Deep Learning for Natural Language Processing

Sidharth Mudgal

April 4, 2017

Table of contents

Intro

Intro

Deep Learning

Core Idea

- Given data that has hard to describe complex intrinsic structure
  - Important: DL usually works well only when data has unknown hidden structure. Not a silver bullet for all tasks
- Learn hierarchical representations of data points to perform a task

Typical flow

- Input representation
- Hidden representations (higher level features)
- Output prediction

Main Challenge

- Assisting the neural network to learn to create good representations
- How?
  - Encoding domain knowledge
  - Make training easier
    - Improved optimization techniques
    - Better gradient flow

Model Engineering

- DL represents a shift in the way we think about ML
  - Traditional ML: Encode domain knowledge using feature engineering
  - “Deep” ML: Encode domain knowledge using model engineering

Intro

Natural Language Processing

What is NLP? (from Stanford CS224n)

- **Natural language processing** is a field at the intersection of
  - computer science
  - artificial intelligence
  - and linguistics.
- **Goal:** for computers to process or "understand" natural language in order to perform tasks that are useful, e.g.,
  - Performing Tasks, like making appointments, buying things
  - Question Answering
    - Siri, Google Assistant, Facebook M, Cortana … thank you, mobile!!!
- Fully **understanding and representing** the **meaning** of language (or even defining it) is a difficult goal.
  - Perfect language understanding is AI-complete

Applications (from Stanford CS224n)

- Search (written and spoken)
- Online advertisement matching
- Automated/assisted translation
- Sentiment analysis for marketing or finance/trading
- Speech recognition
- Chatbots / Dialog agents
  - Automating customer support
  - Controlling devices
  - Ordering goods

The NLP Formula: Hierarchical Feature Generation

- Input representation (Word Vectors)
- Hidden representations (Typically using LSTM RNNs)
- Output prediction (Typically using a variant of softmax)

Word Vectors

Word Vectors

Motivation

What do we want?

We want a representation of words that:

- Captures meaning
- Is efficient

Past attempts to capture meaning of words - WordNet

```
from nltk.corpus import wordnet as wn
panda = wn.synset('panda.n.01')
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

(here, for *good*):

