Natural Language Processing
自然語言處理

黃瀚萱
Department of Computer Science
National Chengchi University
2020 Fall

Deep Neural Networks for NLP

Schedule

Date	Topic
9/16	Introduction
9/23	Linguistic Essentials
9/30	Collocation
10/7	Language Model
10/14	Performance Evaluation and Word Sense Disambiguation
10/21	Text Classification (HW1 will be assigned)
10/28	Invited Talk: NLP and Cybersecurity (Term Project)
11/4	POS Tagging
11/11	Midterm Exam

Schedule

Date	Topic
11/18	Chinese Word Segmentation
11/25	Word Embeddings
12/2	**Neural Networks for NLP**
12/9	Semi-supervised Learning
12/16	**Discussion about your Final Project**
12/23	**Invited Talk**
12/30	**Discourse Analysis**
1/6	Final Project Presentation II
1/13	Final Exam

Important Dates

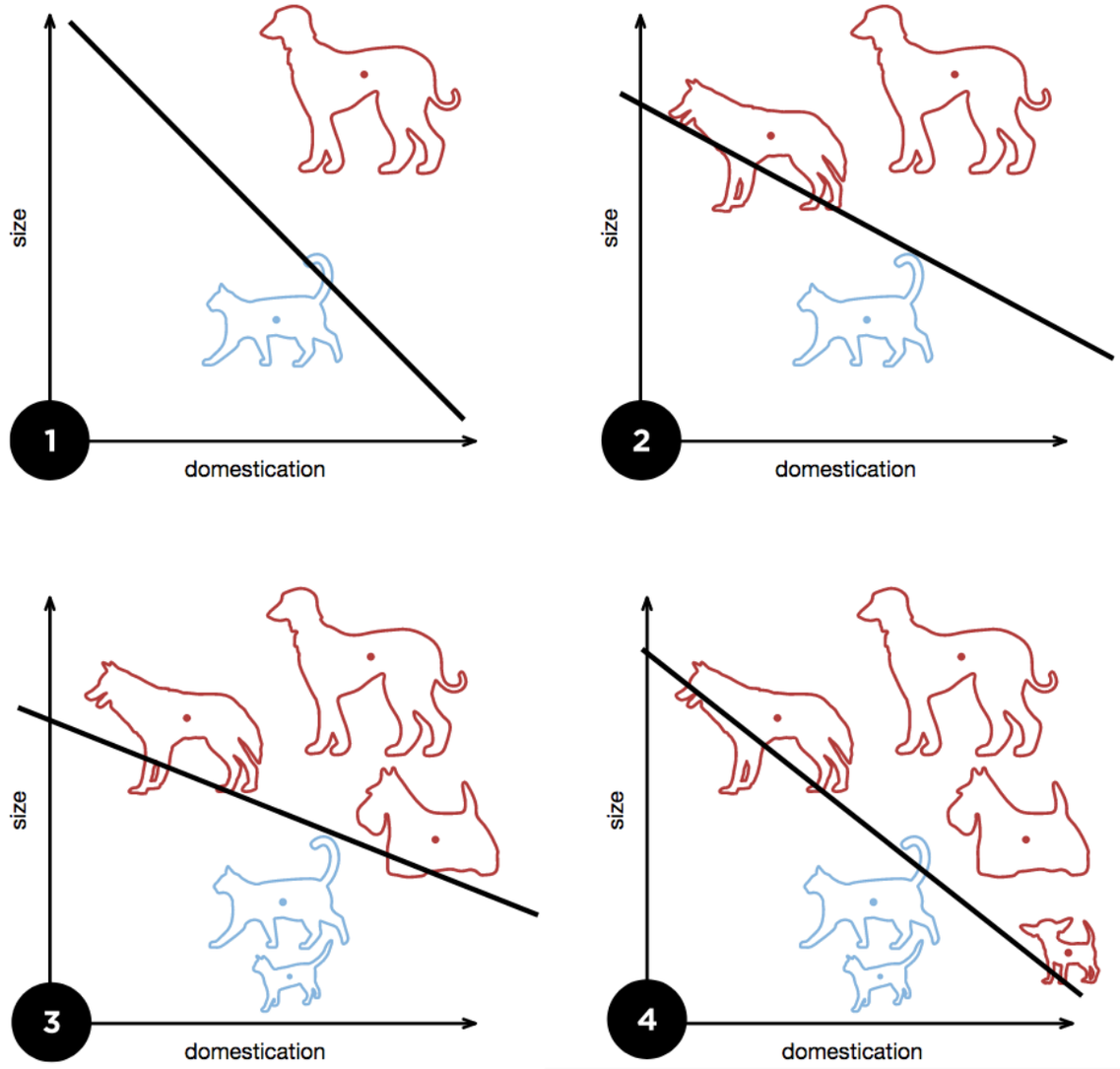
Date	Event
10/05	Release of Dataset Part I
10/28	**Tutorial in Class**
11/10	Release of Dataset Part II
11/18	**Submit Your Team Information to Moodle and Register**
12/13	Registration Due
12/16	**Discussion of Your Final Project (In Class)**
12/14 - 12/21	**Formal Run (Result Submission)**
12/25	Announcement of Formal Run Scores
12/31	Final Report Submission
2020/01/06	**Final Project Presentation**
2021/01/08	Announcement of Final Scores

Agenda

- Feed-forward neural networks
- Convolutional neural networks
- Recurrence neural networks
    - Simple RNN
    - LSTM
    - GRU
    - Attention
- Transformers
- Codelab

Perceptron (Recap)

- Features: x
- Labels: y
  - 1: Cat
  - 0: Dog



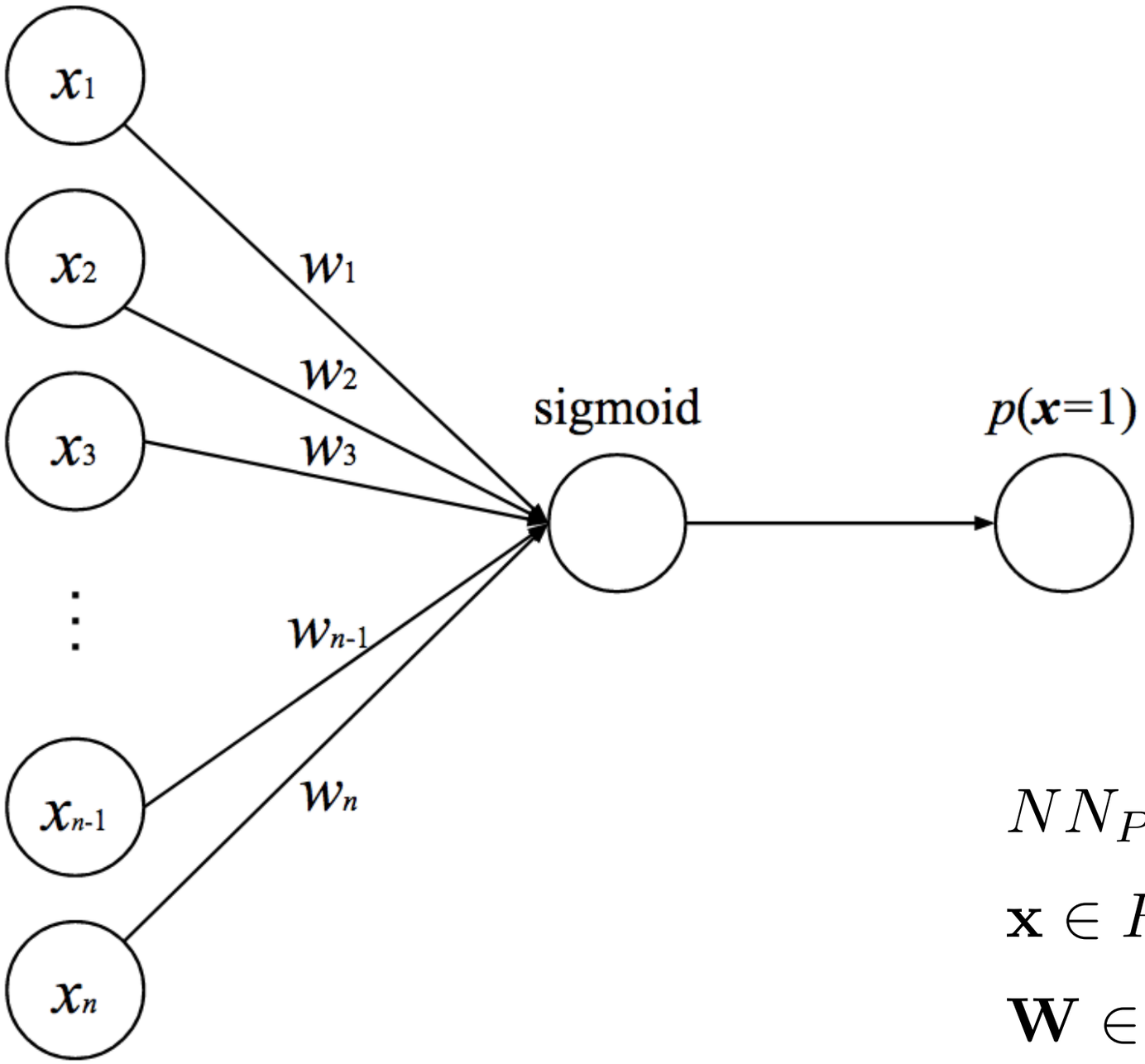
Features $(\mathbf{x})$:
x_1: size
x_2: domestication

Labels (y):
0: Dog
1: Cat

$$f(x) = \begin{cases} 1 & \text{if } w_1 x_1 + w_2 x_2 + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Parameters to estimate: w_1, w_2, and b

Single Layer Perceptron



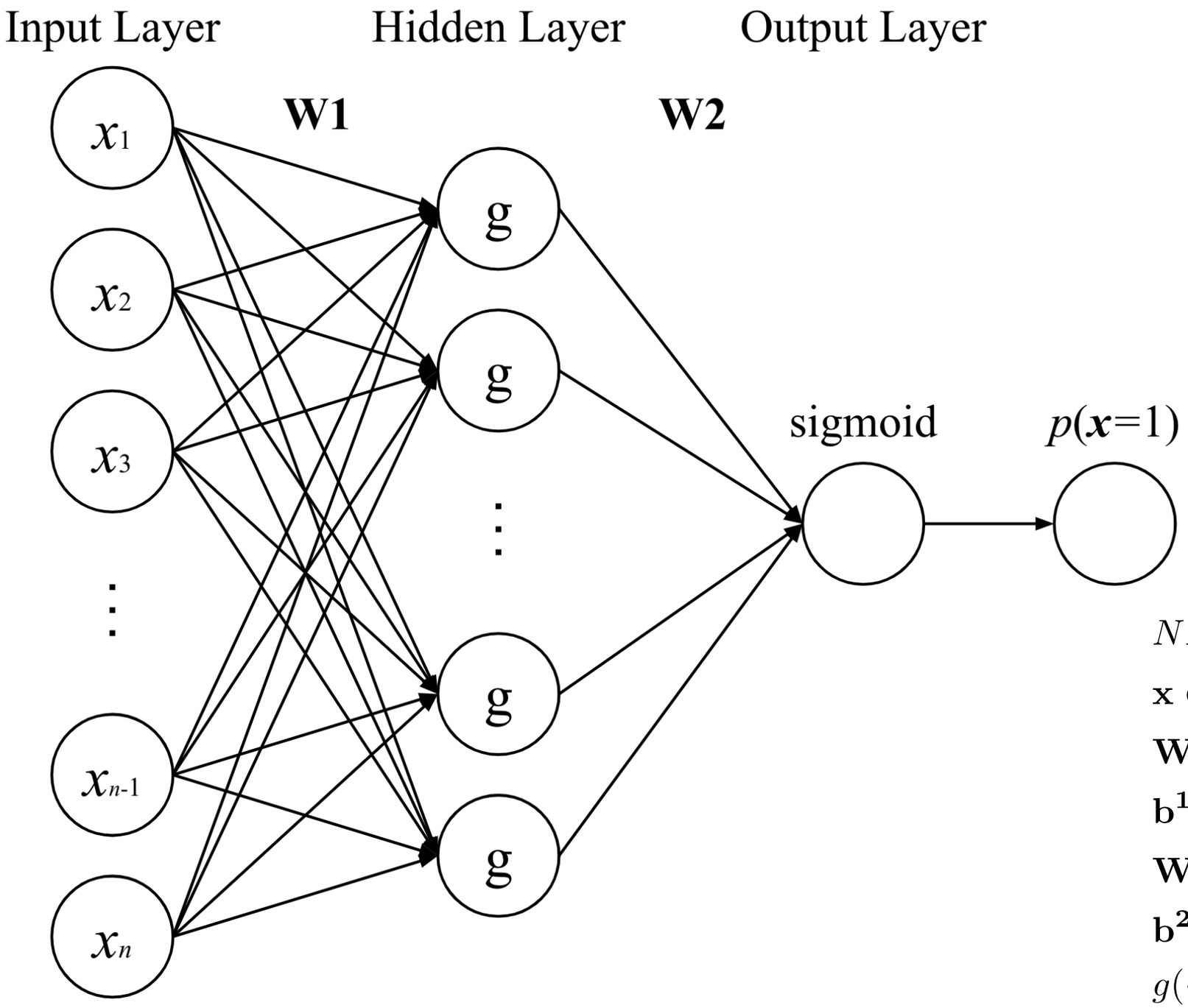
$$NN_{Perceptron}(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}$$

$\mathbf{x} \in R^{d_{in}}$: Feature vector

$\mathbf{W} \in R^{d_{in} \times d_{out}}$: Weight matrix

$\mathbf{b} \in R^{d_{out}}$: Bias terms

Multi-layer Perceptron with 1 Hidden Layer



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

$\mathbf{x} \in R^{d_{in}}$: Feature vector

$\mathbf{W^1} \in R^{d_{in} \times d_h}$: Weight matrix

$\mathbf{b^1} \in R^{d_h}$: Bias terms

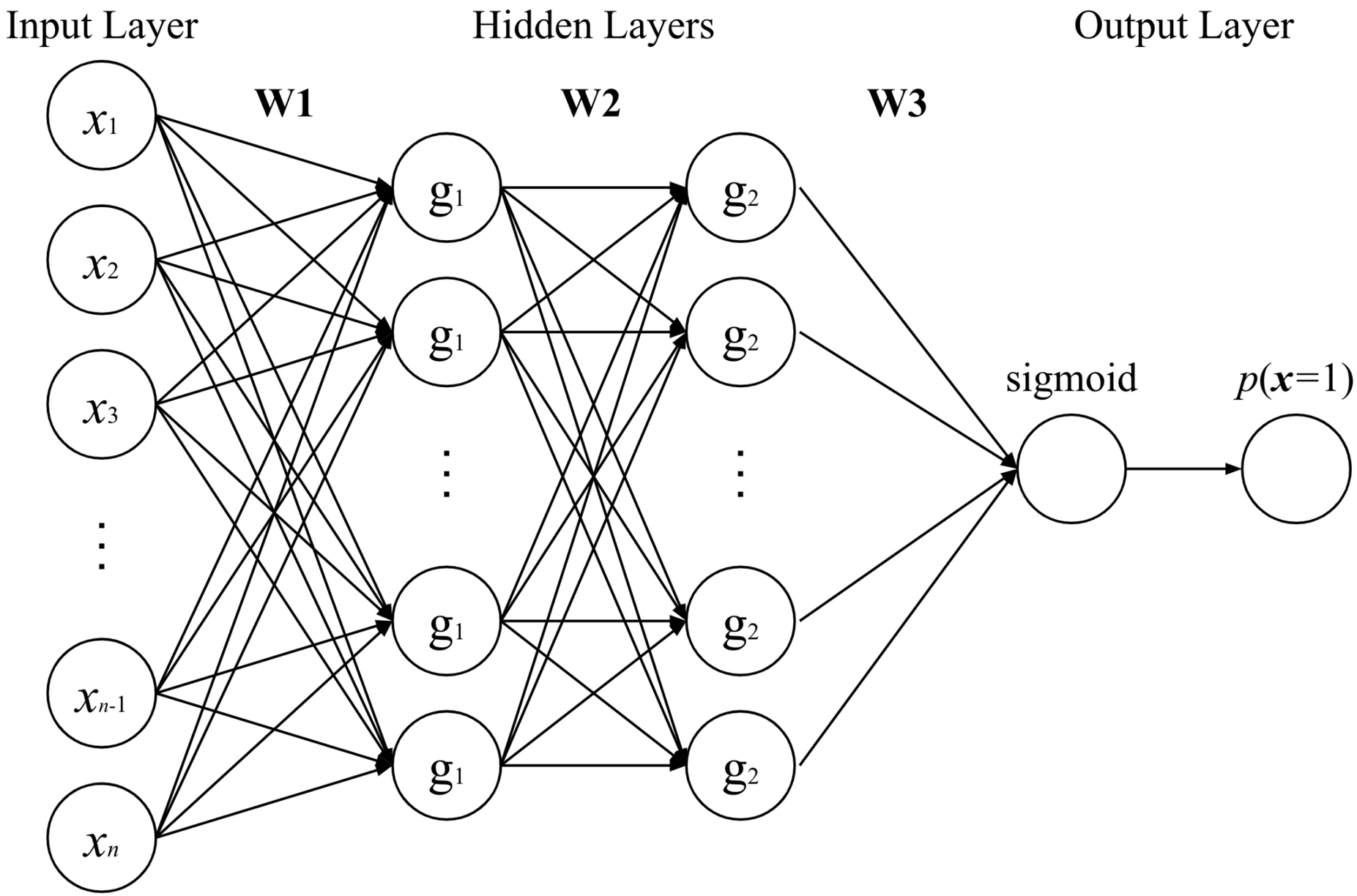
$\mathbf{W^2} \in R^{d_h \times d_{out}}$: Weight matrix

$\mathbf{b^2} \in R^{d_{out}}$: Bias terms

$g(\cdot)$: Activation function

d_h: Size of the hidden layer

One More Hidden Layer



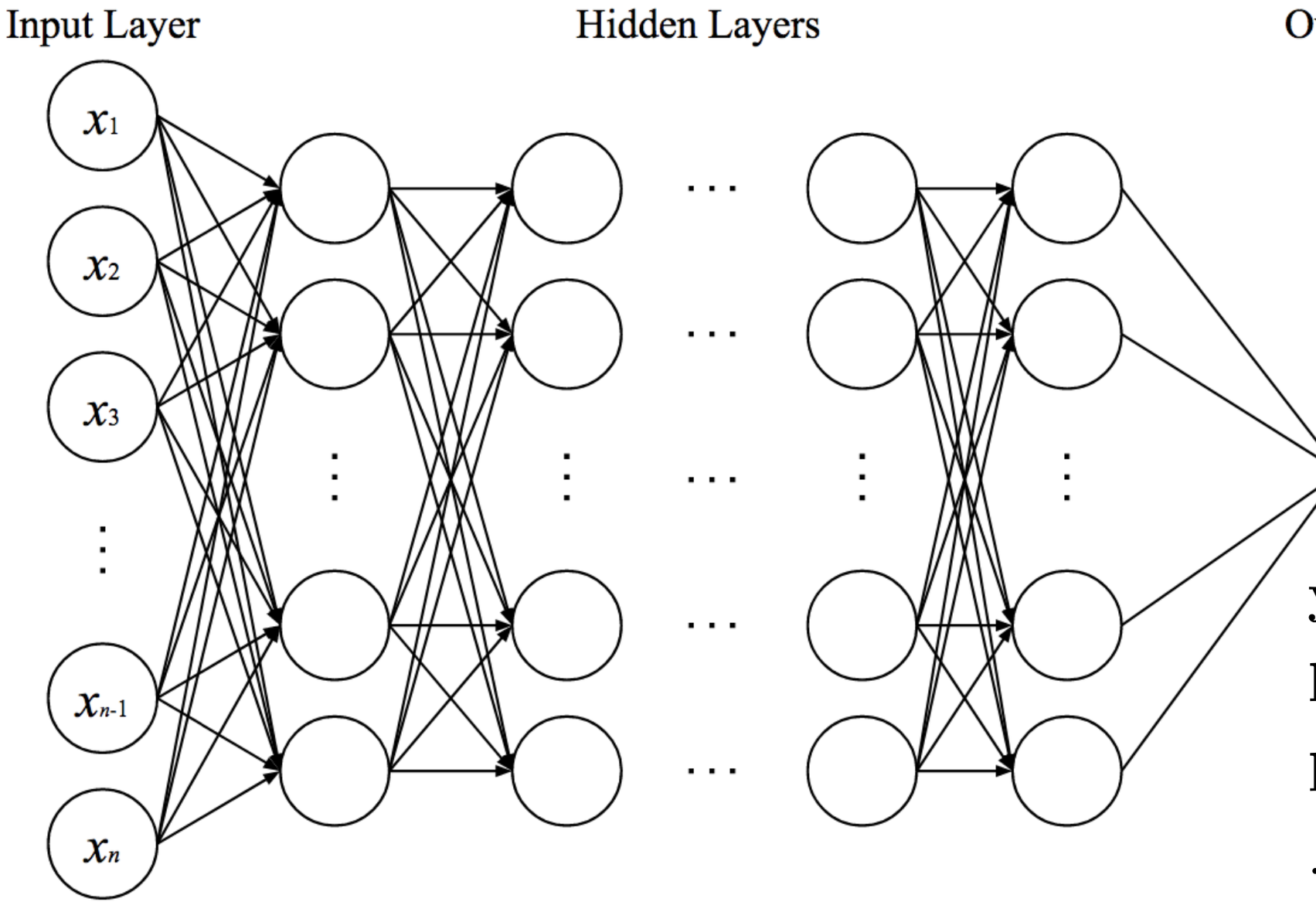
$$NN_{MLP(1)}(\mathbf{x}) = g(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2}$$

$$NN_{MLP(2)}(\mathbf{x}) = g_2(g_1(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2})\mathbf{W^3} + \mathbf{b^3}$$

$$NN_{MLP(1)}(\mathbf{x}) = g(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2}$$
$$NN_{MLP(2)}(\mathbf{x}) = g_2(g_1(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2})\mathbf{W^3} + \mathbf{b^3}$$

$$\mathbf{y} = NN_{MLP(2)}(\mathbf{x})$$
$$\mathbf{h^1} = g_1(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})$$
$$\mathbf{h^2} = g_2(\mathbf{h^1}\mathbf{W^2} + \mathbf{b^2})$$
$$\mathbf{y} = \mathbf{h^2}\mathbf{W^3} + \mathbf{b^3}$$

