CS678 Advanced Natural Language Processing

Feed-Forward Neural Networks & Embedding

Ziyu Yao

https://nlp.cs.gmu.edu/course/cs678-fall22

Acknowledgement: Slides are adapted from Greg Durrett CS388@UT Austin and Antonios Anastasopoulos CS695@GMU

About Course Project

- It is allowed to form teams with students from the other section
- Can choose topics from any section: https://nlp.cs.gmu.edu/course/cs678-fall22/projectideas/
  - But priority is given to teams from the same section (or with more students from the same section, if you form teams across sections)
- Note that the two sections share the same Piazza space

(HW1) Vocabulary

- Word type: distinct words in a corpus
- Word token: running occurrence of a word type
- Vocabulary: a set of word types (i.e., *distinct* word tokens)
  - Vocabulary size: the number of word types
- Word (type) frequency: the number of occurrences (tokens) of that type in a corpus

Construct Vocabulary

A very large number of published documents contain text only. They often look boring, and they are often written in obscure language, using mile-long sentences and cryptic technical terms, using one font only, perhaps even without headings. Such style, or lack of style, might be the one you are strongly expected to follow when writing eg scientific or technical reports, legal documents, or administrative papers. It is natural to think that such documents would benefit from a few illustrative images. (However, just adding illustration might be rather useless, if the text remains obscure and unstructured.)

Construct Vocabulary

a very large number of published documents contain text only . they often look boring , and they are often written in obscure language , using mile-long sentences and cryptic technical terms , using one font only , perhaps even without headings . such style, or lack of style, might be the one you are strongly expected to follow when writing eg scientific or technical reports , legal documents , or administrative papers . it is natural to think that such documents would benefit from a few illustrative images . (however , just adding illustration might be rather useless , if the text remains obscure and unstructured .)

truecase + tokenize

Construct Vocabulary

a very large number of published documents contain text only . they often look boring , and they are often written in obscure language , using mile-long sentences and cryptic technical terms , using one font only , perhaps even without headings . such style, or lack of style, might be the one you are strongly expected to follow when writing eg scientific or technical reports , legal documents , or administrative papers . it is natural to think that such documents would benefit from a few illustrative images . (however , just adding illustration might be rather useless , if the text remains obscure and unstructured .)

count the word type, and sort by frequency

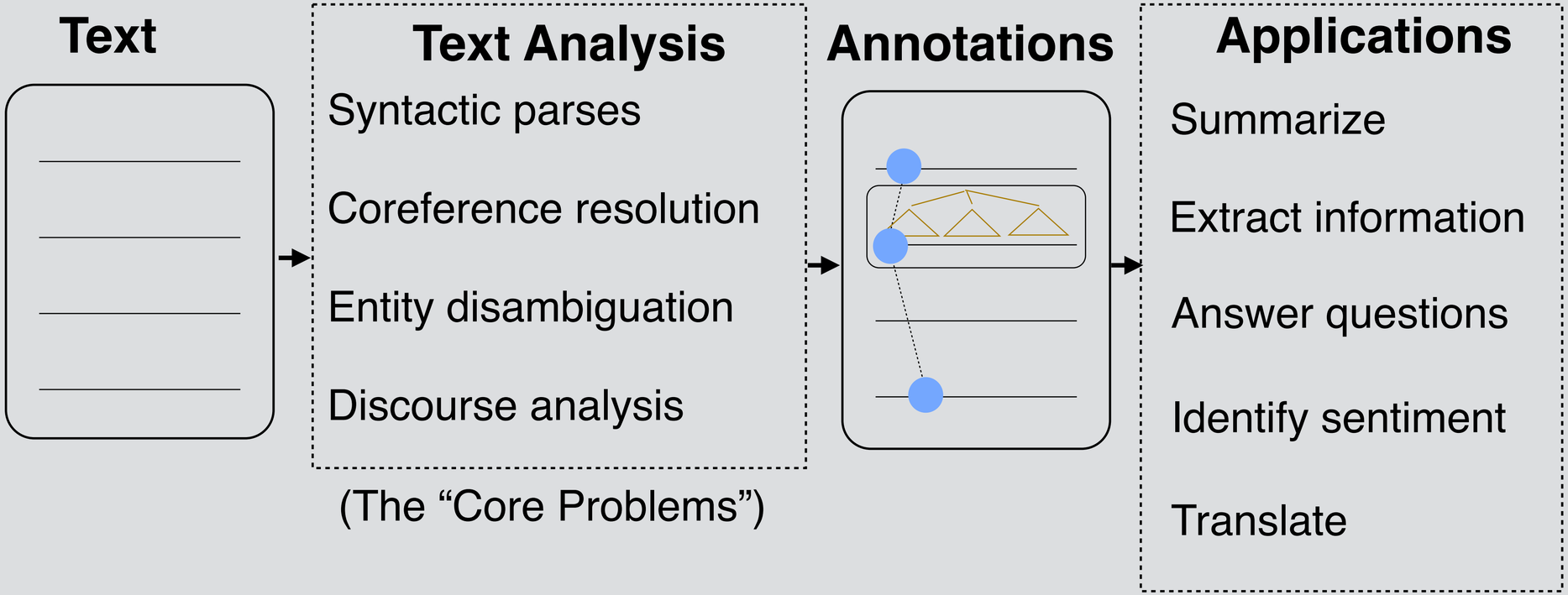
vocabulary (typically after merging rare words *if w/o transfer learning)

,	10	
.	5	
documents		3
and	3	
or	3	
a	2	
of	2	
text	2	
only	2	
…		

Outline

- Refresher: Classification
- Feed-Forward Neural Nets
- Word Embedding
  - Vector Semantics
  - Evaluation of Word Embedding

Recall: NLP Analysis Pipeline



NLP is about building these pieces, which are learned with machine learning approaches nowadays!

Examples: Classification

- *Sentiment Analysis*: What sentiment does the sentence imply?
- *Text Classification*: Is the article about sports or music?
- *Coreference Resolution*: Does the word “She” refers to “Mary” or “Kathy” in this sentence?
- *Text Entailment*: Does sentence A entail or contradict sentence B?

AllenNLP demo: https://demo.allennlp.org/reading-comprehension/bidaf-elmo

Examples: Structured Prediction

- *Part-of-Speech (POS) Tagging*: Predict a sequence of POS tags indicating the token attributes (e.g., noun, verb, etc.)
- *Named Entity Recognition*: Predict a sequence of named entity labels (e.g., ORG, LOC) for qualified words or phrases.
- *Dependency Parsing*: Predict how words are associated syntactically (e.g., noun A is the subject of verb B)
- *Semantic Parsing*: Predict a sequence of grammatically correct symbols (“logical form”) with the same semantic

Stanford CoreNLP demo: https://corenlp.run/

Examples: Some High-level Tasks

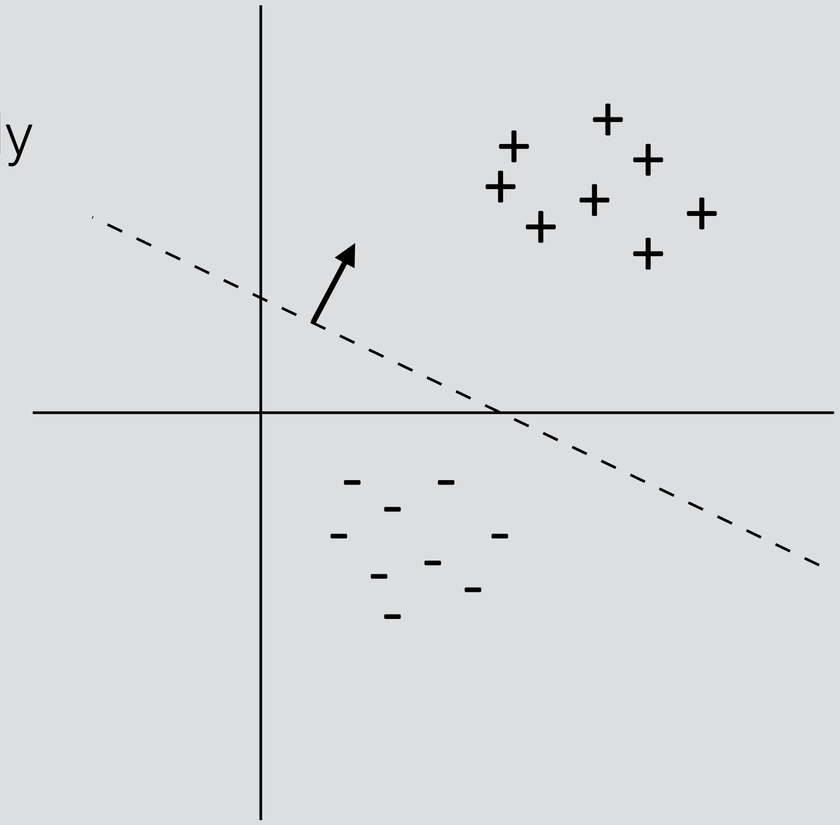
- *Machine Reading Comprehension*: Given a text passage and a question, *predict* or generate a text span as the answer
  - Multi-class classification: predicting the start/end token
- *Machine Translation*: Generate a semantically equivalent target-language sentence based on the source-language sentence
- *Summarization*: Generate a summary for the given text passage
- *Dialogue Generation*: Generate a response to an utterance

Machine Learning Concepts

- Training/development/test set
- Supervised, semi-supervised, unsupervised learning
  - Loss function or objective function, optimization
- “Feature”
- Model parameters vs. hyper-parameters
- Evaluation: held-out test set, task-specific metrics

Binary Classification

- Datapoint x with label $y \in \{0,1\}$
- Embed datapoint in a feature space $f(x) \in \mathbb{R}^n$
  - For simplicity, we will use x and $f(x)$ interchangeably
- *Linear* binary classification: learning a linear decision rule $w^\top f(x) > 0$
  - The bias term b is not needed if we have a lot of features
  - Goal: learning the weight w from data



Example: Sentiment Analysis

this movie was great! would watch again — Positive

that film was awful, I'll never watch again — Negative

- Surface cues can basically tell you what's going on here: presence or absence of certain words (*great*, *awful*) or phrases (*watch again*)
- Steps to classification:
  - Turn examples like this into *feature vectors*
  - Pick a model / learning algorithm
  - Train *weights* on data to get our classifier

Feature Representation

this movie was great! would watch again Positive

- Convert this example to a *vector* using *bag-of-words (BOW)* features

	[contains *the*]	[contains *a*]	[contains *was*]	[contains *movie*]	[contains *film*]	...
	position 0	position 1	position 2	position 3	position 4	
$f(x)$ = [	0	0	1	1	0	...

- Indexing the features
  - Predefined; Must be consistent across all examples (in all of the training, dev, and test sets)!

Feature Representation

- What's wrong?
  - Very large vector space (size of vocabulary + other features)
  - Sparsity (how many per example?)
- Today's NLP: dense features learned from neural networks

$\mathrm{w} = (w_1, \ldots, w_n)^\top, \mathrm{x} = (x_1, \ldots, x_n)^\top$

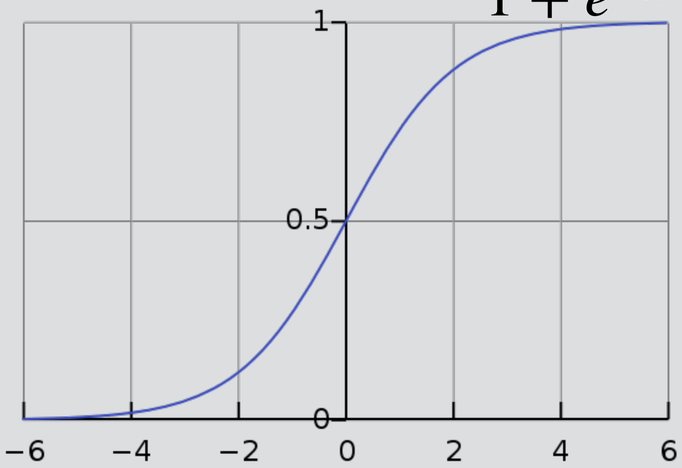
vector scalar

Logistic Regression

$$P(y = + \,|\, \mathrm{x}) = logistic(\mathrm{w}^\top \mathrm{x})$$

$$= \frac{1}{1 + \exp(-\sum_{i=1}^{n} w_i x_i)} = \frac{\exp(\sum_{i=1}^{n} w_i x_i)}{1 + \exp(\sum_{i=1}^{n} w_i x_i)}$$

$$logistic(x) = \frac{1}{1 + e^{-x}}$$



To learn weights: *maximize* log likelihood of data

Assume training data $D_{train} = \{(\mathrm{x}_j, y_j)\}_{j=1}^{|D_{train}|}$

$$L(D_{train}) = \sum_{j=1}^{|D_{train}|} \log P(y_j \,|\, \mathrm{x}_j)$$ corpus-level log-likelihood (LL)

$$L(\mathrm{x}_j, y_j = +) = \log P(y_j = + \,|\, \mathrm{x}_j) = \sum_{i=1}^{n} w_i \mathrm{x}_{ji} - \log(1 + \exp(\sum_{i=1}^{n} w_i \mathrm{x}_{ji}))$$

one (positive) example LL

Logistic Regression: Training

Optimization with gradient *ascent*

$$L(\mathrm{x}_j, y_j = +) = \log P(y_j = + \mid \mathrm{x}_j) = \sum_{i=1}^{n} w_i \mathrm{x}_{ji} - \log(1 + \exp(\sum_{i=1}^{n} w_i \mathrm{x}_{ji}))$$

$$\begin{aligned}
\frac{\partial L(\mathrm{x}_j, y_j = +)}{\partial w_i} &= x_{ji} - \frac{\partial}{\partial w_i} \log(1 + \exp(\sum_{i=1}^{n} w_i x_{ji})) \\
&= x_{ji} - \frac{1}{1 + \exp(\sum_{i=1}^{n} w_i x_{ji})} \frac{\partial}{\partial w_i} (1 + \exp(\sum_{i=1}^{n} w_i x_{ji})) \\
&= x_{ji} - \frac{1}{1 + \exp(\sum_{i=1}^{n} w_i x_{ji})} \, x_{ji} \, \exp(\sum_{i=1}^{n} w_i x_{ji}) \\
&= x_{ji} \, (1 - P(y_j = + \mid \mathrm{x}_j))
\end{aligned}$$

