ILLINOIS INSTITUTE OF TECHNOLOGY

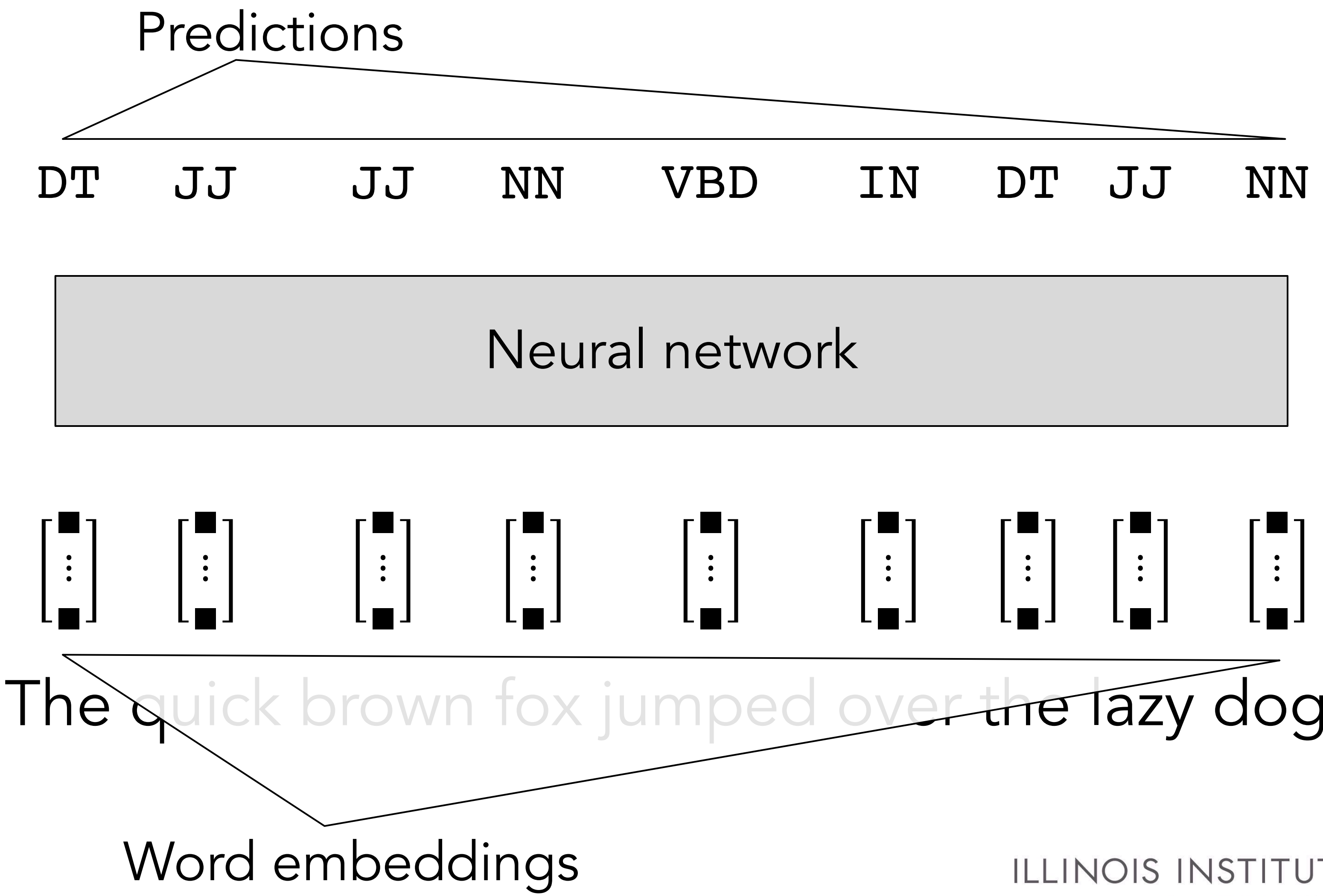
Neural models for sequence labeling

CS-585

Natural Language Processing

Derrick Higgins

Neural networks for sequence labeling

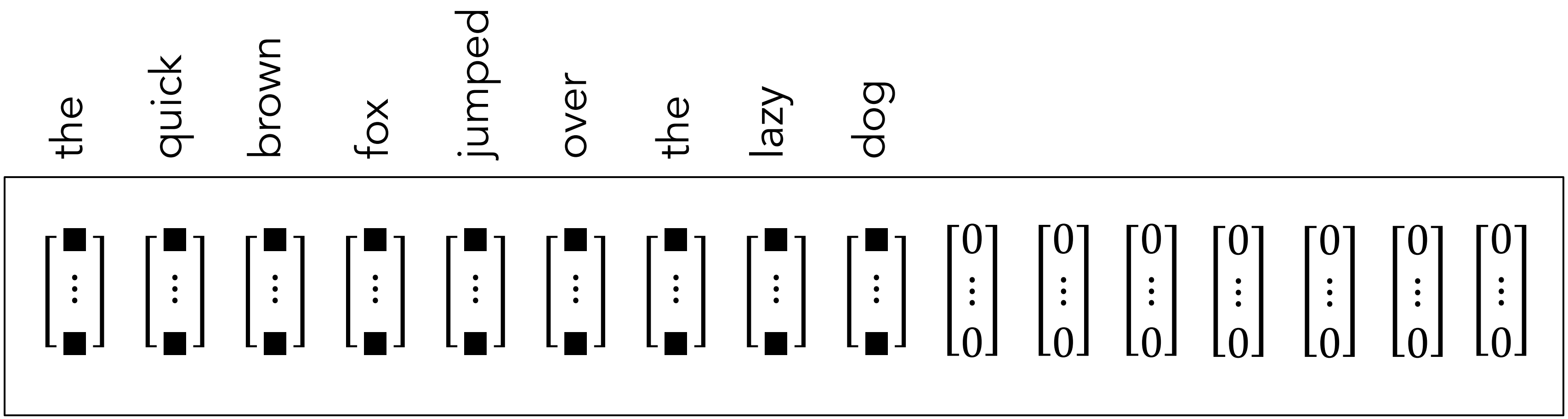


NNs for sequence labeling: preprocessing

- In order to process data efficiently for neural networks, we need to bundle the representations of multiple texts into a minibatch -- a single matrix
  - One row for each text (B)
  - One column for each embedding element of each word (D elements x N words)
  - (Or a BxDxN tensor)
- Problem: N is not a constant – some texts are longer than others
  - Solution: zero-padding and truncation

Zero-padding

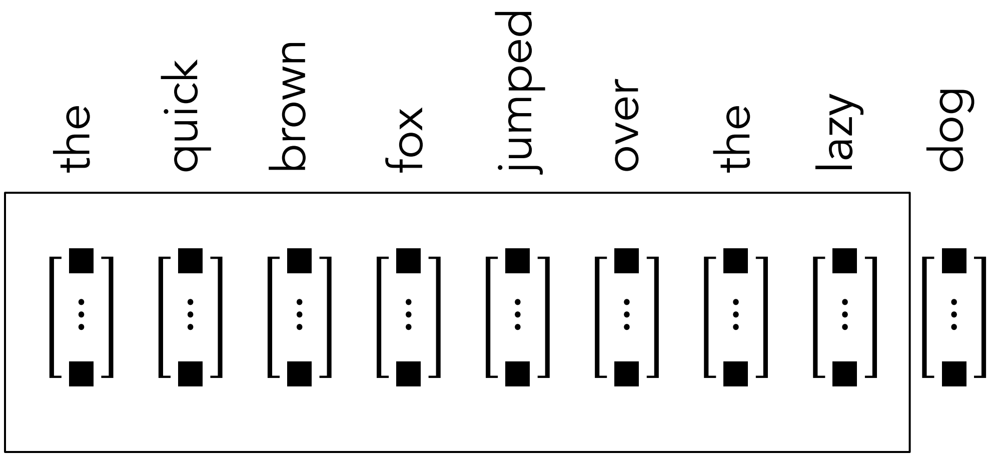
- Choose `sentence_length` for minibatches
- If a given sentence is too short, append zero vectors



`sentence_length = 16`

Truncation

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



`sentence_length = 8`

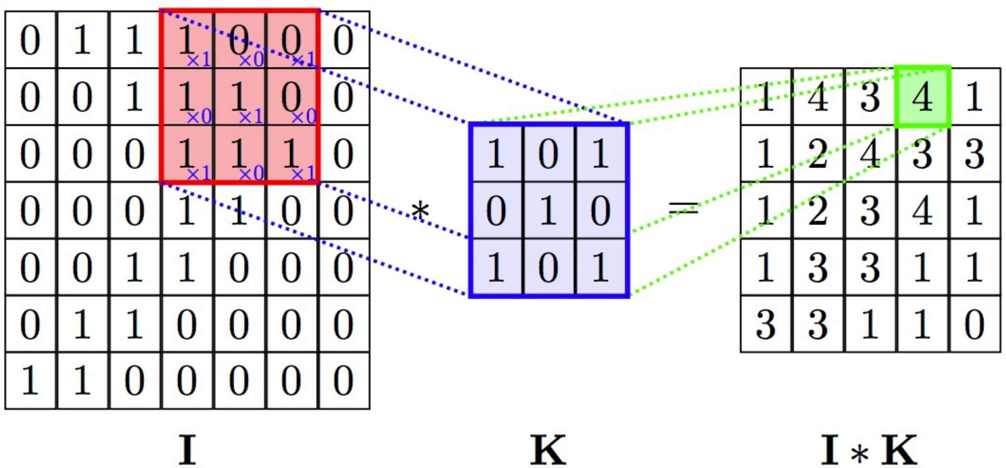
May be handled differently at training vs. prediction time

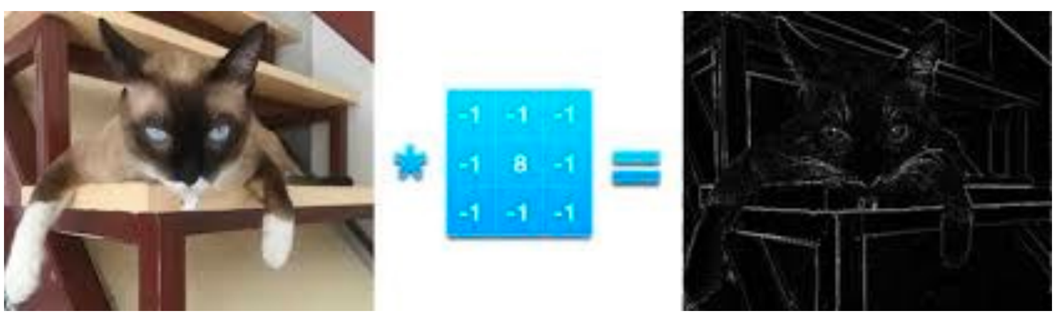
CONVOLUTIONAL NETWORKS FOR TEXT

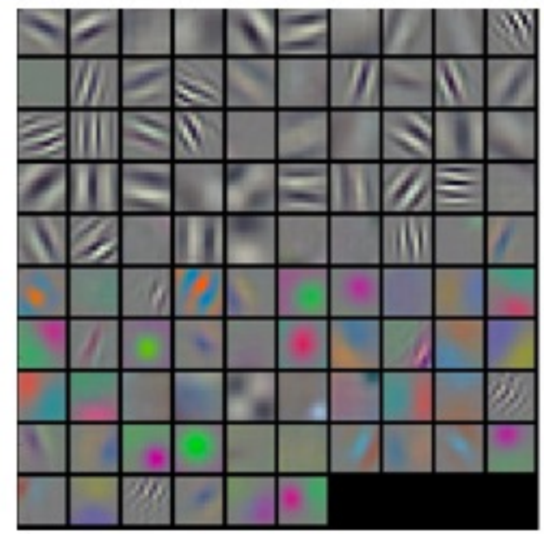
Convolutional networks

- 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







Visualizations of filters

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

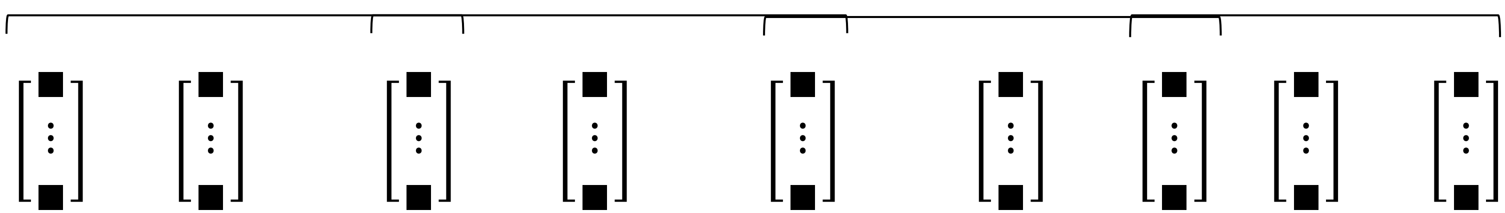
https://adeshpande3.github.io

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

Convolutions in NLP

- Instead of 2d or 3d, 1d-convolutions

$\mathbf{a}^T(\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3)$ $\mathbf{a}^T(\mathbf{w}_3 + \mathbf{w}_4 + \mathbf{w}_5)$ $\mathbf{a}^T(\mathbf{w}_5 + \mathbf{w}_6 + \mathbf{w}_7)$ $\mathbf{a}^T(\mathbf{w}_7 + \mathbf{w}_8 + \mathbf{w}_9)$



