

# **Deep Learning for Natural Language Processing**

## **Lecture 2: Deep Neural Networks and Word Embedding**

Quan Thanh Tho

Faculty of Computer Science and Engineering  
Ho Chi Minh City University of Technology

# Acknowledgement

- Some slides are from Coursera course of Prof. Andrew Ng.

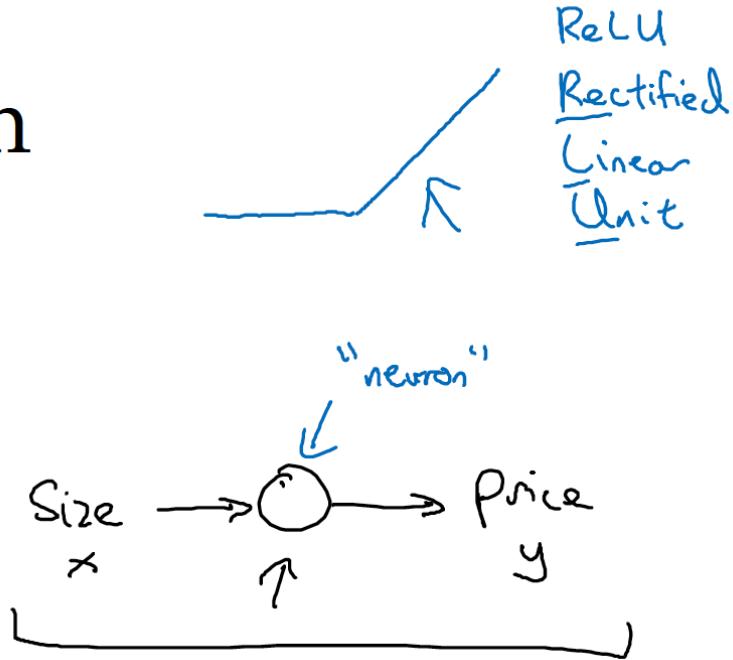
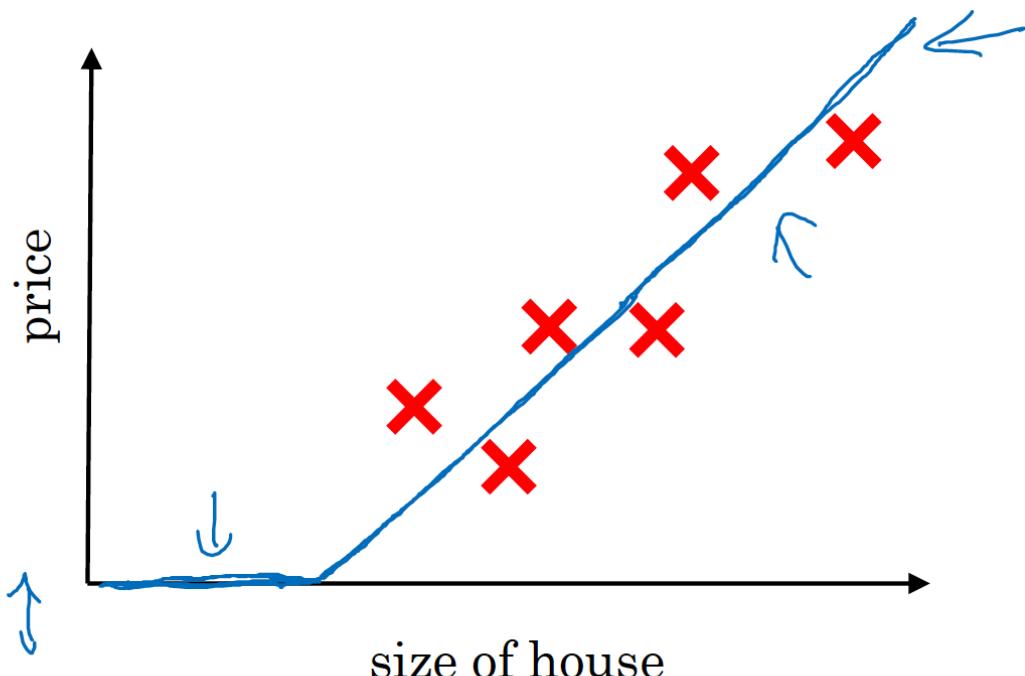
# Agenda

- Intuition of Neural Network for Classification Task
- Neural Network and Deep NN Text Classification
- Specific DNN Architecture: CNN and AutoEncoder
- From AutoEncoder to Word Embedding
- Doc2vec: combining tf.idf and WE

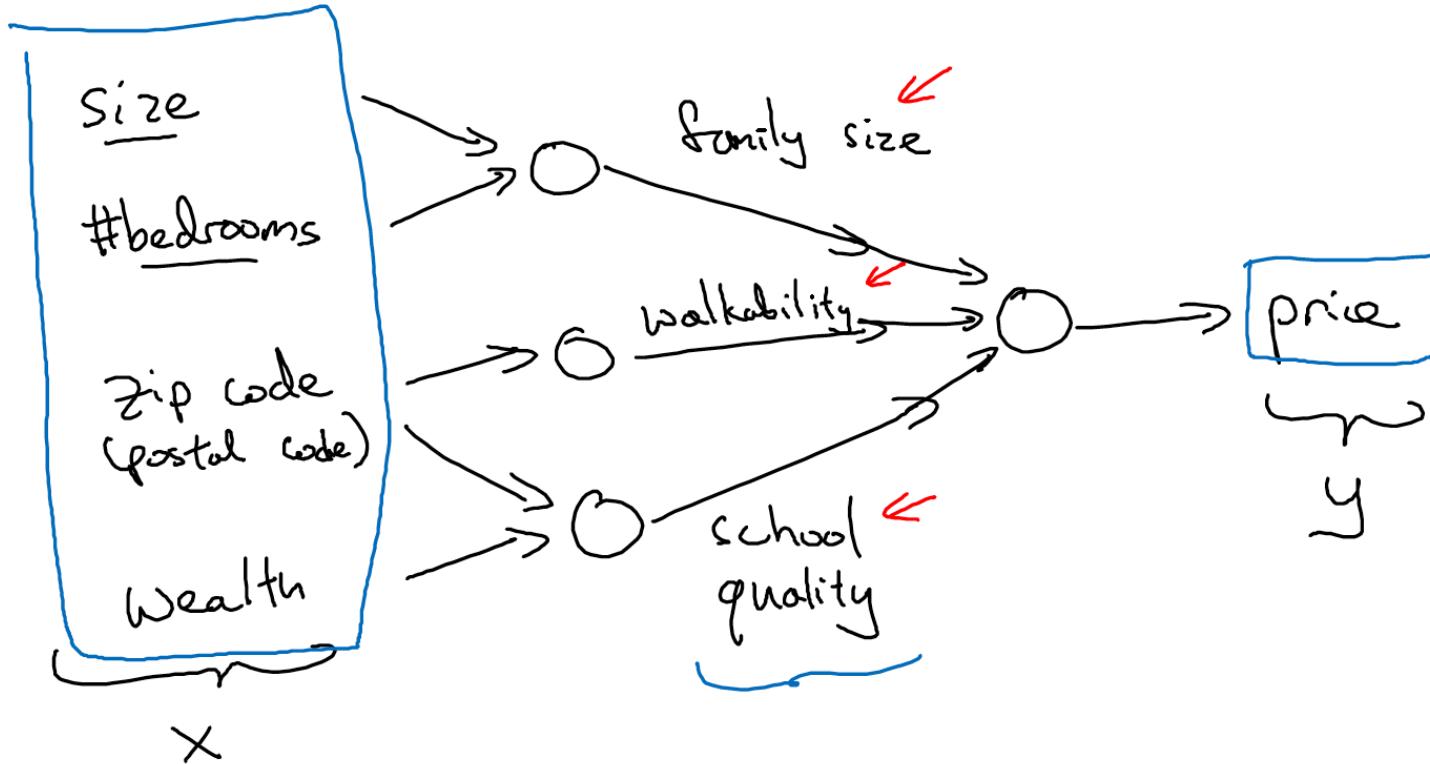
# **Classical Machine Learning Techniques for NLP**

- Machine Learning Task
- The tf.idf weights
- Neural-based approach

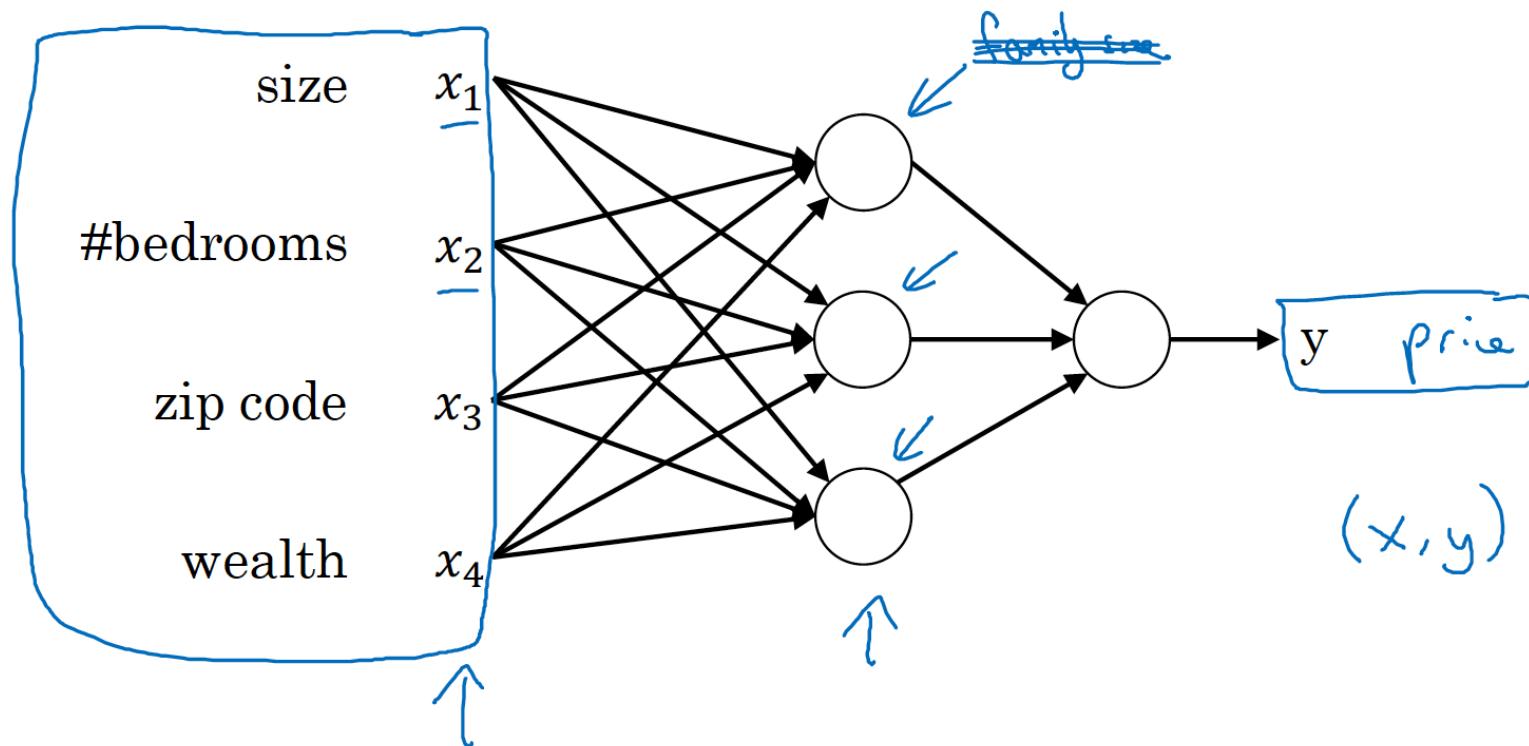
# Housing Price Prediction



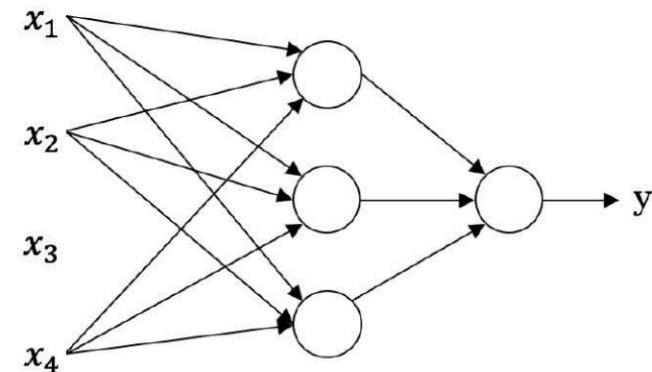
# Housing Price Prediction



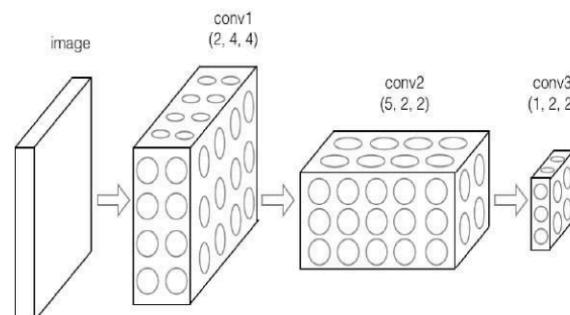
# Housing Price Prediction



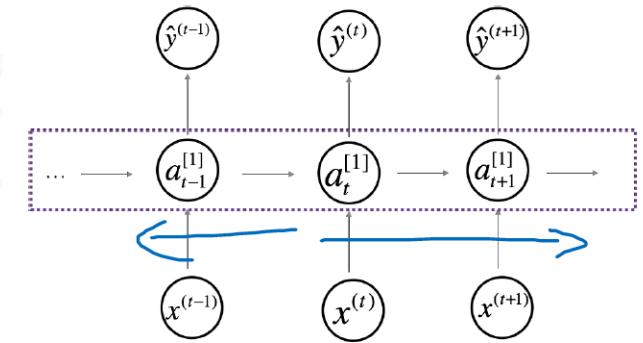
# Neural Network examples



Standard NN

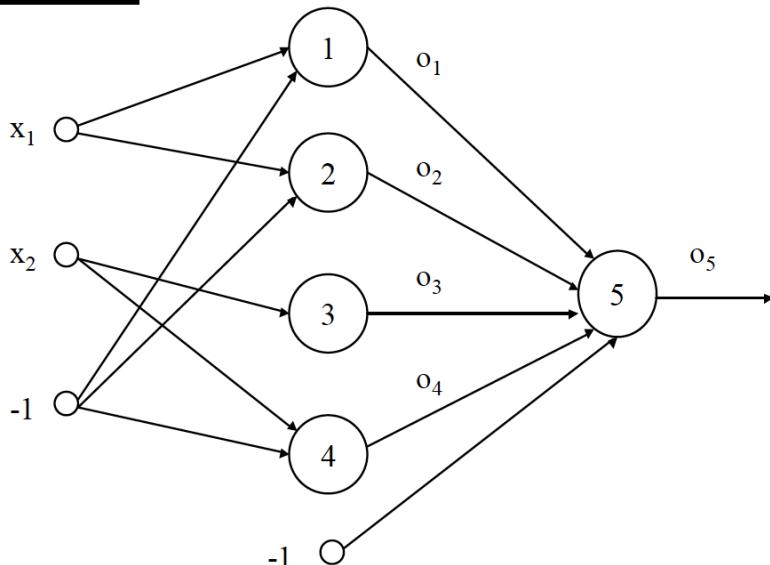


Convolutional NN



Recurrent NN

### Example 1:



The connection weight vectors are:

$$\mathbf{w}_1 = [1 \quad 0 \quad -1]^T \quad \mathbf{w}_2 = [-1 \quad 0 \quad -2]^T$$

$$\mathbf{w}_3 = [0 \quad 1 \quad 0]^T \quad \mathbf{w}_4 = [0 \quad -1 \quad -3]^T$$

$$\mathbf{w}_5 = [1 \quad 1 \quad 1 \quad 1 \quad 3.5]^T$$

The activation function is as follow:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases}$$

