Big Data

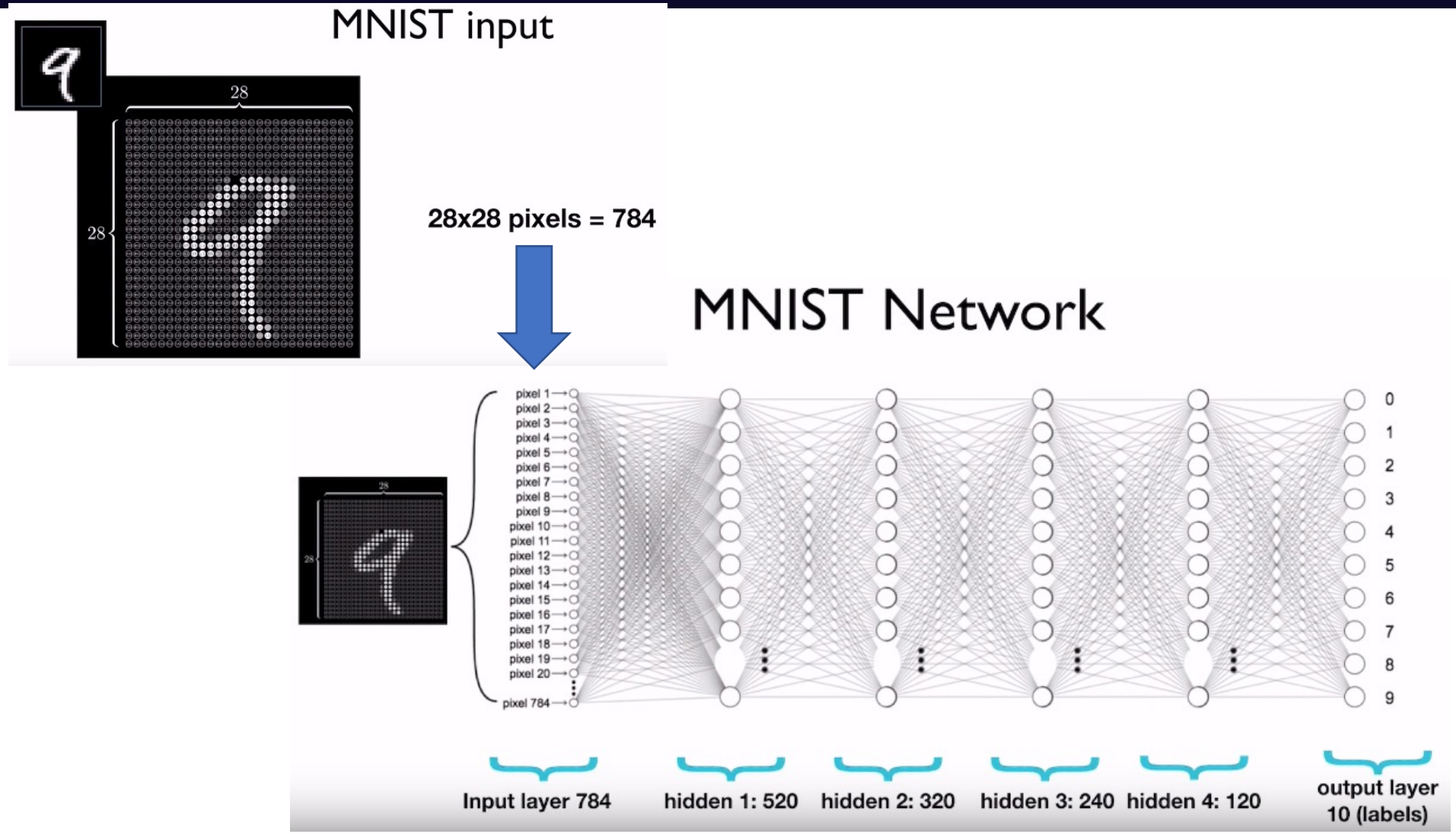
CSI4121

Ch 1. Data Representation (part2)

Jinyoung Yeo

Yonsei AI

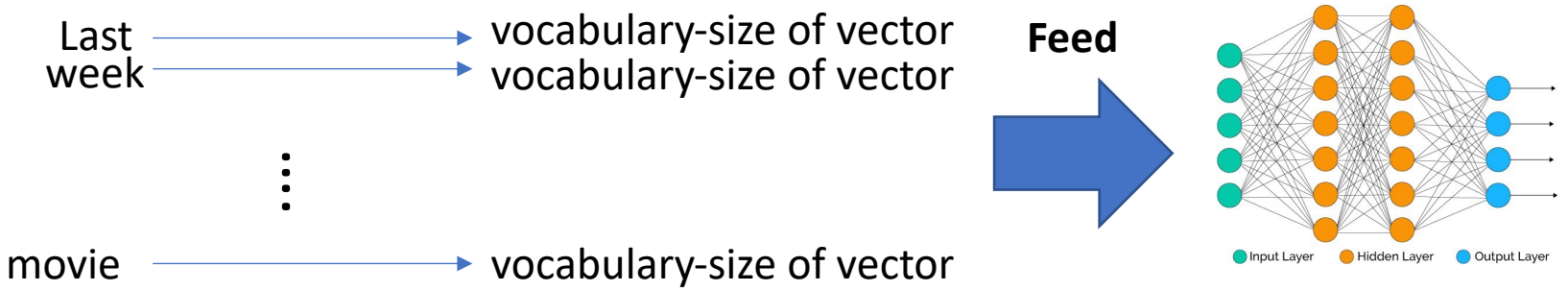
Neural Networks



One-hot Vector/Encoding

- $v \in \{0,1\}^{|V|}, where$ v is one-hot vector and |V| is vocaburary size

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1



Drawbacks of One-hot Represenation

- Curse of Dimensionality

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Mostly zeros → Information quality ↓

