UVA CS 4774: Machine Learning

S3: Lecture 16:

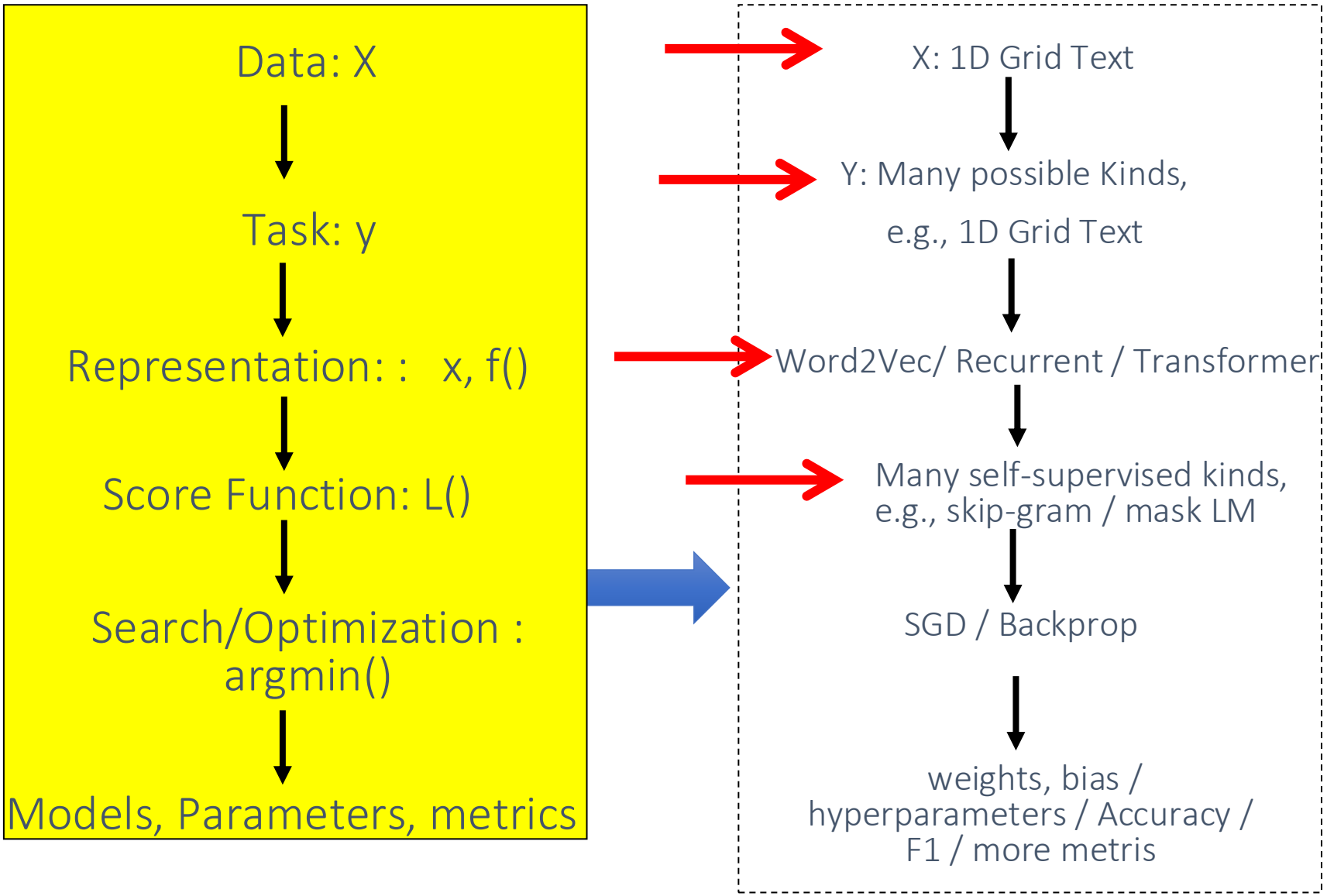
Deep Neural Networks for Natural Language Processing

Dr. Yanjun Qi

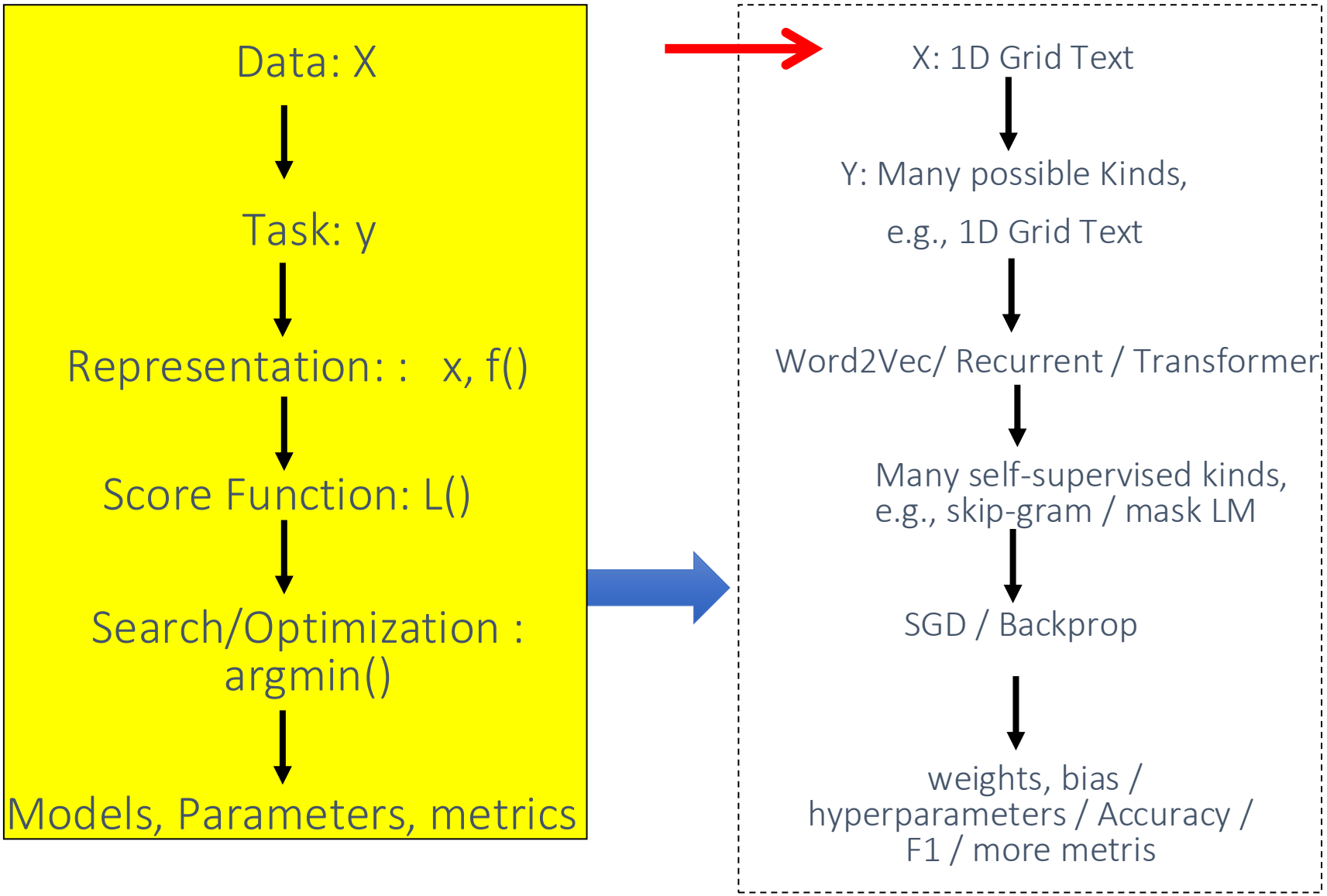
University of Virginia
Department of Computer Science

Module I

Today: Neural Network Models on 1D Grid / Language Data



Today: Neural Network Models on 1D Grid / Language Data



What is NLP

- **Wiki: Natural language processing** (**NLP**) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (**natural**) languages.



Credit: Kai-Wei Chang

Go beyond the keyword matching

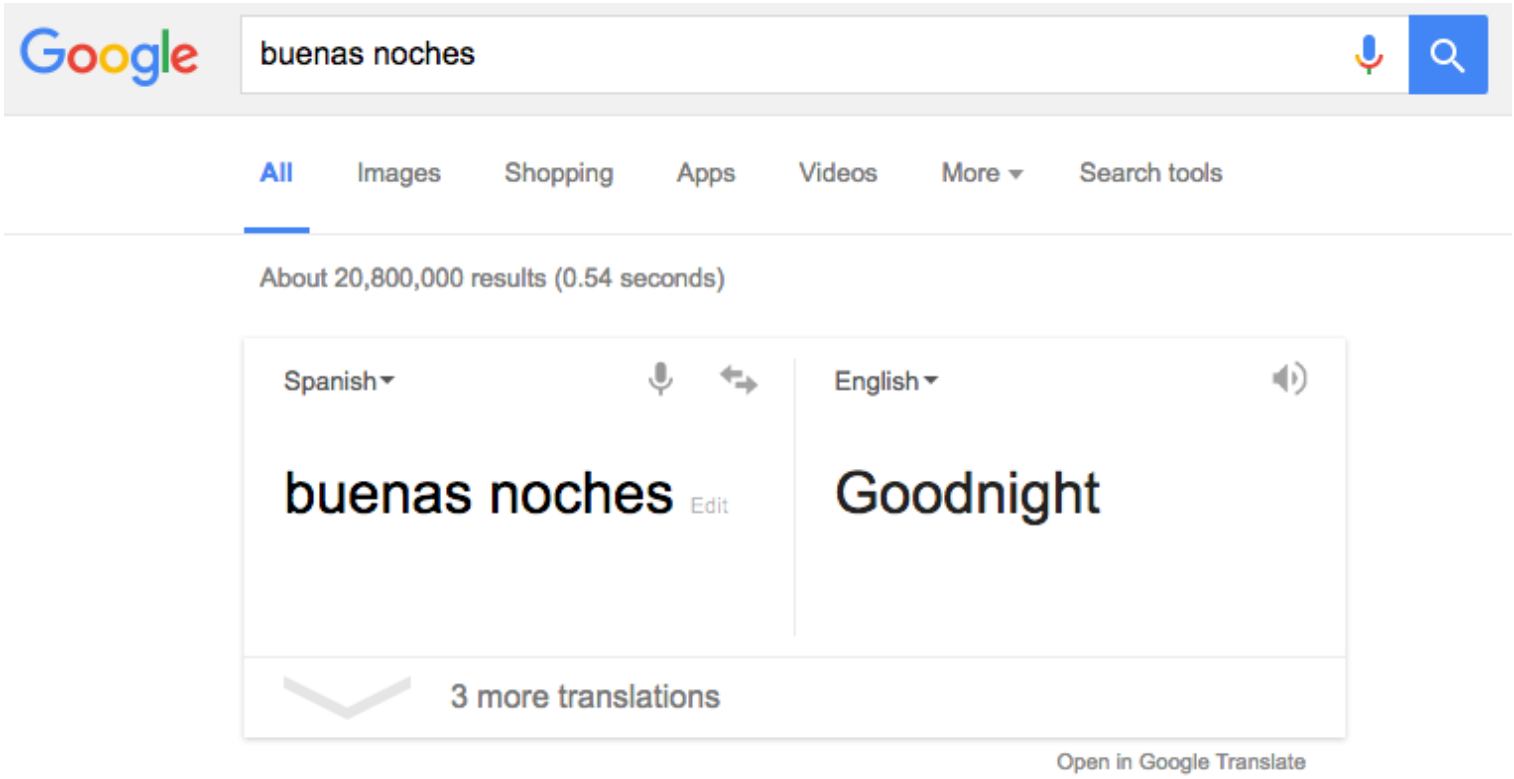




- Identify the structure and meaning of words, sentences, texts and conversations
- Deep understanding of broad language
- NLP is all around us

