Embeddings, NN, Deep Learning, Distributional Semantics … in NLP

Horacio Rodríguez

TALP

Outline

- Introduction
- Approaches to Semantics
- Semantic spaces
- NN models for NLP
- Embeddings
- Embedding of words
- Embedding of more complex units
- Deep Learning
- Applications
- Conclusions

Introduction

- Embedding
  - Representing meaning
    - Words meaning
    - More complex units meaning
      - Phrases, n-grams, clauses, sentences, documents
    - Approaches to semantics
  - Obtaining meaning
    - Semantic parsing
    - SRL
    - Lexical Semantics

Introduction

- What to read
  - “Embedding Methods for Natural Language Processing”, [Bordes, Weston, 2014] Tutorial at EMNLP.
  - “Statistical Language Models Based on Neural Networks”, [Mikolov, 2012], Google,
  - Yoav Goldberg (2015) “A Primer on Neural Network Models for Natural Language Processing”
  - Aurélien Bellet (2013) “Tutorial on Metric Learning”
  - Richard Socher’s tutorial on Deep Learning using Theano
    - http://deeplearning.net/tutorial/

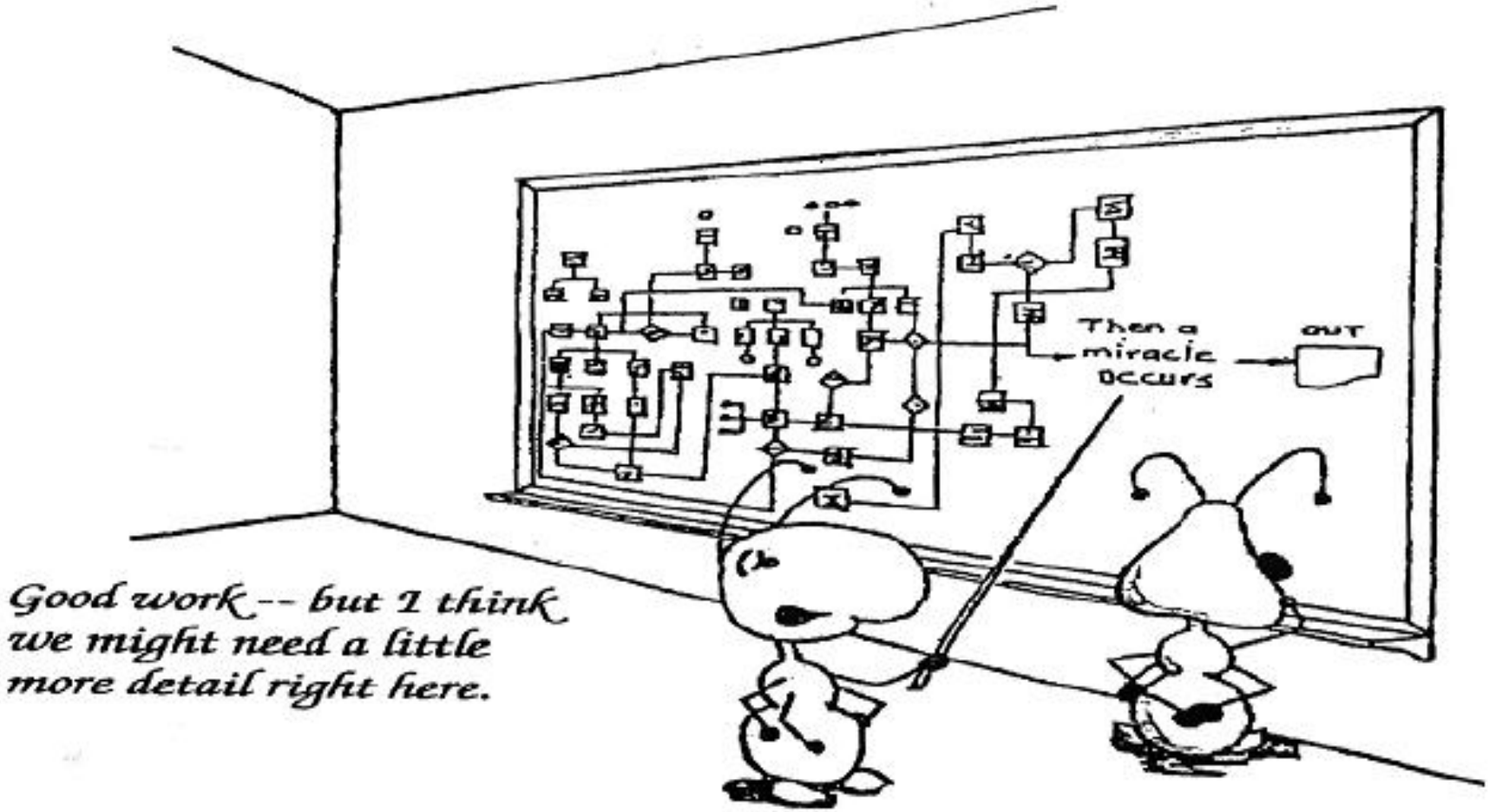
Introduction

- What to read
  - “Deep learning”, Geoffrey Hinton tutorial on IPAM 2012.
  - “Deep Learning, Graphical Models, Energy-Based Models, Structured Prediction” from Yann LeCun.
  - “Deep Learning for NLP (without Magic)” tutorial of Socher and Manning in NAACL2013
  - “Survey on Embeddings, working notes”, Horacio Rodríguez, 2016
    - https://drive.google.com/open?id=0B9kPAXXCRPn5TTlXeVdrMVpjcE0
  - http://deeplearning.net/reading-list/tutorials/

Introduction

- What to read
  - Tutorial TALP
  - Deep Learning for Speech and Language https://telecombcn-dl.github.io/2017-dlsl/

Introduction



My initial guessing

Approaches to Semantics

- Compositional Semantics
  - Semantic complex entities can be built from its simpler constituents
    - Ted Briscoe (2011) Introduction to Formal Semantics for Natural Language
    - Gennaro Chierchia and Sally McConnell-Ginet. (2000) Meaning and Grammar - an Introduction to Semantics (second edition). The MIT Press, 2000.

Approaches to Semantics

– Distributional Semantics

Distributional Hypothesis: the meaning of a word can be obtained from the company it has.

- Nice introduction in Turney & Pantel, 2010.
- M. Baroni and A. Lenci. 2010. Distributional Memory: A general framework for corpus-based semantics.
- M. Baroni and R. Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space.
- William Blacoe and Mirella Lapata. 2012. A Comparison of Vector-based Representations for Semantic Composition.

Semantic spaces

– Semantic spaces

- most recent effort towards solving this problem concern latent factor models because they tend to scale better and to be more robust w.r.t. the heterogeneity of multi-relational data.

- These models represent entities with latent factors (usually low-dimensional vectors or embeddings) and relationships as operators destined to combine them.

- Operators and latent factors are trained to fit the data using reconstruction, clustering or ranking costs.

Semantic spaces

- Ways of organizing the semantic entities
  - Distributional Semantics
    - Vectors, matrices, tensors
    - Sometimes different representations depending on POS
  - Units to be represented
    - Words,
    - Senses
    - Phrases
    - Sentences
    - Relations
      - Rdf triples

Semantic spaces

Definition (Distance function)

A distance over a set $\mathcal{X}$ is a pairwise function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which satisfies the following properties $\forall x, x', x'' \in \mathcal{X}$:

1. $d(x, x') \geq 0$ (nonnegativity),
2. $d(x, x') = 0$ if and only if $x = x'$ (identity of indiscernibles),
3. $d(x, x') = d(x', x)$ (symmetry),
4. $d(x, x'') \leq d(x, x') + d(x', x'')$ (triangle inequality).

Semantic spaces

Definition (Similarity function)

A (dis)similarity function is a pairwise function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. K is symmetric if $\forall x, x' \in \mathcal{X}$, $K(x, x') = K(x', x)$.

Semantic spaces

Minkowski distances

Minkowski distances are a family of distances induced by ℓ_p norms:

$$d_p(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_p = \left(\sum_{i=1}^{d} |x_i - x_i'|^p \right)^{1/p},$$

for $p \geq 1$. We can recover three widely used distances:

- For $p = 1$, the Manhattan distance $d_{man}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} |x_i - x_i'|$.
- For $p = 2$, the "ordinary" Euclidean distance:

$$d_{euc}(\mathbf{x}, \mathbf{x}') = \left(\sum_{i=1}^{d} |x_i - x_i'|^2 \right)^{1/2} = \sqrt{(\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x}')}$$

.

- For $p \to \infty$, the Chebyshev distance $d_{che}(\mathbf{x}, \mathbf{x}') = \max_i |x_i - x_i'|$.

Semantic spaces

- Issues with these similarity functions:
  - All the dimensions have the same importance
  - Alternative: Weighting the dimensions
  - Mahalanobis metric:

$$d_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \mathbf{M} (\mathbf{x} - \mathbf{x}')}$$

  - Original: $M = \Sigma^{-1}$, where Σ is the covariance matrix of the input vectors
  - Extended: Whatever $M \in \mathbb{R}^{dxd}$ being semidefinite positive (SDP). $M \succcurlyeq 0$

Semantic spaces

- Mahalanobis metric:

A matrix $\mathbf{M} \in \mathbb{R}^{d \times d}$ is positive semi-definite (PSD) if all its eigenvalues are nonnegative.

- If $M \succcurlyeq 0$, then $x^T M x \geq 0$
- $M = L^T L$ for some matrix L.

Equivalent to a Euclidean distance after a linear projection $\mathbf{L}$:

$$\begin{aligned} d_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') &= \sqrt{(\mathbf{x} - \mathbf{x}')^T \mathbf{M} (\mathbf{x} - \mathbf{x}')} = \sqrt{(\mathbf{x} - \mathbf{x}')^T \mathbf{L}^T \mathbf{L} (\mathbf{x} - \mathbf{x}')} \\ &= \sqrt{(\mathbf{L}\mathbf{x} - \mathbf{L}\mathbf{x}')^T (\mathbf{L}\mathbf{x} - \mathbf{L}\mathbf{x}')}. \end{aligned}$$

If $\mathbf{M}$ has rank k, $\mathbf{L} \in \mathbb{R}^{k \times d}$ (dimensionality reduction).

Semantic spaces

- Popular Similarities:

The cosine similarity measures the cosine of the angle between two instances, and can be computed as

$$K_{cos}(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x}^T \mathbf{x}'}{\|\mathbf{x}\|_2 \|\mathbf{x}'\|_2}.$$

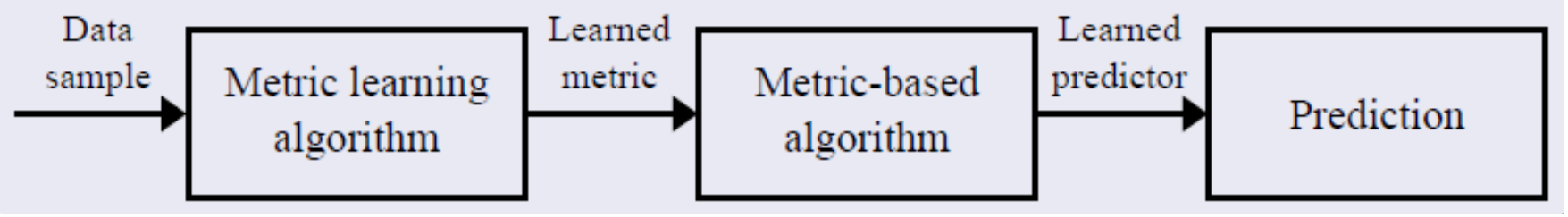
The bilinear similarity is related to the cosine but does not include normalization and is parameterized by a matrix $\mathbf{M}$:

$$K_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{M} \mathbf{x}',$$

where $\mathbf{M} \in \mathbb{R}^{d \times d}$ is not required to be PSD nor symmetric.

Semantic spaces

- Metric Learning:



- Must-link / cannot-link constraints:

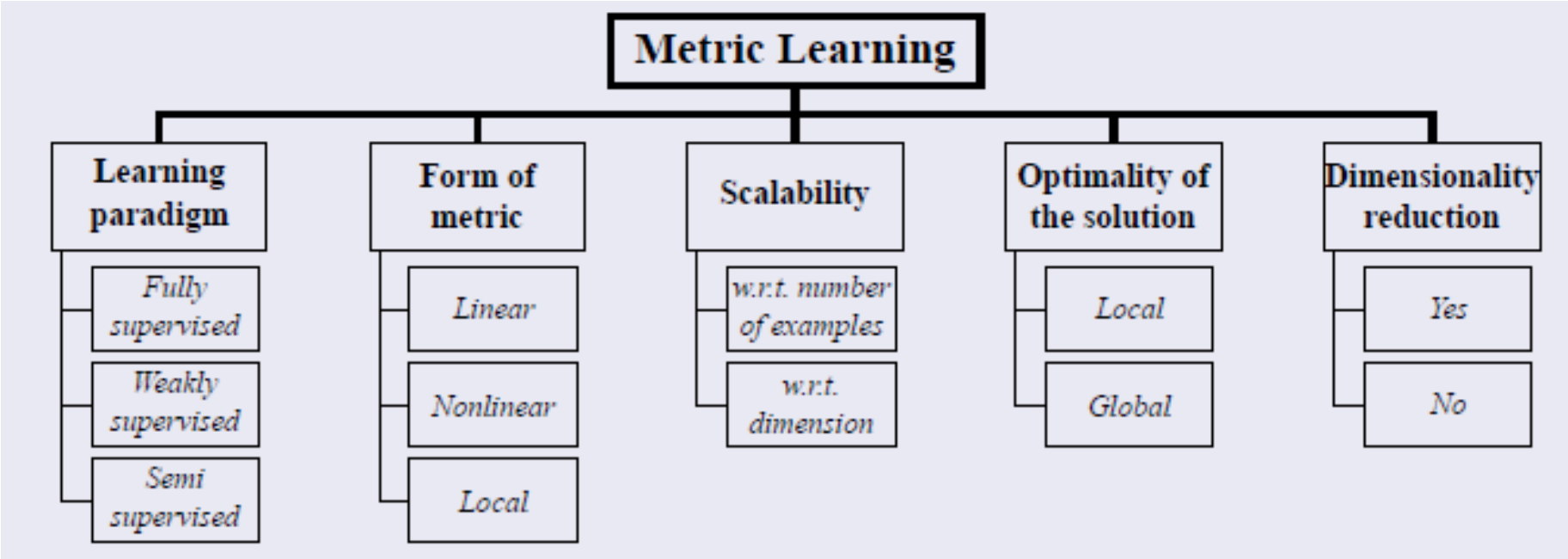
$$\begin{aligned} \mathcal{S} &= \{(x_i, x_j) : x_i \text{ and } x_j \text{ should be similar}\}, \\ \mathcal{D} &= \{(x_i, x_j) : x_i \text{ and } x_j \text{ should be dissimilar}\}. \end{aligned}$$

- Relative constraints:

$$\mathcal{R} = \{(x_i, x_j, x_k) : x_i \text{ should be more similar to } x_j \text{ than to } x_k\}.$$

Semantic spaces

- Metric Learning:



Semantic spaces

- Basic formulation:

$$\max_{\mathbf{M}\in\mathbb{R}^{d\times d}} \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{D}} d_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_j)$$

$$\text{s.t.} \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{S}} d^2_{\mathbf{M}}(\mathbf{x}_i,\mathbf{x}_j) \leq 1,$$

$$\mathbf{M} \succeq 0.$$

Look at Bellet tutorial for many other alternatives

Semantic spaces

- Metric Learning implementations in Python:
  - https://pypi.python.org/pypi/metric-learn
  - Large Margin Nearest Neighbor (LMNN)
  - Information Theoretic Metric Learning (ITML)
  - Sparse Determinant Metric Learning (SDML)
  - Least Squares Metric Learning (LSML)
  - Neighborhood Components Analysis (NCA)
  - Local Fisher Discriminant Analysis (LFDA)
  - Relative Components Analysis (RCA)

Semantic spaces

- Vector space model (VSM)
  - Similarity of Documents
    - Term $\times$ Document Matrix
  - Similarity of Words
    - Word $\times$ Context Matrix
  - Similarity of Relations
    - Pair $\times$ Pattern Matrix

Semantic spaces

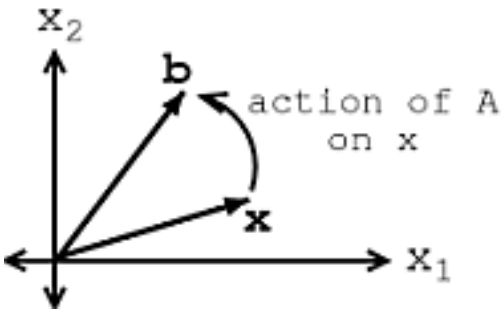
- Reducing the size of semantic spaces
  - Clustering of similar words
    - Point-wise mutual information (PMI)
    - 2nd order PMI
  - Dimensionality reduction
    - Latent Semantic Indexing, LSI (LSA, PCA, …)
      - Deerwester et al, 1988
    - pLSI
      - Hofman, 1999
  - Latent Dirichlet Allocation, LDA
    - Blei et al, 2003
  - Embeddings

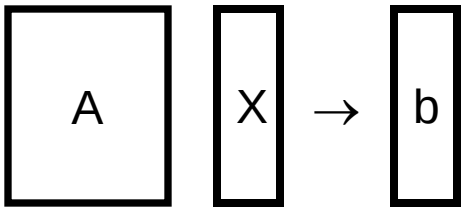
LSI

- Spectral theorem
  - Conditions for a matrix A to be diagonalizable
  - Spectral Decomposition of A
- Diagonalization
  - if A is normal (and thus hermitic and thus if real symmetric) then a decomposition exists:
    - $A = U \Lambda U^T$
    - Λ is diagonal, being its entries the eigenvalues of A
    - U is unitary being its columns the eigenvectors of A
- Singular Values Decomposition (SVD)