Memory use ↑

Computation time ↑

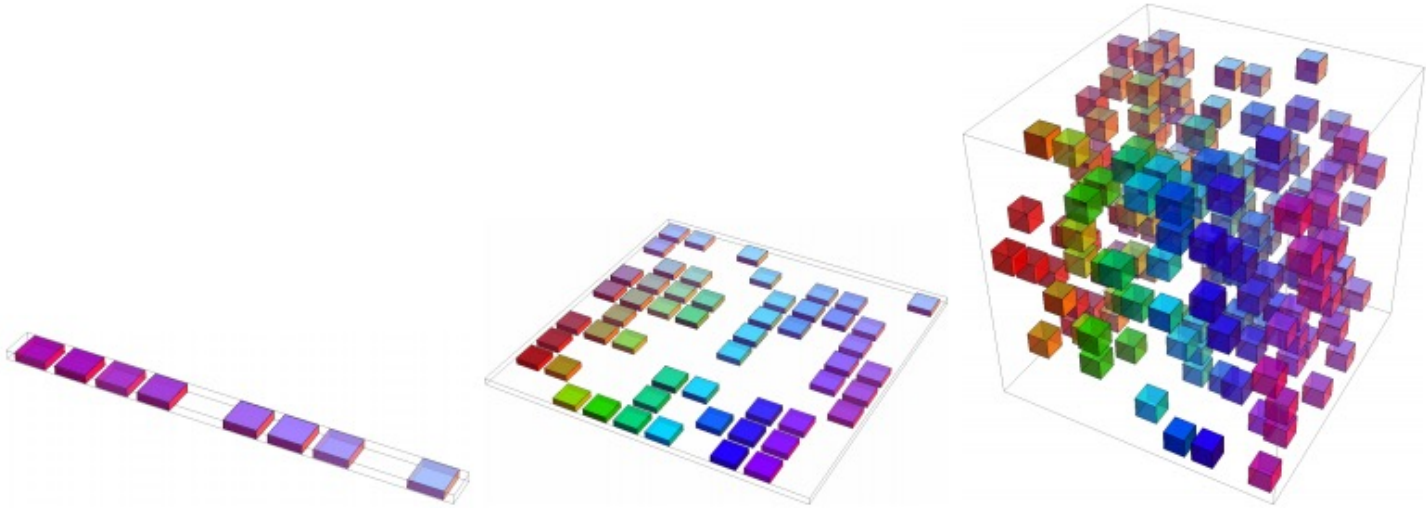
- Orthogonality

$$[0,1,0,\ldots,0] \text{ X } [0,1,0,\ldots,0]^T = 0$$

Distance(“Dog”, “Puppy”) = Distance(“Dog”, “Computer”) = 0

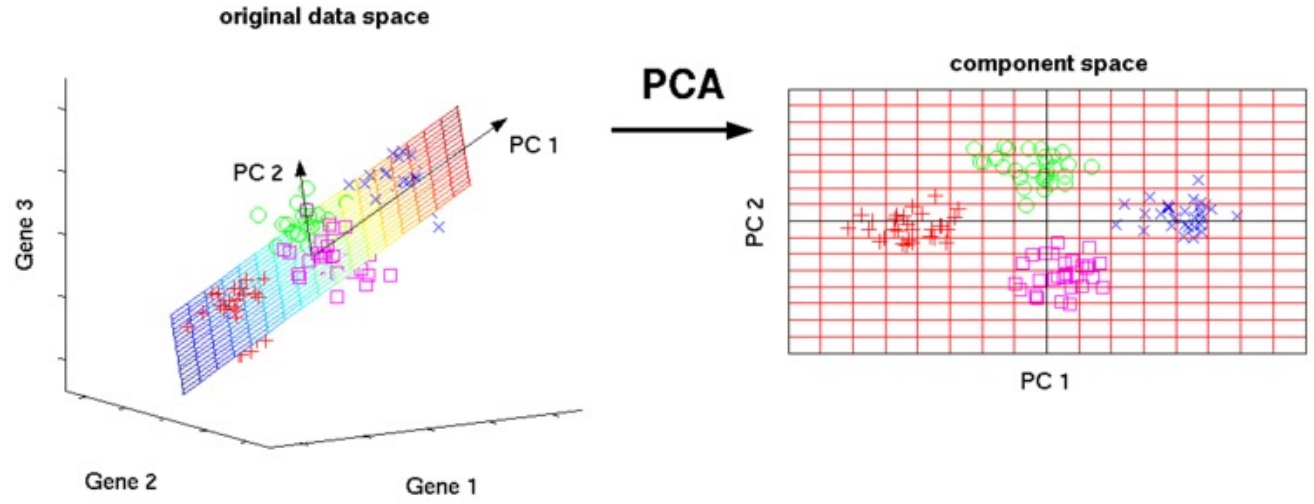
Curse of Dimensionality

- Many (machine learning) problems become exceedingly difficult when the number of dimensions in the data is high.
  - When dimensionality increases, data becomes increasingly sparse.
  - The possible combinations of subspaces will grow exponentially.



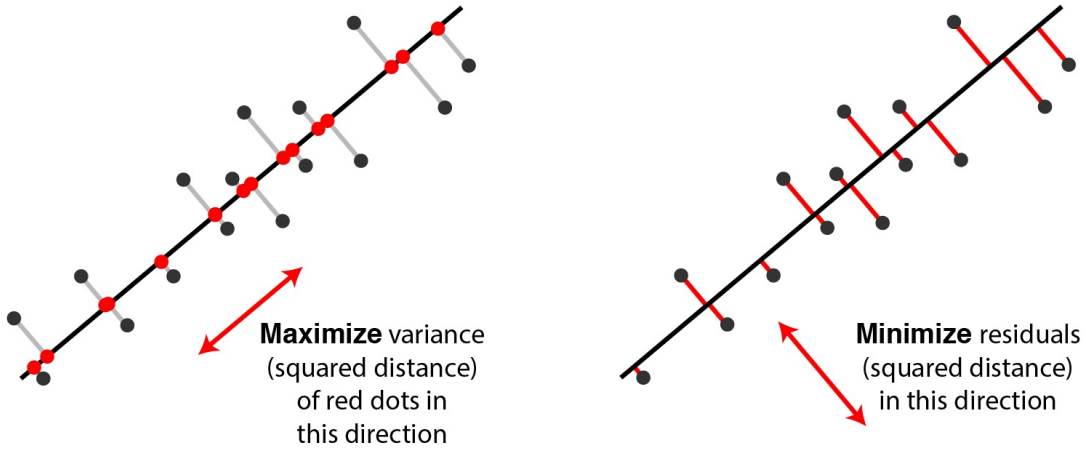
Dimensionality Reduction

- The process of converting a set of high dimensional data into a lower dimensional representation that conveys similar information
- Ex) PCA (Principal Component Analysis), LDA (Linear Discriminana Analysis)



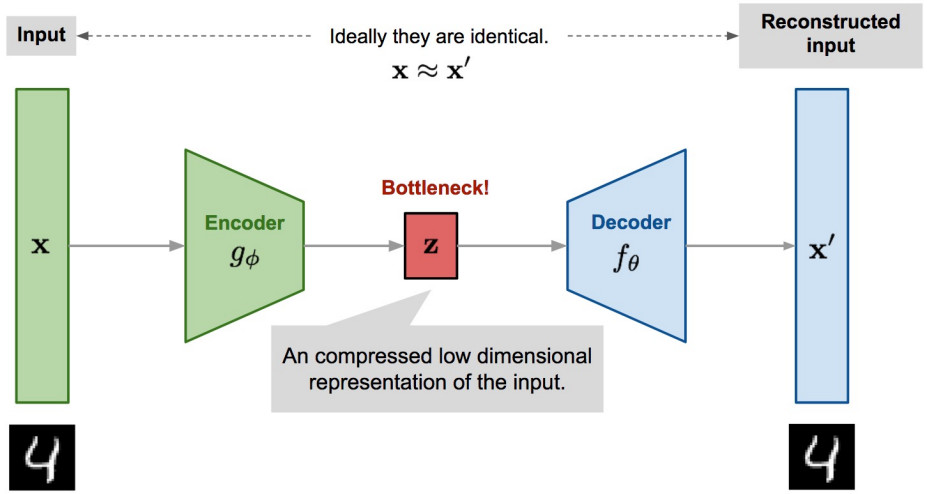
Dimensionality Reduction

- PCA finds a linear subspace that maximizes the variance of the data once it is projected into a lower-dimensional space.
- And also minimizes the distance of the projection in a least-squares sense.



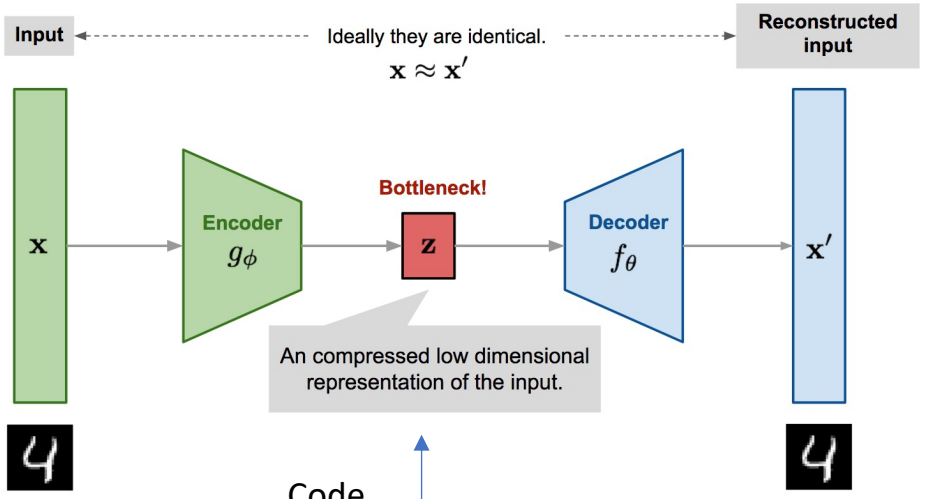
Autoencoder

- **Autoencoder** is a neural network designed to learn an identity function in an unsupervised way to reconstruct the original input while compressing the data in the process so as to discover a more efficient and compressed representation.