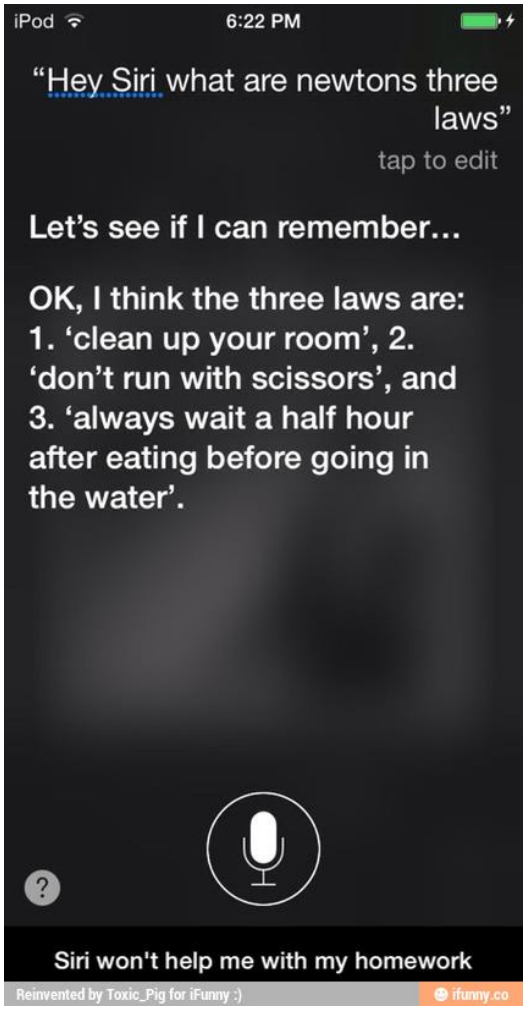
Credit: Kai-Wei Chang

Machine translation

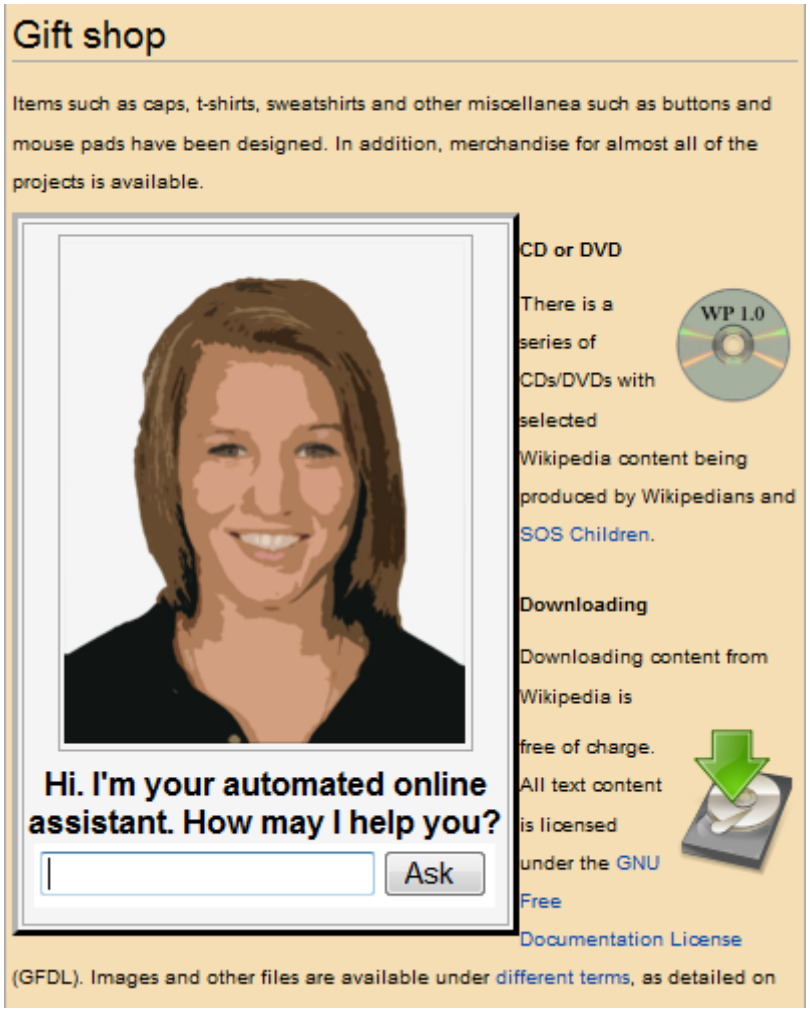


Credit: Kai-Wei Chang

Dialog Systems / Question answering

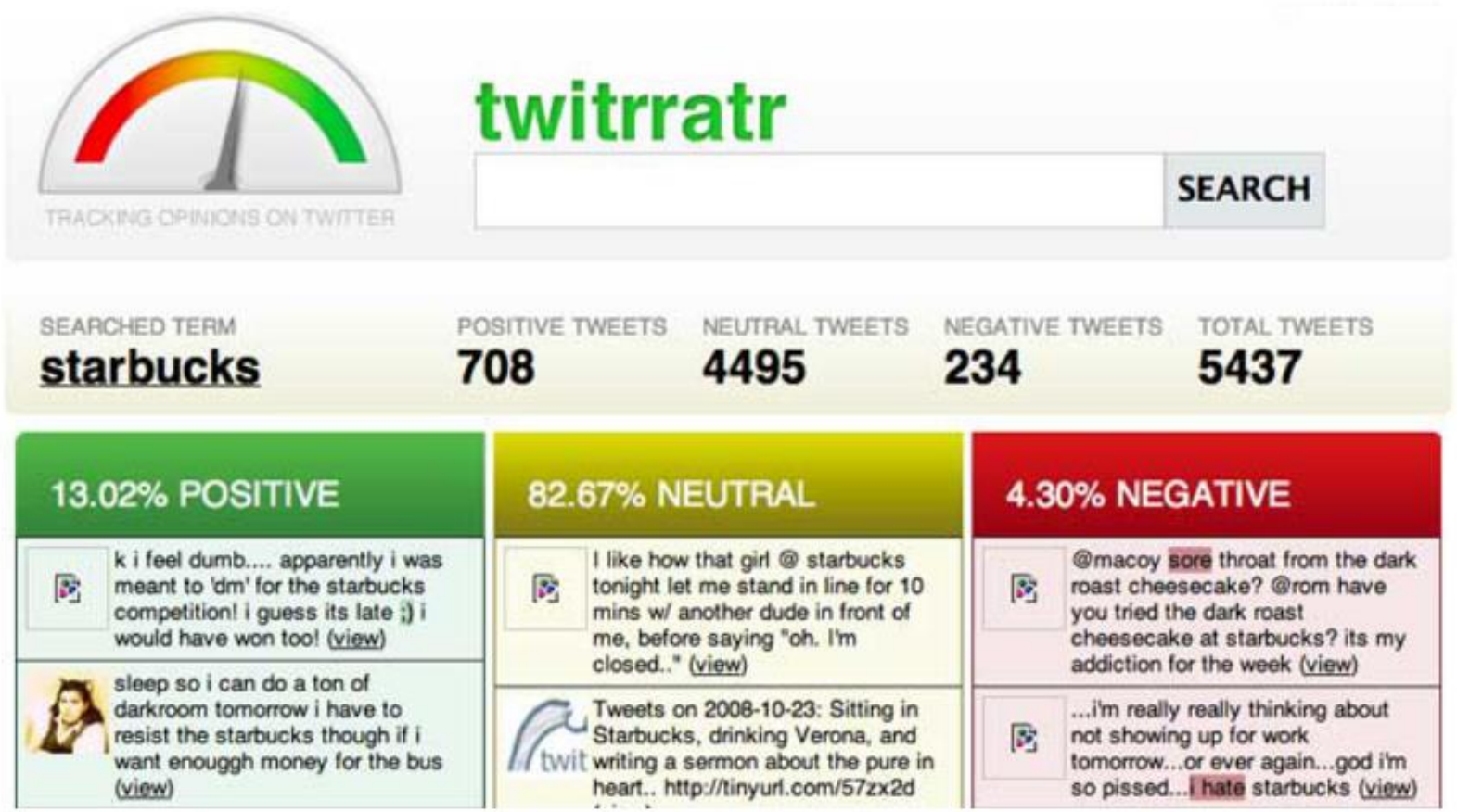


credit: ifunny.com



Credit: Kai-Wei Chang

Sentiment/Opinion Analysis

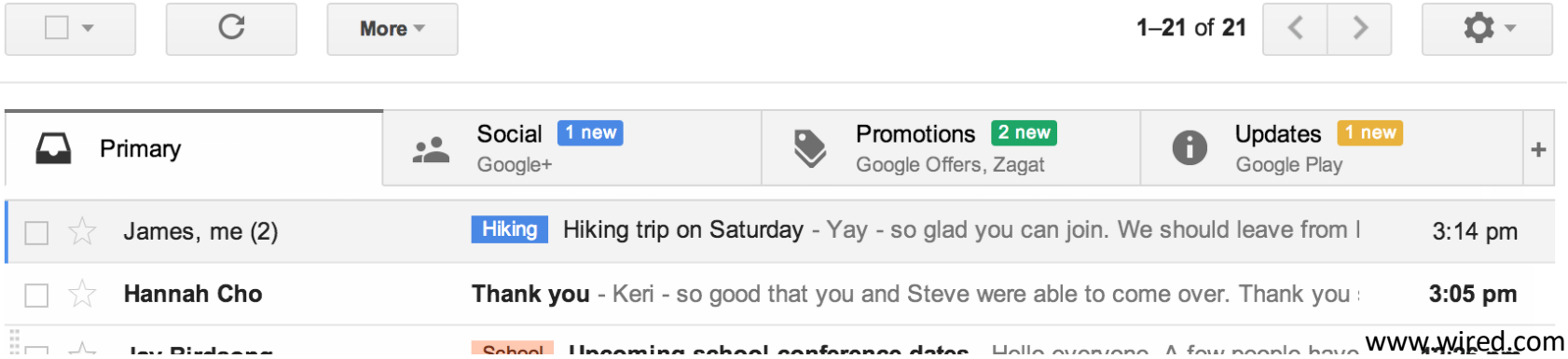


Credit: Kai-Wei Chang

Text Classification







www.wired.com

Credit: Kai-Wei Chang

Language Comprehension

Christopher Robin is alive and well. **He** is the same person that you read about in the book, **Winnie the Pooh.** As **a boy**, **Chris** lived in a pretty home called **Cotchfield Farm**. When **Chris** was three years old, **his father** wrote a poem about **him**. The poem was printed in a magazine for others to read. **Mr. Robin** then wrote a book

- Q: who wrote Winnie the Pooh?
- Q: where is Chris lived?

