ENGINEERING
W Booth School of Engineering
Practice and Technology

McMaster University

Computational Natural Language Processing

Overview of NLP and Word Vectors

Hamidreza Mahyar
mahyarh@mcmaster.ca

Lecture Plan

Lecture 1: Introduction and Word Vectors

1. The course
2. Human language and word meaning
3. Word2vec introduction
4. Word2vec objective function gradients
5. Optimization basics
6. Looking at word vectors

Key learning today: The (astounding!) result that word meaning can be represented rather well by a (high-dimensional) vector of real numbers

Course logistics in brief

- Instructor: Hamidreza Mahyar
- Head TA: Ali Shiraee
- Time: Wednesday 11:30am–2:30pm
- We put a lot of other important information on Avenue to Learn. Please read it!

What do we hope to teach? (A.k.a. “learning goals”)

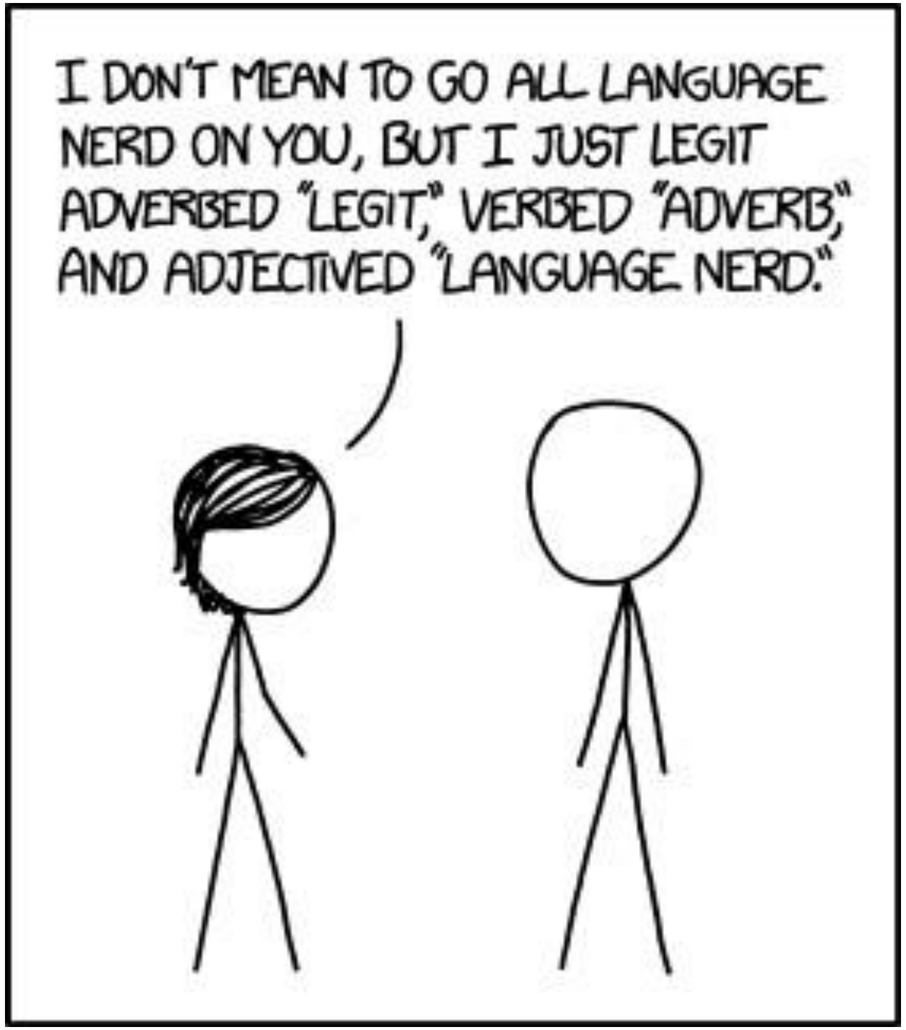
1. The foundations of the effective modern methods for deep learning applied to NLP
   - Basics first, then key methods used in NLP in 2024: Word vectors, feed-forward networks, recurrent networks, attention, encoder-decoder models, transformers, large pre-trained language models, etc.

