Word Representation II: Statistical Methods

Lecture 5

Announcements

- Late submission by a few minutes:
    - No problem. Just don't do it for HW2.
- HW2 will be posted soon (today or tomorrow).

- One more lecture this Thursday, two PSs next week.
    - Today: Language Model before Transformer.
    - Thursday: Transformers.
    - Next 4 PSs: Hovhannes walks through the entire Transformer architecture.

Outline

- Global Word Embeddings
  - Word2vec
  - GloVe
- Contextual Word Embeddings
  - Language Modelling
  - RNN

Softmax

- Say now we have a lot of logits: $\mathbf{x} = [x_1, \dots x_n], x_i \in (-\infty, +\infty)$
- We want a probability distribution that
  - $\mathbf{p} = f(\mathbf{x}) = [p_1, \dots p_n] = f([x_1, \dots, x_n]) = [f(x_1), \dots, f(x_n)]$
  - Probability distribution -> everything sums to 1. $\sum \mathbf{p} = 1$
  - Differentiable everywhere.
- Solution:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Representing Data

- Earlier success in computer vision.
  - Navlab 5 (Jochem et al., 1995)

- Much more intuitive to convert images into vector representations.

Representing Data

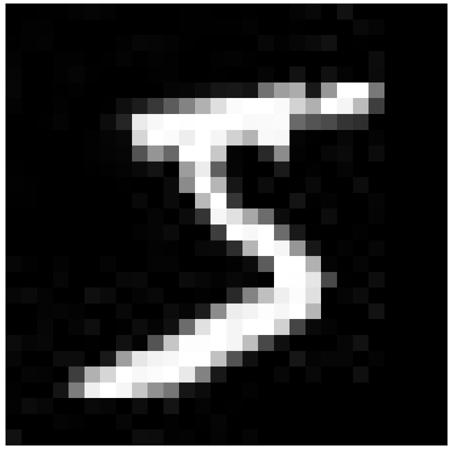
- Numeric Data:
  - E.g. credit score:

Monthly Income	Number Of Open Credit Lines And Loans	Number Of Times 90 Days Late	Number Real Estate Loans Or Lines	Number Of Time 60-89 Days Past Due Not Worse	Number Of Dependents
9120	13	0	6	0	2
2600	4	0	0	0	1
3042	2	1	0	0	0
3300	5	0	0	0	0
63588	7	0	1	0	0
3500	3	0	1	0	1
NA	8	0	3	0	0
3500	8	0	0	0	0

Give Me Some Credit (gmsc): https://www.kaggle.com/c/GiveMeSomeCredit

Representing Data

- Numeric Data:
  - E.g. credit score:
- Images:
  - Gray scale or RGB



MNIST dataset
Handwritten numbers

- Videos:
  - Images on a timeline

Representing `Textual` Data

- The vast majority of rule--based and statistical NLP work regarded words as atomic symbols.
  - Recall Lecture 3:

```
Det → the | a | an
Adj → old | red | happy | …
N   → dog | park | ice-cream | contumely | run | …
V   → saw | ate | run | disdained | …
P   → in | to | on | under | with | … }
```

- Vector space: this is a vector with one 1 and a lot of zeroes:

```
[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
```

  - The “one-hot” representation
  - i-th word in the dictionary:

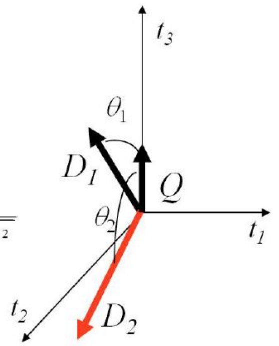
$$v_i = 1, \forall j \neq i, v_j = 0$$

Representing `Textual` Data: Problems

- There are a lot of words!
    - Oxford English Dictionary: 500,000+ entries
    - Longman Dictionary of Contemporary English: 230,000 words
    - Brysbaert et al. (2016): 42,000 lemmas
  - As a result, a lot of BIG vectors!
  - For reference, (L)LM dimensions:
    - BERT, GPT-2: 768
    - Llama-3-8B: 4096
    - Llama-3.1-405B: 16384
- No useful similarity information:
    - Motel: `[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]`
    - Hotel: `[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]`
    - Linguist: `[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]`

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\text{CosSim}(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



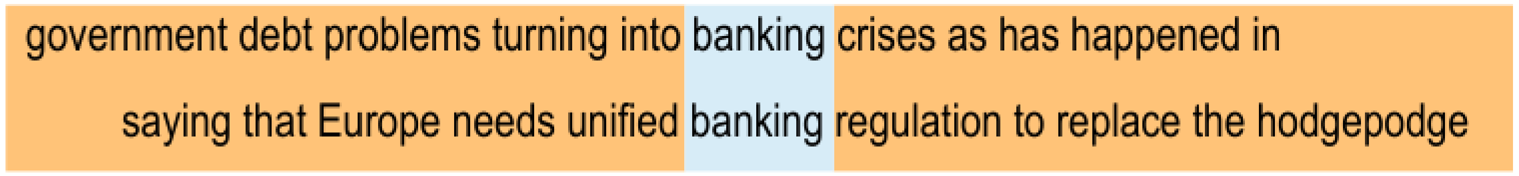
$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim(D_1 , Q) = 10 / $\sqrt{(4+9+25)(0+0+4)}$ = 0.81
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim(D_2 , Q) = 2 / $\sqrt{(9+49+1)(0+0+4)}$ = 0.13
$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

- cos_sim(motel, hotel)=0, cos_sim(motel, linguist)=0, cos_sim(hotel, linguist)=0

Distributional similarity based representations

- You can get a lot of value by representing a word by means of its neighbors:
- "Noscitur a sociis"
  - The meaning of an unclear or ambiguous word should be determined by considering the words with which it is associated in the context.
  - 19th-century rule of interpretation in English civil courts.
- One of the most successful ideas of modern NLP



government debt problems turning into banking crises as has happened in

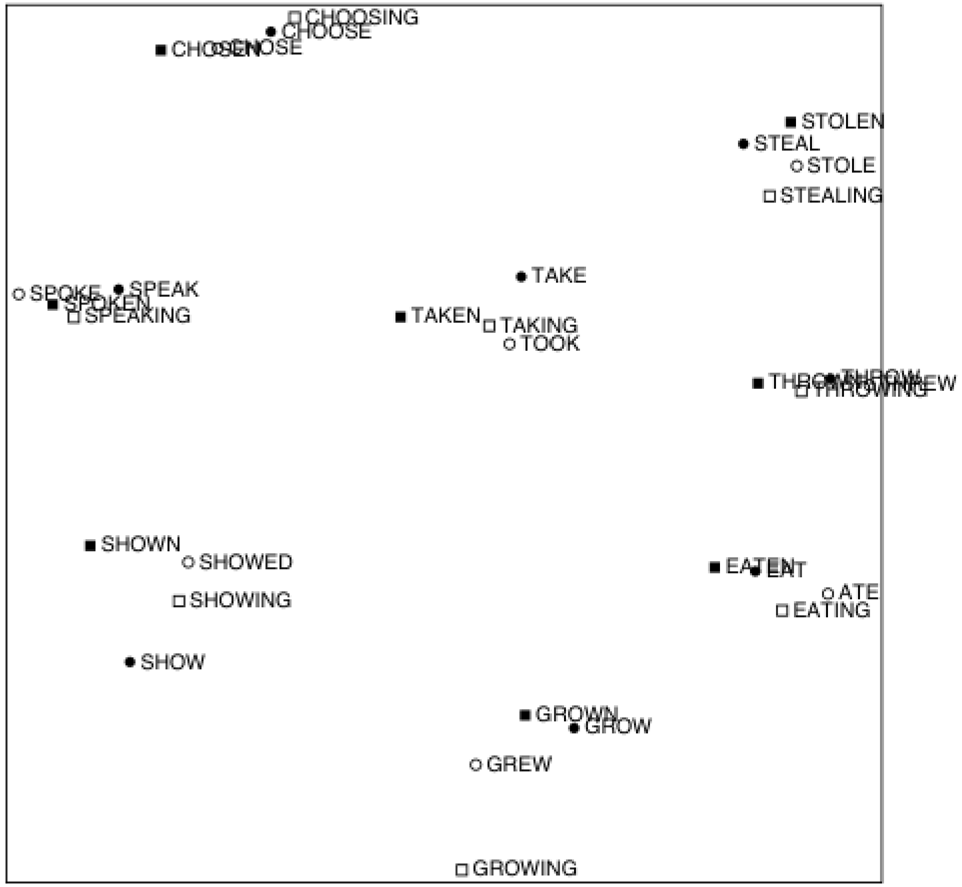
saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

With distributed, distributional representations, syntactic and semantic information can be captured

$$\textit{shown} = \begin{matrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{matrix}$$

Synonymy? Hyponymy?
Morphology?