Based on the connections and the weights, we have the following relationships:

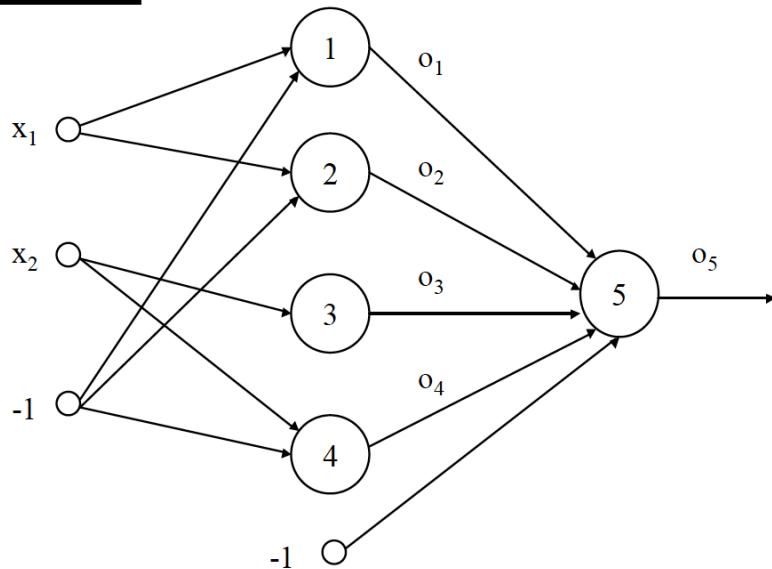
$$v_1 = x_1 - 1$$

$$v_2 = -x_1 + 2$$

$$v_3 = x_2$$

$$v_4 = -x_2 + 3$$

Example 1:



$$o_1 = \text{sgn}(x_1 - 1)$$

$$o_2 = \text{sgn}(-x_1 + 2)$$

$$o_3 = \text{sgn}(x_2)$$

$$o_4 = \text{sgn}(-x_2 + 3)$$

The output vector of the hidden layer is then used as the input of the output layer. The output of the overall network is as follow:

$$o_5 = \text{sgn}(o_1 + o_2 + o_3 + o_4 - 3.5)$$

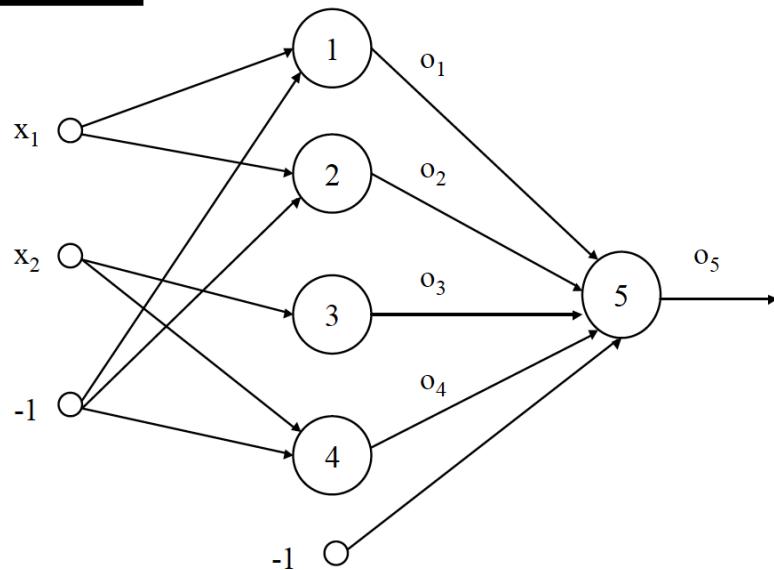
The value of  $o_5$  is 1 if and only if

$$o_1 + o_2 + o_3 + o_4 \geq 3.5$$

This actually requires that:

$$o_1 = o_2 = o_3 = o_4 = 1$$

Example 1:



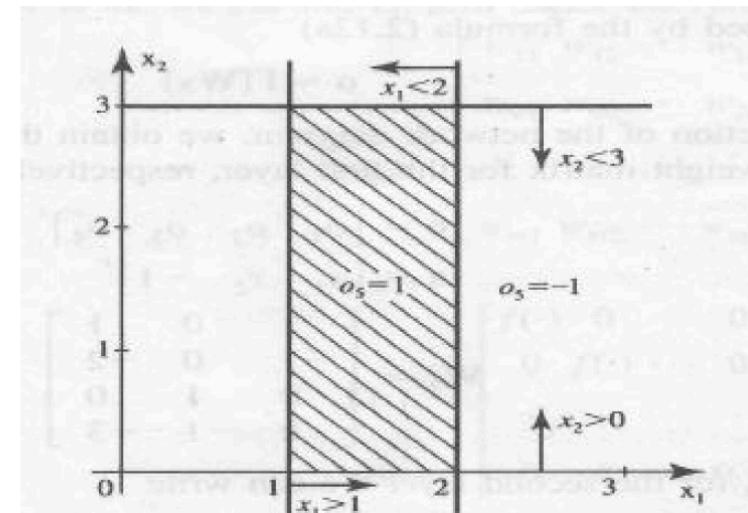
which is equivalent to:

$$x_1 \geq 1$$

$$x_1 \leq 2$$

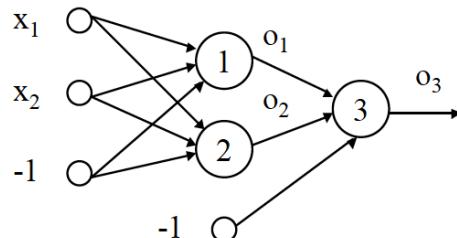
$$x_2 \geq 0$$

$$x_2 \leq 3$$



## Example 2:

Consider the following 2-layer neural network:

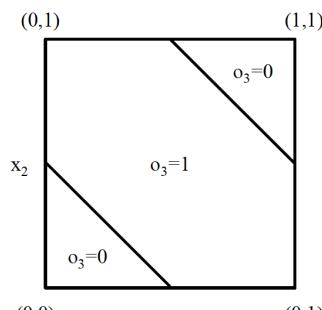


The weight vectors are:

$$\mathbf{w}_1 = [1 \ 1 \ 1.5]^T \quad \mathbf{w}_2 = [1 \ 1 \ 0.5]^T \quad \mathbf{w}_3 = [-2 \ 1 \ 0.5]^T$$

Assume  $x_1$  and  $x_2$  are in the following range:

$$0 \leq x_1 \leq 1$$



Output of the network

If the values of input  $x_1$  and  $x_2$  are either 1 or 0, we can see from the above diagram that:

$$f(1,1) = 1$$

$$f(0,0) = 1$$

$$f(0,1) = 0$$

$$f(1,0) = 0$$

Where  $f$  is the nonlinear mapping realized by the above neural network. Actually,  $f$  realizes the XOR logic operation:

$$1 \oplus 1 = 1$$

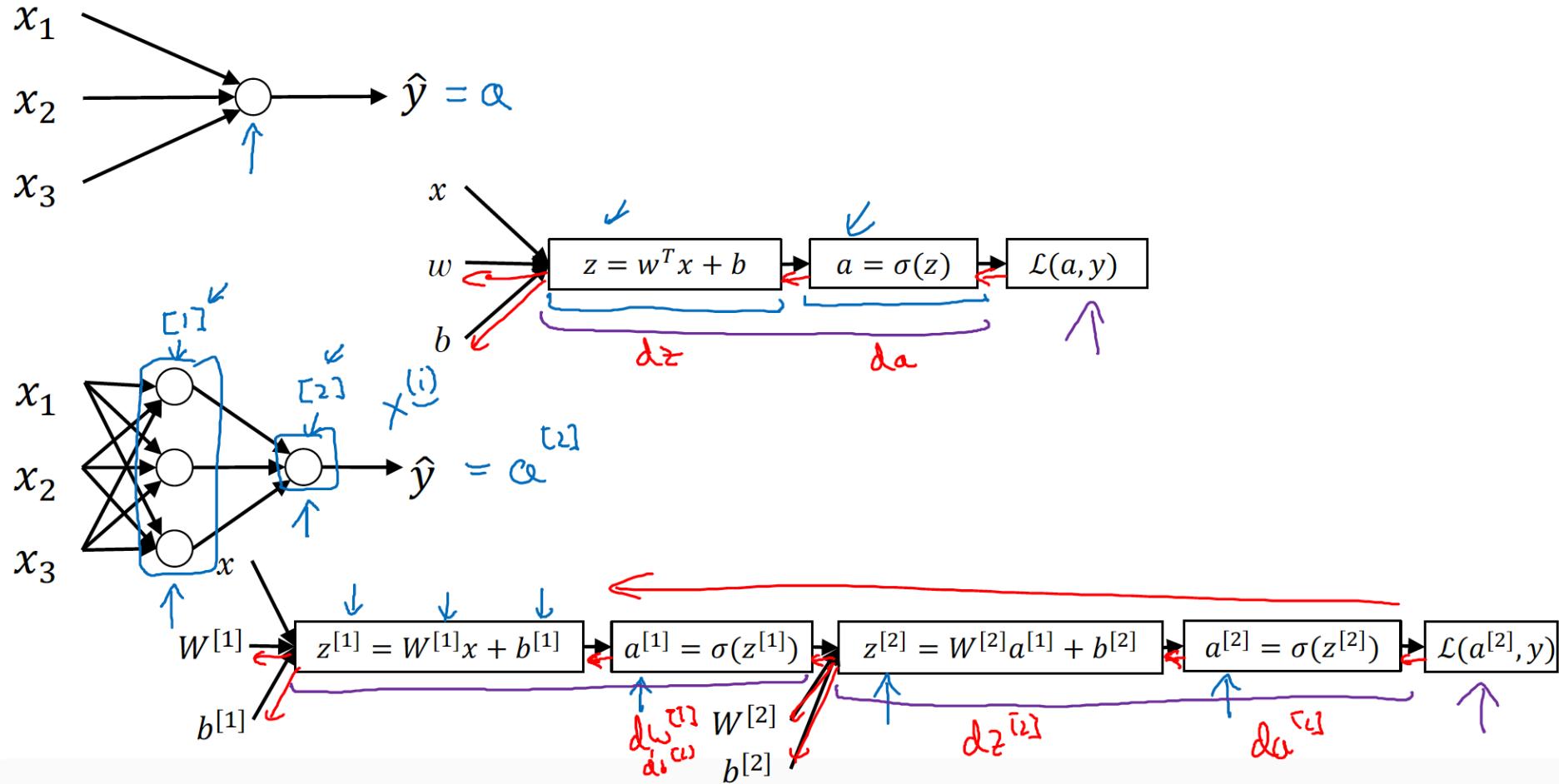
$$0 \oplus 0 = 1$$

$$0 \oplus 1 = 0$$

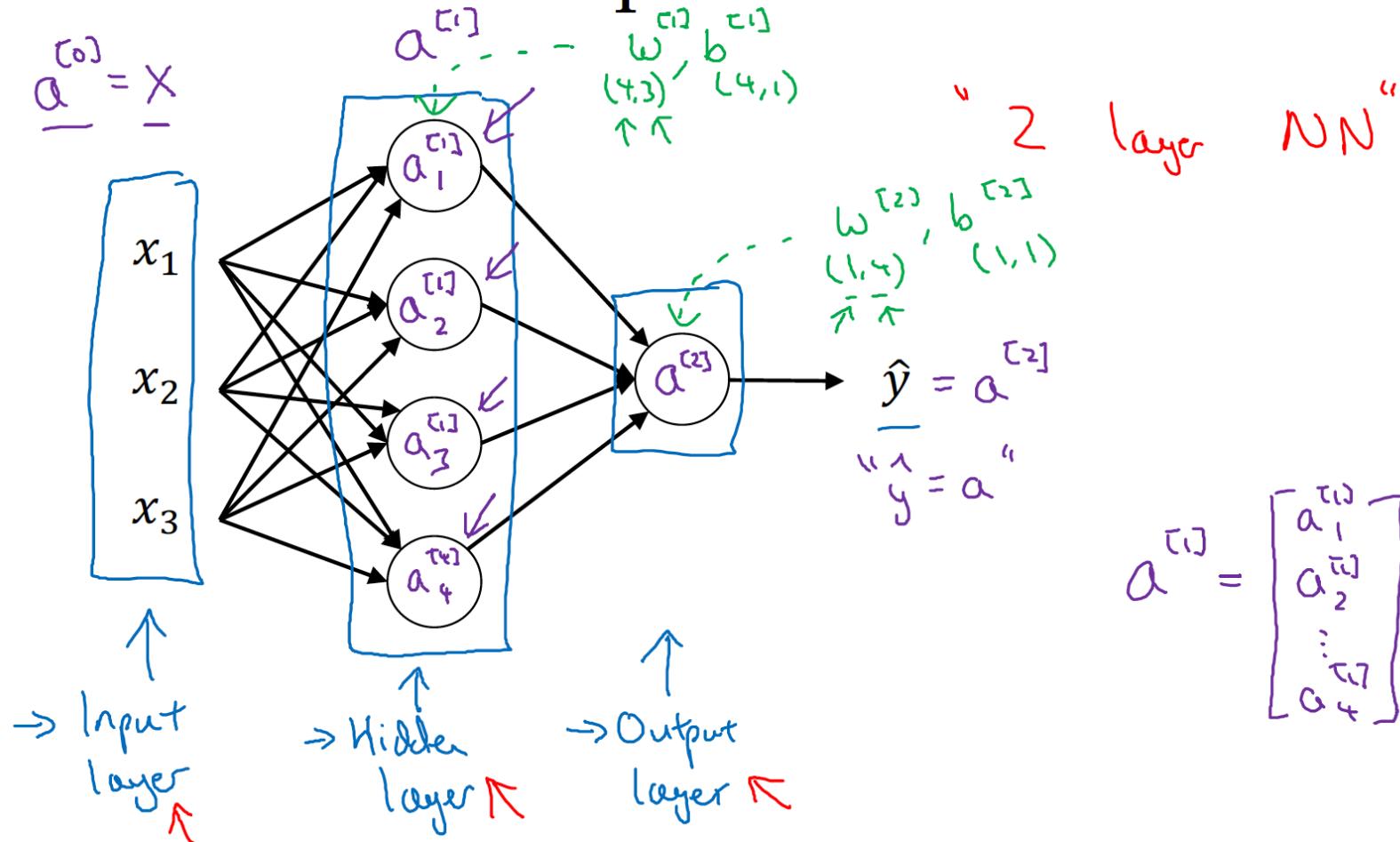
$$1 \oplus 0 = 0$$

Where  $\oplus$  denotes the exclusive OR (XOR) operator.