```
S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced,
proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good
```

A simple idea

"You shall know a word by the company it keeps" (J. R. Firth 1957: 11)

Ways to create word embeddings

- Traditional count based methods
  - SVD, PPMI, etc.
- Neural network based methods
  - Word2Vec et al.
- Hybrid
  - Glove

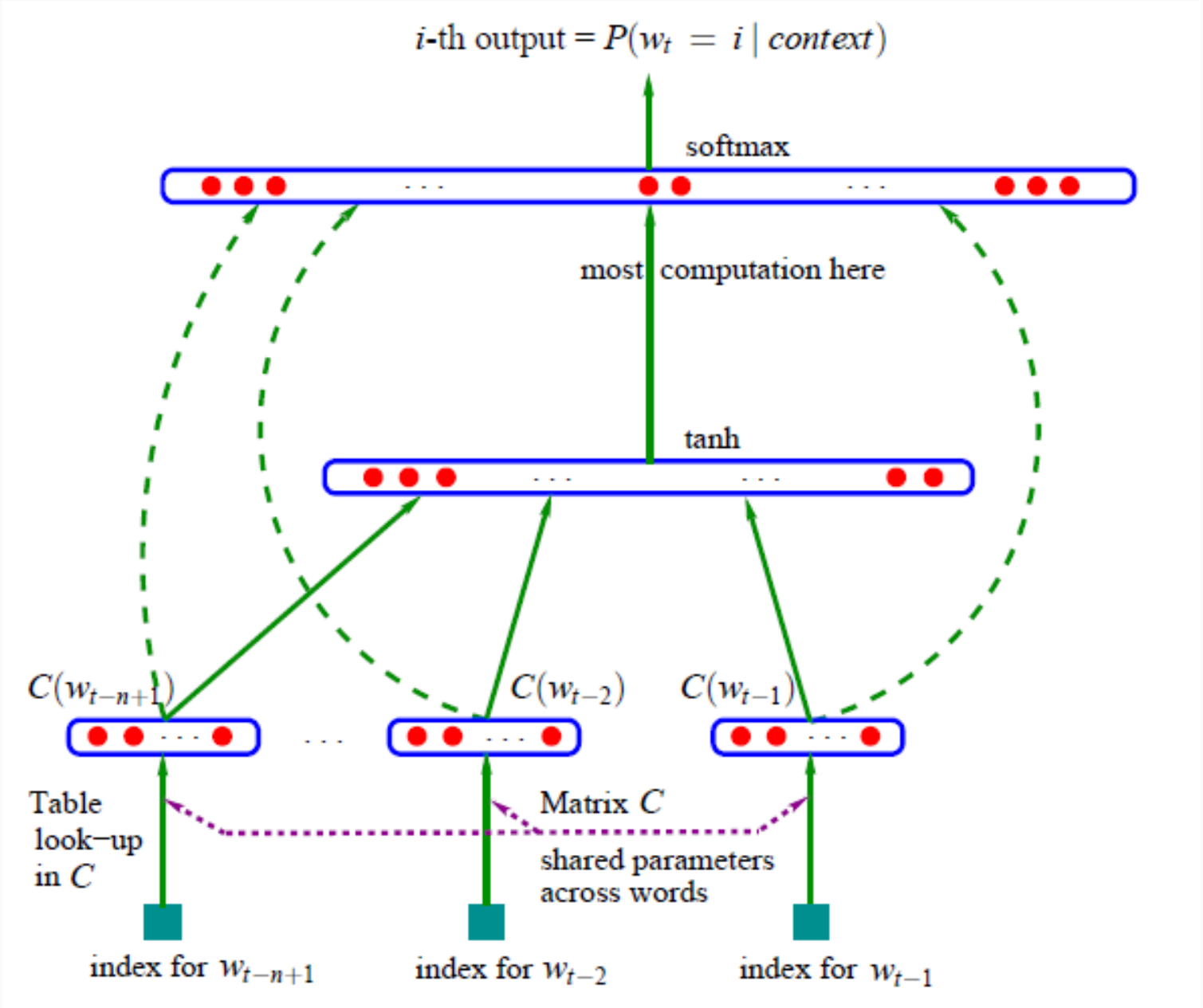
Word Vectors

Language Modeling

What is a Language Model?

- A probability distribution over sequences of words
  $P(w_1, \cdots, w_m) = \prod_{t=1}^{m} P(w_t | w_{t-1}, \cdots, w_1)$
- In practice, we only condition on the last M words
  $P(w_1, \cdots, w_m) = \prod_{t=1}^{m} P(w_t | w_{t-1}, \cdots, w_{t-M})$

Training word vectors using LMs (Bengio et al. 2003

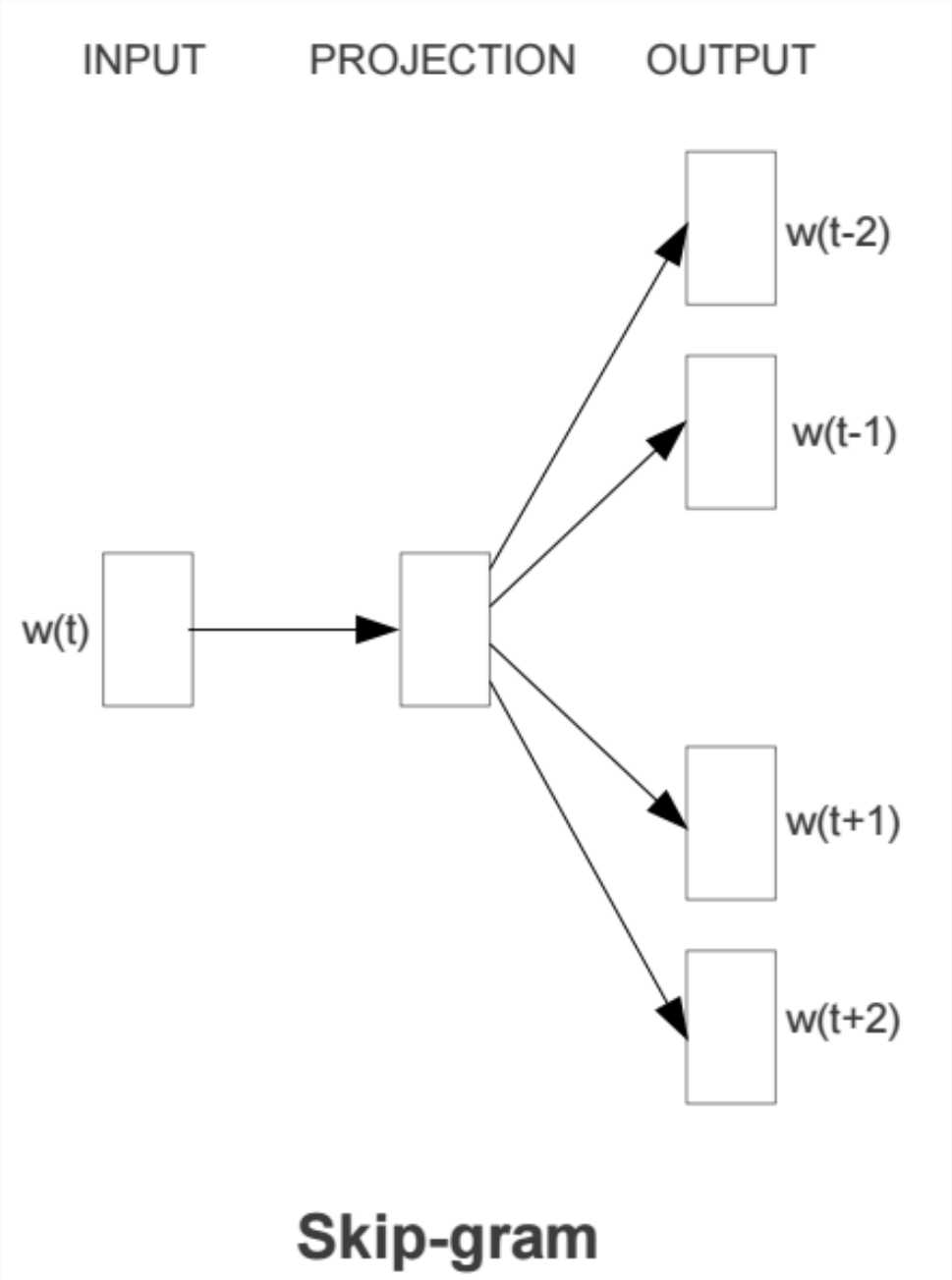


Word2Vec

Core Idea

- Parameterize words as vectors
- Use an NN to *predict* nearby words
  - Skip Gram
  - CBOW
- Hope that learned word vectors are semantically meaningful

Skip Gram Model



Objective Function (from Stanford CS224n)

For each word $t = 1 \ldots T$, predict surrounding words in a window of "radius" m of every word.

Objective function: Maximize the probability of any context word given the current center word:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} p(w_{t+j} | w_t ; \theta)$$

Negative Log Likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log p(w_{t+j} | w_t)$$

Where θ represents all variables we will optimize

How do we predict nearby words? (from Stanford CS224n)

Predict surrounding words in a window of radius *m* of every word

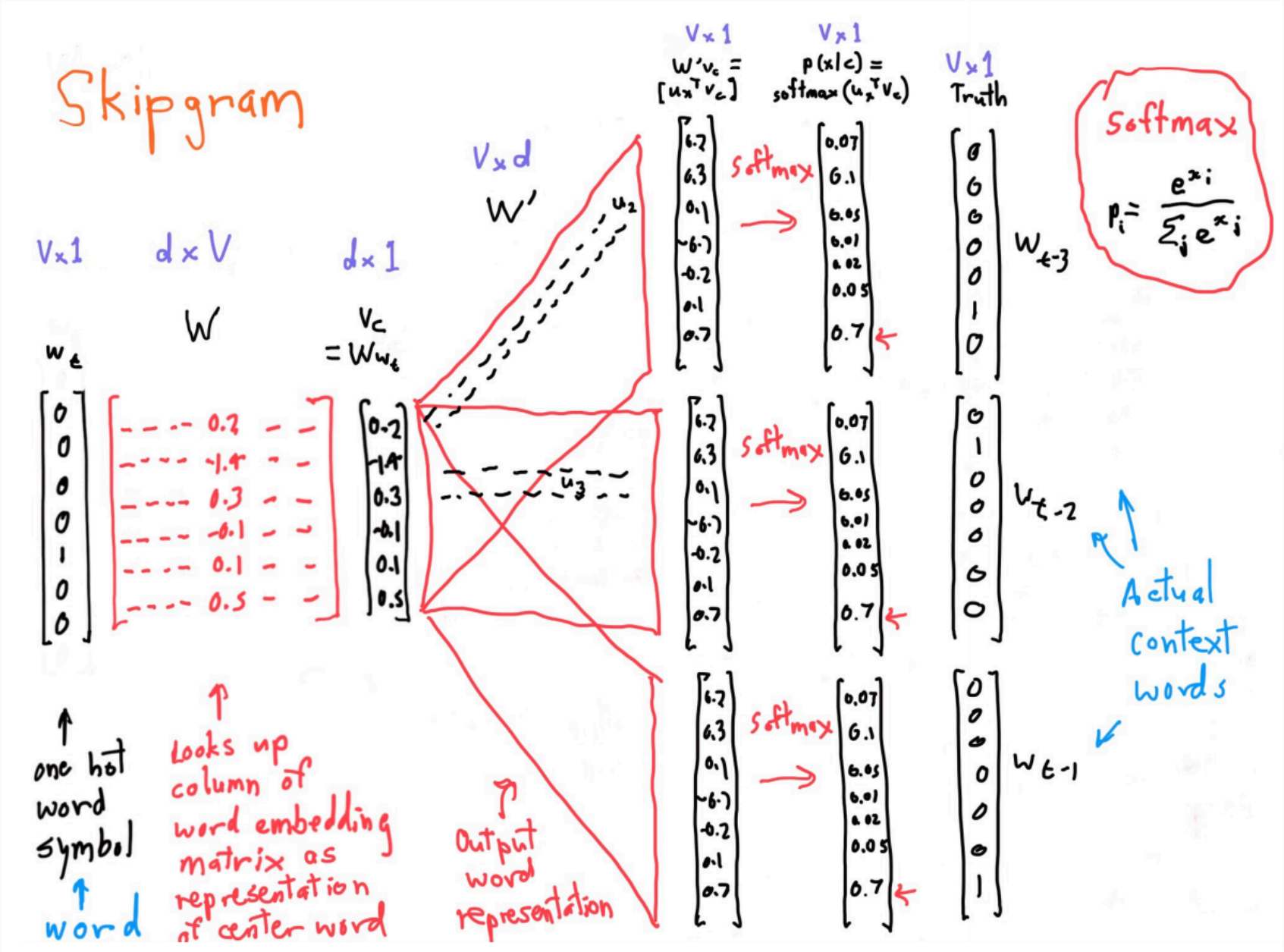
For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

