

# Neural Language Modeling & Word Embeddings

Kuan-Yu Chen (陳冠宇)

2018/03/29 @ TR-409, NTUST

# Language Modeling

---

- A goal of statistical language modeling is to learn the joint probability function of sequences of words in a language

$$P(w_1, w_2, \dots, w_T)$$

- A statistical model of language can be represented by the conditional probability of the next word given all the previous ones (**chain rule**)

$$P(w_1, w_2, \dots, w_T) = P(w_1) \prod_{t=2}^T P(w_t | w_1, w_2, \dots, w_{t-1})$$

# N-gram – Markov Assumption

---

- Unigram Model
  - Each word occurs independently of the other words
  - The so-called “bag-of-words” model

$$P(w_1, w_2, \dots, w_T) = P(w_1) \cdot P(w_2) \cdots P(w_T) = \prod_{t=1}^T P(w_t)$$

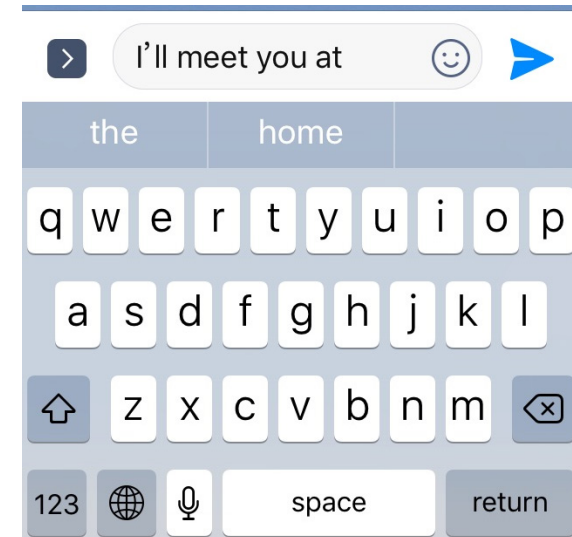
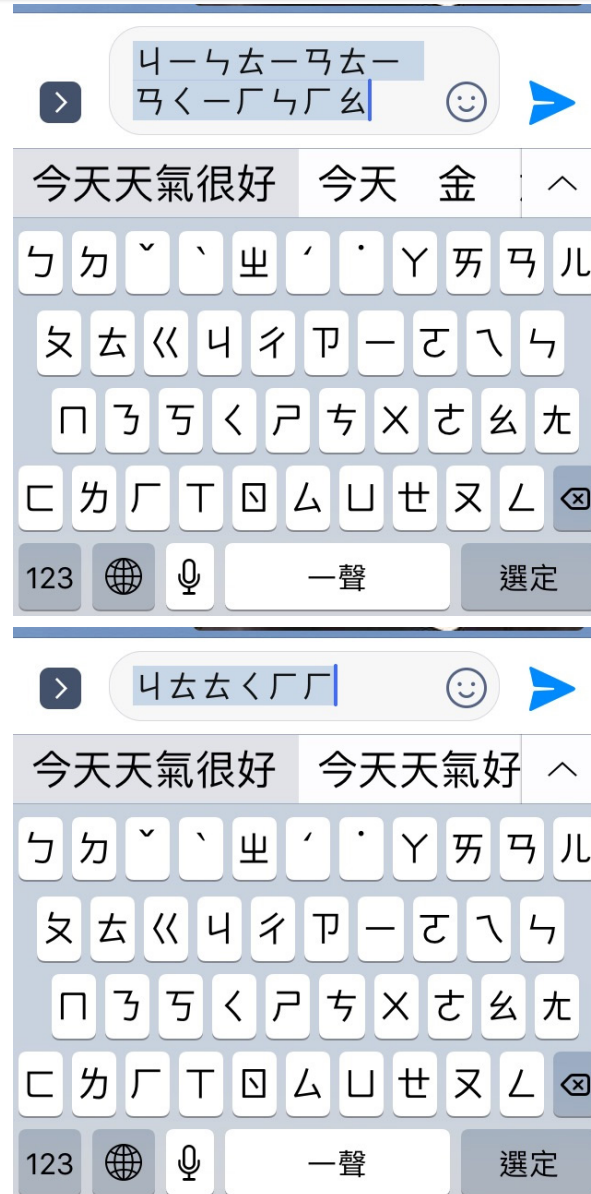
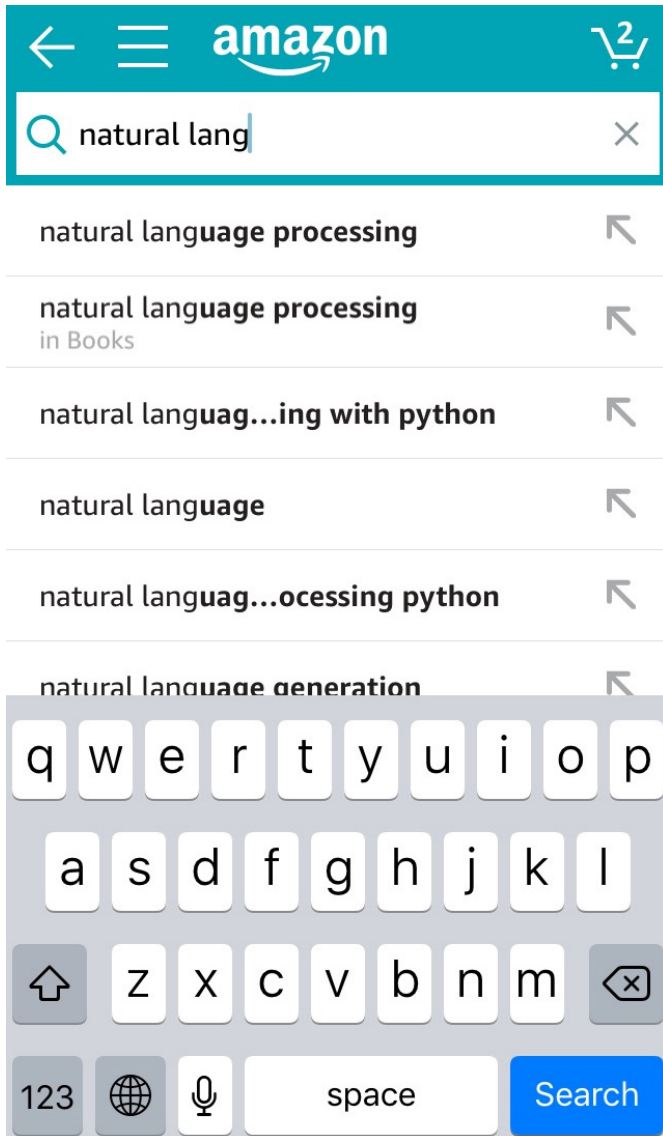
- Bigram Model

