An Introduction to Deep Learning

Marc'Aurelio Ranzato
Facebook AI Research
ranzato@fb.com

Outline

- **PART 0** [lecture 1]
    - Motivation
    - Training Fully Connected Nets with Backpropagation
- **Part 1** [lecture 1 and lecture 2]
    - Deep Learning for Vision: CNN
- **Part 2** [lecture 2]
    - **Deep Learning for NLP: embeddings**
- **Part 3** [lecture 3]
    - Modeling sequences

Representing Symbolic Data

- Lots of data is symbolic. For instance:
  - Text
  - Graphs
- Can DL be useful to represent such data?
  - If we could represent symbolic data in a continuous space, we could easily measure relatedness.
  - We could apply the powerful tools of linear algebra and DL to perform complex reasoning.

Representing Symbolic Data

- Challenges:
  - Discrete nature, easy to count but not obvious how to represent.
  - One cannot use standard backprop through discrete units.
  - The number of entities to represent can be very large, albeit finite; e.g., words in English dictionary.
  - Often times this data is not associated to a regular grid structure like an image. E.g.: text, social graph.

Case Study:
Learning Word Representations

- As a case study, we will consider the problem of learning word representations from raw text (without any supervision).

- We will explore a few approaches to learn such representations.

- Practical applications:
    - Text classification
    - Ranking (e.g., google search, Facebook feeds ranking)
    - Machine translation
    - Chatbot

Latent Semantic Analysis

- Problem: Find similar documents in a corpus.
- Solution:
  - construct the “term”/“document” matrix storing (normalized) occurrence counts
  - SVD

Deerwester et al. “Indxing by Latent Semantic Analysis” JASIS 1990

Latent Semantic Analysis

$$
\begin{array}{c}
\boldsymbol{X} \\
(\mathbf{d}_j) \\
\downarrow \\
(\mathbf{t}_i^T) \rightarrow
\end{array}
\begin{bmatrix}
x_{1,1} & \cdots & x_{1,n} \\
\vdots & \ddots & \vdots \\
x_{m,1} & \cdots & x_{m,n}
\end{bmatrix}
$$

term-document matrix

$x_{i,j}$ (normalized) number of times word i appears in document j

Deerwester et al. “Indxing by Latent Semantic Analysis” JASIS 1990

Latent Semantic Analysis

$$
\begin{array}{c}
X \\
(\mathbf{d}_j) \\
\downarrow \\
(\mathbf{t}_i^T) \rightarrow
\end{array}
\begin{bmatrix}
x_{1,1} & \cdots & x_{1,n} \\
\vdots & \ddots & \vdots \\
x_{m,1} & \cdots & x_{m,n}
\end{bmatrix}
$$

term-document matrix

Example

doc1: the cat is furry

doc2: dogs are furry

	doc1	doc2
are	0	1
cat	1	0
dogs	0	1
furry	1	1
is	1	0
the	1	0

$x_{i,j}$ (normalized) number of times word i appears in document j

Deerwester et al. “Indxing by Latent Semantic Analysis” JASIS 1990

Latent Semantic Analysis

$$
\begin{array}{ccccc}
X & & U & \Sigma & V^T \\
(\mathbf{d}_j) & & & & (\hat{\mathbf{d}}_j) \\
\downarrow & & & & \downarrow \\
(\mathbf{t}_i^T) \rightarrow & \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} & = & (\hat{\mathbf{t}}_i^T) \rightarrow \begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} \dots \begin{bmatrix} \\ \mathbf{u}_l \\ \\ \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} \cdot \begin{bmatrix} [& \mathbf{v}_1 &] \\ & \vdots & \\ [& \mathbf{v}_l &] \end{bmatrix}
\end{array}
$$

term-document matrix

$x_{i,j}$ (normalized) number of times word i appears in document j

Deerwester et al. “Indxing by Latent Semantic Analysis” JASIS 1990

Latent Semantic Analysis

$$
\begin{array}{ccccc}
\boldsymbol{X} & & U & & \Sigma & & V^T \\
(\mathbf{d}_j) & & & & & & (\hat{\mathbf{d}}_j) \\
\downarrow & & & & & & \downarrow \\
(\mathbf{t}_i^T) \rightarrow & \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} & = & (\hat{\mathbf{t}}_i^T) \rightarrow & \begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} \dots \begin{bmatrix} \\ \mathbf{u}_l \\ \\ \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} \cdot \begin{bmatrix} [& & \mathbf{v}_1 & &] \\ & & \vdots & & \\ [& & \mathbf{v}_l & &] \end{bmatrix}
\end{array}
$$

term-document matrix

$x_{i,j}$ (normalized) number of times word i appears in document j

Each column of V^T, is a representation of a document in the corpus.

Deerwester et al. "Indxing by Latent Semantic Analysis" JASIS 1990

Latent Semantic Analysis

$$
\begin{array}{ccccc}
\boldsymbol{X} & & U & & \Sigma & & V^T \\
(\mathbf{d}_j) & & & & & & (\hat{\mathbf{d}}_j) \\
\downarrow & & & & & & \downarrow \\
(\mathbf{t}_i^T) \rightarrow & \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix} & = & (\hat{\mathbf{t}}_i^T) \rightarrow & \begin{bmatrix} \begin{bmatrix} & & \\ \mathbf{u}_1 & & \end{bmatrix} & \cdots & \begin{bmatrix} & & \\ \mathbf{u}_l & & \end{bmatrix} \end{bmatrix} & \cdot & \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_l \end{bmatrix} & \cdot & \begin{bmatrix} [& & \mathbf{v}_1 & &] \\ & & \vdots & & \\ [& & \mathbf{v}_l & &] \end{bmatrix}
\end{array}
$$

term-document matrix

$x_{i,j}$ (normalized) number of times word i appears in document j

Each column of V^T, is a representation of a document in the corpus.

Each column is a D dimensional vector. We can use it to compare & retrieve documents.

Deerwester et al. "Indxing by Latent Semantic Analysis" JASIS 1990

Latent Semantic Analysis

$$
\begin{array}{ccccc}
\boldsymbol{X} & & U & \Sigma & V^T \\
(\mathbf{d}_j) & & & & (\hat{\mathbf{d}}_j) \\
\downarrow & & & & \downarrow \\
(\mathbf{t}_i^T) \rightarrow & \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} & = & (\hat{\mathbf{t}}_i^T) \rightarrow \begin{bmatrix} & & \\ \mathbf{u}_1 & \dots & \mathbf{u}_l \\ & & \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} \cdot \begin{bmatrix} & \mathbf{v}_1 & \\ & \vdots & \\ & \mathbf{v}_l & \end{bmatrix}
\end{array}
$$

term-document matrix

$x_{i,j}$ (normalized) number of times word i appears in document j

Each row of U, is a representation of a word in the dictionary.

Deerwester et al. “Indxing by Latent Semantic Analysis” JASIS 1990

Latent Semantic Analysis

$$
\begin{array}{ccccc}
\boldsymbol{X} & & U & \Sigma & V^T \\
(\mathbf{d}_j) & & & & (\hat{\mathbf{d}}_j) \\
\downarrow & & & & \downarrow \\
(\mathbf{t}_i^T) \rightarrow & \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} & = & (\hat{\mathbf{t}}_i^T) \rightarrow \begin{bmatrix} \begin{bmatrix} \\ \\ \end{bmatrix} & \begin{bmatrix} \\ \\ \end{bmatrix} \\ \mathbf{u}_1 & \mathbf{u}_l \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} \cdot \begin{bmatrix} [& \mathbf{v}_1 &] \\ & \vdots & \\ [& \mathbf{v}_l &] \end{bmatrix}
\end{array}
$$

term-document matrix

$x_{i,j}$ (normalized) number of times word i appears in document j

Each row of U, is a representation of a word in the dictionary.
Each row of U, is a vectorial representation of a word, a.k.a. *embedding*.

Deerwester et al. "Indxing by Latent Semantic Analysis" JASIS 1990

Word Embeddings

- Convert words (symbols) into a D dimensional vector, where D is a hyper-parameter.

- Once embedded, we can:
  - Compare words.
  - Apply our favorite machine learning method (DL) to represent sequences of words.
  - At document retrieval time in LSA, the representation of a new document is a weighted sum of word embeddings (bag-of-words -> bag-of-embeddings): U’ x

bi-gram

- A bi-gram is a model of the probability of a word given the preceding one:

$$p(w_k|w_{k-1}) \qquad w_k \in V$$

- The simplest approach consists of building a (normalized) matrix of counts:

$$c(w_k|w_{k-1}) = \begin{matrix} \\ \text{current word} \end{matrix} \overset{\text{preceding word}}{\begin{bmatrix} c_{1,1} & \dots & c_{1,|V|} \\ \dots & c_{i,j} & \dots \\ c_{|V|,1} & \dots & c_{|V|,|V|} \end{bmatrix}}$$

$c_{i,j}$ number of times word i is preceded by word j

Factorized bi-gram

- We can factorize (via SVD, for instance) the bigram to reduce the number of parameters and become more robust to noise (entries with low counts):

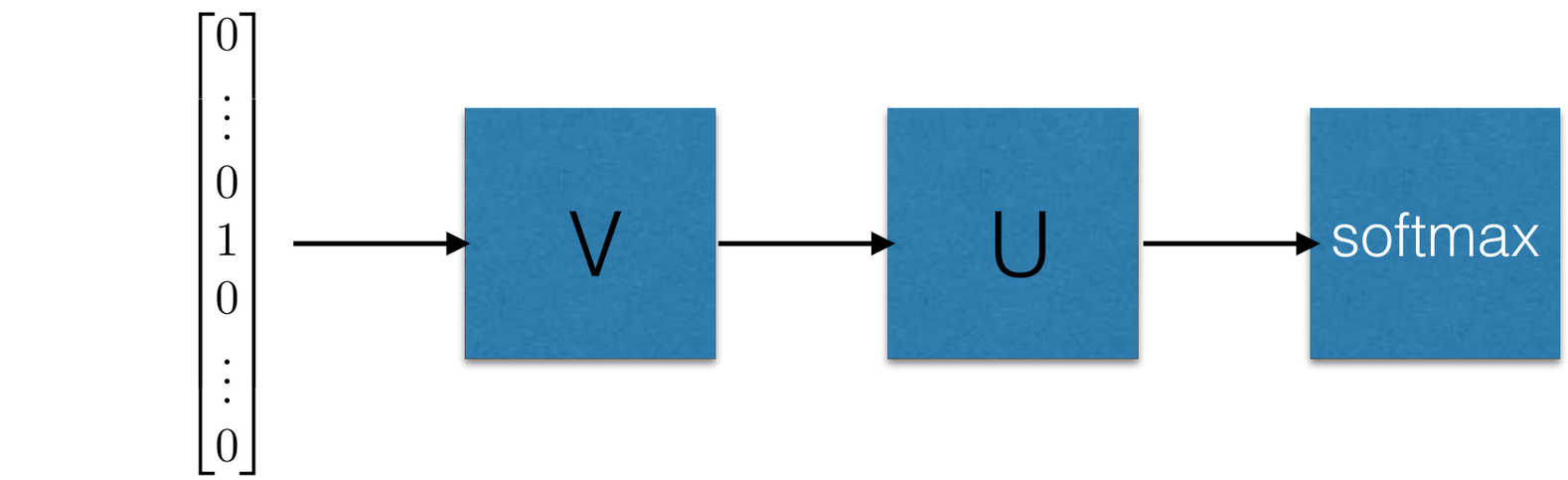
$$c(w_k|w_{k-1}) = \begin{matrix}\text{output word}\end{matrix}\overset{\text{input word}}{\begin{bmatrix} c_{1,1} & \dots & c_{1,|V|} \\ \dots & c_{i,j} & \dots \\ c_{|V|,1} & \dots & c_{|V|,|V|} \end{bmatrix}} = UV \quad \begin{matrix} V \in \mathbb{R}^{D\times|V|} \\ U \in \mathbb{R}^{|V|\times D} \end{matrix}$$

- Rows of U store “output” word embeddings, and columns of V store “input” word embeddings.

Factorized bi-gram

- The same can be expressed as a two layer (linear) neural network:

$$c(w_k|w_{k-1}) = \begin{matrix} \\ \text{output word} \end{matrix} \overbrace{\begin{bmatrix} c_{1,1} & \dots & c_{1,|V|} \\ \dots & c_{i,j} & \dots \\ c_{|V|,1} & \dots & c_{|V|,|V|} \end{bmatrix}}^{\text{input word}} = UV$$

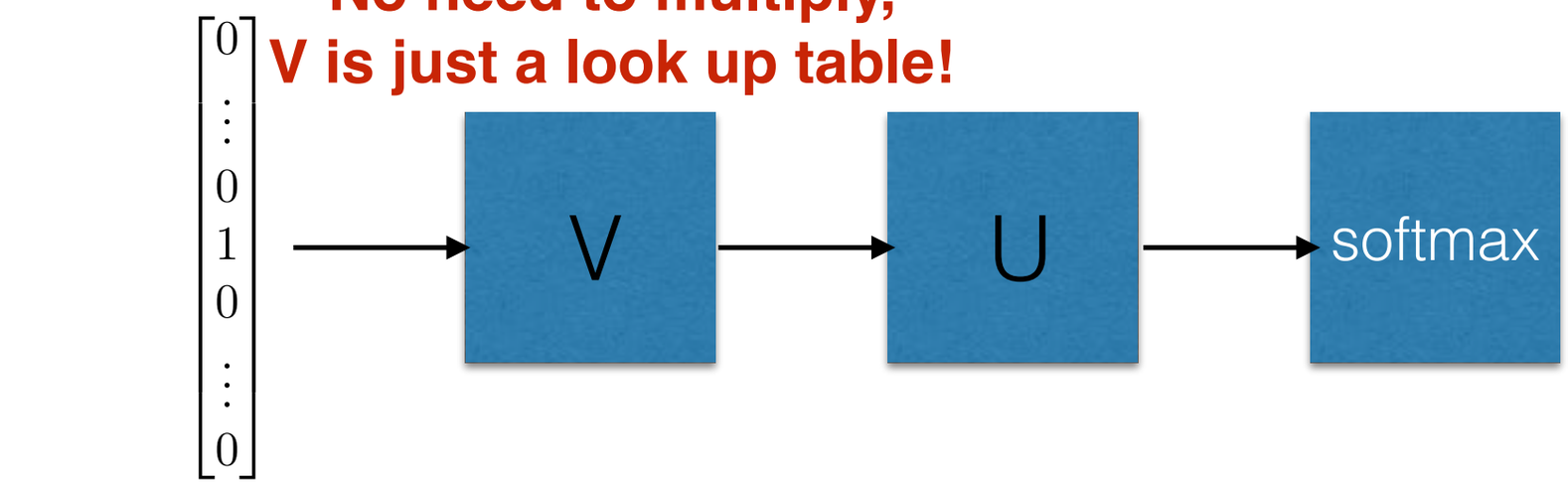


Factorized bi-gram

- The same can be expressed as a two layer (linear) neural network:

$$c(w_k|w_{k-1}) = \begin{matrix} \\ \text{output word} \end{matrix} \overset{\text{input word}}{\begin{bmatrix} c_{1,1} & \cdots & c_{1,|V|} \\ \cdots & c_{i,j} & \cdots \\ c_{|V|,1} & \cdots & c_{|V|,|V|} \end{bmatrix}} = UV$$

No need to multiply,
V is just a look up table!



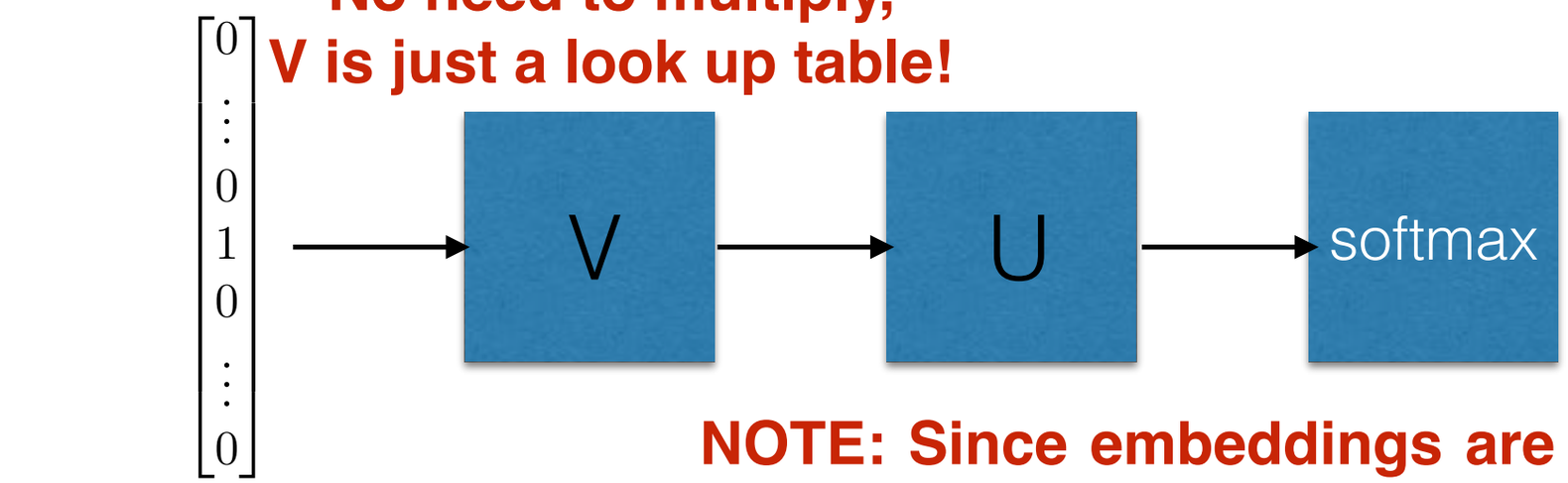
1-hot representation
of the input word

Factorized bi-gram

- The same can be expressed as a two layer (linear) neural network:

$$c(w_k|w_{k-1}) = \begin{matrix} \\ \text{output word} \end{matrix} \overbrace{\begin{bmatrix} c_{1,1} & \dots & c_{1,|V|} \\ \dots & c_{i,j} & \dots \\ c_{|V|,1} & \dots & c_{|V|,|V|} \end{bmatrix}}^{\text{input word}} = UV$$

No need to multiply,
V is just a look up table!



1-hot representation of the input word

NOTE: Since embeddings are free, there is no point adding non-linearities and more layers! Here, depth does not help!

Factorized bi-gram

- bi-gram model could be useful for type-ahead applications (in practice, it's much better to condition upon the past n>2 words).