Logistic Regression: Training

Optimization with gradient *ascent*

$$L(\mathrm{x}_j, y_j = +) = \log P(y_j = + \,|\, \mathrm{x}_j) = \sum_{i=1}^{n} w_i \mathrm{x}_{ji} - \log(1 + \exp(\sum_{i=1}^{n} w_i \mathrm{x}_{ji}))$$

$$\frac{\partial L(\mathrm{x}_j, y_j = +)}{\partial w_i} = x_{ji}\,(1 - P(y_j = + \,|\, \mathrm{x}_j))$$ scalar

$$\frac{\partial L(\mathrm{x}_j, y_j = +)}{\partial \mathrm{w}} = \mathrm{x}_j\,(1 - P(y_j = + \,|\, \mathrm{x}_j))$$ vector

i.e., gradient of w on $(\mathrm{x}, y = +)$ is $\mathrm{x}\,(1 - P(y = + \,|\, \mathrm{x}))$

Logistic Regression: Training

- Gradient of w on positive example $= \mathrm{x}\,(1 - P(y = + \,|\, \mathrm{x}))$
  - If P(+ | x) is close to 1, make very little update
  - Otherwise make w look more like x, which will increase P(+ | x)
- Gradient of w on negative example $= \mathrm{x}\,(-P(y = + \,|\, \mathrm{x}))$
  - If P(+ | x) is close to 0, make very little update
  - Otherwise make w look less like x, which will decrease P(+ | x)
- Let y = 1 for positive instances, y = 0 for negative instances.
- Can combine these gradients as $= \mathrm{x}\,(y - P(y = 1 \,|\, \mathrm{x}))$

Example

(1) *this movie was great! would watch again* +

(2) *I expected a great movie and left happy* +

(3) *great potential but ended up being a flop* —

$f(x_1) = [1 \quad 1]$

$f(x_2) = [1 \quad 1]$

$f(x_3) = [1 \quad 0]$

[contains *great*] [contains *movie*]

position 0 position 1

$w = [0, 0] \longrightarrow P(y = 1 \mid x_1) = \exp(0)/(1 + \exp(0)) = 0.5 \longrightarrow g = [0.5, 0.5]$

$w = [0.5, 0.5] \longrightarrow P(y = 1 \mid x_2) = \text{logistic}(1) \approx 0.75 \longrightarrow g = [0.25, 0.25]$

$w = [0.75, 0.75] \rightarrow P(y = 1 \mid x_3) = \text{logistic}(0.75) \approx 0.67 \longrightarrow g = [-0.67, 0]$

$w = [0.08, 0.75]$ …

$$P(y = 1 \mid \mathrm{x}) = \exp(\mathrm{w}^\top \mathrm{x})/(1 + \exp(\mathrm{w}^\top \mathrm{x}))$$

Gradient: $\mathrm{x}\,(y - P(y = 1 \mid \mathrm{x}))$

Regularization

- Regularizing an objective can mean many things, including an L2-norm penalty to the weights:

$$\sum_j L(\mathrm{x}_j, y_j) - \lambda \|\mathrm{w}\|_2^2$$

- Keeping weights small can prevent overfitting

- For most of the NLP models we build, explicit regularization isn't necessary
  - We always stop early before full convergence
  - For neural networks: dropout and gradient clipping
  - Large numbers of sparse features are hard to overfit in a really bad way

Logistic Regression: Summary

- Model:

$$P(y = 1 \mid \mathrm{x}) = \frac{\exp(\mathrm{w}^\top \mathrm{x})}{(1 + \exp(\mathrm{w}^\top \mathrm{x}))}$$

- Inference: $\hat{y} = \arg\max_y P(y \mid \mathrm{x})$

  $P(y = 1 \mid \mathrm{x}) \geq 0.5 \Leftrightarrow \mathrm{w}^\top \mathrm{x} \geq 0$

- Learning: gradient ascent on the (regularized) log-likelihood
  - True positive example: $\mathrm{w} \leftarrow \mathrm{w} + \mathrm{x}(1 - P(y = 1 \mid \mathrm{x}))$
  - True negative example: $\mathrm{w} \leftarrow \mathrm{w} - \mathrm{x}P(y = 1 \mid \mathrm{x})$

Optimization

- Batch optimization: gradient over the entire dataset
  - e.g., gradient descent

$$\mathrm{w} \leftarrow \mathrm{w} - \alpha \mathrm{g}, \quad \mathrm{g} = \frac{\partial}{\partial \mathrm{w}} L$$

  - α is the learning rate
  - Inefficient when computing over the entire training set
- Online optimization: gradient over a randomly sampled example
  - e.g., stochastic gradient descent
- More commonly used now: *mini-batch* optimization

Outline

- Refresher: Classification
- Feed-Forward Neural Nets
- Word Embedding
  - Vector Semantics
  - Evaluation of Word Embedding

terials

Deep Learning

https://www.deeplearningbook.org/

An MIT Press book

Ian Goodfellow and Yoshua Bengio and Aaron Courville

http://neuralnetworksanddeeplearning.com/

Neural Networks and Deep Learning

Neural Networks and Deep Learning is a free online book. The book will teach you about:

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data
- Deep learning, a powerful set of techniques for learning in neural networks

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. This book will teach you many of the core concepts behind neural networks and deep learning.

Neural Networks

- Linear classification: $argmax_y \mathrm{w}^\top f(\mathrm{x}, y)$

- Want to learn intermediate conjunctive features of the input

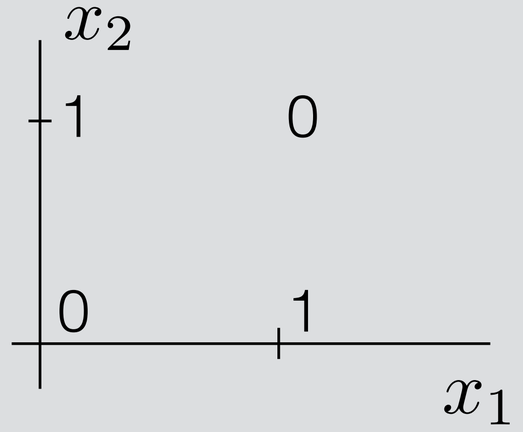
*the movie was **not** all that **good***
I[contains *not* & contains *good*]

- How do we learn this if our feature vector is just the unigram indicators?

I[contains *not*], I[contains *good*]

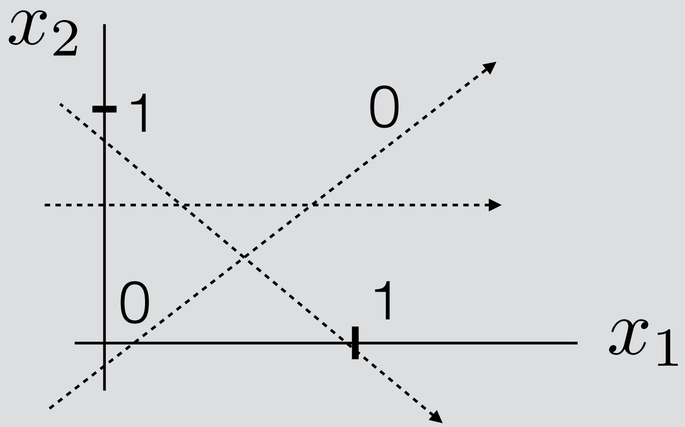
Neural Networks: XOR

- Let's see how we can use neural nets to learn a simple nonlinear function
- Inputs $x_1,\ x_2$
- Outputs y



x_1	x_2	$y = x_1$ XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Neural Networks: XOR

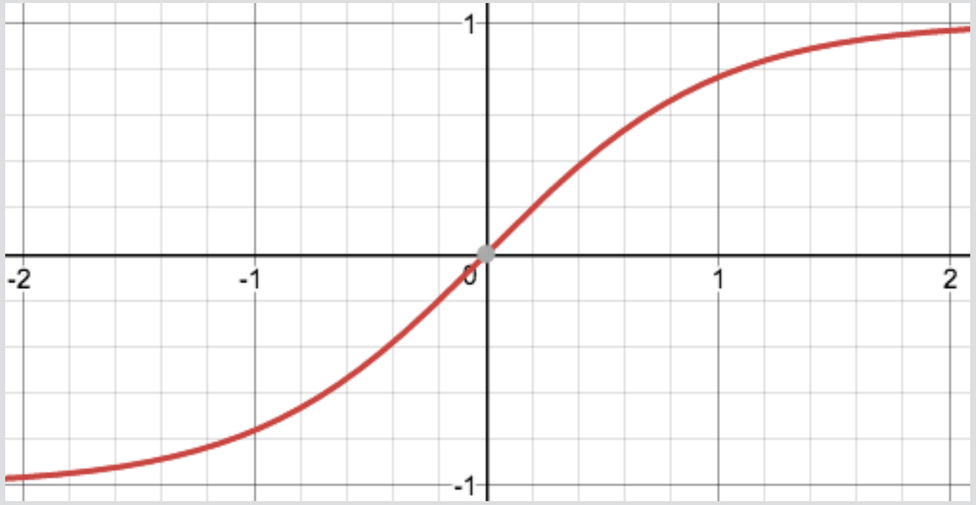


$$y = a_1 x_1 + a_2 x_2$$ X

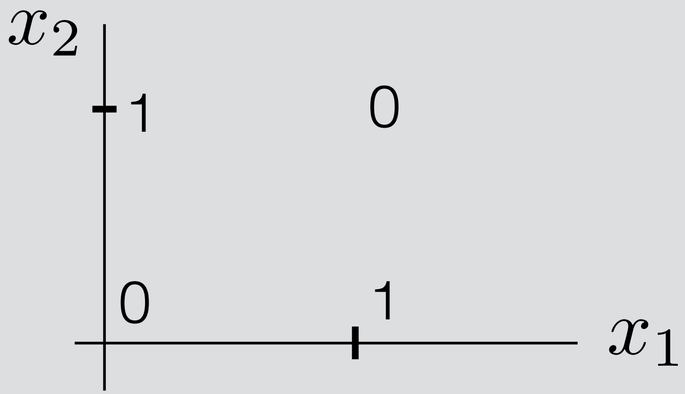
$$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$$ ✓