[Rohde et al. 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence]

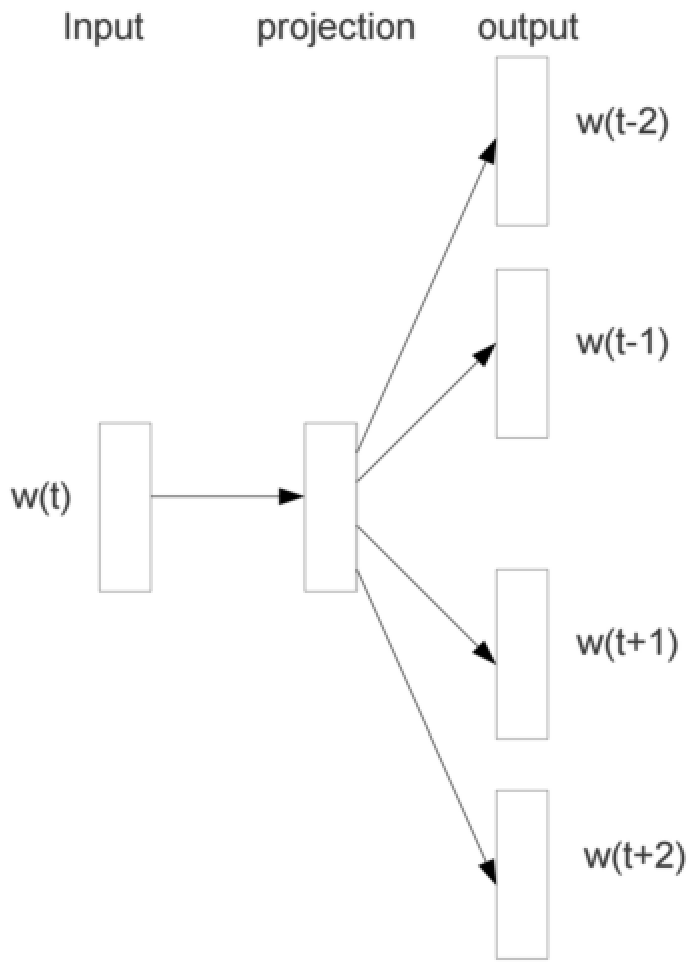
Two Kinds of Vectors

- Count:
  - tf-idf, PMI, LSA
  - **Sparse!**
  - Information Retrieval workhorse!
  - Words are represented by (a simple function of) the counts of nearby words
- Predict:
  - word2vec, GloVe, BERT, GPT-2, GPT-3, GPT-4…
  - **Dense!**
  - Representation is created by training a classifier to predict whether a word is likely to appear nearby
  - *Contextual embeddings*.

Word2vec

Word2vec CBOW/SkipGram: Predict!

- Train word vectors to try to either
  - Predict a word given its bag-of-words context (CBOW); or
  - Predict a context word (position-independent) from the center word
- Update word vectors until they can do this prediction well



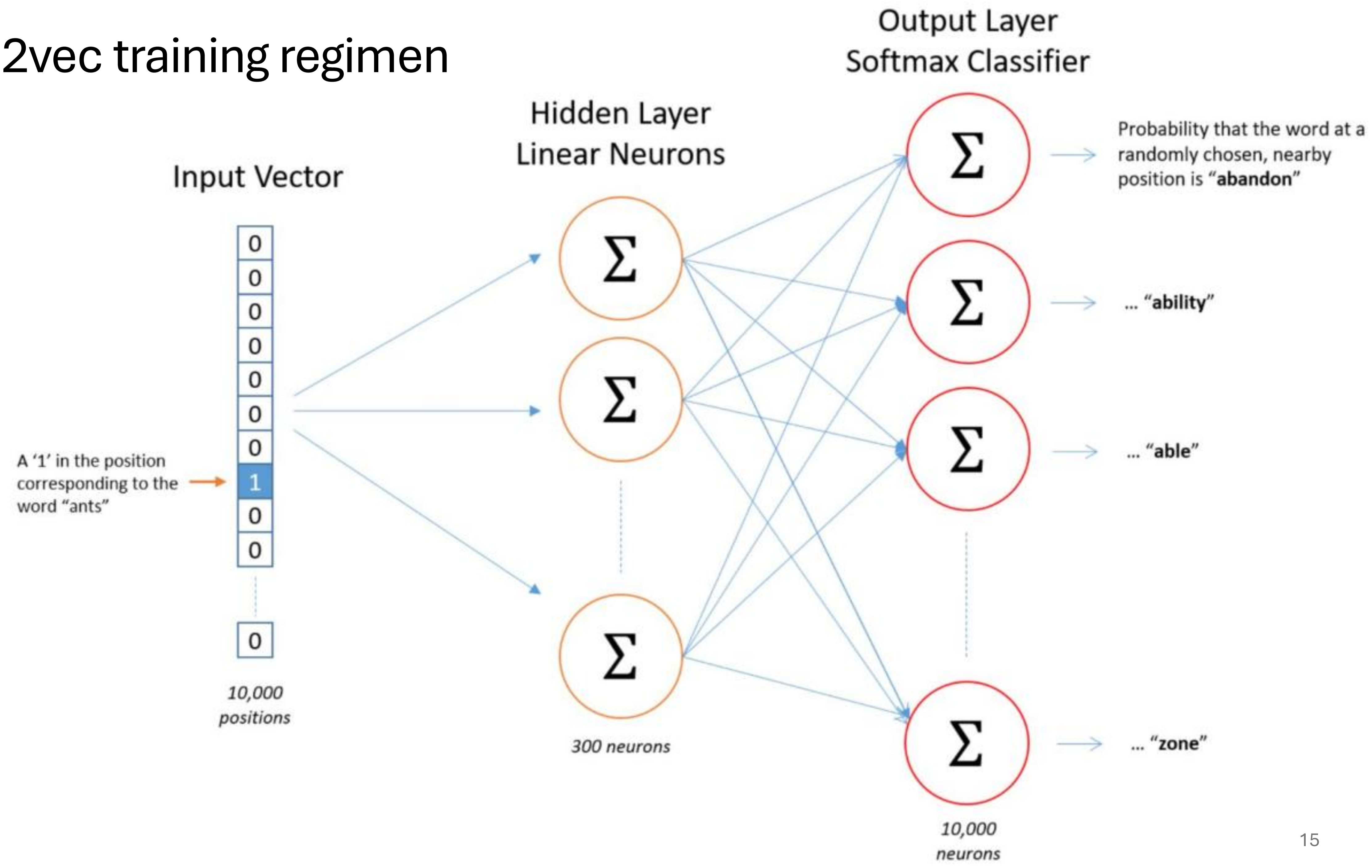
Skip-Gram Training Data

- Assume a +/- 2 word window, given training sentence:

```
…lemon, a [tablespoon of apricot jam, a] pinch
              c1       c2 [target] c3  c4
```

- Goal: train a classifier that is given a candidate (word, context) pair
- And assigns each pair a probability:
  - P(+|w, c)
  - P(−|w, c) = 1 − P(+|w, c)

Word2vec training regimen



```python
class Word2Vec(nn.Module):

    def __init__(self, vocab_size, embedding_size):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, embedding_size)
        self.expand = nn.Linear(embedding_size, vocab_size, bias=False)

    def forward(self, input):
        # Encode input to lower-dimensional representation
        hidden = self.embed(input)
        # Expand hidden layer to predictions
        logits = self.expand(hidden)
        return logits
```

Approach: predict if candidate word *c* is a "neighbor"

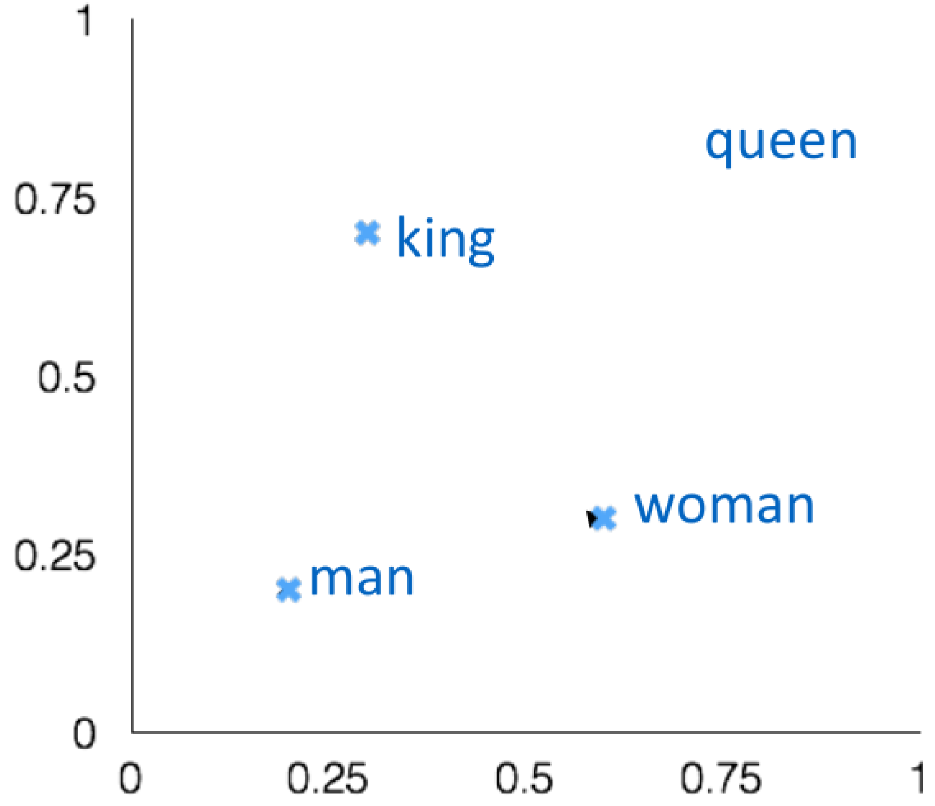
1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples.
3. Use logistic regression to train a classifier to distinguish those two cases.
4. Use the learned model activations as the embeddings.