- Factorized model yields word embeddings as a by-product.

Word Embeddings

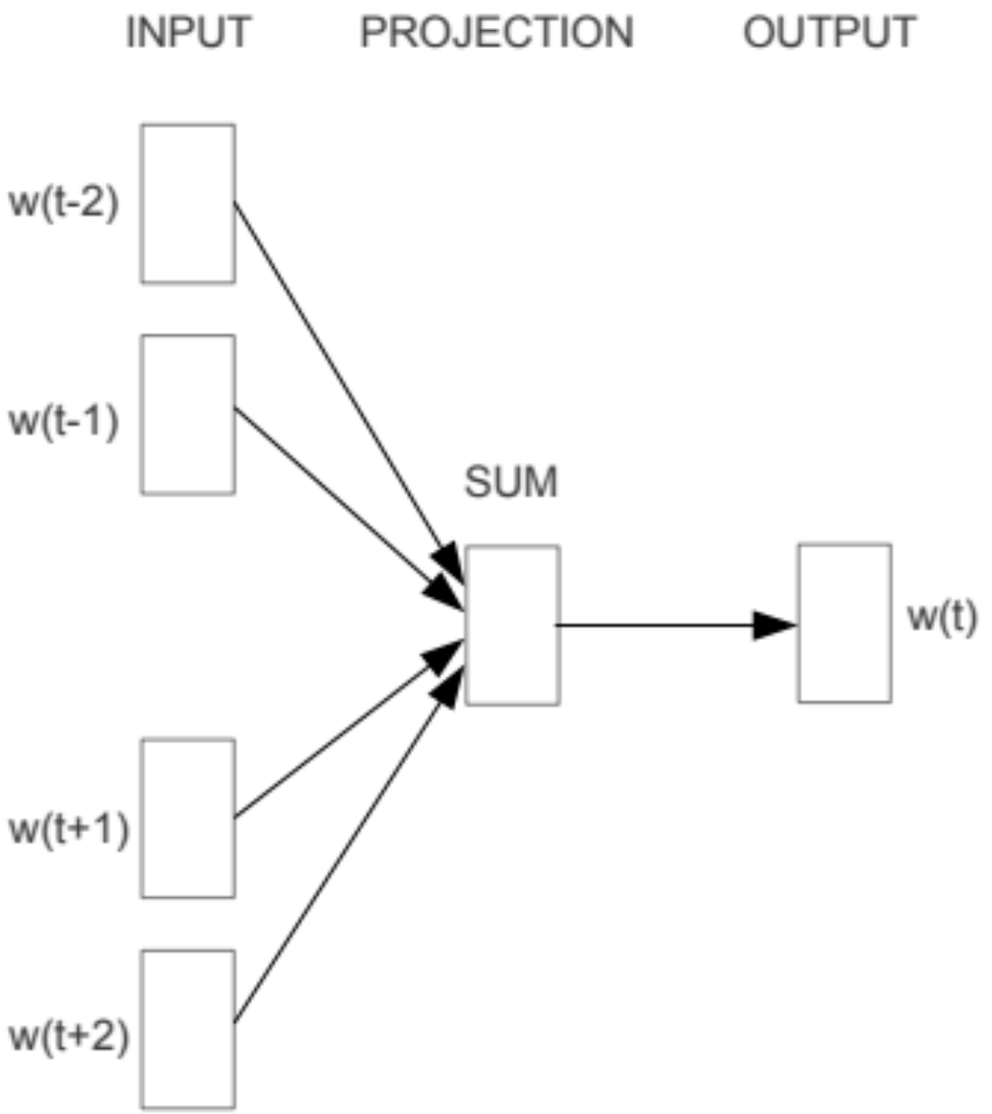
- LSA learns word embeddings that take into account co-occurrences across documents.

- bi-gram instead learns word embeddings that only take into account the next word.

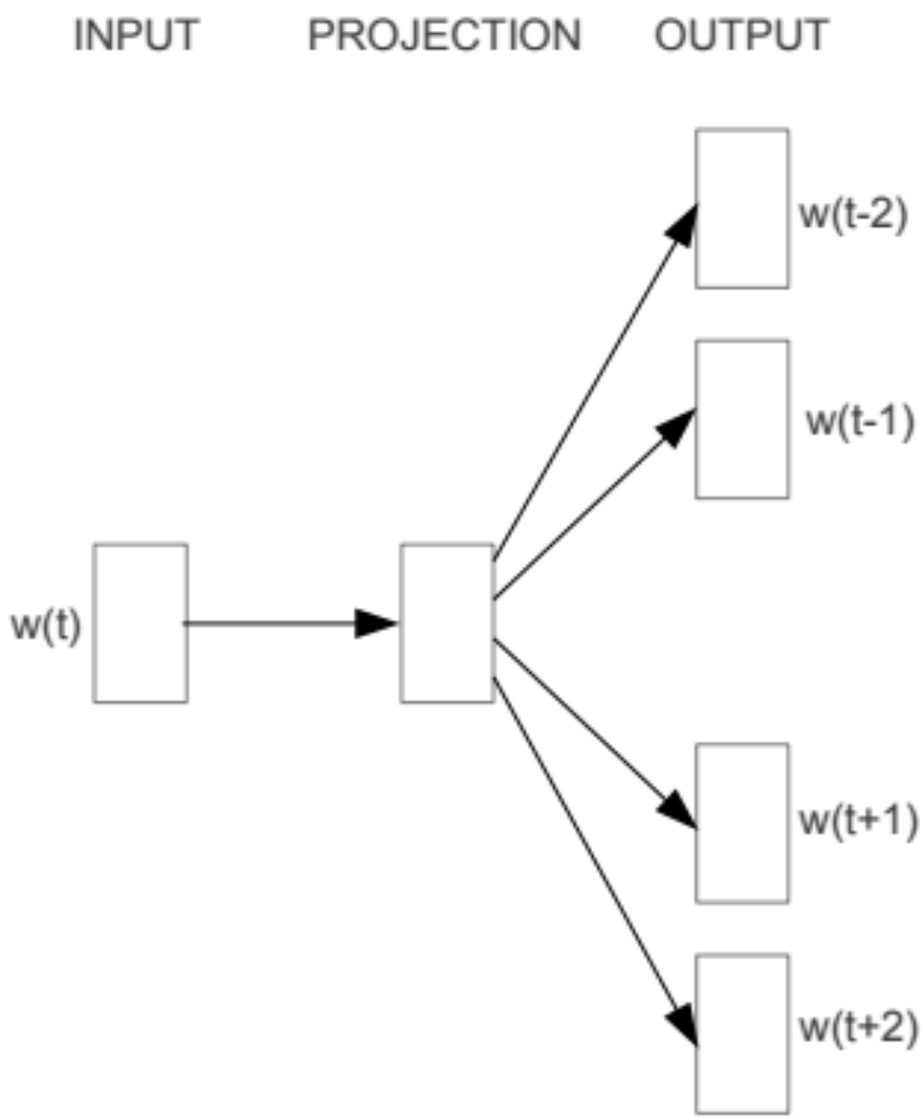
- It seems better to do something in between, using more context but just around the word of interest, yielding a method called **word2vec**.

Mikolov et al. “Efficient estimation of word representations” rejected by ICLR 2013

word2vec



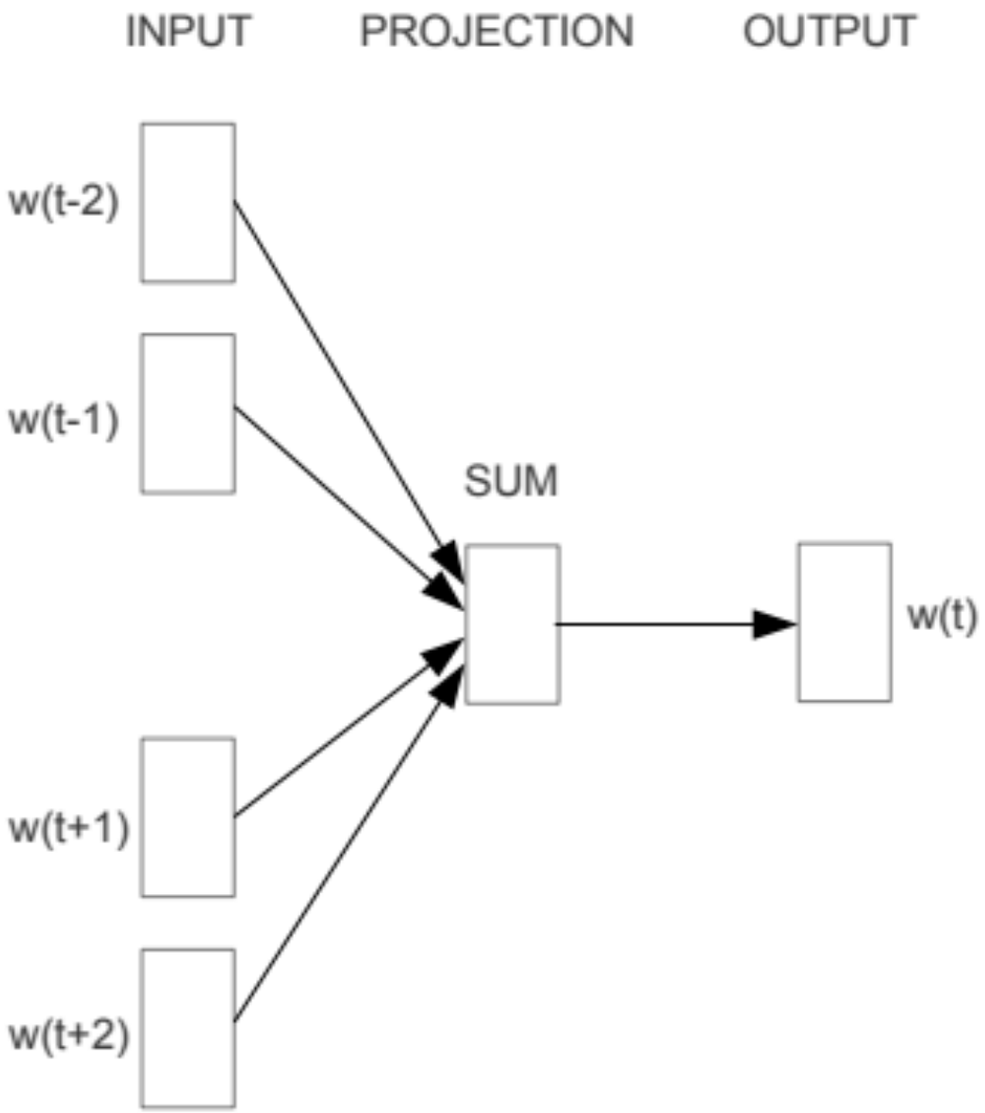
CBOW



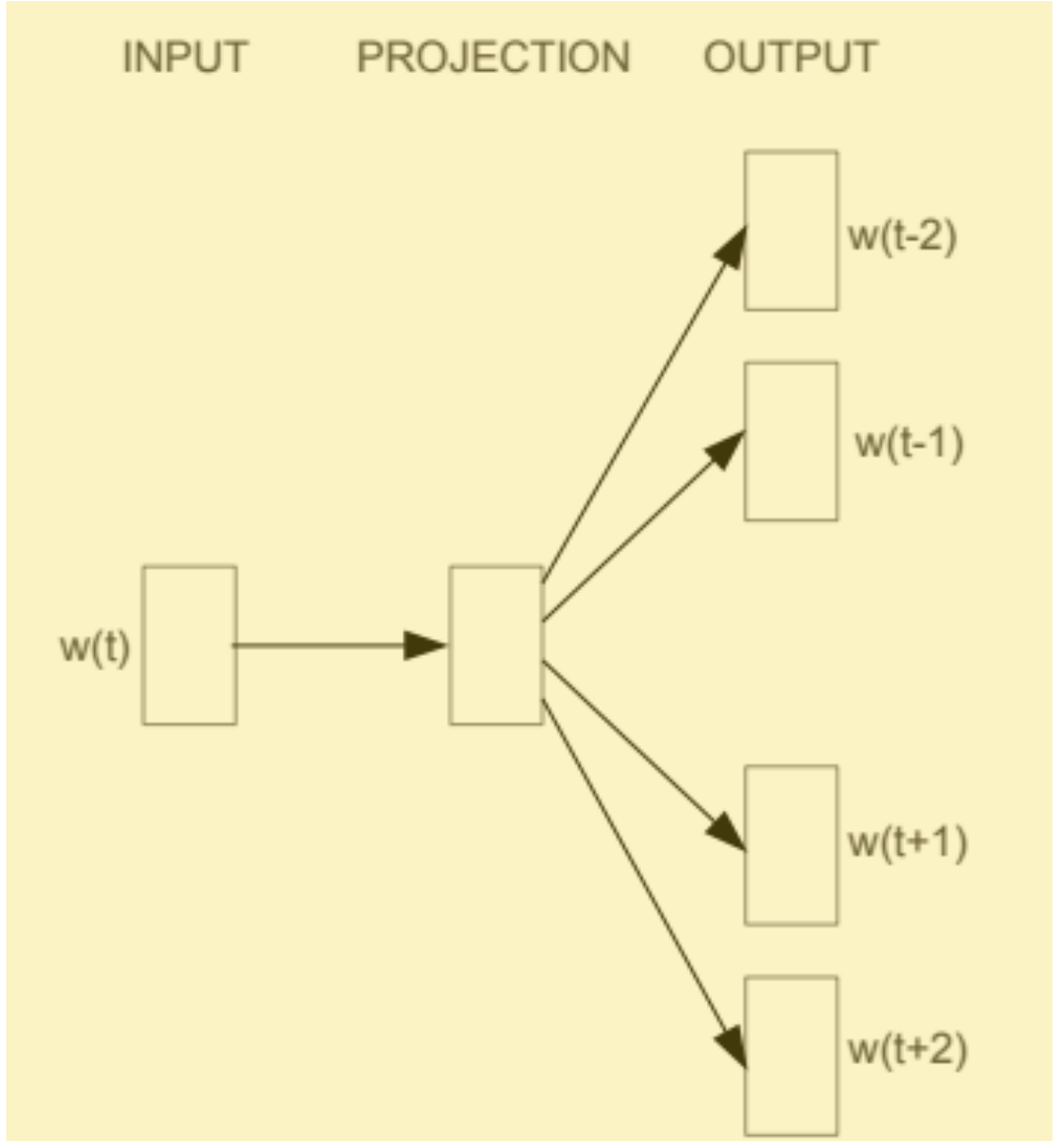
Skip-gram

Mikolov et al. “Efficient estimation of word representations” rejected by ICLR 2013

word2vec



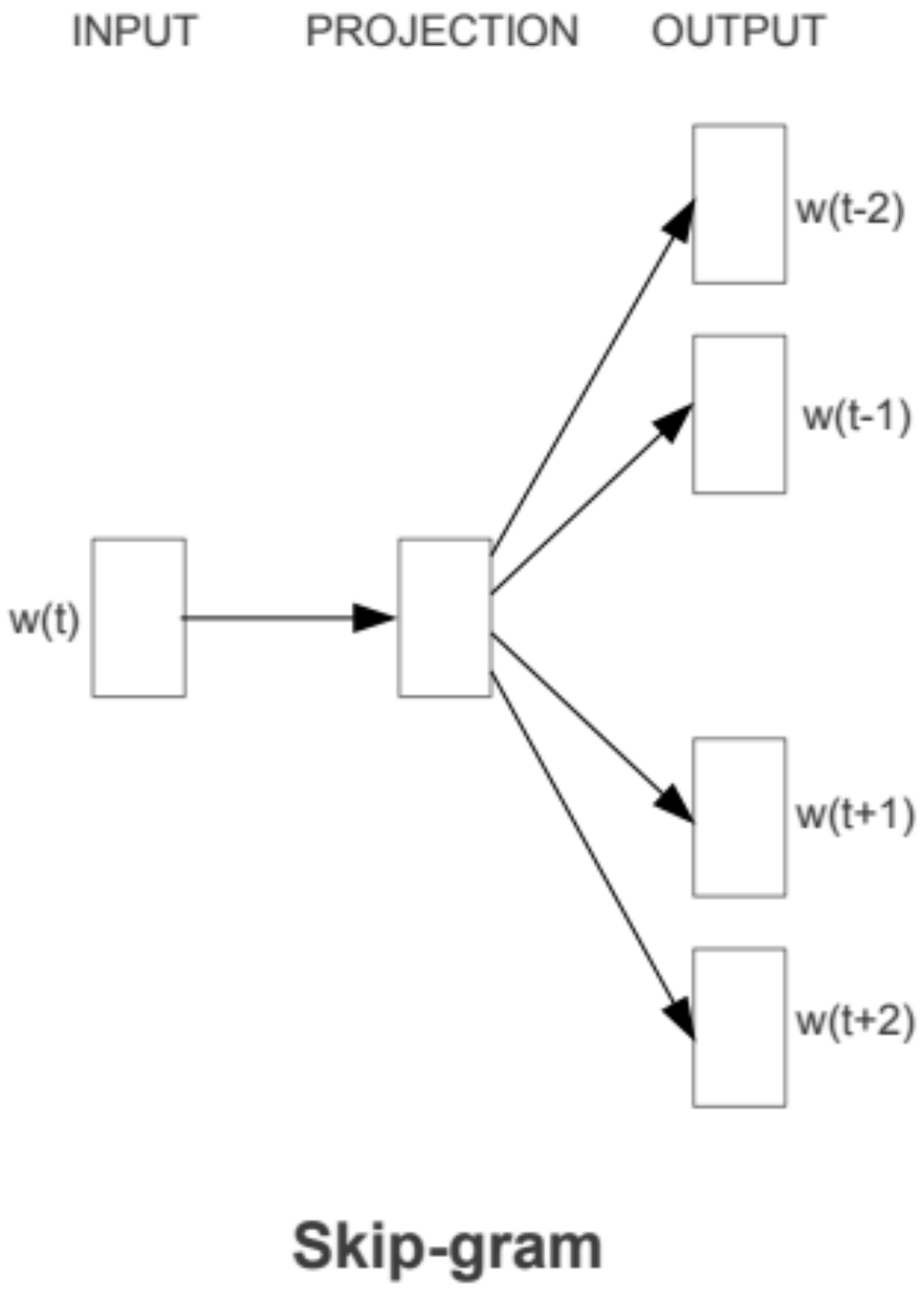
CBOW



Skip-gram

Mikolov et al. “Efficient estimation of word representations” rejected by ICLR 2013

skip-gram



Skip-gram

- Similar to factorized bi-gram model, but predict N preceding and N following words.
- Words that have the same context will get similar embeddings. E.g.: cat & kitty.
- Input projection is just look-up table. Bulk of computation is the the prediction of words in context.
- Learning by cross-entropy minimization via SGD.