LSI

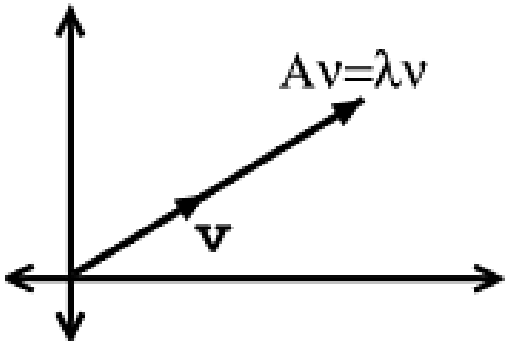
- For a square matrix *A*:
- $A\mathbf{x} = \mathbf{b}$





$A\mathbf{x} = \lambda\mathbf{x}$

where **x** is a vector (eigenvector), and λ a scalar (eigenvalue)



LSI

- SVD decomposes:
  - $A = U \Lambda V^T$
    - Λ is diagonal, being its entries the eigenvalues of A
    - U,V left and right singular vector matrices
    - U: matrix of eigenvectors of $Y=AA^T$
    - V: matrix of eigenvectors of $Y=A^TA$
    - Λ matrix m x n
    - U matrix m x r
    - V matrix n x r
    - Λ matrix r x r, $r \leq \min(m,n)$

LSI

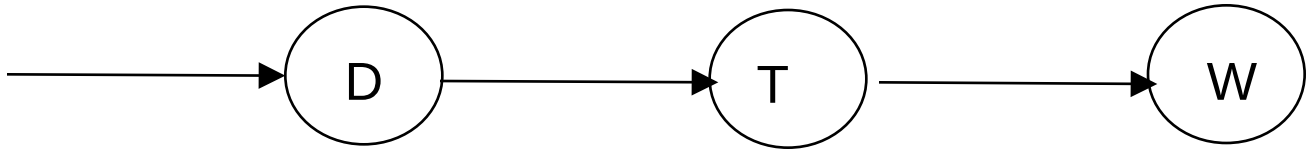
- SVD decomposes exactly:
  - $A = U \Lambda V^T$
- For dimensionality reduction we take only the k highest eigenvalues (and the corresponding eigenvectors), i.e. $U_k \Lambda_k V_k$
  - $k << r$
  - Set k that Minimizes the Frobenius norm (L_2) of A-A'.
- Applying $U_k \Lambda_k V_k^T$ we obtain A' that approximates A

LSI

- Computing distances:
  - Supose A is a term x document matrix
  - in A
    - Comparing two terms: dot product of two rows in A
    - Comparing two documents: dot product of two columns in A
    - Comparing a term and a document: entry in A
  - in A'
    - Comparing two terms: dot product of two rows in $U_k \Lambda_k$
    - Comparing two documents: dot product of two rows in $V_k \Lambda_k$
    - Comparing a term and a document: entry in A'

pLSI

- pLSI is a generative model:
  - Models each word w in a document d, as a simple of a mixture of topics.
    - $p(w) = \sum_{i=1}^{k} p(z_i) p(\mathrm{w}|z_i)$
    - $\sum_{i=1}^{k} p(z_i)$=1
  - Each word is generated by a topic, different words can be generated by different topics.

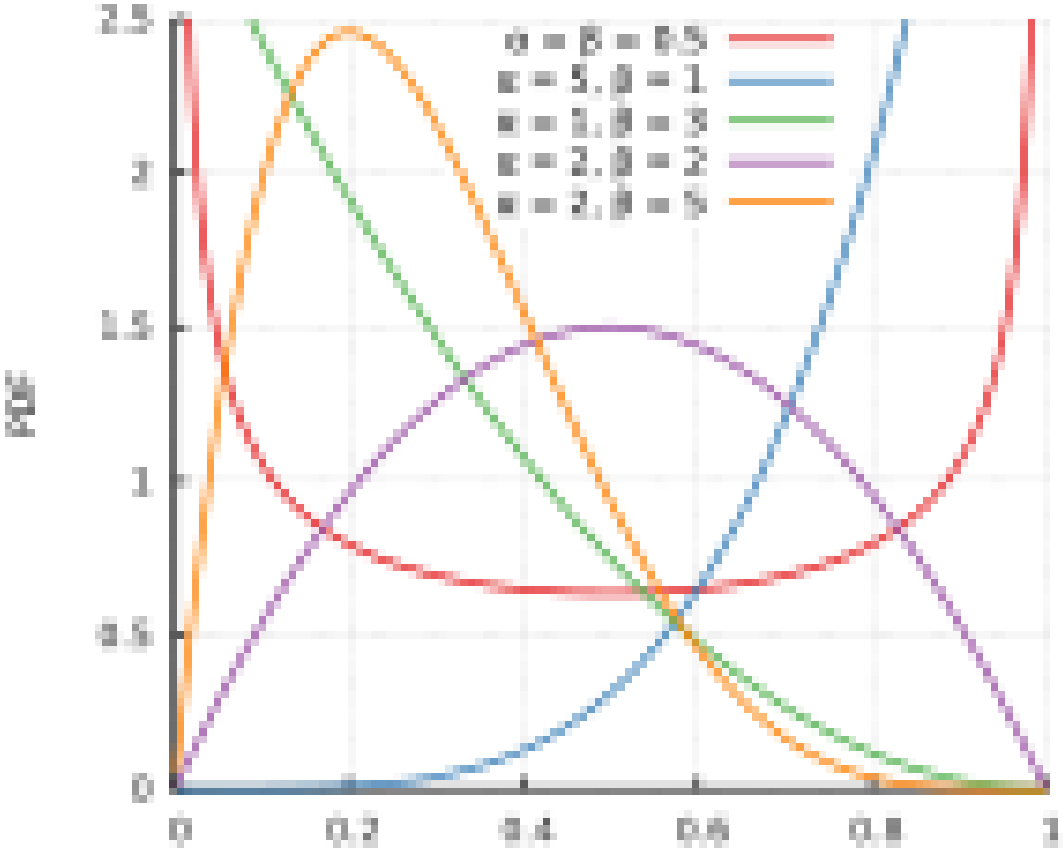


LDA

- Latent Dirichlet Allocation:
  - In the LDA model, the topic mixture proportions for each document are drawn from some distribution, i.e. a Dirichlet prior.
    - Dirichlet is a multivariate generalization of Beta distribution.
  - pLSI can be considered a LDA under an uniform Dirichlet prior distribution
  - The distribution is based on multinomials. That is, k-tuples of non-negative numbers that sum to one.

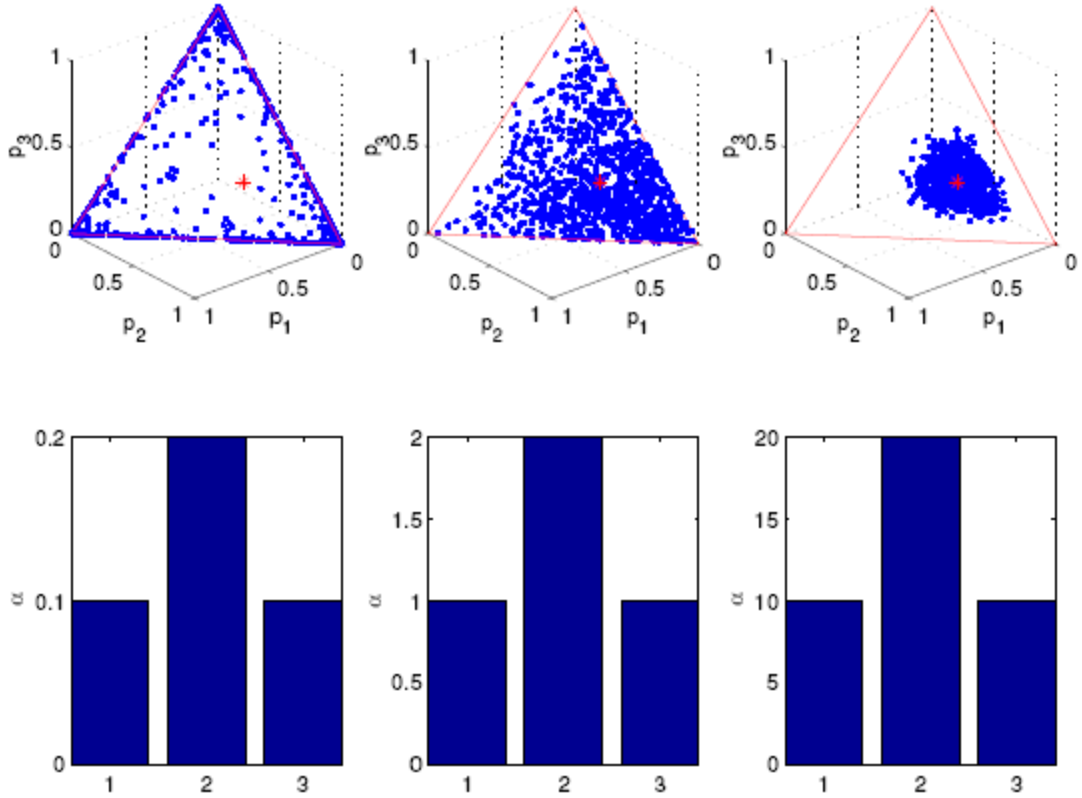
LDA

the **beta distribution** is a family of continuous probability distributions defined on the interval [0, 1] parametrized by two positive shape parameters,
denoted by α and β, that appear as exponents of the random variable and control the shape of the distribution.



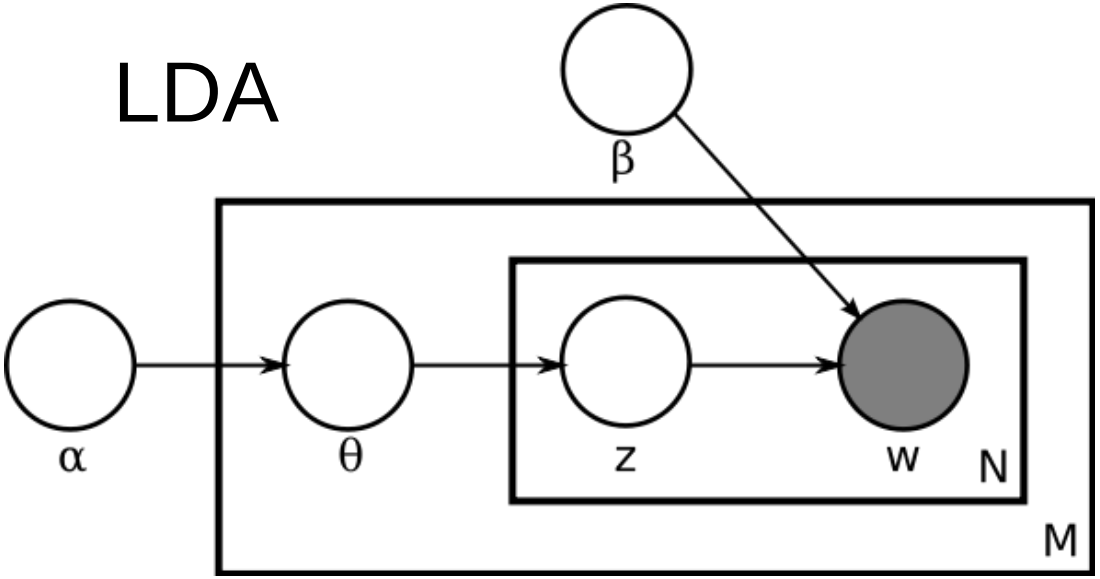
LDA

the **Dirichlet distribution**, often denoted Dir (α), is a family of continuous multivariate probability distributions parameterized by a vector α of positive reals. It is a multivariate generalization of the beta distribution



LDA

- LDA model



  - Dir(α) per document topic distribution
  - Dir(β) per word topic distribution
  - θ_i topic distribution for document i
  - z_{ij} topic for the jth word in document i according to
    - Multinomial(θ_i)
  - w_{ij} specific word

NN models for NLP

- Feed-forward neural Networks (FFNM)
- Convolutional NN (CNN)
- Recurrent NN (RNN)
- Recursive NN (RecNN)
- Long Short Time Models (LSTM)

NN models for NLP

- feed-forward neural Networks
  - general structure for an NLP classification system
    - 1 Extract a set of core linguistic features $f_1, \ldots, f_k$ that are relevant for predicting the output class.
    - 2. For each feature f_i of interest, retrieve the corresponding vector $v(f_i)$.
    - 3. Combine the vectors (either by concatenation, summation or a combination of both into an input vector x.
    - 4. Feed x into a non-linear classifier (feed-forward neural network).

NN models for NLP

- Feed-forward neural Networks
  - Collobert & Weston, Bengio & Schwenk
  - Dense Vectors vs. One-hot Representations
  - Variable Number of Features: Continuous Bag of Words
    - CBOW

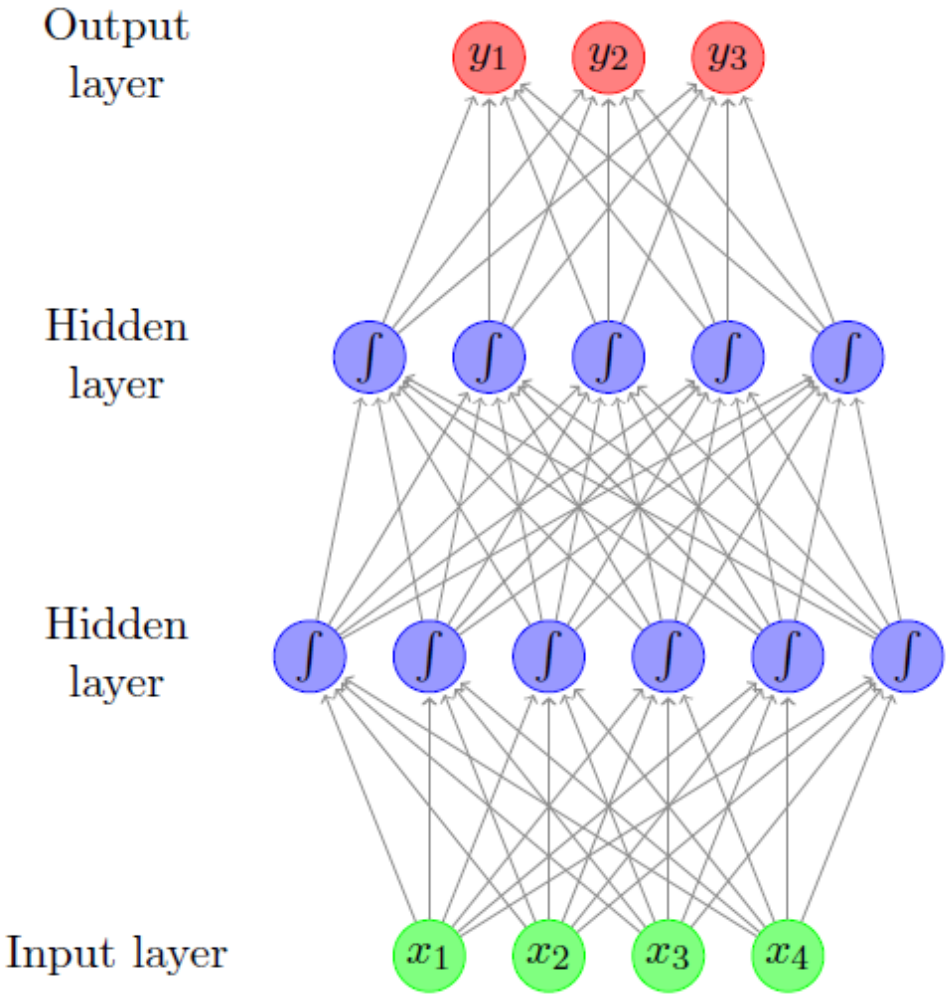
$$CBOW(f_1, ..., f_k) = \frac{1}{k} \sum_{i=1}^{k} v(f_i)$$

    - Weighted CBOW

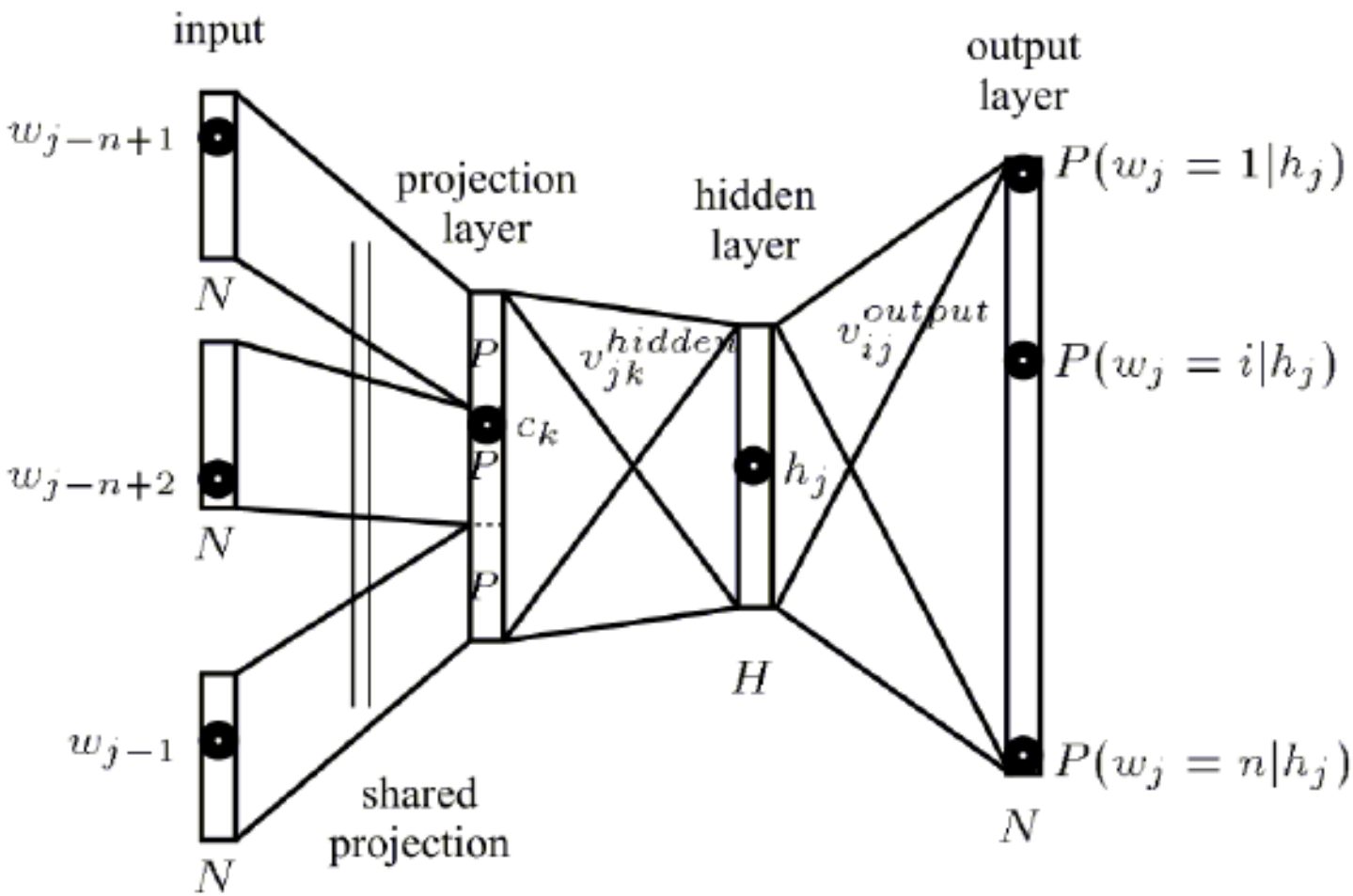
$$WCBOW(f_1, ..., f_k) = \frac{1}{\sum_{i=1}^{k} a_i} \sum_{i=1}^{k} a_i v(f_i)$$