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

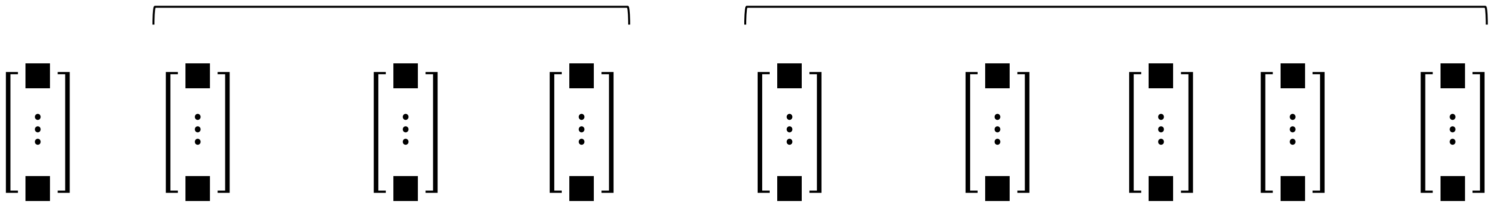
- **Width**: the size of the receptive field around the target location

width=3

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

width=5

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

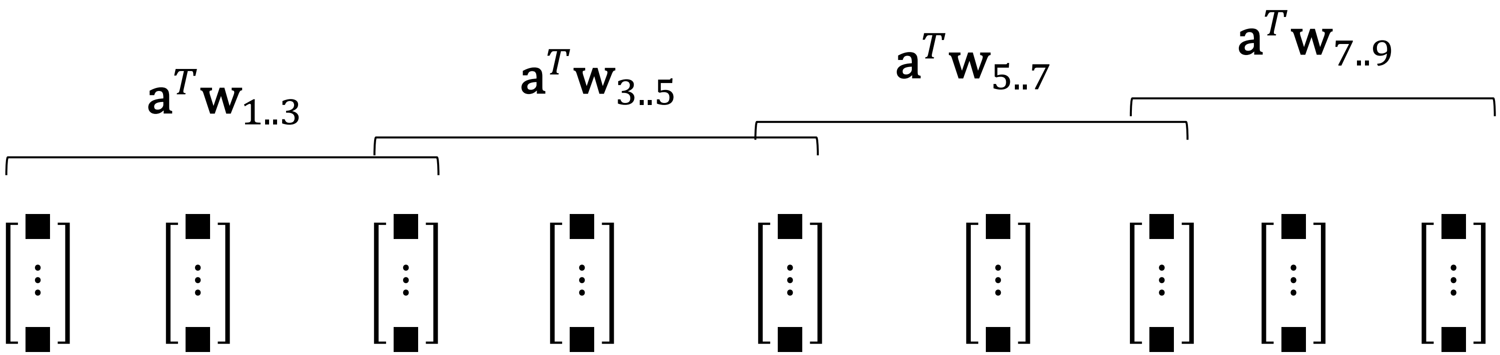


The quick brown fox jumped over the lazy dog

Convolution Parameters

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

stride=2

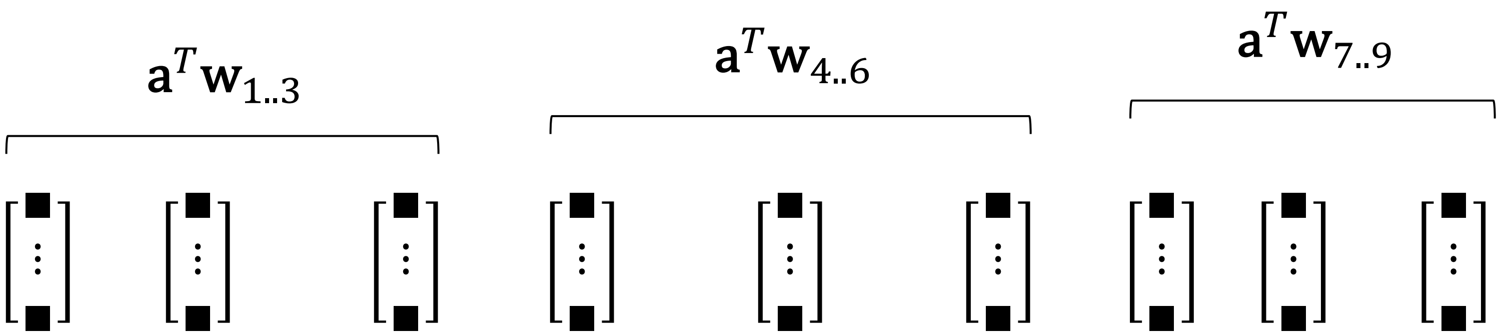


The quick brown fox jumped over the lazy dog

Convolution Parameters

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

stride=3



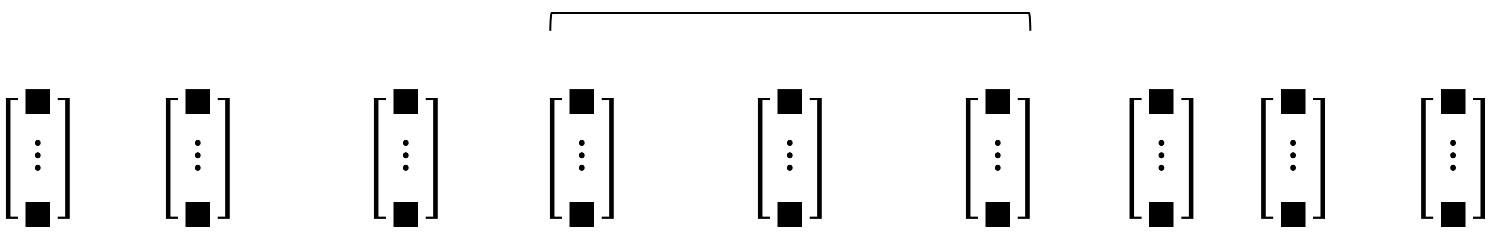
The quick brown fox jumped over the lazy dog

Convolution Parameters

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

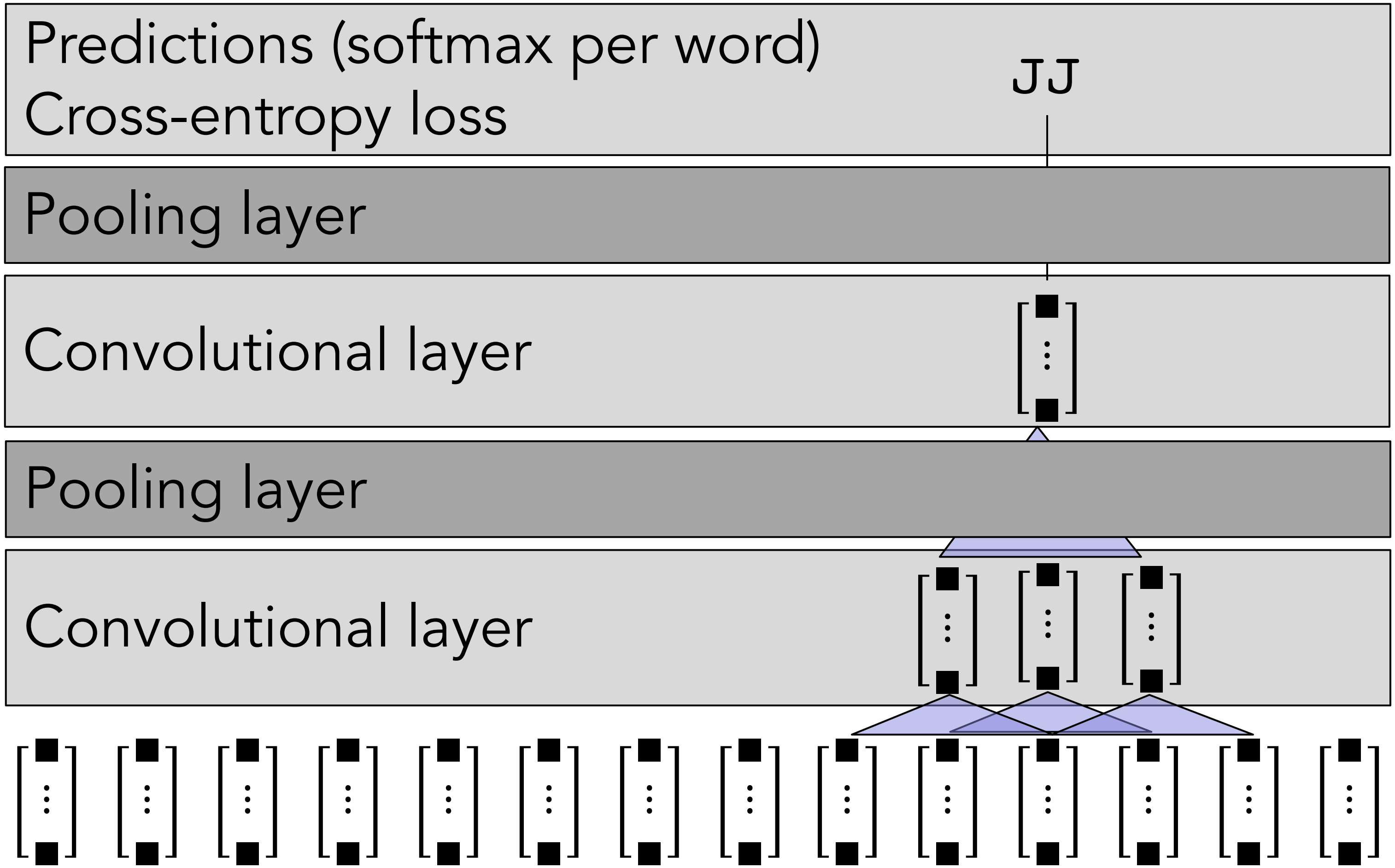
$$\begin{matrix} \mathbf{a}_3^T \mathbf{w}_{4..6} \\ \mathbf{a}_2^T \mathbf{w}_{4..6} \\ \mathbf{a}_1^T \mathbf{w}_{4..6} \end{matrix} = \begin{bmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \end{bmatrix}$$

num_filters=3



The quick brown fox jumped over the lazy dog

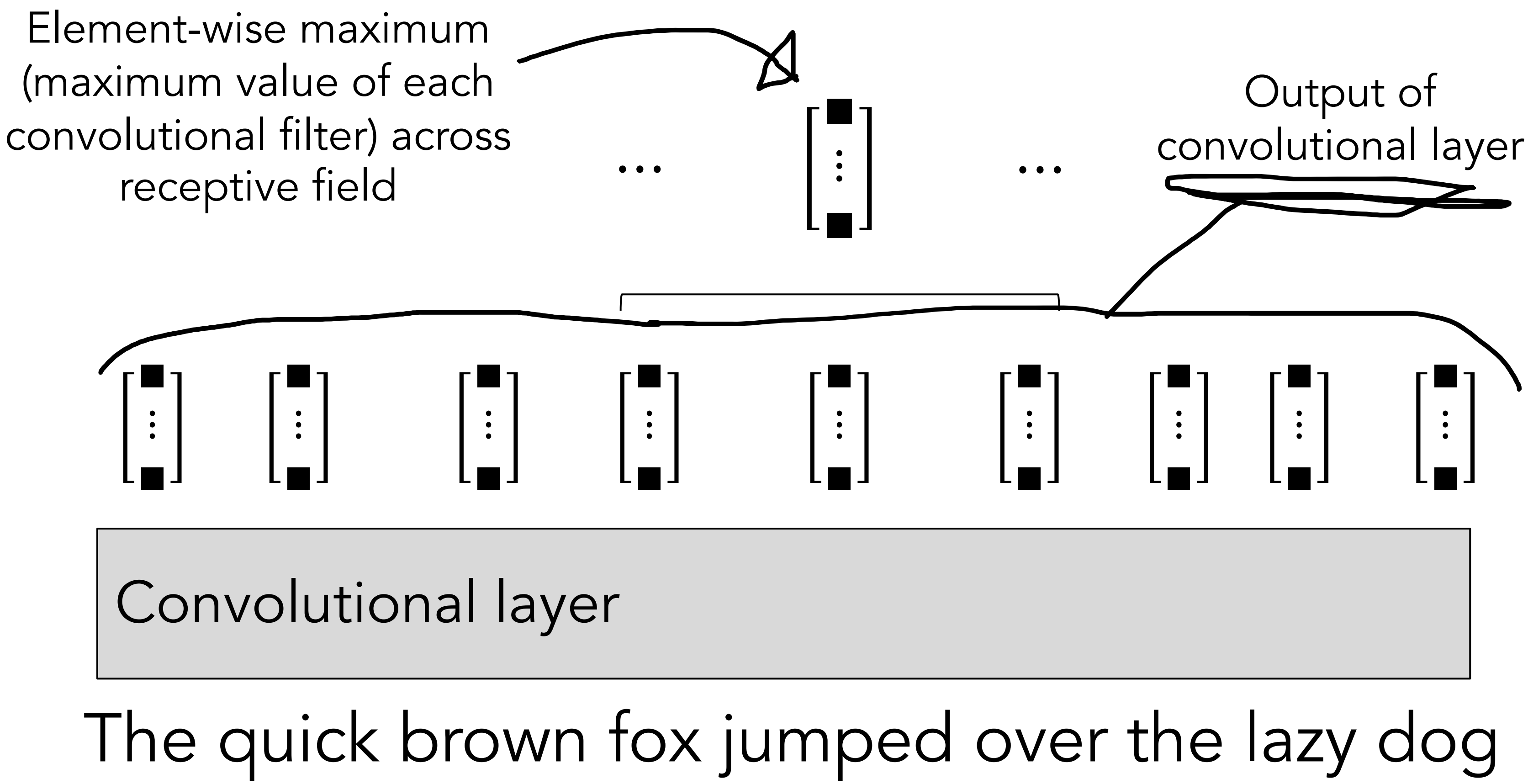
Convolutional architecture for sequence modeling



Pooling layers

- If convolutional operations are feature detectors, then pooling layers aggregate 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 mean) of a convolutional filter within a given region
- Performed separately for each filter

Pooling layers



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.