Word Analogies: word2vec captures dimensions of similarity as linear relations

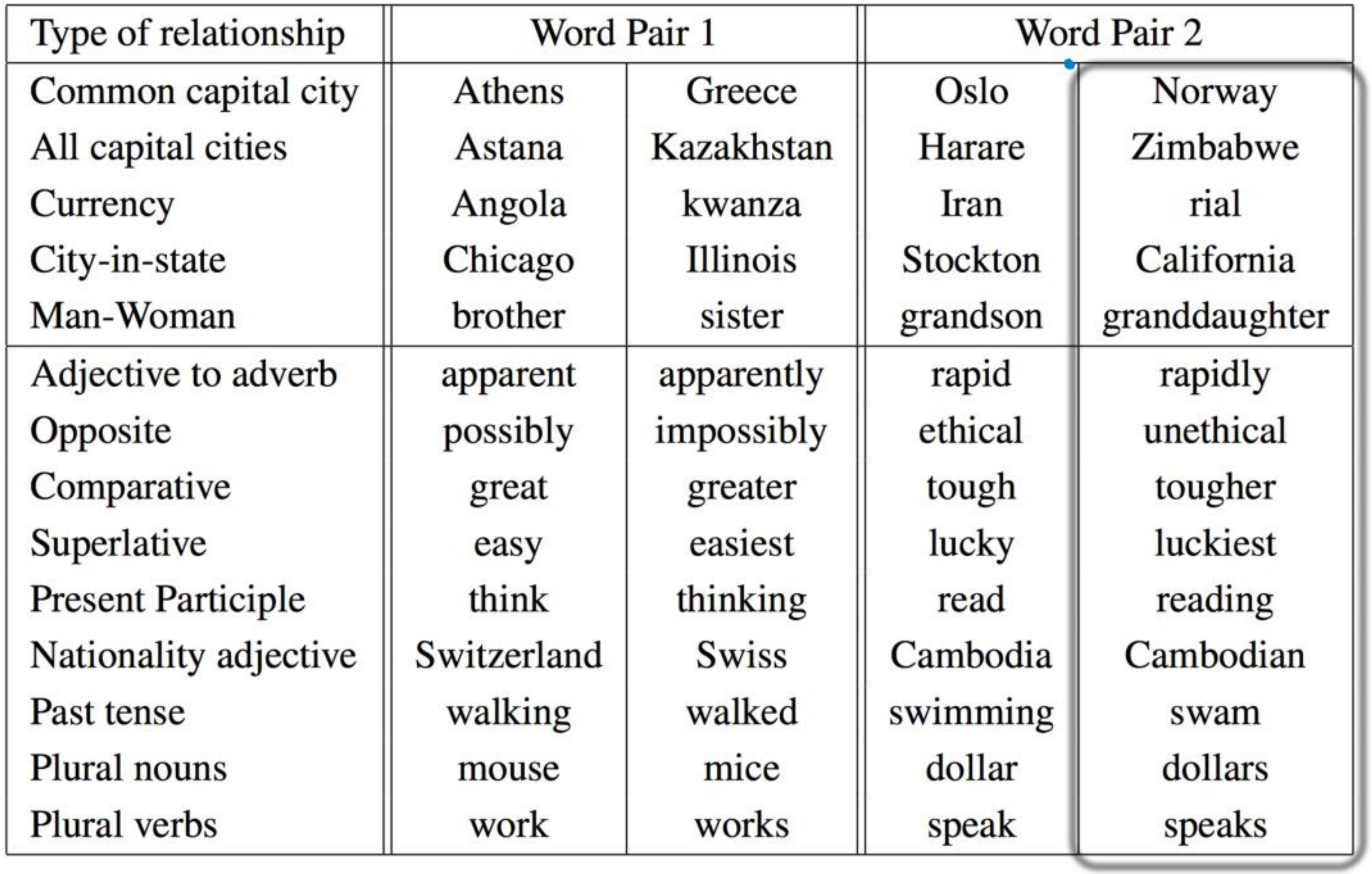
Test for linear relationships, examined by Mikolov et al. (2013)

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
[Mikolov et al., 2012, 2013]



Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Count-based vs. Direct Prediction

Count-based

- Fast training
- Efficient usage of statistics

- Long & Sparse!
  - Length = |V|
  - most elements are zero
- Primarily used to capture word similarity
- Disproportionate importance given to small counts

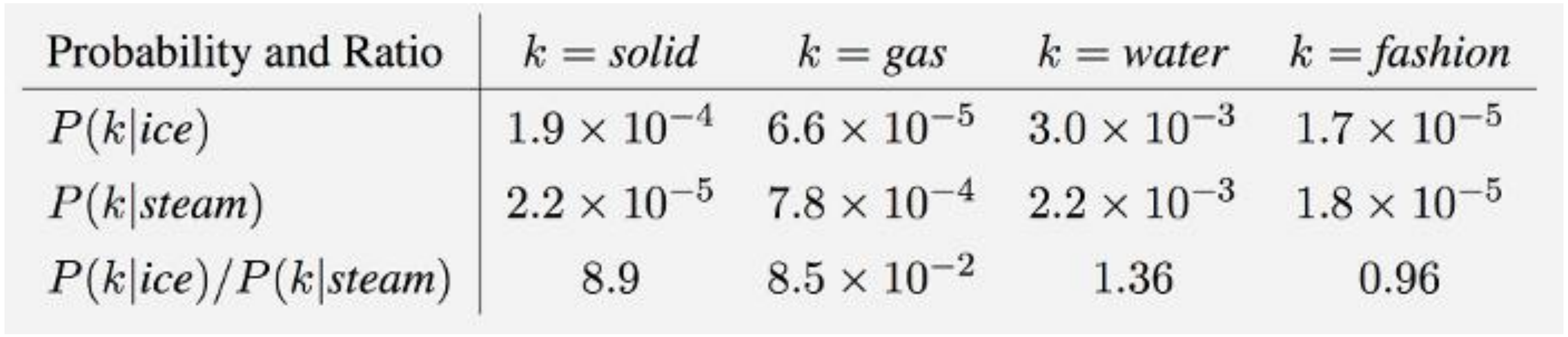
Direct Prediction

- Scales with corpus size
- Inefficient usage of statistics

- Short and Dense
  - Length = any hidden size (50-10000)
  - Nearly nothing is zero
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Encoding meaning in vector differences

- Key idea:
  - Ratios of co-occurrence probabilities can encode meaning components



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Pennington et al. (2014)

Encoding meaning in vector differences

- How can we capture ratios of co-occurrence probabilities as meaning components in a word vector space?
- Solution:
  - Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$
  - with vector differences: $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

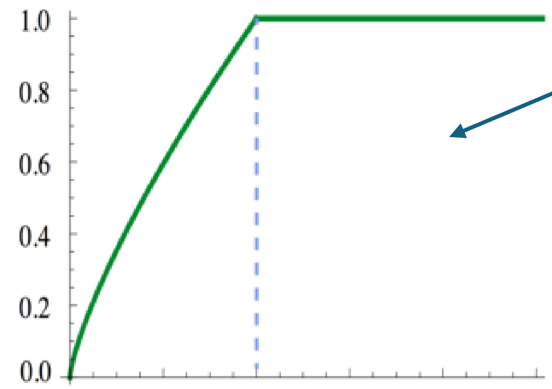
Pennington et al. (2014)

GloVe: A new model for learning word representations

$$w_i \cdot w_j = \log P(i|j)$$

$$w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

$f \sim$



weighting function

biases

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

dot product (similarity)

Co-ocurrence count

```
# https://github.com/noaRricky/pytorch-glove
class GloVeModel(nn.Module):

    def __init__(self, embedding_size, context_size, vocab_siz)
        self.focal_embeddings = nn.Embedding(
            vocab_size, embedding_size)
        self.context_embeddings = nn.Embedding(
            vocab_size, embedding_size)

        self.focal_biases = nn.Embedding(vocab_size, 1)
        self.context_biases = nn.Embedding(vocab_size, 1)

    def loss(self, focal_input, context_input, coocurrence_count):

        focal_embed = self.focal_embeddings(focal_input)
        context_embed = self.context_embeddings(context_input)
        focal_bias = self.focal_biases(focal_input)
        context_bias = self.context_biases(context_input)

        # count weight factor
        weight_factor = torch.pow(coocurrence_count / x_max, alpha)
        weight_factor[weight_factor > 1] = 1

        embedding_products = torch.sum(focal_embed * context_embed, dim=1)
        log_cooccurrences = torch.log(coocurrence_count)

        distance_expr = (embedding_products + focal_bias +
                         context_bias + log_cooccurrences) ** 2

        single_losses = weight_factor * distance_expr
        mean_loss = torch.mean(single_losses)
        return mean_loss
```