Feed-forward Neural Network with *k* Hidden Layers



$$\mathbf{y} = NN_{MLP(2)}(\mathbf{x})$$

$$\mathbf{h^1} = g_1(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})$$

$$\mathbf{h^2} = g_2(\mathbf{h^1}\mathbf{W^2} + \mathbf{b^2})$$

$$\ldots$$

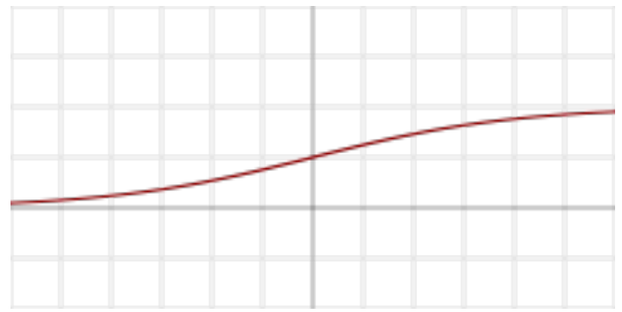
$$\mathbf{h^k} = g_k(\mathbf{h^{k-1}}\mathbf{W^k} + \mathbf{b^k})$$

$$\mathbf{y} = \mathbf{h^k}\mathbf{W^{k+1}} + \mathbf{b^k}$$

Activation Function

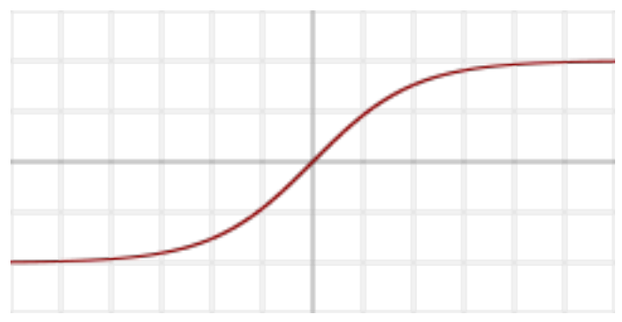
- Sigmoid

$$\frac{1}{1+e^{-x}} \in [0,1]$$



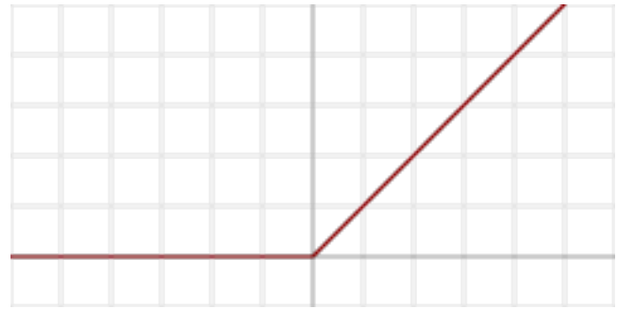
- Hyperbolic tangent (tanh)

$$\frac{e^{2x}-1}{e^{2x}+1} \in [-1,1]$$



- Rectifier (relu)

$$\begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



https://en.wikipedia.org/wiki/Activation_function

Choice of Activation Functions

- Sigmoid
  - Vanish gradient (Very small gradient as x larger/smaller)
  - Suitable for output layer (Softmax for multiple outputs)
- ReLU
  - Constant gradient when $x > 0$
  - Faster to convergence, better performance
  - Suitable for hidden layers

Output Transformation

- The common output transformation function for multi-class models.

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \qquad \text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

- The result is a vector of non-negative real numbers that sum to one, making it a discrete probability distribution over *k* possible outcomes.

- $NN_{Perceptron}$ with the softmax output function:

  - Multinomial logistic regression model, also known as a maximum-entropy classifier.

Minibatch Stochastic Gradient Descent

Until convergence

Sample a batch $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_m, y_m)\}$ of m examples from the training set

$g = 0$

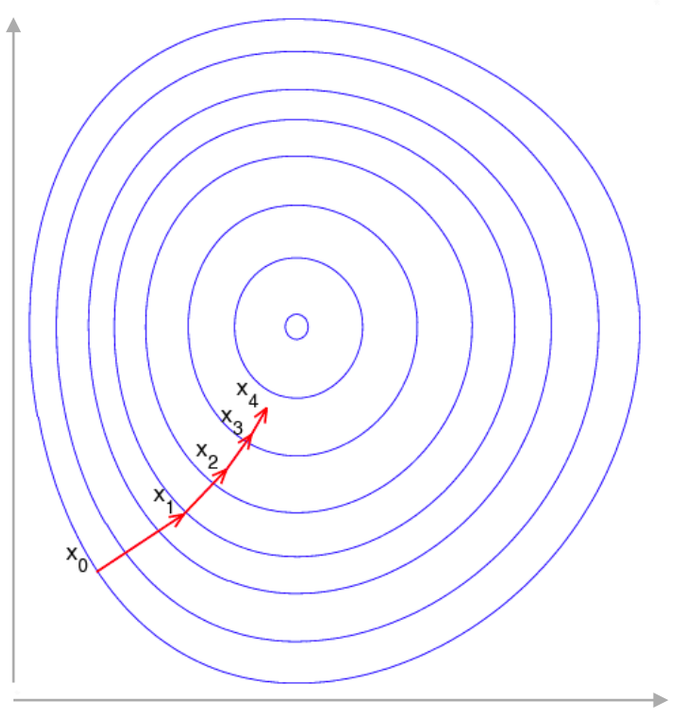
for each sample (x_i, y_i) in the batch:

Compute the loss l of $f(x_i, \theta)$ and y_i

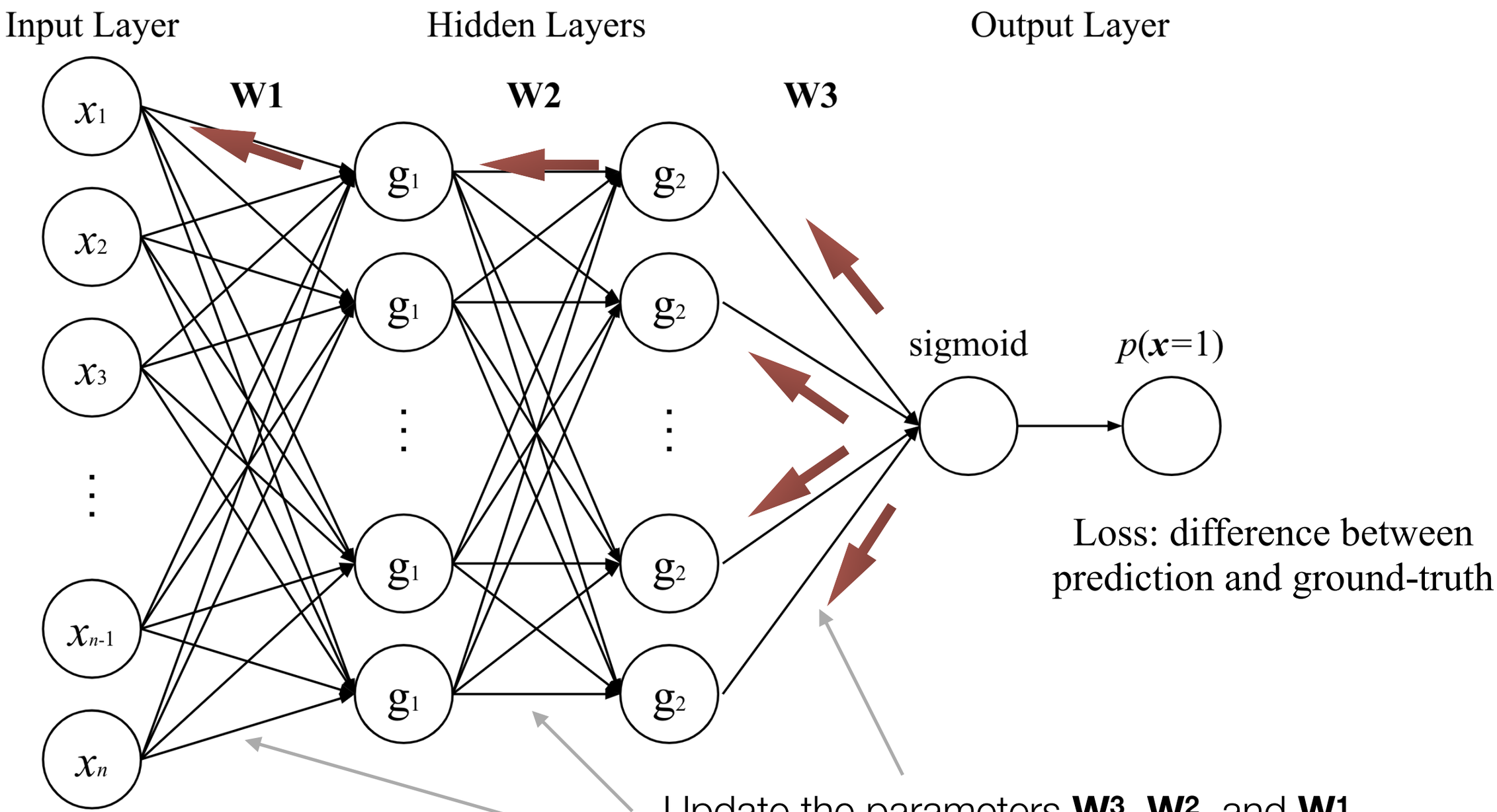
$g = g +$ gradients of l/m

$\theta = \theta + \gamma g$

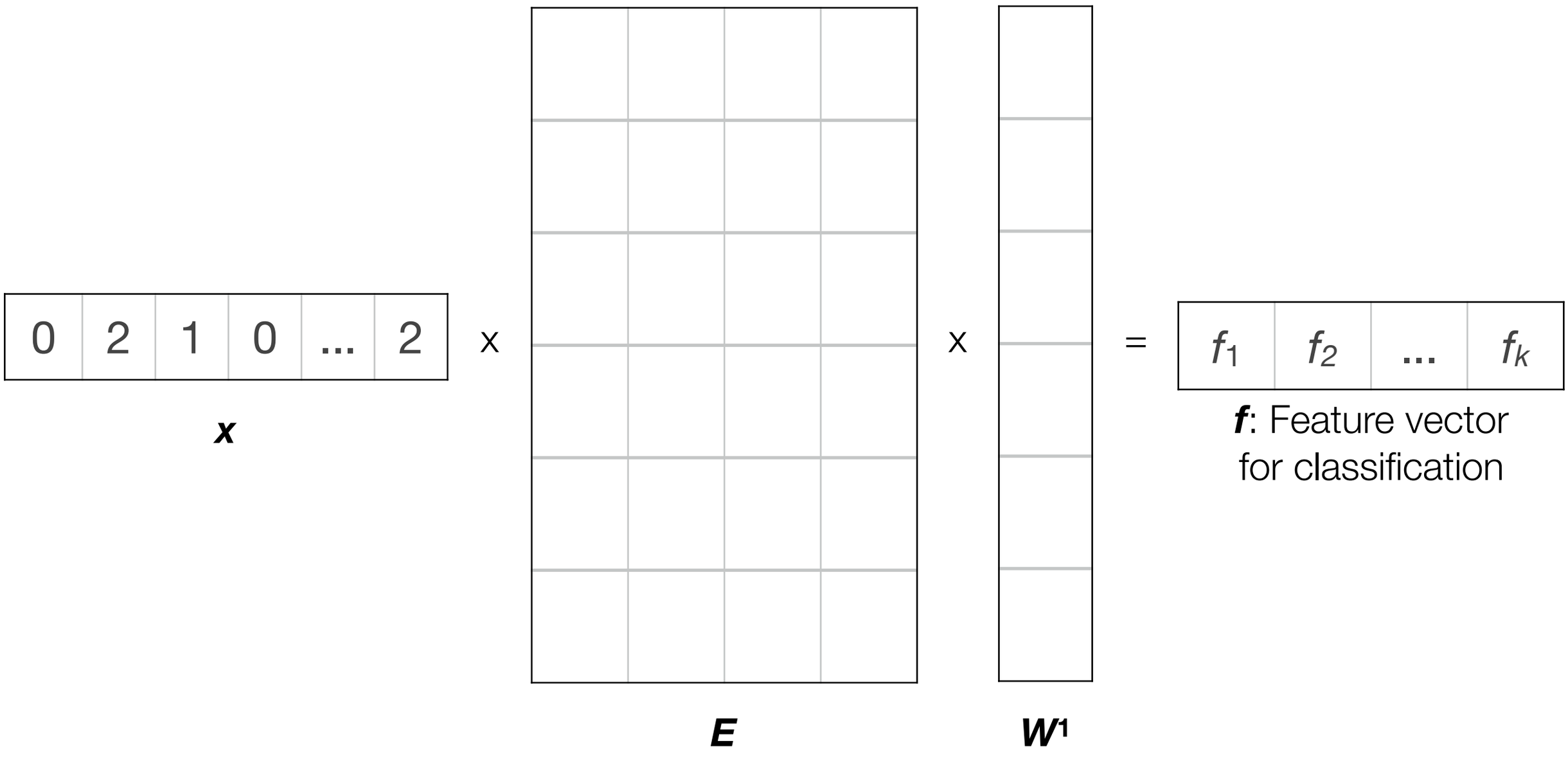
return θ



Back-propagation



Word Embedding NN for Bag of Words



$$\bar{y} = \text{sigmoid}(\boldsymbol{f}\boldsymbol{W}^{\boldsymbol{2}}) = \text{sigmoid}(\boldsymbol{x}\boldsymbol{W}^{\boldsymbol{1}}\boldsymbol{E}\boldsymbol{W}^{\boldsymbol{2}})$$

Parameters to learn

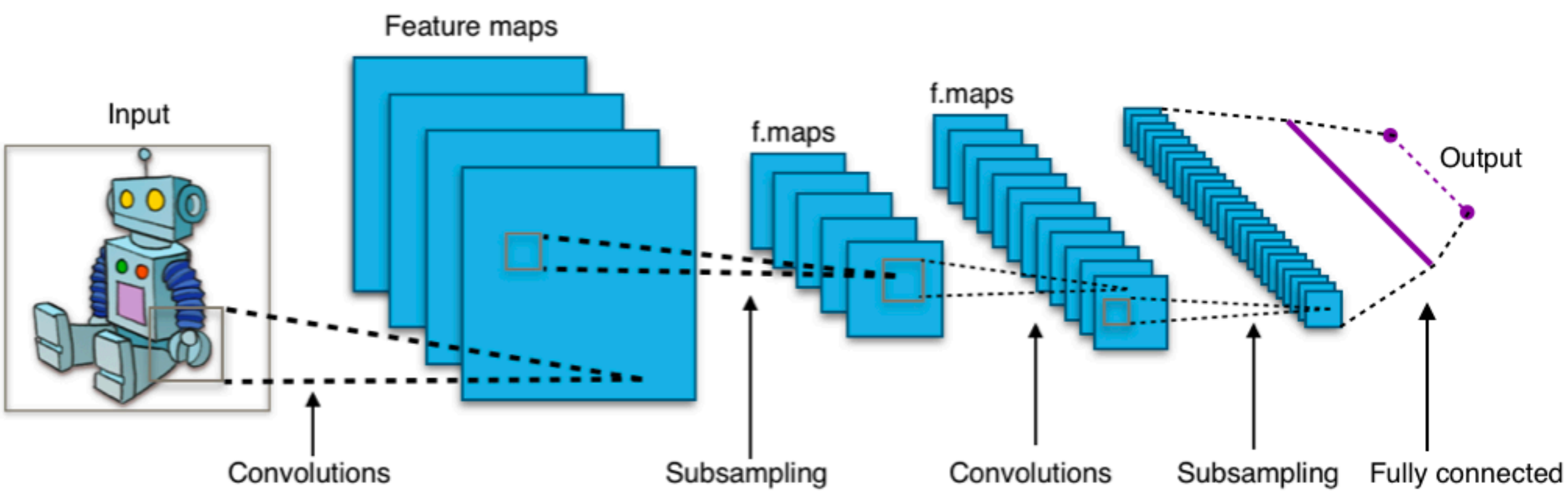
Modeling Sequences

- When dealing with language data, it is very common to work with sequences, such as words (sequences of letters), sentences (sequences of words) and documents.

- Feed-forward neural network, like logistic regression models and naive Bayes models, is hardly to capture the word order information.

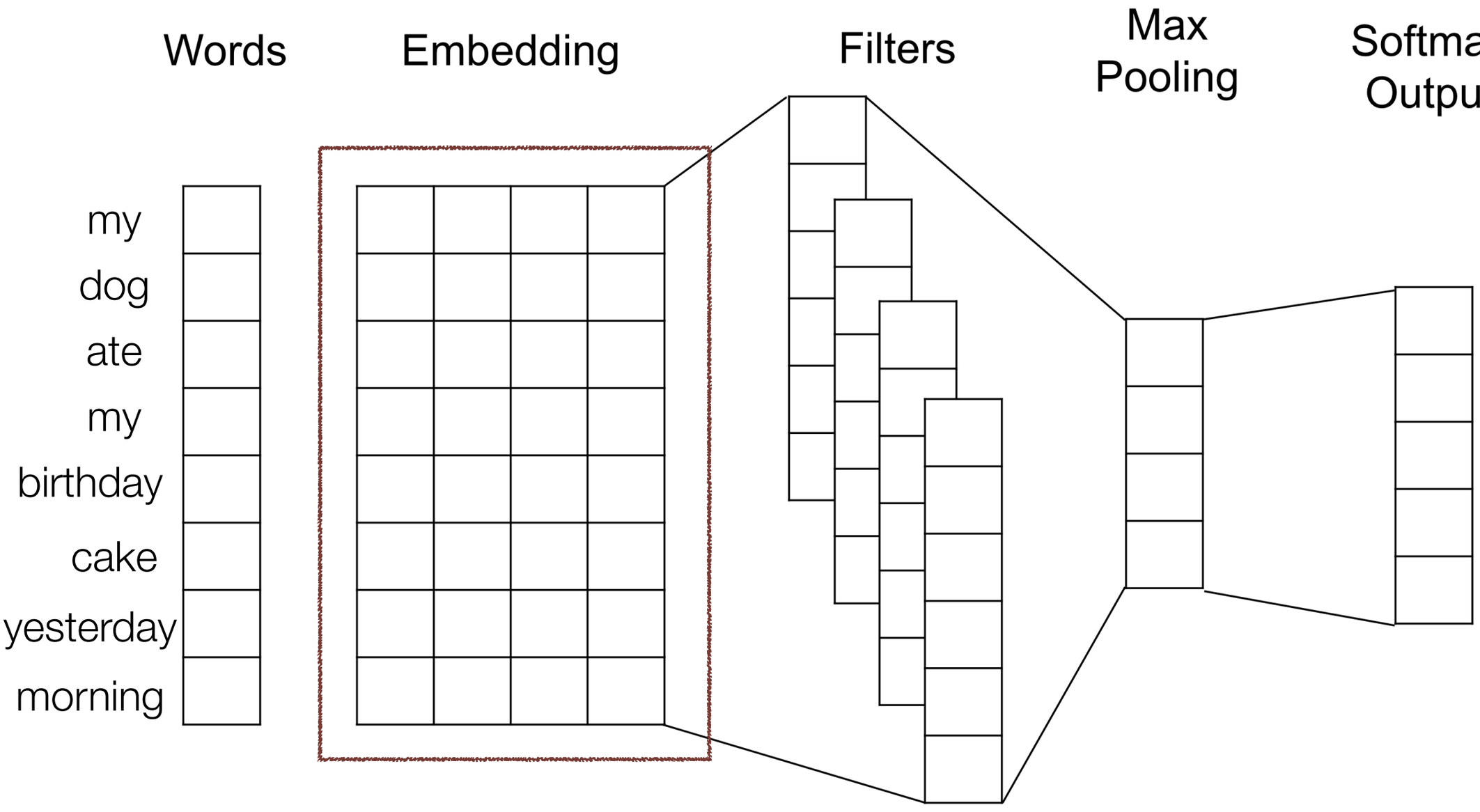
Convolutional Neural Networks

- 2-D (grid) CNN is very powerful and popular in computer vision and image recognition.