where *o* is the outside (or output) word index, *c* is the center word index, v_c and u_o are "center" and "outside" vectors of indices *c* and *o*

Softmax using word *c* to obtain probability of word *o*

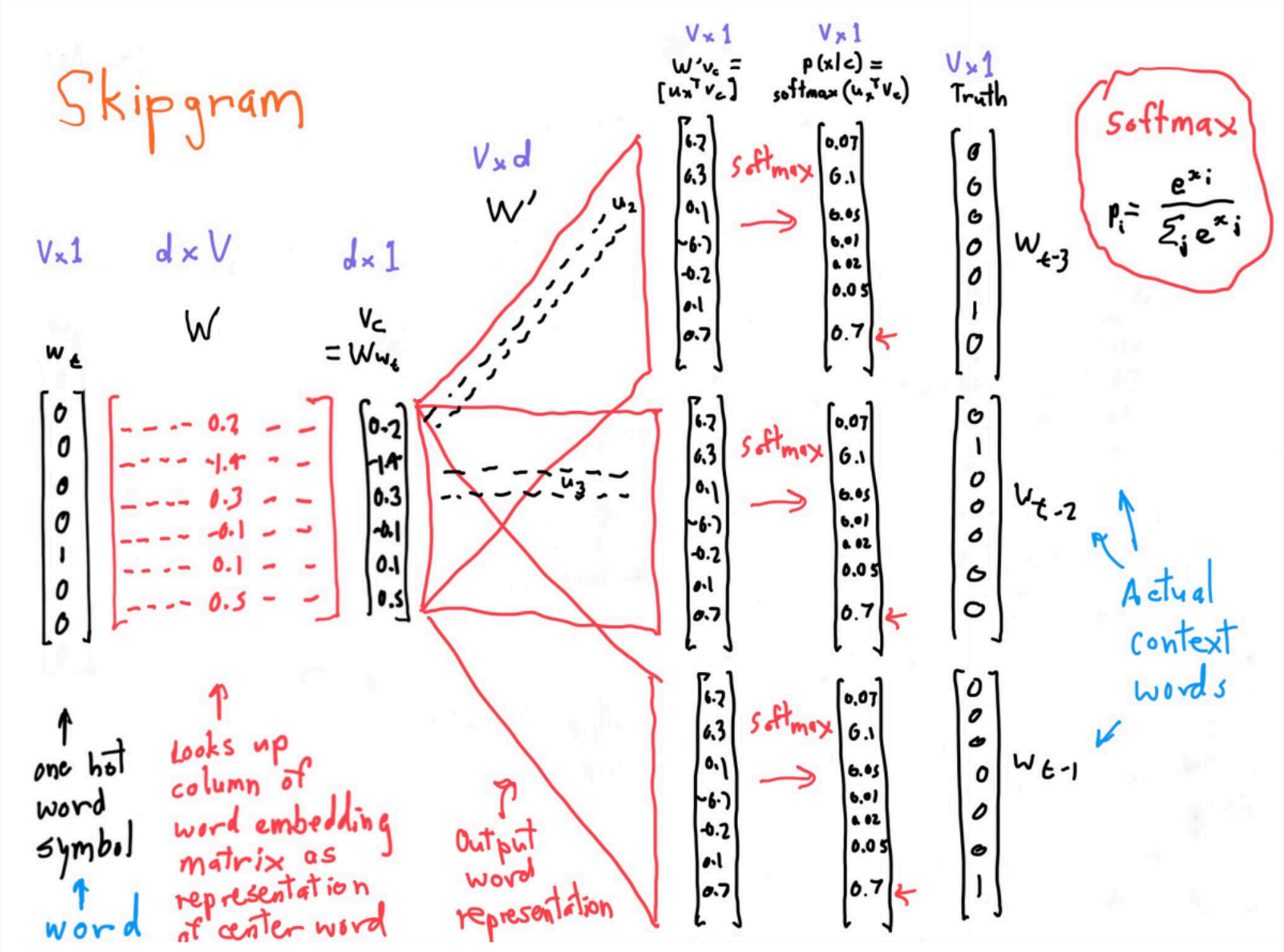
Step by step (from Stanford CS224n)



The problem of Softmax

- Scalability wrt vocabulary size
  - Do we really need to compute the probability of ALL words during training?
  - Do we really need to push down the prob. of ALL wrong words during training?
- Solutions:
  - Hierarchical Softmax
  - Negative sampling

The problem of Softmax



Hierarchical Softmax - Simple version

- "Preprocess" step:
  - Randomly split the vocabulary V into $\sqrt{|V|}$ clusters consisting of approximately $\sqrt{|V|}$ words each
  - I.e., each word w in V is permanently assigned to a cluster $C(w)$ at the beginning of training

Hierarchical Softmax - Simple version

The Idea

- Compute the probability $p(w_{t-1}|\vec{c}_t)$ in three steps:
  - First, compute the probability of the word w_{t-1} belonging to its true cluster $C(w_{t-1})$, i.e., $p(C(w_{t-1})|\vec{c}_t)$
  - Next, compute the probability of the word within its true cluster $C(w_{t-1})$, $p(w_{t-1}|C(w_{t-1}), \vec{c}_t)$
  - Then, $p(w_{t-1}|\vec{c}_t)$ is the product of the two probabilities:

$$\begin{aligned} p(w_{t-1}|\vec{c}_t) &= p(w_{t-1}, C(w_{t-1})|\vec{c}_t) \\ &= p(C(w_{t-1})|\vec{c}_t) \times p(w_{t-1}|C(w_{t-1}), \vec{c}_t) \end{aligned}$$

Hierarchical Softmax - Simple version

- Computing cluster probability $p(C(w_{t-1})|\vec{c}_t)$:
  - Use a regular softmax NN with $\sqrt{|V|}$ classes, 1 for each cluster.

Hierarchical Softmax - Simple version

- Computing word in cluster probability $p(w_{t-1}|C(w_{t-1}), \vec{c}_t)$:
  - To do so, each word cluster has a dedicated softmax NN with approximately $\sqrt{|V|}$ classes, 1 for each word in the cluster.

