New Directions in Vector Space Models of Meaning

Edward Grefenstette[1] Karl Moritz Hermann[1]
Georgiana Dinu[2] Phil Blunsom[1]

[1]Dept of Computer Science
University of Oxford

[2]Centre for Mind/Brain Sciences
University of Trento

ACL 2014 Tutorial

Slides at: `http://www.clg.ox.ac.uk/resources`

What is the meaning of life?

A joke for semanticists

Q: What is the meaning of life?

What is the meaning of life?

A joke for semanticists

Q: What is the meaning of life?

A: *life′*

What is the meaning of life?

A joke for semanticists

Q: What is the meaning of life?
A: $life'$ / $I(life)$ / $[\![\text{life}]\!]$ / etc.

What is the meaning of life?

A joke for semanticists

Q: What is the meaning of life?
A: $life'$ / $I(life)$ / $[\![\text{life}]\!]$ / etc.

- What semantic value to give $life'$?
  - Logical atom?
  - Logical predicate/relation?
  - Just the token itself?
- What is the relation between life and death?
- How can we infer the meaning of life?

Symbolic success

We like the symbolic/discrete approach because...

- Discrete models can be cheap and fast
- Many success stories. E.g.:
  - n-gram language models
  - POS tagging/parsing
- Logical analysis:
  - Long history
  - Powerful inference

Your logic is no good here...

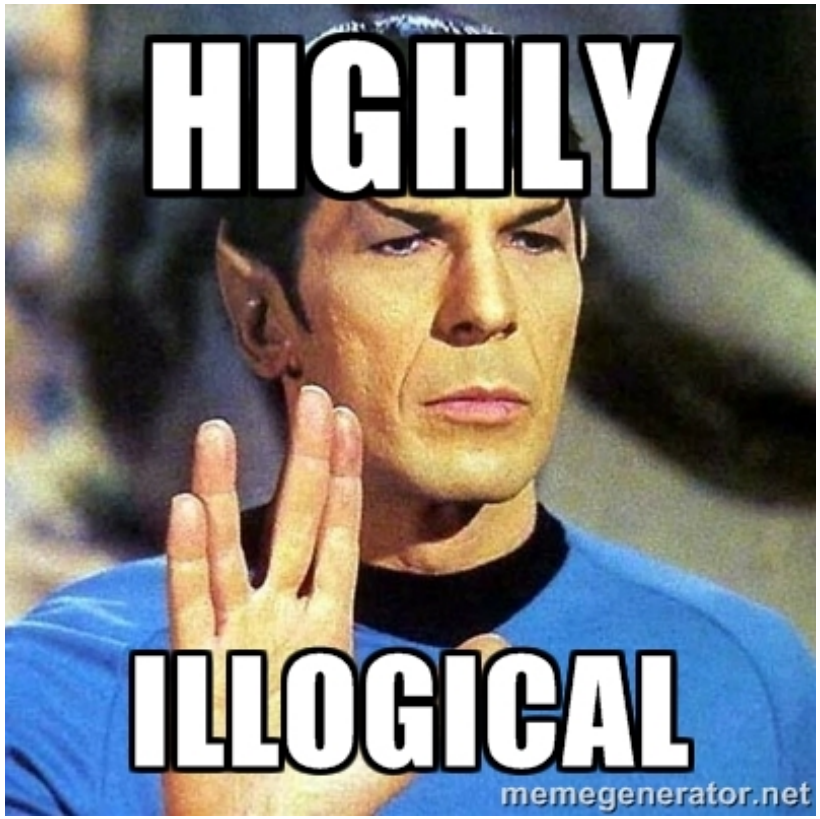
But...

- Doesn't capture "messiness"
- No similarity
- Sparsity
- Rules are hard to learn
- Limited variety of inference

Your logic is no good here...

But...

- Doesn't capture "messiness"
- No similarity
- Sparsity
- Rules are hard to learn
- Limited variety of inference



Vector representations for words

- Go from discrete to distributed representations
- Word meanings are vectors of properties
- Well studied mathematical structure
- Well motivated, theoretically and practically

Background

Philosophy	Hume, Wittgenstein
Linguistics	Firth, Harris
Engineering + Statistics	Feature vectors

Vector representations for words

Many successful applications in lexical semantics:

- Word-sense disambiguation
- Thesaurus extraction

Also many use cases in NLP pipelines, e.g.:

- Automated essay marking
- Plagiarism detection

More than mere words

What's missing?

Word representations alone are not enough to do:

- Machine Translation
- Information Extraction
- Question Answering
- etc.

We need sentence/document representations.

Vector representations for phrases

What could we do with sentence/document vectors?

- Generation
  - English translation from French sentence
  - Next sentence in a conversation
  - Metadata for documents
- Classification
  - Topic/sentiment
  - Stock market predictions ($$$!!)
  - Recommendations (movies, books, restaurants)

Vector representations for phrases

Why can we classify and generate with vectors?

- Learn spatial boundaries to separate subspaces
- Similarity metrics give predictors for next word
- Geometric transforms model contextual influence

Tasks for vector models of meaning

Today's tutorial is about two kinds of basic tasks for the construction of vector models of meaning:

- Learning vector representations for words
- Learning how to compose them to get vector representations for phrases/sentences/documents

Today's menu

1 Distributional Semantics

2 Neural Distributed Representations

3 Semantic Composition

4 Last Words

Goals of this tutorial

By the end of this tutorial, you should have:

- A good understanding of distributed word representations and their usage.
- Some background knowledge about neural language models and (conditional) generation.
- A decent overview of options for integrating compositionality into vector-based models.
- Sufficient knowledge about the terms and mathematics of neural methods to read deep learning papers in NLP.
- Hopefully, some new ideas of your own!

Outline

1 Distributional Semantics

2 Neural Distributed Representations

3 Semantic Composition

4 Last Words

The distributional hypothesis

We found a cute little wampimuk©*MarcoBaroni* ?
sleeping in a tree.

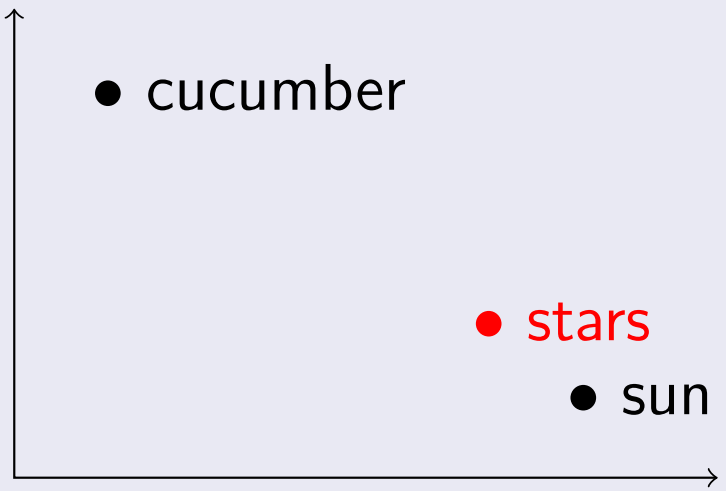
Distributional Semantics in a nutshell

he curtains open and the stars shining in on the barely
ars and the cold , close stars " . And neither of the w
rough the night with the stars shining so brightly , it
made in the light of the stars . It all boils down , wr
surely under the bright stars , thrilled by ice-white
sun , the seasons of the stars ? Home , alone , Jay pla
m is dazzling snow , the stars have risen full and cold
un and the temple of the stars , driving out of the hug
in the dark and now the stars rise , full and amber a
bird on the shape of the stars over the trees in front
But I could n't see the stars or the moon , only the
they love the sun , the stars and the stars . None of
r the light of the shiny stars . The plash of flowing w
man 's first look at the stars ; various exhibits , aer
rief information on both stars and constellations, inc

Construct vector representations

	shining	bright	trees	dark	look
stars	38	45	2	27	12

Similarity in meaning as vector similarity



In more detail

Core components of distributional models of semantics:

- Co-occurrence counts extraction
- Weighting schemes
- Dimensionality reduction
- Similarity measures

Extracting co-occurrence counts

A matrix of co-occurrence counts is built, representing the target linguistic units over context features.

Variations in the type of context features

	Doc1	Doc2	Doc3
stars	38	45	2

	$\xleftarrow{dobj}$ see	$\xrightarrow{mod}$ bright	$\xrightarrow{mod}$ shiny
stars	38	45	44

	The nearest • to Earth	stories of • and their
stars	12	10

Variations in the definition of *co-occurrence*

Co-occurrence with words, window of size 2, scaling by distance to target:

... two *[intensely bright stars in the]* night sky ...

	intensely	bright	in	the
stars	0.5	1	1	0.5

For more details, see:

- Pado and Lapata (2007),
- Turney and Pantel (2010).
- Comparisons: Agirre et al (2009), Baroni and Lenci (2010), Bullinaria and Levy (2012), Kiela and Clark (2014)

Re-weight the counts using corpus-level statistics to reflect co-occurrence *significance*.

Point-wise Mutual Information (PMI)

$$\mathrm{PMI}(\mathrm{target}, \mathrm{ctxt}) = \log \frac{\mathrm{P}(\mathrm{target}, \mathrm{ctxt})}{\mathrm{P}(\mathrm{target})\mathrm{P}(\mathrm{ctxt})}$$

Weighting

Adjusting raw collocational counts:

	bright	in		
stars	385	10788	...	← Counts
stars	43.6	5.3	...	← Pmi

Other weighting schemes:

- TF-IDF
- Local Mutual Information
- Dice

See Ch4 of J.R. Curran's thesis (2004) for a great survey.

Dimensionality reduction

Problem

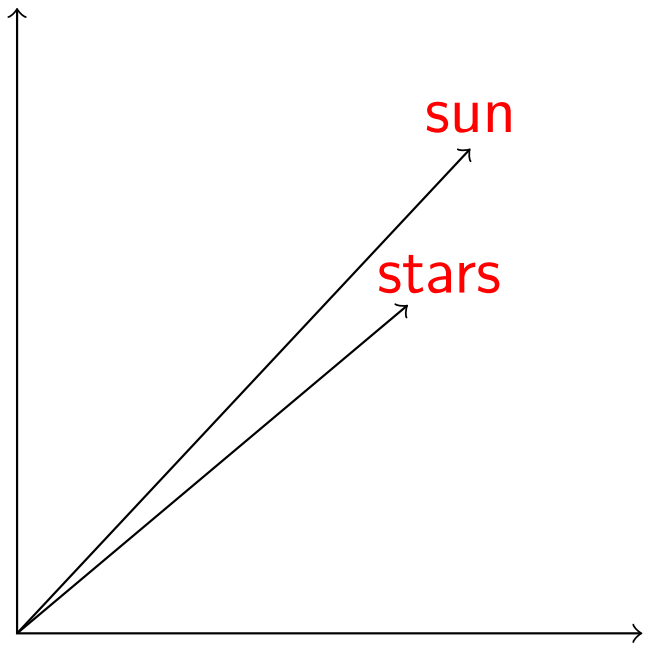
Vectors spaces often range from tens of thousands to millions of dimensions.

Dimensionality reduction

Some of the methods to reduce dimensionality:

- Select context features based on various relevance criteria
- Random indexing
- Having also a *smoothing* effect
  - Singular Value Decomposition
  - Non-negative matrix factorization
  - Probabilistic Latent Semantic Analysis
  - Latent Dirichlet Allocation

Vector similarity measures (or inverted distance measures) are used to approximate similarity in meaning.



Cosine similarity

$$\cos(\mathrm{x}, \mathrm{y}) = \frac{\mathrm{x} \cdot \mathrm{y}}{\|\mathrm{x}\| \times \|\mathrm{y}\|}$$

Distance and similarity

Other similarity measures:

- Euclidean
- Lin
- Jaccard
- Dice
- Kullback-Leibler (for distributions)

Summary

Distributional tradition: Vector representations over intuitive, linguistically-motivated, context features

- Pros: Easy to obtain, vectors are interpretable
- Cons: Involves a large number of design choices (what weighting scheme? what similarity measure?)
- Problems: Going from word to sentence representations is non-trivial, and no clear intuitions exist.

An Open Question

Are there other ways to learn composeable vector representations of meaning, based on the distributional hypothesis, without this parametric burden?

Outline

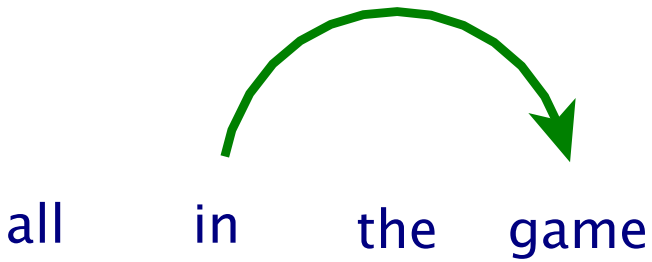
1 Distributional Semantics

2 Neural Distributed Representations

3 Semantic Composition

4 Last Words

Features and NLP



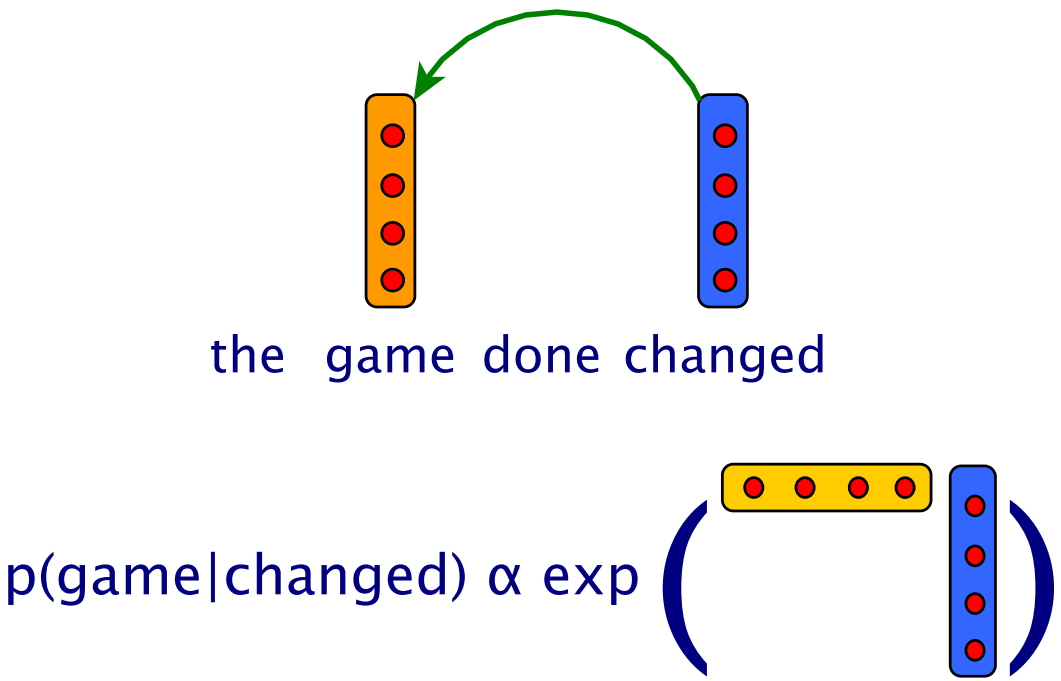
$$p(\text{game}|\text{in}) \propto \exp(\mathbf{w}^T \Phi(\text{game},\text{in}))$$

$$\Phi_1(x,y) = \begin{cases} 1, & \text{if PoS}(x)=\text{Noun} \;\&\; y=\text{in} \\ 0, & \text{otherwise} \end{cases}$$

$$\Phi_2(x,y) = \begin{cases} 1, & \text{if } x=\text{game} \;\&\; \text{PoS}(y)=\text{Prep} \\ 0, & \text{otherwise} \end{cases}$$

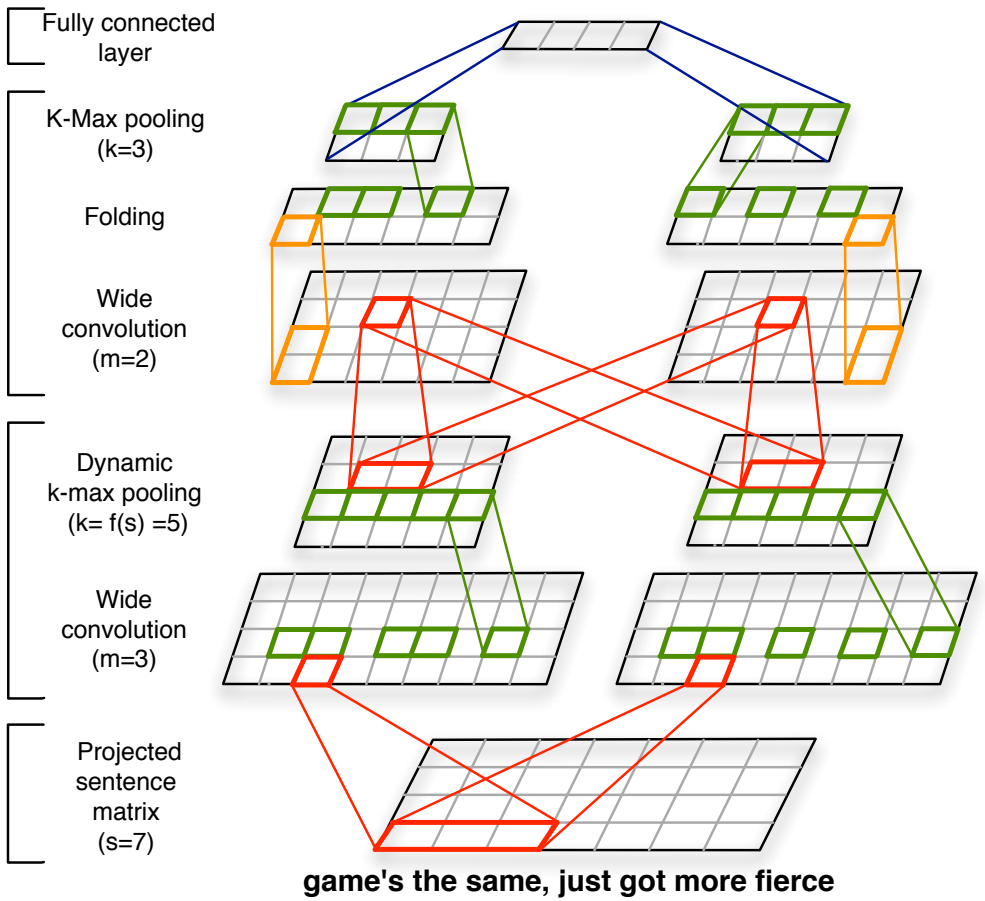
etc.

Twenty years ago log-linear models freed us from the shackles of simple multinomial parametrisations, but imposed the tyranny of feature engineering.



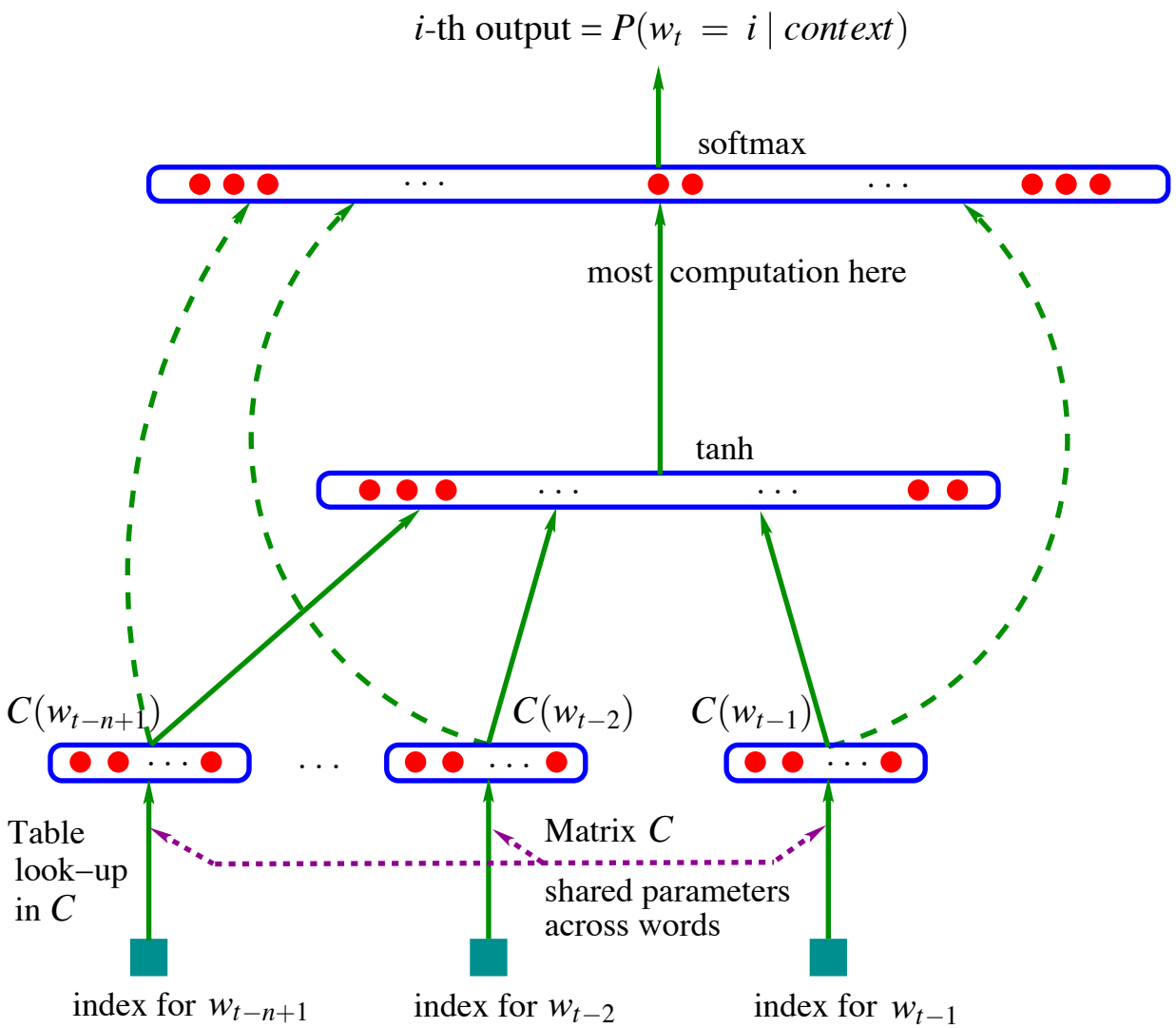
Distributed/neural models allow us to learn shallow features for our classifiers, capturing simple correlations between inputs.

Features and NLP



Deep learning allows us to learn hierarchical generalisations. Something that is proving rather useful for vision, speech, and now NLP...

Neural language models



A Neural Probabilistic Language Model. Bengio et al. JMLR 2003.

Log-linear models for classification

Features: $\phi(\mathbf{x}) \in \mathbb{R}^D$ and weights: $\lambda_k \in \mathbb{R}^D$ for $k \in \{1, ..., K\}$ classes:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{\exp(\lambda_k^\mathsf{T}\phi(\mathbf{x}))}{\sum_j^K \exp(\lambda_j^\mathsf{T}\phi(\mathbf{x}))}$$

Gradient required for training:

$$\begin{aligned}
\frac{\partial}{\partial \lambda_j}\Big[-\log p(\mathcal{C}_k|\mathbf{x})\Big] &= \frac{\partial}{\partial \lambda_j}\log \mathcal{Z}(\mathbf{x}) - \frac{\partial}{\partial \lambda_j}\lambda_k^\mathsf{T}\phi(\mathbf{x}) \\
&= \frac{1}{\mathcal{Z}(\mathbf{x})}\frac{\partial}{\partial \lambda_j}\exp\left(\lambda_j^\mathsf{T}\phi(\mathbf{x})\right) - \frac{\partial}{\partial \lambda_j}\lambda_k^\mathsf{T}\phi(\mathbf{x}) \\
&= \frac{\exp\left(\lambda_j^\mathsf{T}\phi(\mathbf{x})\right)}{\mathcal{Z}(\mathbf{x})}\phi(\mathbf{x}) - \frac{\partial}{\partial \lambda_j}\lambda_k^\mathsf{T}\phi(\mathbf{x}) \\
&= \underbrace{p(\mathcal{C}_j|\mathbf{x})\phi(\mathbf{x})}_{\text{expected features}} - \underbrace{\delta(j,k)\phi(\mathbf{x})}_{\text{observed features}}
\end{aligned}$$

$\delta(j, k)$ is the Kronecker delta function which is 1 if $j = k$ and 0 otherwise, and $\mathcal{Z}(\mathbf{x}) = \sum_j^K \exp(\lambda_j^\mathsf{T}\phi(\mathbf{x}))$ is referred to as the partition function.

A simple log-linear (tri-gram) language model

this is America

$$\Phi_1(w_{n-1}, w_{n-2}) = \begin{cases} 1, \text{ if } w_{w-2}=\text{this } \& \ w_{n-1}=\text{is} \\ 0, \text{ otherwise} \end{cases}$$

$$\Phi_2(w_{n-1}, w_{n-2}) = \begin{cases} 1, \text{ if } w_{w-2}=\text{this} \\ 0, \text{ otherwise} \end{cases}$$

$$\Phi_3(w_{n-1}, w_{n-2}) = \begin{cases} 1, \text{ if } w_{w-1}=\text{is} \\ 0, \text{ otherwise} \end{cases}$$

Classify the next word w_n given w_{n-1}, w_{n-2}: Features: $\phi(w_{n-1}, w_{n-2}) \in \mathbb{R}^D$ and weights: $\lambda_i \in \mathbb{R}^D$:[1]

$$p(w_n|w_{n-1}, w_{n-2}) \propto \exp\left(\lambda_{w_n}^\top \phi(w_{n-1}, w_{n-2}) + b_{w_n}\right)$$

Traditionally the feature maps $\phi(\cdot)$ are rule based, but can we learn them from the data?

[1] we now explicitly include a per-word bias parameter b_{w_n} that is initialised to the empirical log $p(w_n)$.

A simple log-linear (tri-gram) language model

this is America

$$\Phi_1(w_{n-1}, w_{n-2}) = \begin{cases} 1, \text{ if } w_{w-2}=\text{this } \& \ w_{n-1}=\text{is} \\ 0, \text{ otherwise} \end{cases}$$

$$\Phi_2(w_{n-1}, w_{n-2}) = \begin{cases} 1, \text{ if } w_{w-2}=\text{this} \\ 0, \text{ otherwise} \end{cases}$$

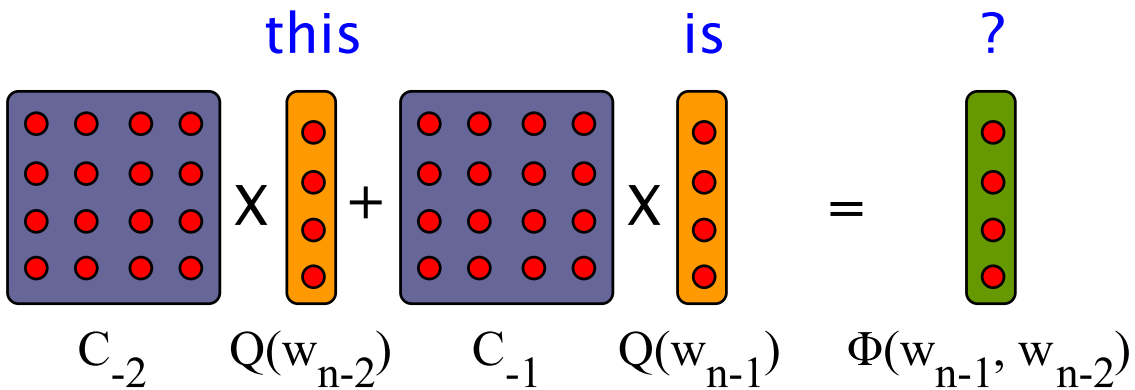
$$\Phi_3(w_{n-1}, w_{n-2}) = \begin{cases} 1, \text{ if } w_{w-1}=\text{is} \\ 0, \text{ otherwise} \end{cases}$$

Traditionally the feature maps $\phi(\cdot)$ are rule based, but can we learn them from the data?
Assume the features factorise across the context words:

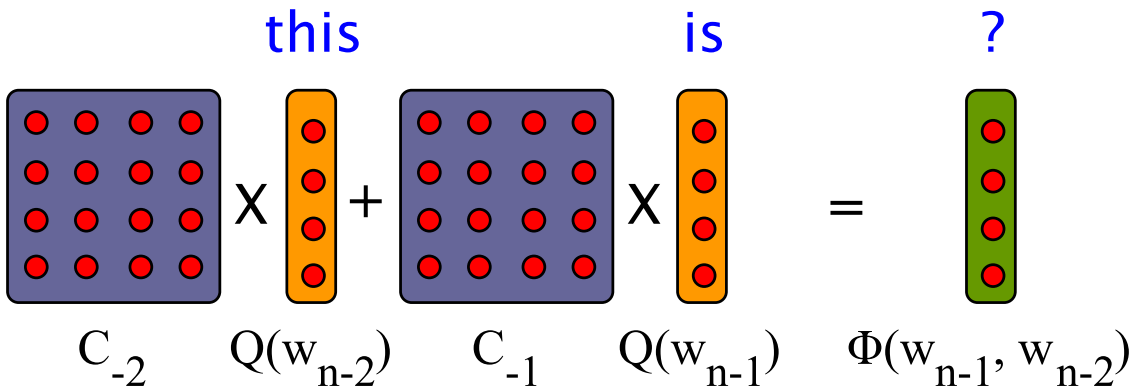
$$p(w_n|w_{n-1}, w_{n-2}) \propto \exp\left(\lambda_{w_n}^{\mathsf{T}}\left(\phi_{-1}(w_{n-1}) + \phi_{-2}(w_{n-2})\right) + b_{w_n}\right)$$

Learning the features: the log-bilinear language model

Represent the context words by the columns of a $D \times |\text{vocab}|$ matrix Q, and output words by the columns of a matrix R; assume ϕ_i is a linear function of these representations parametrised by a matrix C_i:



Learning the features: the log-bilinear language model



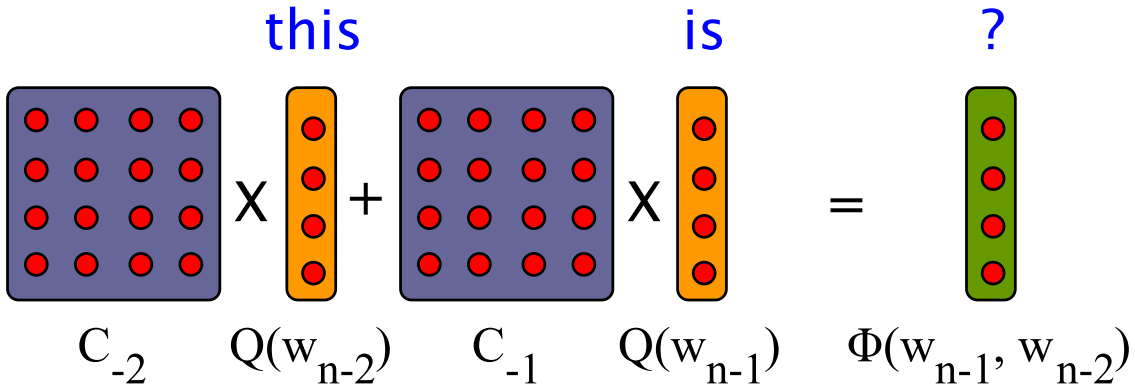
$$\phi(w_{n-1}, w_{n-2}) = C_{-2}Q(w_{n-2}) + C_{-1}Q(w_{n-1})$$

$$p(w_n|w_{n-1}, w_{n-2}) \propto \exp\left(R(w_n)^T \phi(w_{n-1}, w_{n-2}) + b_{w_n}\right)$$

This is referred to as a *log-bilinear model.*[2]

[2] Three new graphical models for statistical language modelling. Mnih and Hinton, ICML'07.

