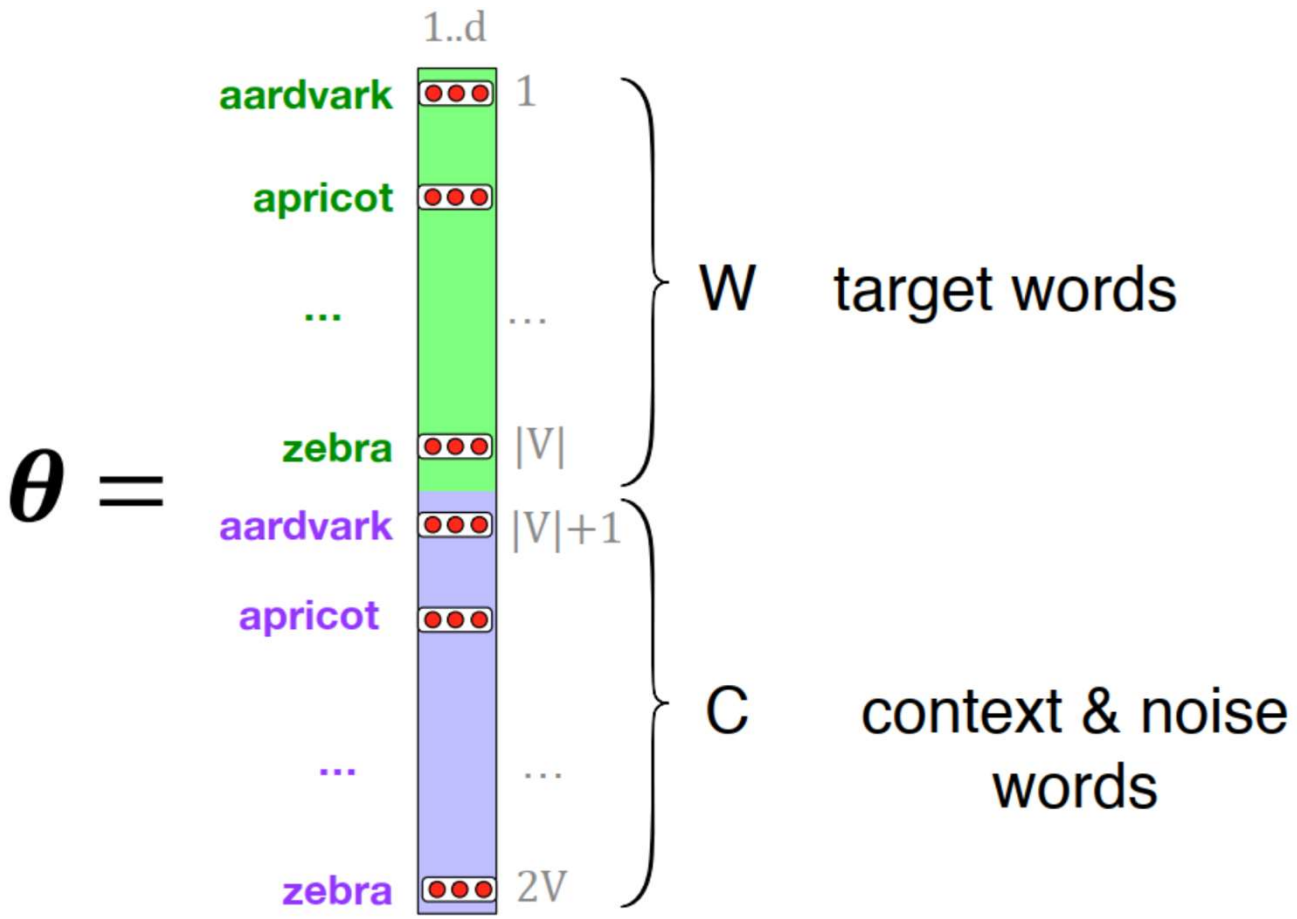
A probabilistic classifier, given

- a test target word w
- its context window of L words $c_{1:L}$

Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

To compute this, we just need embeddings for all the words.

Parameters:Target (W) and Context (C)



Word2Vec: the Approach

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get **negative examples**
3. Use **logistic regression** to train a classifier to distinguish those two cases
4. Use the **learned classifier weights** as the **embeddings**

Word2Vec: Training

Assume a +/- 2 (**L = 4**) word window, given training sentence:

…lemon, a [**tablespoon of apricot jam, a**] pinch…

Positive (+) examples:
(**apricot**, **tablespoon**),(**apricot**, **of**),(**apricot**, **jam**),(**apricot**, **a**)

Negative (-) K (typically double (+)) examples:
(**apricot**, **aardvark**),(**apricot**, **my**),(**apricot**, **where**),(**apricot**, **coaxial**)
(**apricot**, **seven**),(**apricot**, **forever**),(**apricot**, **dear**),(**apricot**, **if**)

Word2Vec: the Approach

Given the set of **positive** and **negative** training instances, and an **initial set of embedding vectors**

The goal of learning is to adjust those word vectors such that we:

- **maximize** the similarity of the target word, context word pairs (w, c_{pos}) drawn from the **positive** data
- **minimize** the similarity of the (w, c_{neg}) pairs drawn from the **negative** data.

Loss Function

Loss function for one w with c_{pos} , c_{neg1} ...c_{negk}

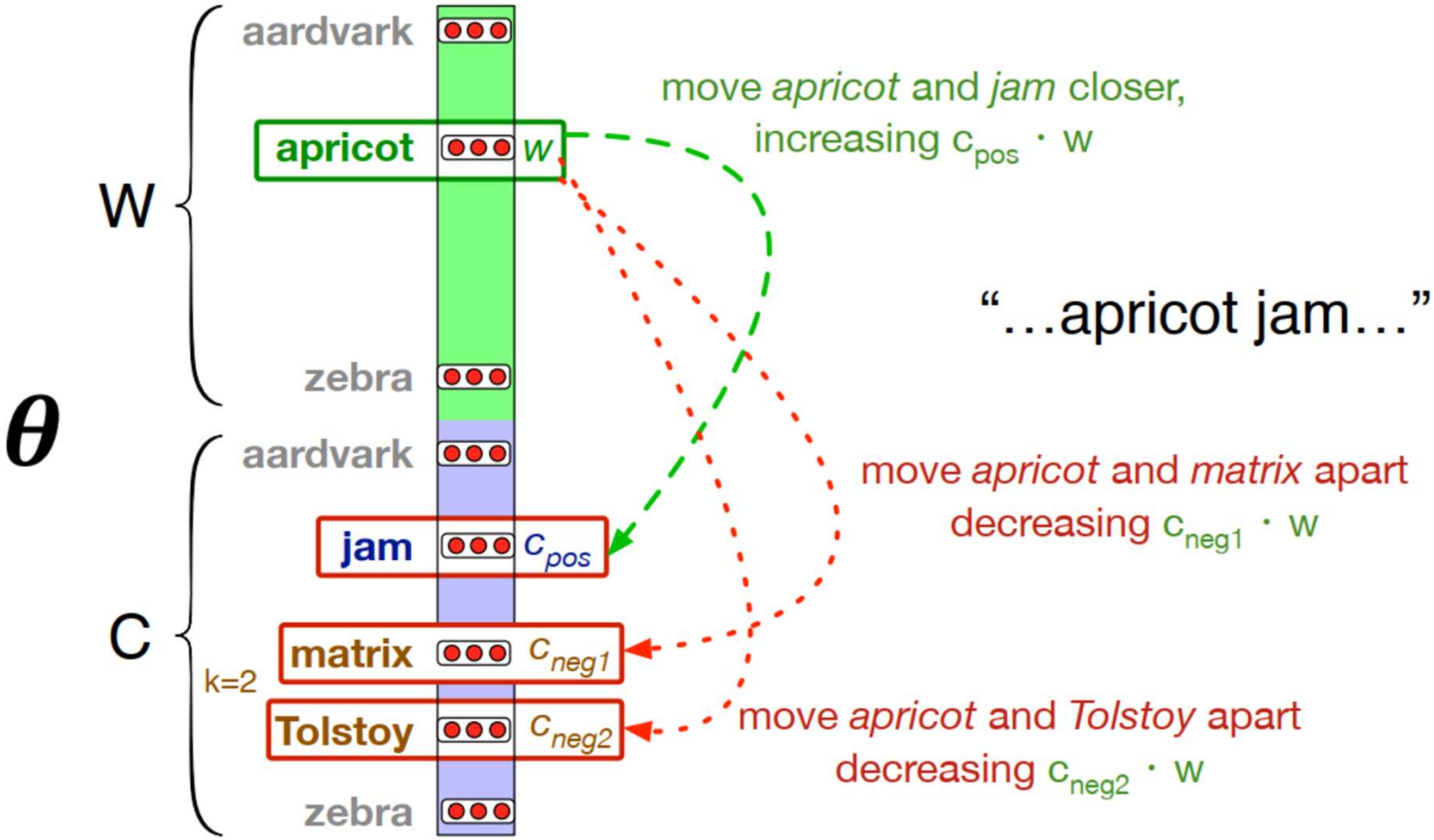
Maximize the **similarity of the target with the actual context words (+)**, and **minimize the similarity of the target with the k negative sampled non-neighbor words (-)**.

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Classifier: Learning Process

- How to learn?
  - use stochastic gradient descent
- Adjust the word weights to:
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - ... for the entire training set.

Gradient Descent: Single Step



Loss Function Derivatives

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

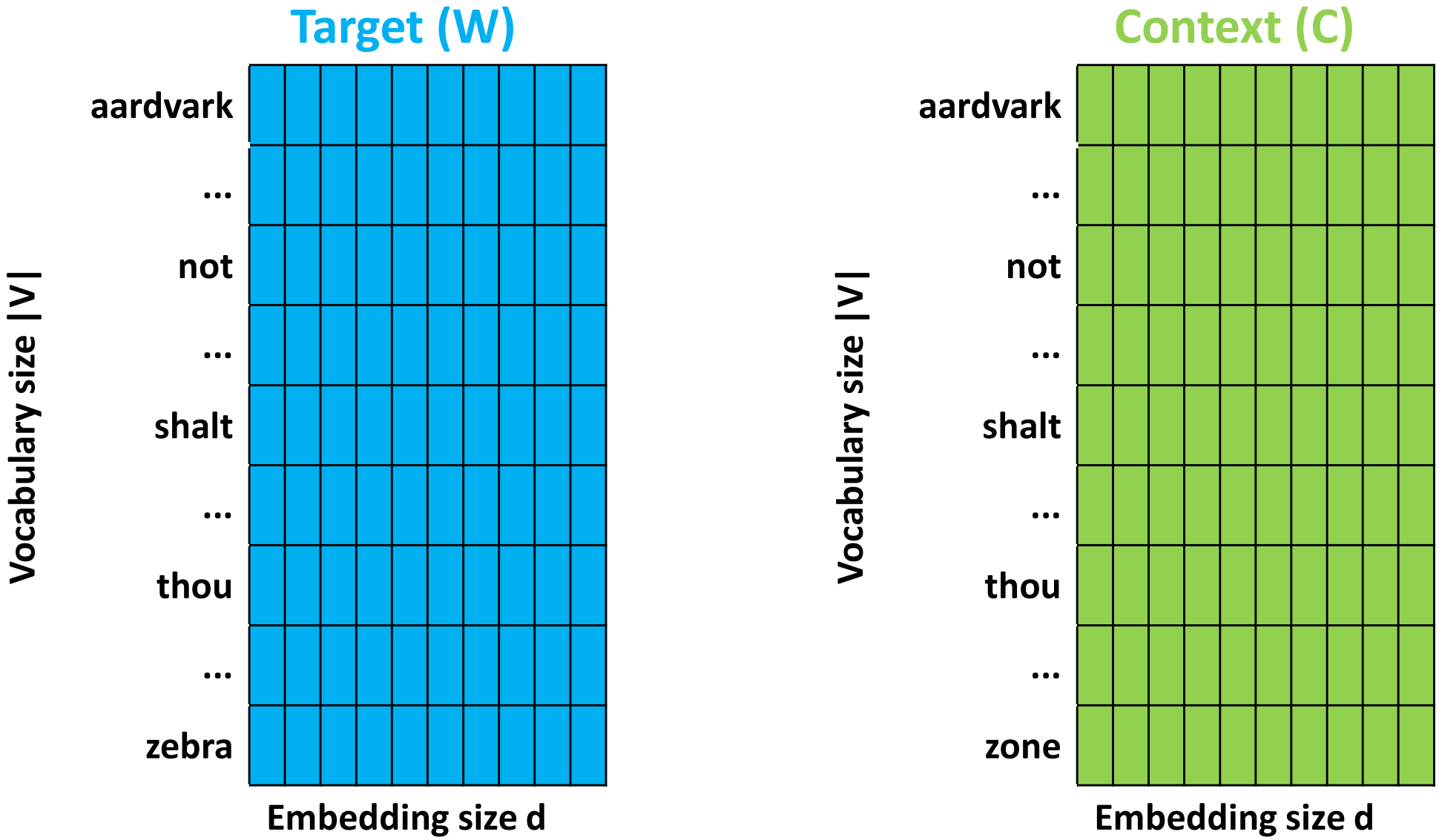
$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Gradient Descent: Updates

Start with randomly initialized W and C matrices



Gradient Descent: Updates

... then incrementally do updates using.

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t}$$

$$c_{neg}^{t+1} = c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t}$$

$$w^{t+1} = w^{t} - \eta \left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w^{t})]c_{neg_i} \right]$$

learning rate

Skip Gram Word2Vec: Summary

- Start with |V| random d-dimensional vectors as initial embeddings
  - Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

Sliding Window Size

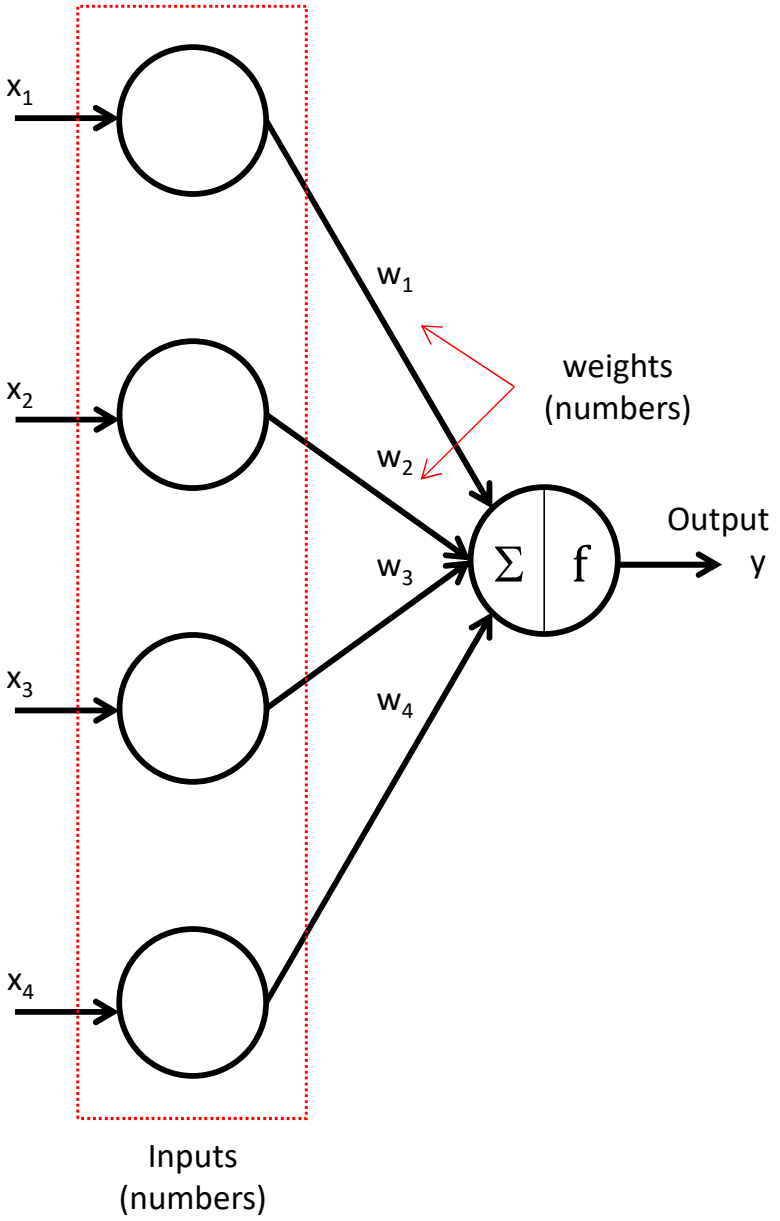
- **Small windows (+/- 2) : nearest words are syntactically similar words in same taxonomy**
  - *Hogwarts* **nearest neighbors are other fictional schools**
    - *Sunnydale*, *Evernight*, *Blandings*

- **Large windows (+/- 5) : nearest words are related words in same semantic field**
  - *Hogwarts* **nearest neighbors are Harry Potter world:**
    - *Dumbledore*, *half-blood*, *Malfoy*

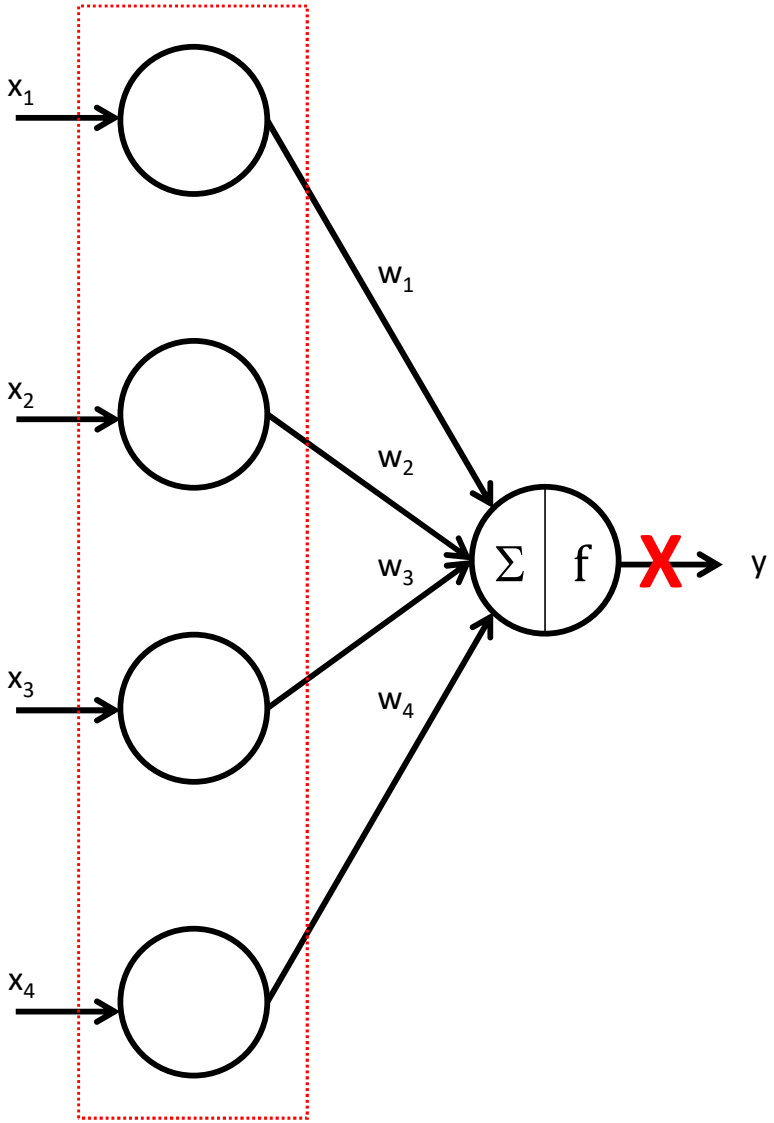
Artificial Neuron (Perceptron)

A (**single-layer**) **perceptron** is a model of a biological neuron. It is made of the following components:

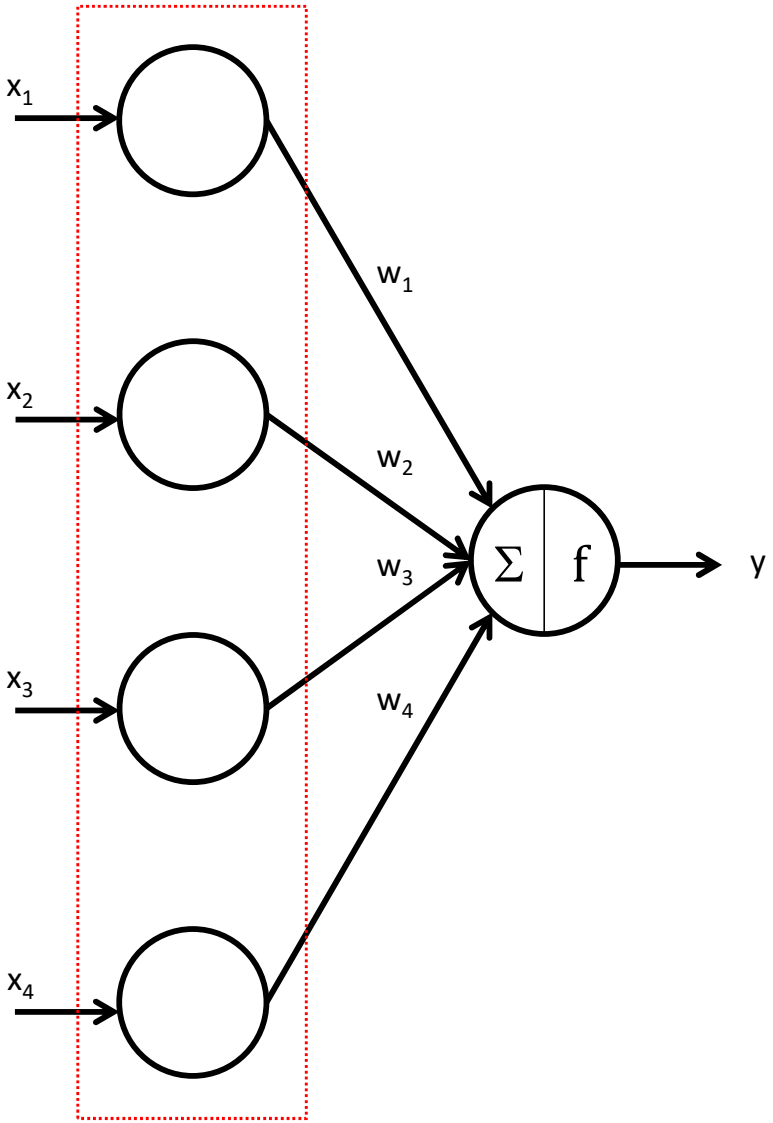
- inputs x_i - numerical values representing information
- weights w_i - numerical values representing how "important" corresponding input is
- weighted sum: $\Sigma\ w_i * x_i$
- activation function f that decides if the neuron "fires"