Hierarchical Softmax

- Time complexity of simple version = $O(\sqrt{|V|})$
- Can do much better ($O(\log |V|)$) with more complex versions
  - Frequency based clustering of words
  - Brown clustering
  - etc...

Negative Sampling

- Radical alternative
- Instead of literally predicting nearby words, we learn to distinguish true nearby words from randomly selected words
- Consider example of predicting w_{t-1}
  - Sample k negative words $n_{t-1}^1, \cdots, n_{t-1}^k \in V$, i.e., k words in V other than w_{t-1}
  - Compute dot products only for the target word w_{t-1} and the k negative samples
  - Instead of computing a probability using a softmax, the model is supposed to make a binary decision for each word using a sigmoid, i.e., predict 1 for word w_{t-1} and 0 for words $n_{t-1}^1, \cdots, n_{t-1}^k$

Results

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

E.g.: France - Paris + Italy $\approx$ Rome

Drawbacks

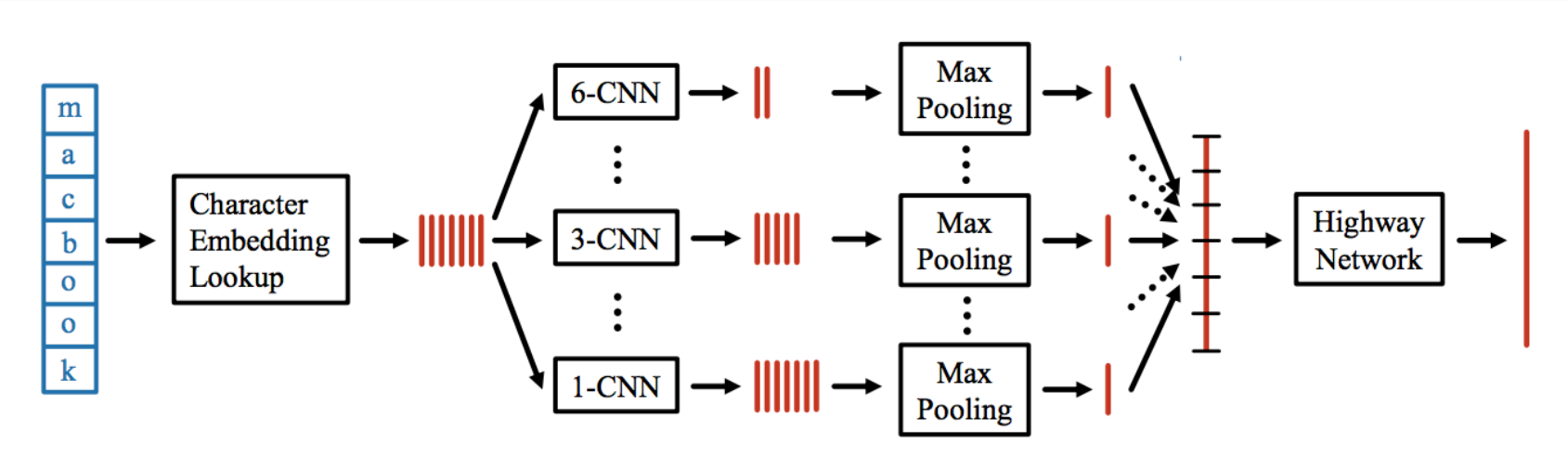
- Learns word vectors independently for each word
- Cannot estimate word embeddings of out-of-vocabulary (OOV) words

Char Level Word Embeddings

Core Idea

- Make use of fact that words are composed of characters
- Parameterize characters as vectors
- Compose char vectors to create word vectors
    - Can use CNN or RNN
- Use an NN to predict nearby words
- Hope that word vectors are semantically meaningful

The Word Embedding Network (WEN)



Step 1: Lookup Char Embeddings

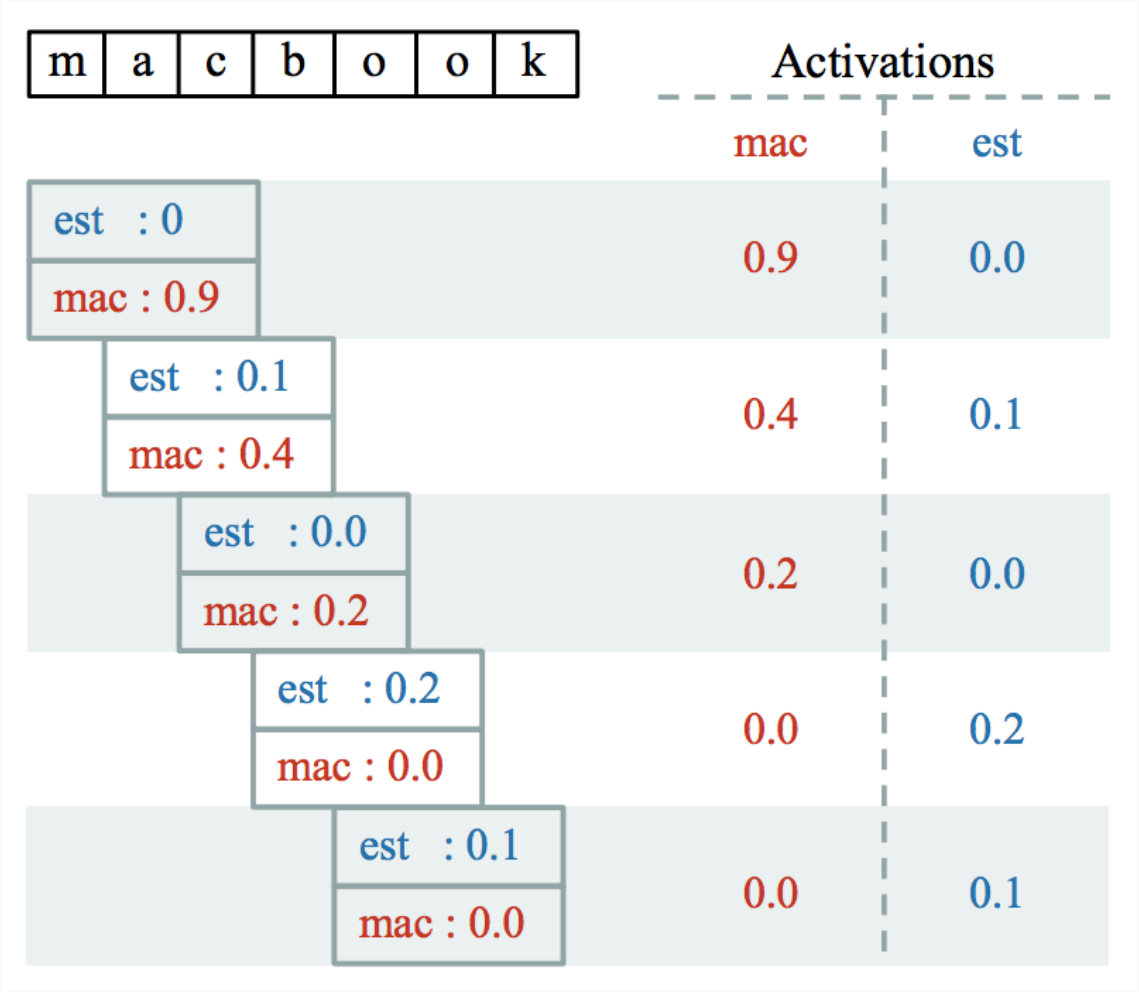
- Same as word vector lookup layer

Step 2: Temporal Convolution over Characters

- Assumption: Words are made of smaller units of character n-grams (morphemes)
- How do we encode this knowledge?
  - Use k Temporal CNNs with k kernel widths