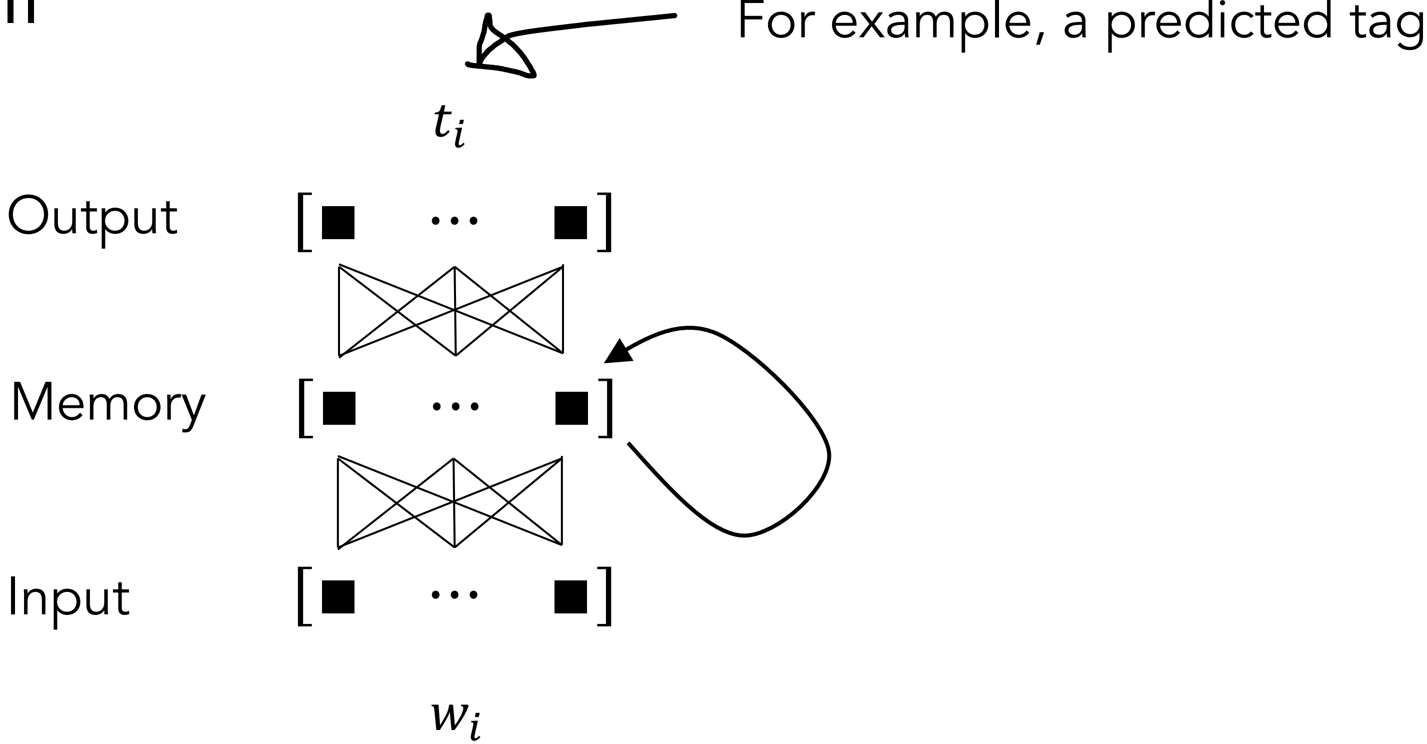
RECURRENT NETWORKS FOR TEXT

Recurrent networks

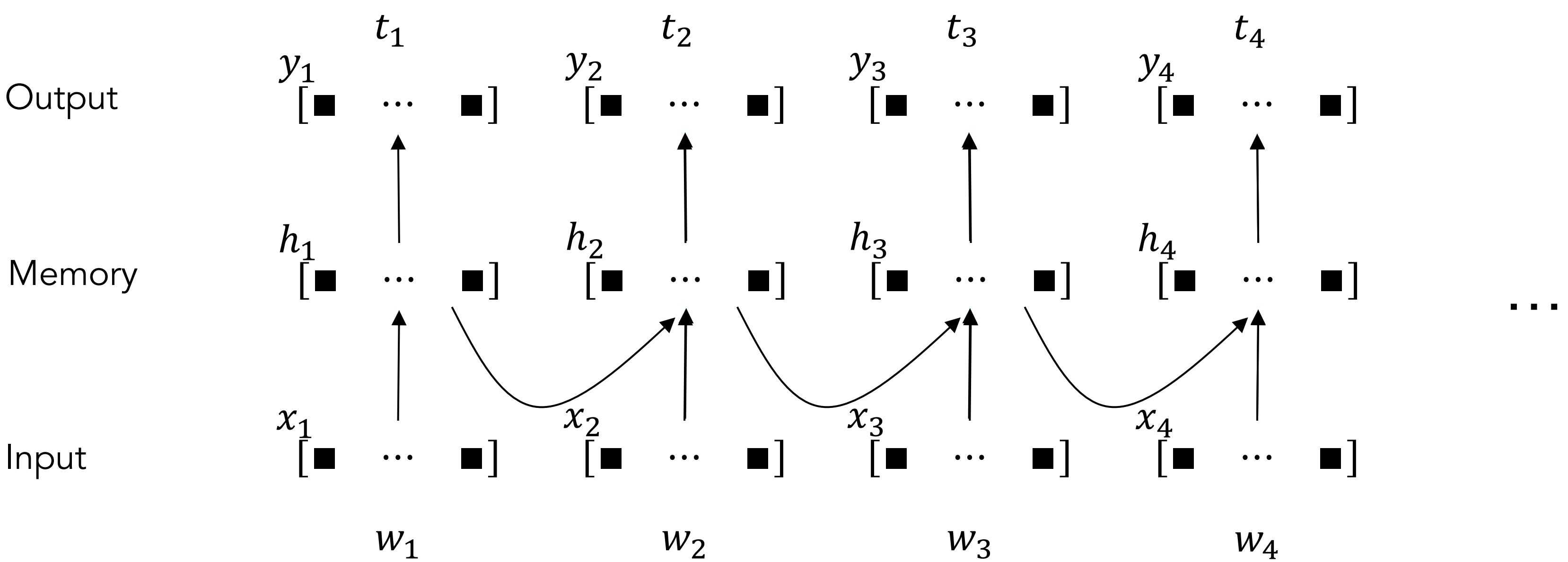
- Recurrent neural networks apply the **same operation to the input** at each time step, producing an **output**, but also updating an **internal memory state** that encodes relevant history to be used in prediction
- This memory state can allow distant information to influence the prediction made for 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)

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

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)

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

Memory state

Output weight matrix

Bias vector

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

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)
- An LSTM 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

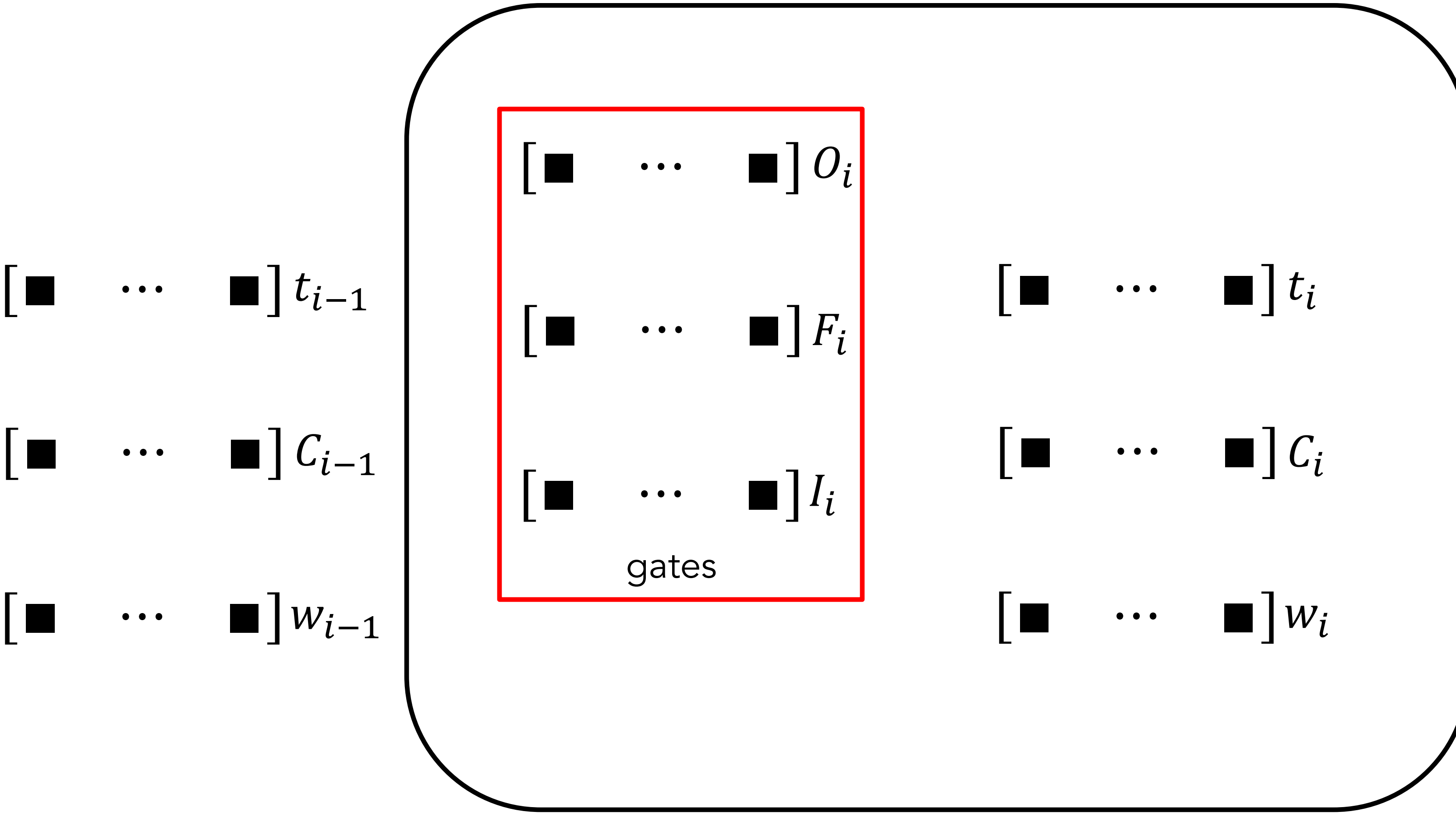
LSTM Cell

$[\blacksquare \quad \cdots \quad \blacksquare]\, t_{i-1}$ Output $[\blacksquare \quad \cdots \quad \blacksquare]\, t_i$

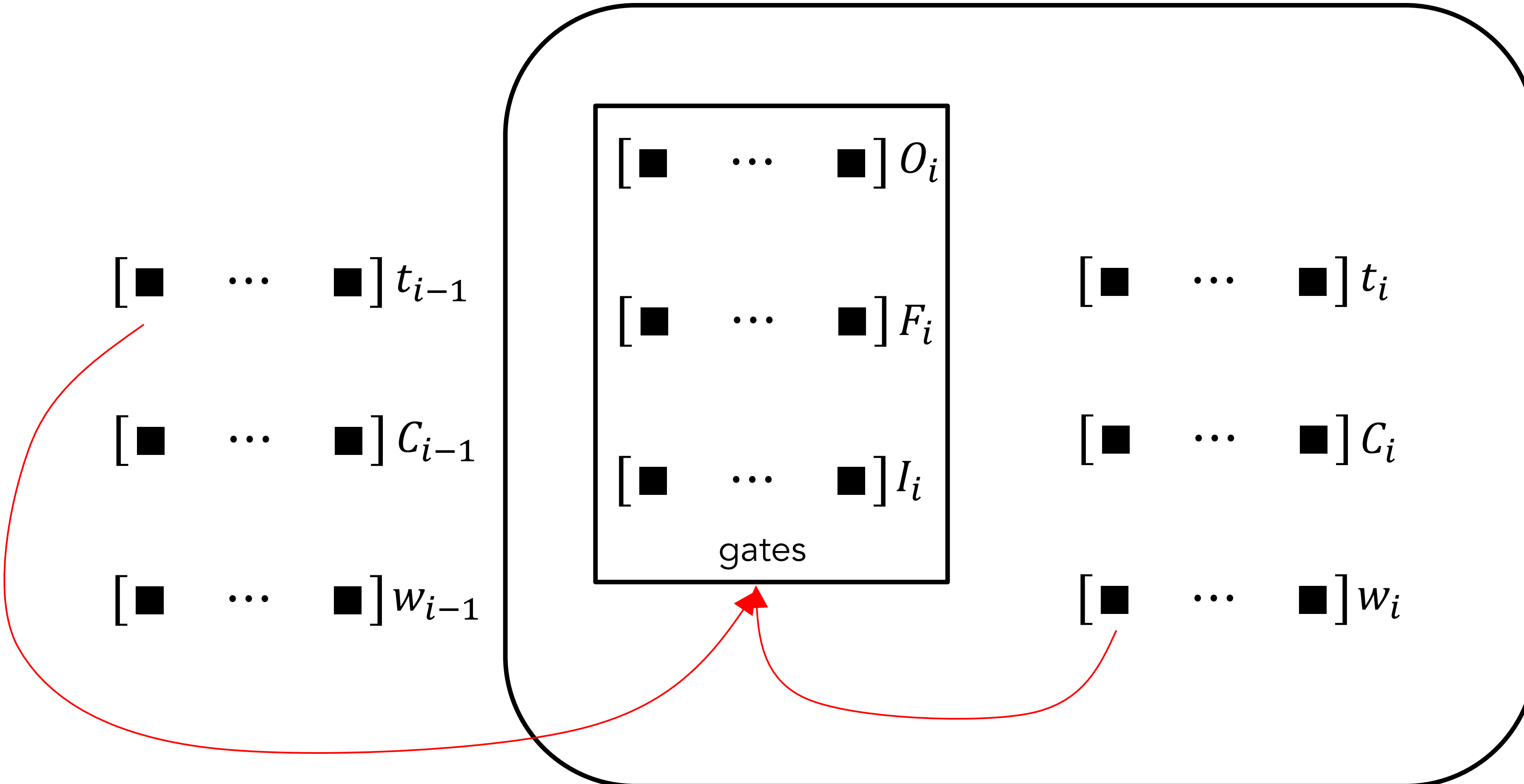
$[\blacksquare \quad \cdots \quad \blacksquare]\, C_{i-1}$ Cell state $[\blacksquare \quad \cdots \quad \blacksquare]\, C_i$