CNN for Sentence Classification

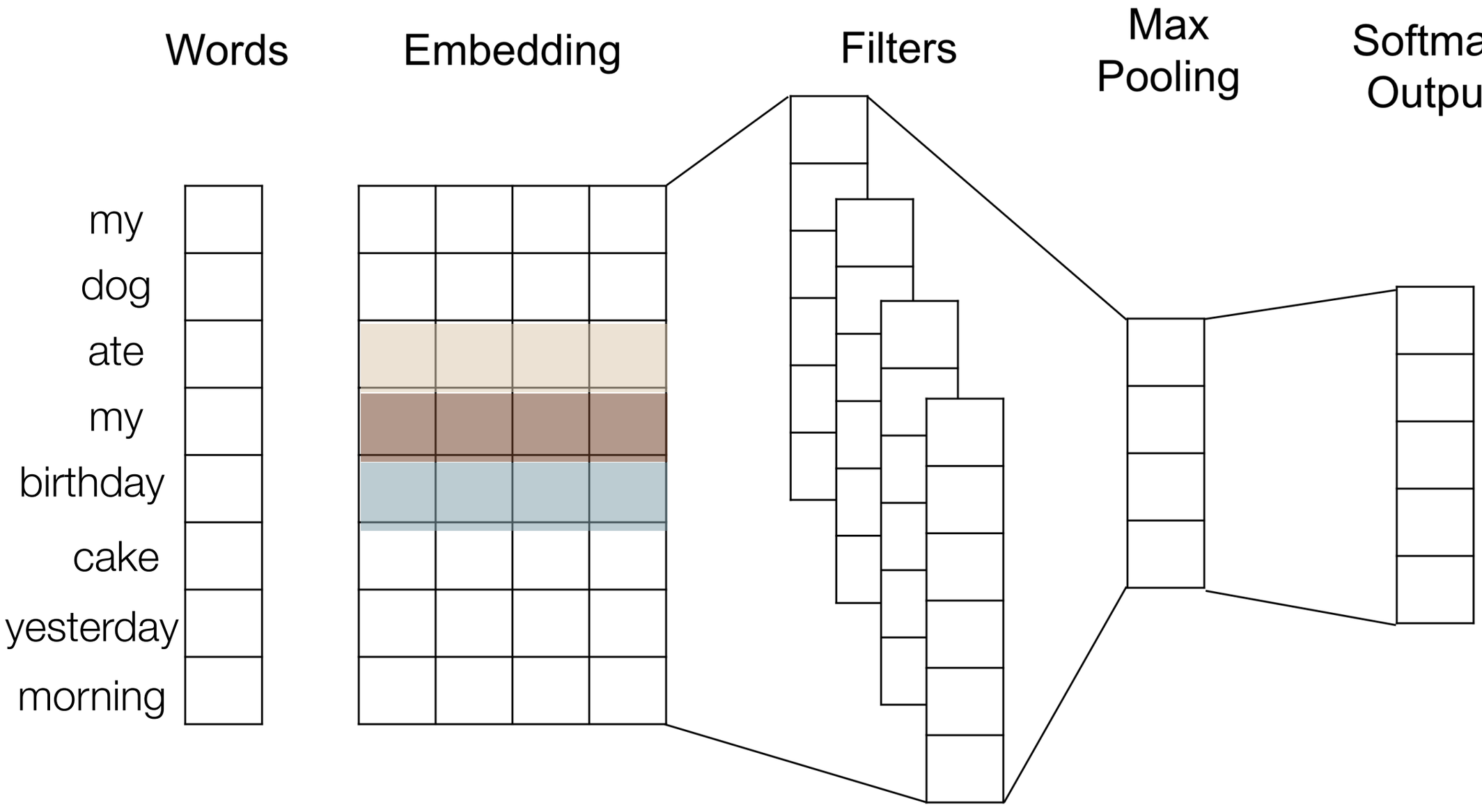
- 1-D (sequence) CNN is very useful and efficient in NLP



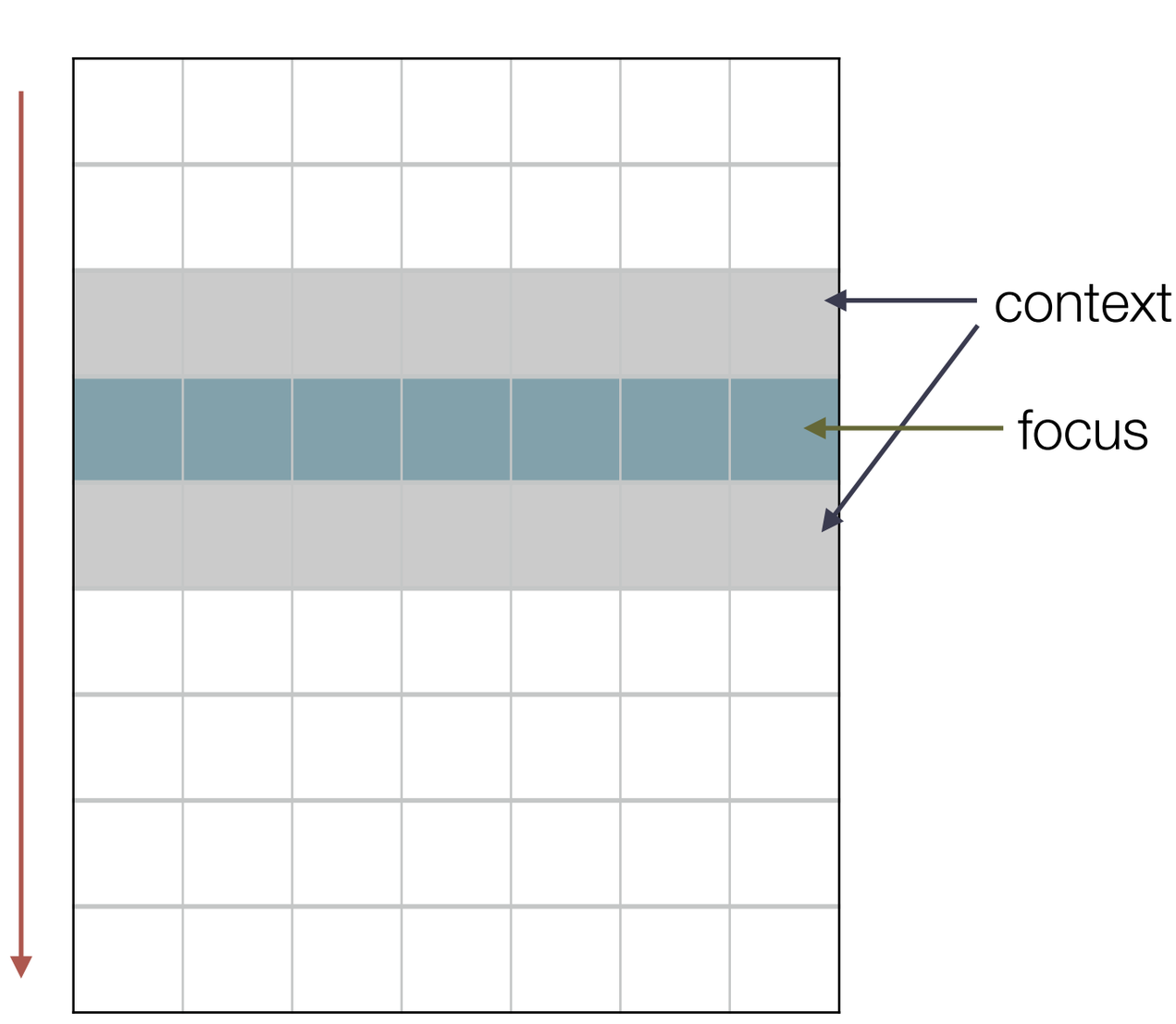
A sentence is represented as a matrix with the dimension of $n * d$,
where n is number of words and d is the dimension of word embeddings

1-D Convolution

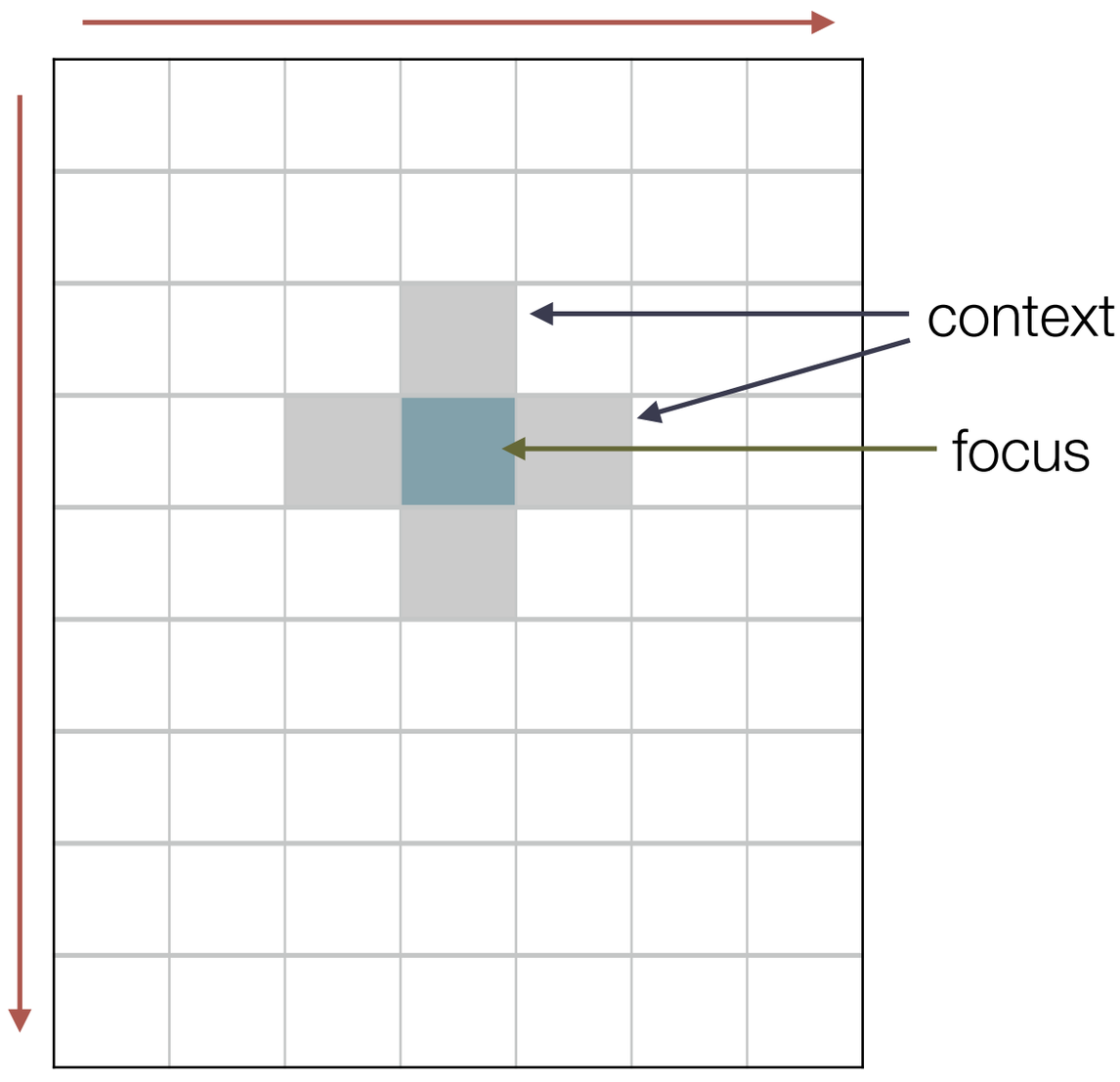
- 1-D convolution layer creates a convolution kernel that is convolved with the layer input over a **single temporal** dimension to produce a tensor of outputs.



1-D Convolution vs 2-D Convolution

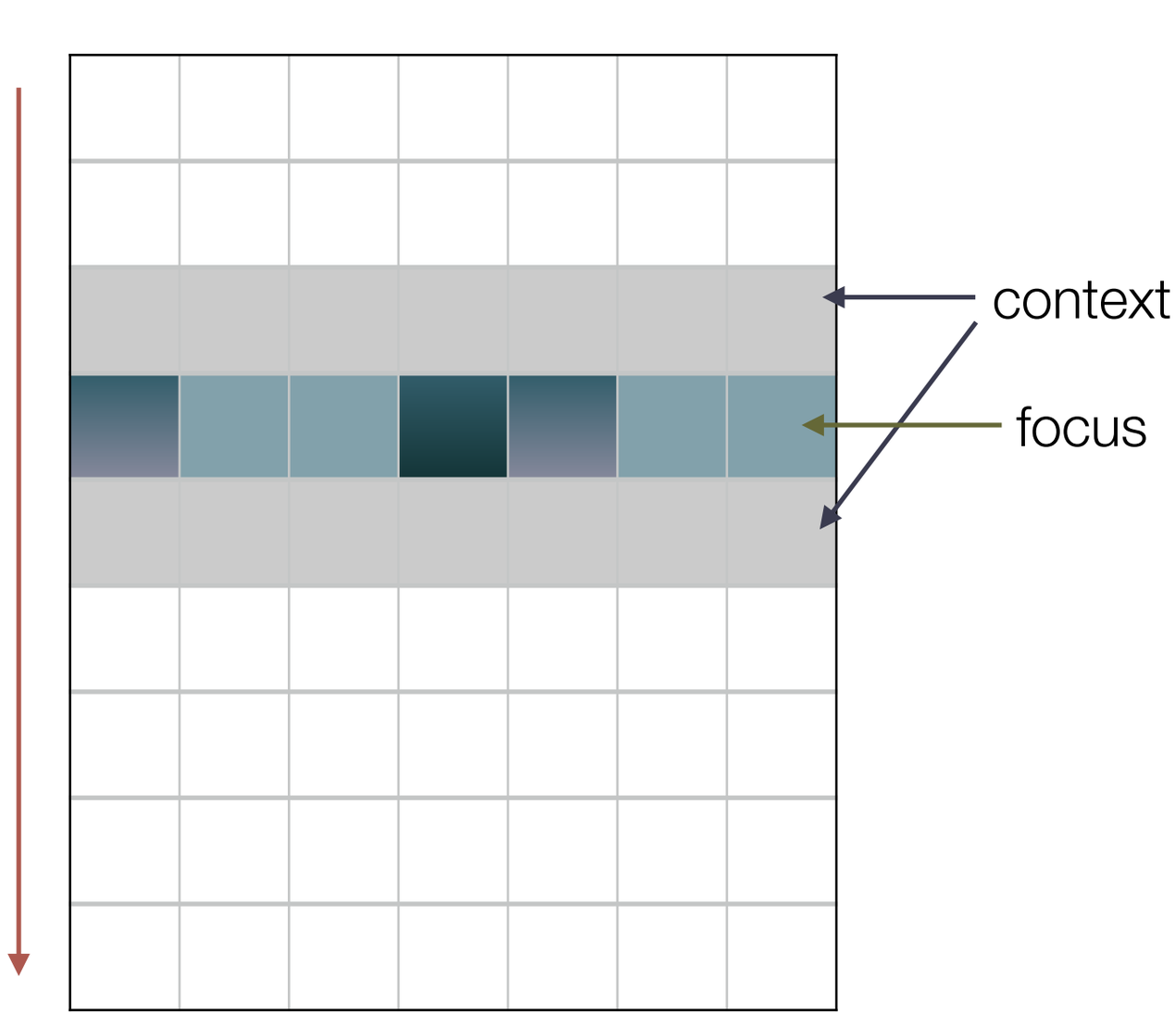


1-D Convolution



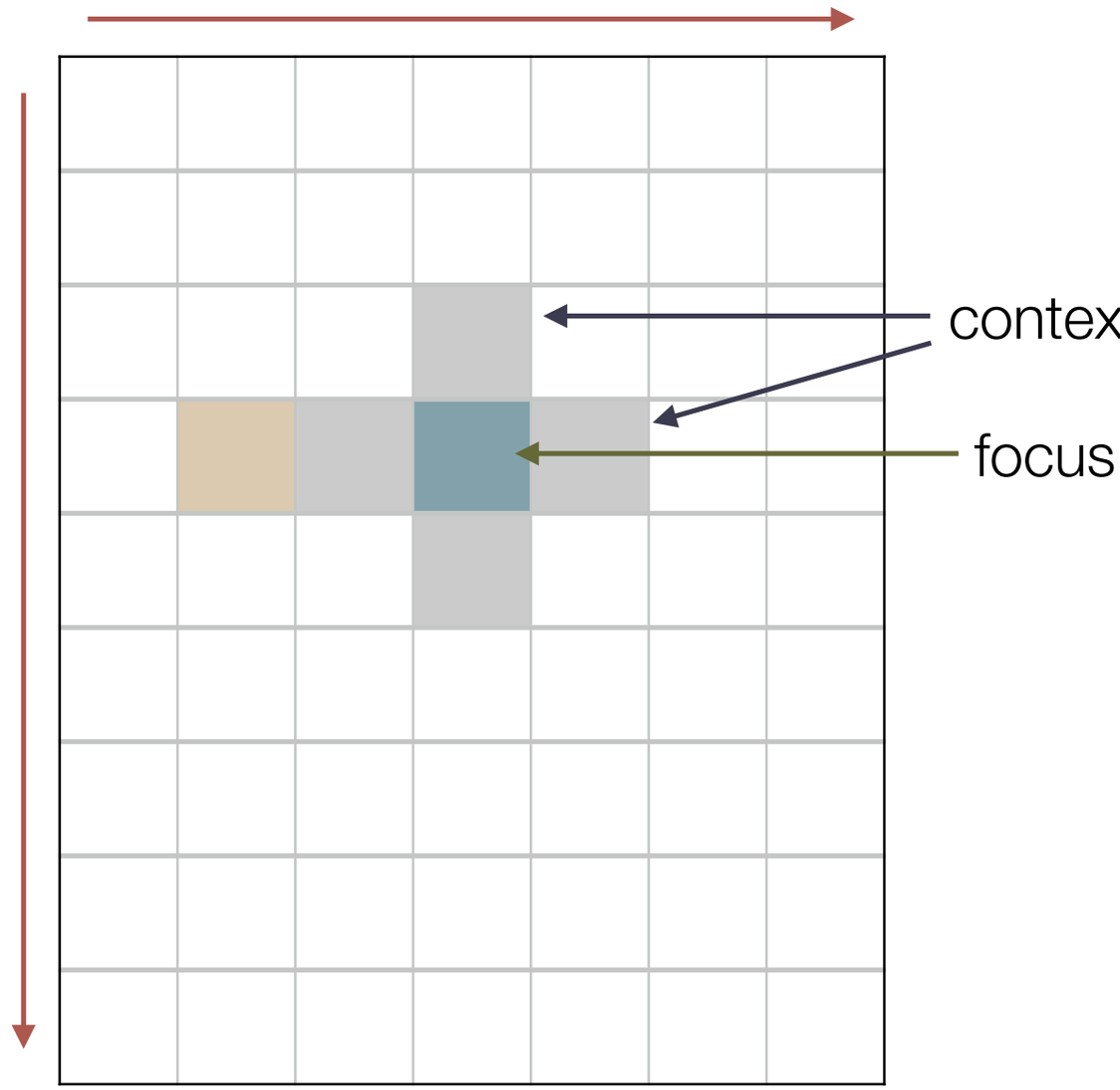
2-D Convolution

1-D Convolution vs 2-D Convolution



1-D Convolution

For the *i*th element in a word embedding, the *i*+1th element is not more important than the *i*+100th element

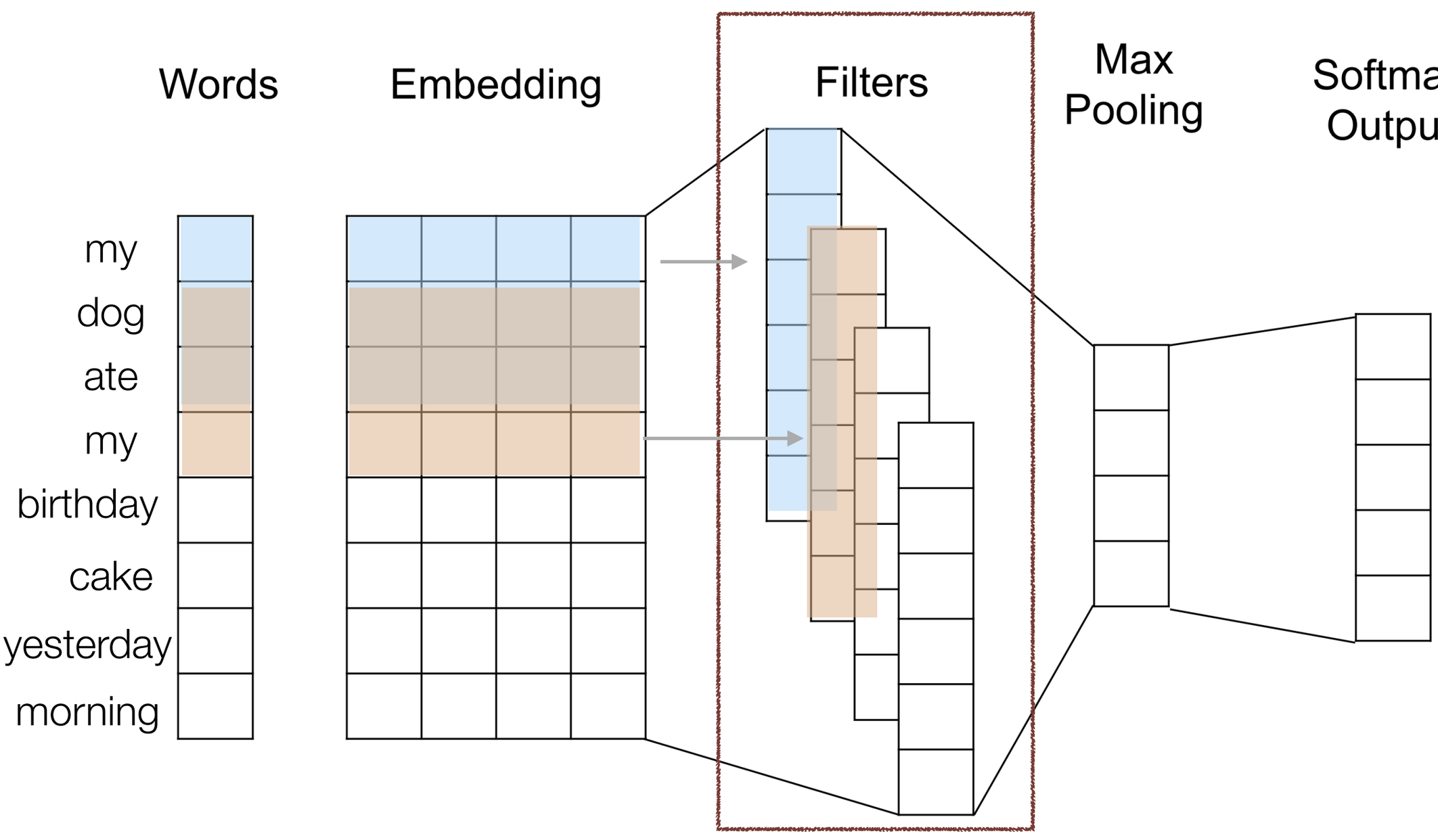


2-D Convolution

For an image, the neighboring pixels are considered more important than distant ones.

