Neural models for sequence labeling

CS-585

Natural Language Processing

Sonjia Waxmonsky

Based on slides from Kai Shu and Derrick Higgins

Word Embeddings Refresher

From HW 2 – Word embeddings encode semantic similarity

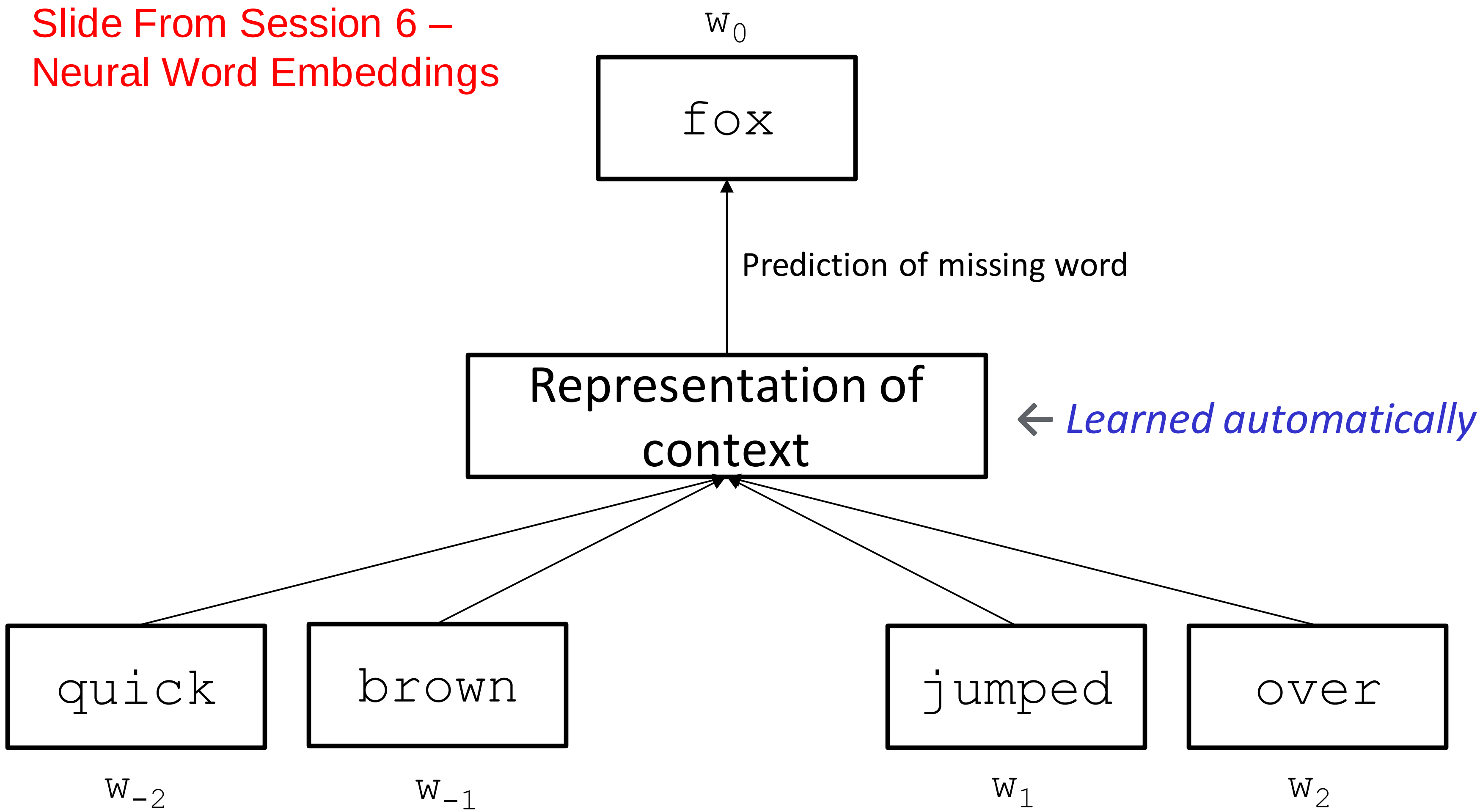
A note on Word2Vec - You can use this code snippet to find similar words; note the binary file "`GoogleNews-vectors-negative300.bin`" needs to be downloaded in advance.

```
import gensim

word2vec = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True, limit=100000)

word2vec.most_similar("blue")
```

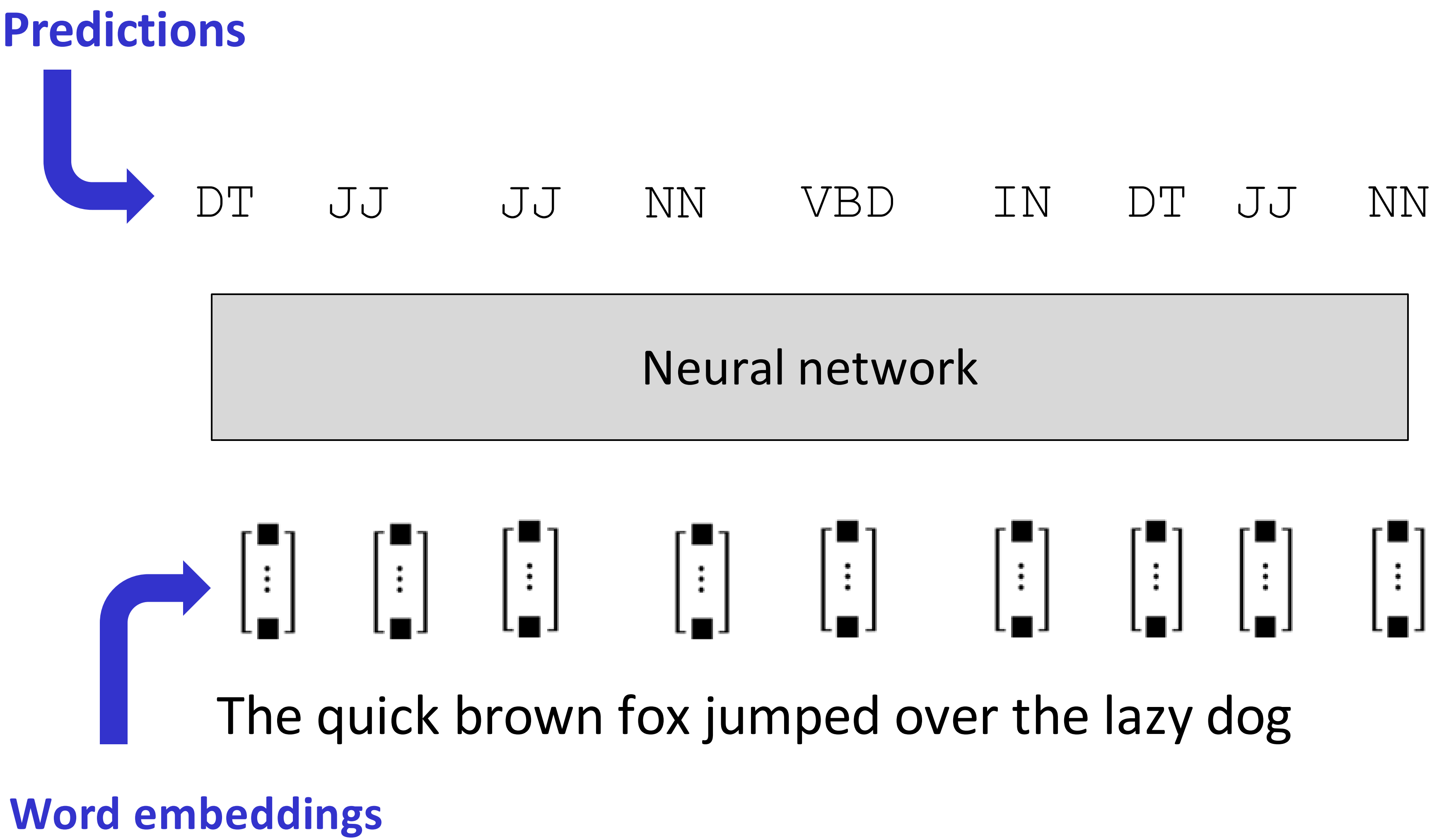
```
[('red', 0.7225173115730286),
 ('purple', 0.7134225964546204),
 ('white', 0.6606027483940125),
 ('maroon', 0.6557417511940002),
 ('colored', 0.6422423720359802),
 ('orange', 0.6421891450881958),
 ('yellow', 0.6376120448112488),
 ('teal', 0.6356961131095886),
 ('pink', 0.6343377232551575),
 ('pale_blue', 0.6308072209358215)]
                                                    [146]:
```

word2vec
Continuous Bag of Words (CBOW)

Slide From Session 6 –
Neural Word Embeddings



Neural networks for sequence labeling



Gradient descent

Slide From Session 10 – Logistic Regression (Model training)

- Batch gradient descent: accumulate updates across entire dataset before applying

η: Learning rate

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \sum_{i=1}^{|D|} \nabla \log L(\Theta; \vec{x_i}; y_i)$$

slow!