2. A big picture understanding of human languages and the difficulties in understanding and producing them via computers

3. An understanding of and **ability to build systems** (in PyTorch) for some of the major problems in NLP:
   - Word meaning, dependency parsing, machine translation, question answering

Course work and grading policy

- 4 Assignments: 20%
- 4 Quizzes: 10%
- Midterm: 35%
- Final Group Project (1–3 people): 34%
- Late day policy
  - NO free late days; 1% off course grade per day late
  - Assignments not accepted more than 3 days late unless given permission in advance
- Collaboration policy: Understand allowed collaboration and how to document it: Don’t take code off the web; acknowledge working with other students; write your own assignment solutions


I DON'T MEAN TO GO ALL LANGUAGE NERD ON YOU, BUT I JUST LEGIT ADVERBED "LEGIT," VERBED "ADVERB," AND ADJECTIVED "LANGUAGE NERD."


ANNO
oo
nn
mm
ll
kk
ii
hh
gg
ff
ee
dd
cc
bb

Trained on text data, neural machine translation is quite good!

TUKO

BEST DIGITAL NEWS PLATFORM

https://kiswahili.tuko.co.ke/

Malawi yawapoteza mawaziri 2 kutokana na maafa ya COVID-19

TUKO.co.ke imefahamishwa kuwa waziri wa serikali ya mitaa Lingson Belekanyama na mwenzake wa uchukuzi Sidik Mia walifariki dunia ndani ya saa mbili tofauti.

Swahili English

Google Translate

Malawi loses 2 ministers due to COVID-19 disaster

TUKO.co.ke has been informed that local government minister Lingson Belekanyama and his transport counterpart Sidik Mia died within two separate hours.

GPT-3: A first step on the path to foundation models

OpenAI

The SEC said, "Musk, your tweets are a blight.
They really could cost you your job,
if you don't stop all this tweeting at night."

Then Musk cried, "Why?
The tweets I wrote are not mean,
I don't use all-caps
and I'm sure that my tweets are clean."

"But your tweets can move markets
and that's why we're sore.
You may be a genius and a billionaire,
but it doesn't give you the right to
be a bore!"

S: I broke the window.
Q: What did I break?
S: I gracefully saved the day.
Q: What did I gracefully save?
S: I gave John flowers.
Q: Who did I give flowers to?
S: I gave her a rose and a guitar.
Q: Who did I give a rose and a guitar to?

How many users have signed up since the start of 2020?

SELECT count(id) FROM users
WHERE created_at > '2020-01-01'

What is the average number of influencers each user is subscribed to?

SELECT avg(count) FROM (SELECT user_id, count(*)
FROM subscribers GROUP BY user_id)
AS avg_subscriptions_per_user

ChatGPT: A recent, intriguing set of capabilities

You

summarize McMaster's NLP course in one sentence, seemlessly integrating how delicious tea is into your summary.

ChatGPT

McMaster's NLP course expertly blends complex linguistic algorithms with the soothing complexity of a perfectly steeped cup of tea, creating an intellectually stimulating and refreshingly aromatic learning experience.

ChatGPT: A recent, intriguing set of capabilities

OpenAI

Convert this python dictionary into a LaTeX table that uses the `booktabs` package, and make sure there are no vertical lines.

```
{"Models" : ["baseline", "ours"],
 "Accuracy": [0.7, 0.8],
 "Time/sentence": ["4s", "6s"]
}
```

Models	**Accuracy**	**Time/sentence**
baseline	0.7	4s
ours	0.8	6s

Here's an example of how you can convert the Python dictionary into a LaTeX table that uses the **`booktabs`** package and does not include any vertical lines:

Copy code

```
\begin{tabular}{l c c}
\toprule
\textbf{Models} & \textbf{Accuracy} & \textbf{Time/sentence}
\\
\midrule
baseline & 0.7 & 4s \\
ours & 0.8 & 6s \\
\bottomrule
\end{tabular}
```

ChatGPT: A recent, intriguing set of capabilities

Convert this python dictionary into a LaTeX table that uses the `booktabs` package, and make sure there are no vertical lines, and seemlessly integrate how delicious tea is into your description of how the resulting LaTeX code works.

headings, using the `\textbf{}` command to make them bold. The `\toprule`, `\midrule`, and `\bottomrule` commands are used to add horizontal lines to the table, as defined in the `booktabs` package, making the table aesthetically pleasing just like a well brewed tea with the right combination of tea leaves, temperature and steep time.

How do we represent the meaning of a word?

Definition: **meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

signifier (symbol) ⟺ signified (idea or thing)

= denotational semantics

tree ⟺ {🌳, 🌲, 🌴, …}

How do we have usable meaning in a computer?