$[\blacksquare \quad \cdots \quad \blacksquare]\, w_{i-1}$ Input $[\blacksquare \quad \cdots \quad \blacksquare]\, w_i$

LSTM Cell

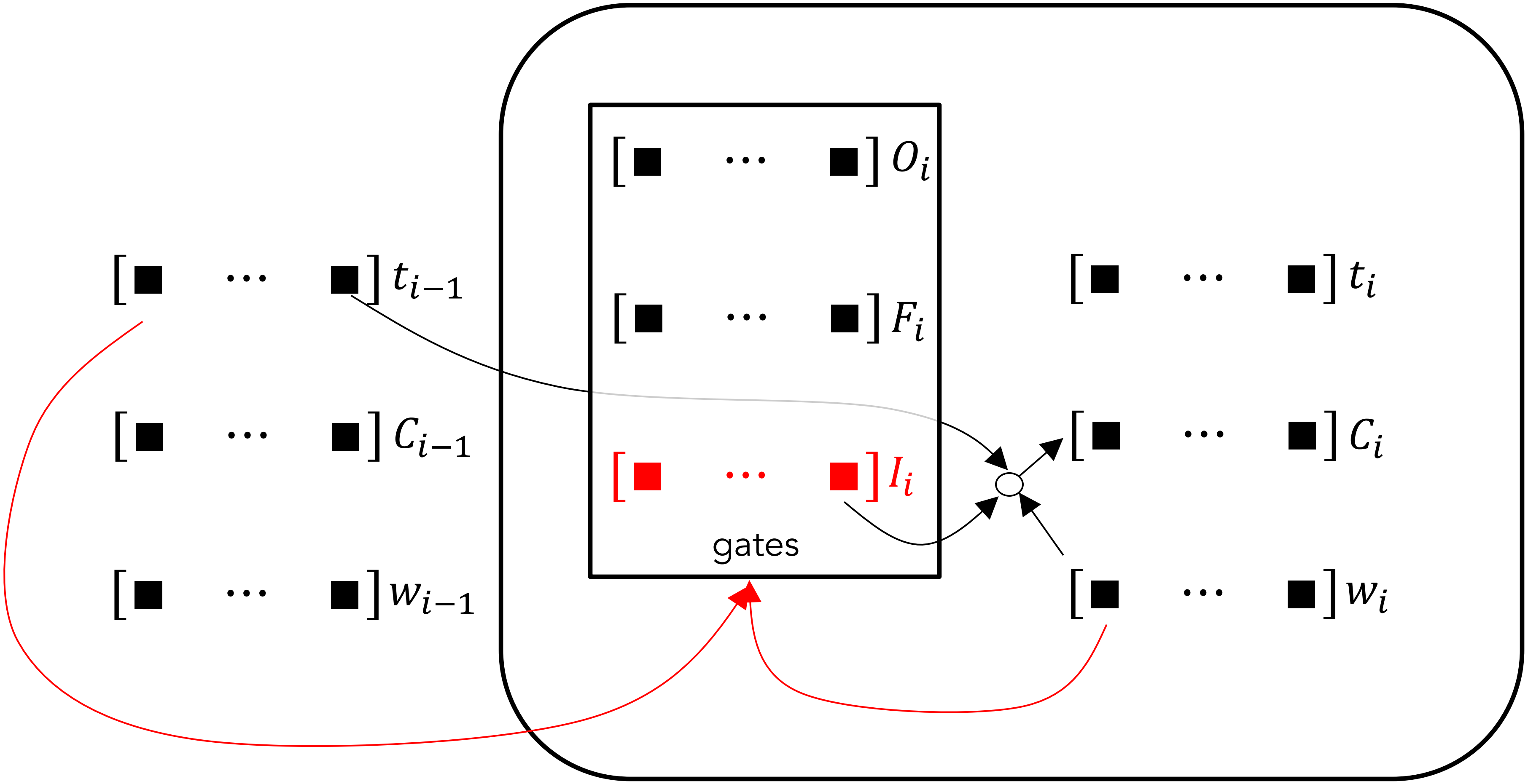


LSTM Cell



Input, output, and forget gates determine how activation/information flows through the network

LSTM Cell



Input gate determines how much of input is incorporated into cell state

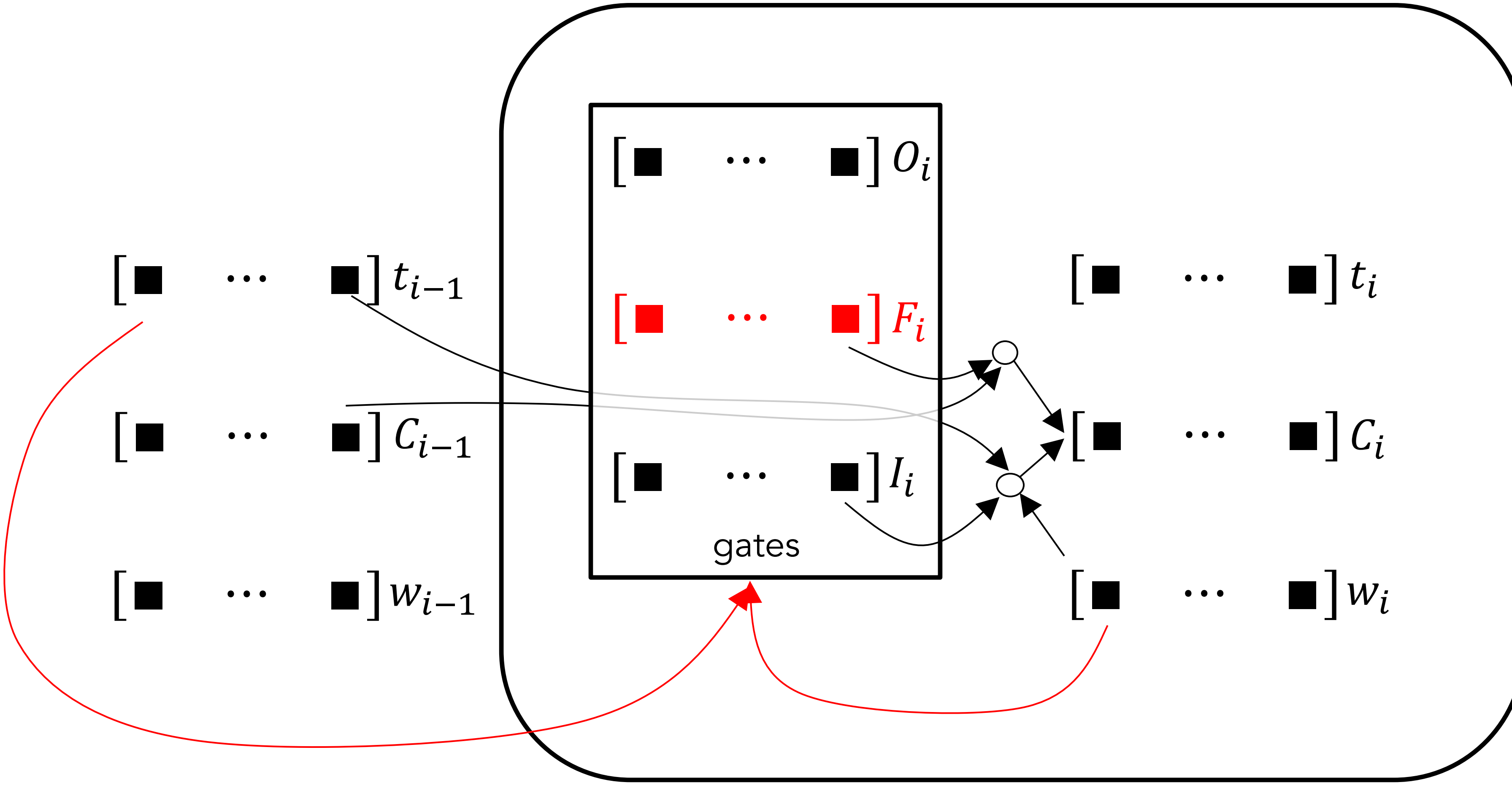
Input gate

$$I_i = \sigma(W_{It}t_{i-1} + W_{Iw}w_i + b_I)$$

Logistic sigmoid to force values into [0,1]

Affine transformation of previous output vector and current input vector

LSTM Cell



Forget gate determines how much of previous cell state is incorporated into current cell state

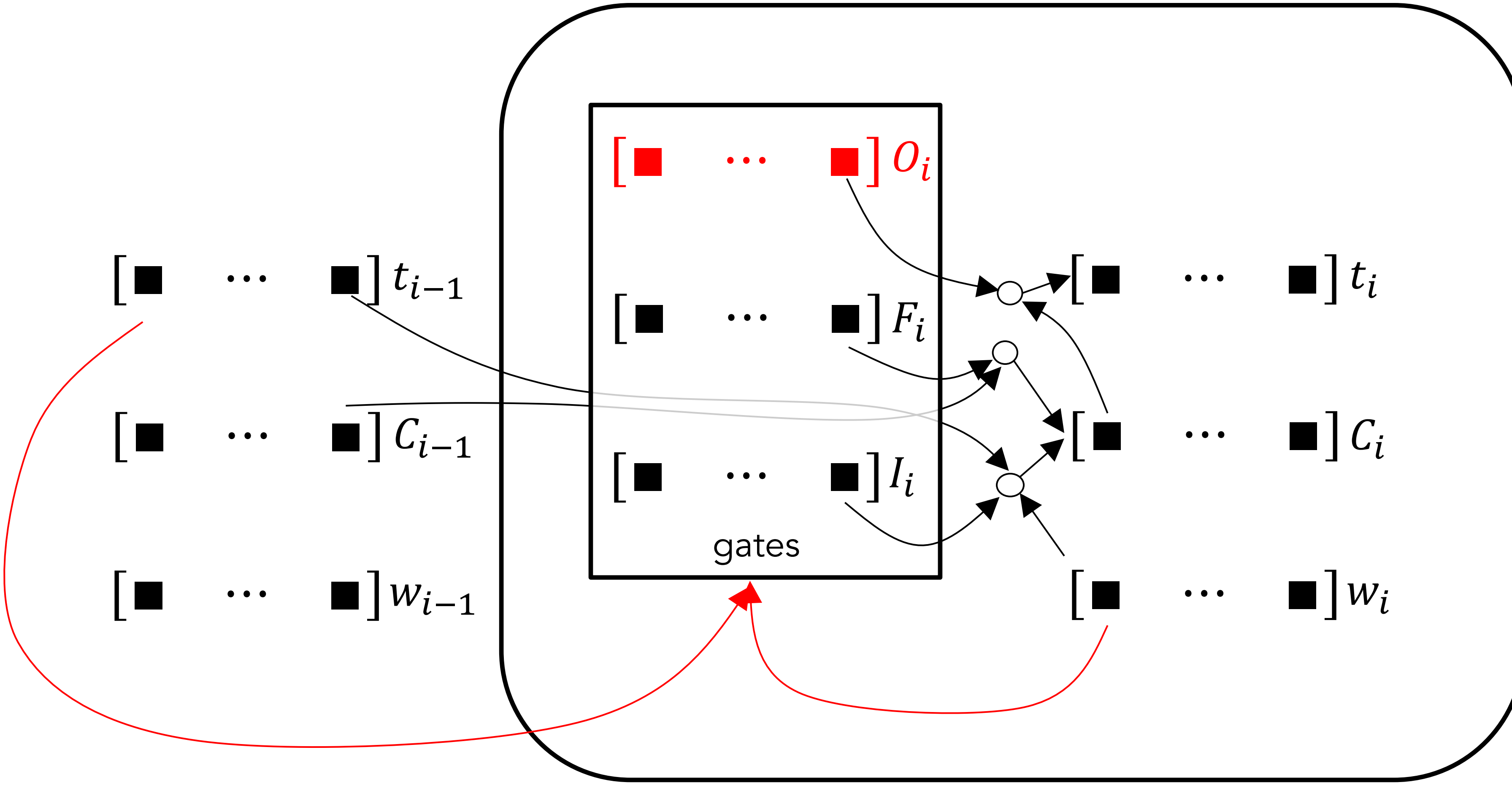
Forget gate

$$F_i = \sigma(\underbrace{W_{Ft} t_{i-1} + W_{Fw} w_i + b_F})$$

Logistic sigmoid to force values into [0,1]