Mikolov et al. “Efficient estimation of word representations” rejected by ICLR 2013

Hierarchical Softmax

- When there are lots of classes to predict (e.g., words in a dictionary, |V| in the order of 100,000 or more), projection in the output space is computationally very expensive.

- Hierarchical softmax speeds up computation at the cost of a little decrease of accuracy:

$$p(w_k|h) = \sum_{n=1}^{N} p(w_k|h, c_n)p(c_n|h)$$

Drop sum: each word belongs to 1 and only 1 cluster

$$= p(w_k|h, c_n)p(c_n|h)$$

n-th cluster

feature (word embedding in skip-gram)

Hierarchical Softmax

$$p(w_k|h) = \sum_{n=1}^{N} p(w_k|h, c_n)p(c_n|h)$$

$$= p(w_k|h, c_n)p(c_n|h)$$

Why is it cheaper to have two softmaxes instead of one?

Because these are much smaller. If clusters have all the same size and contain $\frac{|V|}{N}$ words:

$$D \times |V| \gg D \times N + D \times \frac{|V|}{N} \approx D \times \frac{|V|}{N}$$

In practice, clusters are formed by taking into account word frequency in order to minimize computation cost. Tree can be have more children (binary tree).

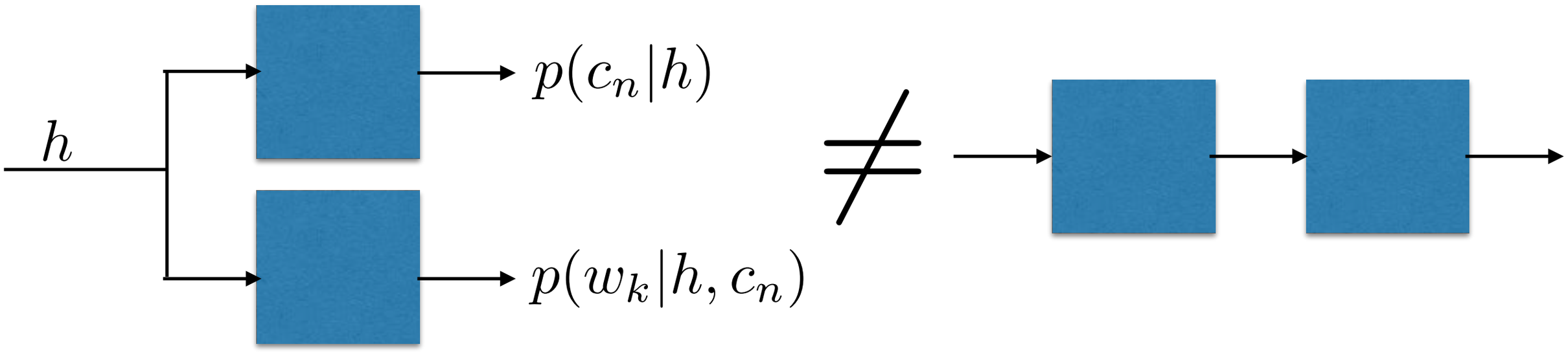
Morin et al. "Hierarchical probabilistic neural network language model" AISTATS 2005
Mikolov et al. "Strategies for training large-scale neural network language models" ASRU 2011

Hierarchical Softmax

$$p(w_k|h) = \sum_{n=1}^{N} p(w_k|h, c_n)p(c_n|h)$$

$$= p(w_k|h, c_n)p(c_n|h)$$

Is hierarchical softmax “deep”? No, as we walk down the tree the representation is not changed.



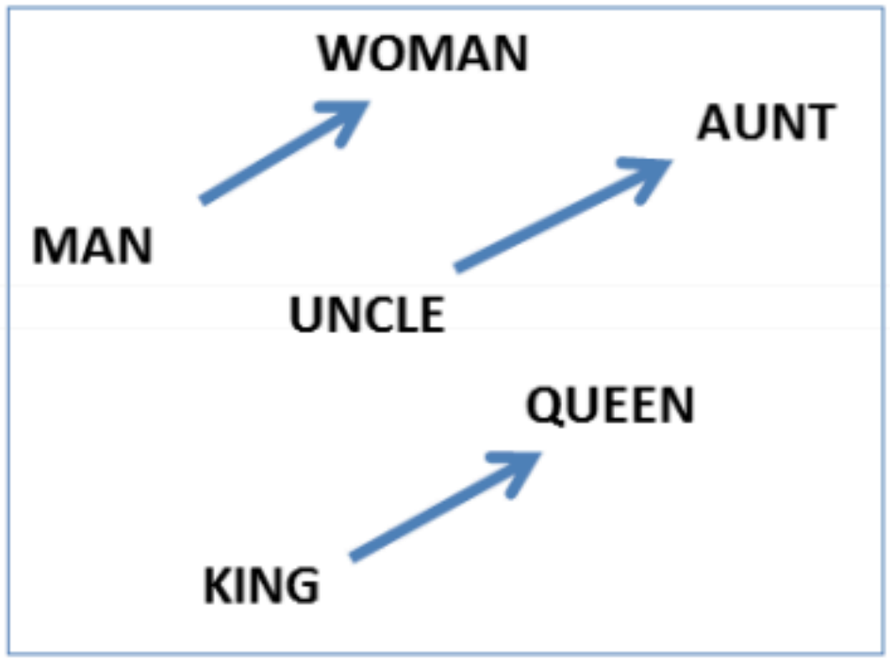
Morin et al. “Hierarchical probabilistic neural network language model” AISTATS 2005
Mikolov et al. “Strategies for training large-scale neural network language models” ASRU 2011

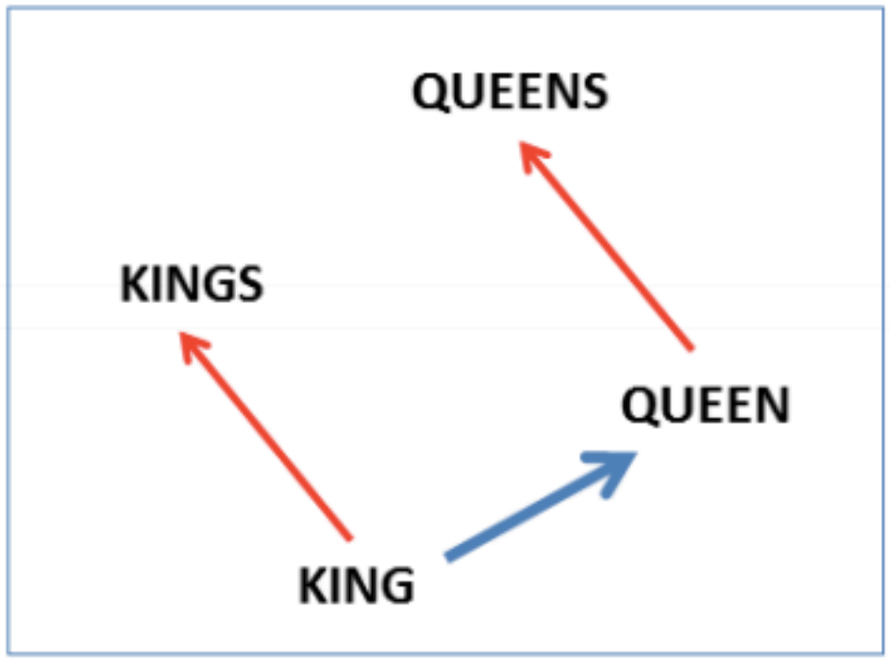
word2vec

- code at: https://code.google.com/archive/p/word2vec/

- next some evaluation from Tomas's NIPS 2013 presentation at: https://drive.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit

Linguistic Regularities in Word Vector Space





- The word vector space implicitly encodes many regularities among words

credit T. Mikolov from https://drive.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit

Linguistic Regularities in Word Vector Space

- The resulting distributed representations of words contain surprisingly a lot of syntactic and semantic information
- There are multiple degrees of similarity among words:
  - **KING** is similar to **QUEEN** as **MAN** is similar to **WOMAN**
  - **KING** is similar to **KINGS** as **MAN** is similar to **MEN**
- Simple vector operations with the word vectors provide very intuitive results

credit T. Mikolov from https://drive.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit

Linguistic Regularities - Results

- Regularity of the learned word vector space is evaluated using test set with about 20K questions
- The test set contains both syntactic and semantic questions
- We measure TOP1 accuracy (input words are removed during search)
- We compare our models to previously published word vectors

credit T. Mikolov from https://drive.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit

Linguistic Regularities - Results

Model	*Vector Dimensionality*	*Training Words*	*Training Time*	*Accuracy [%]*
Collobert NNLM	50	660M	2 months	11
Turian NNLM	200	37M	few weeks	2
Mnih NNLM	100	37M	7 days	9
Mikolov RNNLM	640	320M	weeks	25
Huang NNLM	50	990M	weeks	13
Our NNLM	100	6B	2.5 days	51
Skip-gram (hier.s.)	1000	6B	hours	66
CBOW (negative)	300	1.5B	**minutes**	**72**

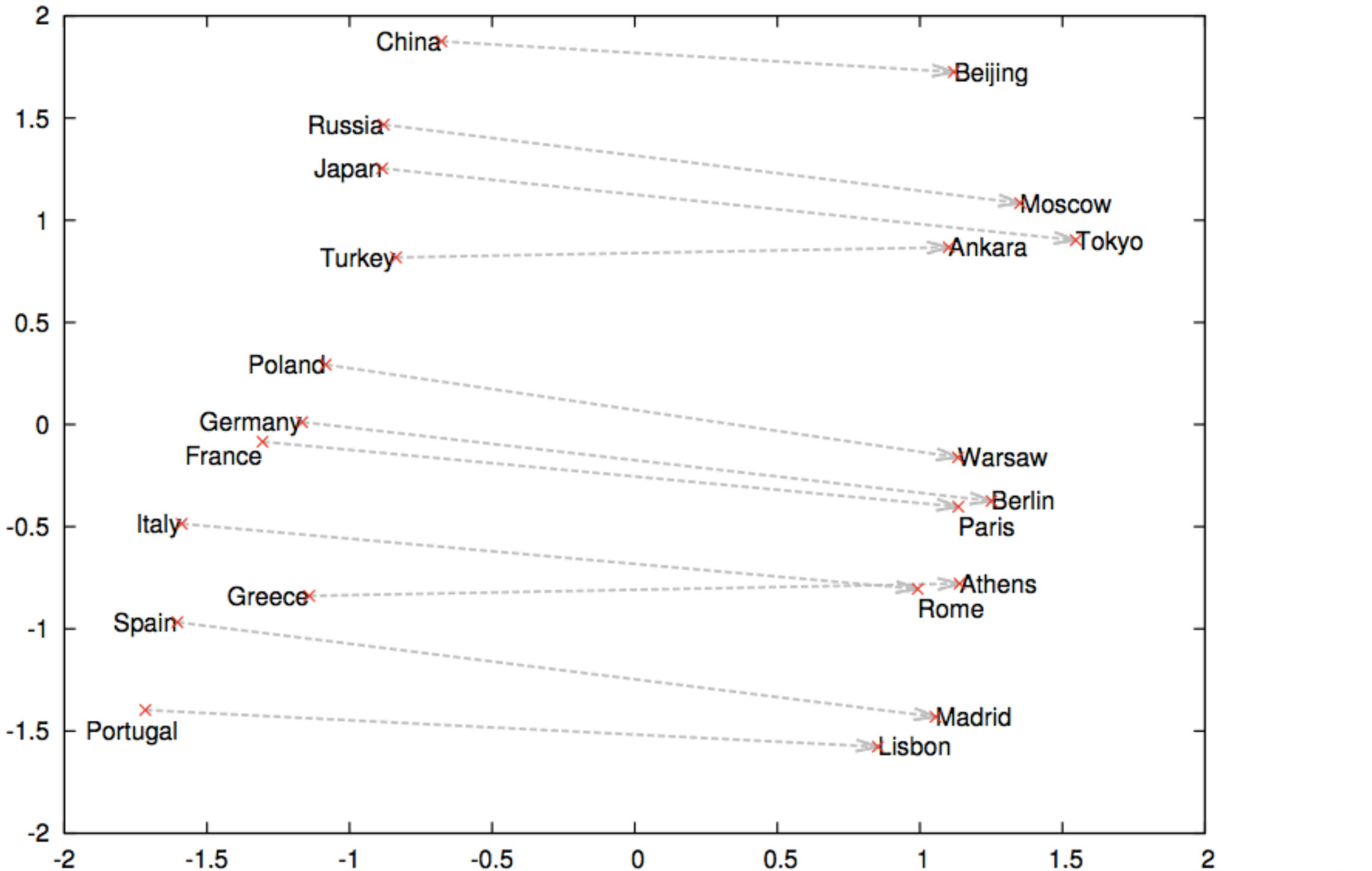
credit T. Mikolov from https://drive.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit

Linguistic Regularities in Word Vector Space

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

credit T. Mikolov from https://drive.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit

Visualization of Regularities in Word Vector Space



credit T. Mikolov from https://drive.google.com/file/d/0B7XkCwpI5KDYRWRnd1RzWXQ2TWc/edit

word2vec demo

Recap

- Embedding words (from a 1-hot to a distributed representation) lets you:
    - understand similarity between words
        - plug them within any parametric ML model
- Several ways to learn word embeddings. word2vec is still one of the most efficient ones.
- Note word2vec leverages large amounts of *unlabeled* data.

Representing Phrases

- How about representing short sequences of words?
- Could we simply average (pool) word embeddings?

word embedding of w_{k+i}

$$\bar{e} = e(w_k, w_{k+1}, \ldots, w_{k+n-1}) = \frac{1}{n} \sum_{i=0}^{n-1} e(w_{k+i})$$

- This is a surprisingly good baseline! E.g.: recommender systems.

credit to: A. Szlam https://learning.mpi-sws.org/mlss2016/slides/Arthur_Szlam_MLSS-2016.pdf

Bag-of-embeddings

- Well-known but counter-intuitive fact about $\mathbb{R}^d$:

[*concentration measure*] with high probability, the inner product of any two random vectors is 0 (therefore their distance is approx.$\sqrt{d}$).

If word embeddings were drawn i.i.d., what's the value of d s.t. we can recover w_{k+i} by finding the nearest neighbor to $\bar{e}$?

$$d > n \log(\frac{n|V|}{\epsilon})$$

number of words in the bag

probability of recovery failure

credit to: A. Szlam https://learning.mpi-sws.org/mlss2016/slides/Arthur_Szlam_MLSS-2016.pdf

Bag-of-embeddings

- Well-known but counter-intuitive fact about $\mathbb{R}^d$:

[*concentration measure*] with high probability, the inner product of any two random vectors is 0 (therefore their distance is approx.$\sqrt{d}$).

If word embeddings were drawn i.i.d., what's the value of d s.t. we can recover w_{k+i} by finding the nearest neighbor to $\bar{e}$?

$$d > n \log(\frac{n|V|}{\epsilon})$$

number of words in the bag

probability of recovery failure

if IVI=100,000, n=10 and d>100, -> perfect (orderless) recovery from a bag!

credit to: A. Szlam https://learning.mpi-sws.org/mlss2016/slides/Arthur_Szlam_MLSS-2016.pdf

Recap

- Given word embeddings, bagging embeddings is often an effective way to represent short sequences of words.
- Theory of sparse recovery explains why.
- What other (better) ways are there?
- How can DL help here?

Language Modeling

- In language modeling, we want to predict a word given some context.

- bi-gram uses only the preceding word.

- More generally, we can use the last N words. E.g.: n-grams and neural net language model.

- Or even better, we can use some sort of running average of all the words seen thus far, as in recurrent neural networks.

- As a by-product, these methods produce a representation of a sequence of (fixed or variable length) words without any supervision.