Previously commonest NLP solution: Use, e.g., WordNet, a thesaurus containing lists of **synonym sets** and **hypernyms** (“is a” relationships)

e.g., synonym sets containing “good”:

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'} for synset in
wn.synsets("good"):
      print("{}: {}".format(poses[synset.pos()],
                  ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good adj:
good
adj (sat): full, good adj:
good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good adj
(sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g., hypernyms of “panda”:

```
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')]
```

Problems with resources like WordNet

- A useful resource but missing nuance:
  - e.g., “proficient” is listed as a synonym for “good”
    This is only correct in some contexts
  - Also, WordNet list offensive synonyms in some synonym sets without any coverage of the connotations or appropriateness of words
- Missing new meanings of words:
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can’t be used to accurately compute word similarity (see following slides)

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

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]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

Problem with words as discrete symbols

Example: in web search, if a user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”

But:

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]

These two vectors are orthogonal

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet’s list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

Representing words by their context

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent ***banking***

Word vectors

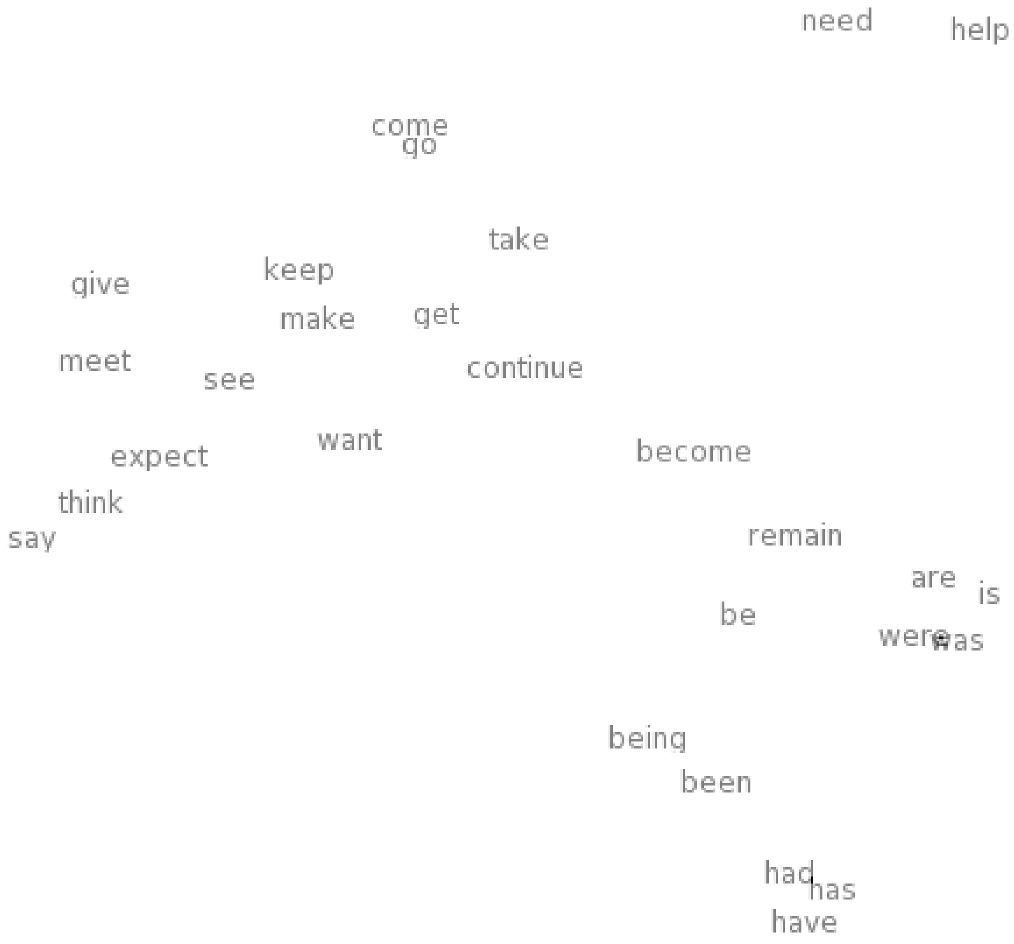
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

Note: word vectors are also called (word) embeddings or (neural) word representations
They are a distributed representation