Learning the features: the log-bilinear language model



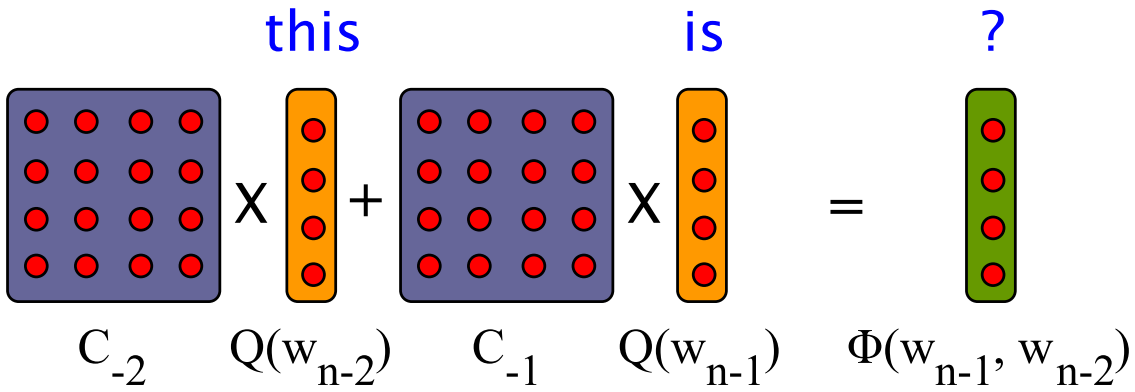
$$p(w_n|w_{n-1}, w_{n-2}) \propto \exp\left(R(w_n)^T \phi(w_{n-1}, w_{n-2}) + b_{w_n}\right)$$

Error objective: $E = -\log p(w_n|w_{n-1}, w_{n-2})$

$$\frac{\partial}{\partial R(j)} E = \frac{\partial}{\partial R(j)} \log \mathcal{Z}(w_{n-1}, w_{n-2}) - \frac{\partial}{\partial R(j)} R(w_n)^\mathsf{T} \phi$$

$$= \Big(p(j|w_{n-1}, w_{n-2}) - \delta(j, w_n)\Big)\phi$$

Learning the features: the log-bilinear language model

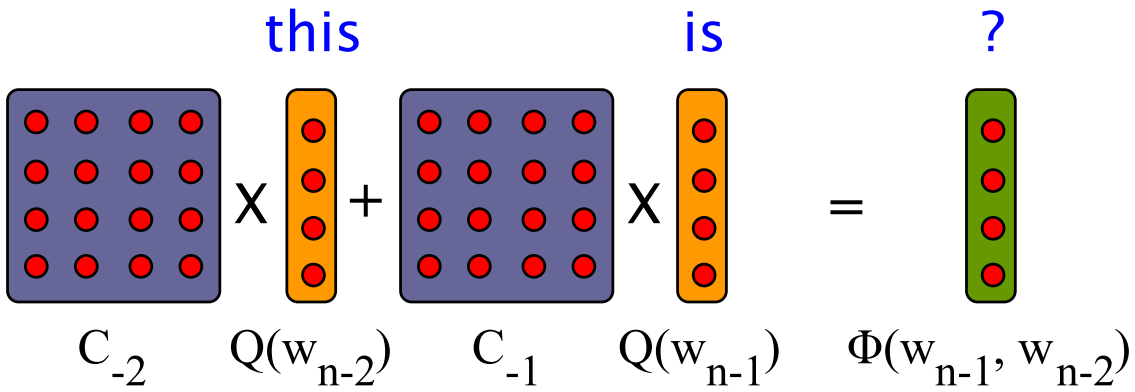


Error objective: $E = -\log p(w_n|w_{n-1}, w_{n-2})$

$$\frac{\partial}{\partial\phi}E = \frac{\partial}{\partial\phi}\log\mathcal{Z}(w_{n-1}, w_{n-2}) - \frac{\partial}{\partial\phi}R(w_n)^{\mathsf{T}}\phi$$

$$= \underbrace{\left[\sum_j p(j|w_{n-1}, w_{n-2})R(w_j)\right]}_{\text{model expected next word vector}} - \underbrace{R(w_n)}_{\text{data vector}}$$

Learning the features: the log-bilinear language model



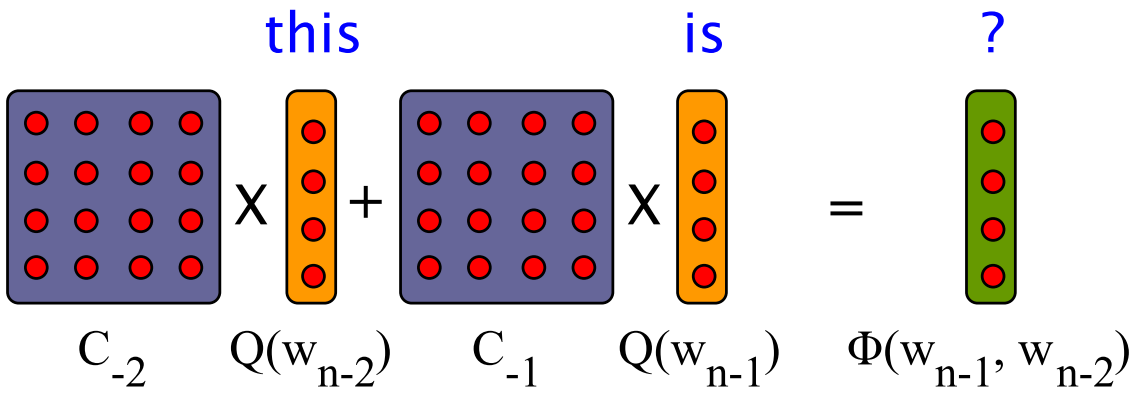
Error objective: $E = -\log p(w_n | w_{n-1}, w_{n-2})$

$$\frac{\partial}{\partial Q(j)} E = \frac{\partial \phi}{\partial Q(j)} \times \frac{\partial E}{\partial \phi}$$

$$\frac{\partial \phi}{\partial Q(j)} = \frac{\partial}{\partial Q(j)} \left[C_{-2} Q(w_{n-2}) + C_{-1} Q(w_{n-1}) \right]$$

$$= \delta(j, w_{n-2}) C_{-2}^{\mathsf{T}} + \delta(j, w_{n-1}) C_{-1}^{\mathsf{T}}$$

Learning the features: the log-bilinear language model



Error objective: $E = -\log p(w_n | w_{n-1}, w_{n-2})$

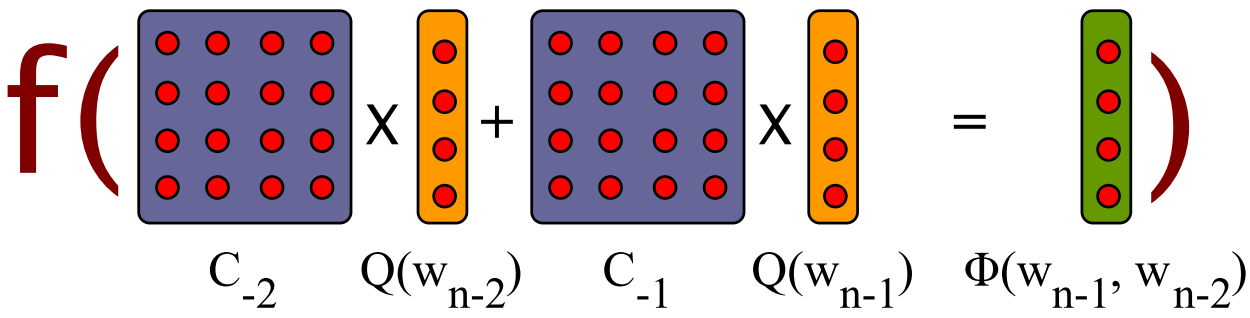
$$\frac{\partial}{\partial C_{-2}} E = \frac{\partial E}{\partial \phi} \times \frac{\partial \phi}{\partial C_{-2}}$$

$$\frac{\partial \phi}{\partial C_{-1}} = \frac{\partial}{\partial A} \Big[C_{-1} Q(w_{n-2}) + C_{-2} Q(w_{n-1}) \Big]$$

$$= Q(w_{n-2})^{\mathsf{T}}$$

Adding non-linearities: the neural language model

Replacing the simple bi-linear relationship between context and output words with a more powerful non-linear function $f(\cdot)$ (logistic sigmoid, tanh, etc.):

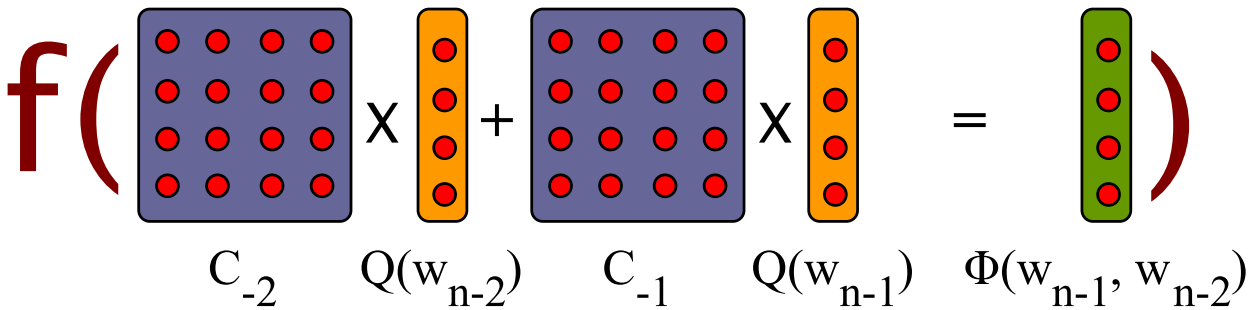


$$p(w_n|w_{n-1}, w_{n-2}) \propto \exp\left[R(w_n)^\top f\left(C^1 Q(w_{n-1}) + C^2 Q(w_{n-2})\right) + b_{w_n}\right]$$

This is a neural language model!

Adding non-linearities: the neural language model

Replacing the simple bi-linear relationship between context and output words with a more powerful non-linear function $f(\cdot)$ (logistic sigmoid, tanh, etc.):



if $f =$ the element wise logistic sigmoid $\sigma(\cdot)$:

$$\frac{\partial}{\partial \phi} E = \frac{\partial \sigma(\phi)}{\partial \phi} \circ \frac{\partial E}{\partial \sigma(\phi)}$$

$$= \sigma(\phi)(1 - \sigma(\phi)) \circ \left[\sum_j p(j|w_{n-1}, w_{n-2}) R(w_j) - R(w_n) \right]$$

where $\circ$ is the element wise product.

Infinite context: a recurrent neural language model

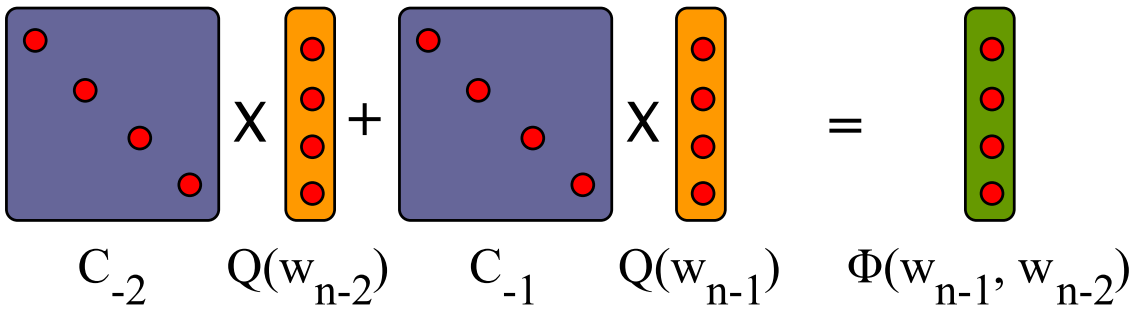
A recurrent LM drops the ngram assumption and directly approximate $p(w_n|w_{n-1}, \ldots, w_0)$ using a recurrent hidden layer:

$$\phi^n = f\left(Cf(\phi^{n-1}) + WQ(w_{n-1})\right)$$

$$p(w_n|w_{n-1}, \ldots, w_0) \propto \exp\left[R(w_n)^\top f(\phi^n) + b_{w_n}\right]$$

Simple RNNs like this are not actually terribly effective models. More compelling results are obtained with complex hidden units (e.g. Long Short Term Memory (LSTM), Clockwork RNNs, etc.), or by making the recurrent transformation C conditional on the last output.

Efficiency



For large D, calculating the context vector-matrix products is costly. Diagonal context transformation matrices (C_x) solve this and result in little performance loss.

Efficiency

Most of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating $R^{\mathsf{T}}\phi$.

Efficiency

Most of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating $R^{\mathsf{T}}\phi$.

Solutions

Short-lists: use the neural LM for the most frequent words, and a vanilla ngram LM for the rest. While this is easy to implement, it nullifies the neural LM's main advantage, i.e. generalisation to rare events.

Efficiency

Most of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating $R^{\mathsf{T}}\phi$.

Solutions

Approximate the gradient/change the objective: if we did not have to sum over the vocabulary to normalise during training it would be much faster. It is tempting to consider maximising likelihood by making the log partition function a separate parameter c, but this leads to an ill defined objective.

$$p_{\text{model}}(w_n|w_{n-1}, w_{n-2}, \theta) \equiv p_{\text{model}}^{\text{unnormalised}}(w_n|w_{n-1}, w_{n-2}, \theta) \times \exp(c)$$

Efficiency

Most of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating $R^{\mathsf{T}}\phi$.

Solutions

Approximate the gradient/change the objective: Mnih and Teh use noise contrastive estimation. This amounts to learning a binary classifier to distinguish data samples from (k) samples from a noise distribution (a unigram is a good choice):

$$p(\text{Data} = 1|w_n, w_{n-1}, \theta) = \frac{p_{\text{model}}(w_n|w_{n-1}, \theta)}{p_{\text{model}}(w_n|w_{n-1}, \theta) + kp_{\text{noise}}(w_n)}$$

Now parametrising the log partition function as c does not degenerate. This is very effective for speeding up training, but has no impact on testing time.[a]

[a] In practice fixing $c = 0$ is effective. It is tempting to believe that this noise contrastive objective justifies using unnormalised scores at test time. This is not the case and leads to high variance results.

Efficiency

Most of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating $R^{\mathsf{T}}\phi$.

Solutions

Factorise the output vocabulary: One level factorisation works well (Brown clustering is a good choice, frequency binning is not):

$$p(w_n|\phi) = p(\text{class}(w_n)|\phi) \times p(w_n|\text{class}(w_n), \phi),$$

where the function class$(\cdot)$ maps each word to one class. Assuming balanced classes, this gives a $\sqrt{|\text{vocab}|}$ speedup.

This renders properly normalised neural LMs fast enough to be directly integrated into an MT decoder.[a]

[a] Compositional Morphology for Word Representations and Language Modelling. Botha and Blunsom, ICML'14

Efficiency

Most of the computational cost of a neural LM is a function of the size of the vocabulary and is dominated by calculating $R^{\mathsf{T}}\phi$.

Solutions

Factorise the output vocabulary: By extending the factorisation to a binary tree (or code) we can get a $\log|\text{vocab}|$ speedup,[a] but choosing a tree is hard (frequency based Huffman coding is a poor choice):

$$p(w_n|\phi) = \prod_i p(d_i|r_i, \phi),$$

where d_i is i^{th} digit in the code for word w_n, and r_i is the feature vector for the i^{th} node in the path corresponding to that code.

[a] A scalable hierarchical distributed language model. Mnih and Hinton, NIPS'09.

Comparison with vanilla n-gram LMs

Good

- Better generalisation on unseen ngrams, poorer on seen ngrams. Solution: direct (linear) ngram features mimicking original log-linear language model features.
- Simple NLMs are often an order magnitude smaller in memory footprint than their vanilla ngram cousins (though not if you use the linear features suggested above!).

Bad

- NLMs are not as effective for extrinsic tasks such as Machine Translation compared to Kneser-Ney models, even when their intrinsic perplexity is much lower.
- NLMs easily beat Kneser-Ney models on perplexity for small training sets ($<$100M), but the representation size must grow with the data to be competitive at a larger scale.

Learning better representations for rich morphology

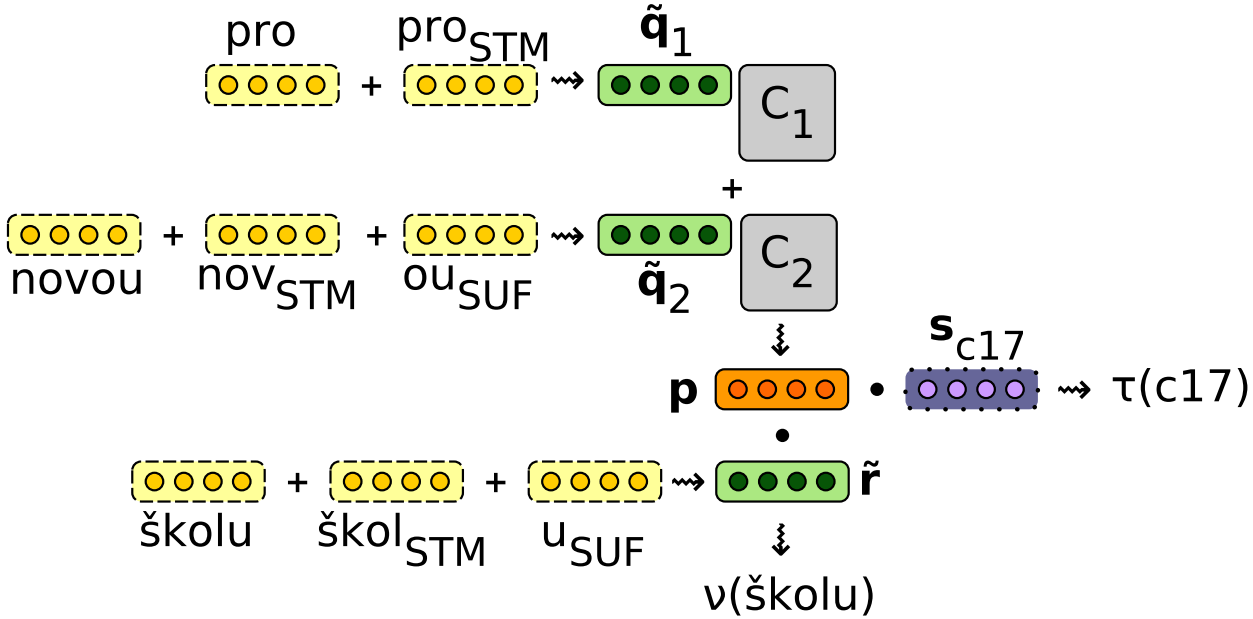
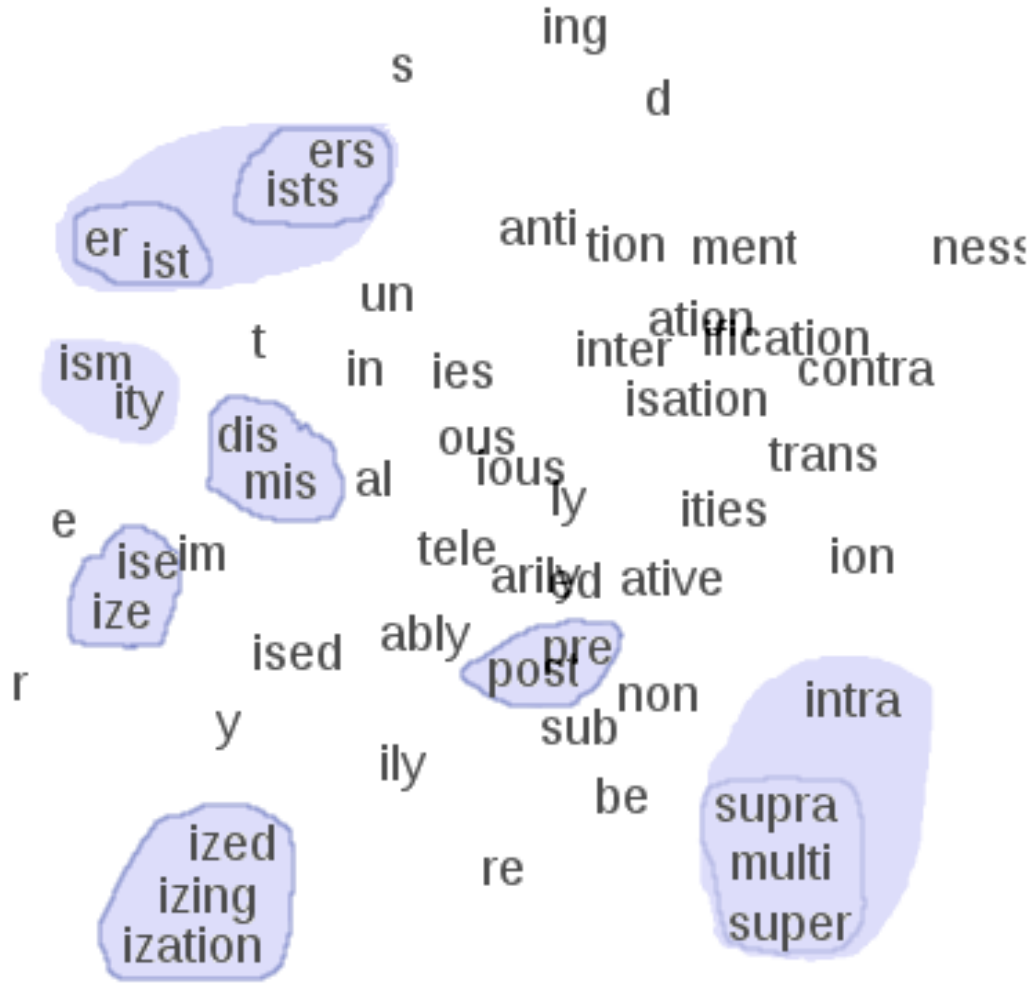


Illustration of how a 3-gram morphologically factored neural LM model treats the Czech phrase "pro novou školu" (for [the] new school).[2]

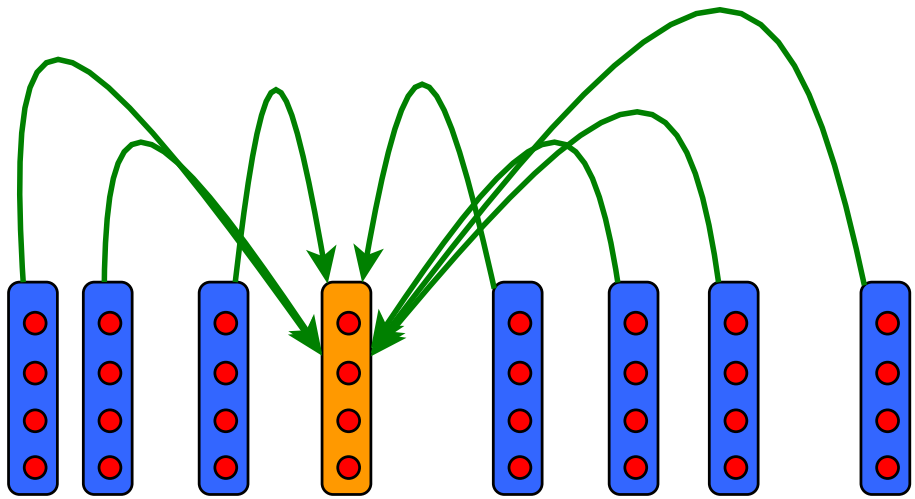
[2] Compositional Morphology for Word Representations and Language Modelling. Botha and Blunsom, ICML'14

Learning better representations for rich morphology



Learning representations directly

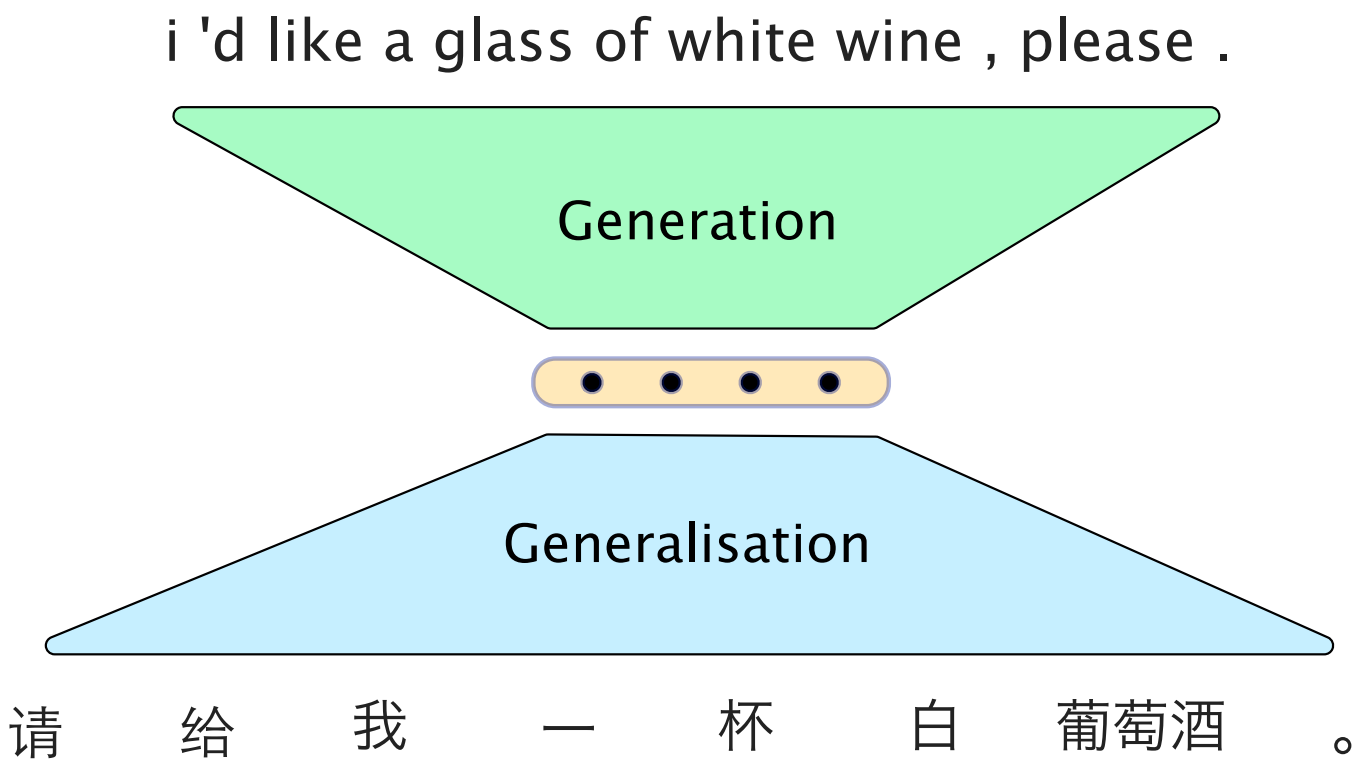
Collobert and Weston, Mikolov et al. `word2vec`, etc.



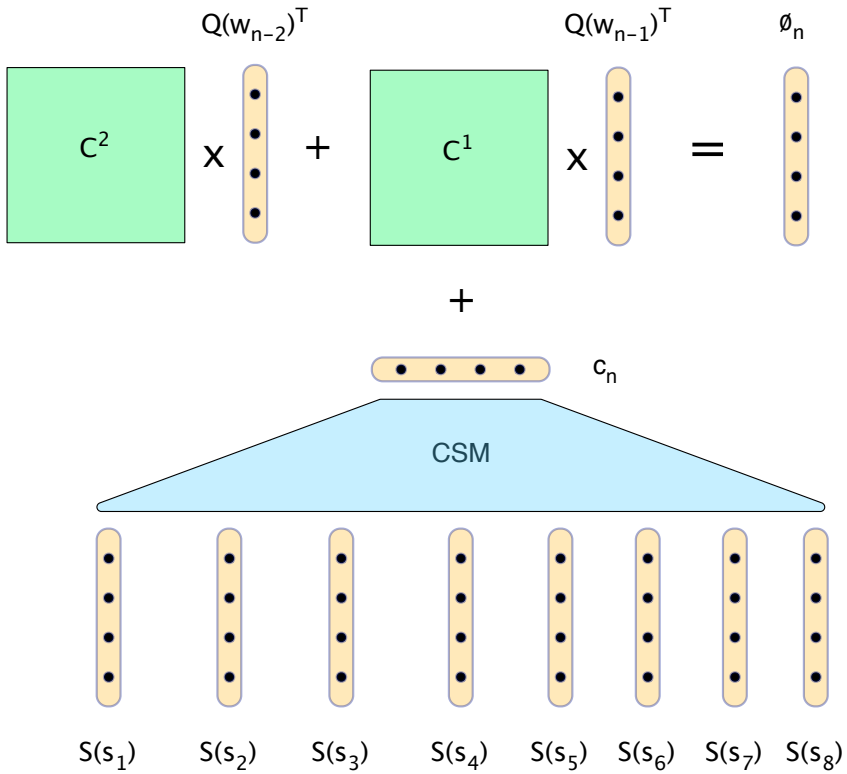
I got the shotgun. You got the briefcase.

If we do not care about language modelling, i.e. $p(\mathbf{w})$, and just want the word representations, we can condition on future context and/or use more efficient margin based objectives.

