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.

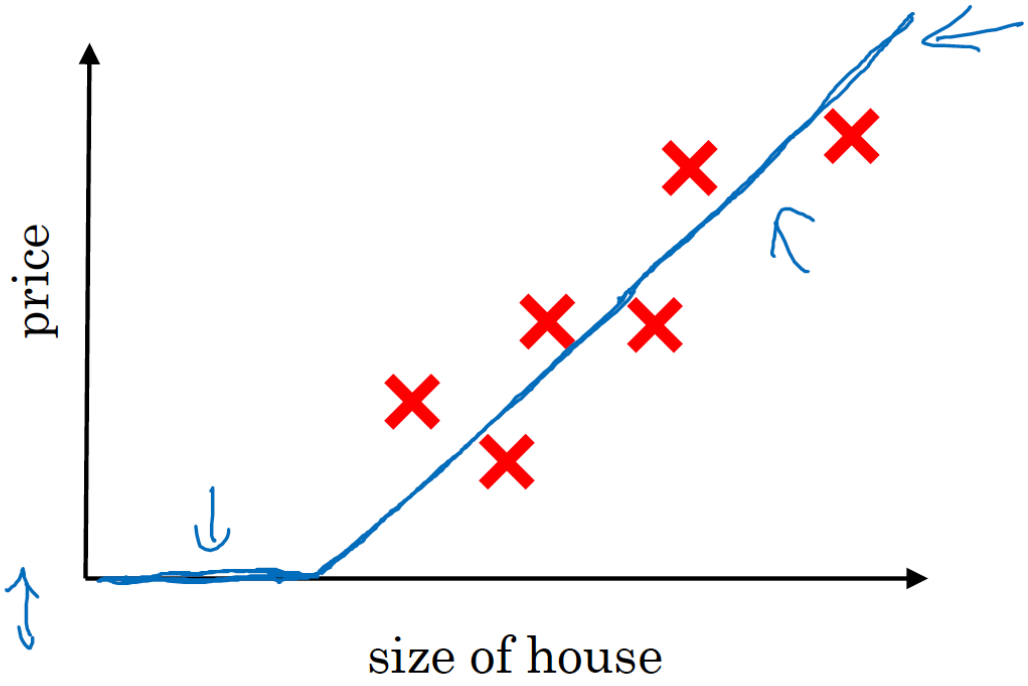
Agenda

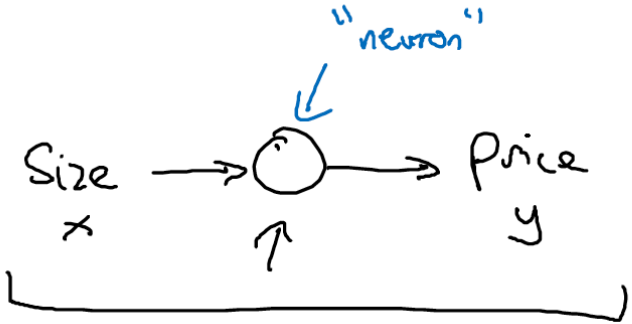
- Intuition of Neural Network for Classification Task
- Neural Network and Deep NN Text Classification
- Specific DNN Architecture: CNN and AutoEncoder
- From AutoEncoder to

Classical Machine Learning Techniques for NLP

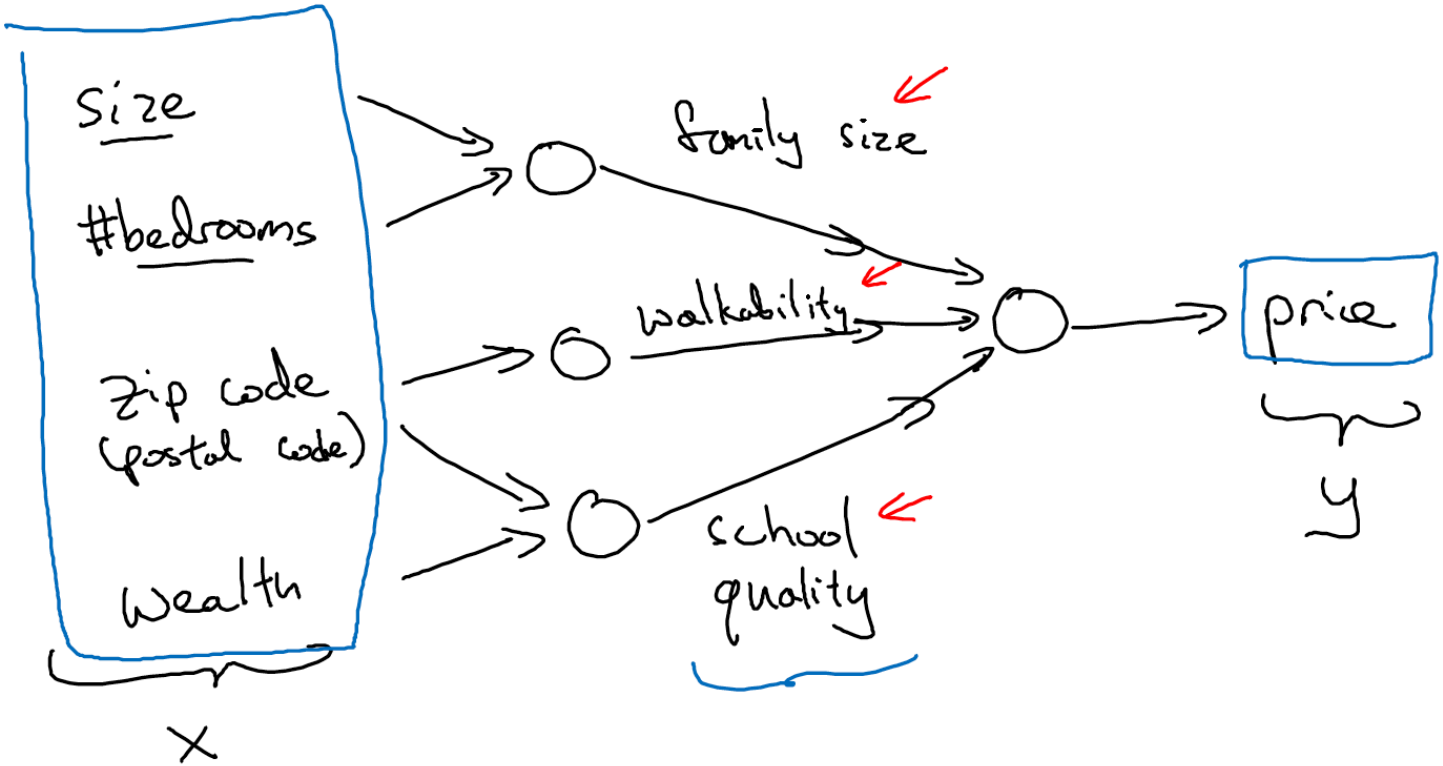
- Machine Learning Task
- The tf.idf weights
- Neural-based approach