Credit: Kai-Wei Chang

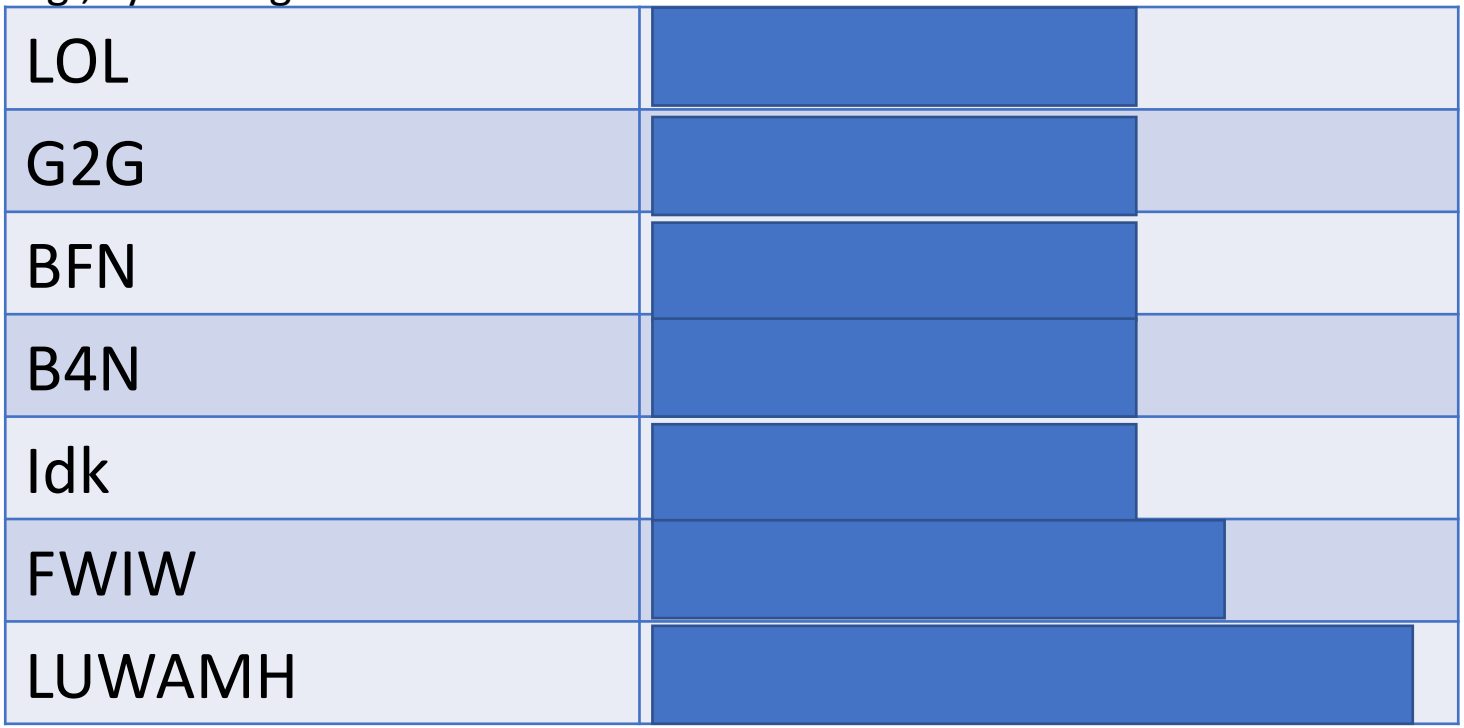
Natural language instruction



Credit: Kai-Wei Chang

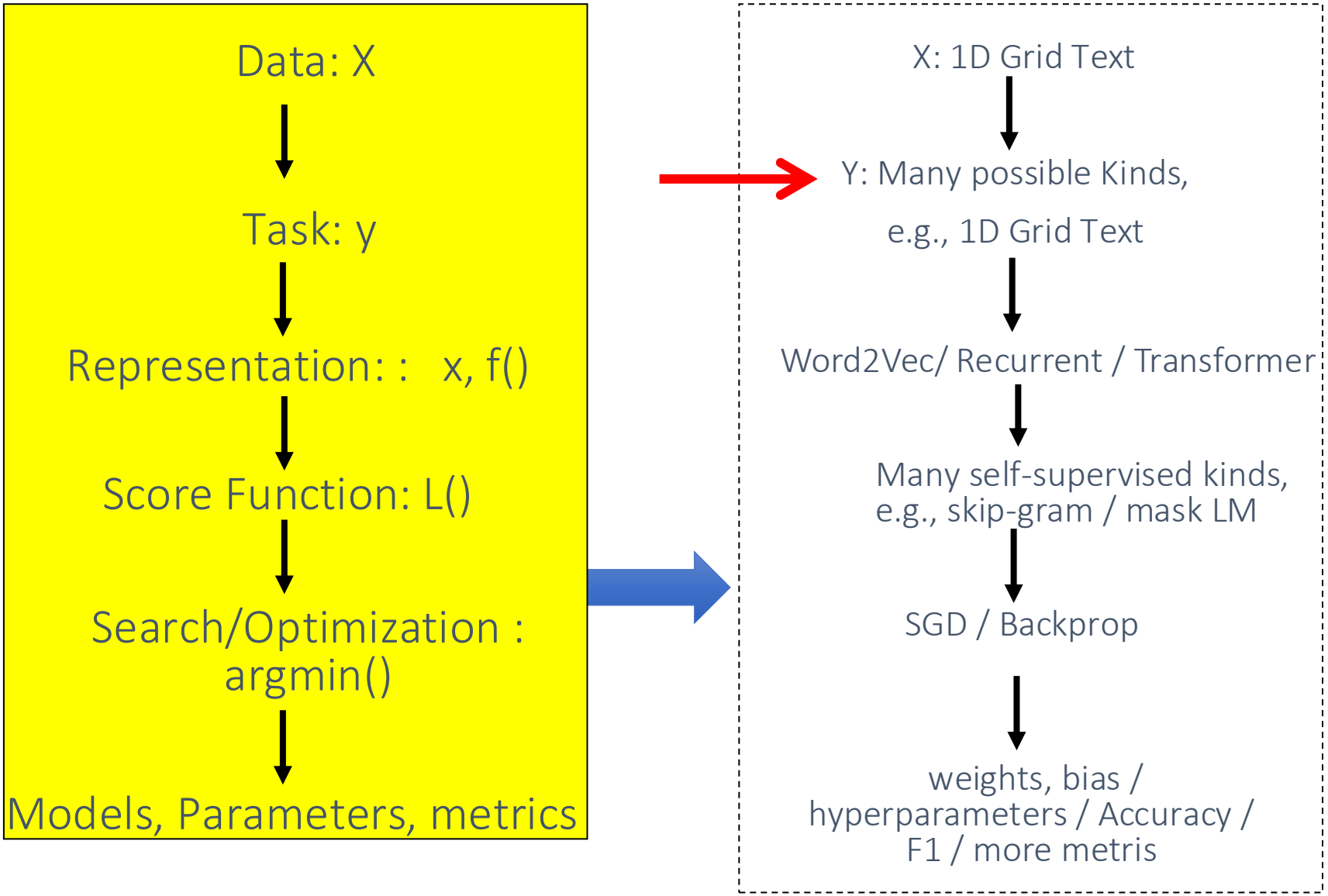
Challenges – language is not static

- Language grows and changes
  - e.g., cyber lingo



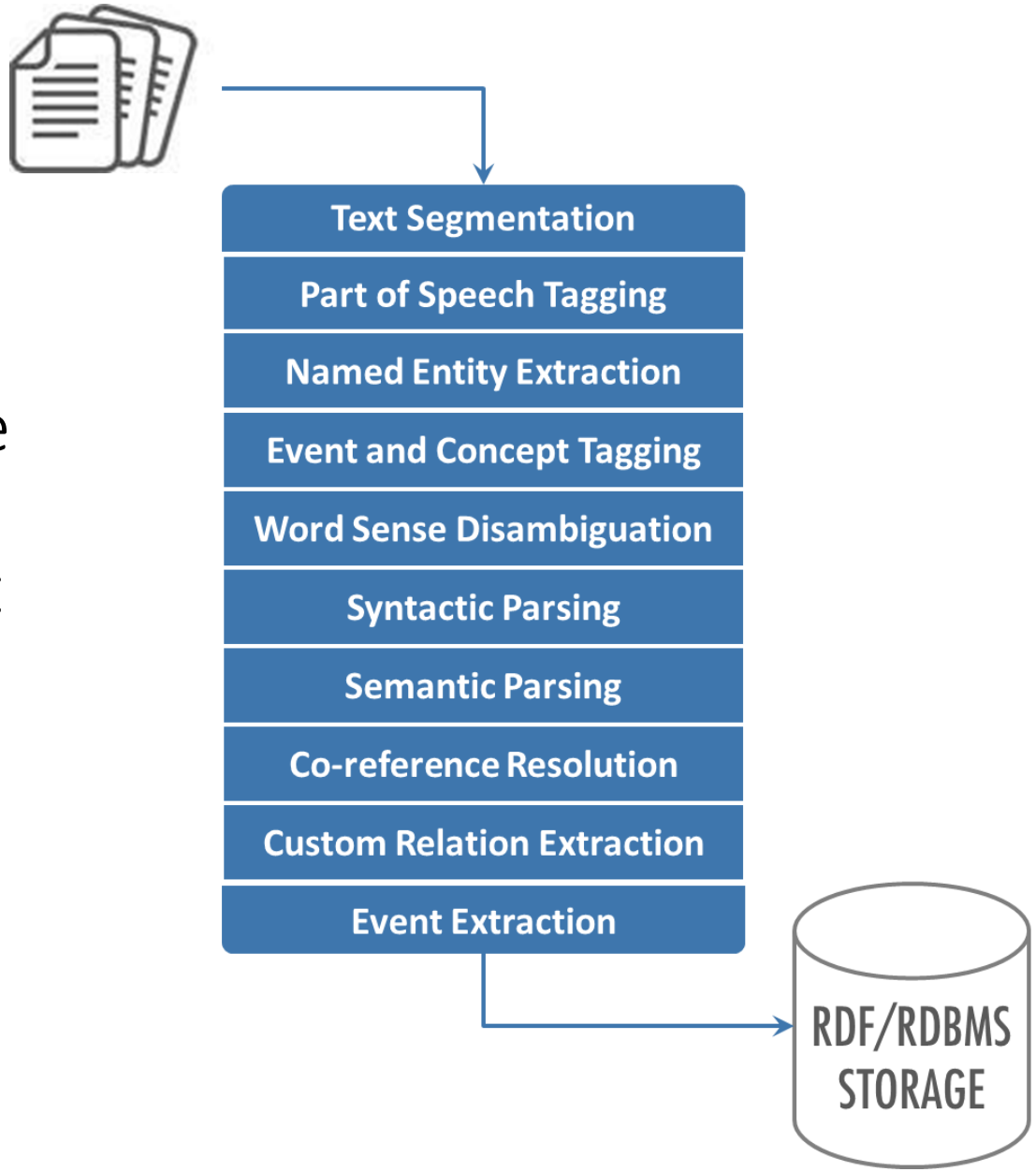
Credit: Kai-Wei Chang

Today: Neural Network Models on 1D Grid / Language Data



Y: Many possible Kinds

Classic NLP Pipeline Components for Understanding Text

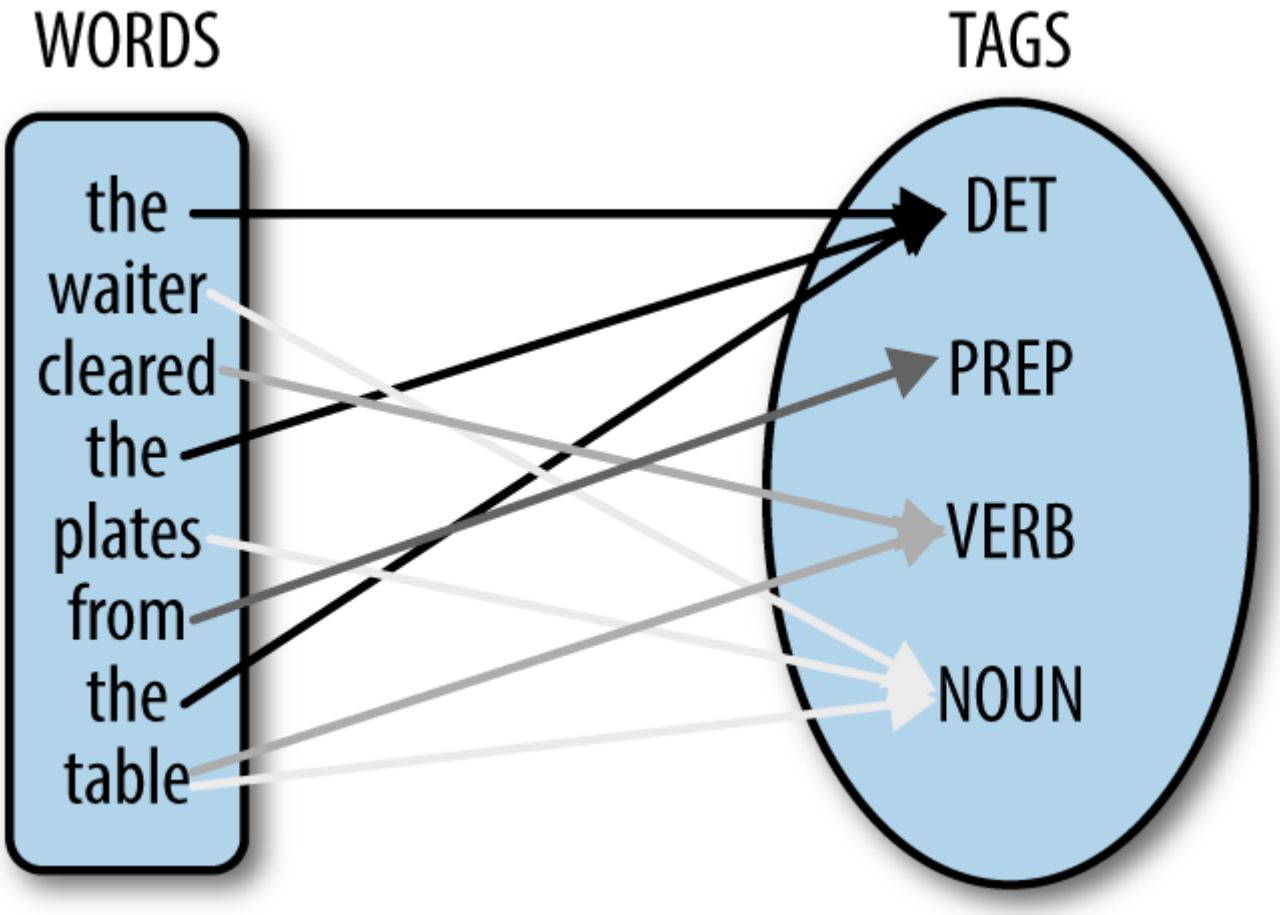


Credit: Kai-Wei Chang

Y: Many possible Kinds

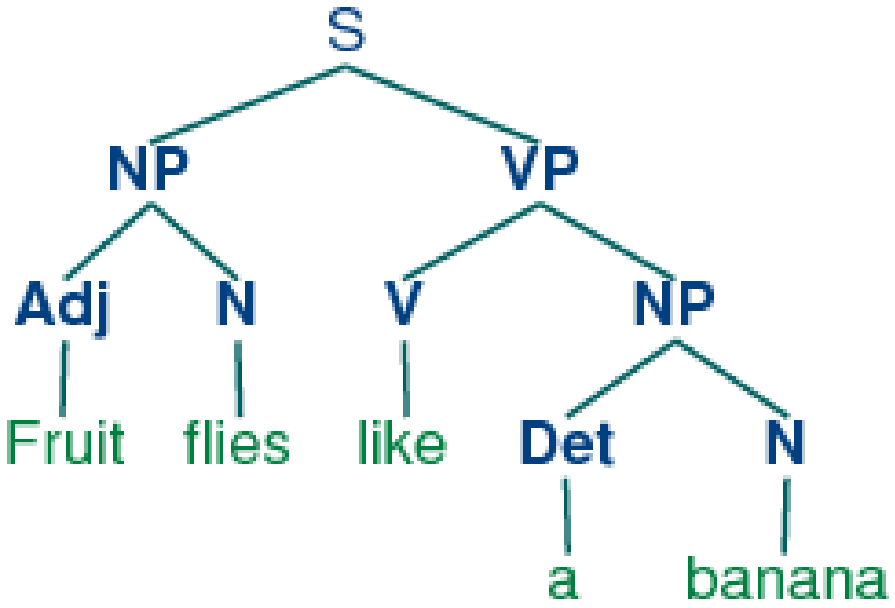
Part of speech tagging

➔ 1D Grid Outputs!



Credit: Kai-Wei Chang

Syntactic (Constituency) parsing



Credit: Kai-Wei Chang

Syntactic structure => meaning

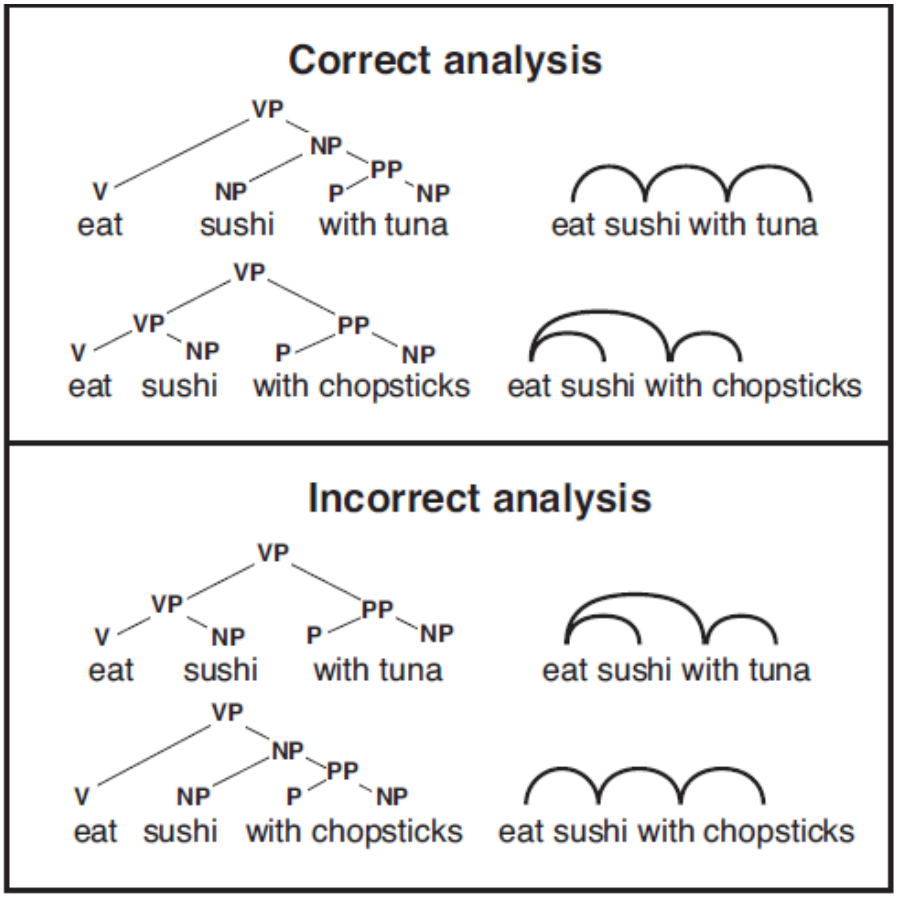


Image credit: Julia Hockenmaier, Intro to NLP

Y: Many possible Kinds

Semantic analysis

Q: [Chris] = [Mr. Robin] ?

Christopher Robin is alive and well. **He** is the same person that you read about in the book, **Winnie the Pooh.** As **a boy**, **Chris** lived in a pretty home called **Cotchfield Farm**. When **Chris** was three years old, **his father** wrote a poem about **him**. The poem was printed in a magazine for others to read. **Mr. Robin** then wrote a book

Slide from Dan Roth

UVA CS 4774: Machine Learning

S3: Lecture 16:

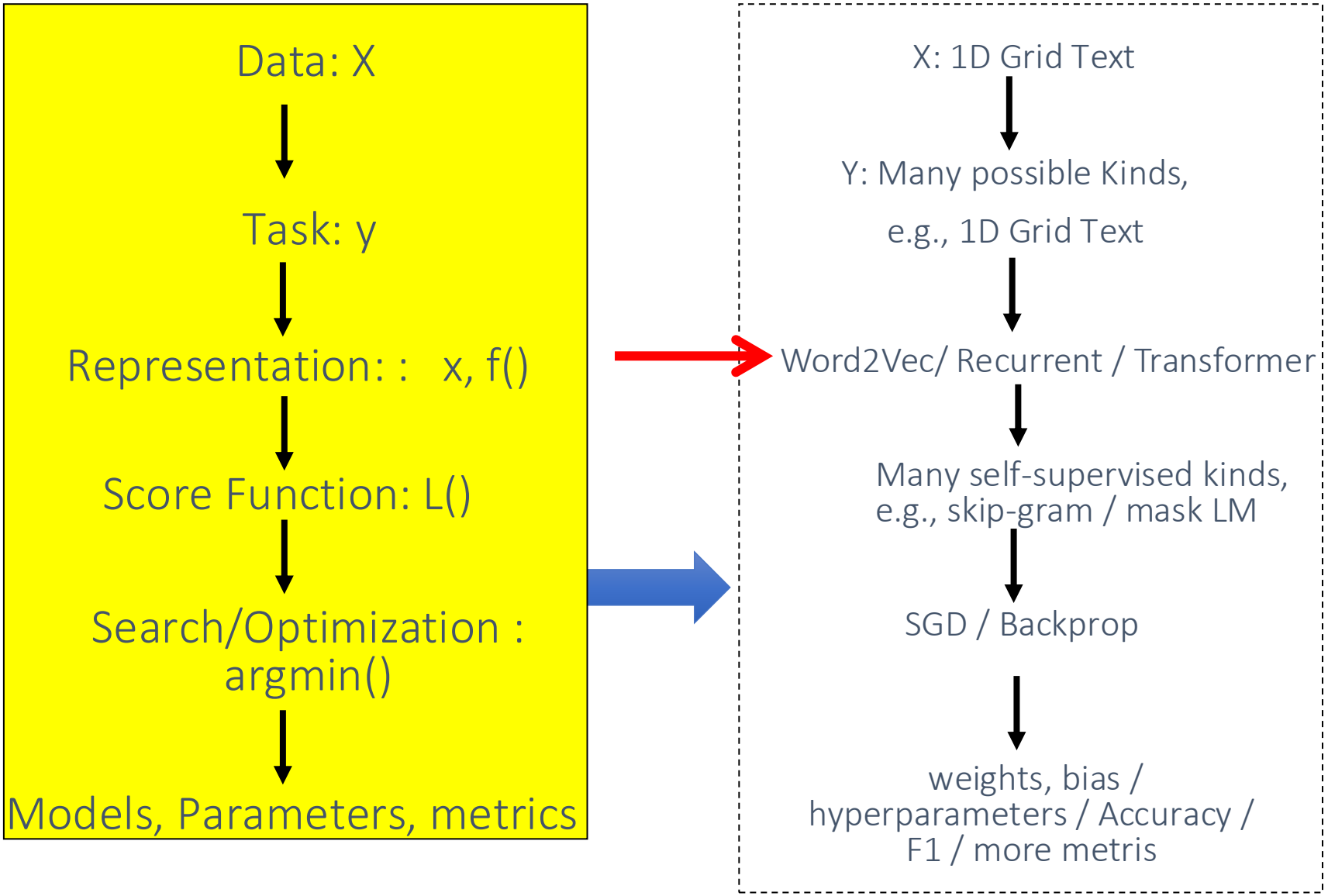
Deep Neural Networks for Natural Language Processing

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Module II

Today: Neural Network Models on 1D Grid / Language Data



Roadmap : f() on natural language inputs



- Before Deep NLP (Pre 2012)
  - (BOW / LSI / Topic LDA)
- Word2Vec (2013-2016)
  - (GloVe/ FastText)
- Recurrent NN (2014-2016)
  - LSTM
  - Seq2Seq
- Attention / Self-Attention (2016 – now)
  - Attention
  - Transformer (self-attention, attention only)
  - BERT / XLNet/ GPT-2 / T5 …

Text document classification, e.g. spam email filtering

- Input: document D
- Output: the predicted class variable C, from $\{c_1,\ldots,c_L\}$

- E.g.,
  - Spam filtering Task: Classify email as {'Spam', 'Other'}.