Conditional Generation



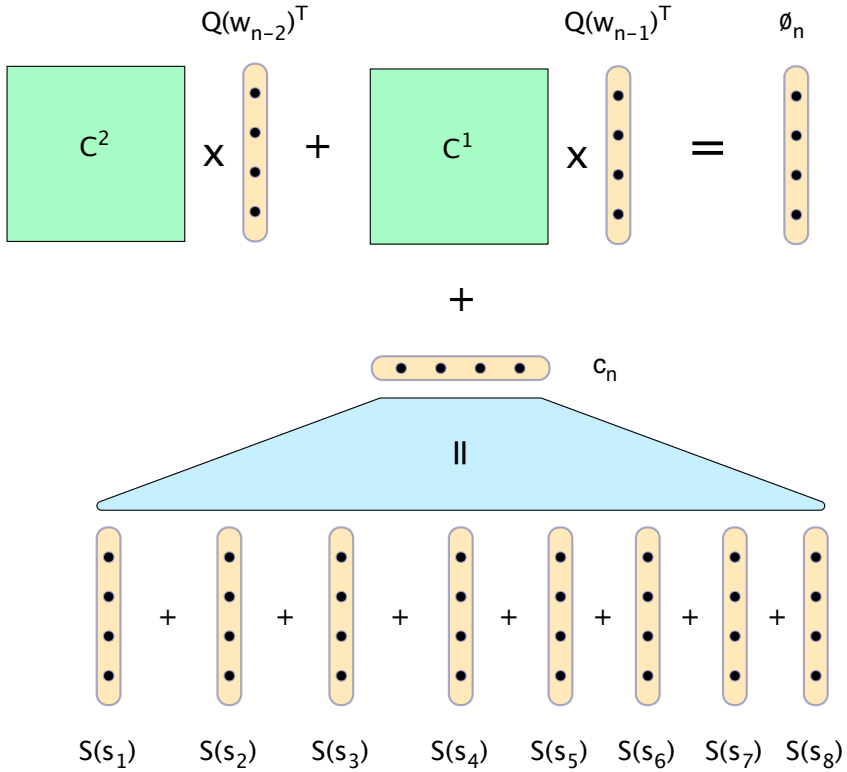
Conditional Generation



$$\phi_n = C^{-2}Q(w_{n-2}) + C^{-1}Q(w_{n-1}) + \texttt{CSM}(n, \mathbf{s})$$

$$p(w_n | w_{n-1}, w_{n-2}, \mathbf{s}) \propto \exp\left(R(w_n)^T \sigma(\phi_n) + b_{w_n}\right)$$

Conditional Generation: A naive additive model



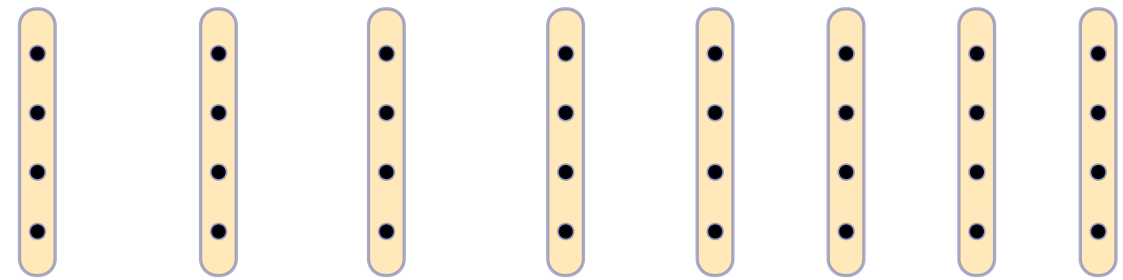
$$p_n = C^{-2}Q(w_{n-2}) + C^{-1}Q(w_{n-1}) + \sum_{j=1}^{|\mathbf{s}|} S(s_j)$$

$$p(w_n|w_{n-1}, w_{n-2}, \mathbf{s}) \propto \exp\left(R(w_n)^T \sigma(\phi_n) + b_{w_n}\right)$$

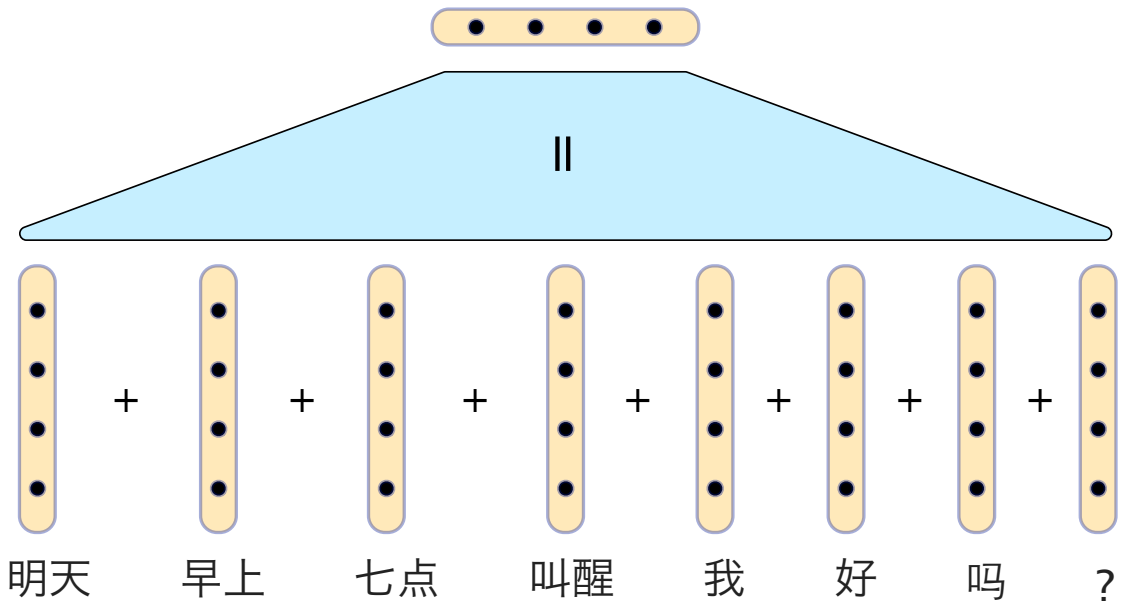
Conditional Generation: A naive additive model

明天 早上 七点 叫醒 我 好 吗 ?

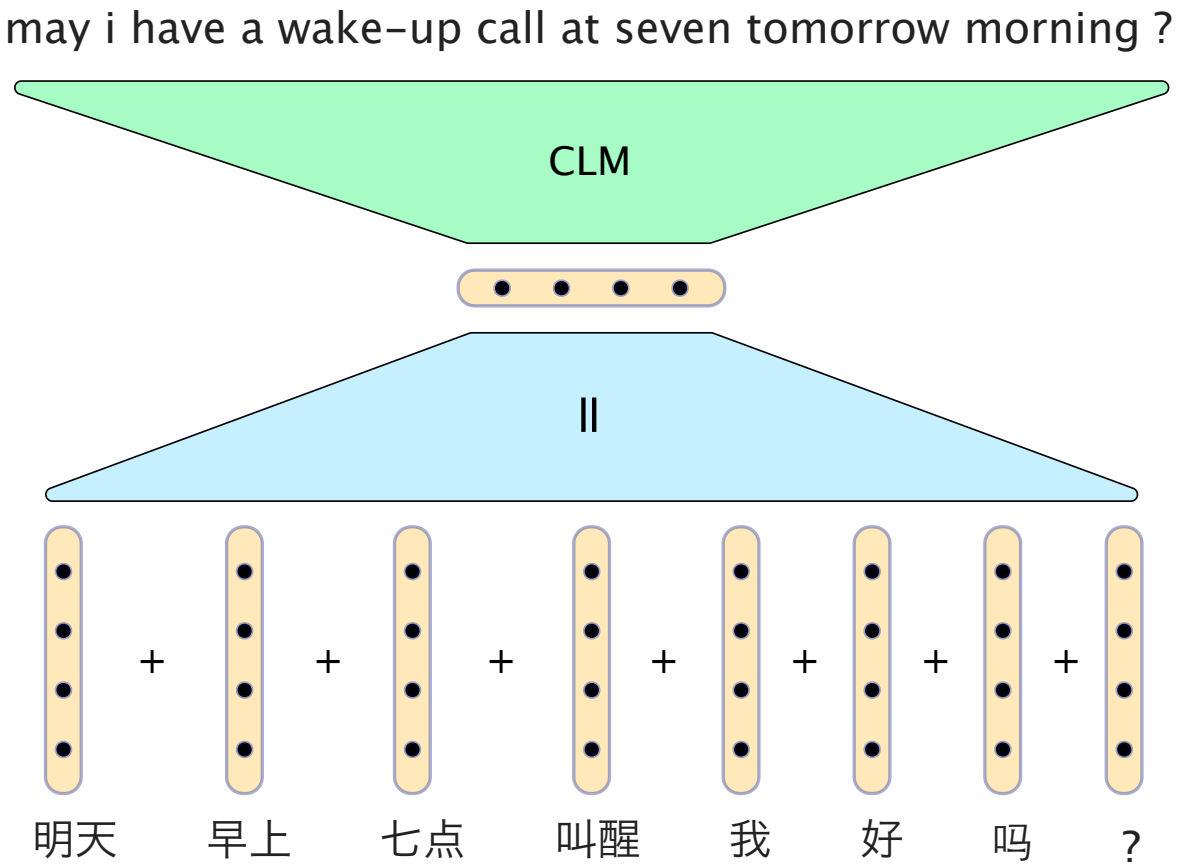
Conditional Generation: A naive additive model



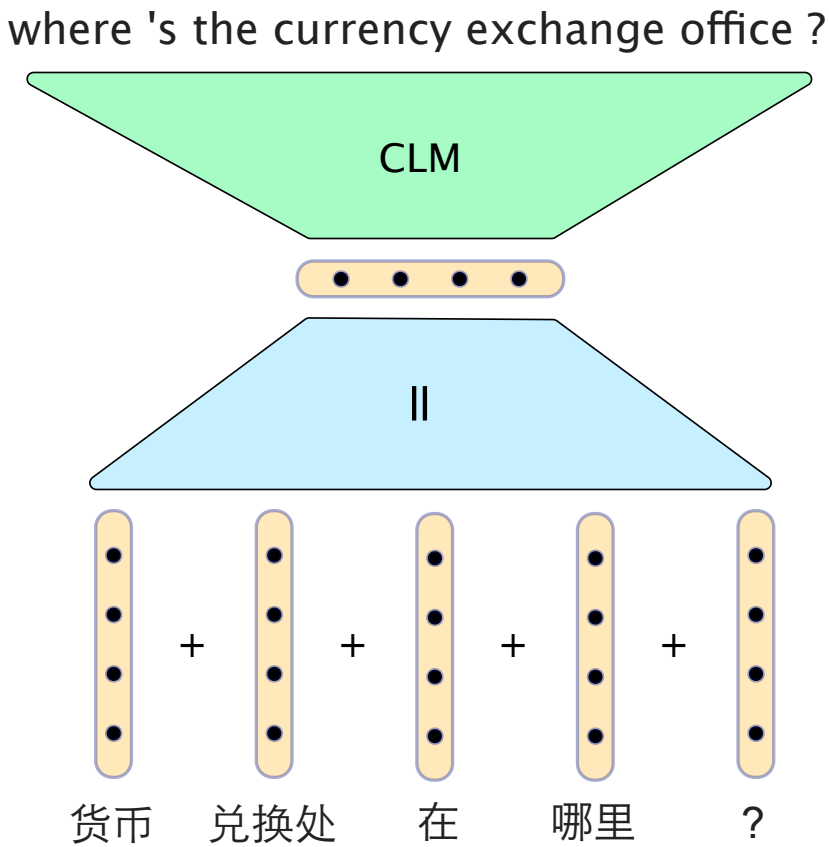
Conditional Generation: A naive additive model



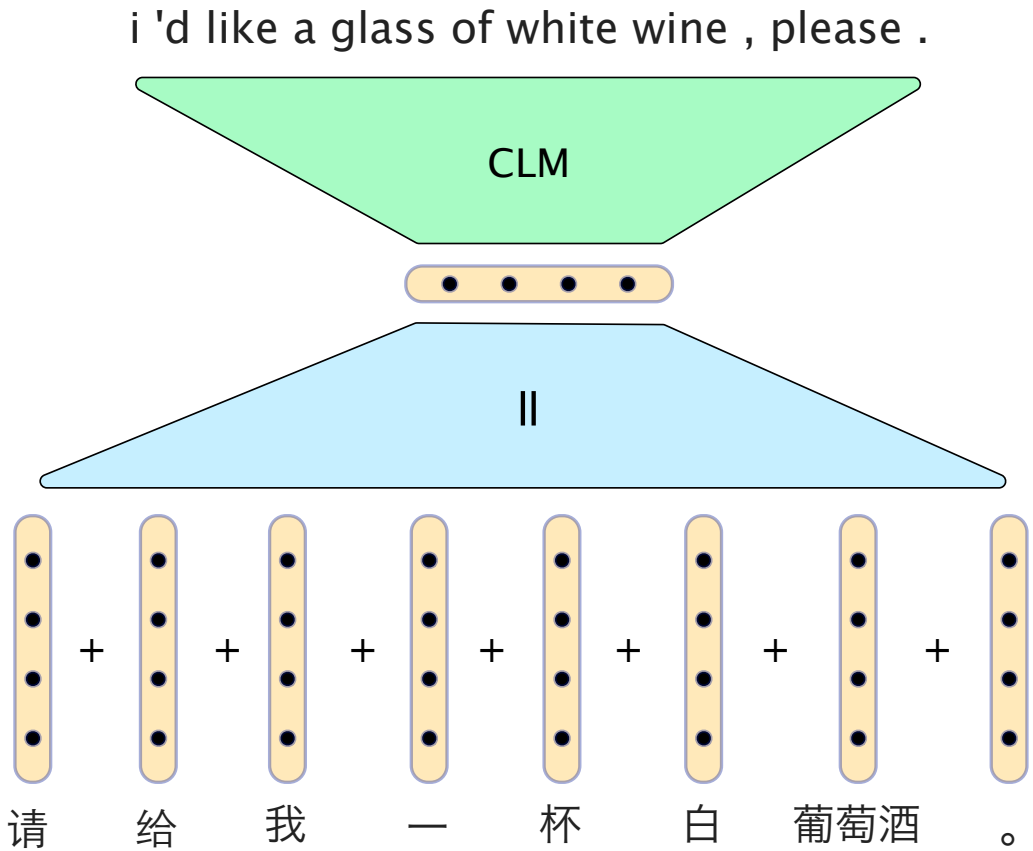
Conditional Generation: A naive additive model



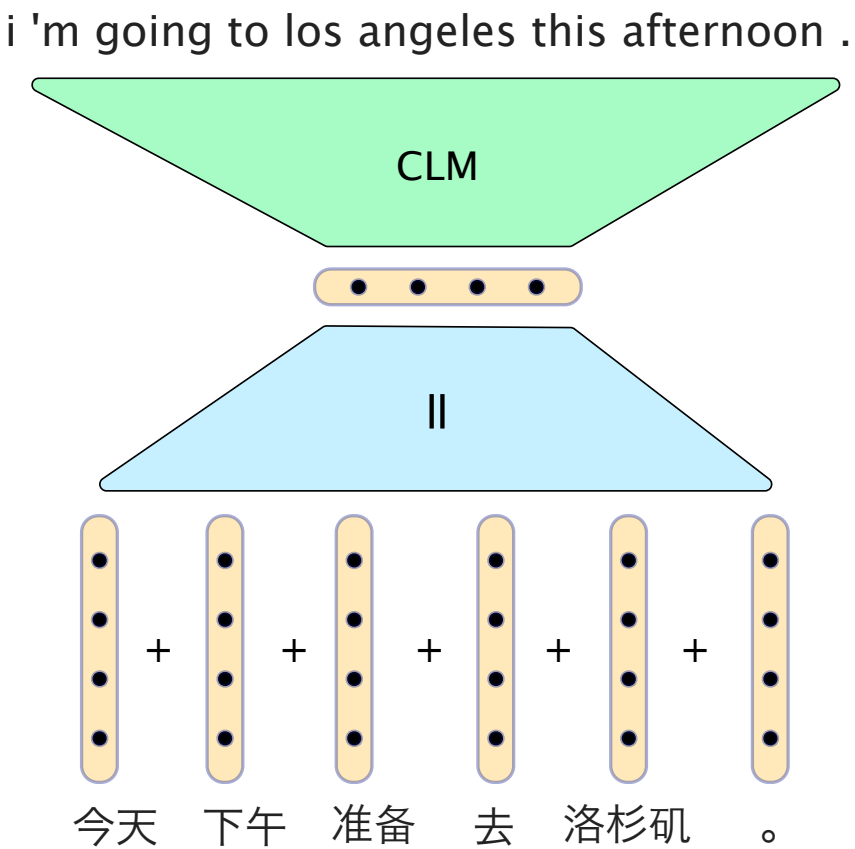
Conditional Generation: A naive additive model



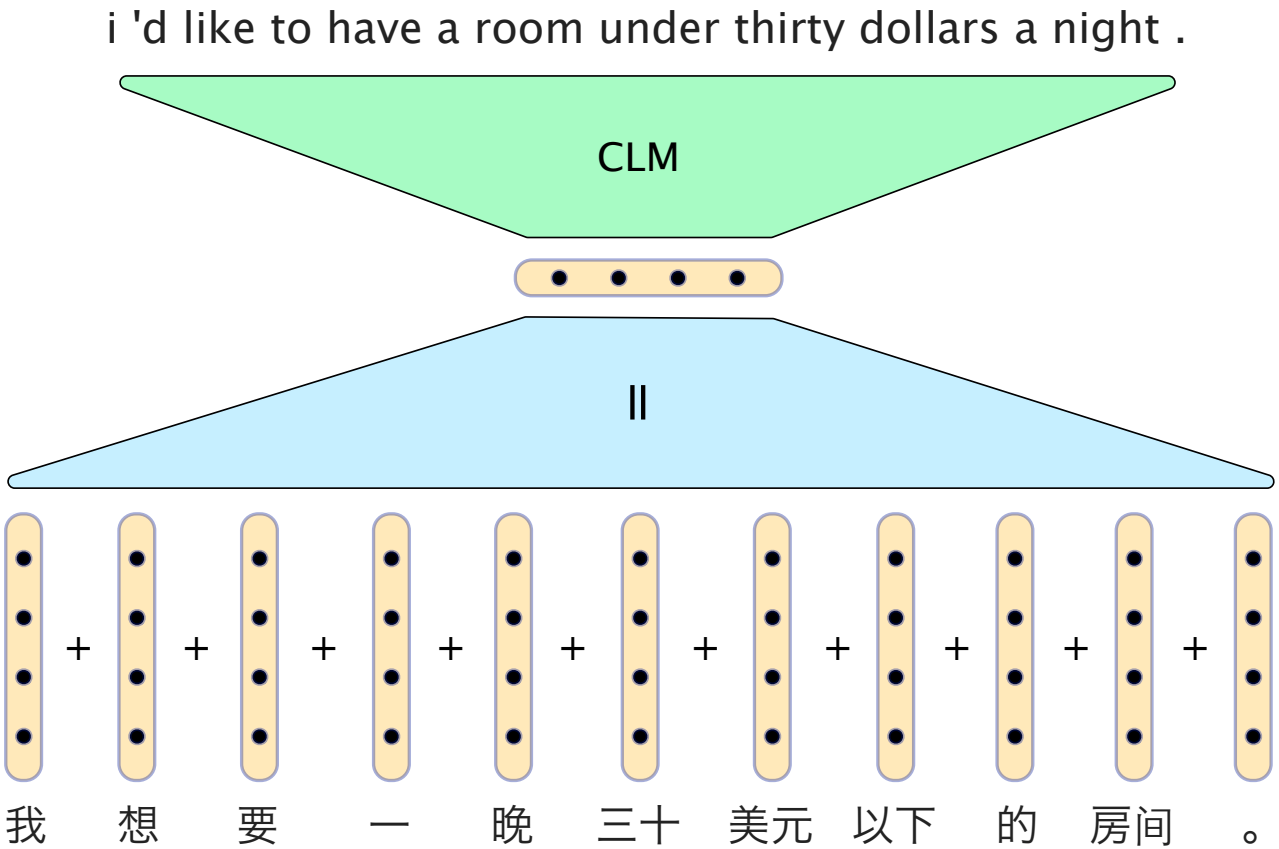
Conditional Generation: A naive additive model



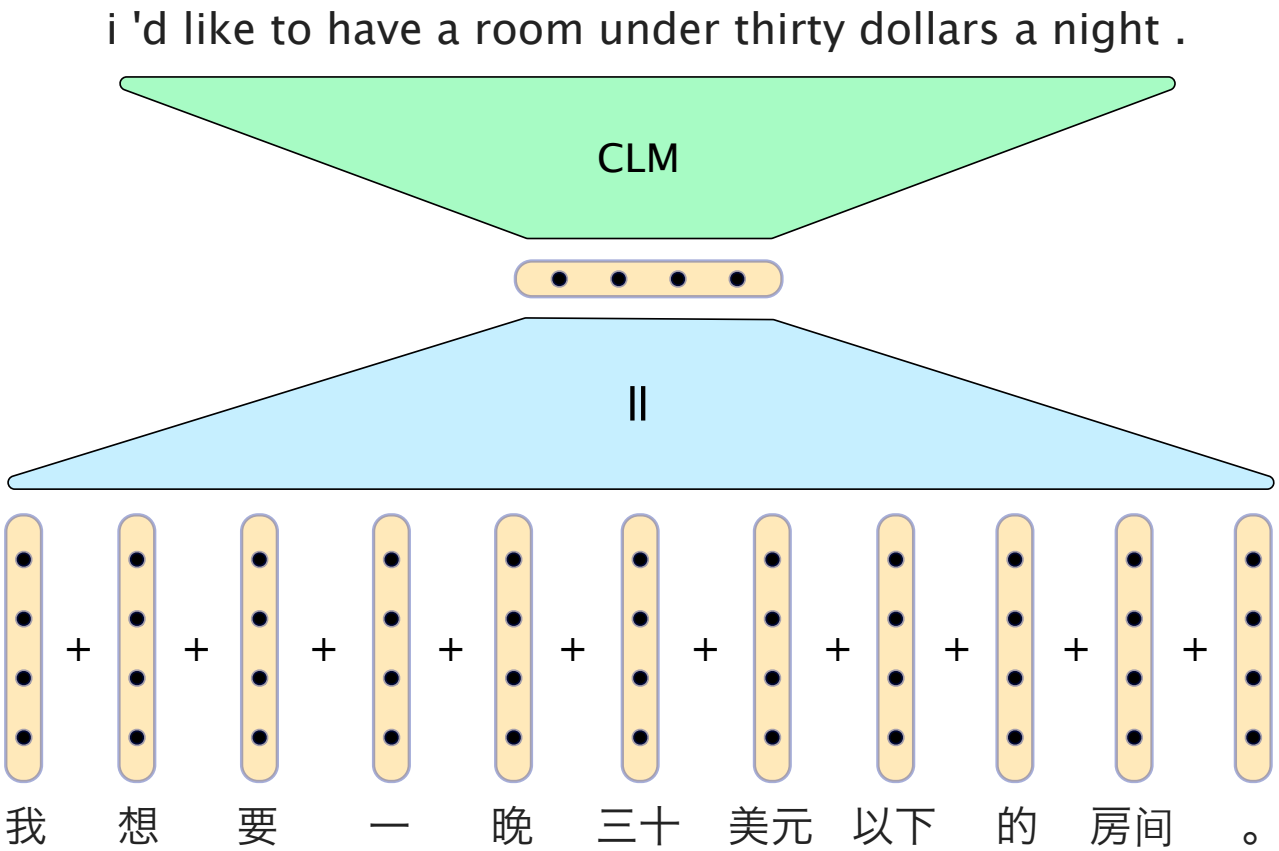
Conditional Generation: A naive additive model



Conditional Generation: A naive additive model



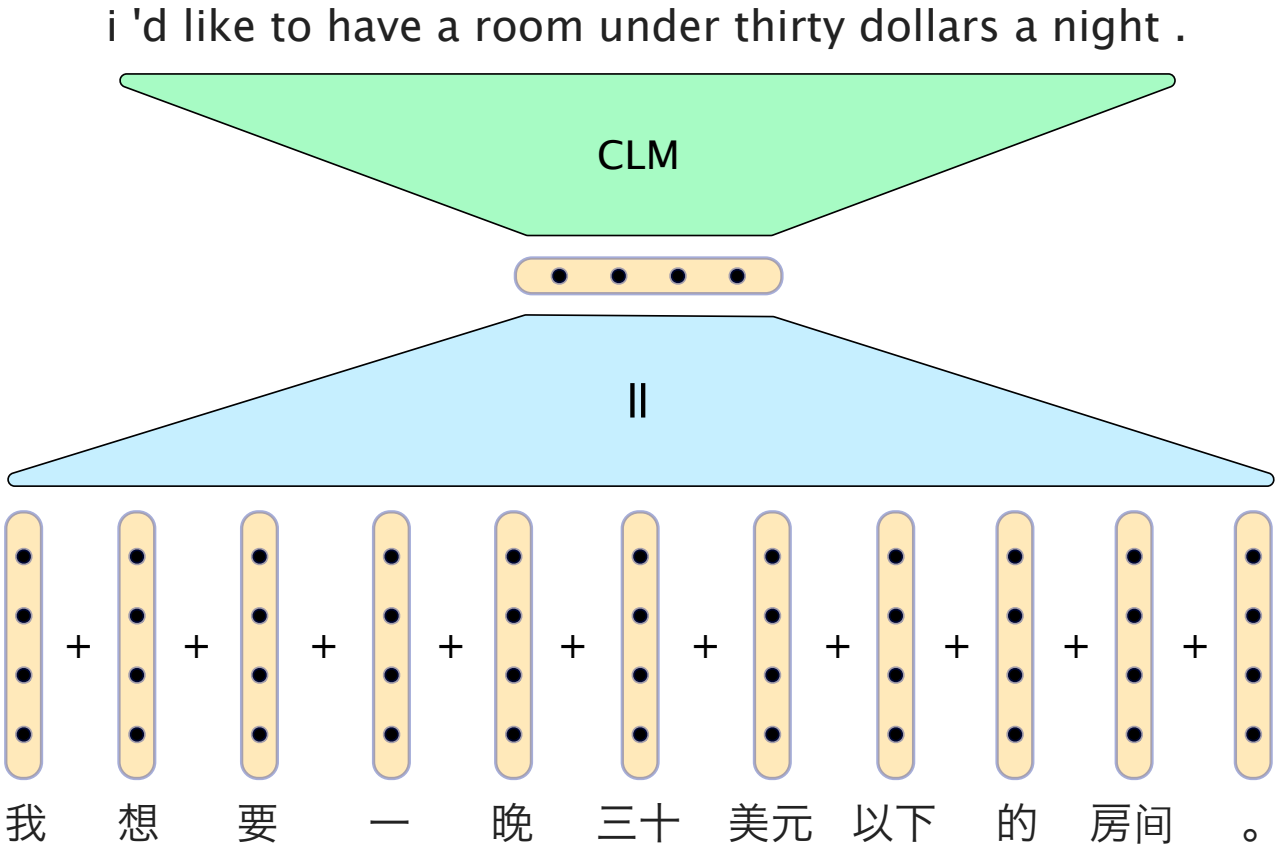
Conditional Generation: A naive additive model



Rough Gloss

I would like a night thirty dollars under room.

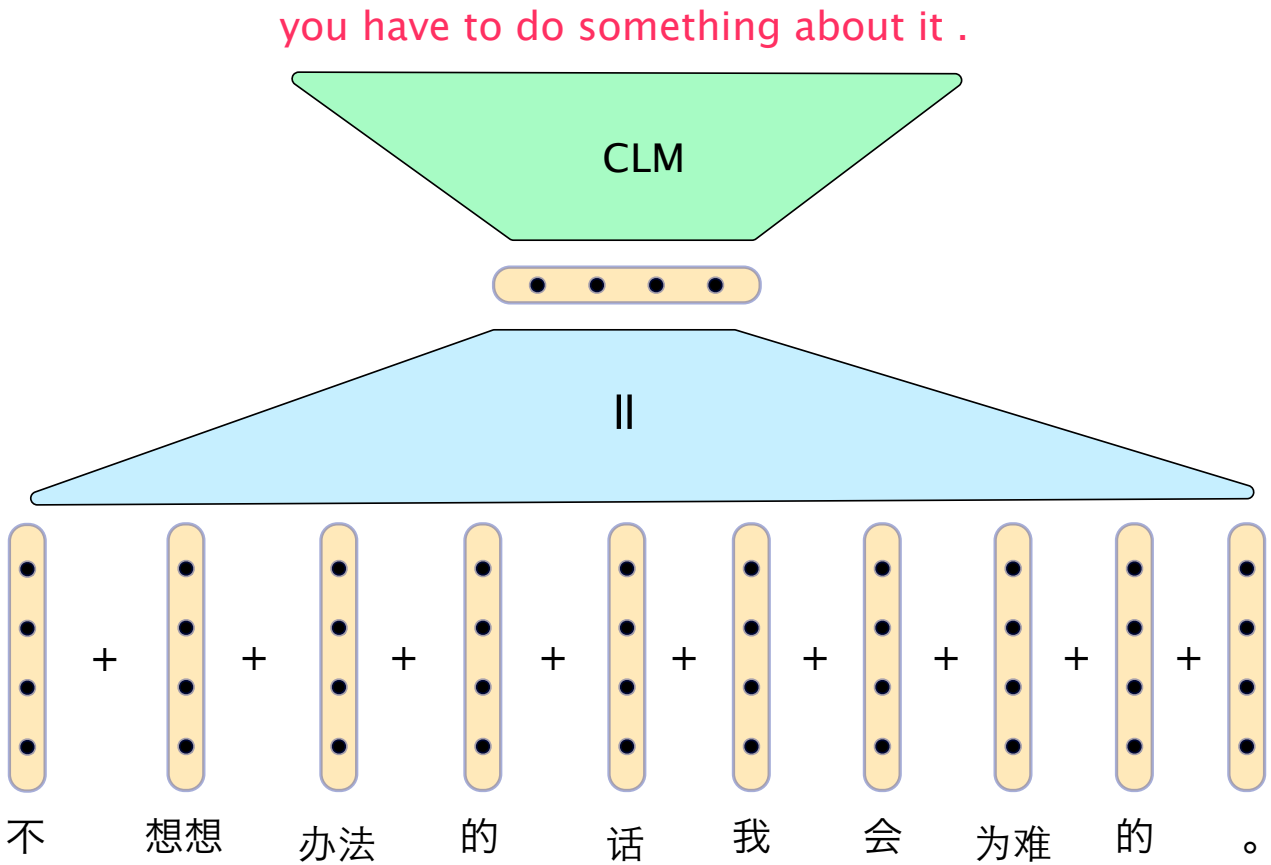
Conditional Generation: A naive additive model



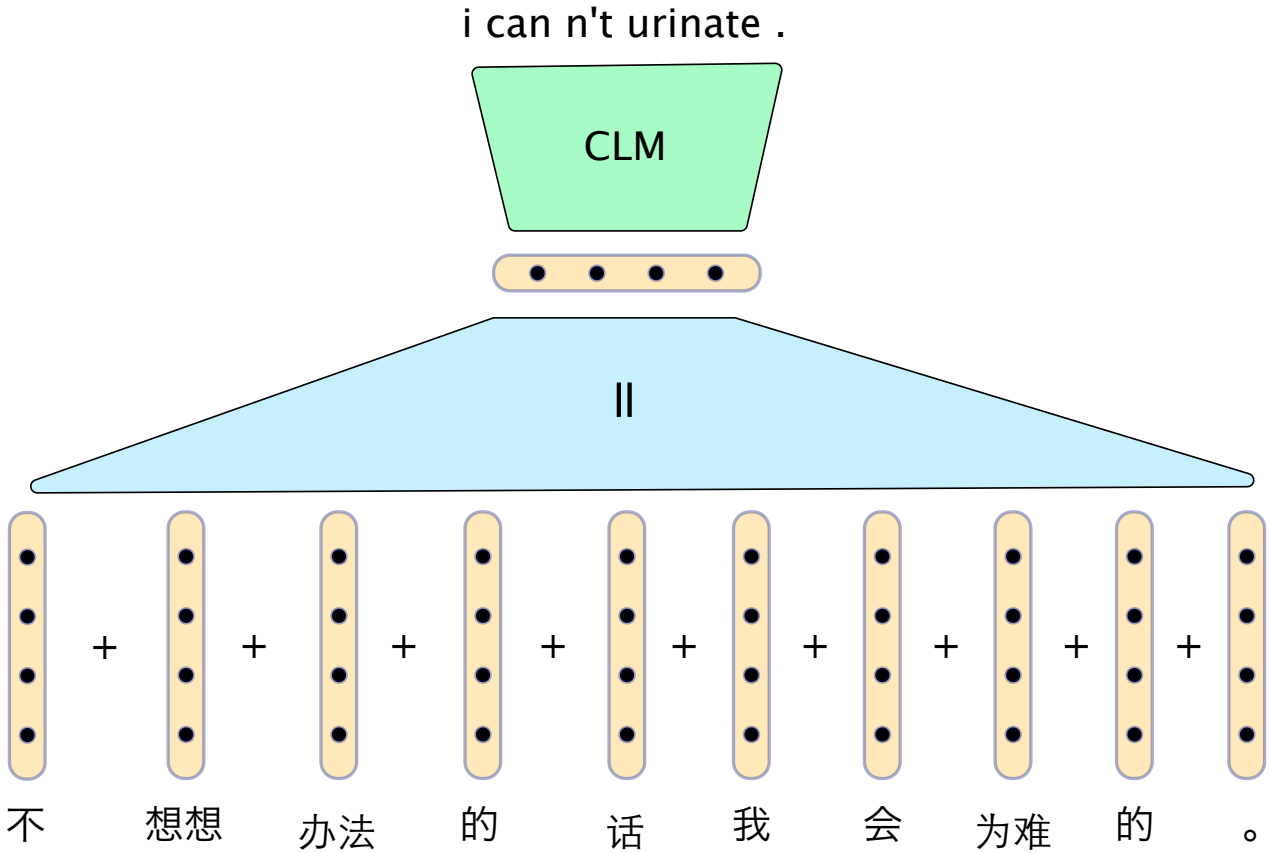
Google Translate

I want a late thirties under $'s room.