Single-layer Perceptron as a Classifier

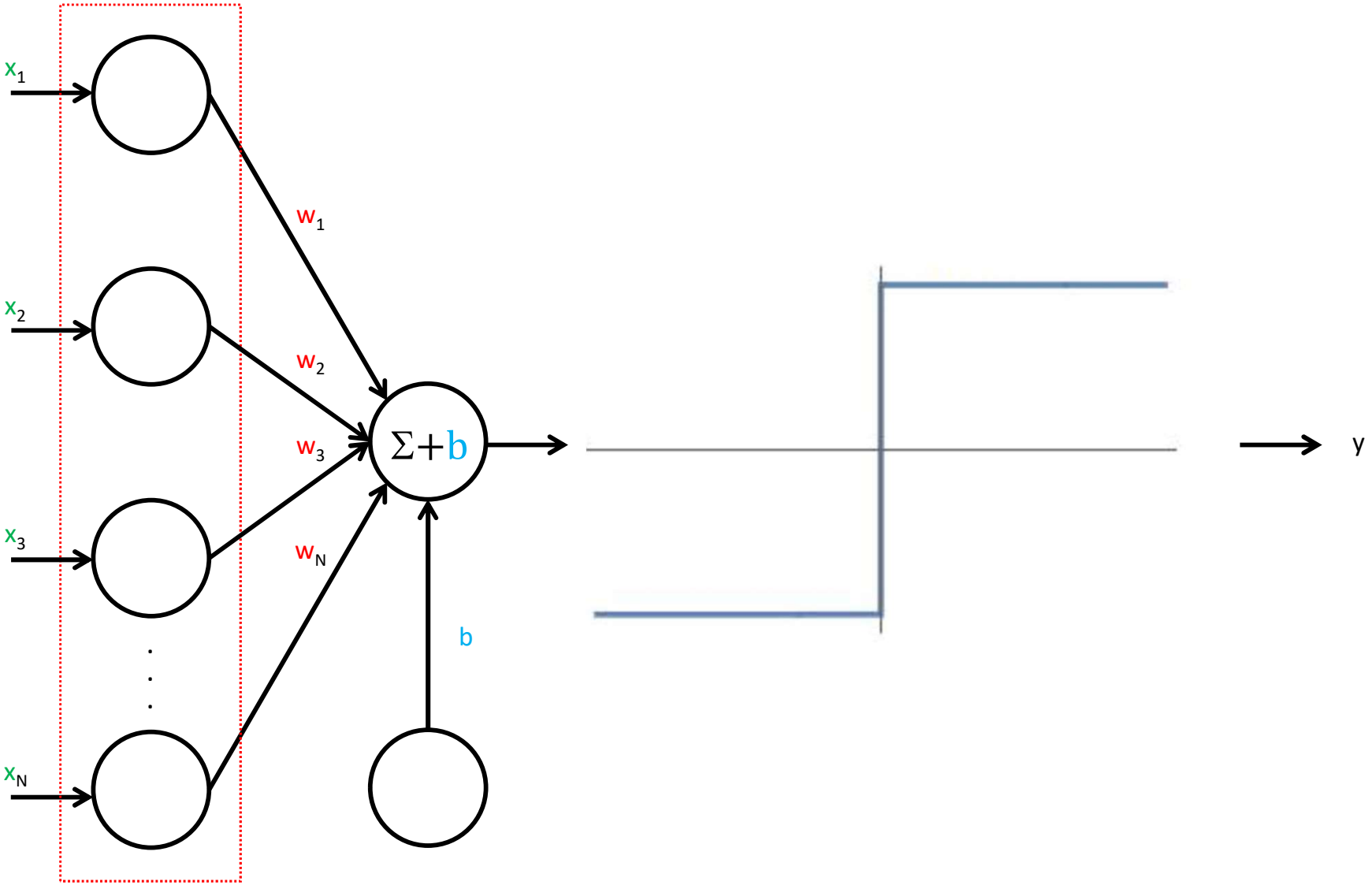


$\Sigma\, w_i * x_i < 0 \rightarrow f = 0 \rightarrow$ **NO**



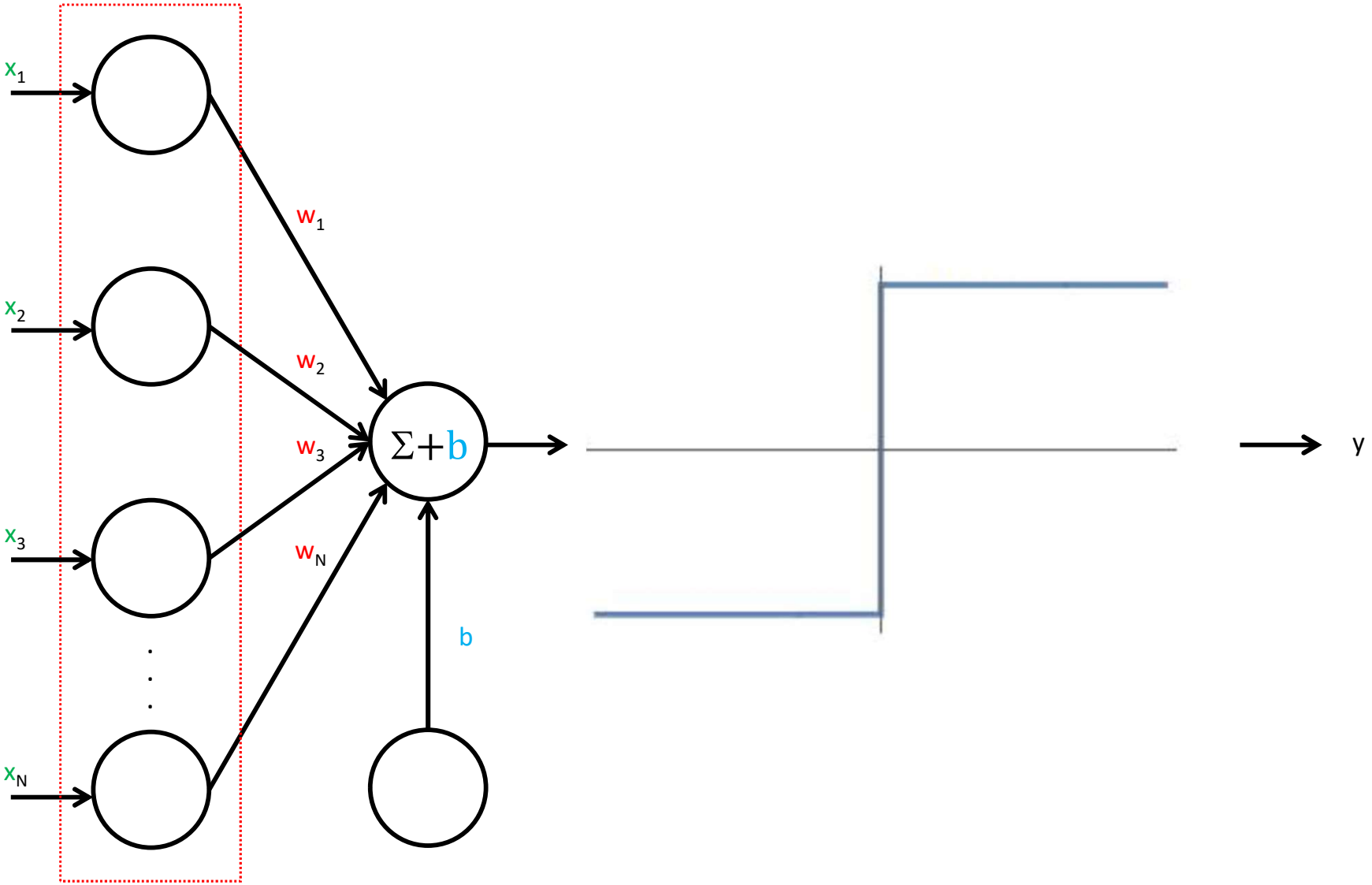
$\Sigma\, w_i * x_i \geq 0 \rightarrow f = 1 \rightarrow$ **YES**

Perceptron with Step Activation



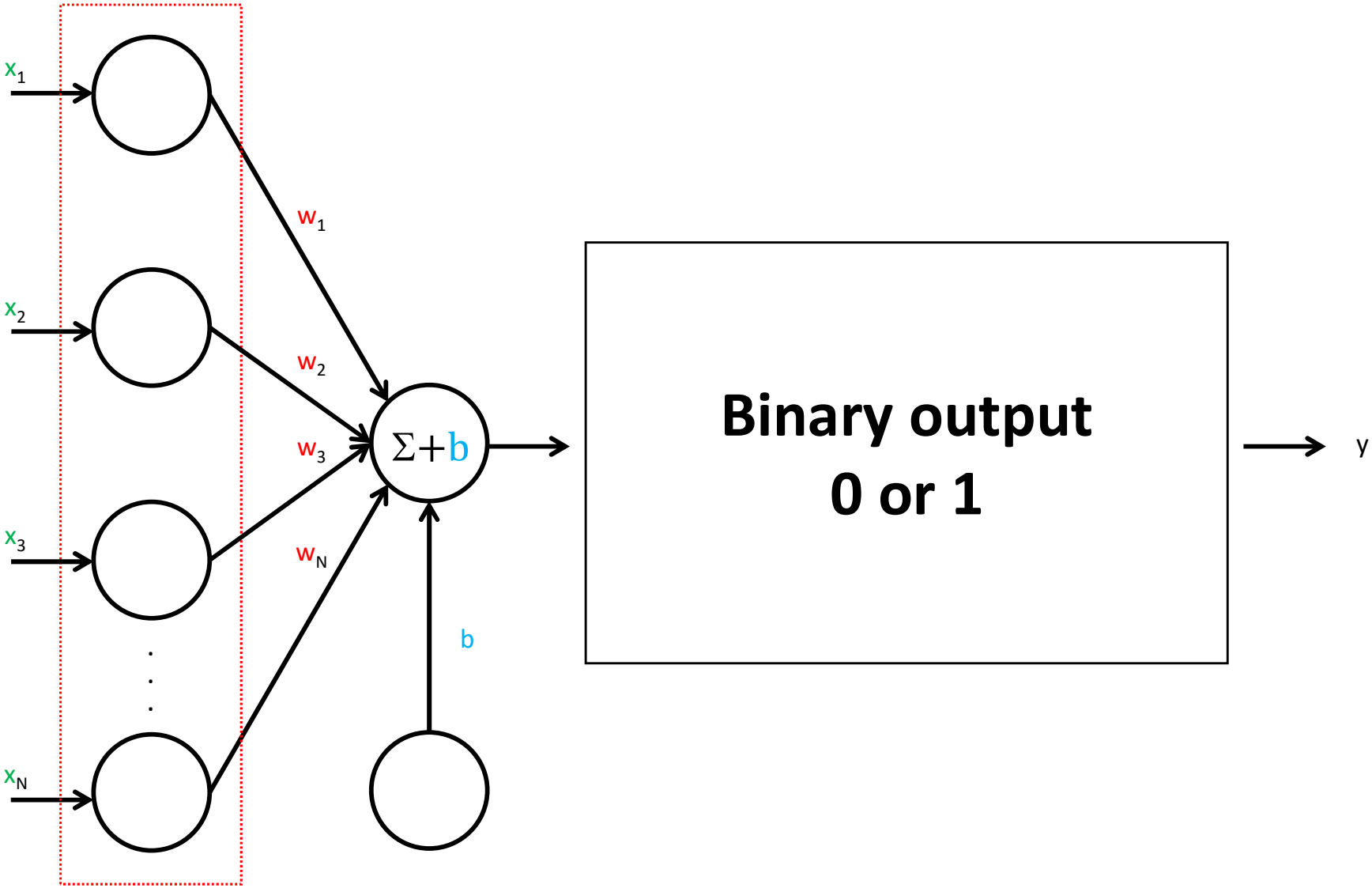
$\Sigma\ w_i * x_i + b \rightarrow f \rightarrow y$

Perceptron with Step Activation



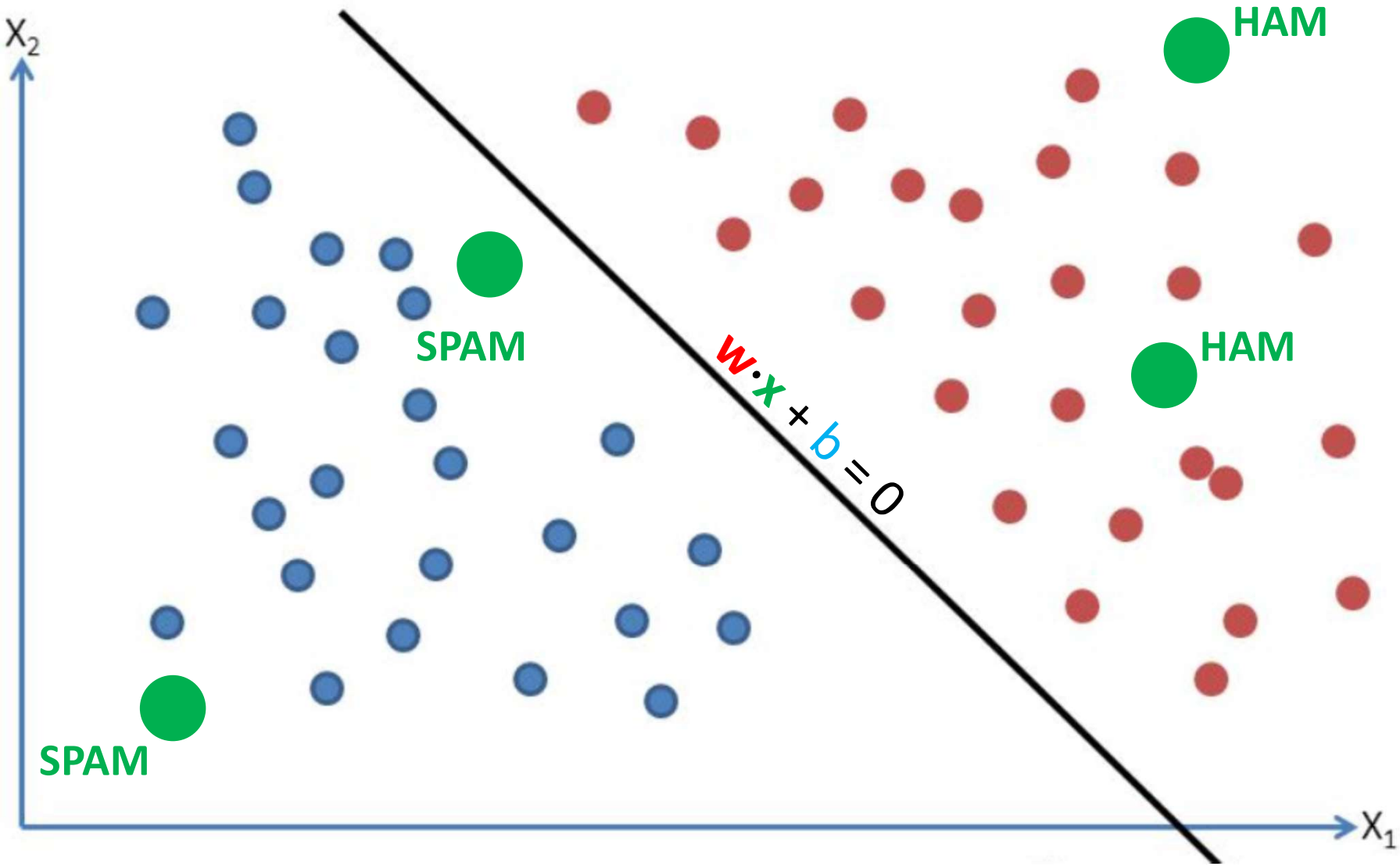
$\Sigma\ w_i * x_i + b \rightarrow f \rightarrow y$

Perceptrons = Linear Classifiers



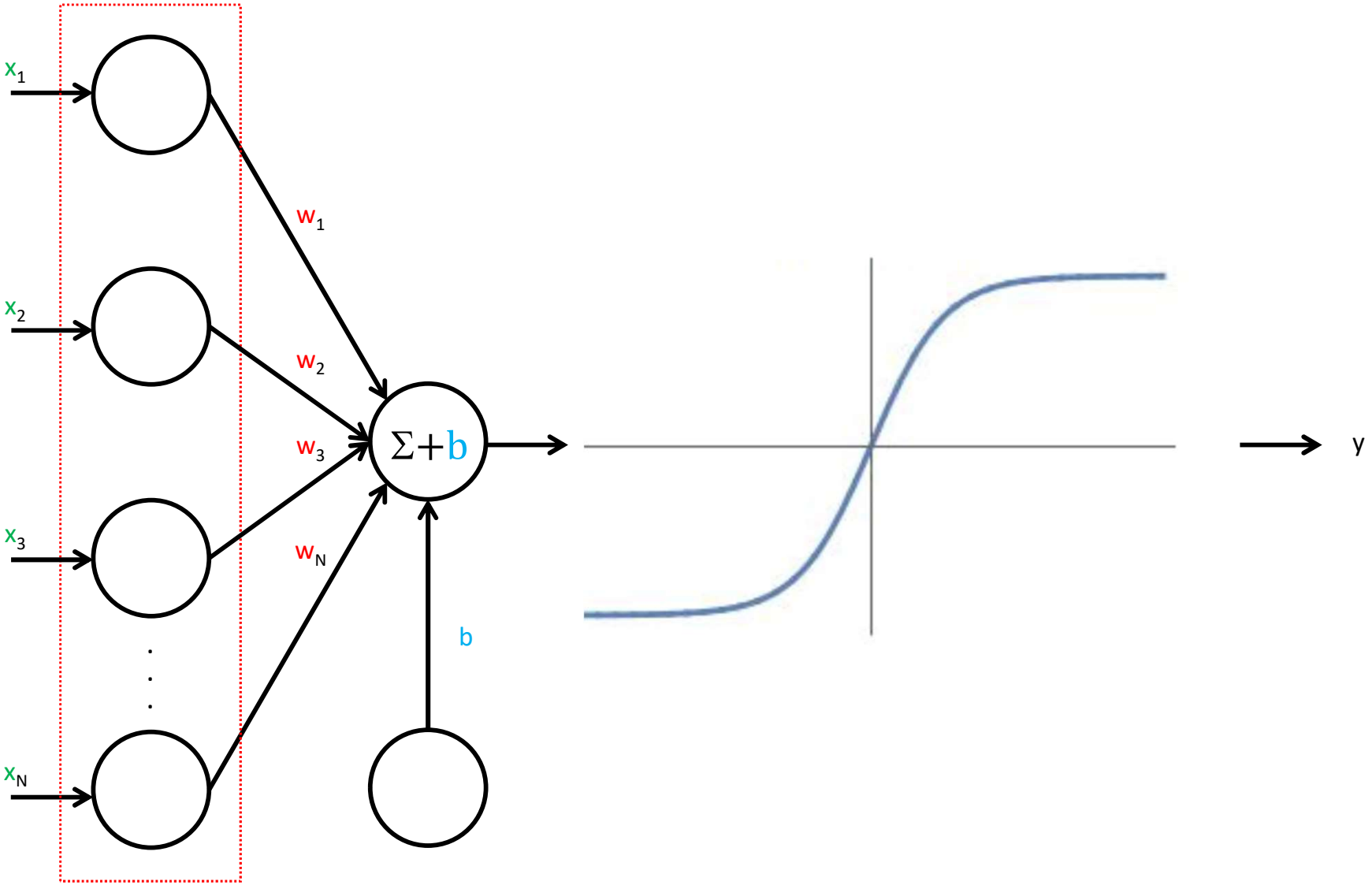
$\Sigma\ w_i * x_i + b \rightarrow f \rightarrow y$

Classification: Linear Separation



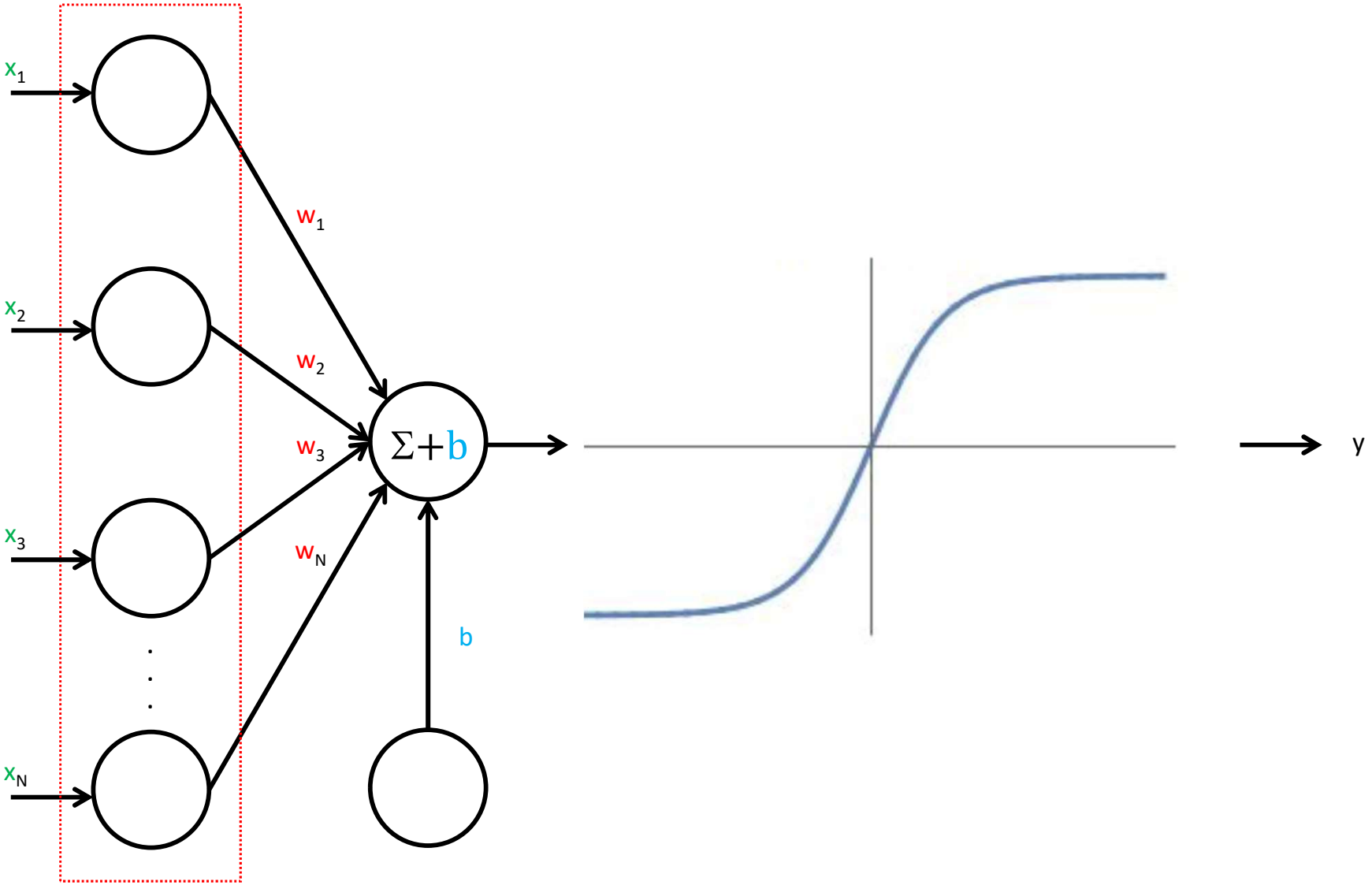
The **w·x** + b = 0 **line is the decision boundary**

Perceptron with Sigmoid Activation



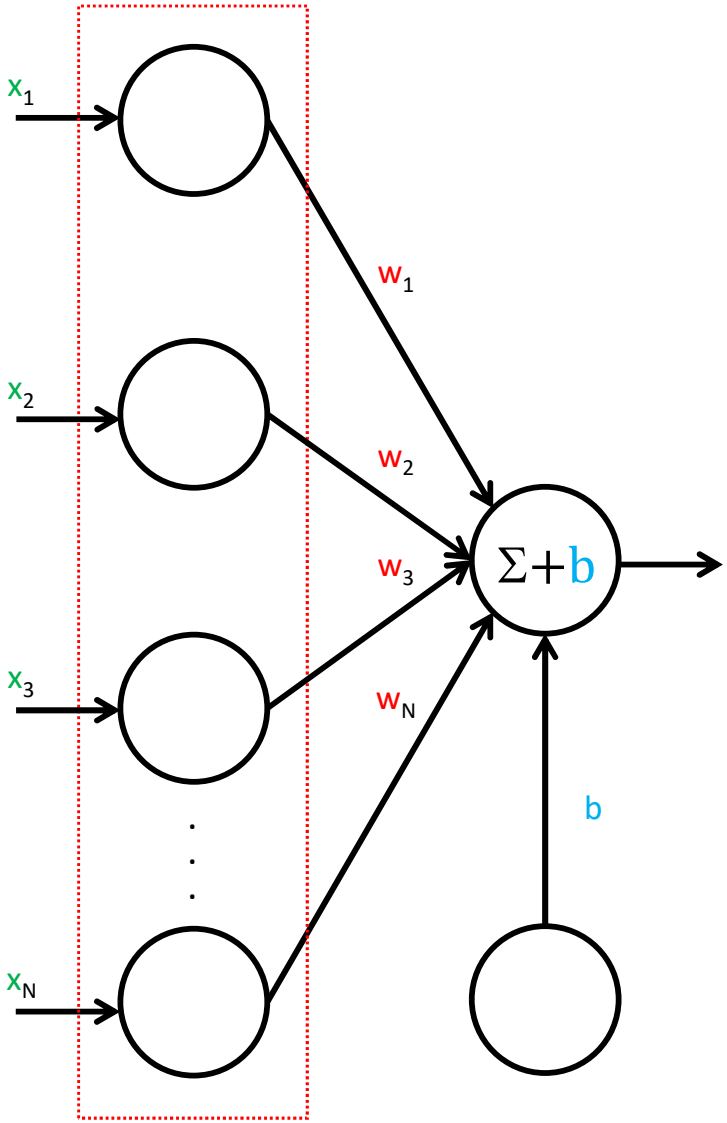
$\Sigma\ w_i * x_i + b \rightarrow f \rightarrow y$