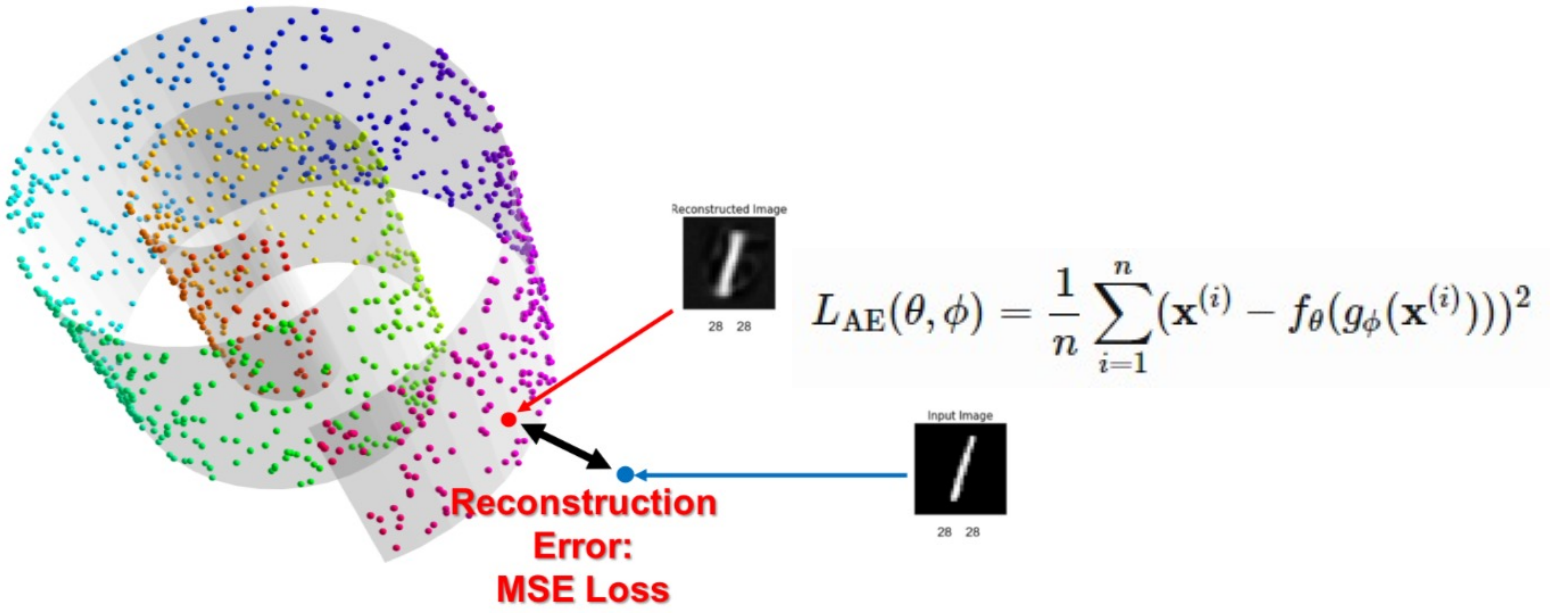
Autoencoder

- *Encoder* network: It translates the original high-dimension input into the latent low-dimensional code. The input size is larger than the output size.
- *Decoder* network: The decoder network recovers the data from the code, likely with larger and larger output layers.



Autoencoder

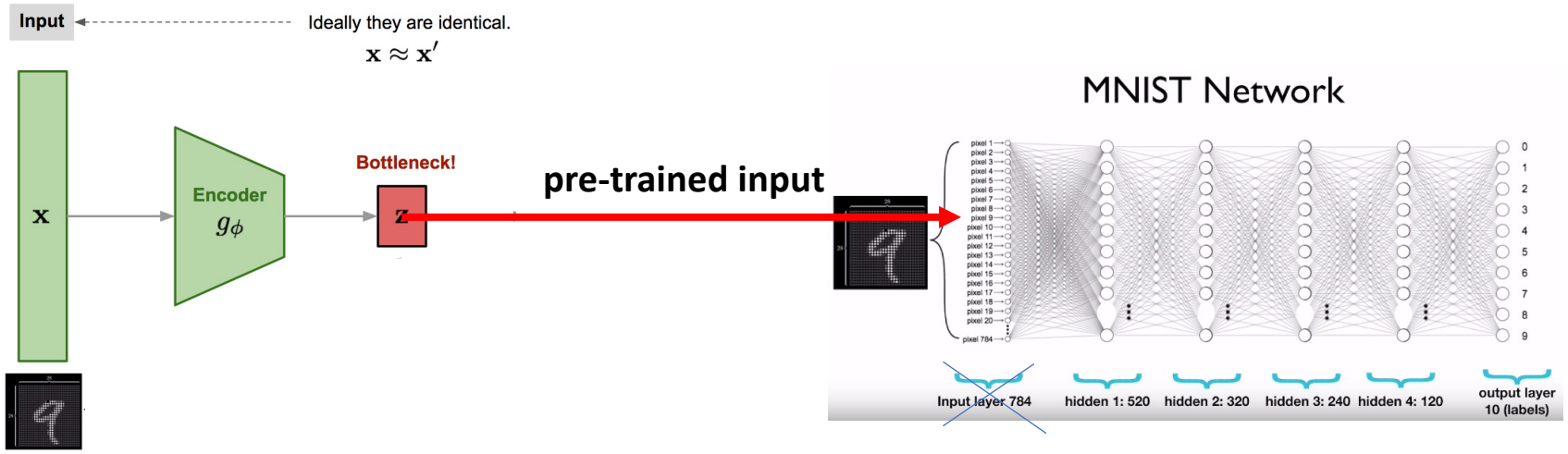
- Learning a reduced representation forces the autoencoder to capture the most important features of the training data.
- Loss function (MSE) penalizes $f(g(x))$ for being dissimilar from x.



$$L_{\mathrm{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$$

Autoencoder

- Learning a reduced representation forces the autoencoder to capture the most important features of the training data.
- Loss function (MSE) penalizes $f(g(x))$ for being dissimilar from x.



Autoencoder

Autoencoder

From Wikipedia, the free encyclopedia

An **autoencoder** is a type of artificial neural network used to learn efficient data codings in an unsupervised manner.[1] The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal "noise". Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name. Several variants

Introduction [edit]

An *autoencoder* is a neural network that learns to copy its input to its output. It has an internal (*hidden*) layer that describes a *code* used to represent the input, and it is constituted by two main parts: an encoder that maps the input into the code, and a decoder that maps the code to a reconstruction of the original input.

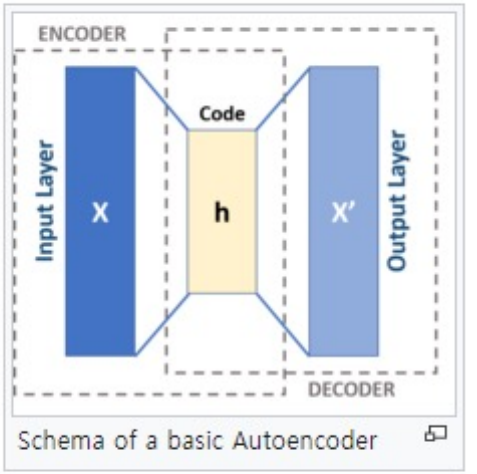
Autoencoder

Basic Architecture [edit]

The simplest form of an autoencoder is a feedforward, non-recurrent neural network similar to single layer perceptrons that participate in multilayer perceptrons (MLP) – having an input layer, an output layer and one or more hidden layers connecting them – where the output layer has the same number of nodes (neurons) as the input layer, and with the purpose of reconstructing its inputs (minimizing the difference between the input and the output) instead of predicting the target value Y given inputs X. Therefore, autoencoders are unsupervised learning models (do not require labeled inputs to enable learning).

An autoencoder consists of two parts, the encoder and the decoder, which can be defined as transitions ϕ and ψ, such that:

$$\phi : \mathcal{X} \to \mathcal{F}$$
$$\psi : \mathcal{F} \to \mathcal{X}$$
$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|X - (\psi \circ \phi)X\|^2$$



Schema of a basic Autoencoder

Autoencoder

In the simplest case, given one hidden layer, the encoder stage of an autoencoder takes the input $\mathbf{x} \in \mathbb{R}^d = \mathcal{X}$ and maps it to $\mathbf{h} \in \mathbb{R}^p = \mathcal{F}$:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

This image $\mathbf{h}$ is usually referred to as *code*, *latent variables*, or *latent representation*. Here, σ is an element-wise activation function such as a sigmoid function or a rectified linear unit. $\mathbf{W}$ is a weight matrix and $\mathbf{b}$ is a bias vector. Weights and biases are usually initialized randomly, and then updated iteratively during training through Backpropagation. After that, the decoder stage of the autoencoder maps $\mathbf{h}$ to the reconstruction $\mathbf{x}'$ of the same shape as $\mathbf{x}$:

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

where $\sigma', \mathbf{W}', \text{ and } \mathbf{b}'$ for the decoder may be unrelated to the corresponding $\sigma, \mathbf{W}, \text{ and } \mathbf{b}$ for the encoder.