- Stochastic gradient descent: update parameters after gradient calculated for each training exemplar

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \nabla \log L(\Theta; \vec{x_i}; y_i)$$

fast!
but less stable

- Minibatch: Process updates for smaller samples of dataset (16-1024)

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \sum_{i=1}^{n} \nabla \log L(\Theta; \vec{x_i}; y_i)$$

faster, more stable

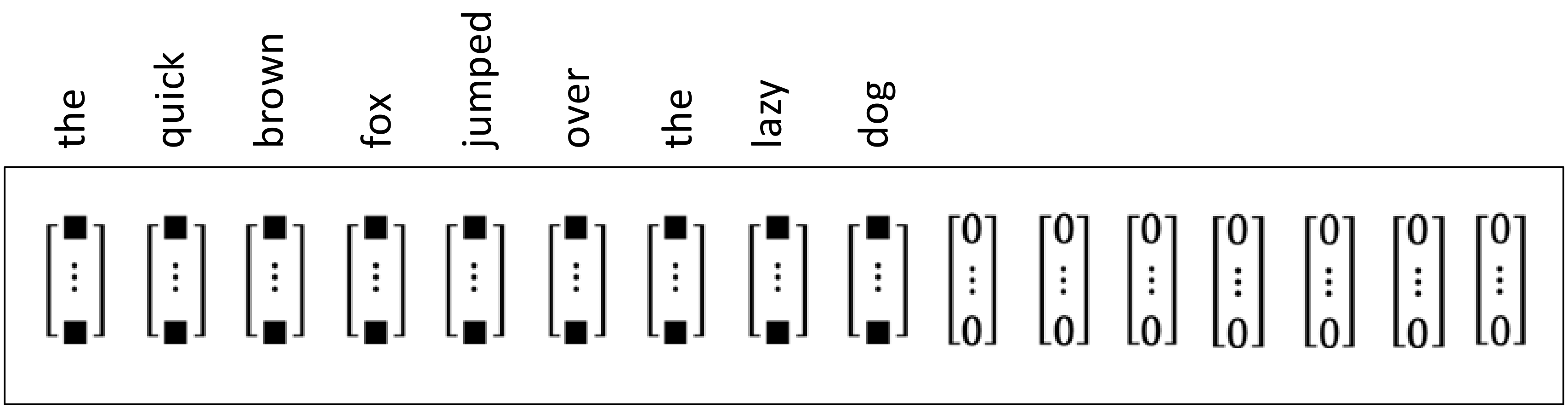
NNs for sequence labeling: preprocessing

- In order to process data efficiently for neural networks, we need to bundle the representations of multiple texts into a **minibatch** -- a single matrix

- Problem: text length is not a constant – some texts are longer than others
  - Solution: **zero-padding** and **truncation** of input sequence

Zero-padding

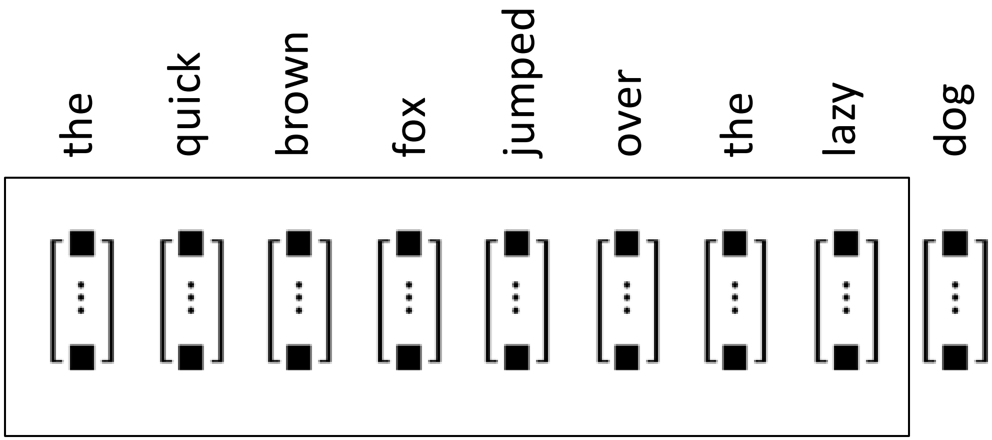
- Choose `sentence_length` for minibatches
- If a given sentence is too short, append zero vectors



`sentence_length = 16`

Truncation

- Choose `sentence_length` for minibatches
- If a given sentence is too long, discard extra words



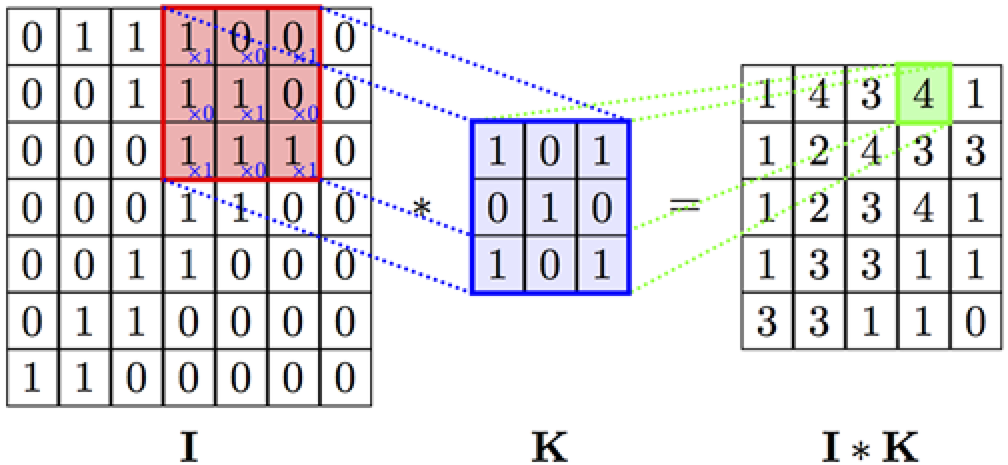
`sentence_length = 8`

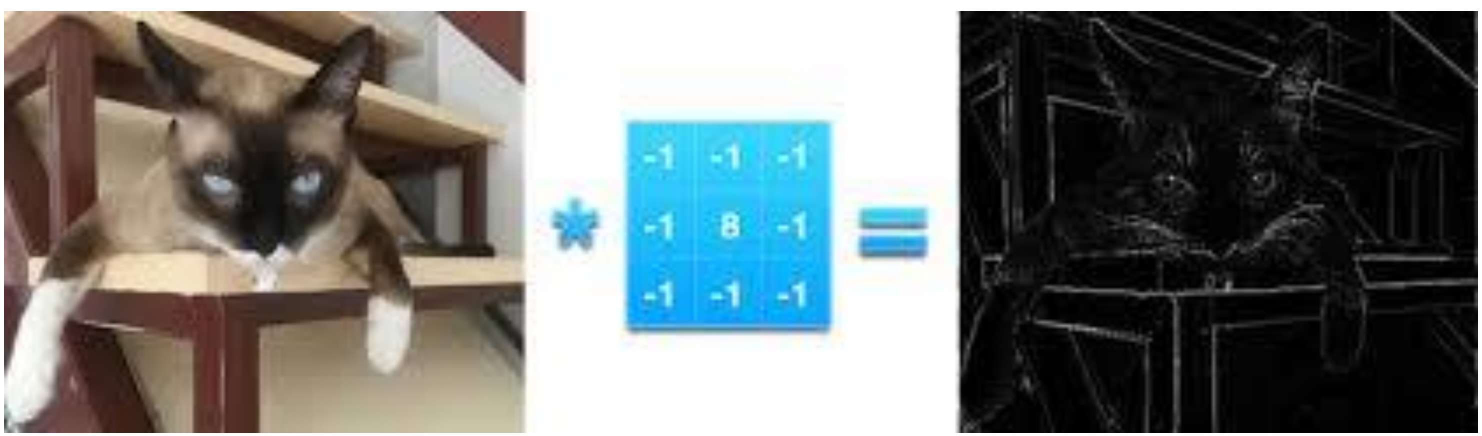
CONVOLUTIONAL NETWORKS FOR TEXT

Convolutional networks

- Convolutional neural networks (convnets, CNNs) use *convolution functions* to collect information from a *local receptive field* for prediction
- Properties
  - Convolutional network operations can be factored into operations that **run in parallel**, because operations at different points in the sequence are independent of one another
  - CNNs can only use **limited contextual** information for prediction, because each layer of the CNN aggregates information from a small local region (distance in words)

Convolutions in computer vision





https://jeiwan.cc/posts/til-convolution-filters-are-weights/

https://adeshpande3.github.io

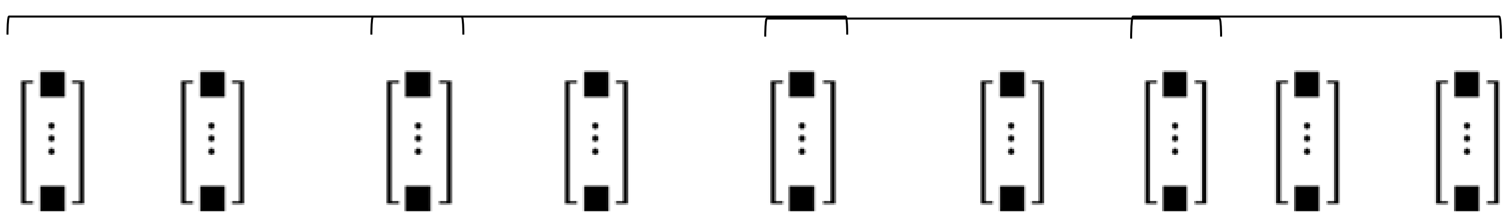
https://github.com/PetarV-/TikZ

Convolutions in NLP

- For NLP: 1d-convolutions (instead of 2-d)

$$\mathbf{a}^T(\mathbf{w}_3 + \mathbf{w}_4 + \mathbf{w}_5) \qquad \mathbf{a}^T(\mathbf{w}_7 + \mathbf{w}_8 + \mathbf{w}_9)$$

$$\mathbf{a}^T(\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3) \qquad \mathbf{a}^T(\mathbf{w}_5 + \mathbf{w}_6 + \mathbf{w}_7)$$