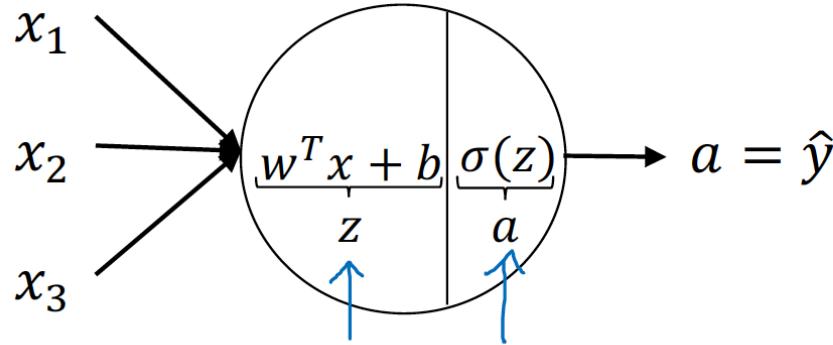
# What is a Neural Network?



# Neural Network Representation

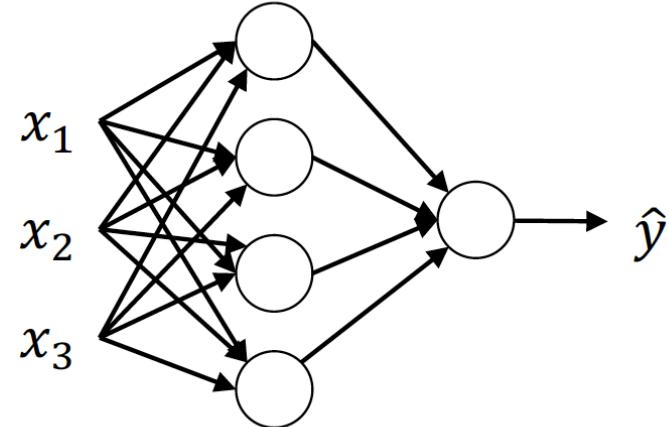


# Neural Network Representation

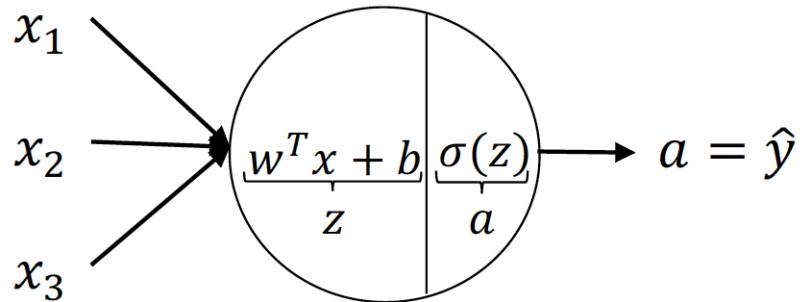


$$z = w^T x + b$$

$$a = \sigma(z)$$

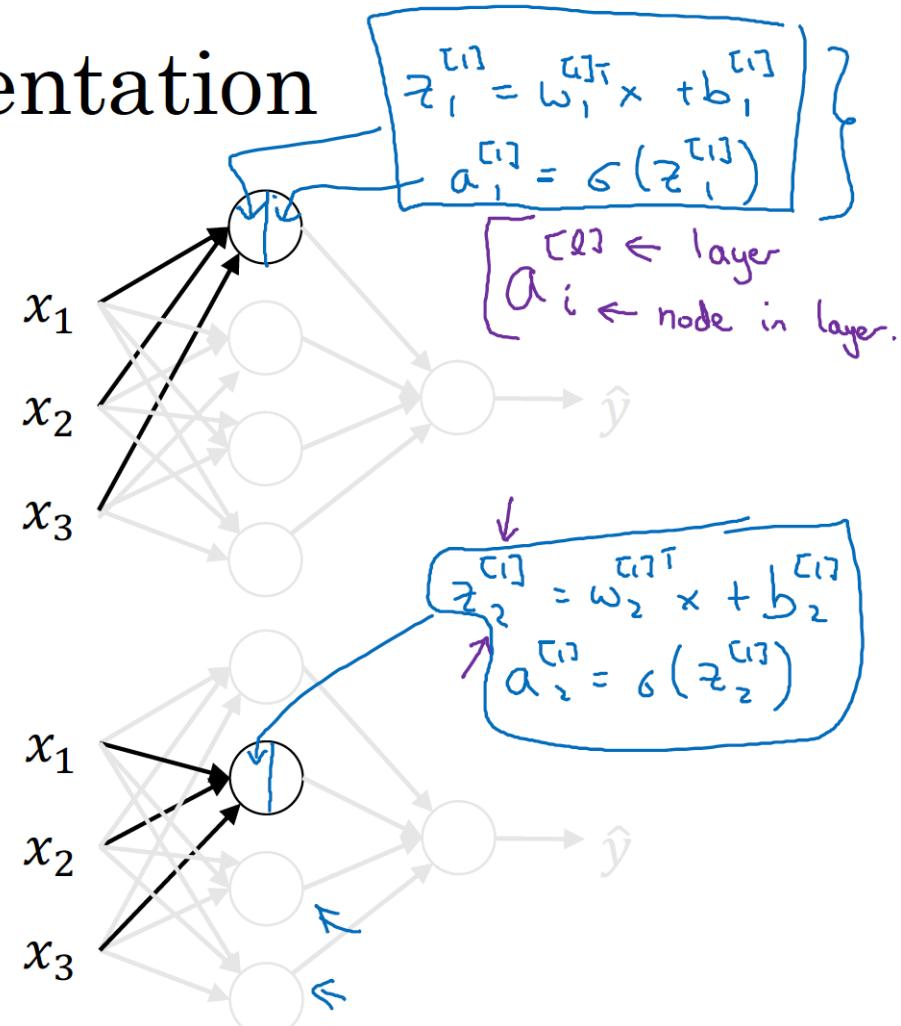


# Neural Network Representation



$$z = w^T x + b$$

$$a = \sigma(z)$$



# Neural Network Representation

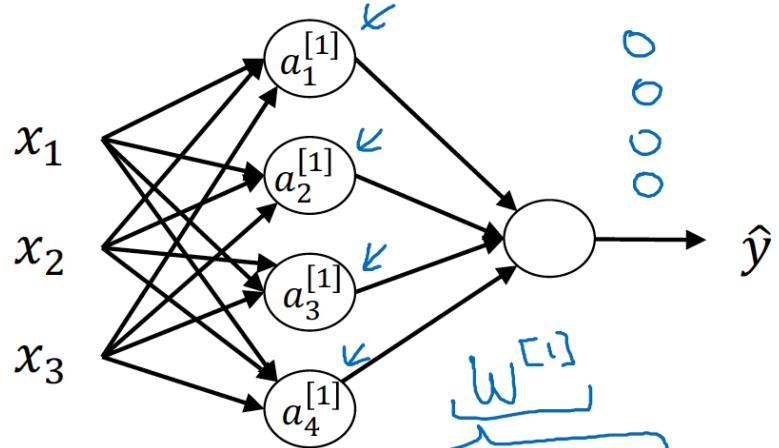


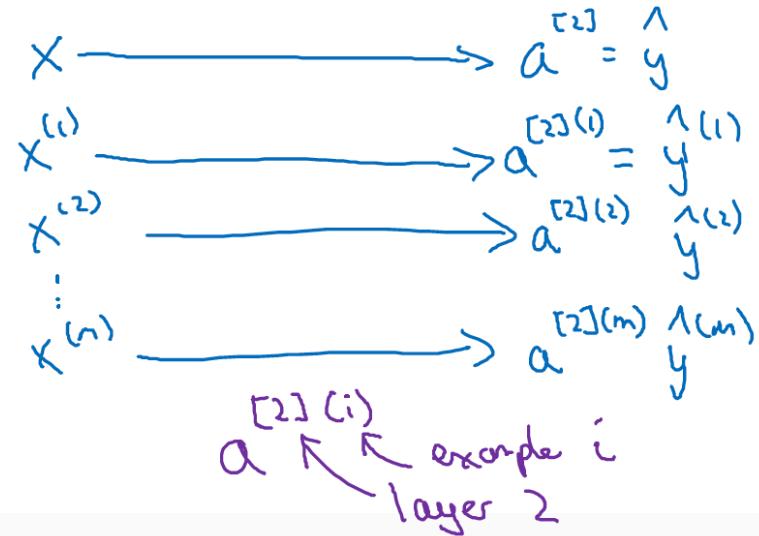
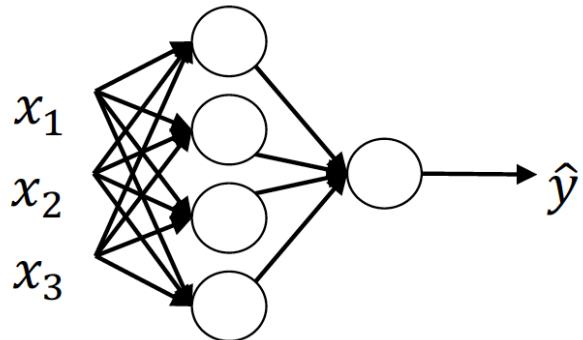
Diagram illustrating the mathematical representation of the neural network:

The diagram shows the forward pass through the first layer. The inputs  $x = [x_1, x_2, x_3]$  are multiplied by the weight matrix  $W^{[1]} = [w_1^{[1]T}, w_2^{[1]T}, w_3^{[1]T}, w_4^{[1]T}]$  and then added to the bias vector  $b^{[1]} = [b_1^{[1]}, b_2^{[1]}, b_3^{[1]}, b_4^{[1]}]$ . The result is passed through the activation function  $\sigma$  to produce the hidden layer activations  $a^{[1]} = [\sigma(z_1), \sigma(z_2), \sigma(z_3), \sigma(z_4)]$ .

Handwritten annotations highlight the components of the equation:

- $(w_i^{[1]})^T x$  is highlighted in red above the first term.
- $a^{[1]}$  is highlighted in red at the end of the equation.
- $z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$  is highlighted in purple.
- $z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$  is highlighted in purple.
- $z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}$  is highlighted in green.
- $z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}$  is highlighted in green.
- $a_1^{[1]} = \sigma(z_1)$  is highlighted in red.
- $a_2^{[1]} = \sigma(z_2)$  is highlighted in red.
- $a_3^{[1]} = \sigma(z_3)$  is highlighted in red.
- $a_4^{[1]} = \sigma(z_4)$  is highlighted in red.
- $z_1^{[1]}$ ,  $z_2^{[1]}$ ,  $z_3^{[1]}$ , and  $z_4^{[1]}$  are grouped by a blue bracket.
- $w_1^{[1]T}$ ,  $w_2^{[1]T}$ ,  $w_3^{[1]T}$ , and  $w_4^{[1]T}$  are grouped by a blue bracket.
- $b_1^{[1]}$ ,  $b_2^{[1]}$ ,  $b_3^{[1]}$ , and  $b_4^{[1]}$  are grouped by a green bracket.
- $a^{[1]}$  is grouped by a red bracket.
- $\sigma(z)$  is grouped by a blue bracket.

# Vectorizing across multiple examples



$$\left\{ \begin{array}{l} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{array} \right. \quad \rightarrow \boxed{\begin{array}{l} \text{for } i = 1 \text{ to } n, \\ z^{[1](i)} = w^{[1]}x^{(i)} + b^{[1]} \\ a^{[1](i)} = \sigma(z^{[1](i)}) \\ z^{[2](i)} = w^{[2]}a^{[1](i)} + b^{[2]} \\ a^{[2](i)} = \sigma(z^{[2](i)}) \end{array}}$$

# Justification for vectorized implementation

$$\underline{z}^{(1)(1)} = \underline{\omega}^{(1)} \underline{x}^{(1)} + \underline{b}^{(1)}, \quad \underline{z}^{(1)(2)} = \underline{\omega}^{(2)} \underline{x}^{(2)} + \underline{b}^{(2)}, \quad \underline{z}^{(1)(3)} = \underline{\omega}^{(3)} \underline{x}^{(3)} + \underline{b}^{(3)}$$

$$\underline{\omega}^{(1)} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

$$\underline{\omega}^{(1)} \underline{x}^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$\underline{\omega}^{(2)} \underline{x}^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$\underline{\omega}^{(3)} \underline{x}^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$\underline{z}^{(1)} = \underline{\omega}^{(1)} \underline{x} + \underline{b}^{(1)}$$

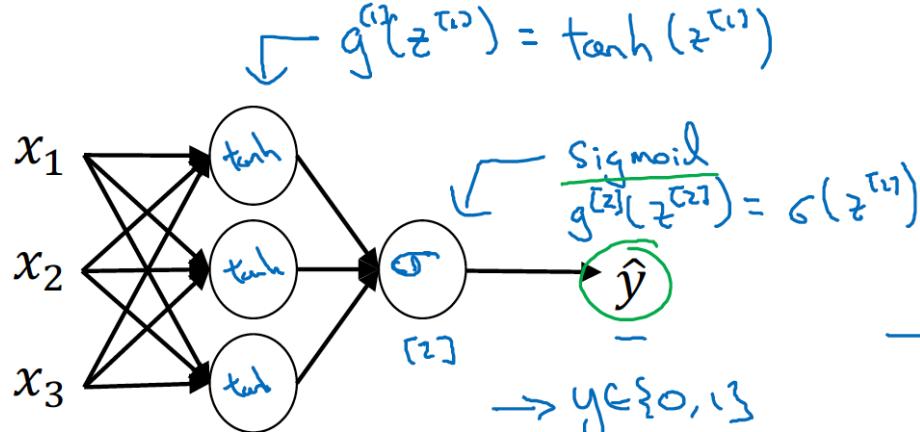
$\times$

$$\underline{\omega}^{(1)} \begin{bmatrix} 1 & | & 1 \\ X^{(1)} & X^{(2)} & X^{(3)} \dots \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} z^{(1)(1)} \\ z^{(1)(2)} \\ z^{(1)(3)} \\ \vdots \\ \vdots \end{bmatrix} = \underline{z}^{(1)}$$