Housing Price Prediction

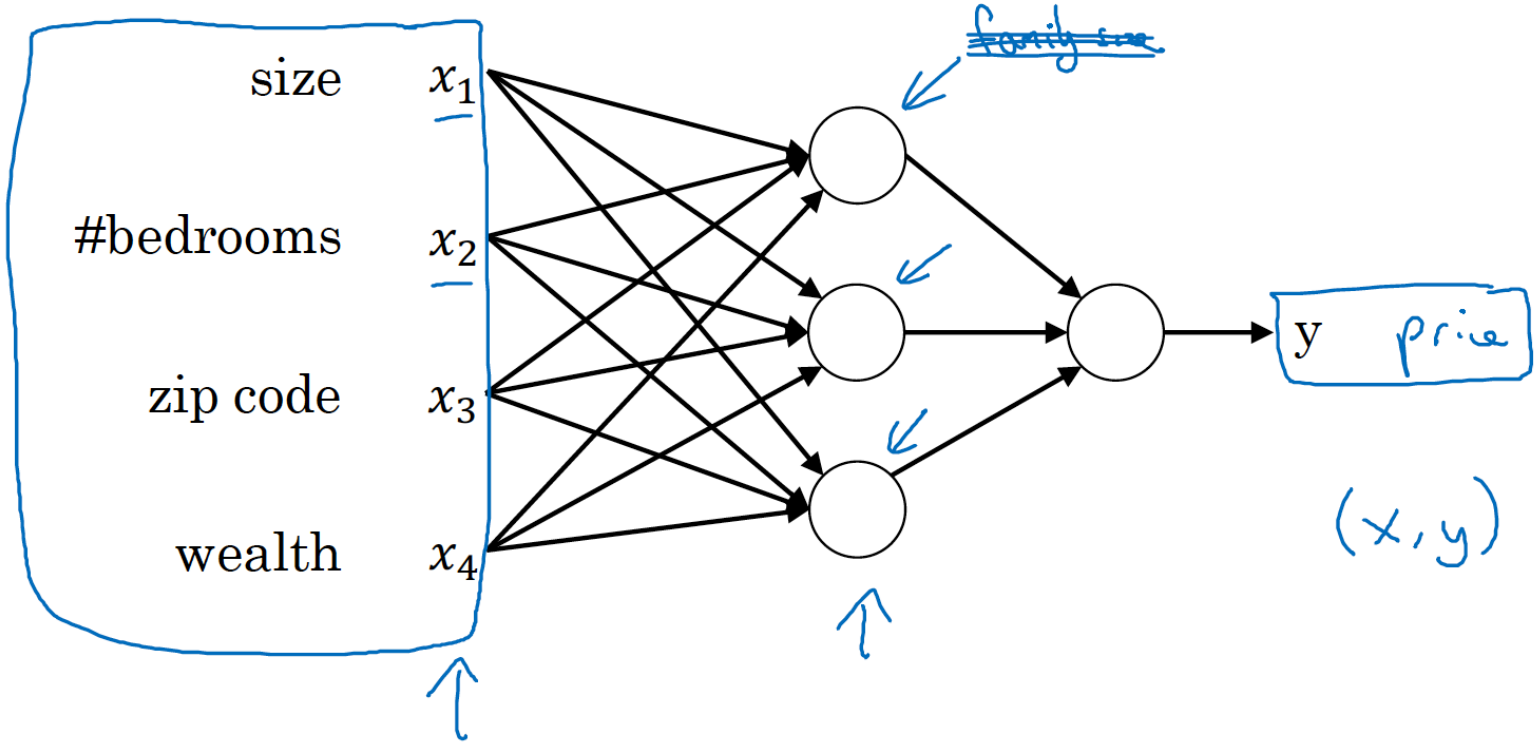




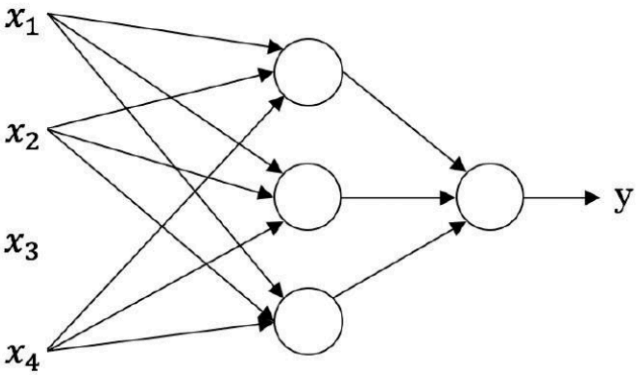
Housing Price Prediction



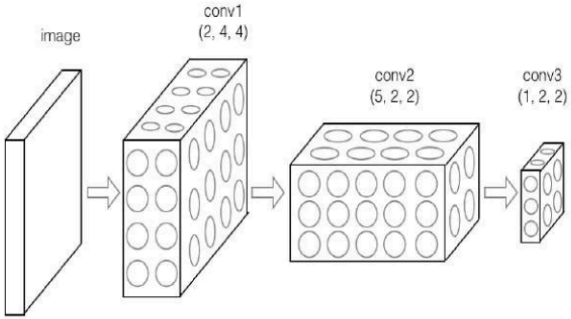
Housing Price Prediction



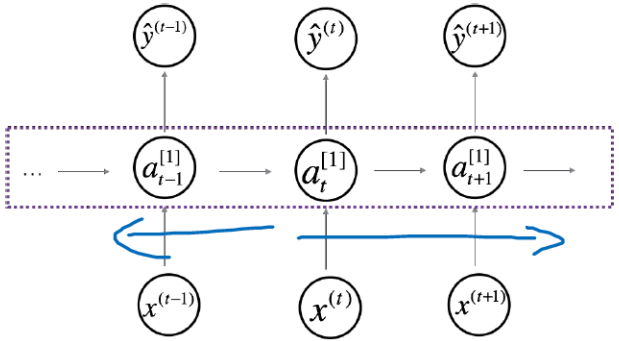
Neural Network examples



Standard NN

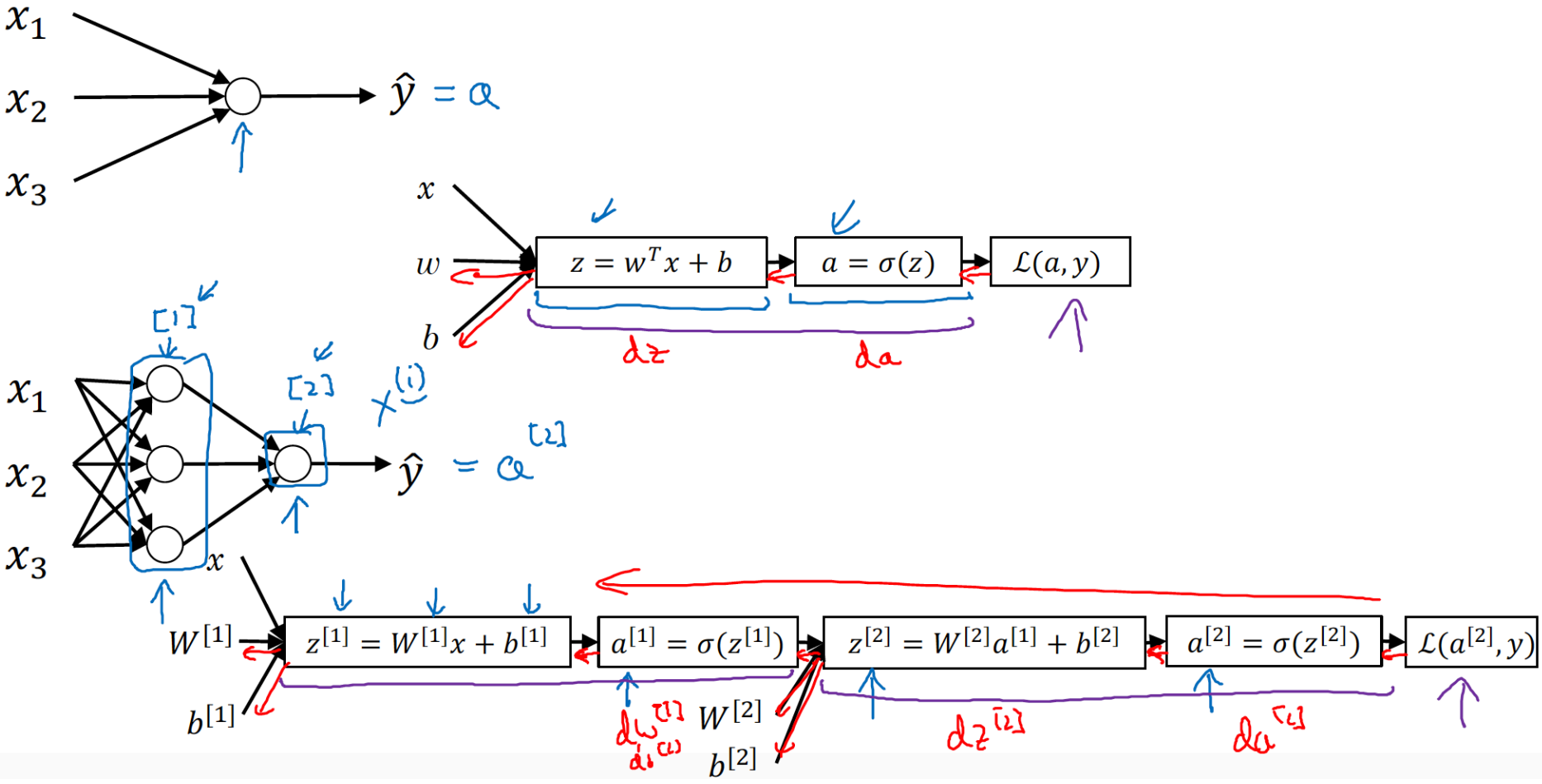


Convolutional NN

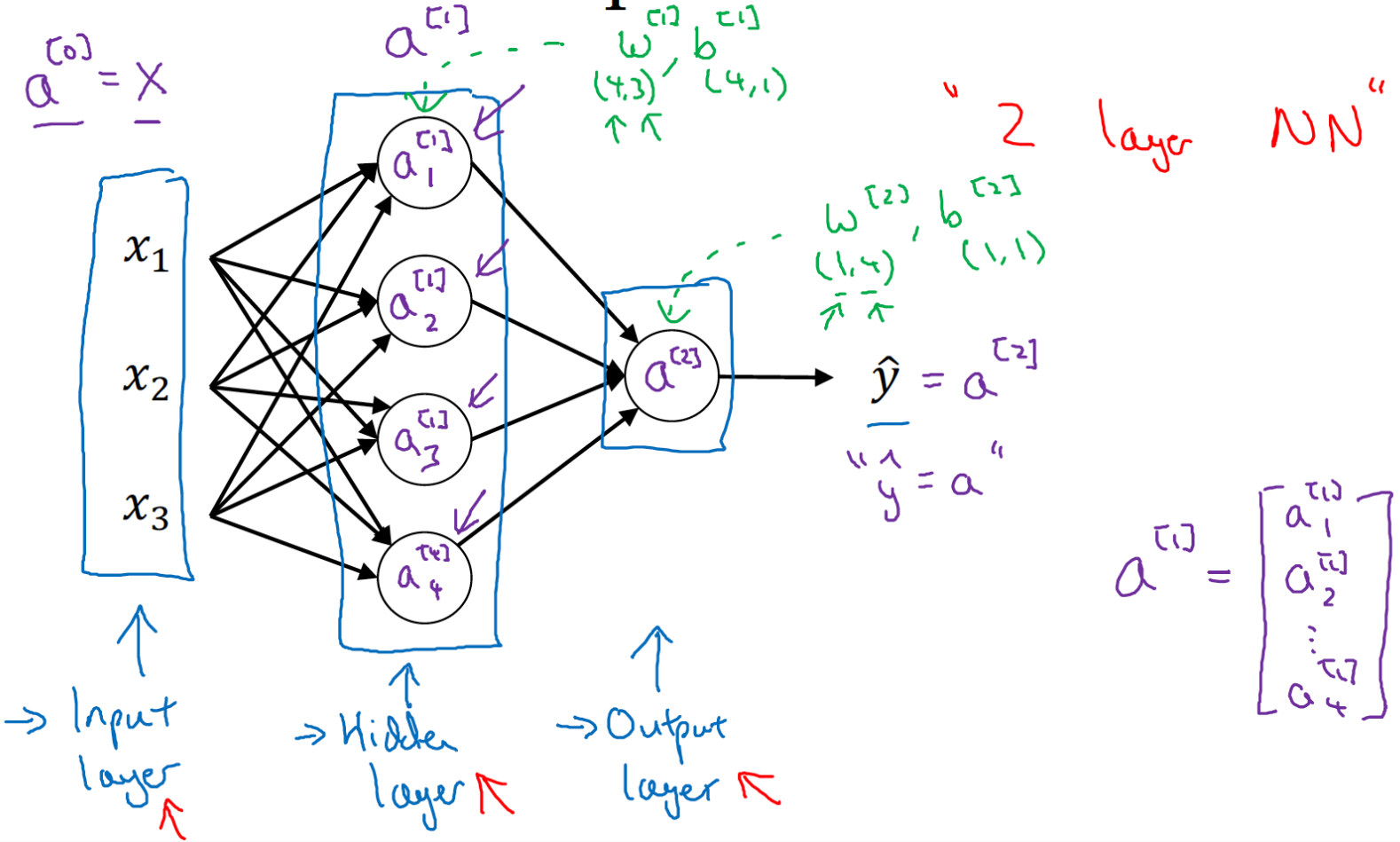


Recurrent NN

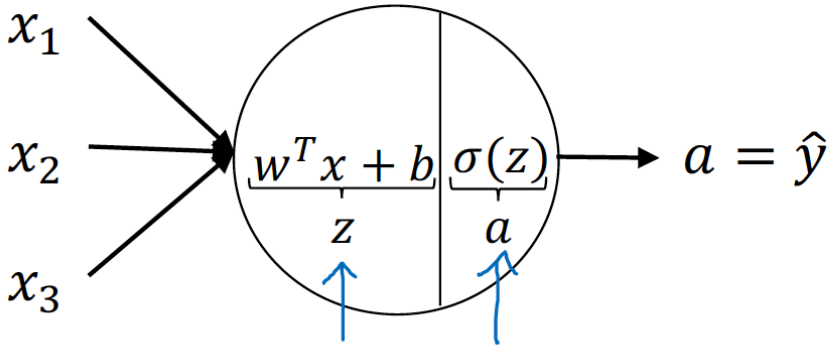
What is a Neural Network?