The quick brown fox jumped over the lazy dog

Convolutions as representation learning

- Convolutions are local feature extractors
  - In vision, detection of edges, corners, facial features, …
  - **In NLP, detection of negation, tense, local syntactic features, …**
- If word embeddings learn good representations for words, convolutions learn good ***higher-level*** representations for making predictions

Convolution Parameters

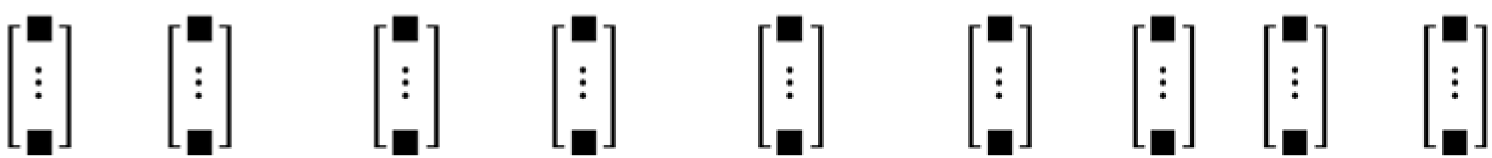
- **Width**: the size of the receptive field around the target location

width=3

$$\mathbf{a}^T(\mathbf{w}_2 + \mathbf{w}_3 + \mathbf{w}_4)$$

width=5

$$\mathbf{a}^T(\mathbf{w}_5 + \mathbf{w}_6 + \mathbf{w}_7 + \mathbf{w}_8 + \mathbf{w}_9)$$

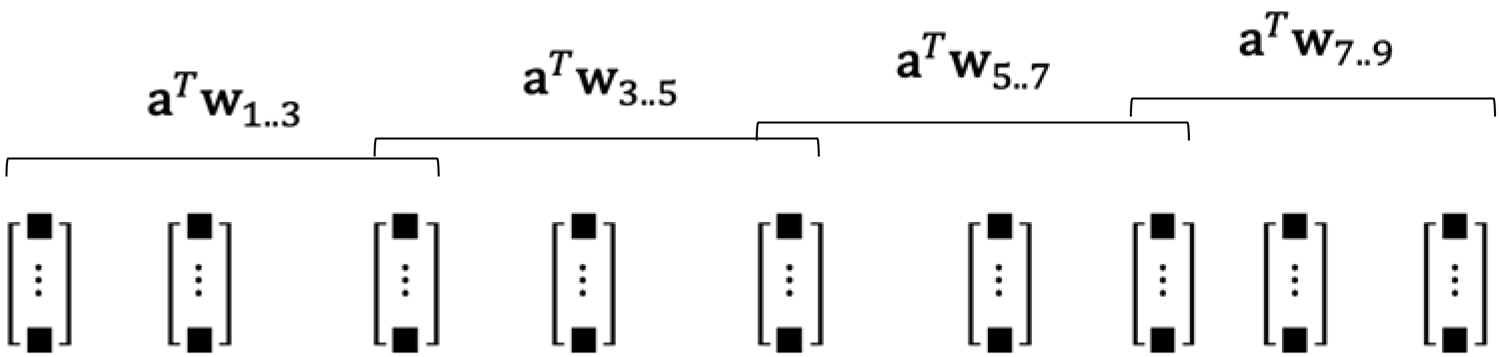


The quick brown fox jumped over the lazy dog

Convolution Parameters

- **Stride**: offset between adjacent applications of the convolution

stride=2

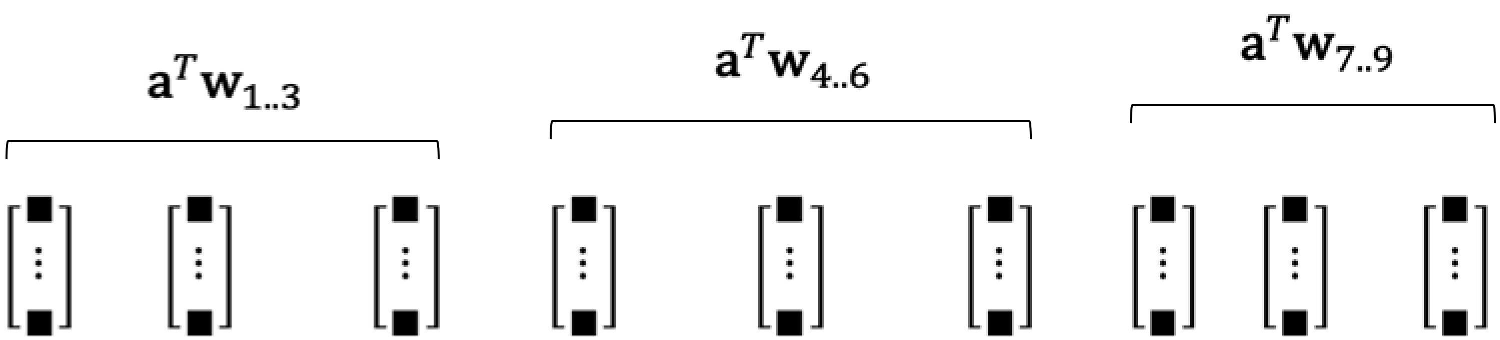


The quick brown fox jumped over the lazy dog

Convolution Parameters

- **Stride**: offset between adjacent applications of the convolution

stride=3



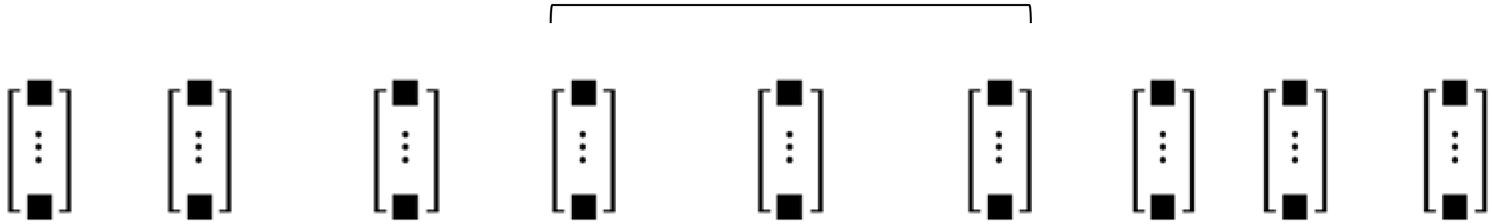
The quick brown fox jumped over the lazy dog

Convolution Parameters

- **Number of filters**: number of independent convolutions applied

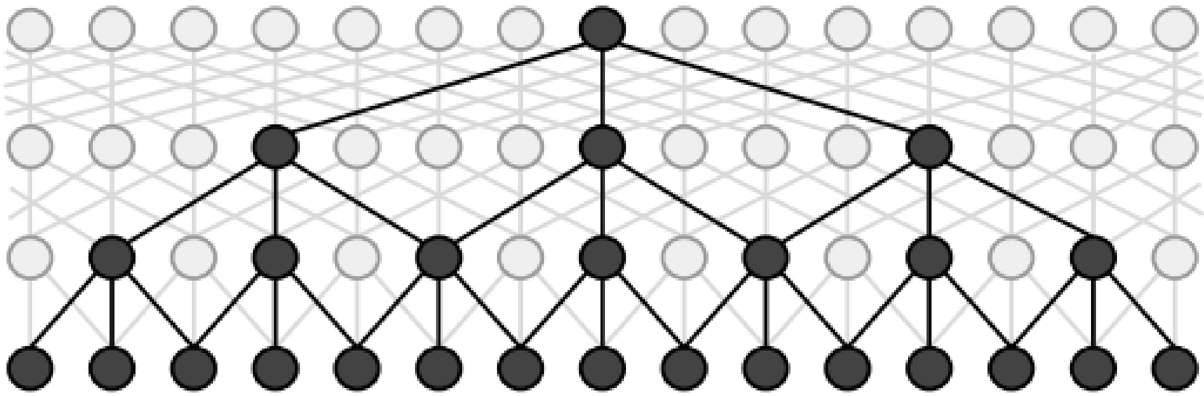
$$\begin{matrix} a_3^T w_{4..6} \\ a_2^T w_{4..6} \\ a_1^T w_{4..6} \end{matrix} = \begin{bmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \end{bmatrix}$$

num_filters=3



The quick brown fox jumped over the lazy dog

Dilated Convolutions



- Adaption for NLP and sequential tagging
- Structure grows with sequence length by adding layers
- Example: 4 stacked layers, convolution width of 3→ covers 31 token sequence

http://aclanthology.lst.uni-saarland.de/D17-1283.pdf

Training convolutional models for sequence labeling

- Loss function for a sentence/labeling is sum of cross-entropy loss across all labels to be predicted
  - Some care required in case of zero-padding
- Train using gradient descent, etc.