Autoencoders are trained to minimise reconstruction errors (such as squared errors), often referred to as the "loss":

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

where $\mathbf{x}$ is usually averaged over some input training set.

Drawbacks of One-hot Represenation

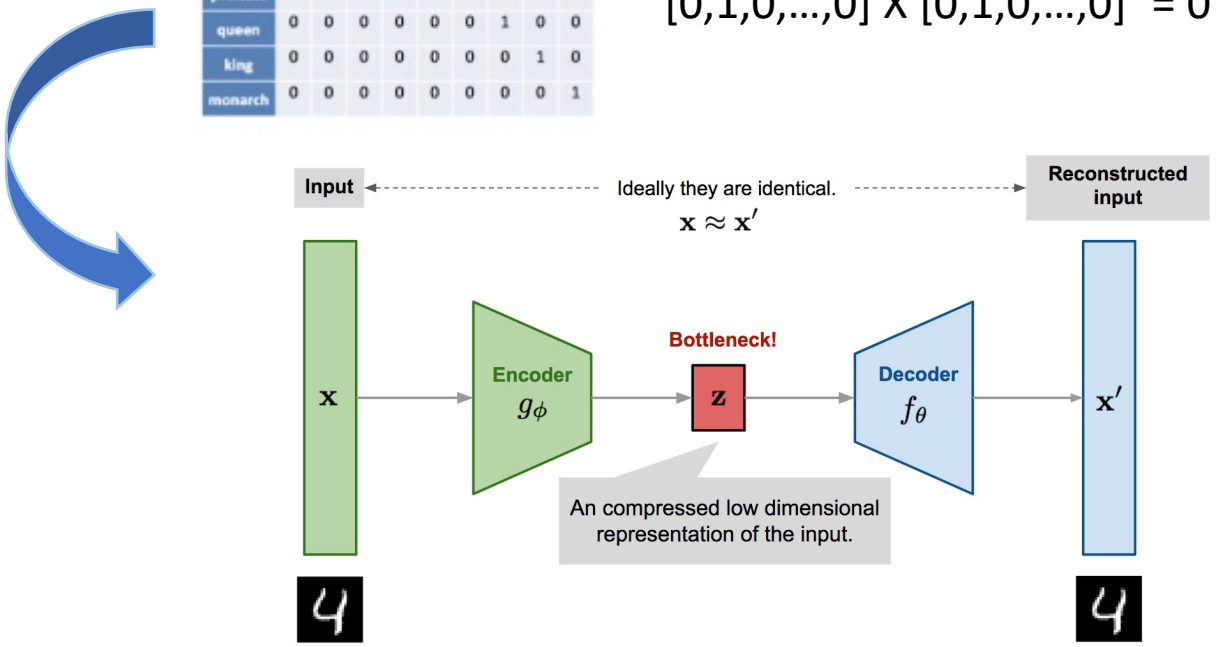
- Curse of Dimensionality
- Orthogonality

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Mostly zeros

$[0,1,0,\ldots,0] \times [0,1,0,\ldots,0]^T = 0$

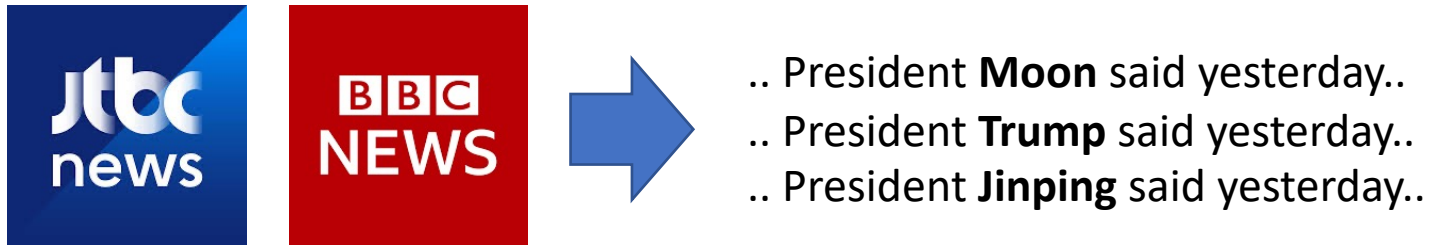
What if



Word Embedding: Word2Vec

Distributional Hypothesis

- “Tell me who your friends are, and I will tell you who you are.”
- “You shall know a word by the company it keeps” (J.R. Firth, British Linguist, 1957)
- “Words that occur in similar contexts tend to be similar” (Z.S. Harris, American Linguist, 1992)

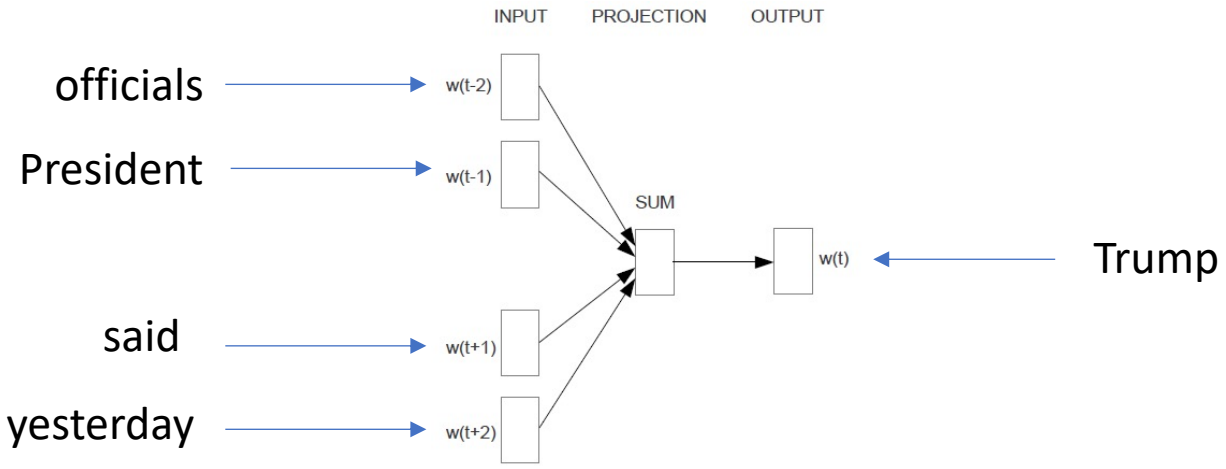


Distributional Hypothesis: Moon, Trump, and Jinping have the similar contexts in news data, which is a strong signal of indicating their high similarity.

Continuous Bag of Words (CBOW)

- Given a set of neighboring words (contexts), predict single words that potentially occur along with the contexts.

… and military officials, President **Trump** said yesterday, The United States is ready...



Word2Vec: CBOW

- Input: a set of one-hot vectors of given context words
- Initialize a matrix for dimensionality reduction

w(t-2) ..001000..

w(t-1) ..0100..0..

w(t+1) ..00..001..

w(t+2) ..00..010..