From: "" <takworlld@hotmail.com>
Subject: real estate is the only way... gem oalvgkay

Anyone can buy real estate with no money down
Stop paying rent TODAY !

Change your life NOW by taking a simple course!
Click Below to order:
http://www.wholesaledaily.com/sales/nmd.htm



P (C=spam | D)

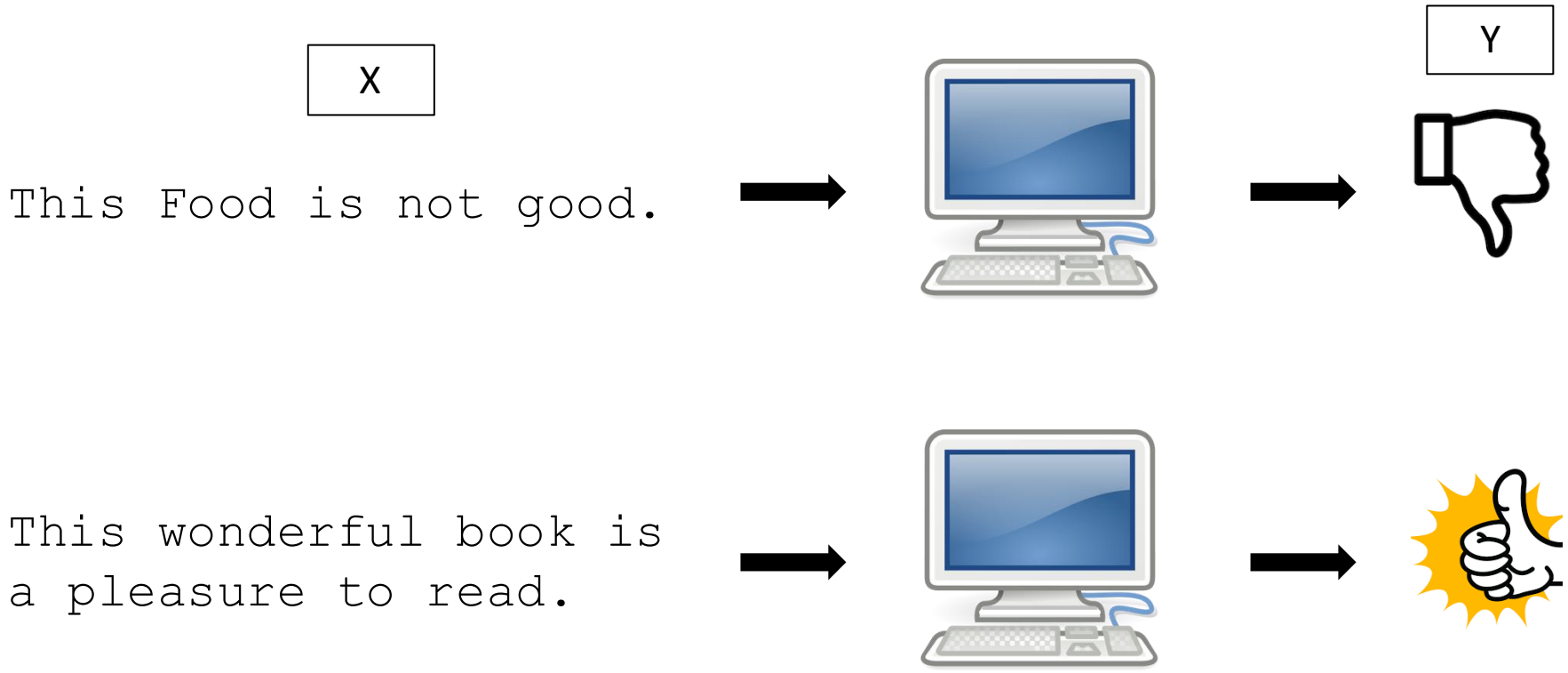
Text classification Tasks

- Input: document D
- Output: the predicted class C, c is from $\{c_1,...,c_L\}$

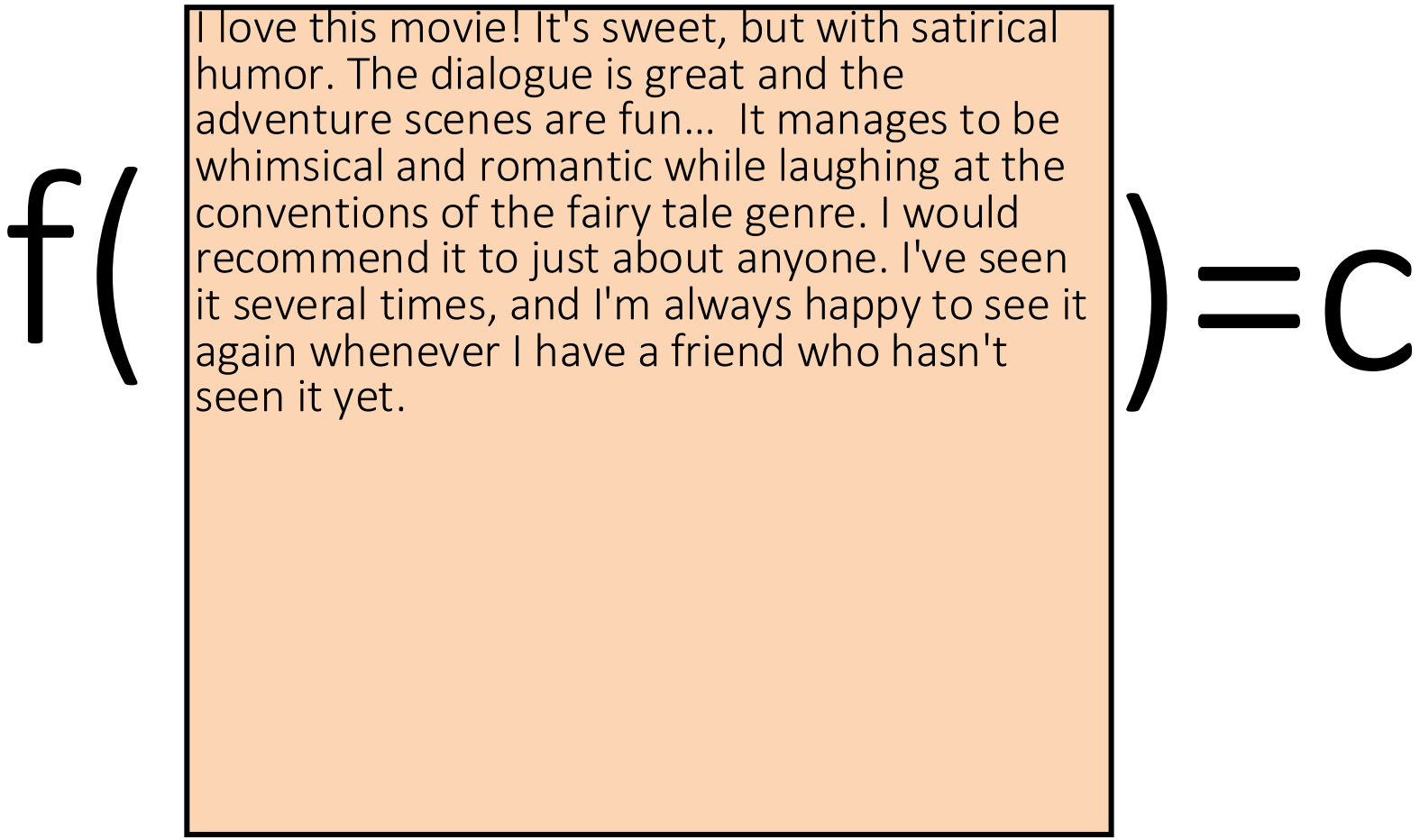
Text classification applicatoins:

- Classify email as 'Spam', 'Other'.
- Classify web pages as 'Student', 'Faculty', 'Other'
- Classify news stories into topics 'Sports', 'Politics'..
- Classify movie reviews as 'Favorable', 'Unfavorable', 'Neutral'
- … and many more.

Variable Length Representation for Natural Language based Classification:



BOW: The bag of words representation



BOW: The bag of words representation



Representing text:
a list of words ➔ Dictionary

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun… It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet.



word	frequency
great	2
love	2
recommend	1
laugh	1
happy	1
…	.

Common refinements: remove stopwords, stemming, collapsing multiple occurrences of words into one….



'Bag of words' representation of text

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun… It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet.



word	frequency
great	2
love	2
recommend	1
laugh	1
happy	1
…	.

Bag of word representation:

Represent text as a vector of word frequencies.

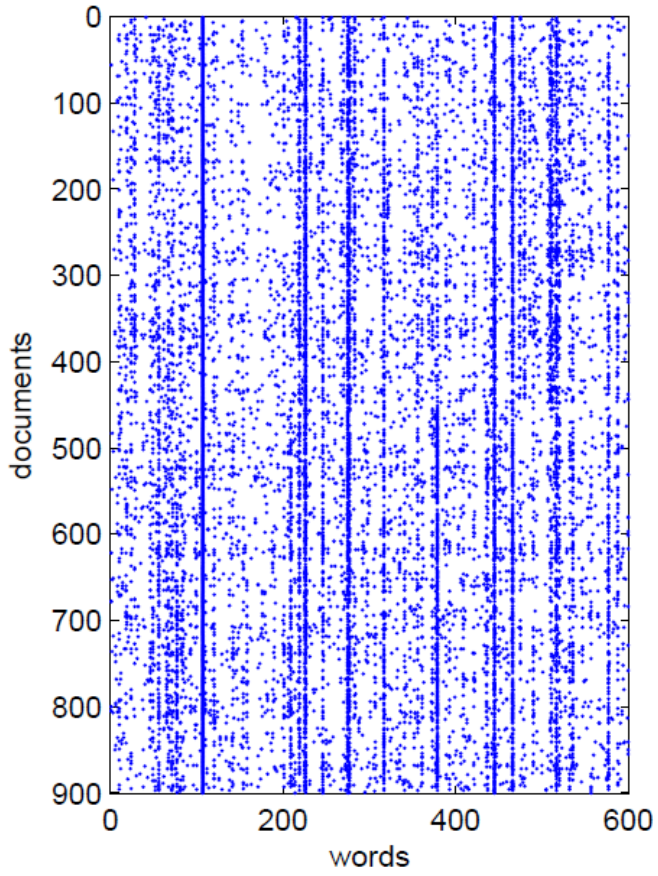
$$D = (w_1, w_2, \cdots, w_K)$$

Bag of words

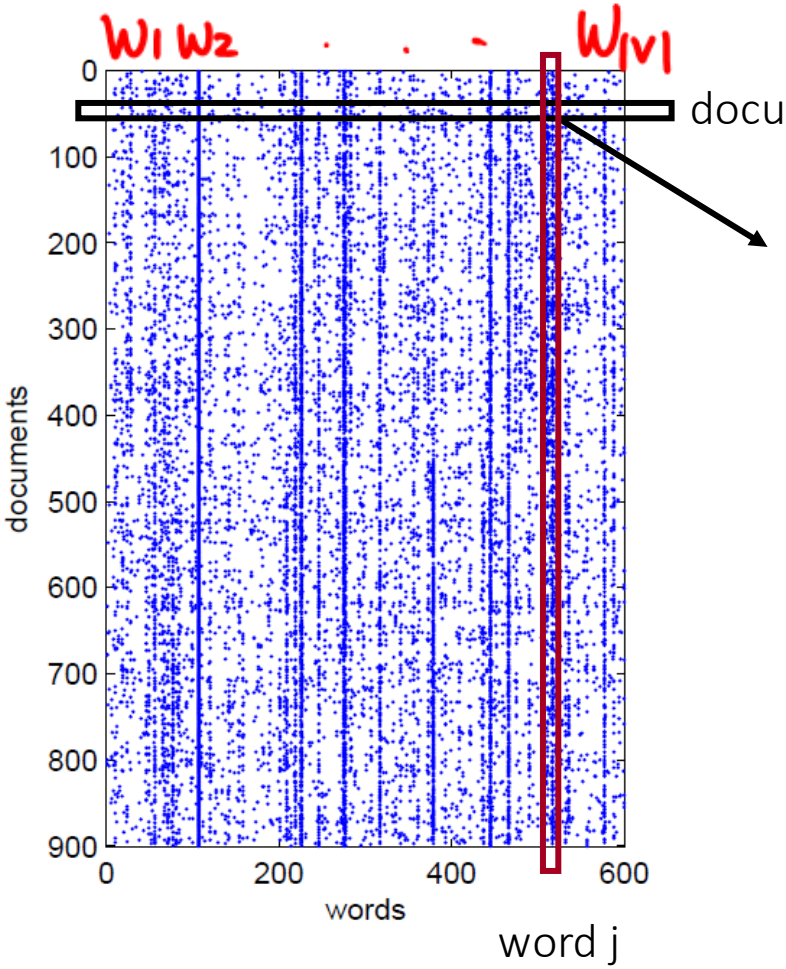
- What simplifying assumption are we taking?

$D = (w_1, w_2, \cdots, w_K)$

We assumed word order
is not important.



Bag of words representation



Term-document matrix.

e.g., X (i,j) = Frequency of word j in document i

➔ Tabular

A collection of documents

	X_1	X_2	X_3	C
s_1				
s_2				
s_3				
s_4				
s_5				
s_6			30	

Unknown Words

- How to handle words in the test corpus that did not occur in the training data, i.e. out of vocabulary (OOV) words?
- Train a model that includes an explicit symbol for an unknown word (<UNK>).
  - Choose a vocabulary in advance and replace other (i.e. not in vocabulary) words in the corpus with <UNK>.
  - Very often, <UNK> also used to replace rare words

EXTRA: Advanced BOW: tf-idf

1. Construct a term-document matrix.
2. Convert word counts to tf-idf scores.

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

A term-document matrix (following is the transpose view).