  - Intuition: A CNN with kernel width j detects the presence of character n-grams of length j

Step 2: Temporal Convolution over Characters

- Example: Temporal convolution with kernel width = 3 and two feature maps sensitive to 3-grams “mac” and “est”



Step 2: Temporal Convolution over Characters

- The output of step 2 consists of two pieces of information:
  - Which character n-grams are present in a given input word?
  - Where do they occur in the word?

Step 3: Max-Pooling

- For each feature map in each CNN, output only the maximum activation
- Intuition: Get rid of the position information of character n-grams
- This yeilds a fixed dimensional vector

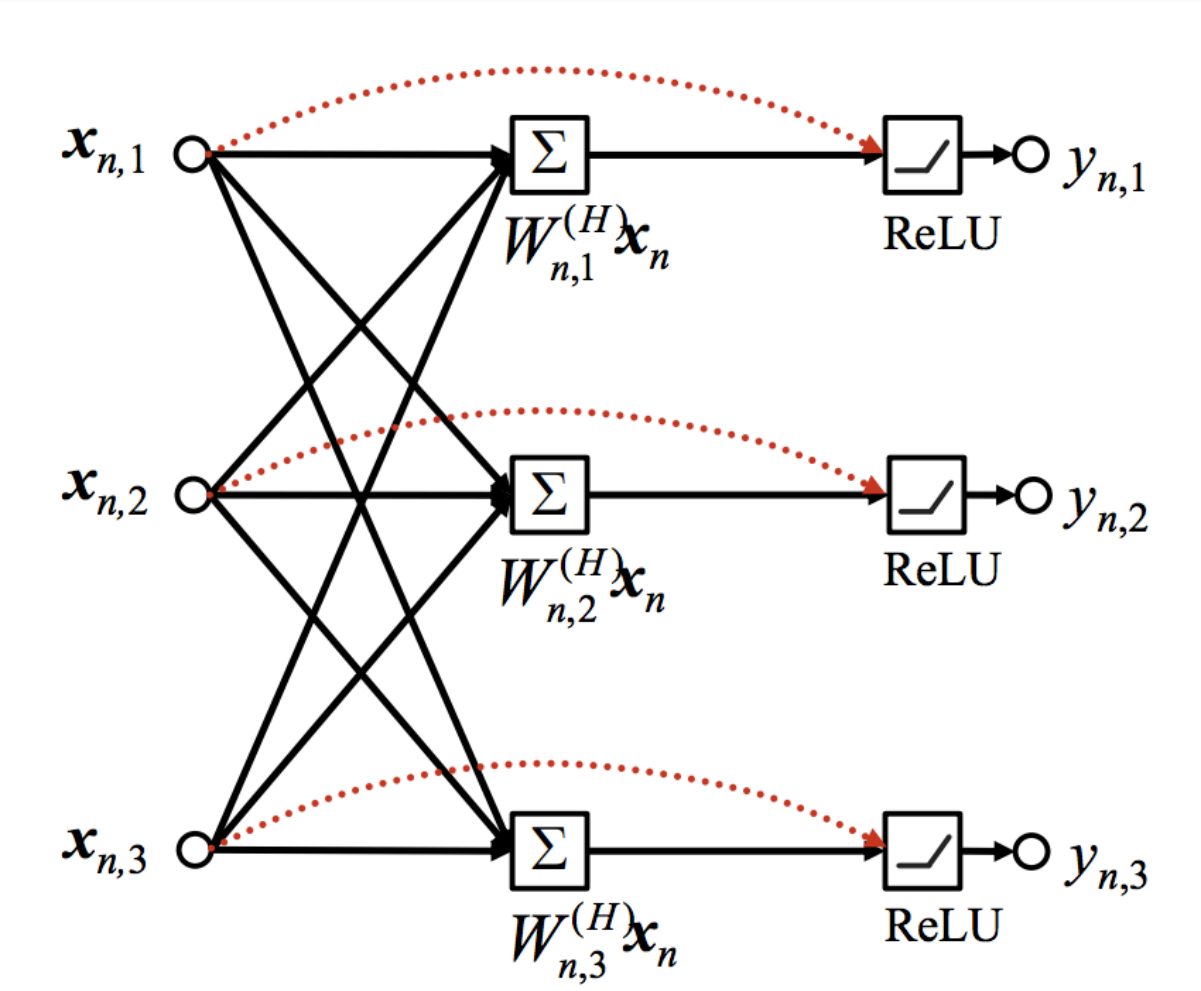
Step 4: Fully Connected Layers

- 3-layer “highway network”
- Intuition: Compose a word vector using information about which character n-grams are present in the word

Step 4: Highway NN Layer Details

- Core Idea: Provide a means to allow inputs to flow unaltered through an NN layer
  - This improves gradient flow
- Alternate pathways (highways) conditionally allows the input to pass through undeterred to the corresponding outputs as if the layer did not exist
- “Gates” determine whether to modify each dimension of the input

Step 4: Highway NN Layer Details



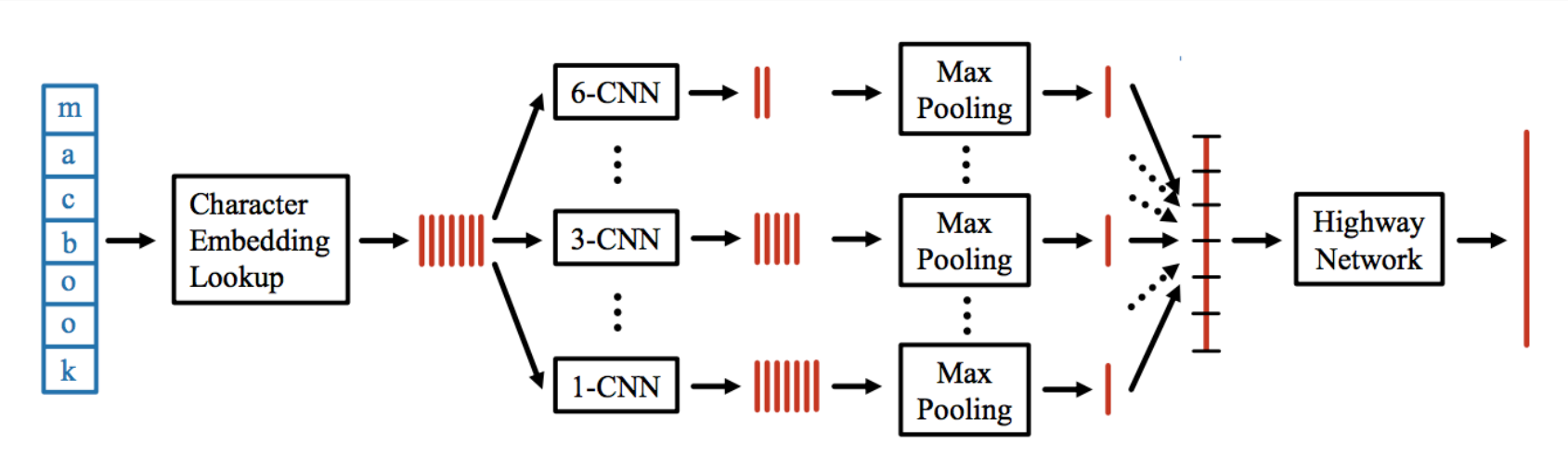
Step 4: Highway NN Layer Details

Given an input vector $\vec{q}$, the output of the n^{th} highway layer with input $\vec{x}_n$ and output $\vec{y}_n$ is computed as follows:

$$
\begin{aligned}
\vec{x}_n &= \begin{cases} \vec{q}, & \text{if } n = 1, \\ \vec{y}_{n-1}, & \text{if } n > 1 \end{cases} \\
H(\vec{x}_n) &= \vec{W}_n^{(H)} \vec{x}_n + \vec{b}_n^{(H)} \\
T(\vec{x}_n) &= \vec{W}_n^{(T)} \vec{x}_n + \vec{b}_n^{(T)} \\