    - Fully connected (affine) layer

NN models for NLP



NN models for NLP



N = |V|
n context size

NN models for NLP

- Feed-forward neural Networks
  - Simple Perceptron

$$NN_{Perceptron}(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}$$

$$\mathbf{x} \in \mathbb{R}^{d_{in}}, \quad \mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}, \quad \mathbf{b} \in \mathbb{R}^{d_{out}}$$

  - 1-layer Multi Layer Perceptron (MLP1)

$$NN_{MLP1}(\mathbf{x}) = g(\mathbf{x}\mathbf{W}^{\mathbf{1}} + \mathbf{b}^{\mathbf{1}})\mathbf{W}^{\mathbf{2}} + \mathbf{b}^{\mathbf{2}}$$

$$\mathbf{x} \in \mathbb{R}^{d_{in}}, \quad \mathbf{W}^{\mathbf{1}} \in \mathbb{R}^{d_{in} \times d_1}, \quad \mathbf{b}^{\mathbf{1}} \in \mathbb{R}^{d_1}, \quad \mathbf{W}^{\mathbf{2}} \in \mathbb{R}^{d_1 \times d_2}, \quad \mathbf{b}^{\mathbf{2}} \in \mathbb{R}^{d_2}$$

NN models for NLP

- Feed-forward neural Networks
  - 2-layer MLP

$$NN_{MLP2}(\mathbf{x}) = (g^2(g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)\mathbf{W}^2 + \mathbf{b}^2))\mathbf{W}^3$$

$$\begin{aligned} NN_{MLP2}(\mathbf{x}) &= \mathbf{y} \\ \mathbf{h}^1 &= g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \\ \mathbf{h}^2 &= g^2(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \\ \mathbf{y} &= \mathbf{h}^2\mathbf{W}^3 \end{aligned}$$

NN models for NLP

- Non-linearities
  - Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

  - Tanh

$$tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

  - Hard tanh

$$hardtanh(x) = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{otherwise} \end{cases}$$

  - Rectifier (ReLu)

$$ReLU(x) = \max(0, x) = \begin{cases} 0 & x < 0 \\ x & \text{otherwise} \end{cases}$$

NN models for NLP

- Output transformations
  - SoftMax

$$\mathbf{x} = x_1, \ldots, x_k$$

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

NN models for NLP

- Embedding Layers
- c(.) is a function from core features to an input vector.
- It is common for c to extract the embedding vector associated with each feature, and concatenate them

$$
\begin{aligned}
\mathbf{x} = c(f_1, f_2, f_3) &= [v(f_1); v(f_2); v(f_3)] \\
NN_{MLP1}(\mathbf{x}) &= NN_{MLP1}(c(f_1, f_2, f_3)) \\
&= NN_{MLP1}([v(f_1); v(f_2); v(f_3)]) \\
&= (g([v(f_1); v(f_2); v(f_3)]\mathbf{W^1} + \mathbf{b^1}))\mathbf{W^2} + \mathbf{b^2}
\end{aligned}
$$

NN models for NLP

- Embedding Layers
- Another common choice for c is to sum the embedding vectors (this assumes the embedding vectors all share the same dimensionality)

$$
\begin{aligned}
\mathbf{x} = c(f_1, f_2, f_3) =& v(f_1) + v(f_2) + v(f_3) \\
NN_{MLP1}(\mathbf{x}) =& NN_{MLP1}(c(f_1, f_2, f_3)) \\
=& NN_{MLP1}(v(f_1) + v(f_2) + v(f_3)) \\
=& (g((v(f_1) + v(f_2) + v(f_3))\mathbf{W^1} + \mathbf{b^1}))\mathbf{W^2} + \mathbf{b^2}
\end{aligned}
$$

NN models for NLP

- Embedding Layers
- Sometimes word embeddings $v(f_i)$ result from an “embedding layer" or “lookup layer". Consider a vocabulary of |V| words, each embedded as a d dimensional vector. The collection of vectors can then be thought of as a |V| x d embedding matrix E in which each row corresponds to an embedded feature.

$$v(f_i) = \mathbf{f_i}\mathbf{E}$$

$$CBOW(f_1, ..., f_k) = \sum_{i=1}^{k}(\mathbf{f_i}\mathbf{E}) = (\sum_{i=1}^{k}\mathbf{f_i})\mathbf{E}$$

NN models for NLP

- Loss Functions
  - Hinge (binary)

$$L_{hinge(binary)}(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$

  - Hinge (multiclass)

$$L_{hinge(multiclass)}(\hat{\mathbf{y}}, \mathbf{y}) = \max(0, 1 - (\hat{y}_t - \hat{y}_k))$$

  - Log loss

$$L_{log}(\hat{\mathbf{y}}, \mathbf{y}) = \log(1 + exp(-(\hat{y}_t - \hat{y}_k))$$

  - Categorical cross-entropy loss

$$L_{cross-entropy}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_i y_i \log(\hat{y}_i)$$

  - Ranking losses

$$L_{ranking(margin)}(\mathbf{x}, \mathbf{x}') = \max(0, 1 - (NN(\mathbf{x}) - NN(\mathbf{x}')))$$

NN models for NLP

- Learning

Algorithm 1 Online Stochastic Gradient Descent Training

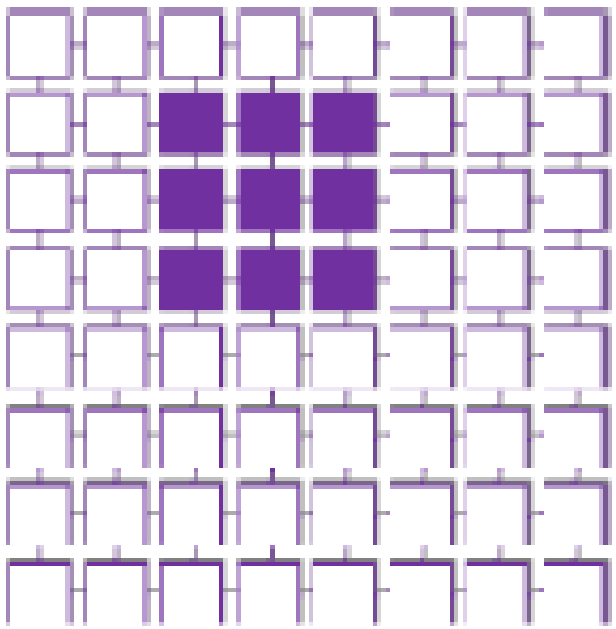
Input: Function $f(\mathbf{x};\theta)$ parameterized with parameters θ.
Input: Training set of inputs $\mathbf{x_1},\ldots,\mathbf{x_n}$ and outputs $\mathbf{y_1},\ldots,\mathbf{y_n}$.
Input: Loss function L.
while stopping criteria not met **do**
    Sample a training example $\mathbf{x_i},\mathbf{y_i}$
    Compute the loss $L(f(\mathbf{x_i};\theta),\mathbf{y_i})$
    $\hat{\mathbf{g}} \leftarrow$ gradients of $L(f(\mathbf{x_i};\theta),\mathbf{y_i})$ w.r.t θ
    $\theta \leftarrow \theta + \eta_k \hat{\mathbf{g}}$
return θ

NN models for NLP

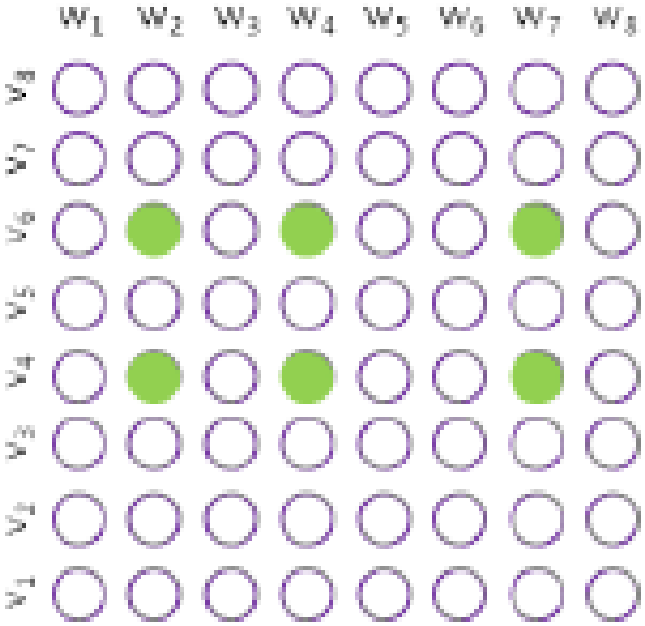
- Basic Convolution & Pooling
  - Predictions based on ordered sets of items (e.g. the sequence of words in a sentence, the sequence of sentences in a document and so on).
  - Apply a non-linear (learned) function over each instantiation of a k-word sliding window over the sentence. This function (also called filter) transforms a window of k words into a d dimensional vector that captures important properties of the words

NN models for NLP

“image patch”

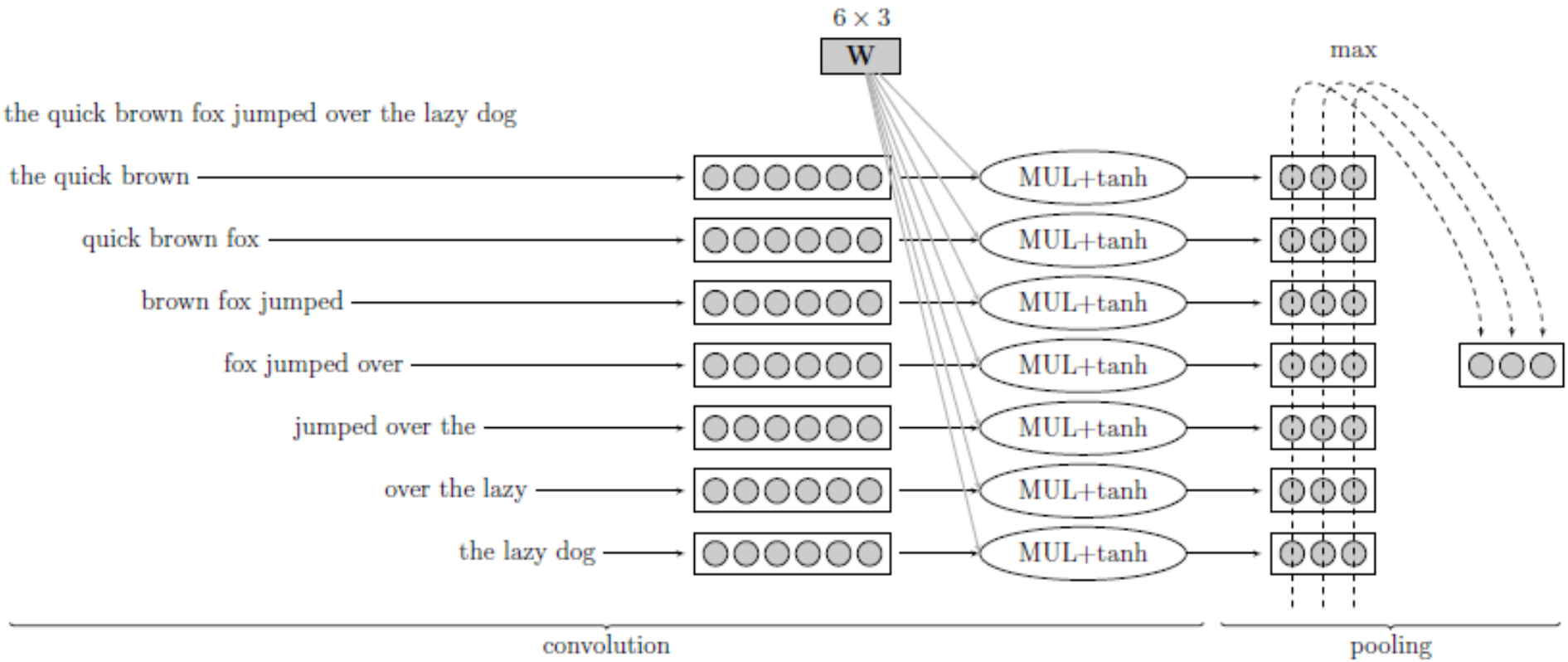


“text patch”



NN models for NLP

- Basic Convolution & Pooling



NN models for NLP

- Basic Convolution & Pooling
  - sequence of words $x = x_1, \ldots, x_n$
  - A 1d convolution layer of width k works by moving a sliding window of size k over the sentence
  - applying the same filter to each window in the sequence $(v(x_i), v(x_{i+1}), \ldots. v(x_{i+k-1}))$
  - The result of the convolution layer is m vectors
  - $P_1, \ldots. p_m$, $p_i \in \mathbb{R}^{d_{conv}}$ where:

$$\mathbf{p_i} = g(\mathbf{w_i}\mathbf{W} + \mathbf{b})$$

NN models for NLP

- Basic Convolution & Pooling
  - The m vectors are then combined using a max pooling layer, resulting in a single d_{conv} dimensional vector c.

$$c_j = \max_{1<i\leq m} \mathbf{p_i}[j]$$

  - we can split the vectors pi into distinct groups, apply the pooling separately on each group, and then concatenate the resulting d_{conv} vectors.

NN models for NLP

- Why not cycles?
  - Recurrent NN (RNN)
    - + More expressive models
    - - More costly to learn

NN models for NLP

- Recurrent NN (RNN)
  - representing arbitrarily sized structured inputs (e.g.sentences) in a fixed-size vector, while paying attention to the structured properties of the input.

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_{1:n}}, \ \mathbf{y_{1:n}}$$

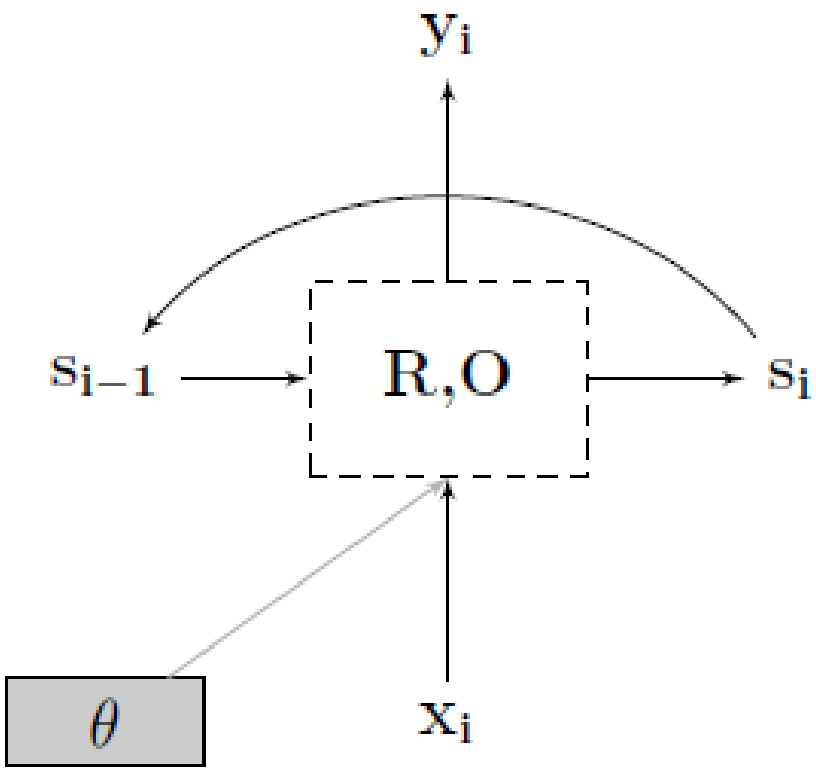
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$

$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}}, \quad \mathbf{y_i} \in \mathbb{R}^{d_{out}}, \quad \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

NN models for NLP

- Recurrent NN (RNN)