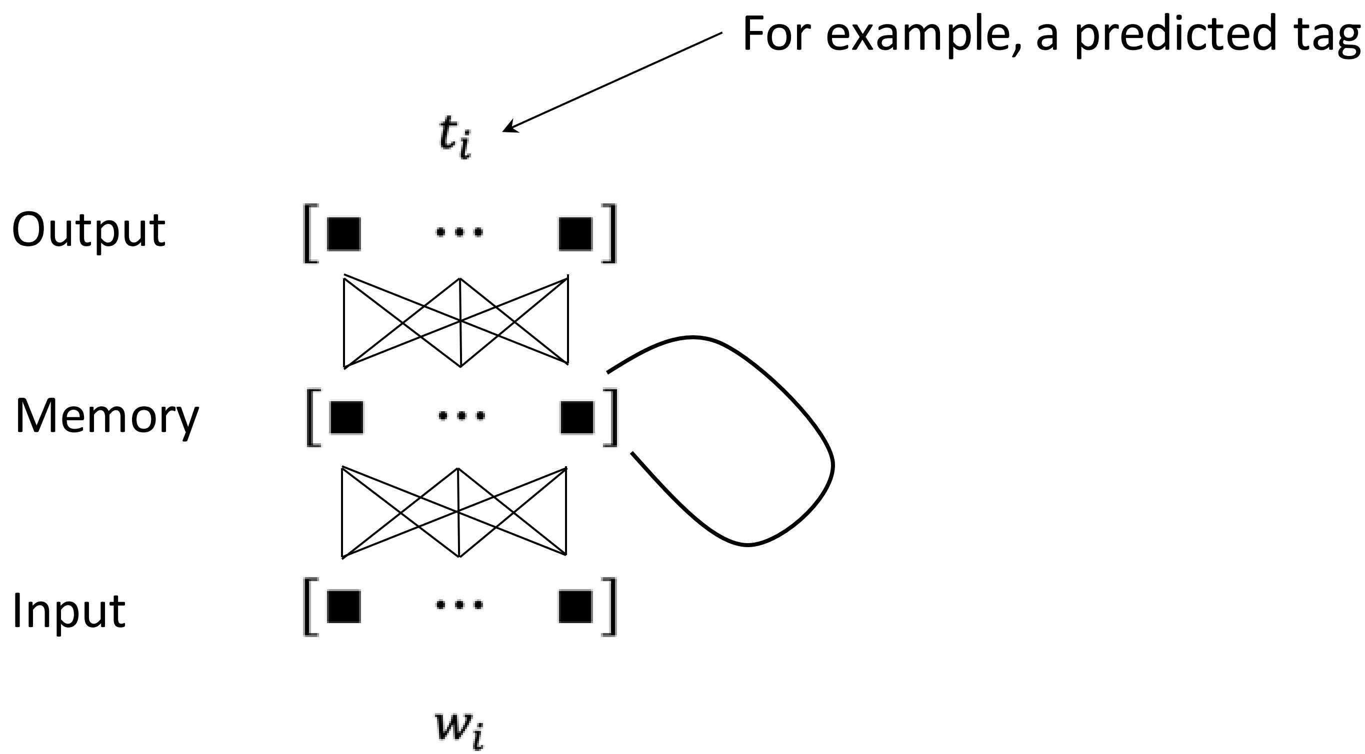
RECURRENT NETWORKS FOR TEXT

Recurrent networks

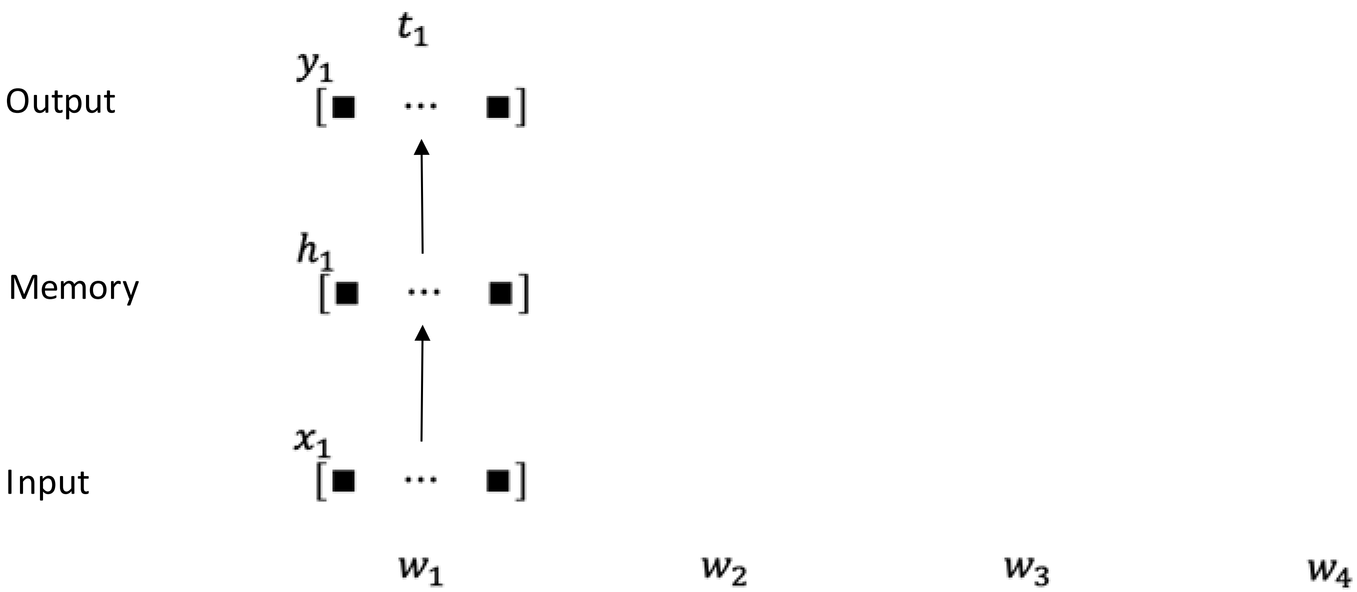
- Recurrent neural networks apply the **same operation to the input** at each time step, producing an output, but also updating an **internal memory state** that encodes relevant history to be used in prediction
- This memory state can allow **distant information** to influence the prediction made for a given word/label
- Because the memory state is **transferred** from time step to time step, the network is intrinsically sequential – it **cannot be effectively parallelized**

Recurrent Neural Networks (RNNs)

RNN Cell



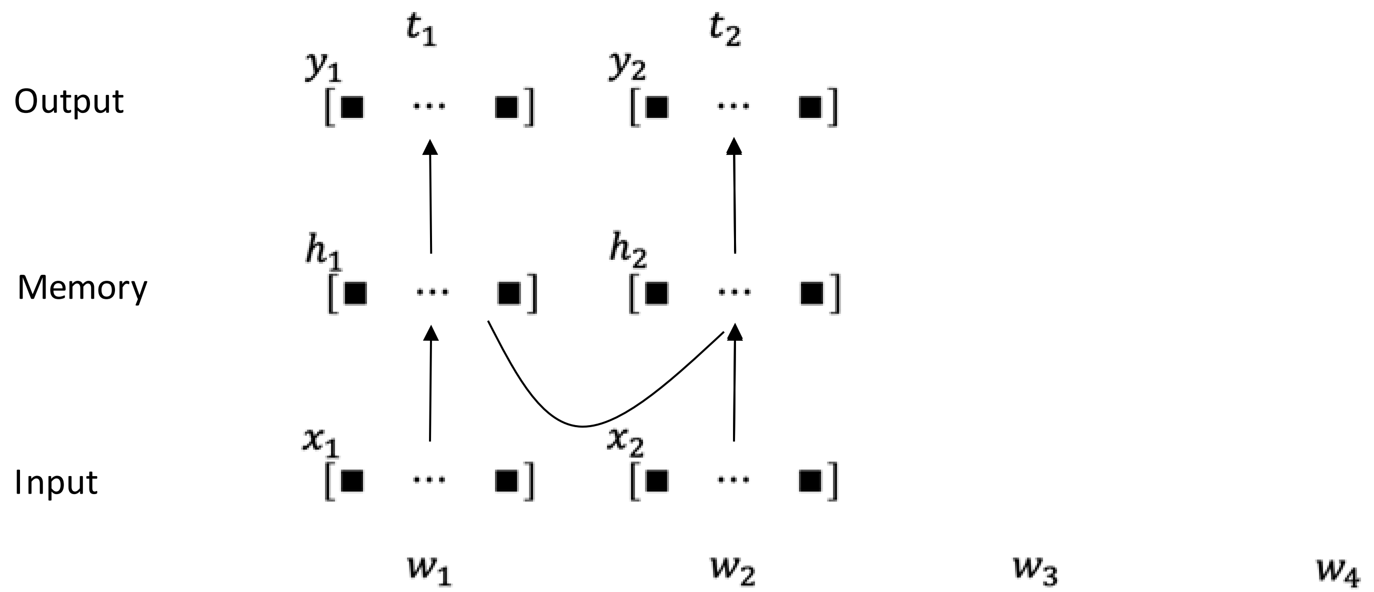
Recurrent Neural Networks (RNNs)