Conditional Generation: A naive additive model



Conditional Generation: A naive additive model



Conditional neural LMs and MT and beyond

Such conditional neural language models are now being exploited in MT and other multi-modal generation problems:

Recurrent Continuous Translation Models.
Kalchbrenner and Blunsom, EMNLP'13.

Joint Language and Translation Modeling with Recurrent Neural Networks.
Auli et al., EMNLP'13.

Fast and Robust Neural Network Joint Models for Statistical Machine Translation.
Devlin et al., ACL'14.

Multimodal Neural Language Models.
Kiros et al., ICML'14.

Outline

1 Distributional Semantics

2 Neural Distributed Representations

3 Semantic Composition
- Motivation
- Models
- Training
- Application Nuggets

4 Last Words

A simple task

Q: Do two words (roughly) mean the same?
"Cat" $\equiv$ "Dog" ?

A: Use a distributional representation to find out.

Given a vector representation, we can calculate the similarity between two things using some distance metric (as discussed earlier).

A different task: paraphrase detection

Q: Do two sentences (roughly) mean the same?
"He enjoys Jazz music" $\equiv$ "He likes listening to Jazz" ?

A: Use a distributional representation to find out?

No

We cannot learn distributional features at the sentence level.

Why can't we extract distributional features?

Linguistic Creativity

We formulate and understand language by composing units (words/phrases), not memorising sentences.

Crucially: this is what allows us to understand sentences we've never observed/heard before.

Why can't we extract distributional features?

The curse of dimensionality

As the dimensionality of a representation increases, learning becomes less and less viable due to sparsity.

Dimensionality for collocation

- One entry per word: Size of dictionary (small)
- One entry per sentence: Number of possible sentences (infinite)

$\Rightarrow$ We need a different method for representing sentences

Why care about compositionality

Paraphrasing

"He enjoys Jazz music" $\equiv$ "He likes listening to Jazz" ?

Sentiment

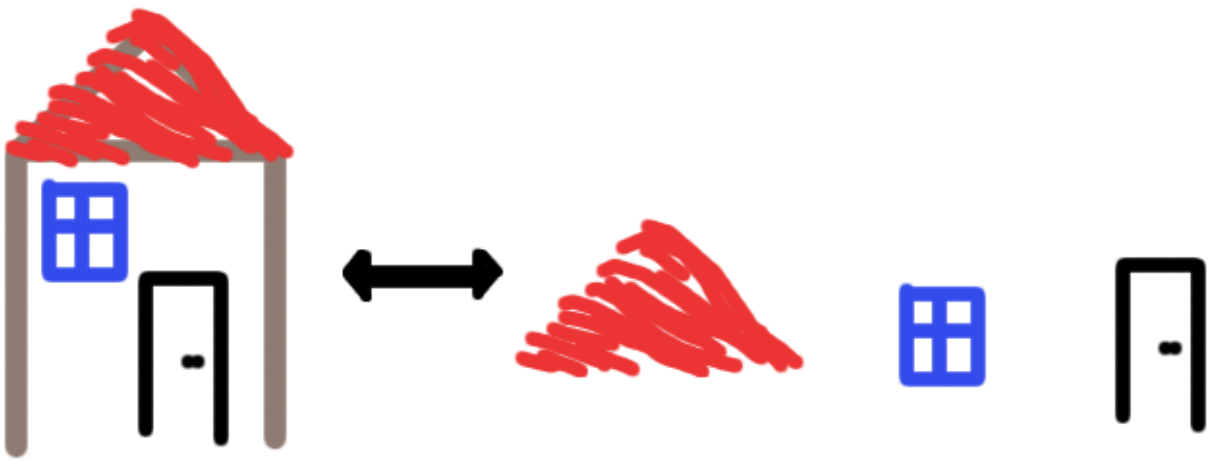
"This film was perfectly horrible" (good;bad)

Translation

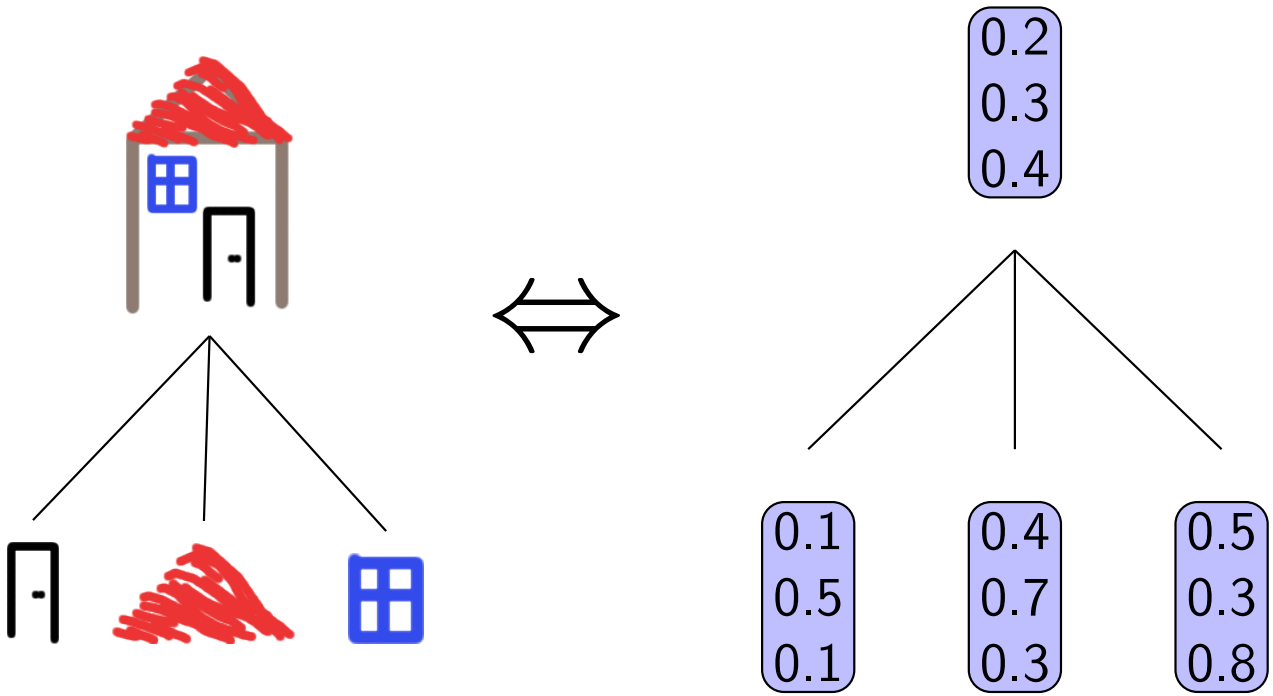
"Je ne veux pas travailler" $\equiv$ "I do not want to work" ?

Semantic Composition

Learning a hierarchy of features, where higher levels of abstraction are derived from lower levels.



A door, a roof, a window: It's a house



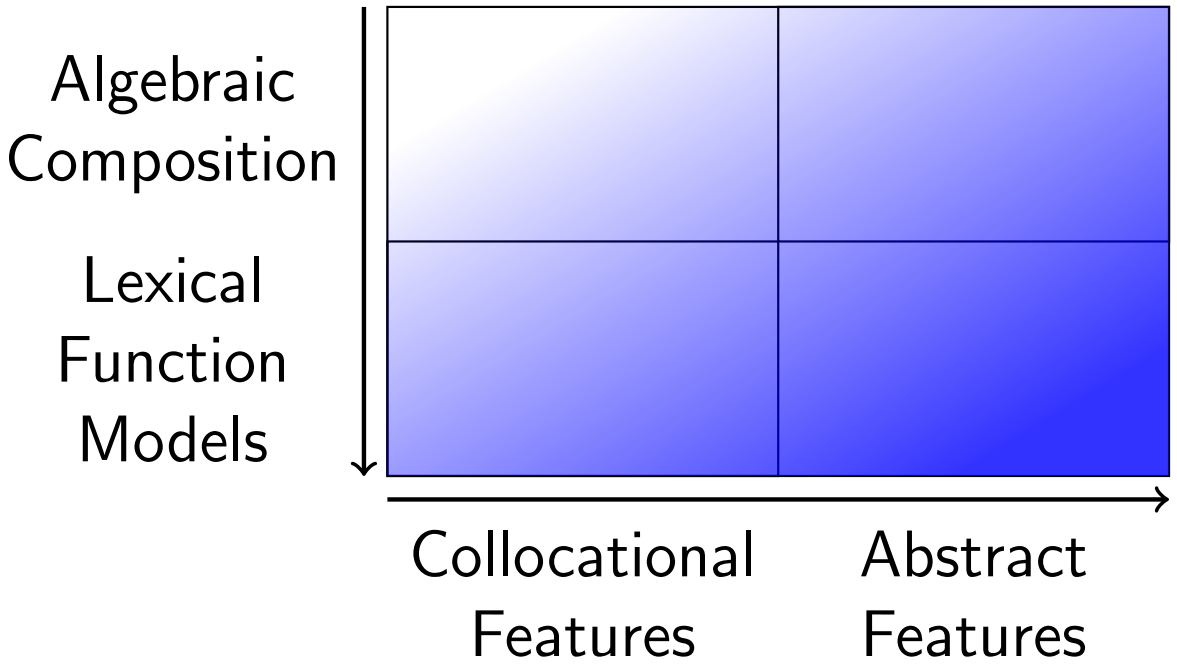
A "generic" composition function

$$p = f(u, v, R, K)$$

Where u, v are the child representations, R the relational information and K the background knowledge. Most composition models can be expressed as some such function f.

$\Rightarrow$ We may also want to consider the action of sentence-, paragraph-, or document-level context on composition.

Composition



Requirements

Not commutative	Mary likes John $\neq$ John likes Mary
Encode its parts?	Magic carpet $\equiv$ Magic + Carpet
More than parts?	Memory lane $\neq$ Memory + Lane

Algebraic vector composition

We take the full composition function ...

$$p = f(u, v, R, K)$$

... and simplify it as follows.

$$p = f(u, v)$$

- Simple mechanisms for composing vectors
- Works well on some tasks
- Large choice in composition functions[a]
  - Addition
  - Multiplication
  - Dilation
  - ...

[a] Composition in Distributional Models of Semantics. Mitchell and Lapata, Cognitive Science 2010

... and simplify it as follows.

$$p = f(u, v)$$

But it's broken

This simplification fails to capture important aspects such as

- Grammatical Relations
- Word order
- Ambiguity
- Context
- Quantifier Scope

Lexical function models

One solution: lexicalise composition.

- Different syntactic patterns indicate difference in composition function.
- Some words modify others to form compounds (e.g. adjectives).
- Let's encode this at the lexical level!

Example: adjectives as lexical functions

$$p = f(red, house) = F_{red}(house)$$

Lexical function model example

Baroni and Zamparelli (2010)[3]

- Adjectives are parameter matrices (θ_{red}, θ_{furry}, etc.).
- Nouns are vectors (**house**, **dog**, etc.).
- Composition is simply **red house** $= \theta_{red} \times$ **house**.

Learning adjective matrices

1. Obtain vector $\mathbf{n}_j$ for each noun n_j in lexicon.
2. Collect adjective noun pairs (a_i, n_j) from corpus.
3. Obtain vector $\mathbf{h}_{ij}$ of each bigram $a_i n_j$.
4. The set of tuples $\{(\mathbf{n}_j, \mathbf{h}_{ij})\}_j$ is a *dataset* D_i for adj. a_i.
5. Learn matrix θ_i from D_i using linear regression.

[3] Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. Baroni and Zamparelli, EMNLP'10

Uses and evaluations for lexical function models

Lexical function models are generally applied to short phrases or particular types of compostion (e.g. noun compounds).

Related Tasks and Evaluations

Semantic plausibility Judge short phrases[a][b]

fire beam	fire glow
table show results	table express results

Morphology Learn composition for morphemes[c]

f(f(shame, less), ness)	shamelessness

Decomposition Extract words from a composed unit[d]

f_{decomp} (reasoning)	deductive thinking
f_{decomp} (f(black, tie))	cravatta nera

---

[a] Vector-based Models of Semantic Composition. Mitchell and Lapata, ACL'08

[b] Experimental support [...]. Grefenstette and Sadrzadeh, EMNLP'11

[c] Lazaridou et al, ACL'13; Botha and Blunsom ICML'14

[d] Andreas and Ghahramani, CVSC'13; Dinu and Baroni, ACL'14

Higher valency functions with tensors

How do we go from predicates (adjectives) to higher-valency relations (verbs, adverbs)?

- Matrices encode linear maps. Good for adjectives.
- What encodes multilinear maps? **Tensors**.
- An order-n tensor T_R represents a function R of $n-1$ arguments.
- Tensor contraction models function application.

For n-ary functions to order $n+1$ tensors

$$R(a, b, c) \Rightarrow ((T_R \times \mathbf{a}) \times \mathbf{b}) \times \mathbf{c}$$

I'm getting tensor every day

We like tensors...

- Encode multilinear maps.
- Nice algebraic properties.
- Learnable through regression[a]
- Decomposable/Factorisable.
- Capture k-way correlations between argument features and outputs.

[a] Grefenstette et al., IWCS'13

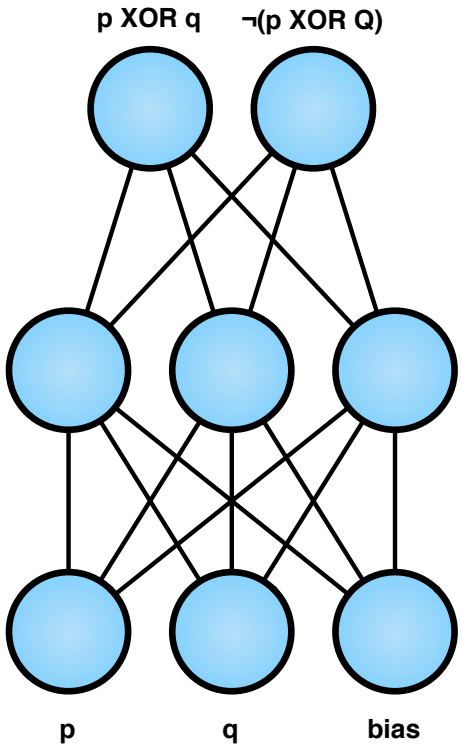
But...

- **Big** data structures (d^n elements).
- Hard to learn (curse of dimensionality).

Q: Can we learn k-way correlations without tensors?

A: Non-linearities + hidden layers!

For example:



- XOR not linearly separable in 2D space.
- Order-3 tensors can model any binary logical operation (Grefenstette 2013).
- Non-linearities and hidden layers offer compact alternative.

A lower-dimensional alternative

Having established nonlinear layers as a low-dimensinonal alternative to tensors, we can redefine semantic composition through some function such as

$$p = f(u, v, R, K) = g\left(W^u_{RK} u + W^v_{RK} v + b_{RK}\right),$$

where g is a nonlinearity, W are composition matrices and b a bias term.

A lower-dimensional alternative

Having established nonlinear layers as a low-dimensinonal alternative to tensors, we can redefine semantic composition through some function such as

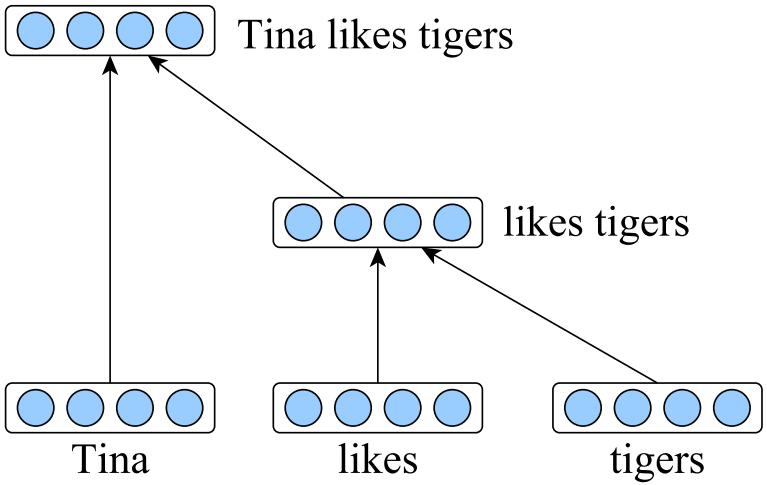
$$p = f(u, v, R, K) = g\left(W^u_{RK} u + W^v_{RK} v + b_{RK}\right),$$

where g is a nonlinearity, W are composition matrices and b a bias term.

Recursion

If W^u_{RK} and W^v_{RK} are square, this class of composition functions can be applied recursively.

Recursive Neural Networks



Composition function:
$f(u, v) = g\left(W(u\|v) + b\right)$

g is a non-linearity
$W \in \mathbb{R}^{n \times 2n}$ is a weight matrix
$b \in \mathbb{R}^n$ is a bias
$u, v \in \mathbb{R}^n$ are inputs

This is (almost) all you need

This is the definition of a simple recursive neural network.[a]
But key decisions are still open: how to parametrise, composition tree, training algorithm, which non-linearity etc.

[a] Pollack, '90; Goller and Küchler, '96; Socher et al., EMNLP'11; Scheible and Schütze, ICLR'13

Choices to make

Decisions, decisions

Tree structure left/right-branching, greedy based on error[a], based on parse[b], ...

Non-linearity [c] tanh, logistic sigmoid, rectified linear ...

Initialisation [d] zeros, Gaussian noise, identity matrices, ...

---

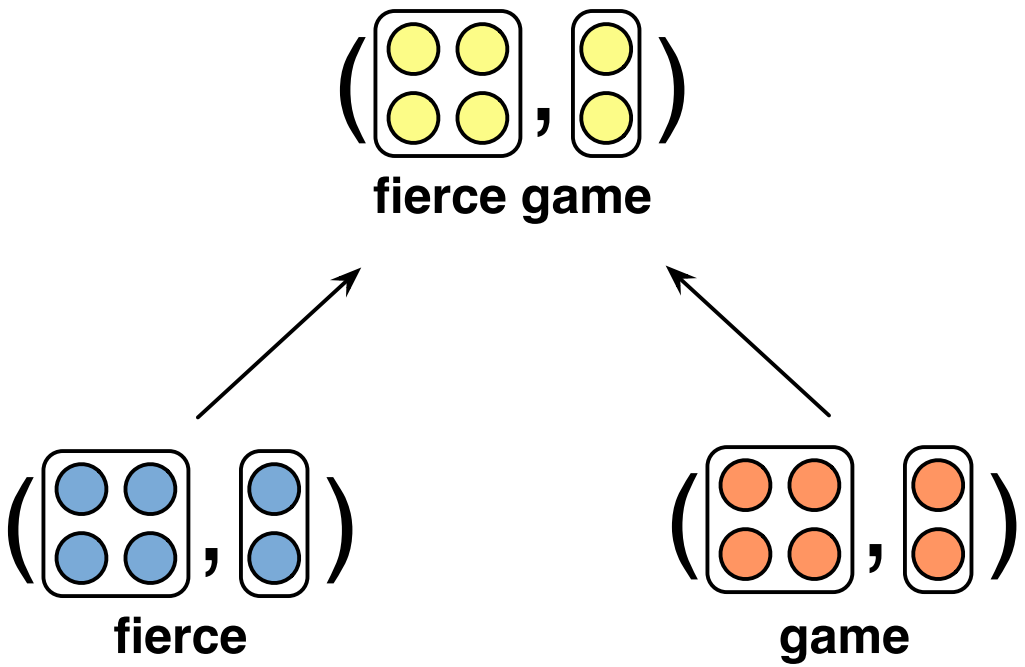
[a] Socher et al., EMNLP'11

[b] Hermann and Blunsom, ACL'13

[c] LeCun et al., Springer 1998

[d] Saxe et al., ICLR'14

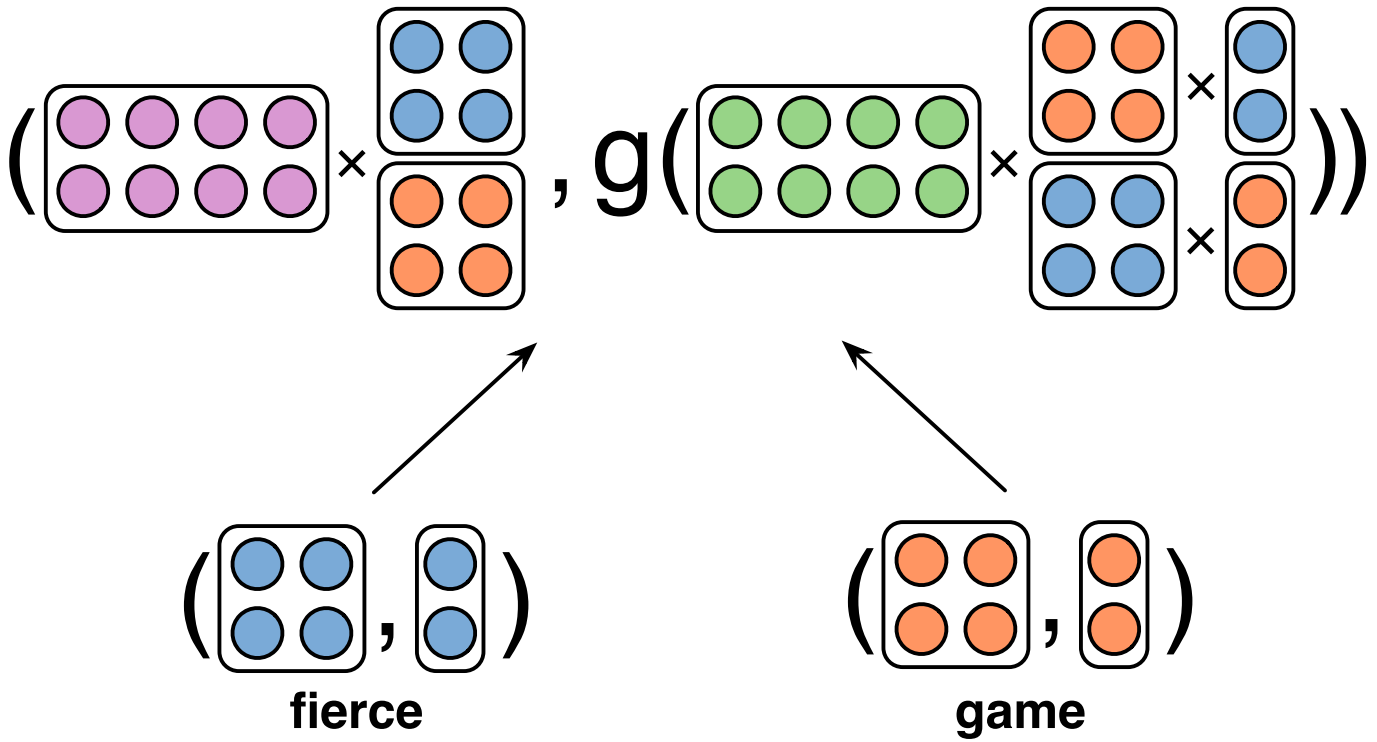
Alternative: Represent everything as both a vector and a matrix (Socher et al. (2012)).



This adds an element similar to the lexical function models discussed earlier.

Matrix-Vector Neural Networks

Alternative: Represent everything by both a vector and a matrix (Socher et al. (2012)).



This adds an element similar to the lexical function models discussed earlier.

Alternative: Represent everything by both a vector and a matrix (Socher et al. (2012)).

Formalizing MVRNNs

$$(C, c) = f(((A, a), (B, b)))$$

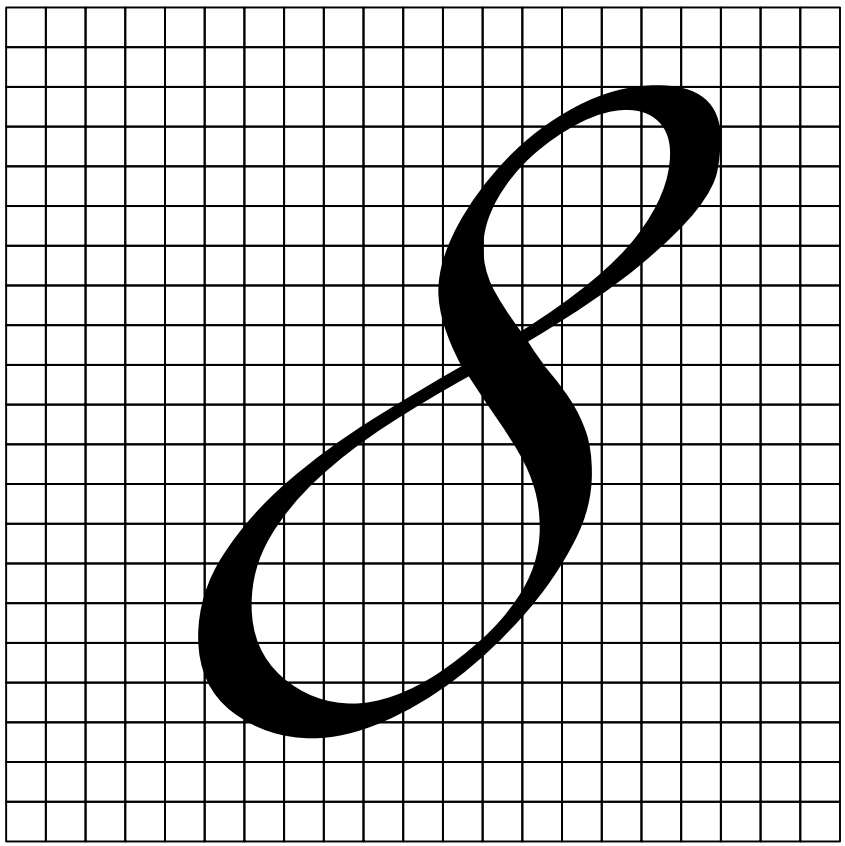
$$c = g(W \times \begin{bmatrix} Ba \\ Ab \end{bmatrix})$$

$$C = W_M \times \begin{bmatrix} A \\ B \end{bmatrix}$$

$$a, b, c \in R^d;\ A, B, C \in R^{d \times d};\ W, W_M \in R^{d \times 2d}$$

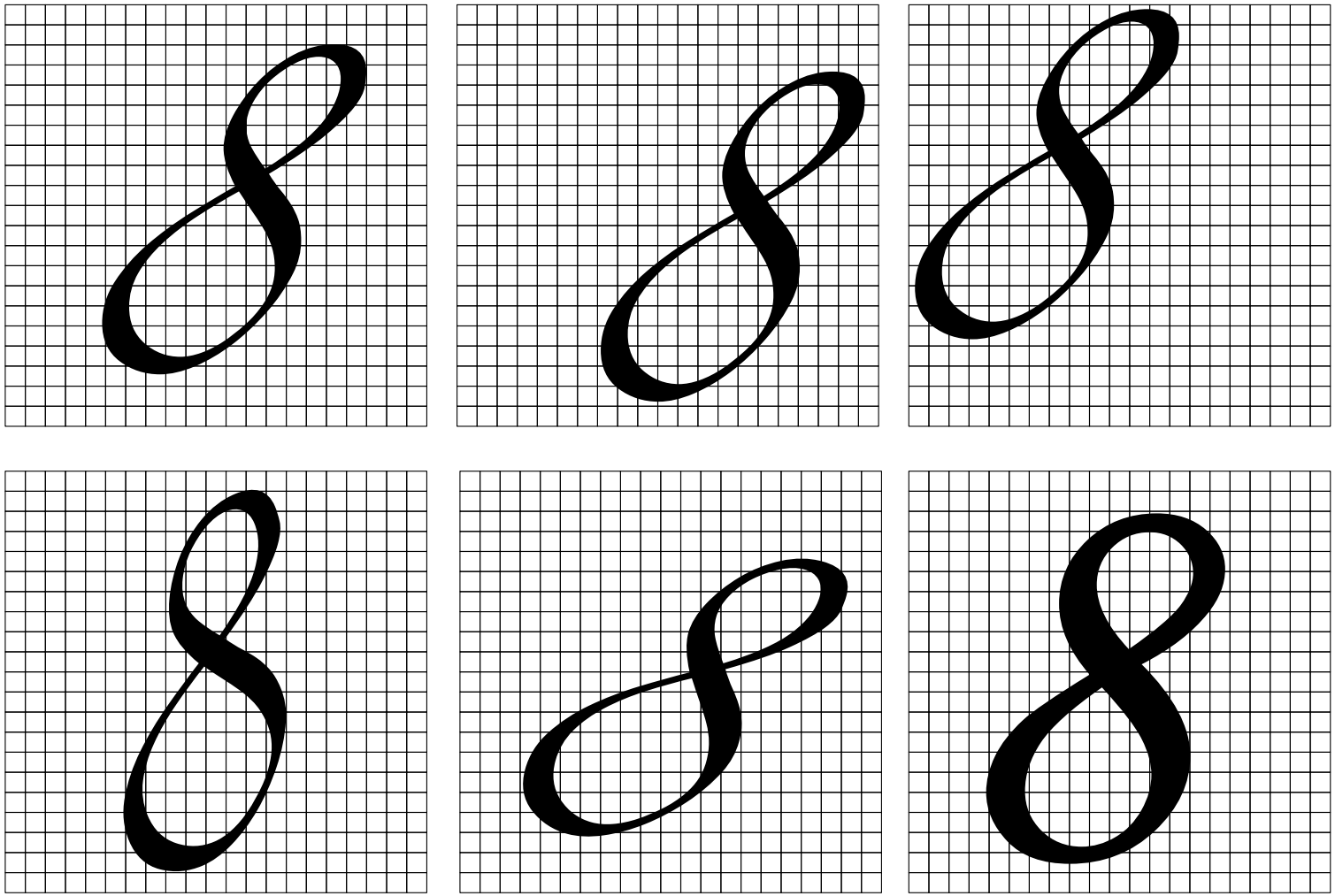
This adds an element similar to the lexical function models discussed earlier.