"or"

x_1	x_2	$y = x_1$ XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0



Neural Networks: XOR

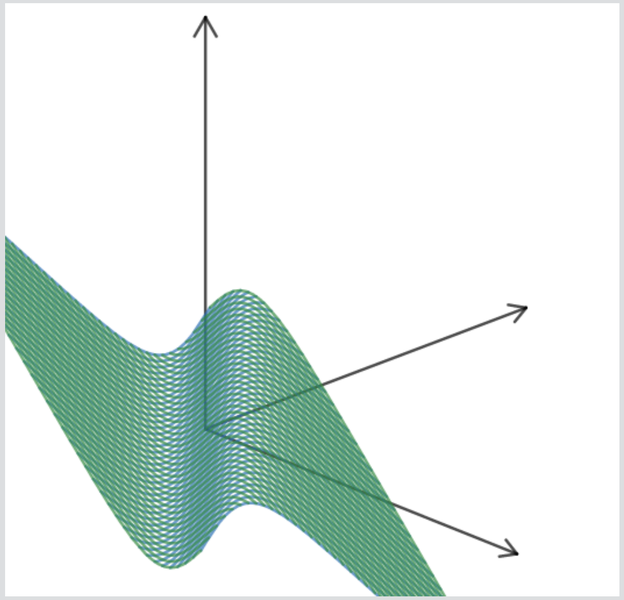


$$y = a_1 x_1 + a_2 x_2$$ X

$$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$$ ✓

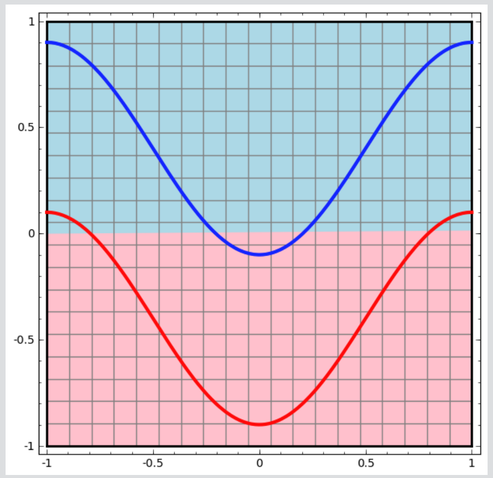
$$y = -x_1 - x_2 + 2\tanh(x_1 + x_2)$$

x_1	x_2	$y = x_1$ XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

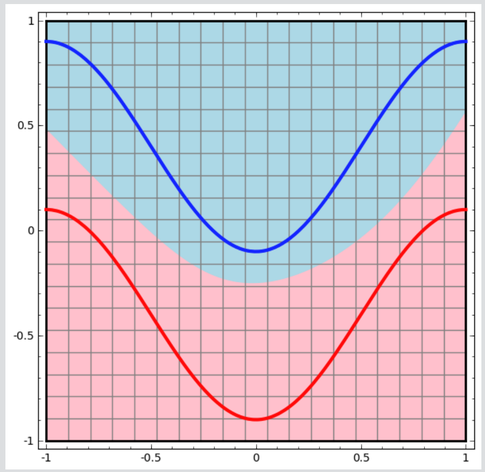


Neural Networks

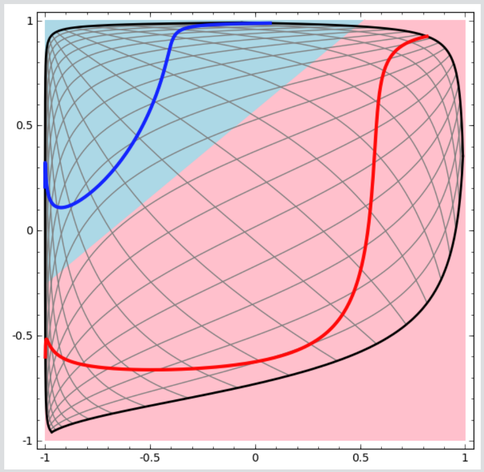
Linear classifier



Neural network



…possible because we transformed the space!



Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Deep Neural Networks

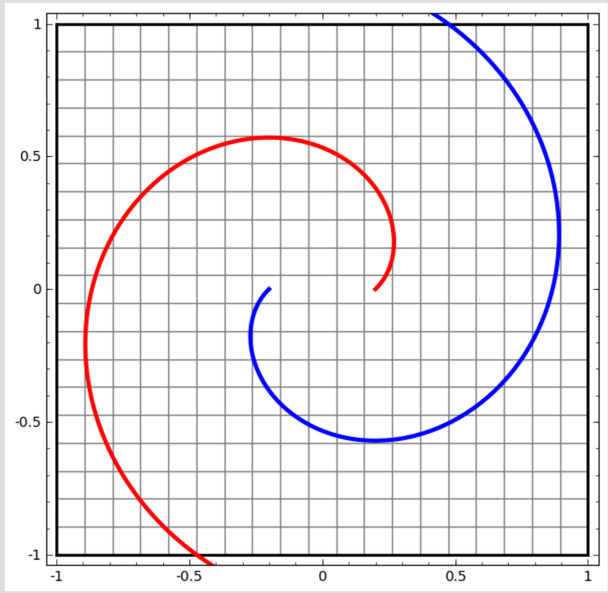
$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\mathbf{z} = g(\mathbf{V}\mathbf{y} + \mathbf{c})$$

$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$

Check: what happens if no nonlinearity?
More powerful than basic linear models?

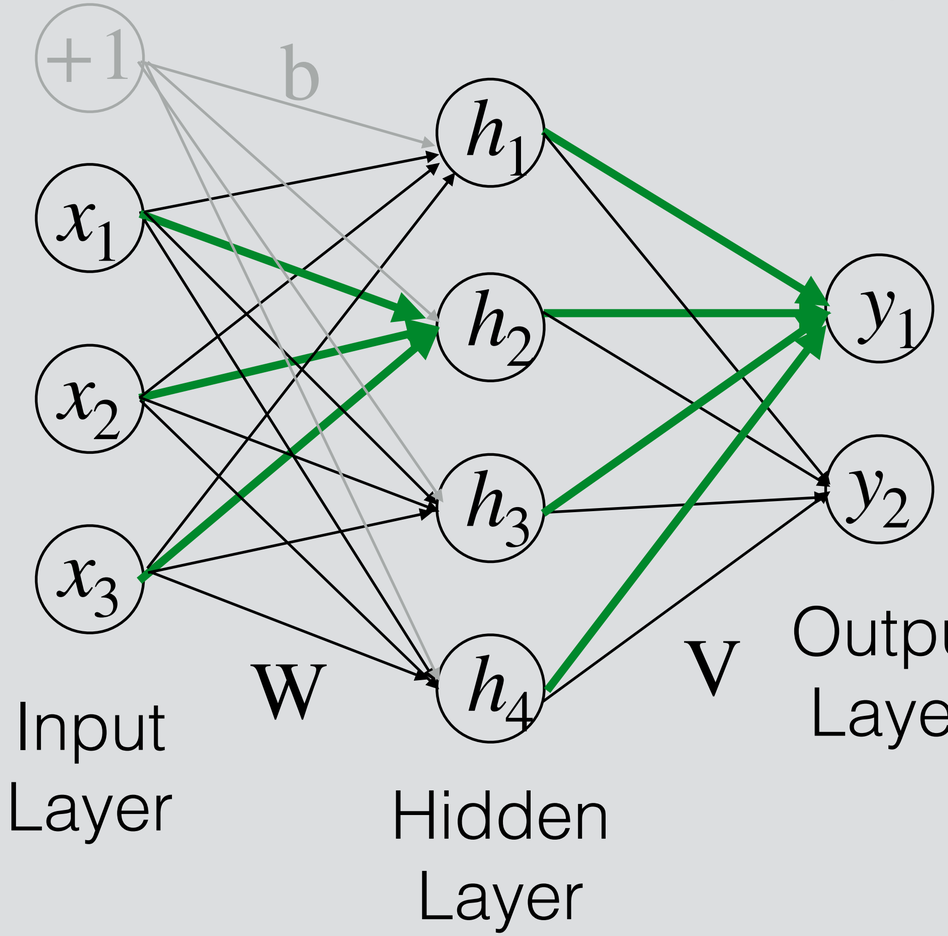
$$\mathbf{z} = \mathbf{V}(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{c}$$



Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Feed-Forward Neural Nets

- Sometimes also called “multi-layer perceptrons” or MLPs



Each hidden unit h takes a weighted sum of its inputs,

$$h_i = \sum_j w_{ij} x_j + b_i$$

(or, $\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$)

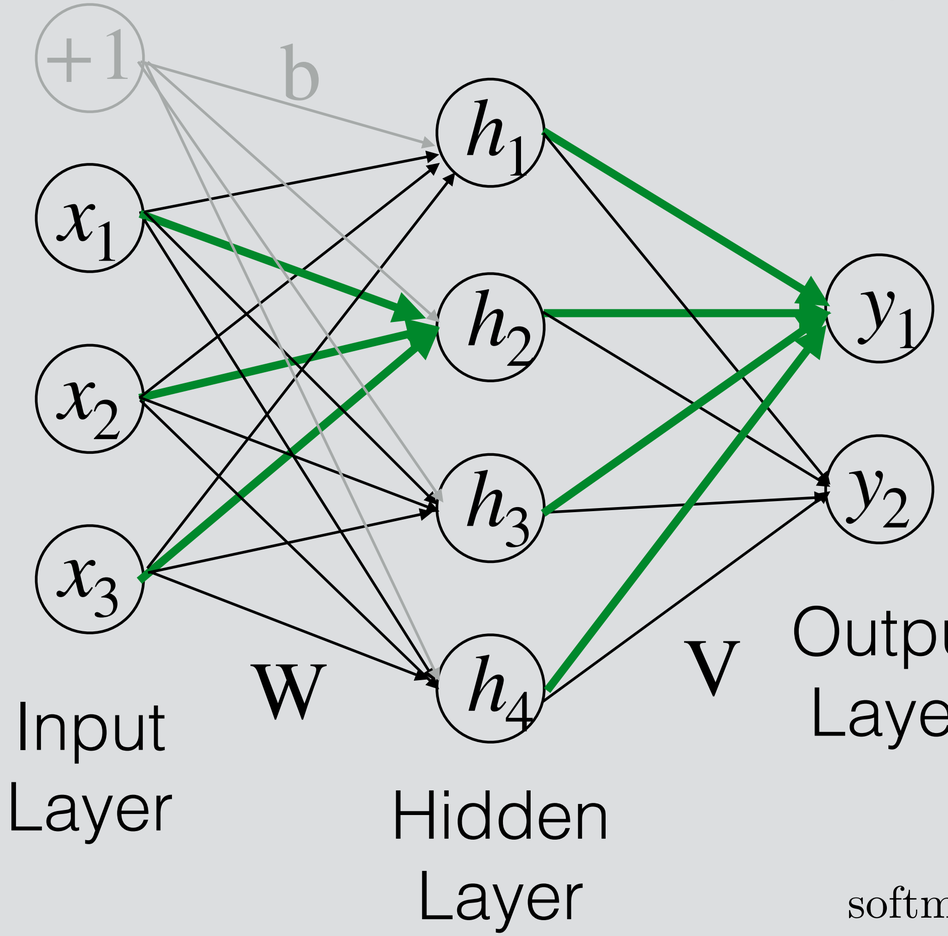
and then apply a non-linear function (the “activation function”):

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

(e.g., tanh, sigmoid, ReLU)

Feed-Forward Neural Nets

- Sometimes also called “multi-layer perceptrons” or MLPs



Each hidden unit h takes a weighted sum of its inputs, and then apply “activation function”:

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

The output (e.g., for classification):

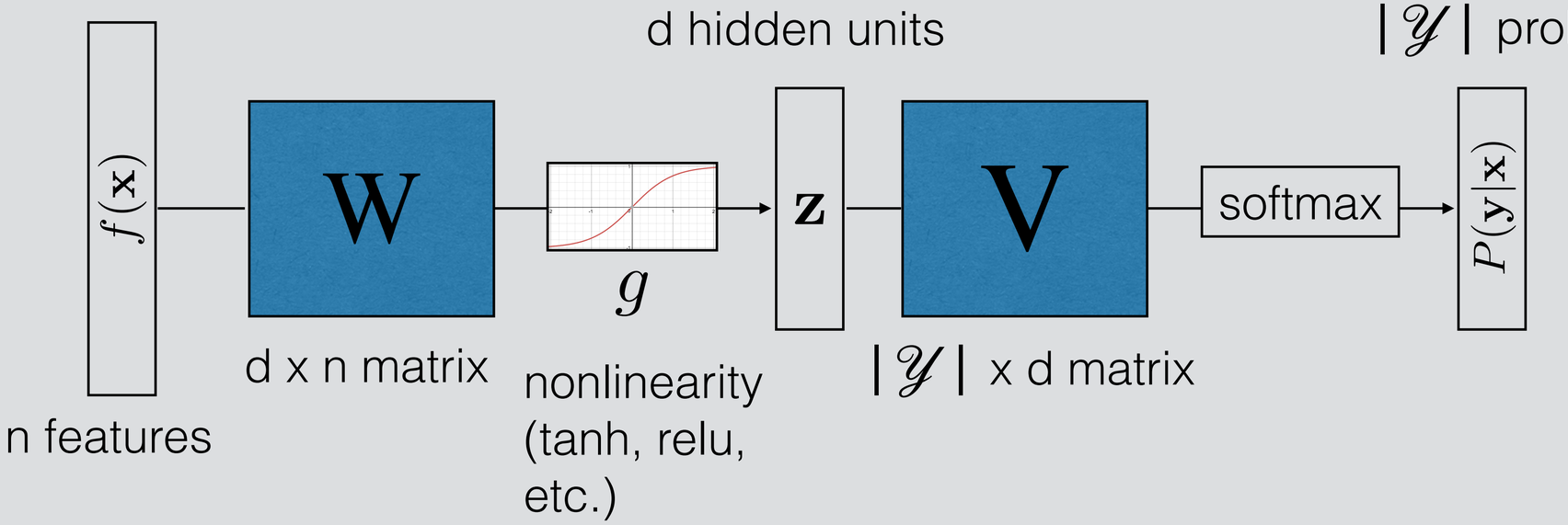
$$\mathbf{y} = softmax(\mathbf{V}\mathbf{h})$$

$$= softmax\Big(\mathbf{V}\big(g(\mathbf{W}\mathbf{x} + \mathbf{b})\big)\Big)$$

$$\text{softmax}(p)_i = \frac{\exp(p_i)}{\sum_{i'} \exp(p_{i'})}$$

Neural Networks for Classification

$$P(\mathrm{y} \mid \mathrm{x}) = softmax\Big(\mathrm{V}\ g(\mathrm{W}\ f(\mathrm{x}))\Big)$$