Language Modeling

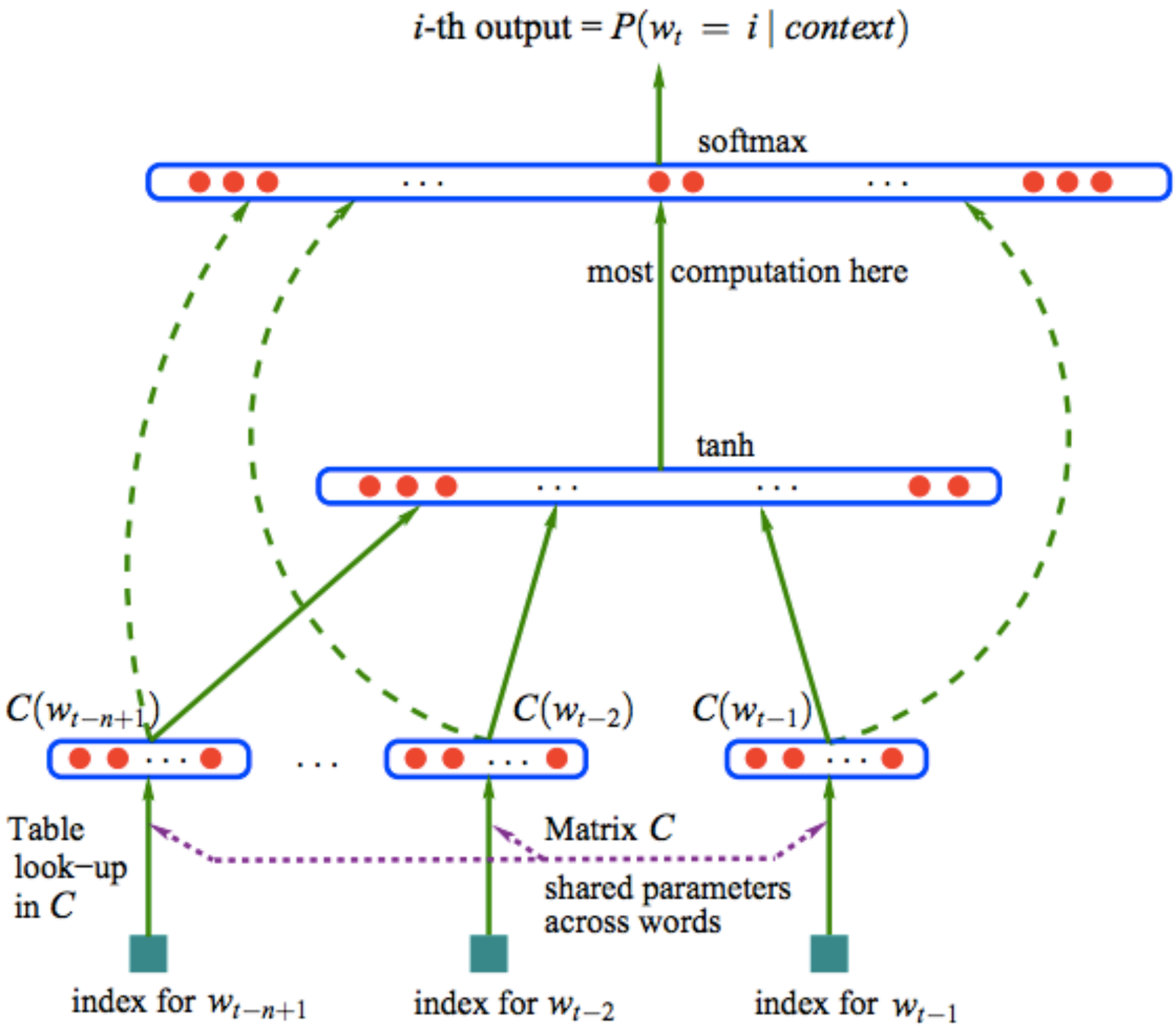
- the math…

$$p_\theta(w_1, w_2, \ldots, w_M) = p_\theta(w_M|w_{M-1}\ldots, w_1)p_\theta(w_{M-1}|w_{M-2}, \ldots, w_1)\ldots p_\theta(w_2|w_1)p_\theta(w_1)$$

- with Markov assumption (used by n-grams):

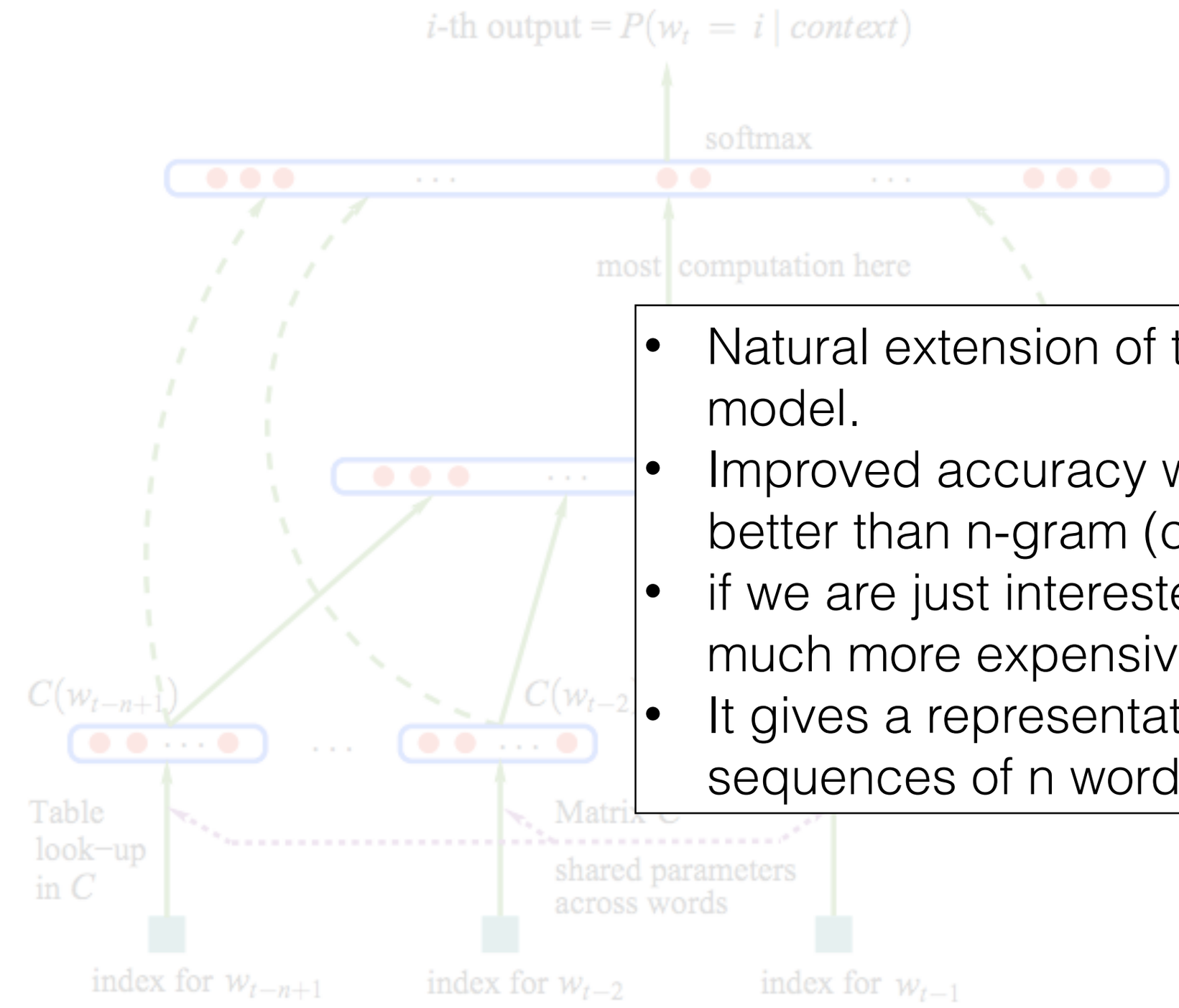
$$p_\theta(w_1, w_2, \ldots, w_M) = p_\theta(w_M|w_{M-1}\ldots, w_{M-n})p_\theta(w_{M-1}|w_{M-2}, \ldots, w_{M-n-1})\ldots p_\theta(w_2|w_1)p_\theta(w_1)$$

Neural Network LM



Y. Bengio et al. “A neural probabilistic language model” JMLR 2003

Neural Network LM



- Natural extension of the factorized bi-gram model.
- Improved accuracy with more context. A bit better than n-gram (count based methods).
- if we are just interested in word embeddings, much more expensive than word2vec.
- It gives a representation to *ordered* sequences of n words.

Y. Bengio et al. "A neural probabilistic language model" JMLR 2003

Recurrent Neural Network

- In NN-LM, the hidden state is the concatenation of word embeddings.

- Key idea of RNNs: compute a (non-linear) running average instead, to increase the size of the context.

- Many variants…

Recurrent Neural Network

- Elman RNN:

$$p(w_{k+1}|h) = \text{softmax}(U^o h_k + b^o)$$

$$h_k = \sigma(U^r h_{k-1} + U^i \mathbf{1}(w_k) + b^r)$$

only difference compared to factorized bi-gram language model

Elman “Finding structure in time” Cognitive Science 1990

Recurrent Neural Network

- Elman RNN:

this could be a hierarchical softmax

$$p(w_{k+1}|h) = \text{softmax}(U^o h_k + b^o)$$

$$h_k = \sigma(U^r h_{k-1} + U^i \mathbf{1}(w_k) + b^r)$$

only difference compared to factorized bi-gram language model

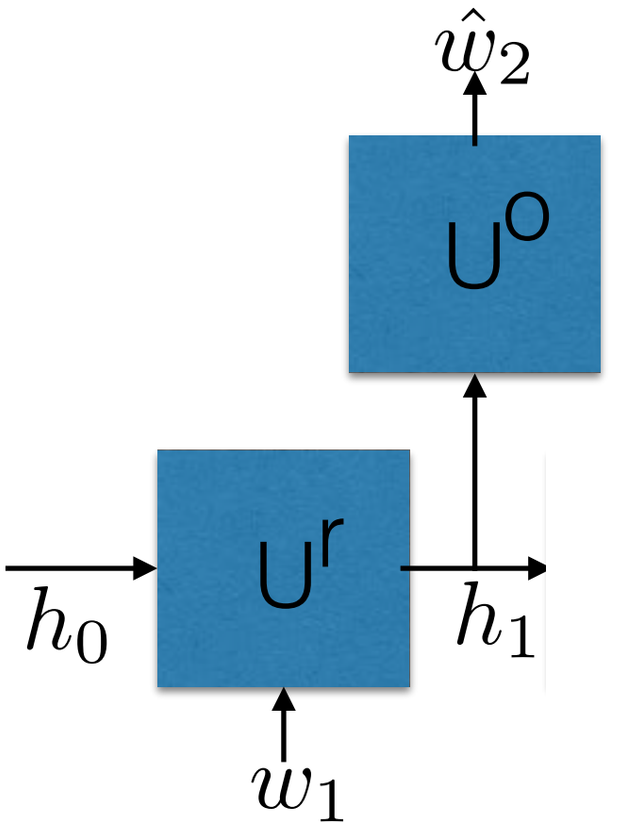
Elman “Finding structure in time” Cognitive Science 1990

RNN: Inference Time

- Elman RNN:

$$p(w_{k+1}|h) = \text{softmax}(U^o h_k + b^o)$$

$$h_k = \sigma(U^r h_{k-1} + U^i \mathbf{1}(w_k) + b^r)$$



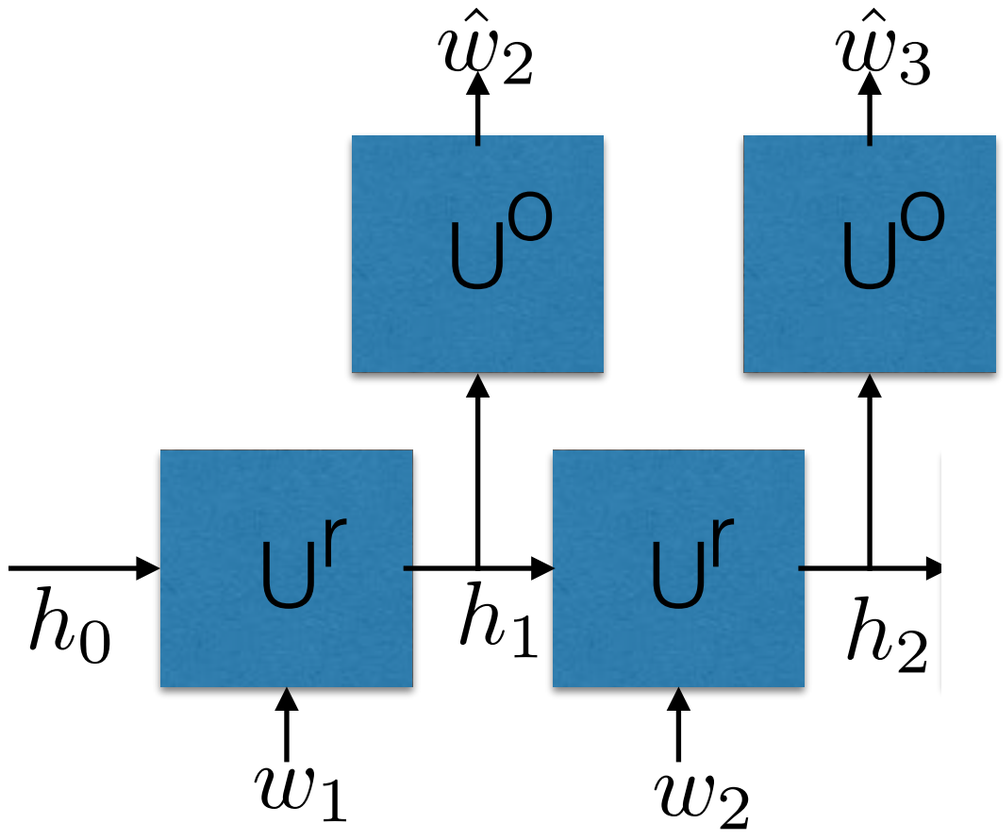
Elman "Finding structure in time" Cognitive Science 1990

RNN: Inference Time

- Elman RNN:

$$p(w_{k+1}|h) = \text{softmax}(U^o h_k + b^o)$$

$$h_k = \sigma(U^r h_{k-1} + U^i \mathbf{1}(w_k) + b^r)$$



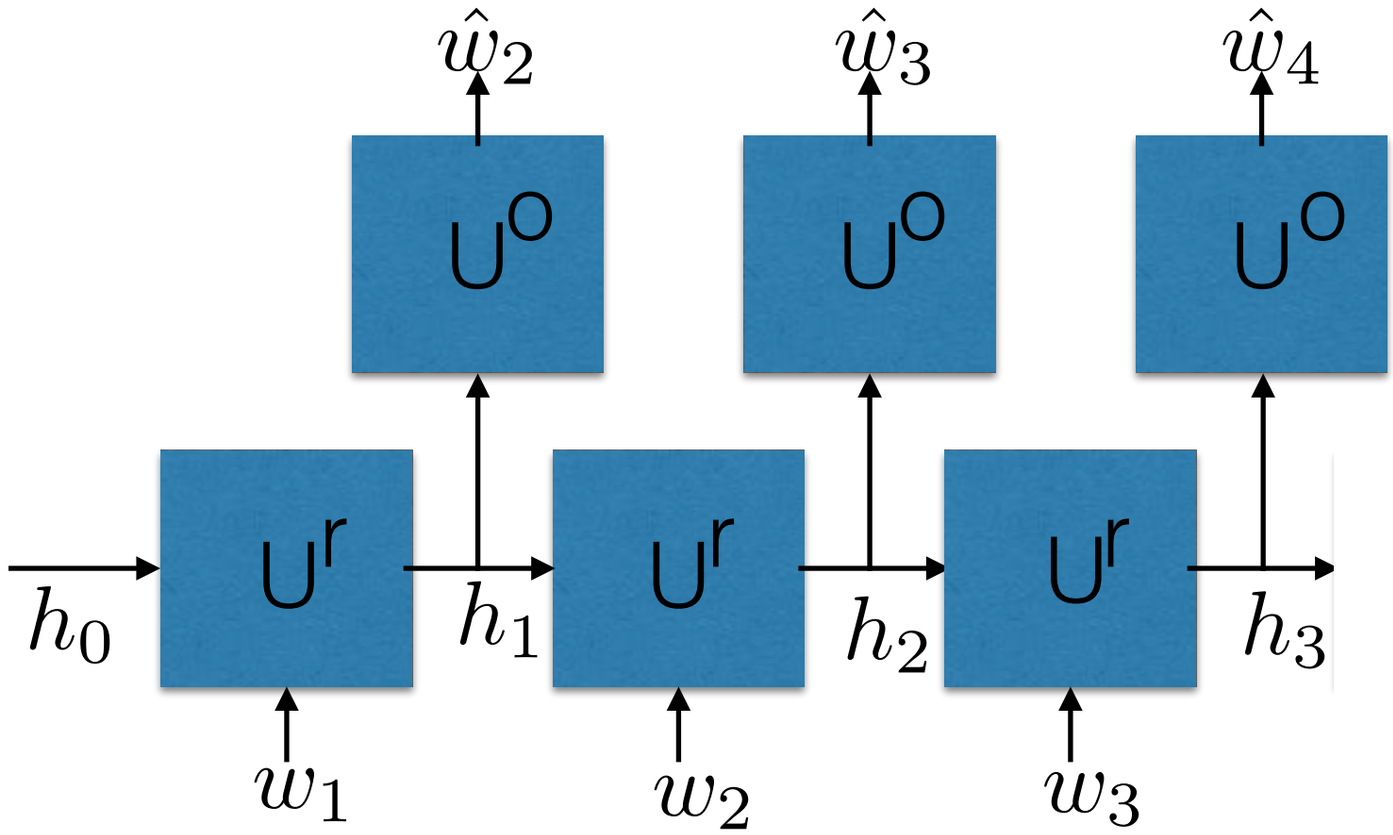
Elman "Finding structure in time" Cognitive Science 1990

RNN: Inference Time

- Elman RNN:

$$p(w_{k+1}|h) = \text{softmax}(U^o h_k + b^o)$$

$$h_k = \sigma(U^r h_{k-1} + U^i \mathbf{1}(w_k) + b^r)$$



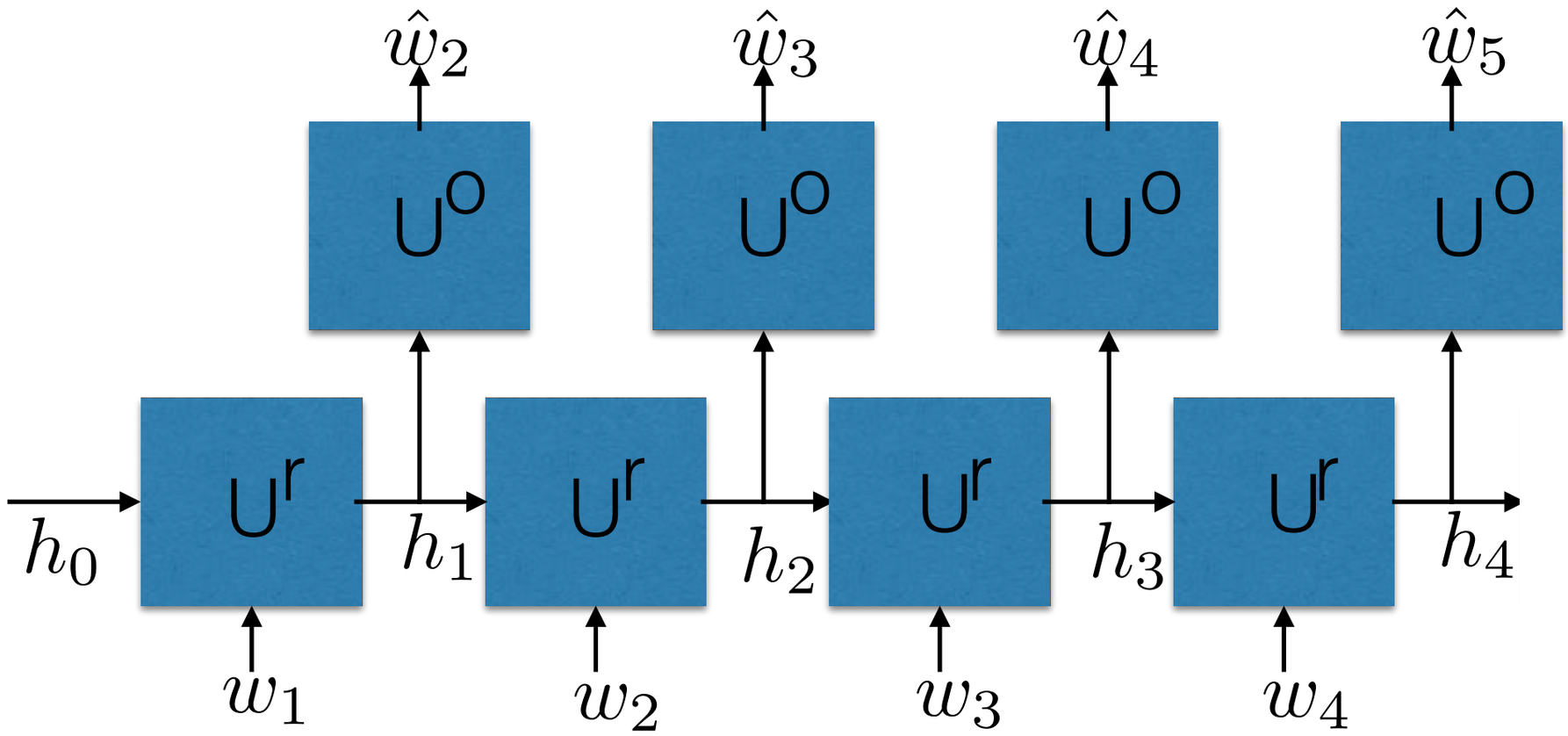
Elman "Finding structure in time" Cognitive Science 1990