Unrolled representation of RNN

- RNN cell at each time step linked to memory state of prior time step

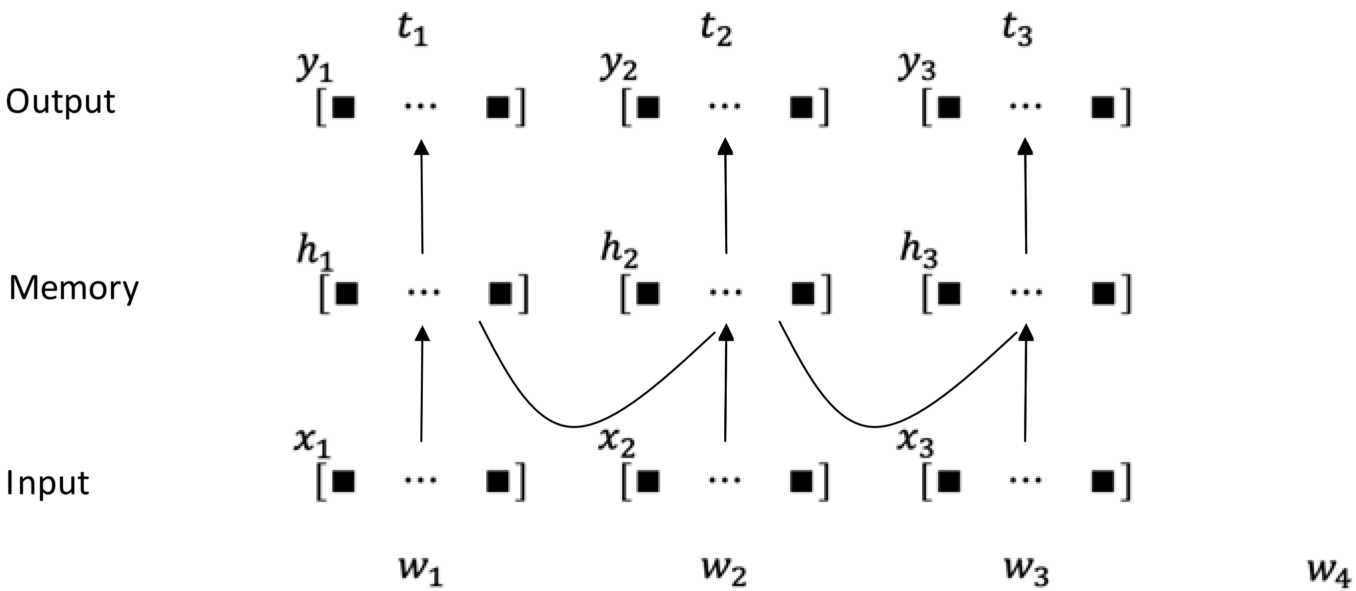
Recurrent Neural Networks (RNNs)



Unrolled representation of RNN

- RNN cell at each time step linked to memory state of prior time step

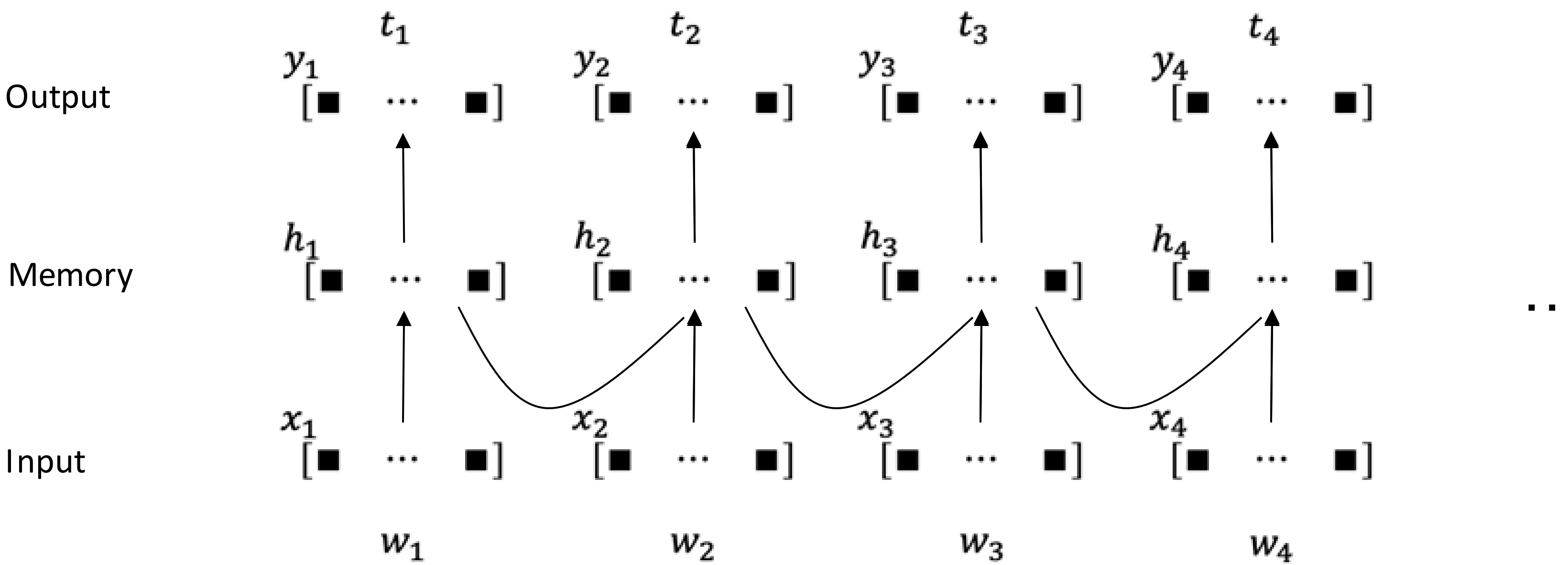
Recurrent Neural Networks (RNNs)



Unrolled representation of RNN

- RNN cell at each time step linked to memory state of prior time step

Recurrent Neural Networks (RNNs)



Unrolled representation of RNN

- RNN cell at each time step linked to memory state of prior time step

Recurrent Neural Networks (RNNs)

$$h_i = \sigma(W x_i + U h_{i-1} + b_h)$$

Memory state

Input vector

Previous memory state

Bias vector

$$y_i = \sigma(V h_i + b_y)$$

Recurrent Neural Networks (RNNs)

$$h_i = \sigma(Wx_i + Uh_{i-1} + b_h)$$

Logistic sigmoid

$$\frac{e^a}{1+e^a}$$

Input weight matrix

Recurrent weight matrix

$$y_i = \sigma(Vh_i + b_y)$$

Recurrent Neural Networks (RNNs)

$$h_i = \sigma(Wx_i + Uh_{i-1} + b_h)$$

Memory state

Output weight matrix

Bias vector

$$y_i = \sigma(Vh_i + b_y)$$

Recurrent Neural Networks (RNNs)

$$y_5 = \sigma(Vh_5 + b_y)$$

$$\sigma(Wx_5 + Uh_4 + b_h)$$

$$\sigma(Wx_4 + Uh_3 + b_h)$$

$$\sigma(Wx_3 + Uh_2 + b_h)$$

$$\sigma(Wx_2 + Uh_1 + b_h)$$

$$\sigma(Wx_1 + U\mathbf{0} + b_h)$$

Problems with RNNs

As we've seen, information needs to travel a long way in an RNN to get from the error signal / loss function (y) to some inputs (x_i)

By the chain rule of differentiation, the gradient of the loss function will have the form

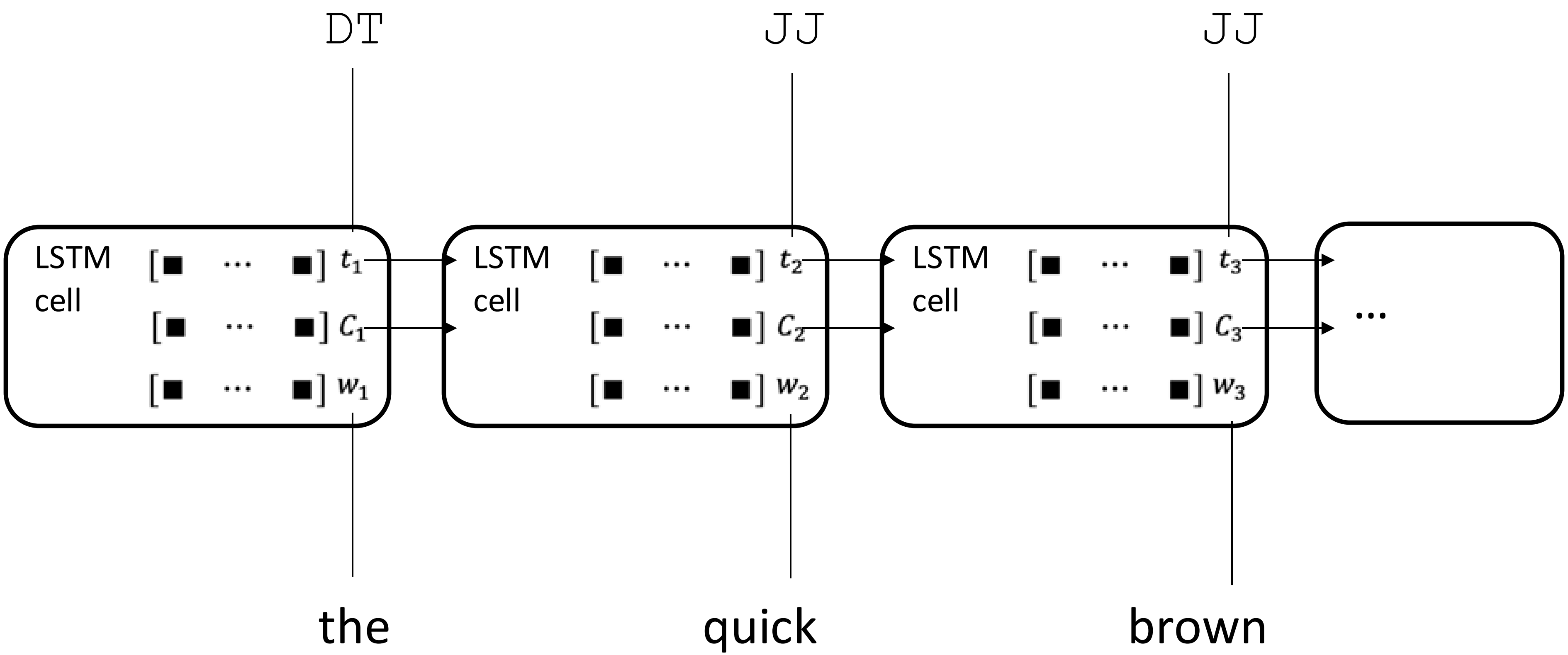
$$W \times \sigma'(z_1) \times U \times \sigma'(z_2) \times U \times \sigma'(z_3) \cdots$$