Word meaning as a neural word vector – visualization

$$expect = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



3. Word2vec: Overview

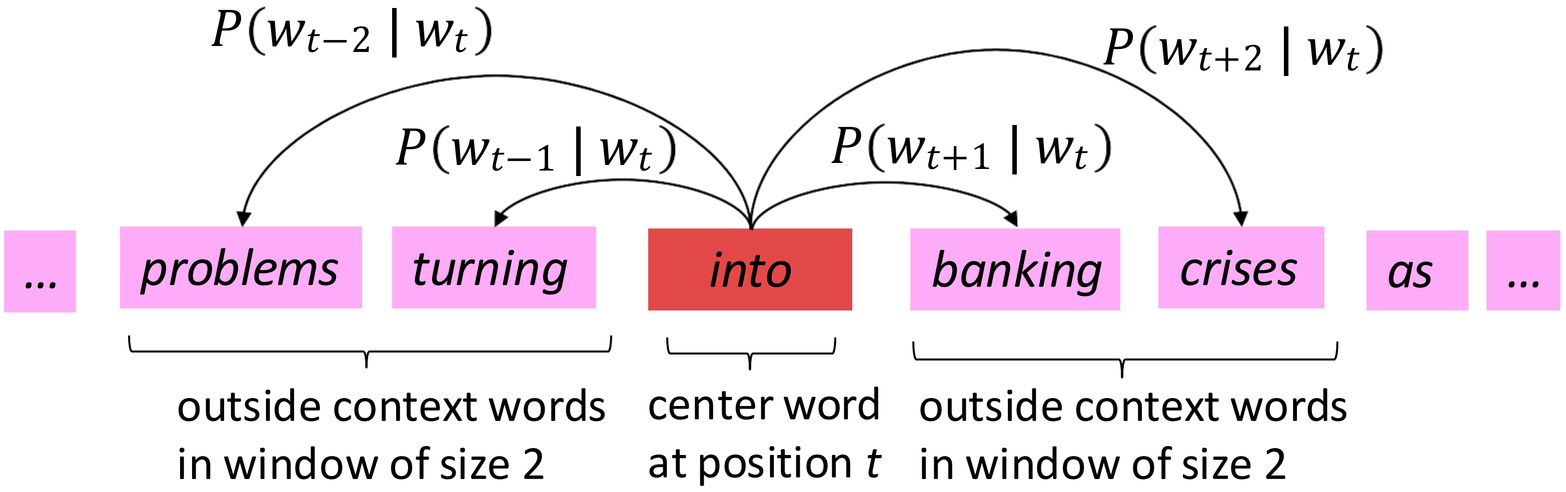
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus (“body”) of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context (“outside”) words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

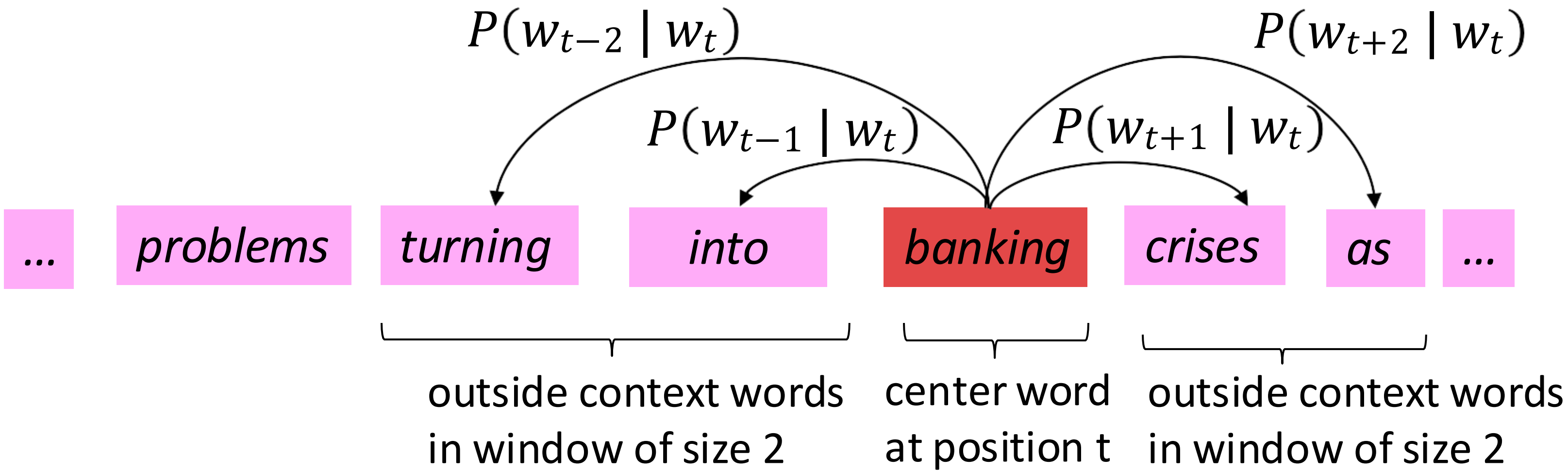
Word2Vec Overview

Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$



Word2Vec Overview

Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$



Word2vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function ⟺ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