\vec{y}_n &= H(\vec{x}_n) \odot T(\vec{x}_n) + \vec{x}_n \odot (1 - T(\vec{x}_n))
\end{aligned}
$$

Here H applies an affine transformation to $\vec{x}_n$ and T acts as the gate deciding which inputs should be transformed and by how much.

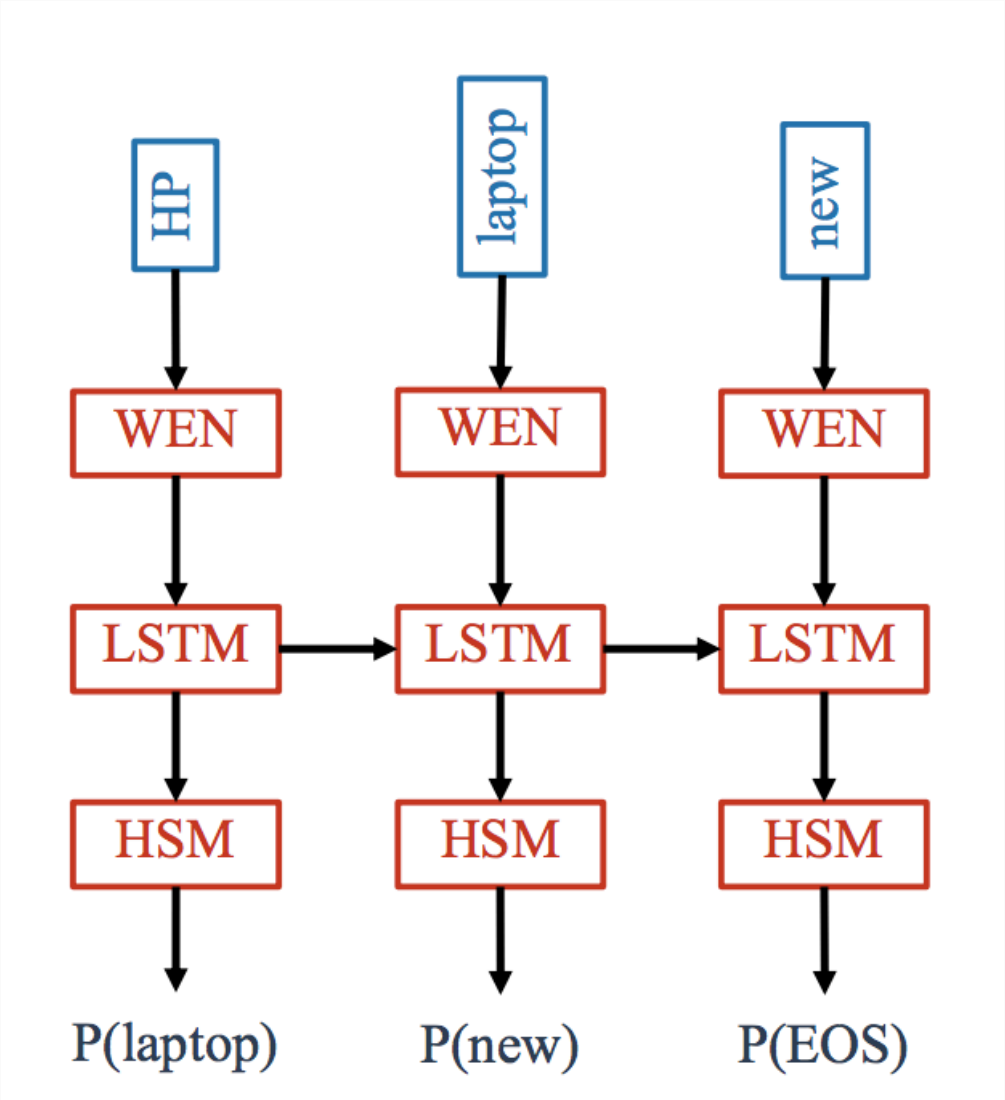
The Word Embedding Network (WEN)



Training the Word Embedding Network

- How can we train this model to produce good word embeddings?
  - Train it as part of a recurrent neural language model

Training


HP
laptop
new
WEN
WEN
WEN
LSTM
LSTM
LSTM
HSM
HSM
HSM
P(laptop)
P(new)
P(EOS)

Results

	In Vocabulary					Out-of-Vocabulary		
	while	*his*	*you*	*richard*	*trading*	*computer-aided*	*misinformed*	*looooook*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*	–	–	–
	letting	*her*	*we*	*robert*	*advertising*	–	–	–
	though	*my*	*guys*	*neil*	*turnover*	–	–	–
	minute	*their*	*i*	*nancy*	*turnover*	–	–	–
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*	*computer-guided*	*informed*	*look*
	whole	*hhs*	*young*	*rich*	*training*	*computerized*	*performed*	*cook*
	meanwhile	*is*	*four*	*richer*	*reading*	*disk-drive*	*transformed*	*looks*
	white	*has*	*youth*	*richter*	*leading*	*computer*	*inform*	*shook*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*	*computer-guided*	*informed*	*look*
	whole	*this*	*your*	*gerard*	*training*	*computer-driven*	*performed*	*looks*
	though	*their*	*doug*	*edward*	*traded*	*computerized*	*outperformed*	*looked*
	nevertheless	*your*	*i*	*carl*	*trader*	*computer*	*transformed*	*looking*

Application: Entity Matching

The Task

- Given text descriptions of two entities, determine if they refer to the same real world entity
  - Binary classification problem