Affine transformation of previous output vector and current input vector

LSTM Cell



Output gate determines how much of current cell state is incorporated into current output

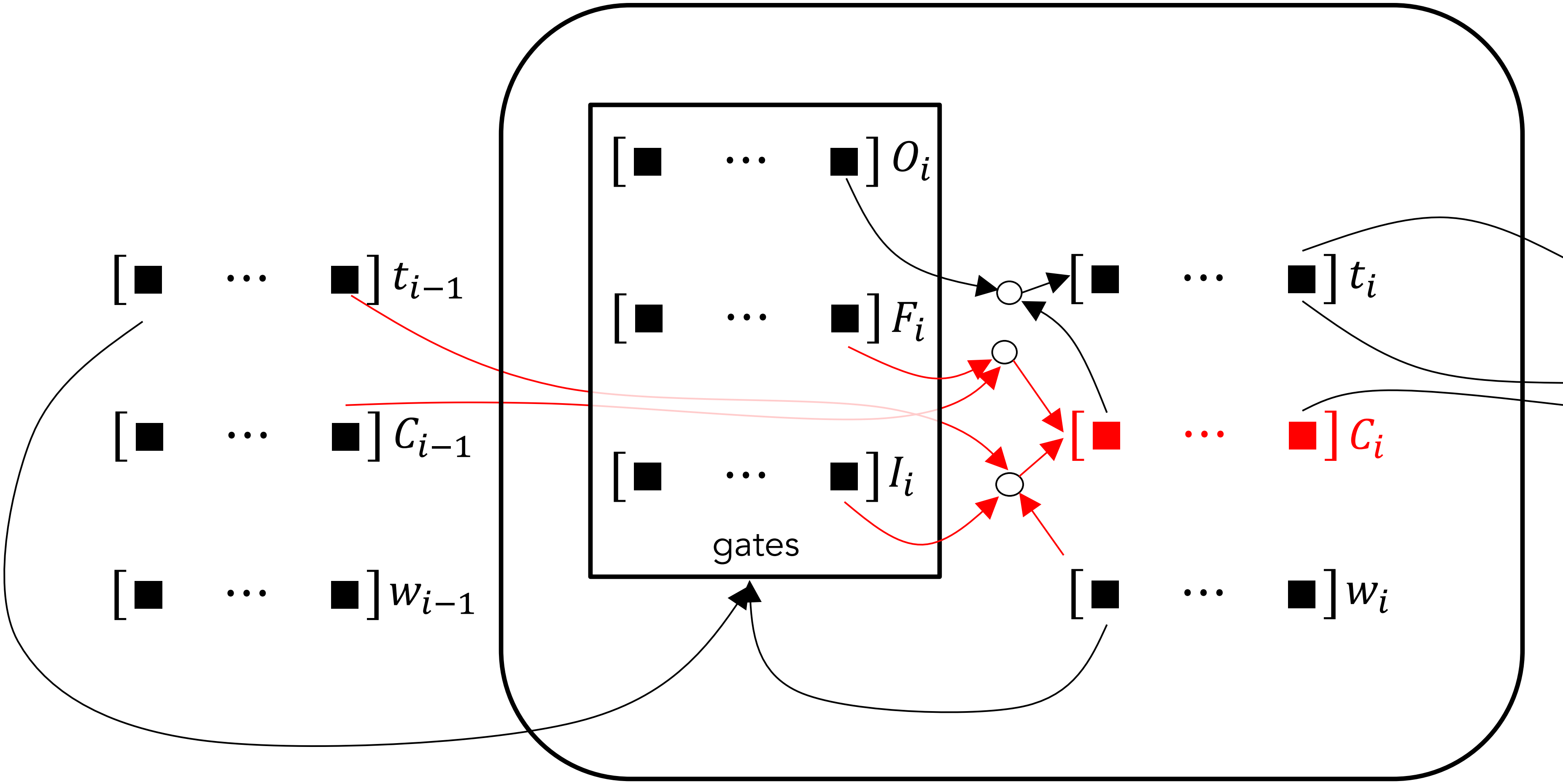
Output gate

$$O_i = \sigma(\underline{W_{Ot}t_{i-1} + W_{Ow}w_i + b_O})$$

Logistic sigmoid to force values into [0,1]

Affine transformation of previous output vector and current input vector

LSTM Cell



$[\blacksquare \cdots \blacksquare]\, O_i$
$[\blacksquare \cdots \blacksquare]\, F_i$
$[\blacksquare \cdots \blacksquare]\, I_i$
gates
$[\blacksquare \cdots \blacksquare]\, t_{i-1}$
$[\blacksquare \cdots \blacksquare]\, C_{i-1}$
$[\blacksquare \cdots \blacksquare]\, w_{i-1}$
$[\blacksquare \cdots \blacksquare]\, t_i$
$[\blacksquare \cdots \blacksquare]\, C_i$
$[\blacksquare \cdots \blacksquare]\, w_i$

Cell state

Elementwise multiplication

$$C_i = F_i \otimes C_{i-1} + I_i \otimes \tanh(W_{Ct} t_{i-1} + W_{Cw} w_i + b_C)$$

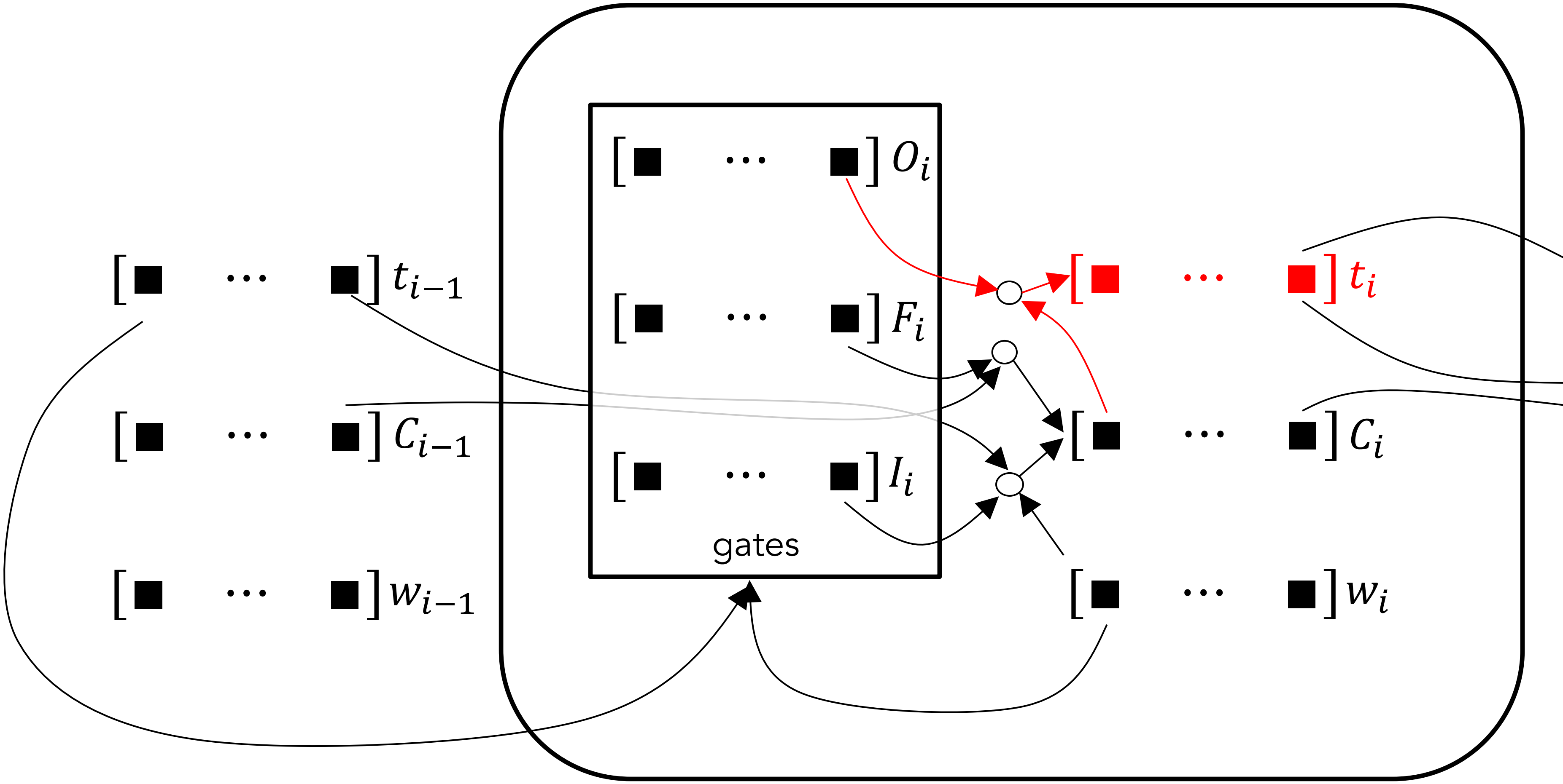
"Candidate" cell state based on current input and previous output

Apply input gate to filter result

Add in previous cell state filtered by forget gate

Nonlinearity to force values into [-1,1]

LSTM Cell



$[\blacksquare \cdots \blacksquare]\, t_{i-1}$
$[\blacksquare \cdots \blacksquare]\, C_{i-1}$
$[\blacksquare \cdots \blacksquare]\, w_{i-1}$
$[\blacksquare \cdots \blacksquare]\, O_i$
$[\blacksquare \cdots \blacksquare]\, F_i$
$[\blacksquare \cdots \blacksquare]\, I_i$
gates
$[\blacksquare \cdots \blacksquare]\, t_i$
$[\blacksquare \cdots \blacksquare]\, C_i$
$[\blacksquare \cdots \blacksquare]\, w_i$