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Figure 6.8 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

A tf-idf matrix (following is the transpose view).

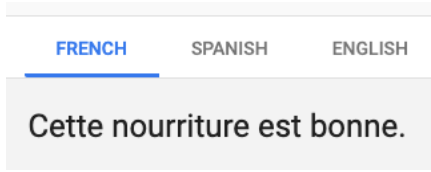
But: BOW NOT Applicable to many NLP tasks:

- removes position information and can not (or hard to) represent word compositions

This Food is good.

X

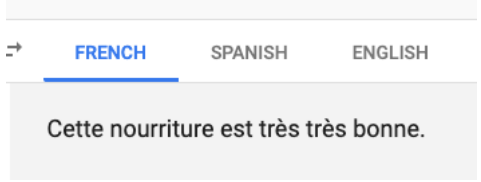




Y: French Translation

This Food is very very good.





UVA CS 4774: Machine Learning

S3: Lecture 16:

Deep Neural Networks for Natural Language Processing

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

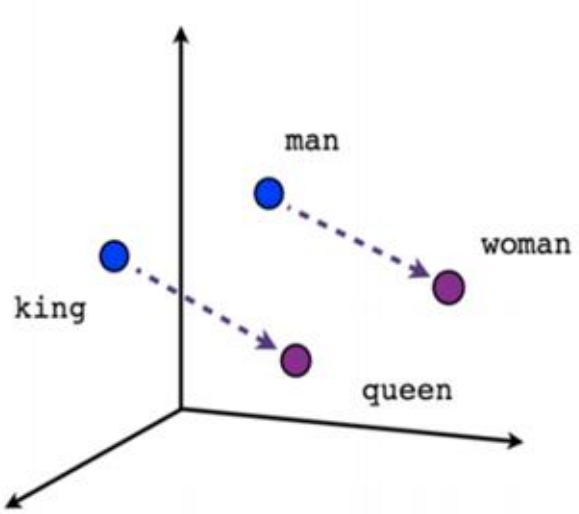
Module III

Roadmap : f() on natural language

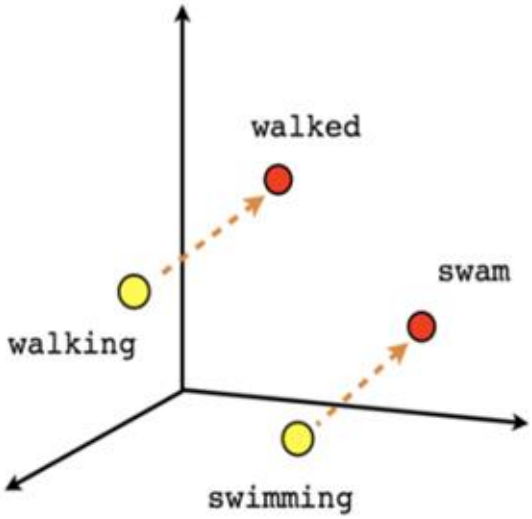
- Before Deep NLP (Pre 2012)
  - (BOW / LSI / Topic LDA)
- Word2Vec (2013-2016)
  - (GloVe/ FastText)
- Recurrent NN (2014-2016)
  - LSTM
  - Seq2Seq
- Attention / Self-Attention (2016 – now)
  - Attention
  - Transformer (self-attention, attention only)
  - BERT / XLNet/ GPT-2 / T5 …

How to Represent A Word in DNN

- Basic approach – "one hot vector"
  - Binary vector
  - Length = | vocab |
  - 1 in the position of the word id, the rest are 0
  - However, does not represent word meaning
  - Extremely high dimensional (there are over 200K words in the English language)
  - Extremely sparse



Male-Female

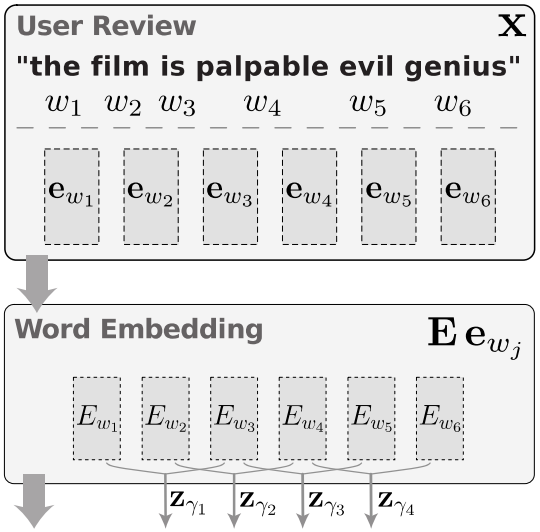


Verb tense

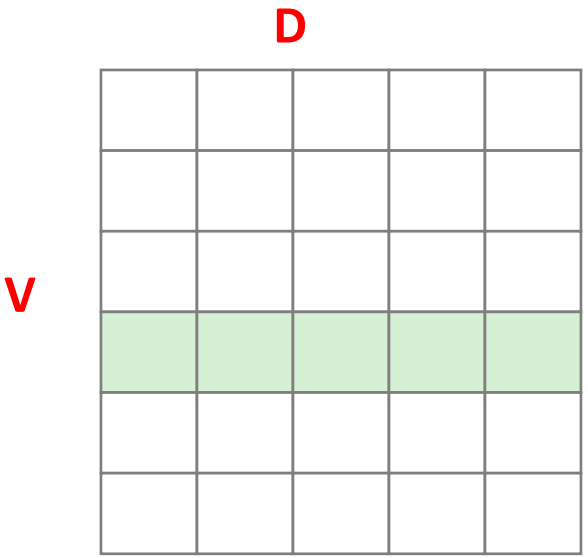
Solution: Distributional Word Embedding Vectors

Popular word embeddings

- Word2Vec:
  - Mikolov et al. 2013
- GloVe (Global Vectors)
  - Pennington et al., 2014
- fasttext
  - Bojanowski et al., 2017
- Elmo
  - Peters, 2018
- BERT
  - Devlin et al., 2018



The lookup table E: V × D weights to learn



$$v_w = x_w^\top E$$

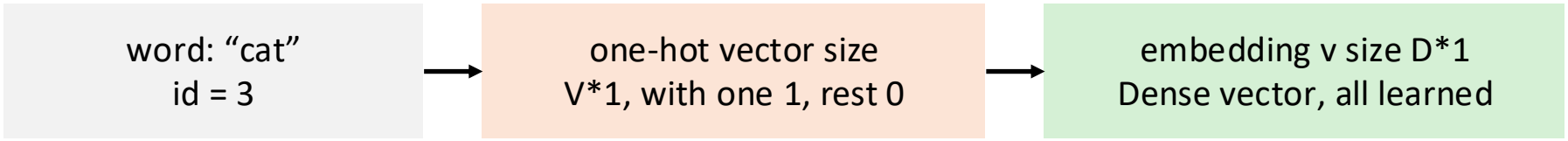
$$(1 \times V)\,(V \times D) = (1 \times D)$$

- Multiplying by a one-hot vector just picks row w
- **Every entry is a weight**: V · D parameters
- Starts random, then training moves the rows
- Example: V = 10,000, D = 300 gives 3,000,000 weights

```
E = nn.Embedding(num_embeddings=V, embedding_dim=D)     # E.weight: V × D
v = E(torch.tensor([3]))                               # row 3, shape (1, D)
```


credit: PyTorch docs, torch.nn.Embedding (weights start from N(0, 1))

Dimensionality of Word Embedding Operation



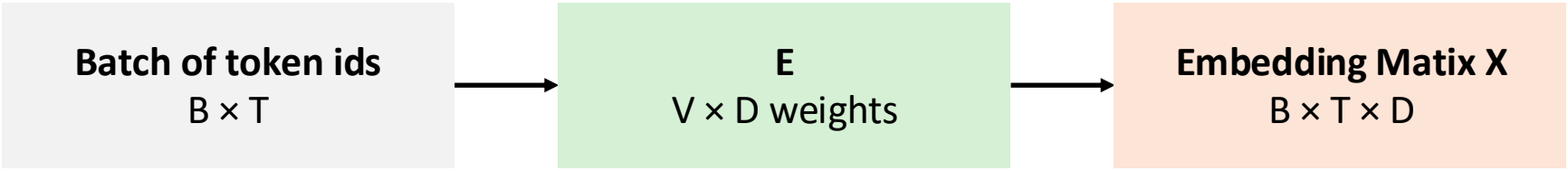
	One-hot x	Embedding v
Length	V (e.g. 10,000)	D (e.g. 300)
Non-zero entries	1	all D
cat vs dog	dot product 0: no similarity	learned: close together
Where it comes from	fixed by the word id	trained from data

D ≪ V: dense vectors usually have 50–1000 dimensions

Rough real use case sizes:
V ≈ 200,000 words, D ≈ 1,024 → E has 200,000 × 1,024 ≈ 205 million weights to learn

credit: Jurafsky & Martin, SLP3 draft, ch. 5

E on a sentence: follow the shapes



B = batch size, T = words per sentence. Each id is replaced by its row in the lookup table E.

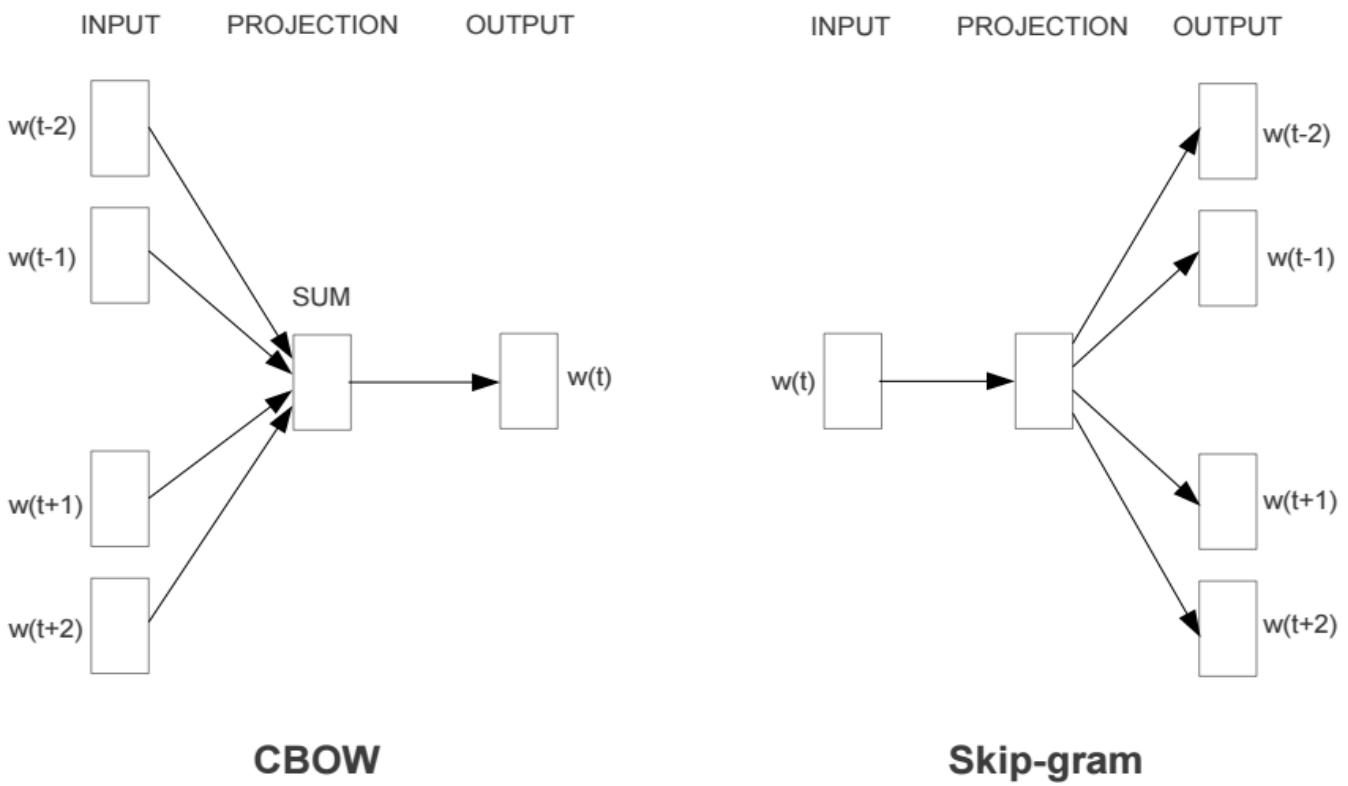
Embedding Size Cases	V	D	E Weights V × D
Our example	10,000	300	3 million
word2vec Google News	3,000,000	300	900 million
GPT-2	50,257	768	38.6 million

E grows with the vocabulary, not with the sentence length

credit: gensim-data list; GPT-2 config (Hugging Face); products computed

Word2vec –Vector Representation of Words (Mikolov et al. 2013)

- Word2vec: computationally-efficient, 2-layer predictive NN for learning word embeddings from raw text



CBOW / SkipGram (Basic Word2Vec) (Mikolov et al. 2013)

- Skip-gram learning:
  - Given $w_{0,}$ predict w_{-2}, w_{-1}, w_1, and w_2

w_{-2}	w_{-1}	w_0	w_1	w_2
?	?	Network	?	?



w_{-2}	w_{-1}	w_0	w_1	w_2
Recurrent	Neural		Language	Model

- Conversely, CBOW tries to predict w_0 when given w_{-2}, w_{-1}, w_1, and w_2

Word2vec –Vector Representation of Words (Mikolov et al. 2013)

- Word2vec comes with two models:

Model	Approach	Speed and Performance	Use case
Continuous Bag-of-Words model (CBOW)	The CBOW predicts the current word based on the context.	Faster to train than the skip-gram model	Predicts frequent words better
Skip-Gram model	Skip-gram predicts surrounding words given the current word.	Usually performs better than CBOW	Predicts rare words better

Skip-gram training data: (target, context) pairs

Slide a window of size m = 2 over the text: **a tablespoon of apricot jam**

Target w	Context c (within 2 words)	Negative samples (k = 1)	Real pairs
a	tablespoon, of	river, blue	2
tablespoon	a, of, apricot	tree, car, piano	3
of	a, tablespoon, apricot, jam	banana, lamp, road, sky	4
apricot	tablespoon, of, jam	door, fish, aardvark	3
jam	of, apricot	moon, cup	2
	Total	**14 fake pairs**	**14**

- **Label 1:** real pair (w, c). **Label 0:** fake pair (w, random word)
- A corpus of N words gives about 2mN pairs; 1 billion words, m = 5 → about 10 billion pairs
- Stored as word ids: a batch is a B × 3 table of (target id, context id, noise id)

No human labels: the text itself supplies every real pair

"Real" means the pair actually occurs in the text within the window, unlike the fake pairs made from random words. For example, the "apricot" row has context words "tablespoon, of, jam", so it makes 3 real pairs: (apricot, tablespoon), (apricot, of) and (apricot, jam).

credit: pairs computed; Mikolov et al. 2013; example from Jurafsky & Martin, SLP3 draft

Training Signals: the vector of a word should be close to the vector of each of its neighbors.