$\underline{\omega}^{(1)} \underline{x}^{(1)} = z^{(1)(1)}$

$$\underline{z}^{(1)(1)} = \underline{\omega}^{(1)} \underline{x}^{(1)} + \underline{b}^{(1)}$$
$$\underline{z}^{(1)(2)} = \underline{\omega}^{(2)} \underline{x}^{(2)} + \underline{b}^{(2)}$$
$$\underline{z}^{(1)(3)} = \underline{\omega}^{(3)} \underline{x}^{(3)} + \underline{b}^{(3)}$$

# Activation functions



Given  $x$ :

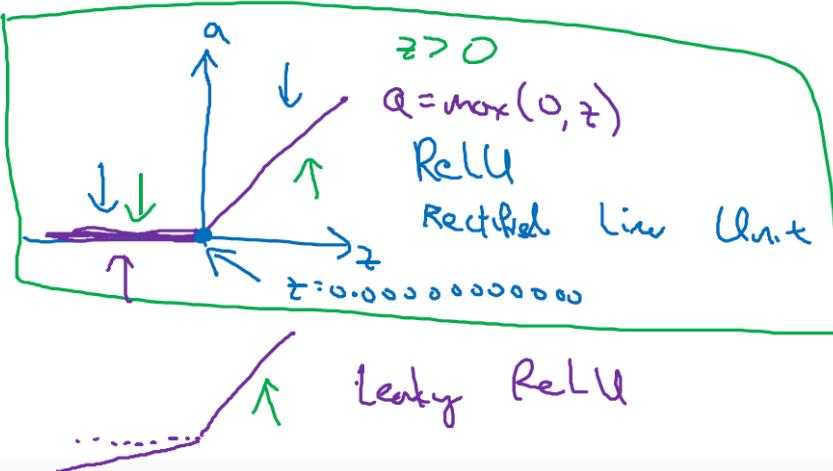
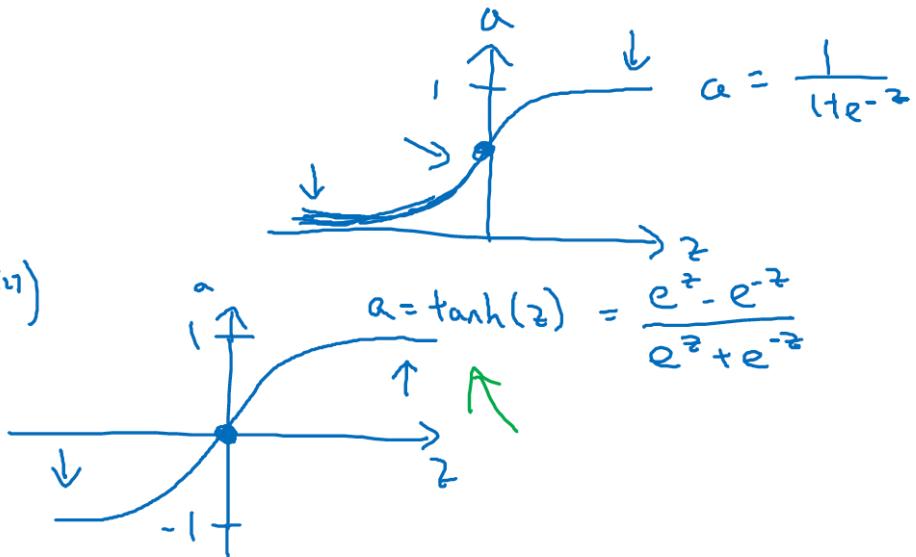
$$\begin{matrix} 0 \\ -1 \\ 1 \end{matrix}$$

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

$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})} \quad g^{(1)}(z^{(1)})$$

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

$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})} \quad g^{(2)}(z^{(2)})$$



# Computing gradients

## Logistic regression

$$z = w^T x + b$$

$$\delta w = \delta z \cdot x$$

$$\delta b = \delta z$$

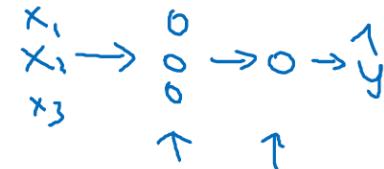
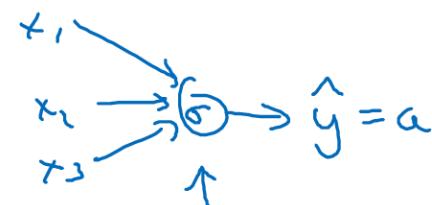
$$\underline{\delta z} = a - y$$

$$\delta z = \delta a \cdot g'(z)$$

$$g(z) = \sigma(z)$$

$$\frac{\delta L}{\delta z} = \frac{\delta L}{\delta a} \cdot \frac{\delta a}{\delta z}$$

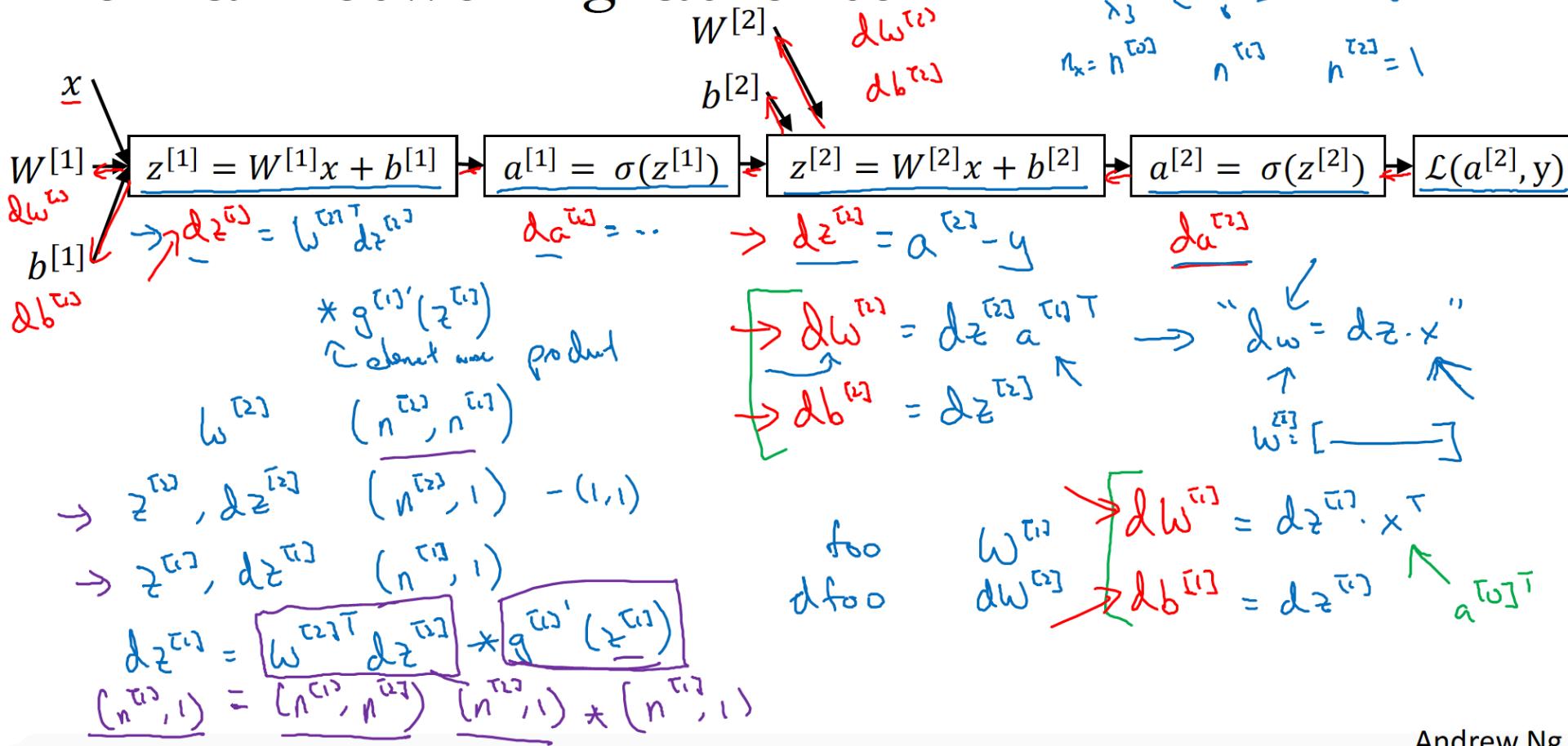
$$\text{"d}z" = \text{"da"}$$



$$\begin{aligned}\delta a &= \frac{d}{da} L(a, y) = -y \log a - (1-y) \log(1-a) \\ &= -\frac{y}{a} + \frac{1-y}{1-a}\end{aligned}$$

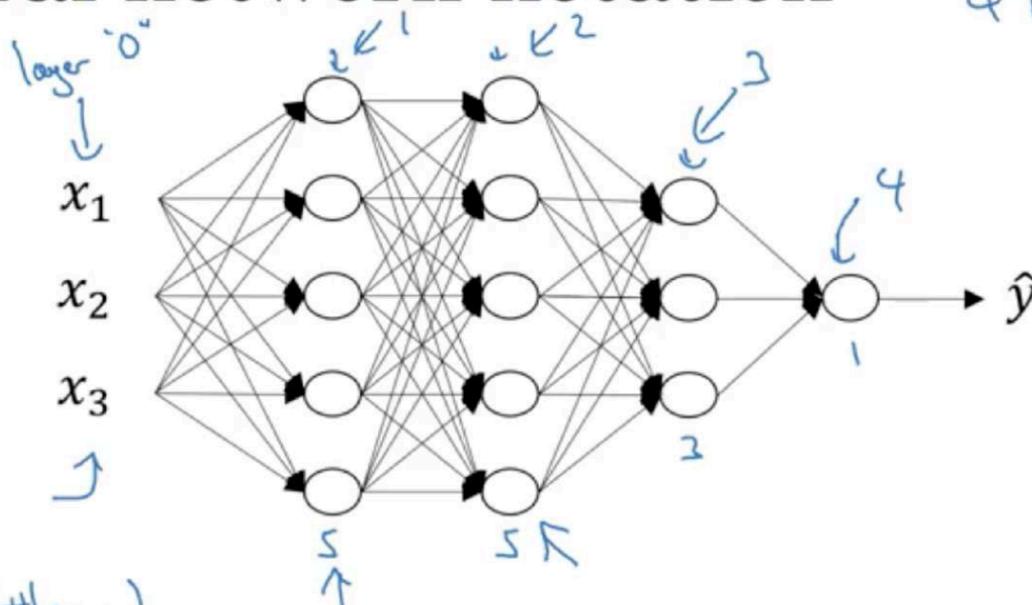
$$\frac{d}{dz} g(z) = g'(z)$$

# Neural network gradients



# Deep neural network notation

4 layer NN



$$l = 4 \text{ (#layers)}$$

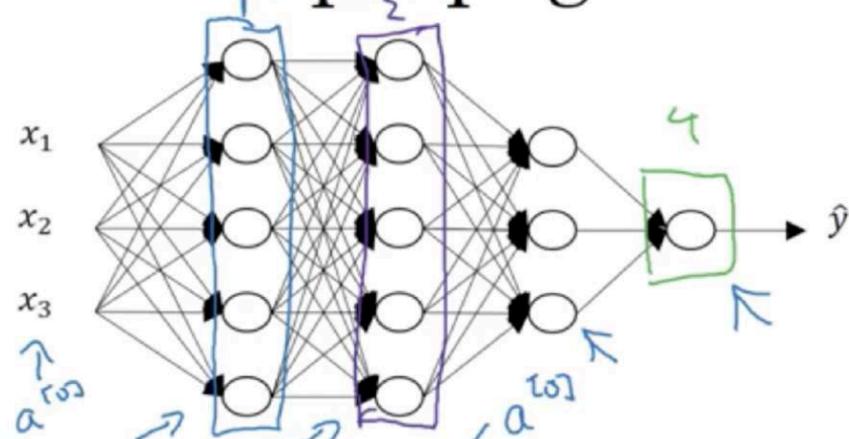
$$n^{[l]} = \# \text{units in layer } l$$

$$a^{[l]} = \text{activations in layer } l$$

$$a^{[l]} = g^{[l]}(z^{[l]}), \quad w^{[l]}$$

$$n^{[1]} = 5, \quad n^{[2]} = 5, \quad n^{[3]} = 3, \quad n^{[4]} = n^{[L]} = 1$$
$$n^{[0]} = n_x = 3$$

# Forward propagation in a deep network



$$x : z^{[0]} = \underline{w}^{[0]} \underline{a} + \underline{b}^{[0]}$$

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

$$\underline{z}^{[2]} = \underline{w}^{[2]} \underline{a}^{[1]} + \underline{b}^{[2]}$$

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

$$z^{[4]} = w^{[4]} a^{[3]} + b^{[4]}, \quad a^{[4]} = g^{[4]}(z^{[4]})$$

$$\begin{aligned} z^{[l]} &= w^{[l]} a^{[l-1]} + b^{[l]} \\ a^{[l]} &= g^{[l]}(z^{[l]}) \end{aligned}$$

Verteilt:

$$\begin{aligned} z^{[1]} &= (w^{[1]} \otimes a^{[0]} + b^{[1]}) \\ a^{[1]} &= g^{[1]}(z^{[1]}) \end{aligned} \quad \rightarrow x = a^{[0]}$$