$|\mathcal{Y}|$: number of classes

Training Neural Networks

$$P(\mathrm{y} \mid \mathrm{x}) = softmax\big(\mathrm{V}\mathrm{z}\big) \qquad \mathrm{z} = g(\mathrm{W}f(\mathrm{x}))$$

- Maximize log likelihood of training data

$$L(\mathrm{x}, y^*) = \log P(y^* \mid \mathrm{x}) = \log\big(softmax(\mathrm{V}\mathrm{z})_{i^*}\big)$$

$$= (\mathrm{V}\mathrm{z})_{i^*} - \log \sum_{i'} \exp\big((\mathrm{V}\mathrm{z})_{i'}\big)$$

i^*: the index of y^* in the output space $\mathcal{Y}$

$$\mathrm{softmax}(p)_i = \frac{\exp(p_i)}{\sum_{i'} \exp(p_{i'})}$$

Compute Gradients

$$L(\mathrm{x}, y^*) = = (\mathrm{Vz})_{i^*} - \log \sum_{i'} \exp\big((\mathrm{Vz})_{i'}\big)$$

- Gradient with respect to $\mathbf{V}_{ij}$

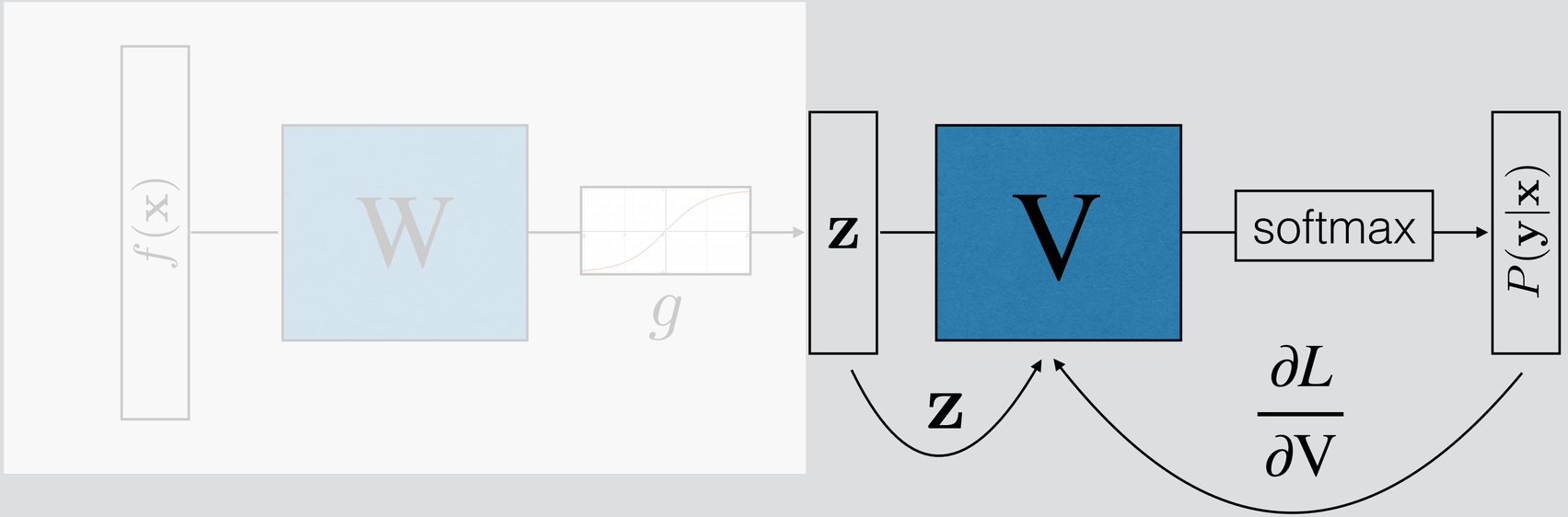
$$\frac{\partial}{\partial \mathrm{V}_{ij}} L(\mathrm{x}, y^*) = \begin{cases} z_j - P(y = i \,|\, \mathrm{x}) z_j & \text{if } i = i^* \\ -P(y = i \,|\, \mathrm{x}) z_j & \text{otherwise} \end{cases}$$

- Looks like logistic regression with $\mathbf{z}$ as the features!

Logistic Regression:
Gradient $= \mathrm{x}\,(y - P(y = 1 \,|\, \mathrm{x}))$

Neural Networks for Classification

$$P(\mathrm{y} \mid \mathrm{x}) = softmax\Big(\mathrm{V}\, g(\mathrm{W}\, f(\mathrm{x}))\Big) \quad \mathrm{z} = g(\mathrm{W} f(\mathrm{x}))$$



Computing Gradients: Backpropagation

$$L(\mathrm{x}, y^*) = = (\mathrm{Vz})_{i^*} - \log \sum_{i'} \exp\left((\mathrm{Vz})_{i'}\right) \qquad \mathrm{z} = g(\mathrm{W}f(\mathrm{x}))$$

- Gradient with respect to $\mathbf{W}_{ij}$: apply the chain rule

$$\frac{\partial L(\mathrm{x}, y^*)}{\partial \mathrm{W}_{ij}} = \frac{\partial L(\mathrm{x}, y^*)}{\partial \mathrm{z}} \frac{\partial \mathrm{z}}{\partial \mathrm{W}_{ij}}$$

https://en.wikipedia.org/wiki/Matrix_calculus

[some math…]

$$err(\mathrm{root}) = e_{i^*} - P(\mathbf{y}|\mathbf{x})$$

dim = num_classes

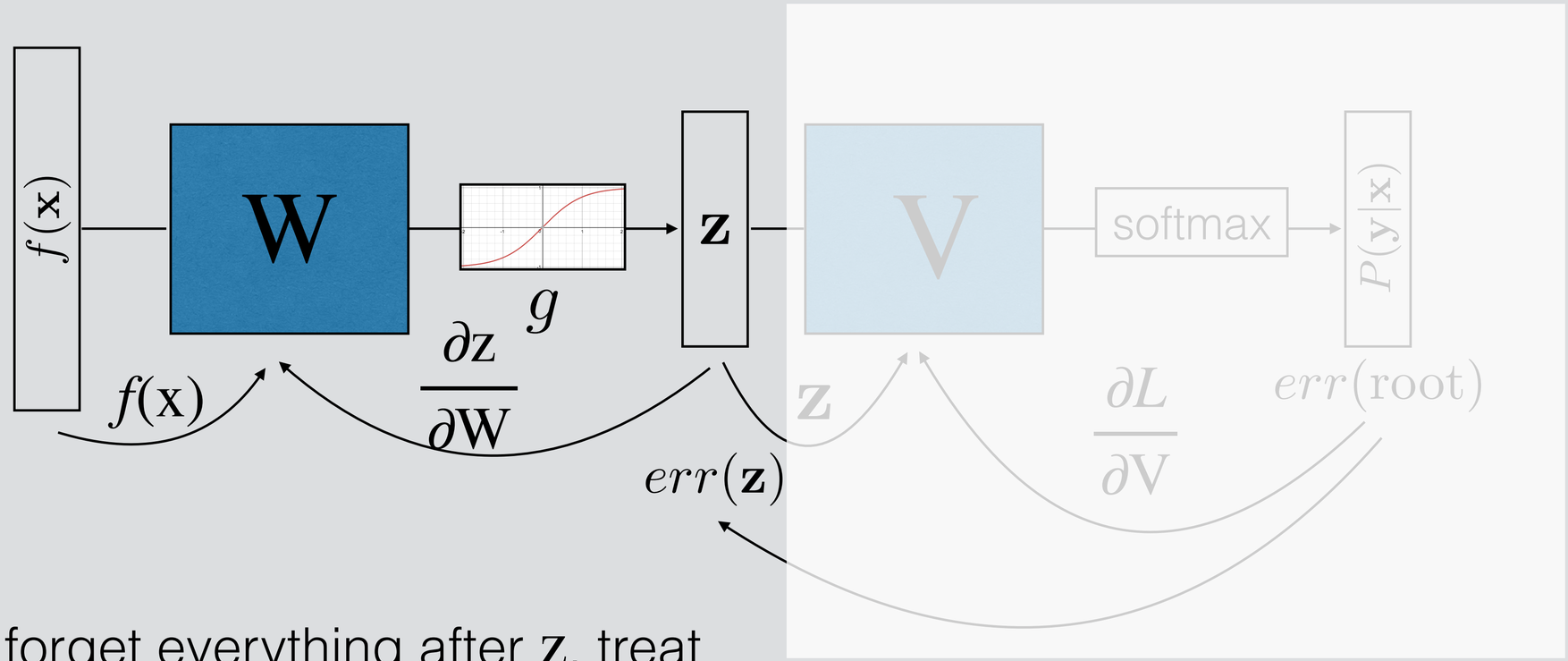
(*err*: the “error signal”)

$$\frac{\partial L(\mathrm{x}, y^*)}{\partial \mathrm{z}} = err(\mathrm{z}) = \mathrm{V}^\top err(\mathrm{root})$$

dim = d

Neural Networks for Classification

$$P(\mathrm{y} \mid \mathrm{x}) = softmax\big(\mathrm{V}\, g(\mathrm{W}\, f(\mathrm{x}))\big) \quad \mathrm{z} = g(\mathrm{W} f(\mathrm{x}))$$



Can forget everything after z, treat it as the output and keep backpropagating

Backpropagation: Takeaways

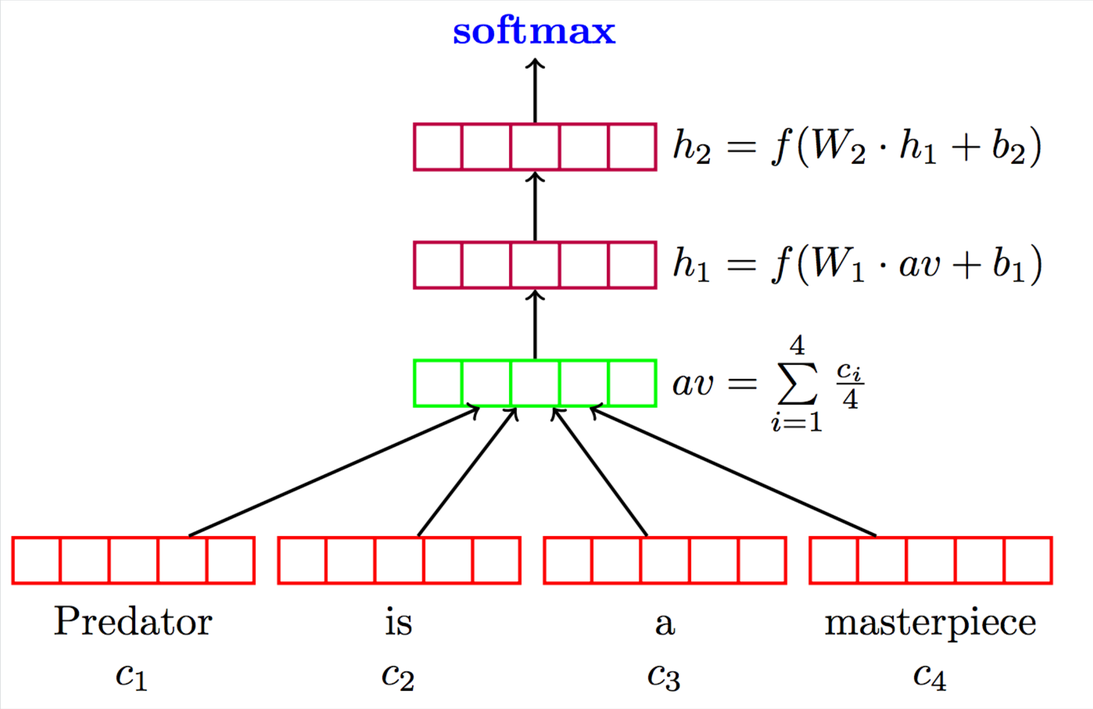
- Gradients of output weights V are easy to compute — looks like logistic regression with hidden layer z as feature vector
- Can compute derivative of loss with respect to z to form an “error signal” for backpropagation
- Easy to update parameters based on “error signal” from next layer, keep pushing error signal back as backpropagation
- Need to remember the values from the *forward* computation

Computation Graphs

- Computing gradients is hard! Computation graph abstraction allows us to define a computation symbolically and will do this for us
- Automatic differentiation: keep track of derivatives / be able to backpropagate through each function
- Use a library like PyTorch or Tensorflow. This class: PyTorch.