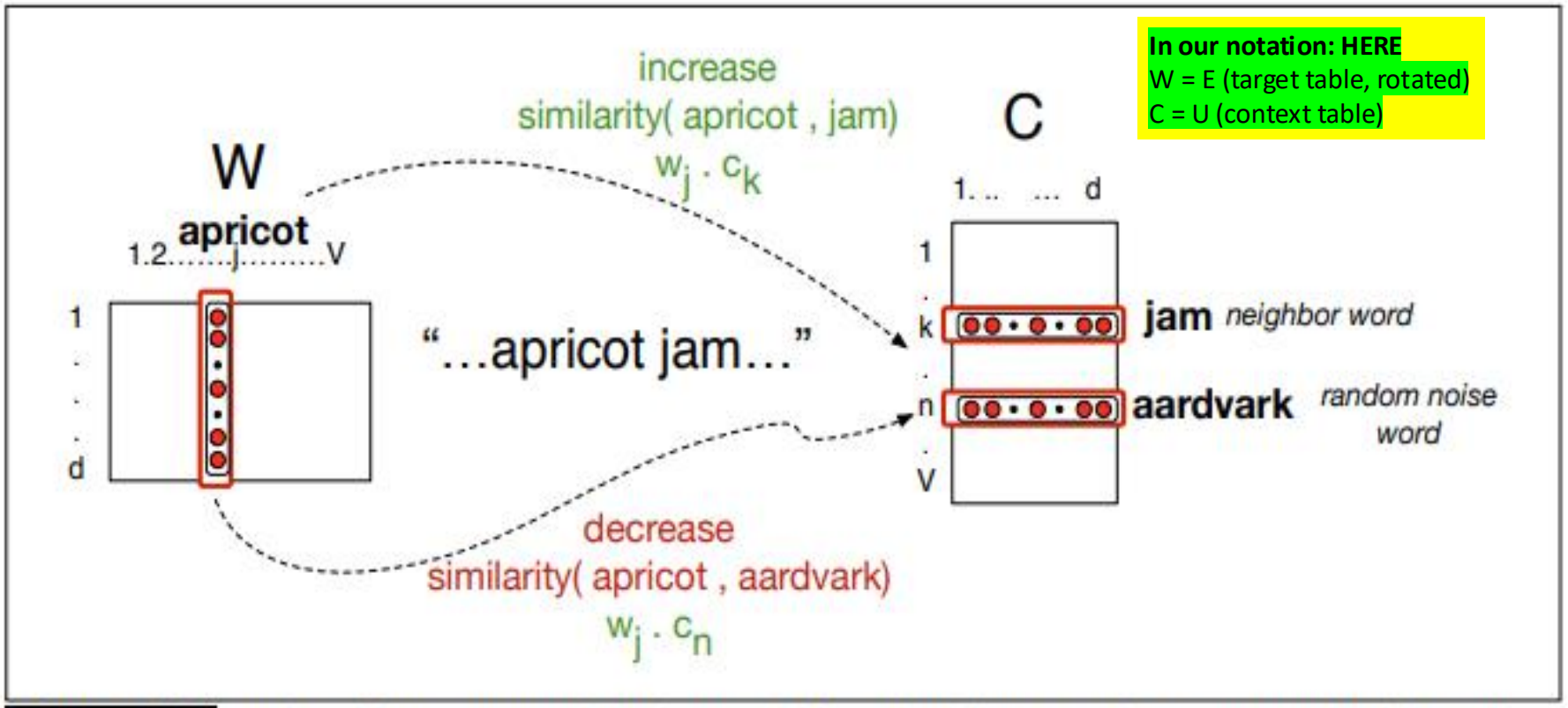


Figure 6.13 The skip-gram model tries to shift embeddings so the target embedding (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (have a lower dot product with) context embeddings for words that don't occur nearby (here *aardvark*).

Negative Sampling based Binary Cross-Entropy Loss (or Log-Sigmoid Loss) framed as a binary classification task.

$$J_{\text{NEG}} = -\log \sigma(u_{w_{\text{context}}}^{\top} v_{w_{\text{target}}}) - \sum_{i=1}^{K} \log \sigma(-u_{w_{n_i}}^{\top} v_{w_{\text{target}}})$$

Skip-gram equations: two V × D tables to learn!!

Original Skip-Gram Loss:

$$P(c \mid w) = \frac{\exp(u_c^\top v_w)}{\sum_{k=1}^{V} \exp(u_k^\top v_w)}$$

$$J = -\frac{1}{N} \sum_{t=1}^{N} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t)$$

Symbol	Meaning	Shape
v_w	row of E (target)	D
u_c	row of U (context)	D
E, U	Two weights to learn	V × D each
m	window size	number

- Softmax over the vocabulary: a V-way classifier for each pair
- **Weights to learn: 2 · V · D** (6 million in our example)
- Problem: the bottom line sums over all V words, for every pair
- After training, keep E and usually drop U

Training Signals: the vector of a word should be close to the vector of each of its neighbors.

credit: Mikolov et al. 2013; Jurafsky & Martin, SLP3 draft, ch. 5

Negative sampling: from V to k + 1

Revised Skip-Gram Loss:

$$J = -\log \sigma(u_c^\top v_w) \; - \; \sum_{i=1}^{k} \log \sigma(-u_{n_i}^\top v_w)$$

$$\frac{\partial J}{\partial v_w} = -\left(1 - \sigma(u_c^\top v_w)\right) u_c \; + \; \sum_{i=1}^{k} \sigma(u_{n_i}^\top v_w) \, u_{n_i}$$

- Real pair (w, c): push the dot product up
- k random words n: push the dot product down
- Replaces one V-way softmax with k + 1 yes/no questions

Per training pair	Softmax Full (V=)	Negative sampling (k = 5)
Dot products	10,000	6
Rows of U updated	10,000	6

credit: Mikolov et al., NeurIPS 2013; Jurafsky & Martin, SLP3 draft, ch. 5; counts computed

Quiz Check: count the weights

- V = 50,000 and D = 100. What is the shape of E? How many weights?
- A batch has 8 sentences of 20 words each. What is the shape of X?
- Skip-gram with both tables: how many weights in total?
- Doubling the sentence length: what happens to E? To X?
- With k = 10, how many dot products does one training pair need? And with the full softmax?

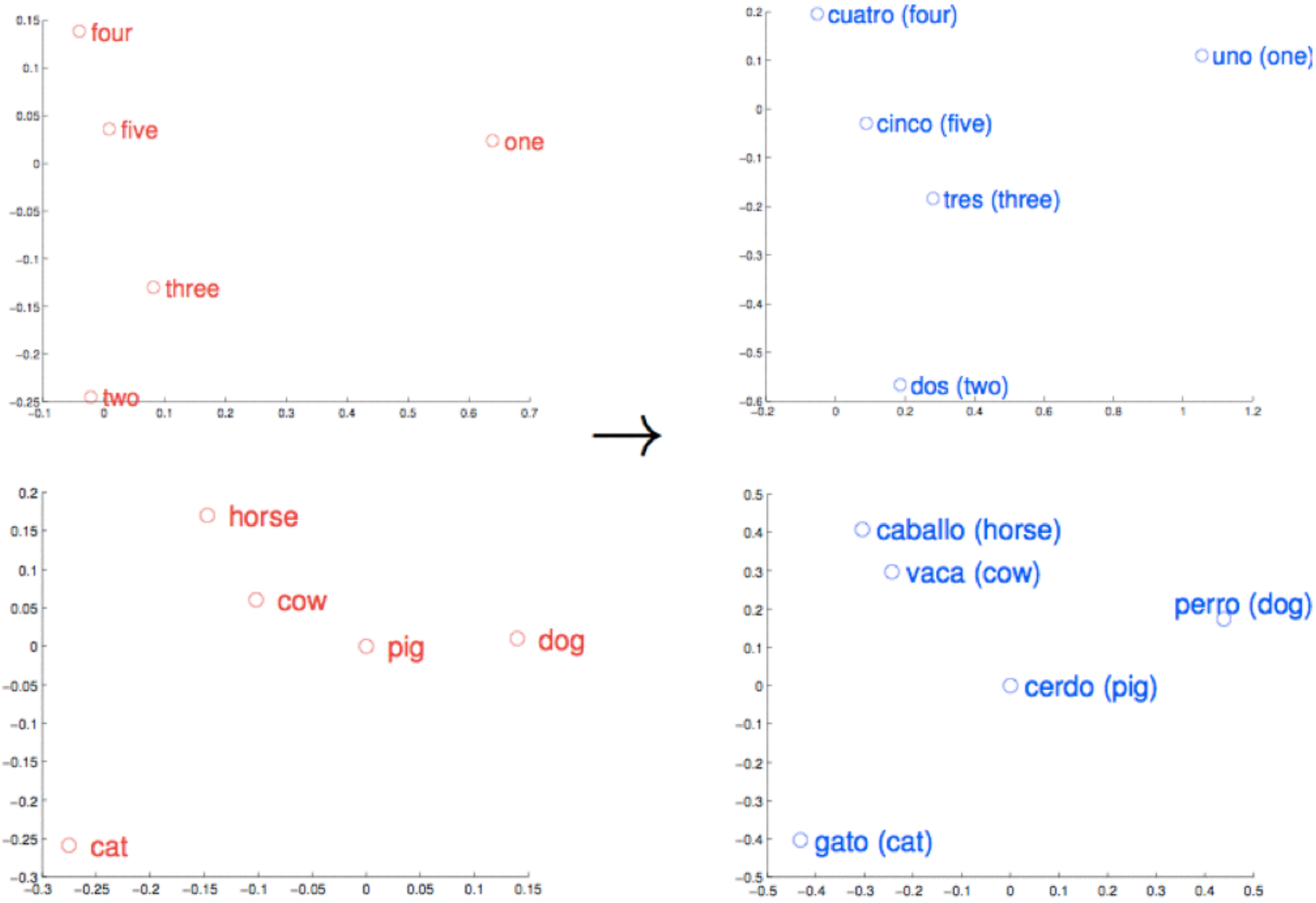
Word2vec –Vector Representation of Words (Mikolov et al. 2013)

- Similarity between two word embeddings represents conceptual similarity of words

- Example results: words associated with *Sweden*, in order of proximity:

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

credit: Mikolov, Le & Sutskever 2013 (arXiv 1309.4168), Figure 1



source: Exploiting Similarities among Languages for MT

- Word Embedding Learn to embed each word into a vector of real values
  - Semantically similar words have closer embedding representations
  - Can uncover semantic /syntactic word relationships
    - [king] - [male] + [female] ~= [queen]
    - [Berlin] - [Germany] + [France] ~= [Paris]
    - [eating] - [eat] + [fly] ~= [flying]

figure via Dr. Li Deng's talk

Analogies: Mikolov, Yih & Zweig, NAACL 2013

figure via Dr. Jeff Dean's talk

Word Embeddings can be used for

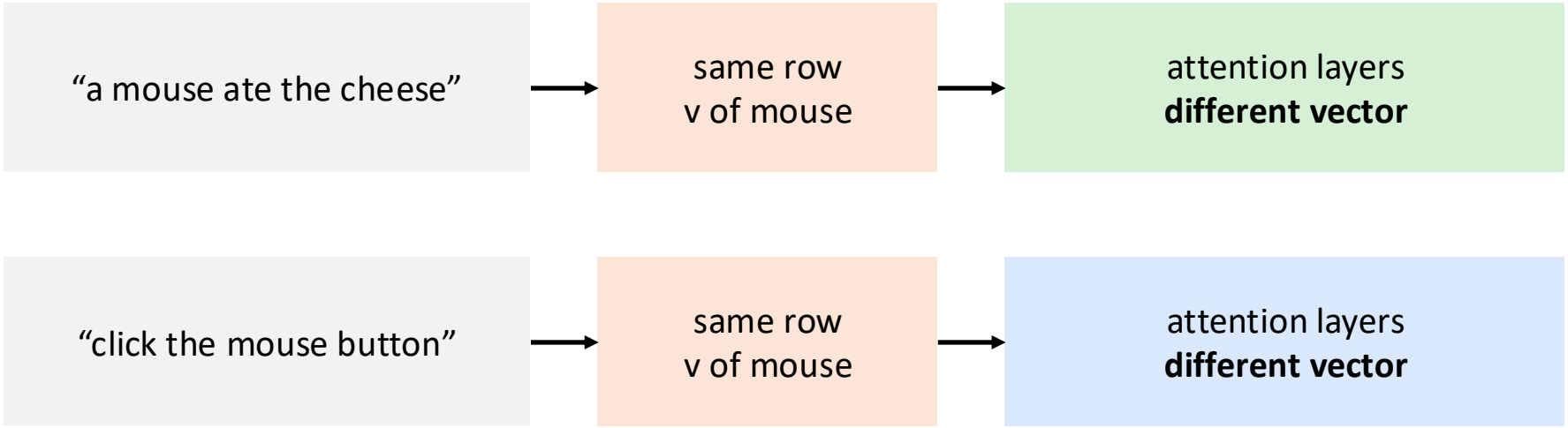
- Relationships between different words
- Useful for language modeling, sentiment analysis, and more

However: word embeddings

- ~~**True**~~ **/ False**: Word2Vec embeddings crash if given an unknown word.
- ~~**True**~~ **/ False**: Word2Vec embeddings lemmatize (sing, sang are the same vector)
- ~~**True**~~ **/ False**: Word2Vec embeddings can intelligently handle homonyms (mouse the animal and mouse the cursor device).
- ~~**True**~~ **/ False**: Word2Vec embeddings can handle context-dependent meaning.

Credit: Kording et al., UPenn CIS 700, 2019

One row per word is not enough



- Word2vec: the vector is the final answer
- Transformers: E is only the first layer; attention then mixes in the context
- E is still V × D, and is now trained with the whole model

Embeddings in 2026

Model	One vector per	Depends on context?	D
word2vec	word	no	300
GloVe (Wikipedia + Gigaword)	word	no	100
GPT-2 token embedding	token	after attention	768
all-MiniLM-L6-v2	sentence	yes	384
BGE-M3	text passage	yes	1024
text-embedding-3-small / large	text passage	yes	1536 / 3072

- Word vectors still teach the idea, and are cheap
- **Today's default:** one vector for a whole text, from a pretrained Transformer
- Usually a pooled summary of the T × D outputs

credit: gensim-data; Hugging Face model cards; OpenAI embeddings guide; as of Oct 2026

From word vectors to retrieval

	Bag of words / TF-IDF	Dense embedding
Length of one text vector	V, mostly zeros	D, all non-zero
Matches	the same words	similar meaning
Needs	just counting	a pretrained model

$$\text{scores} = Mq \qquad (N \times D)\,(D \times 1) = (N \times 1)$$

- Embed N text chunks once: the matrix M is N × D
- Embed the question: a vector q of length D
- With unit-length vectors, each score is a cosine similarity
- 10,000 chunks × 384 = 3.84 million numbers, about 15 MB

Where this leads: the retrieval step of RAG

credit: numbers computed (32-bit floats)

Pretrain, load, then fine-tune or not

1. Pretrain
someone trains on huge text data

→

2. Publish
weights + config on a hub (Hugging Face)

→

3. Load
from_pretrained(name)

4. Then decide:

Use as is
no training at all

Train a new head
freeze the pretrained weights

Fine-tune all
update every weight

"Fine-tuning is identical to pretraining except you don't start with random weights."

credit: Hugging Face Transformers docs, Fine-tuning (v5)

Load: the same step at every scale

```
# word vectors: load E, frozen or trainable
wv = gensim.downloader.load("glove-wiki-gigaword-100")    # 400K words, D = 100
E = nn.Embedding.from_pretrained(torch.tensor(wv.vectors), freeze=True)
```

```
# sentence embeddings: one vector per chunk
m = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")
M = m.encode(chunks)                                          # N × 384
```

```
# a whole LLM: use as is
pipe = pipeline(task="text-generation", model="openai-community/gpt2")
pipe("the secret to baking a good cake is ")
```

freeze=True keeps the weights fixed; freeze=False lets training update them

credit: gensim-data; PyTorch docs; sentence-transformers model card; Transformers docs, Pipeline

Fine-tune or not?