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Word Similarities

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

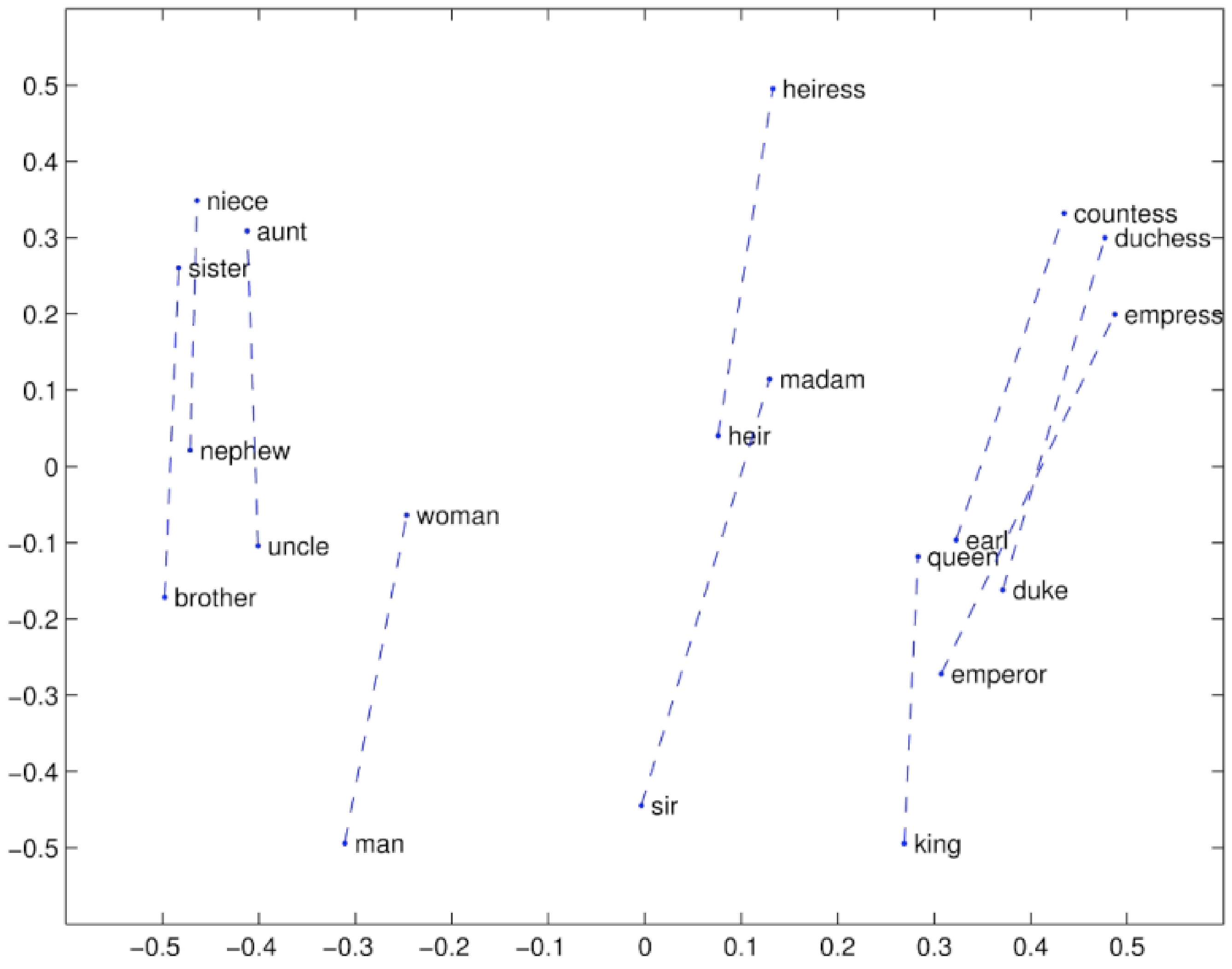
3. litoria

4. leptodactylidae

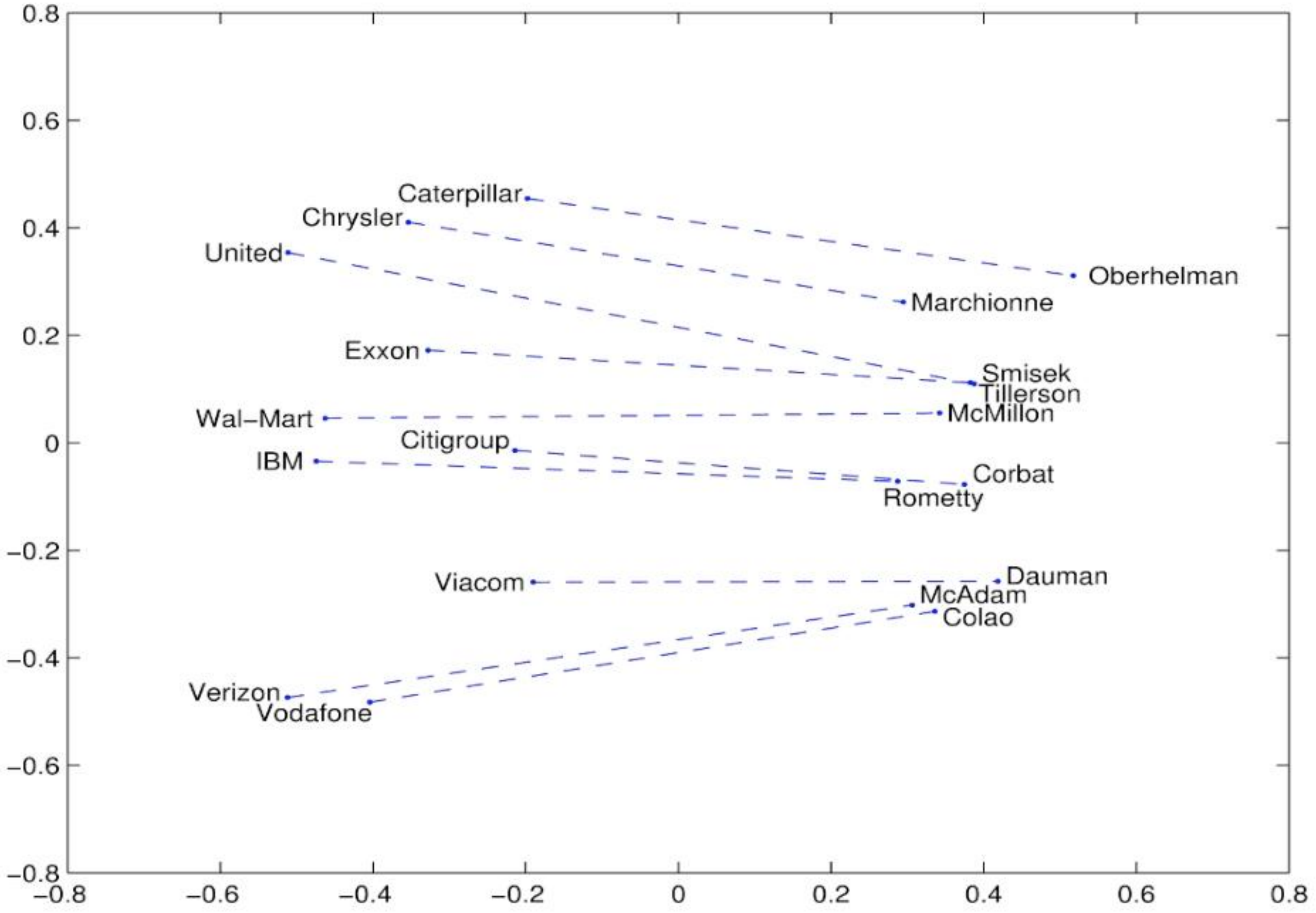
5. rana

7. eleutherodactylus

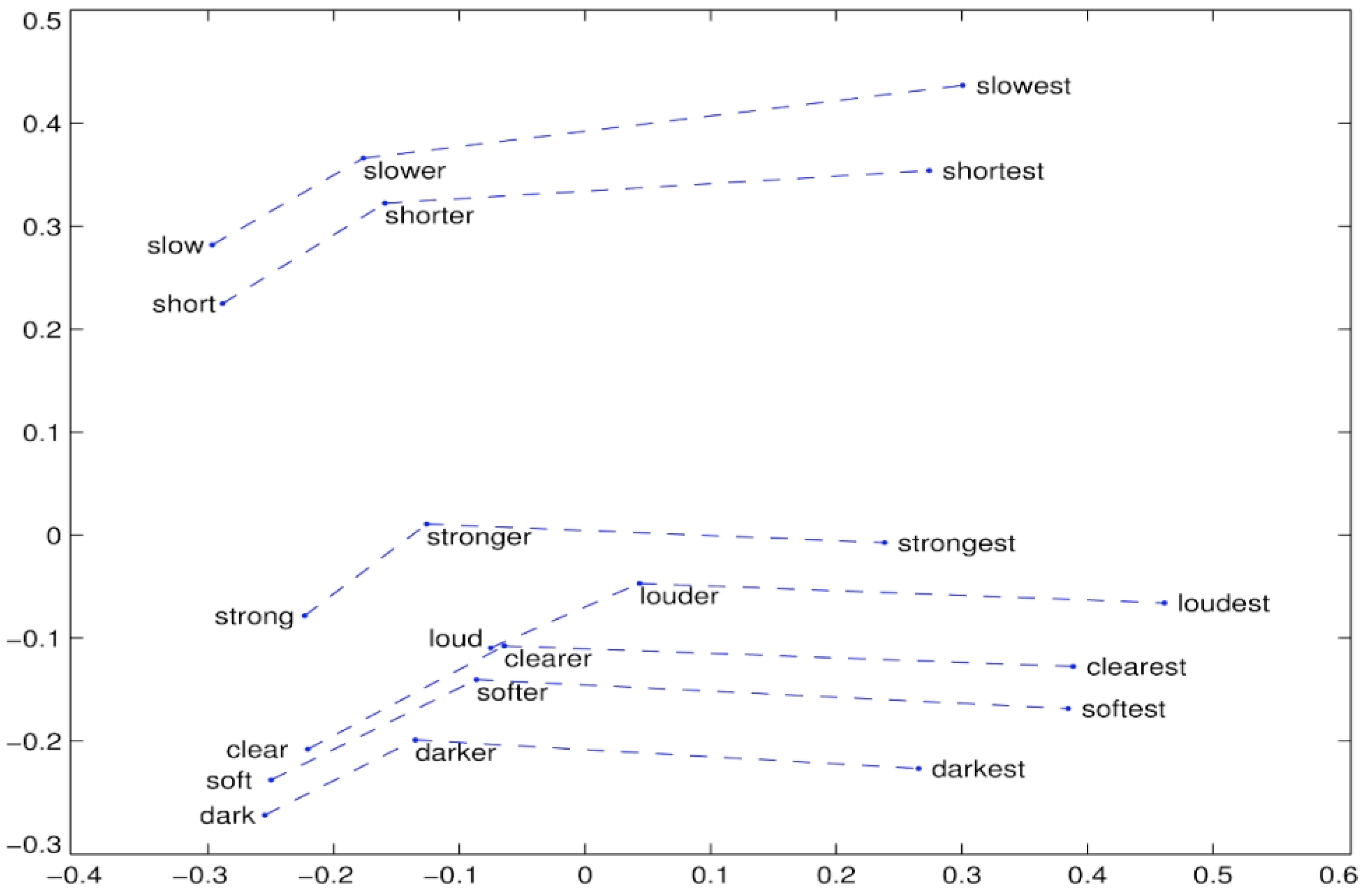
Linear Structures: Visualizations



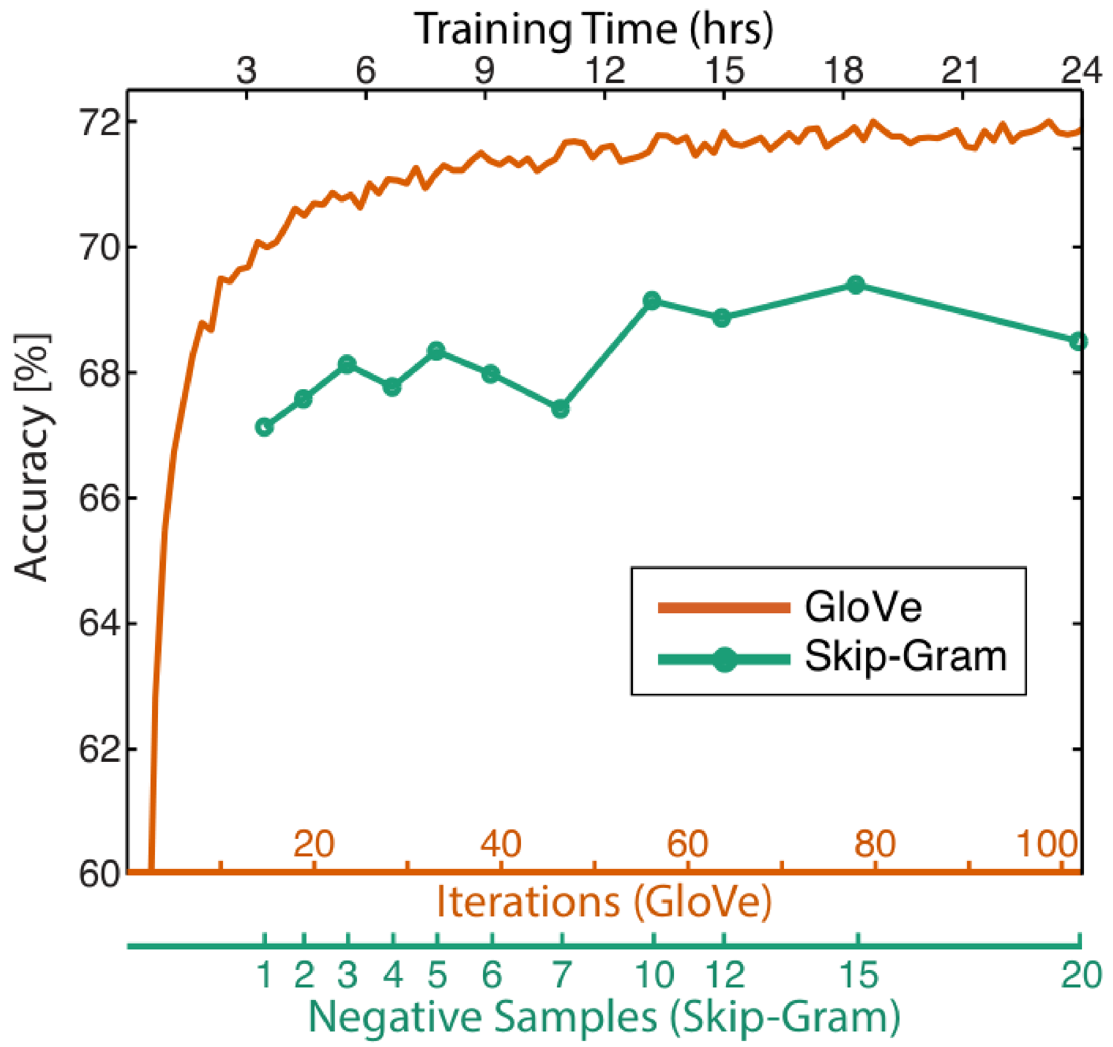
Linear Structures: Visualizations



Linear Structures: Visualizations



Analogy evaluation and hyperparameters

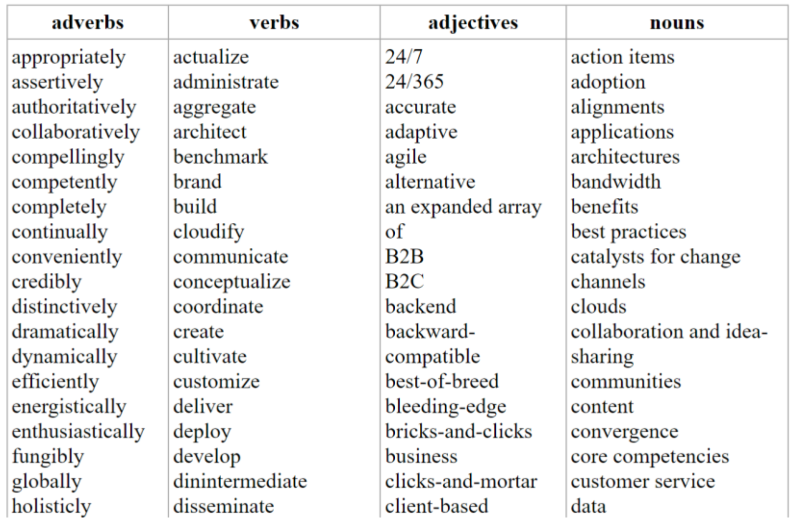


Word Embedding Conclusion

- Developed a model that can translate meaningful relationships between word-word co-occurrence probabilities into linear relations in the word vector space.
- GloVe shows the connection between Count! work and Predict! work – appropriate scaling of counts gives the properties and performance of Predict! models

Terminology Hell

- “Embedding”
  - Embedding layer: `torch.nn.Embedding`
    - Linear layer: one hot index -> vectorized representation
    - Basically, a big look up table
  - Vector(ized) Representation
    - Using an n-dim vector to represent a word. The vector.
  - Hidden Representation; Hidden State
    - The intermediate output of a neural network
    - Neural LM: use this as the vectorized representation
  - Word Embedding:
    - The model/system/algorithm that generate a vectorized representation given a word.
  - Word Embedding:
    - The generated vectorized representation.



adverbs	verbs	adjectives	nouns
appropriately	actualize	24/7	action items
assertively	administrate	24/365	adoption
authoritatively	aggregate	accurate	alignments
collaboratively	architect	adaptive	applications
compellingly	benchmark	agile	architectures
competently	brand	alternative	bandwidth
completely	build	an expanded array	benefits
continually	cloudify	of	best practices
conveniently	communicate	B2B	catalysts for change
credibly	conceptualize	B2C	channels
distinctively	coordinate	backend	clouds
dramatically	create	backward-	collaboration and idea-
dynamically	cultivate	compatible	sharing
efficiently	customize	best-of-breed	communities
energistically	deliver	bleeding-edge	content
enthusiastically	deploy	bricks-and-clicks	convergence
fungibly	develop	business	core competencies
globally	dinintermediate	clicks-and-mortar	customer service
holisticly	disseminate	client-based	data

The Corporate B.S. Generator

WSD!