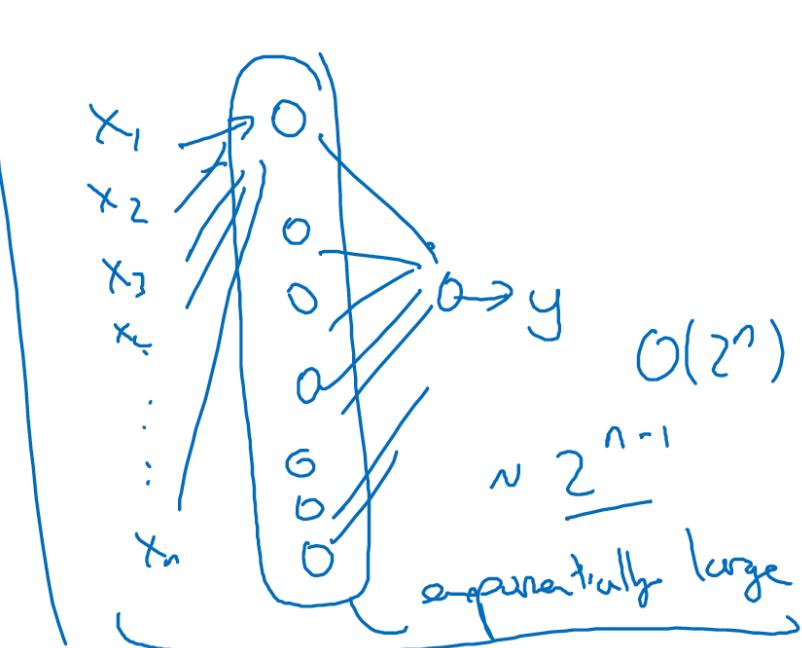
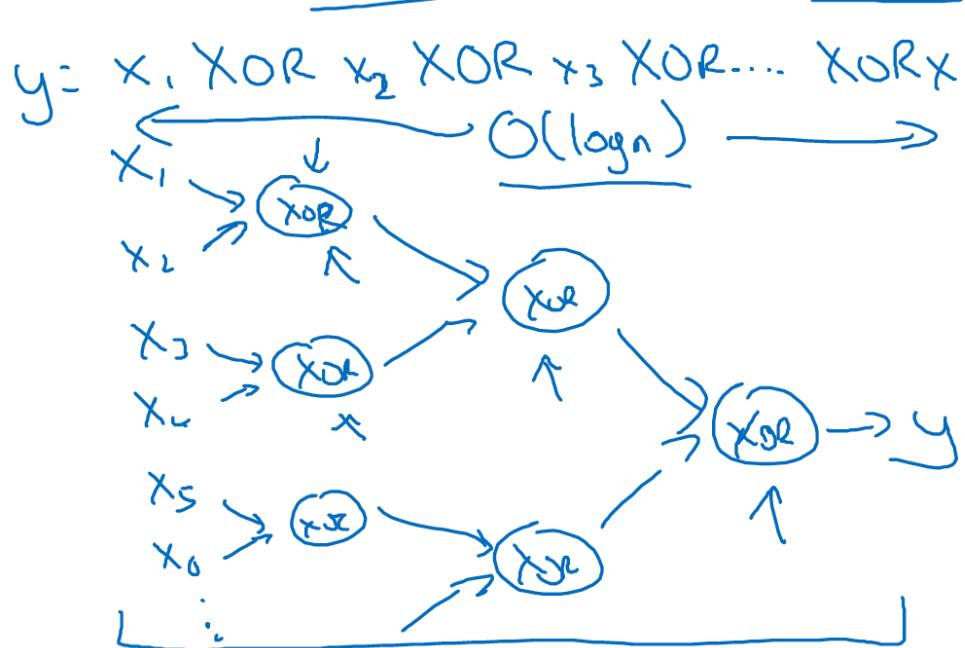
for  $l=1..4$

$$\begin{aligned} z^{[2]} &= w^{[2]} a^{[1]} + b^{[2]} \\ a^{[2]} &= g^{[2]}(z^{[2]}) \end{aligned}$$

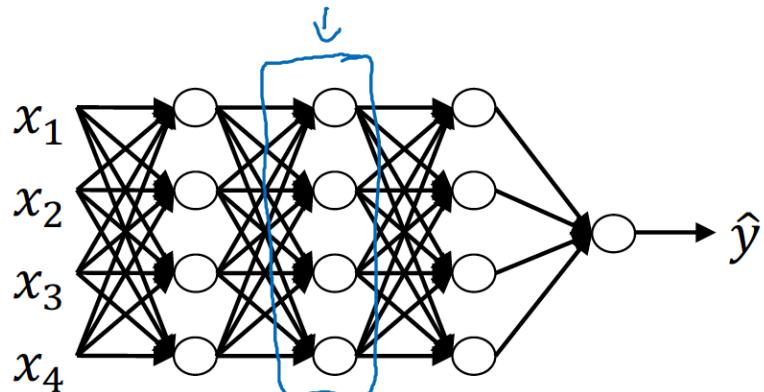
$$\hat{y} = g(z^{[4]}) = a^{[4]}$$

# Circuit theory and deep learning

Informally: There are functions you can compute with a “small” L-layer deep neural network that shallow networks require exponentially more hidden units to compute.



# Forward and backward functions

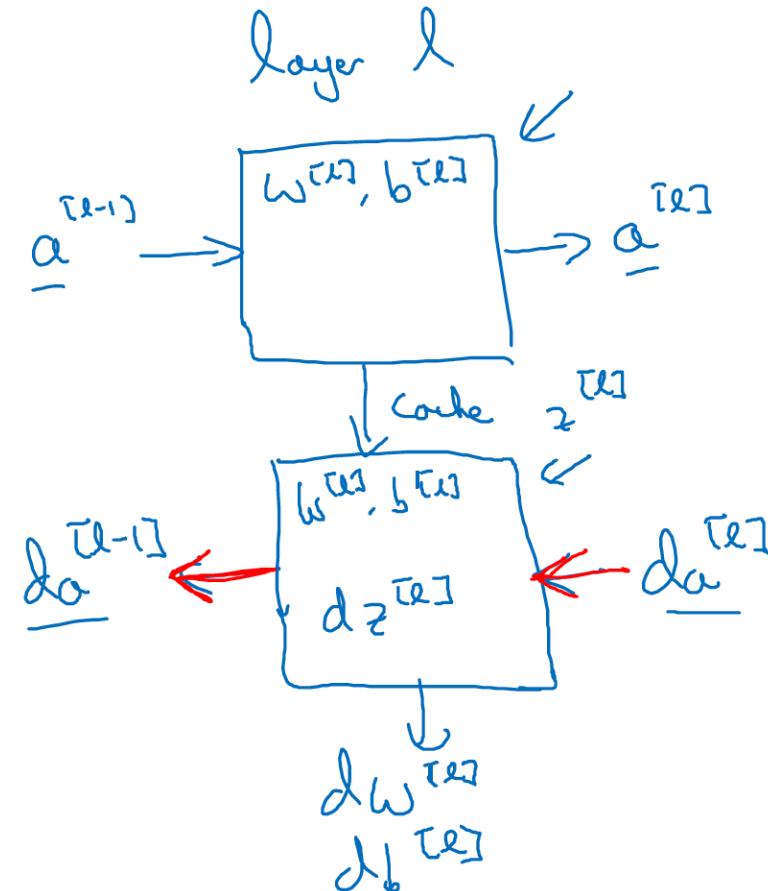


Layer l:  $W^{[l]}, b^{[l]}$

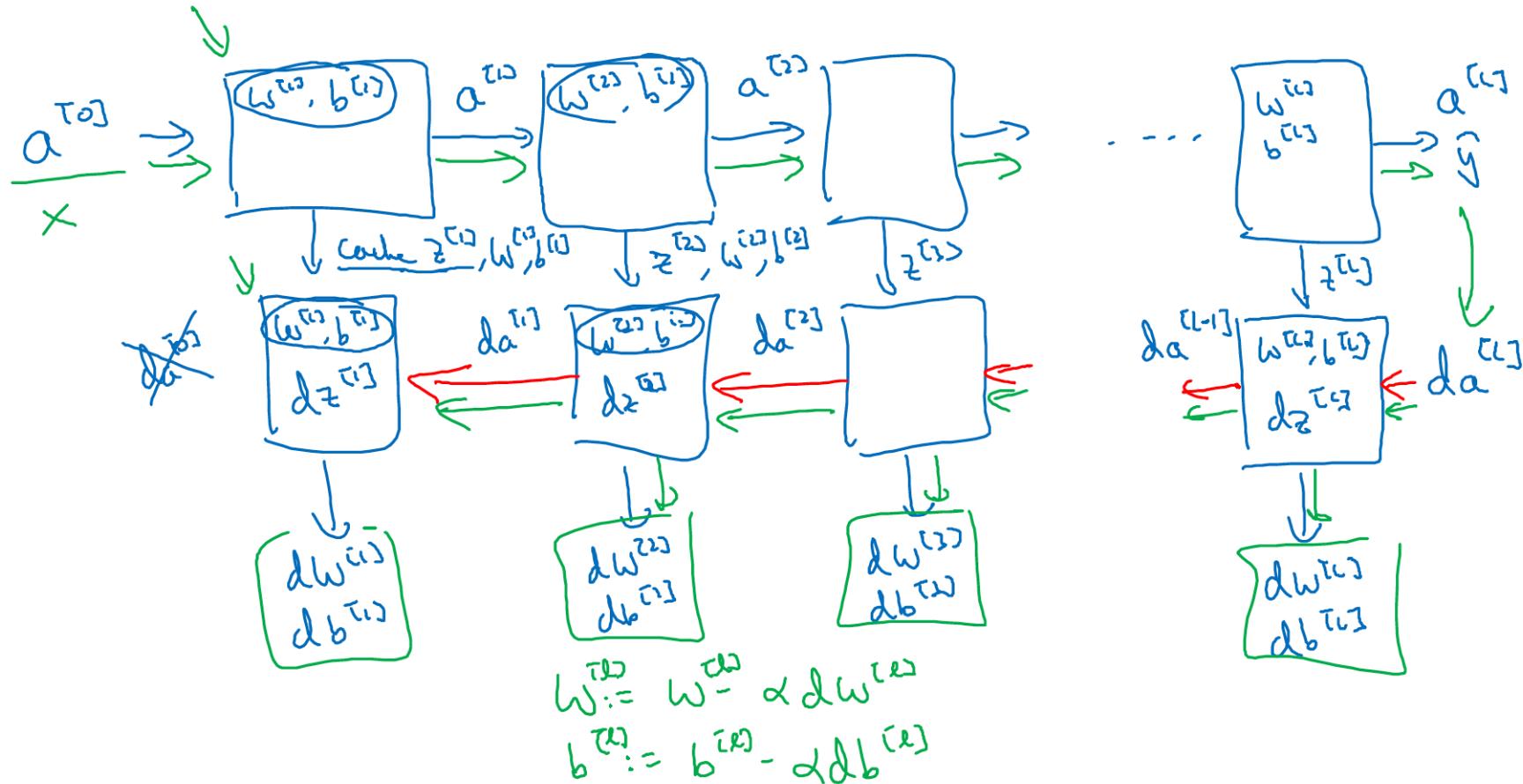
→ Forward: Input  $a^{[l-1]}$ , output  $a^{[l]}$

$$\underline{z}^{[l]} = W^{[l]} \underline{a}^{[l-1]} + b^{[l]}$$
$$\underline{a}^{[l]} = g^{[l]}(\underline{z}^{[l]})$$

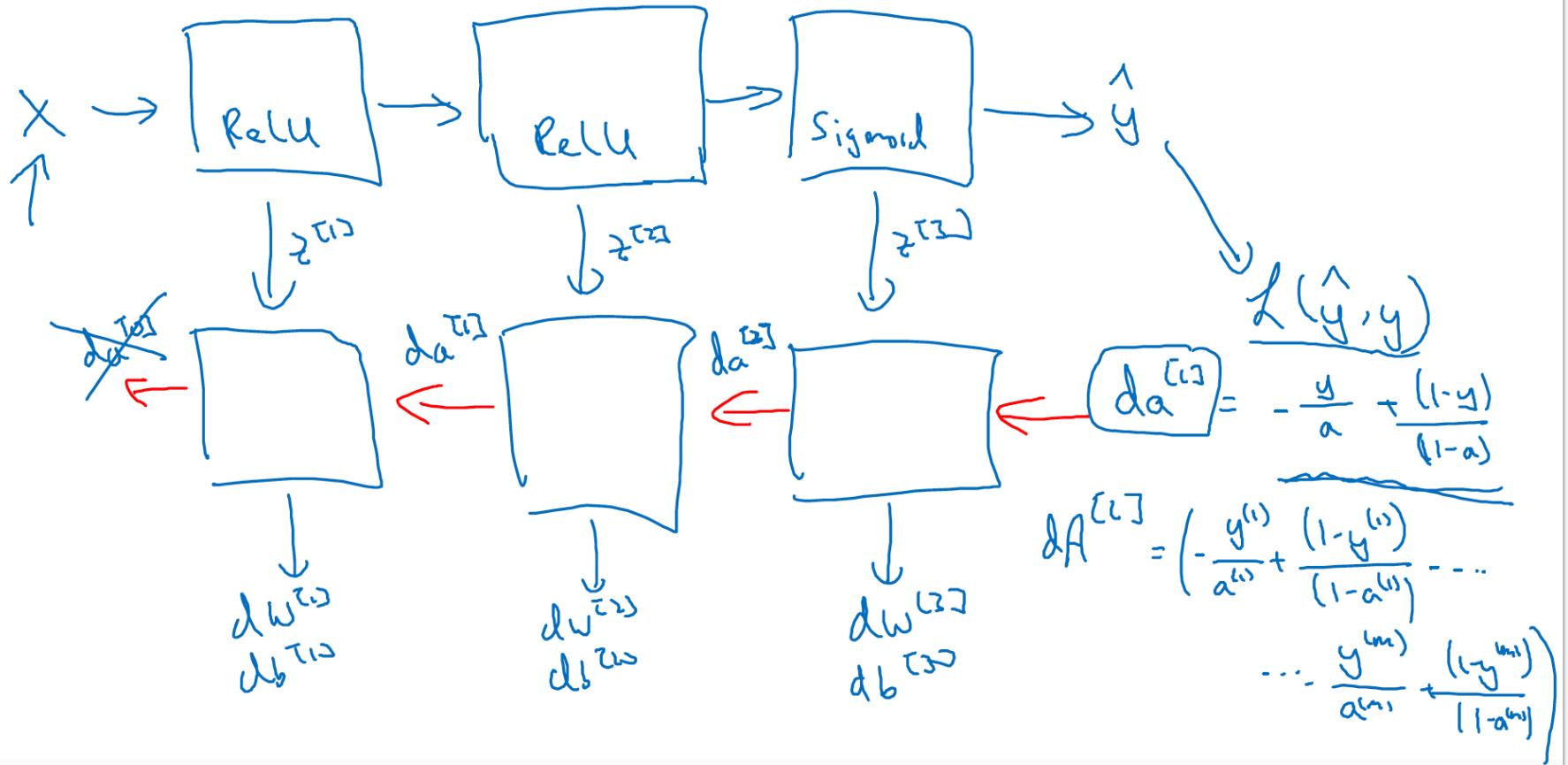
→ Backward: Input  $da^{[l]}$ , output  $\frac{da^{[l-1]}}{dw^{[l]}}$   
cache ( $\underline{z}^{[l]}$ )