Your situation	Try
General text, no labels, retrieval or search	Use as is
A few labeled examples, new labels	Freeze, train a new head
Plenty of labels, your text looks different	Fine-tune all, small learning rate
Out of data or GPU	Stay with as-is or a head

```
model = AutoModelForCausalLM.from_pretrained(model_name, dtype="auto")
# head only: for p in model.base_model.parameters(): p.requires_grad = False
args = TrainingArguments(output_dir="finetuned", learning_rate=2e-5, num_train_epochs=3)
trainer = Trainer(model=model, args=args, train_dataset=train, eval_dataset=dev,
                  processing_class=tokenizer, data_collator=collator)
trainer.train()
trainer.push_to_hub()
```

Start with as-is and measure. Fine-tune only if the result is not good enough

credit: Hugging Face Transformers docs, Fine-tuning (v5); PyTorch docs; rules of thumb

Quiz Check: which path?

- You want to search 5,000 course notes by meaning. Use as is, a new head, or fine-tune all?
- With 384-dimensional vectors, what is the shape of M? Of the score vector?
- You have 200 labeled tweets for 5 new categories. Which path, and why?
- You call from_pretrained(weights, freeze=True) and train. What happens to E?
- A brand-new word appears in your text. What does a word2vec table return?

UVA CS 4774: Machine Learning

S3: Lecture 16: Deep Neural Networks for Natural Language Processing

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Module IV

Roadmap : f() on natural language

- Before Deep NLP (Pre 2012)
  - (BOW / LSI / Topic LDA)
- Word2Vec (2013-2016)
  - (GloVe/ FastText)
- Recurrent NN (2014-2016)
  - LSTM
  - Seq2Seq
- Attention / Self-Attention (2016 – now)
  - Attention
  - Transformer (self-attention, attention only)
  - BERT / XLNet/ GPT-2 / T5 …

Variable Length Nature of Natural Language

X

This Food is not good.

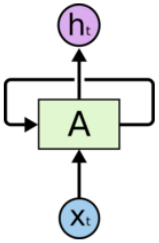


This wonderful book is a pleasure to read.

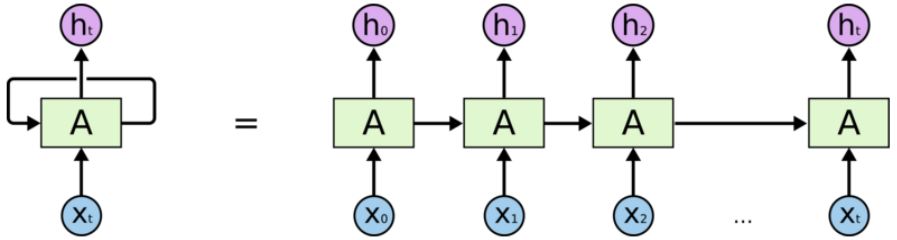


Recurrent Neural Networks

- Allow us to operate over sequences of vectors (with variable length)
- Allow Sequences in the input, as the output, or in the most general case both



Recurrent Neural Networks have loops.



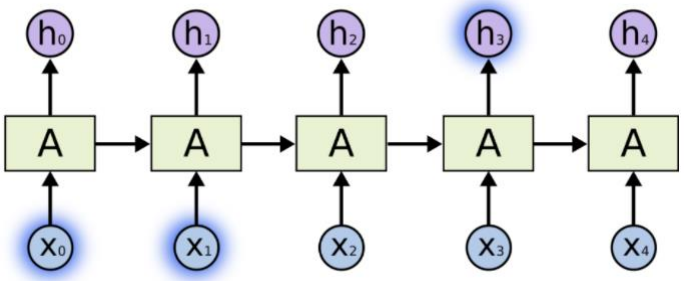
An unrolled recurrent neural network.

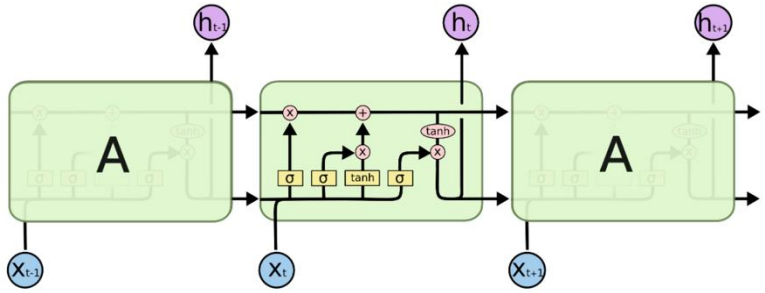
Recurrent Neural Networks are networks with loops in them, allowing information to persist.

Image Credits from Christopher Olah

Deep RNN in the 90's

- Hochreiter & Schmidhuber (1997) invented the LSTM: "Long short-term memory" – Recurrent NN (LSTM-RNN)





The repeating module in an LSTM contains four interacting layers.

Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780.

Image Credits from Christopher Olah

Recurrent Neural Networks Got Popular (2015-20)

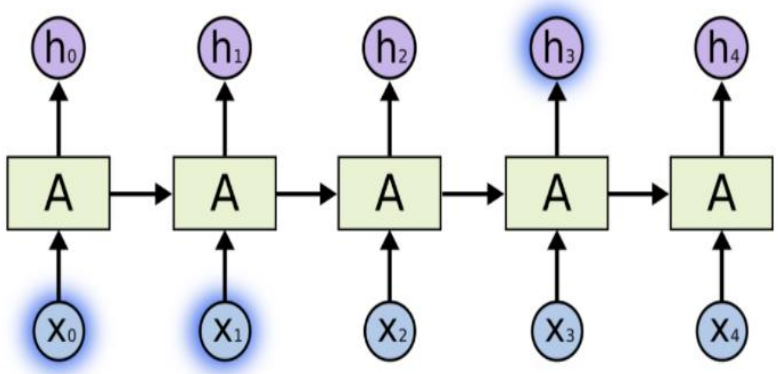
- Incredible success applying RNNs to language modeling and sequence learning problems

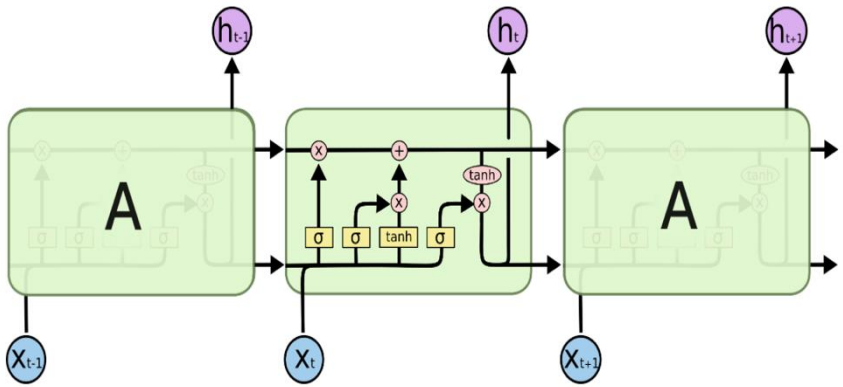
Task	**Input Sequence**	**Output Sequence**
Machine translation (Sutskever et al. 2014)	English	French
Question answering (Bordes et al. 2014)	Question	Answer
Speech recognition (Graves et al. 2013)	Voice	Text
Handwriting prediction (Graves 2013)	Handwriting	Text
Opinion mining (Irsoy et al. 2014)	Text	Opinion expression

LSTM

- "Long short-term memory" – Recurrent NN (LSTM-RNN)

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) = \overrightarrow{LSTM}(\mathbf{x}_t)$$





The repeating module in an LSTM contains four interacting layers.

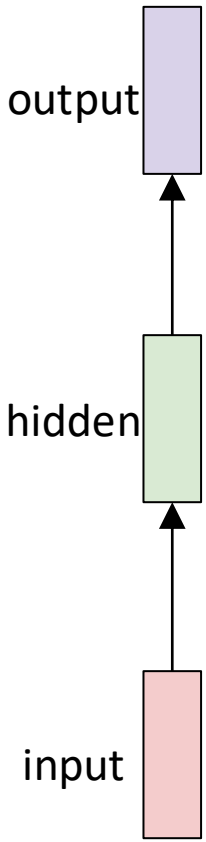
Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780.

Image Credits from Christoph

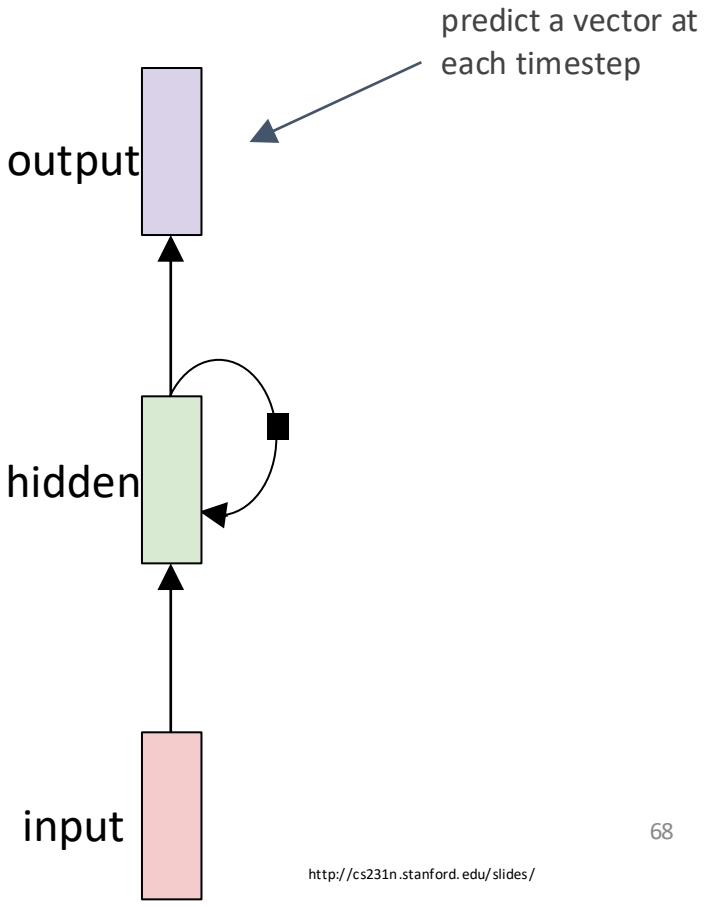
RNN models dynamic temporal dependency

- Make fully-connected layer model each unit recurrently
- Units form a directed chain graph along a sequence
- Each unit uses recent history and current input in modeling

Traditional "Feed Forward" Neural Network



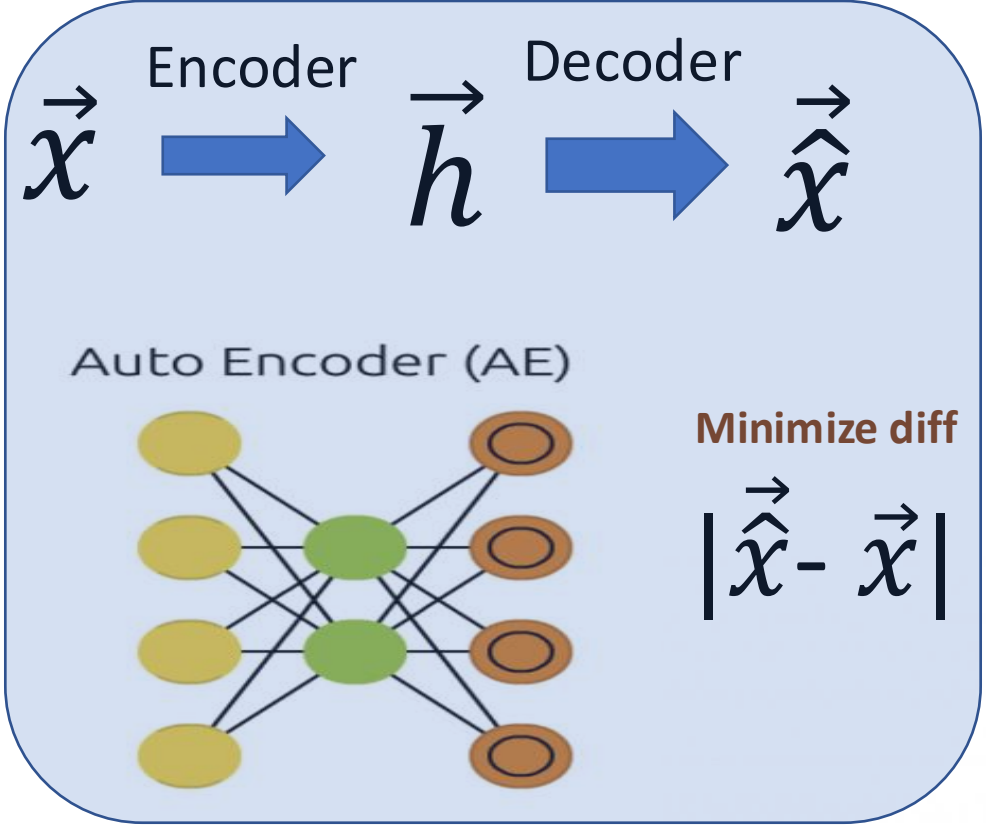
Recurrent Neural Network



RNN can model variable-length input / output

Anything requiring long-range patterns	Anything generative
• Question detection • Natural language context understanding • Entity disambiguation • Sentence embedding	• Machine translation • Natural language generation • Question answering • Skip-thoughts