RNN: Inference Time

- Elman RNN:

$$p(w_{k+1}|h) = \text{softmax}(U^o h_k + b^o)$$

$$h_k = \sigma(U^r h_{k-1} + U^i \mathbf{1}(w_k) + b^r)$$



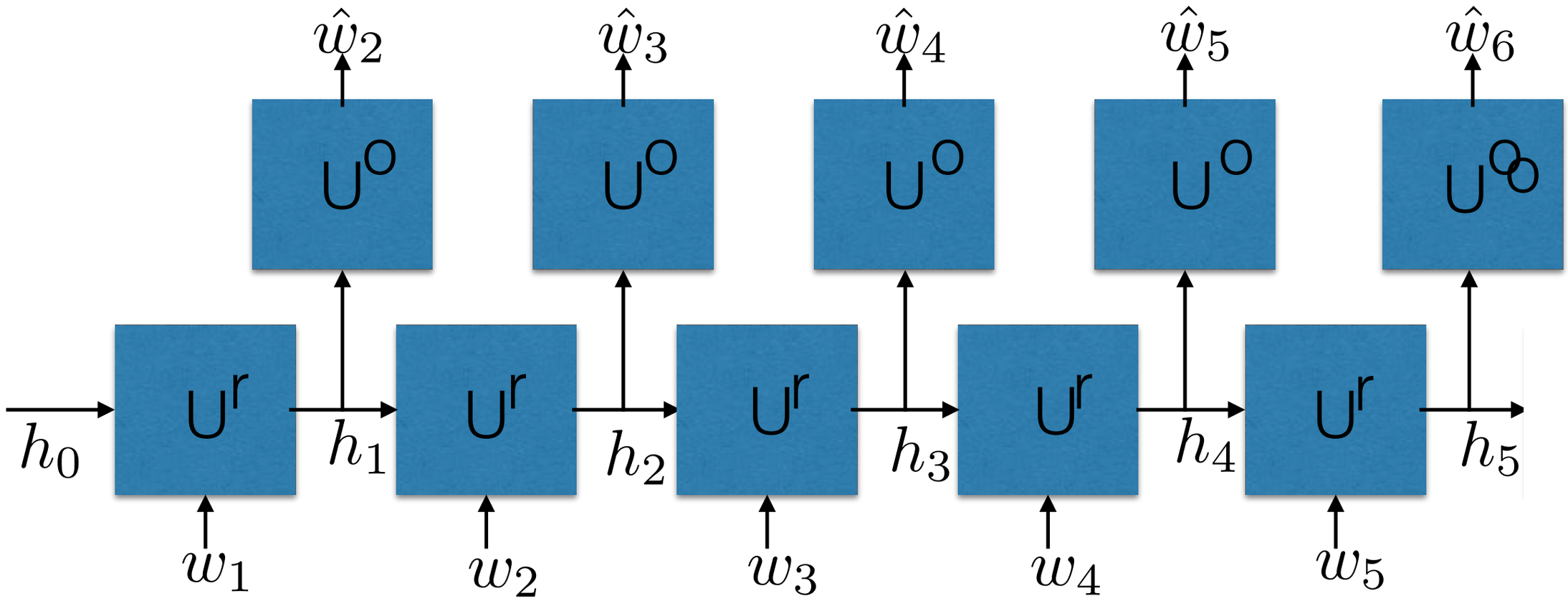
Elman "Finding structure in time" Cognitive Science 1990

RNN: Inference Time

- Elman RNN:

$$p(w_{k+1}|h) = \text{softmax}(U^o h_k + b^o)$$

$$h_k = \sigma(U^r h_{k-1} + U^i \mathbf{1}(w_k) + b^r)$$



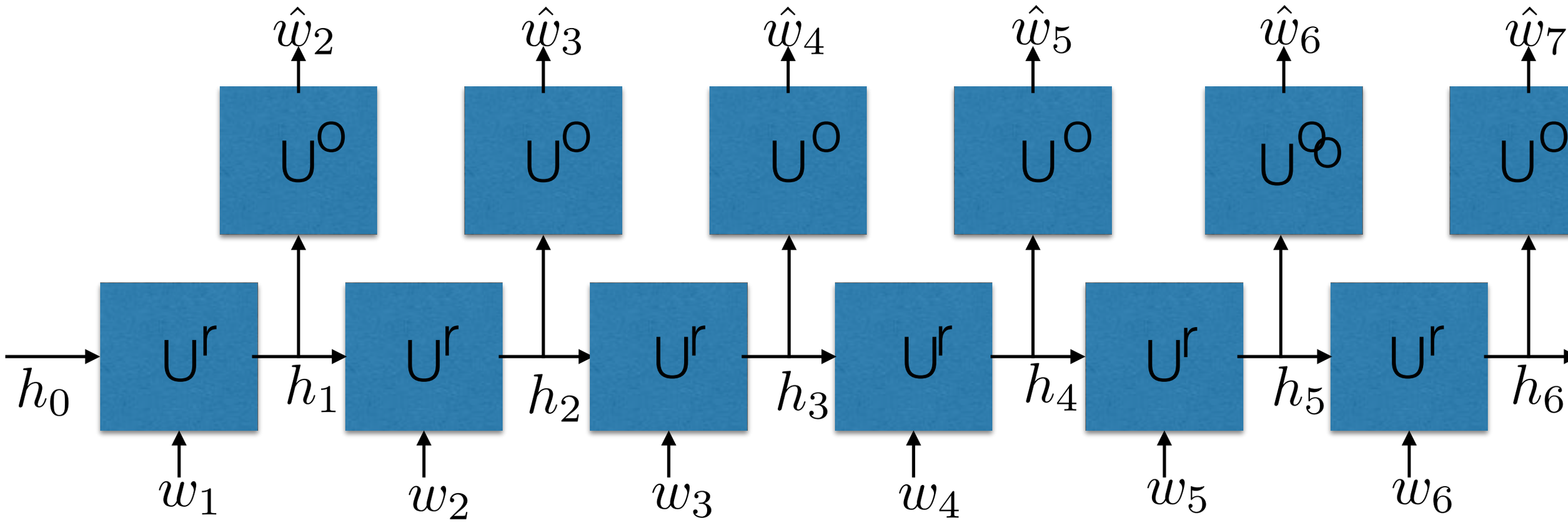
Elman "Finding structure in time" Cognitive Science 1990

RNN: Inference Time

- Elman RNN:

$$p(w_{k+1}|h) = \text{softmax}(U^o h_k + b^o)$$

$$h_k = \sigma(U^r h_{k-1} + U^i \mathbf{1}(w_k) + b^r)$$



Elman "Finding structure in time" Cognitive Science 1990

RNN: Inference Time

- Inference in an RNN is like a regular forward pass in a deep neural network, with two differences:
    - Weights are shared at every layer.
    - Inputs are provided at every layer.

RNN: Inference Time

- Inference in an RNN is like a regular forward pass in a deep neural network, with two differences:
    - Weights are shared at every layer.
    - Inputs are provided at every layer.

- Two possible applications:
    - Scoring: compute the log-likelihood of an input sequence (sum the log-prob scores at every step).
    - Generation: sample or take the max from the predicted distribution over words at each time step, and feed that prediction as input at the next time step.

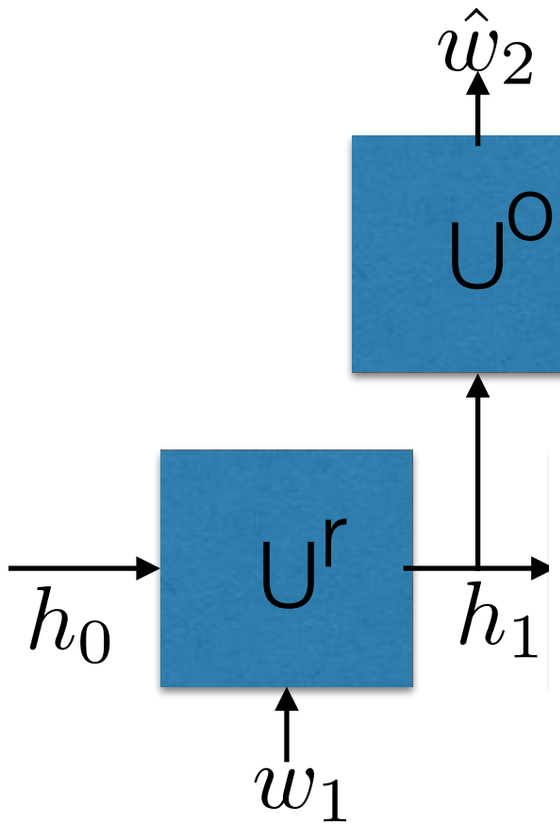
RNN: Training Time

- Truncated Back-Propagation Through Time:
    - Unfold RNN for only N steps and do:
        - Forward
        - Backward
        - Weight update
- Repeat the process on the following sequence of N words, but carry over the value of the last hidden state.

Werbos “Backpropagation through time: what does it do and how to do it” IEEE 1990

RNN: Truncated BPTT

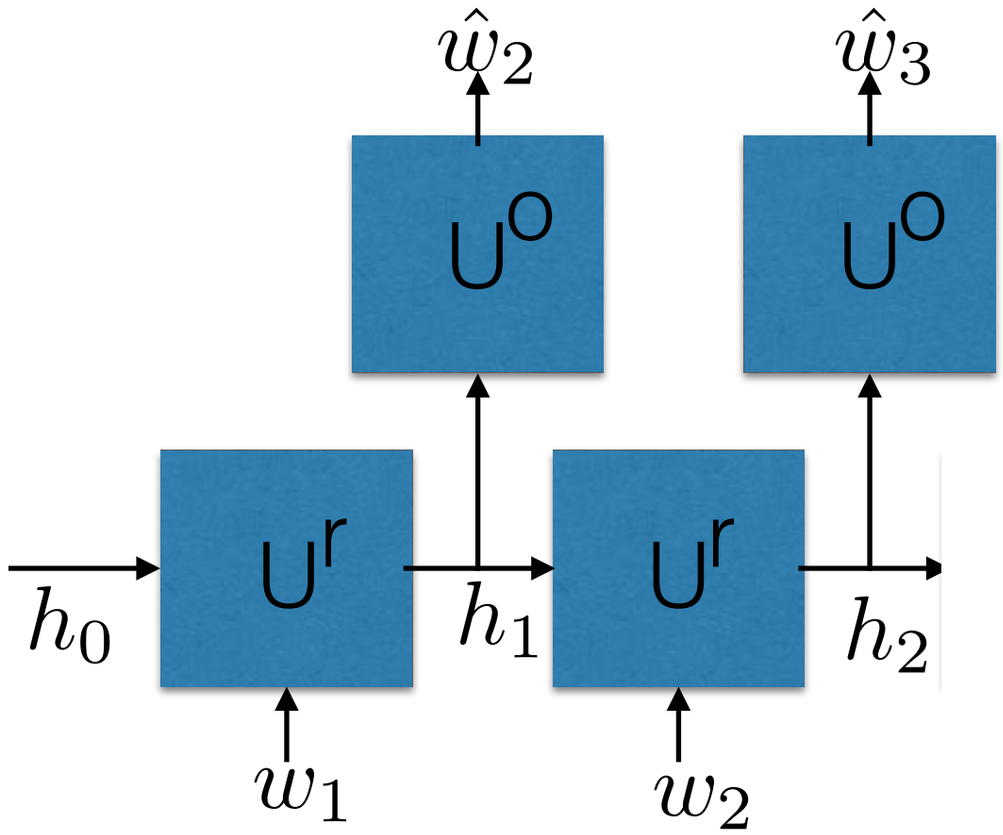
Forward Pass



Elman “Finding structure in time” Cognitive Science 1990

RNN: Truncated BPTT

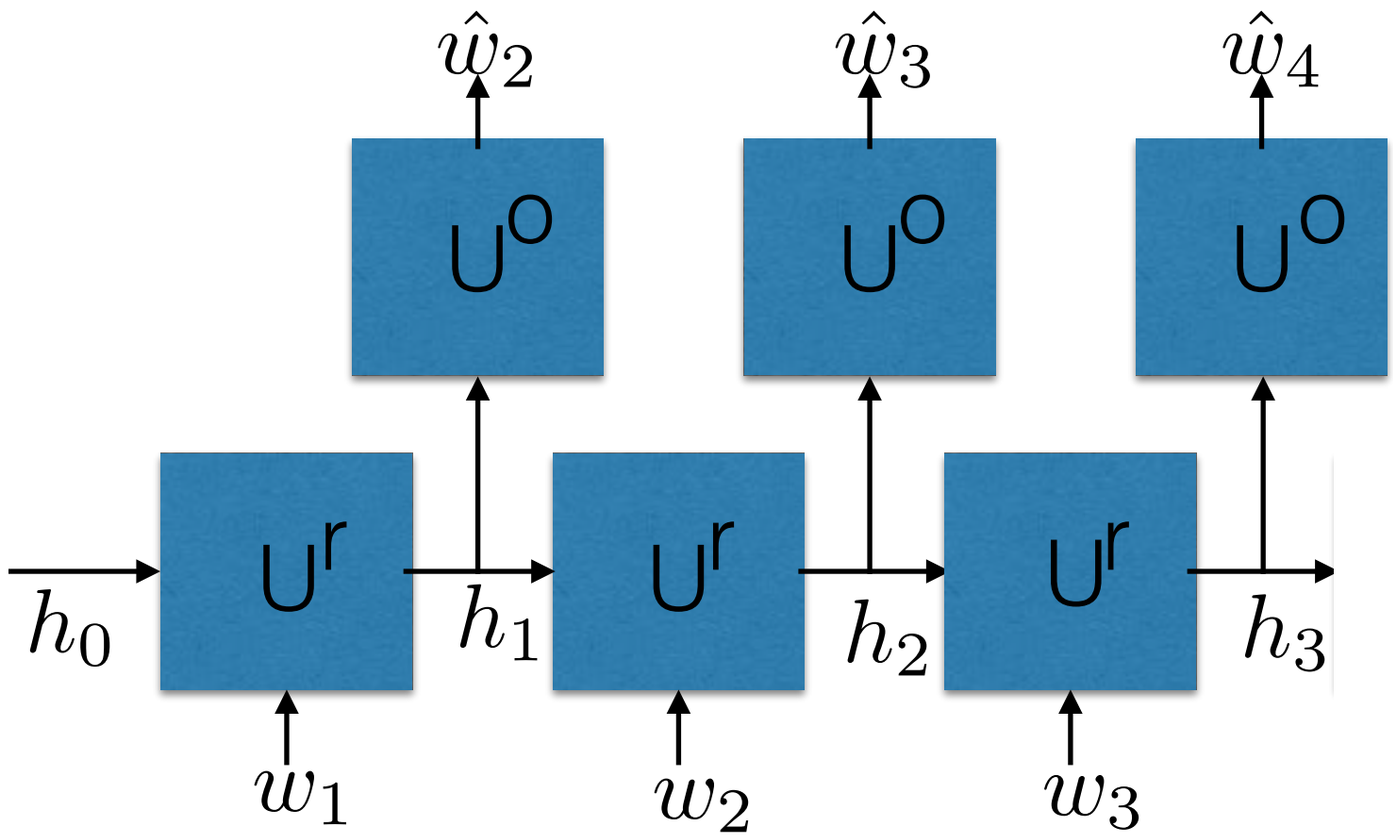
Forward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

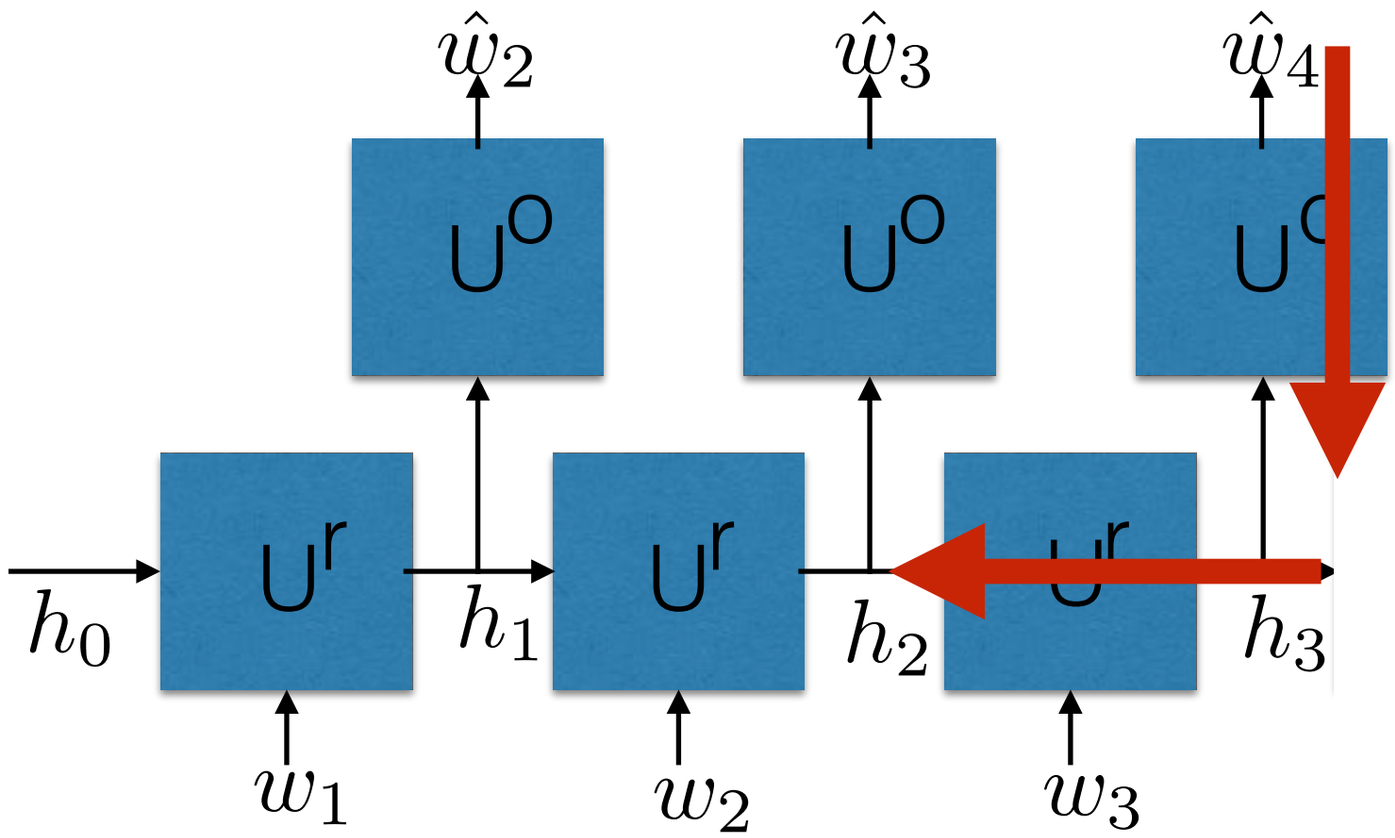
Forward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