$$P(w_1, w_2, \dots, w_T) = P(w_1)P(w_2|w_1) \cdots P(w_T|w_{T-1}) = P(w_1) \prod_{t=2}^T P(w_t|w_{t-1})$$

- Trigram Model

$$\begin{aligned} P(w_1, w_2, \dots, w_T) &= P(w_1)P(w_2|w_1) \cdots P(w_T|w_{T-1}) \\ &= P(w_1)P(w_2|w_1) \prod_{t=3}^T P(w_t|w_{t-2}, w_{t-1}) \end{aligned}$$

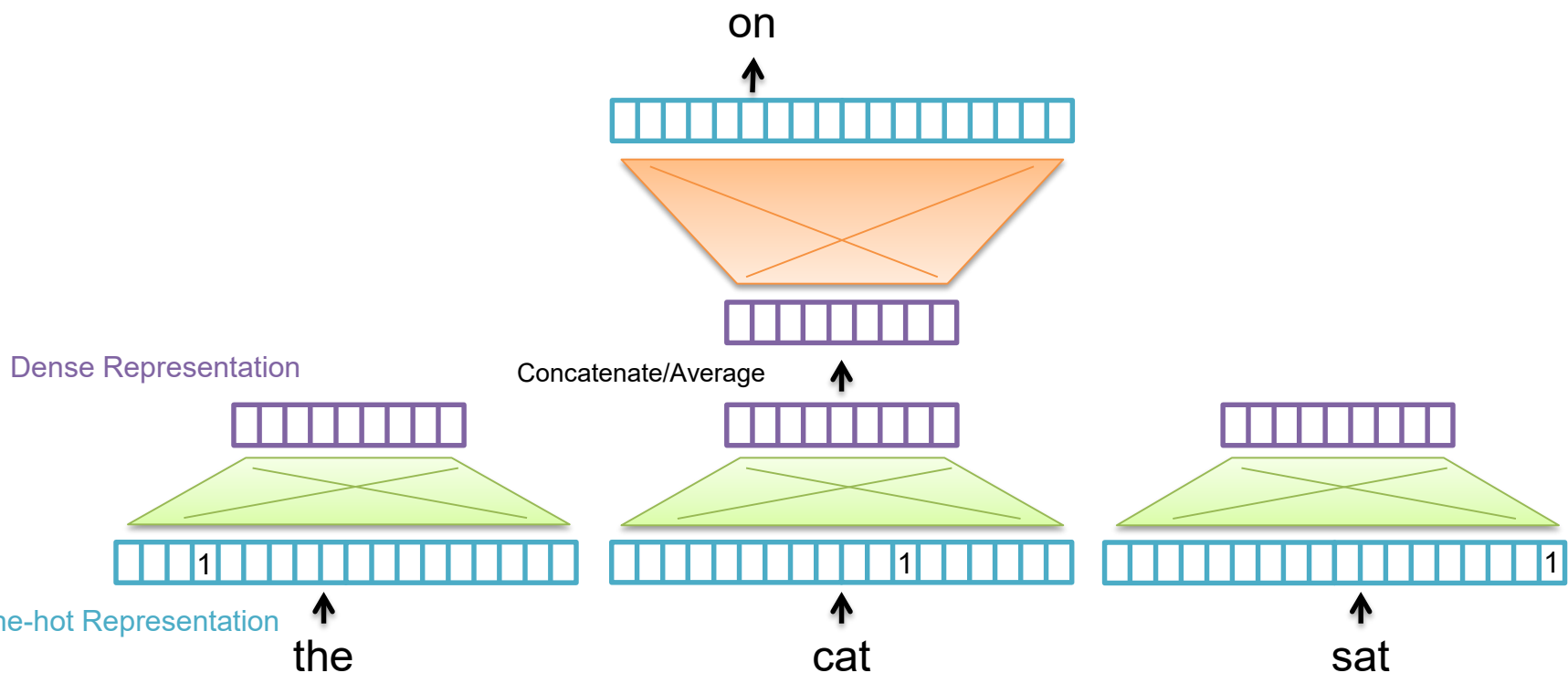
# Language Modeling is All Around



# Neural Network Language Modeling – 1

- The Neural Network Language Model (NNLM) estimated a statistical ( $n$ -gram) language model for **predicting future words**