# Forward and backward functions



# Summary



# What are hyperparameters?

Parameters:  $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \dots$

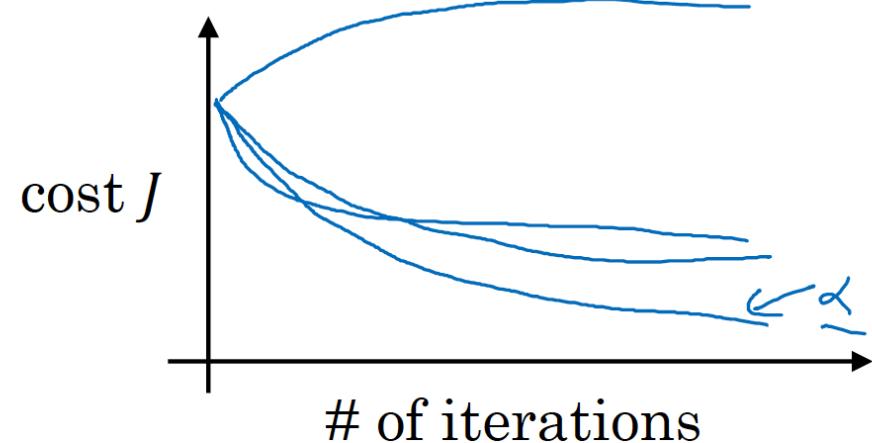
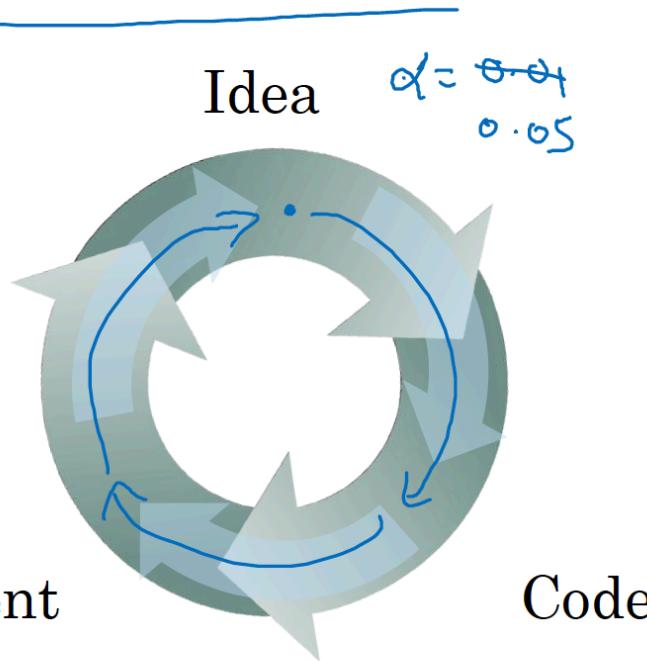
Hyperparameters:

- learning rate  $\alpha$
- # iterations
- # hidden layers  $L$
- # hidden units  $n^{[1]}, n^{[2]}, \dots$
- choice of activation function

}

Others: Momentum, mini-batch size, regularizations, ...

# Applied deep learning is a very empirical process



Vision, Speech, NLP, Ad, Search, Reinforcement.

# **Word Embedding**

...an efficient method for learning high quality distributed vector ...



# To compare pieces of text

- We need effective representation of :
  - Words
  - Sentences
  - Text
- Approach 1: Use existing thesauri or ontologies like WordNet and Snomed CT (for medical). Drawbacks:
  - Manual
  - Not context specific
- Approach 2: Use co-occurrences for word similarity. Drawbacks:
  - Quadratic space needed
  - Relative position and order of words not considered

# Approach 3: low dimensional vectors

- Store only “important” information in fixed, low dimensional vector.
- Single Value Decomposition (SVD) on co-occurrence matrix
  - is the best rank  $k$  approximation to  $X$ , in terms of least squares
  - Motel = [0.286, 0.792, -0.177, -0.107, 0.109, -0.542, 0.349, 0.271]
- $m = n = \text{size of vocabulary}$

$$n \begin{array}{|c|} \hline m \\ \hline \end{array} X = n \begin{array}{|c|} \hline r \\ \hline \end{array} U \begin{array}{|c|} \hline r \\ \hline \end{array} S \begin{array}{|c|} \hline m \\ \hline \end{array} V^T$$
$$n \begin{array}{|c|} \hline m \\ \hline \end{array} \hat{X} = n \begin{array}{|c|} \hline k \\ \hline \end{array} \hat{U} \begin{array}{|c|} \hline k \\ \hline \end{array} \hat{S} \begin{array}{|c|} \hline m \\ \hline \end{array} \hat{V}^T$$

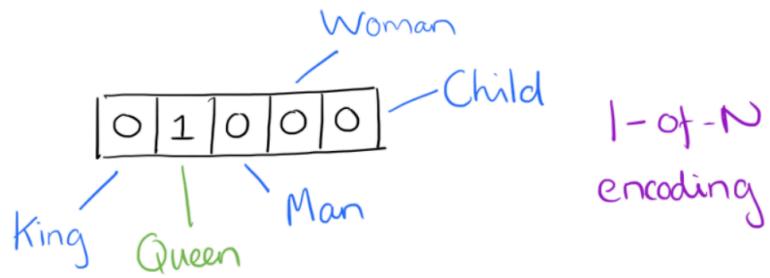
## Problems with SVD

- Computational cost scales quadratically for  $n \times m$  matrix:  $O(mn^2)$  flops (when  $n < m$ )
- Hard to incorporate new words or documents
- Does not consider order of words

# **word2vec Approach to represent the meaning of word**

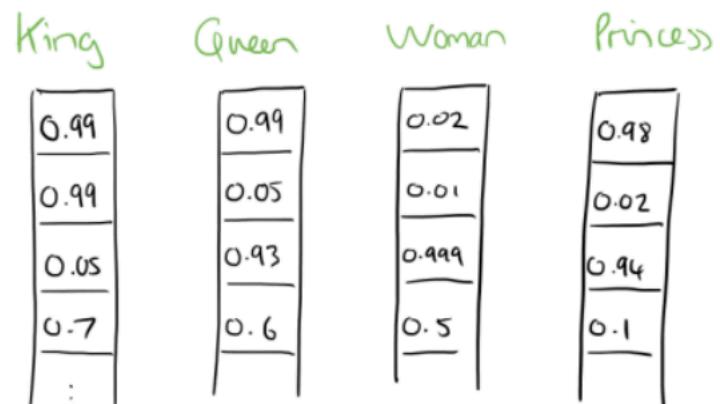
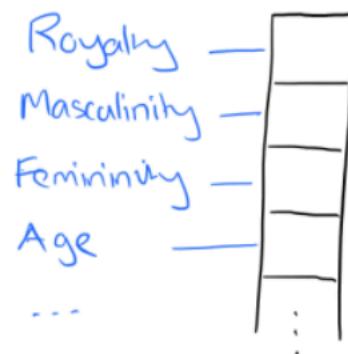
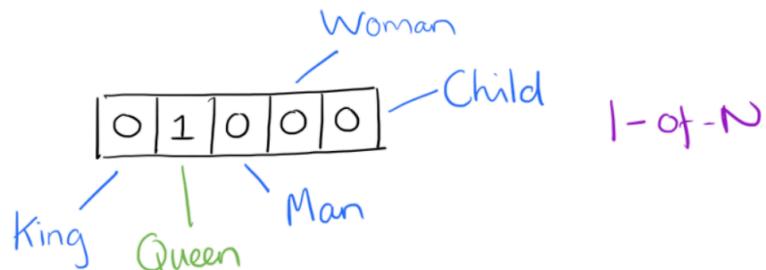
- Represent each word with a low-dimensional vector
- Word similarity = vector similarity
- Key idea: Predict surrounding words of every word
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

# Word2Vec

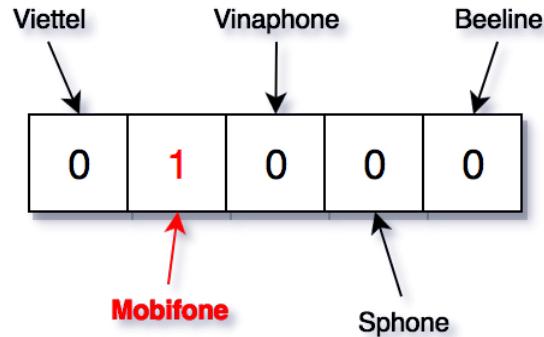


1-of-N  
encoding

# Word2Vec

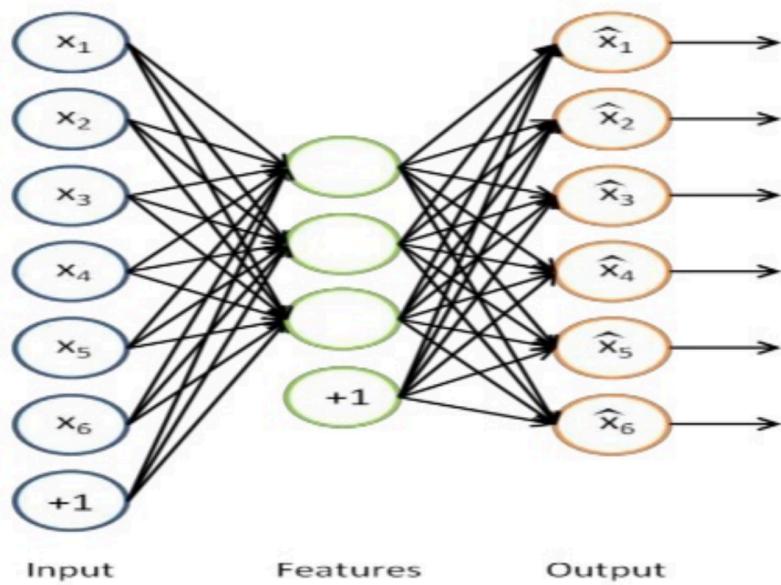


# Word2Vec

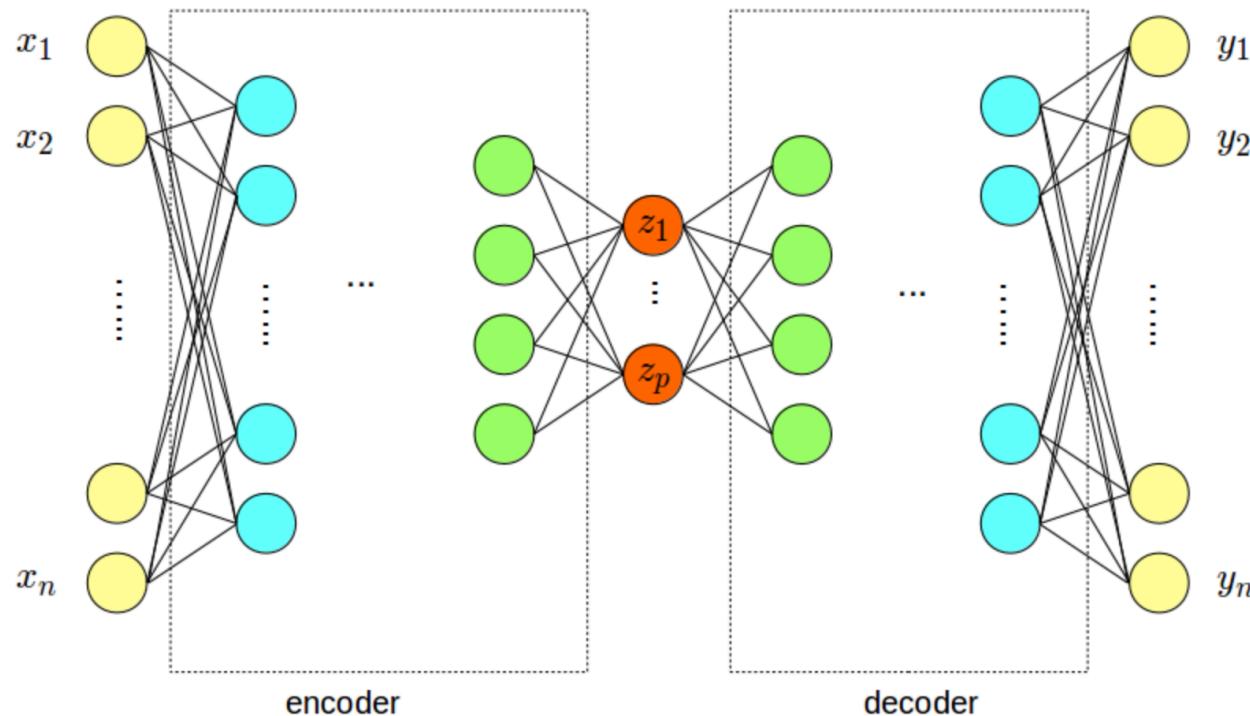


Viettel	Mobifone	VinaPhone	Sphone	Beeline
0.12	0.41	0.25	0.66	0.34
0.02	0.63	0.33	0.49	0.52
0.01	0.44	0.21	0.23	0.37