Logistic Regression Classifier



$\Sigma\ w_i * x_i + b \rightarrow f \rightarrow y$

Basic Neural Unit



Nonlinear transform

σ

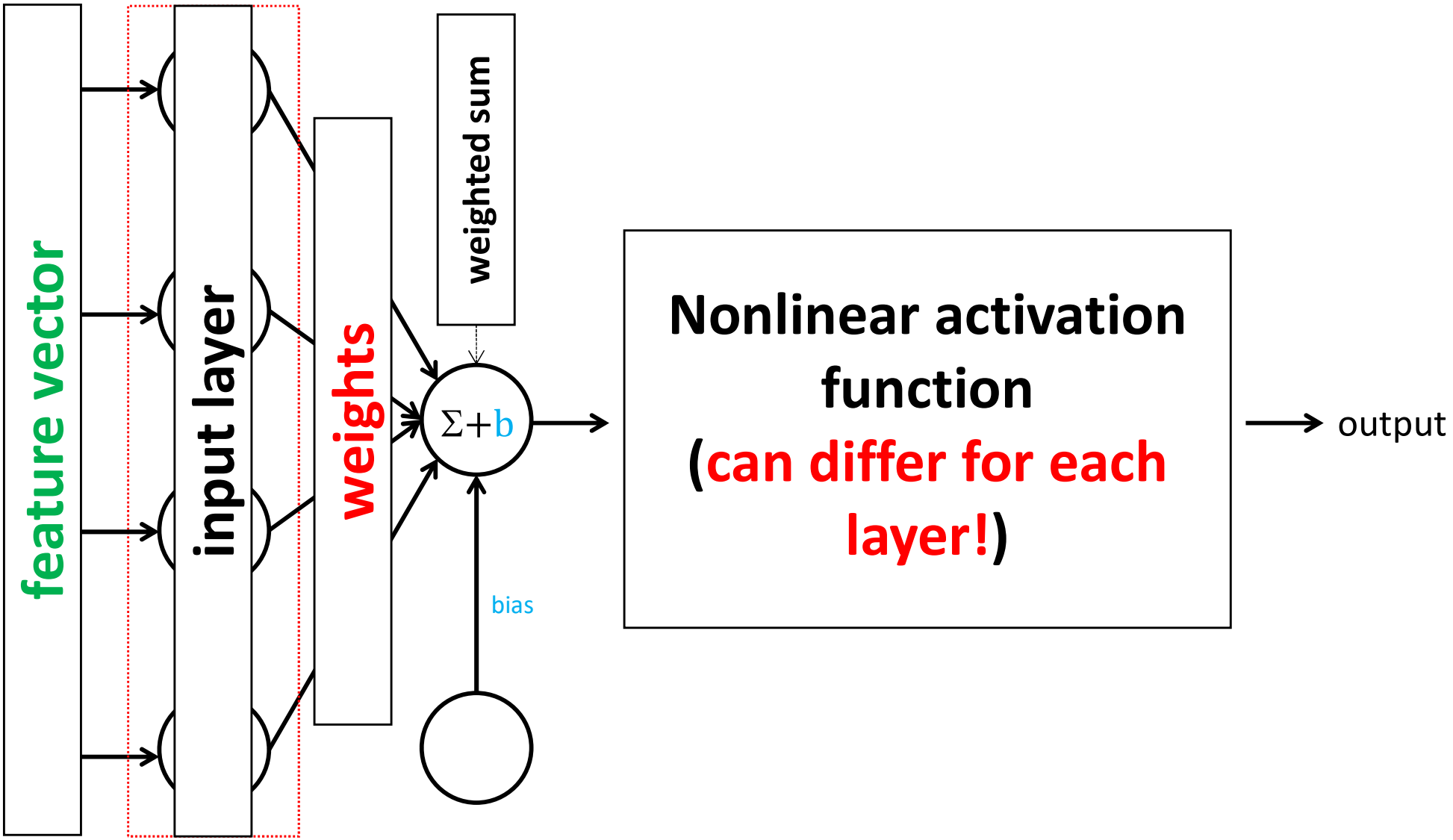
$\sigma(\Sigma+b) = \sigma(z)$

$\rightarrow$ y

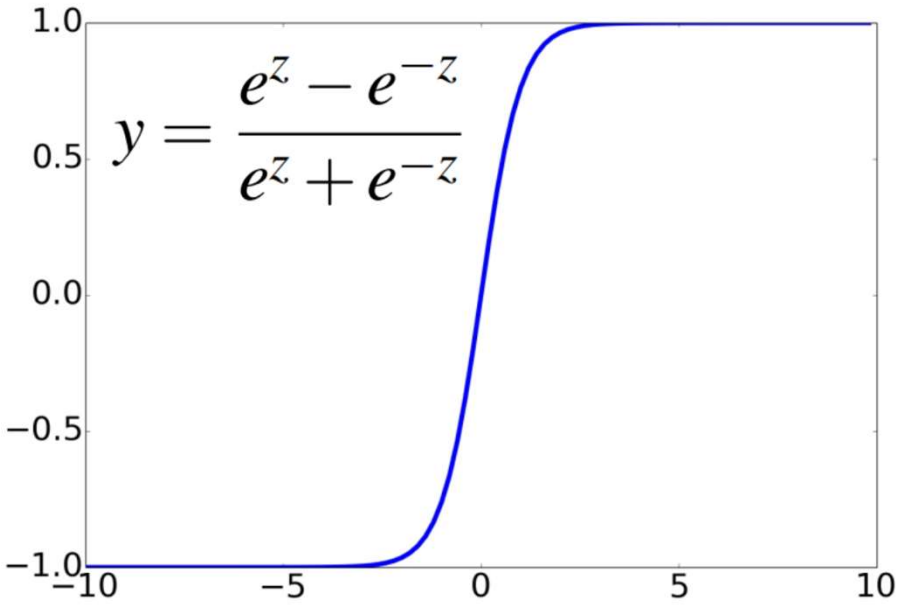
$z = \Sigma\, w_i * x_i + b$

$y = \sigma(z)$

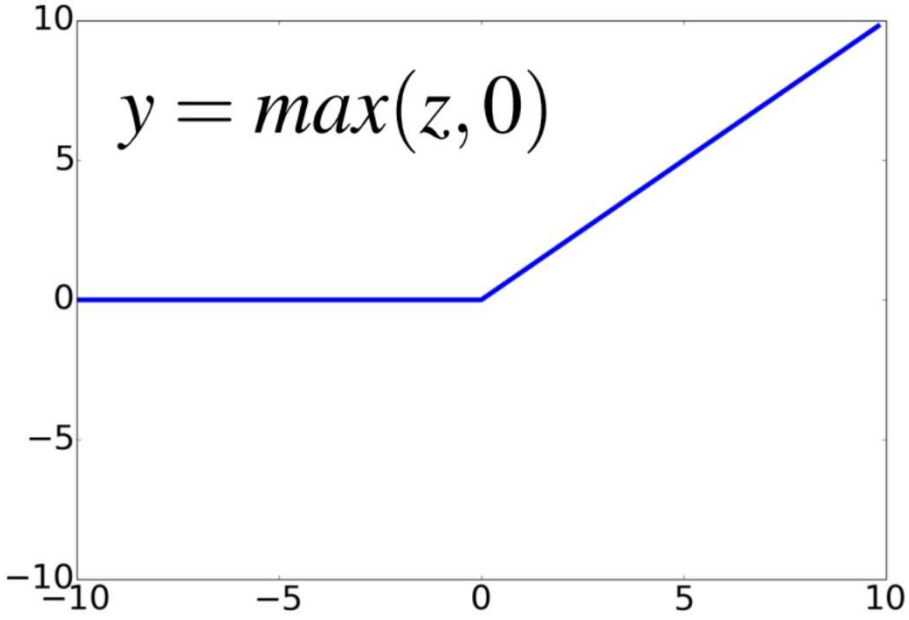
Basic Neural Unit



Other Nonlinear Activation Functions

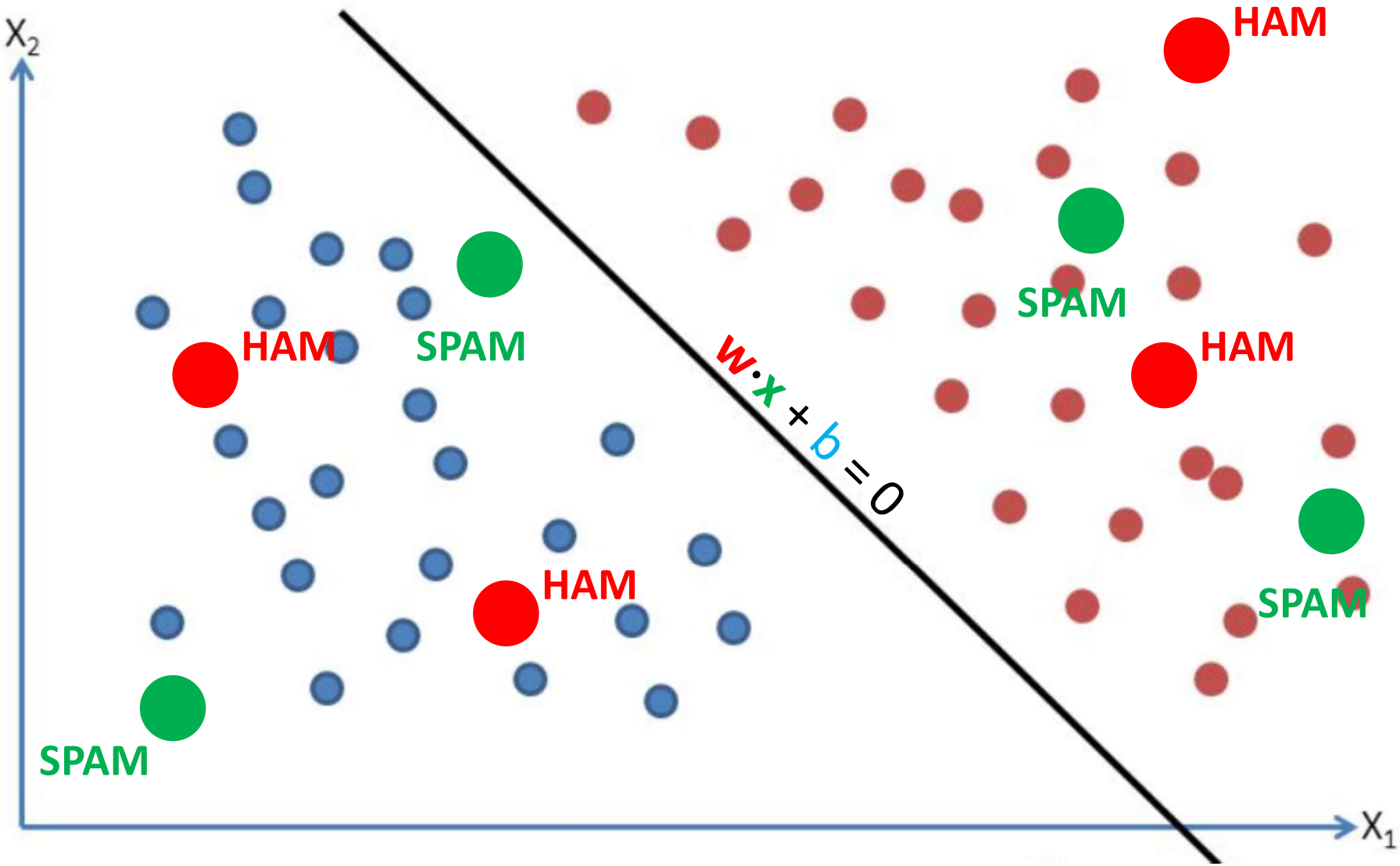


tanh
(hyperbolic tangent)



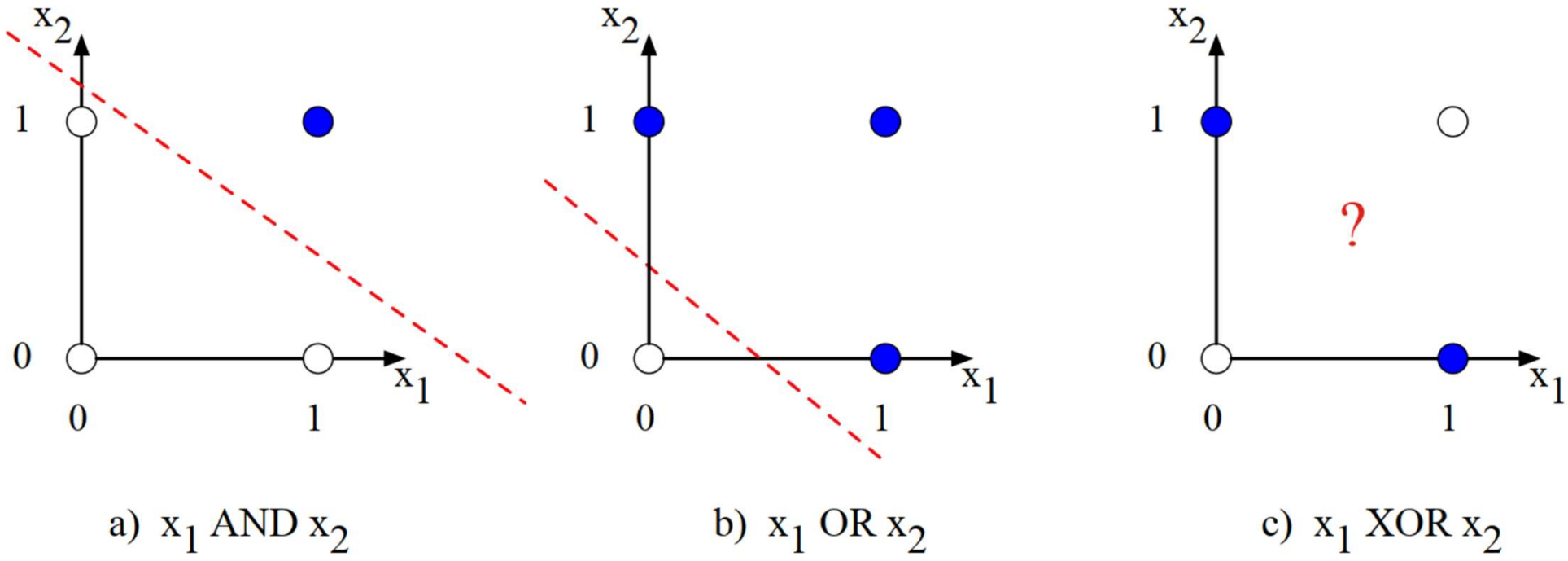
ReLU
(Rectified Linear Unit)

Classification: Linear Separation?



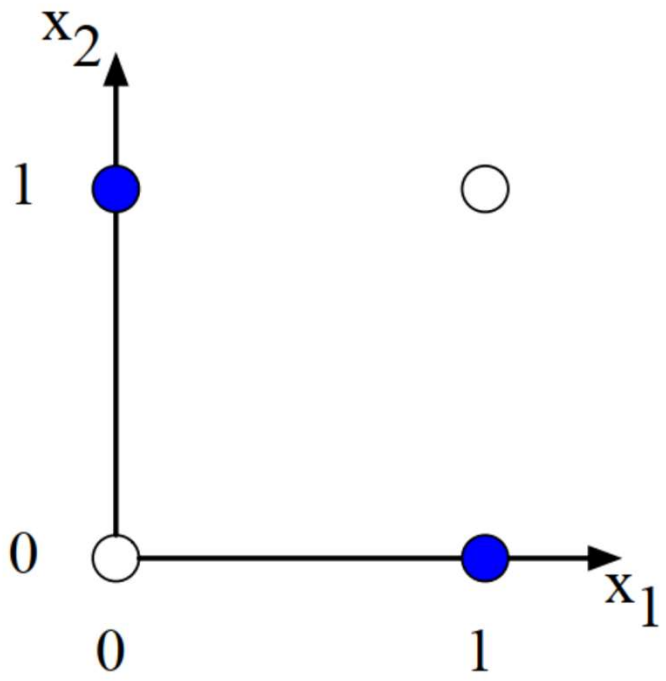
Sometimes decision boundary CANNOT be linear? Not linearly separable f()

XOR: Not a Linearly Separable f()

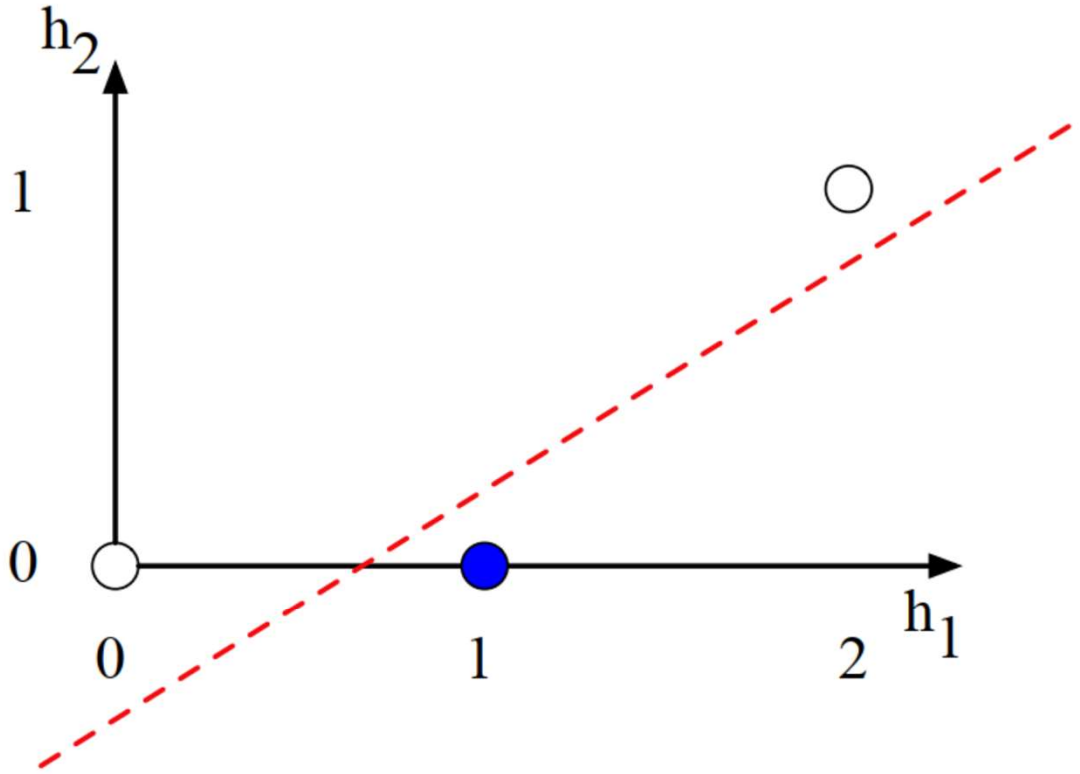


Logical XOR is an example of a function that is NOT linearly separable

XOR: Not a Linearly Separable f()?