LG 32MA68HY-P 32-Inch IPS Monitorr. The 32-inch Class screen provides 75% more viewable area compared to a 24-Inch monitor. The 32-inch Class screen (31.5" measured diagonally) provides 75% more Viewable area compared to a 24-Inch monitor. The full HD clarity and true Color reproduction of the 16:9 IPS panel enhance every project, even when viewed off-angle. […]

(a)

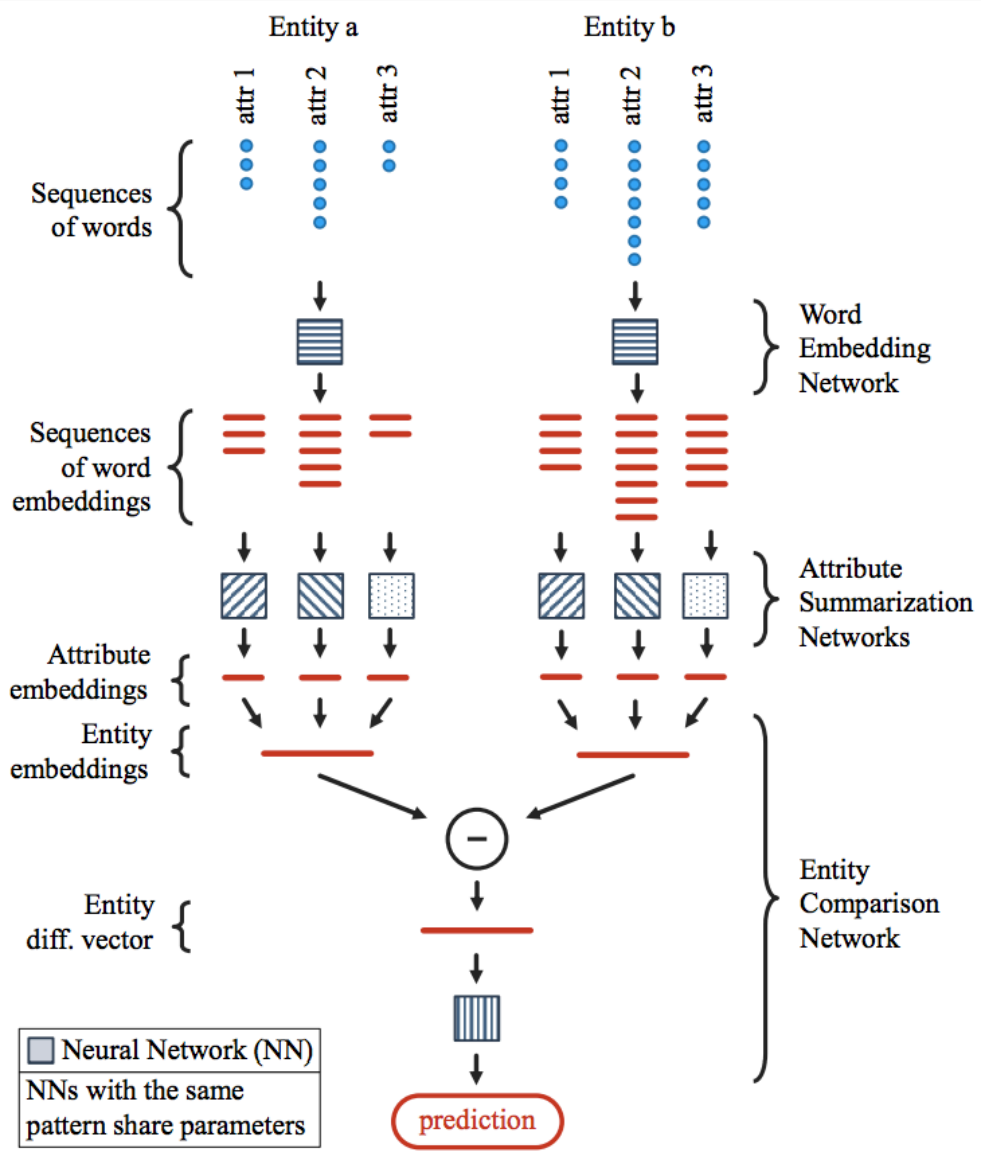
The 48in class 32MA68HY-P 18:10 monitor from LG offers more than a 60% larger screen area vs. a 32in class monitor. That's plenty of room for work on spreadsheets, editing documents, watching videos, viewing photos and more. Supports Display Port, HDMI, D-Sub, USB 2.0 (x2), and USB 3.0. […]

(b)

The Task

- Typical approach: Information Extraction (IE)
  - Followed by training a traditional classifier: SVM, Random Forest etc.
- Proposed alternate approach: Neural Entity Matching Model (Nemo)
  - Use deep learning to perform this end to end
  - Train a single model to perform extraction and entity matching

Nemo Overview



Baseline - Magellan

- Generates string similarity metrics based on heursitics
  - Jaccard, Cosine, Levenshtein etc.
- Picks the best standard ML model using cross validation
  - SVM, Random forest etc.

Results

Dataset	Nemo			Magellan			F_1 absolute improvement	F_1 relative improvement (%)
	P	R	F_1	P	R	F_1		
Home	89.2	90.0	**89.6**	82.7	75.7	**79.0**	10.6	50.4
Electronics	91.2	93.7	**92.4**	89.6	77.2	**82.9**	9.5	55.7
Tools	94.5	95.9	**95.2**	92.6	83.8	**88.0**	7.2	60.1
Clothing	96.4	97.5	**97.0**	86.0	80.3	**83.0**	13.9	82.0
Company	90.1	87.8	**89.0**	90.3	79.7	**84.7**	4.3	28.1

Results - Extensive extraction

Dataset	Nemo			Magellan									
				Extracting top-5 attributes					Extracting top-30 attributes				
	P	R	F_1	P	R	F_1	F_1 absolute improvement	F_1 relative improvement (%)	P	R	F_1	F_1 absolute improvement	F_1 relative improvement (%)