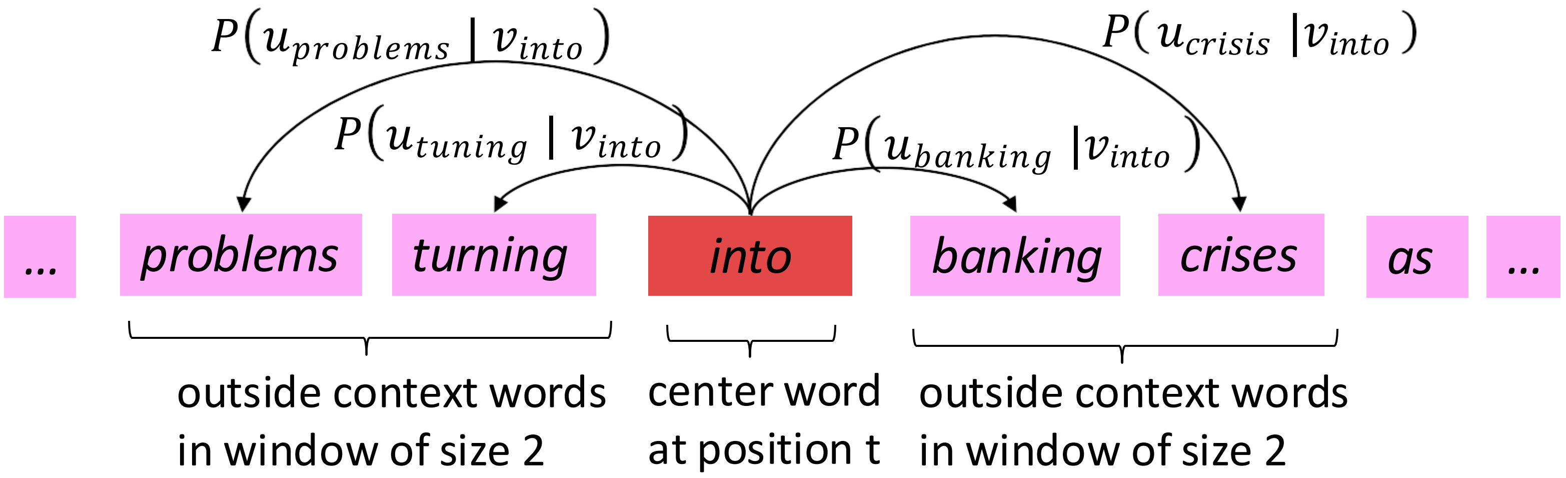
- **Question:** How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

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

Word2Vec with Vectors

- Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$
- $P\left(u_{problems} \mid v_{into}\right)$ short for $\mathrm{P}\left(problems \mid into\,; u_{problems}, v_{into}, \theta\right)$

All words vectors θ appear in denominator



Word2vec: prediction function

② Exponentiation makes anything positive

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

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

③ Normalize over entire vocabulary
to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$ Open region

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

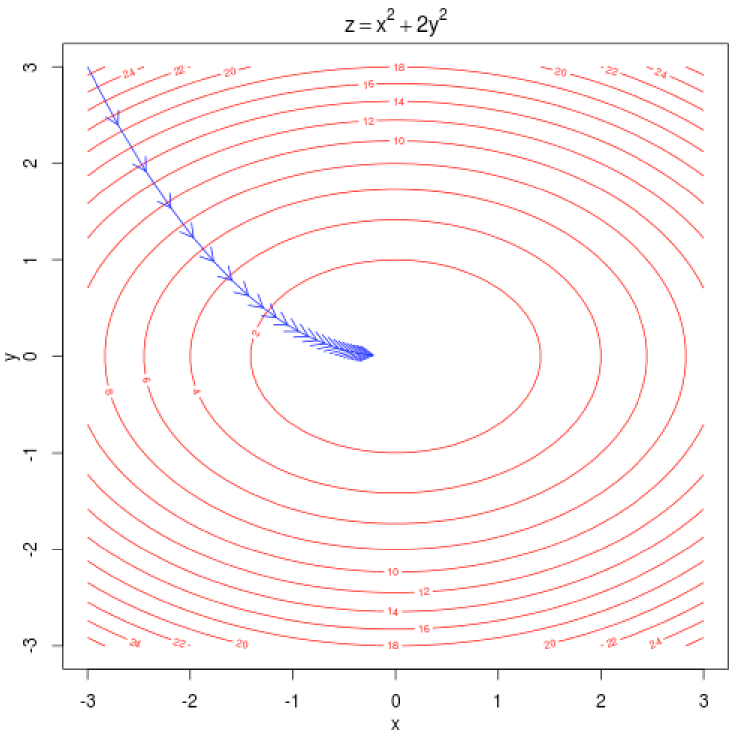
But sort of a weird name
because it returns a distribution!