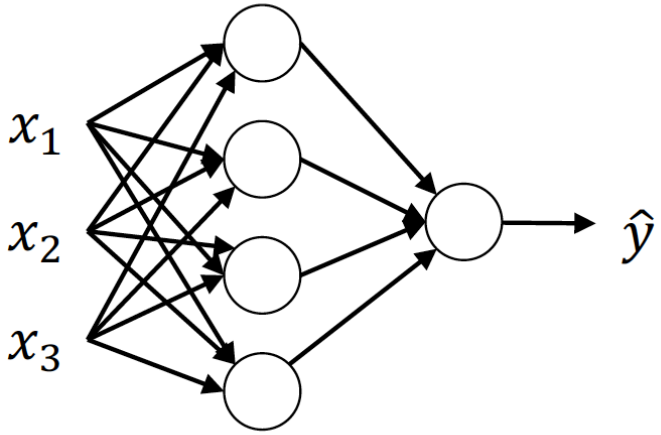


Neural Network Representation



Neural Network Representation

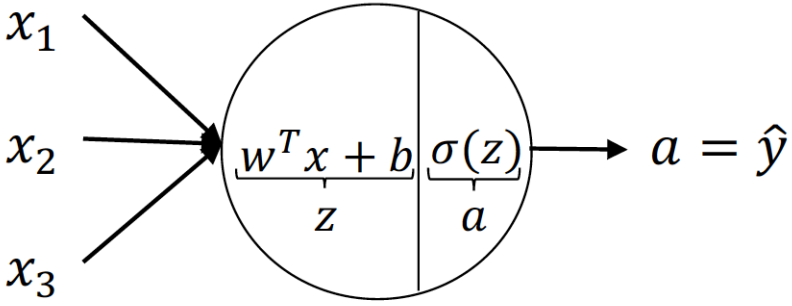




$$z = w^T x + b$$

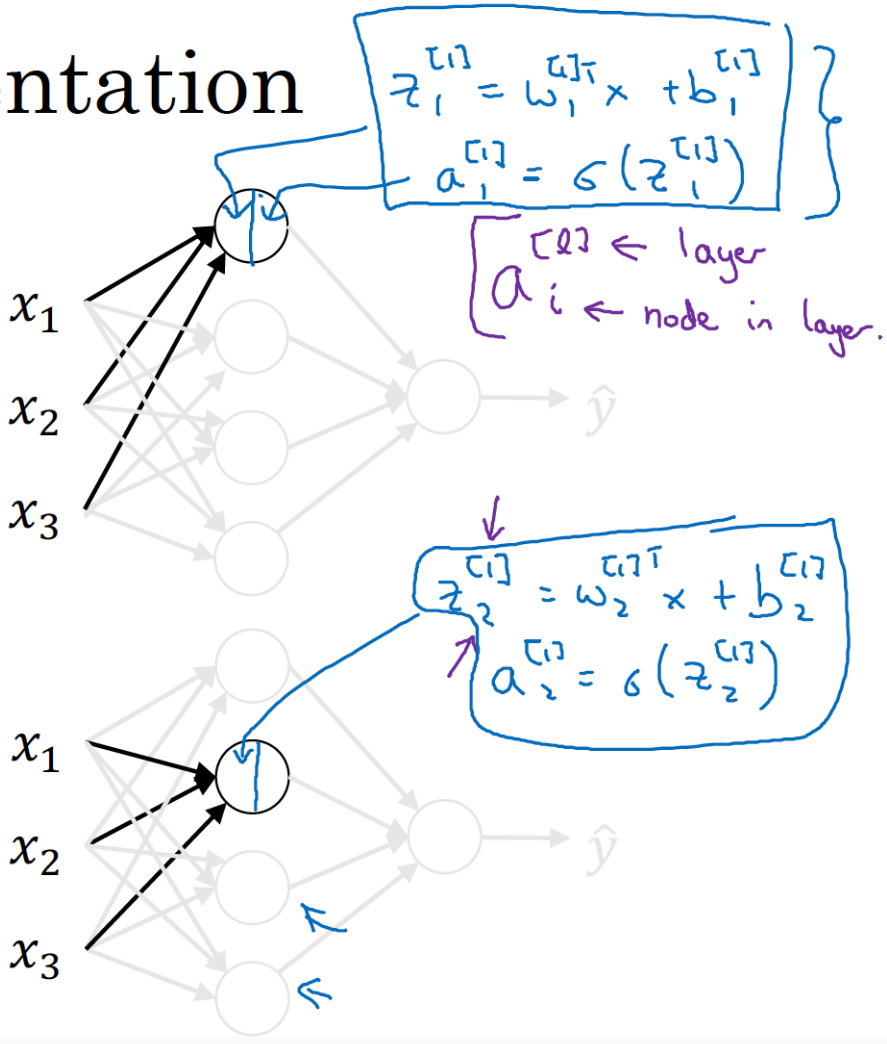
$$a = \sigma(z)$$

Neural Network Representation

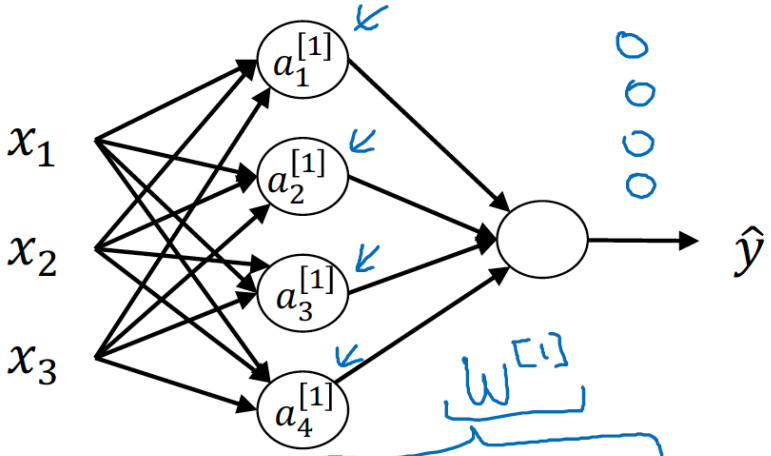


$$z = w^T x + b$$

$$a = \sigma(z)$$



Neural Network Representation



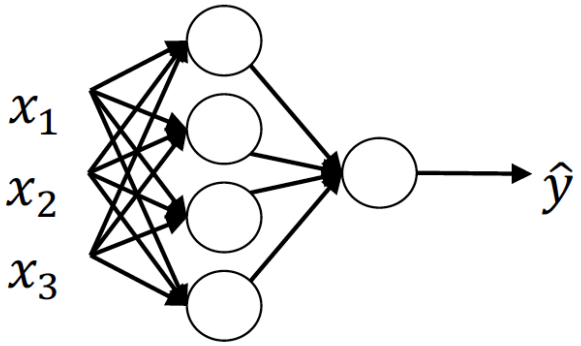
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$W^{[1]}$ $(4,3)$ $\quad b^{[1]}$ $(4,1)$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Vectorizing across multiple examples



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

$X \rightarrow a^{[2]} = \hat{y}$

$x^{(1)} \rightarrow a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)} \rightarrow a^{[2](2)} = \hat{y}^{(2)}$

$\vdots$

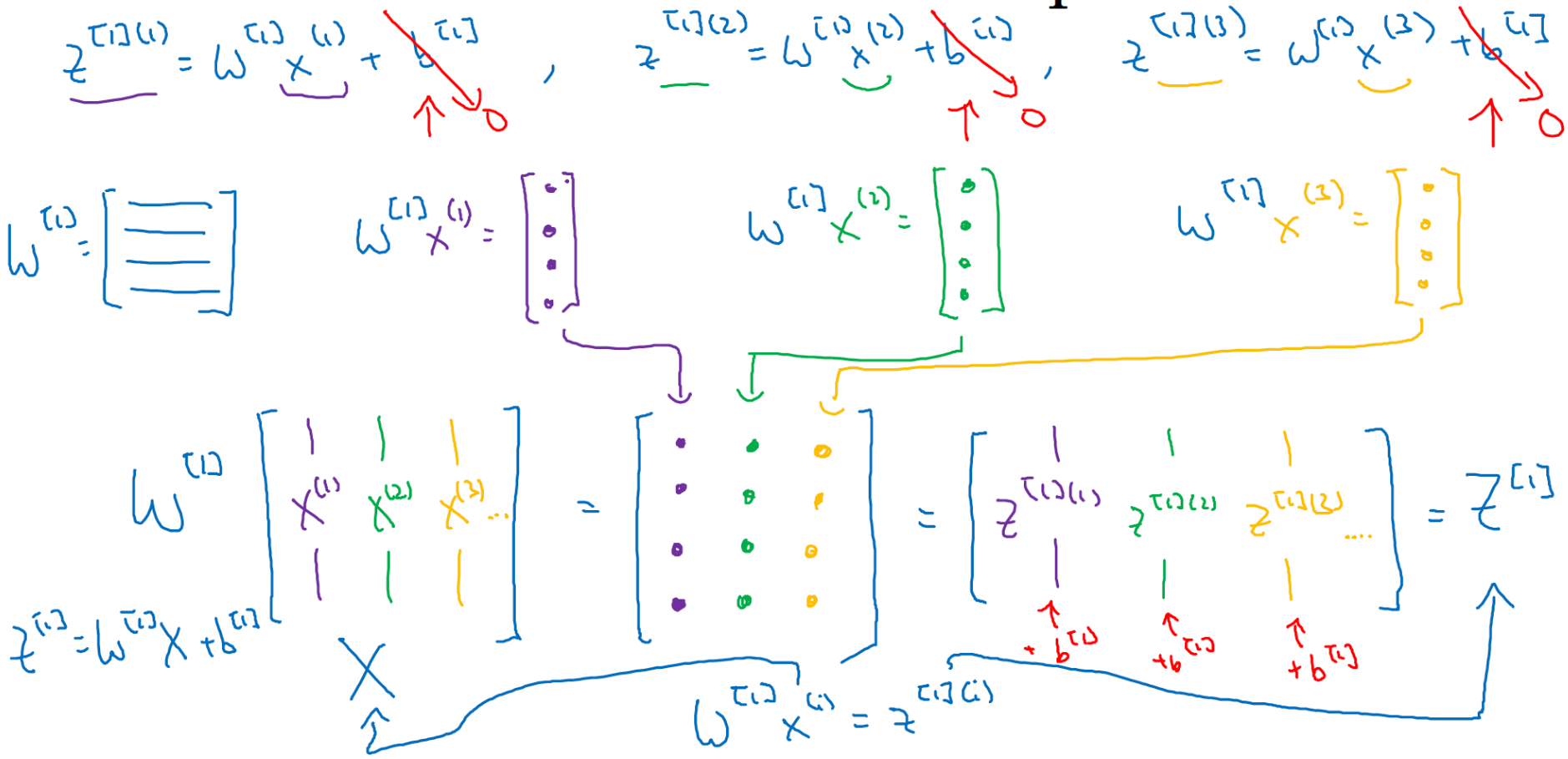
$x^{(m)} \rightarrow a^{[2](m)} = \hat{y}^{(m)}$

$a^{[2](i)}$ ← example i, layer 2

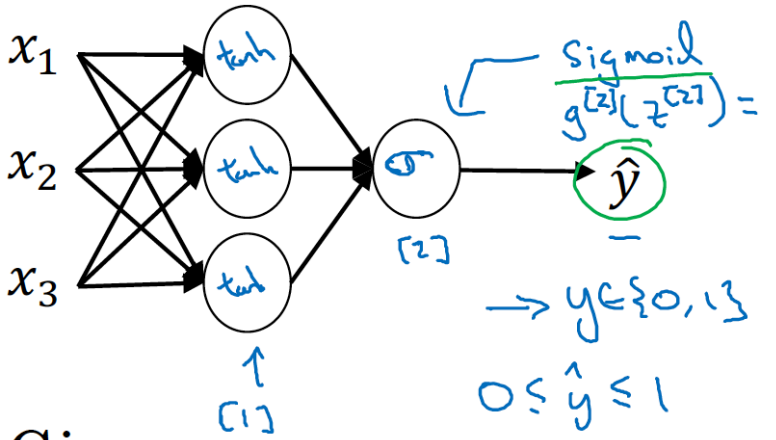
for $i = 1$ to m,

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

Justification for vectorized implementation



Activation functions



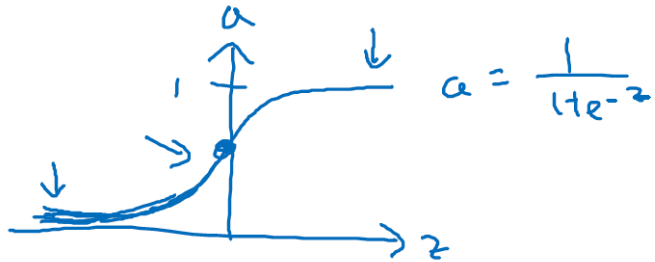
$g^{[1]}(z^{[1]}) = \tanh(z^{[1]})$

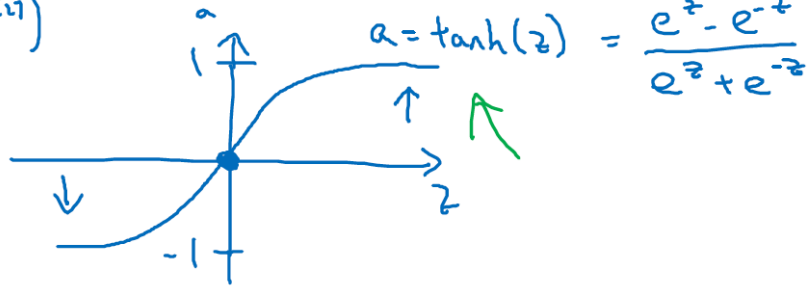
Sigmoid
$g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$

$\rightarrow y \in \{0, 1\}$

$0 \le \hat{y} \le 1$

$-1 \quad 1$





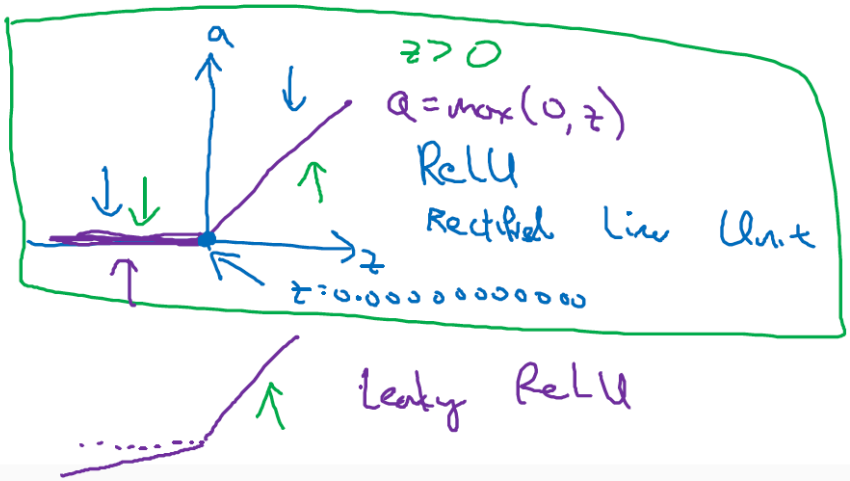
Given x:

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

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

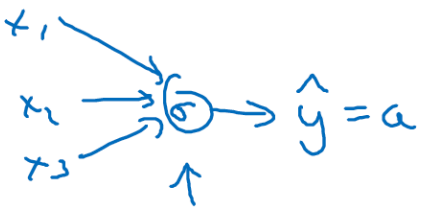
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

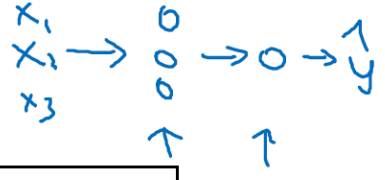
$$\rightarrow a^{[2]} = \sigma(z^{[2]}) \quad g^{[2]}(z^{[2]})$$

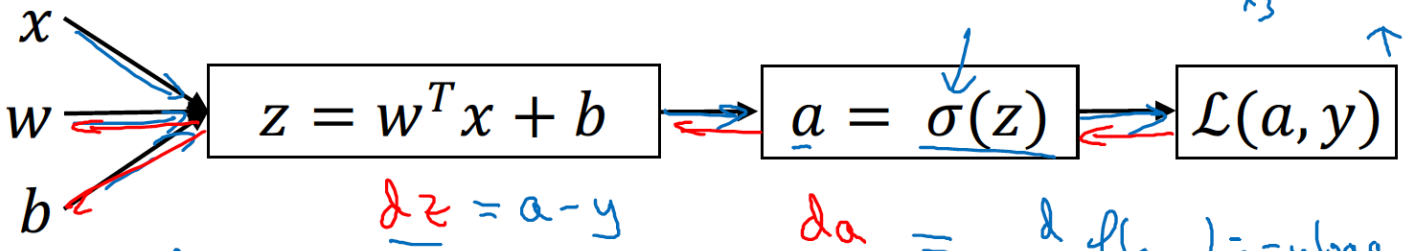


Computing gradients

Logistic regression







$dz = a - y$

$da = \frac{d}{da}\mathcal{L}(a,y) = -y\log a - (1-y)\log(1-a)$

$= -\frac{y}{a} + \frac{1-y}{1-a}$

$dw = dz \cdot x$

$db = dz$

$dz = da \cdot g'(z)$

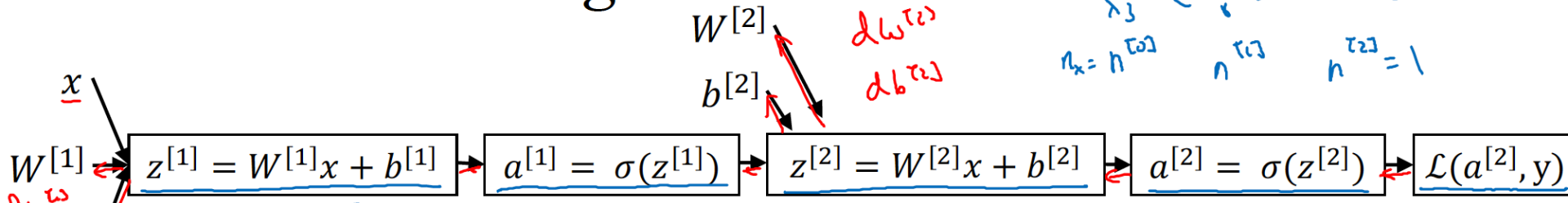
$g(z) = \sigma(z)$

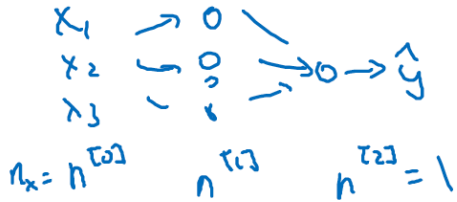
$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{da}{dz}$