Contextual vs. Global Word Embedding

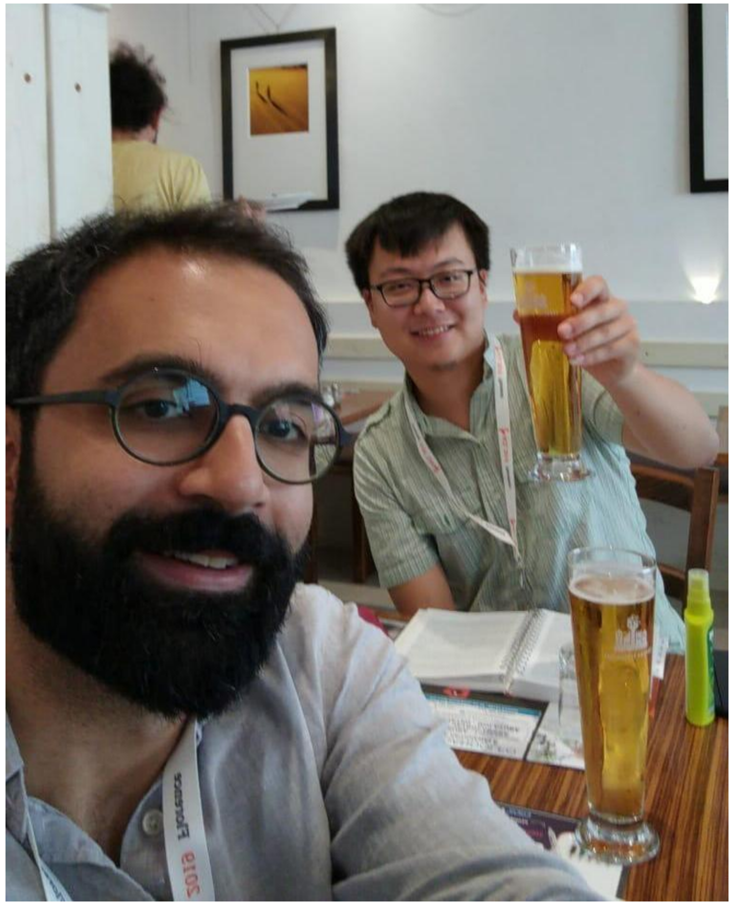
- Global Word Embedding
  - One vector representation word-type
  - word2vec, GloVe
- Contextual Word Embedding
  - One vector representation word-token
  - RNN, LSTM, BERT, GPT…

Recall: Primitives: lexical categories or parts of speech.

- Each word-type is a member of one or more.
- Each word-token is an instance of exactly one.

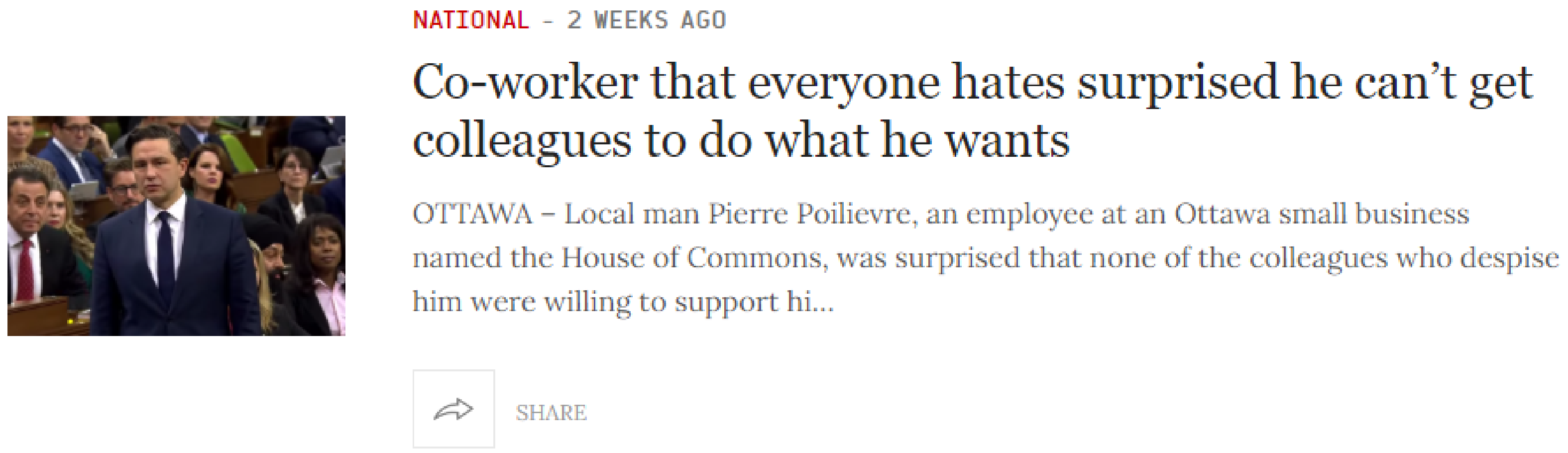
Context is important

- WSD:
  - The lawyer approached the **bar**.
  - Frank approached the **bar**.
- Anything beyond lexical level:
  - Anaphora:
    - Mary is a doctor. **She** works at the ____
    - Anne is a farmer. **She** works at the ____
  - Long distance agreement:
    - <u>The books</u> that every student have read **are** ...
- Tasks:
  - What is the capital of Germany. It is ___


Bro conferencing at ACL 2019.

Also, we need more! What of larger semantic units?

- How can we know when larger units are similar in meaning?
  - *CTV News*: Poilievre-led attempt to bring down Trudeau minority over carbon tax fails.
  - *CBC News*: Liberals survive non-confidence vote on carbon tax with Bloc, NDP backing.
  - The Beaverton: Co-worker that everyone hates surprised he can’t get colleagues to do what he wants.


NATIONAL - 2 WEEKS AGO

Co-worker that everyone hates surprised he can’t get colleagues to do what he wants

OTTAWA – Local man Pierre Poilievre, an employee at an Ottawa small business named the House of Commons, was surprised that none of the colleagues who despise him were willing to support hi…

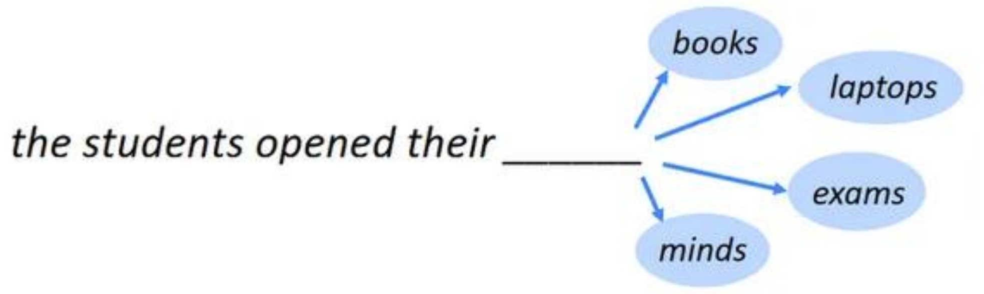
SHARE

Language Modelling Task

- Final goal: predict/estimate the probability of a sequence

Probability(Some sentence over here.)

- Language Models:
  - Estimate the likelihood of sequence of texts.
  - Model that assigns probabilities to sequences of words.
  - (All good definitions)
- Actual task:
  - Predict the next word
  - MLM



Likelihood, Probability of What?

- A really vague, abstract notion of how likely this sentence can appear in real-world, natural language discourse.
- Ultimate Goal: create a statistical model to describe some large quantity of corpora data, that ultimately capture the statistical structure of "Language."
- Again, the practical definition of Language Model is:
  - Estimate the likelihood of sequence of texts.
  - Model that assigns probabilities to sequences of words.

(both are fine)

- Likelihood is not: (related terms)
  - Grammaticality, acceptability, syntactic well-formedness, "make sense," well written …
  - Colorless green ideas sleep furiously. -> Chomsky was terribly wrong.

The Language Modelling Pipeline

- Collect large quantity of unstructured data
  - Wikipedia articles, social media post, news articles…
  - Famous open-source: WikiText-2/103 (100M Tokens), Dolma (3T tokens)
- Tokenization
  - `The Technische Universität Darmstadt, commonly known as TU Darmstadt, is a public research university in the city of Darmstadt, Germany.`
  - `['The', 'ĠTechn', 'ische', 'ĠUnivers', 'it', 'Ã¤', 't', 'ĠD', 'arm', 'stadt', ',', 'Ġcommonly', 'Ġknown', 'Ġas', 'ĠT', 'U', 'ĠD', 'arm', 'stadt', ',', 'Ġis', 'Ġa', 'Ġpublic', 'Ġresearch', 'Ġuniversity', 'Ġin', 'Ġthe', 'Ġcity', 'Ġof', 'ĠD', 'arm', 'stadt', ',', 'ĠGermany', '.']`
- Train the actual model by performing the language modelling task:
  - Next Word Prediction, Masked Language Modelling, …

Language Modelling is **NOT Unsupervised**!

- Supervised vs. unsupervised learning.
    - Supervised learning: labelled data.
    - Unsupervised learning: unlabelled data, find structure in data itself.
        - E.g., Clustering (e.g., K-Means) and Dimensionality Reduction (e.g., PCA, t-SNE).
- Language modelling tasks require direct supervision.
    - Label: next token.
- Why many people call Language Modelling *unsupervised*?
    - Data collection: unprocessed, unstructured raw text from the internet.
    - Basically free!
    - Indeed, we may not need to hire people to do any labelling or annotation.