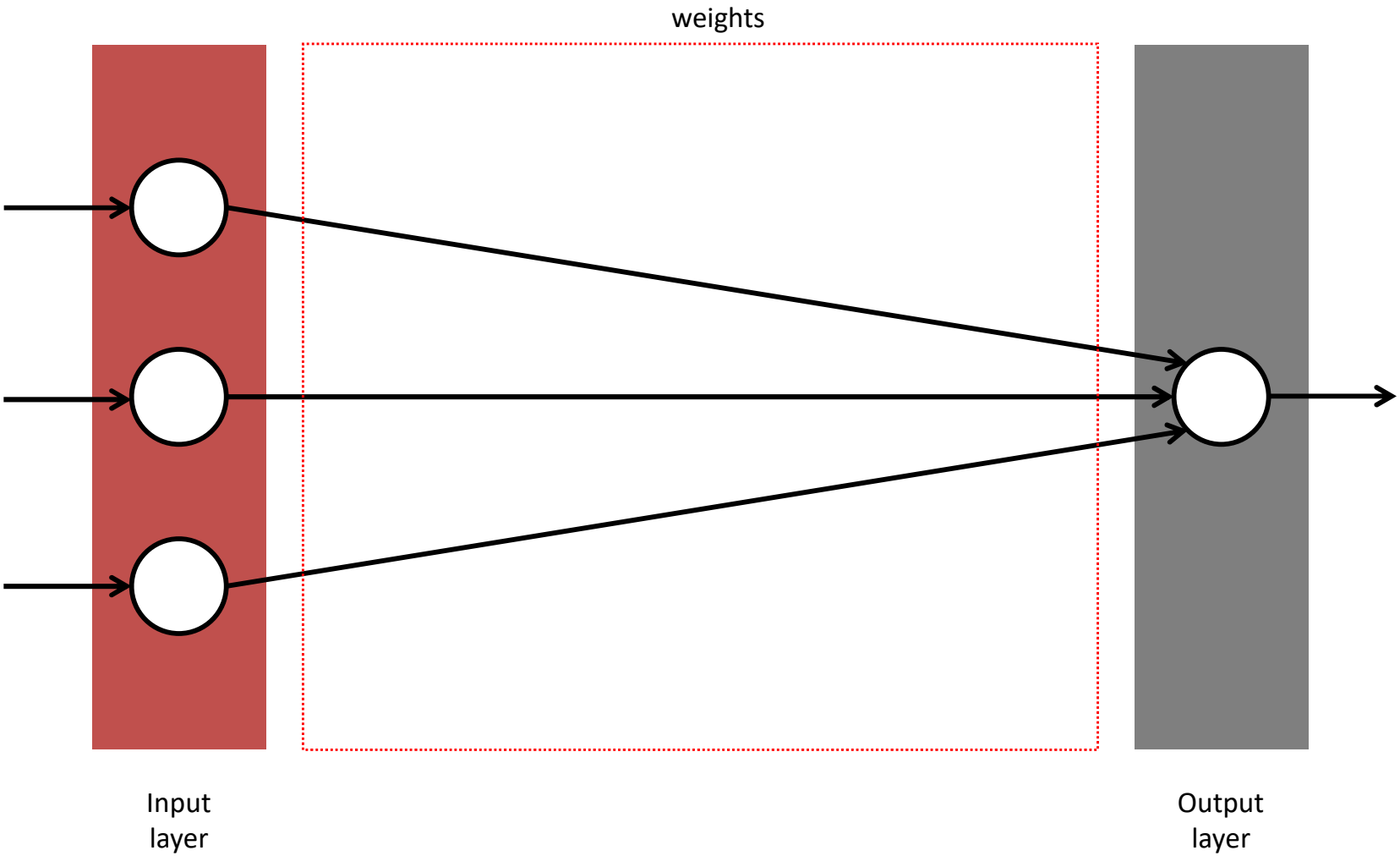


a) The original *x* space

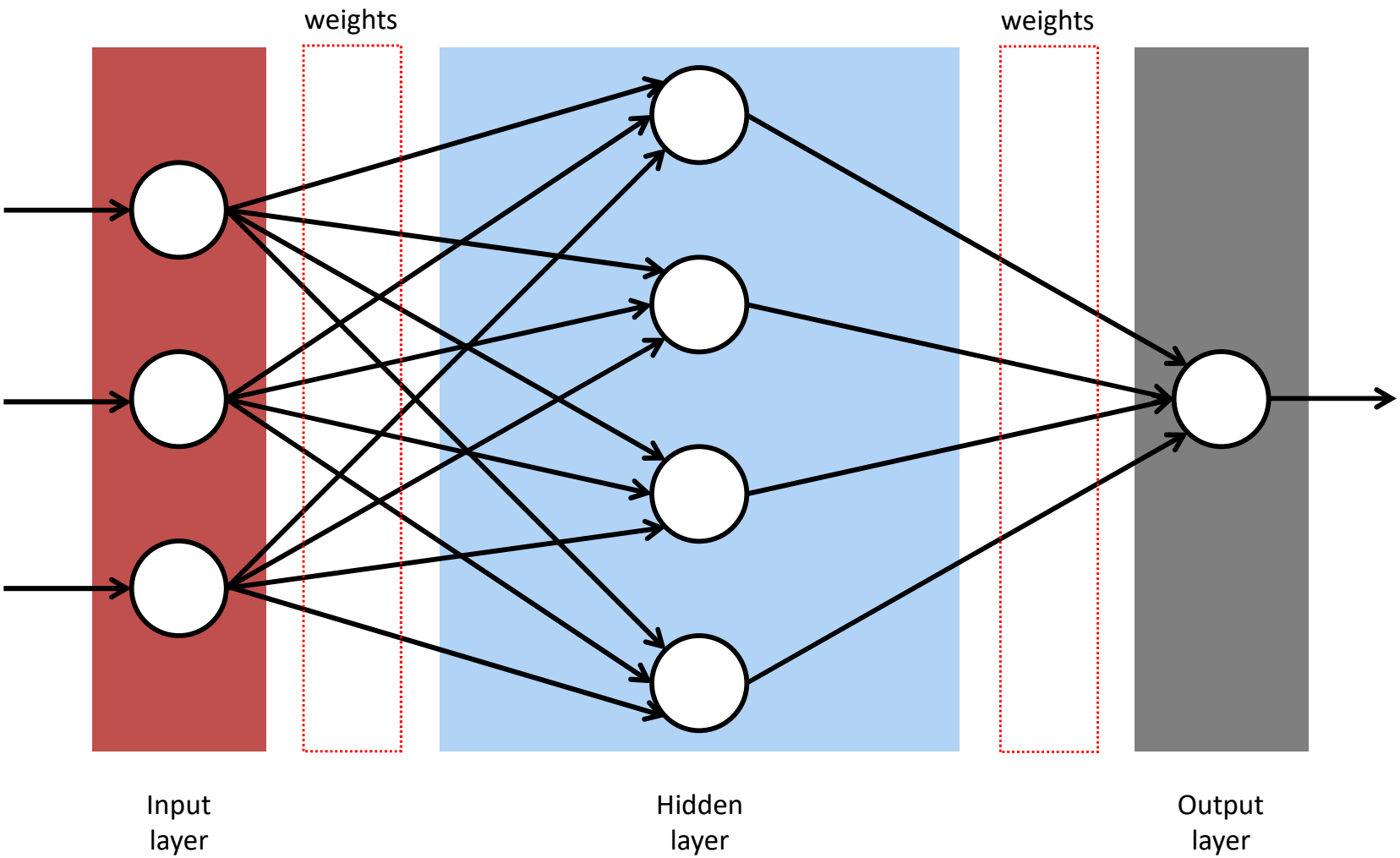


b) The new (linearly separable) *h* space

Basic Neural Unit

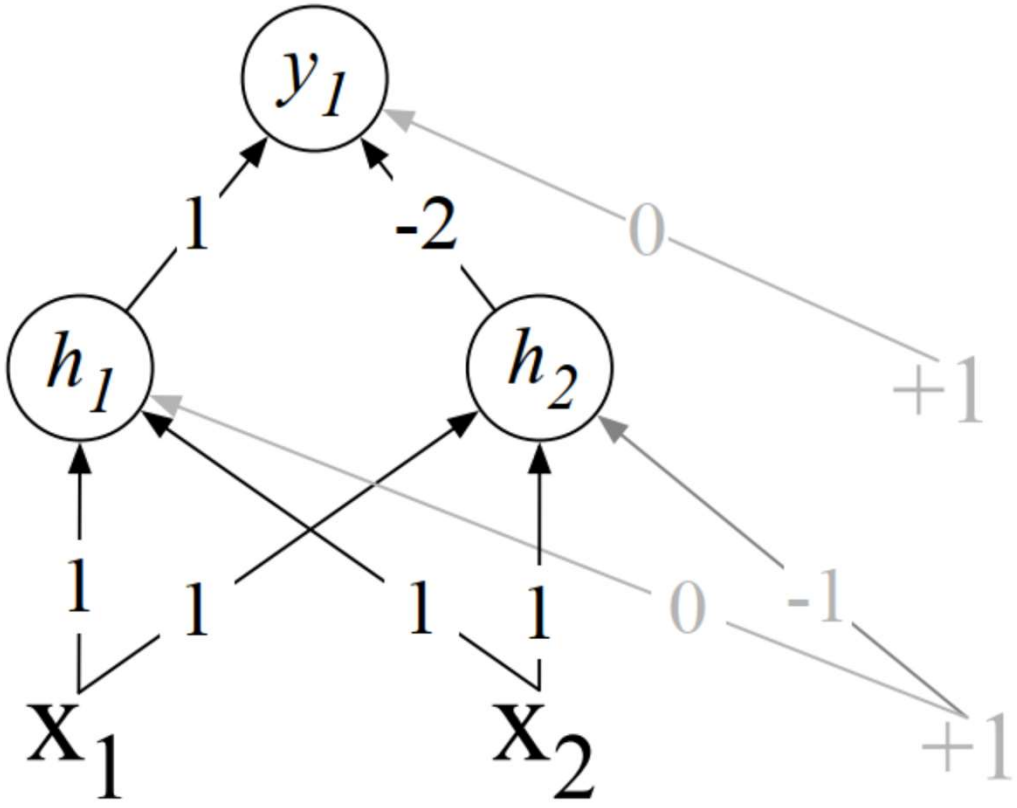


Hidden Layer

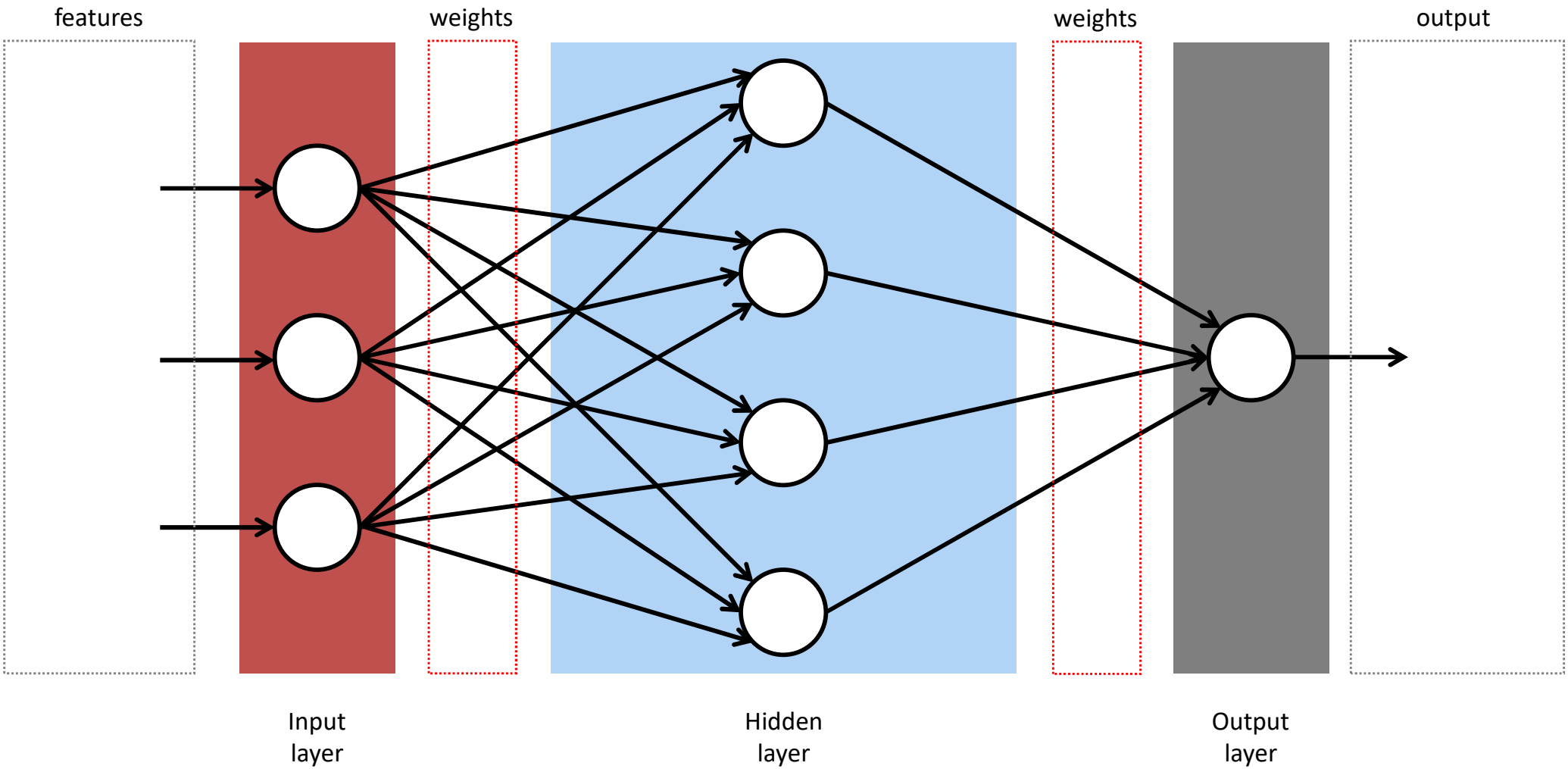


XOR: Hidden Layer Approach

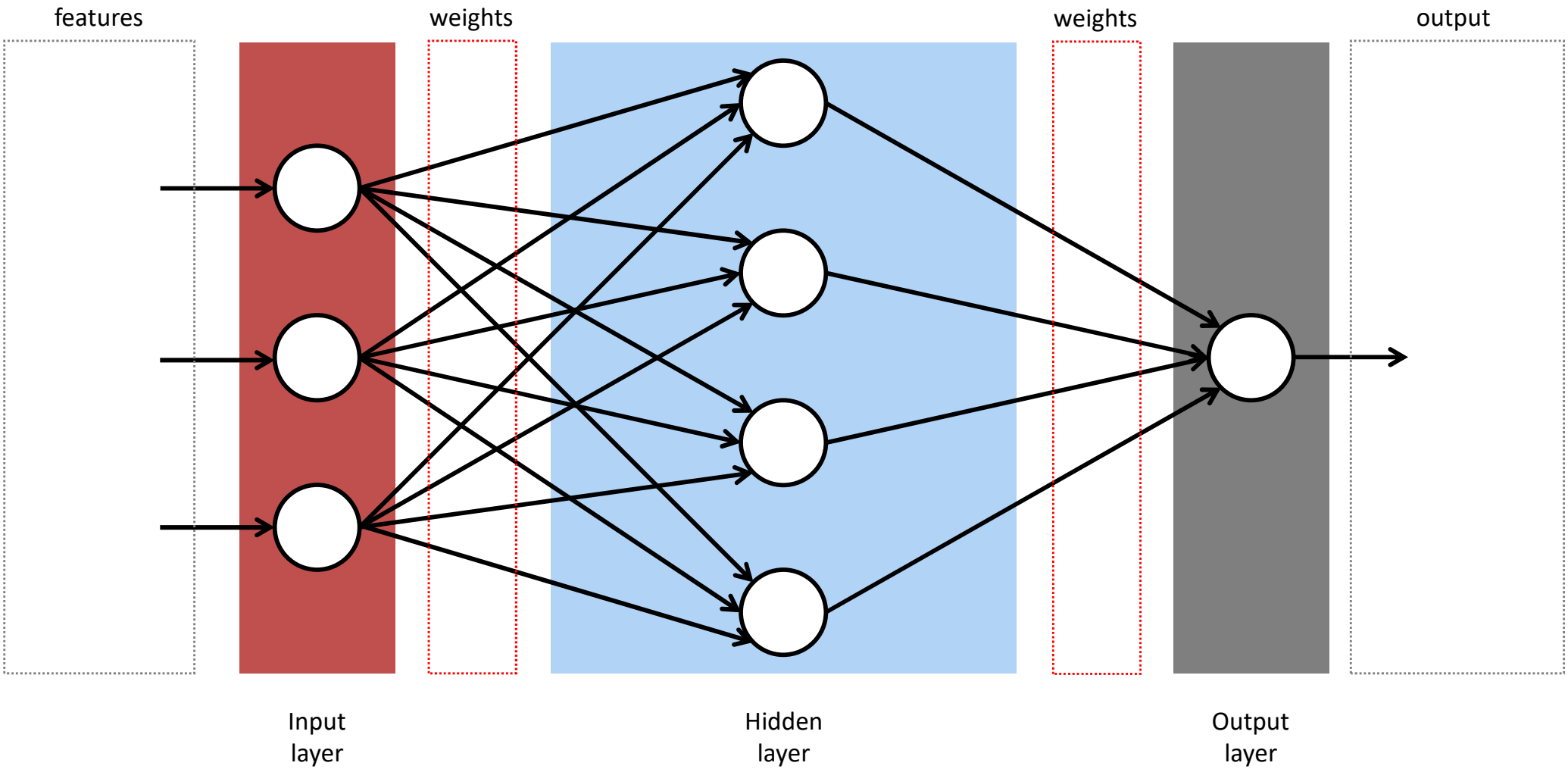
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



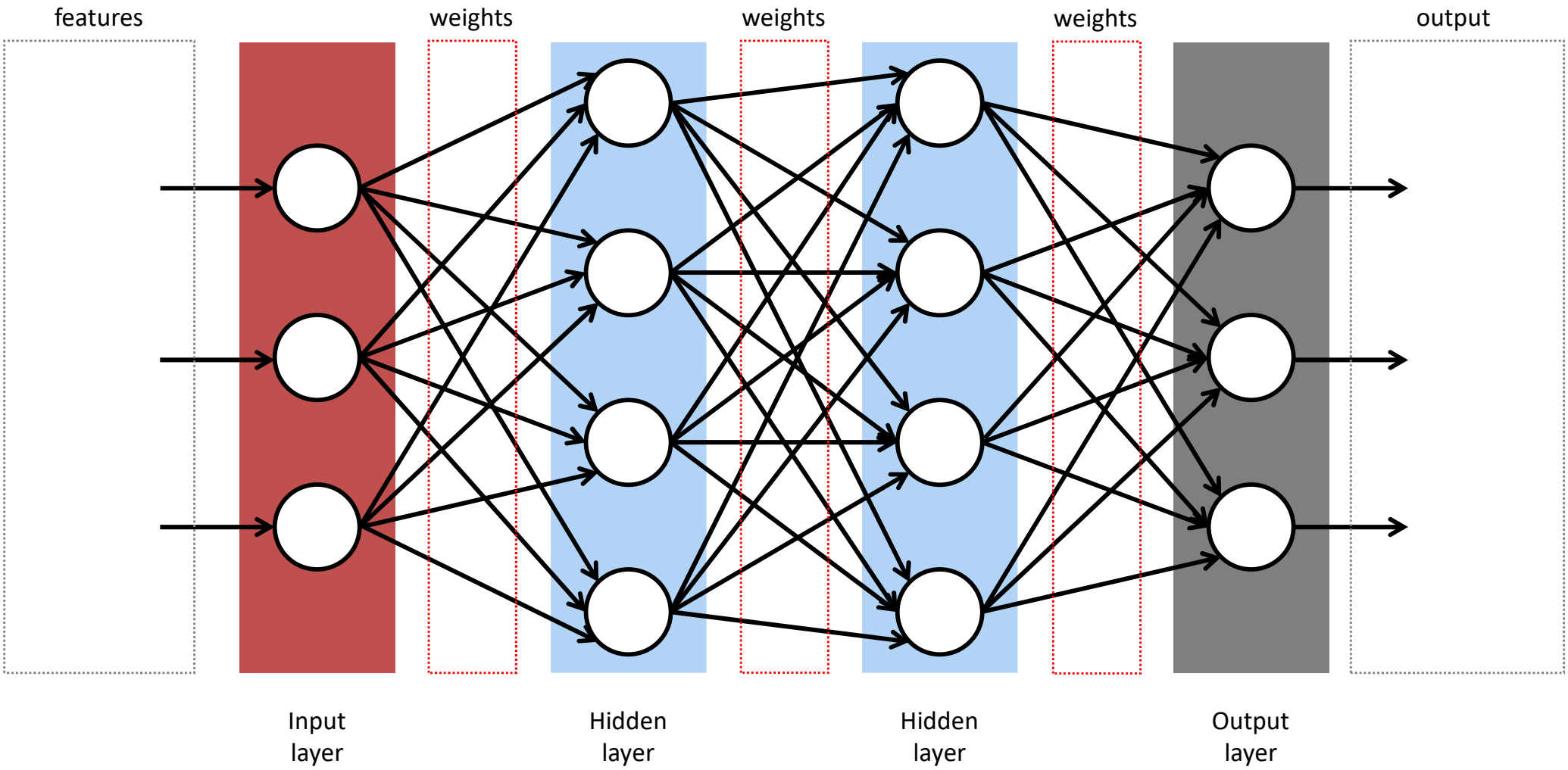
Hidden Layer



Hidden Layer

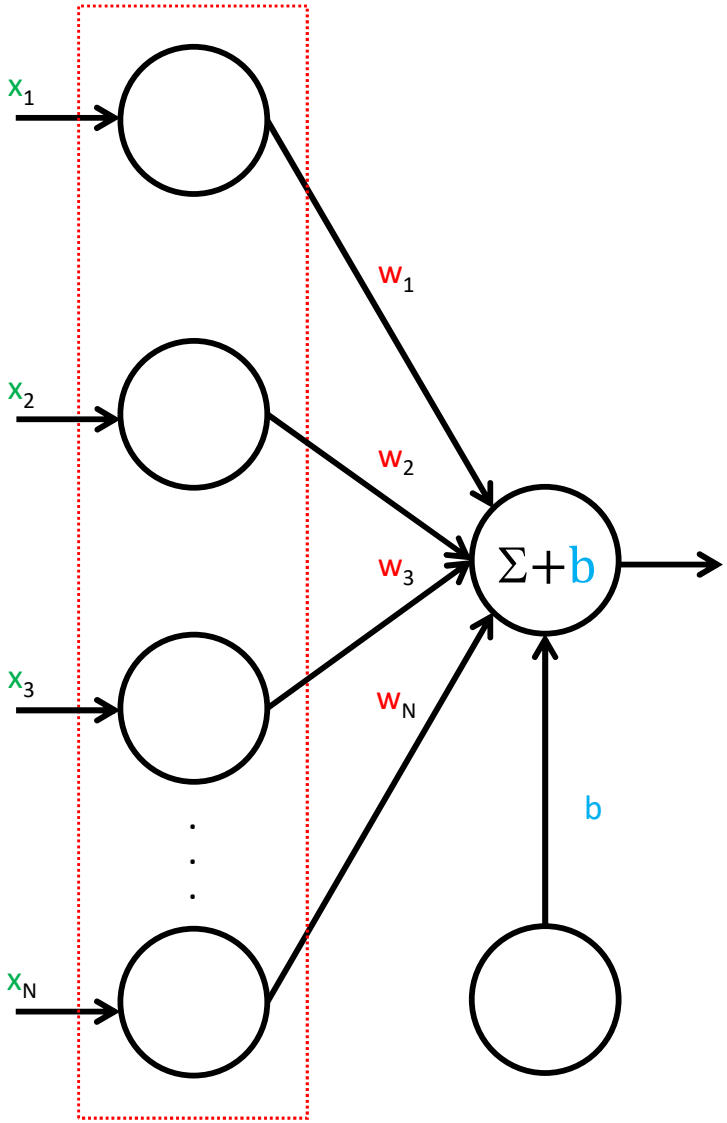


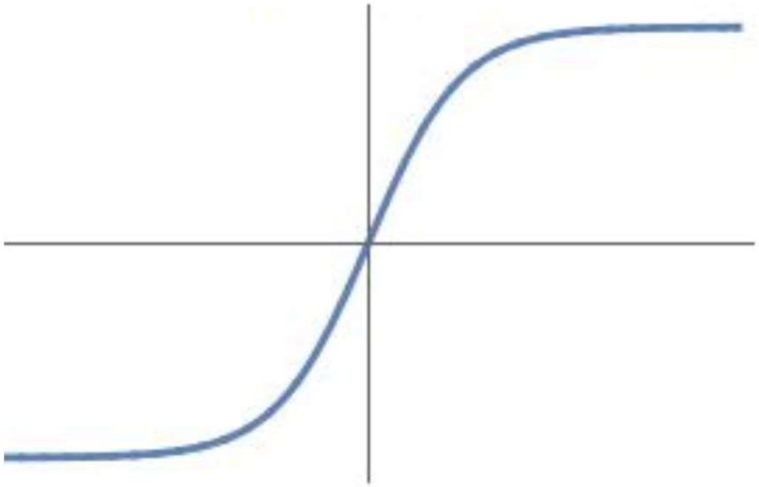
Feedforward Neural Network



Also called (historically): multi-layer perceptron

Logistic Regression = 1 Layer Network



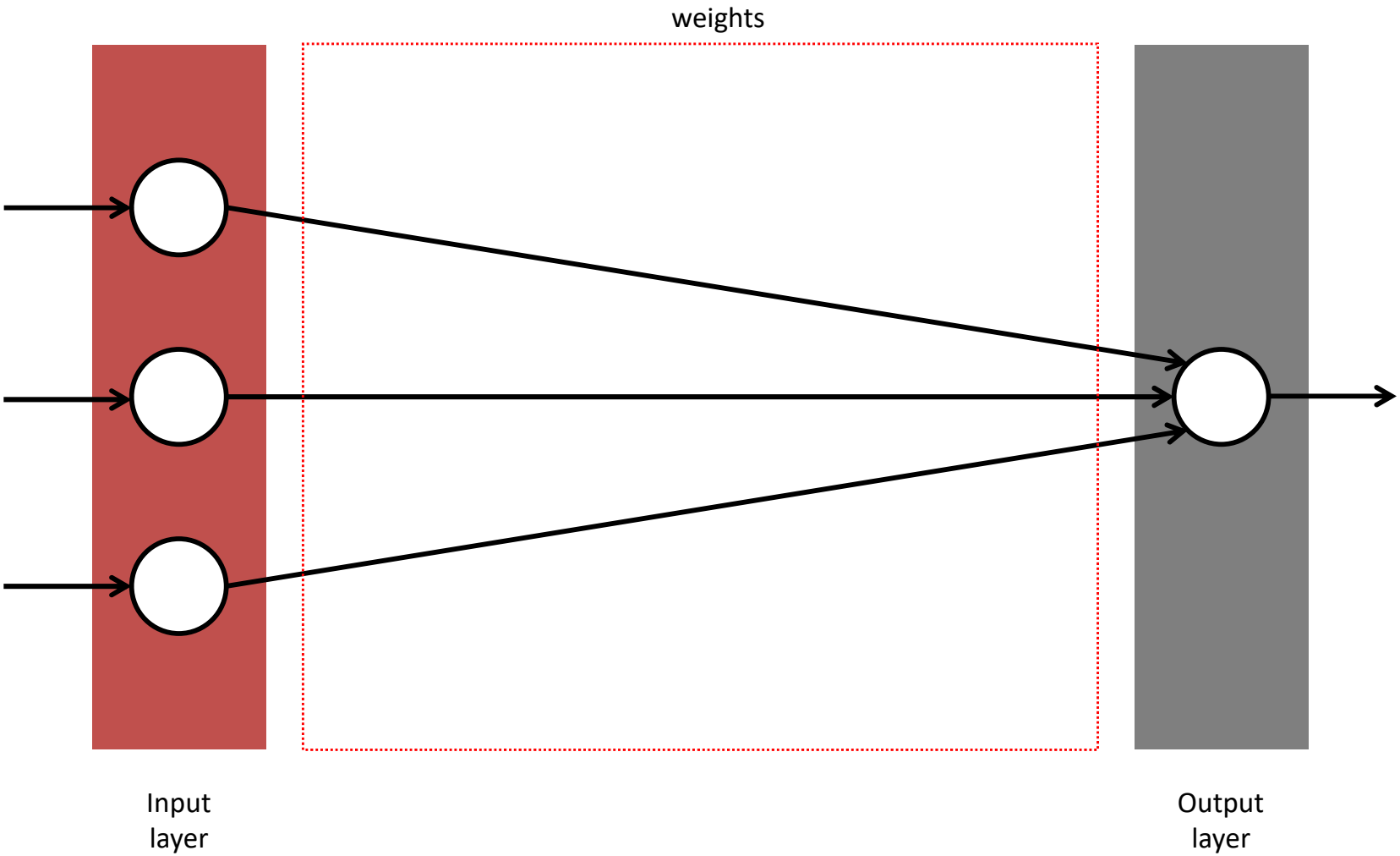


y

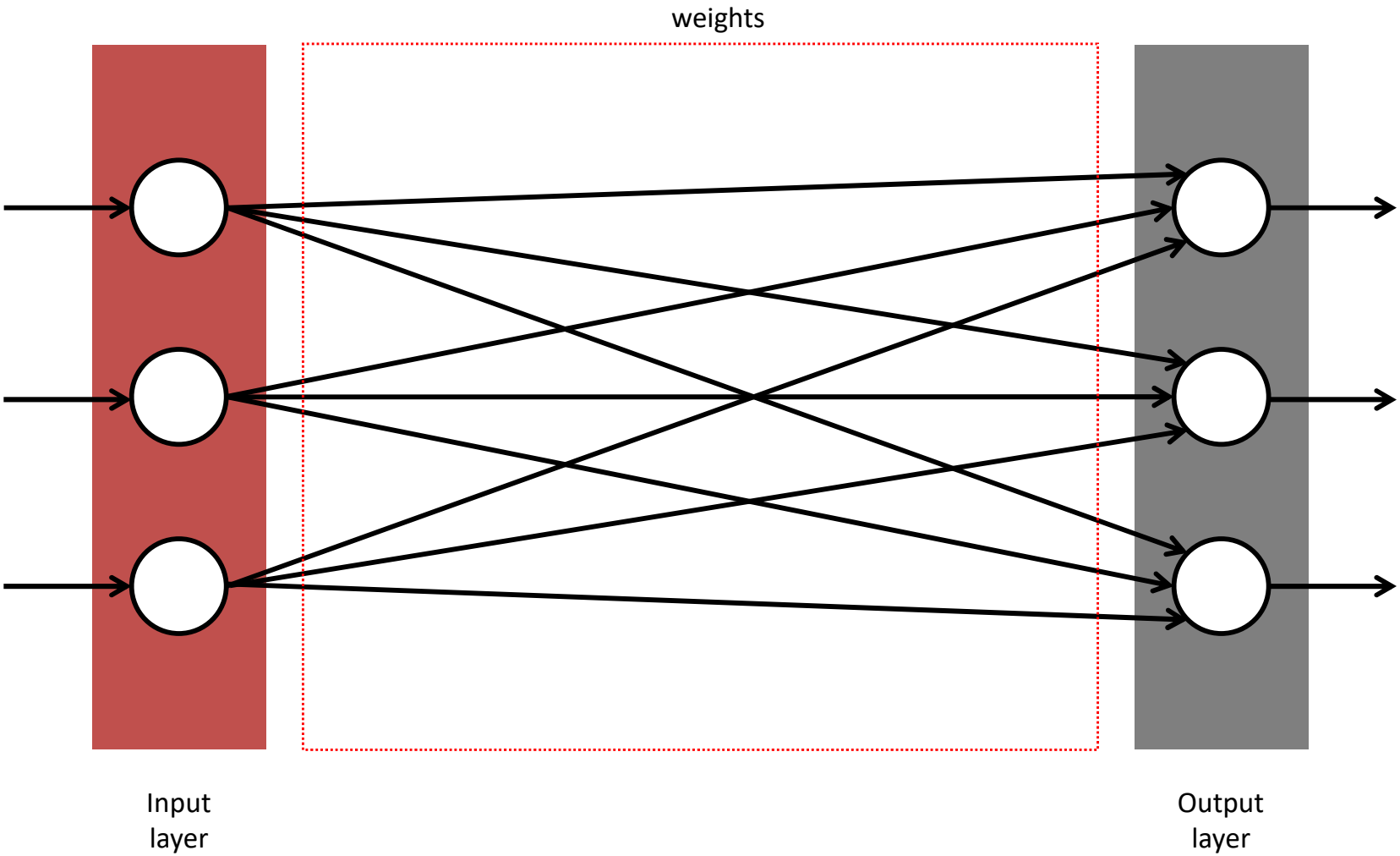
scalar

$\Sigma\ w_i * x_i + b \rightarrow f \rightarrow y$

Binary Logistic Regression = 1 Layer

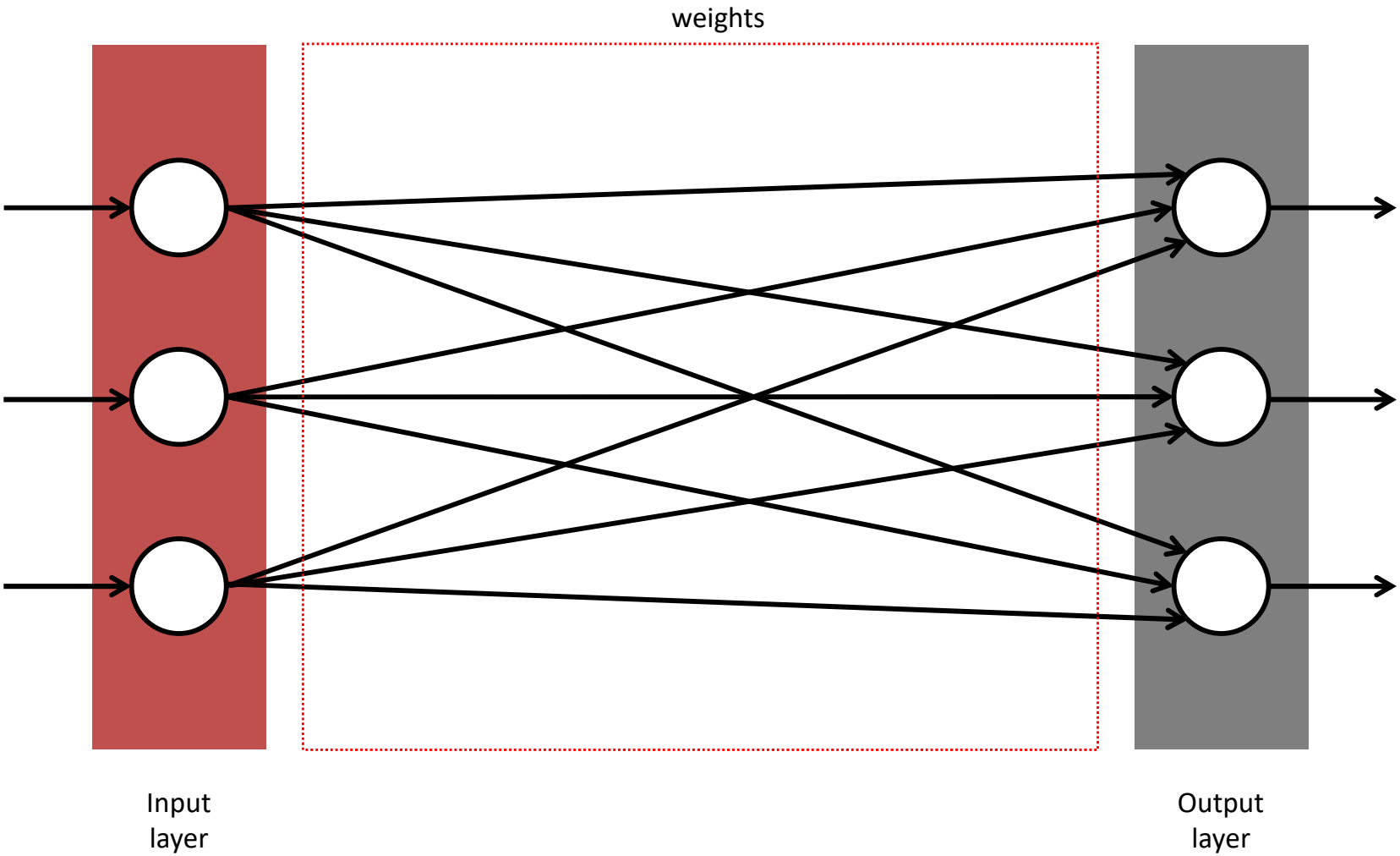


Multinomial Logistic Regression



Multinomial Logistic Regression is still a 1 Layer Network

Fully Connected Network



This Multinomial Logistic Regression network is fully connected network

Softmax: Sigmoid Generalization

For a vector z of dimensionality k, the softmax is:

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

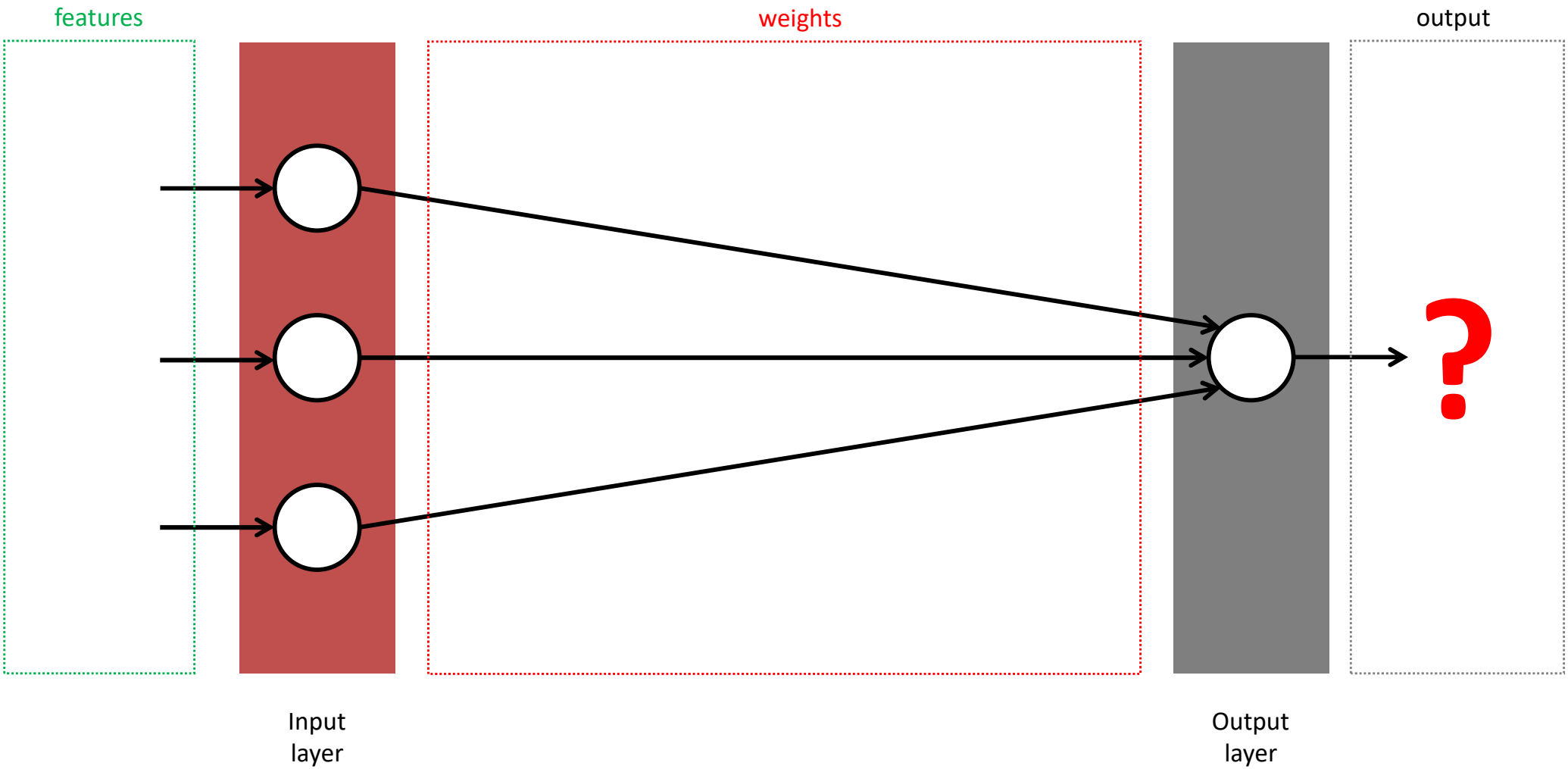
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \leq i \leq k$$

Example:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

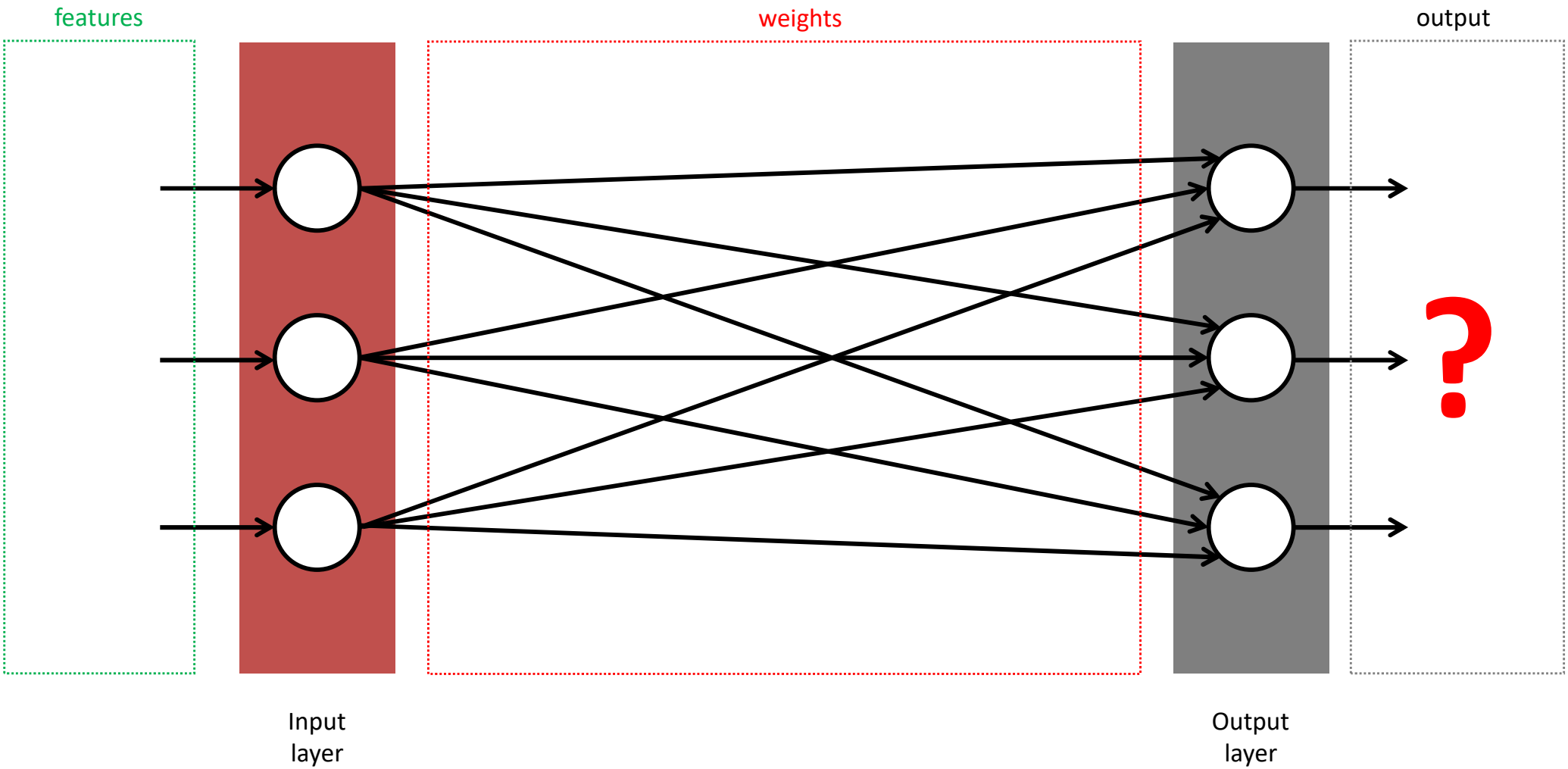
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Binary Logistic Regression



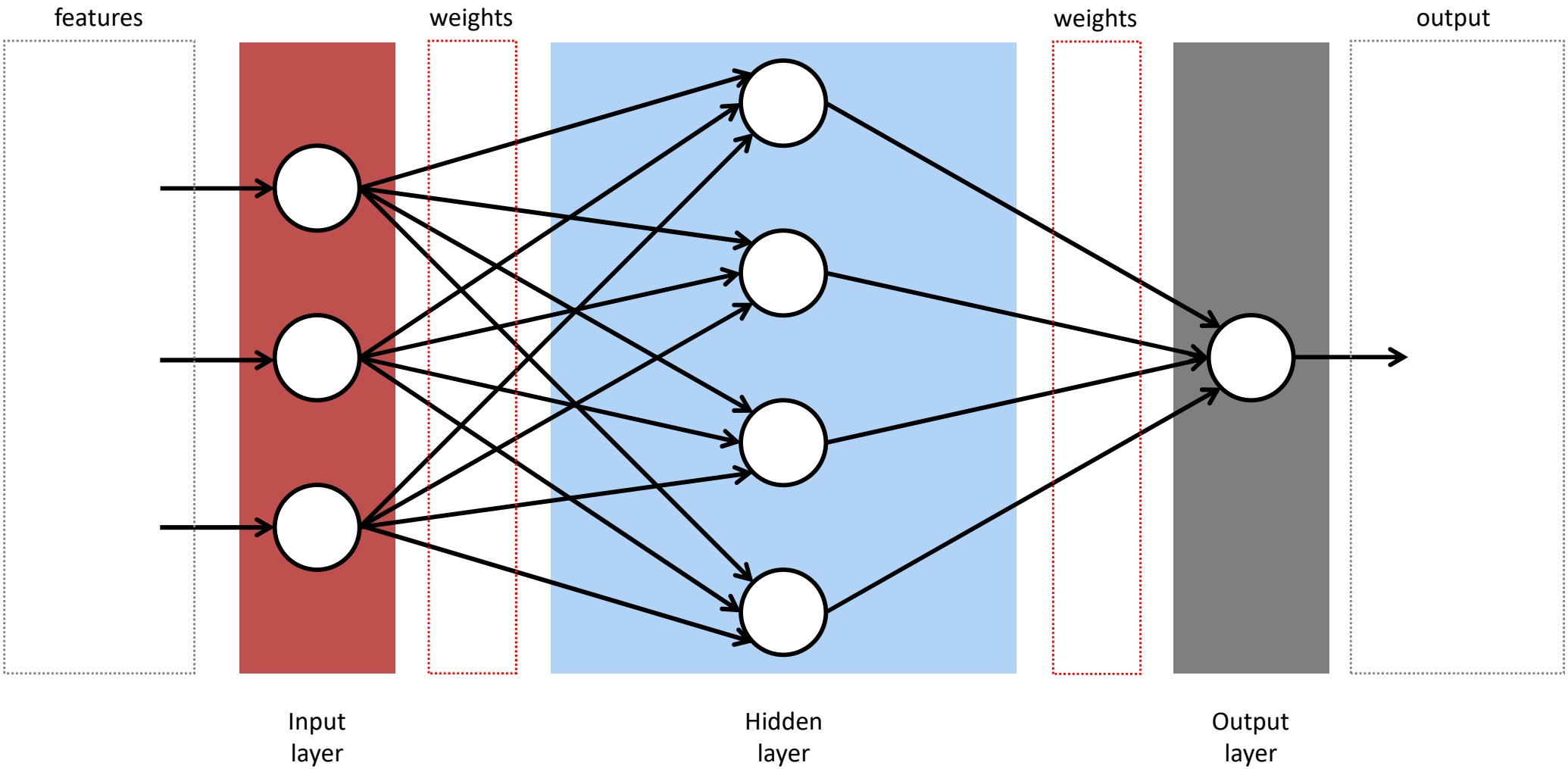
In Binary Logistic Regression output is a SCALAR $y = \sigma(z) = \sigma(w_i * x_i + b)$ **[σ - sigmoid]**

Multinomial Logistic Regression

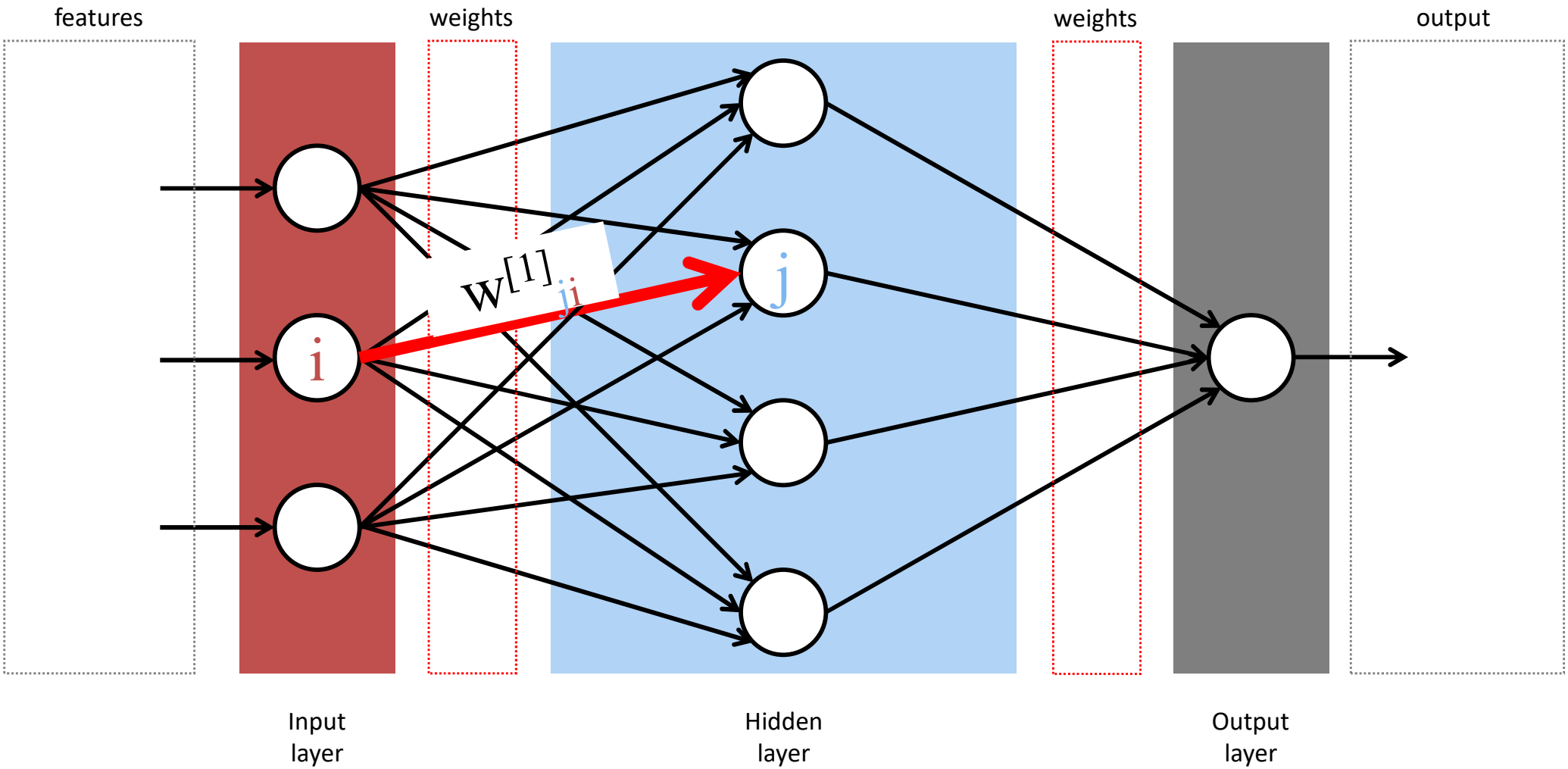


In Multinomial Logistic Regression output is a VECTOR y = s(z)= s(w_i * x_i + b)
[s - softmax]

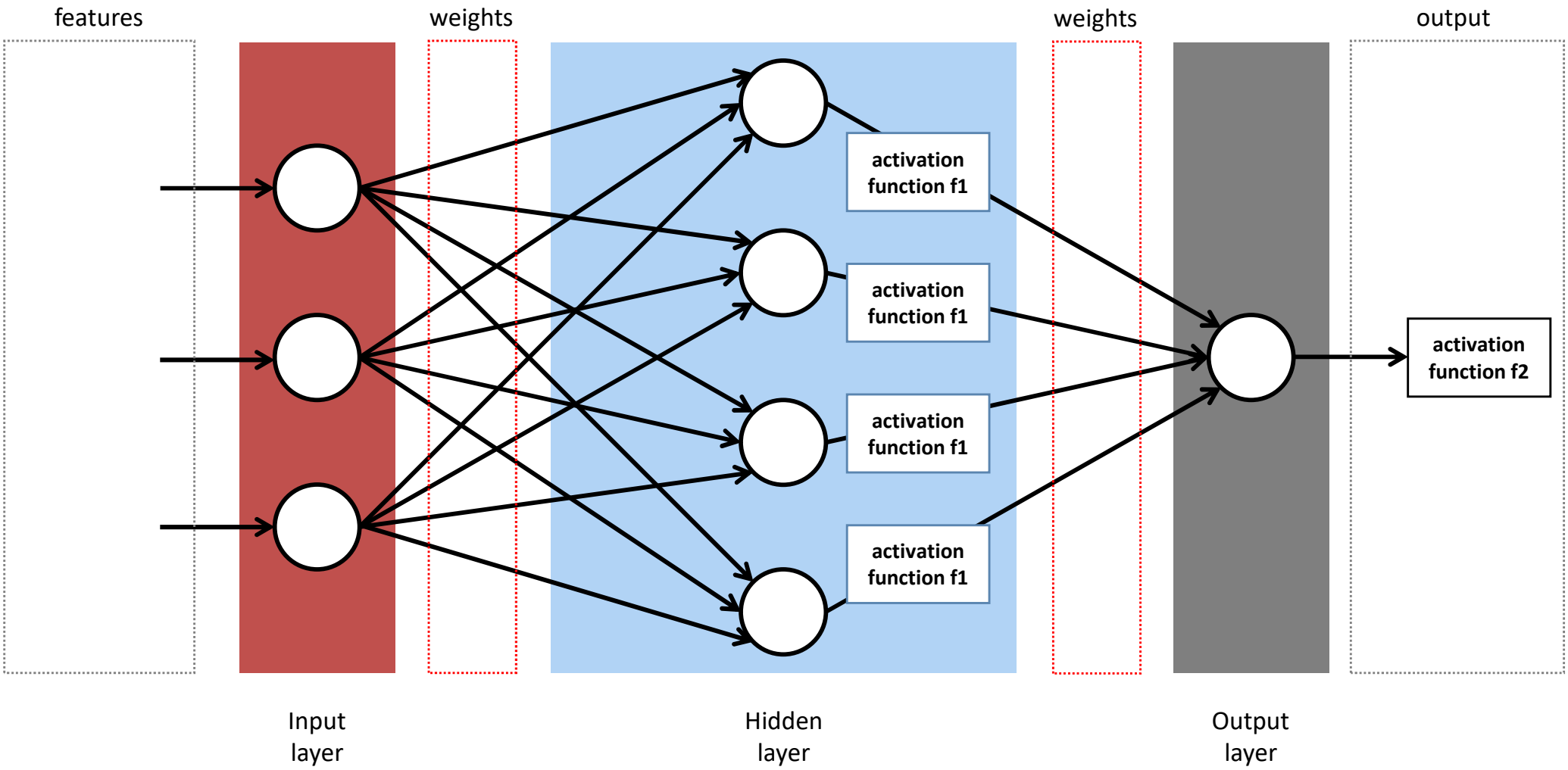
2 Layer Network



2 Layer Network

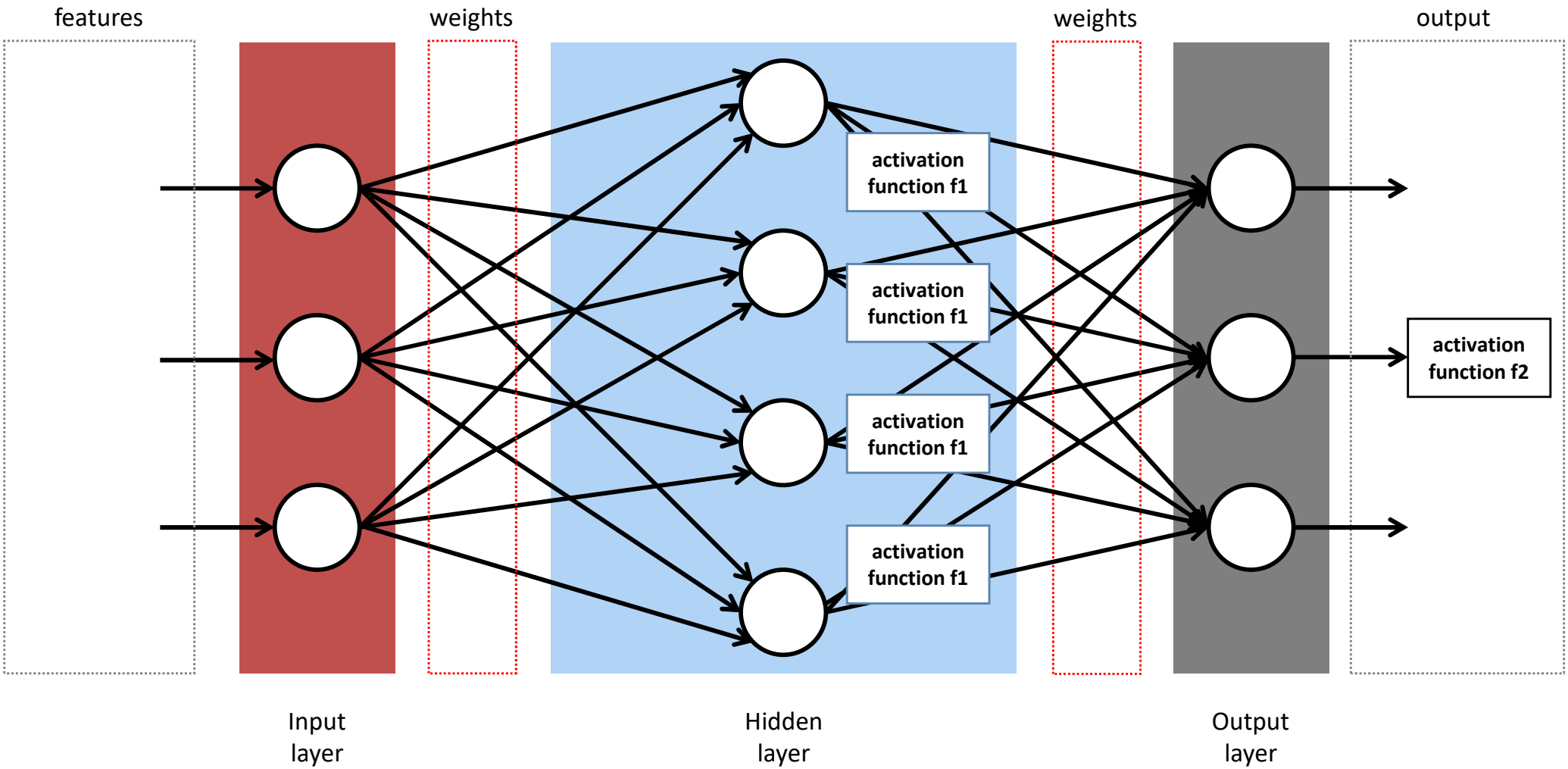


2 Layer Network



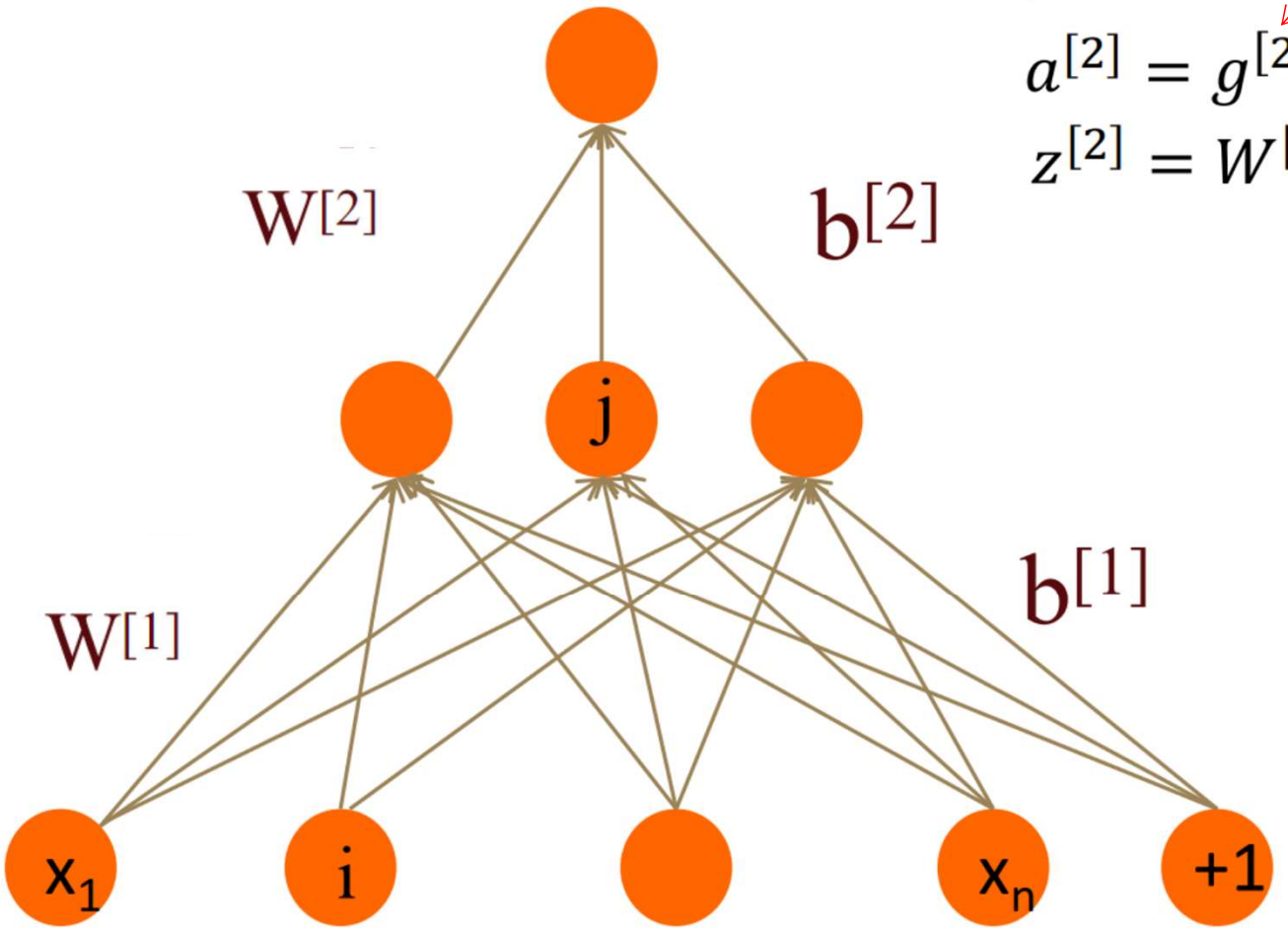
Activation function f1: sigmoid, tanh, ReLU, etc. | Activation function f2: sigmoid