NN models for NLP

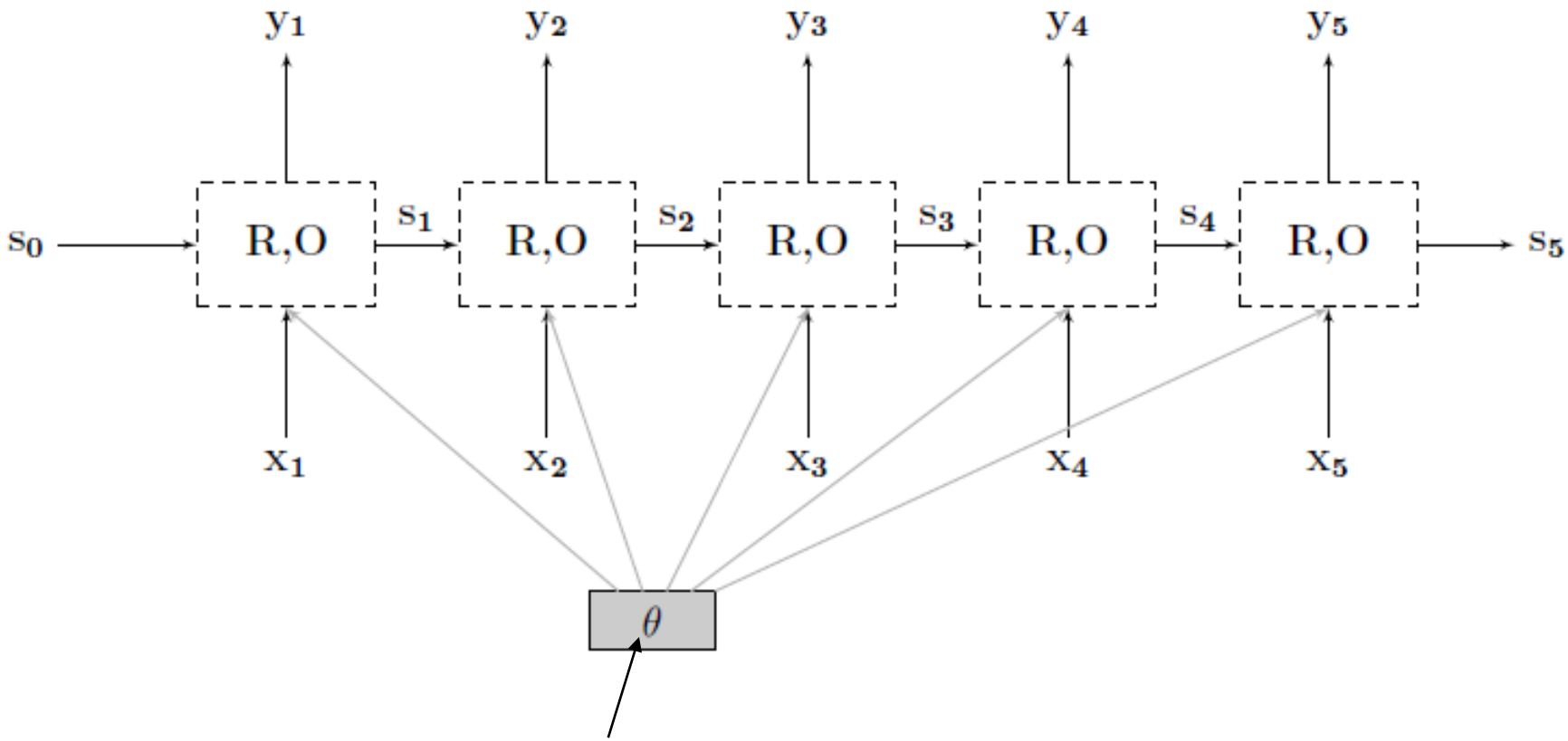
- Simple RNN architecture
  - The state at position i is a linear combination of the input at position i and the previous state, passed through a non-linear activation (commonly tanh or ReLU). The output at position i is the same as the hidden state in that position.

$$\mathbf{s_i} = R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = g(\mathbf{x_i}\mathbf{W^x} + \mathbf{s_{i-1}}\mathbf{W^s} + \mathbf{b})$$
$$\mathbf{y_i} = O_{SRNN}(\mathbf{s_i}) = \mathbf{s_i}$$

$$\mathbf{s_i}, \mathbf{y_i} \in \mathbb{R}^{d_s}, \ \mathbf{x_i} \in \mathbb{R}^{d_x}, \ \mathbf{W^x} \in \mathbb{R}^{d_x \times d_s}, \ \mathbf{W^s} \in \mathbb{R}^{d_s \times d_s}, \ \mathbf{b} \in \mathbb{R}^{d_s}$$

NN models for NLP

- Recurrent NN (RNN)

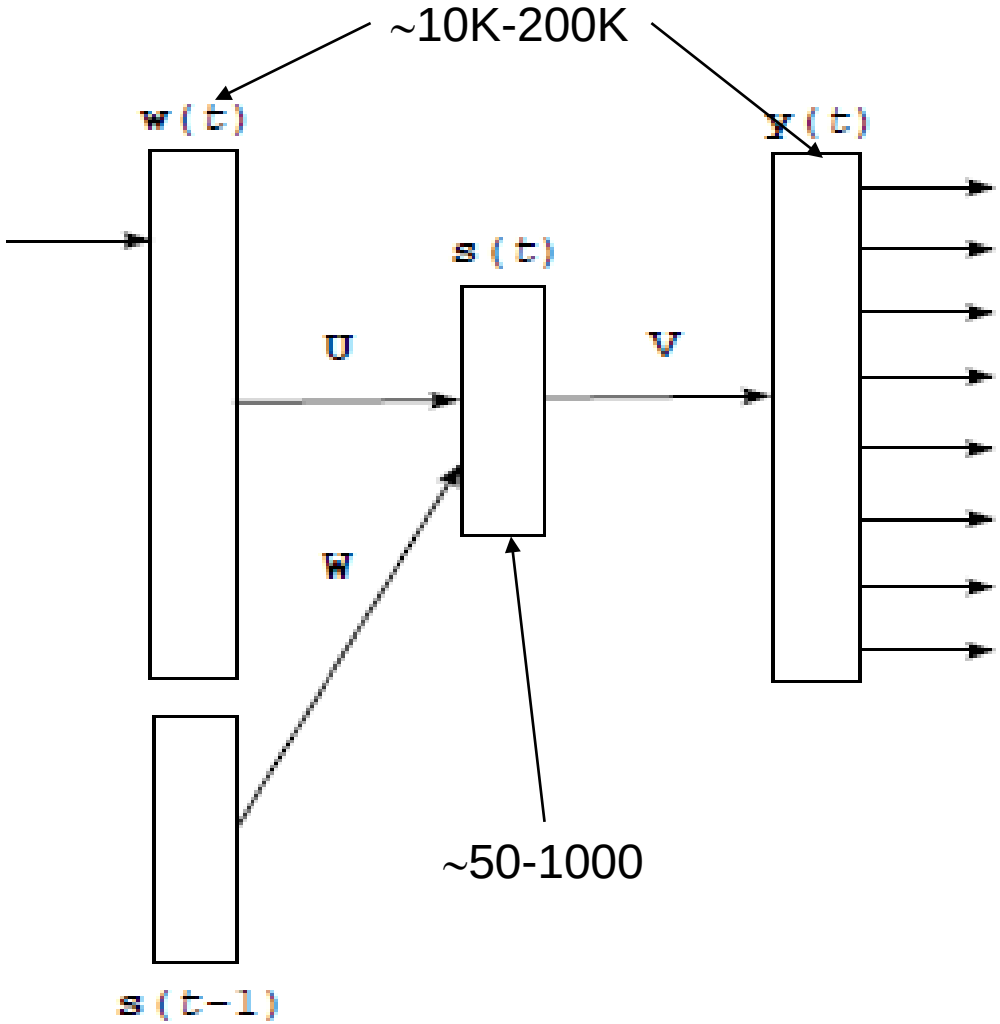


Same parameterization

NN models for NLP

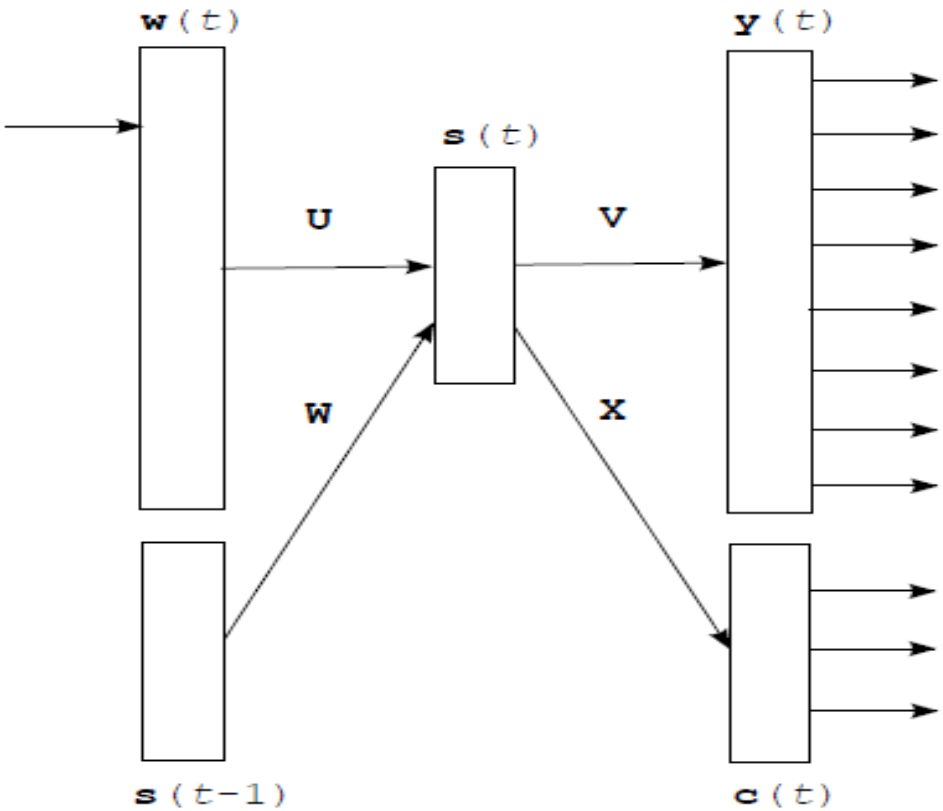
RNN

Without W bigram NN LM
s(t) = f(Uw(t) + Ws(t-1))
f sigmoid or tanh
y(t) = g(Vs(t))
g SoftMax



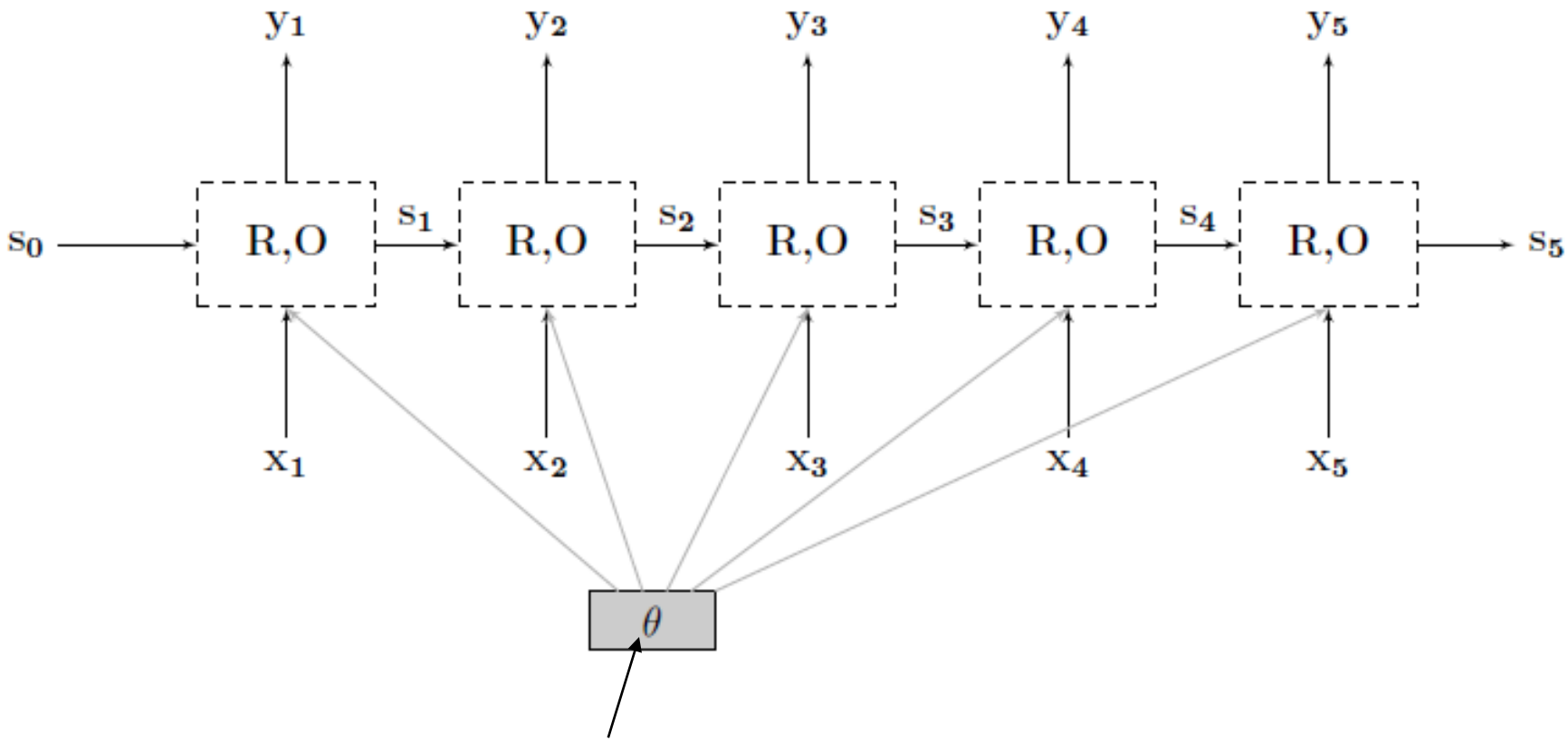
NN models for NLP

Factorization of the output layer, c(t) is the class layer. Assignment word to class



NN models for NLP

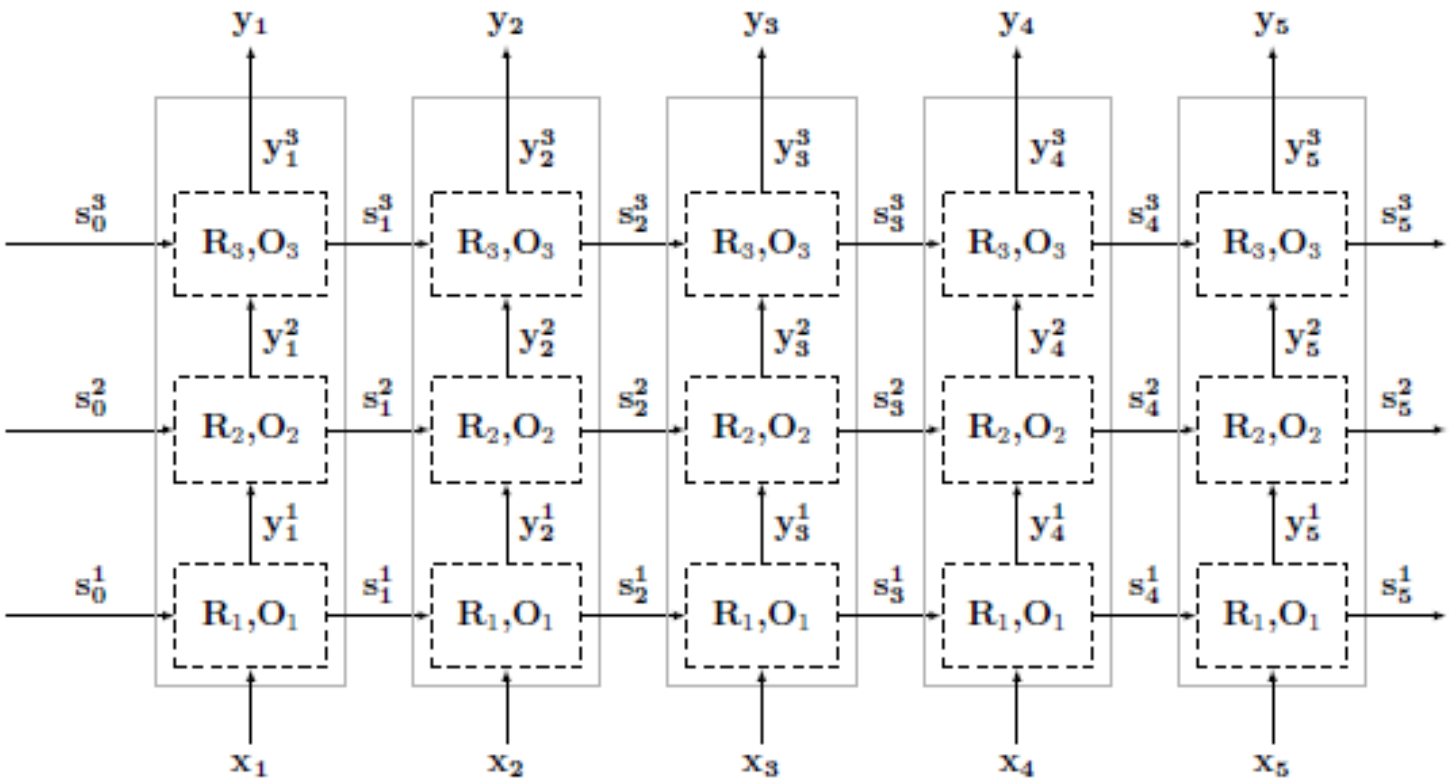
- Recurrent NN (RNN)