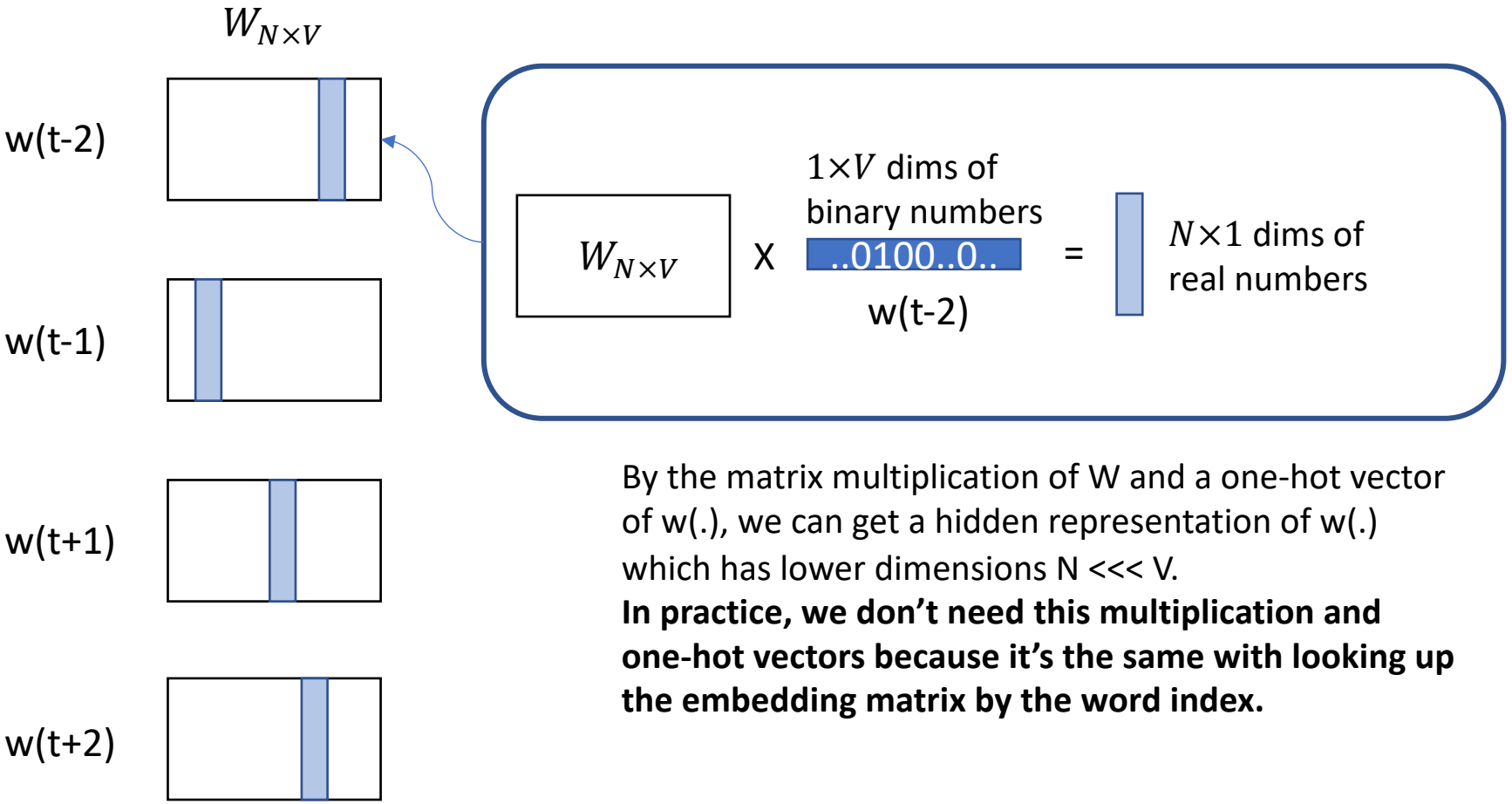
Learnable (Trainable) matrix

$W_{N \times V}$

V: the number of words in vocabulary (hyperparameter)
N: the size of hidden representation (hyperparameter)

Word2Vec: CBOW

- Look up the hidden representations of context words.

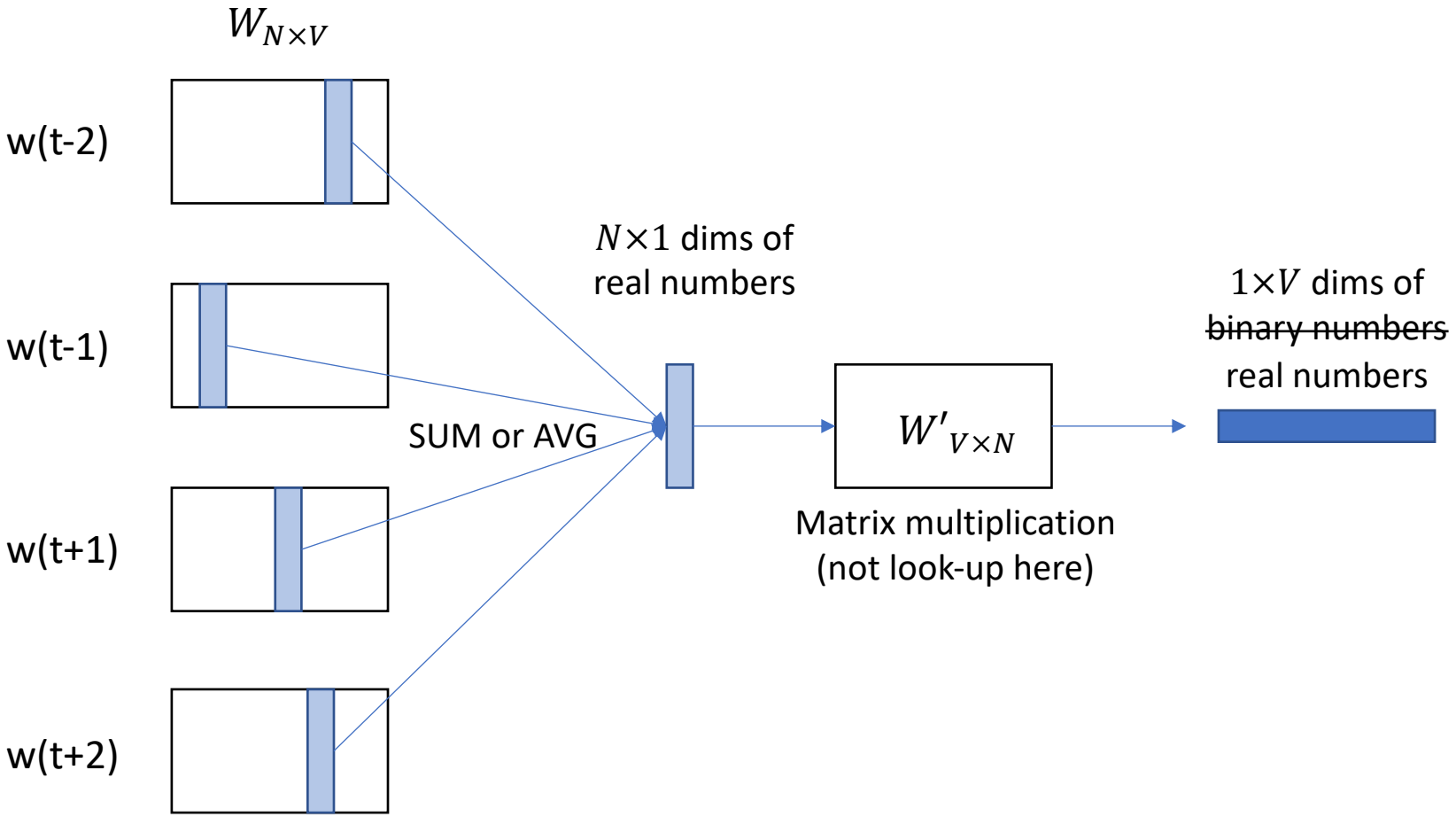


By the matrix multiplication of W and a one-hot vector of w(.), we can get a hidden representation of w(.) which has lower dimensions N <<< V.

In practice, we don't need this multiplication and one-hot vectors because it's the same with looking up the embedding matrix by the word index.