Filtering

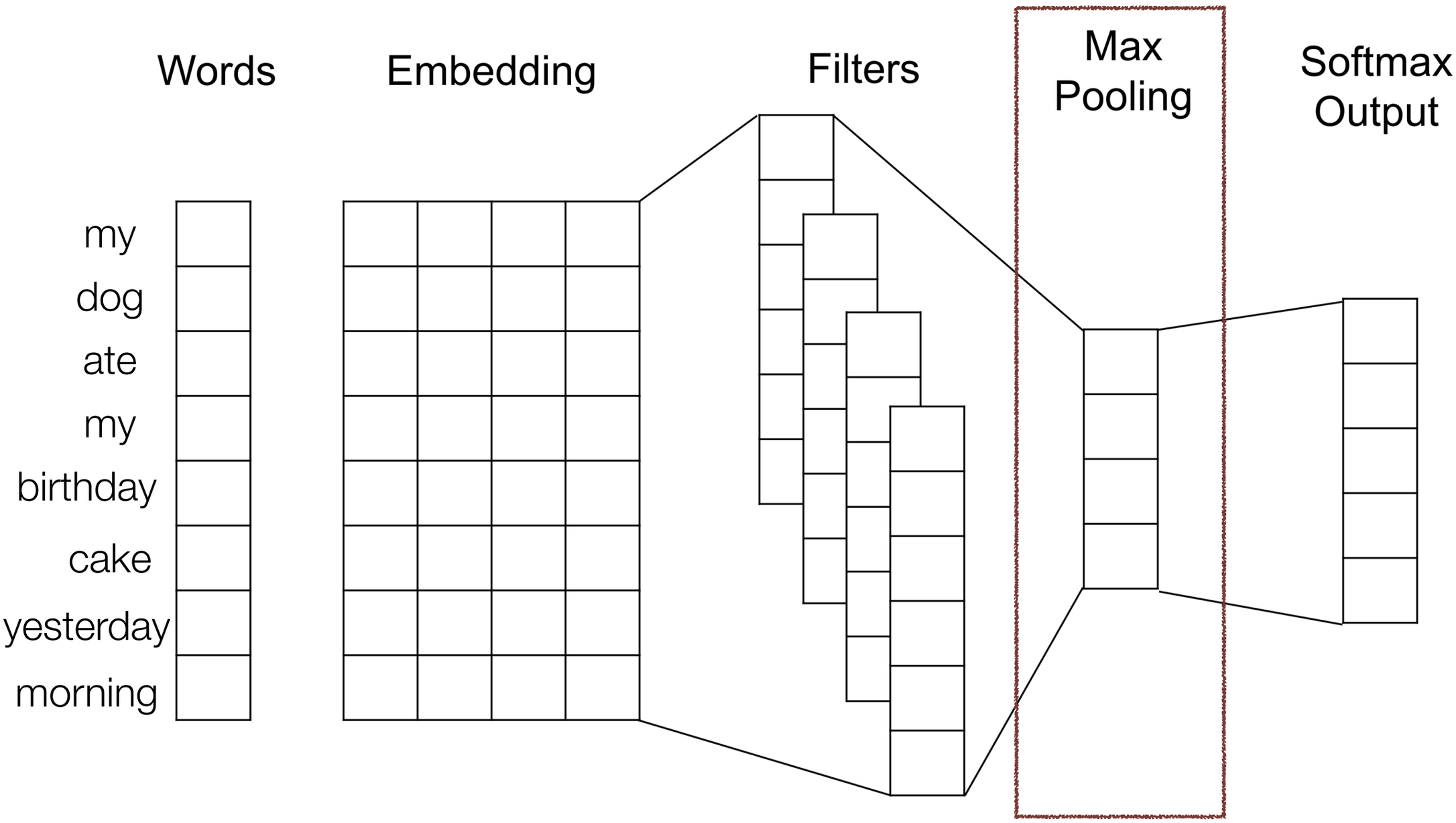
N-grams of word embeddings converted to dense vectors



Extracting neighboring word embeddings and converting to filtered representations

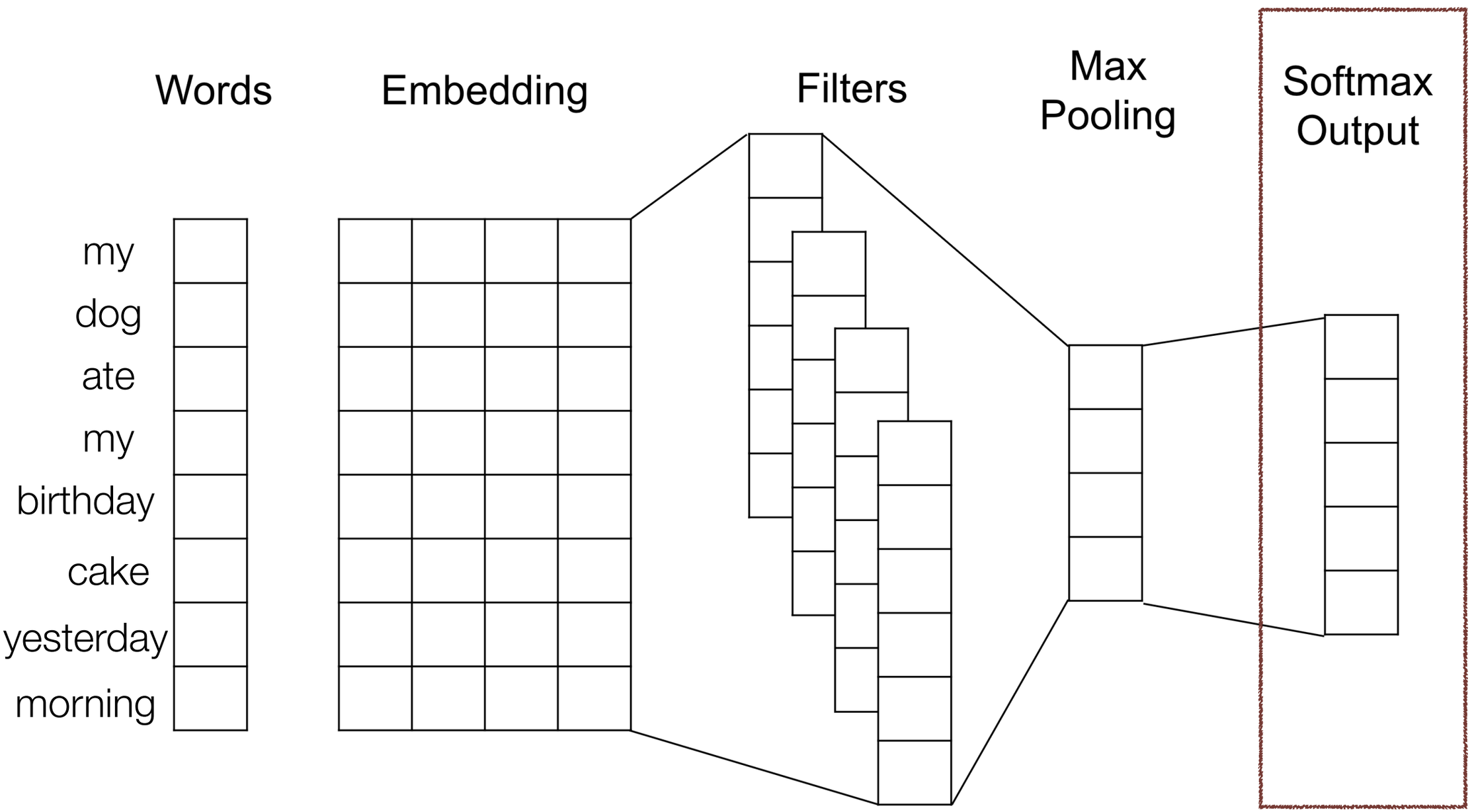
Pooling

Perform classification based on the important information in all n-grams' representations



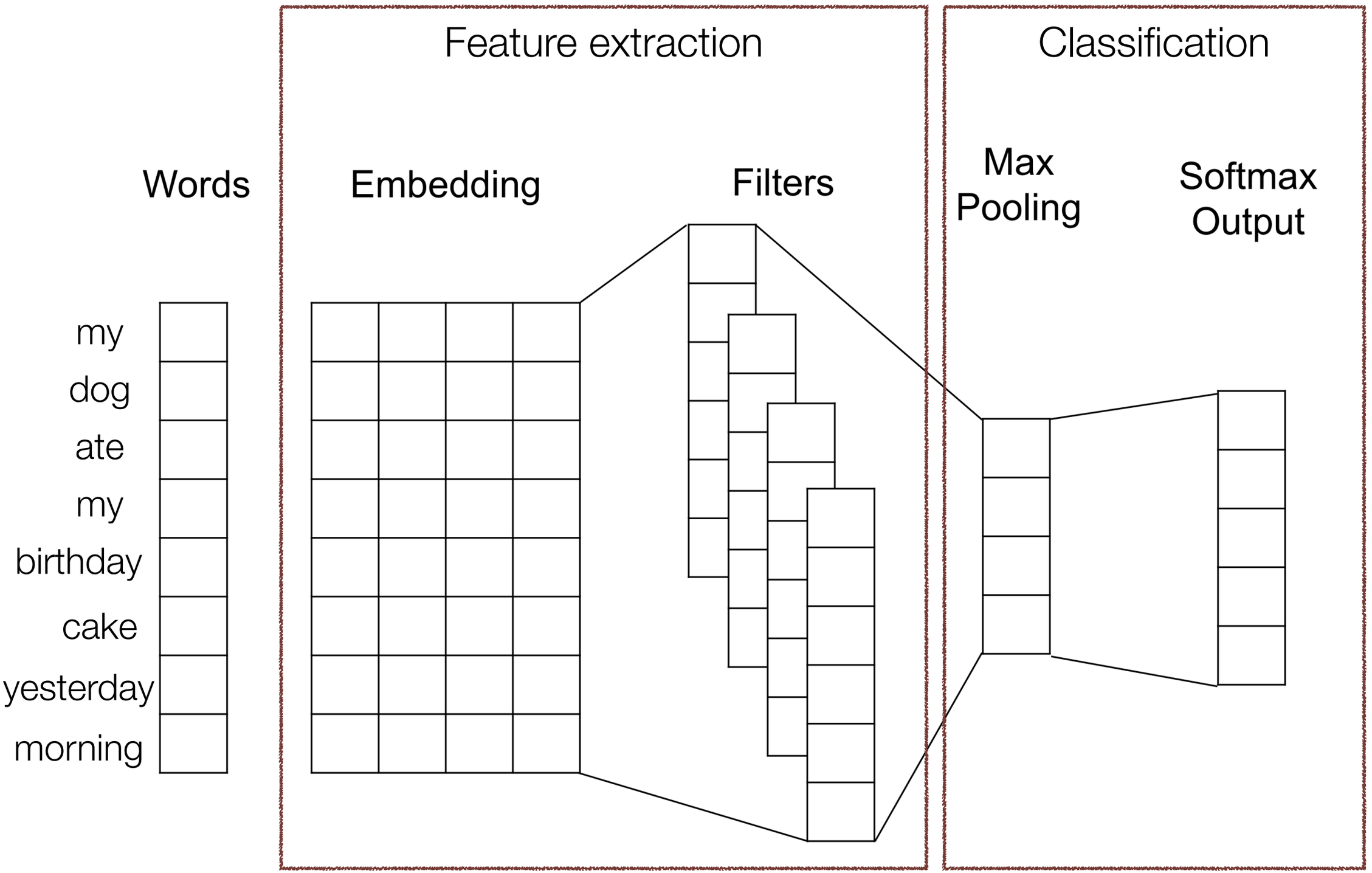
Max-pooling operation takes the max over each dimension, resulting in a pooled vector

Output



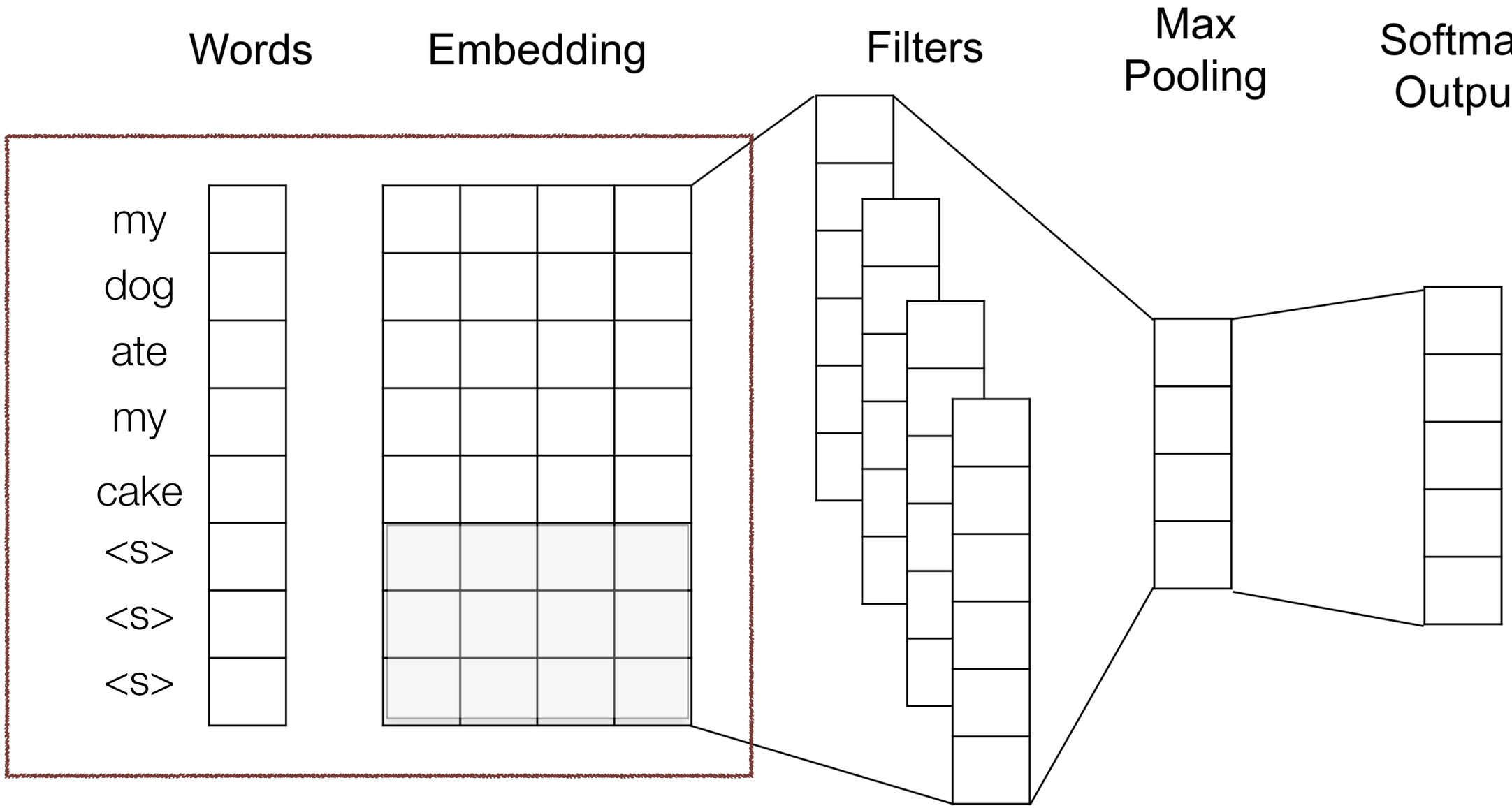
The output layer finally generates the probability of each category based on the pooled vector

Compared with Traditional Text Classifier



Padding

- The max length of input sentences is usually predefined as a hyper-parameter.
- Padding is applied to the input shorter than the max length.



Modeling Longer Dependency

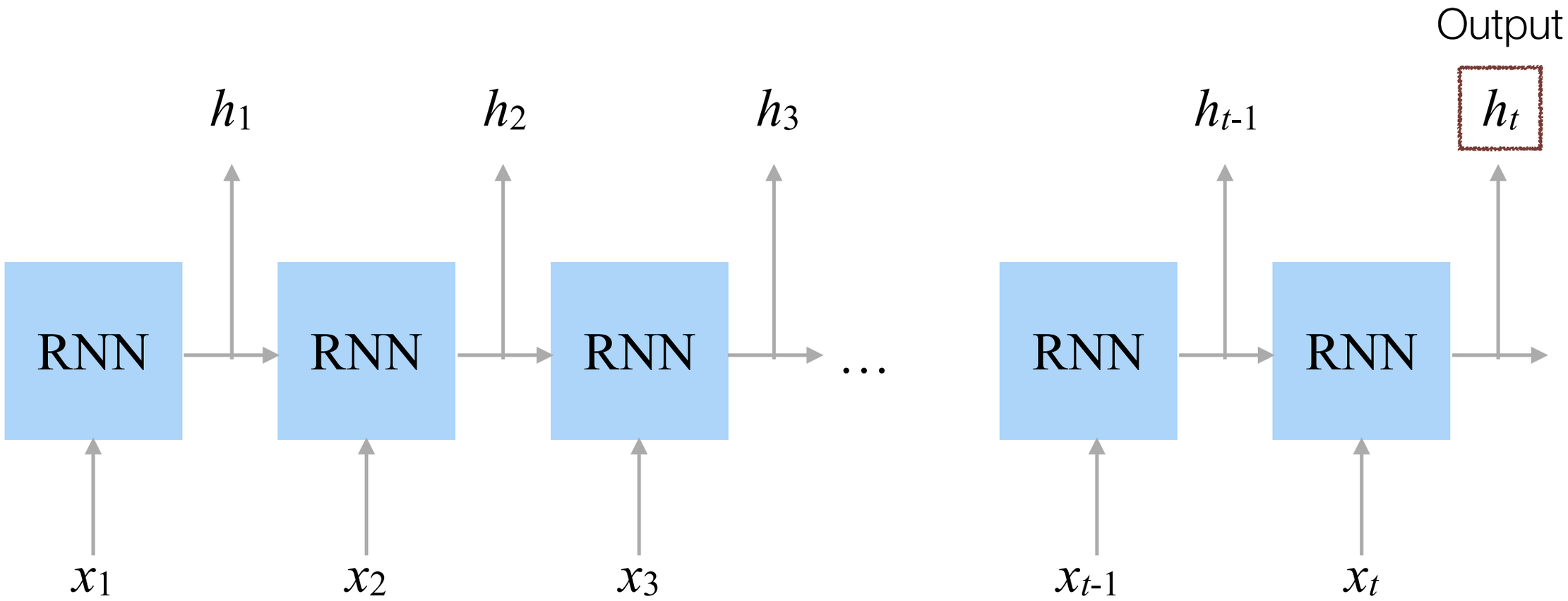
- The convolutional networks also allow encoding a sequence into a fixed size vector.

- While representations derived from convolutional networks are an improvement above the CBOW representation as they offer some sensitivity to word order, their order sensitivity is restricted to mostly local patterns, and disregards the order of patterns that are far apart in the sequence.

- Recurrent neural networks (RNNs) (Elman, 1990) allow representing arbitrarily sized structured inputs in a fixed-size vector, while paying attention to the structured properties of the input.