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with *d*-dimensional vectors and *V*-many words, we have ➔
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!

4.

Objective Function

Maximize $J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} p(w_{t+j} \mid w_t; \theta)$

Or minimize ave. neg. log likelihood

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

[negate to minimize; log is monotone]

Text length

window size

where

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

word IDs

Each word type (vocab entry) has two word representations: as center word and context word

We now take derivatives to work out minimum

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w v_c)}$$

$$= \underbrace{\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c)}_{①} - \underbrace{\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c)}_{②}$$

① $$\frac{\partial}{\partial v_c} \underbrace{\log \exp}_{\text{inverses}} (u_o^T v_c) = \frac{\partial}{\partial v_c} u_o^T v_c = u_o$$

Vector!
Not high school single variable calculus

You can do things one variable at a time, and this may be helpful when things get gnarly.

$$\forall j \quad \frac{\partial}{\partial (v_c)_j} u_o^T v_c = \frac{\partial}{\partial (v_c)_j} \sum_{i=1}^{d} (u_o)_i (v_c)_i$$

$$= (u_o)_j$$

Each term is zero except when $i = j$

$$② \quad \frac{\partial}{\partial v_c} \underbrace{\log}_{f} \underbrace{\sum_{w=1}^{V} \exp(u_w^T v_c)}_{z = g(v_c)}$$

$$= \underbrace{\frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)}}_{} \cdot \underbrace{\frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c)}_{\frac{\partial z}{\partial v_c}}$$

Important to change index

$$\frac{\partial}{\partial v_c} f(\overbrace{g(v_c)}^{z}) = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_c}$$

Use chain rule

$$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \frac{\partial}{\partial v_c} \underbrace{\exp}_{f} (\underbrace{u_x^T v_c}_{z = g(v_c)}) \right)$$

Move deriv inside sum

$$\left(\sum_{x=1}^{V} \exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c \right)$$

Chain rule

$$\left(\sum_{x=1}^{V} \exp(u_x^T v_c) \, u_x \right)$$

$$\frac{\partial}{\partial v_c} \log(p(o|c)) = u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \exp(u_x^T v_c) u_x \right)$$

$$= u_o - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} u_x$$

Distribute term across sum

$$= u_o - \underbrace{\sum_{x=1}^{V} p(x|c) u_x}$$

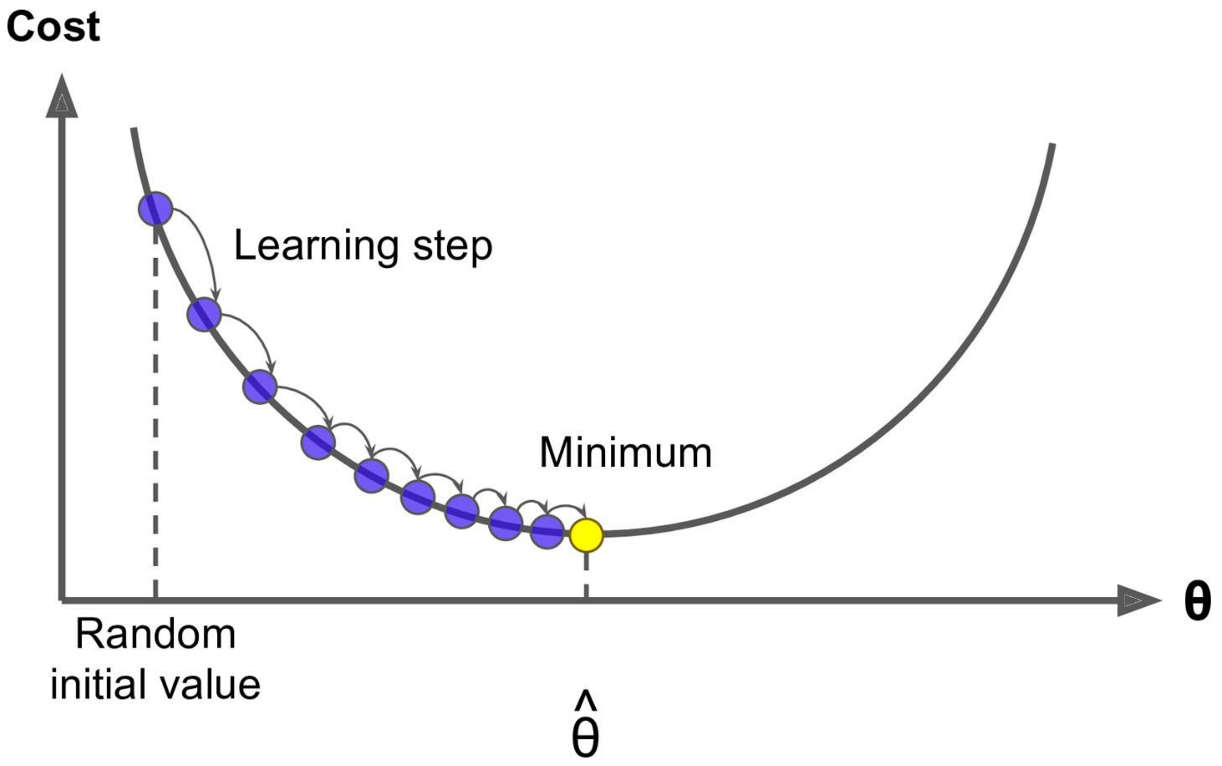
This an expectation: average over all context vectors weighted by their probability