"dz" = "da"

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

Neural network gradients





$dW^{[2]}$ $db^{[2]}$ $dW^{[1]}$ $db^{[1]}$

$dz^{[1]} = W^{[2]T} dz^{[2]}$ $da^{[1]} = \ldots$ $dz^{[2]} = a^{[2]} - y$ $da^{[2]}$

$* g^{[1]'}(z^{[1]})$ ← element-wise product

$dW^{[2]} = dz^{[2]} a^{[1]T}$ $db^{[2]} = dz^{[2]}$ → "$dw = dz \cdot x$"

$W^{[2]}$ $(n^{[2]}, n^{[1]})$

$z^{[2]}, dz^{[2]}$ $(n^{[2]}, 1)$ $-(1,1)$

$z^{[1]}, dz^{[1]}$ $(n^{[1]}, 1)$

foo $W^{[1]}$ $dW^{[1]} = dz^{[1]} \cdot x^T$

dfoo $dW^{[2]}$ $db^{[1]} = dz^{[1]}$ $a^{[0]T}$

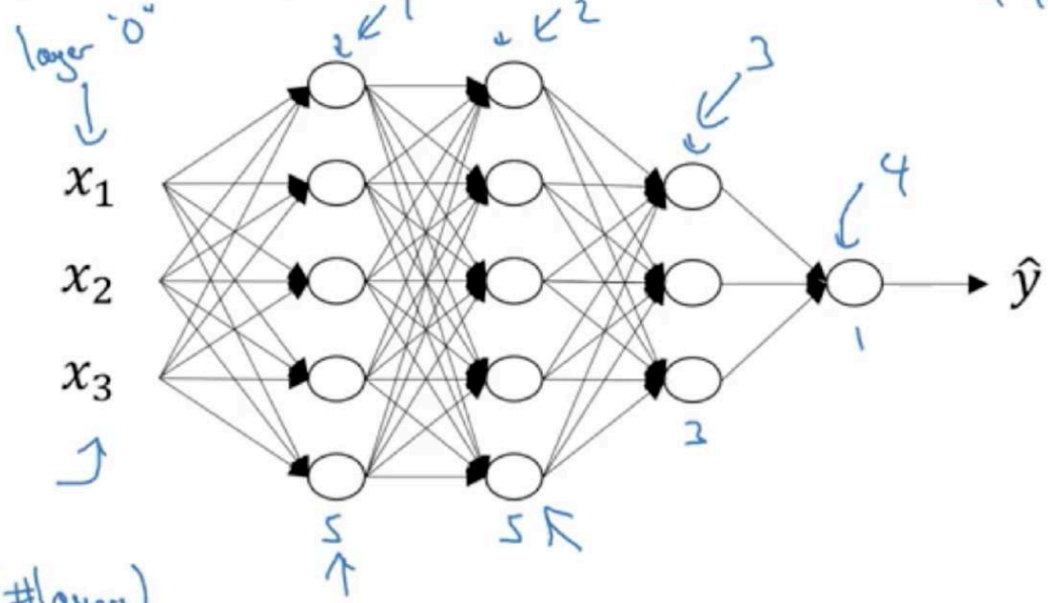
$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]})(n^{[2]}, 1) * (n^{[1]}, 1)$

Andrew Ng

Deep neural network notation

4 layer NN



$L = 4$ (#layers)

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

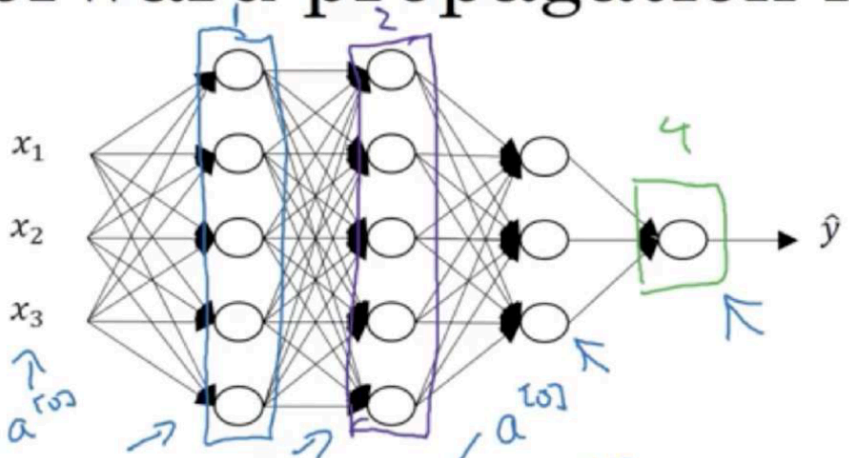
$n^{[1]} = 5$, $n^{[2]} = 5$, $n^{[3]} = 3$, $n^{[4]} = n^{[L]} = 1$

$n^{[0]} = n_x = 3$

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

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

Forward propagation in a deep network



$A^{[0]} = X$

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

$X: \; z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

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

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

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

$z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}, \; a^{[4]} = g^{[4]}(z^{[4]}) = \hat{y}$

Vectorized:

$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[1]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\hat{Y} = g(Z^{[4]}) = A^{[4]}$$

$\rightarrow X = A^{[0]}$

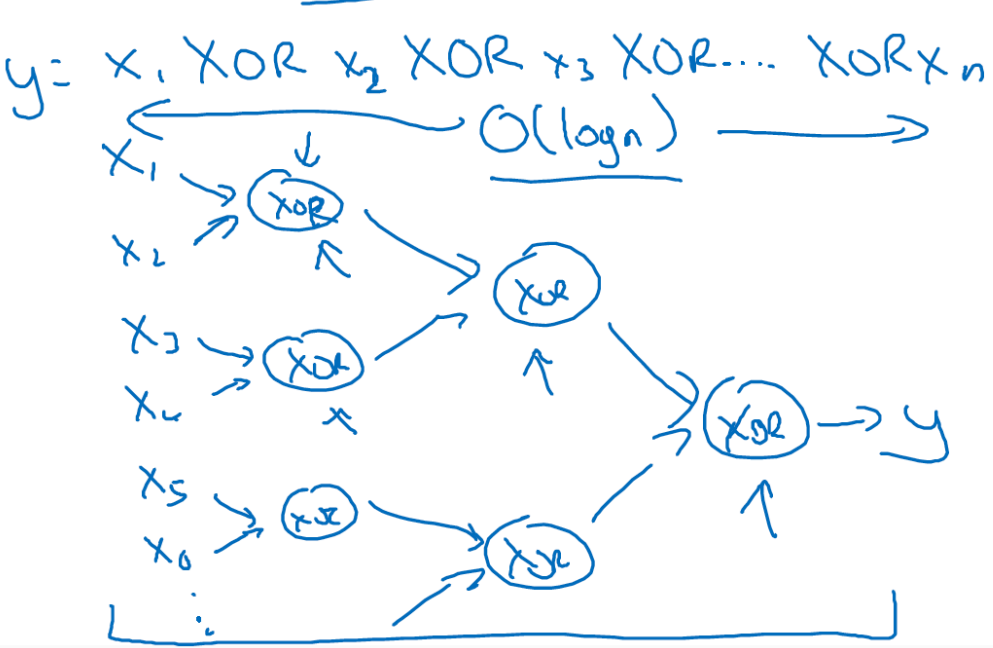
for $l = 1 \ldots 4$

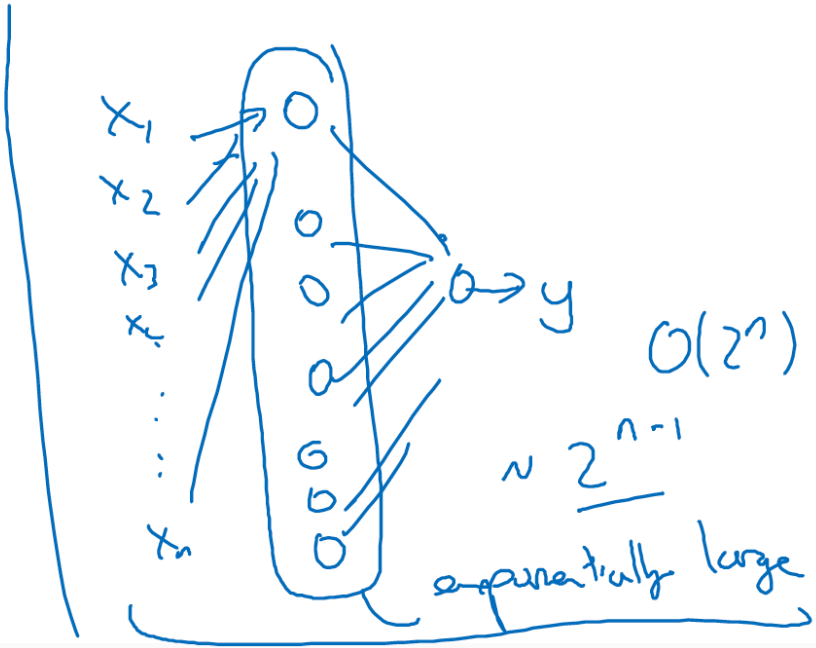
$\begin{bmatrix} z^{[1](1)} & z^{[1](2)} & \cdots \\ | & | & | \end{bmatrix}$