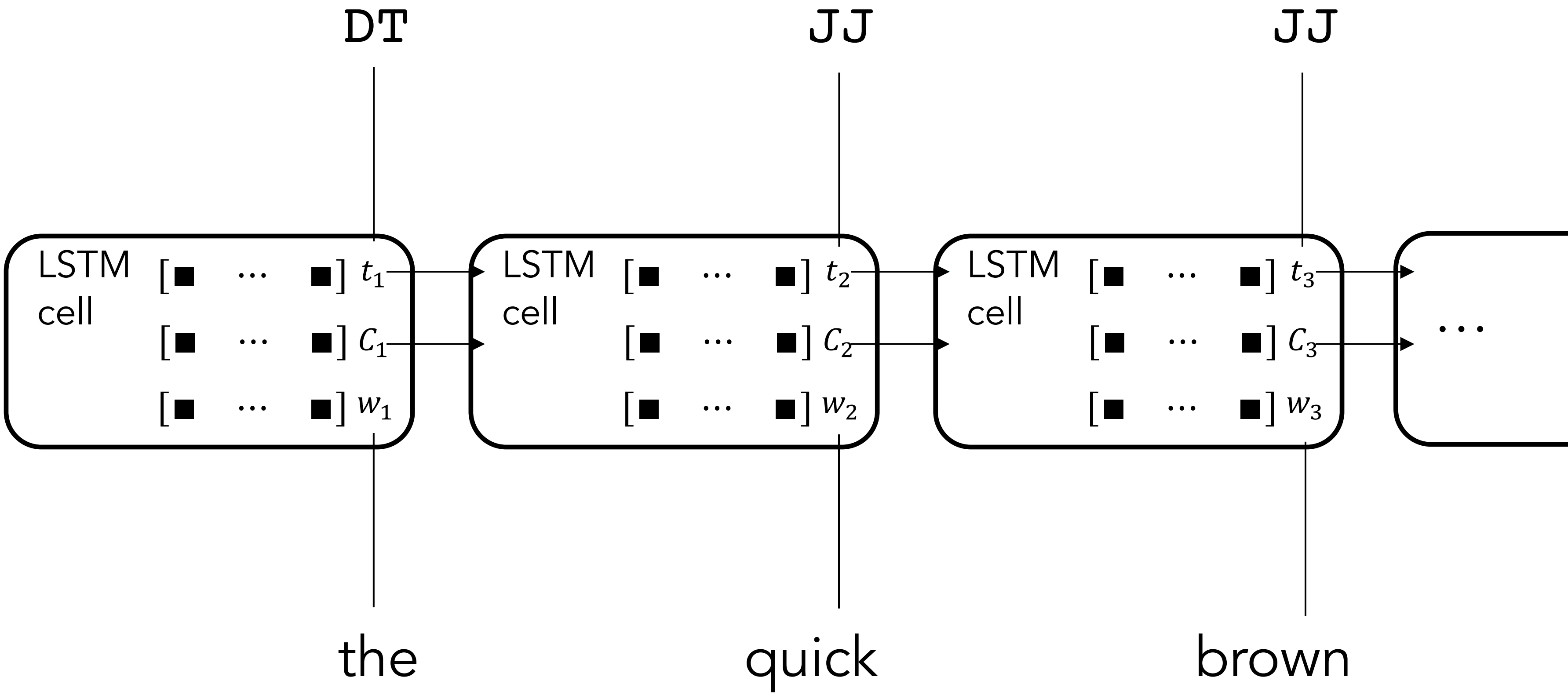
Cell output

$$t_i = O_i \otimes \sigma(C_t)$$

Output gate determines how much of cell state to put into the cell output

Nonlinearity of your choice

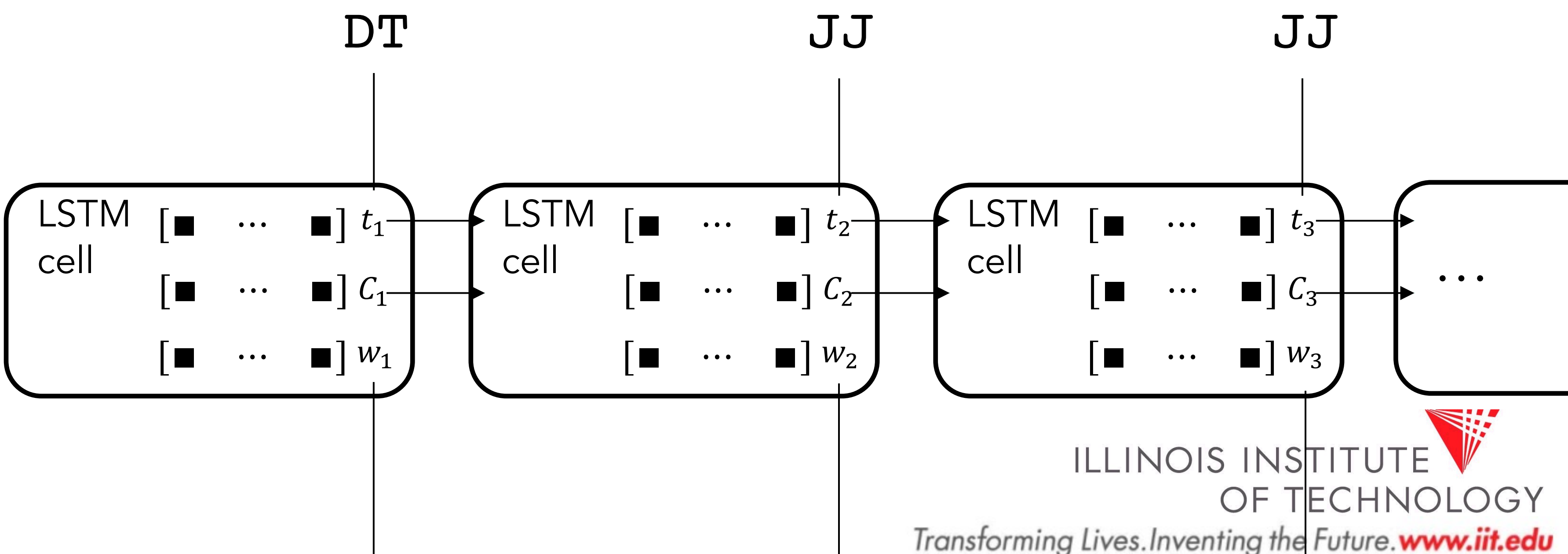
Long short-term memory (LSTM)



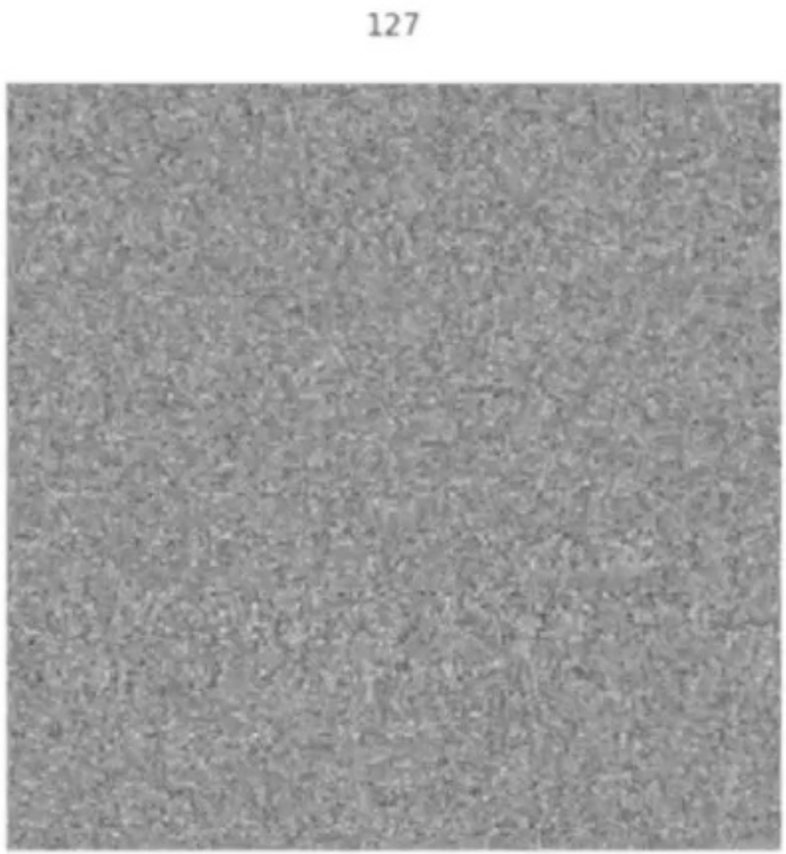
May run in either direction

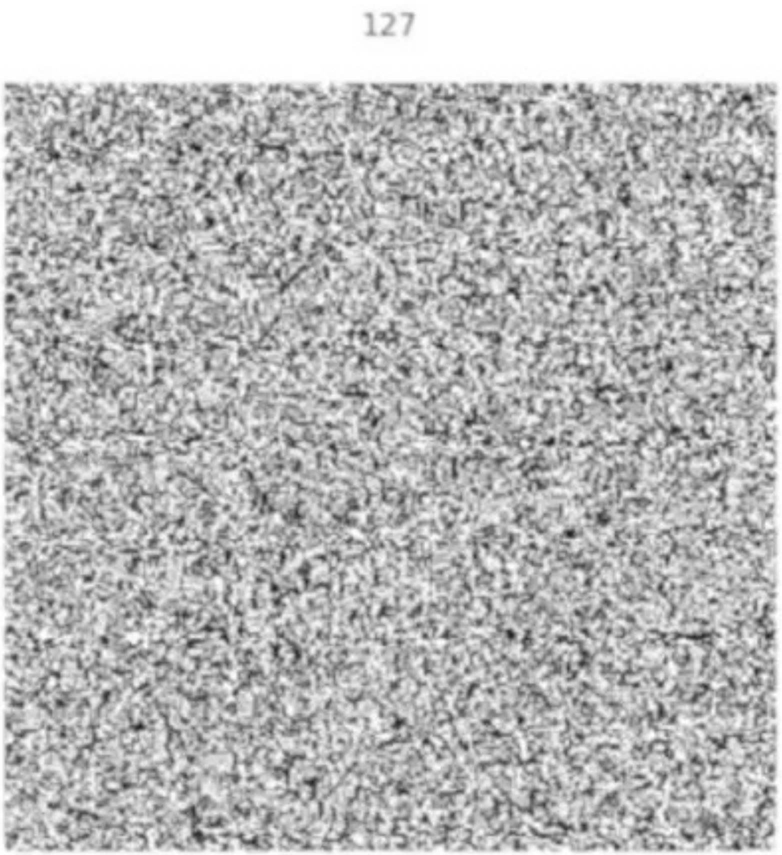
Long short-term memory (LSTM)

- 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: $\mathcal{L} = \sum_i \mathcal{L}_i$



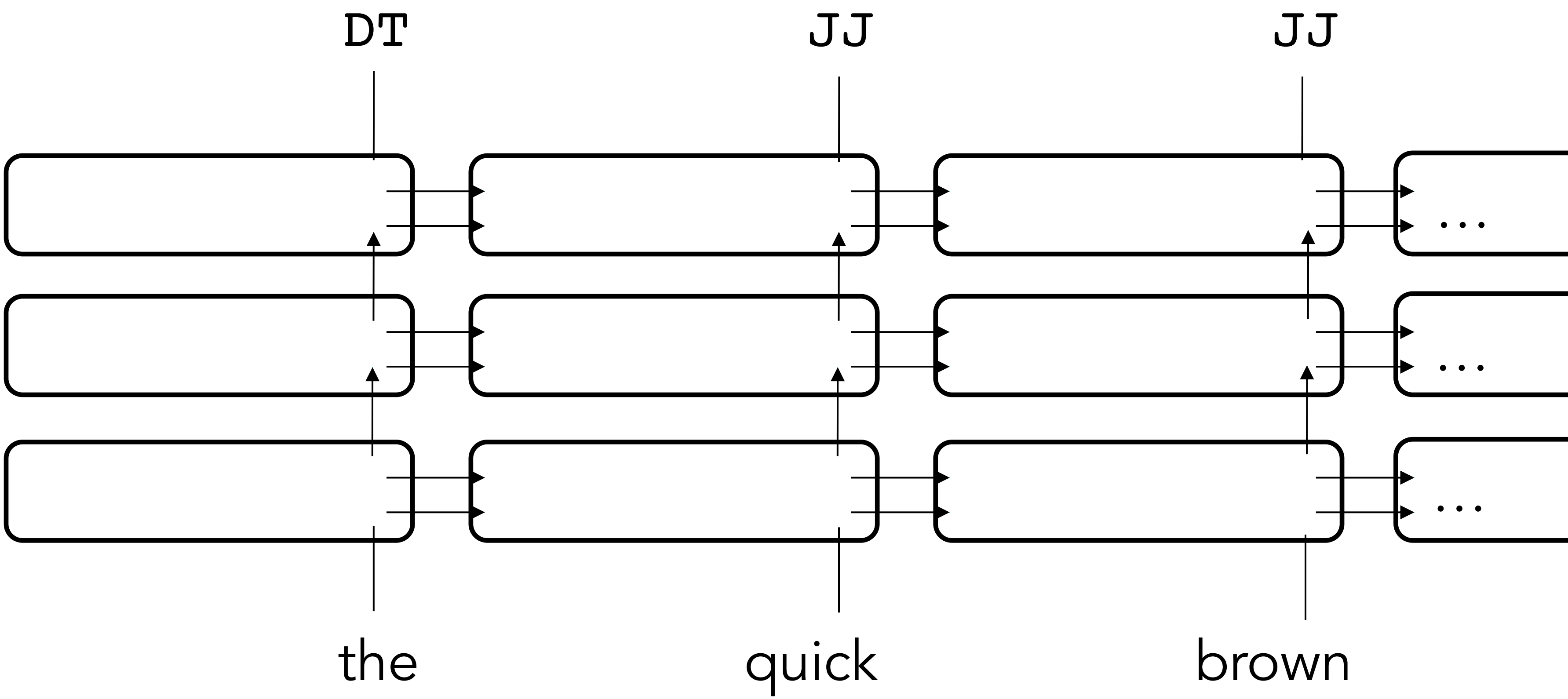
LSTMs and the vanishing gradient problem