Same parameterization

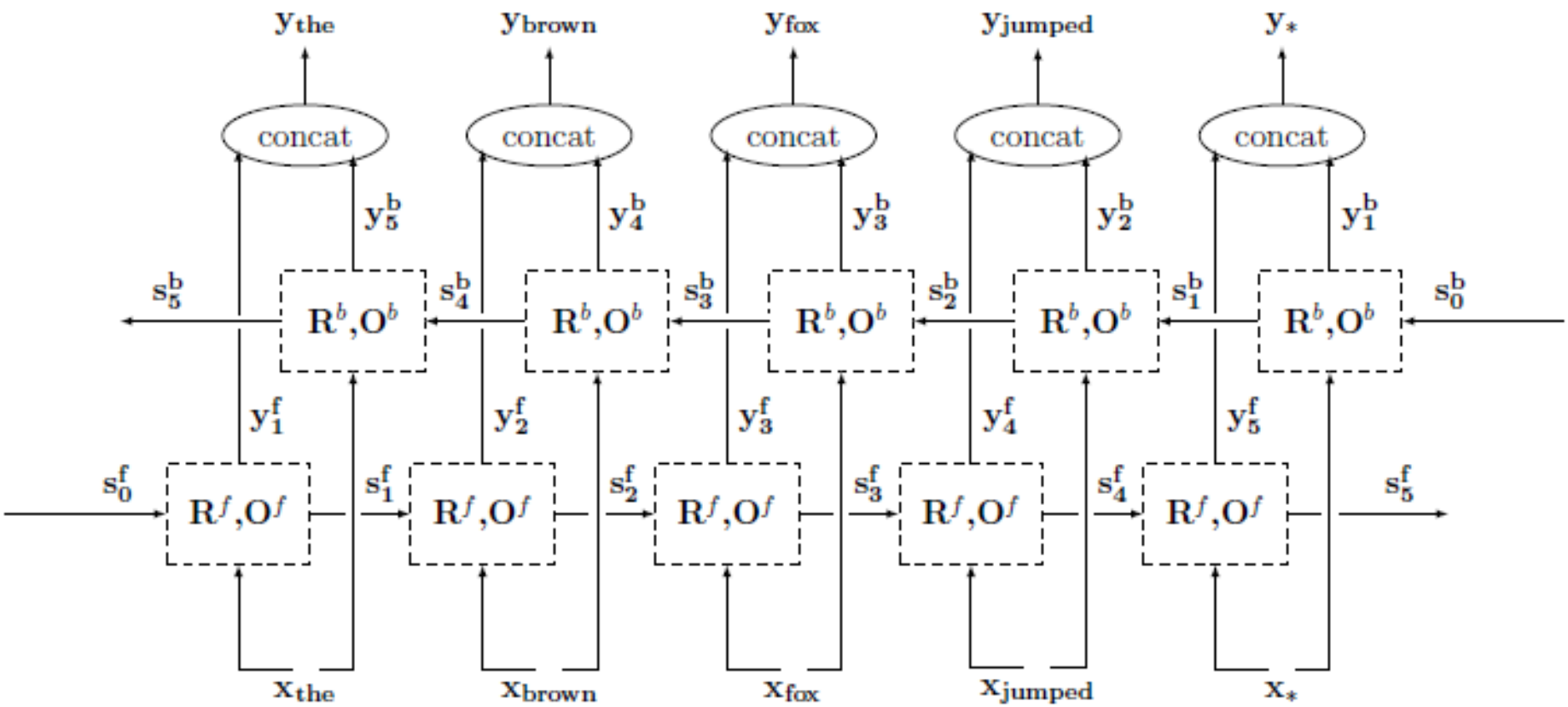
NN models for NLP

- Multi-layer (stacked) RNNs



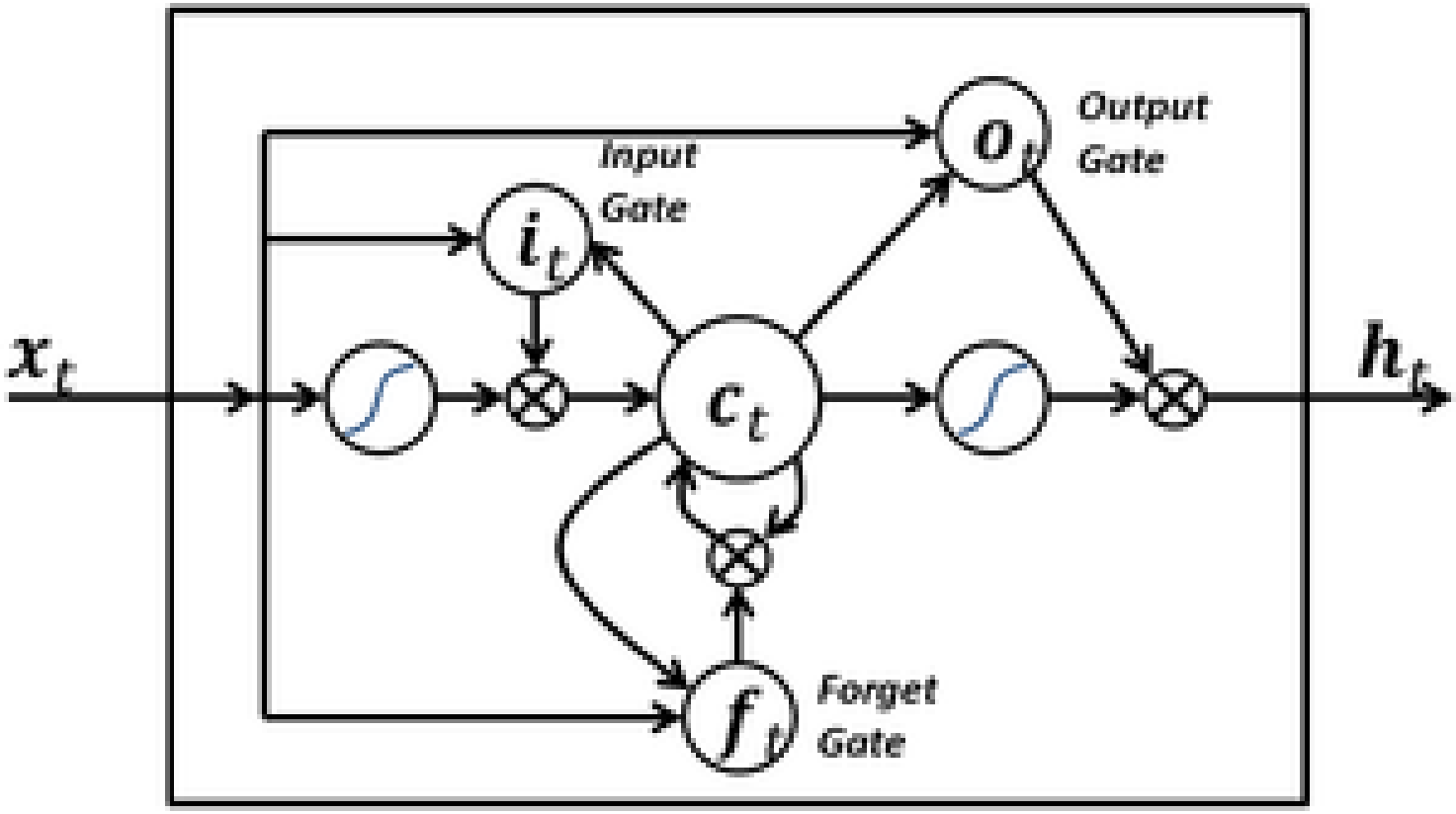
NN models for NLP

- bidirectional-RNN (BI-RNN)



NN models for NLP

- Long Short-Term Models (LSTM)

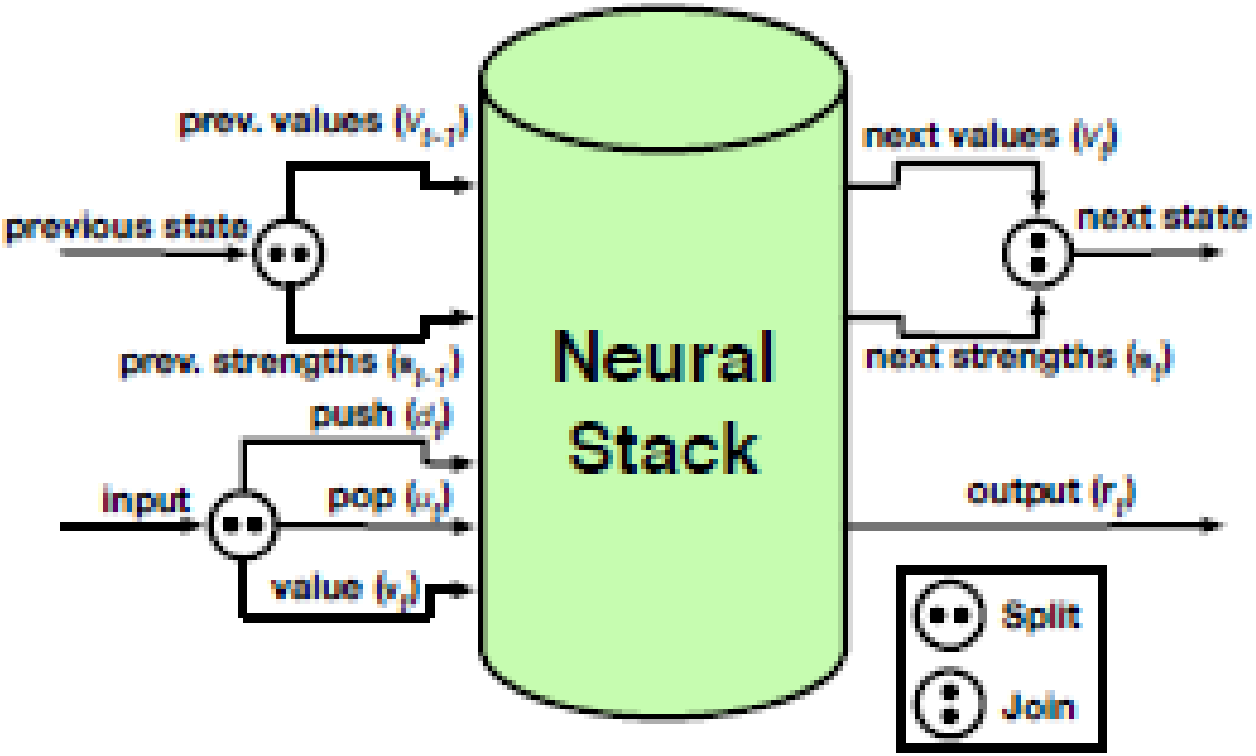


NN models for NLP

- Introducing additional dynamic memory
  - Stack-augmented RNN
    - Joulin, Mikolov, 2015
  - Other structured linear memories
    - Queues, DeQues, Double linked lists
    - Grefenstette at al, 2015
  - Neural Turing Machines
    - Graves et al, 2014
  - Memory Networks
    - Weston et al, 2015, Sukhbaatar et al, 2015

NN models for NLP

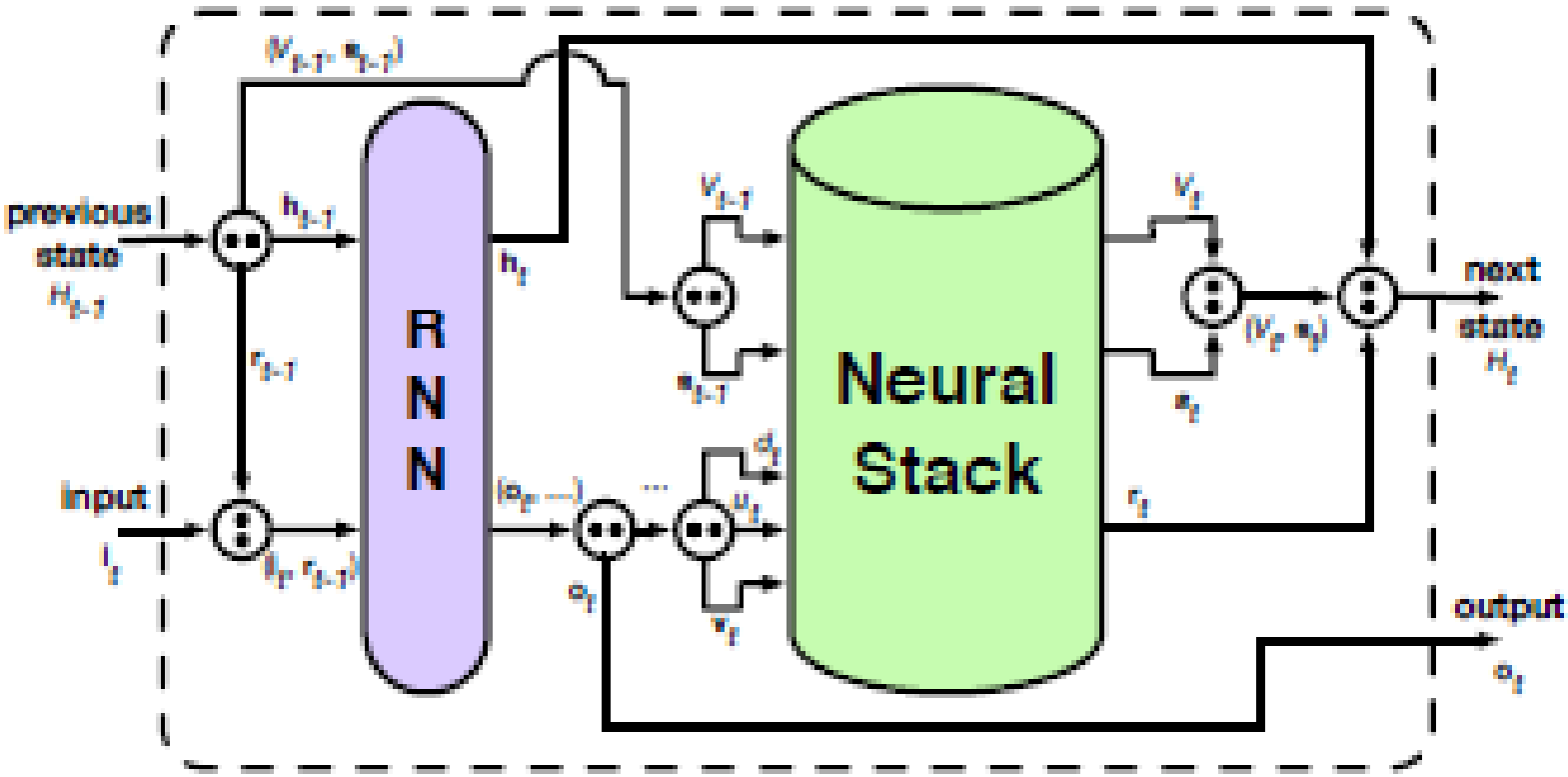
Stack-augmented RNN (Grefenstette et al, 2015)



(b) Neural Stack as a Recurrent Layer

NN models for NLP

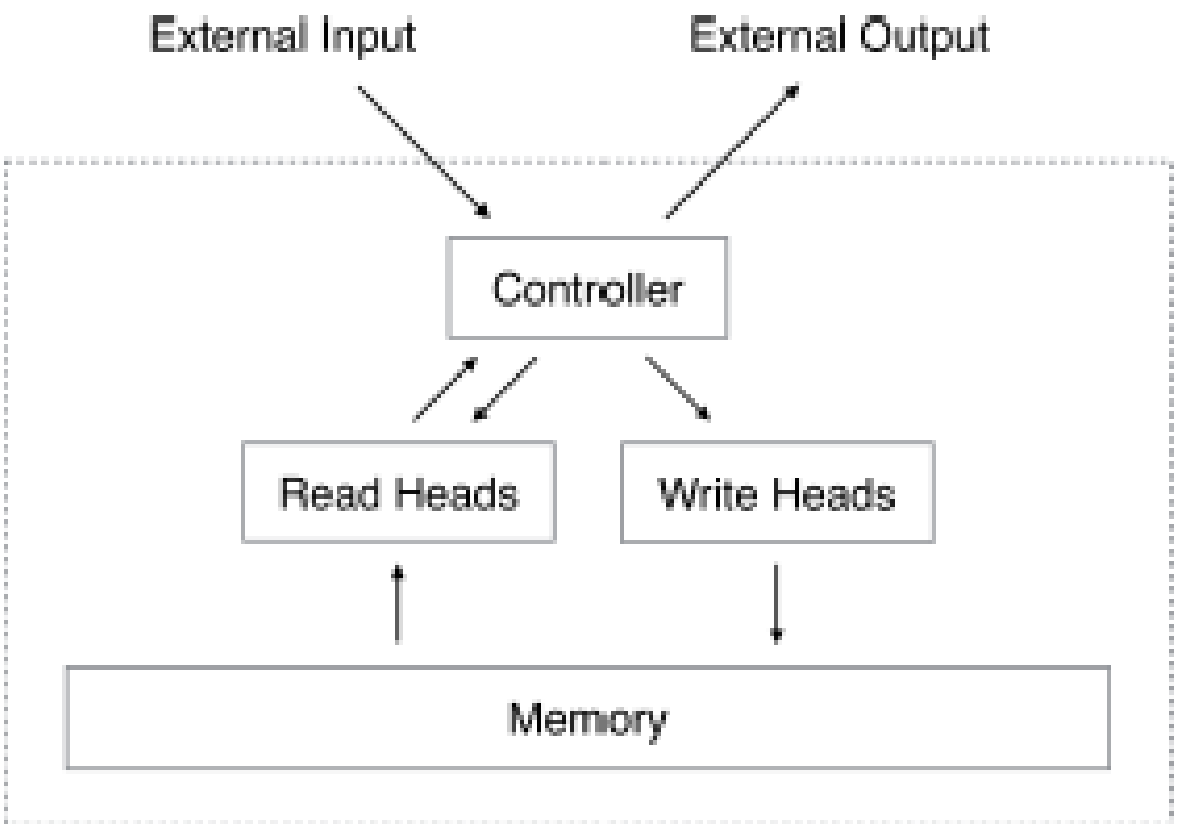
Stack-augmented RNN (Grefenstette et al, 2015)



(c) RNN Controlling a Stack

NN models for NLP

Neural Turing Machines NMM (Graves et al, 2014)