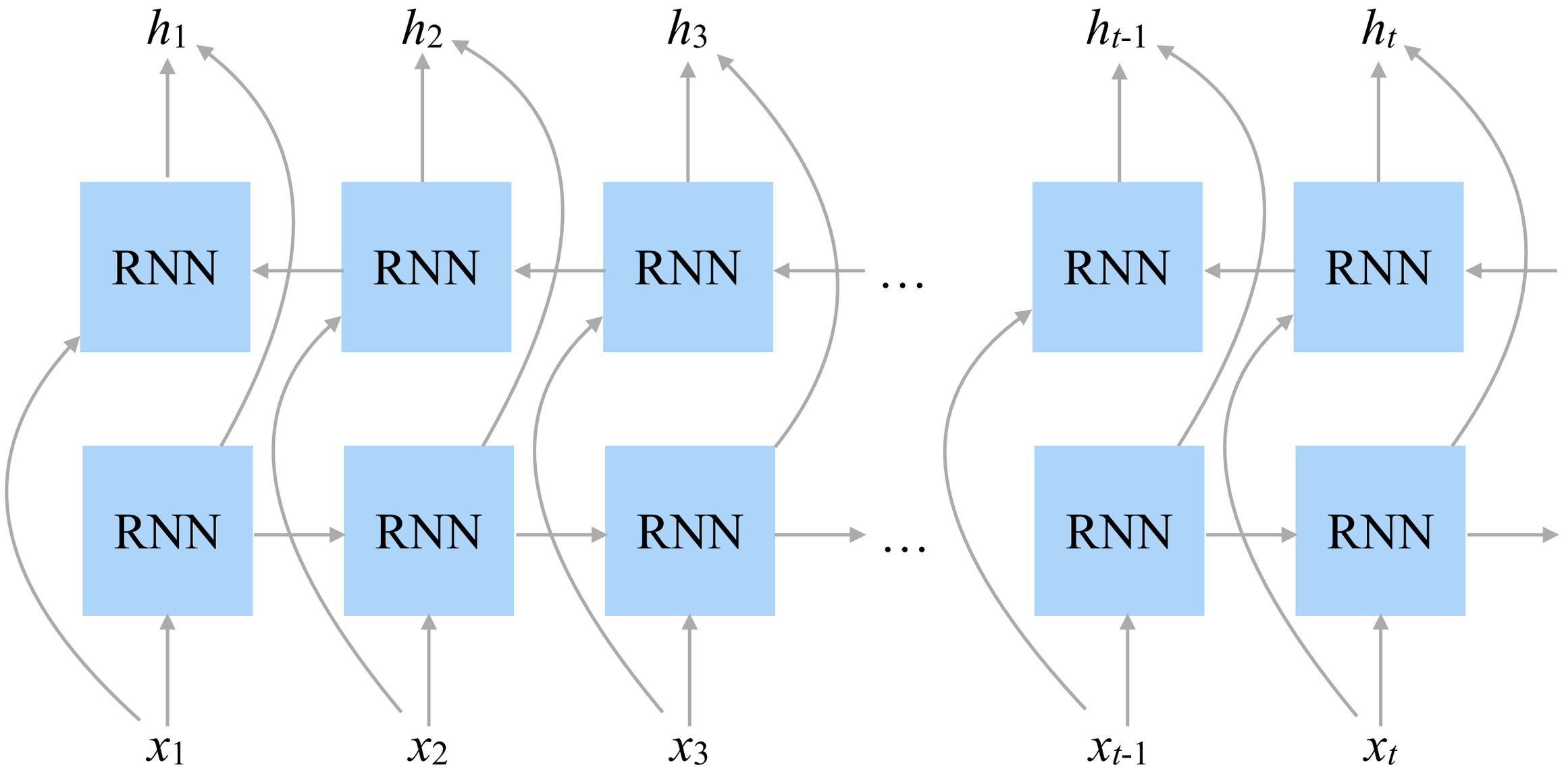
Recurrent Neural Network

- The output of step t-1 is passed to next step
- Unlike HMM, the output of step t is determined by not only the information of x_t and y_{t-1}, but also the information from 1, 2, 3, …, t-2
- No Markov assumption



Bi-Directional RNN

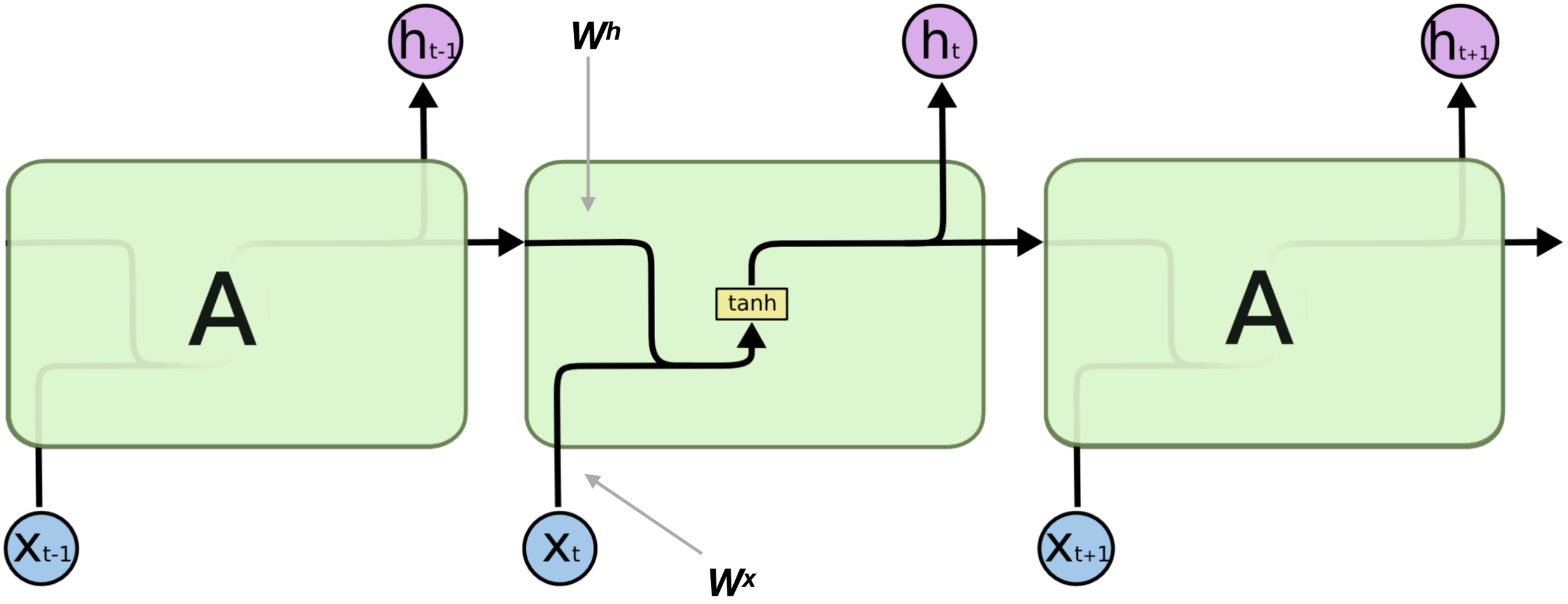
- Adding additional RNN in the opposite direction.
- Take both forward/backward contextual information at the same time.



RNN Architectures

- Simple RNN
- Long Short-term Memory (LSTM)
- Gated Recurrent Unit (GRU)

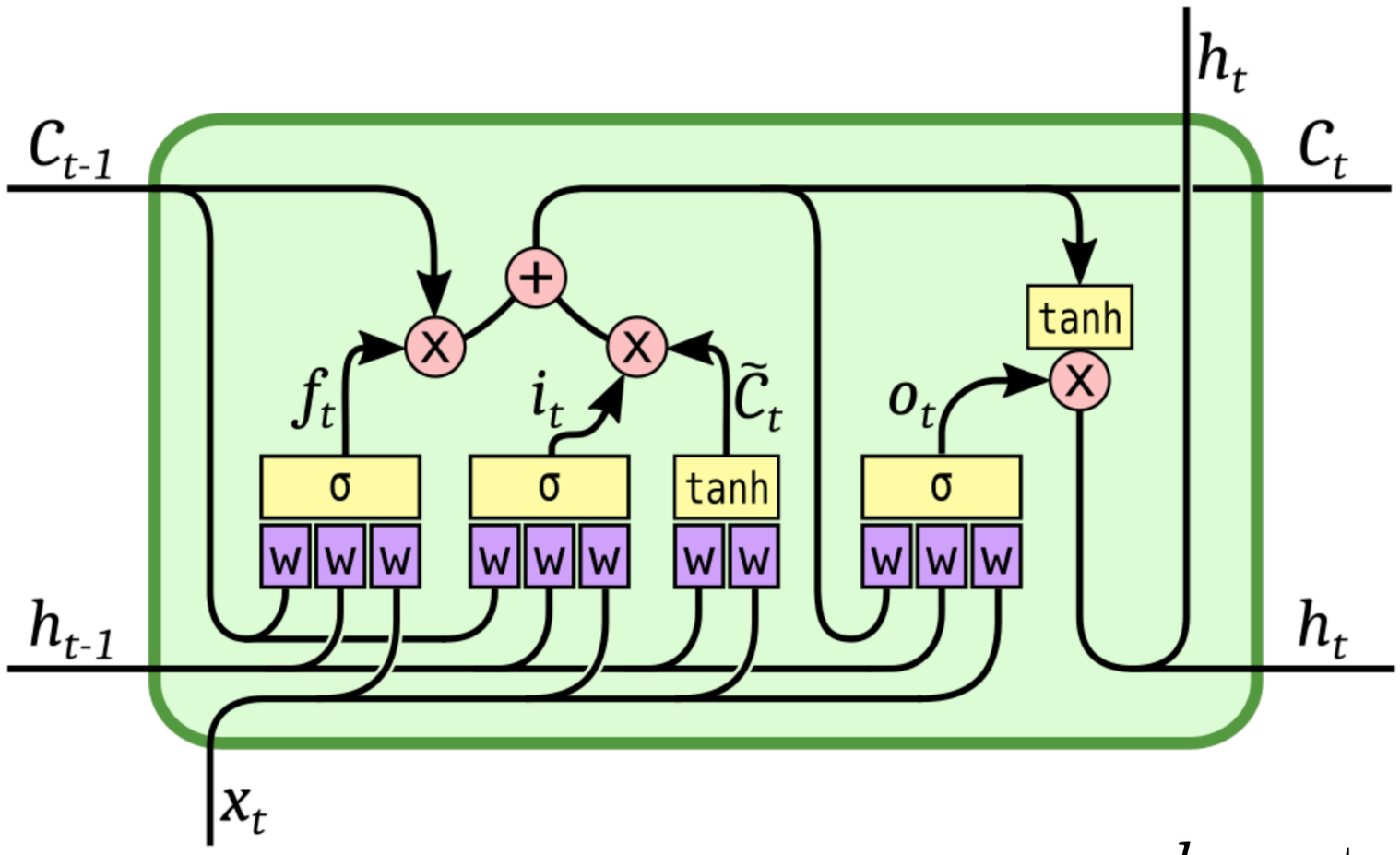
Simple RNN



$$h_t = R_{\text{SRNN}}(h_{t-1}, x_t) = \tanh(h_{t-1}W^h + x_tW^x + b)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-term Memory (LSTM)



More parameters are used to keep the information of earlier steps

Parameters are 4 times as Simple RNN

https://kvitajakub.github.io/2016/04/14/rnn-diagrams/

$$h_t = \tanh(C_t) \times o_t$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

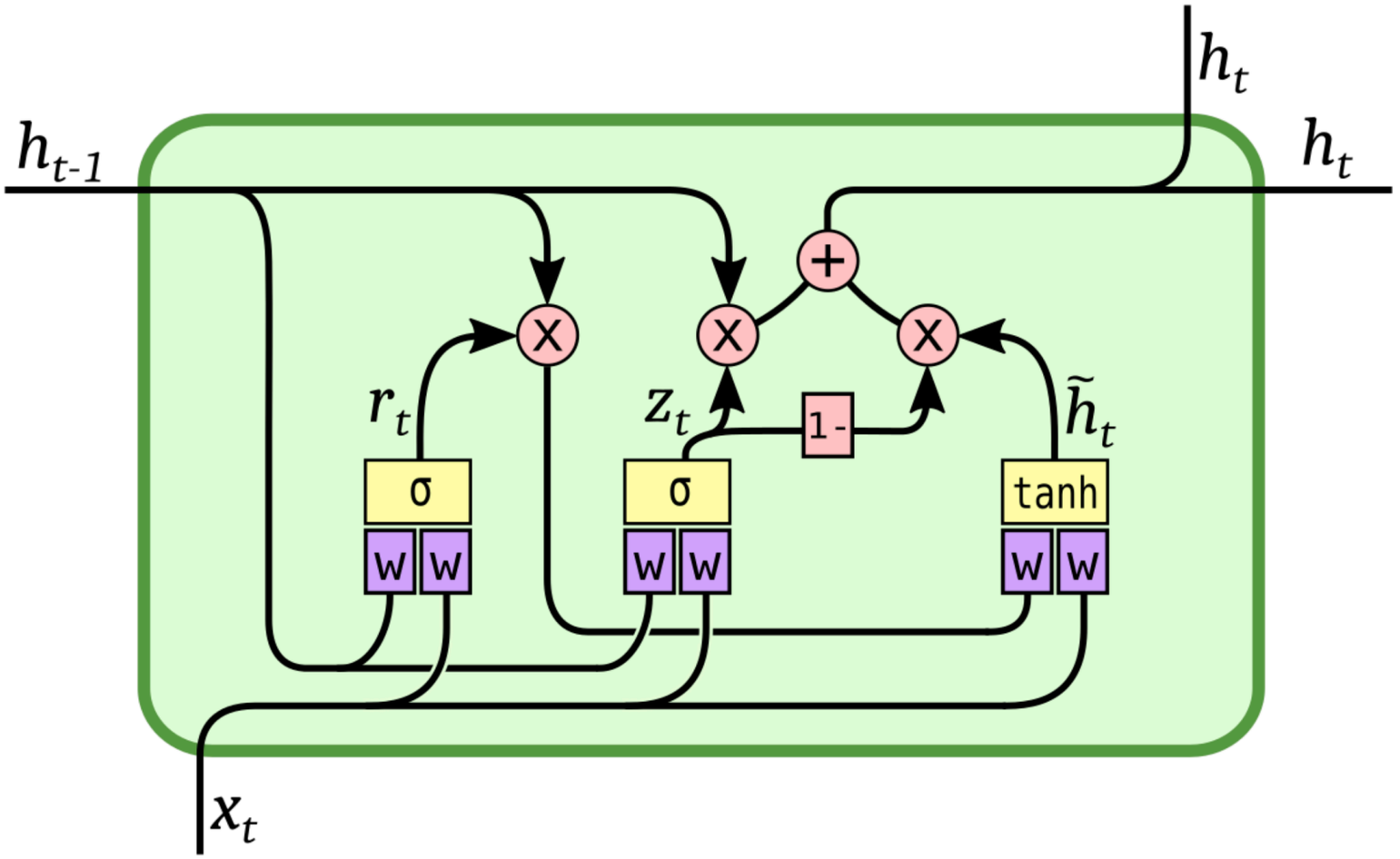
$$o_t = \text{sigmoid}(x_t W^{xo} + h_{t-1} W^{ho})$$

$$f_t = \text{sigmoid}(x_t W^{xf} + h_{t-1} W^{hf})$$

$$i_t = \text{sigmoid}(x_t W^{xi} + h_{t-1} W^{hi})$$

$$\tilde{C}_t = \tanh(x_t W^{xc} + h_{t-1} W^{hc})$$

Gated Recurrent Unit



Simpler and faster than LSTM with comparable performance

Widely-used these days

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t$$

$$z_t = \text{sigmoid}(x_t W^{xz} + h_{t-1} W^{hz})$$

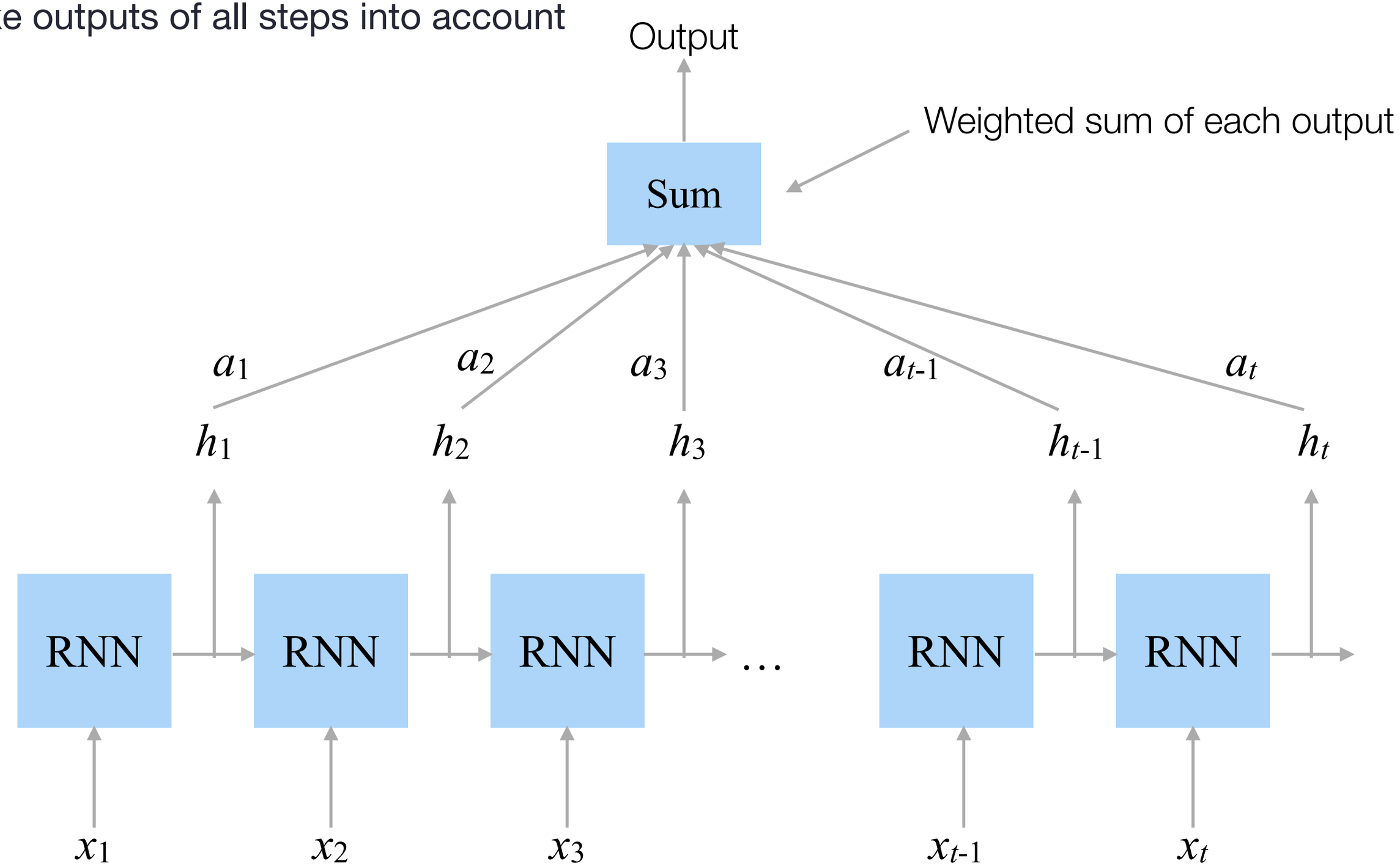
$$r_t = \text{sigmoid}(x_t W^{xr} + h_{t-1} W^{hr})$$

$$\tilde{h}_t = \tanh(x_t W^{xh} + (h_{t-1} \cdot r_t) W^{hg})$$

https://kvitajakub.github.io/2016/04/14/rnn-diagrams/

Attention Mechanism

Take outputs of all steps into account



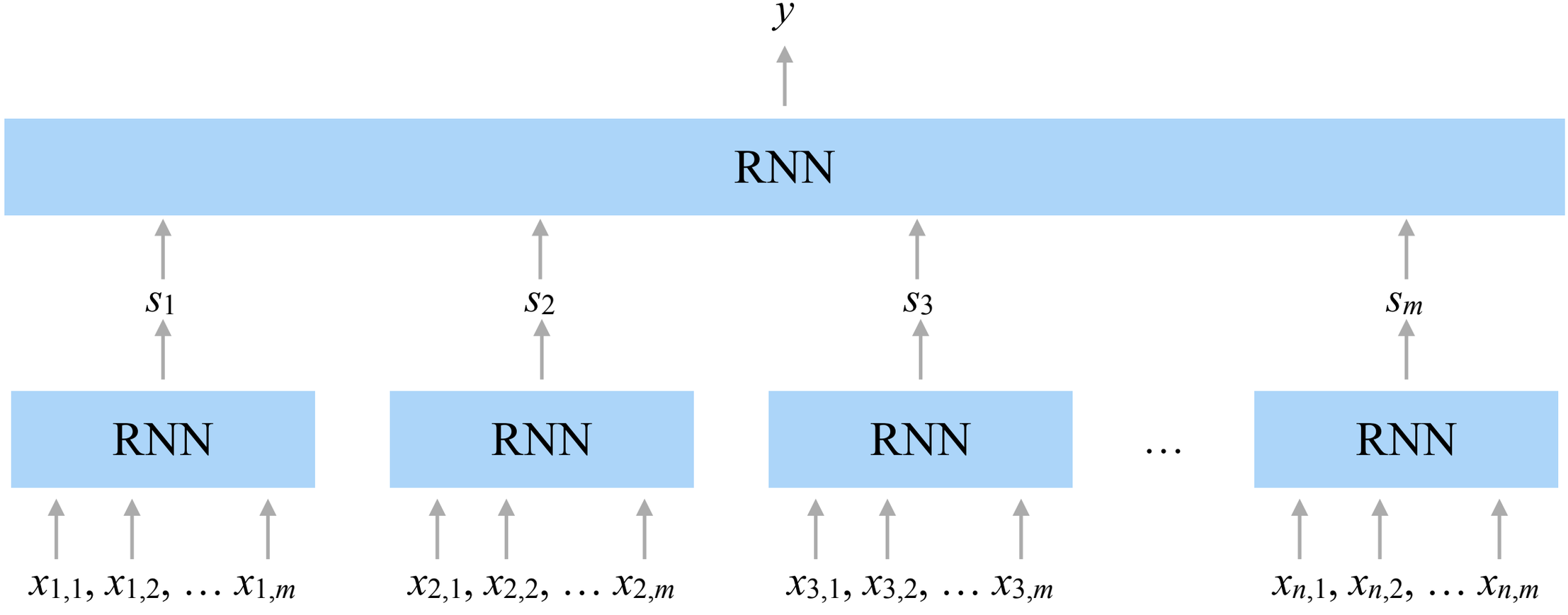
NN Models for Document Classification

- For document classification, a common issue of the RNN or the CNN models are that they do not work well with the very long input sequence.