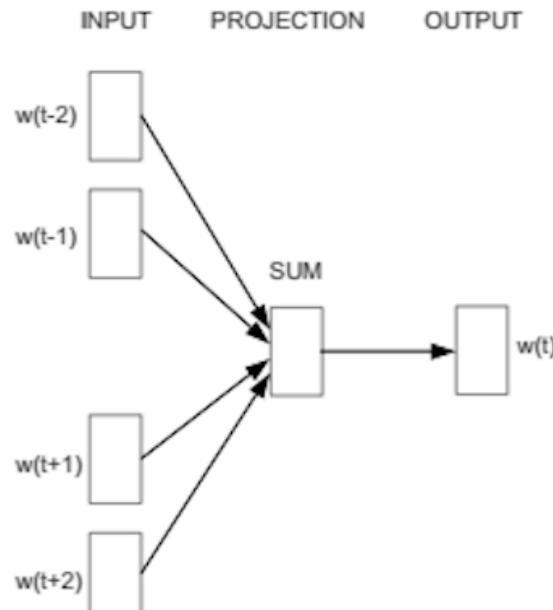
# Auto-Encoder



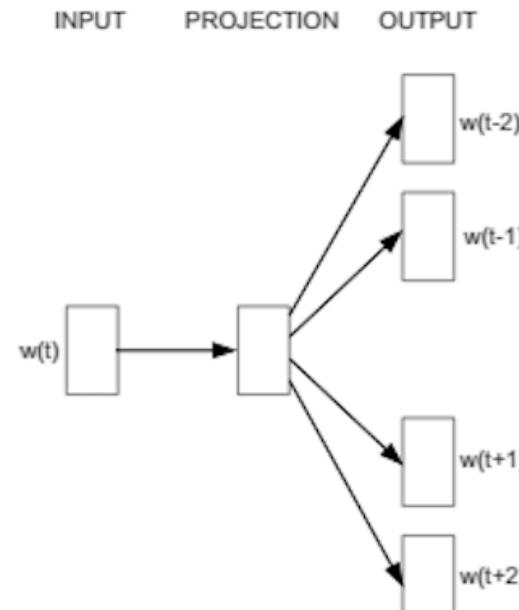
# Stacked Auto-Encoder



# Word2vec



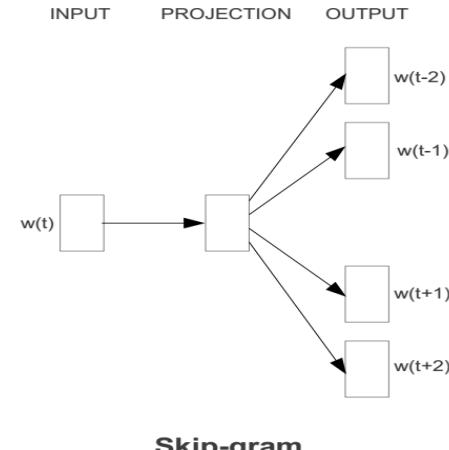
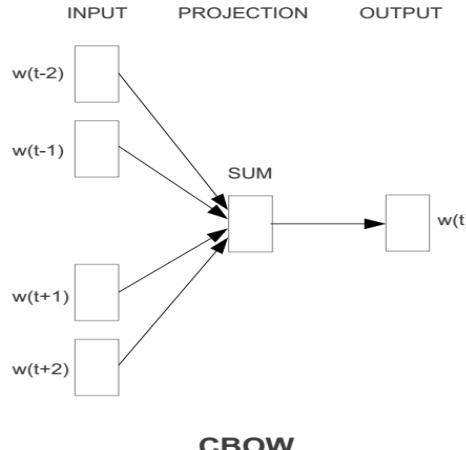
CBOW



Skip-gram

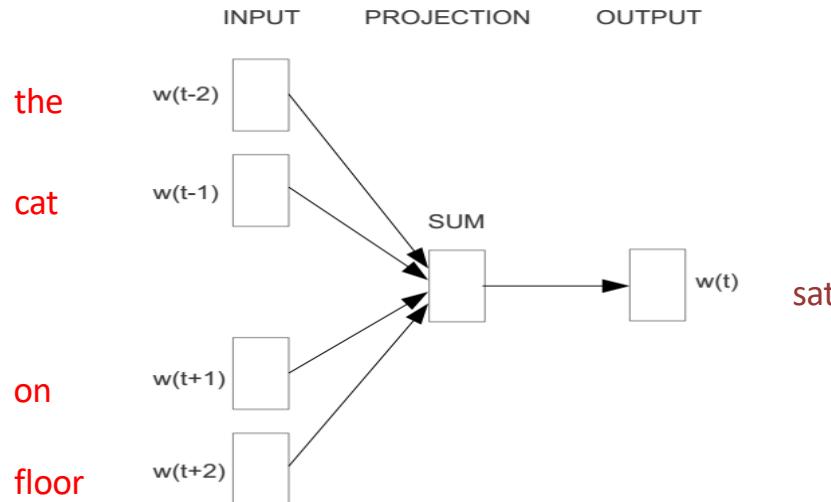
# Represent the meaning of word – word2vec

- 2 basic neural network models:
  - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
  - Skip-gram (SG): use a word to predict the surrounding ones in window.



# Word2vec – Continuous Bag of Word

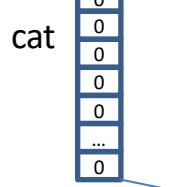
- E.g. “The cat sat on floor”
  - Window size = 2



