Information Retrieval

Amin Nazari

Spring 2025

Chapter 8

Modern text classification

Agenda

- Introduction to Text Classification
- MLP for Text Classification
- RNN for Text Classification
- LSTM for Text Classification
- Comparison & Results
- Conclusion

What is Text Classification?

- Assigning categories to text data
- Applications: Spam detection, sentiment analysis, topic labeling
- Challenges: High-dimensional data, sequence dependency, contextual meaning

Preprocessing for Text Classification

- Tokenization
- Lowercasing
- Stopword Removal
- Vectorization (TF-IDF, Word2Vec, Embeddings)

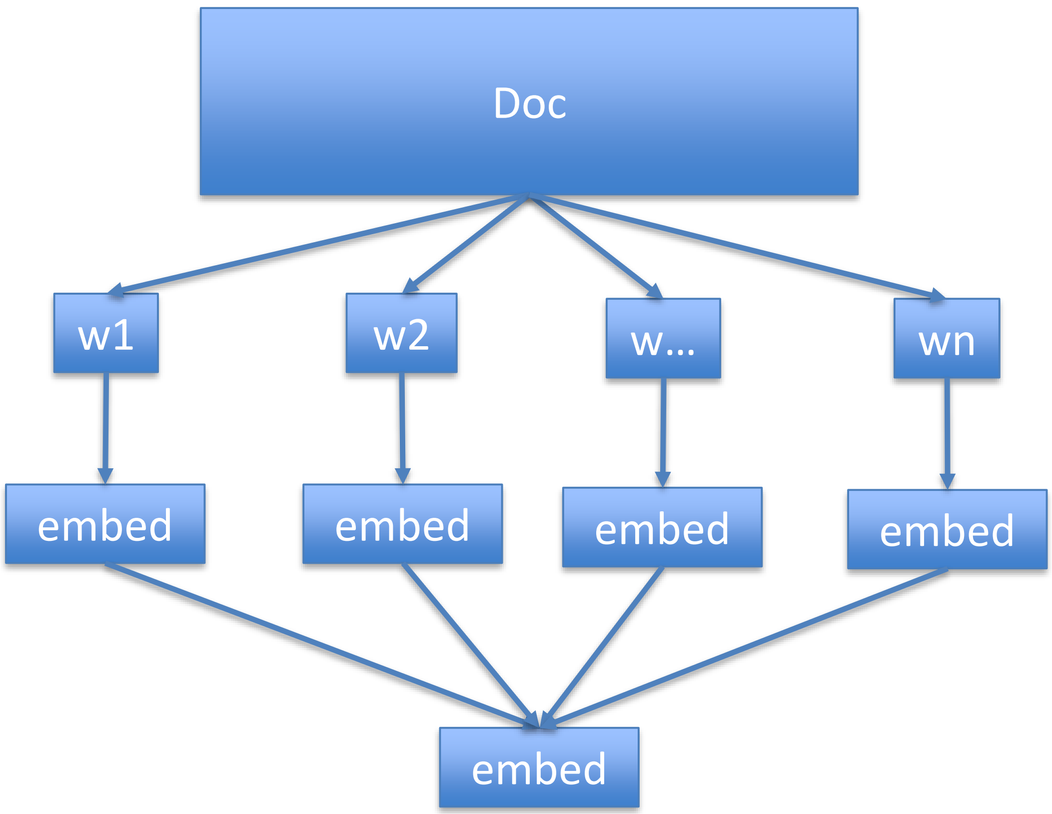
Preprocessing for Text Classification

	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

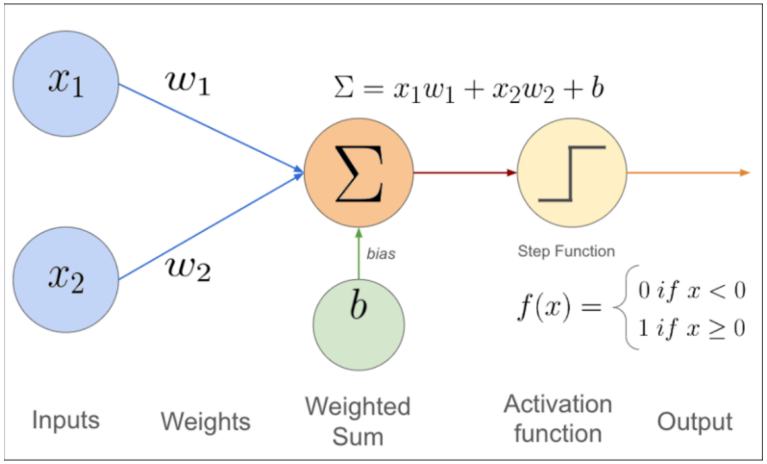
Word Vector (Passage Vector)