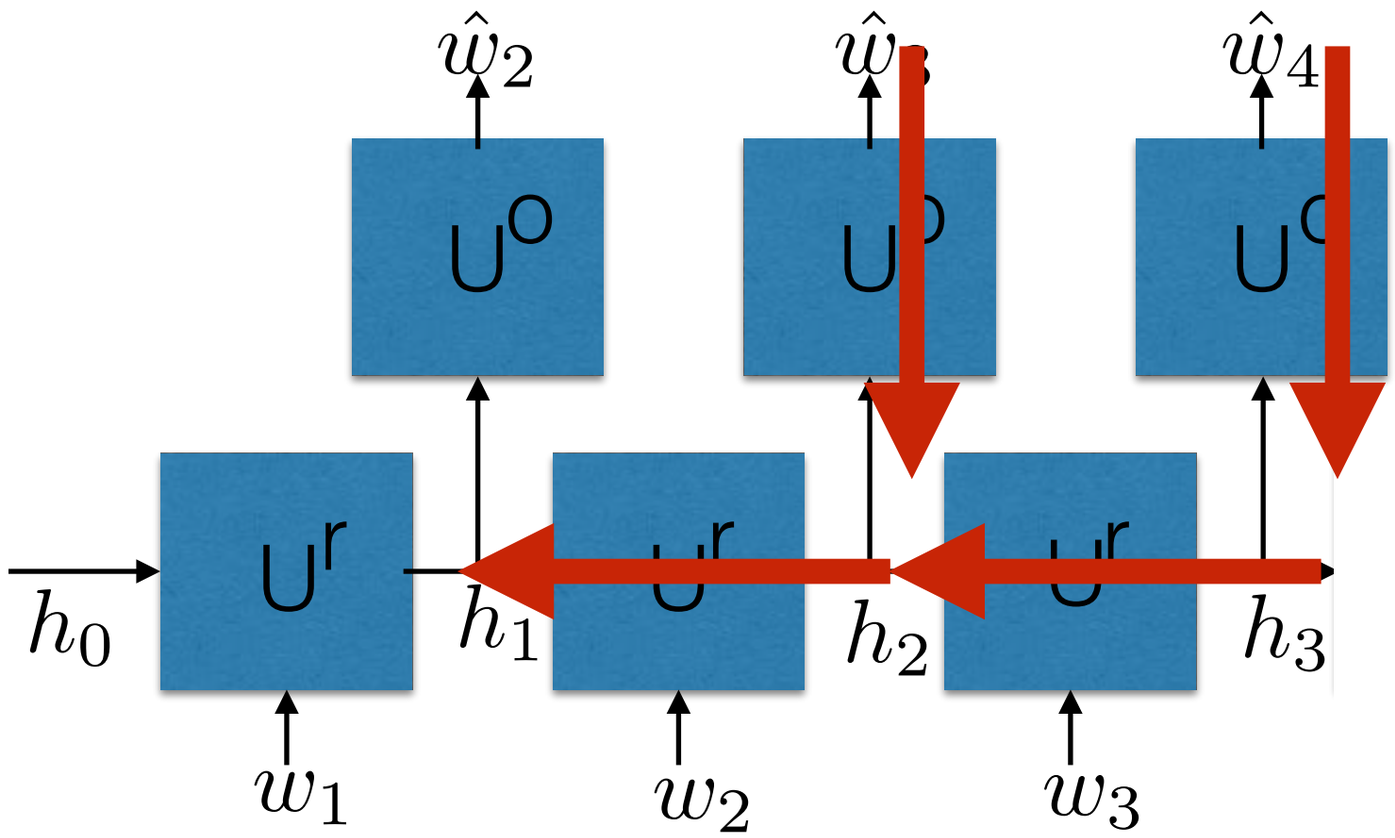
Backward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

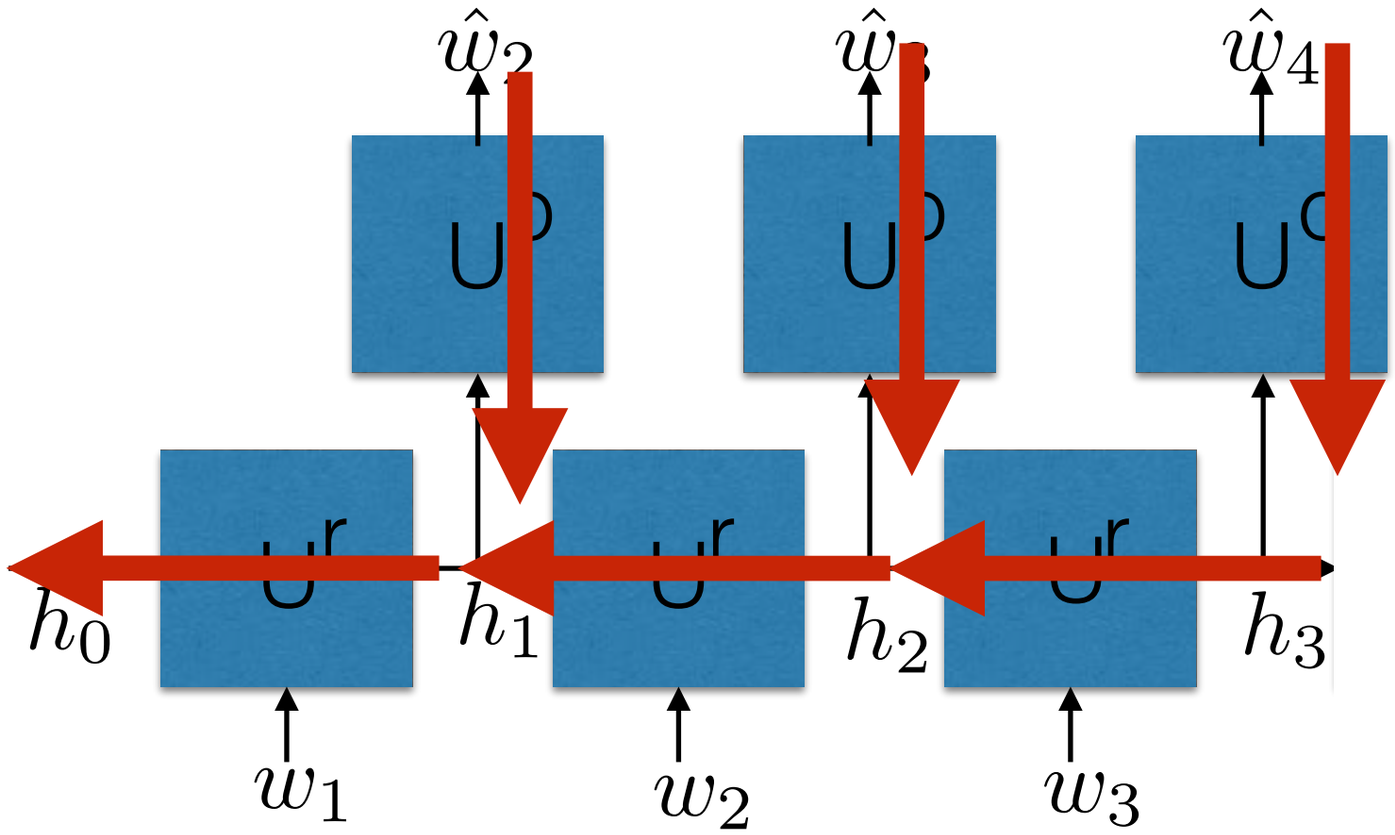
Backward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

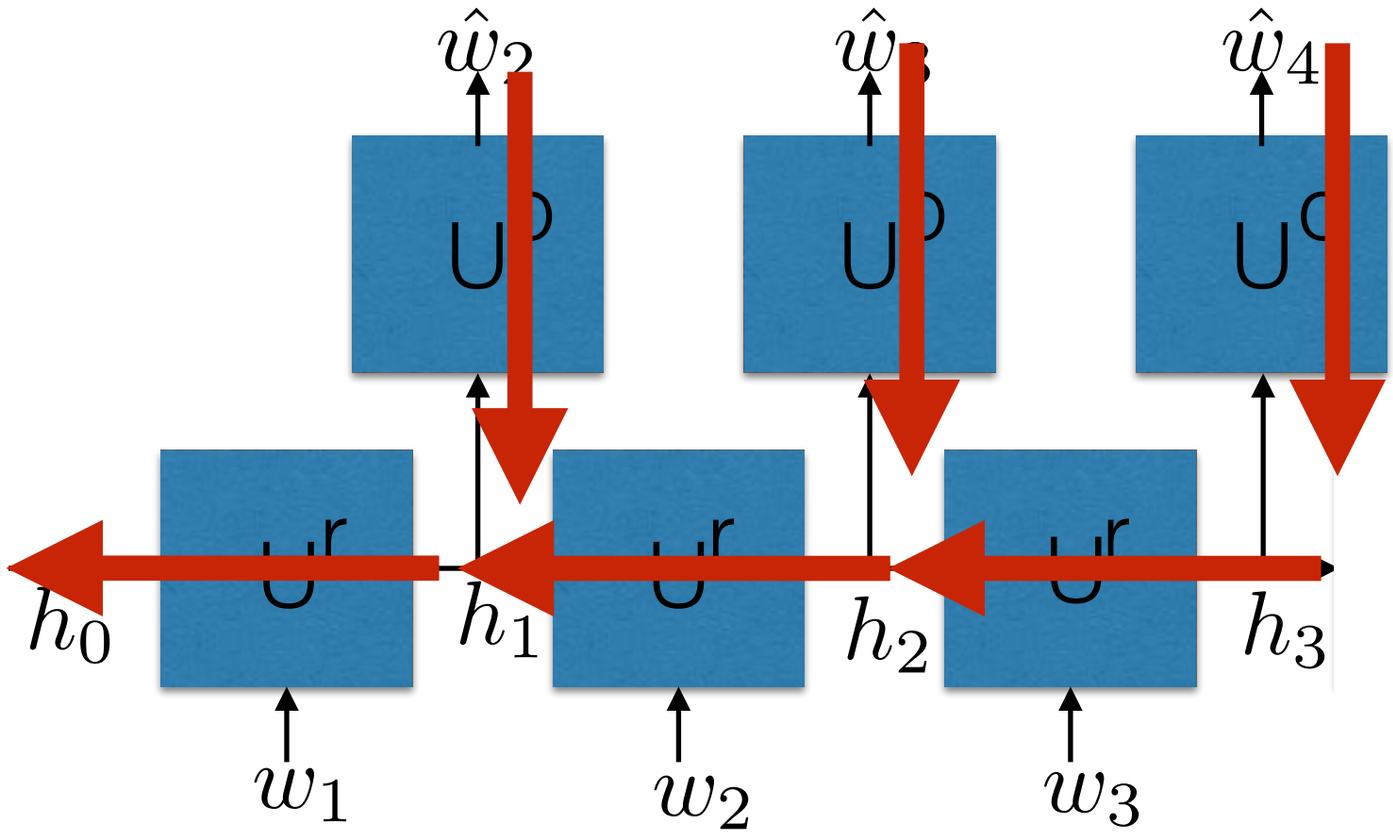
Backward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

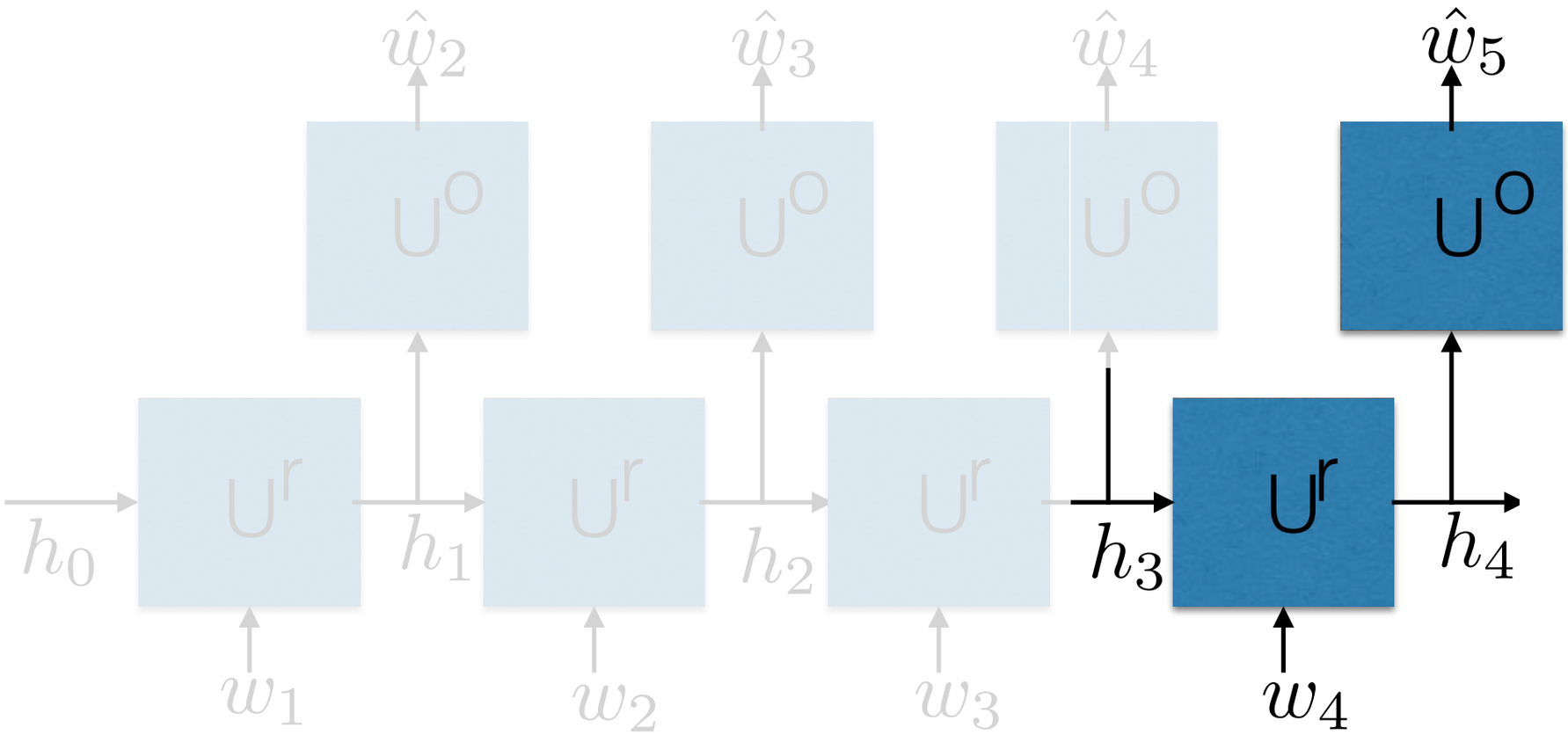
Parameter Update



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

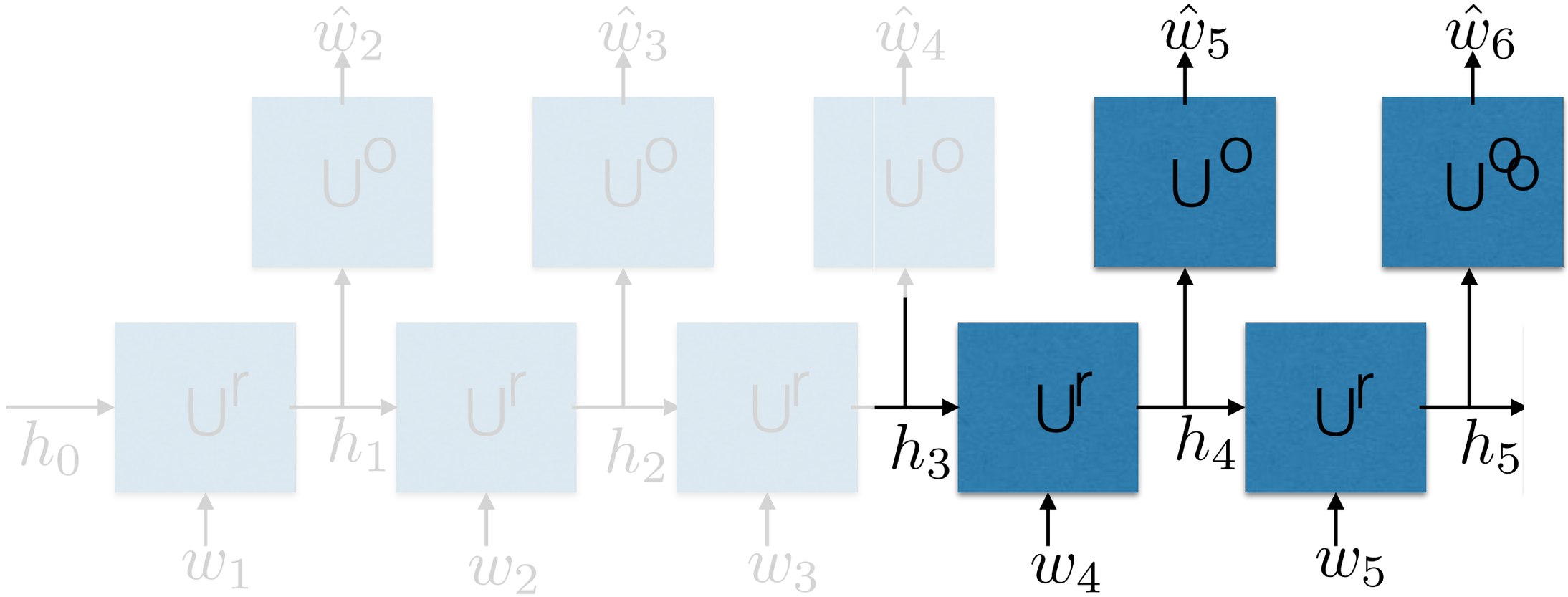
Forward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