Andrew Ng

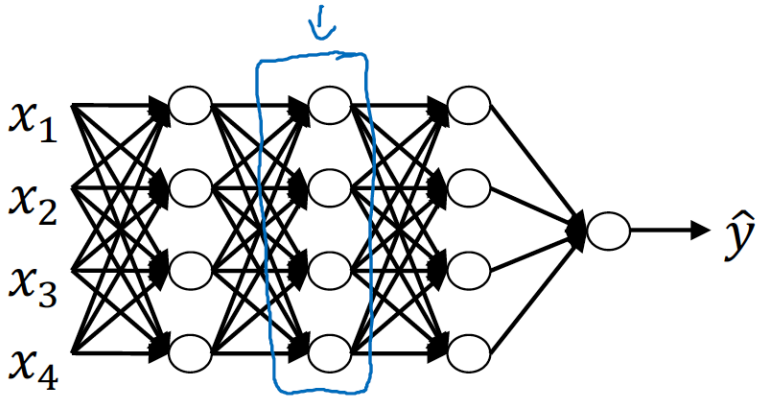
Circuit theory and deep learning

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





Forward and backward functions



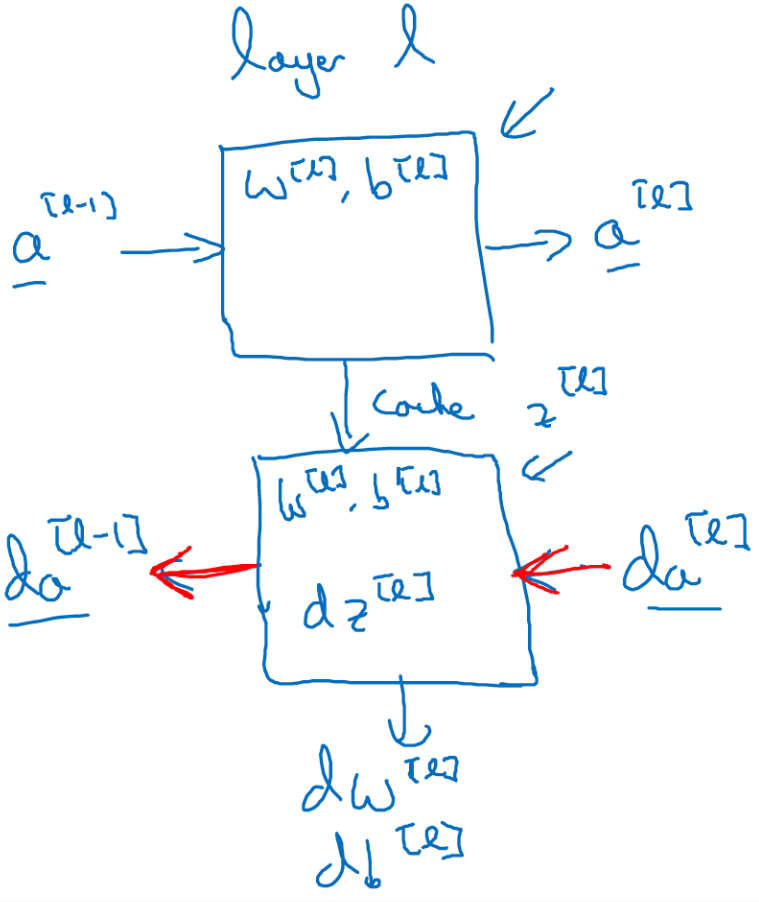
Layer l: $W^{[l]}, b^{[l]}$

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

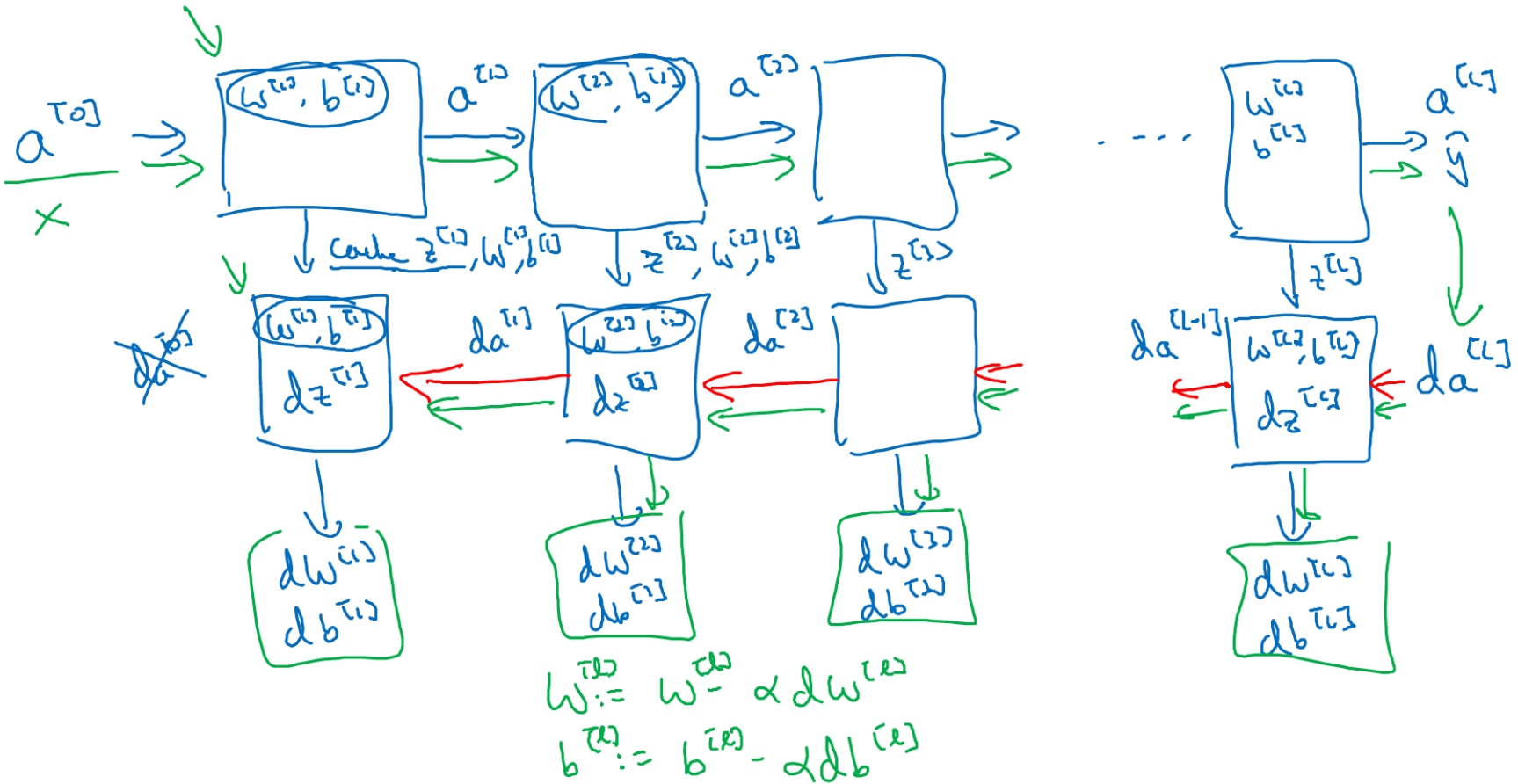
$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ cache $z^{[l]}$

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

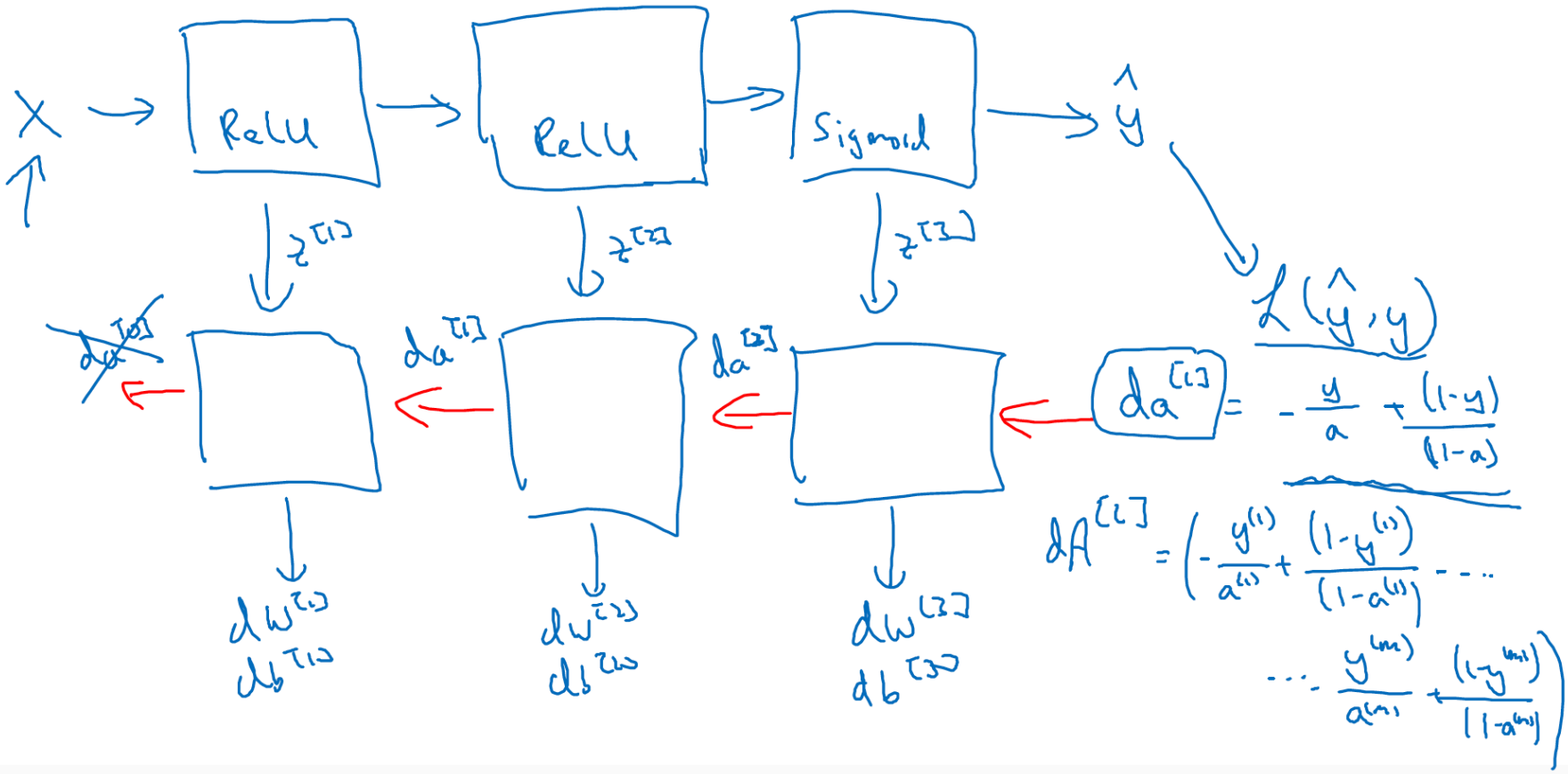
→ Backward: Input $da^{[l]}$, cache ($z^{[l]}$), output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$



Forward and backward functions



Summary


X
ReLU
ReLU
Sigmoid
ŷ
$z^{[1]}$
$z^{[2]}$
$z^{[3]}$
$\mathcal{L}(\hat{y}, y)$
$da^{[1]} = -\frac{y}{a} + \frac{(1-y)}{(1-a)}$
$da^{[0]}$
$da^{[1]}$
$da^{[2]}$
$dA^{[1]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \cdots \right.$
$\left. \cdots \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})} \right)$
$dW^{[1]}$
$db^{[1]}$
$dW^{[2]}$
$db^{[2]}$
$dW^{[3]}$
$db^{[3]}$

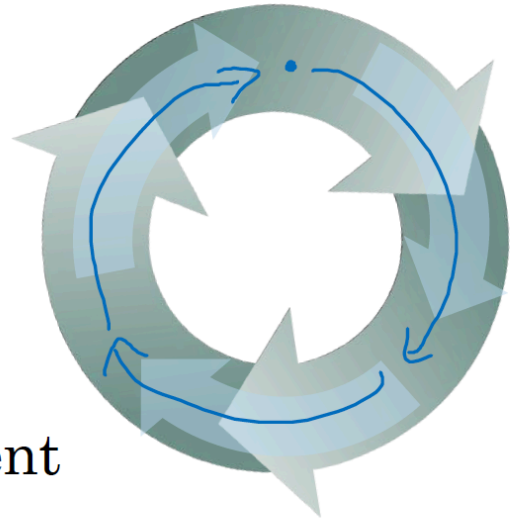
What are hyperparameters?

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

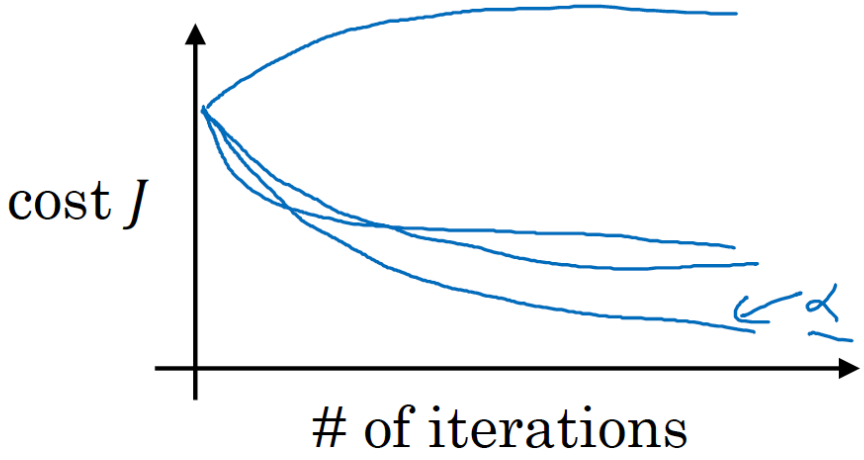
Hyperparameters:
- learning rate α
- #iterations
- # hidden layers L
- # hidden units $n^{[1]}, n^{[2]}, \ldots$
- choice of activation function

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

Applied deep learning is a very empirical process

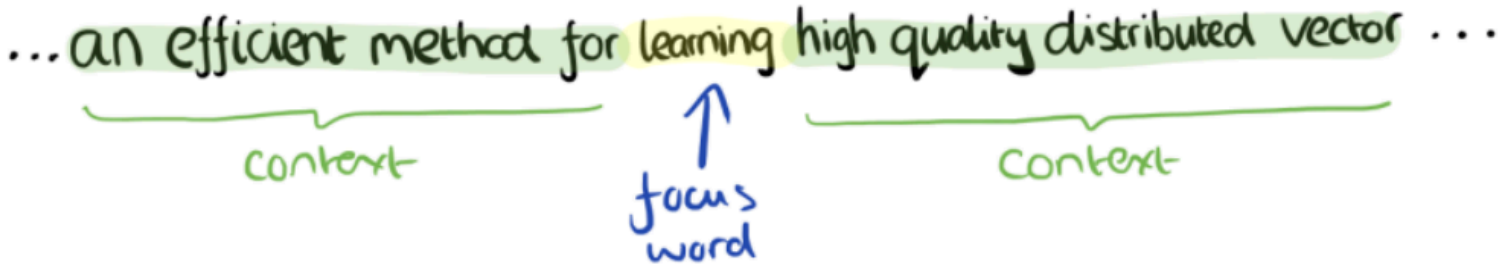


$\alpha = 0.04$ ~~0.04~~ 0.05



Vision, Speech, NLP, Ad, Search, Recommendation.