Language Modelling is **NOT Unsupervised**!

- Why many people call Language Modelling *unsupervised*?
  - ~~Data collection: unprocessed, unstructured raw text from the internet.~~
  - ~~Basically free!~~
  - ~~Indeed, we may not need to hire people to do any labelling or annotation.~~
- No longer the case now (2023, 2024ish)!
  - The major LLM research labs (Qwen, Llama, OpenAI…) have used nearly all the data human ever created.
  - Most commercial models need large amount of curated data.

What's the best way to follow how the news is changing?

Our daily email, with all the freshest future-of-journalism news.

your email address he

Subscribe

THE LATEST FROM NIEMAN LAB

BBC resignations are a symbol of mounting pressure on public broadcasters in the Trump era

DENIS MULLER

Meet the journalists training AI models for Meta and OpenAI

The gig work platform Outlier is one of several companies courting journalists to train large language models (LLMs).

https://www.niemanlab.org/2025/02/meet-the-journalists-training-ai-models-for-meta-and-openai/

Tokenization and Tokenizers

- Character-level language modeling:
  - Classifying Names with a Character-Level RNN
  - Good with Chinese
  - Other languages: inefficient use of data
- Tokenization: breaks down text into smaller units, often called tokens.
  - `text.split()`
- The only difficulty: unknown token.
  - Special <unk> token

#longexposurephotography	Rechtsschutzversicherungsgesellschaft
Long exposure photography	Rechts Schutz Versicherung s Gesellschaft
	legal protection insurance company

Agglutinative language

Çekoslovakya	Czechia(in ussr)
-(l)ı	from
-(l)aştır	to (let it) become
-(a)ma	not
-dı	past
-k	1st pronoun(us)
-lar	plural
-(ı)mız	(from) us
-dan	out of /from
-mış	past (heard)
-sı(n)ız	you

Eiskaffee

Çekoslovakyalılaştıramadıklarımızdanmışsınız

"You are reportedly one of those that we could not make Czechoslovakian."

Tokenization and Tokenizers

- Solution: **Break a word down into subwords, or word pieces!**

- Slightly different encoding styles

  colorless green ideas sleep furiously

  - BERT: `'color', '##less', 'green', 'ideas', 'sleep', 'furiously’`
  - GPT/LLaMA: `'color', 'less', 'Ġgreen', 'Ġideas', 'Ġsleep', 'Ġfuriously’`
  - XLM: `'color', 'less</w>', 'green</w>', 'ideas</w>', 'sleep</w>', 'furiously</w>’`

Byte Pair Encoding (BPE)

- Subword tokenization technique.
- Used for data compression and dealing with unknown words.

- Initialization:
- Vocabulary = set of all individual characters.
- V = {A, B, C, … a, b, c, … 1, 2, 3, … !, $, %, …}

- Repeat:
  - Choose two symbols that appear as a pair most frequently (say “a” and “t”).
  - Add new merged symbol (“at”).
  - Replace each occurrence with the new symbol (“t”,”h”,”a”,”t” -> “t”,”h”,”at”).

- Until *k* merges have been done. Usually, we select a pre-defined vocabulary size (*k*).

Byte Pair Encoding (BPE)

```
Segments:                               Vocabulary:
5       l o w </w>                      </w>, d, e, I, l, n, o, r, s, t, w
2       l o w e s t </w>
6       n e w e r </w>
3       w i d e r </w>
2       n e w </w>

Most frequent symbol pair: er (9 times)

Segments:                               Vocabulary:
5       l o w </w>                      </w>, d, e, I, l, n, o, r, s, t, w, er
2       l o w e s t </w>
6       n e w er </w>
3       w i d er </w>
2       n e w </w>
```

Byte Pair Encoding (BPE)

```
Segments:                                   Vocabulary:
5       l o w </w>                          </w>, d, e, I, l, n, o, r, s, t, w, er
2       l o w e s t </w>
6       n e w er </w>
3       w i d er </w>
2       n e w </w>

Most frequent symbol pair: er</w> (9 times)

Segments:                                   Vocabulary:
5       l o w </w>                          </w>, d, e, I, l, n, o, r, s, t, w, er,
2       l o w e s t </w>                    er</w>
6       n e w er</w>
3       w i d er</w>
2       n e w </w>
```

Byte Pair Encoding (BPE)

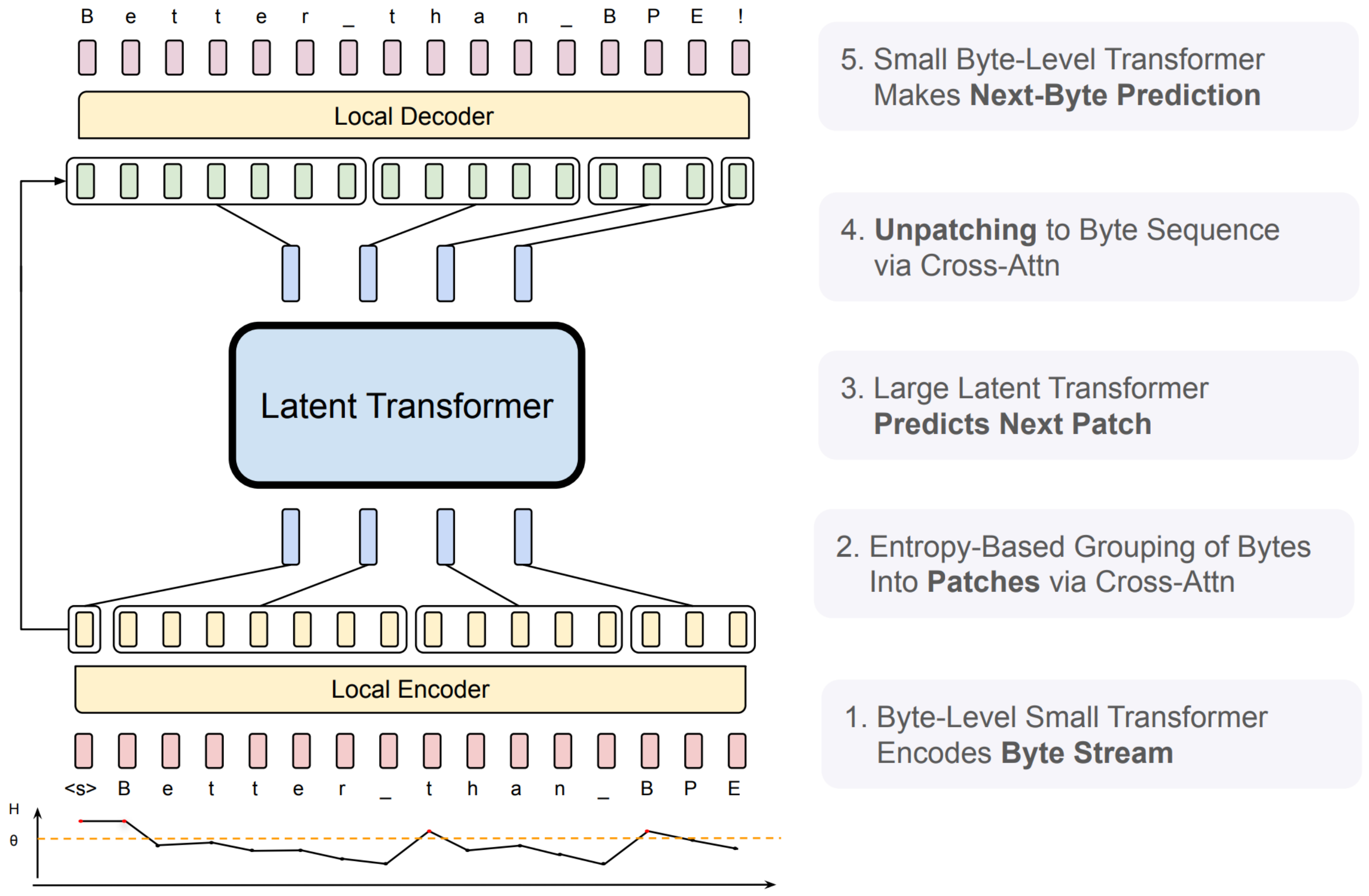
```
Segments:                                   Vocabulary:
5       l o w </w>                          </w>, d, e, I, l, n, o, r, s, t, w, er,
2       l o w e s t </w>                    er</w>
6       n e w er</w>
3       w i d er</w>
2       n e w </w>

Most frequent symbol pair: ne (8 times)

Segments:                                   Vocabulary:
5       l o w </w>                          </w>, d, e, I, l, n, o, r, s, t, w, er,
2       l o w e s t </w>                    er</w>, ne
6       ne w er</w>
3       w i d er</w>
2       ne w </w>
```

Limitations

- Doesn’t work well for other writing systems:
    - Especially Arabic.
- Typos:
    - participating -> `'Ġparticipating'`
    - partcipating -> `'Ġpart', 'c', 'ip', 'ating’`
- “How many r’s does the word strawberry has?”