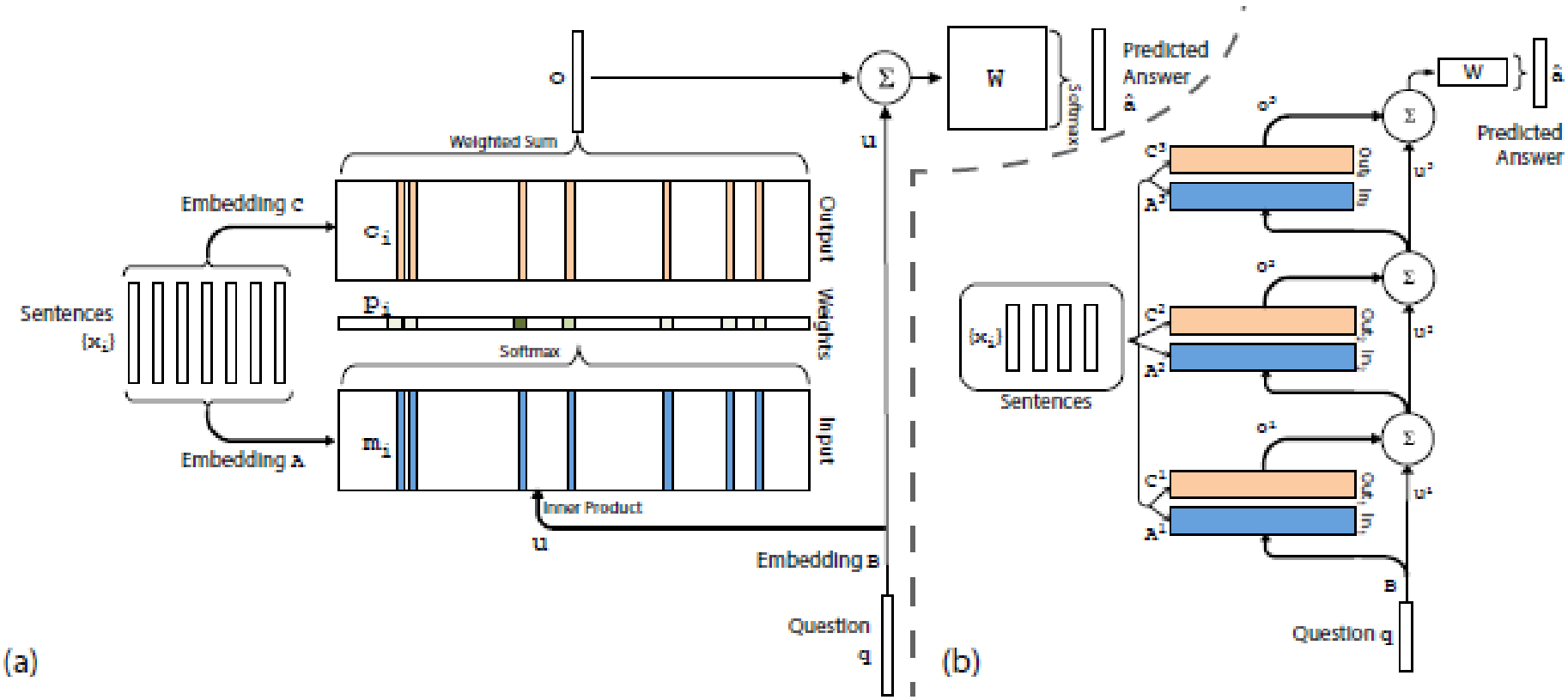
NN models for NLP

- MN (Sukhbaatar et al, 2015)

Sam walks into the kitchen.	Brian is a lion.
Sam picks up an apple.	Julius is a lion.
Sam walks into the bedroom.	Julius is white.
Sam drops the apple.	Bernhard is green.
Q: Where is the apple?	Q: What color is Brian?
A. Bedroom	A. White

NN models for NLP

- MN (Sukhbaatar et al, 2015)



NN models for NLP

- Long Short-Term Models (LSTM)

$$
\begin{aligned}
\mathbf{s_j} = R_{LSTM}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} =& \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} =& \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} =& \sigma(\mathbf{x_j}\mathbf{W^{xi}} + \mathbf{h_{j-1}}\mathbf{W^{hi}}) \\
\mathbf{f} =& \sigma(\mathbf{x_j}\mathbf{W^{xf}} + \mathbf{h_{j-1}}\mathbf{W^{hf}}) \\
\mathbf{o} =& \sigma(\mathbf{x_j}\mathbf{W^{xo}} + \mathbf{h_{j-1}}\mathbf{W^{ho}}) \\
\mathbf{g} =& \tanh(\mathbf{x_j}\mathbf{W^{xg}} + \mathbf{h_{j-1}}\mathbf{W^{hg}})
\end{aligned}
$$

$$
\mathbf{y_j} = O_{LSTM}(\mathbf{s_j}) = \mathbf{h_j}
$$

$$
\mathbf{s_j} \in \mathbb{R}^{2 \cdot d_h}, \ \ \mathbf{x_i} \in \mathbb{R}^{d_x}, \ \ \mathbf{c_j}, \mathbf{h_j}, \mathbf{i}, \mathbf{f}, \mathbf{o}, \mathbf{g} \in \mathbb{R}^{d_h}, \ \ \mathbf{W^{xo}} \in \mathbb{R}^{d_x \times d_h}, \ \ \mathbf{W^{ho}} \in \mathbb{R}^{d_h \times d_h},
$$

Embeddings

- What is an embedding?
  - A mapping of an element of a textual space (a word, a phrase, a sentence, an entity, a relation, a predicate with or without arguments, an rdf triple, an image , etc.) into an element of a, frequently low dimensional, vectorial space.
  - Within NLP, initial embedding models were first applied to words, basically for building accurate language models (e.g. ASR, MT).
  - Lexical embeddings can serve as useful representations for words for a variety of NLP tasks, but learning embeddings for phrases or other complex linguistic units, can be challenging but needed for many NLP tasks.
  - While separate embeddings are learned for each word, this is infeasible for every phrase. $|V| \sim 10^5$

Embeddings of words

- Word Embeddings
  - A **word representation** is a mathematical object associated with each word, often a **vector**. Each dimension's value corresponds to a feature and might even have a semantic or grammatical interpretation, i.e. each dimension represents a syntactic or semantic property of the word, so we can call it a word feature.

Embeddings of words

- Word Embeddings
  - Conventionally, supervised lexicalized NLP approaches take a word and convert it to a symbolic ID, which is then transformed into a feature vector using a **one-hot representation**. The feature vector has the same length as the size of the **vocabulary**, and only one dimension is on. However, the one-hot representation of a word suffers from data sparsity. For facing this problem several approaches have been followed.

x = [0 0 0 … 1 … 0 0 0]

Embeddings of words

- Learning Word Embeddings
  - The simpler word embedding system consists of mapping a word token into a vector of reals.
    - $\mathbf{w} \rightarrow \mathbf{w} \in \mathbb{R}^n$, where n is the dimension of the embedding space.
  - [Mikolov et al, 2013] **Word2Vec**
  - [Pennington et al, 2014] consider that the two main model families for learning word vectors are:
    - global matrix factorization methods, such as **LSI**, **LDA**, and their derivates (e.g. **pLSI**, **sLDA**)
    - local context window methods, such as the **skip-gram** or the **CBOW** models of [Mikolov et al. 2013]. **Word2Vec**
  - [Pennington et al, 2014] global log-bilinear regression model, **GloVe.**

Embeddings of words

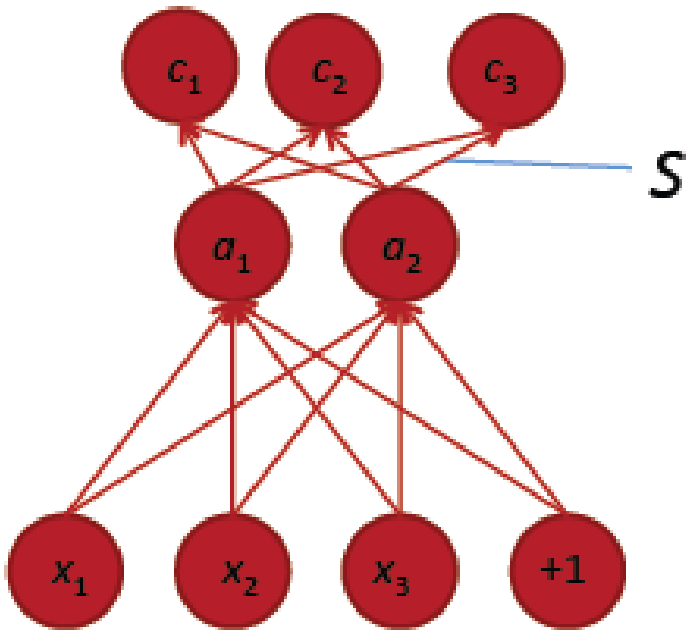
- Learning Word Embeddings
  - The **skip-gram** and continuous bag-of-words (**CBOW**) models of Mikolov et al. (2013) propose a simple single-layer architecture based on the inner product between two word vectors.
  - Mnih and Kavukcuoglu (2013) also proposed closely-related vector log-bilinear models, **vLBL** and **ivLBL**
  - Levy et al. (2014) proposed explicit word embeddings based on a **PPMI** metric.
  - All these systems use **unsupervised** learning
  - An interesting system, using **semi-supervised** learning approach, is [Turian et al, 2010]. This system combines different word representations and use also clustering techniques.

Embeddings of words

- Learning Word Embeddings
  - A word and its context is a positive training example, a random word in that same context gives a negative training example.
  - Each word is associated with a n-dimensional vector (context, n-gram, sentence, …)
  - Word embedding matrix (word + context)
  - $\mathbf{L} \in \mathbb{R}^{n \times |V|}$

Embeddings of words

- Shallow (Word level) NLP tasks
  - Pos tagging, NER
  - C&W, SENNA
    - Similar to word vector learning but adding a SoftMax (supervised) layer
    - Training with BackPropagation



Embeddings of words

- Some examples downloadble
  - Glove, [Pennington et al, 2014],
    - http://nlp.stanford.edu/projects/glove/
  - [Turian et al, 2010]
    - http://metaoptimize.com/projects/wordreprs/
  - Senna, [Collobert, Weston, 2008
    - http://ml.nec-labs.com/senna/
  - HPCA, [Lebret, Collobert, 2014]
    - http://www.lebret.ch/words/
  - HLBL, [Mnih, Hinton, 2007]
    - http://www.cnts.ua.ac.be/conll2003/ner/
  - Word2vec, [Mikolov et al, 2013]
    - Code.google.com/p/word2vec

Embeddings of phrases

- Phrase Embeddings
  - Usually approached following two lines:
    - Using pre-defined composition operators (Mitchell and Lapata, 2008), e.g., component-wise sum/multiplication, we learn composition functions that rely on phrase structure and context.
  - Using matrices/tensors as transformations (Socher et al., 2011), (Socher et al., 2013).
    - A popular system is Socher's use of RNN, which recursively computes phrase embeddings based on the binary sub-tree associated with the phrase with matrix transformations. For a phrase p = (w_1,w_2). The model then computes the phrase embedding $\mathbf{e_p}$ as:

$$e_p = \sigma\left(W \cdot [e_{w_1} : e_{w_2}]\right)$$

Embeddings of phrases

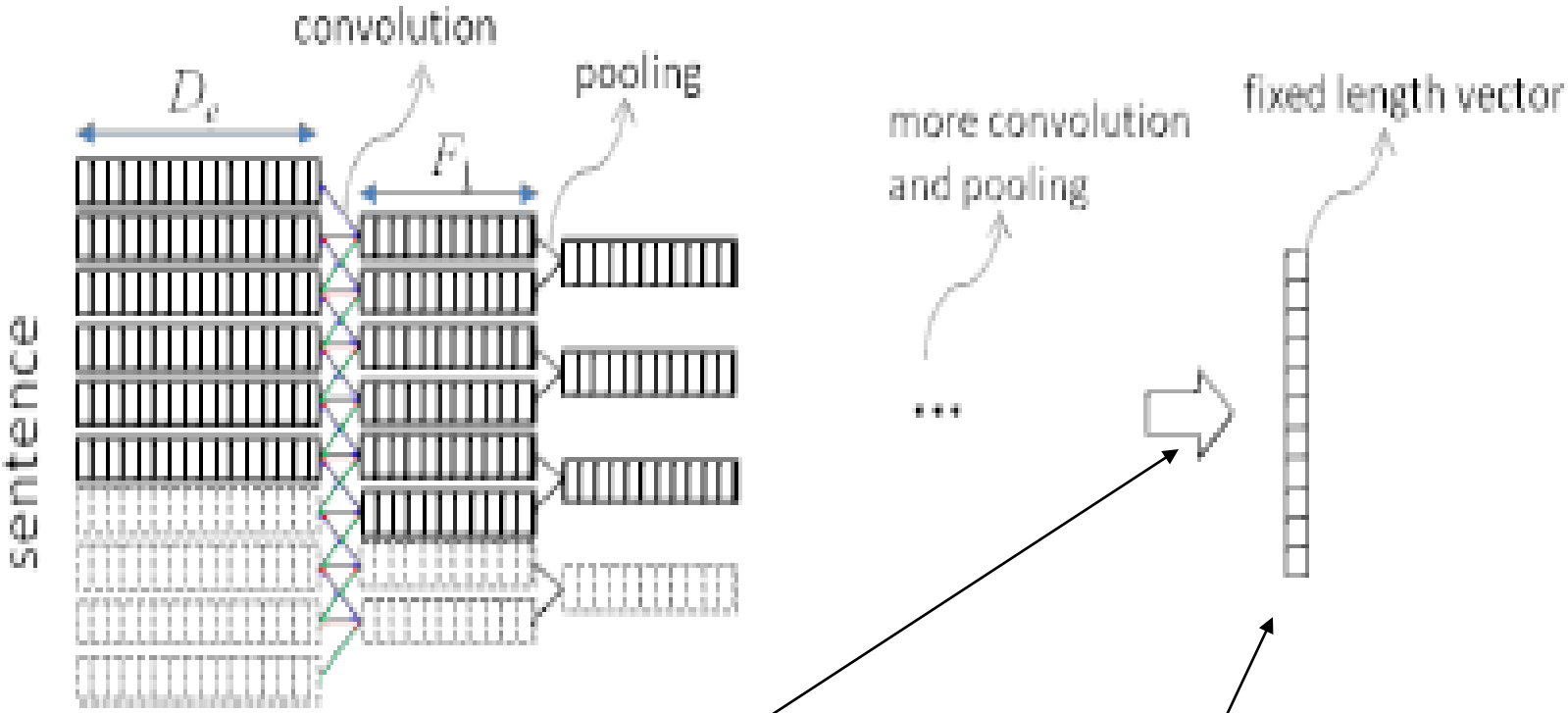
- **Neural sentence models**
  - Problem of variable sentence length.
  - Represent the variable length sentence as a fixed-length vector. These models generally consist of a **projection layer** that maps words, subword units or n-grams to high dimensional embeddings (often trained beforehand with unsupervised methods); the latter are then combined with the different architectures of neural networks, such as Neural Bag-of-Words (**NBOW**), recurrent neural network (**RNN**), recursive neural network (**RecNN**), convolutional neural network (**CNN**) and so on.

Embeddings of phrases

- **Convolutional Sentence Model**
  - [Hu et al, 2015] is a new convolutional architecture for modeling sentences. It takes as input the embedding of words (often trained beforehand with unsupervised methods) in the sentence aligned sequentially, and summarize the meaning of a sentence through layers of convolution and pooling, until reaching a fixed length vectorial representation in the final layer.
  - This is an interesting example of deep learning architecture.

Embeddings of phrases

Hu et al's overall architecture of the convolutional sentence model



Non linear function (tanh)

Embedding of the sentence

Embeddings of more complex units

- Some examples downloadble
  - Phrases: FCT, [Yu,Dredze, 2015] ,
    - https://github.com/Gorov/FCT_PhraseSim_TACL
  - Entities, Triples, Questions: [Bordes et al, 2014]
  - Words, Entities, Triples, WN synsets, [Bordes et al, 2012]
  - Multi-relational Data, Entities, relationships: TransE, [Bordes et al, 2013]
  - Structured embeddings of KBs, WN, FreeBase: [Bordes et al, 2011]
  - Sentences: FCM, [Yu et al, 2014]
    - https://github.com/Gorov/FCM_nips_workshop/
  - Sentences: LRFCM, [Yu et al, 2015]
    - https://github.com/Gorov/ERE_RE

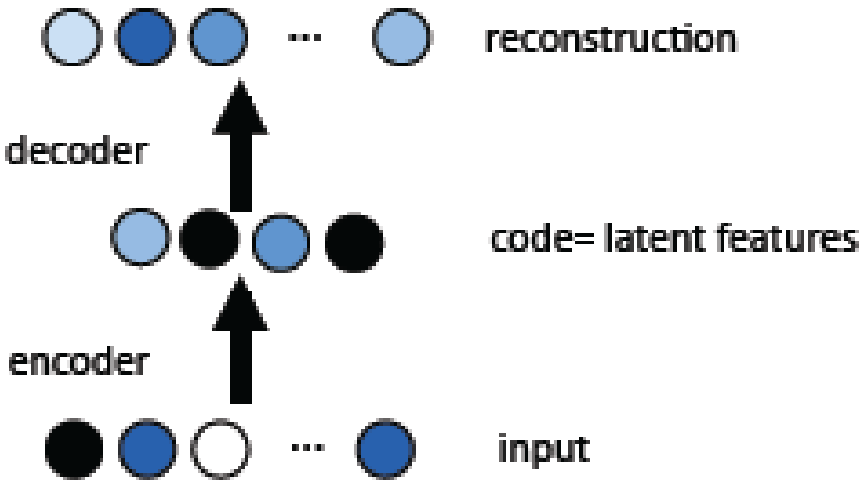
Embeddings of more complex units

- Recursive Autoencoders (RAE)
  - From Socher et al, 2011, used for paraphrase detection (see later)
  - A RAE is a multilayer recursive neural network with input = output
  - It learns feature representations for each node in the tree such that the word vectors underneath each node can be recursively reconstructed.
  - Reconstruction = decoder(encoder(input))