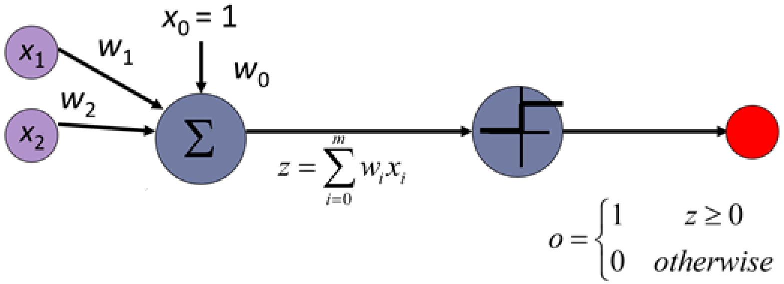
Document Vector

Preprocessing for Text Classification



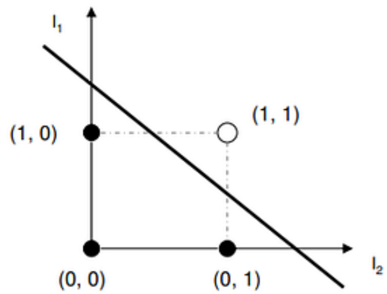
Basic Components of Perceptron



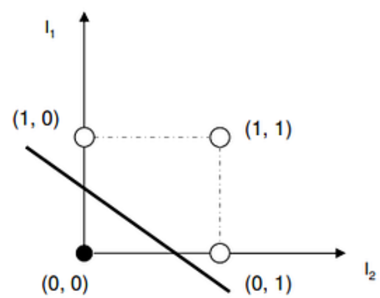


XOR

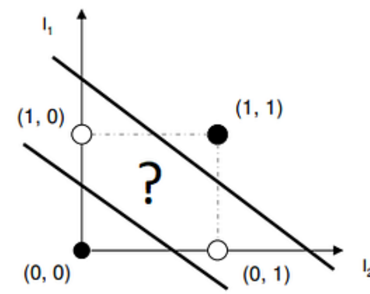
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



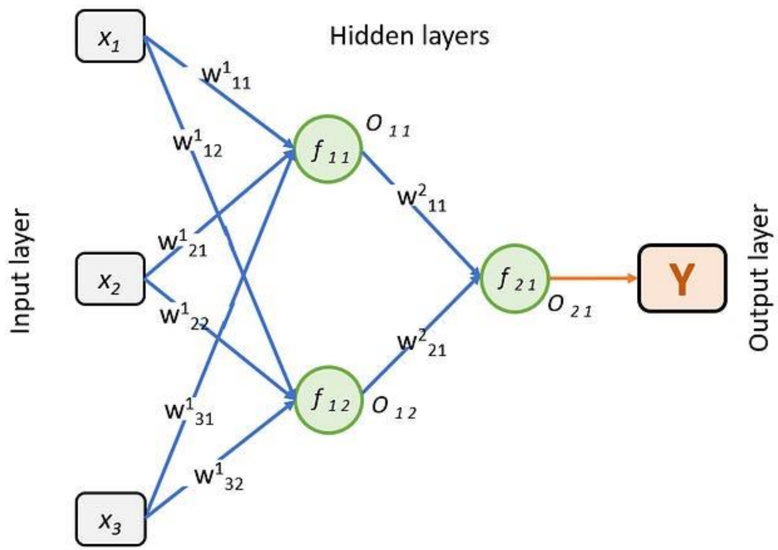
OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



Forward Propagation



here,

x - inputs

w^n_{ij} - n is num of next hidden layer,

i is neuron num of previous layer,

j is neuron num of next layer

(w^2_{21}- n is next layer,

i is 2nd neuron of previous layer,

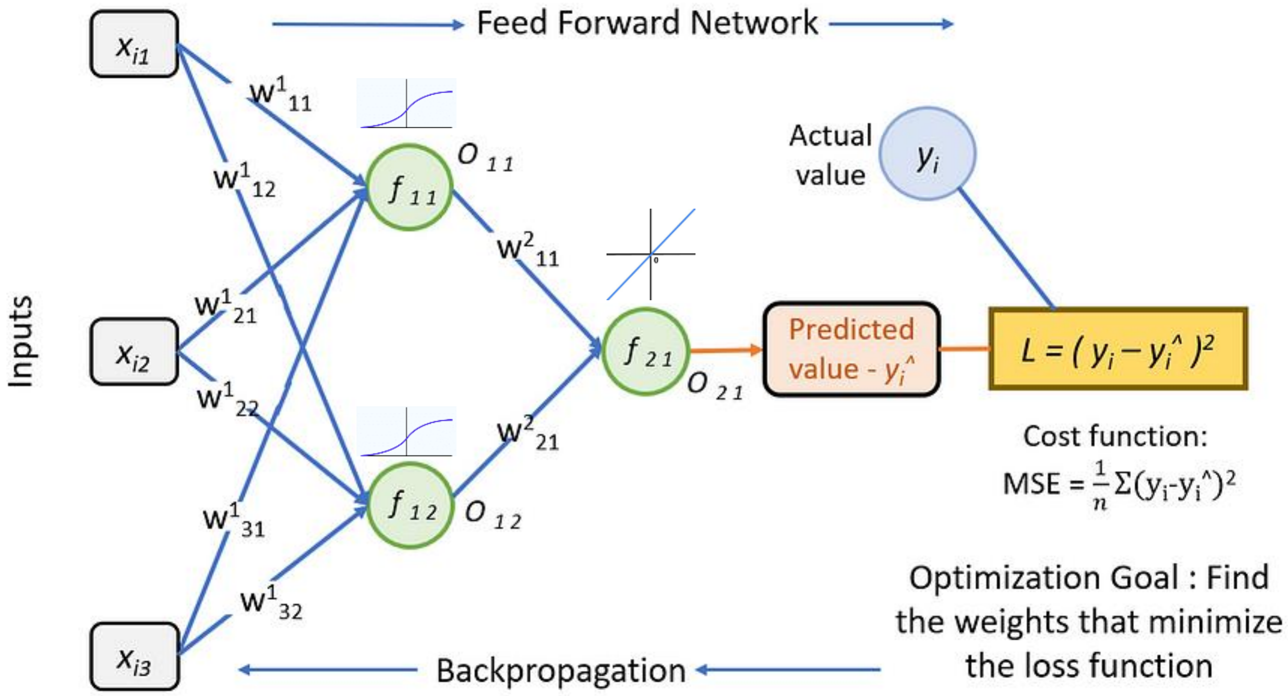
j is 1st neuron of next layer)