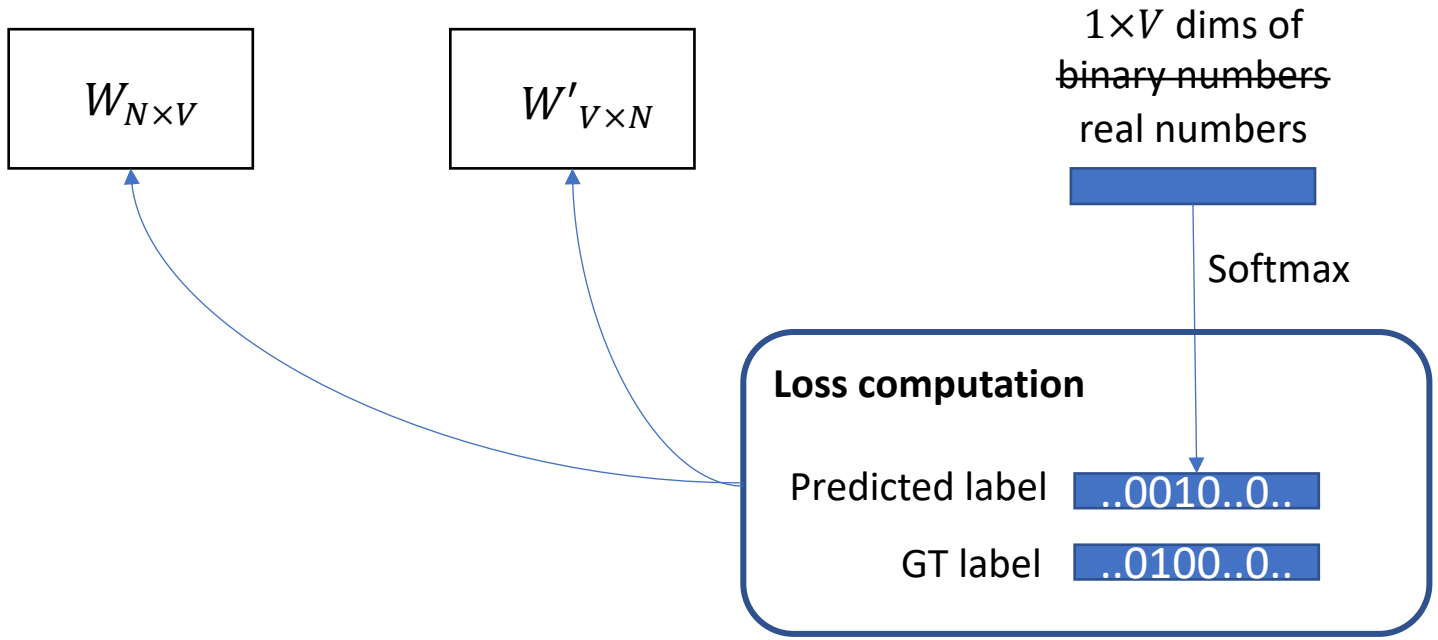
Word2Vec: CBOW

- "Encode" into one representation and then "decode".

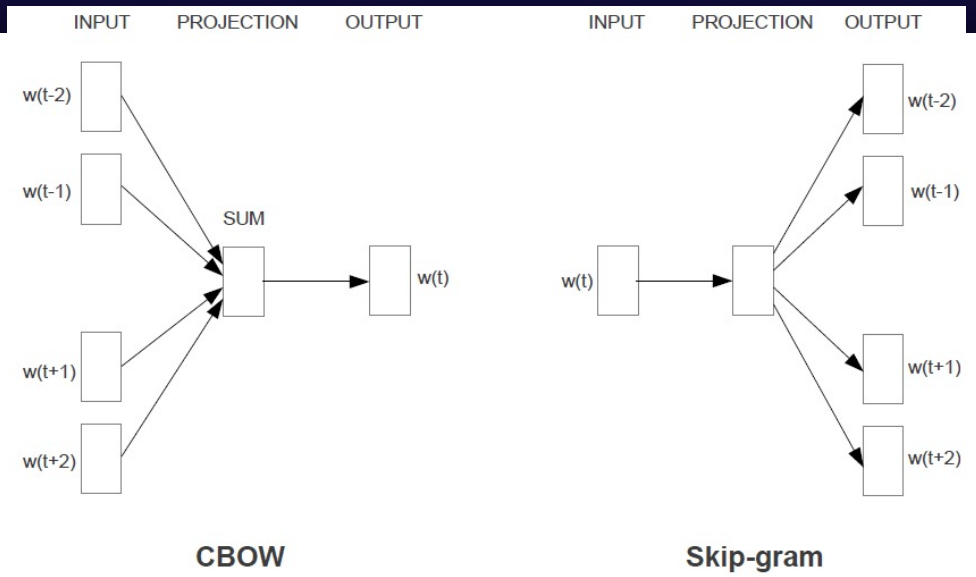


Word2Vec: CBOW

- Compute a loss value and its gradients and backpropagate them for training.