$A = \tanh(Wx+b)$

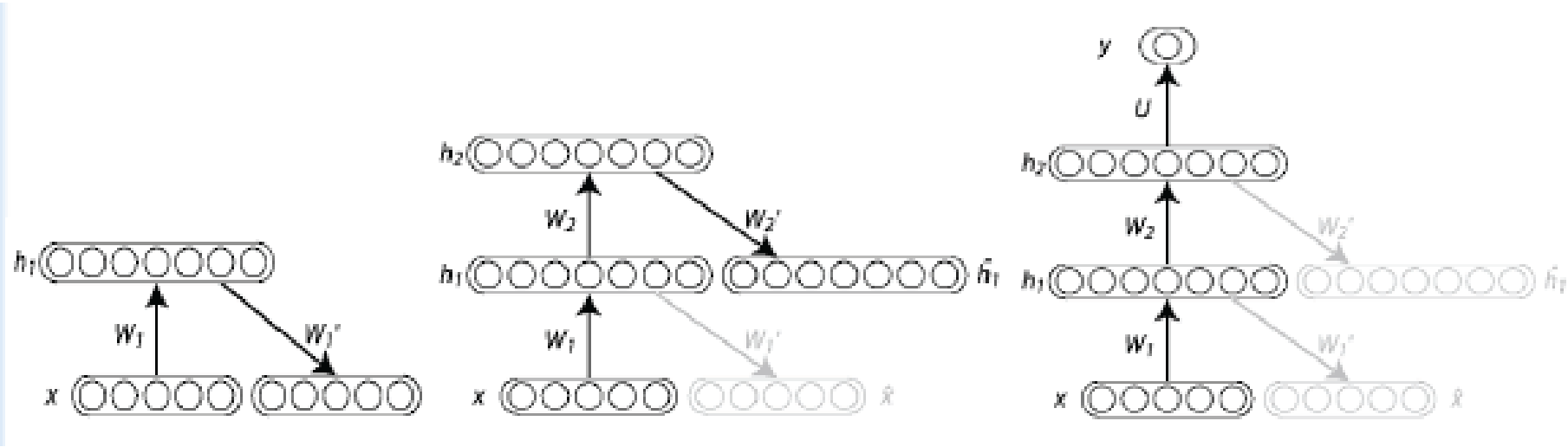
$x' = \tanh(W^T a+c)$

Cost of reconstruction
$= ||x' - x||^2$

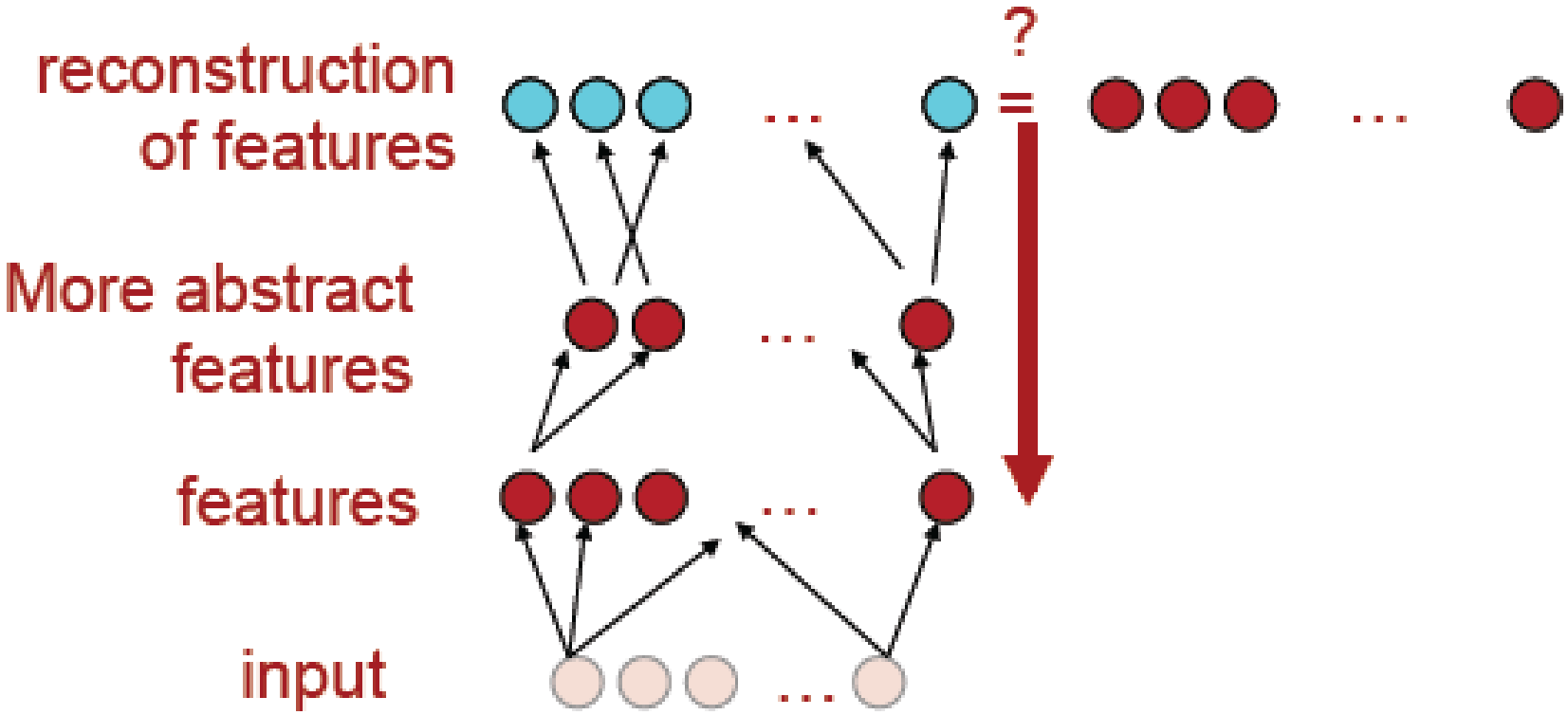


Embeddings of more complex units

- Stacking RAE
  - RAE can be stacked to form highly non-linear representations



Embeddings of more complex units



Deep Learning

- Deep Learning motivation
  - From Socher & Manning, 2013
  - Frequently ML uses handcrafted representations as features, in a way that ML task consists of optimizing weights.
  - Most practical NLP ML methods use supervised approaches
    - $\rightarrow$ need of labeled training data
    - But almost all data is unlabeled
  - Representation learning attempts to automatically learn good features or representations
  - Deep learning attempts to learn multiple levels of representation of increasing complexity/abstraction
  - At least as pre-training or in the most superficial layers unsupervised learning is used in Deep Learning

Deep Learning

- Deep Learning Basics
  - From Hinton, 2012
  - Greedy learning
    - Because of overfitting many parameters
      - $\rightarrow$ Regularization
      - Diffusion of gradient
    - Learning multilayer generative models of unlabelled data by learning one layer of features at a time
    - Hinton's layer-based pre-training

Deep Learning

- Deep Learning Basics
  - It is easy to generate an unbiased example at the leaf nodes, so we can see what kinds of data the network believes in.
  - It is hard to infer the posterior distribution over all possible configurations of hidden causes.
  - It is hard to even get a sample from the posterior.
  - We need a way of learning one layer at a time that takes into account the fact that we will be learning more hidden layers later.
  - Possibility: Learning Feature Extraction layers and classifier layer into two independent steps

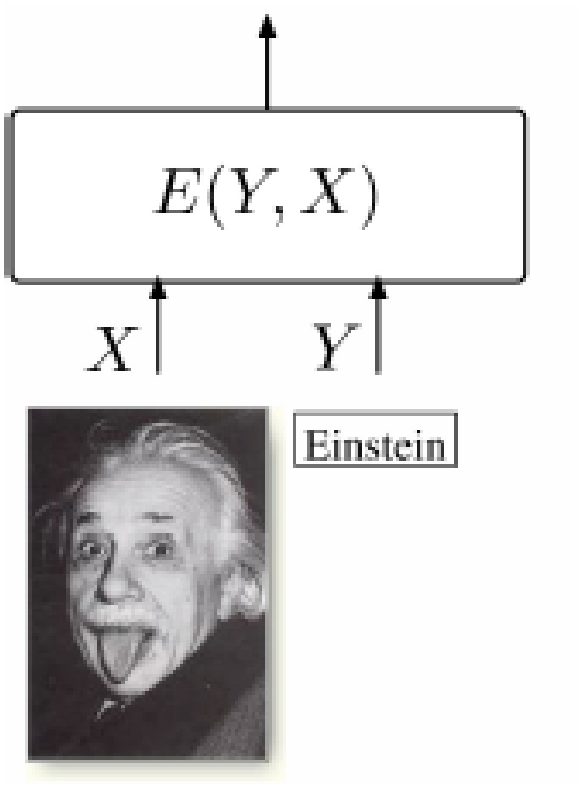
Deep Learning

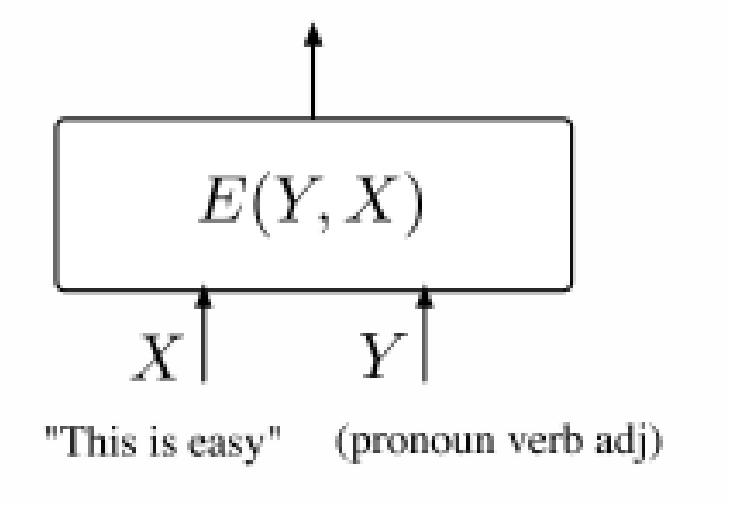
- Energy-based learning
  - From LeCun, 2012
  - Measures the compatibility between an observed variable X and a variable to be predicted Y through an energy function E(Y,X).
  - Search for the Y that minimizes the energy within a set

$$Y^* = \mathrm{argmin}_{Y \in \mathcal{Y}} E(Y, X)$$

Deep Learning

Energy-based learning





Deep Learning

- Energy-based learning
  - Family of energy functions
  - Training set
  - Loss functional / Loss function

$$\mathcal{E} = \{E(W, Y, X) : \quad W \in \mathcal{W}\}$$

$$\mathcal{S} = \{(X^i, Y^i) : i = 1 \ldots P\}$$

$$\mathcal{L}(E, \mathcal{S})$$

$$\mathcal{L}(E, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} L(Y^i, E(W, \mathcal{Y}, X^i)) + R(W).$$

Per-sample loss

Desired answer

Energy surface for a given Xi as Y varies

Regularizer

Deep Learning

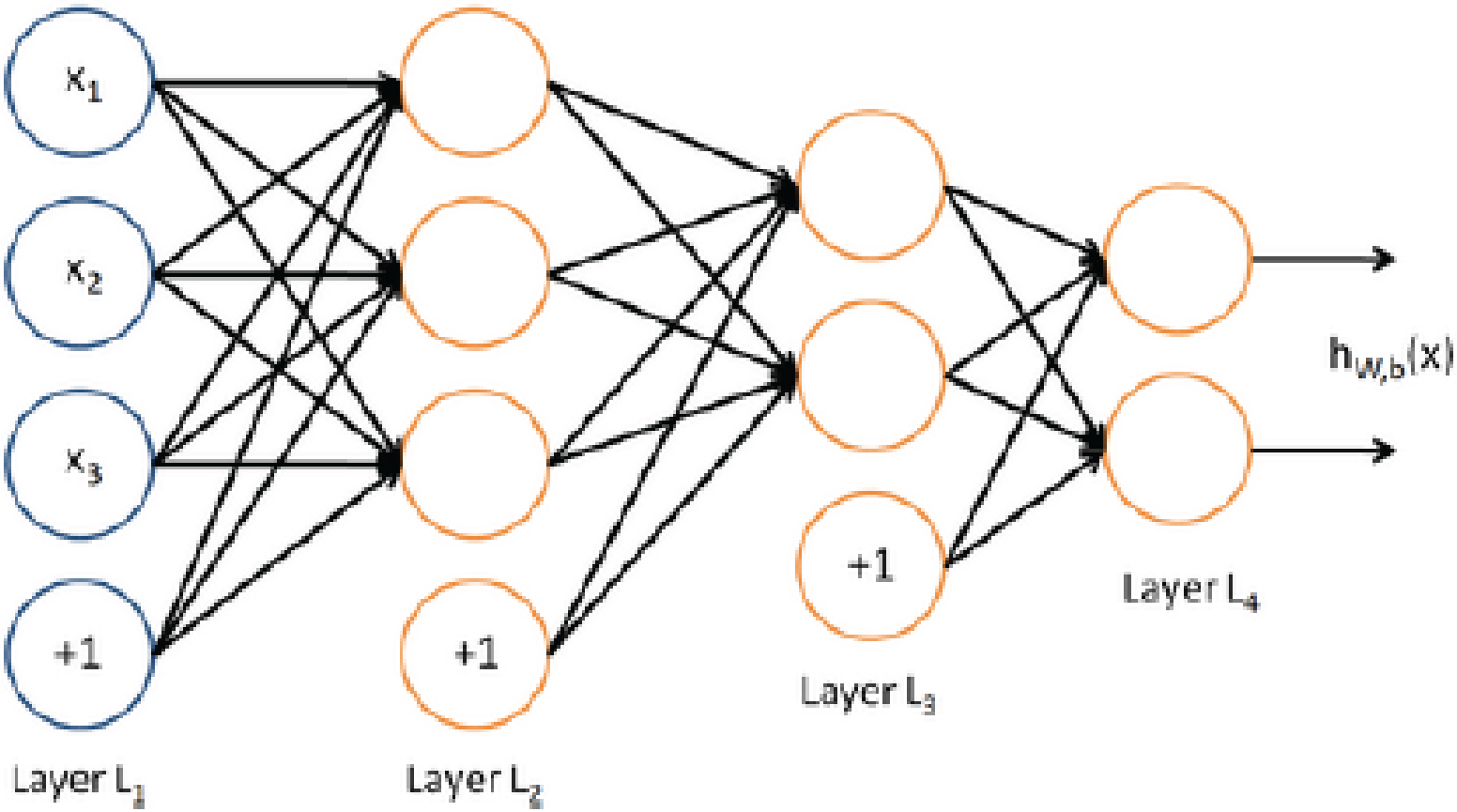
Loss functions

Loss (equation #)	Formula	Margin
energy loss	$E(W, Y^i, X^i)$	none
perceptron	$E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i)$	0
hinge	$\max\left(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)$	m
log	$\log\left(1 + e^{E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)}\right)$	> 0
LVQ2	$\min\left(M, \max(0, E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)$	0
MCE	$\left(1 + e^{-\left(E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)}\right)^{-1}$	> 0
square-square	$E(W, Y^i, X^i)^2 - \left(\max(0, m - E(W, \bar{Y}^i, X^i))\right)^2$	m
square-exp	$E(W, Y^i, X^i)^2 + \beta e^{-E(W, \bar{Y}^i, X^i)}$	> 0
NLL/MMI	$E(W, Y^i, X^i) + \frac{1}{\beta} \log \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)}$	> 0
MEE	$1 - e^{-\beta E(W, Y^i, X^i)} / \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)}$	> 0

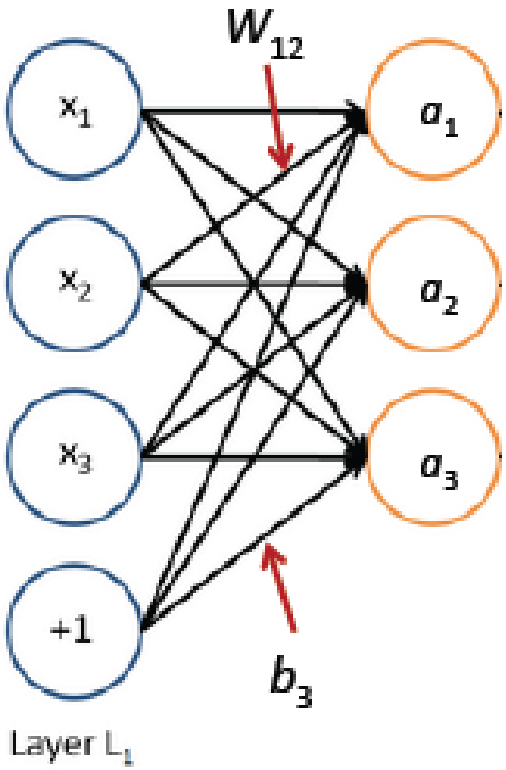
Deep Learning

- Deep Learning Results in NLP tasks
  - Good results in Language Modeling & Machine Translation
    - Mikolov's word2vec
  - Good results in shallow (linear) NLP tasks (pos tagging, NER, Co-reference, SRL, …)
    - C&W, SENNA
  - State of the Art results in structure building tasks (not better)
    - Parsing
    - Relational learning
    - Semantic parsing
  - Good results in a wide range of applications
    - Q&A
    - Paraphrase detection
    - KB population (EL, SF)
    - …

Deep Learning



Deep Learning



$Z = Wx+b$
$A=f(z)$

The neural activations ***a*** can be used to compute some function, for instance score in LM.

$score(x) = U^T a \in R$

Deep Learning

- Alternatives
  - RNN
  - LSTM
  - Transfer Learning
    - Yosinski et al 2014
    - Learn a NN N_B from an existing data set D_B, then transfer into N_T

Applications

- QA
  - Bordes 2014's approach is based on converting questions to (uninterpretable) embeddings which require no pre-defined grammars or lexicons and can query any KB independent of its schema.
  - He focuses on answering simple factual questions on a broad range of topics, more specifically, those for which single KB triples stand for both the question and an answer.
    - automatically generating questions from KB triples and treating this as training data
    - Supplementing this with a data set of question collaboratively marked as paraphrases but with no associated answers.