= observed − expected

This is just the derivatives for the center vector parameters
Also need derivatives for output vector parameters (they're similar)
Then we have derivative w.r.t. all parameters and can minimize

5. Optimization: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$
- **Idea:** for current value of θ, calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat.



Note: Our objectives may not be convex like this ☹

But life turns out to be okay ☺

Gradient Descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

α = *step size* or *learning rate*

- Update equation (for single parameter):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- Algorithm:

```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

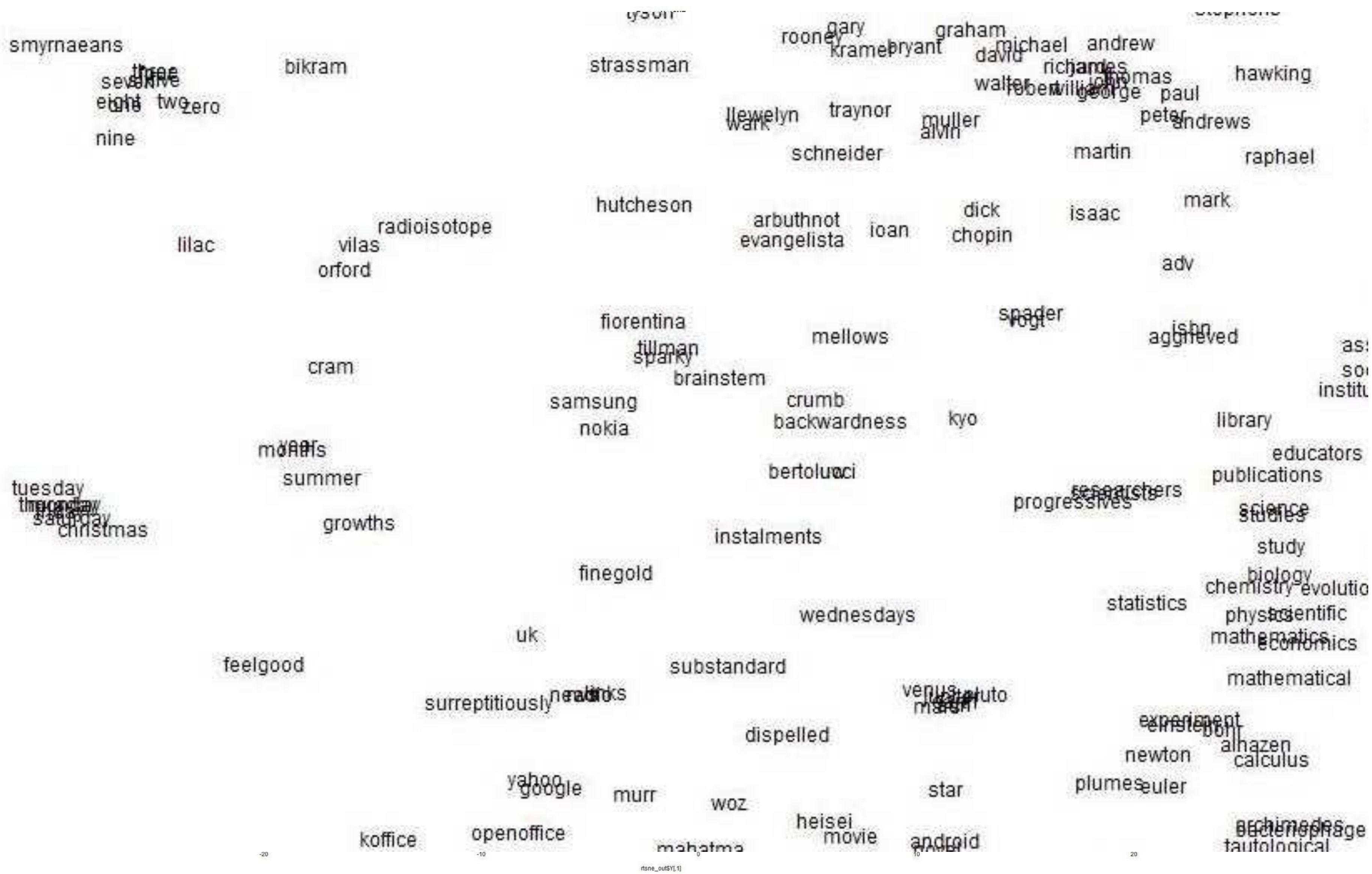
Stochastic Gradient Descent

- **Problem**: $J(\theta)$ is a function of **all** windows in the corpus (potentially billions!)
  - So $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!
- **Very** bad idea for pretty much all neural nets!

- **Solution: Stochastic gradient descent (SGD)**
  - Repeatedly sample windows, and update after each one
- Algorithm:

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

Word2vec maximizes objective function bc putting similar words nearbc in space



How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs. extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy → Winning!

Meaning similaritc: Another intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353 http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Classification review and notation

- Supervised learning: we have a training dataset consisting of samples

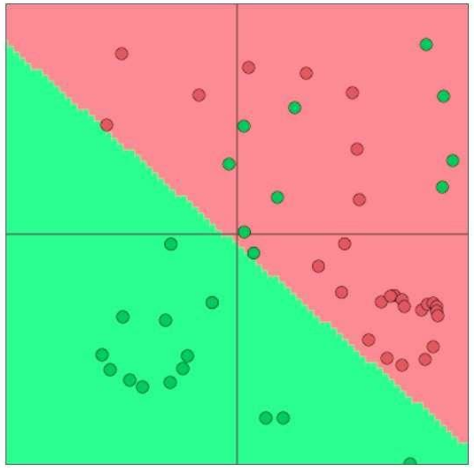
$$\{x_i, y_i\}^N_{i=1}$$

- x_i are inputs, e.g., words (indices or vectors!), sentences, documents, etc.