Word2Vec: Skip-gram

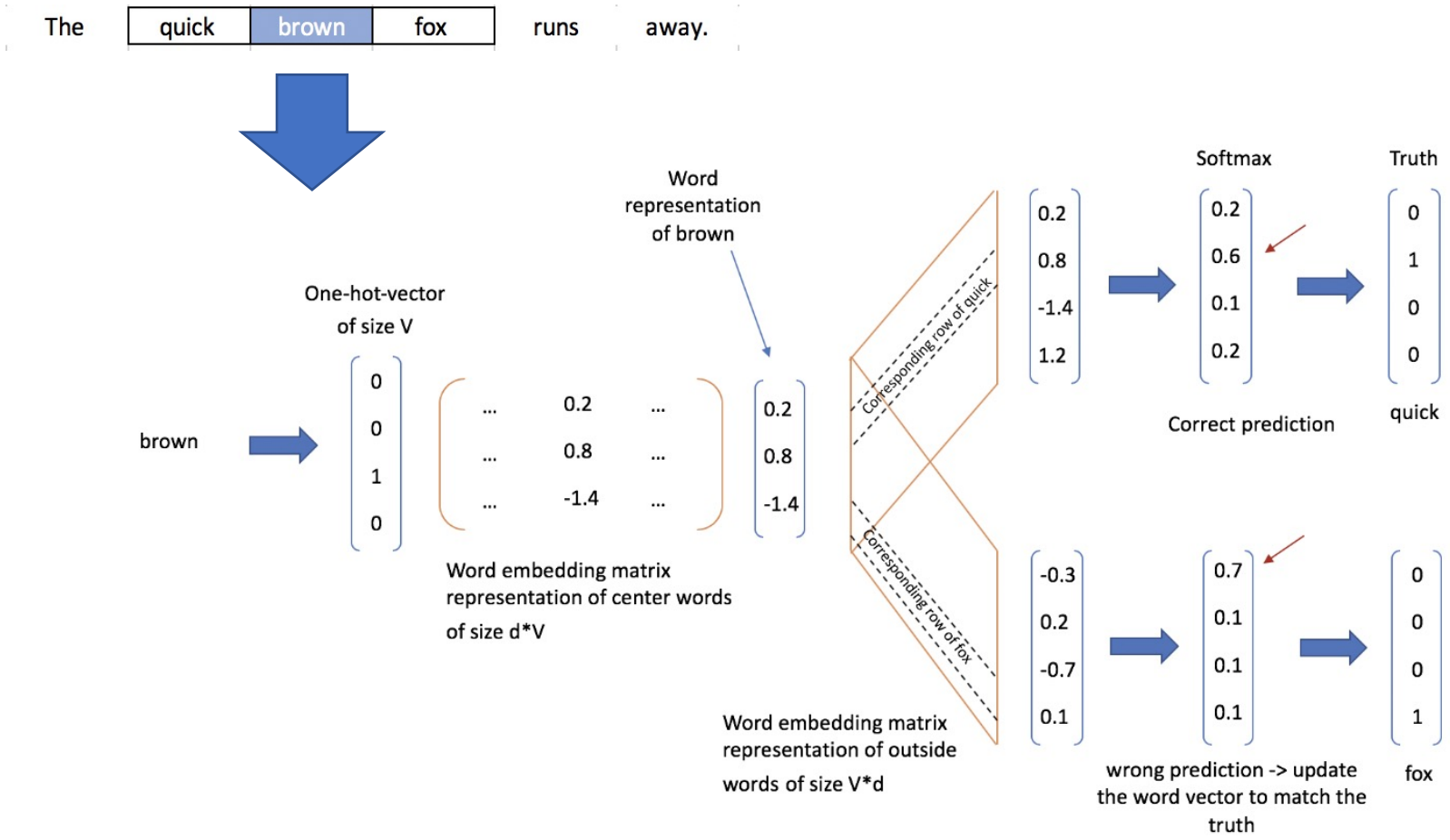


Source Text

Training Samples

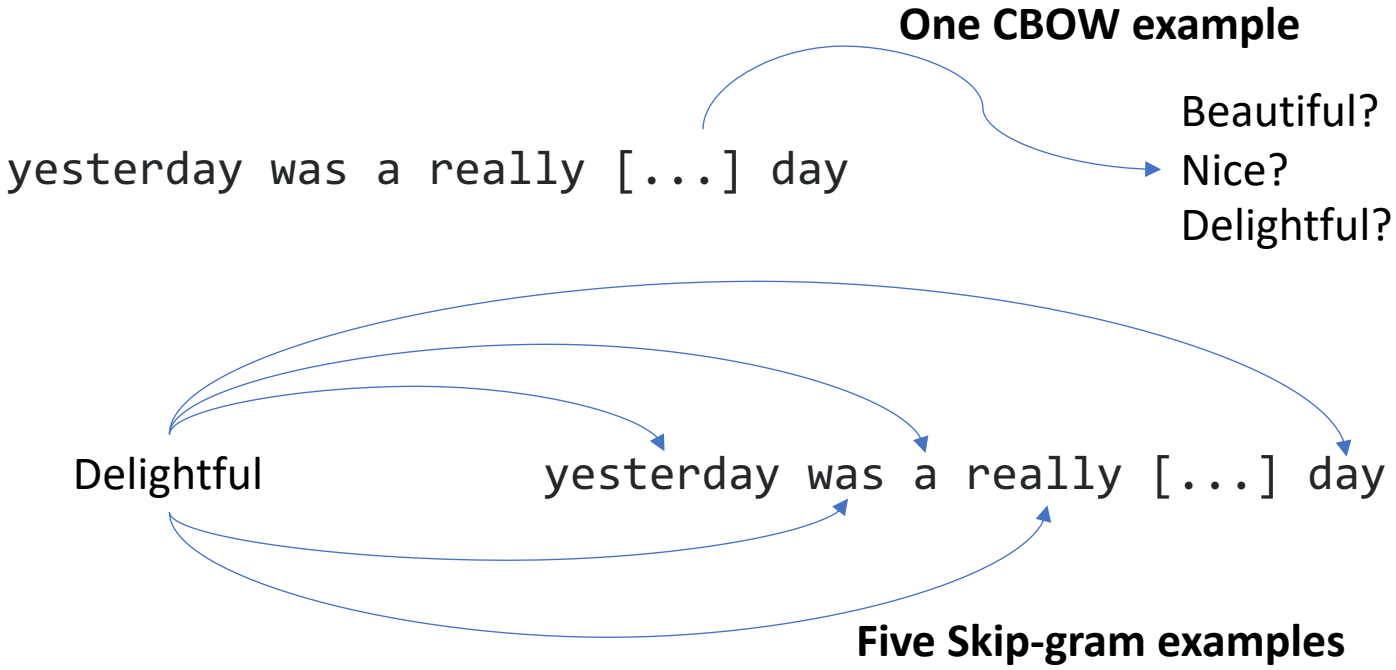
The quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Word2Vec: Skip-gram



Difference between CBOW and Skip-gram

- **CBOW:** several times faster to train than the skip-gram, slightly better accuracy for the frequent words
- **Skip-gram:** works well with small amount of the training data, represents well even rare words or phrases.



Other techniques in Word2Vec

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log p(w_{t+j}|w_t) \longleftarrow log\frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V}\exp(u_w^T v_c)}$$

- Computing the normalization factor is expensive.

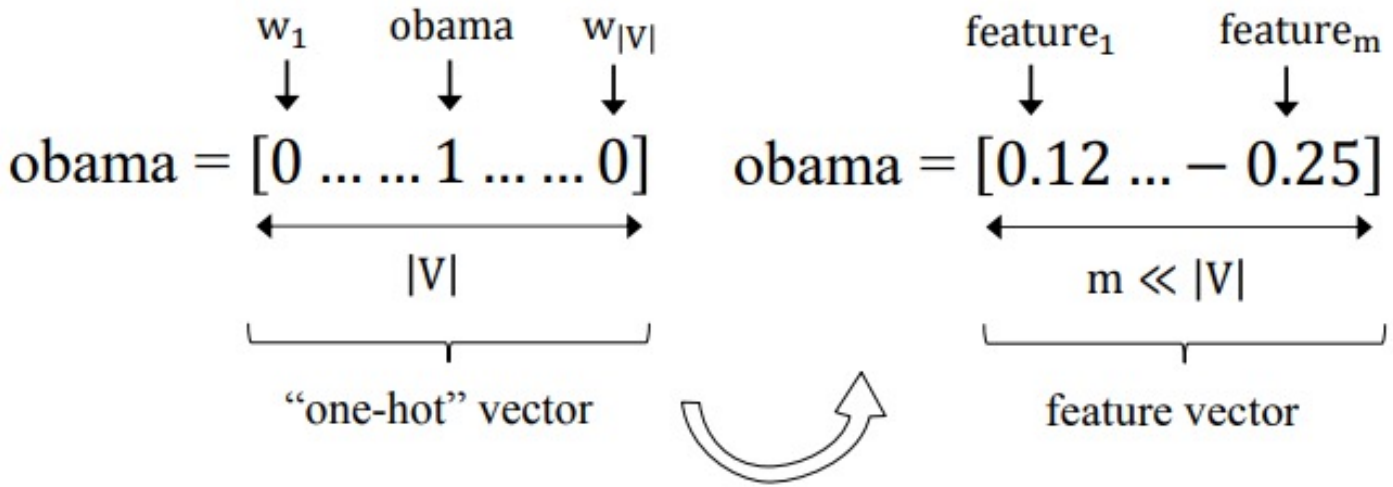
- **Hierarchical Softmax:** We assign the words to the leaves of a binary tree, turning the prediction problem into maximizing the probability of a specific path in the hierarchy.

- **Negative Sampling:** Selecting 5~20 negative samples instead of using the overall set of words in the vocabulary

Word embeddings: distributed representation of words

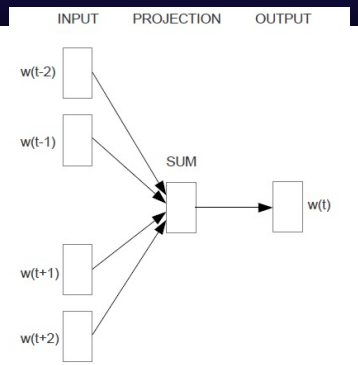
- Each unique word is mapped to a point in a real continuous m-dimensional space
- Typically, $|V| > 10^6$, $100 < m < 500$

$$w_i \in V \xrightarrow{\text{mapping } C} \mathbb{R}^m$$

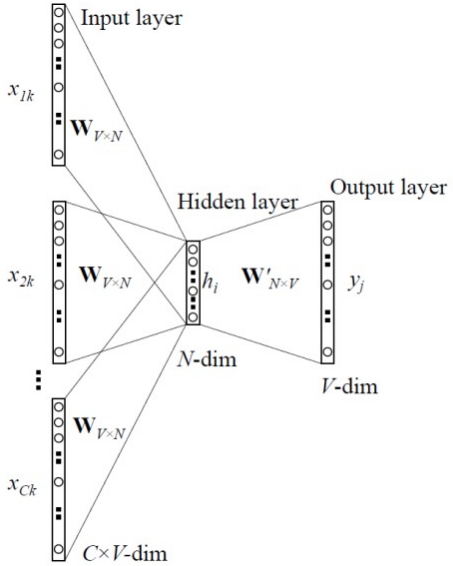


- Fighting the curse of dimensionality with:
  - **compression** *(dimensionality reduction)*
  - **smoothing** *(discrete to continuous)*
  - **densification** *(sparse to dense)*
- Similar words end up close to each other in the feature space

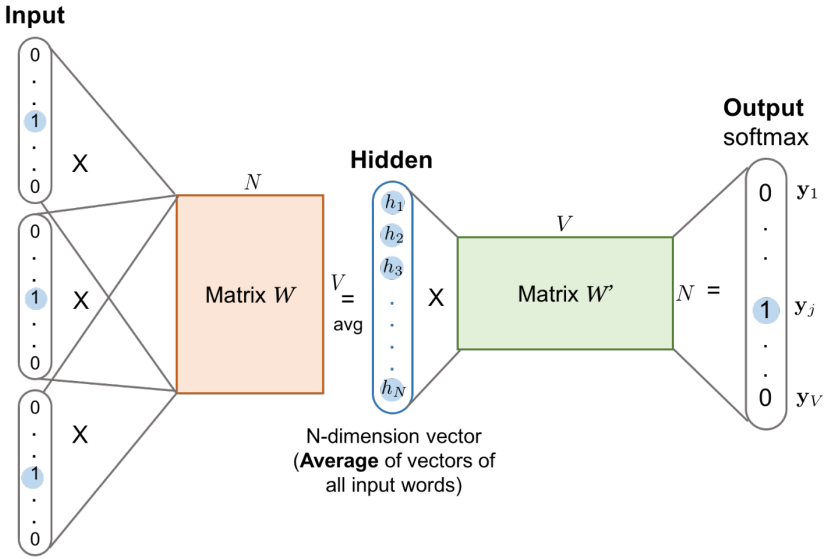
Supplement (read one of these)



http://solarisailab.com/archives/959
(Korean)