Index of cat in vocabulary

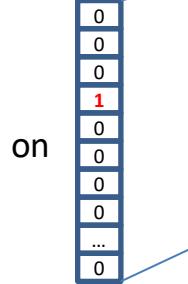
Input layer

cat



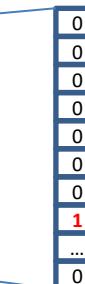
Hidden layer

on

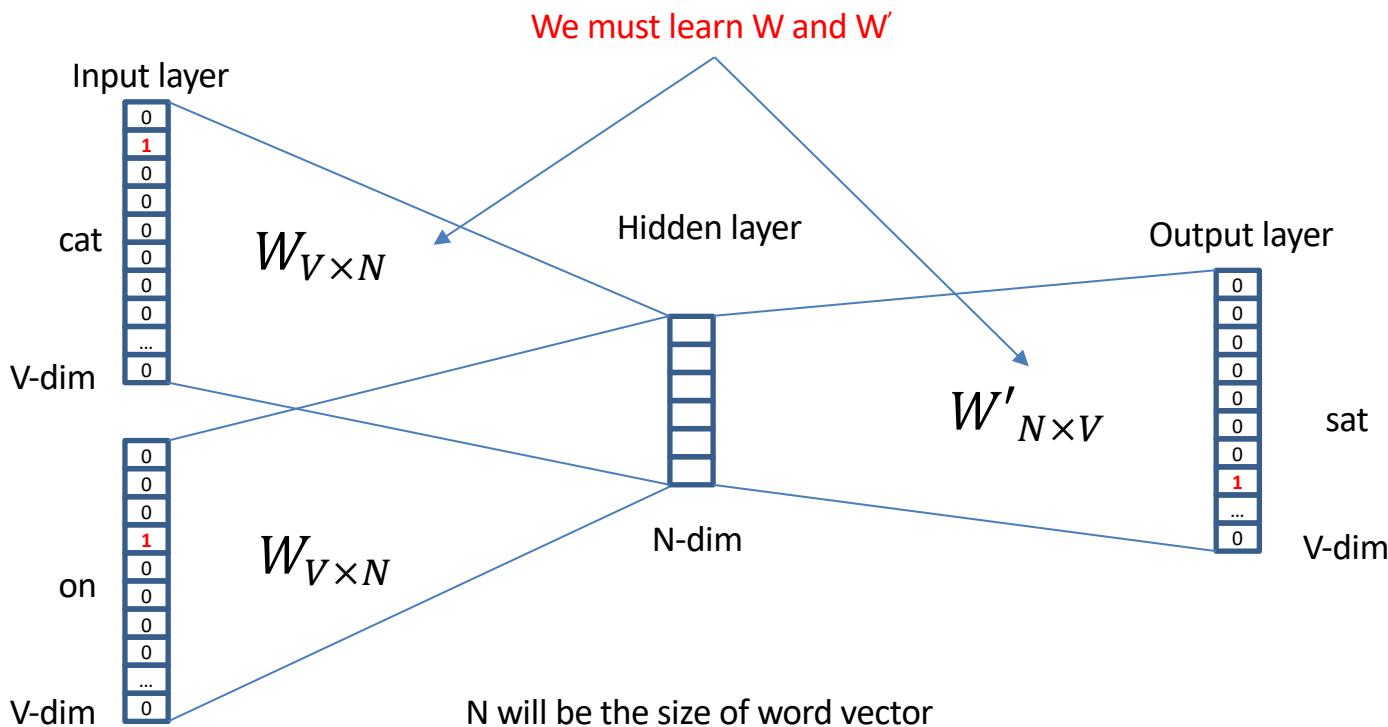


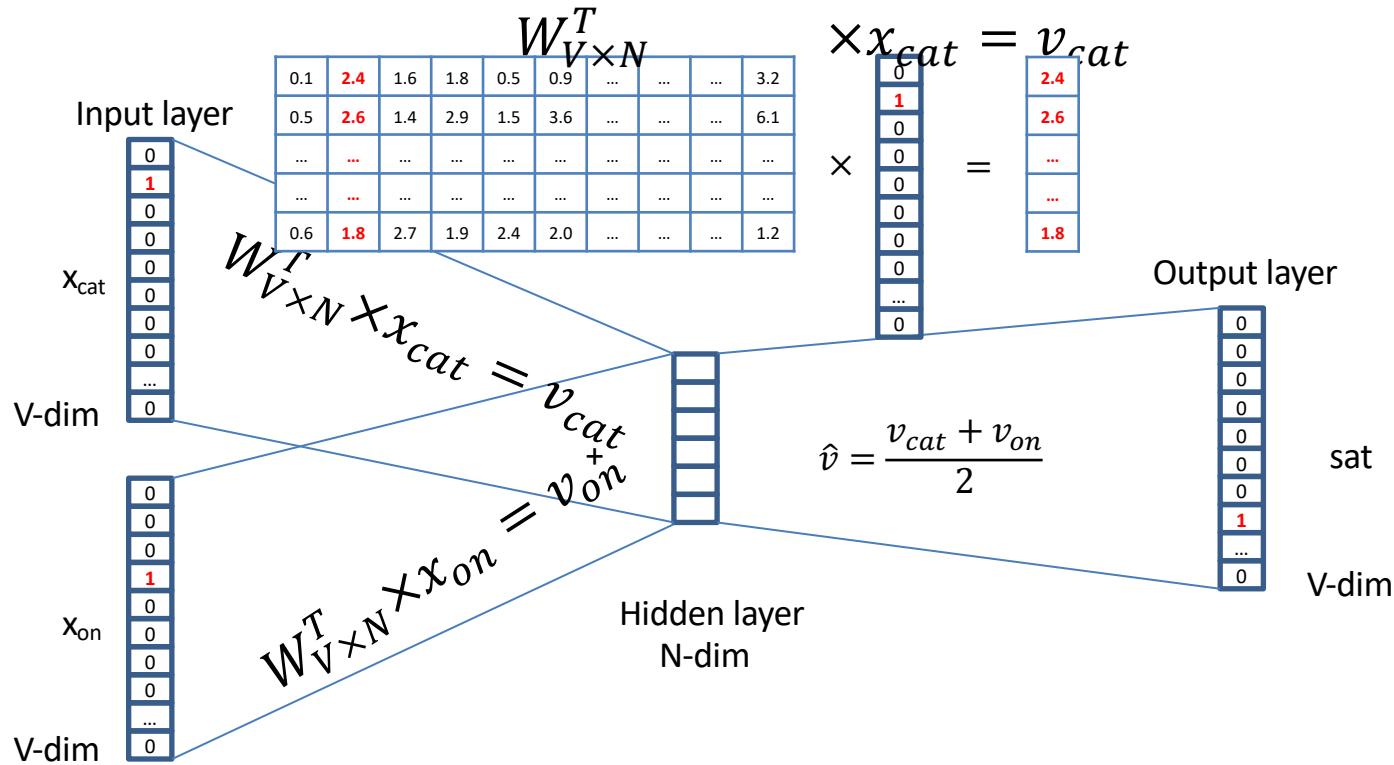
Output layer

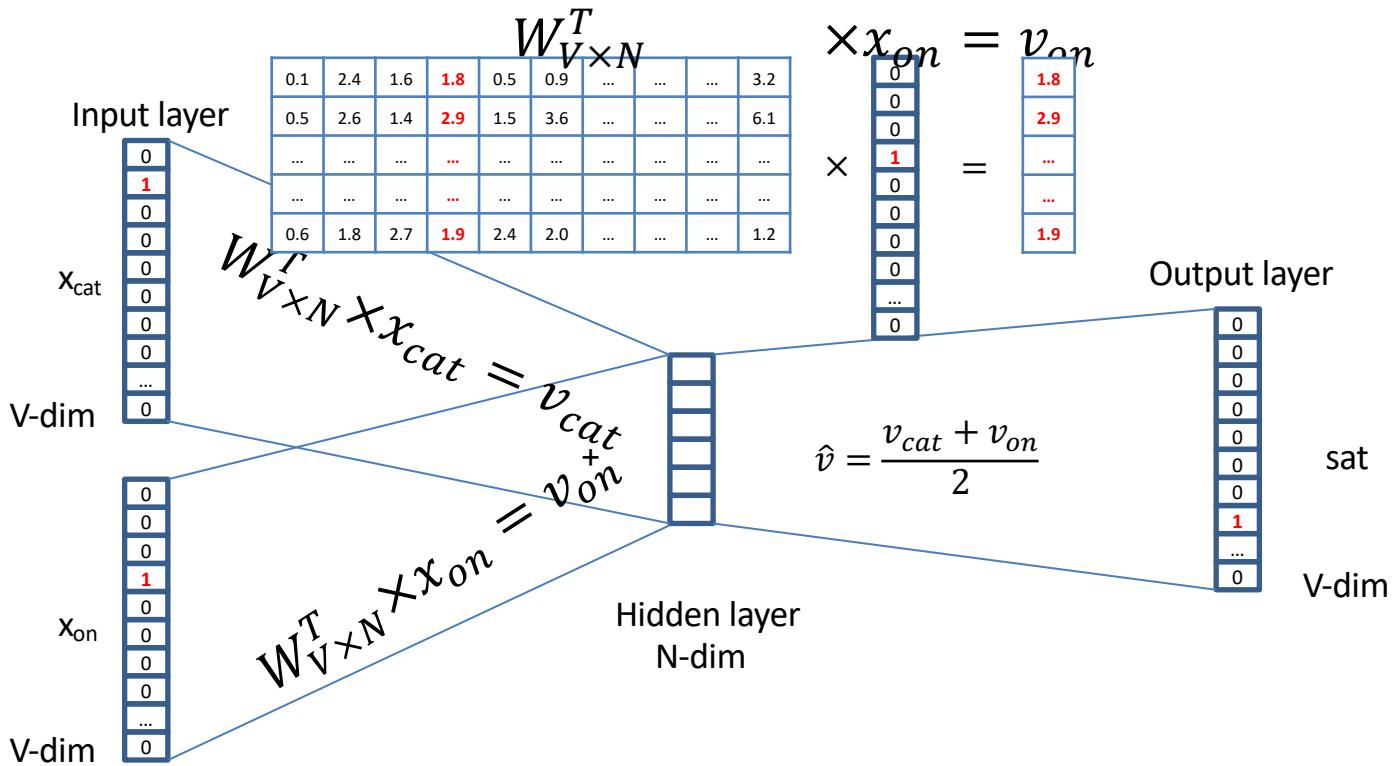
sat

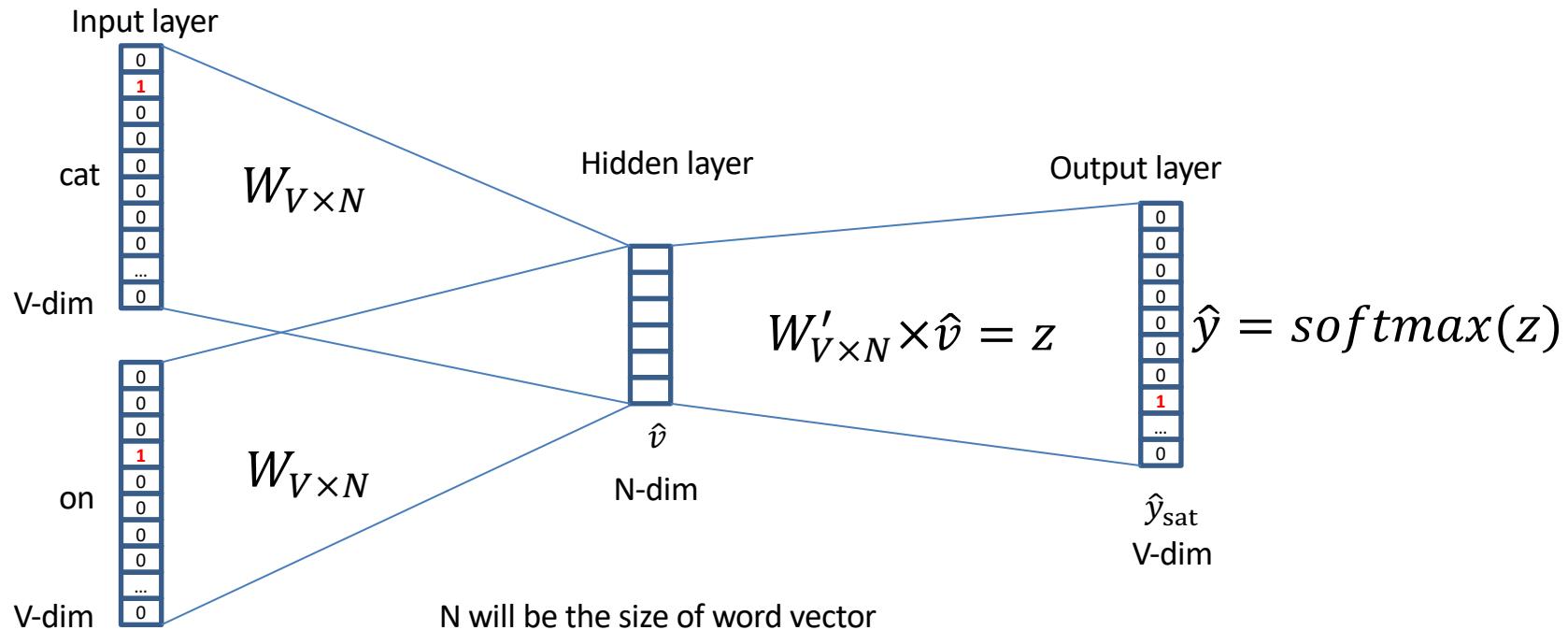


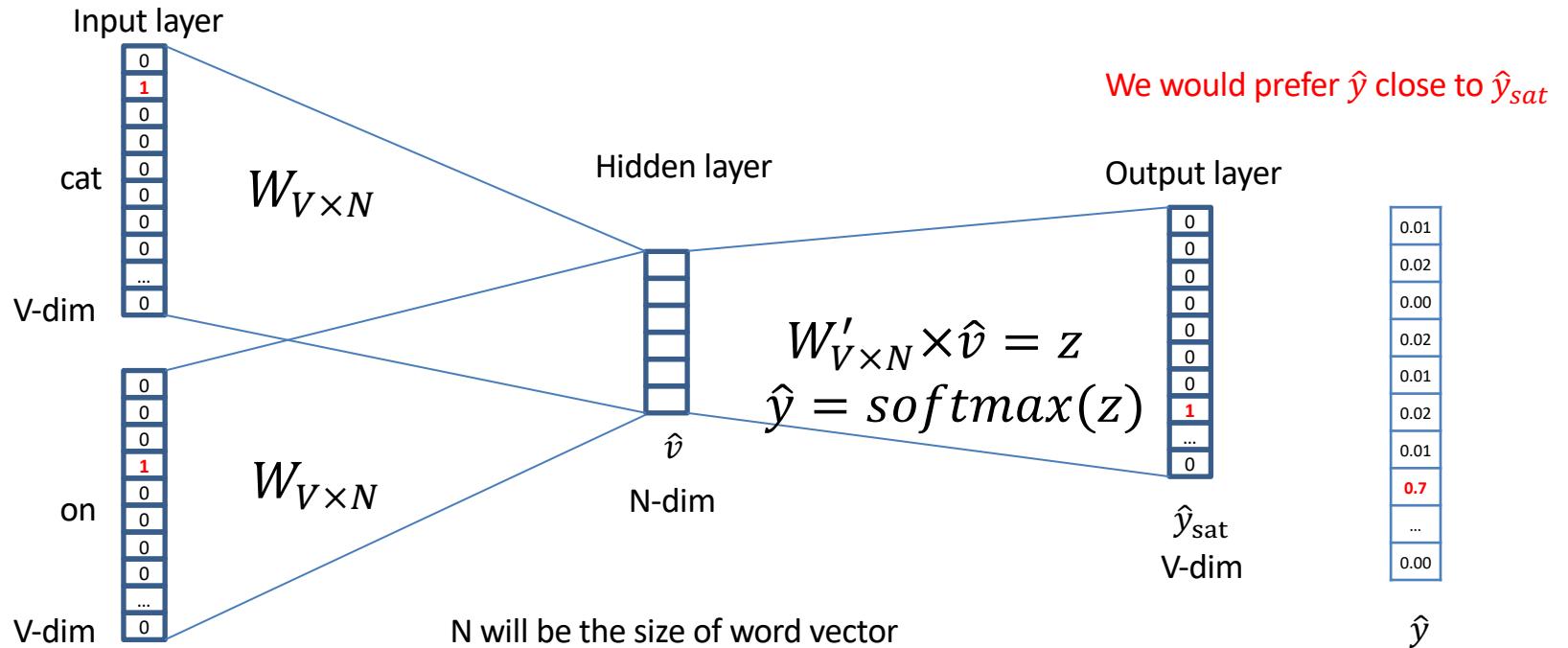
one-hot vector

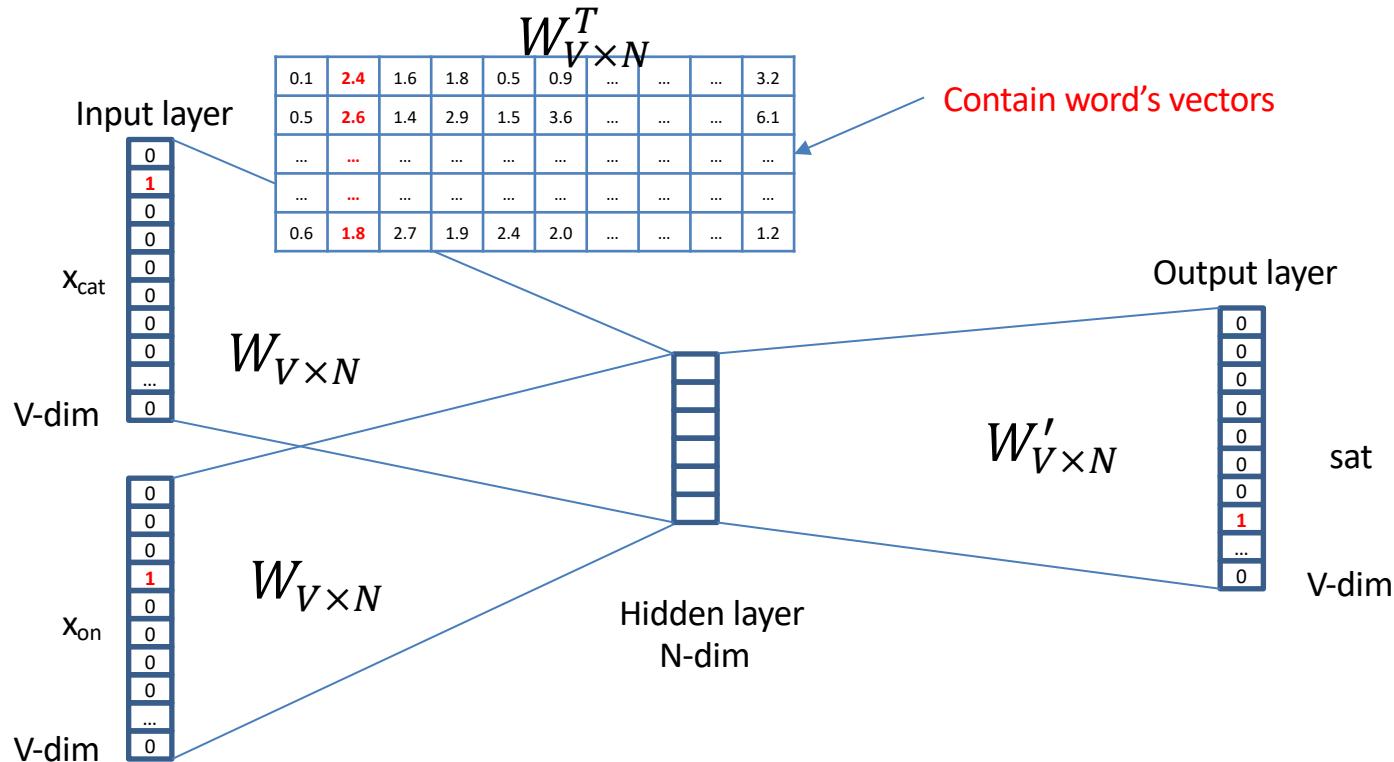












We can consider either  $W$  or  $W'$  as the word's representation. Or even take the average.

# Some interesting results

## Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

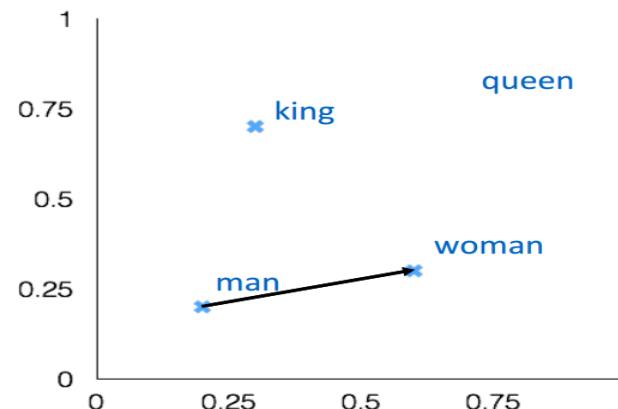
a:b :: c:?



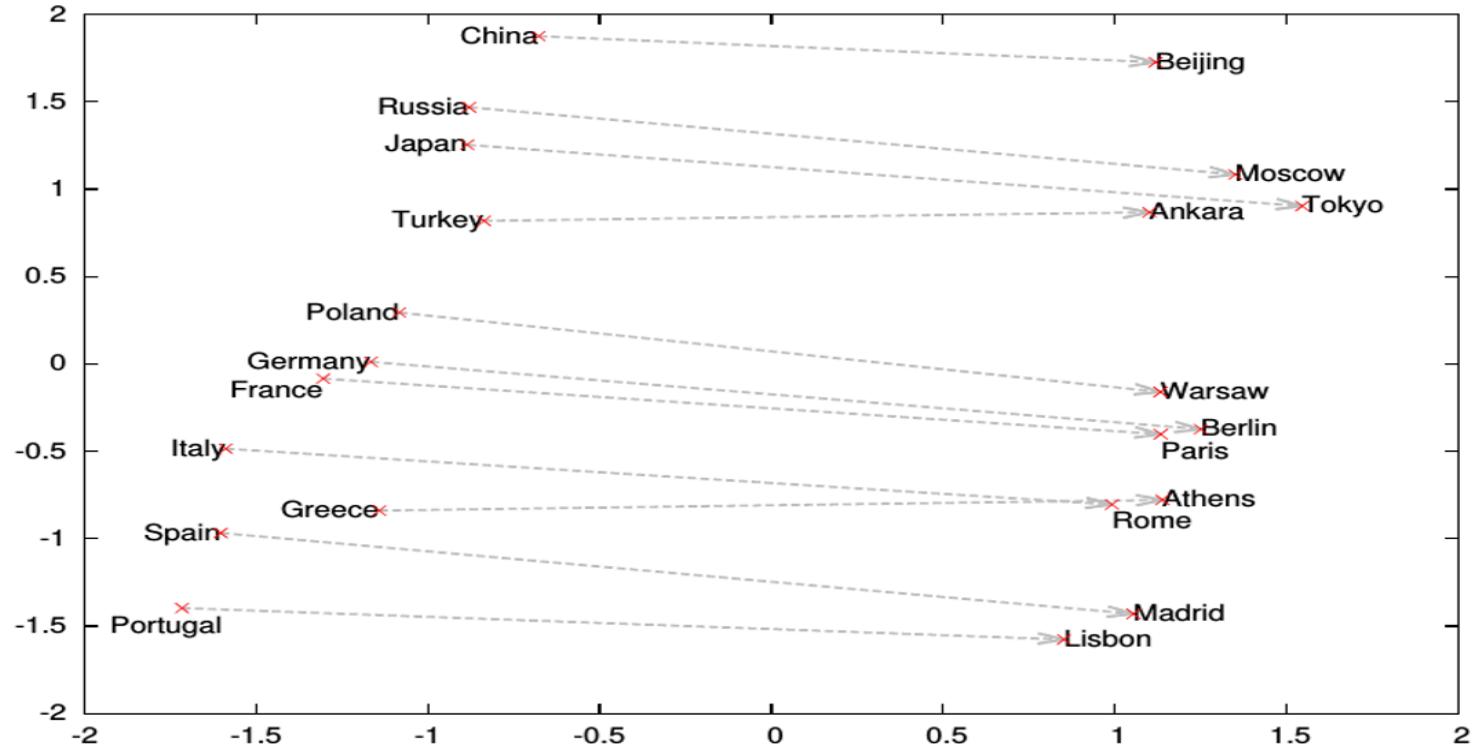
$$d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\|}$$

man:woman :: king:?

+	king	[ 0.30 0.70 ]
-	man	[ 0.20 0.20 ]
+	woman	[ 0.60 0.30 ]
<hr/>		
		queen [ 0.70 0.80 ]



# Word analogies



## Combination of tf.idf and WE

- Recall
  - $\text{Doc} = \{\text{tfidf}_1, \text{tfidf}_2, \dots, \text{tfidf}_t\}$
  - $\text{W1} = \{x_{11}, x_{12}, \dots, x_{\_t}\}$
  - ...
  - $\text{Wt} = \{x_{t1}, x_{t2}, \dots, x_{\_tt}\}$
- Then  $d\_1 = W1 * \text{tfidf}_1, d\_2 = W2 * \text{tfidf}_2, \dots$  etc
- Finally  $\text{DocVect} = \text{average of all } d\_i$