A step back: How do we learn to recognise pictures?
Will a fully connected neural network do the trick?

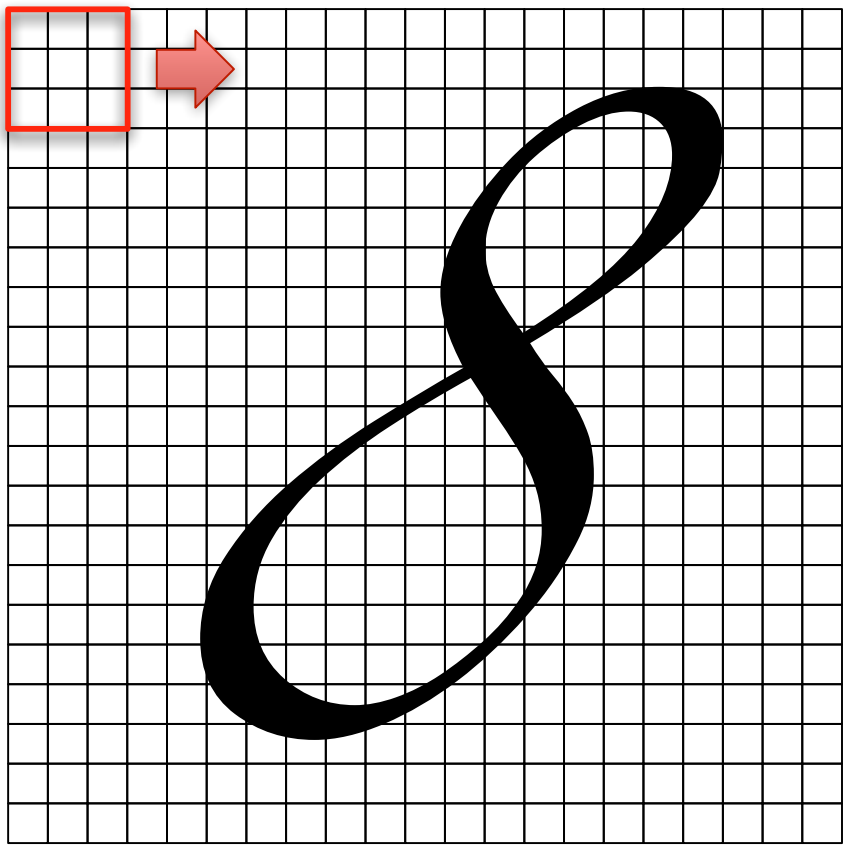


ConvNets for pictures

Problem: lots of variance that shouldn't matter (position, rotation, skew, difference in font/handwriting).

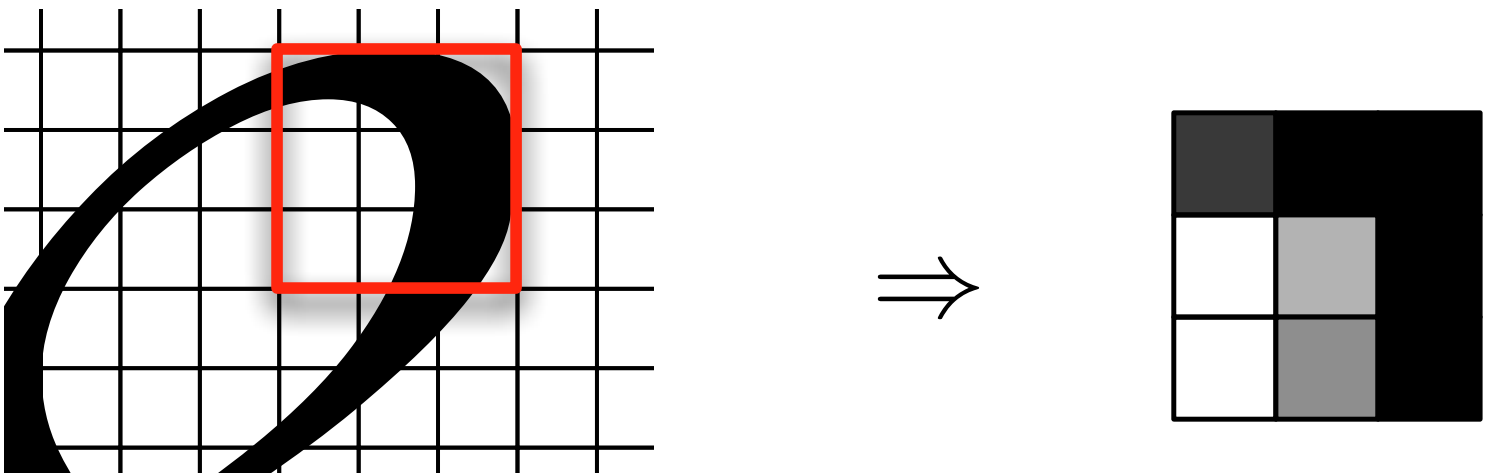


Solution: Accept that features are **local**. Search for local features with a window.



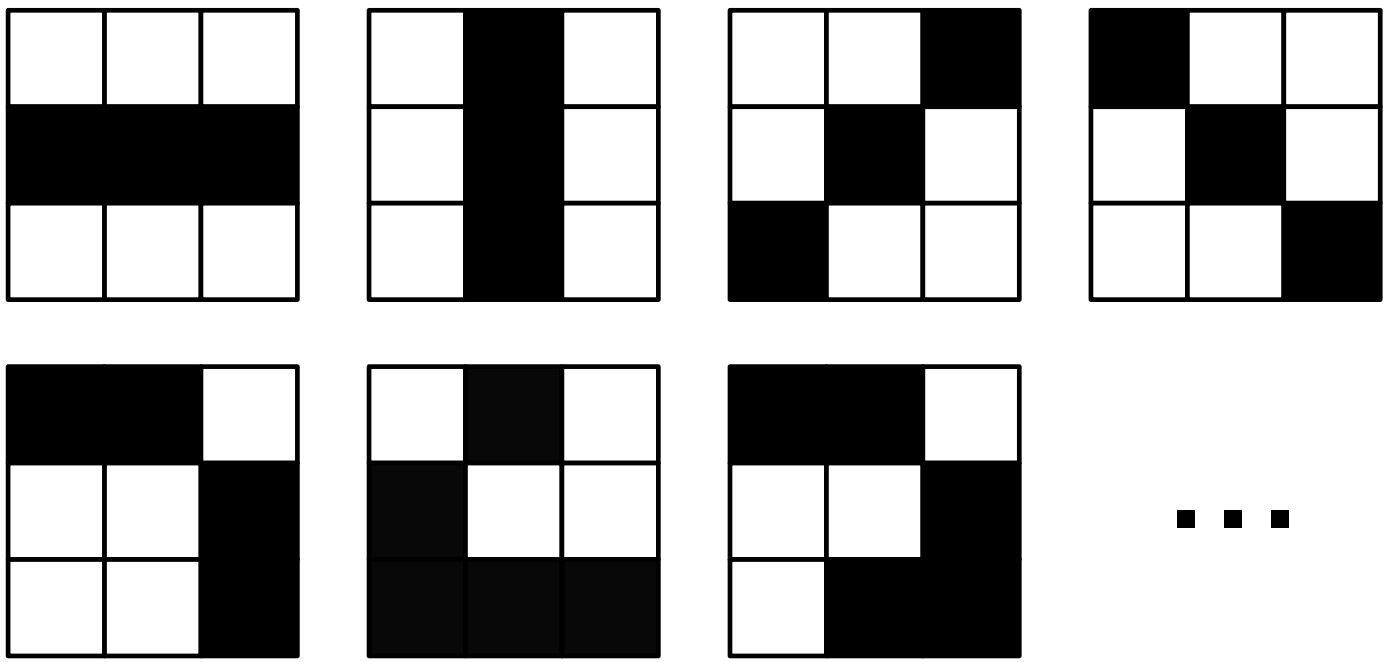
ConvNets for pictures

Convolutional window acts as a classifer for local features.



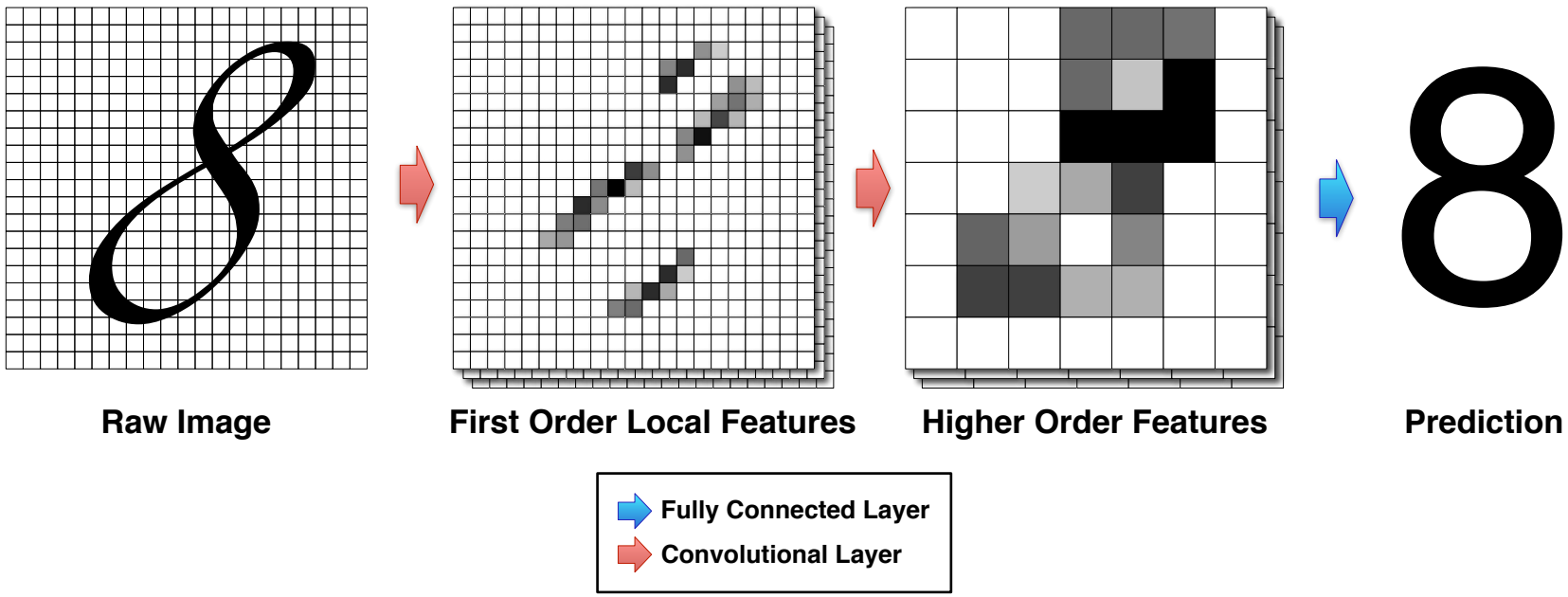
ConvNets for pictures

Different convolutional maps can be trained to recognise different features (e.g. edges, curves, serifs).



ConvNets for pictures

Stacked convolutional layers learn higher-level features.



One or more fully-connected layers learn classification function over highest level of representation.

ConvNets for language

Convolutional neural networks fit natural language well.

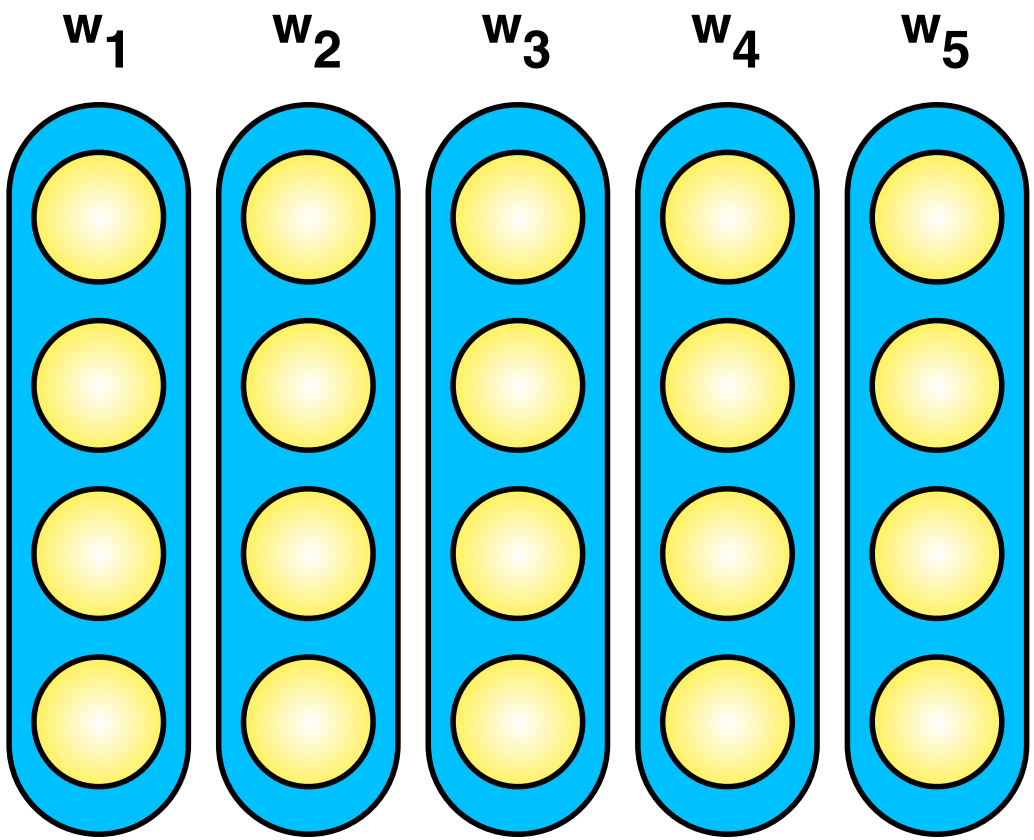
Deep ConvNets capture:

- Positional invariances
- Local features
- Hierarchical structure

Language has:

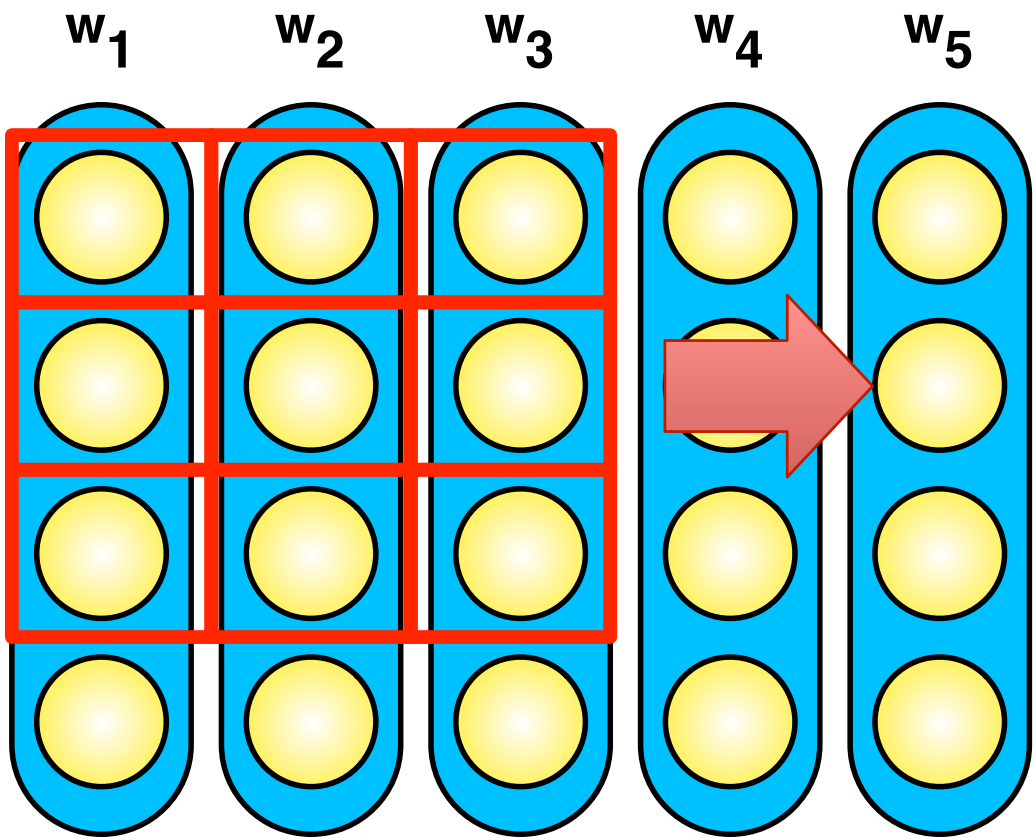
- Some positional invariance
- Local features (e.g. POS)
- Hierarchical structure (phrases, dependencies)

How do we go from images to sentences? Sentence matrices!

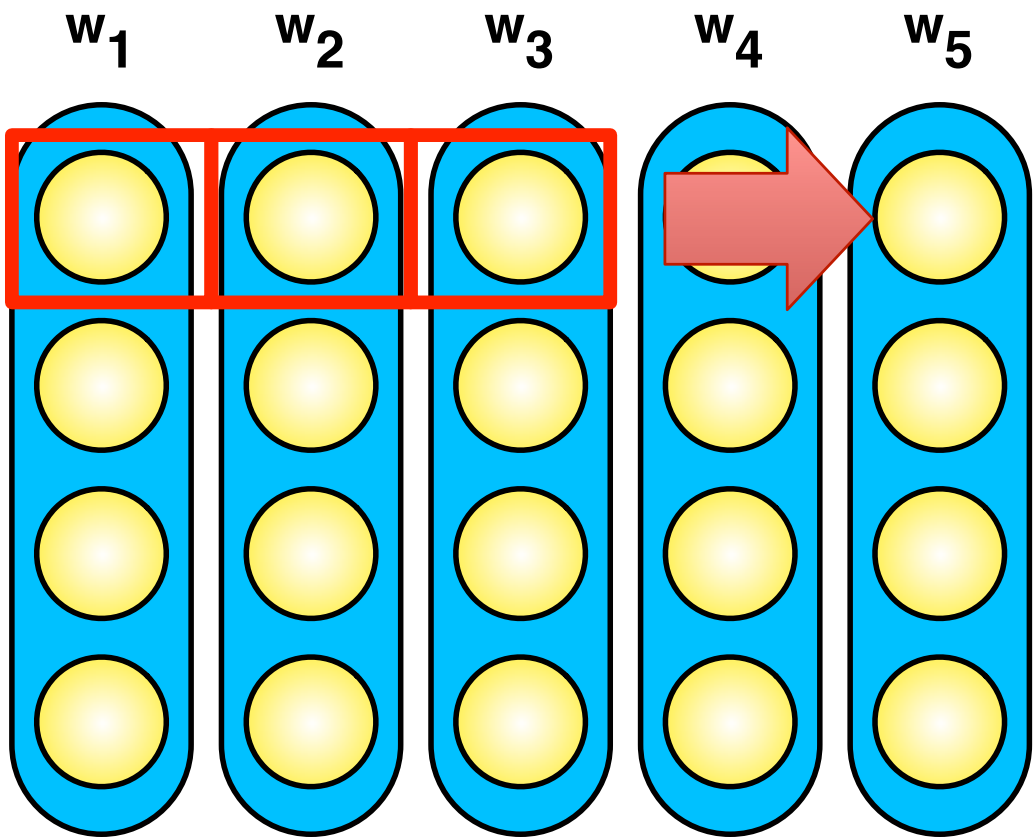


ConvNets for language

Does a convolutional window make sense for language?



A better solution: feature-specific windows.

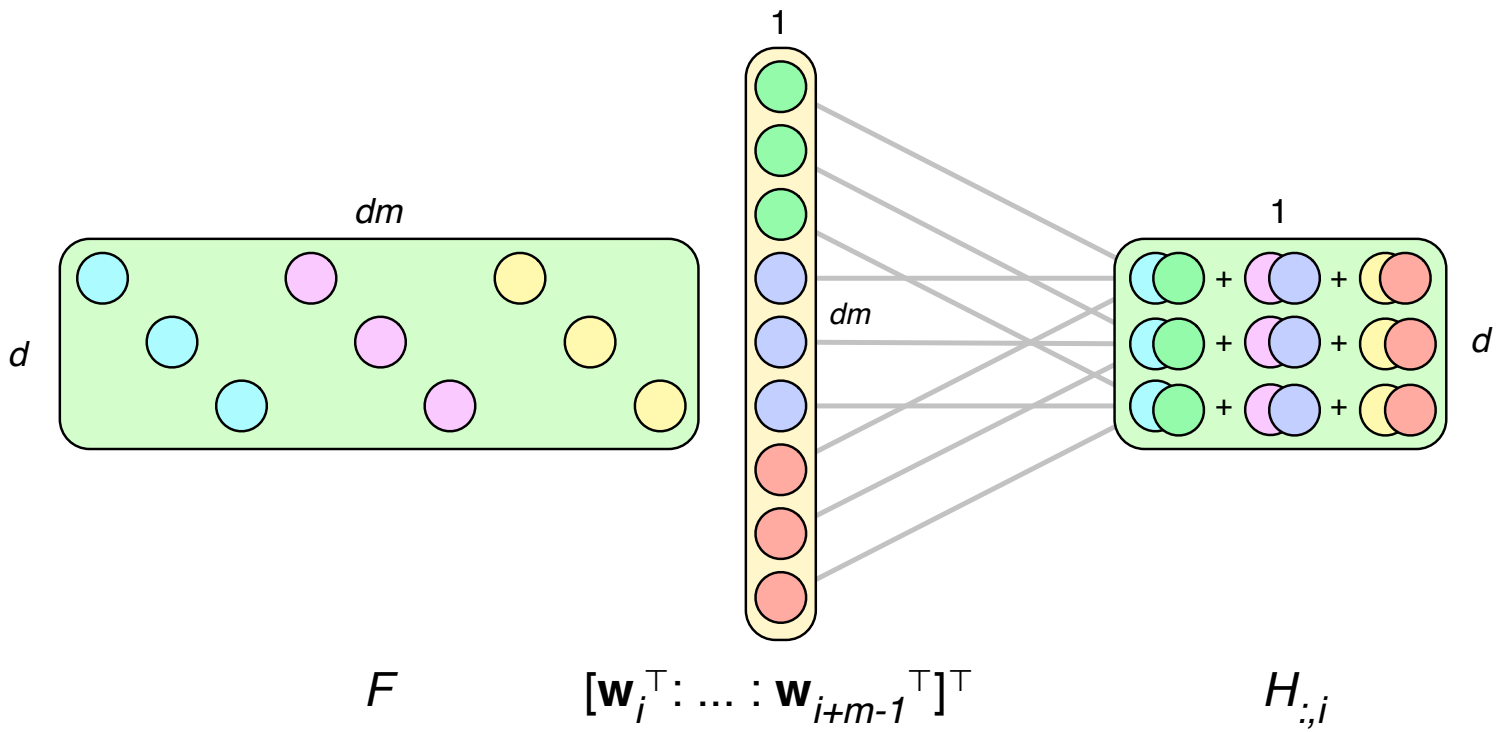


To compute the layerwise convolution, let:

- m be the width of the convolution window
- d be the input dimensionality
- $M \in \mathbb{R}^{d \times m}$ be a matrix with filters as rows
- $F \in R^{d \times dm} = [diag(M_{:,1}), \ldots, diag(M_{:,m})]$ be the filter application matrix
- $\mathbf{w}_i \in \mathbb{R}^d$ be the embedding of the ith word in the input sentence
- $H \in \mathbb{R}^{d \times l}$ be the "sentence" matrix obtained by applying the convolution to the input layer of l word embeddings
- $\mathbf{b} \in \mathbb{R}^d$ a bias vector

Applying the convolution

$$\forall i \in [1, l] \quad H_{:,i} = g(F \begin{bmatrix} \mathbf{w}_i \\ \vdots \\ \mathbf{w}_{i+m-1} \end{bmatrix} + \mathbf{b})$$



A full convolutional sentence model

Come and see the poster for Kalchbrenner *et al.* (2014),
A Convolutional Neural Network for Modelling Sentences.

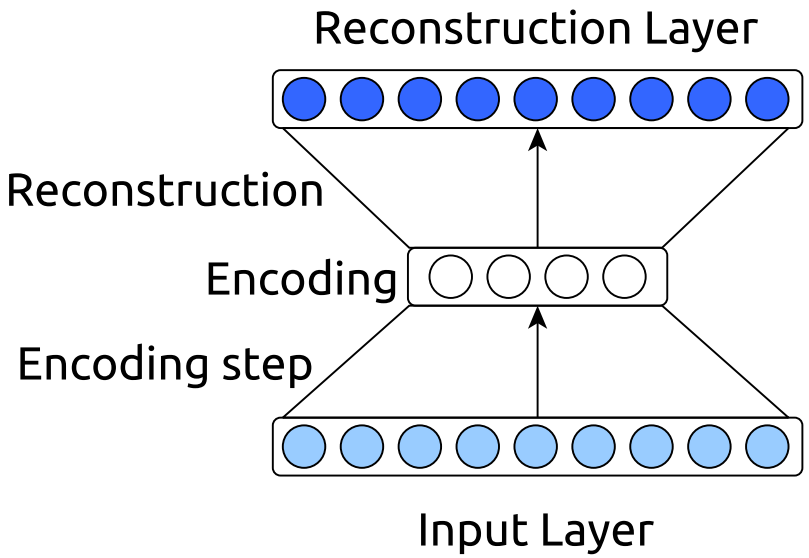
Monday, 18:50-21:30pm, Grand Ballroom, LP17

Training Compositional Vector Space Models

Several things to consider

- **Training Signals** autoencoders, classifiers, unsupervised signals
- **Gradient Calculation** backpropagation
- **Gradient Updates** SGD, L-BFGS, AdaGrad, ...
- **Black Magic** drop-out, layer-wise training, initialisation, ...

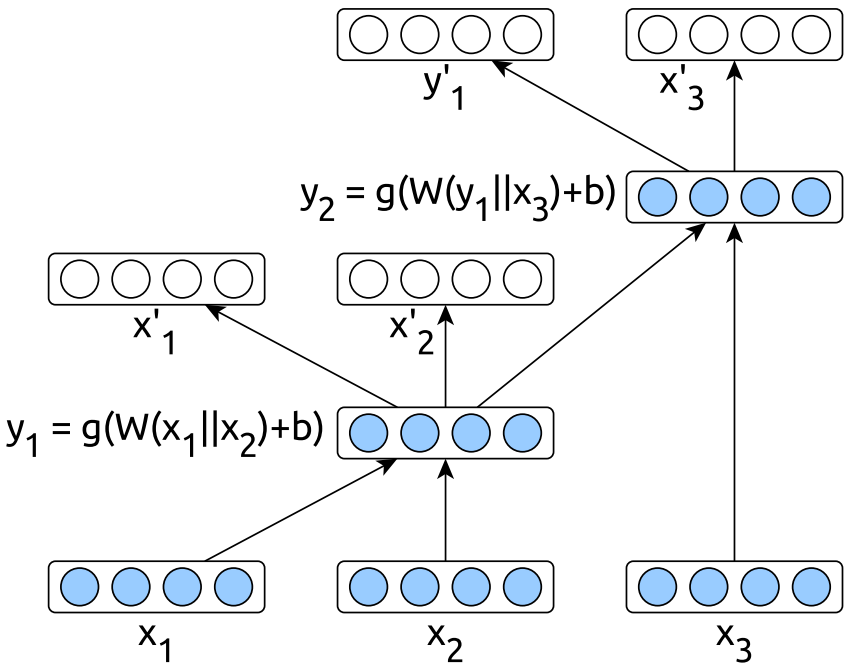
Autoencoders can be used to minimise information loss during composition:



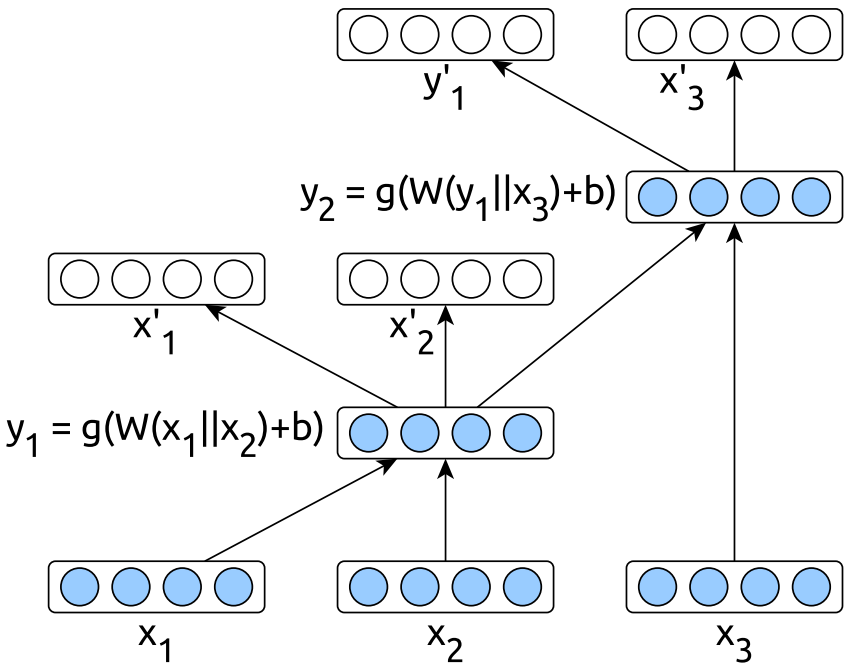
We minimise an objective function over inputs $x_i, i \in N$ and their reconstructions x_i':

$$J = \frac{1}{2} \sum_{i}^{N} \|x_i' - x_i\|^2$$

Recursive Autoencoders



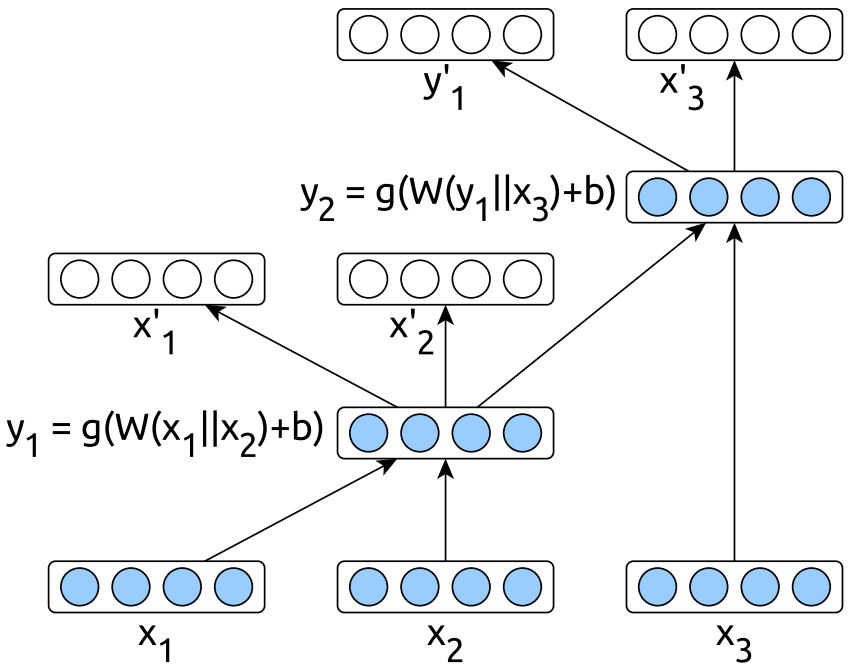
We still want to learn how to represent a full sentence (or house). To do this, we chain autoencoders to create a recursive structure.