HW 1: Sentiment Analysis

Deep Averaging Networks (DAN): feedforward neural network on average of word embeddings from input



Two non-linear layers

f(**x**) = 300-d embedding for each word of **x** (“word embedding”)

Iyyer, Mohit, et al. "Deep unordered composition rivals syntactic methods for text classification." *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*. 2015.

BREAK

Shuffling the Training Data

- Stochastic gradient methods update the parameters a little bit at a time
  - What if we have the sentence “I love this sentence so much!” at the end of the training data 10000 times?
  - Or, what if in the training set all the first half examples are positive and all the second half are negative?
- To train correctly, we should randomly shuffle the order at each time step

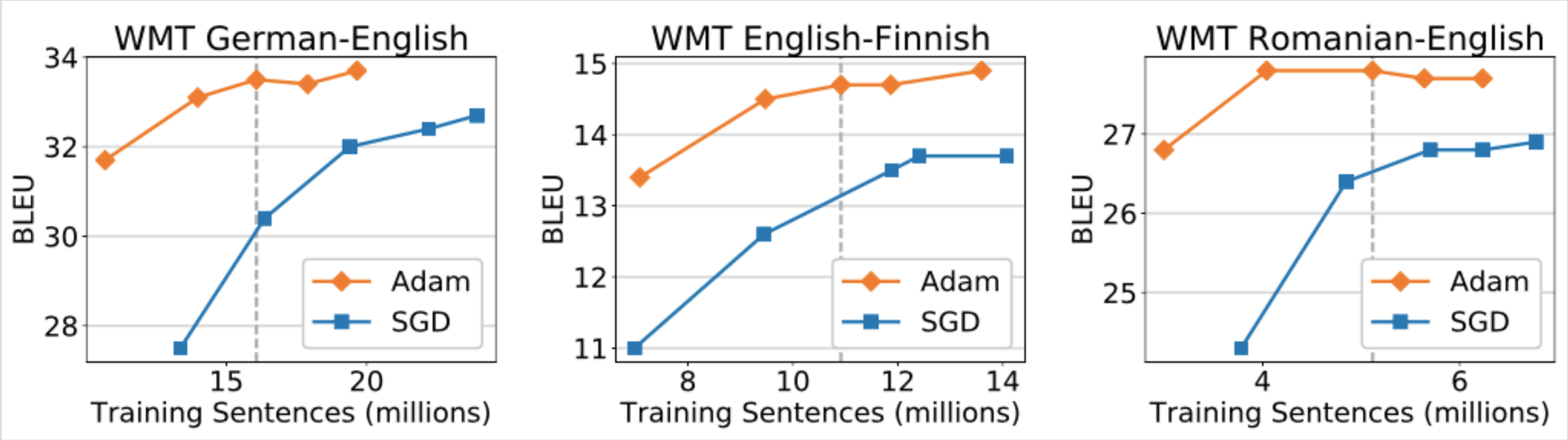
Other Optimizer Options

- **SGD with Momentum:** Remember gradients from past time steps to prevent sudden changes
- **Adagrad:** Adapt the learning rate to reduce learning rate for frequently updated parameters (as measured by the variance of the gradient)
- **Adam:** Like Adagrad, but keeps a running average of momentum and gradient variance
- **Many others:** RMSProp, Adadelta, etc.

A very nice overview by Sebastian Ruder:
https://ruder.io/optimizing-gradient-descent/

Which One to Use?

- Adam is usually fast to converge and stable
- But simple SGD tends to do very well in terms of generalization (Wilson et al. 2017)
- You should use learning rate decay, (e.g. on Machine translation results by Denkowski & Neubig 2017)



Early Stopping

- Neural nets have tons of parameters: we want to prevent them from overfitting
- We can do this by monitoring our performance on held-out development set and stopping training when it starts to get worse
- At the end of training, return the best parameters on the held-out dev set (“best” as measured by a certain metric)

Dropout
(Srivastava+ 14)

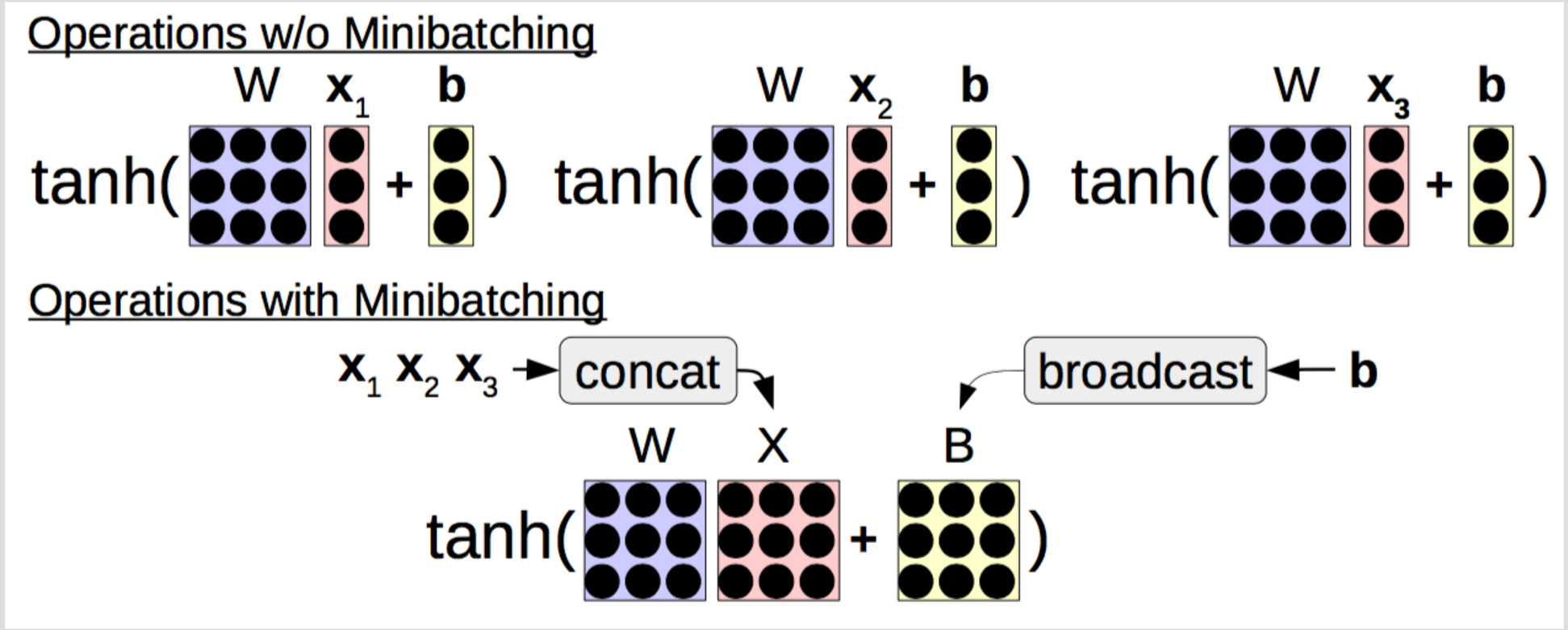
- Neural nets have lots of parameters, and are prone to overfitting
- Dropout: randomly zero-out nodes in the hidden layer with probability *p* at **training time only**

- Because the number of nodes at training/test is different, scaling is necessary:
  - **Standard dropout:** scale by *p* at test time
  - **Inverted dropout:** scale by 1/(1-*p*) at training time
- Easy to implement in PyTorch, **but remember to switch between `model.train()` and `model.eval()`**

Efficiency Tricks: Mini-batching

- On modern hardware 10 operations of size 1 is **much slower than** 1 operation of size 10
- Mini-batching combines together smaller operations into one big one
- In PyTorch (almost) all operations already automatically support batches

Mini-batching



Outline

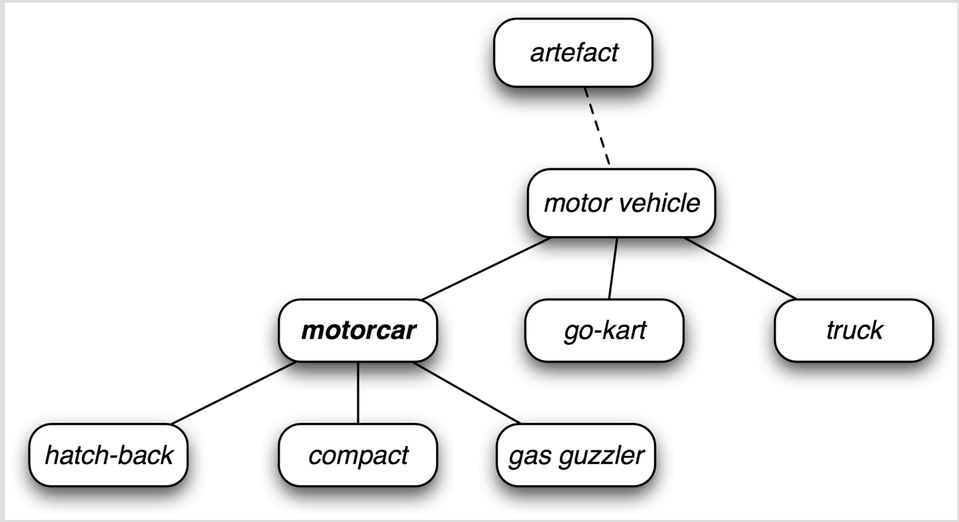
- Refresher: Classification
- Feed-Forward Neural Nets
- Word Embedding
  - Vector Semantics
  - Evaluation of Word Embedding

What do we want to know about words?

- Are they the same part of speech?
- Do they have the same conjugation?
- Do these two words mean the same thing?
- Do they have some semantic relation (is-a, part-of, went-to-school-at)?

A Manual Attempt: WordNet

- WordNet is a large database of words including parts of speech, semantic relations, etc. Covering more than 200 languages.



- But can we do something similar, more complete, and without the effort?

Image Credit: NLTK;
http://wordnetweb.princeton.edu/perl/webwn More details see JM Ch18.3

An Answer (?): Word Embeddings!

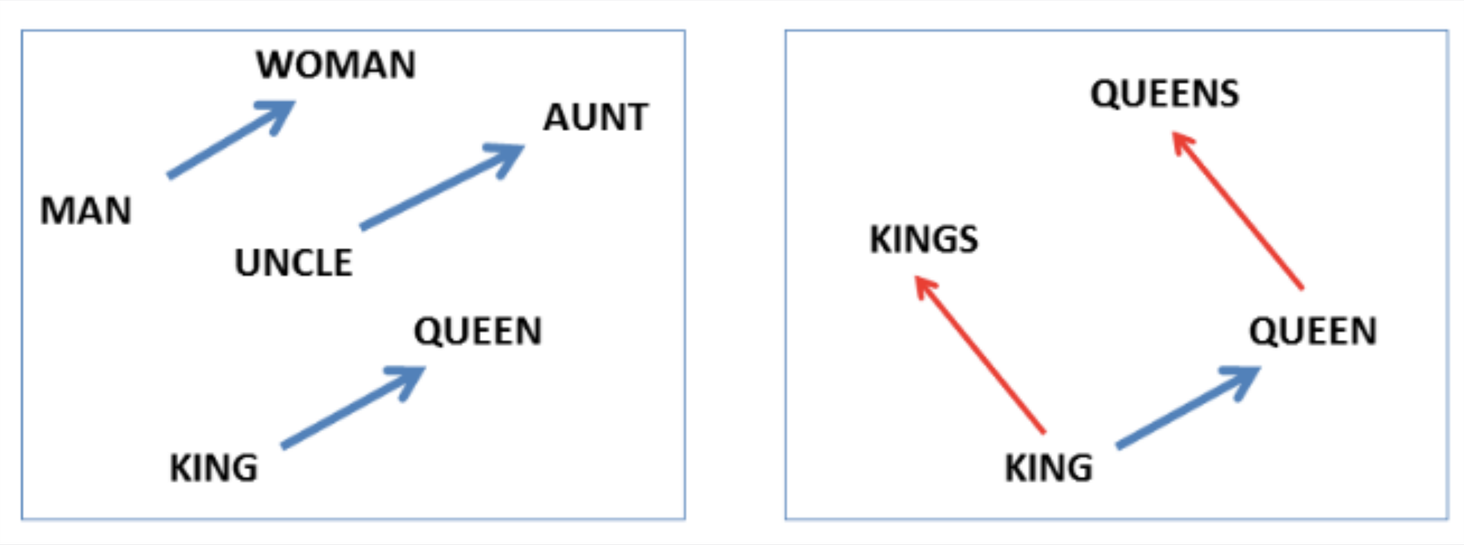
- A continuous vector representation of words

- Within the word embedding, these features of syntax and semantics may be included
  - Element 1 might be **more positive for nouns**
  - Element 2 might be **positive for animate objects**
  - Element 3 might have **no intuitive meaning whatsoever**

Word Embeddings are Cool!