- *Vanishing gradients*: Elements of U are less than one, and gradients drop off to zero
- *Exploding gradients*: Elements of U are greater than one and gradients increase without limit

Long short-term memory (LSTM)

- A more sophisticated version of the recurrent network is the **long short term memory (LSTM)**
- Adds **memory cell** to hidden state
- Uses **gates** to determine what information feeds forward from one time step of the network to the next
  - This helps to address the vanishing/exploding gradients problems and make learning more stable
- Gates are function of current input and previous hidden state

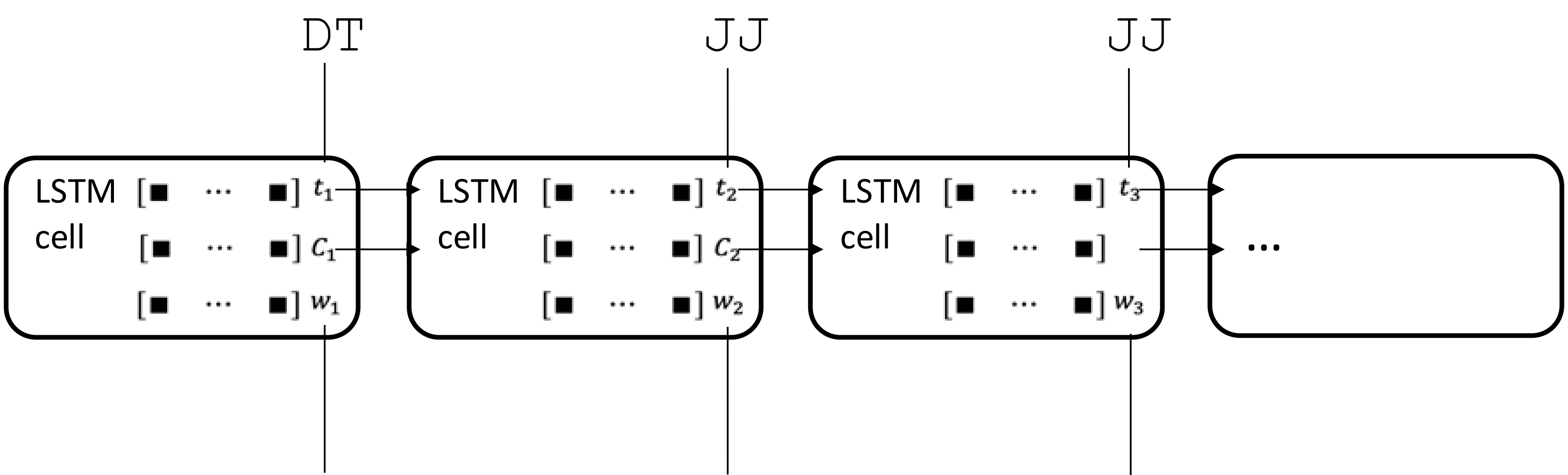
Long short-term memory (LSTM)



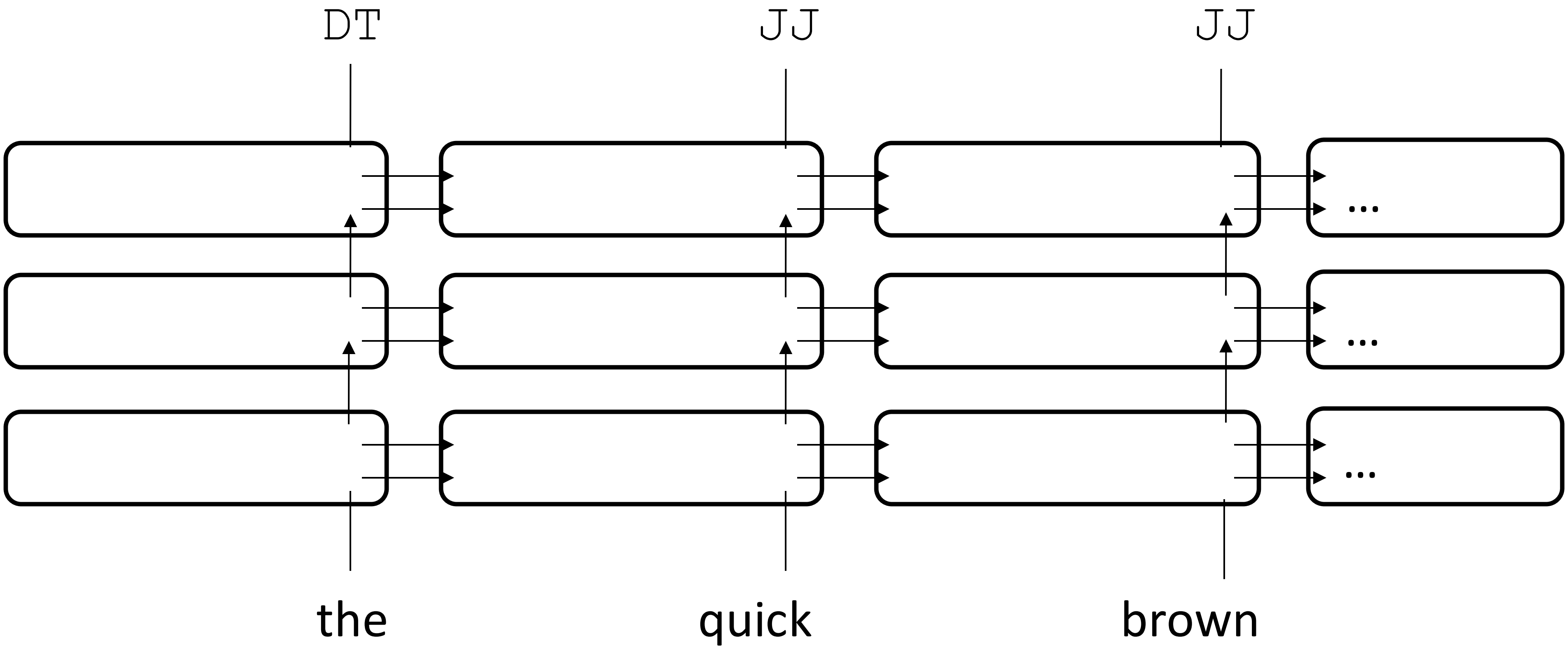
May run in either direction

Long short-term memory (LSTM)

- Softmax transformation to make categorical prediction of each tag at output layer
- Cross-entropy loss function: $\mathcal{L}_i = -\log P(t_i = t_i^*)$
- Total loss is sum of losses across labels for full text: $\mathcal{L} = \sum_i \mathcal{L}_i$