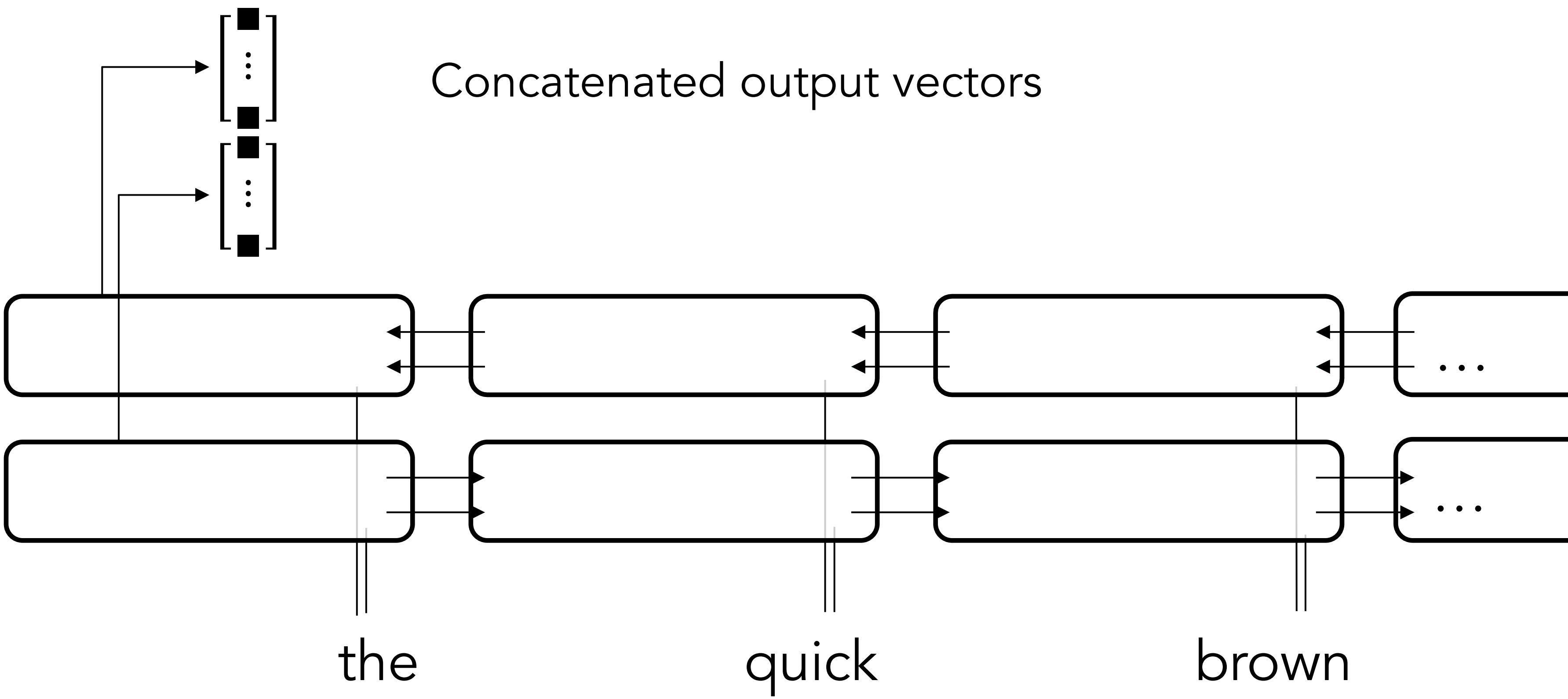
https://imgur.com/vaNahKE

Multi-layer LSTM



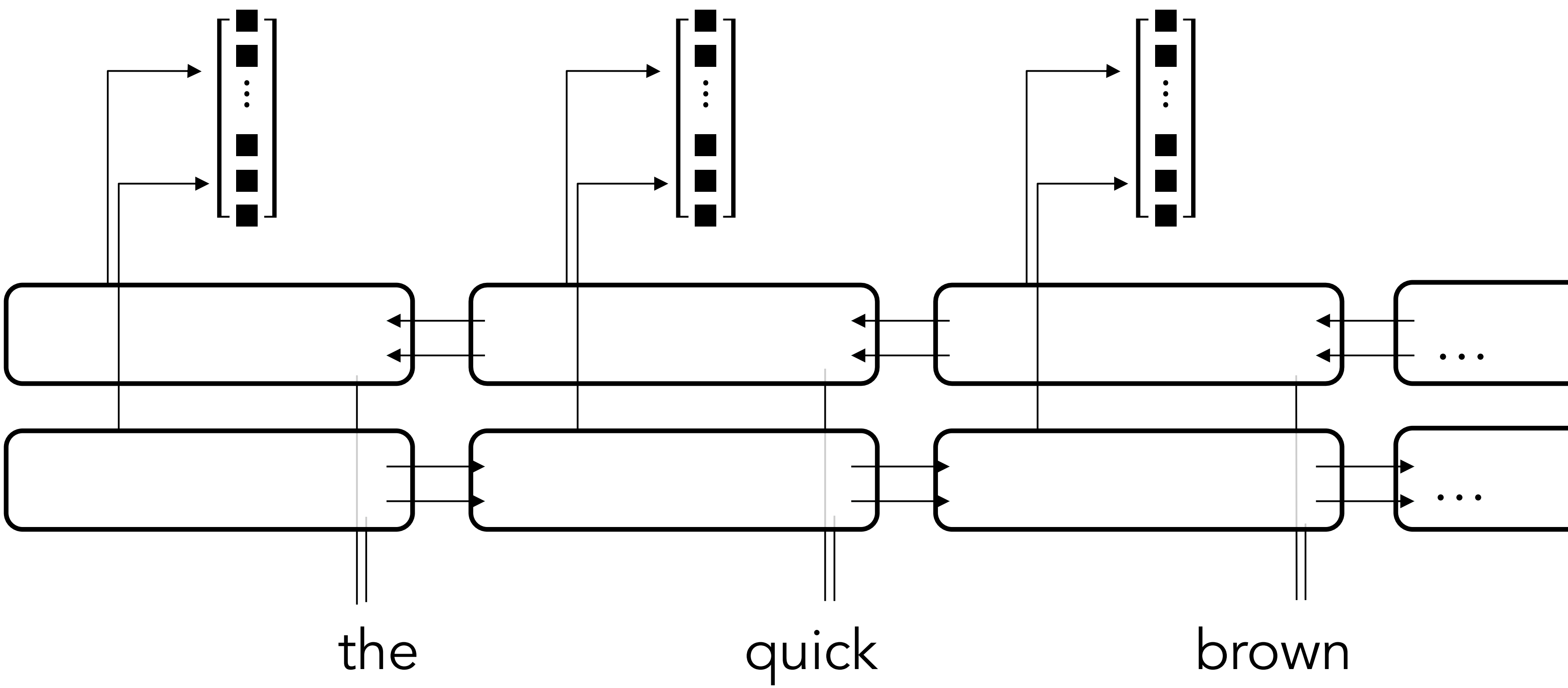
Output vector from each layer is provided as input to next layer up

Bidirectional LSTM (BiLSTM)



Concatenated output from two LSTM layers running in opposite directions

Bidirectional LSTM (BiLSTM)



Concatenated output from two LSTM layers running in opposite directions

Next Level: CRF layer on top of neural sequence model

- LSTMs and other recurrent models for sequence labeling do a very good job of flexibly incorporating evidence from a potentially unbounded history (preceding set of words and label predictions)
- But they don't always do a great job of incorporating top-down constraints on label sequences (e.g., an `I-Place` has to be preceded by a `B-Place`). Therefore, a CRF is often used as the last layer of a recurrent model
- Modules available for
  - Pytorch: https://github.com/allenai/allennlp/blob/master/allennlp/modules/conditional_random_field.py
  - Keras: https://github.com/keras-team/keras-contrib/blob/master/keras_contrib/layers/crf.py
- Can be trained end-to-end using gradient descent and similar optimizers

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

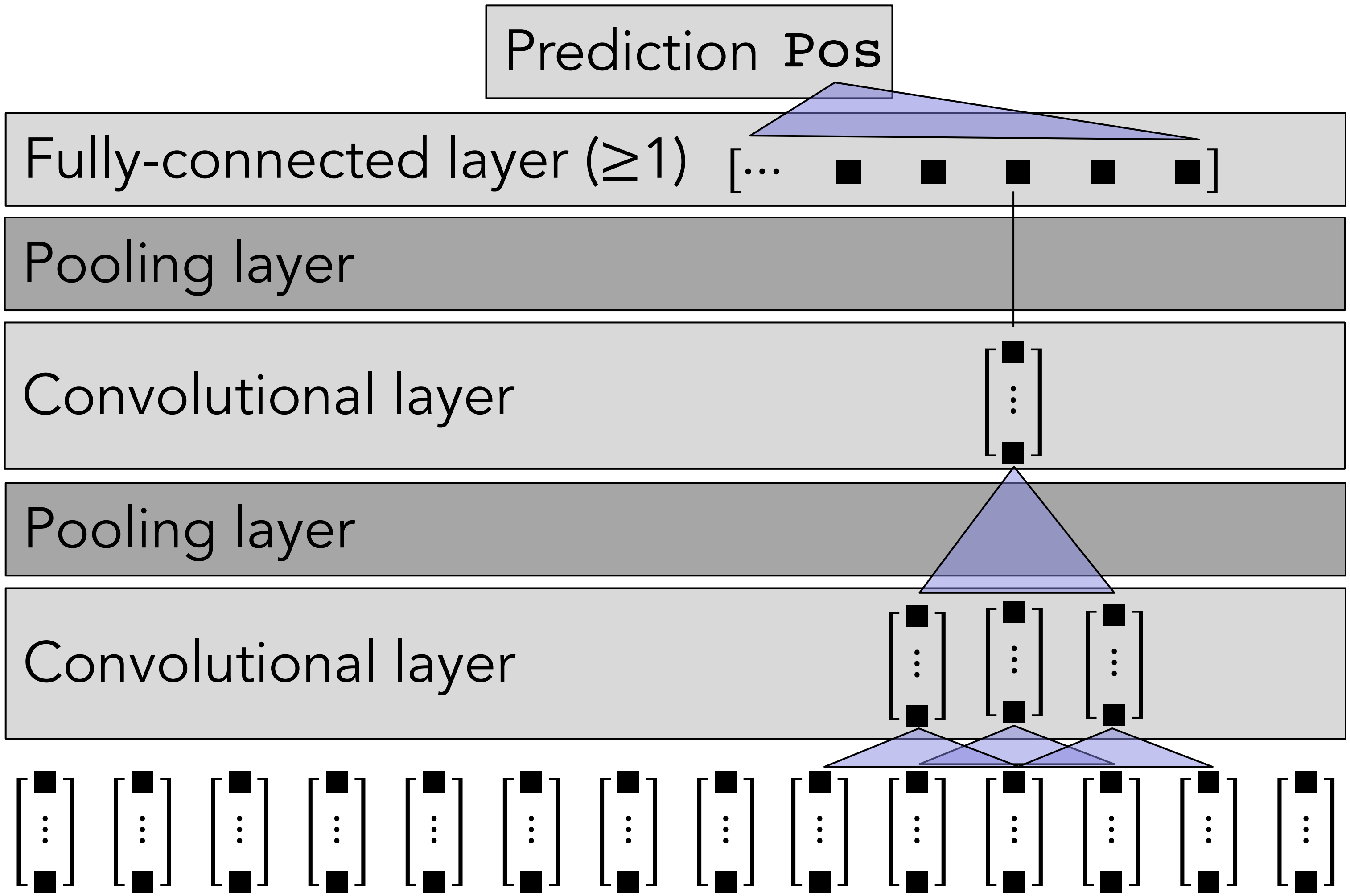
I would never buy this product again. It clearly failed under high-stress testing in my home.	I would clearly buy this product again. It never failed under high-stress testing in my home.

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

CNNs for text categorization

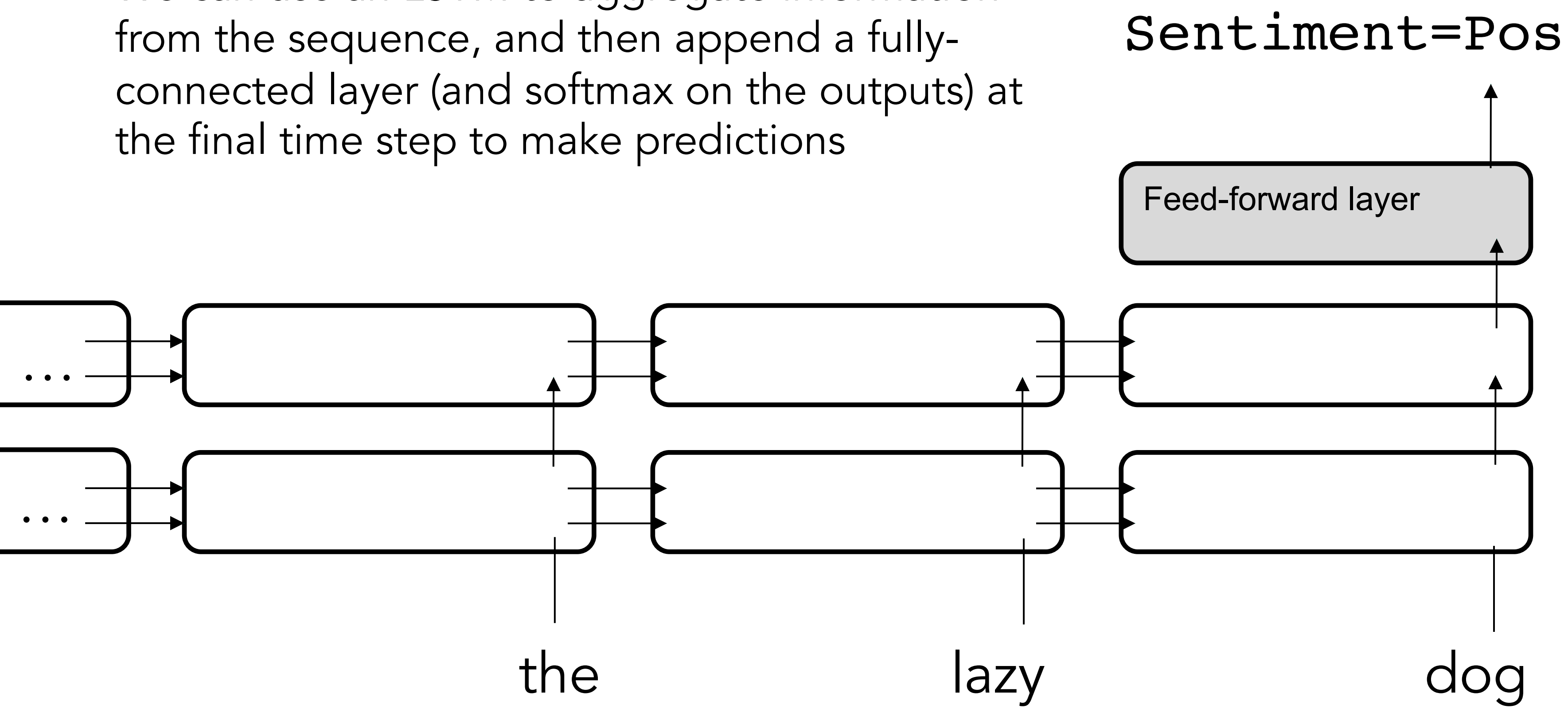
- In a convolutional model, we can use pooling operations to aggregate features across the entire sentence / text
- And then use this representation as an input to a standard feed-forward neural network for text categorization