(An Obligatory Slide)

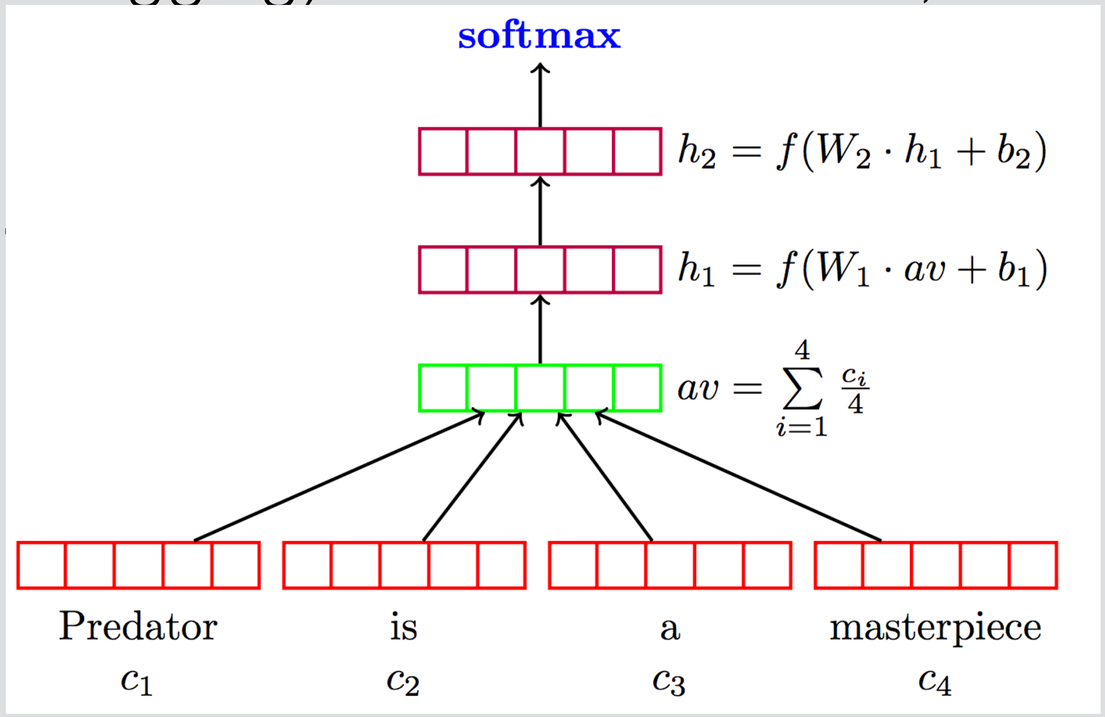
- e.g. king-man+woman = queen (Mikolov et al. 2013)



- “What is the female equivalent of king?” is not easily accessible in many traditional resources

How to Train Word Embeddings?

- **Initialize randomly**, and then *train jointly* with the task
- Pre-train on a **supervised** task (e.g. POS tagging) and test on another, (e.g. parsing)
- Pre-train on an **unsupervised** task (e.g.



(Iyyer et al., 2015)

Unsupervised Pre-training of Word Embeddings

(Summary of Goldberg 10.4)

Distributional vs. *Distributed* Representations

- **Distributional representations**
  - Words are similar if they appear in similar contexts (Harris 1954);
  - In contrast: *non-distributional* representations are created from lexical resources such as WordNet, etc.
- **Distributed representations**
  - Basically, each item is represented by a vector of values, and each value represents some activations
  - In contrast: *local* representations are represented using discrete symbols (e.g., one-hot vector)

Distributional Representations

(see Goldberg 10.4.1)

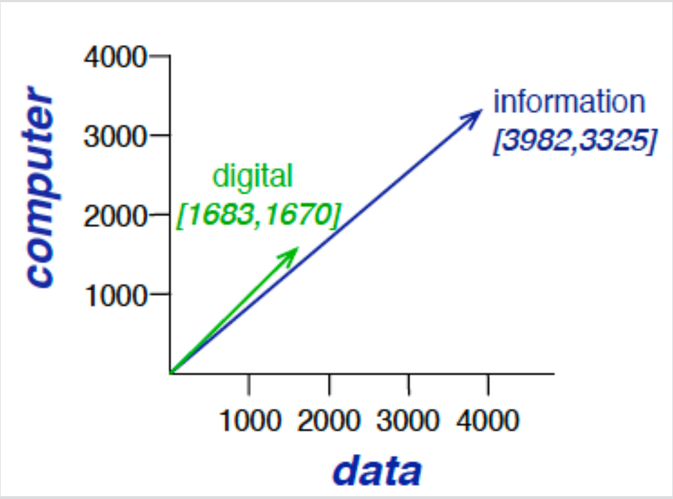
- Words appear in a context

<s>	<s>	<unk>	communications	pittsburgh	acquired	<unk>	&	co.
investment	management	inc.	a	pittsburgh	firm	that	runs	a
<s>	mr.	allen	's	pittsburgh	firm	advanced	investment	management
look	stupid	<unk>	former	pittsburgh	<unk>	second	<unk>	<unk>
through	the	university	of	pittsburgh	law	school	<s>	<s>
with	the	university	of	pittsburgh	<s>	<s>	<s>	<s>
<unk>	he	heads	the	pittsburgh	branch	of	the	committee
at	the	university	of	pittsburgh	earn	up	to	$

for	society	corp.	a	cleveland	bank	said	demand	for
as	washington	<unk>	r.i.	cleveland	<unk>	n.c.	minneapolis	and
<s>	<s>	<unk>	a	cleveland	merchant	bank	owns	about
new	stadiums	ranging	from	cleveland	to	san	antonio	and
<s>	the	philadelphia	and	cleveland	districts	for	example	reported
mcdonald	&	co.	in	cleveland	said	<unk>	's	unanticipated
<unk>	tumor	at	the	cleveland	clinic	in	N	<s>
at	mcdonald	&	co.	cleveland	<s>	<s>	<s>	<s>

Count-based Metho

(JM Ch6)



- Create a word-context count matrix
  - **Count** the number of co-occurrences of word/context, with rows as word, columns as contexts

Example corpus

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

Vector Representations

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Count-based Methods

(JM Ch6)

- Create a word-context count matrix
  - **Count** the number of co-occurrences of word/context, with rows as word, columns as contexts

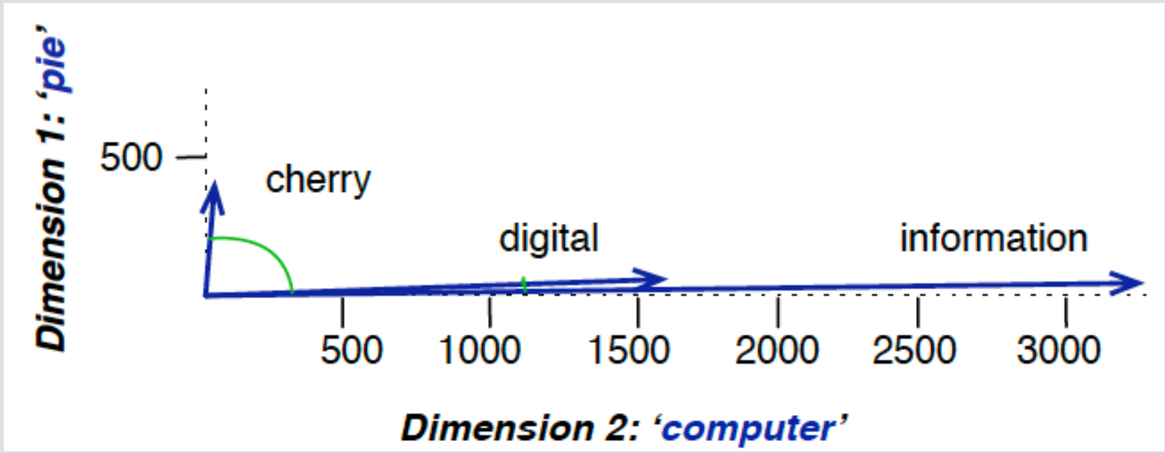
- Dealing with “word importance”: co-occurring with words like “they” “the” “it” should not mean anything!
  - Solution 1: weight with point-wise mutual information (PMI)
  - Solution 2 (when context is a document): Term Frequency-Inverse Document Frequency (TF-IDF)

$$\text{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

Measuring the Closeness
(JM Ch6)

- We now use a vector to represent each word
- How do we know whether two words are similar?
- **Measure their closeness** using cosine similarity

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$



Prediction-based Methods

(See Goldberg 10.4.2)

- Instead, try to **predict** the words within a neural network
  - Word embeddings are the byproduct
  - Much *denser* compared with count-based vectors
- Example: Neural Language Models (next lecture)
  - i.e., learning good word embeddings, such that the LM model could be driven to predict the correct next word

Context Window Methods

- If we don’t need to calculate the probability of the sentence, other methods are still possible!
- These can move **closer to the contexts used in count-based methods**
- These drive word2vec, etc.

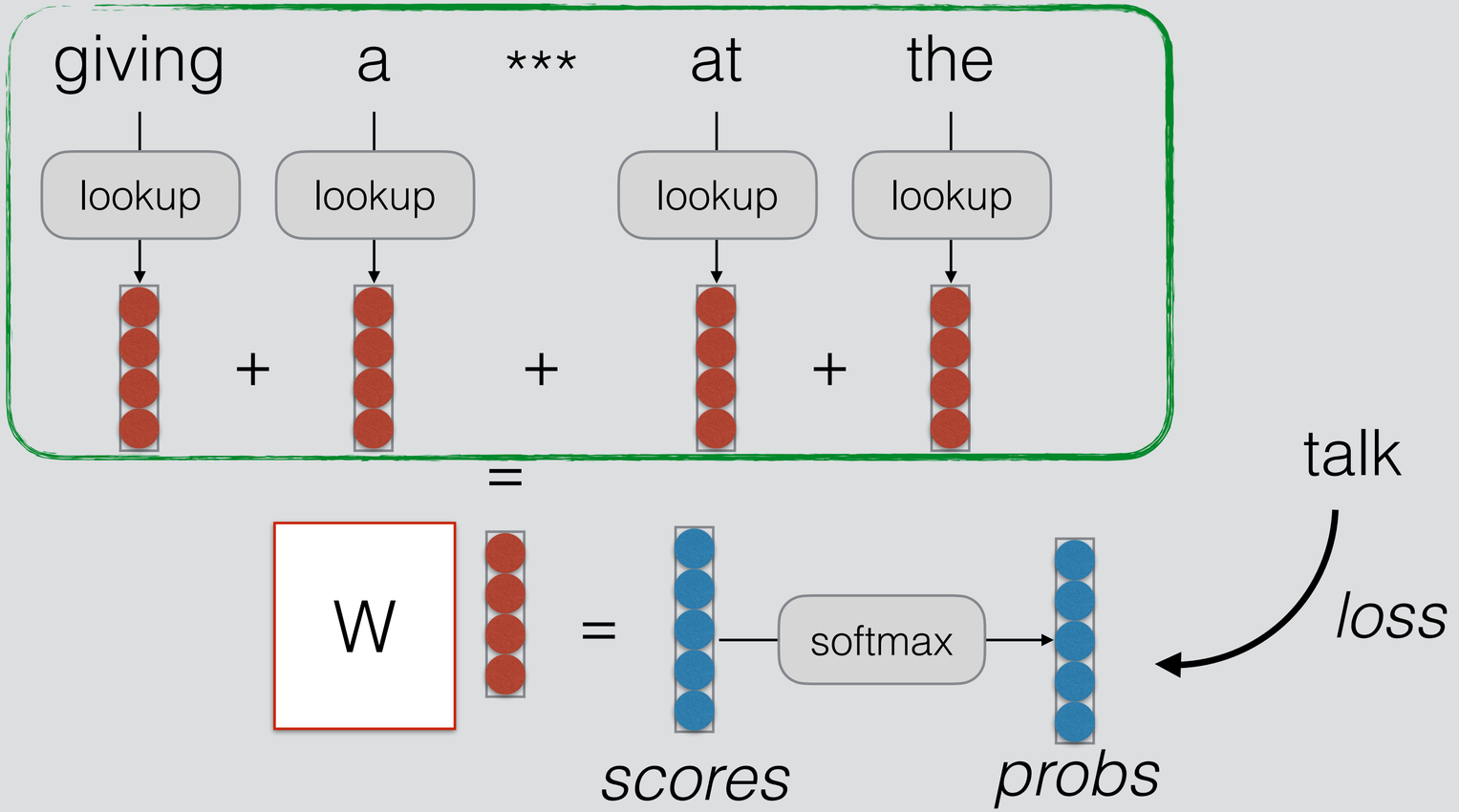
Continuous Bag-Of-Word

CBOW

(Mikolov et al. 2013)