f_{ij} - i is num of layer and j is num of neuron

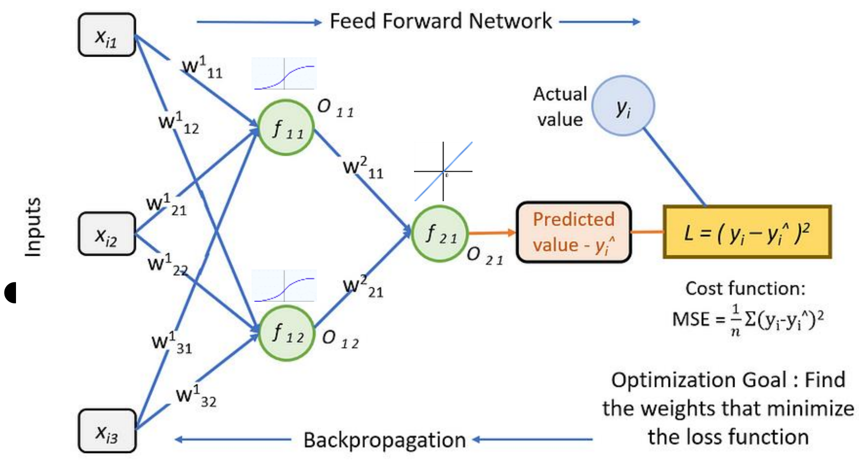
O_{ij} - output from ith layer jth neuron

Y - output

Back Propagation



Back Propagation (Regression)



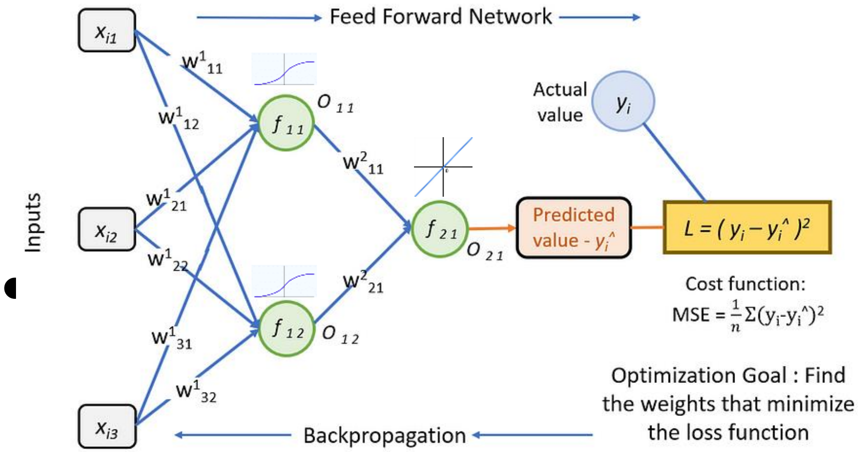
$$f_{11} = w_{11} * x_1 + w_{21} * x_2 + w_{31} * x_3$$
$$f_{12} = w_{12} * x_1 + w_{22} * x_2 + w_{32} * x_3$$

$$O_{11} = Sigmoid(f_{11})$$
$$O_{12} = Sigmoid(f_{12})$$

$$f_{21} = v_{11} * O_{11} + v_{21} * O_{21}$$
$$O_{21} = f_{21}$$

$$E = (\mathrm{y} - \mathrm{O}_{21})^2$$

Back Propagation (Regression)



$$f_1 = x_{(1*3)} * w_{(3*2)}$$
$$O_1 = Sigmoid(f_1)$$

$$f_2 = O_{1(1*2)} * v_{(2*1)}$$
$$O_2 = (f_2)$$

$$E = (\mathrm{y} - \mathrm{O}_2)^2$$

Back Propagation (Regression)