Convolutional architecture for text categorization

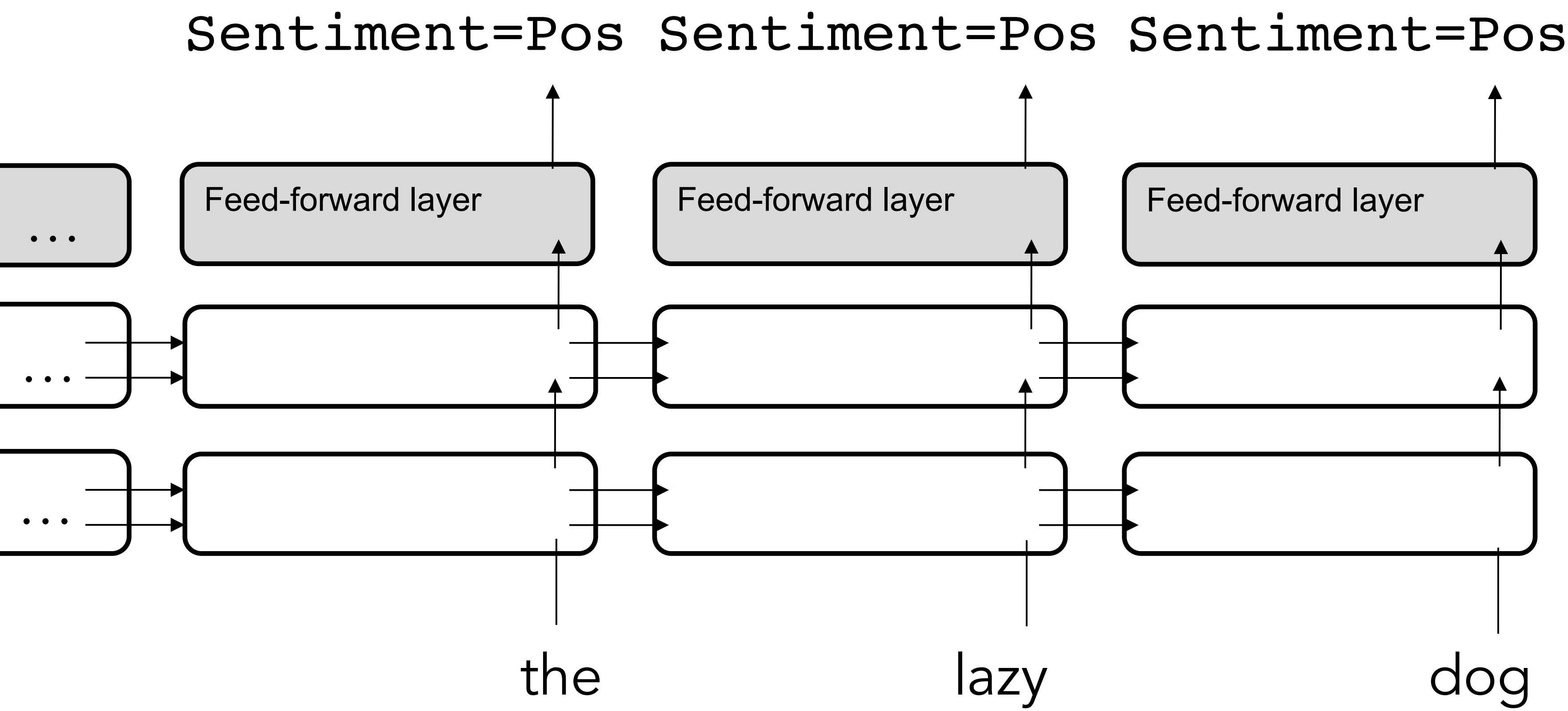


RNNs for text categorization

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.:

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

TRANSFORMERS AND BERT (ETC.)

Attention

- Attention is a mechanism used in neural network models to determine how much weight is given to different evidence (pixels, time steps or word vectors) in making a prediction
- Imagine a two-step process
  - First we determine what information is relevant for the prediction we want to make
  - Then we make a prediction using only the relevant information (or giving it more weight)

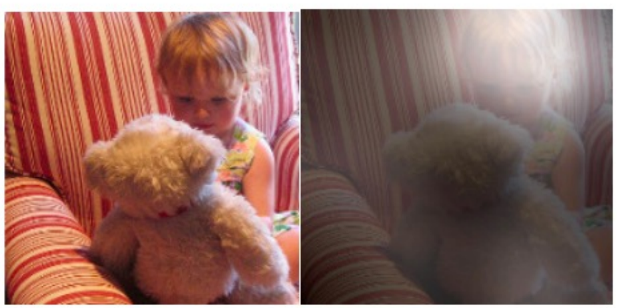
Attention

- For instance, in computer vision, attention mechanisms are used to identify regions of the image that are relevant for classification

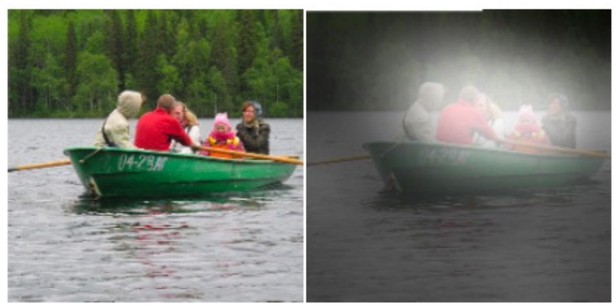
A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

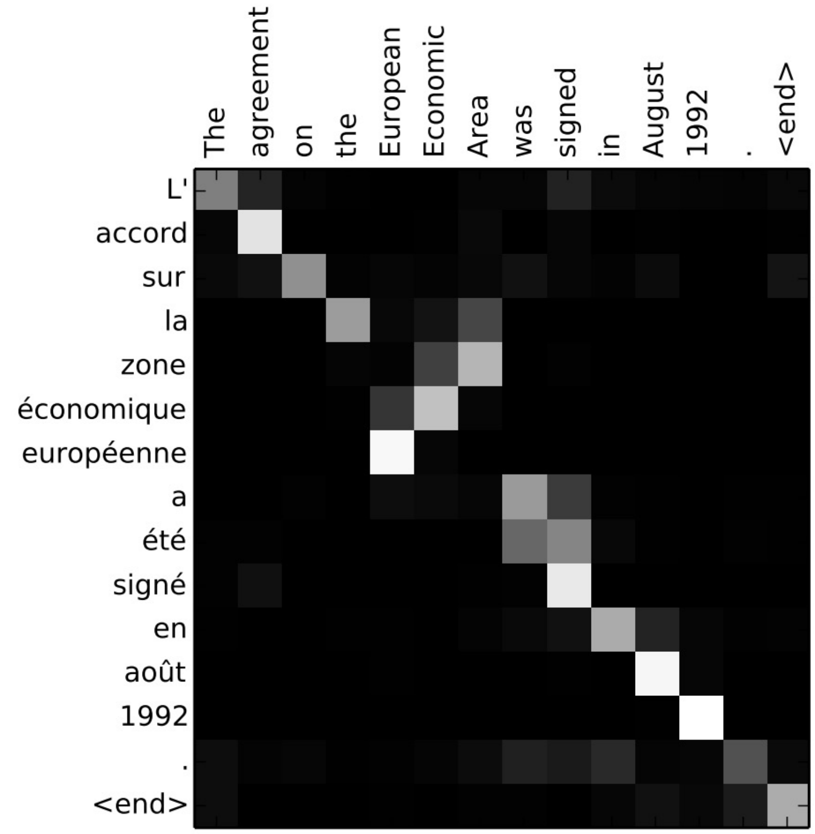
Xu, Kelvin, et al. *"Show, attend and tell: Neural image caption generation with visual attention." International Conference on Machine Learning*. 2015.

ILLINOIS INSTITUTE OF TECHNOLOGY

*Transforming Lives.Inventing the Future.***www.iit.edu**

Attention

- Attention can also be used to determine how strongly words or context vectors are weighted in NLP



Bahdanau, D. et al. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR.

Attention

- Mathematically, an attention function computes a distribution over a set of input vectors. This distribution can be used as a basis for a weighted sum of the vectors, giving higher weight to those elements to which the model is "attending"
- This distribution is calculated by applying a *compatibility* function f to each input vector, and applying a softmax transformation to the result.

$$Attention(q, \mathbf{k}, \mathbf{v}) = \sum_i \frac{e^{f(q,\mathbf{k}_i)}}{\sum_j e^{f(q,\mathbf{k}_j)}} \mathbf{v}_i$$

query: element we are deciding about when we need to focus attention

Attention

- Mathematically, an attention function computes a distribution over a set of input vectors. This distribution can be used as a basis for a weighted sum of the vectors, giving higher weight to those elements to which the model is "attending"
- This distribution is calculated by applying a *compatibility* function f to each input vector, and applying a softmax transformation to the result.

$$Attention(q, \mathbf{k}, \mathbf{v}) = \sum_i \frac{e^{f(q,\mathbf{k}_i)}}{\sum_j e^{f(q,\mathbf{k}_j)}} \mathbf{v}_i$$

keys: elements to be checked for compatibility with the key

Attention

- Mathematically, an attention function computes a distribution over a set of input vectors. This distribution can be used as a basis for a weighted sum of the vectors, giving higher weight to those elements to which the model is "attending"
- This distribution is calculated by applying a *compatibility* function f to each input vector, and applying a softmax transformation to the result.

$$Attention(q, \mathbf{k}, \mathbf{v}) = \sum_i \frac{e^{f(q,\mathbf{k}_i)}}{\sum_j e^{f(q,\mathbf{k}_j)}} \mathbf{v}_i$$

values: elements to be aggregated based on compatibility of keys

The Transformer Architecture

- Interleaved self-attention and feed-forward layers
- Special encoding necessary in order to preserve information about order of words

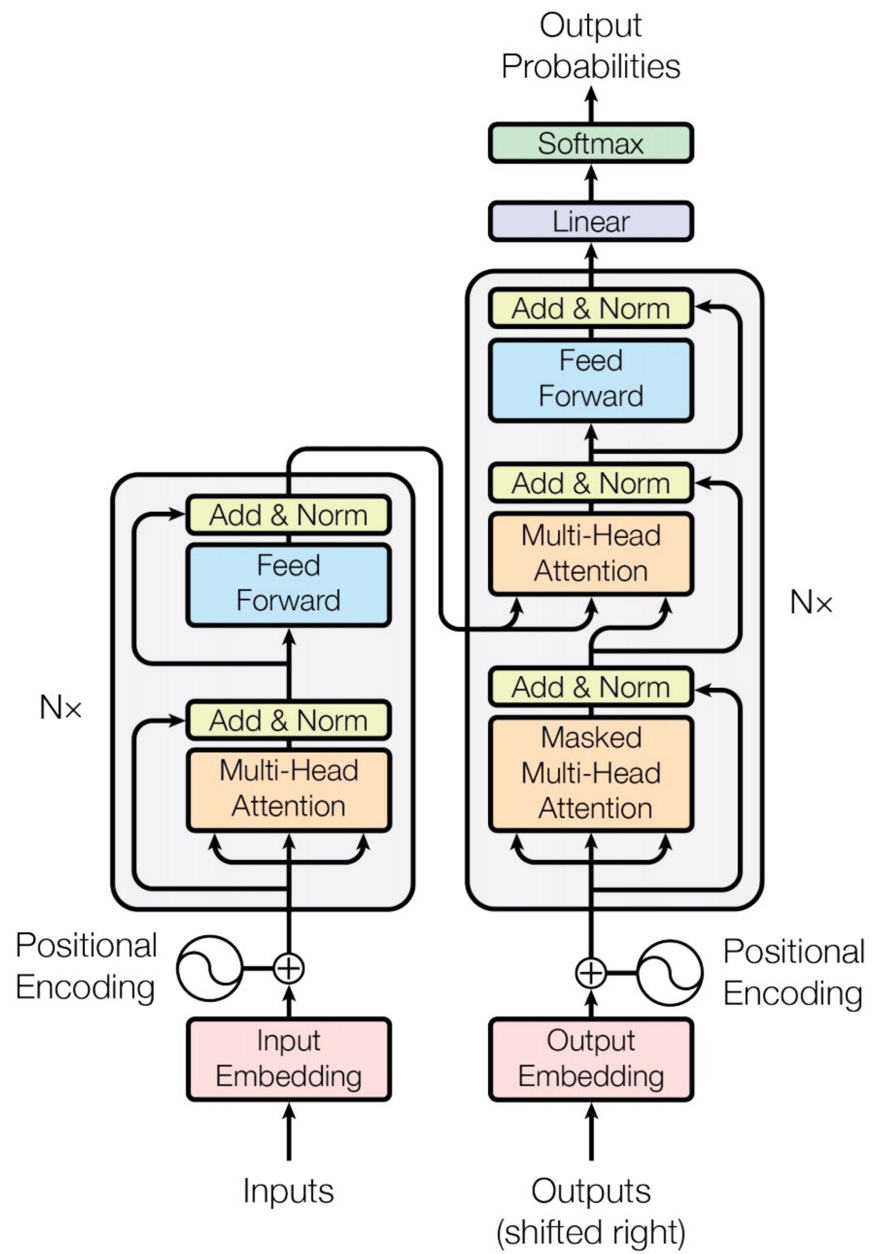


Figure 1: The Transformer - model architecture.

BERT and friends

- In the last 5 years or so, big NLP labs have developed powerful new neural network frameworks that facilitate transfer learning: learning good representations that will perform well across a range of tasks
- They
  - Are computationally intensive
  - Leverage deep networks (recurrent or transformer-based)
  - Can be pre-trained on a language modeling objective and fine-tuned on specific tasks

BERT and friends

- GPT / GPT-2 / GPT-3 (AI^2)
  - Unidirectional transformer-based language model
- ELMo (AI^2)
  - Word representations based on internal states of biLSTM
- BERT (Google AI)
  - Transformer-based model for text categorization and sequence modeling
  - Trained on special masked language modeling objective