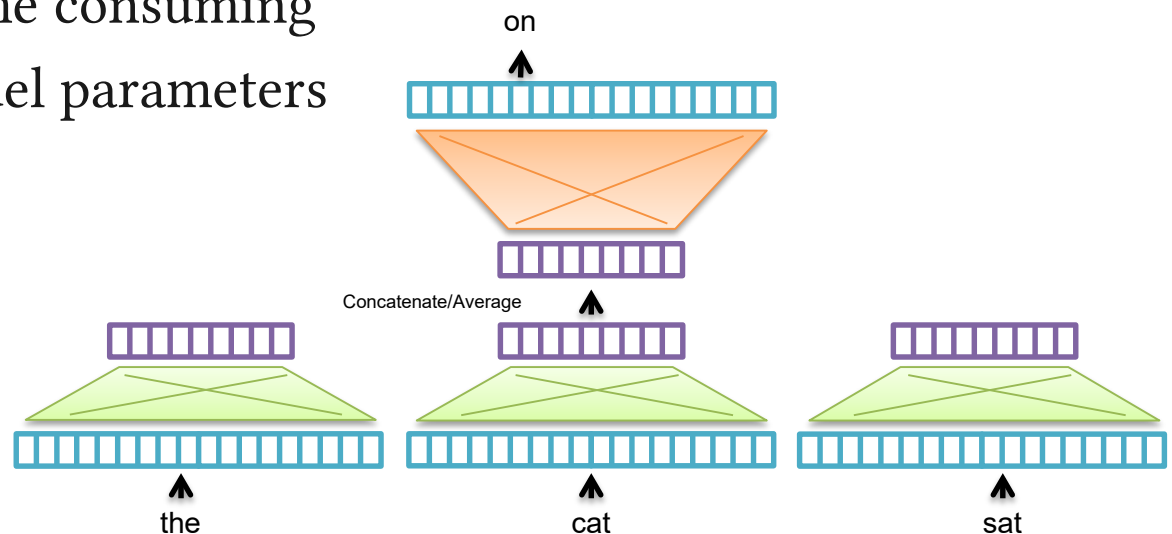
$$P(w_1, w_2, \dots, w_T) \approx \prod_{t=1}^T P(w_t | w_{t-n+1}, \dots, w_{t-1})$$



# Neural Network Language Modeling – 2

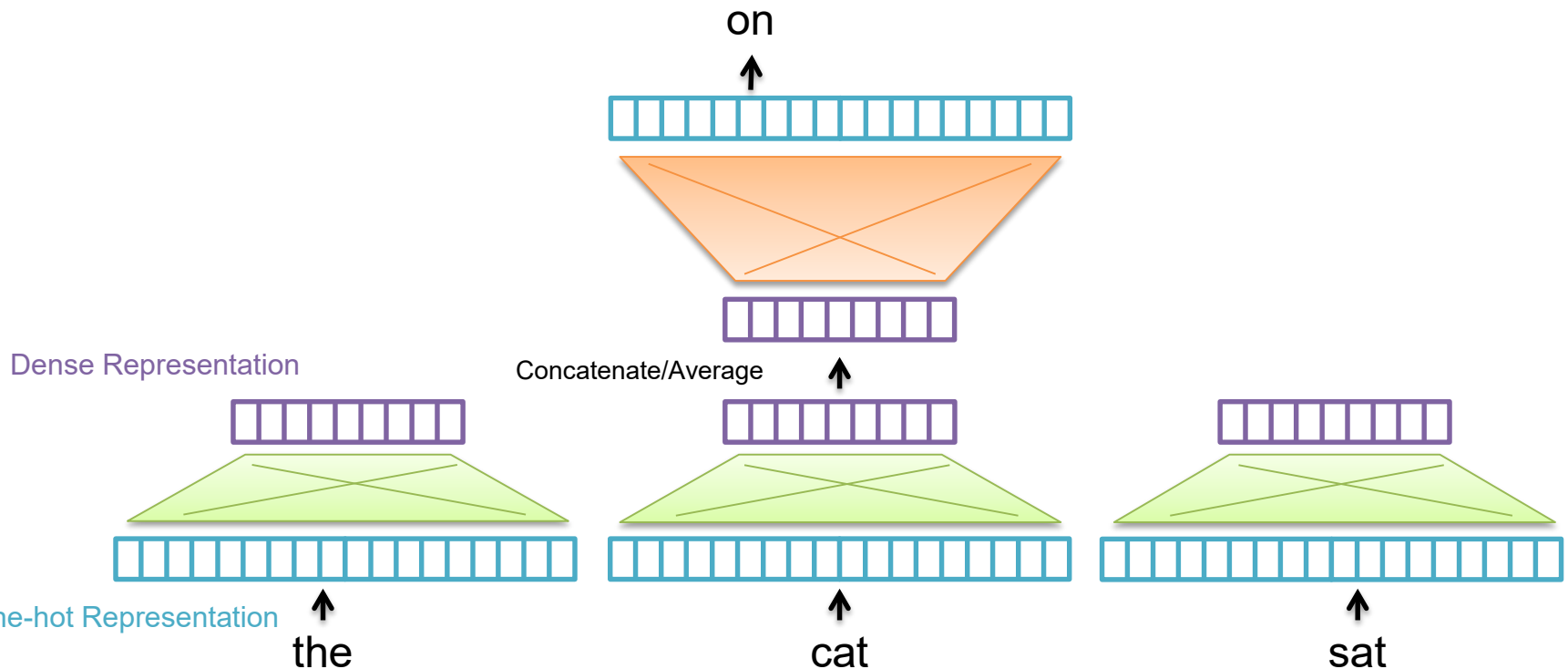
---

- Advantages:
  - It has a build-in smoothing technique
  - (Semantically or Syntactically) Similar words will have similar probabilities
    - Generalization ability
- Disadvantages:
  - The training is time consuming
  - A large set of model parameters