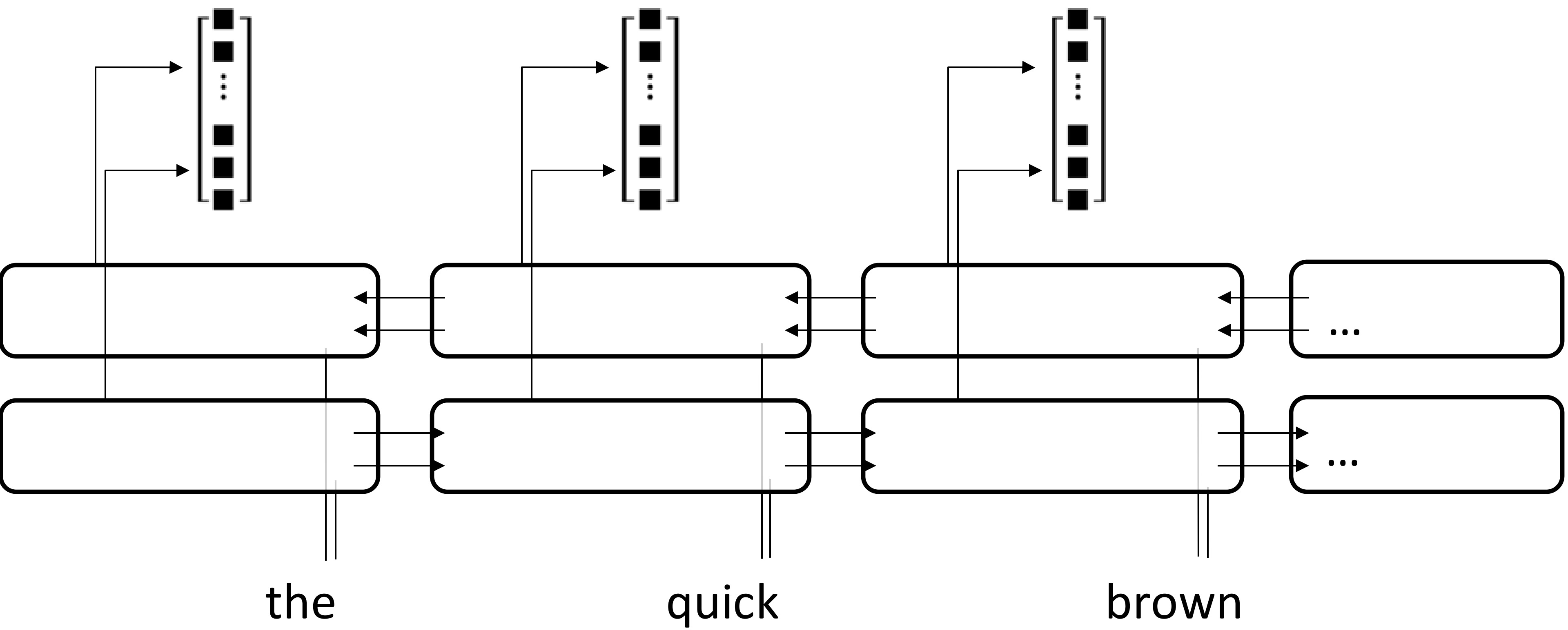


Multi-layer LSTM



Output vector from each layer is provided as input to next layer up

Bidirectional LSTM (BiLSTM)



Concatenated output from two LSTM
layers running in opposite directions

CONVNETS AND RNNS FOR TEXT CATEGORIZATION

Using sequence information for text categorization

- We noted before that some text categorization tasks (like sentiment analysis) could also benefit from using sequential information about the words in a text

I would never buy this product again. It clearly failed under high-stress testing in my home.	I would clearly buy this product again. It never failed under high-stress testing in my home.

- We can also use these CNN/RNN architectures for text classification

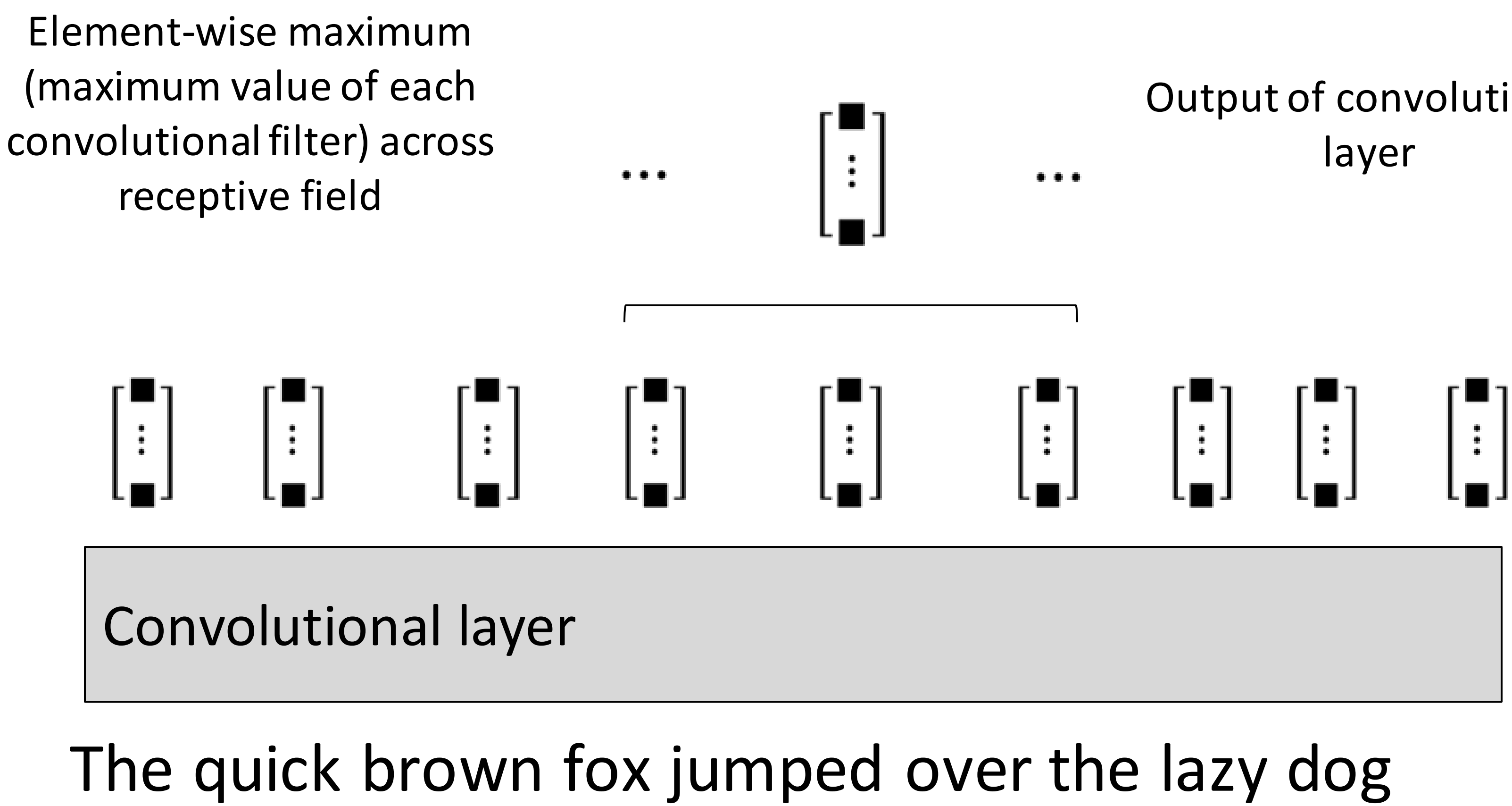
CNNs for text categorization

- In a convolutional model, we can use pooling operations to aggregate features across the entire sentence / text
- And then use this representation as an input to a standard feed-forward neural network for text categorization

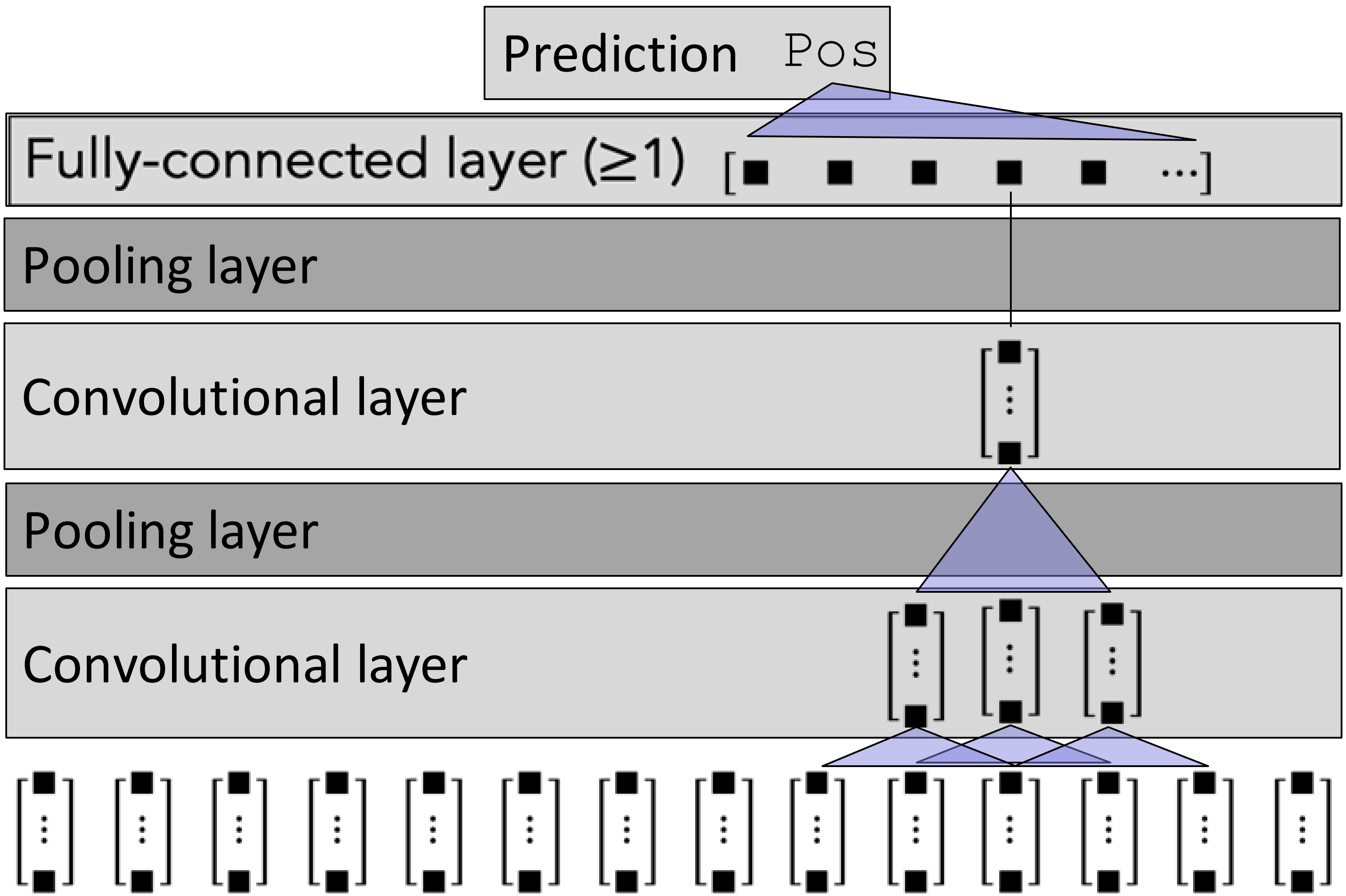
Pooling layers

- If convolutional operations are feature detectors, then **pooling layers <u>aggregate</u> the outputs of the feature detectors** to indicate whether a given feature is activated in the neighborhood of a word
- The output of the pooling layer is typically the **maximum** value (sometimes the **average**) of a convolutional filter within a given region
- Performed separately for each filter
- Can also be applied to sequential tagging