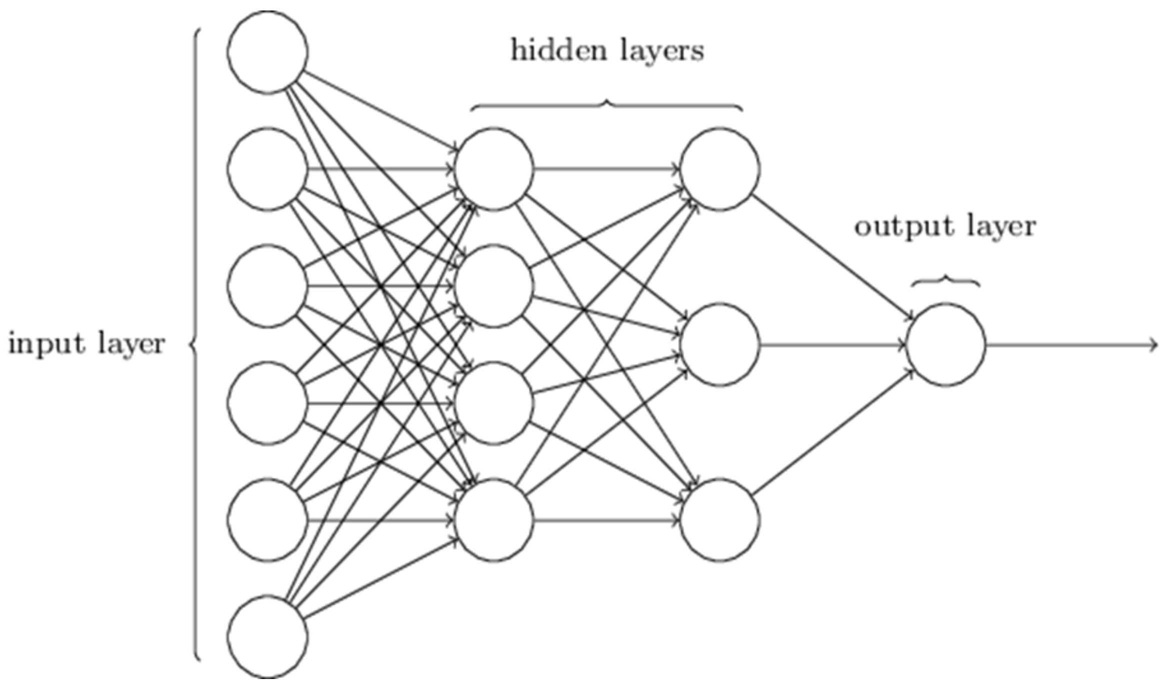
- $w = w - \alpha \frac{\partial E(w)}{\partial w}$
- $v = v - \alpha \frac{\partial E(v)}{\partial v}$
- $f_1 = x_{(1*3)} * w_{(3*2)};\ O_1 = Sigmoid(f_1)$
- $f_2 = O_{1(1*2)} * v_{(2*1)}; O_2 = f_2;$
- $E = (\mathrm{y} - \mathrm{O}_2)^2;$
- $\frac{\partial E}{\partial v} = \frac{\partial E}{\partial \mathrm{O}_2} \cdot \frac{\partial \mathrm{O}_2}{\partial f_2} \cdot \frac{\partial f_2}{\partial v} = -(\mathrm{y} - \mathrm{O}_2) \times 1 \times O_1$
- $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \mathrm{O}_2} \cdot \frac{\partial \mathrm{O}_2}{\partial f_2} \cdot \frac{\partial f_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial f_1} \cdot \frac{\partial f_1}{\partial w} = -(\mathrm{y} - \mathrm{O}_2) \times 1 \times v \times (O_1(1-O_1)) \times x$

Back Propagation (Classification)

- $w = w - \alpha \frac{\partial E(w)}{\partial w}$
- $v = v - \alpha \frac{\partial E(v)}{\partial v}$
- $f_1 = x_{(1*3)} * w_{(3*2)}$; $O_1 = Sigmoid(f_1)$
- $f_2 = O_{1(1*2)} * v_{(2*1)}$; $O_2 = Sigmoid(f_2)$;
- $E = \mathrm{y} * \log(\mathrm{O}_2) + (1 - y) * \log(1 - O_2)$;
- $\frac{\partial E}{\partial v} = \frac{\partial E}{\partial \mathrm{O}_2} \cdot \frac{\partial \mathrm{O}_2}{\partial f_2} \cdot \frac{\partial f_2}{\partial v} = -(\frac{\mathrm{y}}{\mathrm{O}_2} + \frac{-(1-y)}{1-O_2}) \times (O_2(1\text{-}O_2)) \times O_1 = -(y - O_2) \times O_1$
- $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \mathrm{O}_2} \cdot \frac{\partial \mathrm{O}_2}{\partial f_2} \cdot \frac{\partial f_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial f_1} \cdot \frac{\partial f_1}{\partial w} = -(\frac{\mathrm{y}}{\mathrm{O}_2} + \frac{-(1-y)}{1-O_2}) \times (O_2(1\text{–}O_2)) \times v \times (O_1(1\text{-}O_1)) \times x$
- $\frac{\partial E}{\partial w} = -(y - O_2) \times v \times (O_1(1\text{–}O_1)) \times x$

MLP (Multilayer Perceptron)