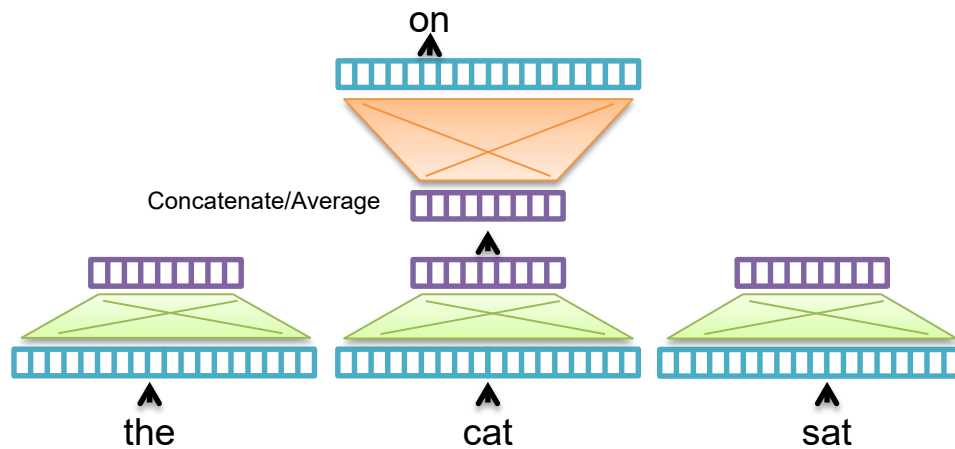
# NNLM

- Perhaps one of the most-known seminal studies on developing word embedding methods was rooted in the Neural Network Language Modeling (NNLM)



- It estimated a statistical ( $n$ -gram) language model for **predicting future words** in context while inducing word embeddings as a by-product

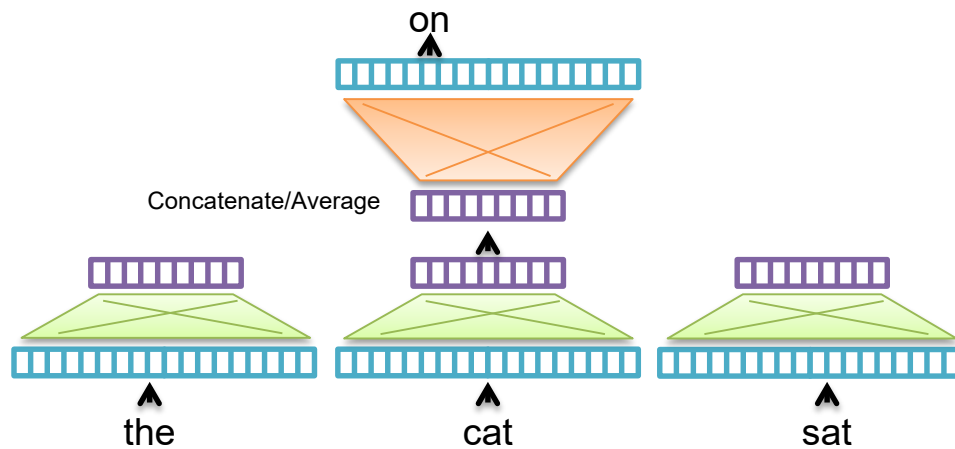
# From One-hot to Dense – 1



The diagram shows the multiplication of a one-hot vector by a dense matrix to produce a dense vector. On the left is a one-hot vector (blue bar with 15 segments). This is followed by a multiplication symbol ( $\times$ ) and a large green rectangle with a diagonal cross, representing a dense matrix. This is followed by an equals sign ( $=$ ) and a resulting dense vector (purple bar with 10 segments).



# From One-hot to Dense – 2



$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} \text{Dense Embedding Matrix} \end{bmatrix} = \begin{bmatrix} \text{Dense Embedding Vector} \end{bmatrix}$$

The diagram shows a 16-unit one-hot vector (blue bar) with the third unit set to 1, representing the word "the". This is multiplied (indicated by  $\times$ ) by a large green rectangle representing a dense embedding matrix. The result is a 16-unit purple vector, representing the dense embedding of the word "the".