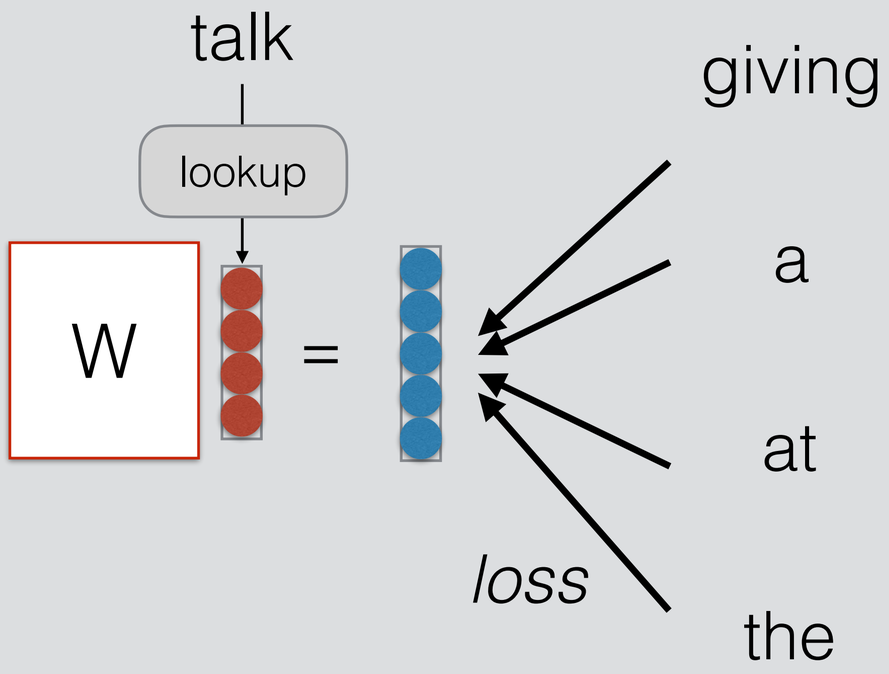
- Predict word based on sum of surrounding embeddings

Order-insensitive => *Bag* of words



Skip-gram
(Mikolov et al. 2013)

- Predict each word in the context given the word



Skip-gram
(Mikolov et al. 2013)

- Predict each word in the context given the word
  - Formally in the paper:

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^{\top} v_{w_I}\right)}$$

  - v_w and v'_w are “input” and “output” vector representations of word w
  - W: the vocabulary set

Negative Sampling

- Idea: encourage “good” word-context pairs and discourage “bad” ones
  - A probabilistic version of margin-based ranking loss

$\sigma(-x) = 1 - \sigma(x)$
(the sigmoid function)

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right]$$

- w_O: the “good” word for the context word w_I
- $P_n(w)$: the noise distribution
- $w_i \sim P_n(w)$: negative samples

What Contexts?

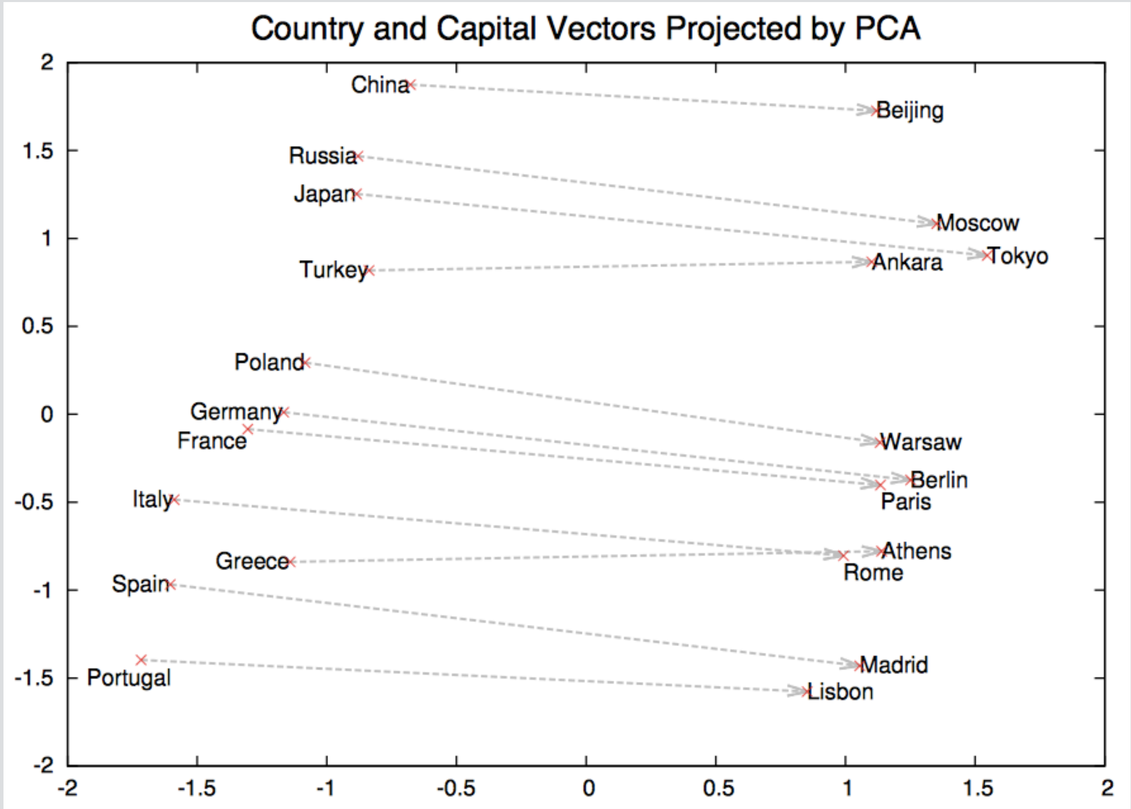
- Context has a large effect!
- **Small context window:** more syntax-based embeddings
- **Large context window:** more semantics-based, topical embeddings
- **Context based on syntax:** more functional, w/ words with same inflection grouped
  - e.g., by leveraging a dependency parser

Outline

- Refresher: Classification
- Feed-Forward Neural Nets
- Word Embedding
  - Vector Semantics
  - Evaluation of Word Embedding

Visualization of Embeddings

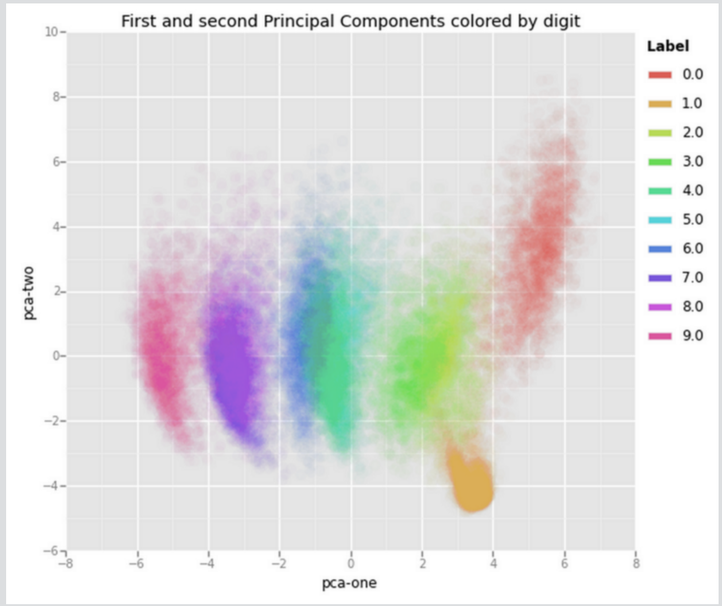
- Reduce high-dimensional embeddings into 2/3D for visualization (e.g. Mikolov et al. 2013)

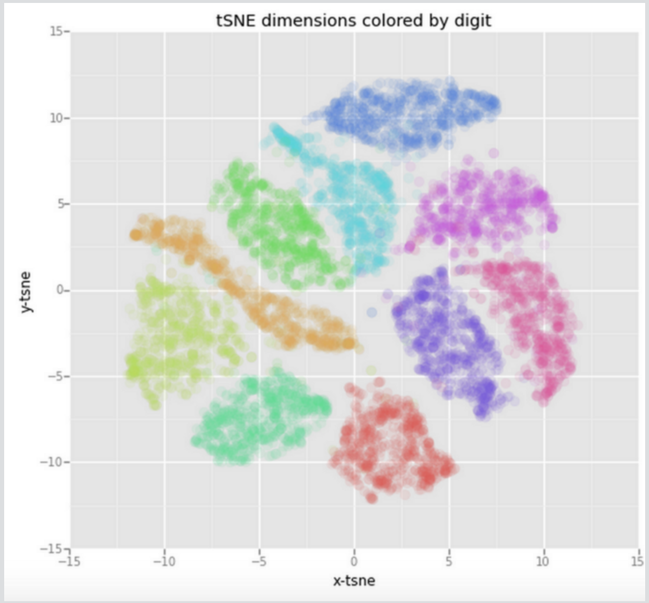


Non-linear Projection

- Non-linear projections group things that are close in high-dimensional space
- e.g. SNE/t-SNE (van der Maaten and Hinton 2008) group things that give each other a high probability according to a Gaussian

PCA





t-SNE

(Image credit: Derksen 2016)

Types of Evaluation

- Intrinsic vs. Extrinsic
  - **Intrinsic:** How good is it based on its features?
  - **Extrinsic:** How useful is it for the downstream tasks?
- Qualitative vs. Quantitative
  - **Qualitative:** Examine the characteristics of examples.
  - **Quantitative:** Calculate statistics

Intrinsic Evaluation of Embeddings (Mikolov et al., 2013)

- A Seman
  - Analogy
    - e.g., if X
      “smalles
    - 5 types

uld be

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

Intrinsic Evaluation of Embeddings (Mikolov et al., 2013)

- A Semantic-Syntactic Word Relationship test set
  - Analogy via algebraic operations:
    - e.g., if X = V(“biggest”) - V(“big”) + V(“small”), then the closet word to X should be “smallest”
  - 5 types of semantic questions & 9 types of syntactic questions
- Evaluation: accuracy on retrieving the correct word as the closest word

Intrinsic Evaluation of Embeddings (Mikolov et al., 2013)

- Accuracy vs. model architecture

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

- Observation: more training data works only with sufficiently high dimensionality

Intrinsic Evaluation of Embeddings (Mikolov et al., 2013)

- Accuracy vs. model architecture

(syntactic)

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

- Observations:
  - (1) both better than prev LMs; (2) CBOW slightly better at syntactic relationship, while Skip-gram much better at semantic relationship

Intrinsic Evaluation of Embeddings

(categorization from Schnabel et al 2015)

- **Relatedness:** The correlation between embedding cosine similarity and human eval of similarity?
- **Analogy:** Find x for "*a is to b, as x is to y*".
- **Categorization:** Create clusters based on the embeddings, and measure purity of clusters.
- **Selectional Preference:** Determine whether a noun is a typical argument of a verb.

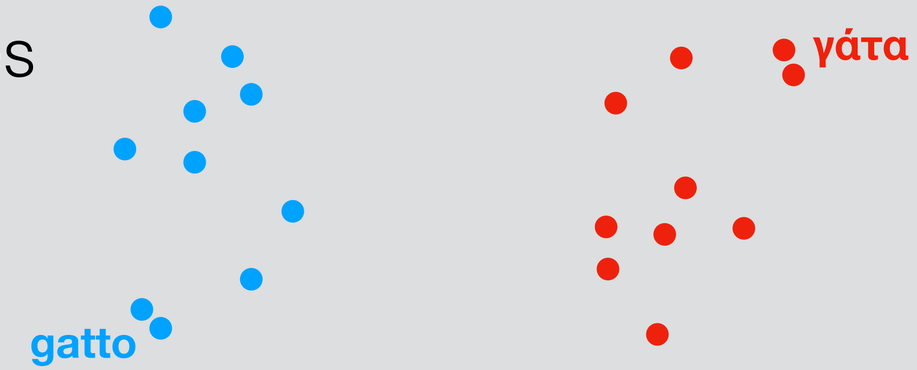
Schnabel, Tobias, et al. "Evaluation methods for unsupervised word embeddings."
Proceedings of the 2015 conference on empirical methods in natural language processing. 2015.