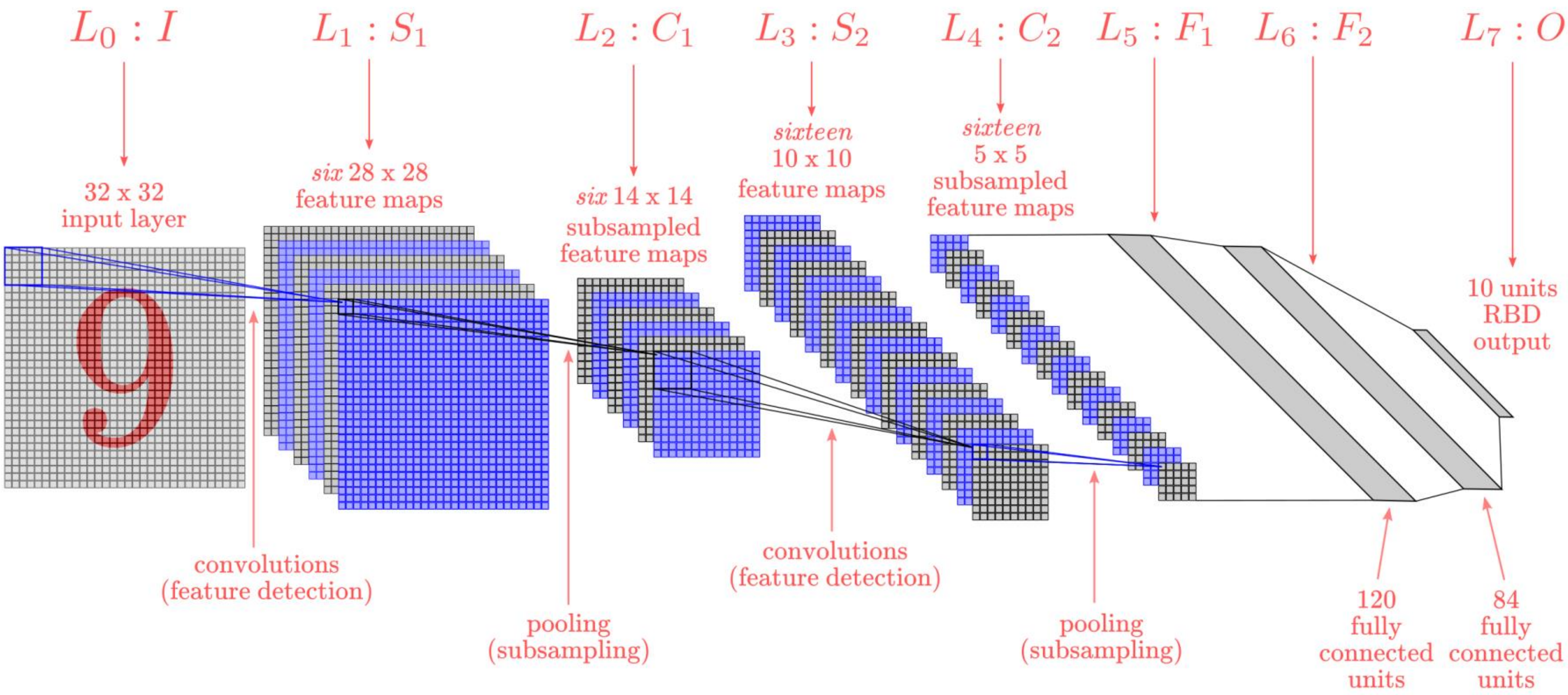
- Input → Hidden Layers → Output
- Uses fixed-size input (TF-IDF, bag-of-words)
- Limitation: Ignores word order and sequence



Convolutional Neural Networks (CNN)

- Originally designed for image recognition
- Captures local features and spatial hierarchies
- Effective for detecting patterns in text sequences
- Handles variable input length (via padding/truncation)

Convolutional Neural Networks (CNN)



CNN Architecture for Text

- Layers:
  - **Input: Tokenized and embedded sentences (e.g., Word2Vec, GloVe)**
  - **Embedding**
  - **Convolution Layer(s):** Filters slide over embedded text (like n-grams) and detects key patterns
  - **Pooling Layer(s):** Max pooling selects the most important features
  - **Fully Connected Layer**
  - **Softmax Output Layer**