2 Layer Network



Activation function f1: sigmoid, tanh, ReLU, etc. | Activation function f2: softmax

Multilayer Neural Net: Notation



$$y = a^{[2]}$$ sigmoid or softmax

$$a^{[2]} = g^{[2]}(z^{[2]})$$

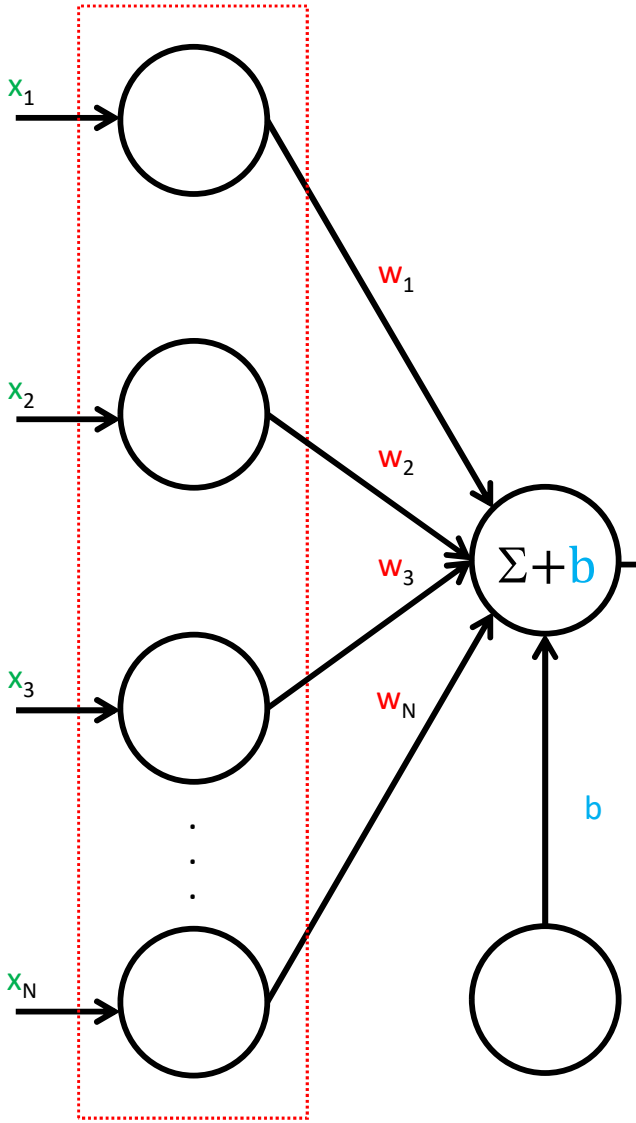
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

ReLU

$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

Multilayer Neural Net: Notation



$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$

z

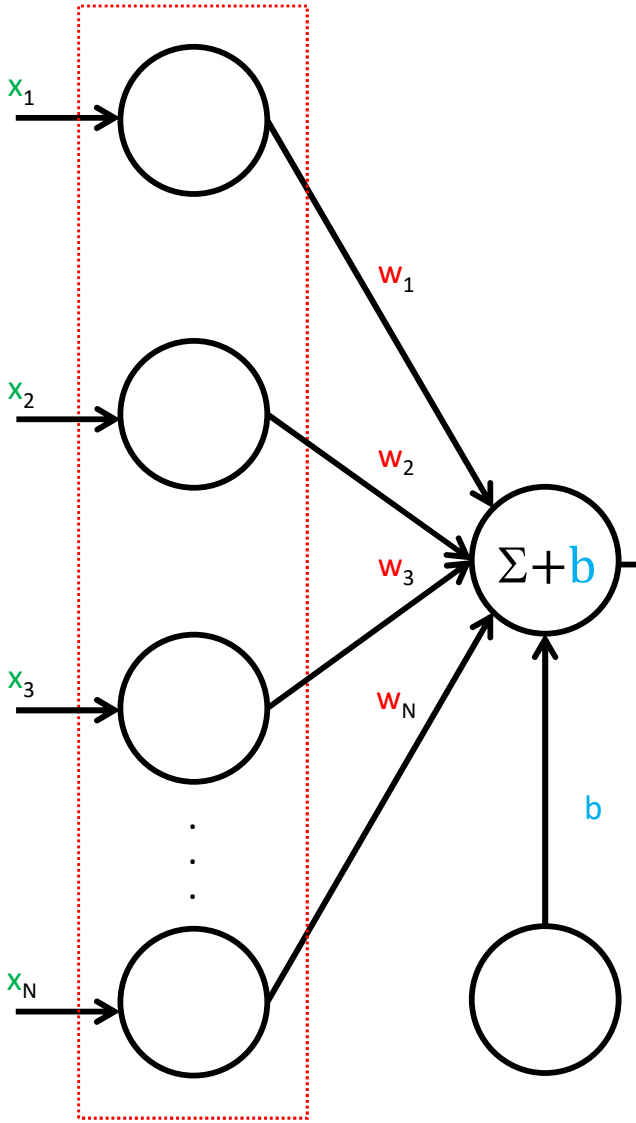
Nonlinear transform
g

a → y

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$
$$\hat{y} = a^{[2]}$$

$z = \Sigma\, w_i * x_i + b$

Multilayer Neural Net: Notation



z

Nonlinear transform

g

a

y

$z = \Sigma\, w_i * x_i + b$

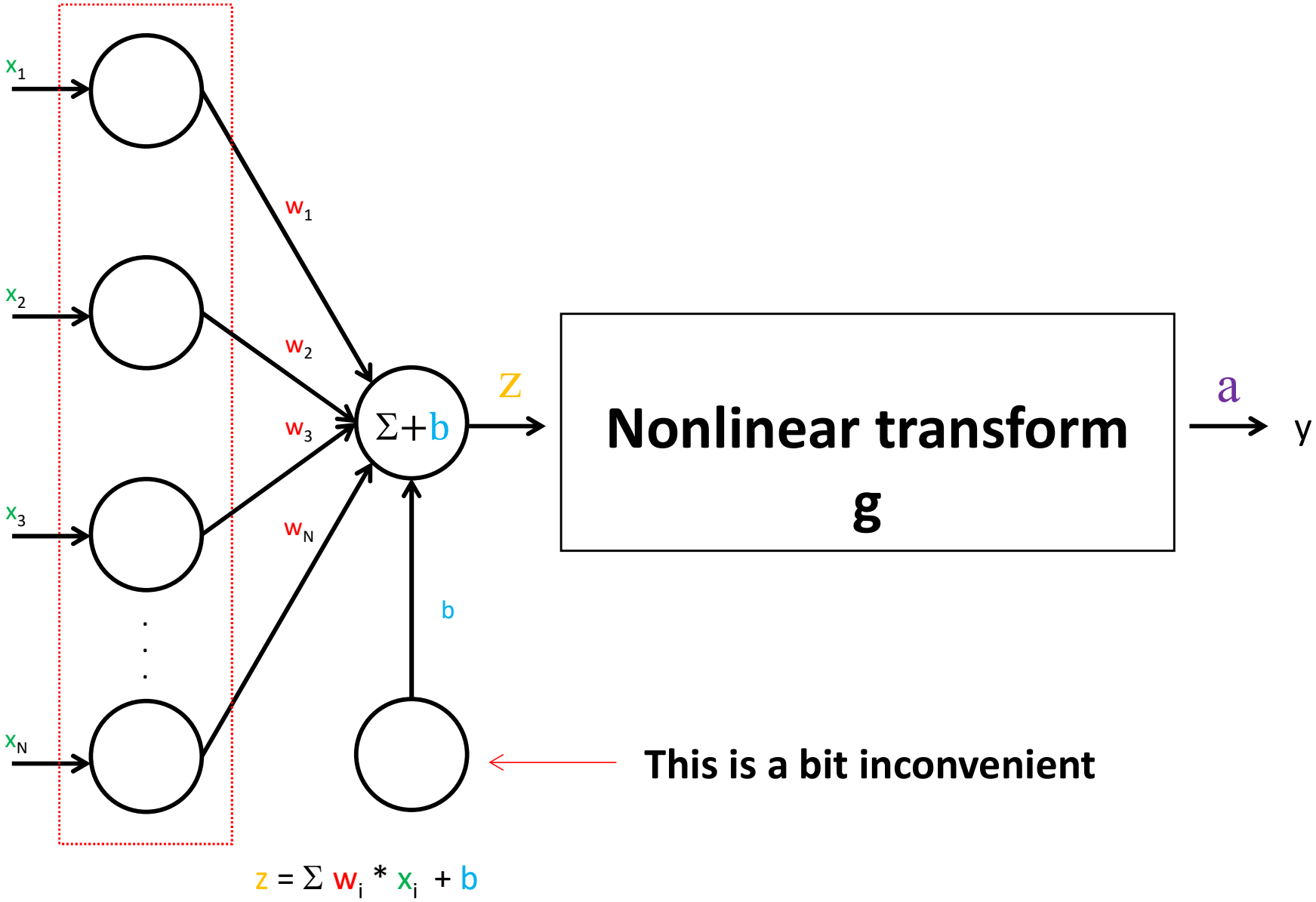
for i **in** 1..n

$$z^{[i]} = W^{[i]}\, a^{[i-1]} + b^{[i]}$$

$$a^{[i]} = g^{[i]}(z^{[i]})$$

$$\hat{y} = a^{[n]}$$

Replacing the Bias Unit



$z = \Sigma\, w_i * x_i + b$

Replacing the Bias Unit

Let's switch to a notation without the bias unit

1. **Add a dummy node $a_0=1$ to each layer**
2. **Its weight w_0 will be the bias**
3. **So input layer $a^{[0]}_0=1$, $a^{[1]}_0=1$, $a^{[2]}_0=1$, …**

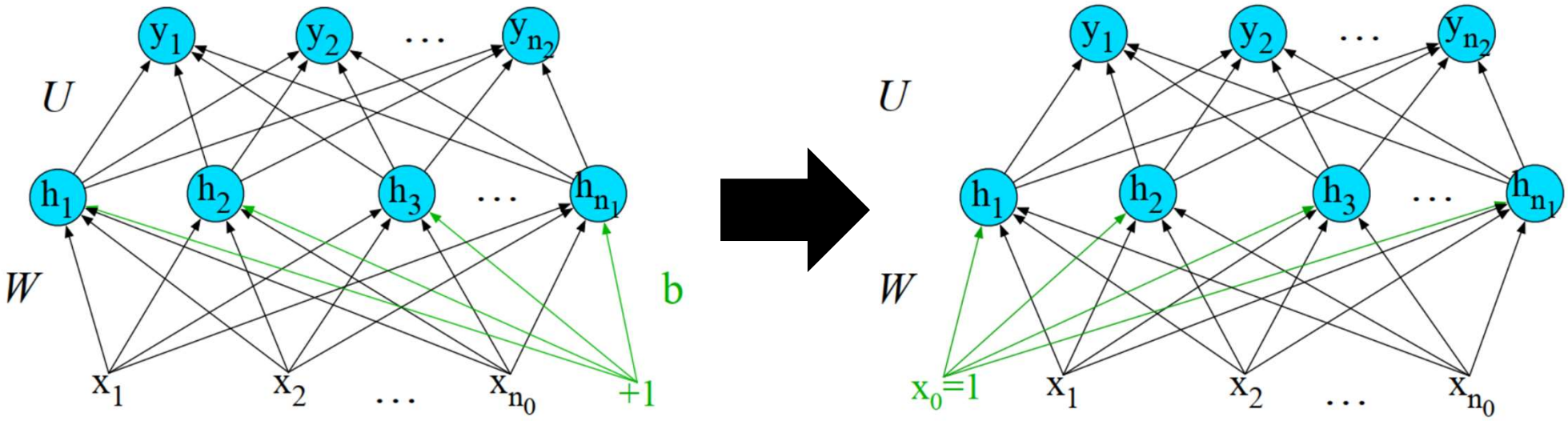
and

$$x = x_1, x_2, \ldots, x_{n0} \qquad x = x_0, x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx + b) \qquad h = \sigma(Wx)$$

$$h_j = \sigma\left(\sum_{i=1}^{n_0} W_{ji}x_i + b_j\right) \qquad \sigma\left(\sum_{i=0}^{n_0} W_{ji}x_i\right)$$

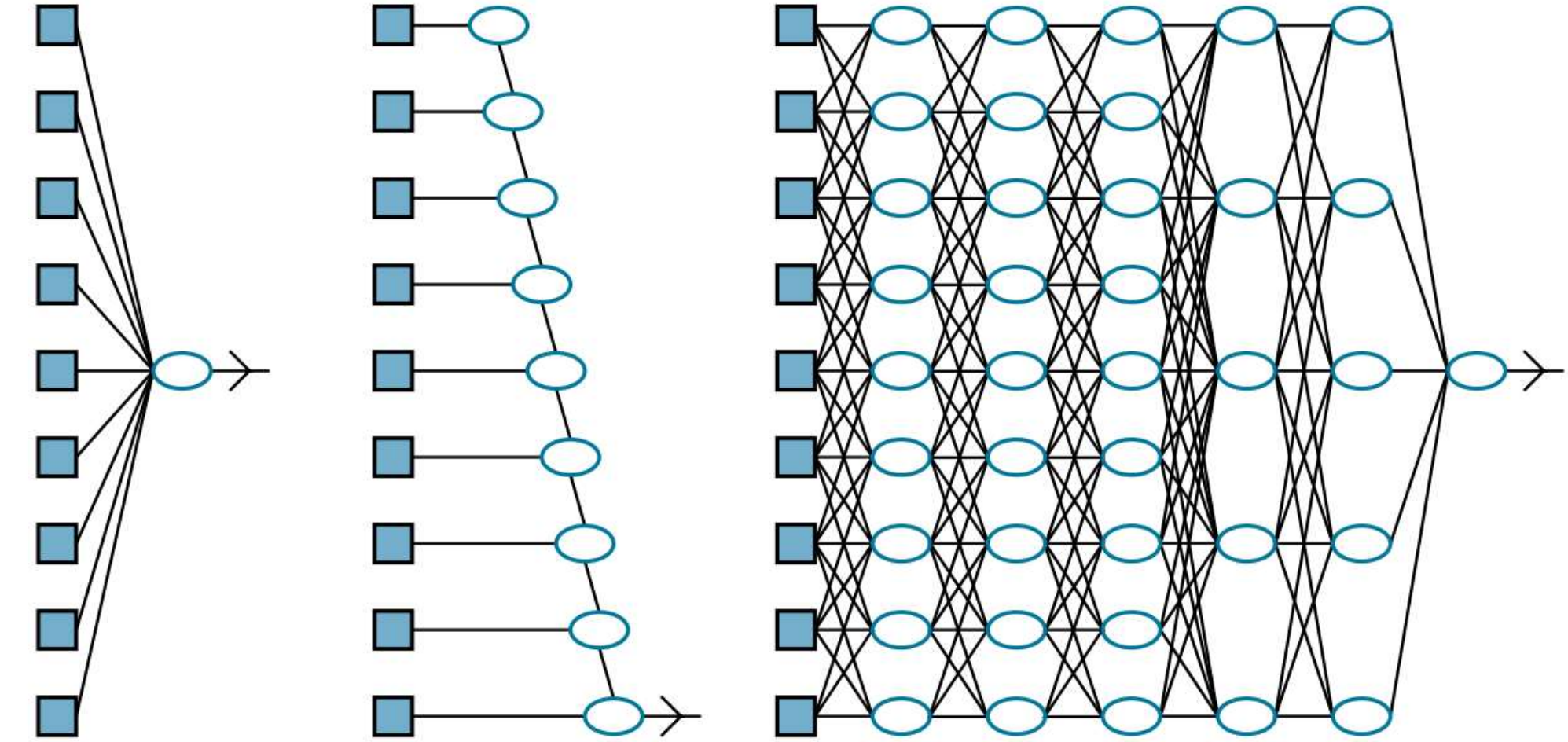
Replacing the Bias Unit



Deep Learning

Deep learning is a broad family of techniques for machine learning (also a sub-field of ML) in which hypotheses take the form of complex algebraic circuits with tunable connections. The word "deep" refers to the fact that the circuits are typically organized into many layers, which means that computation paths from inputs to outputs have many steps.

Shallow vs. Deep Models



Shallow Model

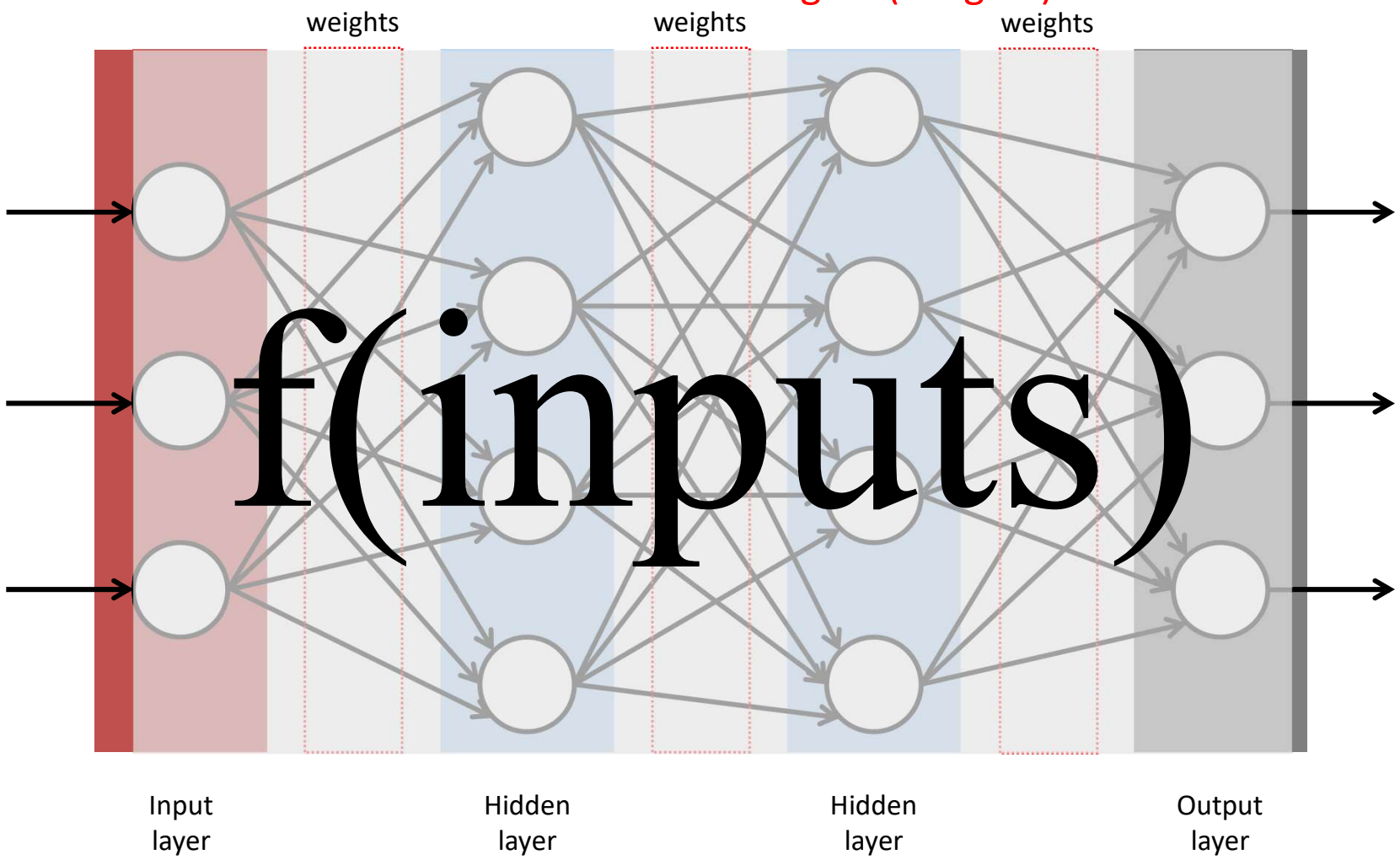
Shallow Model

Deep Model

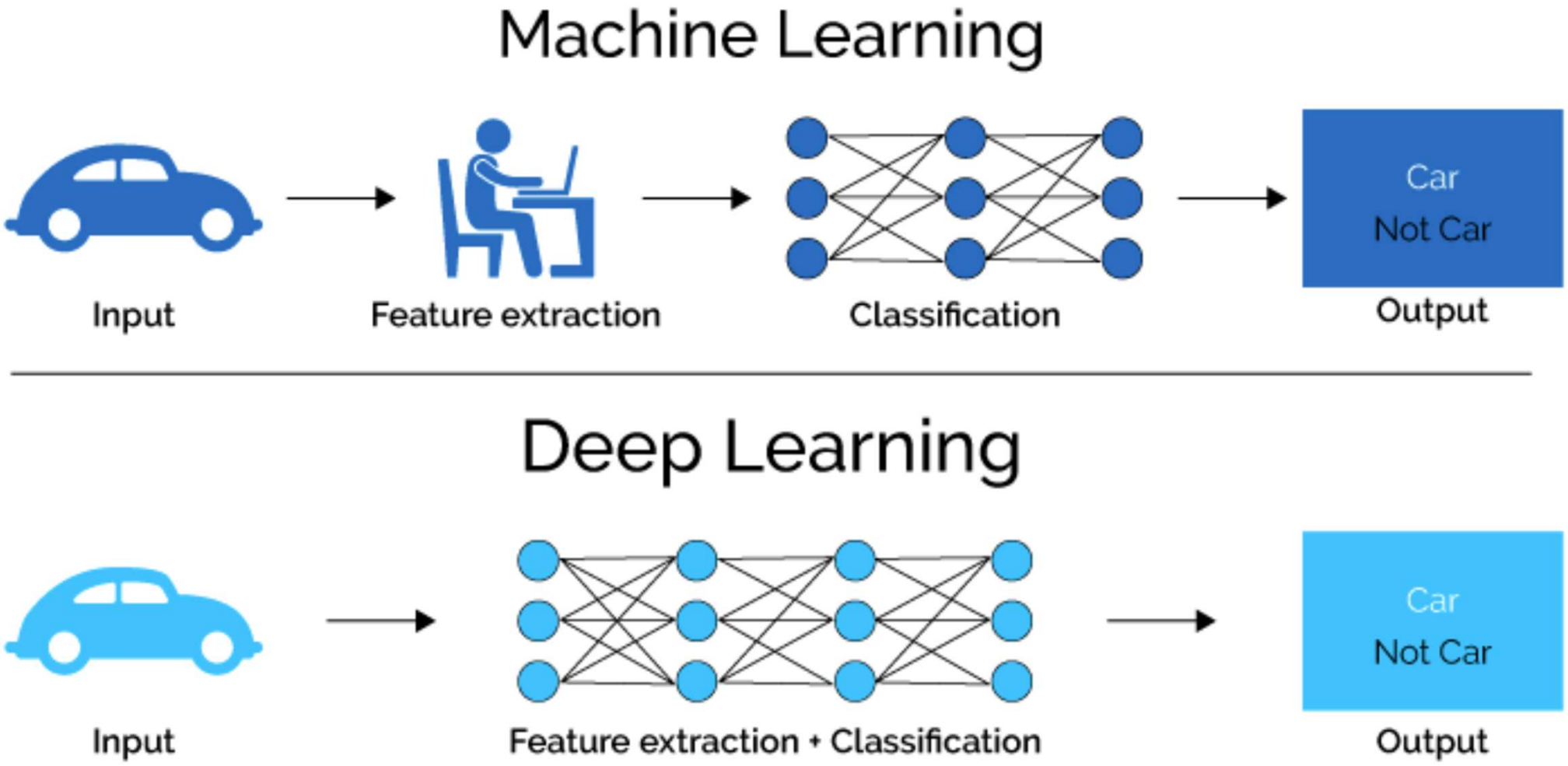
Longer computation path

ANN as a Complex Function

In ANNs **hypotheses take form of complex algebraic circuits** with tunable connection strengths (weights).