Objective Function

Minimizing the reconstruction error will learn a compression function over the inputs:

$$E_{rec}(i,\theta) = \frac{1}{2}\left\|x_i - x_i'\right\|^2$$



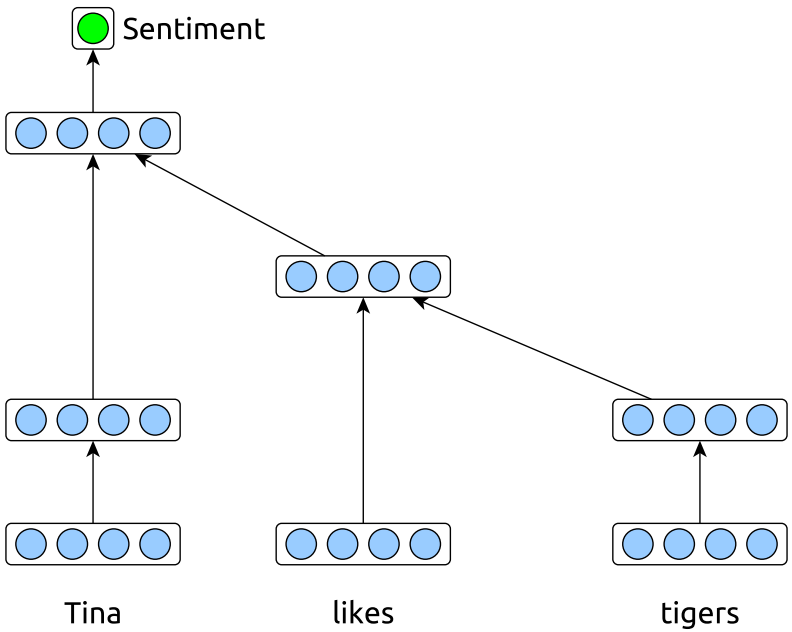
Objective Function

Minimizing the reconstruction error will learn a compression function over the inputs:

$$E_{rec}(i,\theta) = \frac{1}{2}\left\| x_i - x_i' \right\|^2$$

Question: Composition = Compression?

Classification signals

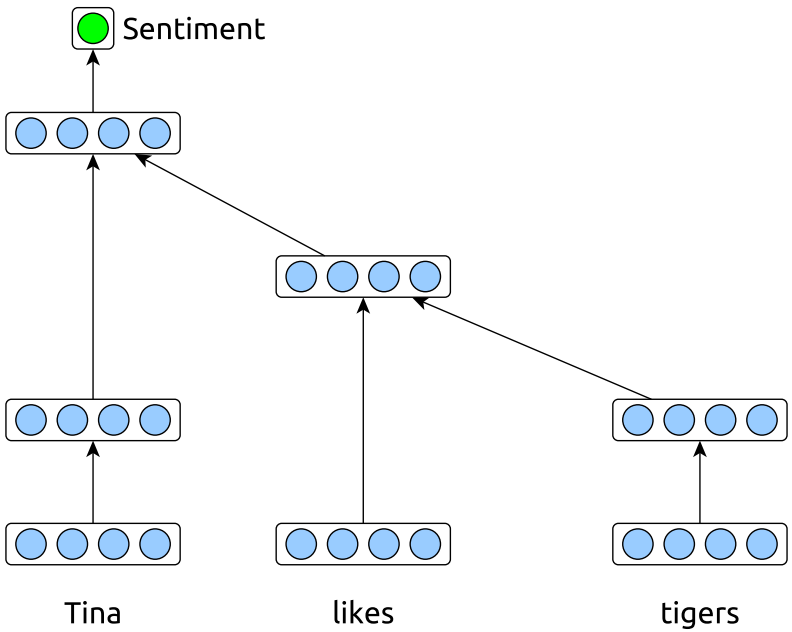


Classification error

$$E(N, I, \theta) = \sum_{n \in N} \frac{1}{2} \|I - v_n\|^2$$

where v_n is the output of a softmax layer on top of the neural network.

Classification signals



Classification error

$$E(N, I, \theta) = \sum_{n \in N} \frac{1}{2} \|I - v_n\|^2$$

where v_n is the output of a softmax layer on top of the neural network.

Question: Sentiment = Semantics?

Simple Energy Function

Strongly align representations of semantically equivalent sentences (a, b)

$$E_{dist}(a, b) = \|f(a) - g(b)\|^2$$

- Works if CVM and representations in one model are fixed (semantic transfer).
- Will degenerate if representations are being learned jointly (i.e. in a multilingual setup).

Representations in both models can be learned in parallel with a modified energy function as follows.

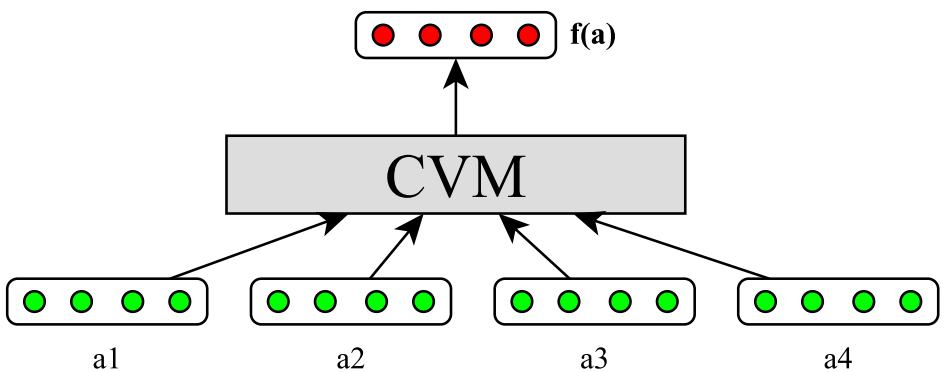
A large-margin objective function

Enforce a margin between unaligned sentences (a, n)

$$E_{noise}(a, b, n) = [m + E_{dist}(a, b) - E_{dist}(a, n)]_+$$

Objective function for a parallel corpus $\mathcal{C}_{A,B}$

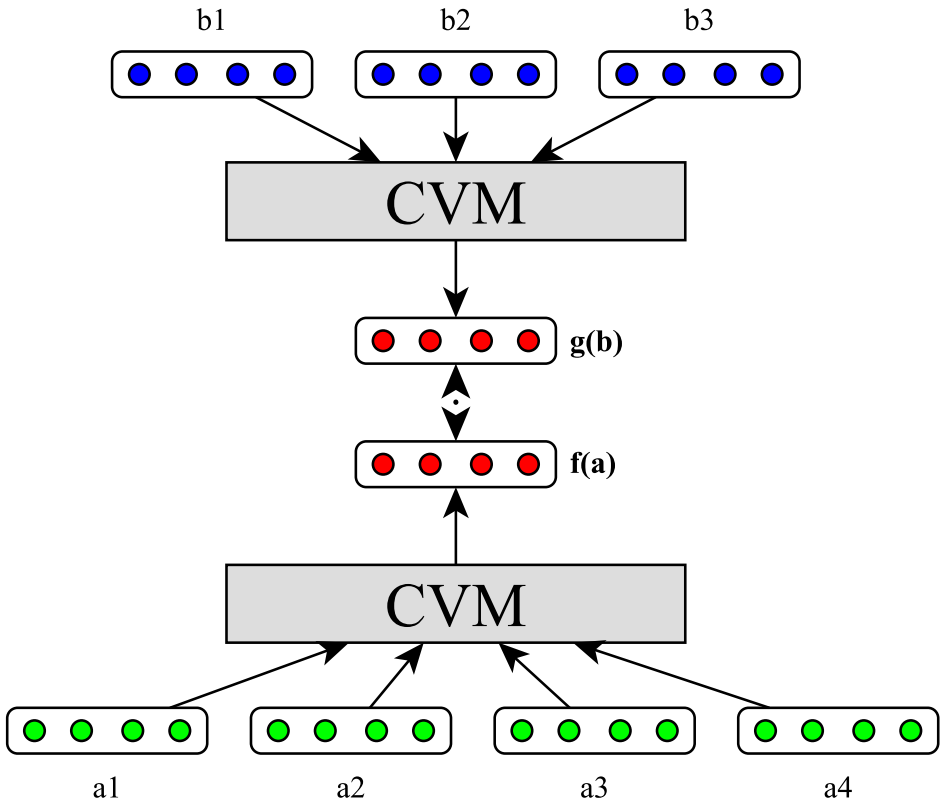
$$J(\theta_{bi}) = \sum_{(a,b)\in\mathcal{C}_{A,B}} \left(\sum_{i=1}^{k} E_{noise}(a, b, n_i) \right) + \frac{\lambda}{2} \|\theta_{bi}\|^2$$



Monolingual Composition Model

- Needs objective function
- Supervised or Autoencoder?
- Compression or Sentiment?

Multilingual Models with Large-Margin Training



Monolingual Composition Model

- Needs objective function
- Supervised or Autoencoder?
- Compression or Sentiment?

Multilingual Model

- Task-independent learning
- Multilingual representations
- Joint-space representations
- Composition function provides large context

Backpropagation

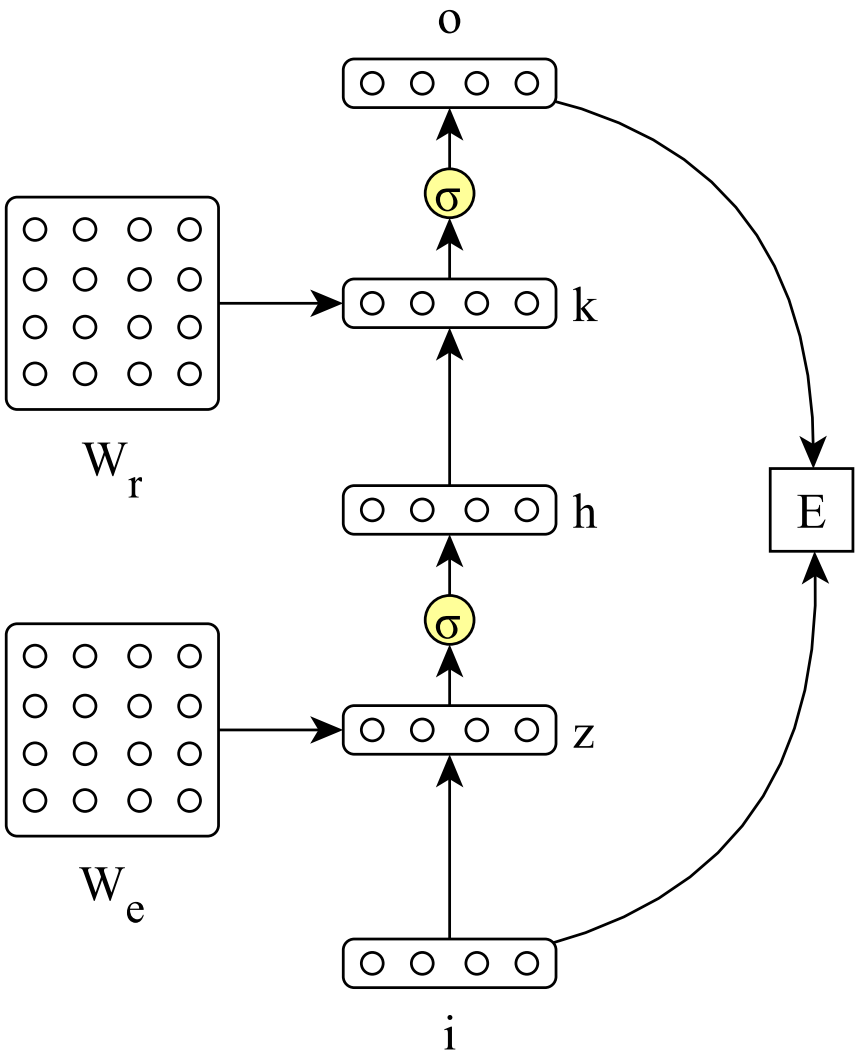
Calculating gradients is simple and fast with backprop:

- Fast
- Uses network structure for efficient gradient calculation
- Simple to adapt for dynamic structures
- Fast

Gradient-descent based strategies

- Stochastic Gradient Descent
- L-BFGS
- Adaptive Gradient Descent (AdaGrad)

Backpropagation (autoencoder walk-through)



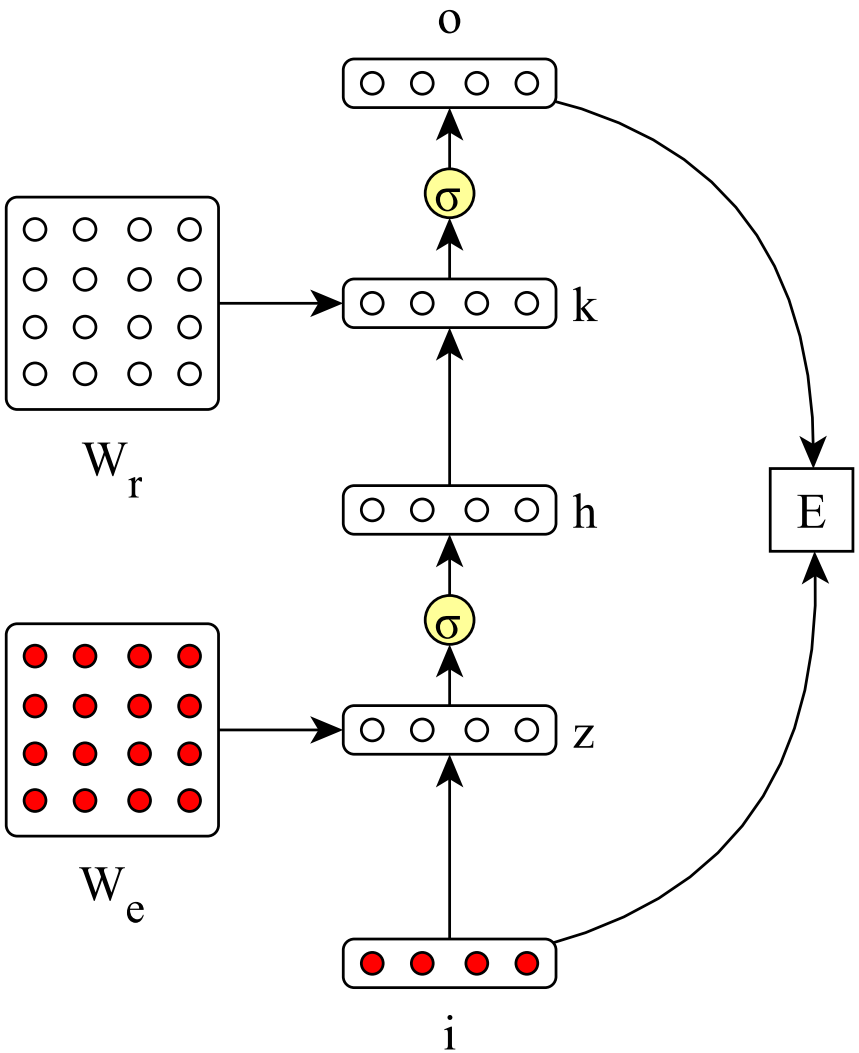
Autoencoder

This is a simple autoencoder:

- intermediary layers z, k
- input i
- output/reconstruction o
- hidden layer h
- weight matrices W_e, W_r
- $E = \frac{1}{2}(\|o - i\|)^2$

We omit bias terms for simplicity.

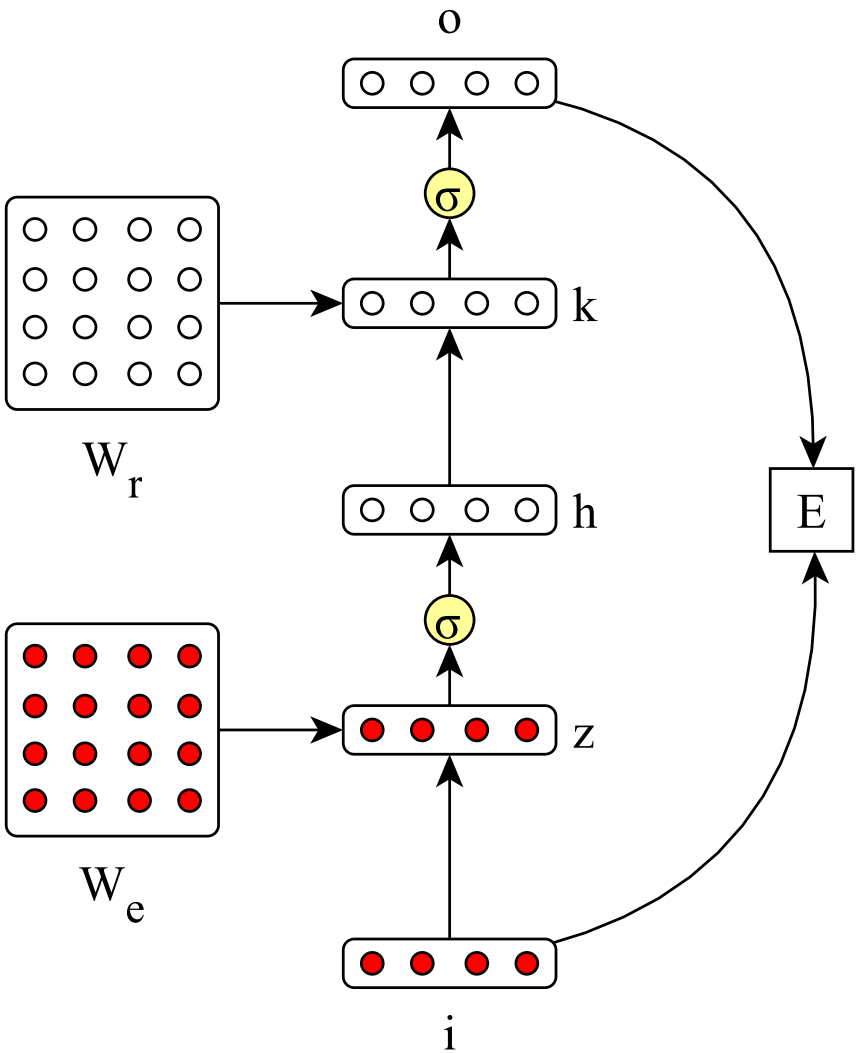
Backpropagation (autoencoder walk-through)



Forwardpagate

$$z = W^e i$$

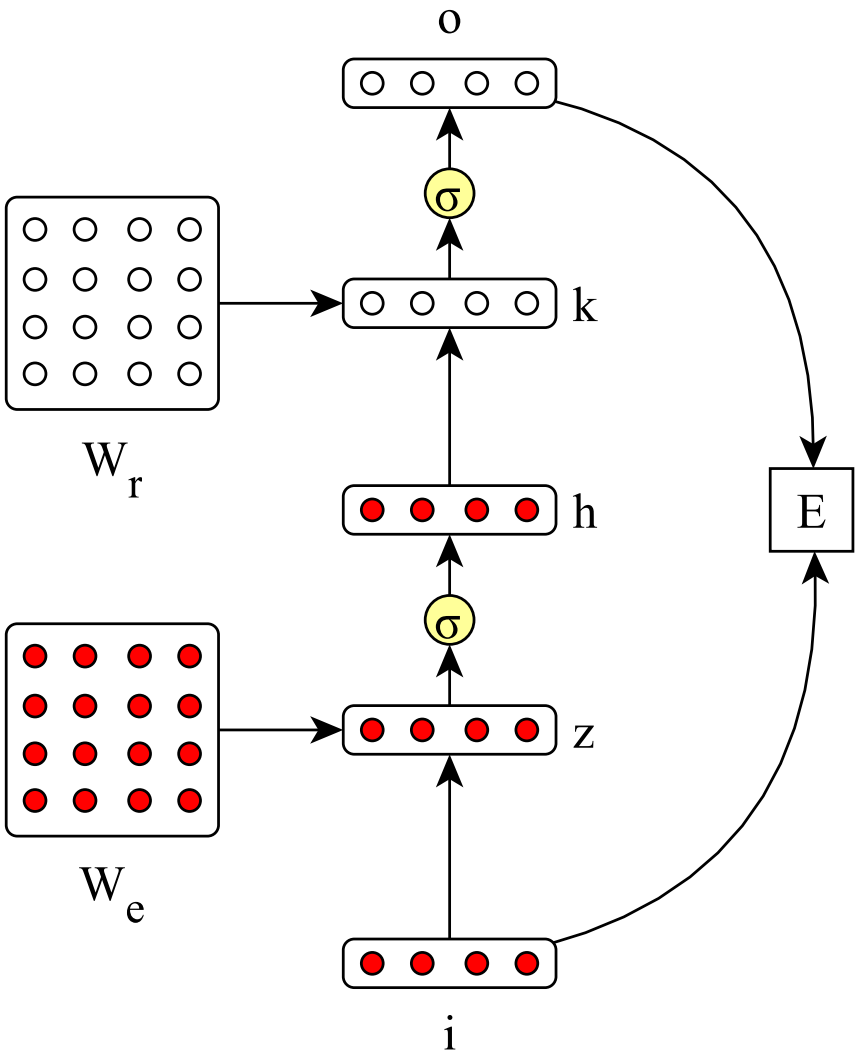
Backpropagation (autoencoder walk-through)



Forwardpagate

$$z = W^e i$$

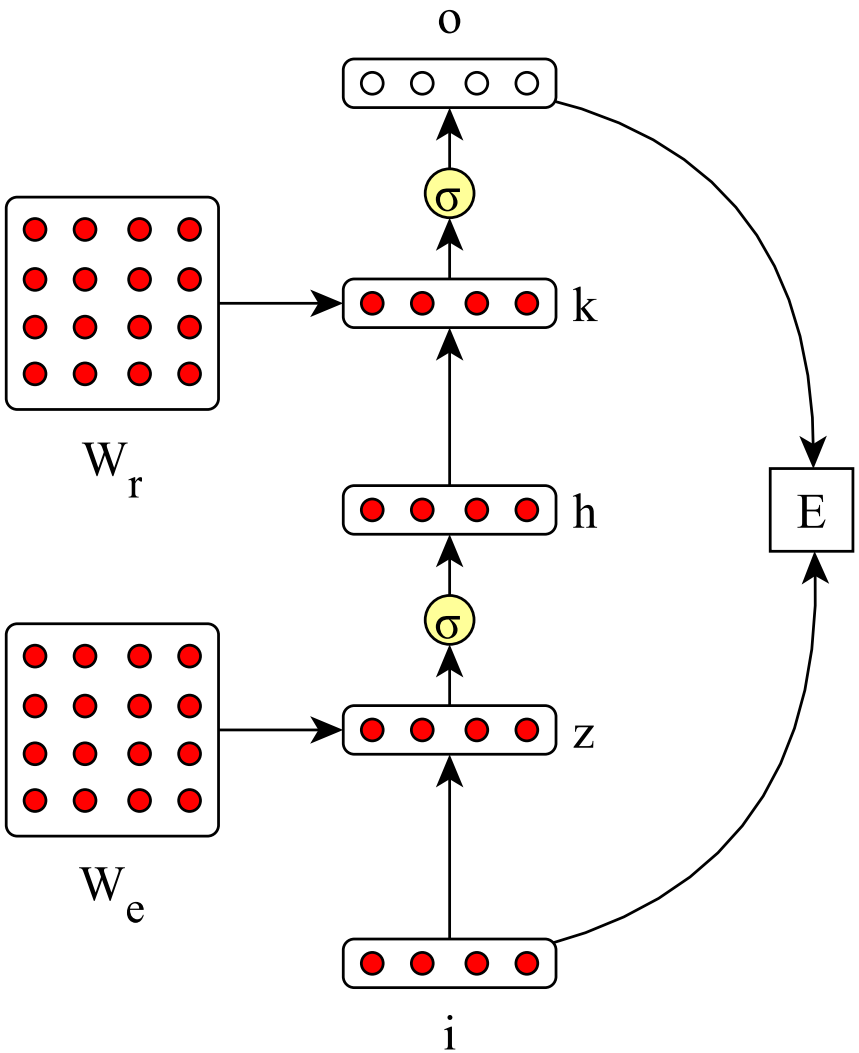
Backpropagation (autoencoder walk-through)



Forwardpagate

$$h = \sigma(z)$$

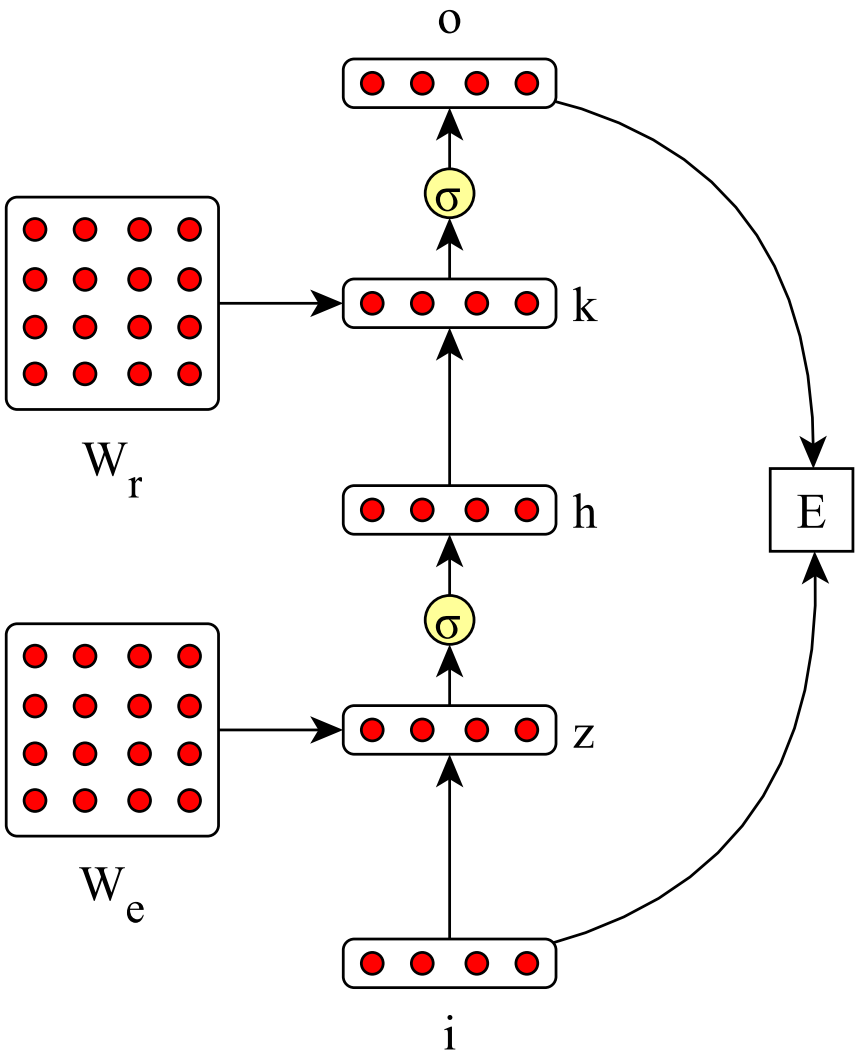
Backpropagation (autoencoder walk-through)



Forwardpagate

$$k = W^r h$$

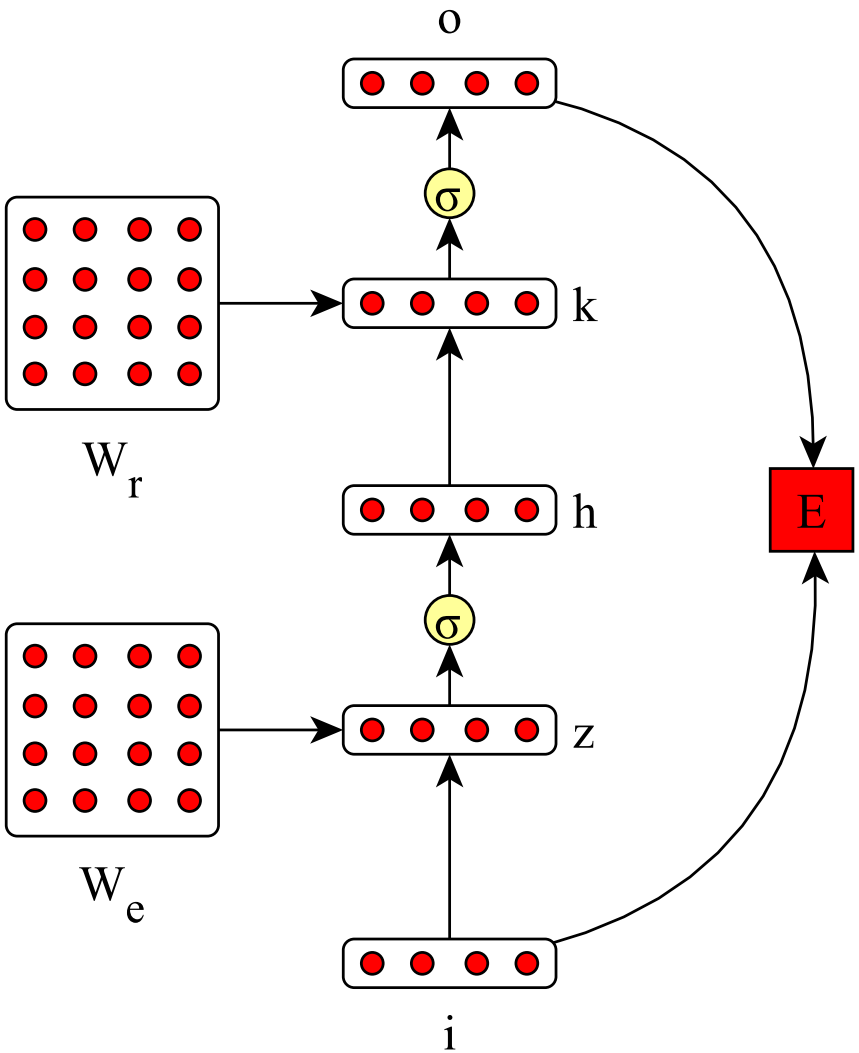
Backpropagation (autoencoder walk-through)