CNN Architecture for Text

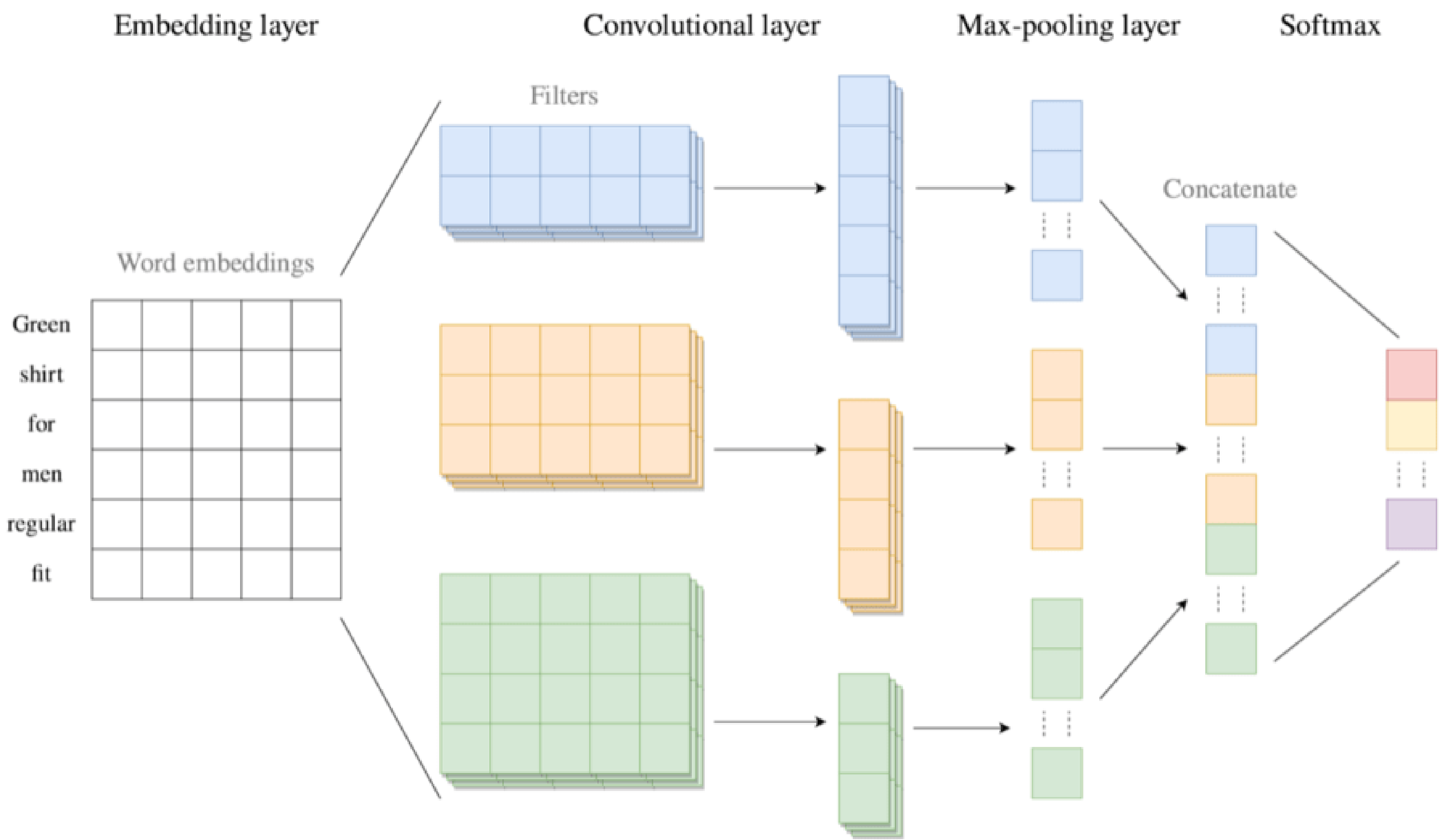


Figure 3: General CNN architecture for text classification

CNN Architecture for Text

- Input: Sentence of 100 words
- Embedding: 100 x 300 matrix
- Conv layer: 100 filters (kernel size = 3x300)
- Pooling: Global max pooling
- Dense: 128 neurons
- Output: Softmax with 3 classes

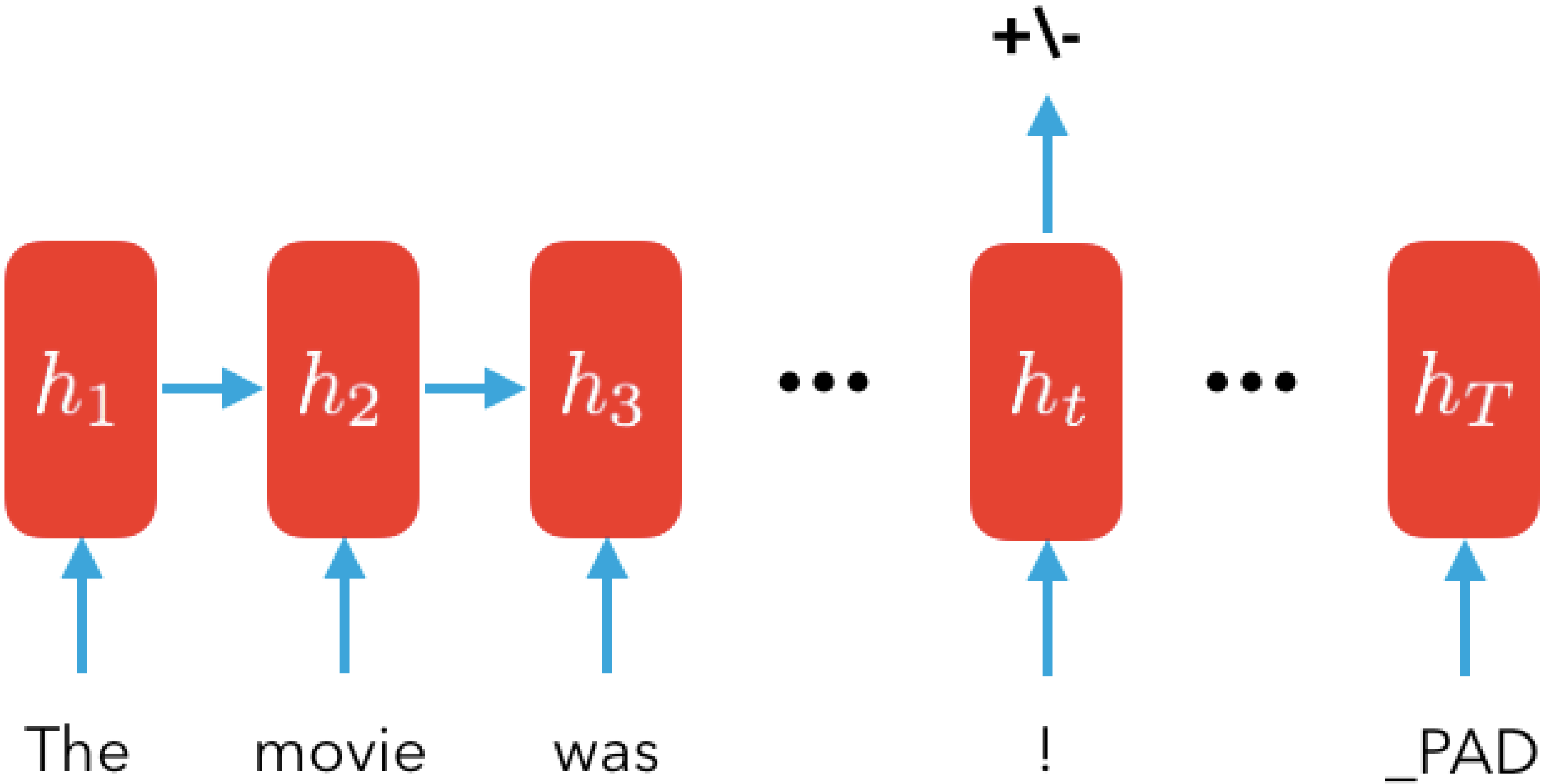
CNN Architecture for Text

- To capture **different n-gram features**:
  - Filter size = 3 might capture short phrases like "free money now"
  - Filter size = 5 might capture longer phrases like "you have won a prize"