Word Embedding

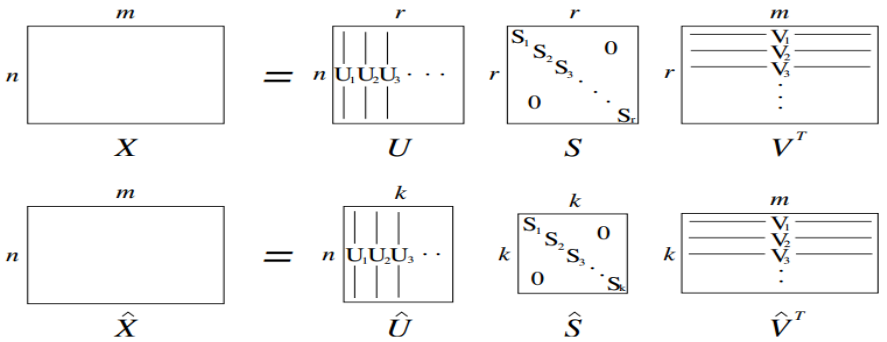

...an efficient method for learning high quality distributed vector ...
context
focus word
context

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 = size of vocabulary



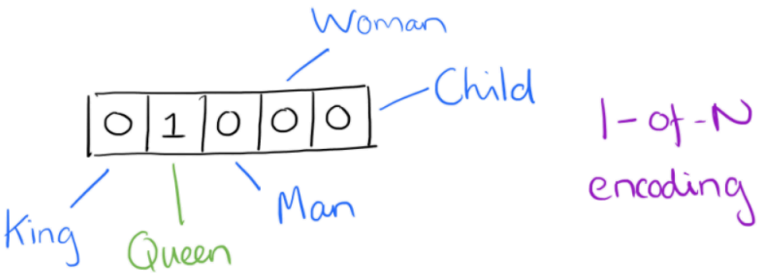
Problems with SVD

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

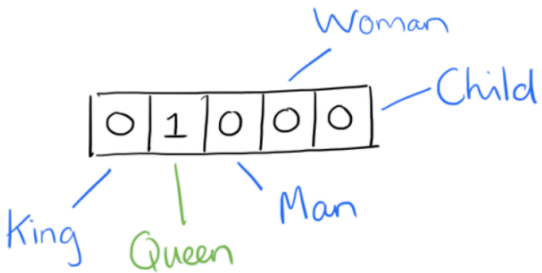
word2vec Approach to represent the meaning of word

- 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

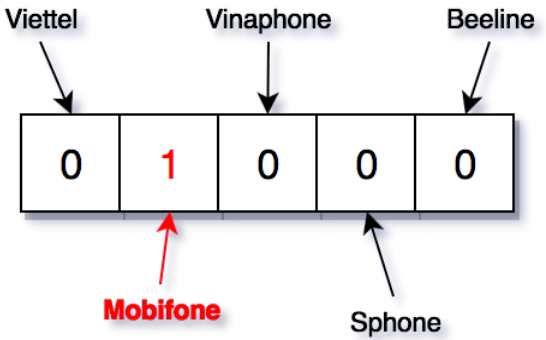


Word2Vec



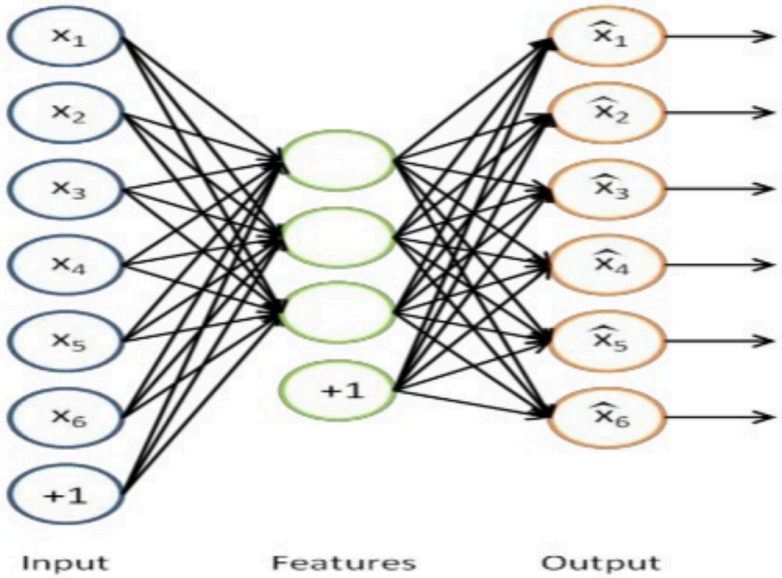
1-of-N

Word2Vec

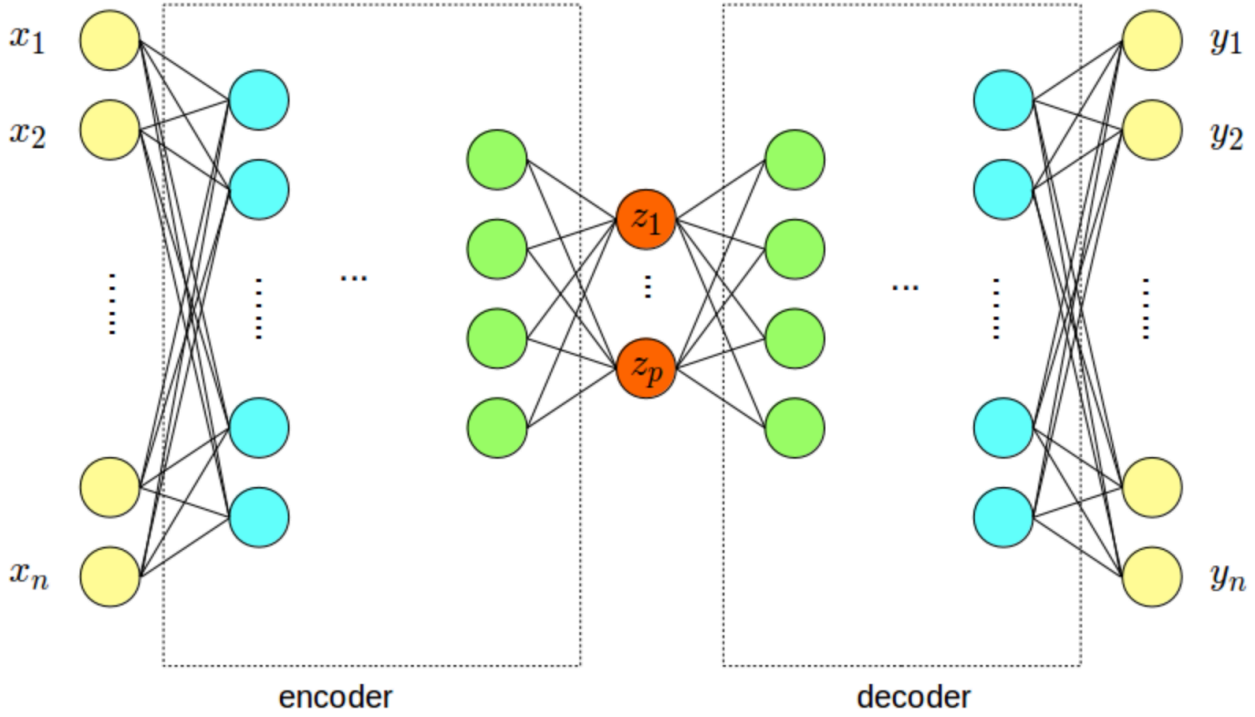


Viettel	Mobifone	VinaPhone	Sphone	Beeline
0.12	0.41	0.25	0.66	0.34
0.02	0.63	0.33	0.49	0.52
0.01	0.44	0.21	0.23	0.37

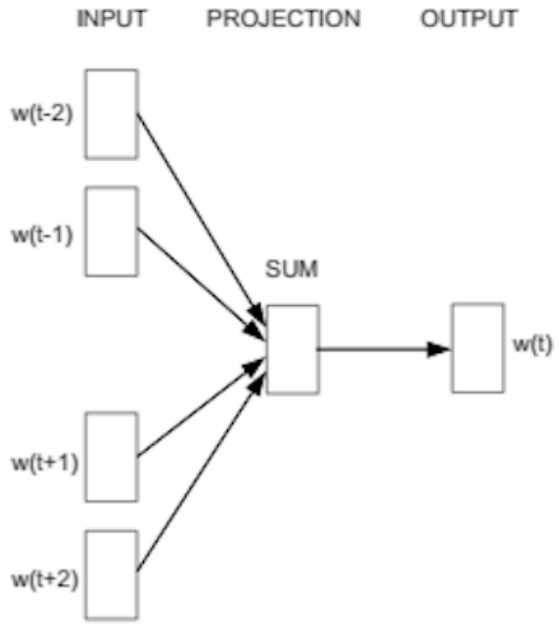
Auto-Encoder



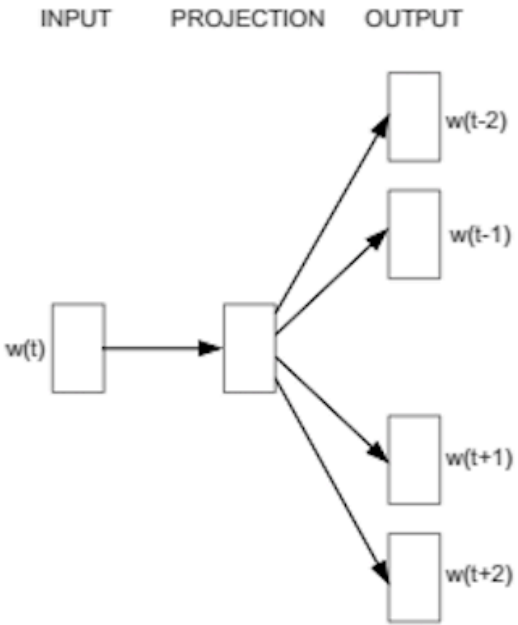
Stacked Auto-Encoder



Word2vec



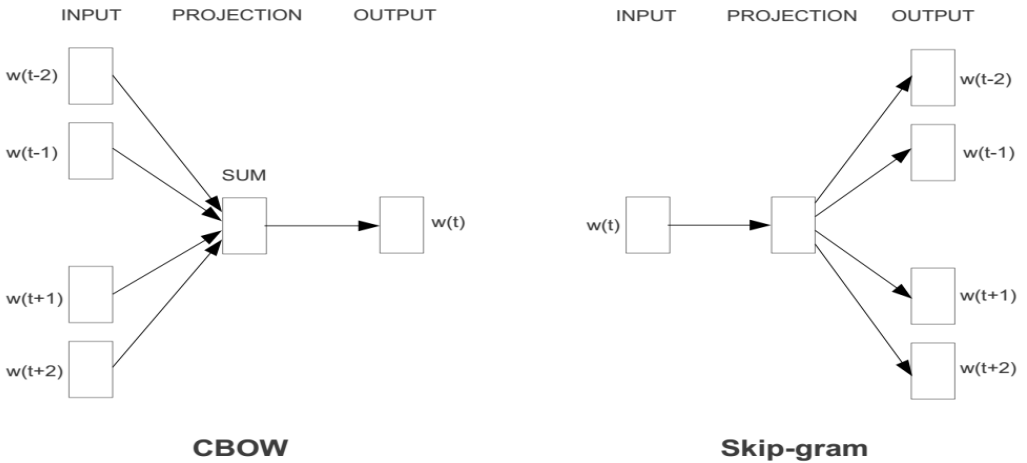
CBOW



Skip-gram

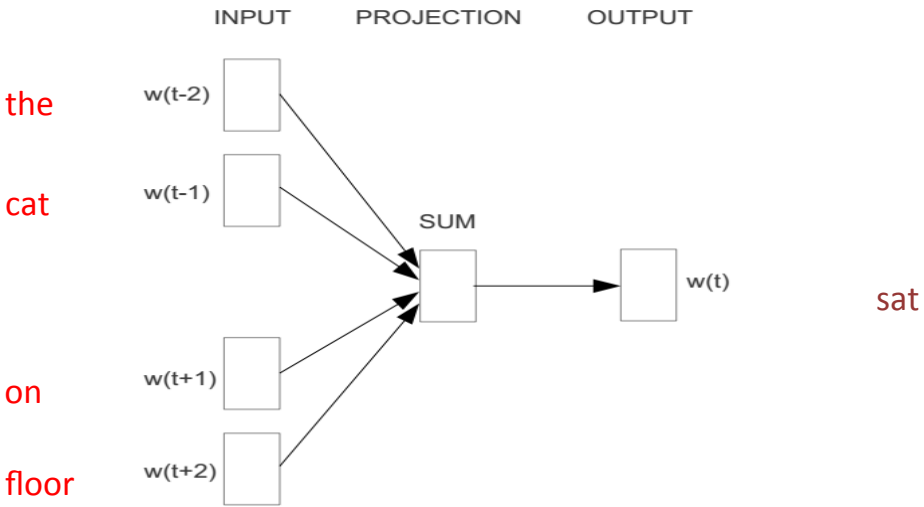
Represent the meaning of word – word2vec