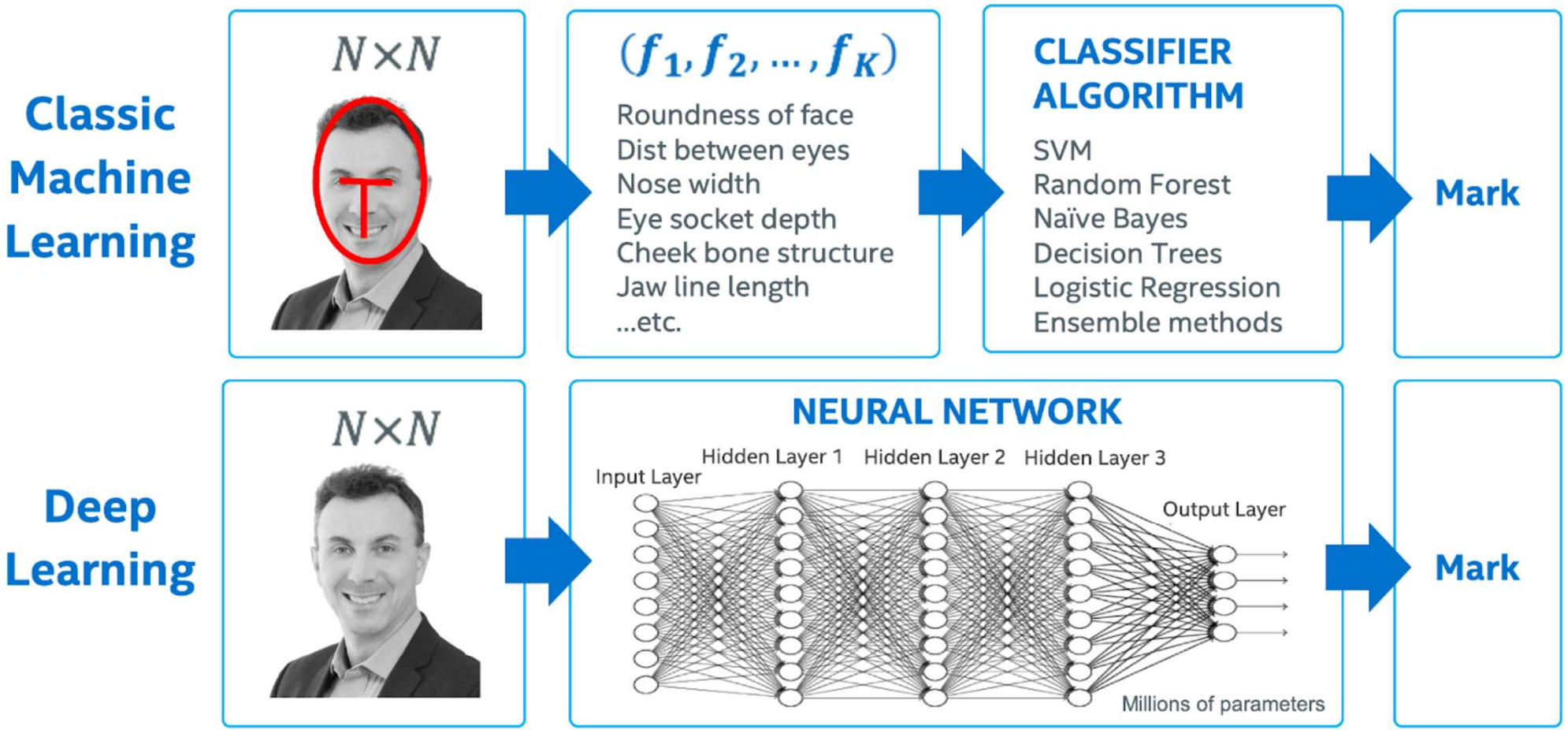


Machine Learning vs. Deep Learning



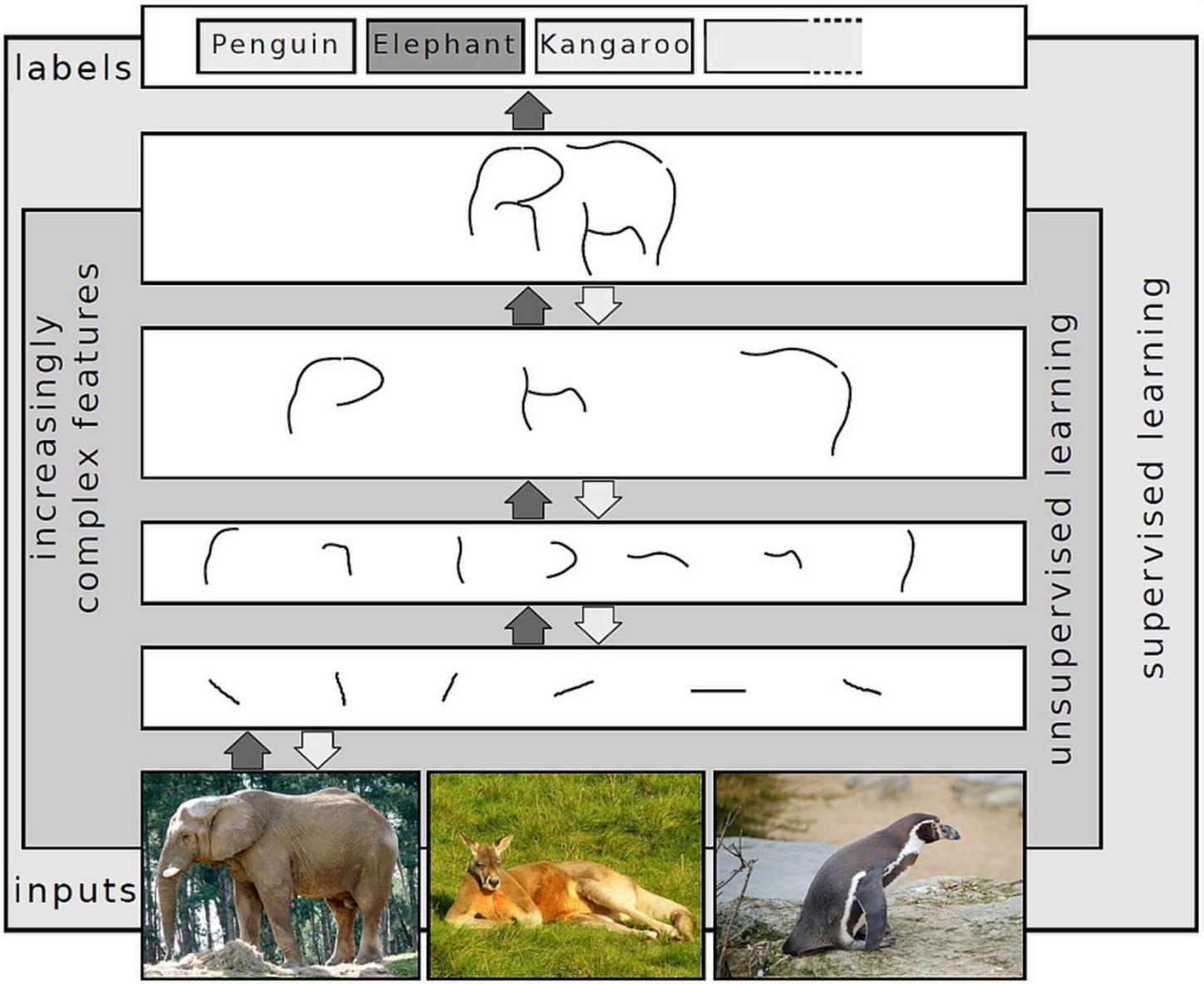
Source: https://www.quora.com/What-is-the-difference-between-deep-learning-and-usual-machine-learning

Machine Learning vs. Deep Learning



Source: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/difference-between-ai-machine-learning-deep-learning.html

Deep Learning: Feature Extraction



Source: https://en.wikipedia.org/wiki/Deep_learning

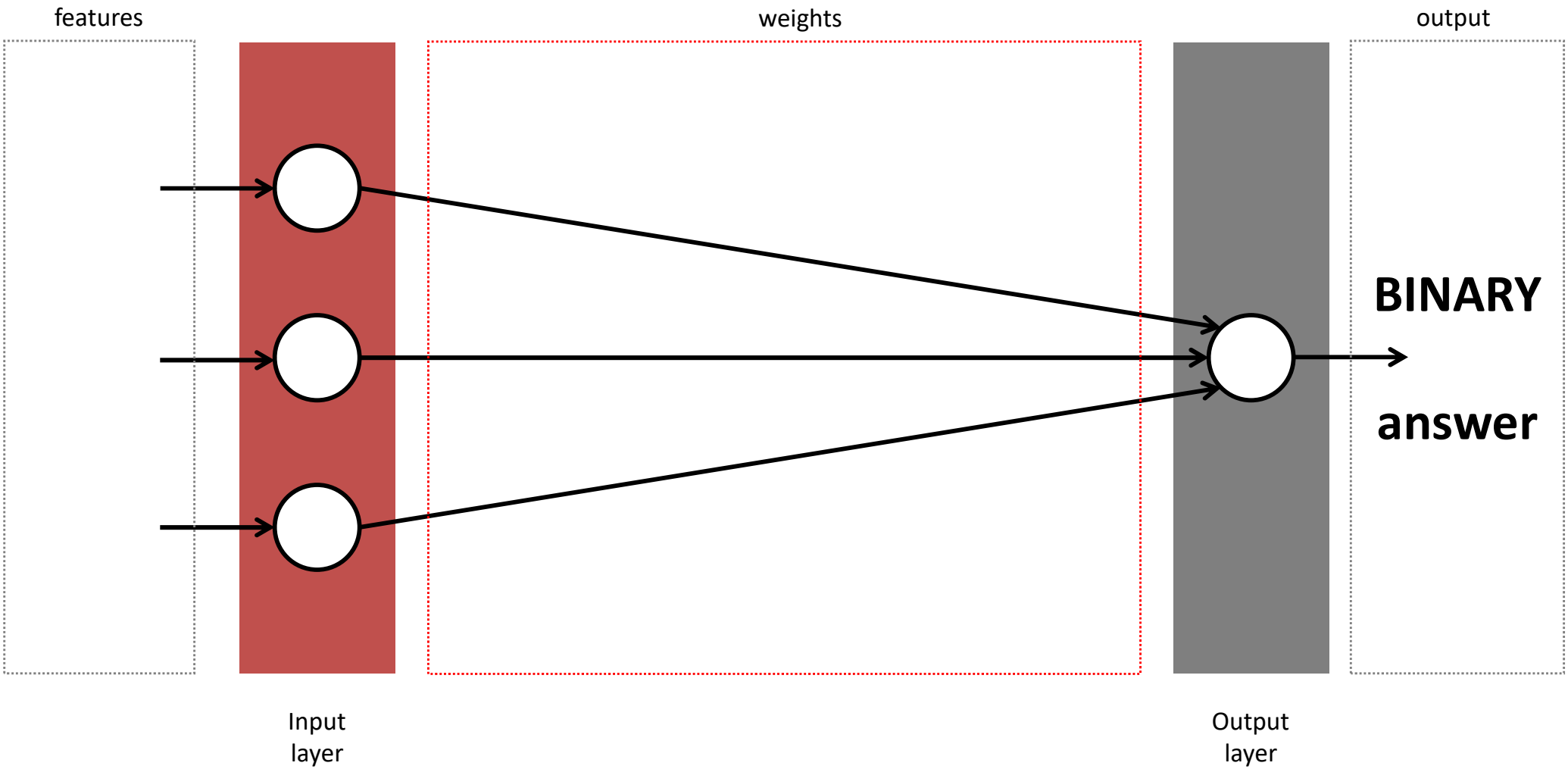
Neural Networks in NLP

Let's consider the NLP modeling we explored so far:

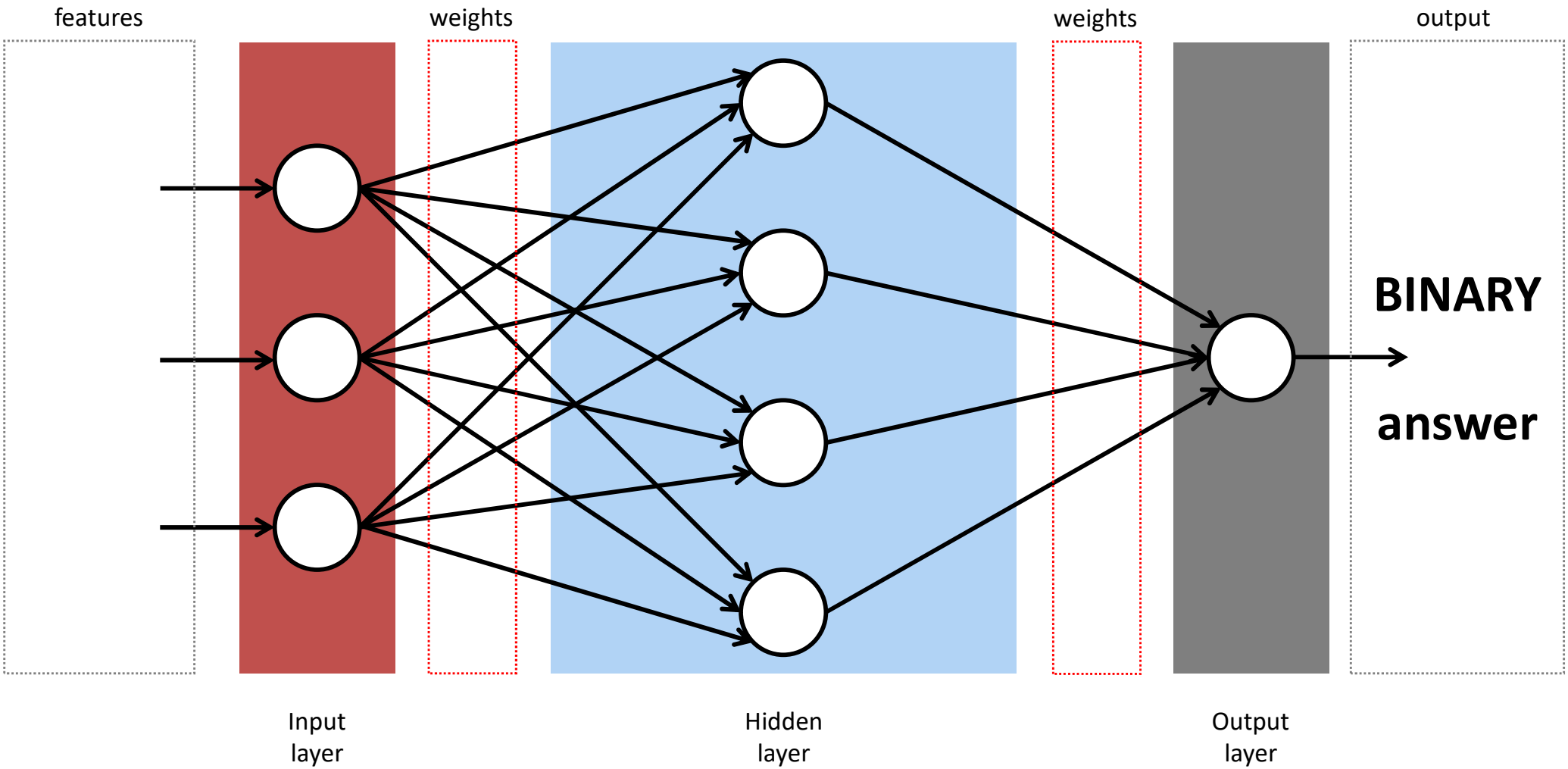
- **Classification**
- **Language Modeling**

Can we apply Neural Networks?

Logistic Regression Sentiment Analysis



Logistic Regression Sentiment Analysis

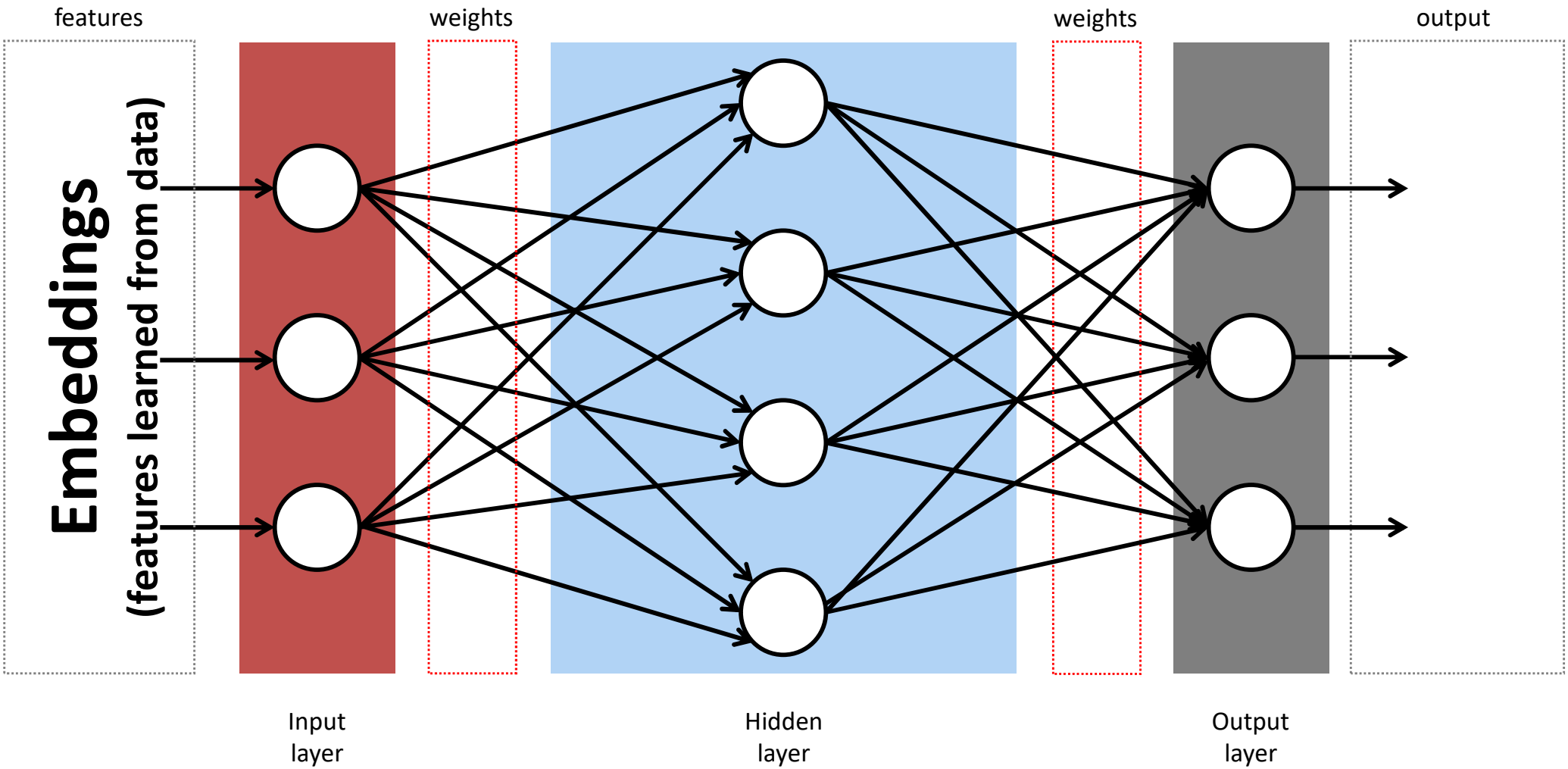


Complex Feature Vector Relationships

Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if “no” } \in \text{ doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if “!” } \in \text{ doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

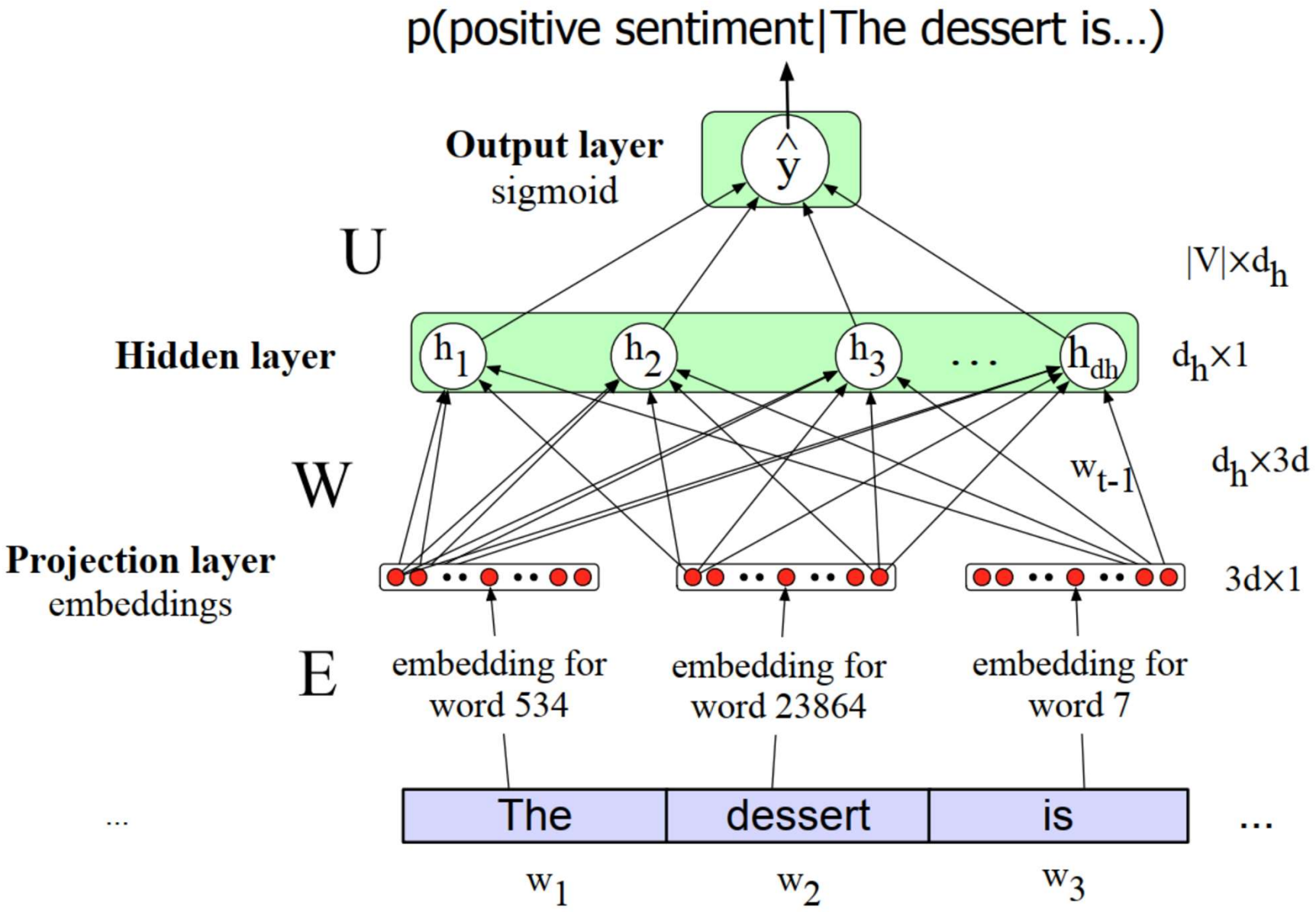
Adding hidden layers can help capture non-linear relationships between features!

Embeddings as Input Features

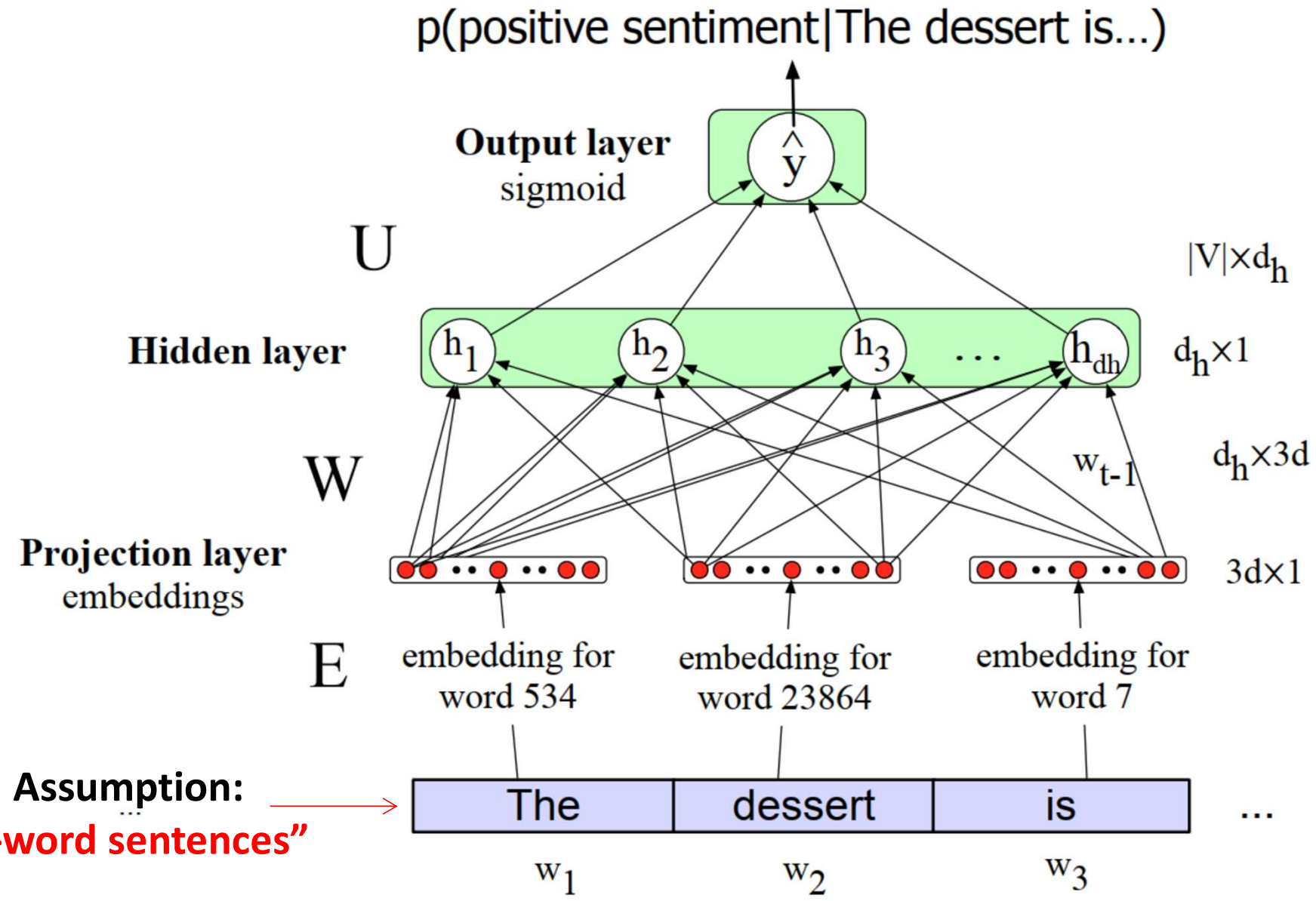


Multiclass output: add more output layer nodes + use softmax (instead of sigmoid)

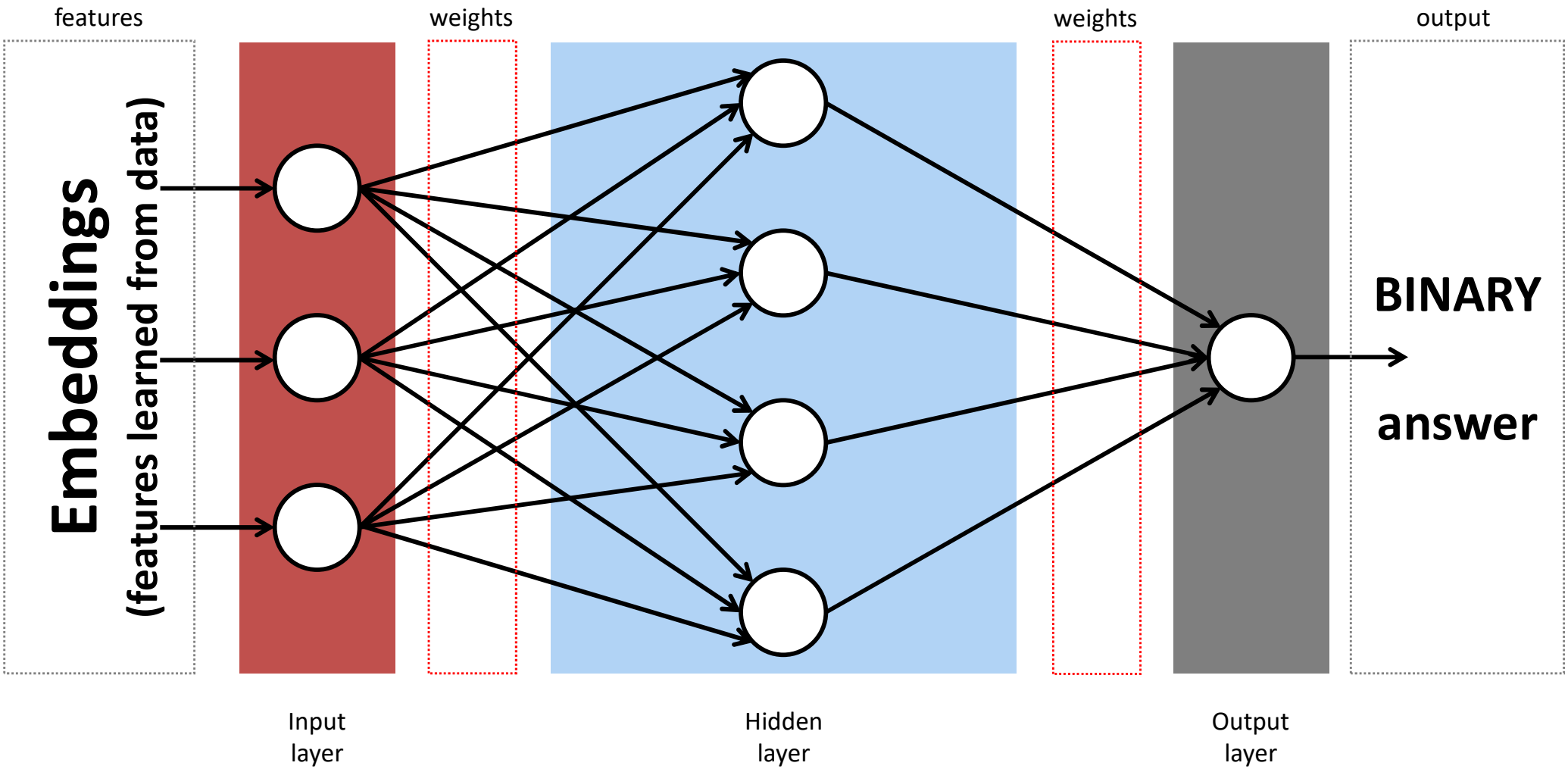
Embeddings as Input Features



Embeddings as Input Features



Embeddings as Input Features



Texts in Different Sizes: Ideas

Some simple solutions:

1. Make the input the length of the longest sample
   - if shorter then pad with zero embeddings
   - truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - take the mean of all the word embeddings
   - take the element-wise max of all the word embeddings
     - for each dimension, pick the max value from all words

Language Models Revisited

Language Modeling: Calculating the probability of the next word in a sequence given some history.