- A NN model for sentence classification usually has a working window up to 100~200 words.

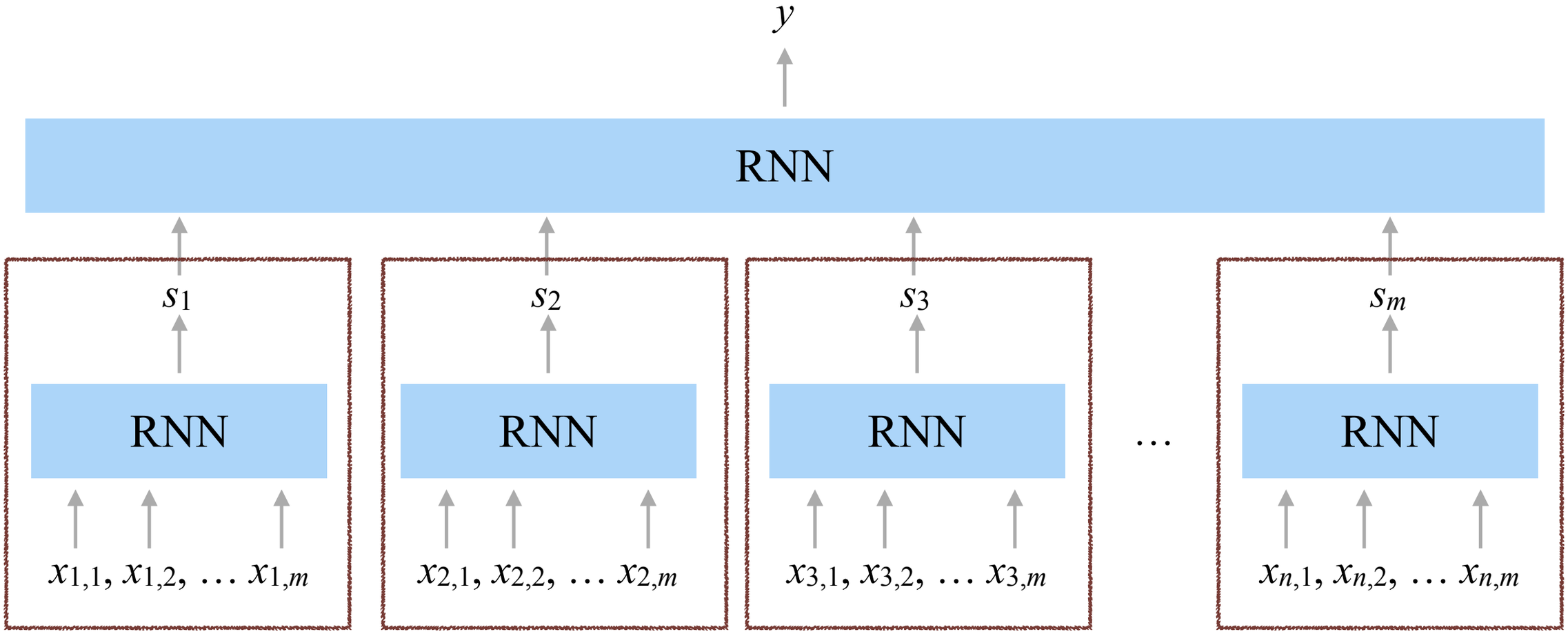
Hierarchical Neural Network Models

- The lower RNN extracts features from raw sentences and represents each sentence as a dense vector s_i.
- The upper RNN extracts the document-level information from all sentences.

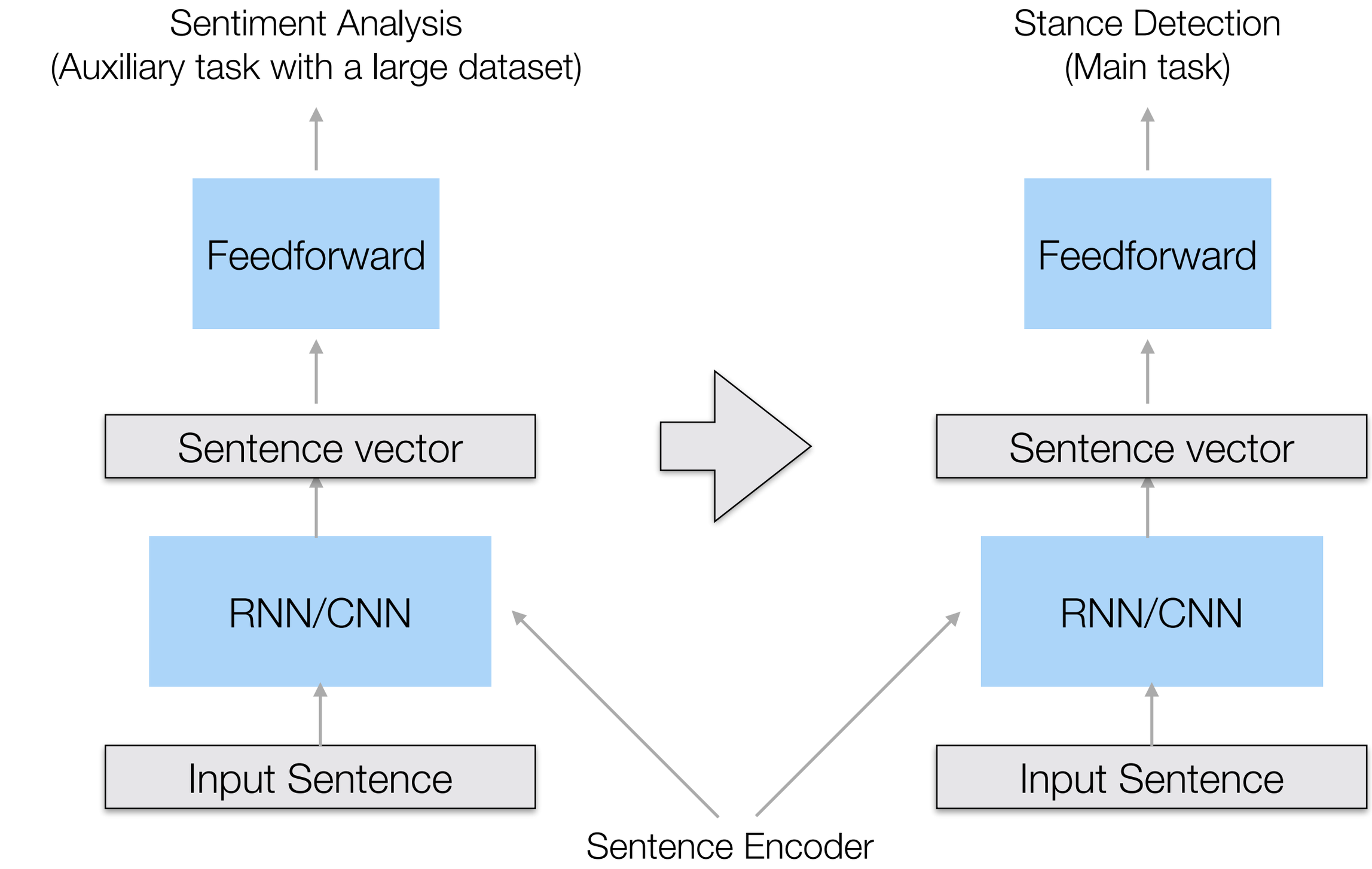


Sentence Encoder

- The role of the lower RNN is the **sentence encoder**, which converts a sequence of words into a dense vector
- Not limited to RNN, CNN also performs well for sentence encoding.



Pre-Trained Sentence Encoder



ELMo

- Different from word embeddings, ELMo word representations are functions of the entire input sentence.
- They are computed on top of two-layer biLMs with character convolutions, as a linear function of the internal network states.
- This setup allows us to do semi-supervised learning, where the biLM is pre-trained at a large scale.
- Easily incorporated into a wide range of existing neural NLP architectures.

Bidirectional Language Model

- Forward language model

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1})$$

- Backward language model

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N)$$

- Bidirectional language model

$$\sum_{k=1}^{N} (\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s))$$

ELMo

- For token t_k, R_k is its L-layer representation.

$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

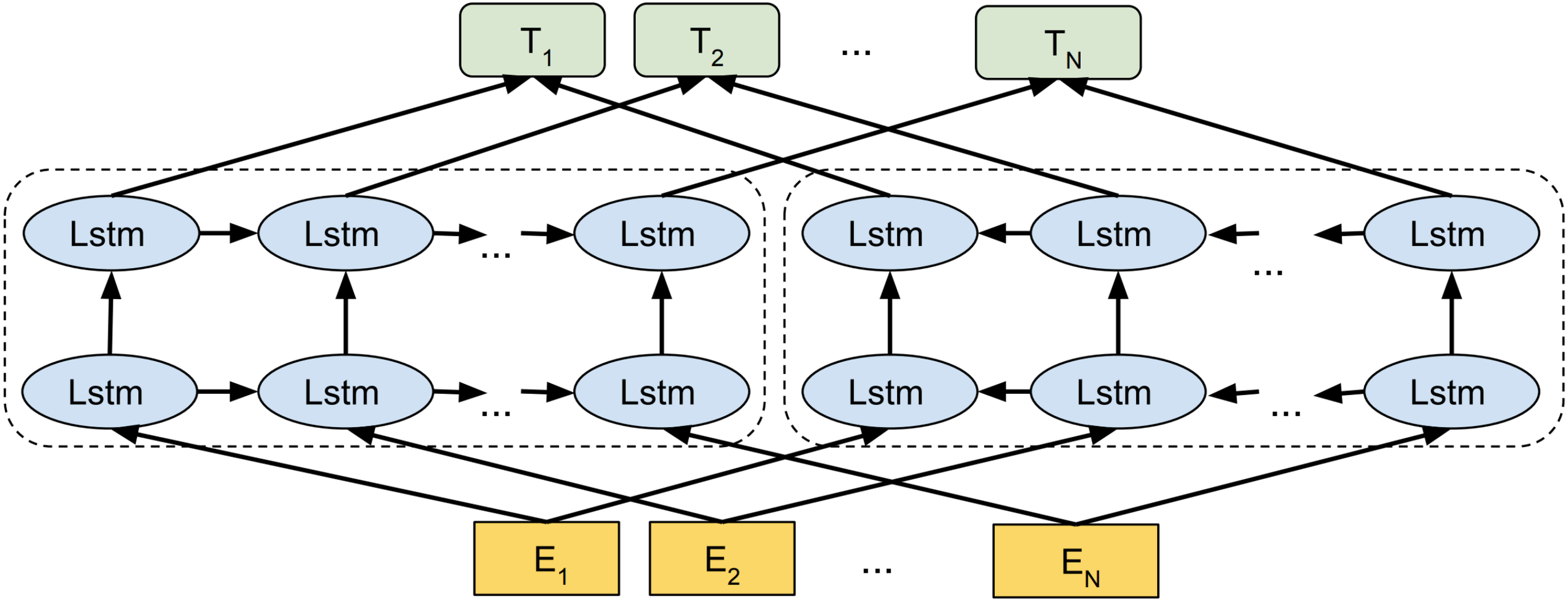
- Task specific weighting of all biLM layers:

$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}
$$

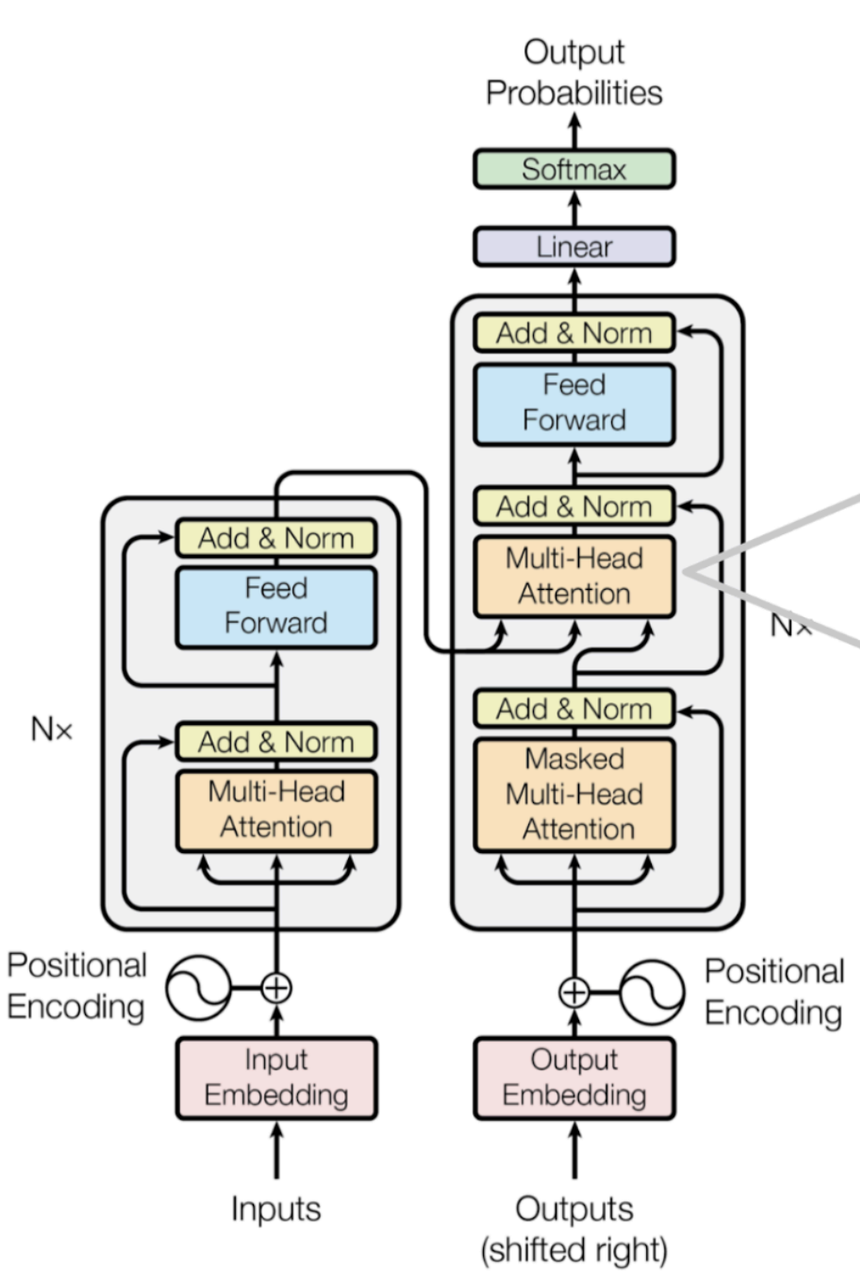
- s^{task} are softmax-normalized weights and the scalar parameter γ^{task} allows the task model to scale the entire ELMo vector.

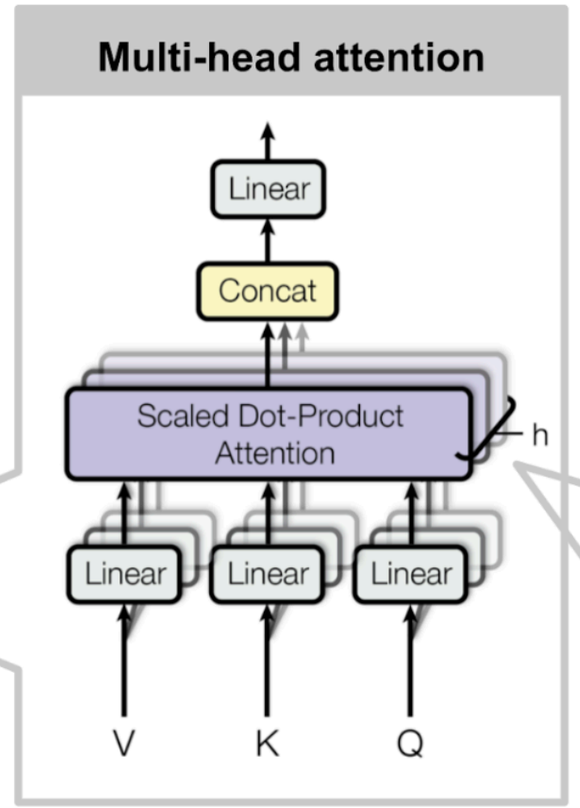
Limitation of ELMo

- ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks.

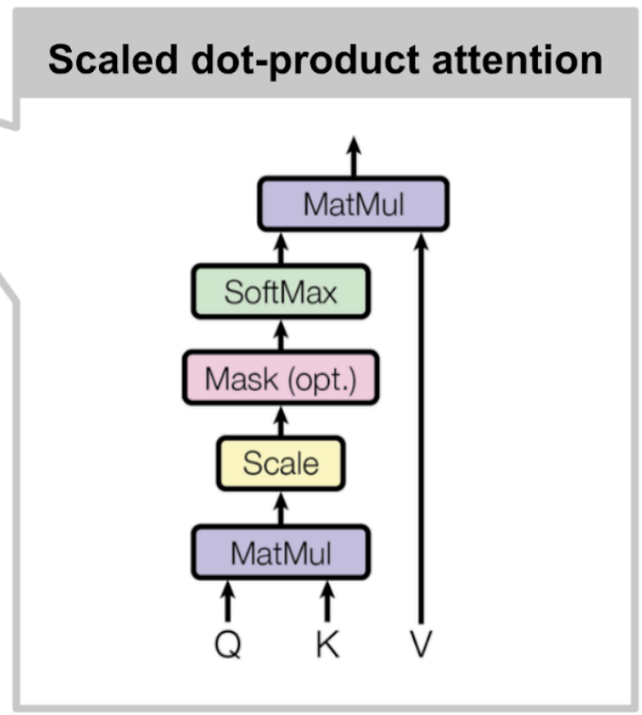


Transformer





Zoom-In!



Zoom-In!