RNN (Recurrent Neural Network)

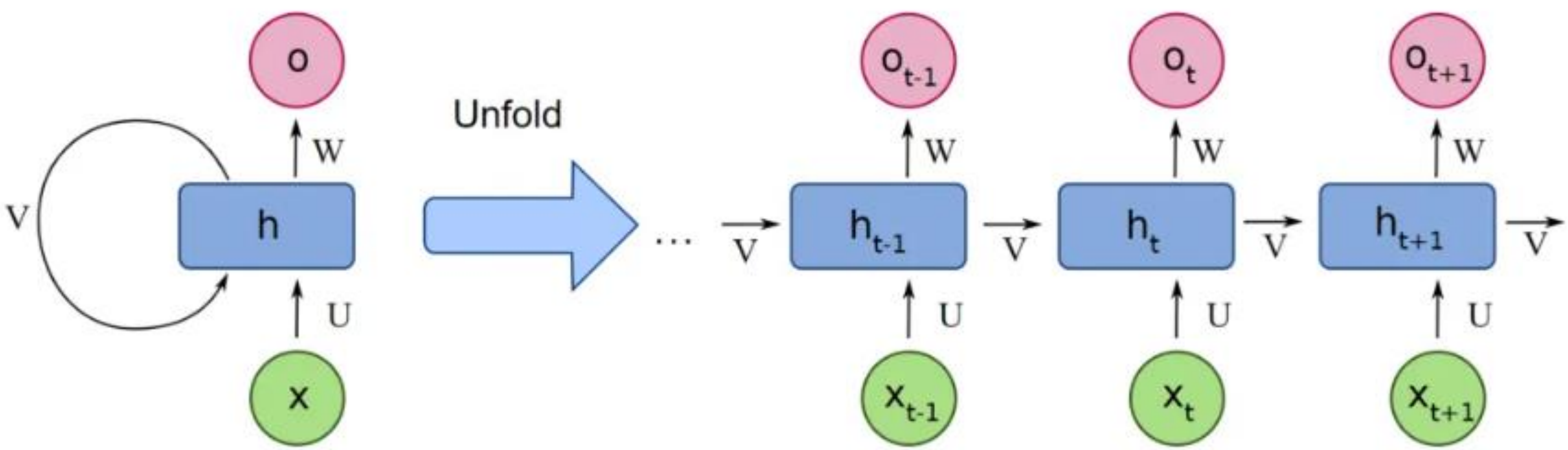
- Designed for sequence data
- Maintains hidden state h_t
- Processes text word by word
- Limitation: Vanishing gradient, poor long-term memory

RNN (Recurrent Neural Network)



RNN – Architecture

- RNN cells process sequence: x1, x2, ..., xn → h1, h2, ..., hn
- Output from final hidden state goes to classifier



RNN – Architecture

For input sequence $\{x_1, x_2, ..., x_T\}$, the hidden state is updated as:

$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

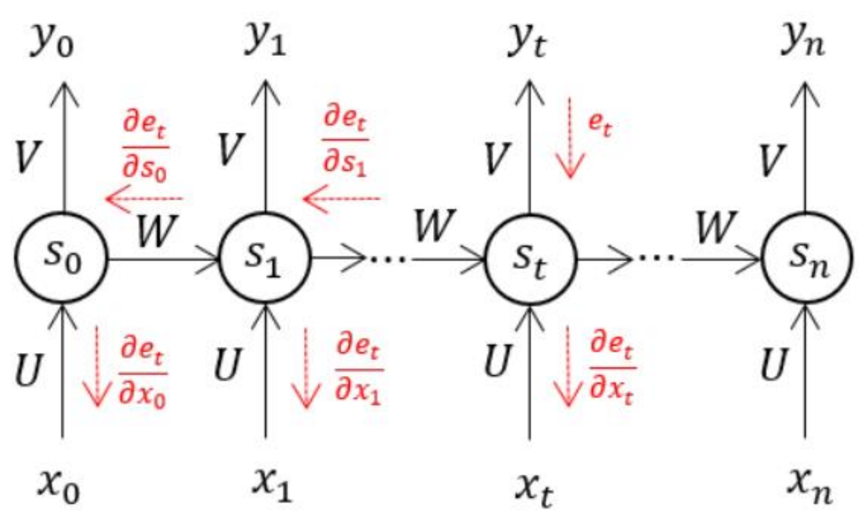
Where:

- W: **Input weight matrix** (same for all t)
- U: **Hidden-to-hidden weight matrix** (also same)
- b: Bias (shared)
- h_t: Hidden state at time t

✅ These matrices W, U, b are **learned during training** and reused at **each time step**.

- **Forward pass:** process sequence
- **Loss:** sum of losses at all time steps
- **Backward pass:** compute gradients at each time step and sum them up
- **Update:** use the summed gradients to update parameters once per sequence (or batch)

$$\frac{\partial L}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \theta}$$



Why Do We Need LSTM?

- Traditional RNNs struggle with long sequences.
- They forget important past information due to:
  - Vanishing gradients
  - Exploding gradients
- Example: Predicting words in long sentences → needs memory of earlier context.