# From Modeling to Vectorization

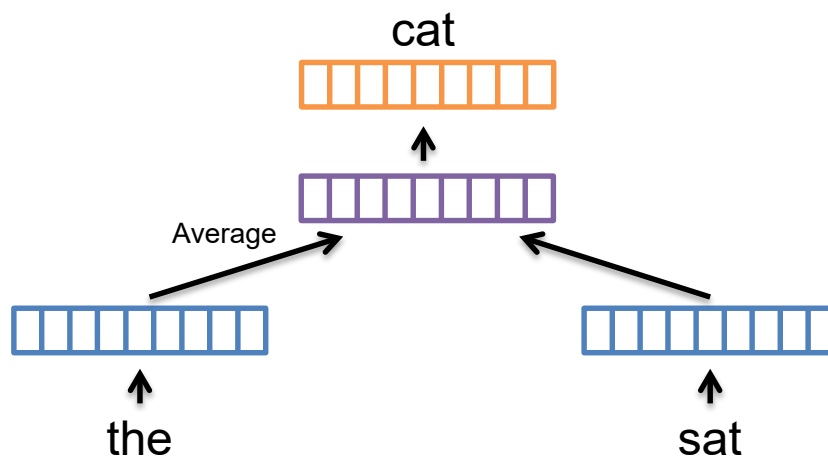
---

- Recent methods for learning vector space representations of words have succeeded in capturing **fine-grained semantic** and **syntactic** regularities
  - One-hot representation vs. Distributed representation
  - High-dimensional vector vs. Low-dimensional representation
  - Sparse vs. Dense
  - Orthogonal vs. Similar words will have similar representations
- The two main model families for learning word vectors
  1. Global matrix factorization methods
    - Global Vector
  2. Local context window methods
    - Continuous Bag-of-Words model, and Skip-gram model

# Continuous Bag-of-Words Modeling – 1

- Rather than seeking to learn a statistical language model, the CBOW model manages to obtain a dense vector representation (embedding) of each word directly

$$\prod_{t=1}^T P(w_t | w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c}) = \prod_{t=1}^T \frac{\exp(v_{\bar{w}_t} \cdot v_{w_t})}{\sum_{w \in V} \exp(v_{\bar{w}_t} \cdot v_w)}$$