Home	89.2	90.0	**89.6**	90.1	85.9	**87.9**	1.7	13.8	91.2	88.1	**89.6**	0.0	0.0
Electronics	91.2	93.7	**92.4**	94.8	87.1	**90.8**	1.6	17.9	94.3	88.3	**91.2**	1.2	13.8
Tools	94.5	95.9	**95.2**	95.9	91.5	**93.7**	1.5	24.1	95.9	91.8	**93.8**	1.4	22.9
Clothing	96.4	97.5	**97.0**	97.5	95.3	**96.4**	0.6	15.9	97.3	95.6	**96.5**	0.5	13.8

Understanding the model

- Compute first order saliency scores for word embeddings
  - Essentially the derivative of word embeddings wrt the output of the model
  - Measure of sensitivity

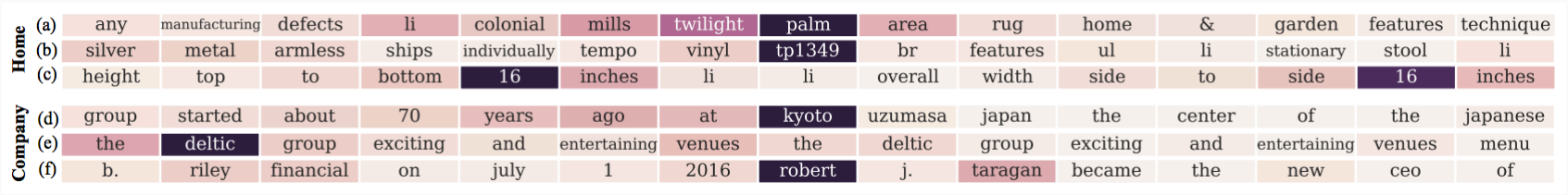
Understanding the model

Example:

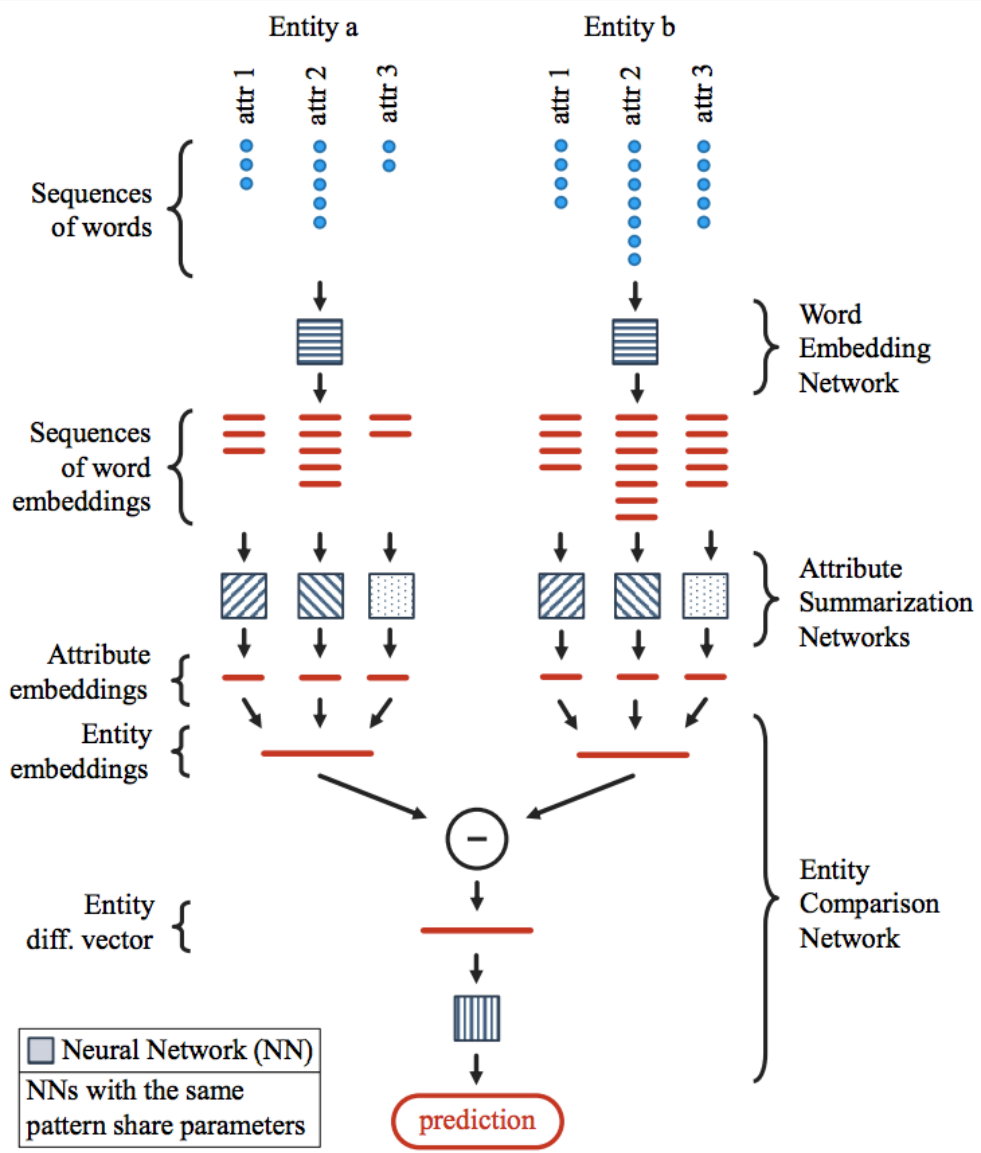
- Assume we have an entity pair (e_1, e_2)
- We want to compute the saliency of the t^{th} word in e_1
- If y_L represents the score that Nemo assigns to the correct label L (match or mismatch) for the pair, $\vec{x_t}$ represents Nemo's input units for the t^{th} word and $\vec{w_t}$ is the word embedding of the t^{th} word in e_1, then the saliency s of the word is calculated as

$$s = \left\| \frac{\partial y_L}{\partial \vec{x_t}} |_{\vec{x_t} = \vec{w_t}} \right\|$$

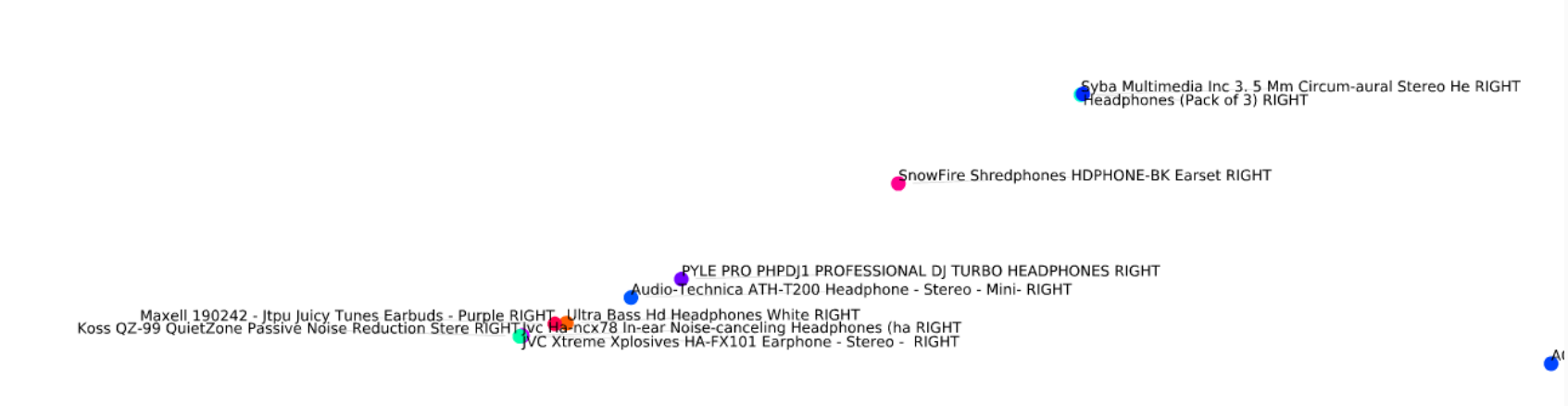
Saliency Visualization



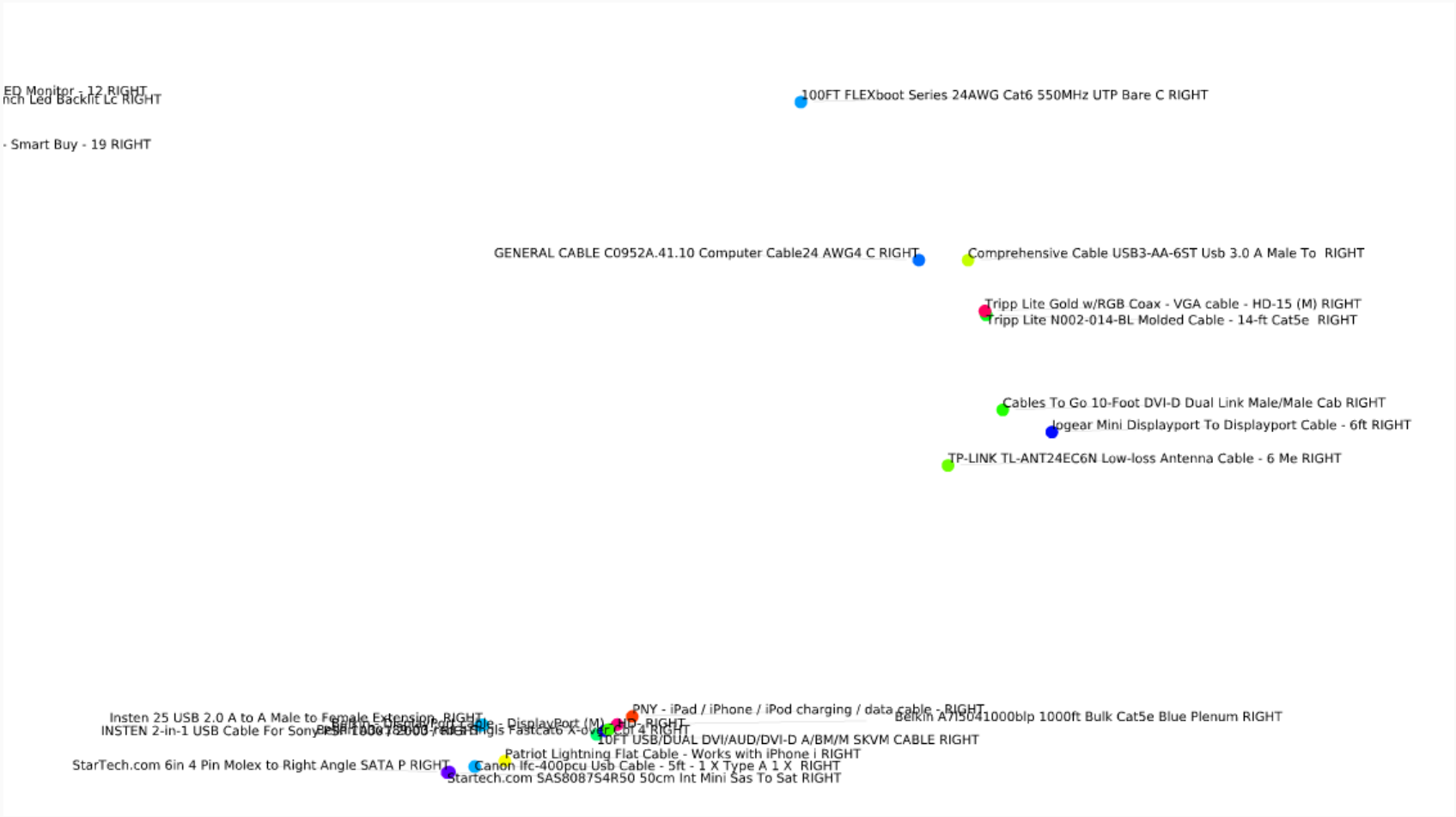
Entity Embedding Visualization



Entity Embedding Visualization - Headphones

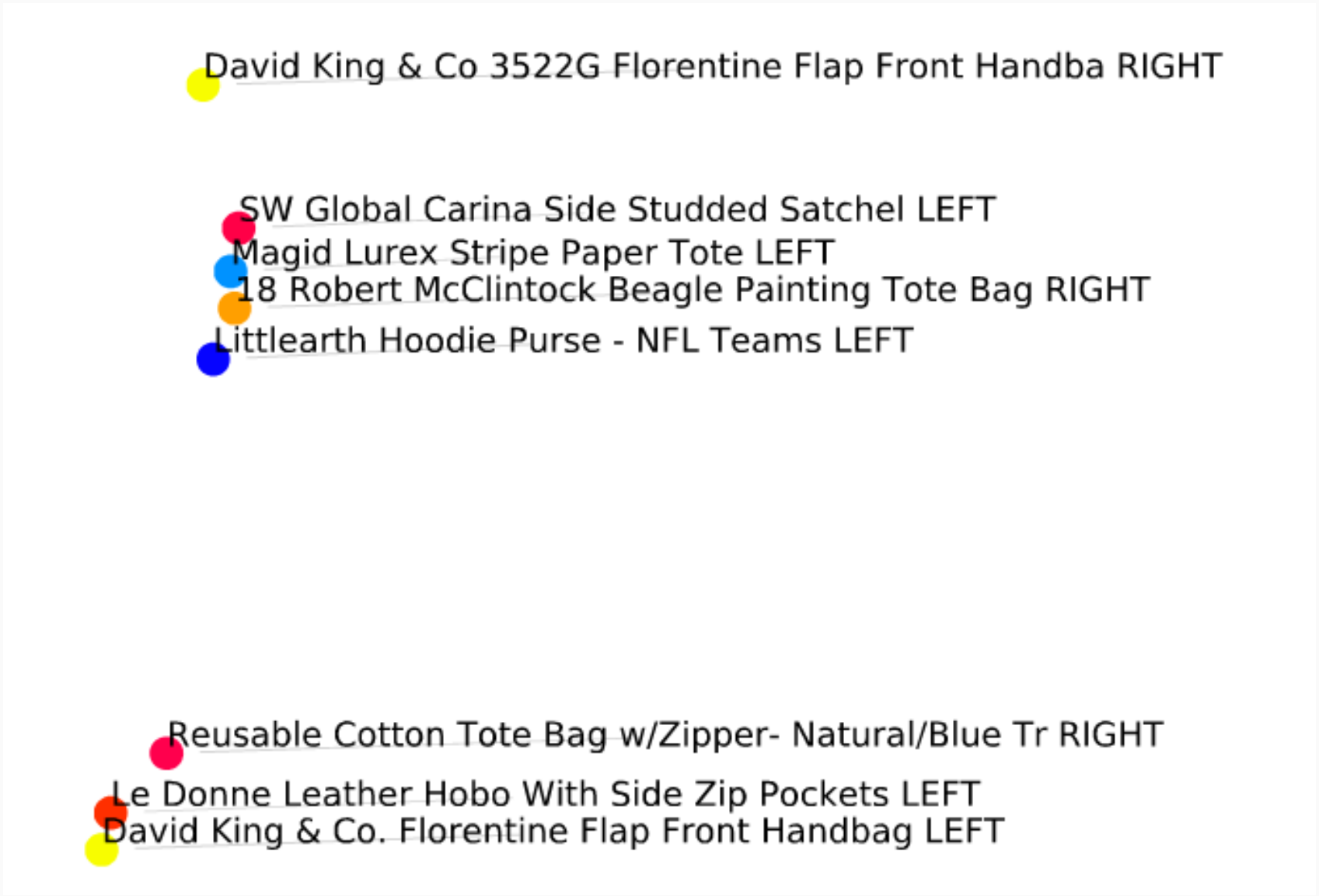


Entity Embedding Visualization - Cables

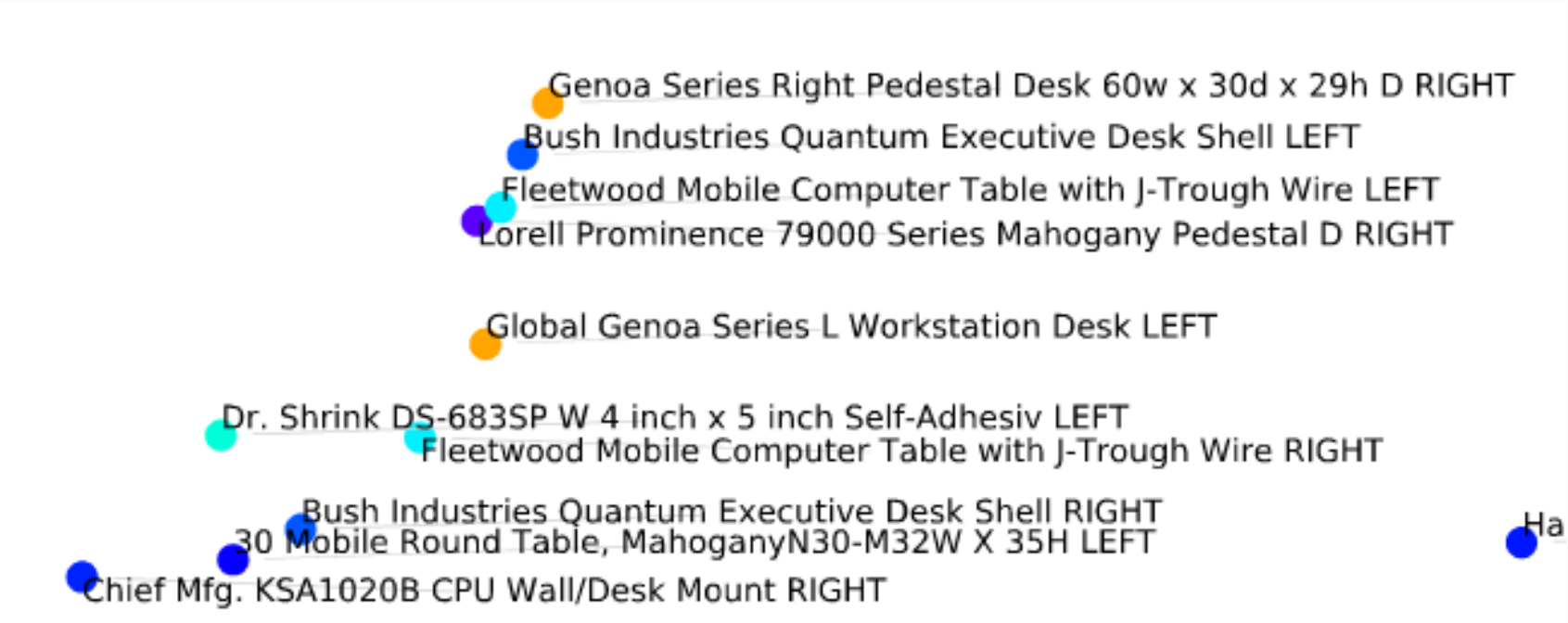


Entity Embedding Visualization - Shirts

Entity Embedding Visualization - Bags



Entity Embedding Visualization - Desks



Conclusion

Deep Learning: An incredibly promising tool

- Has produced breakthroughs in several domains
- But has a lot of open problems
  - Large quantities of labeled data
  - Huge number of parameters
  - Optimization challenges