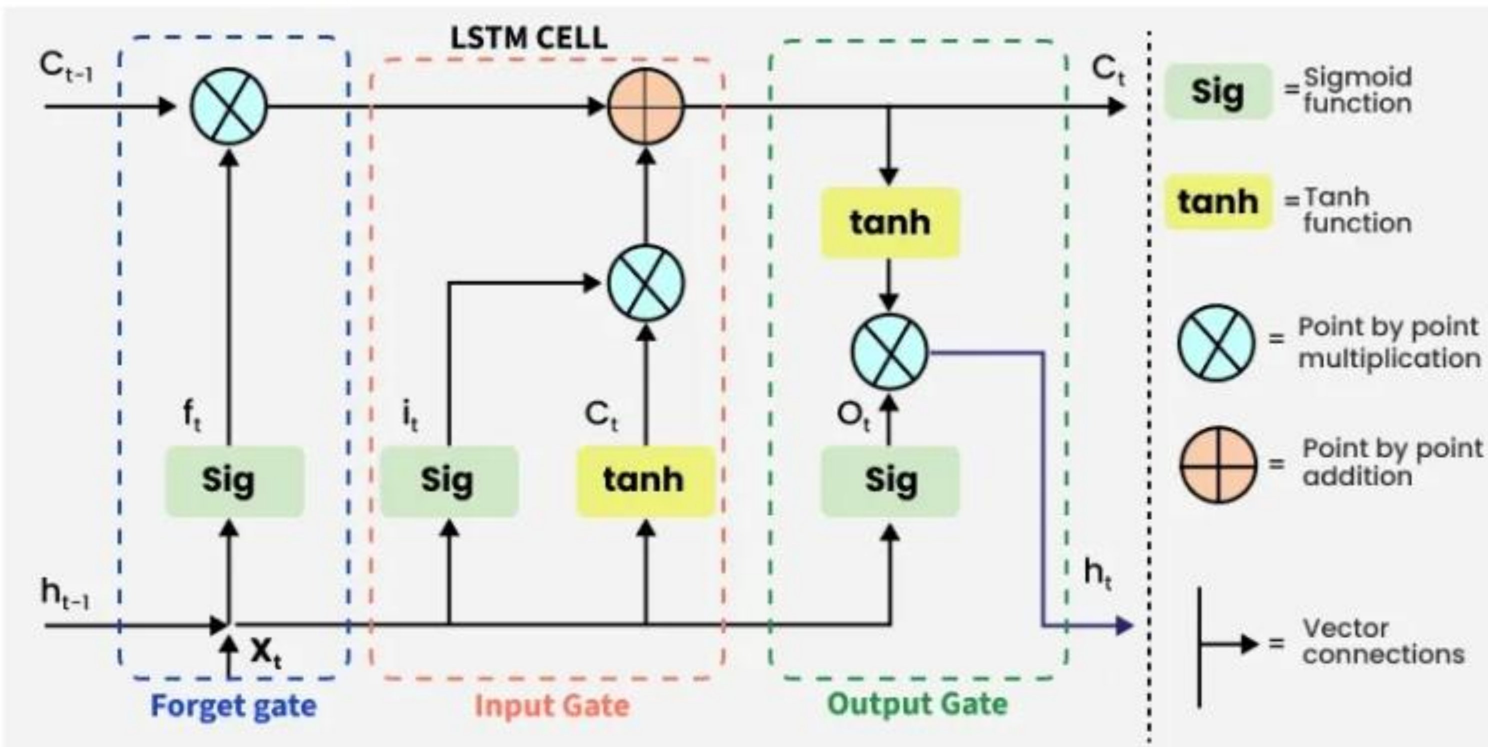
LSTM in a Nutshell

- A type of Recurrent Neural Network (RNN).
- Designed to remember long-term dependencies.
- Introduced by Hochreiter & Schmidhuber (1997).
- Uses gates to control memory.

LSTM Cell: Overview

- Maintains two states:
  - Hidden state (h_t)
  - Cell state (c_t)
- Controls memory using three gates:
  - Forget gate
  - Input gate
  - Output gate

LSTM



LSTM Model

Step 1: Forget Gate

$$f_t = \sigma(W_f\, x_t + U_f\, h_{\{t-1\}} + b_f)$$

-Decides what information to discard from the cell state.

- Values close to 0 → forget
- Values close to 1 → keep

Step 2: Input Gate

$$i_t = \sigma(W_i x_t + U_i h_{\{t-1\}} + b_i)$$
$$\sim c_t = \tanh(W_c x_t + U_c h_{\{t-1\}} + b_c)$$

- Input gate: decides what new info to add.
- Candidate state: possible new values to add to memory.

Step 3: Update Cell State

$$c_t = f_t \odot c_{\{t-1\}} + i_t \odot \sim c_t$$

-Combine old memory with new info.

- Element-wise operations preserve gradient flow.

Step 4: Output Gate

$$o_t = \sigma(W_o \ x_t + U_o \, h_{\{t-1\}} + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

- Output gate: controls what to send to the next time step or output layer.

Summary of LSTM Flow

-$x_t, h_{\{t-1\}}$ →all gates

- Gates control:
  - What to forget
  - What to update
  - What to output
- Cell state (c_t) carries long-term memory
- Hidden state (h_t) carries short-term output

Why Use LSTM?

- Solves vanishing gradient problem
- Retains important info over long sequences
- Works well in:
  - Text generation
  - Language modeling
  - Time series prediction
  - Speech recognition

Training Setup

- Models: KNN, SVM, …
- Models: MLP (TF-IDF), CNN, RNN & LSTM

Conclusion

- MLP: Simple, ignores word order
- RNN: Captures sequence, weak on long dependencies
- LSTM: Best performance via memory and context
- Future: Try BiLSTM, GRU, attention