Forwardpagate

$$o = \sigma(k)$$

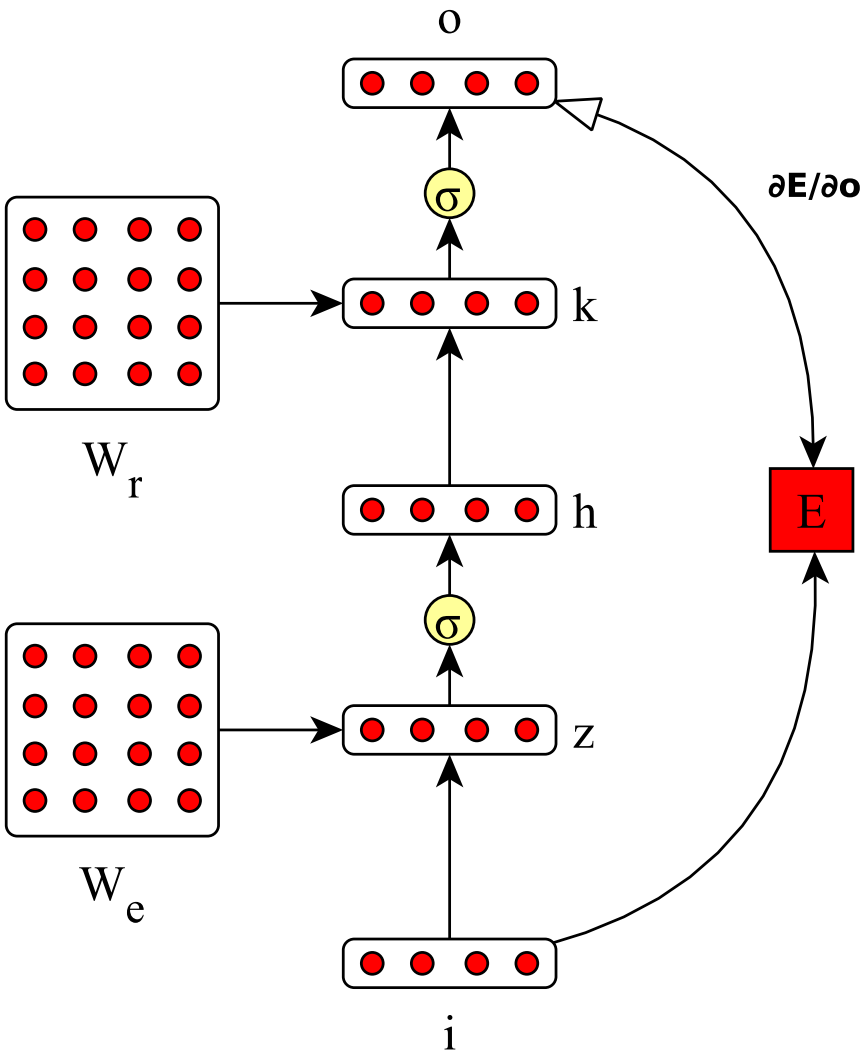
Backpropagation (autoencoder walk-through)



Error function

$$E = \frac{1}{2}(\|o - i\|)^2$$

Backpropagation (autoencoder walk-through)



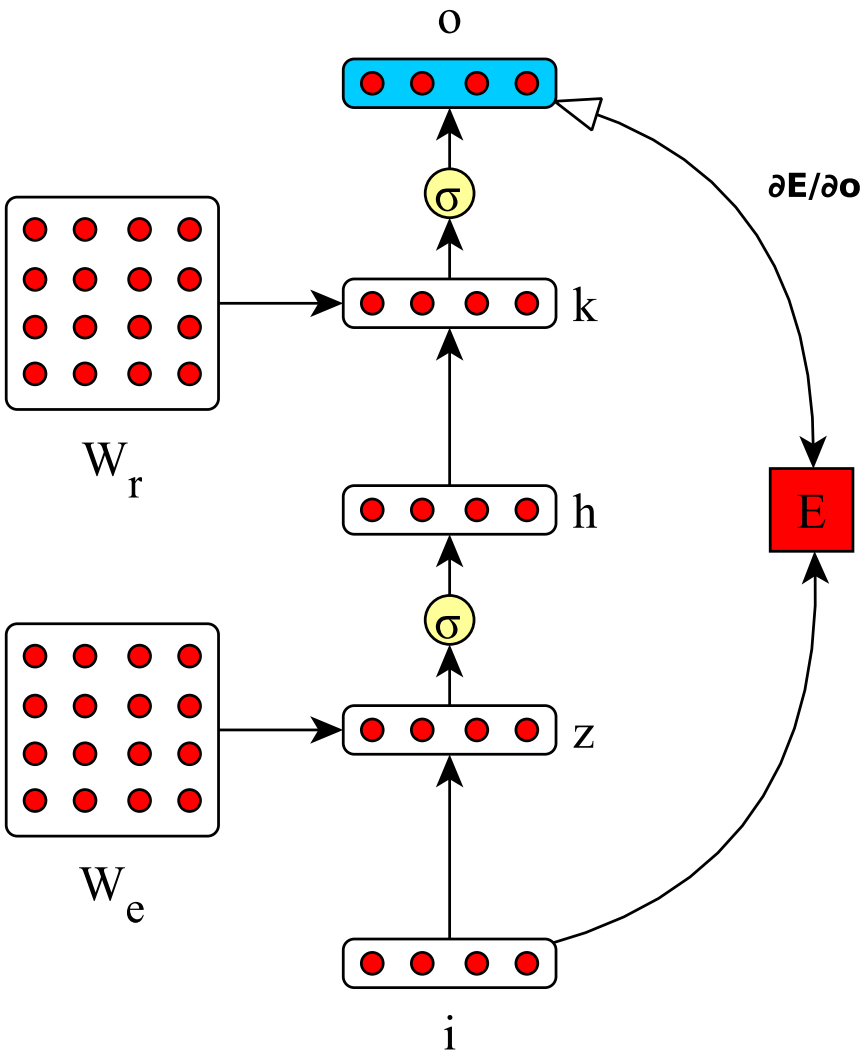
Error function

$$E = \frac{1}{2}(\|o - i\|)^2$$

Backpropagation

We begin by calculating the error with respect to the output node o.

Backpropagation (autoencoder walk-through)



Error function

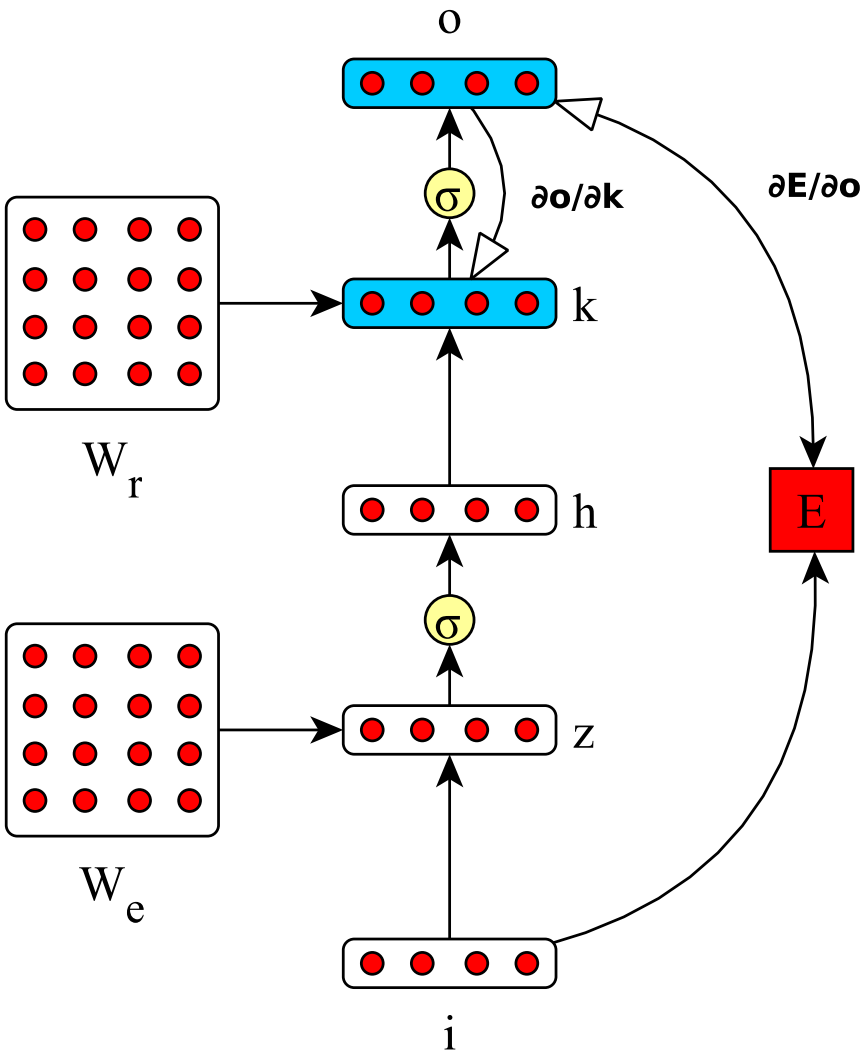
$$E = \frac{1}{2}(\|o - i\|)^2$$

Backpropagation

We begin by calculating the error with respect to the output node o.

$$\frac{\partial E}{\partial o} = (o - i)$$

Backpropagation (autoencoder walk-through)



Forwardpagate

$$o = \sigma(k)$$

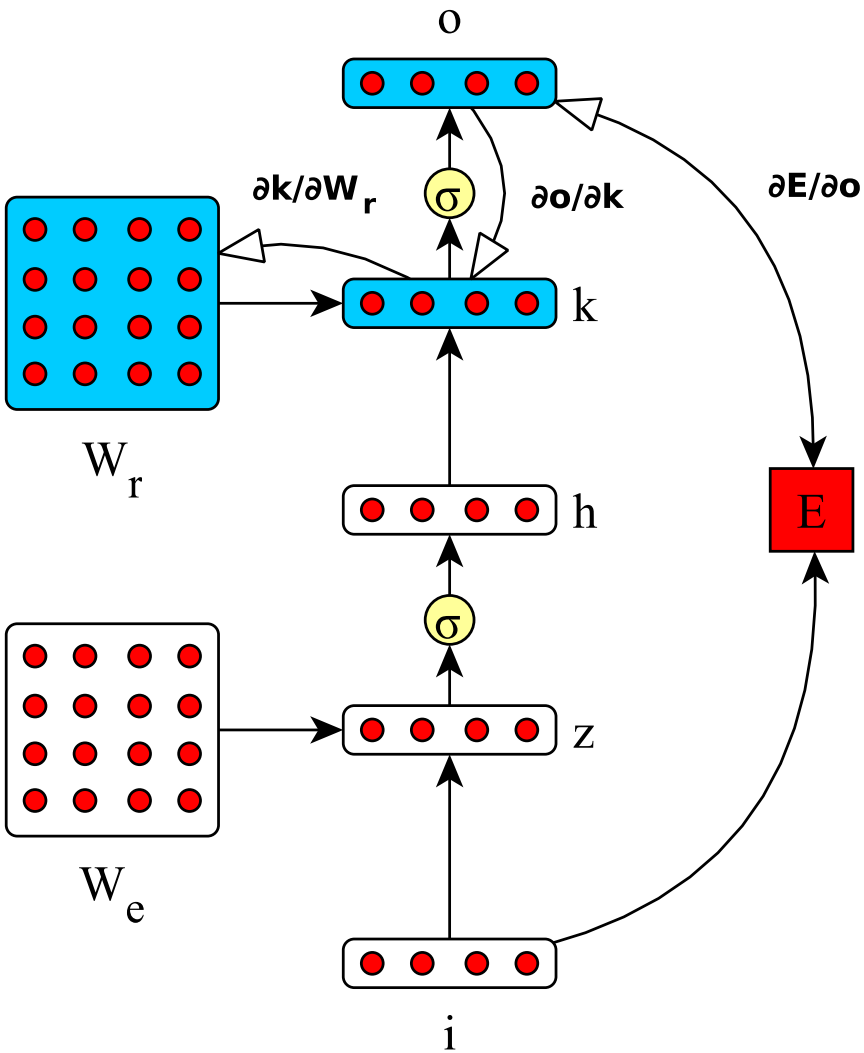
Backpropagation

$$\frac{\partial E}{\partial k} = \frac{\partial o}{\partial k}\frac{\partial E}{\partial o}$$

$$\frac{\partial E}{\partial o} = \checkmark$$

$$\frac{\partial o}{\partial k} = \sigma'(k) = \sigma(k)(1-\sigma(k))$$

Backpropagation (autoencoder walk-through)



Forwardpagate

$$k = W^r h$$

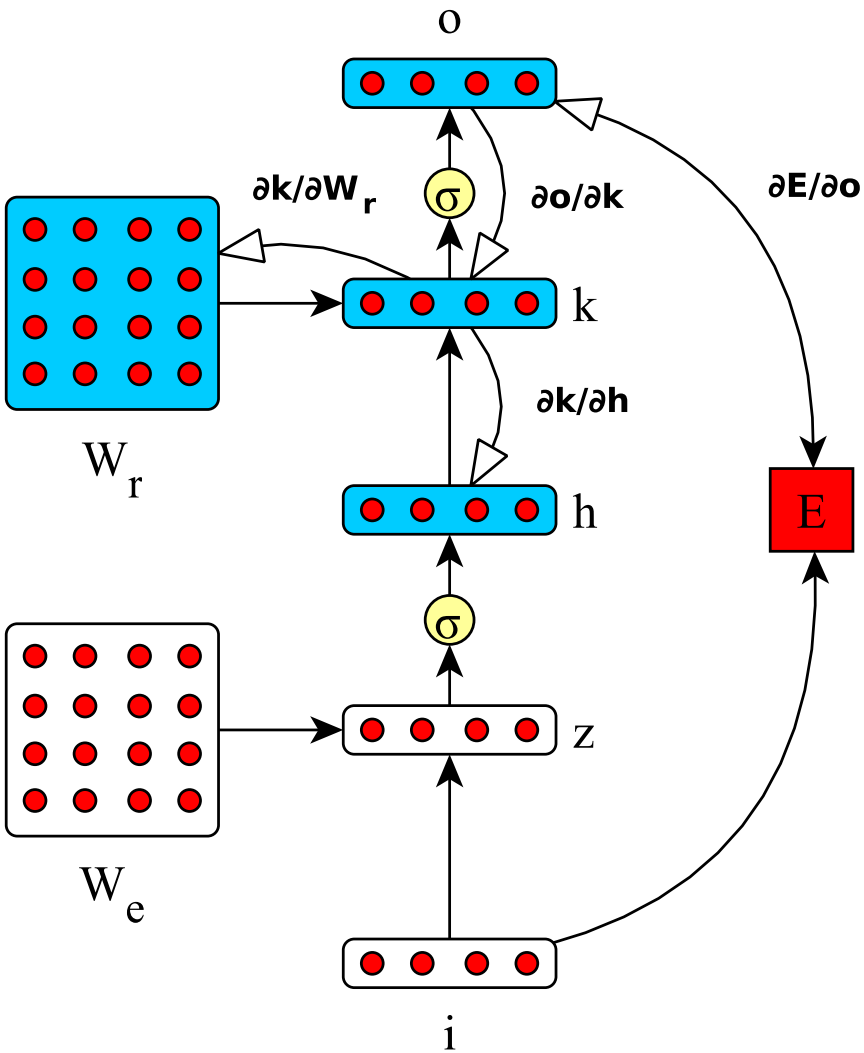
Backpropagation

$$\frac{\partial E}{\partial W_r} = \frac{\partial E}{\partial k}\frac{\partial k}{\partial W_r}$$

$$\frac{\partial E}{\partial k} = \checkmark$$

$$\frac{\partial k}{\partial W_r} = h$$

Backpropagation (autoencoder walk-through)



Forwardpagate

$$k = W^r h$$

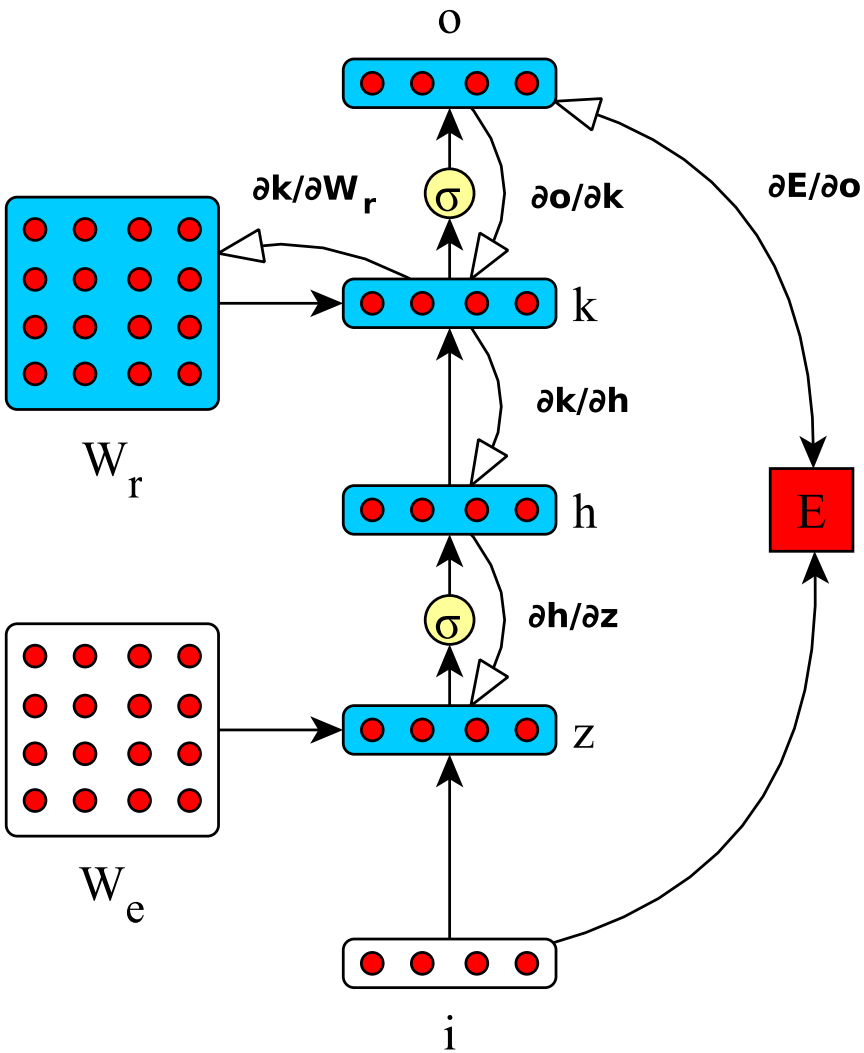
Backpropagation

$$\frac{\partial E}{\partial h} = \frac{\partial k}{\partial h}\frac{\partial E}{\partial k}$$

$$\frac{\partial E}{\partial k} = \checkmark$$

$$\frac{\partial k}{\partial h} = W_r$$

Backpropagation (autoencoder walk-through)



Forwardpagate

$$h = \sigma(z)$$

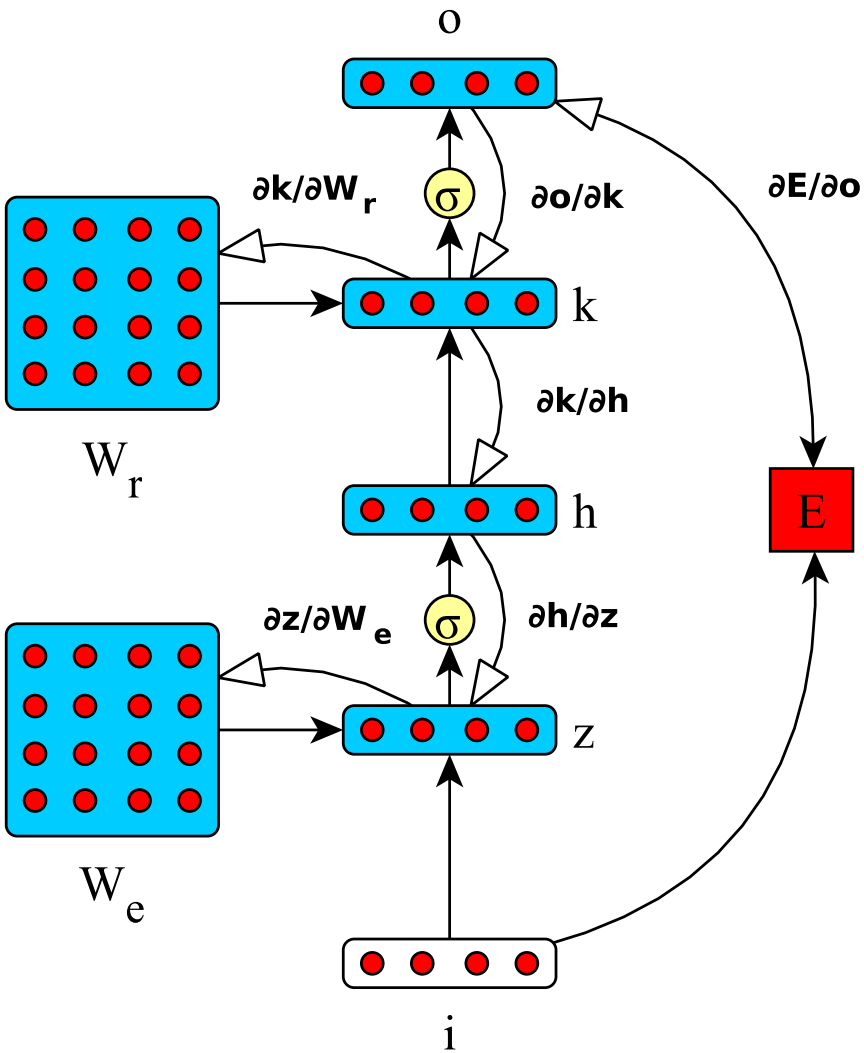
Backpropagation

$$\frac{\partial E}{\partial z} = \frac{\partial h}{\partial z}\frac{\partial E}{\partial h}$$

$$\frac{\partial E}{\partial h} = \checkmark$$

$$\frac{\partial h}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Backpropagation (autoencoder walk-through)



Forwardpagate

$$z = W^e i$$

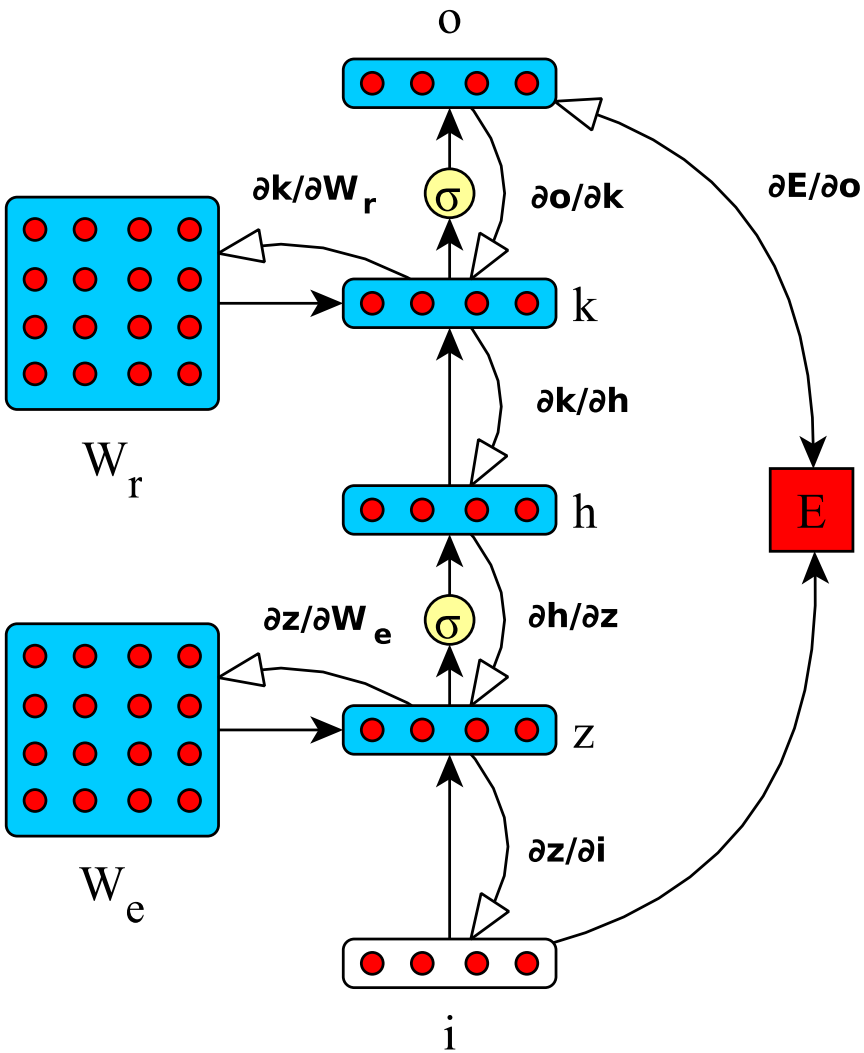
Backpropagation

$$\frac{\partial E}{\partial W_e} = \frac{\partial E}{\partial z}\frac{\partial k}{\partial W_e}$$

$$\frac{\partial E}{\partial k} = \checkmark$$

$$\frac{\partial k}{\partial W_e} = i$$

Backpropagation (autoencoder walk-through)



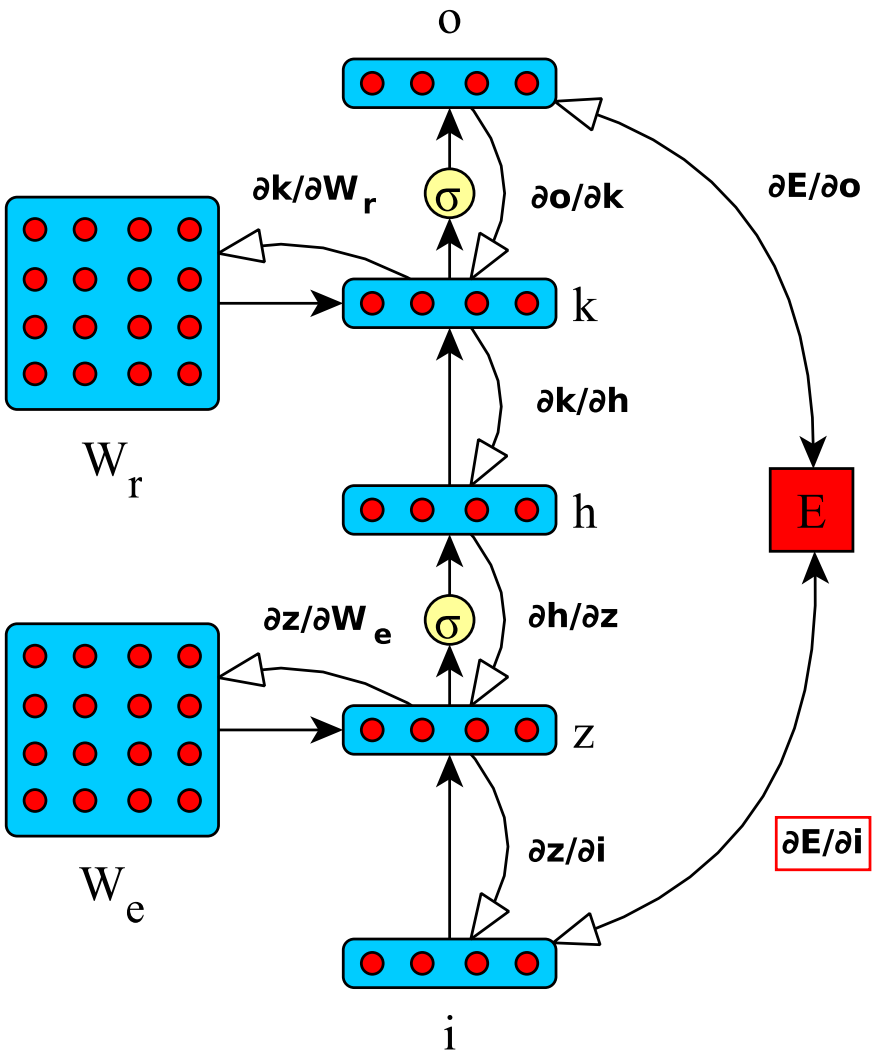
Forwardpagate

$$z = W^e i$$

Backpropagation

$$\frac{\partial E}{\partial i} = \frac{\partial z}{\partial i}\frac{\partial E}{\partial z}$$
$$\frac{\partial E}{\partial z} = \checkmark$$
$$\frac{\partial z}{\partial i} = W_e$$

Backpropagation (autoencoder walk-through)



Forwardpagate + Error

$$z = W^e i$$

$$E = \frac{1}{2}(\|o - i\|)^2$$

Backpropagation

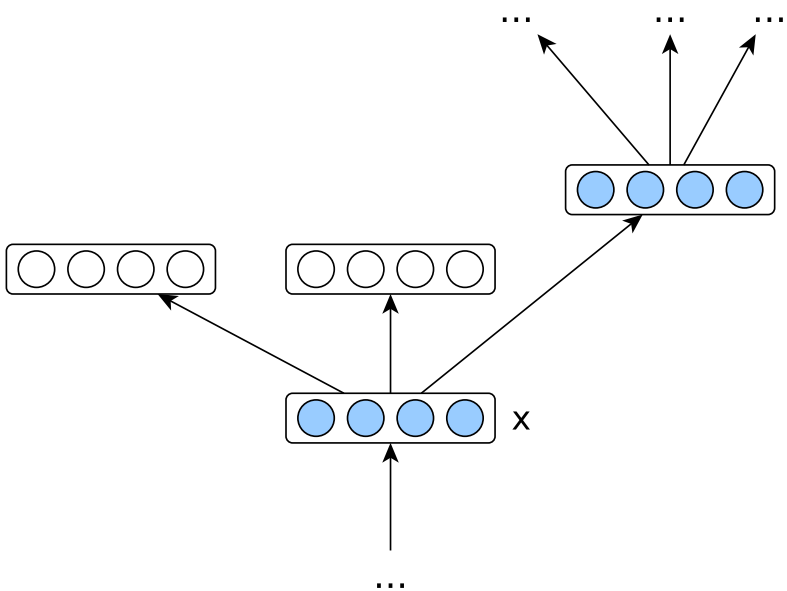
$$\frac{\partial E}{\partial i} = \frac{\partial z}{\partial i}\frac{\partial E}{\partial z} + \frac{\partial E}{\partial i}$$

$$\frac{\partial z}{\partial i} = W_e$$

$$\frac{\partial E}{\partial i} = -(o - i)$$

Backpropagation for recursive neural nets

Backpropagation can be modified for tree structures and to adjust for a distributed error function.