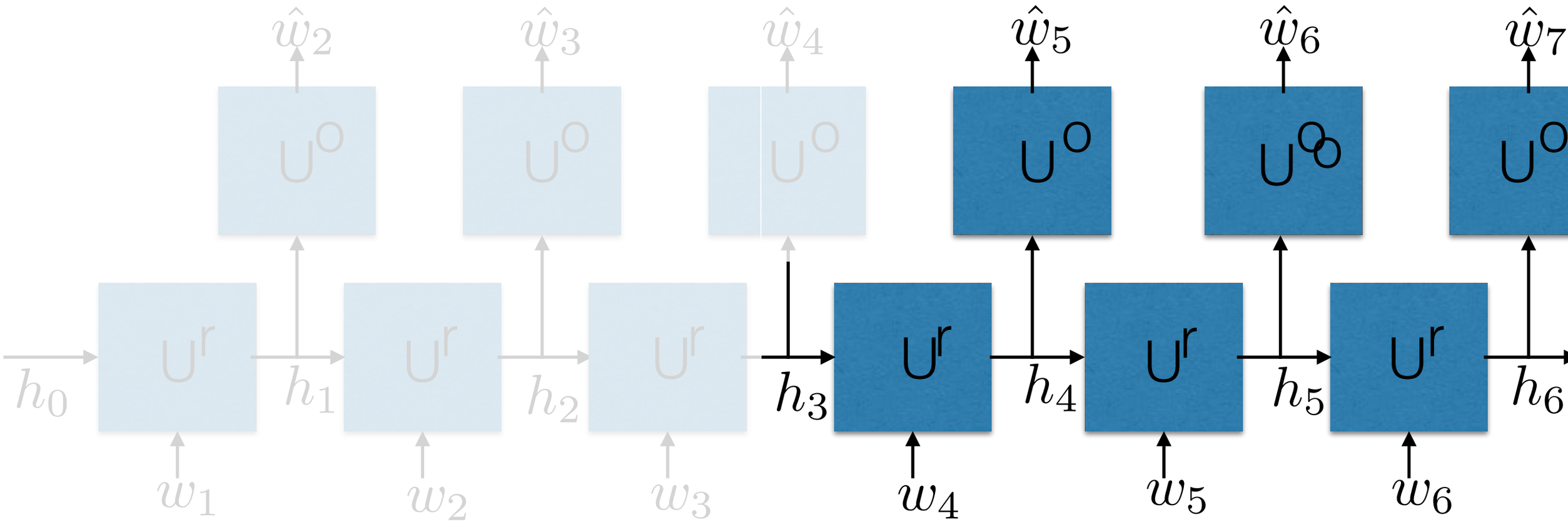
Forward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

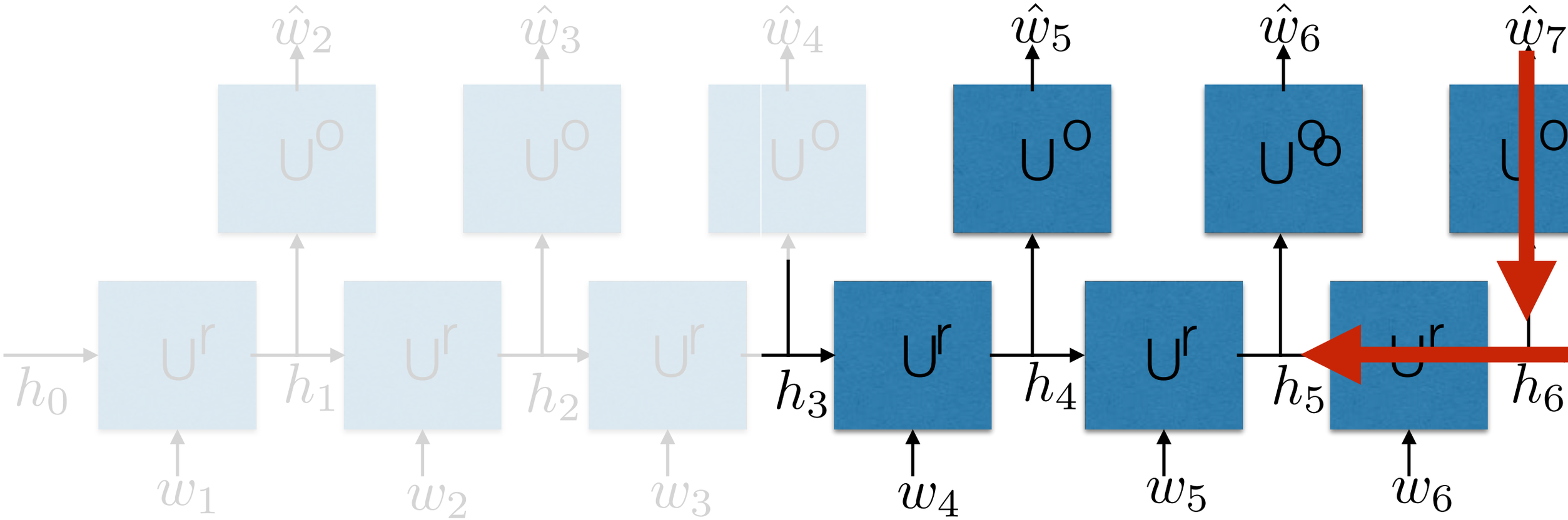
Forward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

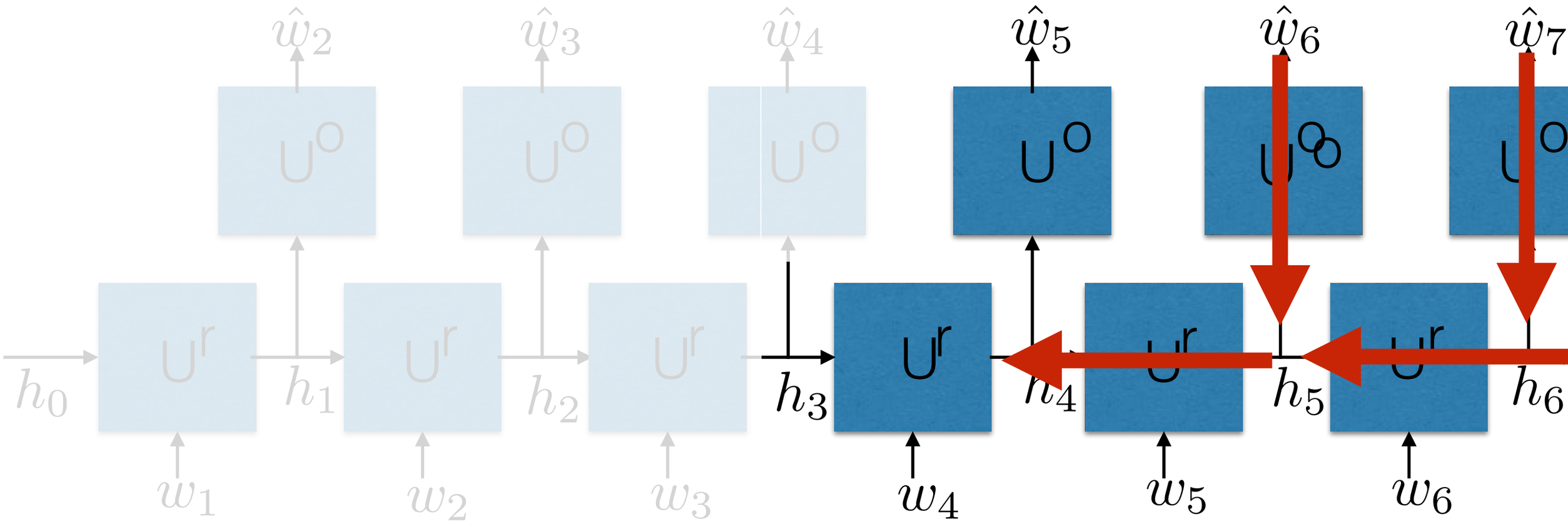
Backward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

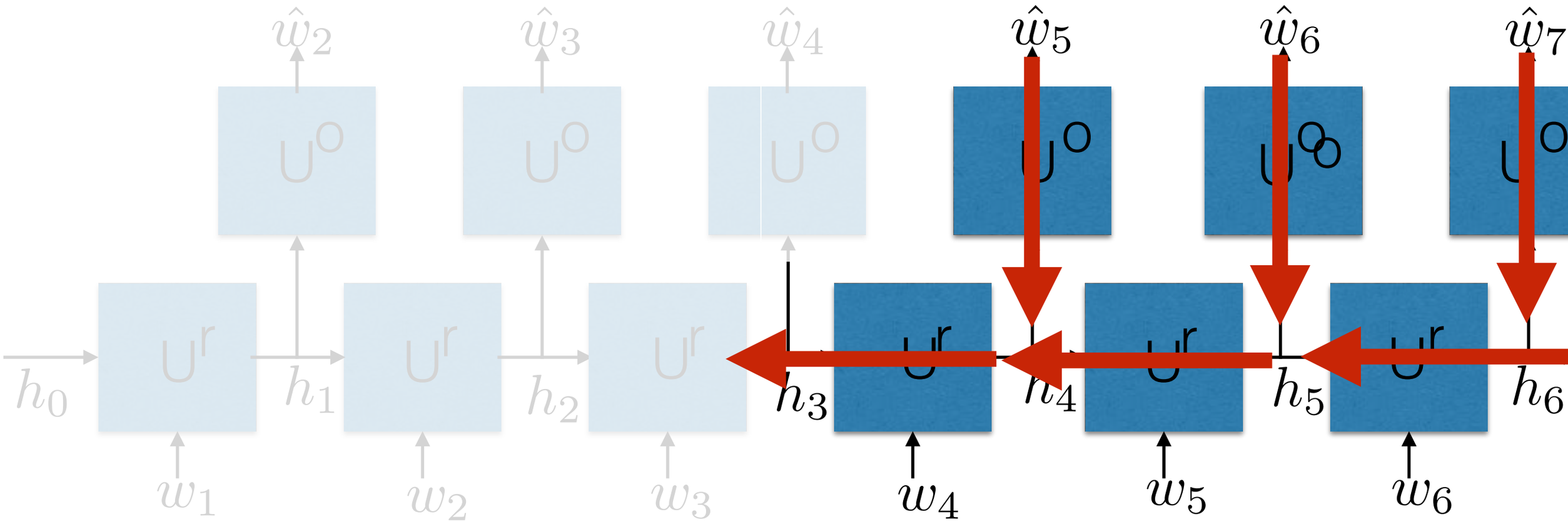
Backward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

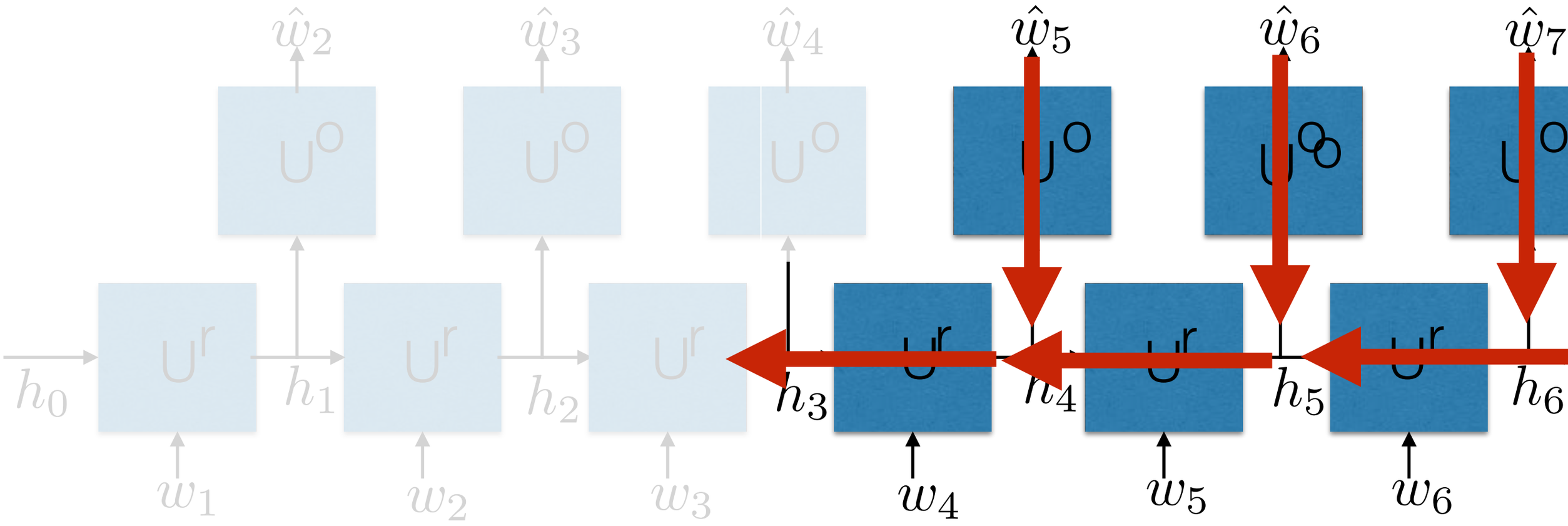
Backward Pass



Elman "Finding structure in time" Cognitive Science 1990

RNN: Truncated BPTT

Parameter Update



Elman "Finding structure in time" Cognitive Science 1990

Recap

- RNNs are more powerful because they capture a context of potentially “infinite” size.

- The hidden state of a RNN can be interpreted as a way to represent the history of what has been seen so far.

- RNNs can be useful to represent variable length sentences.

- There are lots of RNN variants. The best working ones have gating (units that multiply other units): e.g.: LSTM and GRU.

Gated Recurrent Unit RNN

Key idea: add gating units that enable hidden units to maintain (or reset) their state over time.

$$z_k = \sigma(S^i \mathbf{1}(w_k) + S^r h_{k-1})$$ **update gates**

$$r_k = \sigma(V^i \mathbf{1}(w_k) + V^r h_{k-1})$$ **reset gates**

Cho et al. “On the properties of NMT: encoder-decoder approaches” arXiv 2014

Gated Recurrent Unit RNN

Key idea: add gating units that enable hidden units to maintain (or reset) their state over time.

$$z_k = \sigma(S^i \mathbf{1}(w_k) + S^r h_{k-1})$$ **update gates**

$$r_k = \sigma(V^i \mathbf{1}(w_k) + V^r h_{k-1})$$ **reset gates**

$$\bar{h}_k = \tanh(U^i \mathbf{1}(w_k) + U^r (r_k \cdot h_{k-1}))$$ **candidate hiddens**

$$h_k = (1 - z_k) h_{k-1} + z_k \bar{h}_k$$ **new hiddens**

Cho et al. “On the properties of NMT: encoder-decoder approaches” arXiv 2014

Comparison

PennTreeBank	perplexity
n-gram	141
neural net	141
Elman RNN	123
GRU	-
LSTM	82

perplexity = $2^{H(p)}$

interpretation: average number of words the model is uncertain among (ideal value is 1).

Hochreiter et al. "Long short term memory" Neural Computation 1997
Grave et al. "Improving neural LMs with continuous cache" ICLR 2017
Mikolov et al. "Extensions of RNN LMs" ICASSP 2011

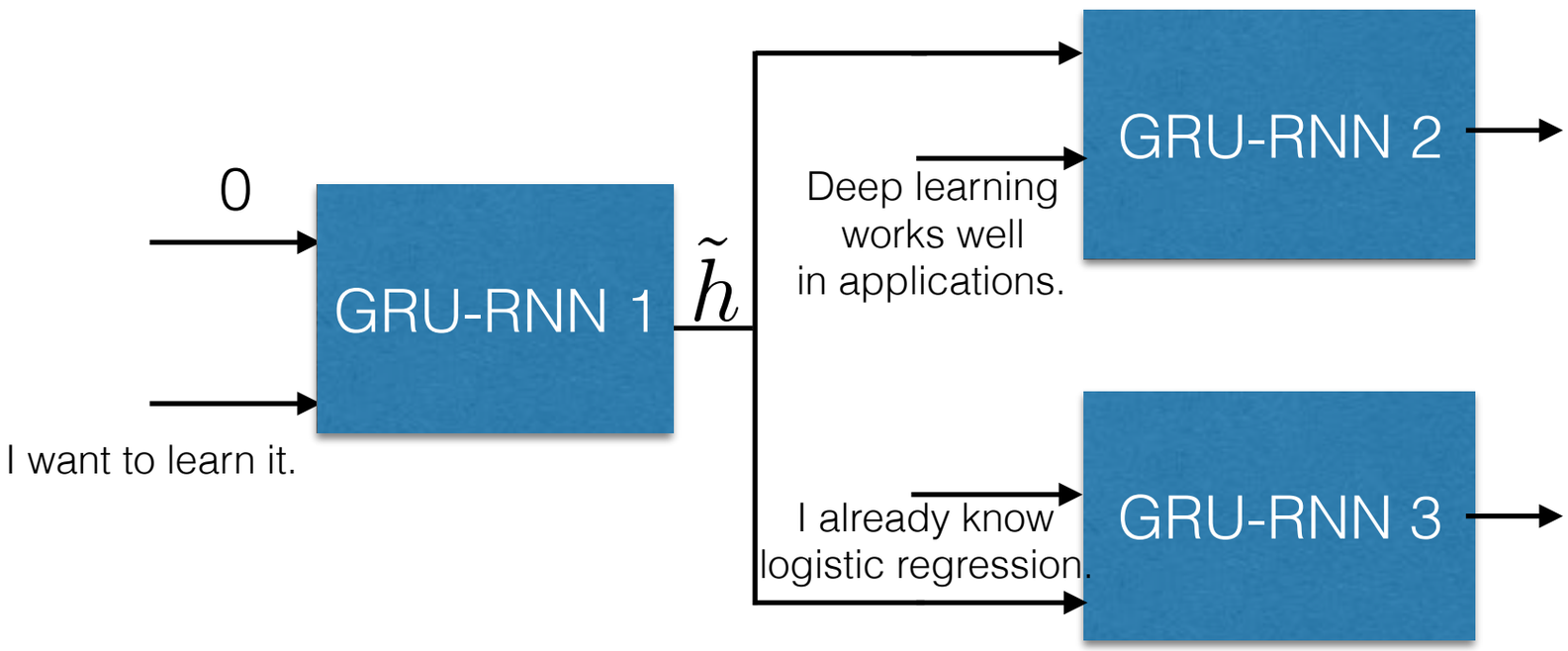
Recap

- There are several ways to represent sentences:
    - Bag of embeddings: strong baseline.
    - neural net language model: assumes fixed context, good for predicting the next word.
    - RNN: longer context, particularly good for predicting the next word.
- Why predicting just future words? How about predicting surrounding words in the context?