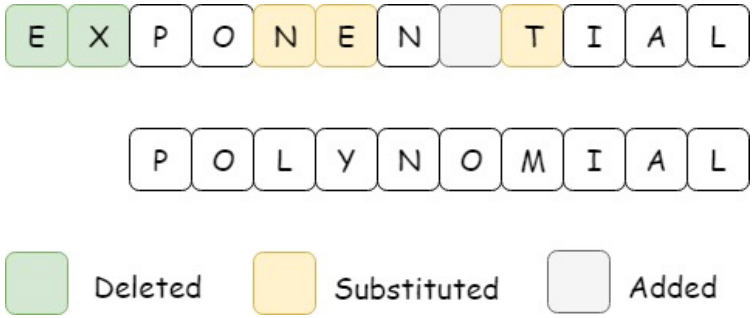
https://reniew.github.io/21/
(Korean)



https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html
(English)

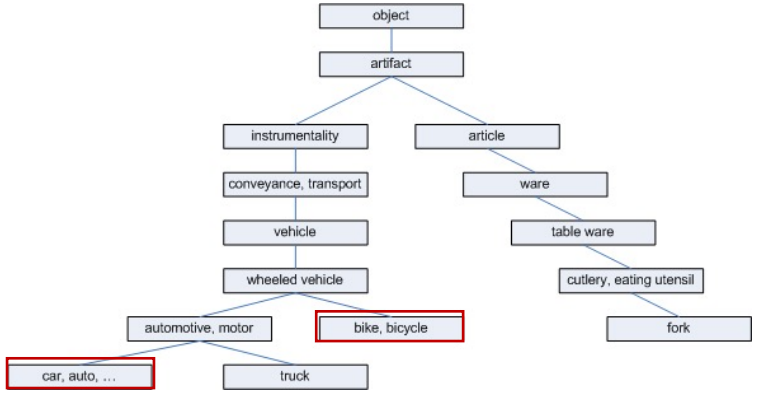
Once upon a time in NLP…

- In computer science, traditional ways to handle text data is to consider it as discrete symbols.
- Extreme examples are rule-based distance/similarity metrics between two strings/words.



Edit Distance: minimum number of operations {delete, substitute, add} to transform one into another.

Dist(“exponential”, “polynomial”) = 6

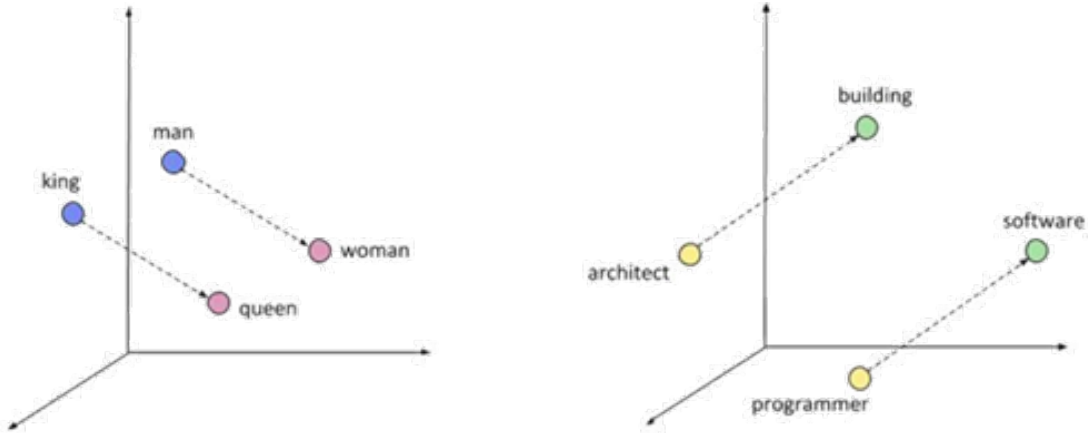


WORDNET: a hierarchical structure of words where the shortest path between two words is defined as another distance.

Dist(“car”, “bike”) = 3

Word Embedding

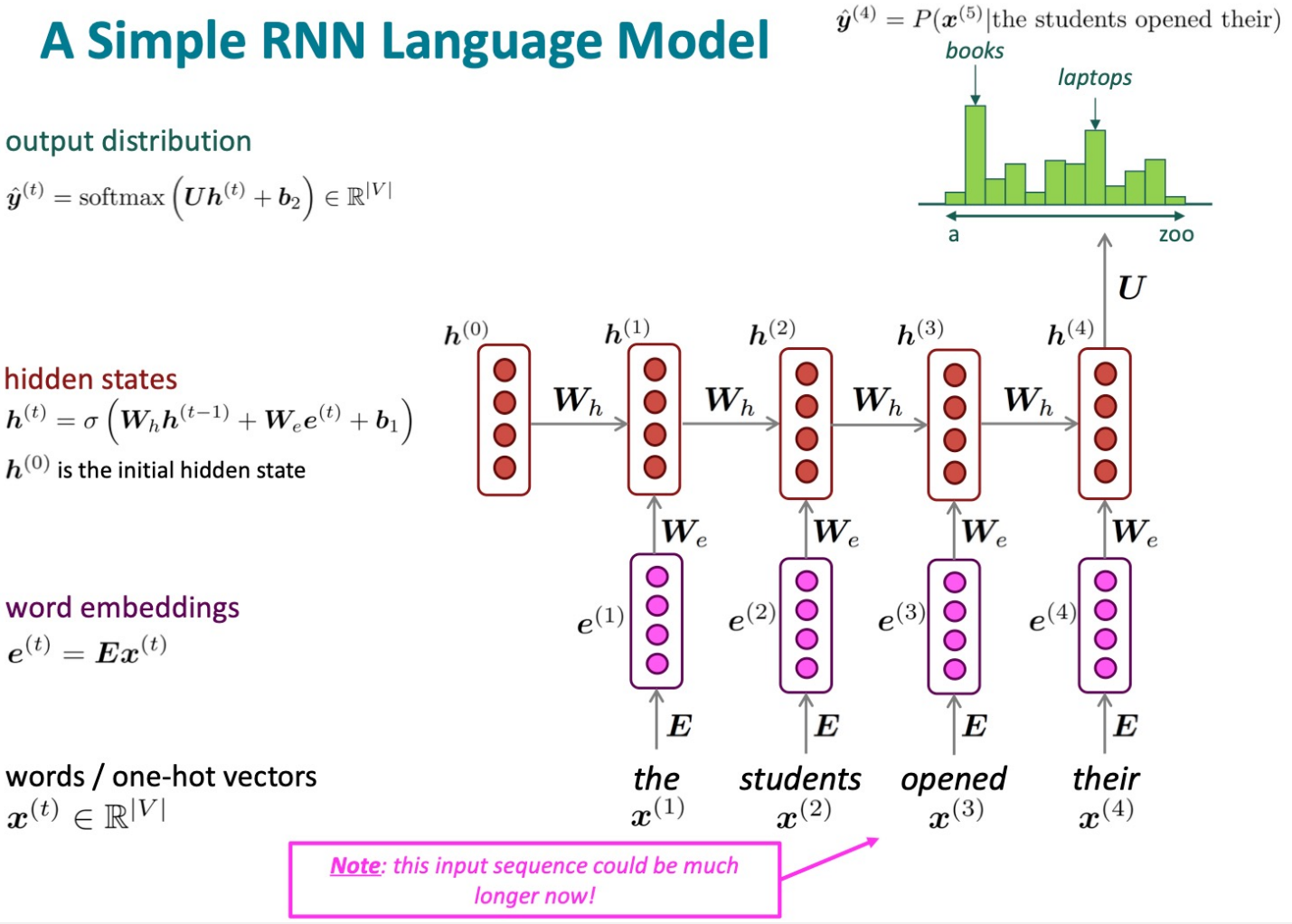
- In the age of AI, a word/sentence is encoded into a numeric and continuous vector that is understandable format for machine learning.
- In this lecture, we studied several shifts
  - 1) from discrete symbols to continuous vector
  - 2) from naïve continuous vector to “effective” representation



Popular examples of word embedding:
Vec(“king”)-Vec(“man”)+Vec(“woman”) ≈ Vec(“queen”)

Wrap up

A Simple RNN Language Model



Thanks, any question?