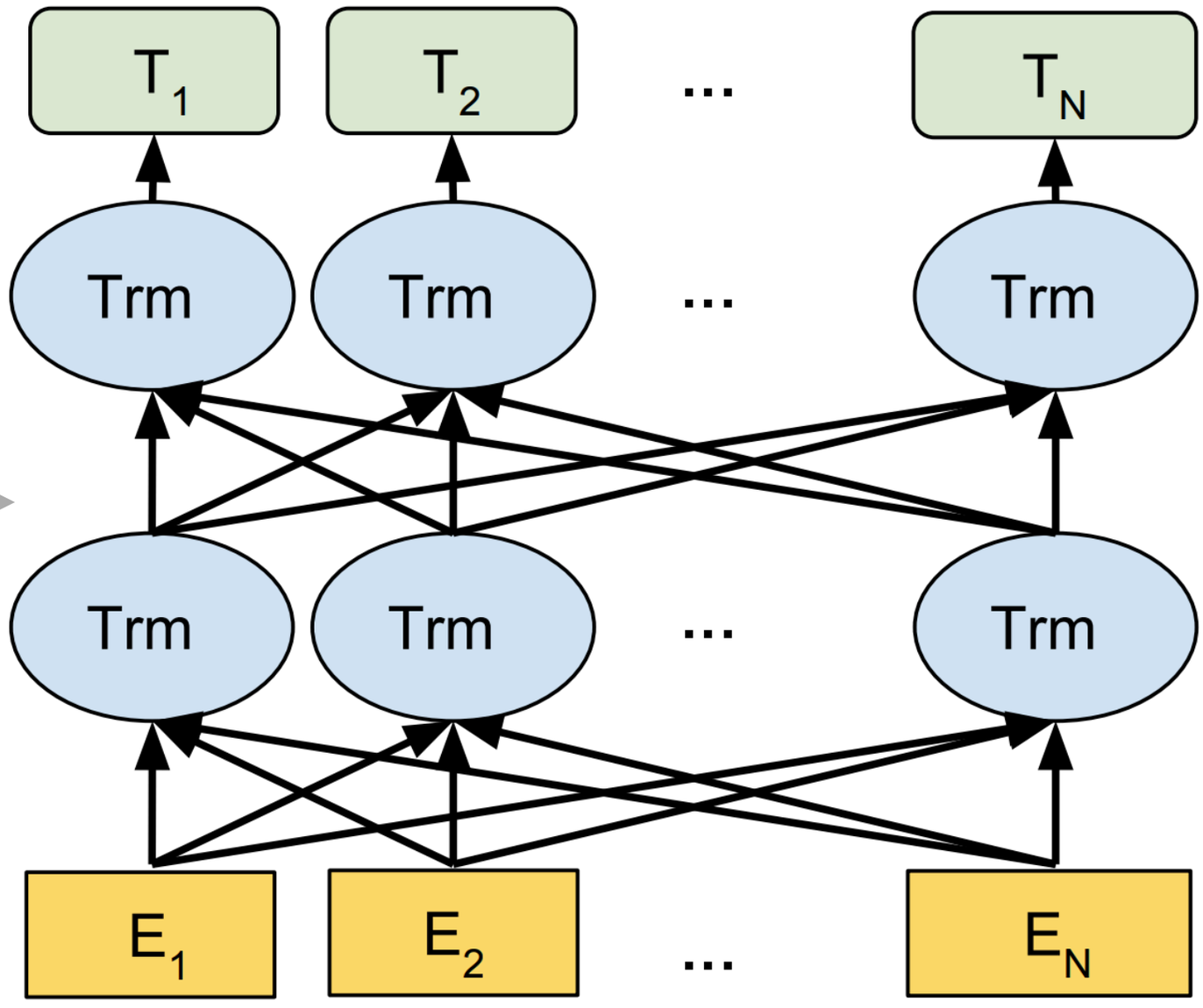
Google BERT

- BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
  - Transformer is an encoder playing the role as LSTM
- A sophisticated sentence encoder with pre-trained models provided by Google in the late 2018.
- Multilingual
- For Chinese, no word segmentation is required.
  - Character-based

https://arxiv.org/abs/1810.04805

Architecture of Google BERT

Transformer with 12 layers

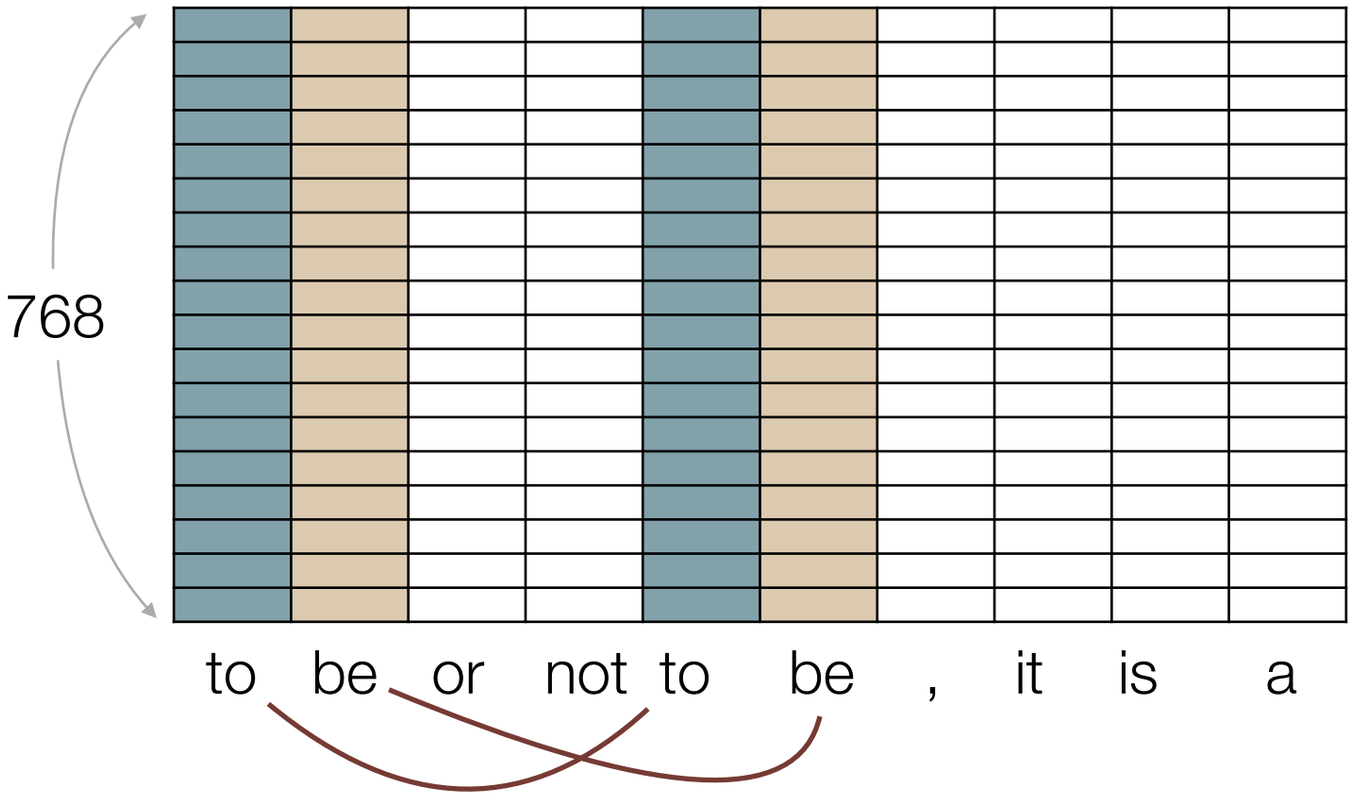


https://github.com/google-research/bert

Google BERT Encoder

- Each sentence with *n* words is represented as a matrix with the dimension of 768 * *n*

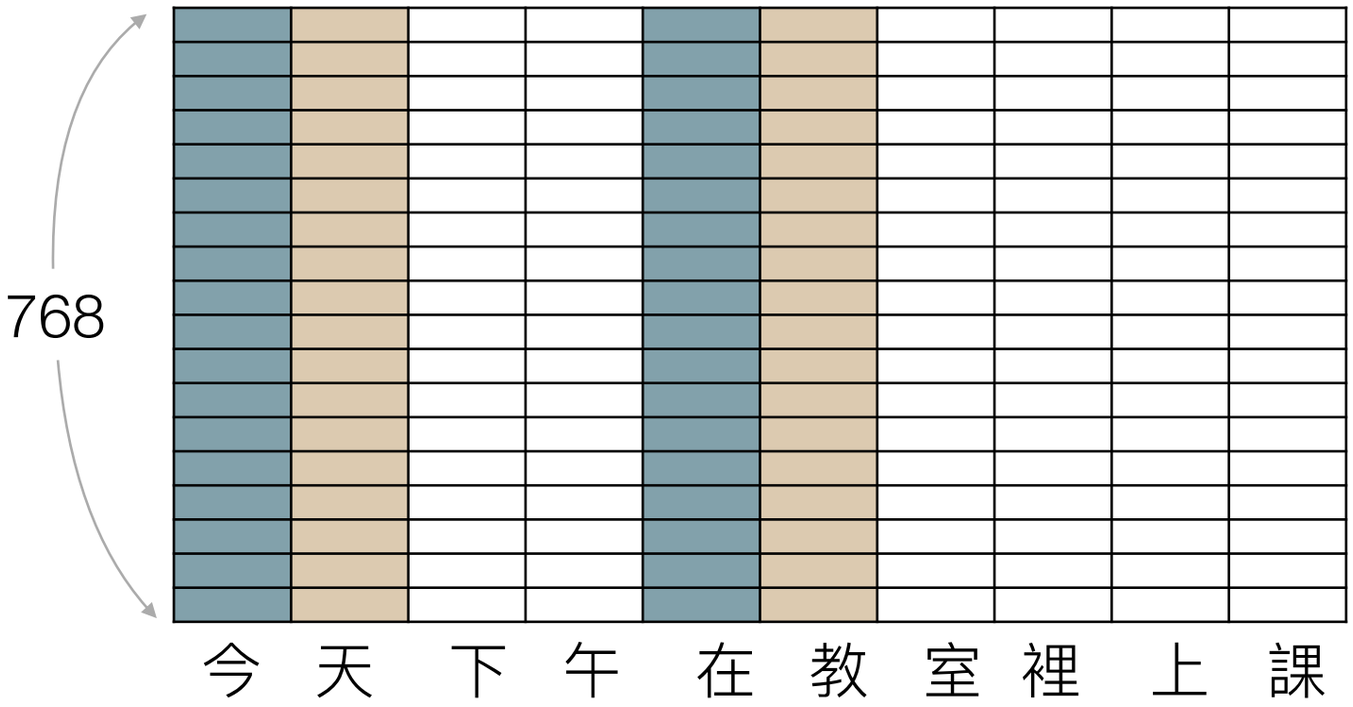
- Words from more than 100 languages are projected to a single space.



The embedding of the same word differs according to the context

Google BERT Encoder for Chinese

- Character-based representation traditional/simplified Chinese.
- No Chinese word segmentation is required anymore.

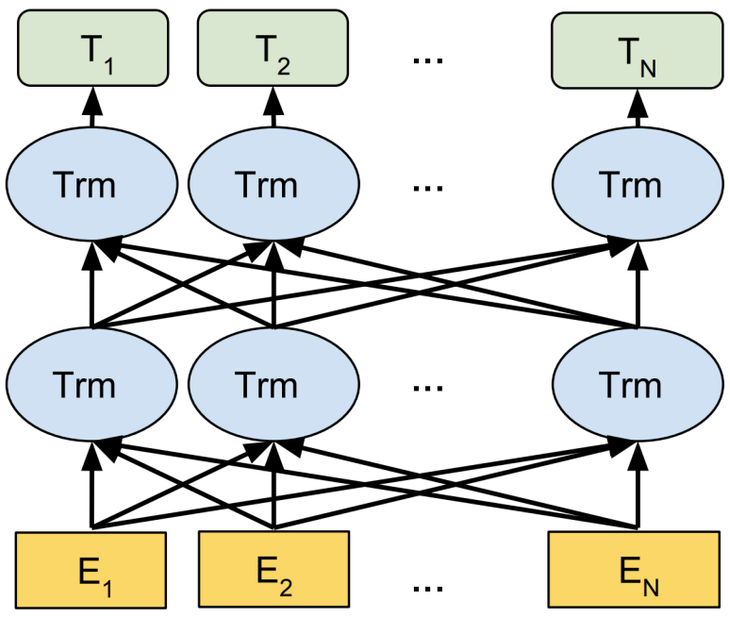


The embedding of the same word differs according to the context

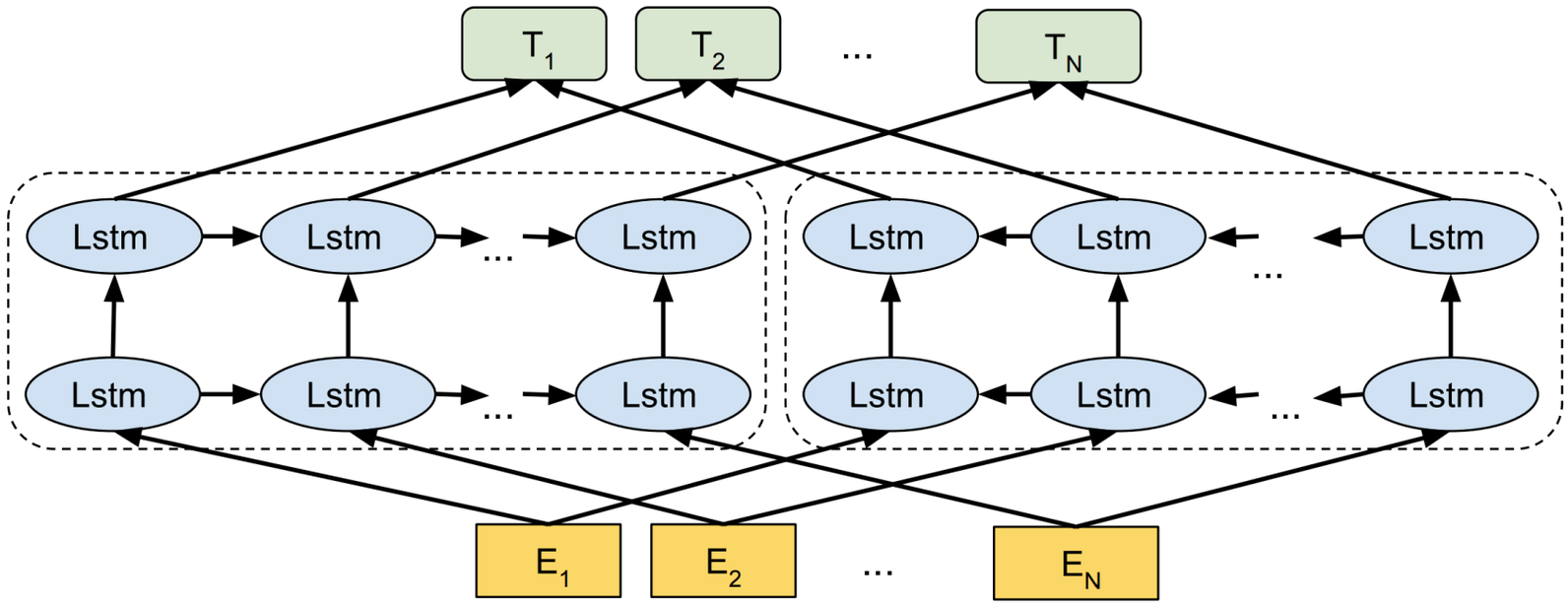
Transformer vs Bi-LSTM

- Each step in the LSTM can only directly access the information of its neighboring steps.

- Each step in the transformer can directly access the information of **any** steps in the sequence.



BERT



ELMO

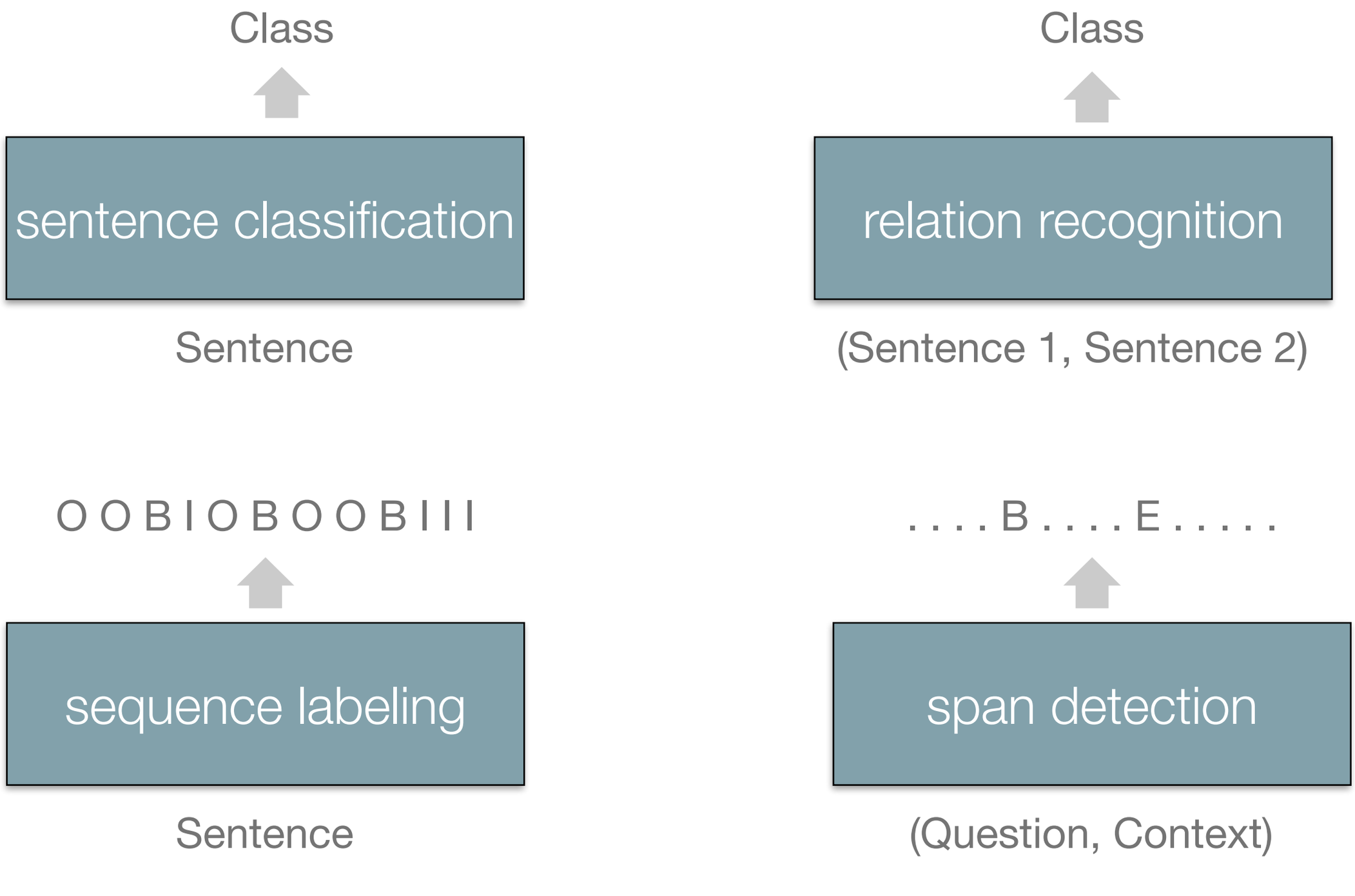
Self-supervision Pre-training of BERT

- The pre-training tasks play an important role for Google BERT.
- Task 1: Masked language modeling (Cloze task)
    - My dog is [MASK] => hairy
- Task 2: Next sentence prediction
    - the man went to [MASK] store => he bought a gallon [MASK] milk
- Two tasks are co-trained at the same time.

Fine-tuning of BERT

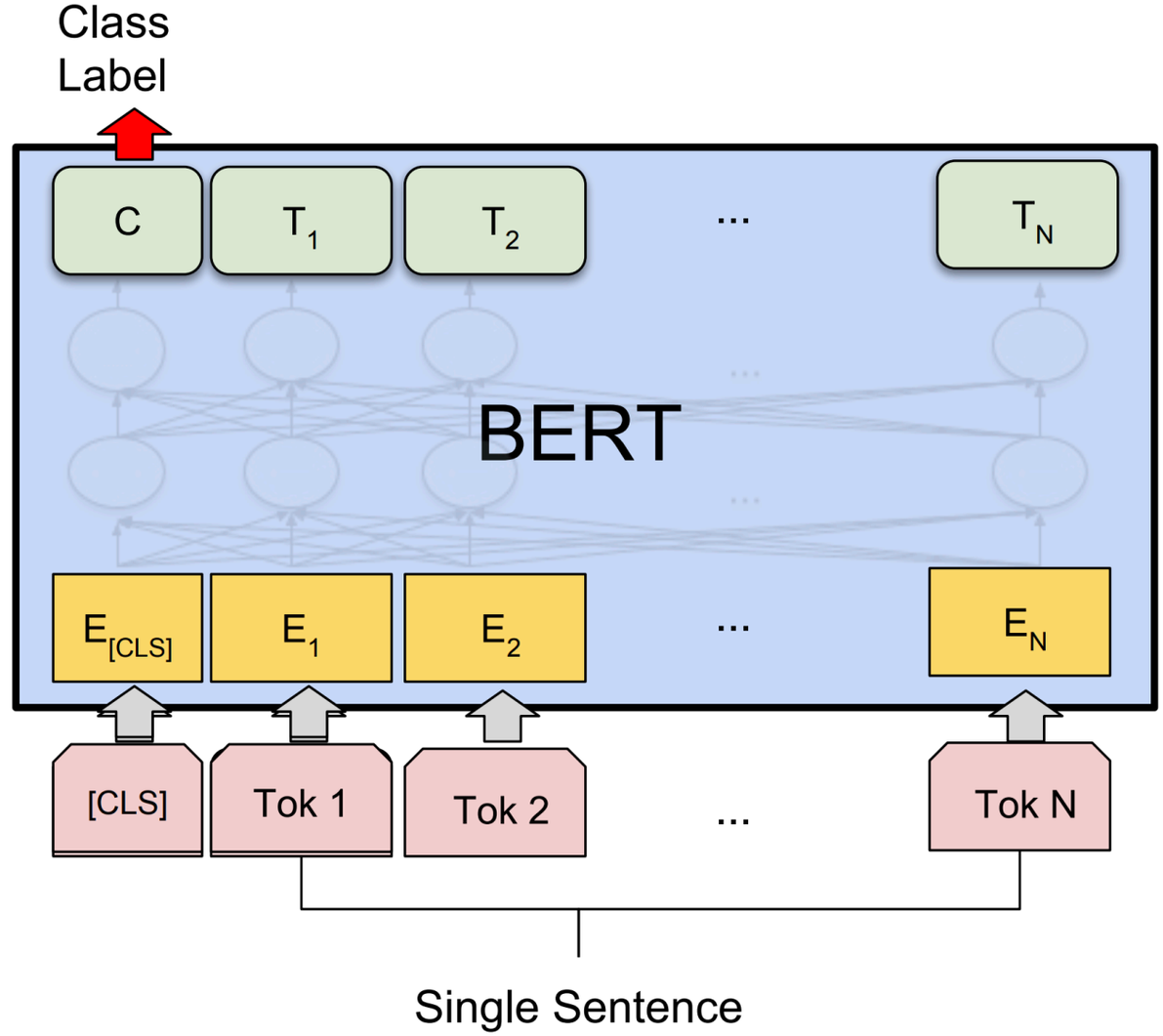
- The sentence embeddings generated by BERT can be used as static information for subsequent stages of the neural network.
    - By freezing the part of the BERT in the whole NN.
- Like word embeddings, the sentence embeddings can also be used as static features for traditional machine learning models such as SVM and logistic regression.
- However, fine-tuning the BERT for your target task is generally better in most situation.