Skip-Thought Vectors

Key idea: 1) encode a sentence with an RNN, 2) use final hidden state to bias two other RNNs, one predicting the next sentence and one predicting the previous sentence.

Given: "Deep learning works well in applications. I want to learn it. I already know logistic regression."

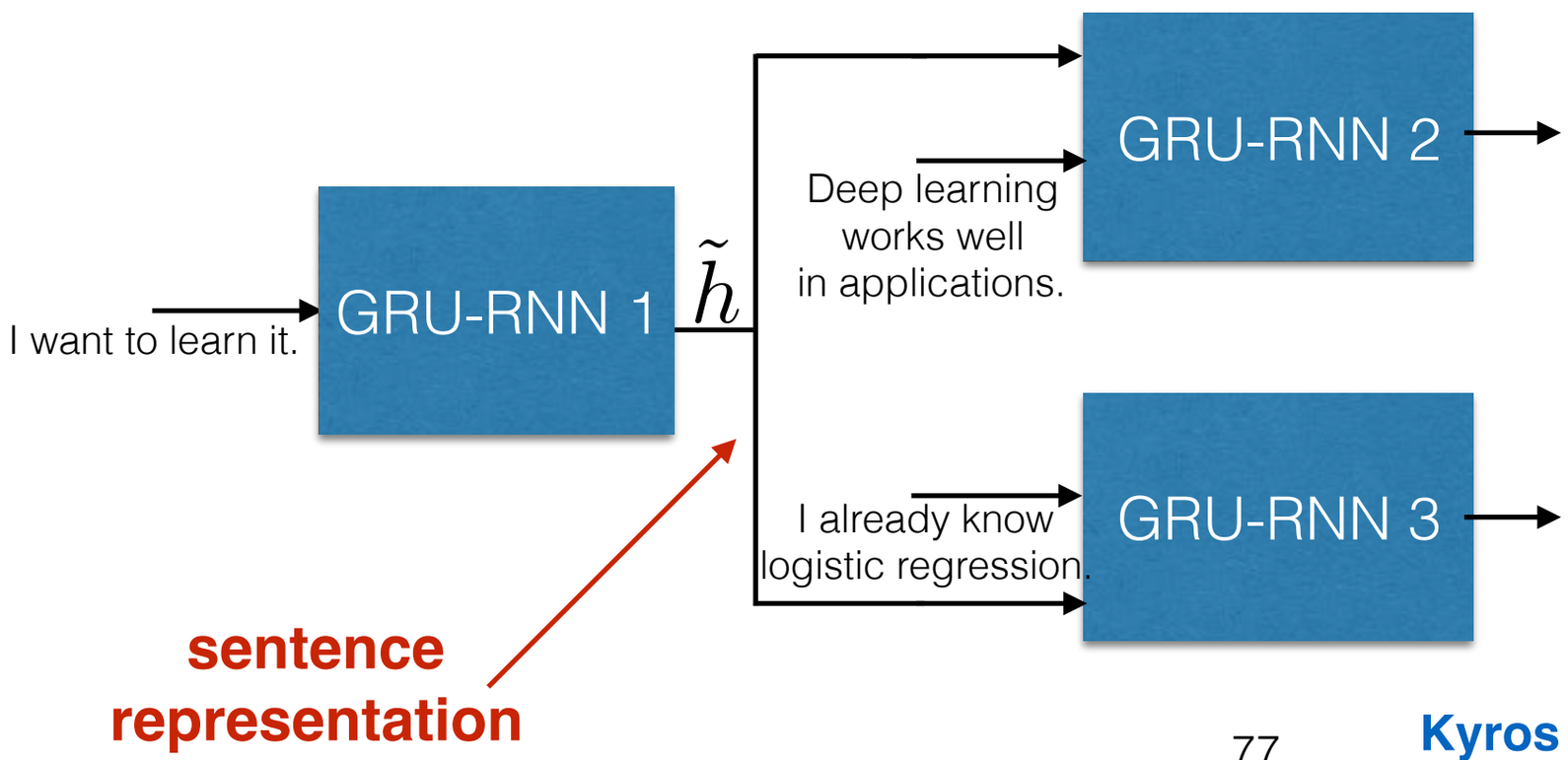


Kyros et al. "Skip-Thought vectors" arXiv 2015
https://github.com/ryankiros/skip-thoughts

Skip-Thought Vectors

Key idea: 1) encode a sentence with an RNN, 2) use final hidden state to bias two other RNNs, one predicting the next sentence and one predicting the previous sentence.

Given: "Deep learning works well in applications. I want to learn it. I already know logistic regression."

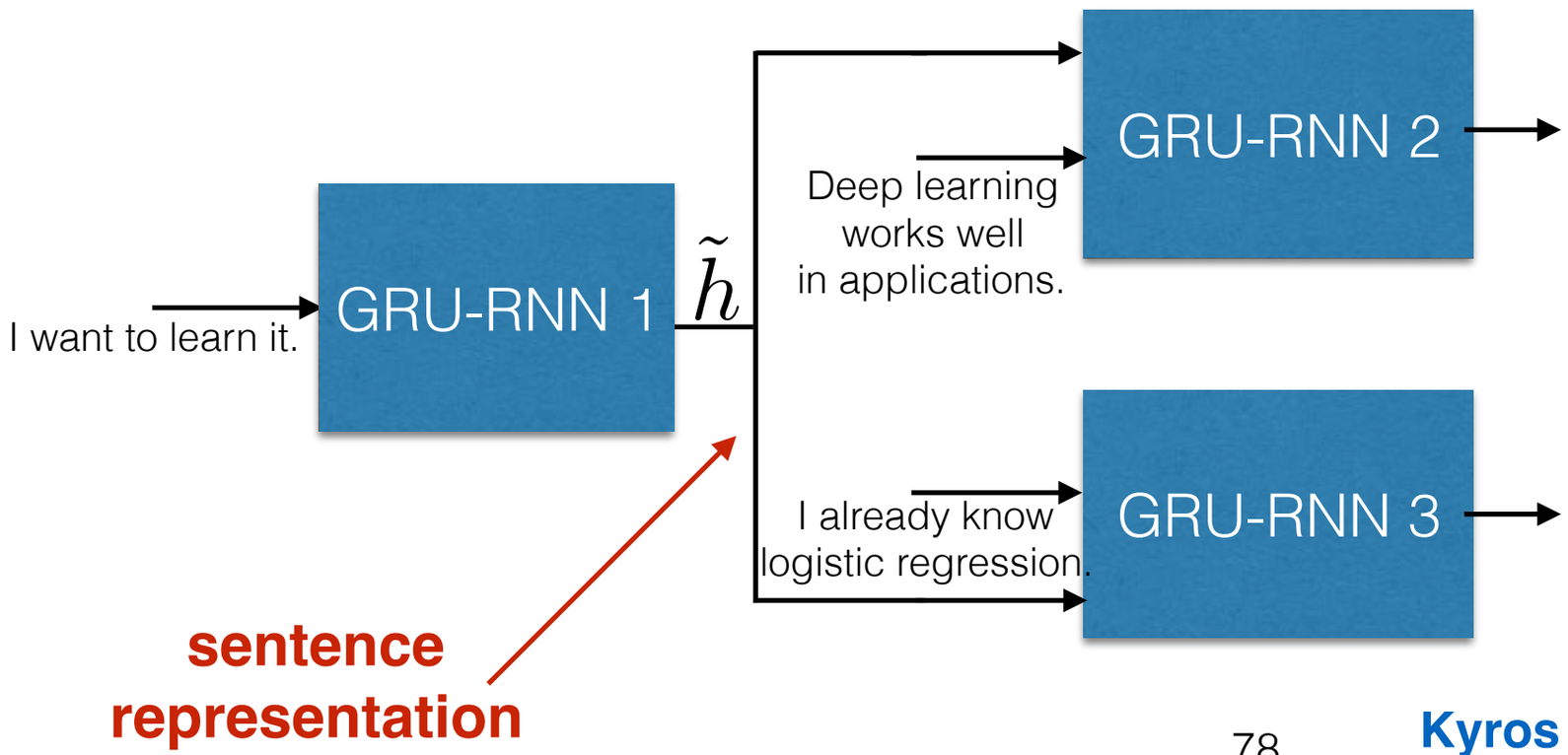


Kyros et al. "Skip-Thought vectors" arXiv 2015
https://github.com/ryankiros/skip-thoughts

Skip-Thought Vectors

Key idea: 1) encode a sentence with an RNN, 2) use final hidden state to bias two other RNNs, one predicting the next sentence and one predicting the previous sentence.

Given: "Deep learning works well in applications. I want to learn it. I already know logistic regression."



GRU-RNN 2 & 3 have slightly modified recurrent equations

$$z_k = \sigma(S^i \mathbf{1}(w_k) + S^r h_{k-1} + S^c \tilde{h})$$

$$r_k = \sigma(V^i \mathbf{1}(w_k) + V^r h_{k-1} + V^c \tilde{h})$$

...

Kyros et al. "Skip-Thought vectors" arXiv 2015
https://github.com/ryankiros/skip-thoughts

Skip-Thought Vectors

It's a generalization of word2vec to sentences, using RNNs to represent sentences.

Training:

Loss = cross entropy of previous sentence + cross entropy of next sentence.

It uses the BookCorpus dataset with sentences from 11,000 books.

Kyros et al. "Skip-Thought vectors" arXiv 2015
https://github.com/ryankiros/skip-thoughts

Skip-Thought Vectors

Method	MR	CR	SUBJ	MPQA	TREC
NB-SVM [41]	79.4	81.8	93.2	86.3	
MNB [41]	79.0	80.0	93.6	86.3	
cBoW [6]	77.2	79.9	91.3	86.4	87.3
GrConv [6]	76.3	81.3	89.5	84.5	88.4
RNN [6]	77.2	82.3	93.7	90.1	90.2
BRNN [6]	82.3	82.6	94.2	90.3	91.0
CNN [4]	81.5	85.0	93.4	89.6	**93.6**
AdaSent [6]	**83.1**	**86.3**	**95.5**	**93.3**	92.4
Paragraph-vector [7]	74.8	78.1	90.5	74.2	91.8
uni-skip	75.5	79.3	92.1	86.9	91.4
bi-skip	73.9	77.9	92.5	83.3	89.4
combine-skip	76.5	80.1	93.6	87.1	92.2
combine-skip + NB	80.4	81.3	93.6	87.5	

Table 7: Classification accuracies on several standard benchmarks. Results are grouped as follows: (a): bag-of-words models; (b): supervised compositional models; (c) Paragraph Vector (unsupervised learning of sentence representations); (d) ours. Best results overall are **bold** while best results outside of group (b) are underlined.

Kyros et al. “Skip-Thought vectors” arXiv 2015

Skip-Thought Vectors

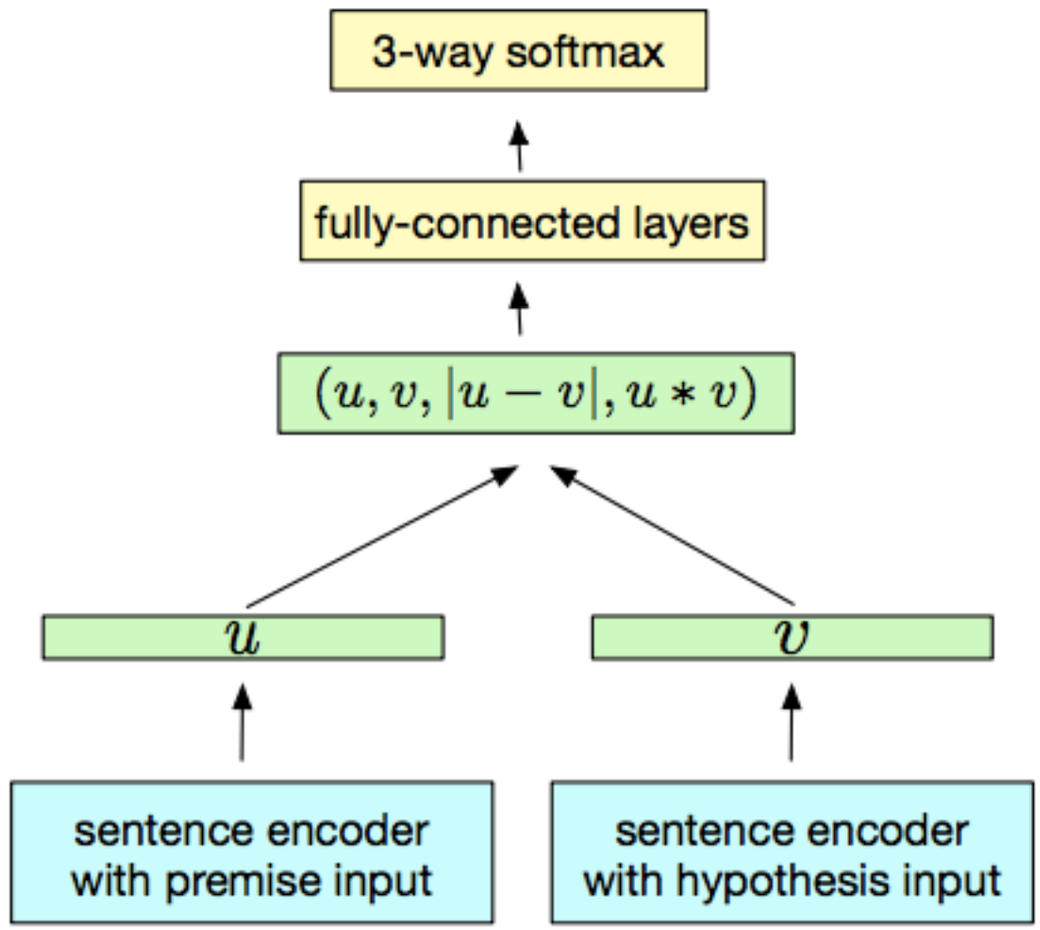
Example of generation:

she grabbed my hand . “ come on . ” she fluttered her bag in the air . “ i think we ’re at your place . i ca n’t come get you . ” he locked himself back up . “ no . she will . ” kyrian shook his head . “ we met ... that congratulations ... said no . ” the sweat on their fingertips ’s deeper from what had done it all of his flesh hard did n’t fade . cassie tensed between her arms suddenly grasping him as her sudden her senses returned to its big form . her chin trembled softly as she felt something unreadable in her light . it was dark . my body shook as i lost what i knew and be betrayed and i realize just how it ended . it was n’t as if i did n’t open a vein . this was all my fault , damaged me . i should have told toby before i was screaming . i should ’ve told someone that was an accident . never helped it . how can i do this , to steal my baby ’s prints ? ”

Kyros et al. “Skip-Thought vectors” arXiv 2015

Supervised Learning of Sentence Representations

If one has available labeled data on related tasks, it's always better to train in supervised mode. Representations transfer well to other tasks.



Conneau et al. "Supervised learning of universal sentence representations" arXiv 2017

Model	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	SICK-R	SICK-E	STS14
Unsupervised representation training (unordered sentences)										
Unigram-TFIDF	73.7	**79.2**	90.3	82.4	-	85.0	73.6/81.7	-	-	.58/.57
word2vec BOW	73.6	77.3	89.2	85.0	-	82.2	69.3/77.2	-	-	.58/.57
SIF	-	-	-	-	82.2	-	-	-	**84.6**	**.68**/ -
ParagraphVec (DBOW)	60.2	66.9	76.3	70.7	-	59.4	72.9/81.1	-	-	.42/.43
SDAE	74.6	78.0	**90.8**	86.9	-	78.4	**73.7**/80.7	-	-	.37/.38
GloVe BOW†	**78.7**	78.8	90.6	87.6	79.4	77.4	73.0/81.6	0.799	78.7	.46/.50
GloVe Positional Encoding†	76.3	77.4	90.4	87.1	80.6	80.8	72.5/81.2	0.789	77.9	.44/.48
BiLSTM-Max (untrained)†	77.5	**81.3**	89.6	**88.7**	80.7	**85.8**	73.2/81.6	**0.860**	83.4	.39/.48
Unsupervised representation training (ordered sentences)										
FastSent	70.8	78.4	88.7	80.6	-	76.8	72.2/80.3	-	-	**.63/.64**
FastSent+AE	71.8	76.7	88.8	81.5	-	80.4	71.2/79.1	-	-	.62/.62
SkipThought	76.5	80.1	93.6	87.1	82.0	**92.2**	**73.0/82.0**	**0.858**	82.3	.29/.35
SkipThought-LN	**79.4**	**83.1**	**93.7**	**89.3**	82.9	88.4	-	**0.858**	79.5	.44/.45
Supervised representation training										
CaptionRep (bow)	61.9	69.3	77.4	70.8	-	72.2	-	-	-	.46/.42
DictRep (bow)	76.7	78.7	90.7	87.2	-	81.0	68.4/76.8	-	-	**.67/.70**
NMT En-to-Fr	64.7	70.1	84.9	81.5	-	82.8	-	-		.43/.42
Paragram-phrase	-	-	-	-	79.7	-	-	0.849	83.1	- /**.71**
BiLSTM-Max (on SST)†	(*)	83.7	90.2	89.5	(*)	86.0	72.7/80.9	0.863	83.1	.55/.54
BiLSTM-Max (on SNLI)†	79.9	84.6	92.1	**89.8**	83.3	**88.7**	75.1/82.3	**0.885**	**86.3**	.66/.64
BiLSTM-Max (on AllNLI)†	**81.1**	**86.3**	**92.4**	**90.2**	**84.6**	88.2	**76.2**/**83.1**	**0.884**	**86.3**	**.68**/**.65**
Supervised methods (directly trained for each task – no transfer)										
Naive Bayes - SVM	79.4	81.8	93.2	86.3	83.1	-	-	-	-	-
AdaSent	83.1	86.3	95.5	93.3	-	92.4	-	-	-	-

Conneau et al. “Supervised learning of universal sentence representations” arXiv 2017

Recap

- Predict surrounding context is a general principle. It can be used to learn word and sentence representations in an unsupervised manner.

- Learning from labeled datasets, lets you transfer better features usually.

- Choice of sentence representation depends on sequence length, task, computational and memory constraints.

Questions?

Acknowledgements

I would like to thank Arthur Szlam for sharing his material about sparse recovery from bag-of-word embeddings.