Possible Future Directions

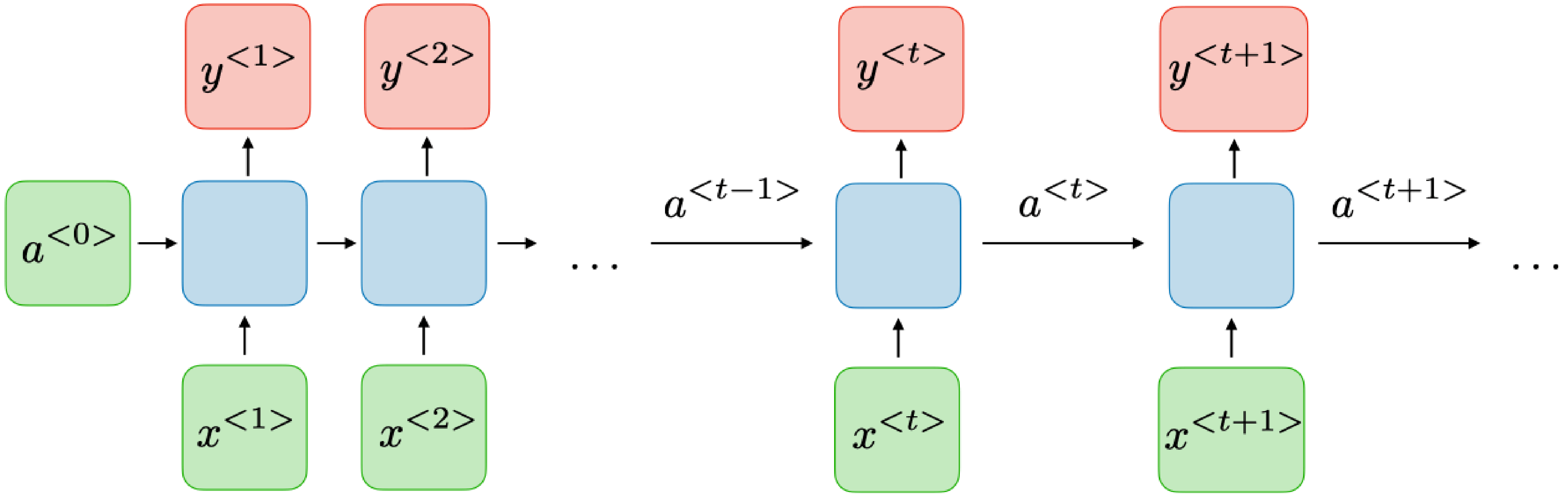
- Byte-based tokenization
  - Byte Latent Transformer
  - Pagnoni et al. (2025).
- Language (writing system) specific tokenization.
  - E.g., Arabic: morphological methods.
- Modularity & model merging: combine models together. (More on this later)



B e t t e r _ t h a n _ B P E !
Local Decoder
Latent Transformer
Local Encoder
<s> B e t t e r _ t h a n _ B P E
H
θ
5. Small Byte-Level Transformer Makes **Next-Byte Prediction**
4. **Unpatching** to Byte Sequence via Cross-Attn
3. Large Latent Transformer **Predicts Next Patch**
2. Entropy-Based Grouping of Bytes Into **Patches** via Cross-Attn
1. Byte-Level Small Transformer Encodes **Byte Stream**

Training a Language Model

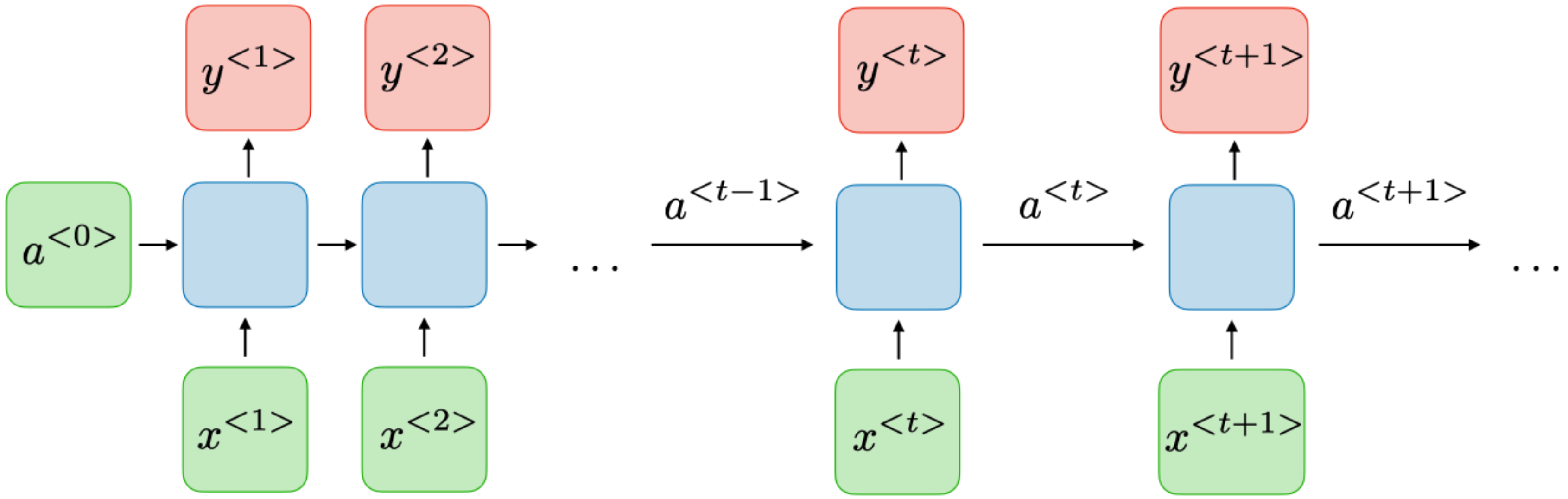
- Let's take recurrent neural network (RNN) as an example.



We will run this quick, full details in DL4NLP.

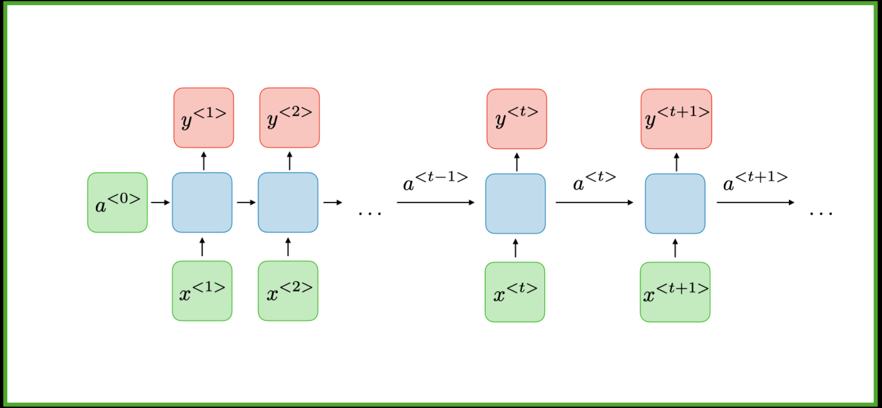
$$\mathbf{x} \leftarrow f(\mathbf{x}\mathbf{W} + \mathbf{b})$$

$$\mathbf{a}_i \leftarrow f(\underset{\substack{\uparrow \\ \text{Input token representation}}}{\mathbf{x}_i}\mathbf{W}_x + \underset{\substack{\uparrow \\ \text{RNN hidden state}}}{\mathbf{a}_{i-1}}\mathbf{W}_a + \mathbf{b})$$



$$y_i \leftarrow \mathbf{a}_i\mathbf{W}_{out}$$

```python
class RNN(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):
        # i: input token, h: hidden state, o: output
        self.i2h = nn.Embedding(input_size, hidden_size)
        self.h2h = nn.Linear(hidden_size, hidden_size)
        self.h2o = nn.Linear(hidden_size, output_size) # output_size: number of labels

    def forward(self, x, hidden_state):
        x = self.i2h(x)
        hidden_state = self.h2h(hidden_state)
        hidden_state = torch.tanh(x + hidden_state)
        out = self.h2o(hidden_state)
        return out, hidden_state
```



Training Regime

- Model:
  - Let’s take this RNN model as an example.
- Data:
  - A corpus with sentences like: `a tablespoon of apricot jam`
- Input:
  - `a tablespoon of apricot`
- Output label:
  - `jam`

Training Regime

- Model:
  - Let’s take this RNN model as an example.
- Data:
  - A corpus with sentences like: `a tablespoon of apricot jam`
- Input:
  - `a tablespoon of`
- Output label:
  - `apricot`

Training Regime

- Model:
  - Let’s take this RNN model as an example.
- Data:
  - A corpus with sentences like: `a tablespoon of apricot jam`
- Input:
  - `a tablespoon`
- Output label:
  - `of`

Training Regime

- Model:
  - Let's take this RNN model as an example.
- Data:
  - A corpus with sentences like: `a tablespoon of apricot jam`
- Input:
  - `a`
- Output label:
  - `tablespoon`

Summary: How to train a neural LM

- Collect training data.
- Train the model:
  - Take a tokenized sequence
  - Input: $t_1, t_2, t_3, \ldots, t_{n-1}$
  - Output: logits (or probability) of t_n.
- The LM is typically a neural model,
  - trained as a typical neural model.