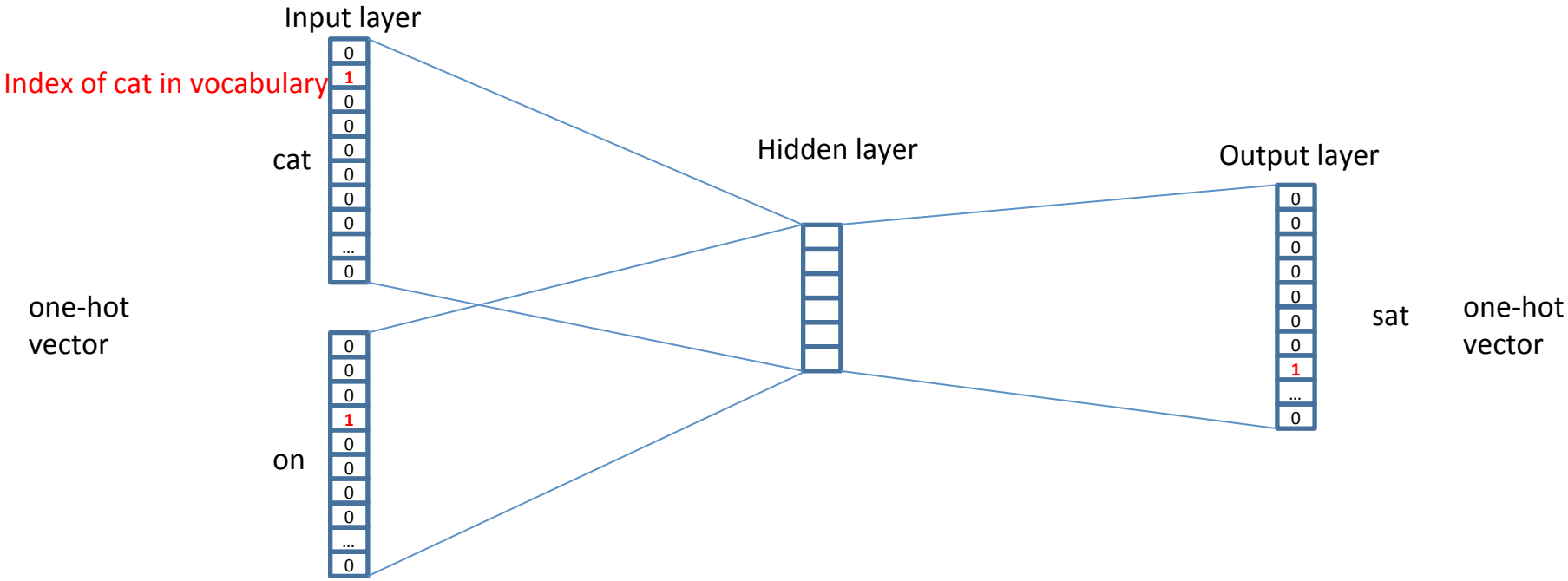
- 2 basic neural network models:
  - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
  - Skip-gram (SG): use a word to predict the surrounding ones in window.

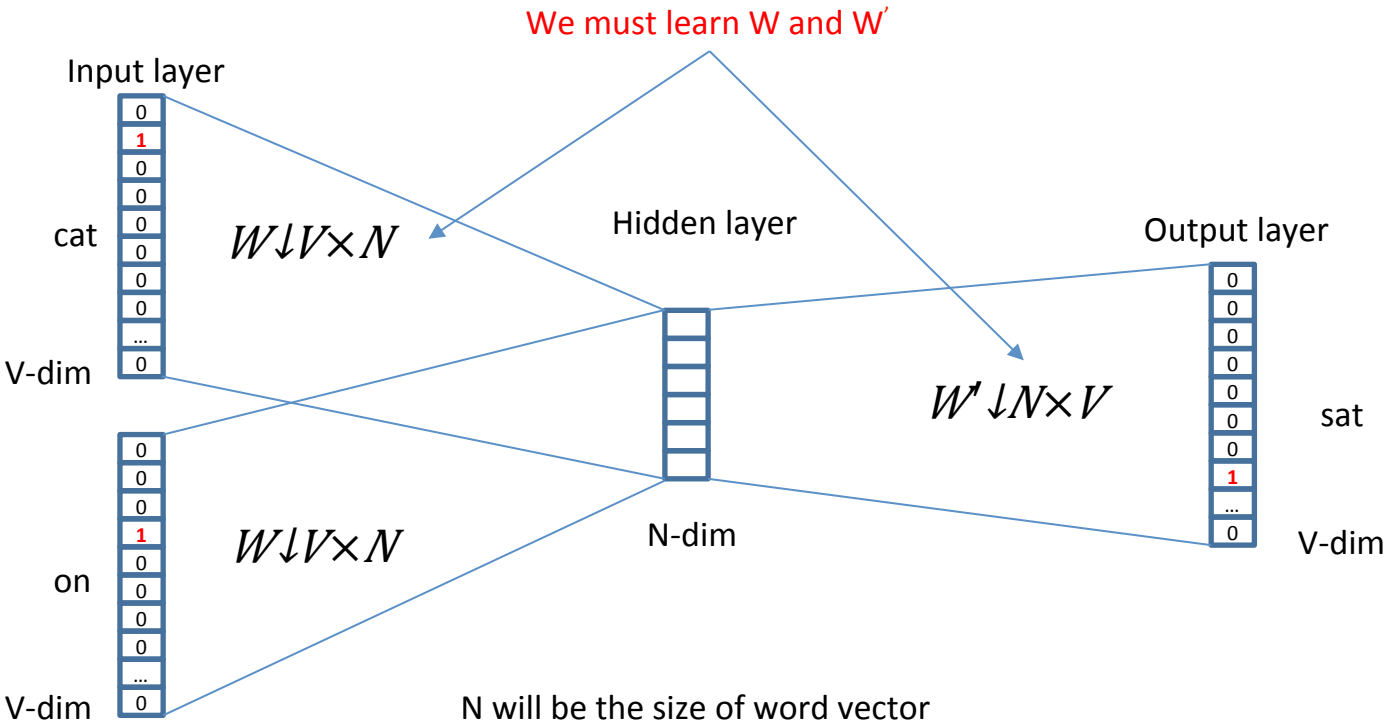


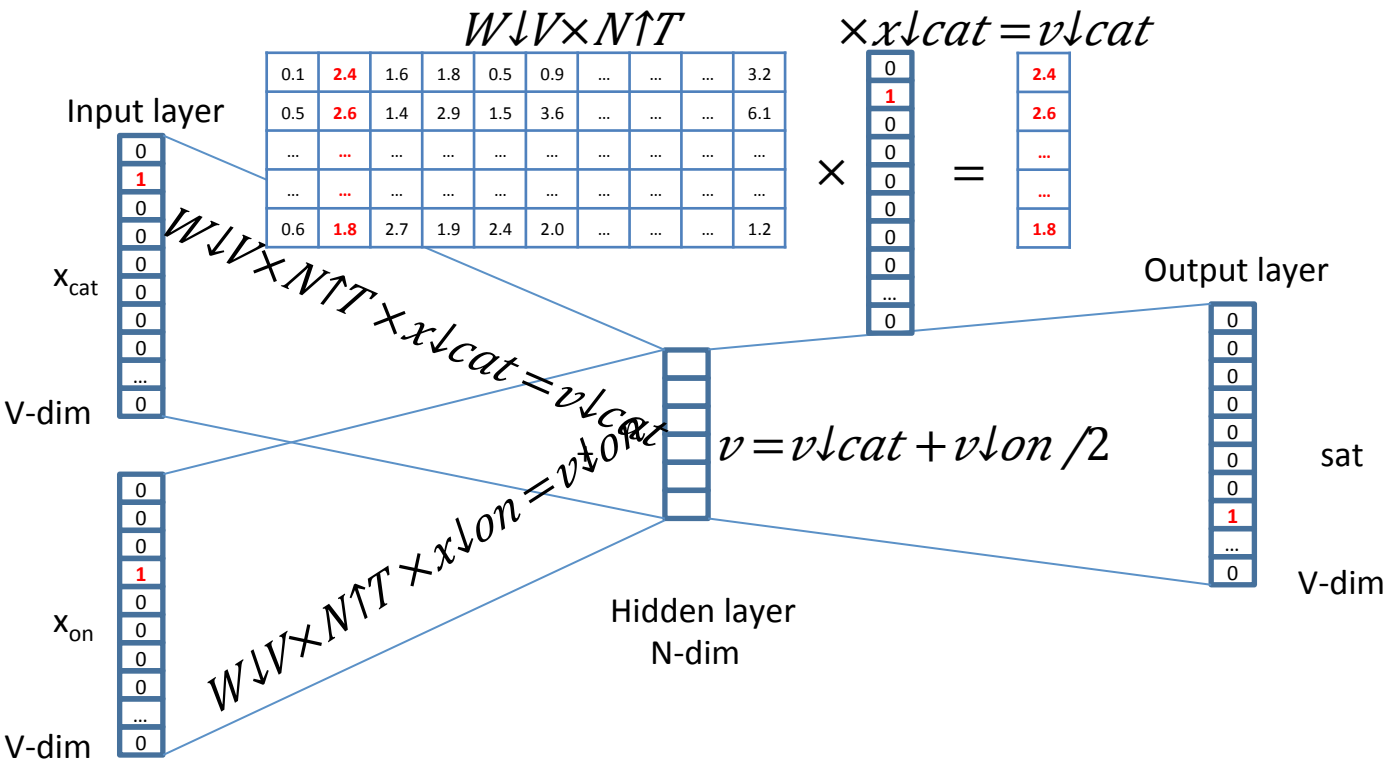
Word2vec – Continuous Bag of Word

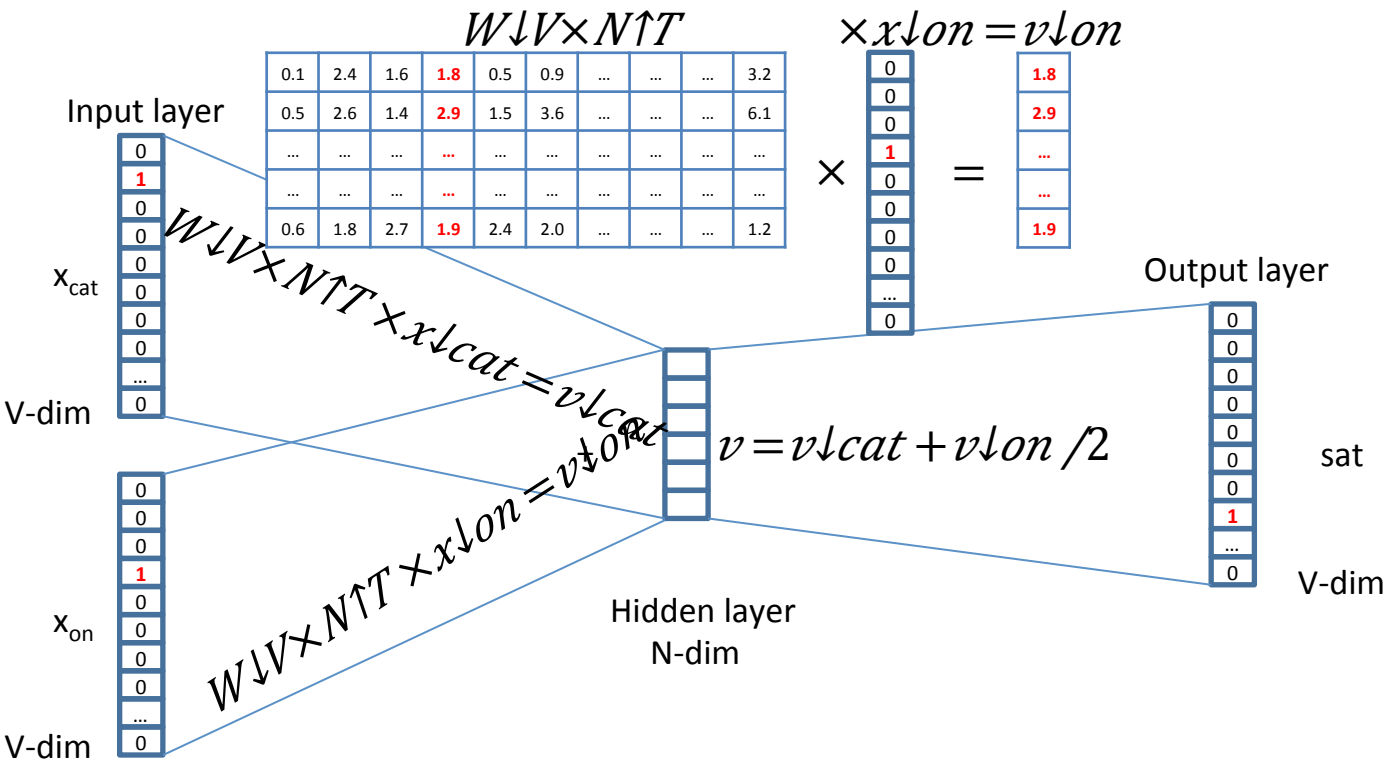
- E.g. “The cat sat on floor”
  - Window size = 2