Applications

Patterns for generating questions from ReVerb triples

KB Triple	Question Pattern
(?, r, e)	*who r e ?*
(?, r, e)	*what r e ?*
(e, r, ?)	*who does e r ?*
(e, r, ?)	*what does e r ?*
(?, r, e)	*what is the r of e ?*
(?, r, e)	*who is the r of e ?*
(e, r, ?)	*what is r by e ?*
(?, r, e)	*who is e's r ?*

KB Triple	Question Pattern
(?, r, e)	*what is e's r ?*
(e, r, ?)	*who is r by e ?*
(e, r-in, ?)	*when did e r ?*
(e, r-on, ?)	*when did e r ?*
(e, r-in, ?)	*when was e r ?*
(e, r-on, ?)	*when was e r ?*
(e, r-in, ?)	*where was e r ?*
(e, r-in, ?)	*where did e r ?*

Applications

- QA
  - Embedding Reverb
  - The model ends up learning vector embeddings of symbols, either for entities or relationships from ReVerb, or for each word of the vocabulary. The embeddings are used for scoring the similarities of a question q and a triple t, i.e. learning the function S(q,t).
  - It consists of projecting questions, treated as a bag of words (and possibly n-grams as well), on the one hand, and triples on the other hand, into a shared embedding space and then computing a similarity measure (the dot product in this paper) between both projections.

Applications

- QA
  - Scoring function: $S(q, t) = f(q)^T g(t)$
    - $f(\cdot)$ a function mapping words from questions into R^k, $f(q) = V^T \Phi(q)$.
    - V is the matrix of $R^{nv \times k}$ containing all word embeddings v that will be learned,.
    - $\Phi(q)$ is the (sparse) binary representation of q ($\in \{0, 1\}^{nv}$) indicating absence or presence of words.
    - Similarly, $g(\cdot)$ is a function mapping entities and relationships from KB triples into R^k, $g(t) = W^T \Psi(t)$.

Applications

- QA
  - Scoring function: $S(q, t) = f(q)^T g(t)$
    - W is the matrix of $R^{ne \times k}$ containing all entity and relationship embeddings w, that will also be learned.
    - $\Psi(t)$ is the (sparse) binary representation of t ($\in \{0, 1\}^{ne}$) indicating absence or presence of entities and relationships.
    - An entity does not have the same embedding when appearing in the left-hand or in the right-hand side of a triple.

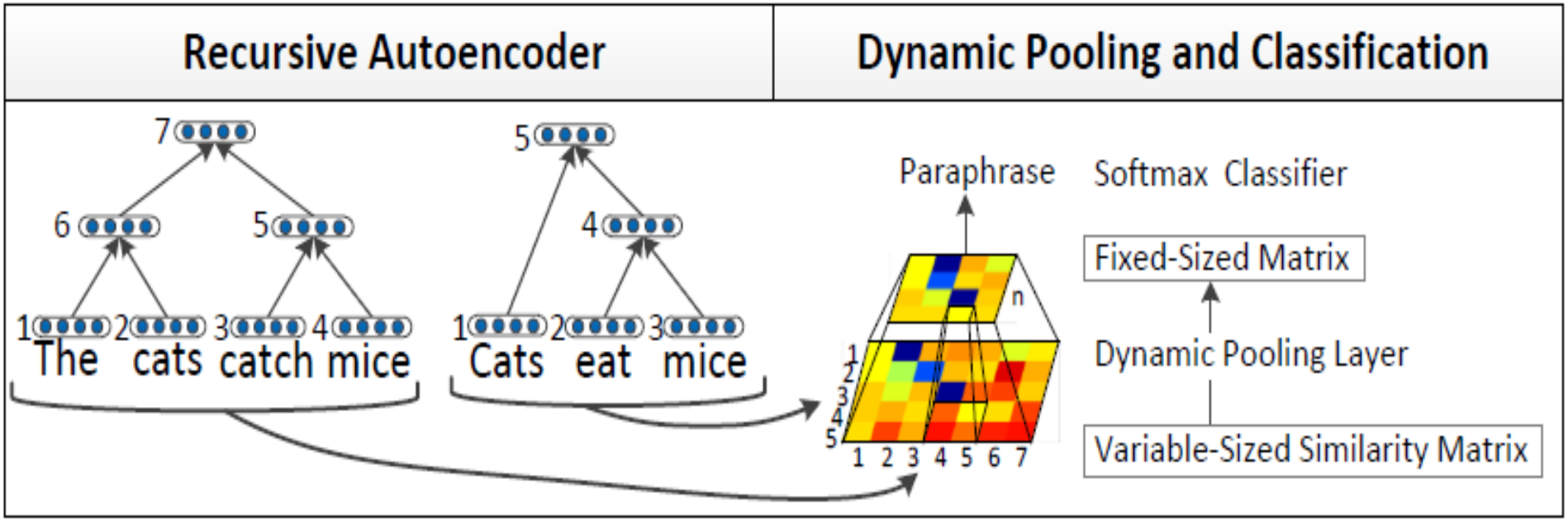
$$\hat{t}(q) = \arg\max_{t' \in \mathcal{K}} S(q, t') = \arg\max_{t' \in \mathcal{K}} \left(\boldsymbol{f}(q)^\top \boldsymbol{g}(t')\right).$$

Applications

- RAE for paraphrase detection
  - From Socher et al, 2011
  - RAE learns feature representations for each node in the tree such that the word vectors underneath each node can be recursively reconstructed.
  - These feature representations are used to compute a similarity matrix that compares both the single words as well as all nonterminal node features in both sentences.
  - In order to keep as much of the resulting global information of this comparison as possible and deal with the arbitrary length of the two sentences, a new dynamic pooling layer which outputs a fixed-size representation. Any classifier such as a softmax classifier can then be used to classify whether the two sentences are paraphrases or not.

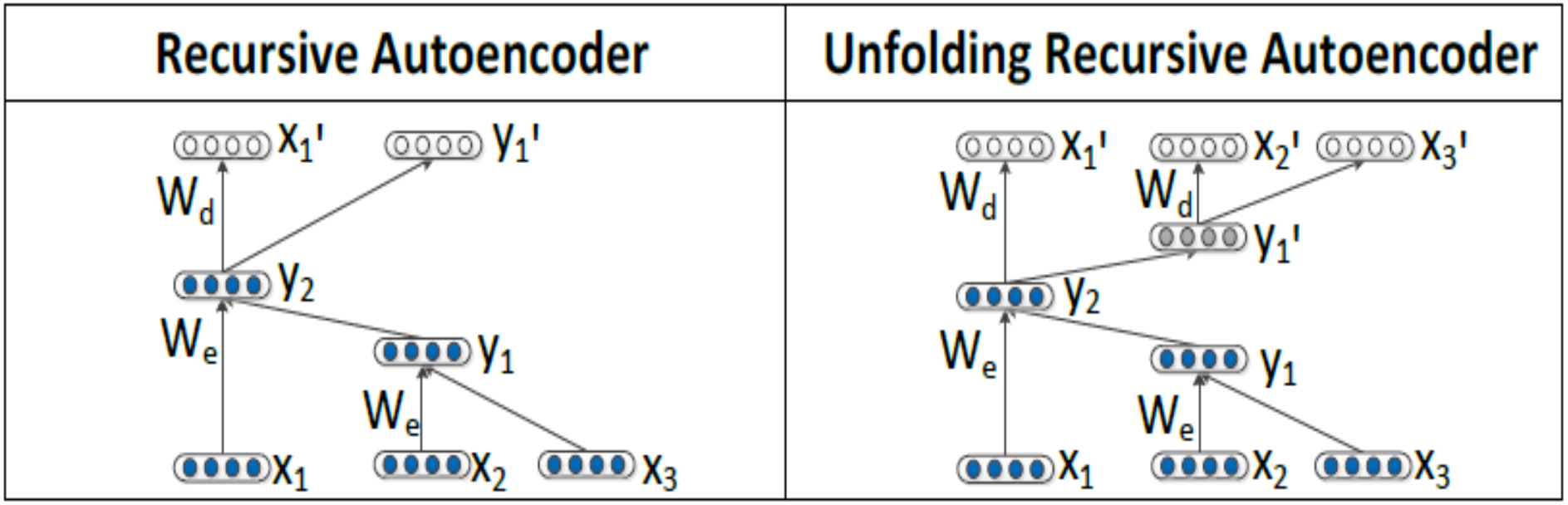
Applications

Paraphrase detection with RAE



Applications

Paraphrase detection with RAE



Applications

Paraphrase detection with RAE

Representing a sentence as an ordered list of these vectors (x_1, …, x_m)
This word representation is better suited for RAEs than the binary number
representations used in previous related models.
A tree is given for each sentence by a parser.

Applications

Paraphrase detection with RAE

The binary parse tree for this input is in the form of branching triplets of parents with children: $(p \rightarrow c_1 c_2)$.
Each child can be either an input word vector x_i or a nonterminal node in the tree.
For both examples in last slide, we have the following triplets:
$((y_1 \rightarrow x_2 x_3), (y_2 \rightarrow x_1 y_1)), \forall x, \ y \in R^n$.

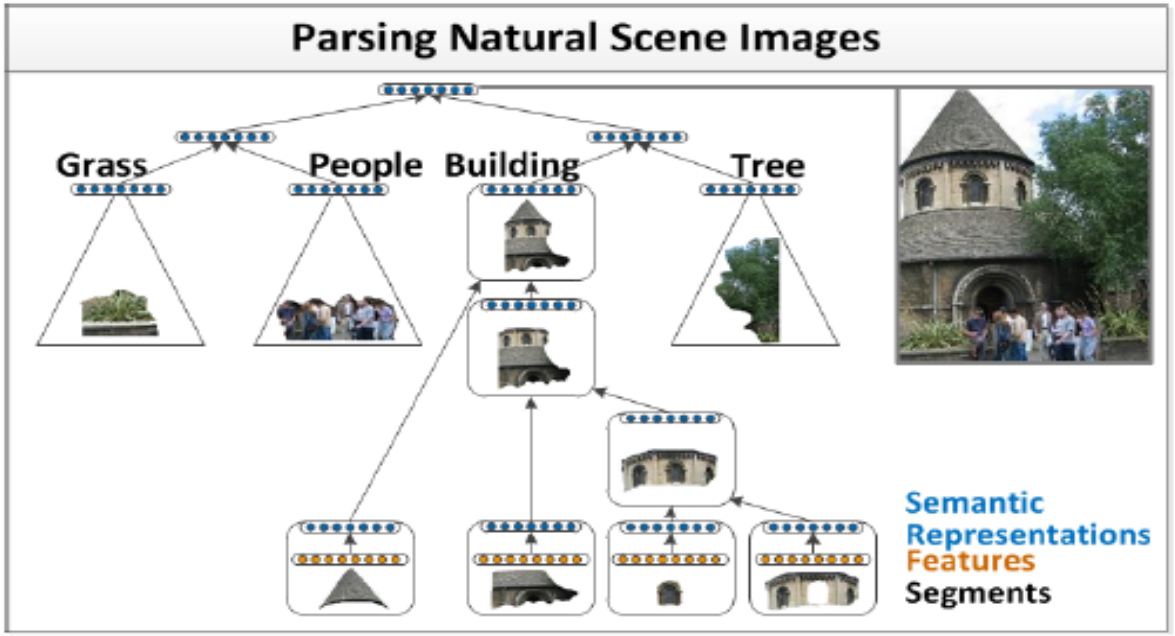
Applications

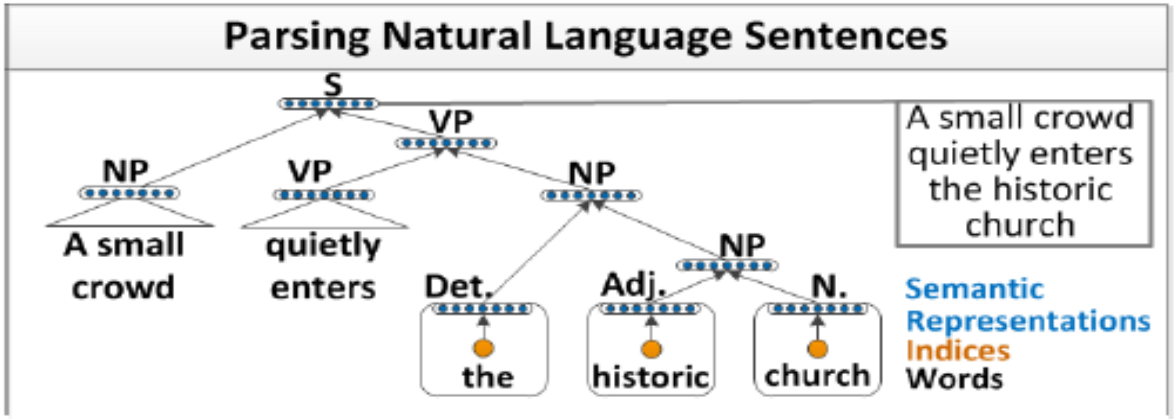
Paraphrase detection with RAE

Compute the parent representations.
$p = y_1$ is computed from the children $(c_1, c_2) = (x_2, x_3)$ by one standard neural network layer: $p = f(W_e[c_1; c_2] + b)$,
where $[c_1; c_2]$ is simply the concatenation of the two children,
f an element-wise activation function and $W_e \in R^{nX2n}$ (the encoding matrix).
how well this n-dimensional vector represents its direct children?
decode their vectors in a reconstruction layer and then compute the Euclidean distance between the original input and its reconstruction.

Applications

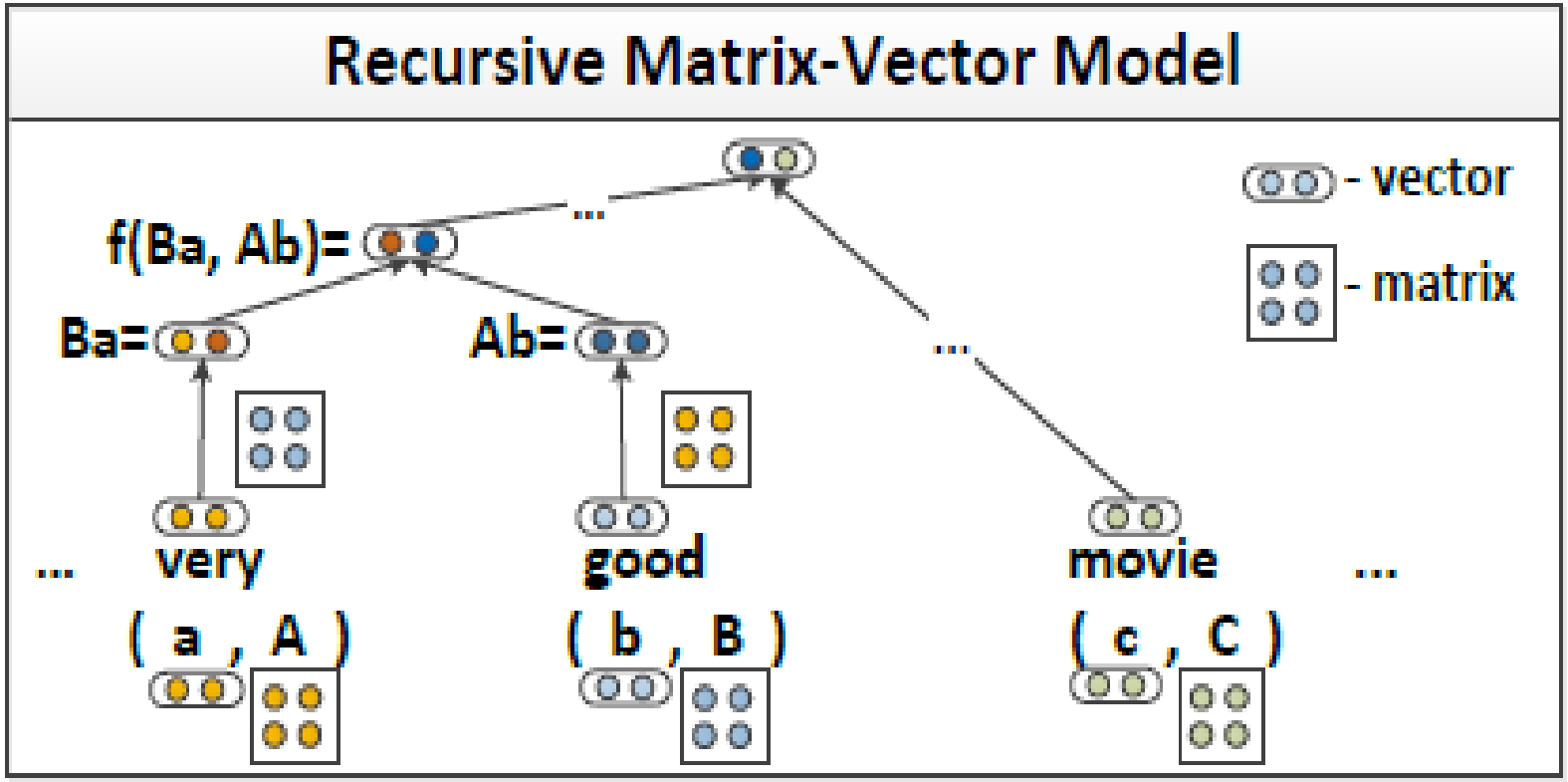
Parsing using Matrix Vector RNN,
Socher et al, 2011





Applications

Parsing using Matrix Vector RNN,
Socher et al, 2011



Conclusions

- Embeddings
  - Good for words, LM, MT
    - Billion of words for learning models
    - Probably better than LSI; LDA, …
    - Combining unsupervised learning with task-dependent supervised layers
  - Not so good for composition of words into more complex units
    - Convolution and pooling seem to be rather naïve approaches for dealing with word order and relevance
    - Socher's approaches seem to go in the good direction
      - Including additional information beyond words: pos, parse, synsets, …
  - Nice to embed KB
    - Freebase, dbpedia, BioPortal, …
    - Other rdf (why not owl) modeled KB

Conclusions

- Deep Learning
  - Good Results in many NLP tasks
  - Good learning capabilities
    - Big models
    - Efficient use of computer resources, GPU, …
  - Difficult to interpret
    - Magic, miracle ???
    - Can we get conclusions from a successful model ??
  - Greedy learning of layers is ok??
  - How many layers ??
  - How many neurons in each layer ??
  - How about not NN-based models (deep graphical models, …) ??