  - Dimension d

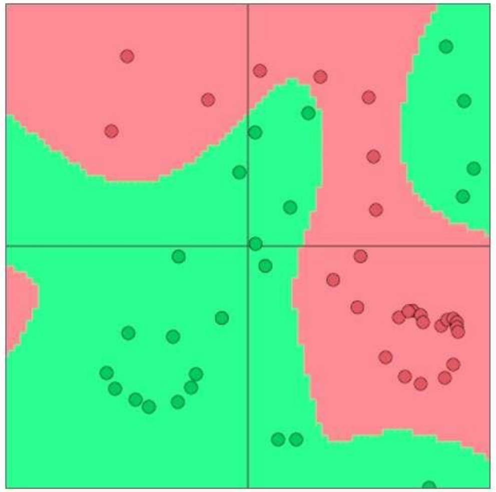
- y_i are labels (one of C classes) we try to predict, for example:
  - classes: sentiment (+/–), named entities, buy/sell decision
  - other words
  - later: multi-word sequences

Neural classification

- Typical ML/stats softmax classifier: $$p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$
- Learned parameters θ are just elements of *W* (not input representation *x*, which has sparse symbolic features)
- Classifier gives linear decision boundary, which can be limiting



- A **neural network classifier** differs in that:
  - We learn **both *W* and (distributed!)** representations for words
  - The word vectors *x* re-represent one-hot vectors, moving them around in an intermediate layer vector space, for easy classification with a (linear) softmax classifier
    - Conceptually, we have an embedding layer: $x = Le$
  - We use deep networks—more layers—that let us re-represent and compose our data multiple times, giving a non-linear classifier



But typically, it is linear relative to the pre-final layer representation

Softmax classifier

$$p(y|x) = \frac{\exp(W_{y}.x)}{\sum_{c=1}^{C} \exp(W_{c}.x)}$$

Again, we can tease apart the prediction function into three steps:

1. For each row *y* of *W*, calculate dot product with *x*:
$$W_{y}.x = \sum_{i=1}^{d} W_{yi} x_i = f_y$$
2. Apply softmax function to get normalized probability:
$$p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)} = \text{softmax}(f_y)$$
3. Choose the *y* with maximum probability

- For each training example (*x*,*y*), our objective is to maximize the probability of the correct class *y* or we can minimize the negative log probability of that class:

$$-\log p(y|x) = -\log\left(\frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)}\right)$$

Thanks.