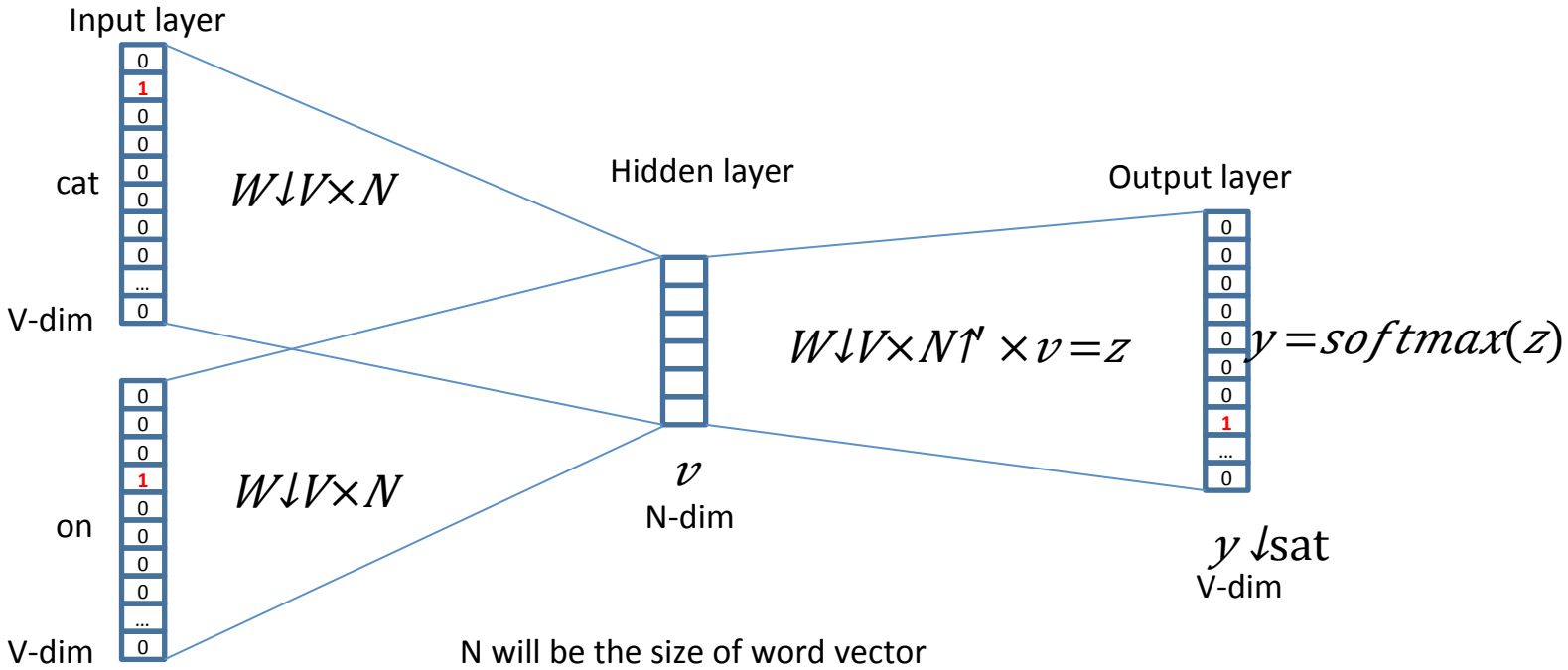


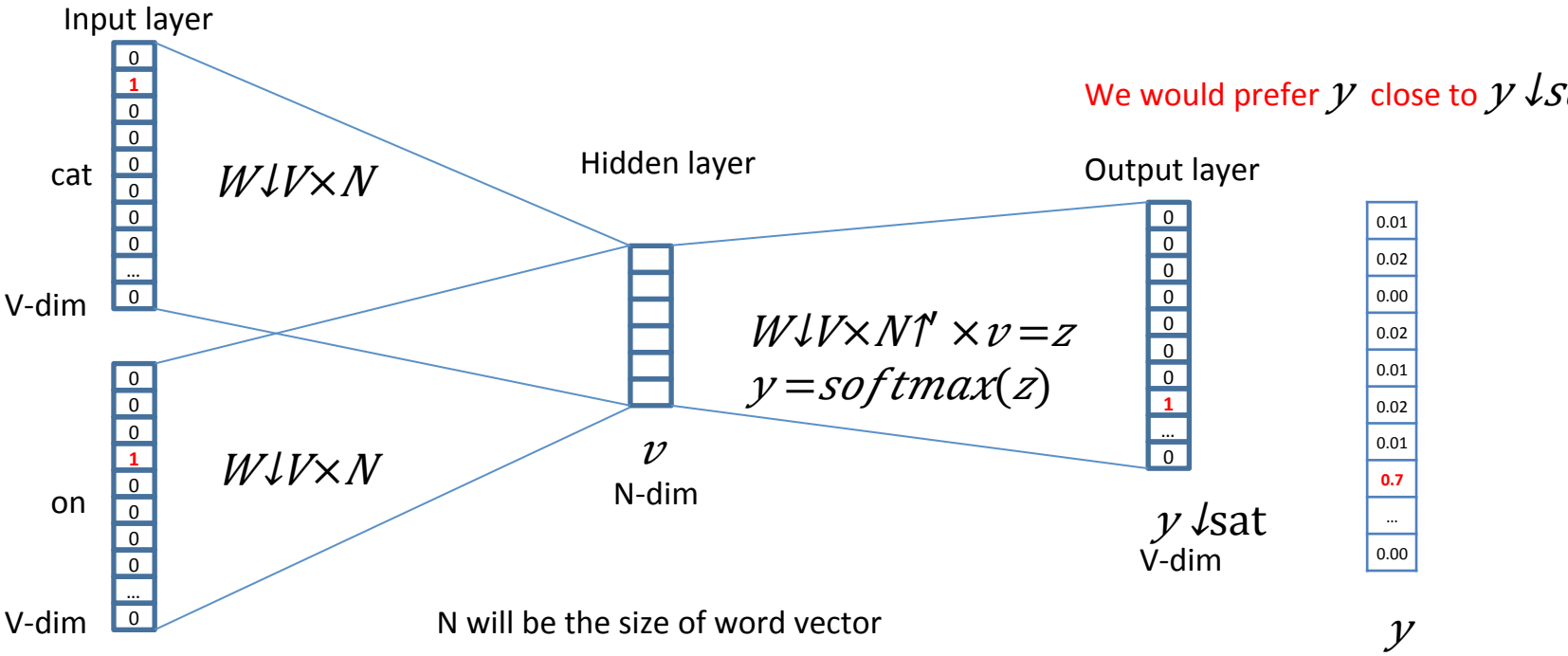

Input layer
Index of cat in vocabulary
cat
one-hot vector
on
Hidden layer
Output layer
sat
one-hot vector

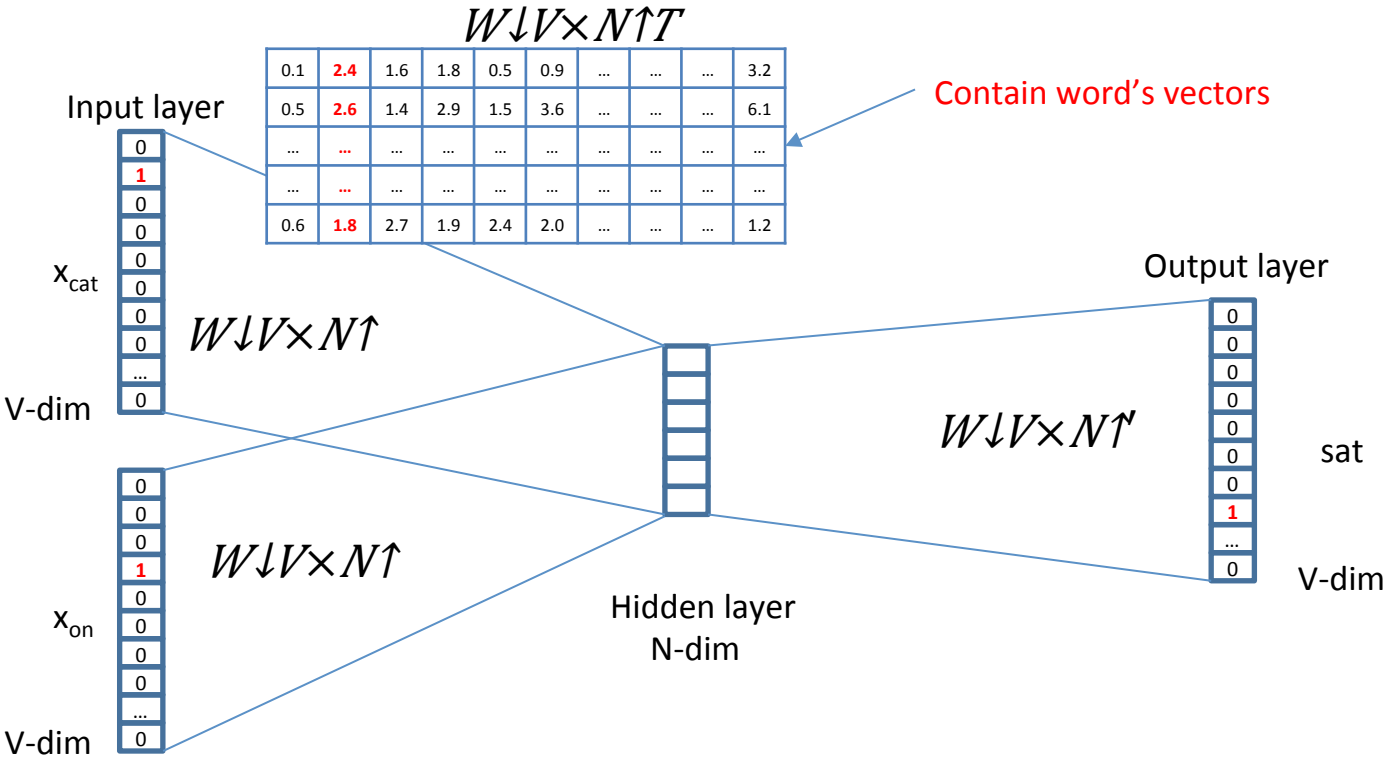

We must learn W and W'
Input layer
cat
V-dim
on
V-dim
$W \downarrow V \times N$
$W \downarrow V \times N$
Hidden layer
N-dim
$W' \downarrow N \times V$
Output layer
sat
V-dim
N will be the size of word vector


$W\downarrow V\times N\uparrow T$
$\times x\downarrow cat = v\downarrow cat$
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W\downarrow V\times N\uparrow T \times x\downarrow cat = v\downarrow cat$
$W\downarrow V\times N\uparrow T \times x\downarrow on = v\downarrow on$
$v = v\downarrow cat + v\downarrow on \,/2$
Hidden layer
N-dim
Output layer
sat
V-dim


$W\downarrow V\times N\uparrow T$
$\times x\downarrow on = v\downarrow on$
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W\downarrow V\times N\uparrow T \times x\downarrow cat = v\downarrow cat$
$W\downarrow V\times N\uparrow T \times x\downarrow on = v\downarrow on$
$v = v\downarrow cat + v\downarrow on /2$
Hidden layer
N-dim
Output layer
sat
V-dim


Input layer
cat
V-dim
on
V-dim
$W \downarrow V \times N$
$W \downarrow V \times N$
Hidden layer
v
N-dim
$W \downarrow V \times N \uparrow \times v = z$
Output layer
$y = softmax(z)$
$y \downarrow sat$
V-dim
N will be the size of word vector


Input layer
cat
V-dim
on
V-dim
$W\downarrow V\times N$
$W\downarrow V\times N$
Hidden layer
v
N-dim
$W\downarrow V\times N\uparrow \times v=z$
$y=softmax(z)$
Output layer
$y\downarrow sat$
V-dim
We would prefer y close to $y\downarrow s$
y
N will be the size of word vector



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

Some interesting results

Word Analogies

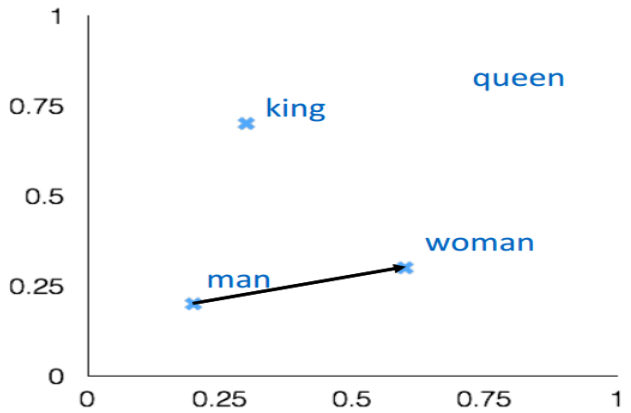
Test for linear relationships, examined by Mikolov et al. (2014)

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]
	queen	[0.70 0.80]



Word analogies