$$v_{\bar{w}_t} = \frac{1}{2c} \sum_{j=-c \& \& j \neq 0}^c v_{w_{t+j}}$$

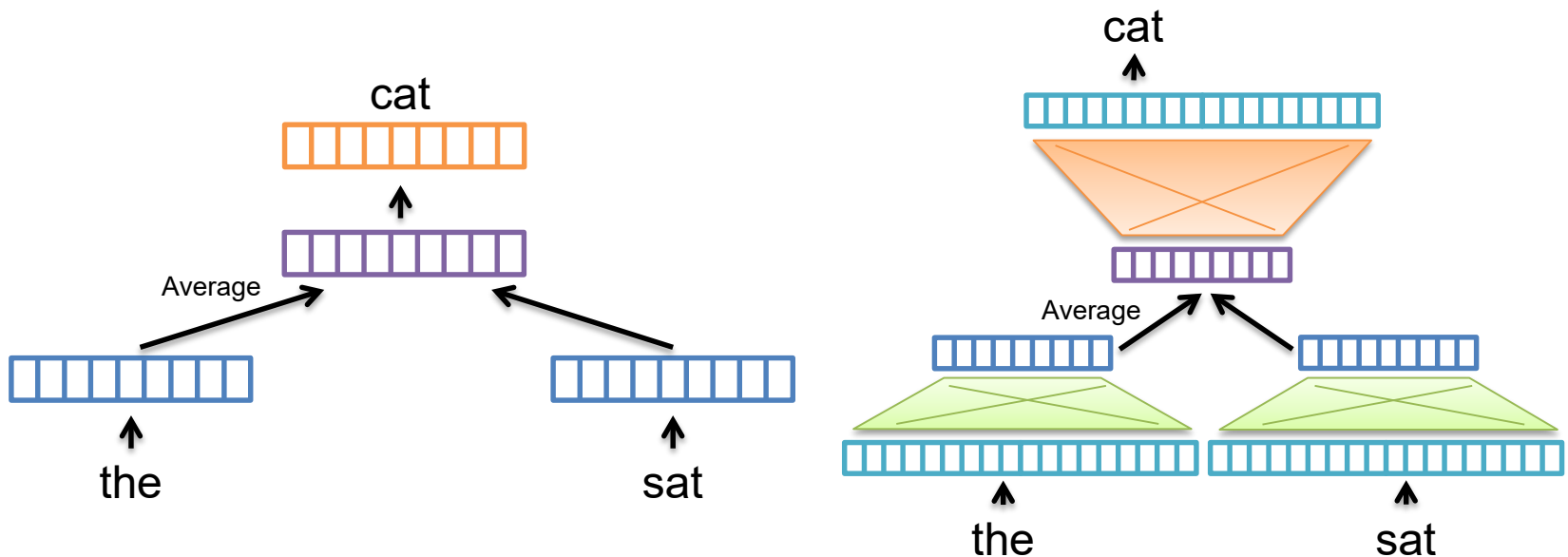
# Continuous Bag-of-Words Modeling – 2

---

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

# Continuous Bag-of-Words Modeling – 3

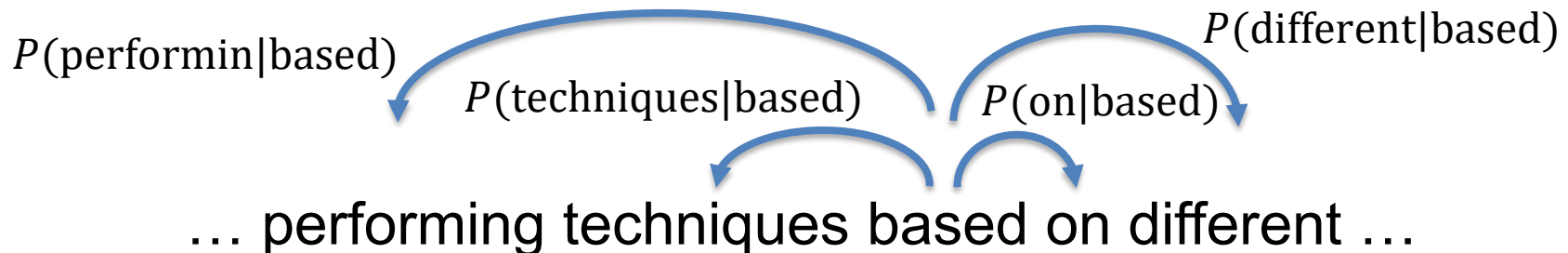
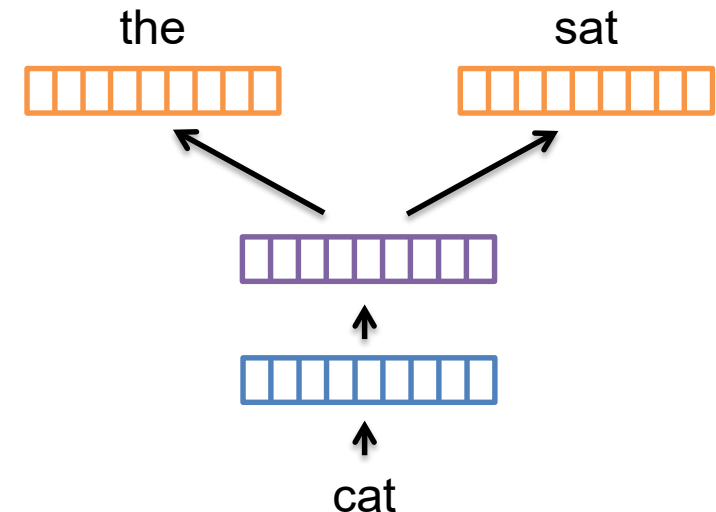
$$\prod_{t=1}^T P(w_t | w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c}) = \prod_{t=1}^T \frac{\exp(v_{\bar{w}_t} \cdot v_{w_t})}{\sum_{w \in V} \exp(v_{\bar{w}_t} \cdot v_w)}$$



# Skip-Gram Modeling

- In contrast to the CBOW model, the SG model employs an inverse training objective for learning word representations

$$\begin{aligned}
 & \prod_{t=1}^T P(w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c} | w_t) \\
 &= \prod_{t=1}^T \prod_{j=-c \& \& j \neq 0}^c P(w_{t+j} | w_t) \\
 &= \prod_{t=1}^T \prod_{j=-c \& \& j \neq 0}^c \frac{\exp(v_{w_{t+j}} \cdot v_{w_t})}{\sum_{w \in V} \exp(v_w \cdot v_{w_t})}
 \end{aligned}$$

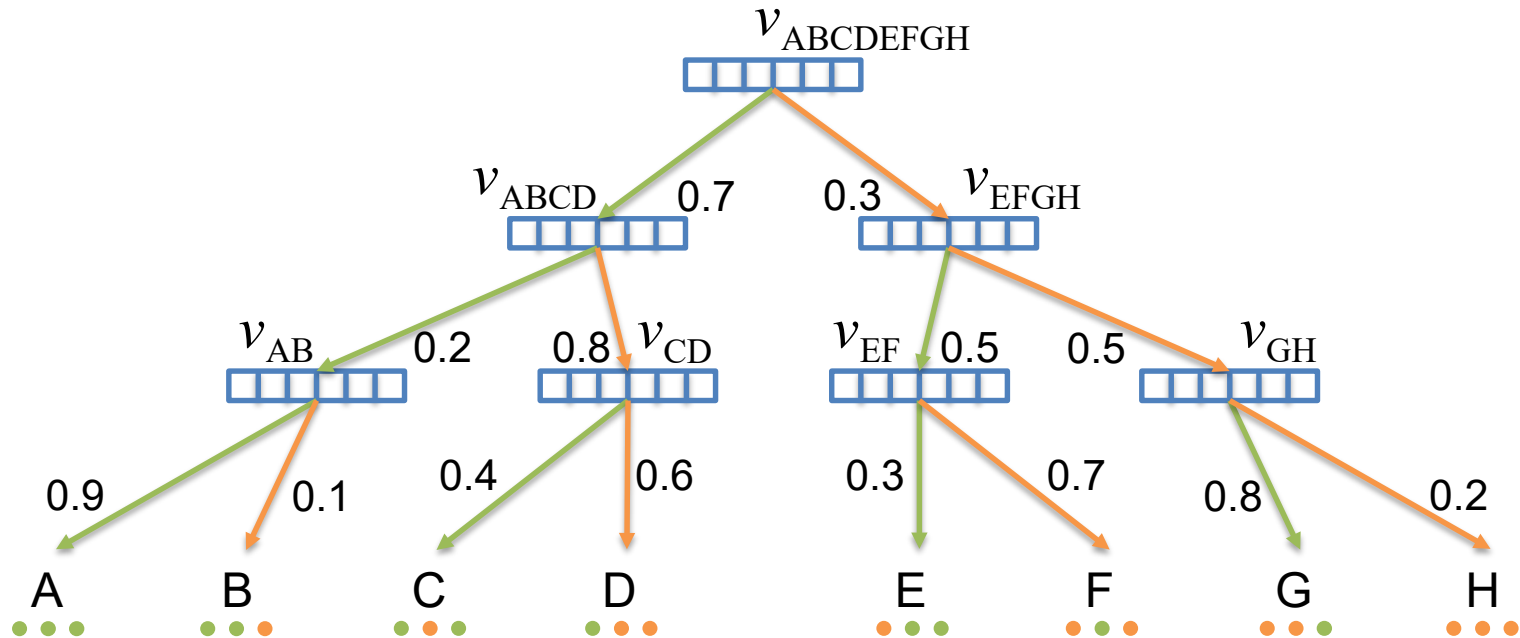


# The Training Process

---

- Negative Sampling
  - Noise contrastive estimation (NCE) posits that a good model should be able to differentiate data from noise
- Hierarchical Softmax
  - The main advantage is that it is needed to evaluate only about  $\log_2(V)$  nodes