- N-gram based language models
- other: neural network-based?

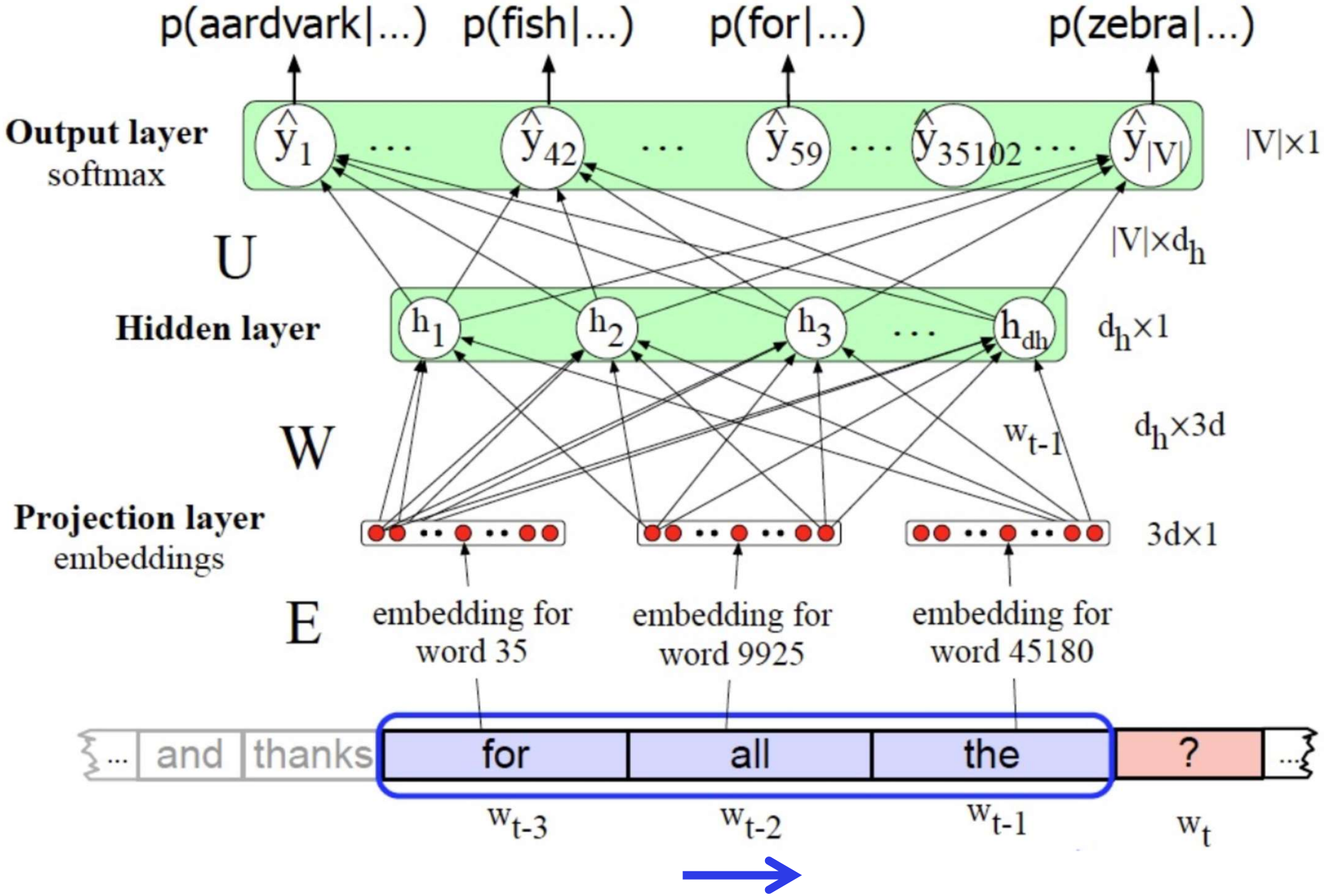
Task: predict next word w_t
given prior words w_{t-1}, w_{t-2}, w_{t-3}, …

Problem: Now we're dealing with sequences of arbitrary length.

Solution: Sliding windows (of fixed length)

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Language Model



Neural LM Better Than N-Gram LM

Training data:

We've seen: I have to make sure that the cat gets fed.

Never seen: dog gets fed

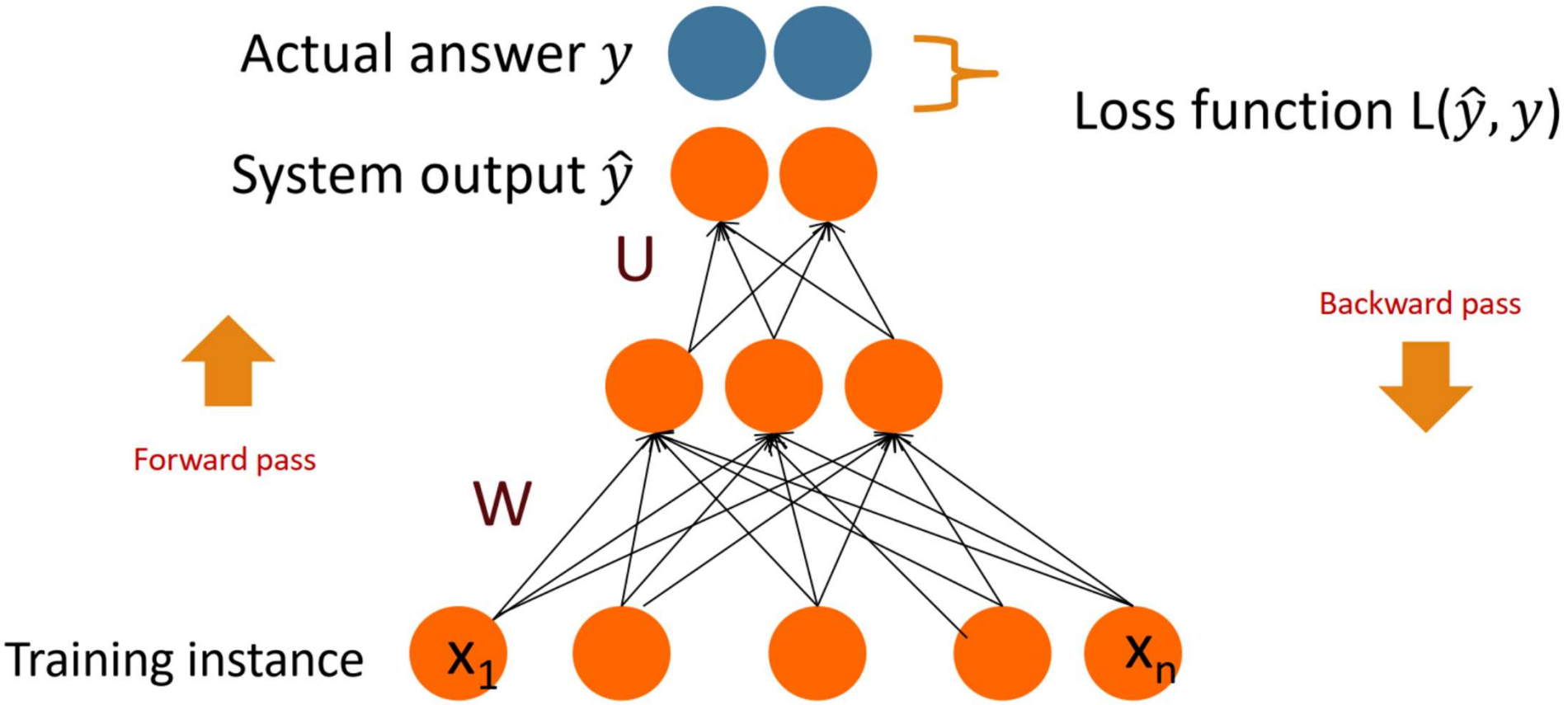
Test data:

I forgot to make sure that the dog gets ___

N-gram LM can't predict "fed"!

Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict "fed" after dog

Training Neural Networks



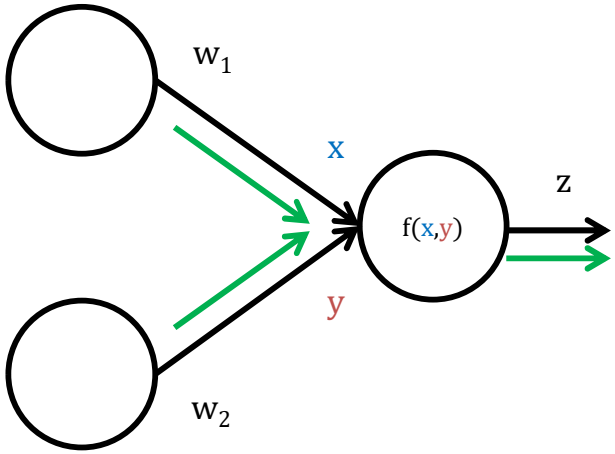
Training Neural Networks: Intuition

For every training tuple (x ,y) = (feature vector, label)

- **Run forward computation to find estimate ŷ**
- **Run backward computation to update weights:**
  - **For every output node**
    - **Compute loss L between true y and the estimated ŷ**
    - **For every weight w from hidden layer to the output layer**
    - **Update the weight**
  - **For every hidden node**
    - **Assess how much blame it deserves for the current answer**
    - **For every weight w from input layer to the hidden layer**
      - **Update the weight**

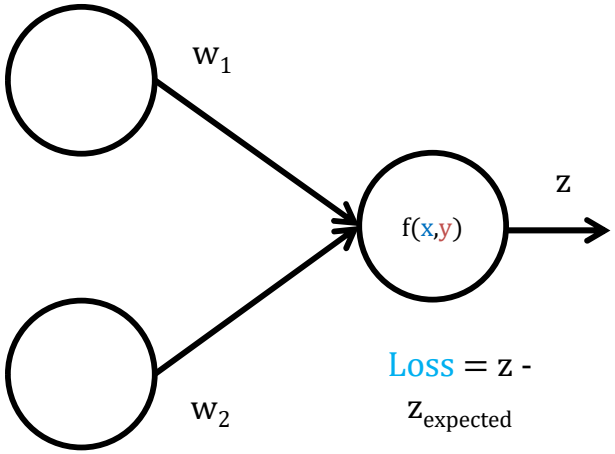
Back-propagation

Feed forward



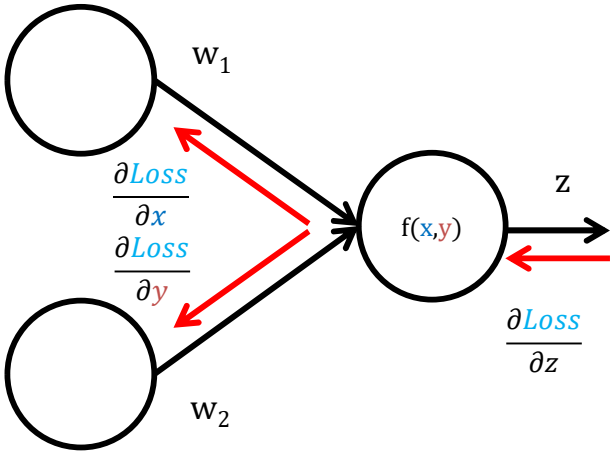
Feed a **labeled sample** through the network

Evaluate Loss



How "incorrect" is the result compare to the label?

Back-propagation



Update weights (use **Gradient Descent**)

Cross-Entropy Loss: Calculation

Goal: **maximize** probability of the correct label P(*y*|x)

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^y(1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y}+(1-y)\log(1-\hat{y})\end{aligned}$$

Goal: **minimize** error / cross-entropy loss

$$L_{\text{CE}}(\hat{y},y) = -\log p(y|x) = -[y\log\hat{y}+(1-y)\log(1-\hat{y})]$$

which yields:

$$L_{\text{CE}}(\hat{y},y) = -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))]$$

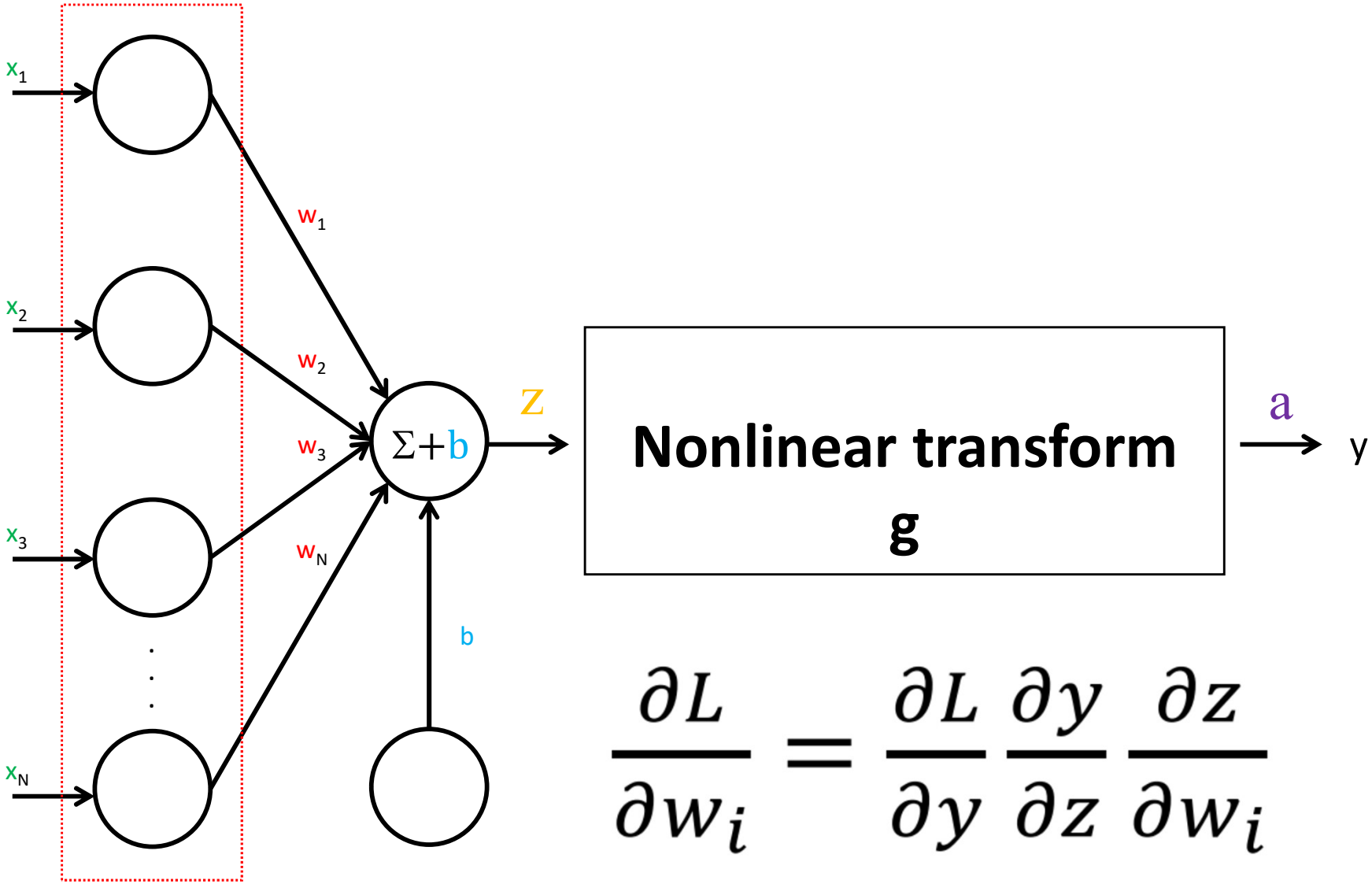
Gradients and Learning Rate

- The value of the gradient (slope in our example) $\frac{d}{dw} L(f(x;w), y)$ weighted by a **learning rate** η

- Higher learning rate means move **w** faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)$$

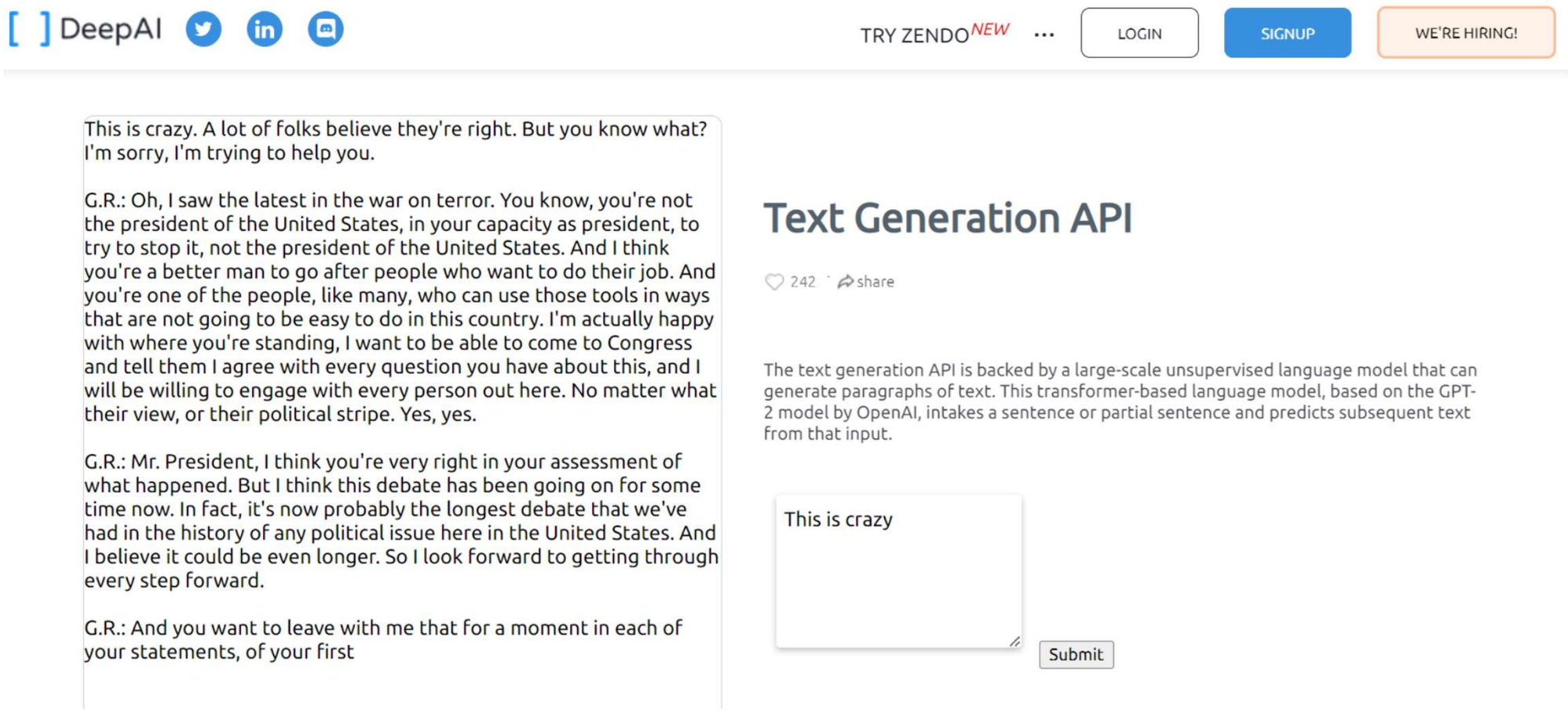
NN Node: Derivative of the Loss



$$z = \Sigma\, w_i * x_i + b$$

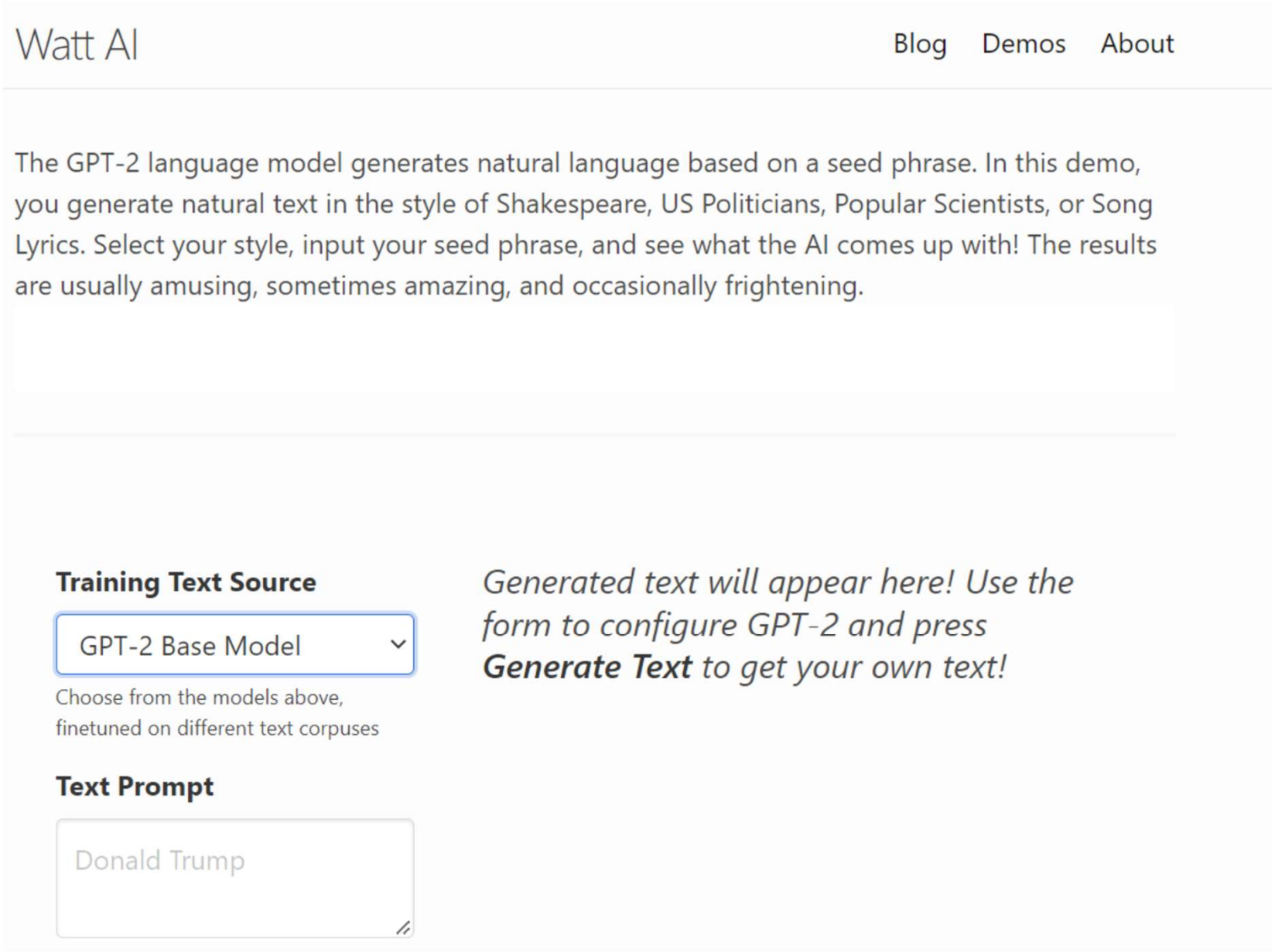
$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}$$

Language Models and Transformers



Source: https://deepai.org/machine-learning-model/text-generator

Language Models and Transformers



Source: https://watt-ai.github.io/demos/gpt2