Four Fundamental Tasks in NLP



Sentence Classification

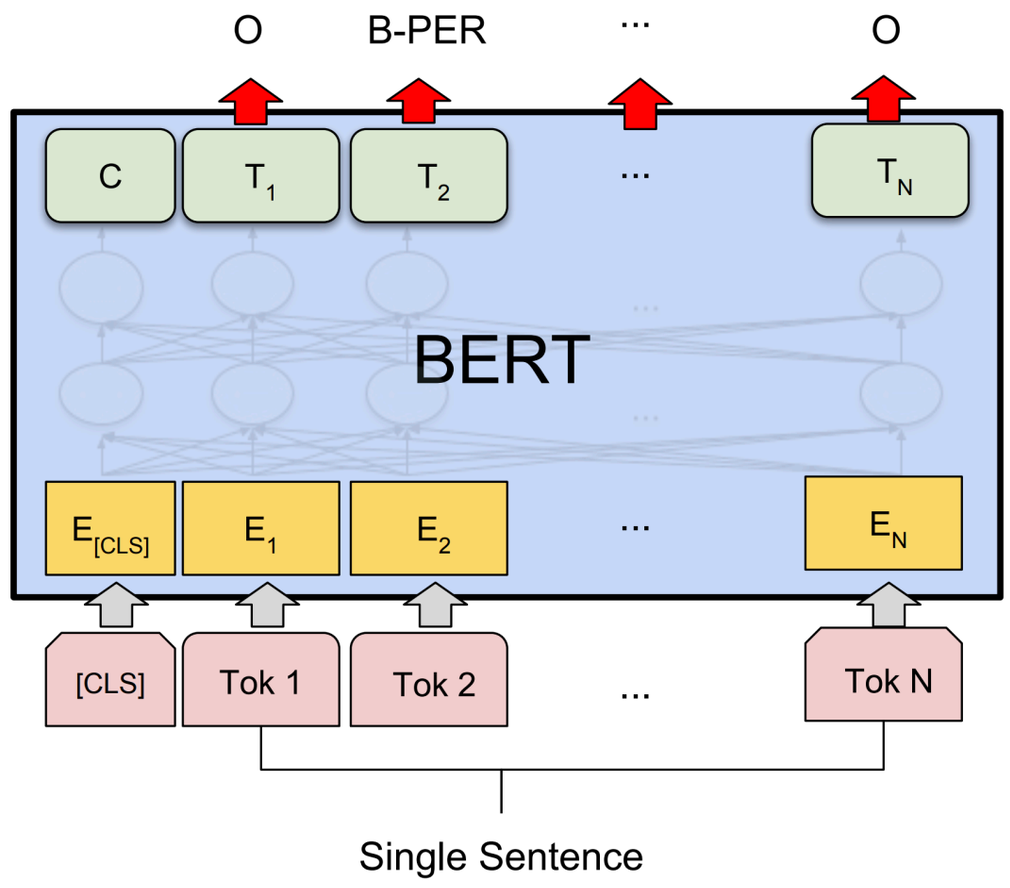
- Given a sentence, predict its category
- The most fundamental task in NLP
  - Sentiment analysis
  - Irony detection
  - Fake news detection



Sequence Labeling

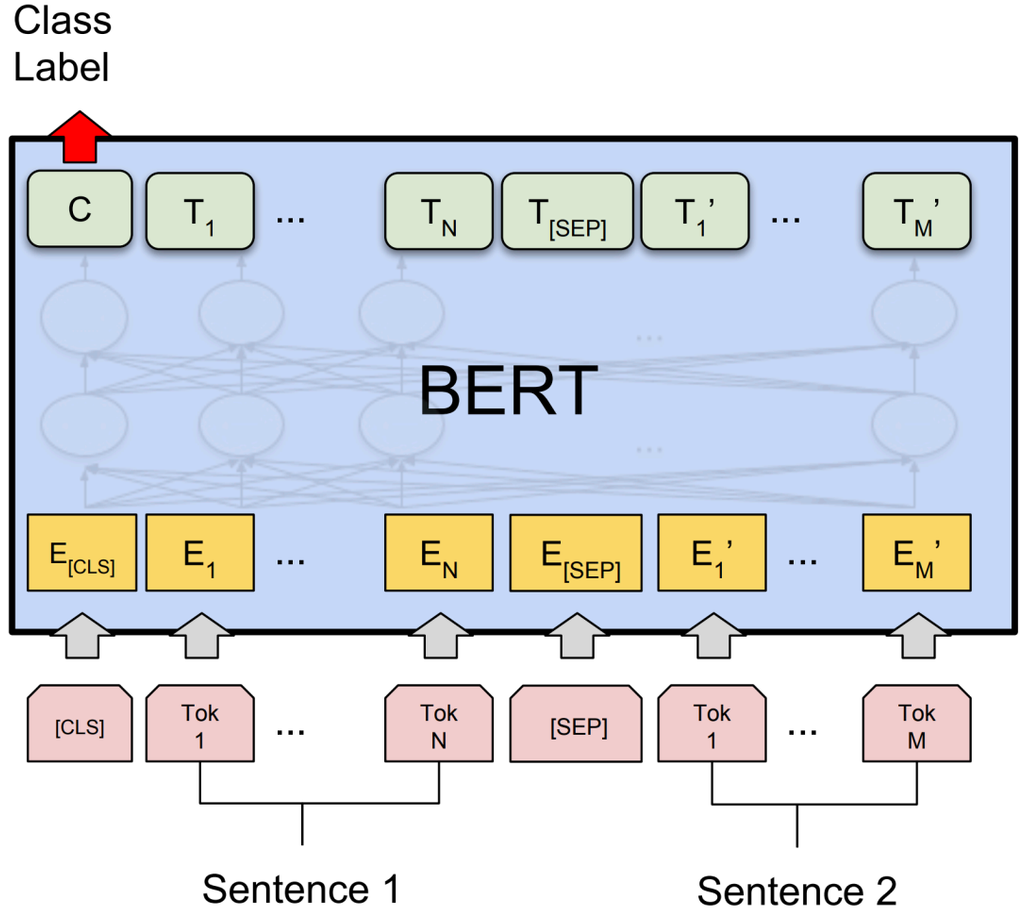
- Given a sentence as a sequence of tokens, predict the label for each tokens
  - Token is a word (most languages) or a character (Chinese)

- POS tagging
- Chinese word segmentation
- Information extraction



Relation Recognition

- Given a pair of sentences, predict their relation.
- Similarity of two sentences
- Discourse relation recognition
- Textual entailment



Textual Entailment

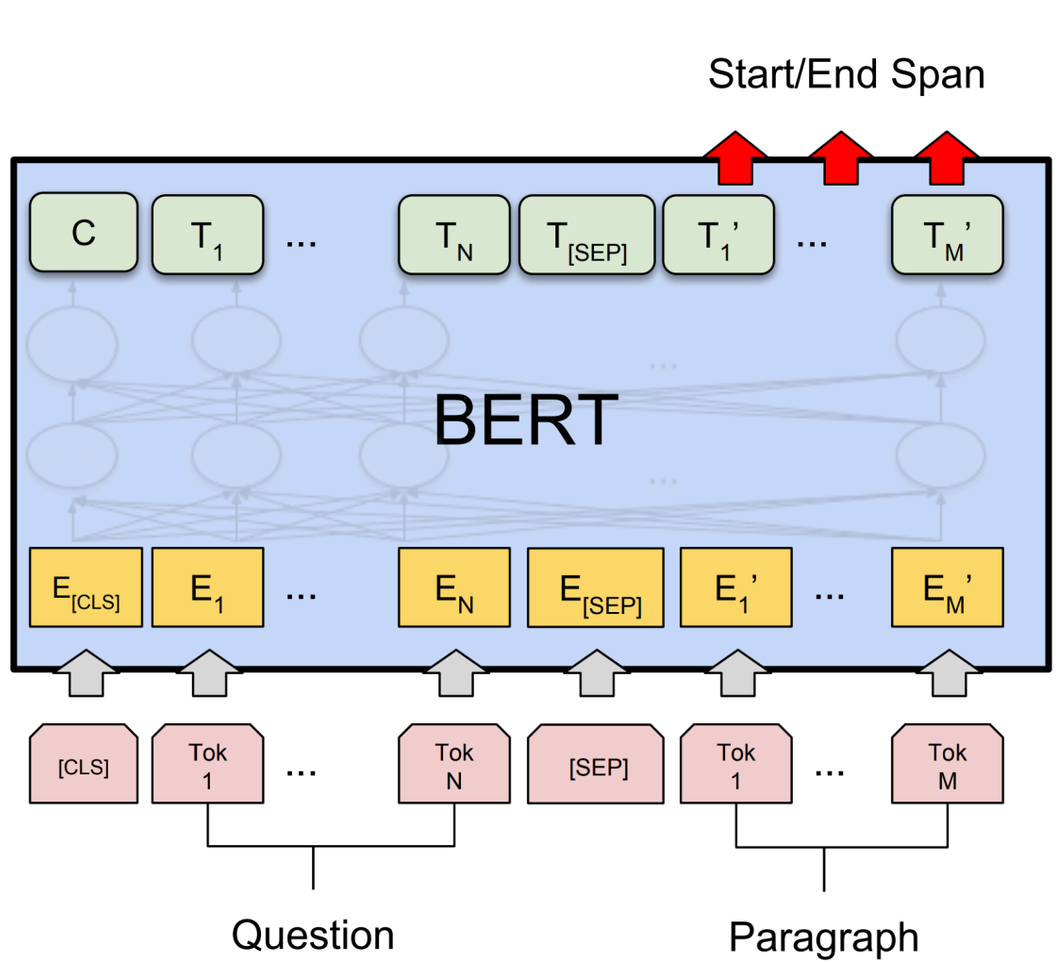
- Aka natural language inference (NLI)
- A very important task in NLP
- Given two sentence H and T, predict their relation as one of the three relations:
  - Entailment
  - Contradiction
  - Neutral

Textual Entailment

At 8:34, the Boston Center controller received a third transmission from American 11	**Entailment**	The Boston Center controller got a third transmission from American 11.
Met my first girlfriend that way	**Contradiction**	I didn't meet my first girlfriend until later
I am a lacto-vegetarian. (我是奶蛋素食者)	**Neutral**	I enjoy eating cheese too much to abstain from dairy

Span Detection

- Given a query and a paragraph, suggest a span of text from the paragraph.
- Question answering
- Reading comprehension



Question Answering Dataset: SQuAD

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”.

1. What causes precipitation to fall?

   gravity

2. What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?

   graupel (霰)

3. Where do water droplets collide with ice crystals to form precipitation?

   with in a cloud

https://rajpurkar.github.io/SQuAD-explorer/

Useful Resources

- KERAS
  - https://github.com/CyberZHG/keras-bert
- PyTorch
  - https://github.com/codertimo/BERT-pytorch
- Huggingface
  - https://github.com/huggingface/transformers
- Simpletranformers
  - https://github.com/ThilinaRajapakse/simpletransformers

Feed-Forward Neural Network with Keras

```
model = Sequential()

model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

Advantages of Keras

- In Python
- Very simple like scikit-learn
- Many latest, powerful features are built-in
- Rich resource

https://pixabay.com/vectors/swiss-army-knife-pocket-knife-blade-154314/

Advantages of PyTorch

- Flexible
  - Keras is convenient but not that flexible
  - TensorFlow is a bit lower level side.
- In Python
- Fast and simple
- Rich resource

https://pixabay.com/vectors/toolbox-tools-red-kit-box-29058/

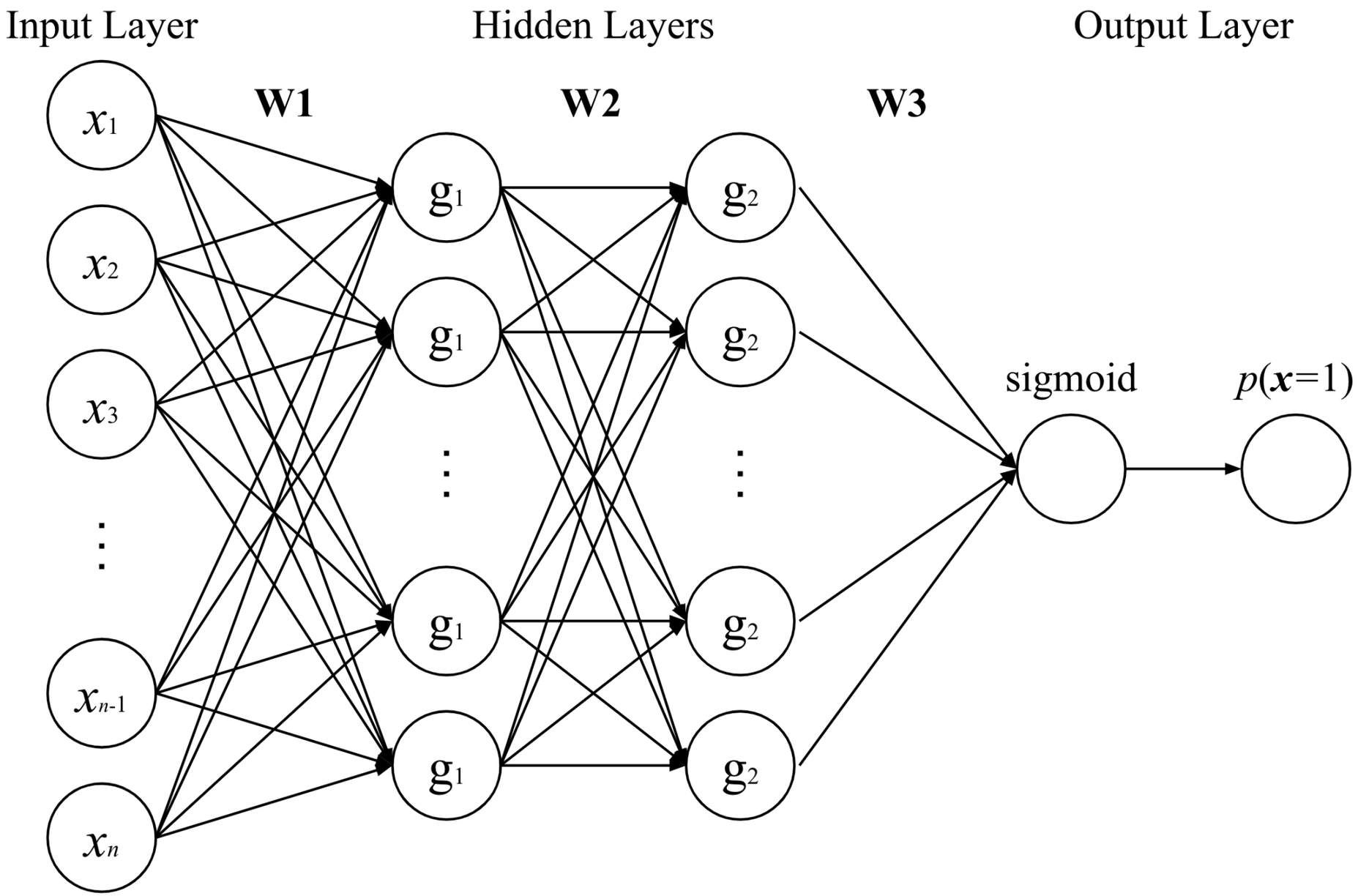
Using Simpletransformer

```
from simpletransformers.classification import ClassificationModel
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO)
transformers_logger = logging.getLogger("transformers")
transformers_logger.setLevel(logging.WARNING)

training_data = [["a good movie", 1],
                 ["the plot is too boring", 0]]

train_df = pd.DataFrame(training_data)

model = ClassificationModel('bert', 'bert-base-uncased')

model.train_model(train_df)

predictions, raw_outputs = model.predict(["i saw a nice film"])
```

Model Tuning

- Optimizer
    - sgd, adam, adagrad, and so on
- 2 ~ 50 Epochs
- Number of hidden layers
    - Start with 1 to 10
- Hidden size
    - 20 to 1000
- Embedding size
    - 20 to 1000
- Batch size
    - 16 to ?

Regularization



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

$$NN_{MLP(2)}(\mathbf{x}) = g_2(g_1(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2})\mathbf{W^3} + \mathbf{b^3}$$

Regularization

- L1 or L2 is usually used for regularization

$$\hat{y} = MLP(\mathbf{x}) = g_2(g_1(\mathbf{x}\mathbf{W^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2})\mathbf{W^3} + \mathbf{b^3}$$

$$\mathcal{L}_{MLP} = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

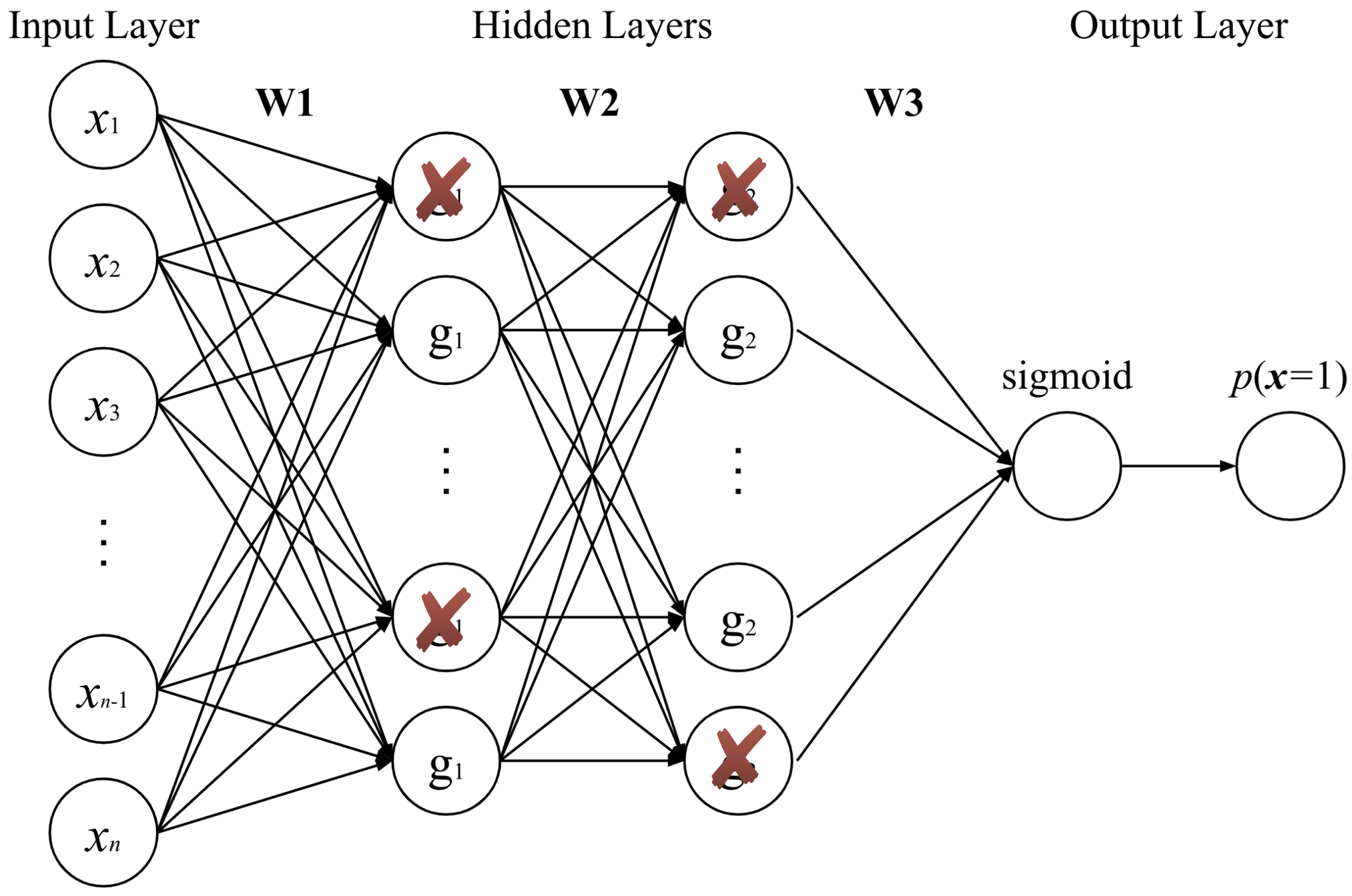
$$\mathcal{L}_{R1} = \frac{1}{2}|\mathbf{W^1}|_2^2$$

$$\mathcal{L}_{R2} = \frac{1}{2}|\mathbf{W^2}|_2^2$$

$$\mathcal{L}_{R3} = \frac{1}{2}|\mathbf{W^3}|_2^2$$

$$\mathcal{L} = \mathcal{L}_{MLP} + \mathcal{L}_{R1} + \mathcal{L}_{R2} + \mathcal{L}_{R3}$$

Prevent Overfitting: Dropout



Early Stopping

- Applied to prevent overfitting
- Reserving a part of training data for validation in each epoch.
- Terminating the training when no lower loss is found in recent *n* epochs.
  - *n*: How much patience