# Hierarchical Softmax – 1



$$P(A|w) = 0.7 \times 0.2 \times 0.9$$

$$\vdots$$

$$P(D|w) = 0.7 \times 0.8 \times 0.6$$

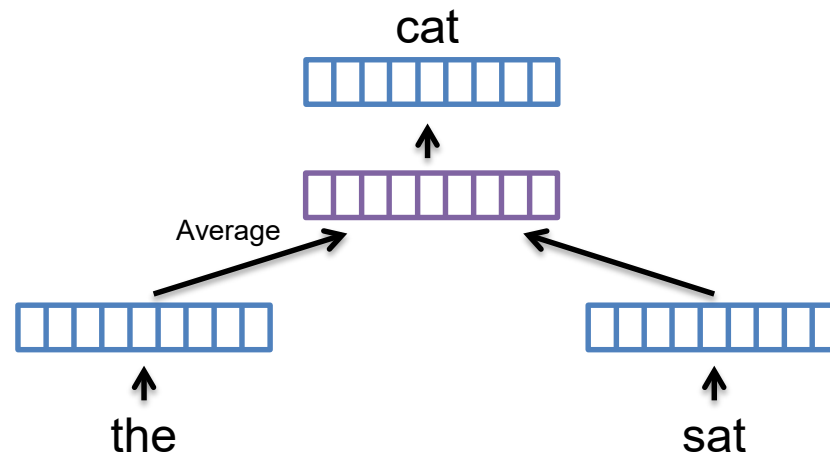
$$\vdots$$

$$P(H|w) = 0.3 \times 0.5 \times 0.2$$



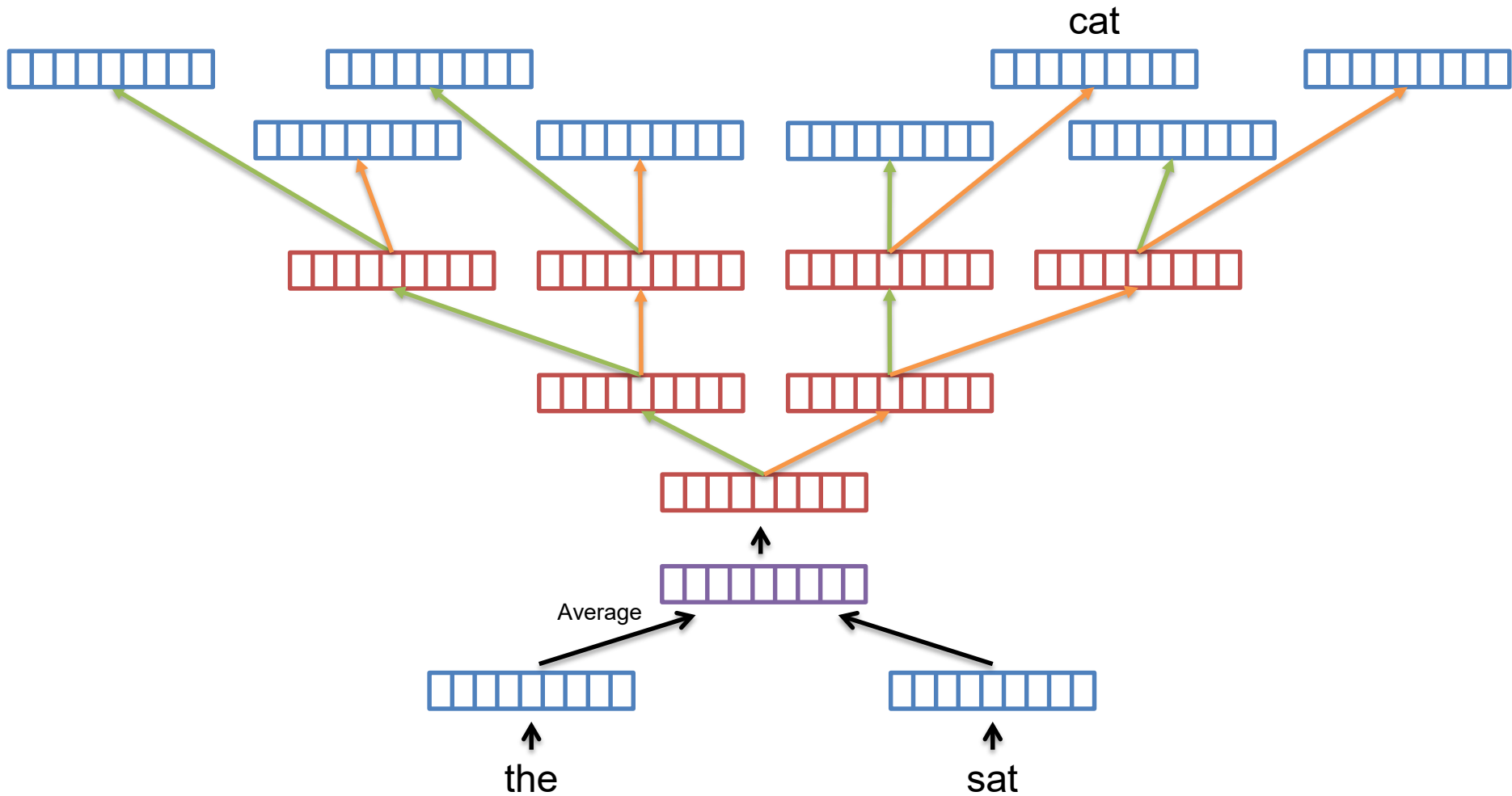
# Hierarchical Softmax – 2

---



# Hierarchical Softmax – 3

---



# Global Vector Model (GloVe)

- The idea is:
  - For word **solid** related to **ice** but not **steam**, we expect the ratio:
    - $P_{ice-solid} / P_{steam-solid}$  will be large
    - $P_{ice-gas} / P_{steam-gas}$  will be small
    - $P_{ice-fashion} / P_{steam-fashion}$  should be close to one
- The **starting point** for word vector learning should be with **ratios of co-occurrence probabilities** rather than the probabilities themselves

$$P(w_i, w_j) \propto c(w_i, w_j)$$

$$v_{w_i} \cdot v_{w_j} \propto \log(P(w_i, w_k)) \propto \log(c(w_i, w_k))$$

$$\sum_{i=1}^{|V|} \sum_{j=1}^{|V|} f(c(w_i, w_j)) \left( v_{w_i} \cdot v_{w_j} + b_i + b_j - \log(c(w_i, w_j)) \right)^2$$

# Classic Word Embeddings

---

- Various word embeddings have been proposed and applied to several NLP-related tasks
  - Prediction-based Methods
    - CBOW and Skip-gram
      - Local context
  - Count-based Methods
    - GloVe
      - Global matrix
- The **interpretation** of the learned value of each dimension in the representation is not intuitively clear

# Word Embeddings with VSM

---

- The vector space model usually refers to the combination of the **vector representations** and the **cosine similarity measure**

$$\text{sim}(d_m, d_n) = \frac{\vec{d}_m \cdot \vec{d}_n}{|\vec{d}_m| |\vec{d}_n|}$$

- A straightforward way to leverage the word embedding methods for NLP applications is to represent a document by averaging the vector representations of words occurring in the document

$$\vec{d}_m = \frac{1}{|d_m|} \sum_{i=1}^{|d_m|} v_{w_i} \quad \vec{d}_n = \frac{1}{|d_n|} \sum_{i=1}^{|d_n|} v_{w_i}$$