Extrinsic Evaluation: Using Word Embeddings in Systems

- Typically two ways of usage:
  - **Initialize** w/ the embeddings
  - **Concatenate** pre-trained embeddings with learned embeddings
- Latter is more expressive, but leads to increase in model parameters

Evaluation with Lexicon Induction

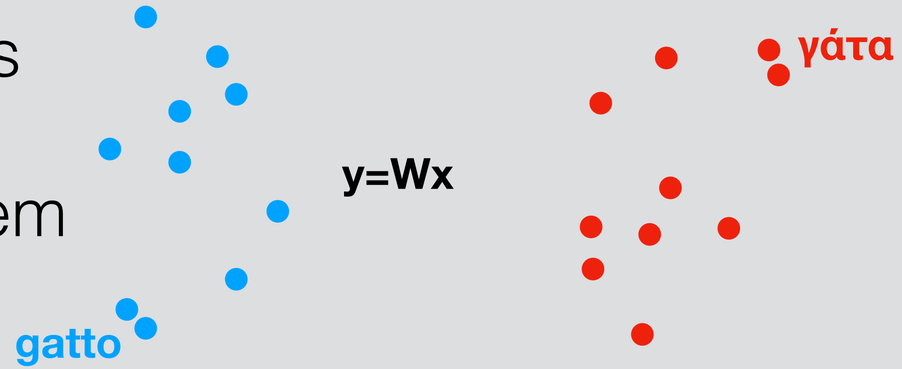
Learn embeddings for two languages



Evaluation with Lexicon Induction

Learn embeddings for two languages

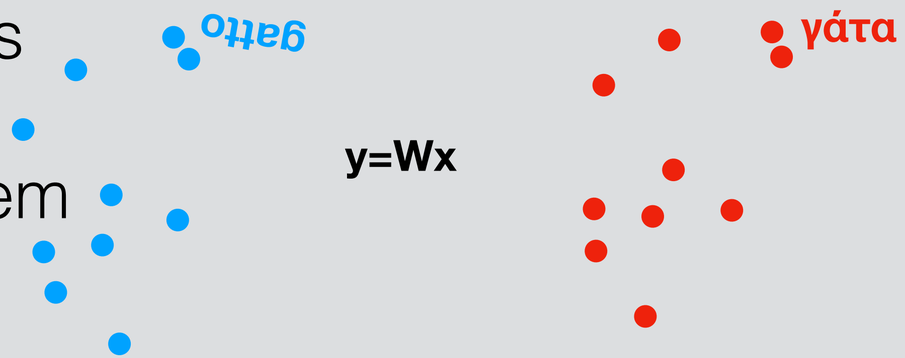
Learn a transformation that aligns them



Evaluation with Lexicon Induction

Learn embeddings for two languages

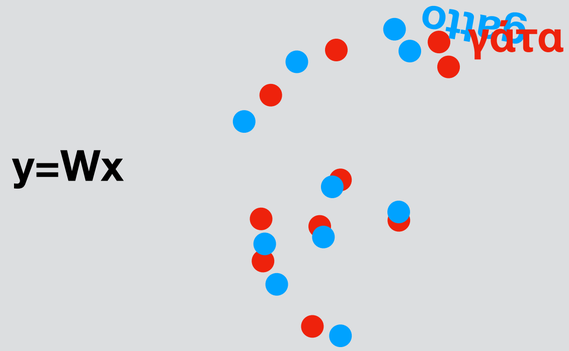
Learn a transformation that aligns them



Evaluation with Lexicon Induction

Learn embeddings for two languages

Learn a transformation that aligns them

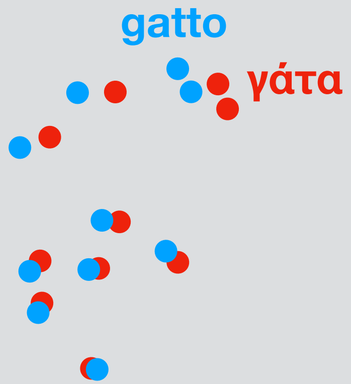


Evaluation with Lexicon Induction

Learn embeddings for two languages

Learn a transformation that aligns them

Extract Lexicon



gatto	γάτα
gatti	γάτες
purtroppo	δυστυχώς
...	...

Zou et al. 2013,
Smith et al. 2013,
Dinu and Baroni 2014,
Zhang et al. 2017,
Artetxe et al. 2017,
Conneau et al. 2018
Artetxe et al. 2018

How Do I Choose Embeddings?

- No one-size-fits-all embedding (Schnabel et al 2015)

	relatedness						categorization			sel. prefs		analogy			
	rg	ws	wss	wsr	men	toefl	ap	esslli	batt.	up	mcrae	an	ansyn	ansem	average
CBOW	**74.0**	**64.0**	**71.5**	**56.5**	**70.7**	66.7	**65.9**	**70.5**	**85.2**	24.1	13.9	**52.2**	**47.8**	**57.6**	**58.6**
GloVe	63.7	54.8	65.8	49.6	64.6	**69.4**	64.1	65.9	77.8	27.0	**18.4**	42.2	44.2	39.7	53.4
TSCCA	57.8	54.4	64.7	43.3	56.7	58.3	57.5	**70.5**	64.2	**31.0**	14.4	15.5	19.0	11.1	44.2
C&W	48.1	49.8	60.7	40.1	57.5	66.7	60.6	61.4	80.2	28.3	16.0	10.9	12.2		
H-PCA	19.8	32.9	43.6	15.1	21.3	54.2	34.1	50.0	42.0	-2.5	3.2	3.0	2.4		
Rand. Proj.	17.1	19.5	24.9	16.1	11.3	51.4	21.9	38.6	29.6	-8.5	1.2	1.0	0.3		

Table 1: Results on absolute intrinsic evaluation. The best result for each dataset is highligl
The second row contains the names of the corresponding datasets.

(Intrinsic Evaluation)

- Be aware, and use the best one for the task

	dev	test	p-value
Baseline	94.18	93.78	0.000
Rand. Proj.	94.33	93.90	0.006
GloVe	94.28	93.93	0.015
H-PCA	94.48	93.96	0.029
C&W	**94.53**	**94.12**	
CBOW	94.32	93.93	0.012
TSCCA	**94.53**	94.09	0.357

Table 4: F1 chunking results using different word embeddings as features. The p-values are with respect to the best performing method.

(Extrinsic Evaluation)

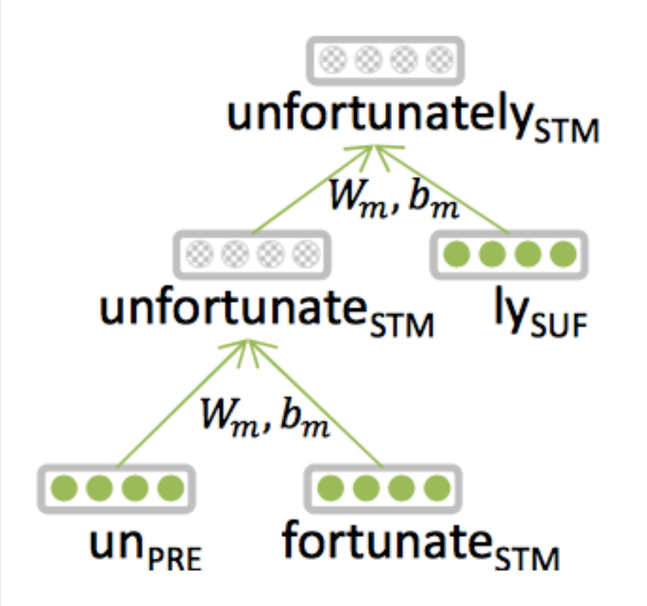
Limitations of Embeddings

- Sensitive to **superficial differences** (dog/dogs)
- **Insensitive to context** (financial bank, bank of a river)
- **Not necessarily coordinated** with knowledge or across languages
- **Not interpretable**
- Can **encode bias** (encode stereotypical gender roles, racial biases)

(Brief; offline study)

Sub-word Embeddings (1)

- Can capture sub-word regularities

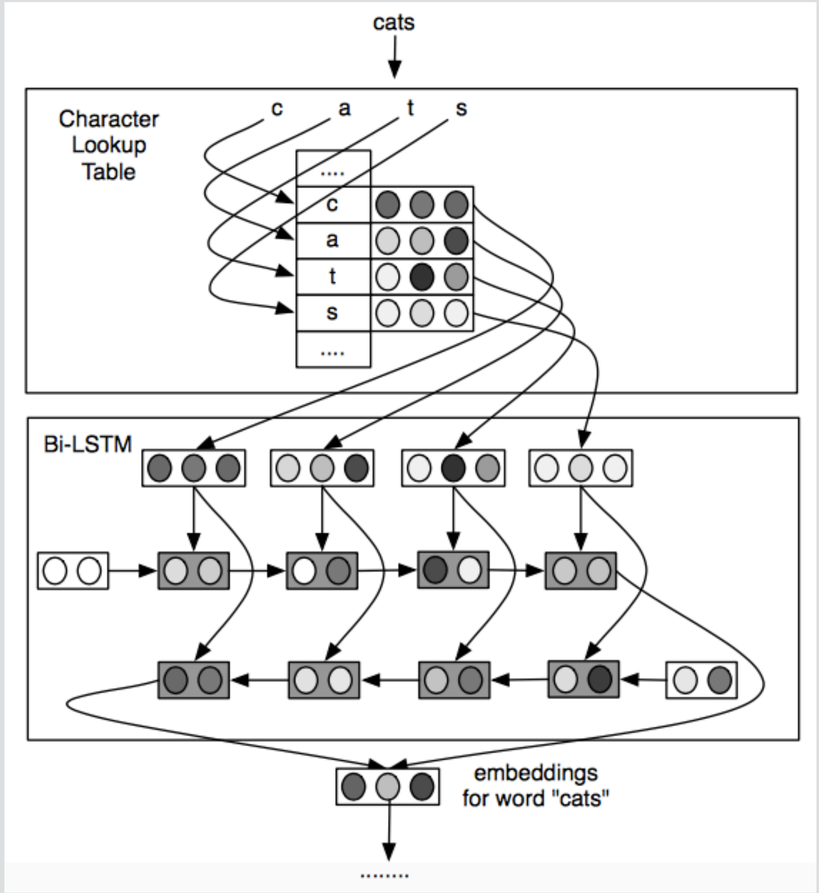


Morpheme-based
(Luong et al. 2013)

(Brief; offline study)

Sub-word Embeddings (2)

- Can capture sub-word regularities

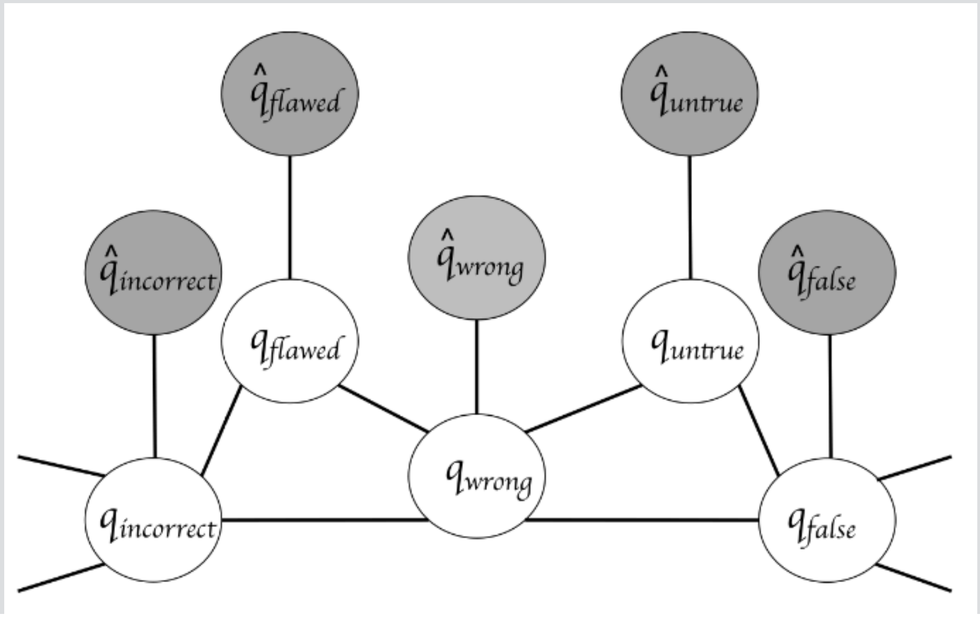


Character-based (Ling et al. 2015)

(Brief; offline study)

Retrofitting of Embeddings to Existing Lexicons

- We have an existing lexicon like WordNet, and would like our vectors to match (Faruqui et al. 2015)



$$\Psi(Q) = \sum_{i=1}^{n} \left[\alpha_i \|q_i - \hat{q}_i\|^2 + \sum_{(i,j) \in E} \beta_{ij} \|q_i - q_j\|^2 \right]$$

(Brief; offline study)

De-biasing Word Embeddings

(Bolukbasi et al. 2016)

- Word embeddings reflect bias in statistics

Extreme *she*	**Extreme *he***
1. homemaker	1. maestro
2. nurse	2. skipper
3. receptionist	3. protege
4. librarian	4. philosopher
5. socialite	5. captain
6. hairdresser	6. architect
7. nanny	7. financier
8. bookkeeper	8. warrior
9. stylist	9. broadcaster
10. housekeeper	10. magician

Gender stereotype *she-he* analogies		
sewing-carpentry	registered nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	lovely-brilliant

Gender appropriate *she-he* analogies		
queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

- Identify pairs to “neutralize”, find the direction of the trait to neutralize, and ensure that they are neutral in that direction

Outline

- Refresher: Classification
- Feed-Forward Neural Nets
- Word Embedding
  - Vector Semantics
  - Evaluation of Word Embedding