Pooling layers

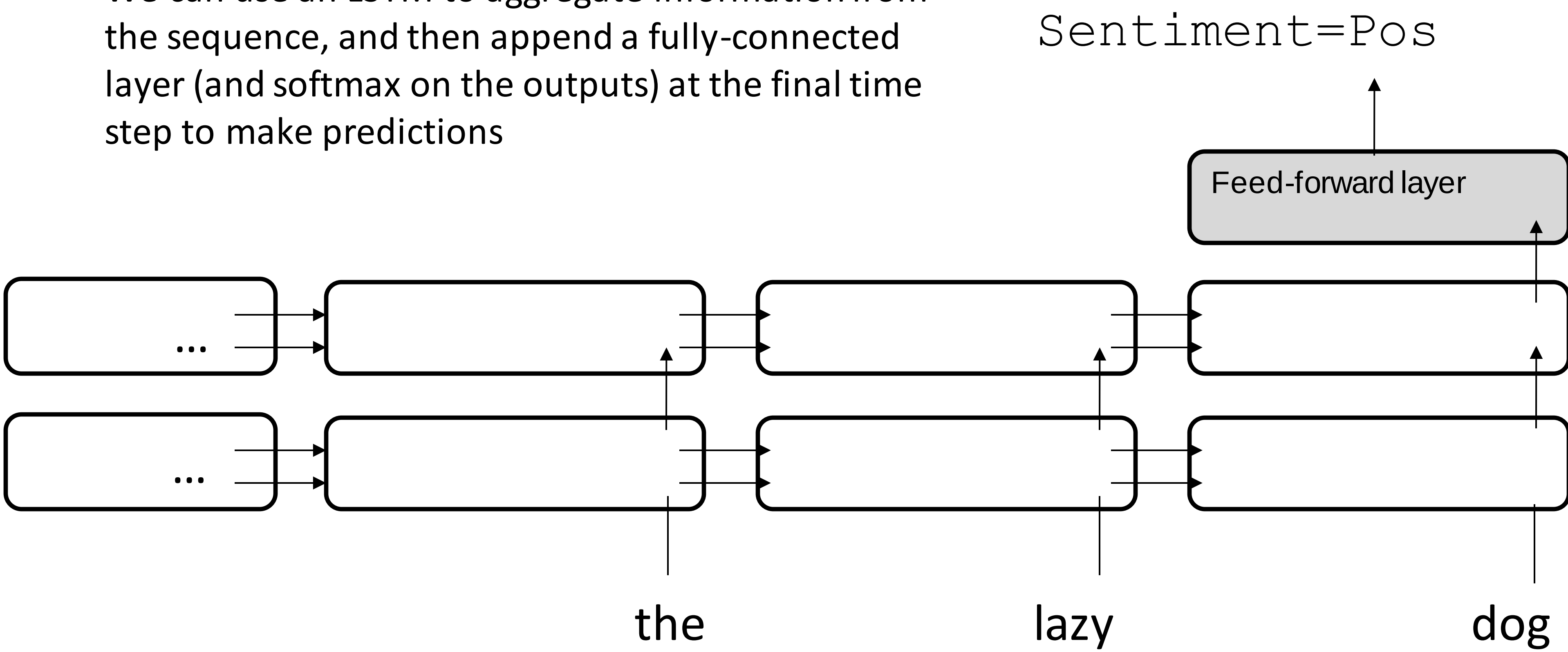


Convolutional architecture for text categorization

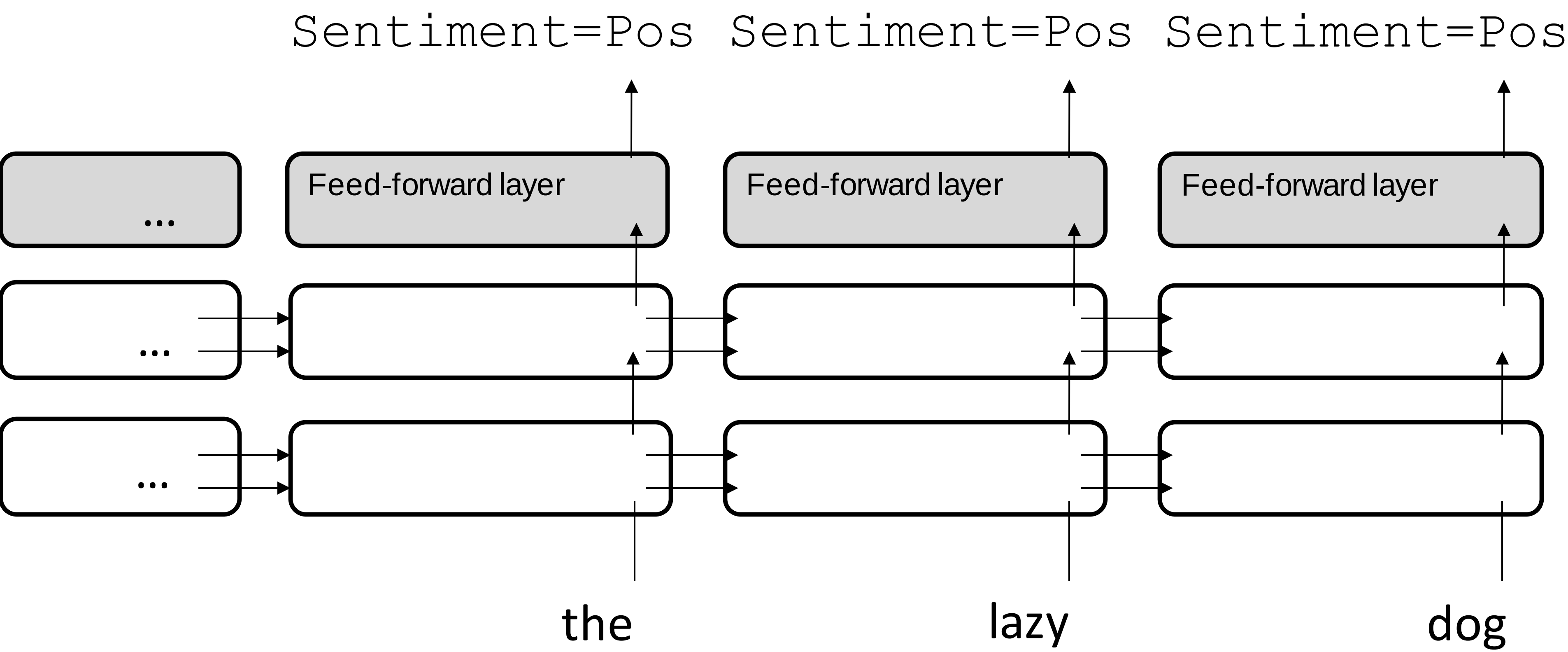


RNNs for text categorization

We can use an LSTM to aggregate information from the sequence, and then append a fully-connected layer (and softmax on the outputs) at the final time step to make predictions



RNNs for text categorization



In practice, it works better if we predict the text class at *every* time step instead of just at the final time step (*target replication*)

Target replication

- If the prediction is only made at the final time step, information has to travel a long way through the network to get to the error signal
- The solution is to make predictions (and calculate a loss on which we can backpropagate error) closer to each word – specifically, at each time step
- We define a loss function that incorporates the prediction error at each time step, giving more weight to the final prediction, e.g.:

$$\mathcal{L} = \alpha \mathcal{L}_N + \frac{(1-\alpha)}{N} \sum_{i=1}^{N} \mathcal{L}_i$$

Regularization

Similar regularization techniques of feed-forward neural networks also apply to CNNs, RNNs, and LSTMs, including:

- Early stopping
- Dropout
- L1 and L2 penalties

Textbook reading

- Relevant readings in Eisenstein-NLP textbook
  - **3.4: Convolutional neural networks** [Chapter: non-linear classification]
  - **6.3: Recurrent neural network language models**
  - **7.6: Neural sequence labeling**
- LSTMs and RNNs are both appropriate for sequence labeling <u>and</u> language modeling (both covered in Chapters 6 and 7)

Word embeddings – looking ahead

- Word2vec – Embedding is static, does not vary by context or word sense
- **Contextualized** word embeddings
  - Conditioned on context
  - Vector of each word/token is function of the entire input sentence