- Combined with traditional TF-IDF features can usually boost the performance!

$$\text{sim}(d_m, d_n) = \alpha \times \text{sim}_{TFIDF}(d_m, d_n) + (1 - \alpha) \times \text{sim}_{WE}(d_m, d_n)$$

# Word Embeddings with LM – 1

---

- In addition to the vector space model, we can construct a language model for a given document by using the word embeddings

$$\vec{d}_m = \frac{1}{|d_m|} \sum_{i=1}^{|d_m|} v_{w_i} \quad P(w_j|d_m) = \frac{\exp(v_{w_j} \cdot \vec{d}_m)}{\sum_{w \in V} \exp(v_w \cdot \vec{d}_m)}$$

$$\vec{d}_n = \frac{1}{|d_n|} \sum_{i=1}^{|d_n|} v_{w_i} \quad P(w_j|d_n) = \frac{\exp(v_{w_j} \cdot \vec{d}_n)}{\sum_{w \in V} \exp(v_w \cdot \vec{d}_n)}$$

- By doing so, the KL-divergence measure can be used to quantify the similarity degree between a pair of documents

$$KLD(d_m||d_n) = \sum_{w \in V} P(w|d_m) \log \frac{P(w|d_m)}{P(w|d_n)}$$


# Word Embeddings with LM – 2

- An opposite strategy to obtain a document language model is to leverage a translation mechanism

$$P(w_j|w_i) = \frac{\exp(v_{w_j} \cdot v_{w_i})}{\sum_{w \in V} \exp(v_w \cdot v_{w_i})}$$

- Consequently, the document model can be obtained by linearly combining the associated word-based language models of the words occurring in the document

$$P(w_j|d_m) = \sum_{i=1}^{|d_m|} \underbrace{P(w_j|w_i)P(w_i|d_m)}_{\text{Translation Model}}$$

Unigram Model 

$$P(w_j|d_n) = \sum_{i=1}^{|d_n|} P(w_j|w_i)P(w_i|d_n)$$

# Likelihood Measure

---

- Take the information retrieval task for example
  - For a given query  $q$ , we want to rank a set of documents
  - In the likelihood retrieval model, we rank documents by considering the probability that the query could be generated by the document language model

$$P(d_m|q) = \frac{P(q|d_m)P(d_m)}{P(q)} \propto P(q|d_m)P(d_m)$$
$$\approx P(q|d_m) \approx \prod_{i=1}^{|q|} P(w_i|d_m)$$

Document Model

- A document is treated as a model to predict (generate) the query



# Questions?

---



[kychen@mail.ntust.edu.tw](mailto:kychen@mail.ntust.edu.tw)