We know that

$$\frac{\partial E}{\partial x} = \sum_{y \in Y} \frac{\partial y}{\partial x} \frac{\partial E}{\partial y}$$

$Y =$ Successors of x

This allows us to efficiently calculate all gradients with respect to E.

Once we have gradients, we needs some function

$$\theta_{t+1} = f(G_t, \theta_t)$$

that sets models parameters given previous model parameters and gradients.

Gradient Update Strategies

- Stochastic Gradient Descent
- L-BFGS
- Adaptive Gradient Descent

AdaGrad

Fine-tune the learning rate for each parameter based on the historical gradient for that parameter.

First, initialise $H_i = 0$ for each parameter W_i and set step-size hyperparameter λ. During training, at each iteration:

1. Calculate gradient $G_i = \frac{\delta E}{\delta W_i}$. Update $H_i = H_i + G_i^2$.
2. Calculate parameter-specific learning rate $\lambda_i = \frac{\lambda}{\sqrt{H_i}}$.
3. Update parameters as in SGD: $W_i = W_i - \lambda_i G_i$.

Explanation

Parameter-specific learning rate λ_i decays over time, and more quickly when weights are updated more heavily.

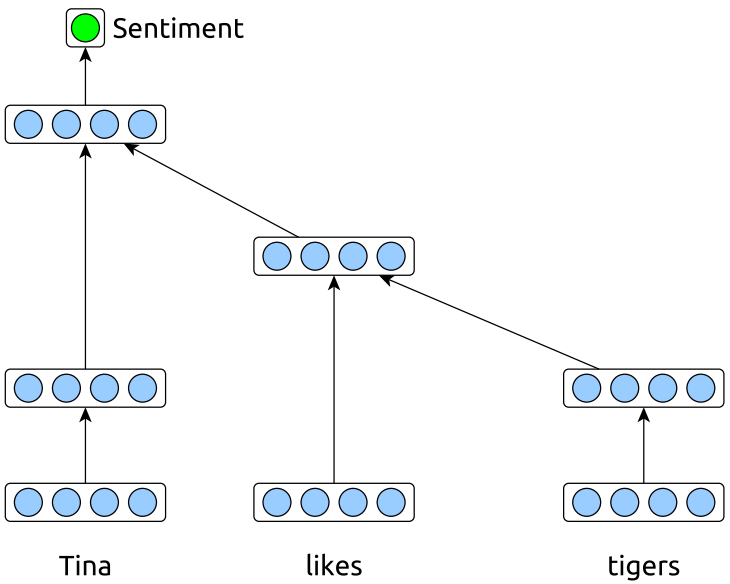
Various things will improve your odds

- Pre-train any deep model with layer-wise autoencoders
- Regularise all embeddings (with L1/L2 regulariser)
- Train in randomised mini-batches rather than full batch
- Use patience/early stopping instead of training to convergence

Application: Sentiment labelling with RecNNs

We can use a recursive neural network to learn sentiment:

- sentiment signal attached to root (sentence) vector
- trained using softmax function and backpropagation



Sentiment Analysis

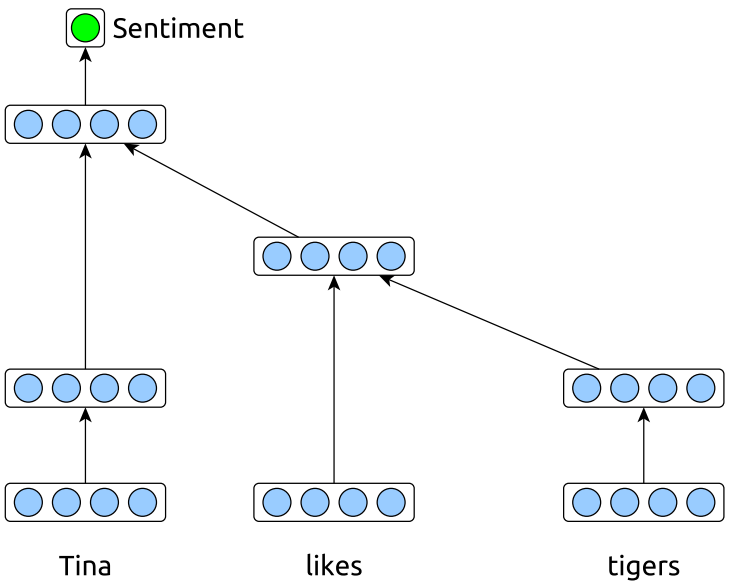
Assume the simplest composition function to begin:

$$p = g\left(W(u\|v) + b\right)$$

Application: Sentiment labelling with RecNNs

We can use a recursive neural network to learn sentiment:

- sentiment signal attached to root (sentence) vector
- trained using softmax function and backpropagation



Sentiment Analysis

Assume the simplest composition function to begin:

$$p = g\left(W(u\|v) + b\right)$$

This will work ...

... sort of.

Making sentiment analysis work better

The basic system will work. However, to produce state-of-the-art results, a number of improvements and tricks are necessary.

Composition Function

- Parametrise the composition function
- More complex word representations
- Structure the composition on parse trees
- Convolution instead of binary composition

Other Changes

- Instead of the root node, evaluate on all nodes
- Add autoencoders as a second learning signal
- Initialise with pre-trained representations
- Drop-out training and similar techniques

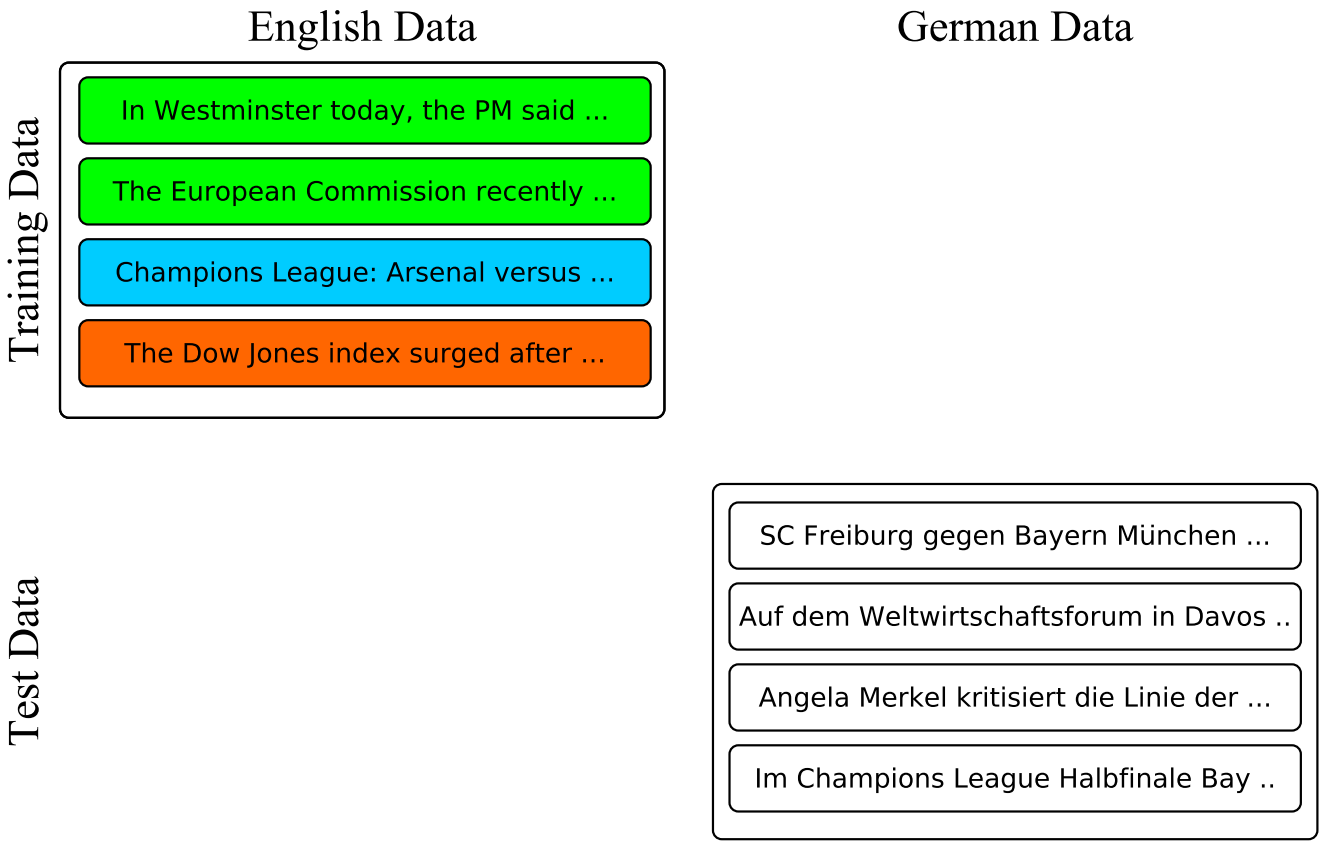
Corpora for sentiment analysis

Corpora

- Movie Reviews (Pang and Lee)
  - Relatively small, but has been used extensively
  - SOTA ∼87% accuracy (Kalchbrenner et al., 2014)
  - `http://www.cs.cornell.edu/people/pabo/movie-review-data/`
- Sentiment Treebank
  - Sentiment annotation for sentences and sub-trees
  - SOTA ∼49% accuracy (Kalchbrenner et al., 2014)
  - `http://nlp.stanford.edu/sentiment/treebank.html`
- Twitter Sentiment140 Corpora
  - Fairly large amount of data
  - Twitter language is strange!
  - SOTA ∼87% (Kalchbrenner et al., 2014)
  - `http://help.sentiment140.com/for-students/`

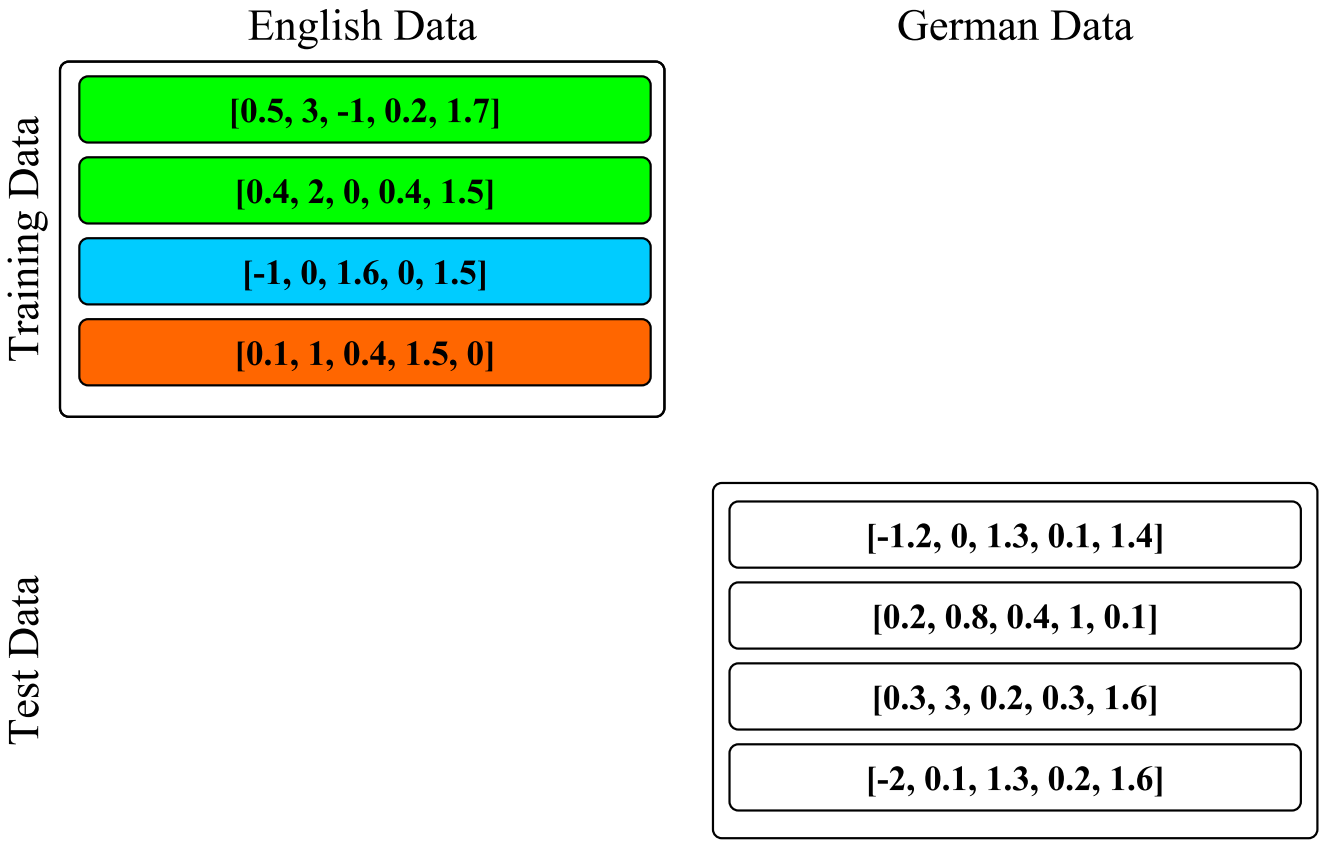
Application: Cross-lingual Document Classification

One application for multilingual representations is cross-lingual annotation transfer. This can be evaluated with cross-lingual document classification (Klementiev et al., 2012):



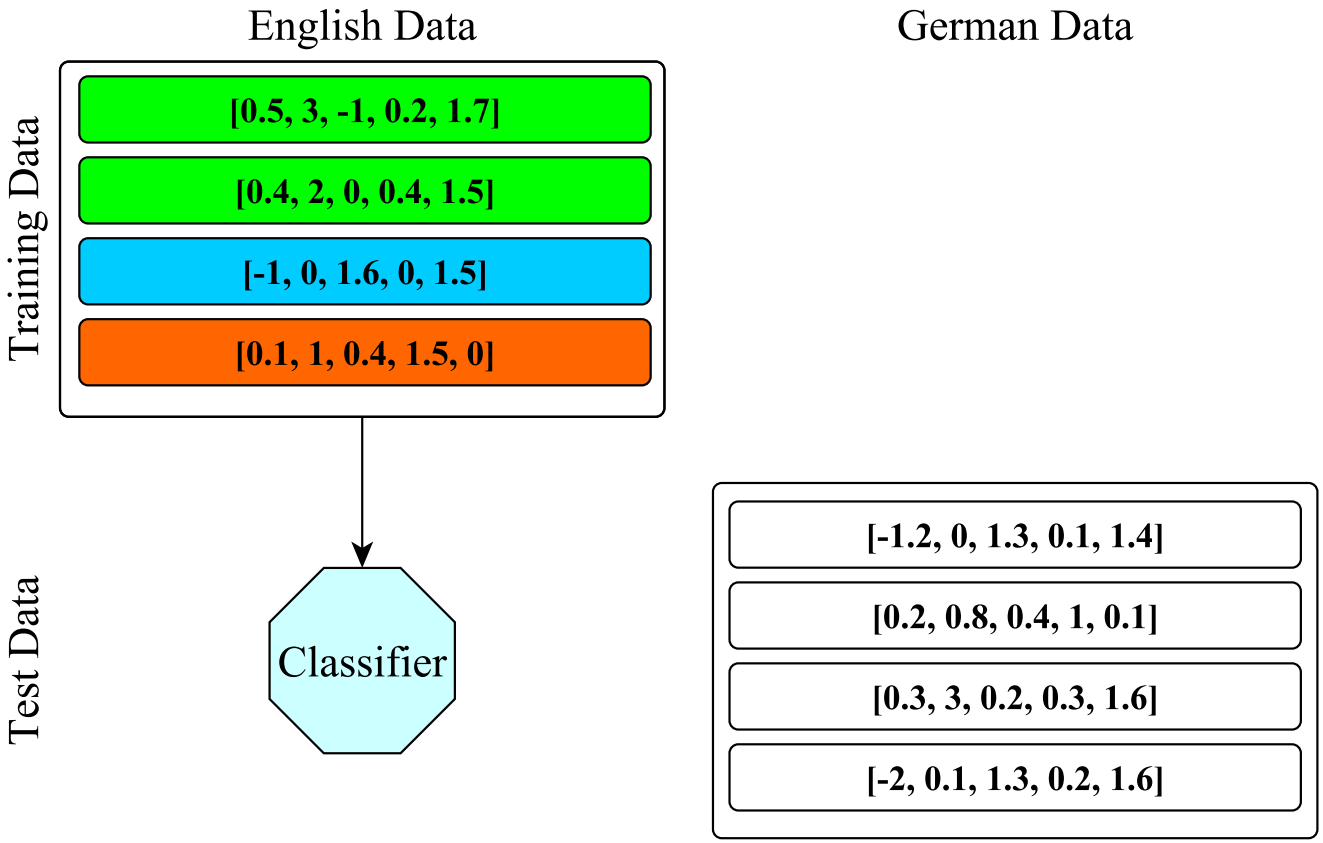
Application: Cross-lingual Document Classification

One application for multilingual representations is cross-lingual annotation transfer. This can be evaluated with cross-lingual document classification (Klementiev et al., 2012):



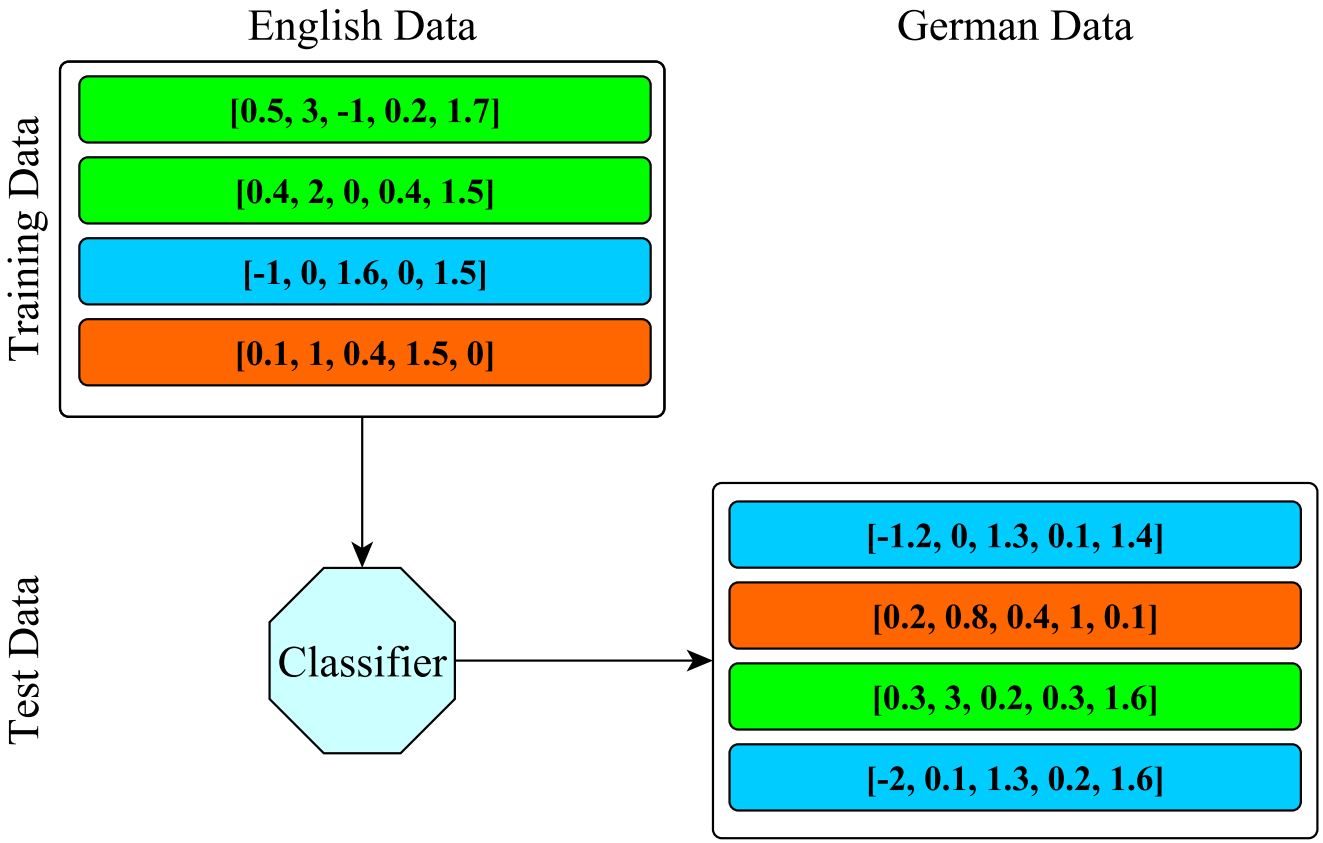
Application: Cross-lingual Document Classification

One application for multilingual representations is cross-lingual annotation transfer. This can be evaluated with cross-lingual document classification (Klementiev et al., 2012):



Application: Cross-lingual Document Classification

One application for multilingual representations is cross-lingual annotation transfer. This can be evaluated with cross-lingual document classification (Klementiev et al., 2012):

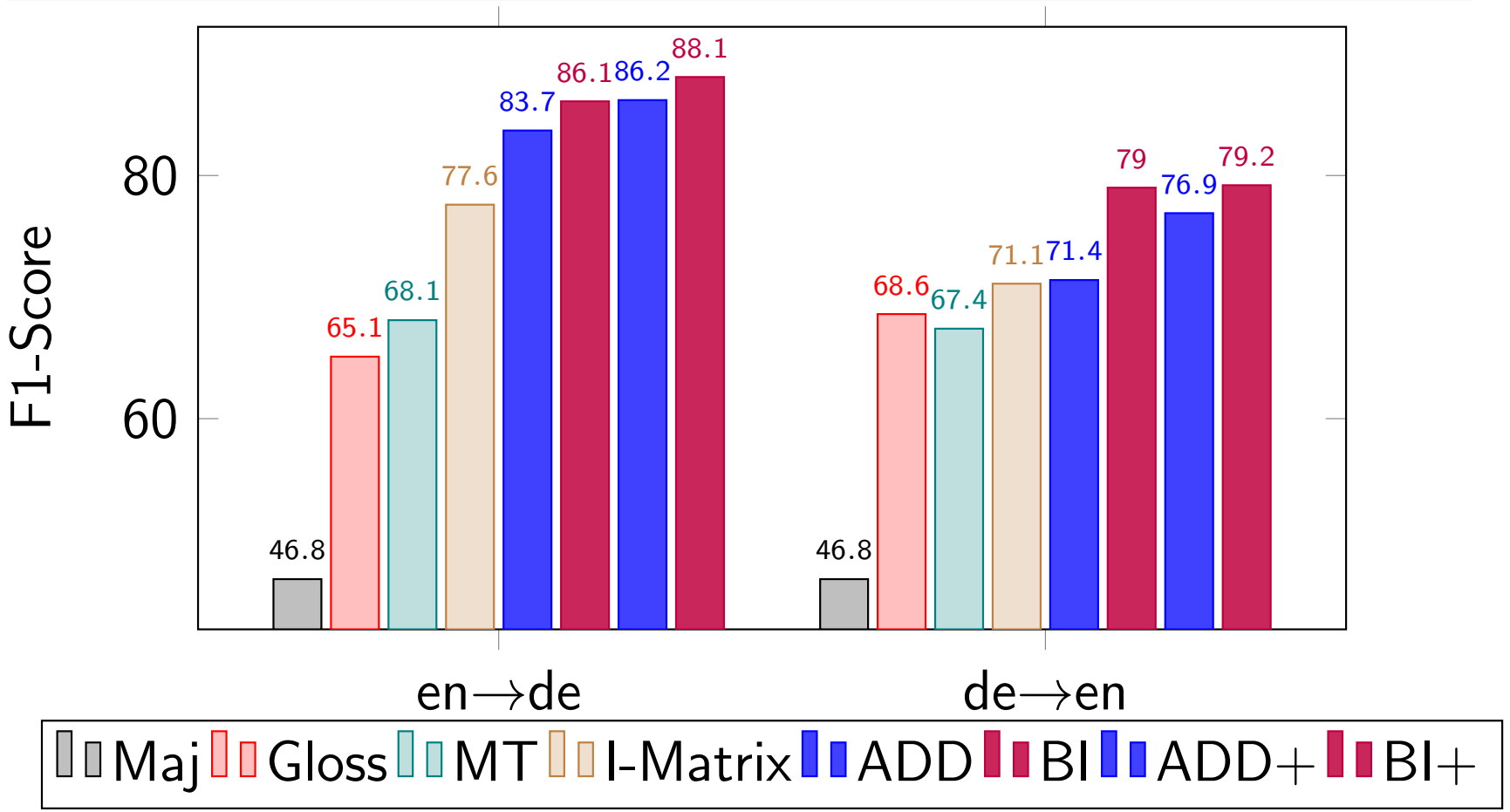


Two Stage Strategy

1. **Representation Learning**
   Using the large-margin objective introduced earlier, it is easy to train a model on large amounts of parallel data (here: Europarl) using any composition function together with AdaGrad and an L_2 regularizer.
2. **Classifier training**
   Subsequently, sentence or document representations can be used as input to train a supervised classifier (here: Averaged Perceptron). Assuming the vectors are semantically similar across languages this classifier should be useful independent of its training language.

Two composition models in the multilingual setting

$$f_{ADD}(a) = \sum_{i=0}^{|a|} a_i \qquad f_{BI}(a) = \sum_{i=1}^{|a|} \tanh\left(x_{i-1} + x_i\right)$$



Two composition models in the multilingual setting

$$f_{ADD}(a) = \sum_{i=0}^{|a|} a_i \qquad f_{BI}(a) = \sum_{i=1}^{|a|} \tanh\left(x_{i-1} + x_i\right)$$

More details on these results

Come and see the talk for Hermann and Blunsom (2014), *Multilingual Models for Compositional Distributed Semantics*

Monday, 10:10am, Grand Ballroom VI, Session 1B

Outline

1 Distributional Semantics

2 Neural Distributed Representations

3 Semantic Composition

4 Last Words

Recap

Distributional models:

- Well motivated
- Empirically successful at the word level
- Useable at the phrase level

But...

- No easy way from word to sentence
- Primarily oriented towards measuring word similarity
- Large number of discrete hyperparameters which must be set manually

Recap

Distributed neural models:

- Free us from the curse of distributional hyperparameters
- Fast
- Compact
- Generative
- Easy to jointly condition representations

Recap

Distributed compositional models:

- Allow classification over and generation from phrase, sentence, or document representations
- Recursive neural networks integrate syntactic structure
- ConvNets go from local to global context hiearchically
- Multimodal embeddings

- Neural methods provide us with a powerful set of tools for embedding language.
- They are easier to use than people think.
- They are true to a generalization of the distributional hypothesis: meaning is inferred from use.
- They provide better ways of tying language learning to extra-linguistic contexts (images, knowledge-bases, cross-lingual data).
- You should use them.

Thanks for listening!

References

Distributional Semantics

- Baroni, M. and Lenci, A. (2010). Distributional Memory: A general framework for corpus-based semantics.
- Bullinaria, J. and Levy, J. (2012). Extracting semantic representations from word co-occurrence statistics: Stop lists, stemming and SVD.
- Firth, J.R. (1957). A synopsis of linguistic theory 1930-1955.
- Grefenstette, G. (1994). Explorations in automatic thesaurus discovery.
- Harris, Z.S. (1968). Mathematical structures of language.
- Hoffman, T. and Puzicha, J. (1998). Unsupervised learning from dyadic data.
- Landauer, T.K. and Dumais, S.T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge.
- Lin, D. and Pantel, P. (2001). DIRT — Discovery of Inference Rules from Text.
- Padó, S. and Lapata, M. (2007). Dependency-based construction of semantic space models.
- Turney, P.D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics.

References

Neural Language Modelling

- Bengio, Y., Schwenk, H., Senécal, J. S., Morin, F. and Gauvain, J.L. (2006). Neural probabilistic language models.
- Brown, P.F., Desouza, P.V., Mercer, R.L., Pietra, V.J.D. and Lai, J.C. (1992). Class-based n-gram models of natural language.
- Grefenstette, E., Blunsom, P., de Freitas, N. and Hermann, K.M. (2014). A Deep Architecture for Semantic Parsing.
- Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models.
- Mikolov, T., Karafiát, M., Burget, L., Cernocký, J. and Khudanpur, S. (2010). Recurrent neural network based language model.
- Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling.
- Mnih, A. and Hinton, G. (2008). A Scalable Hierarchical Distributed Language Model.
- Sutskever, I., Martens, J. and Hinton, G. (2011). Generating text with recurrent neural networks.

References

Compositionality

- Dinu, G. and Baroni, M. (2014). How to make words with vectors: Phrase generation in distributional semantics.
- Grefenstette, E. (2013). Towards a formal distributional semantics: Simulating logical calculi with tensors.
- Grefenstette, E., Dinu, G., Zhang, Y.Z., Sadrzadeh, M. and Baroni, M. (2013). Multi-step regression learning for compositional distributional semantics.
- Grefenstette, E. and Sadrzadeh, M. (2011). Experimental support for a categorical compositional distributional model of meaning.
- Hermann, K.M. and Blunsom, P. (2013). The role of syntax in vector space models of compositional semantics.
- Hermann, K. M. and Blunsom, P. (2014). Multilingual Models for Compositional Distributed Semantics.
- Kalchbrenner, N. and Blunsom, P. (2013). Recurrent convolutional neural networks for discourse compositionality.
- Kalchbrenner, N., Grefenstette, E. and Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences.
- Lazaridou, A., Marelli, M., Zamparelli, R. and Baroni, M. (2013). Compositionally derived representations of morphologically complex words in distributional semantics.

References

Compositionality (continued)

- LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series.
- Marelli, M., Menini, S., Baroni, M., Bentivogli, L., Bernardi, R. and Zamparelli, R. (2014). A SICK cure for the evaluation of compositional distributional semantic models.
- Mitchell, J. and Lapata, M. (2008). Vector-based Models of Semantic Composition.
- Socher, R., Pennington, J., Huang, E.H., Ng, A.Y. and Manning, C.D. (2011). Semi-supervised recursive autoencoders for predicting sentiment distributions.