http://cs231n.stanford.edu/slides/



RNN can models variable-length input / output

Seq2Seq

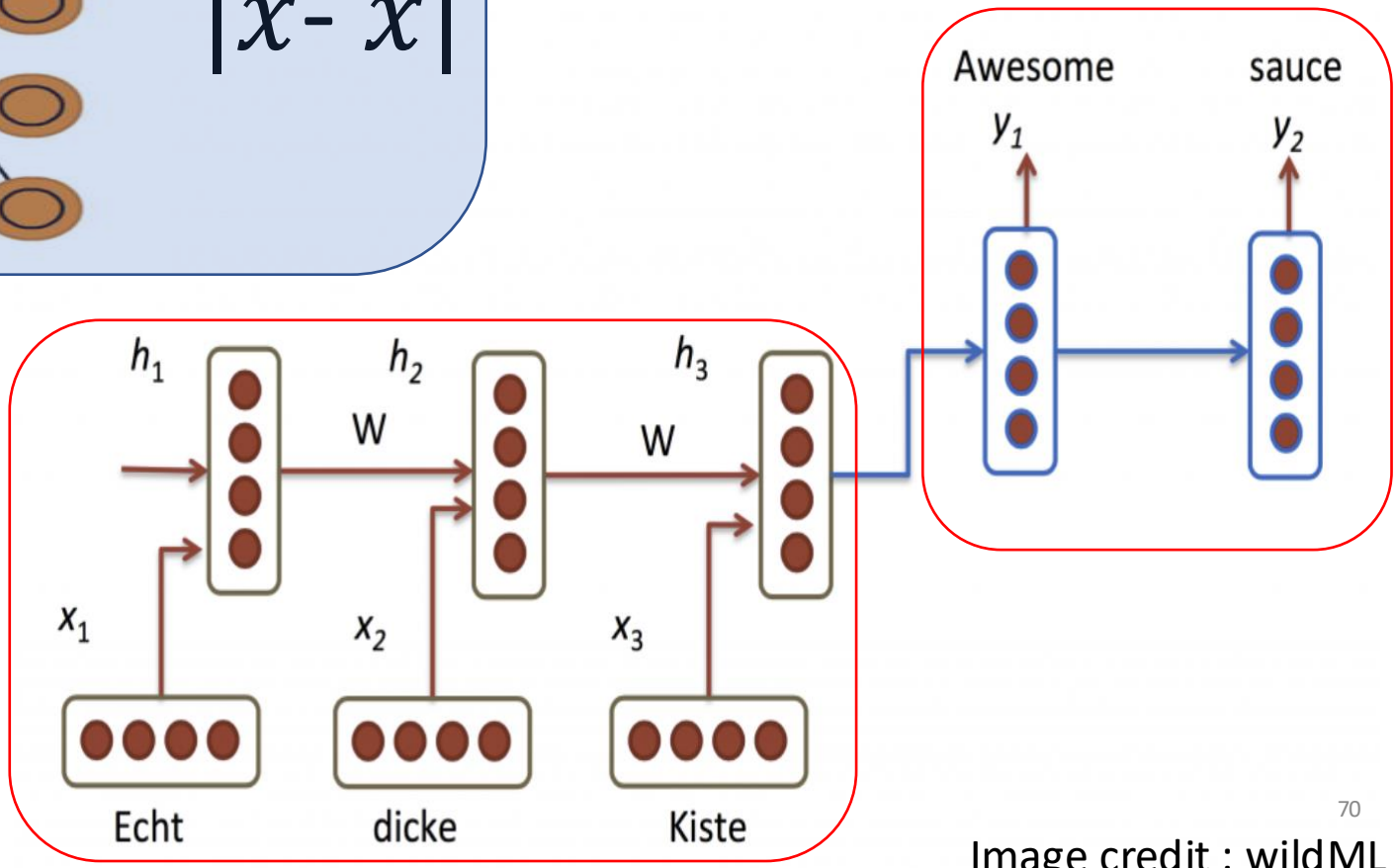


Image credit : wildML

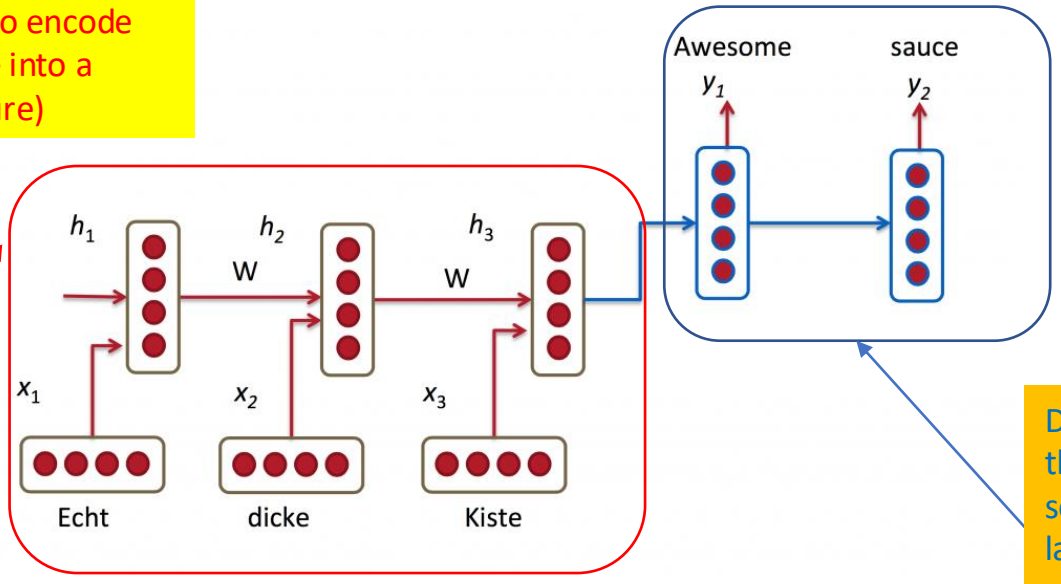
Seq2Seq for Machine Translation

In machine translation, the **input** is a sequence of words in source language, and the **output** is a sequence of words in target language.

Two LSTMs for Machine Translation (German to English)

- Encoder LSTM (on Germany)
- Decoder LSTM (on English)

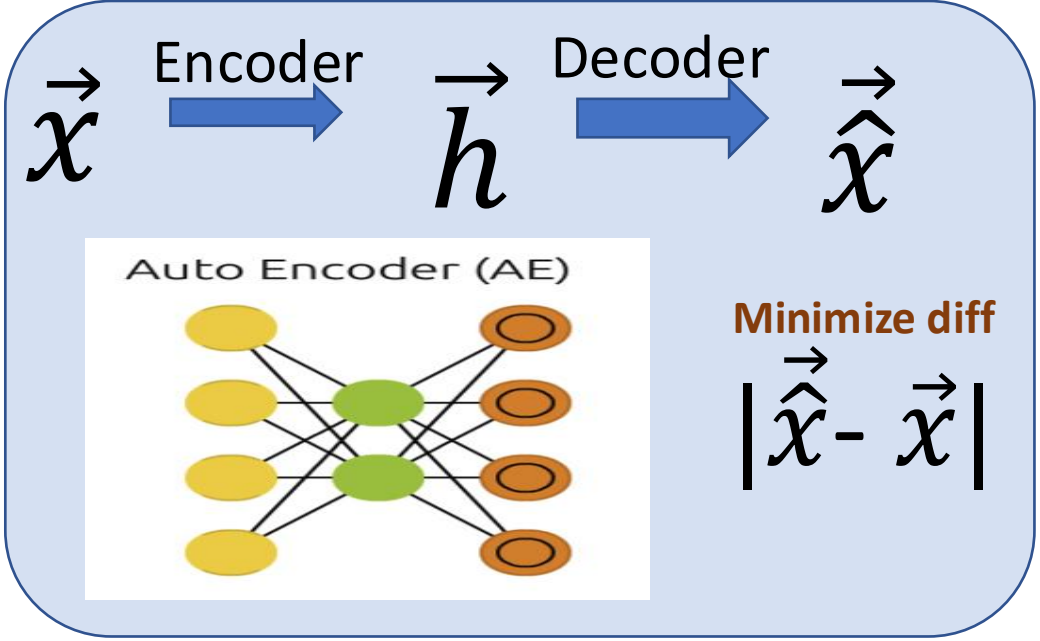
Encoder: An RNN to encode the input sentence into a hidden state (feature)



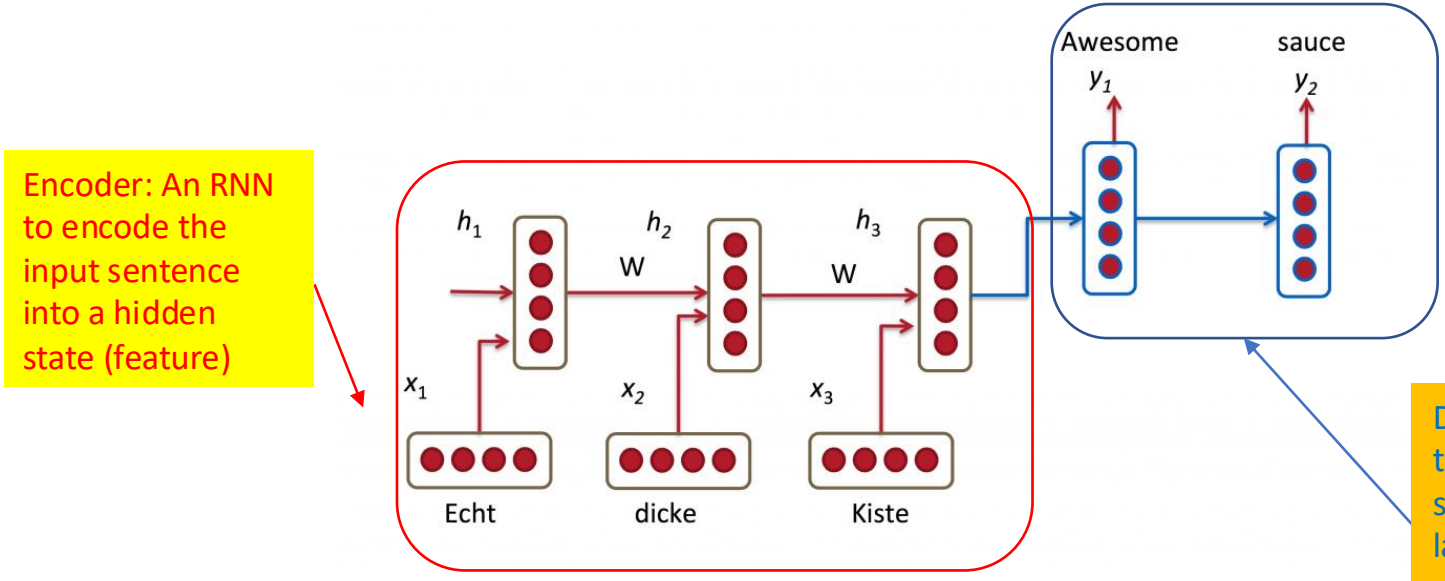
Decoder: An RNN with the hidden state of the sentence in source language as the input and output the translated sentence

Encoder-decoder architecture for machine translation

Adapt from Professor Qiang Yang of HK UST



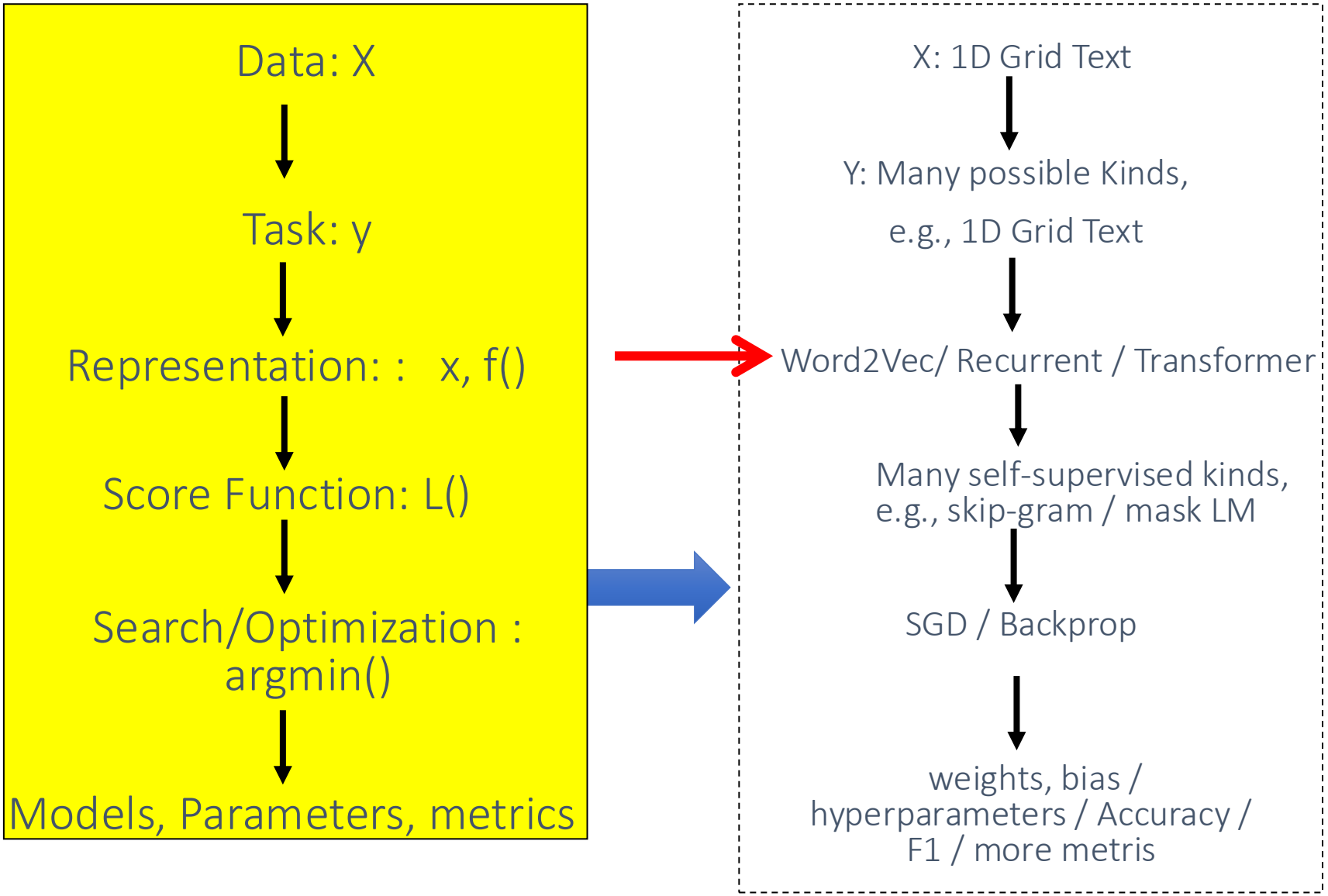
Encoder-decoder architecture for machine translation



Encoder: An RNN to encode the input sentence into a hidden state (feature)

Decoder: An RNN with the hidden state of the sentence in source language as the input and output the translated sentence

Today Recap: : Neural Network Models on 1D Grid / Language Data



References

- Dr. Yann Lecun's deep learning tutorials
- Dr. Li Deng's ICML 2014 Deep Learning Tutorial
- Dr. Kai Yu's deep learning tutorial
- Dr. Rob Fergus' deep learning tutorial
- Prof. Nando de Freitas' slides
- Olivier Grisel's talk at Paris Data Geeks / Open World Forum
- Hastie, Trevor, et al. *The elements of statistical learning*. Vol. 2. No. 1. New York: Springer, 2009.
- Dr. Hung-yi Lee's CNN slides
- NIPS 2017 DL Trend Tutorial

References: embeddings and workflow

- Mikolov et al., Efficient Estimation of Word Representations in Vector Space, 2013
- Mikolov et al., Distributed Representations of Words and Phrases, NeurIPS 2013
- Jurafsky & Martin, Speech and Language Processing, 3rd ed. draft, ch. 5
- PyTorch docs: torch.nn.Embedding
- Hugging Face Transformers docs: Fine-tuning, Pipeline
- Model cards: all-MiniLM-L6-v2, BGE-M3; OpenAI embeddings guide; gensim-data