Compositional Semantics

Introduction

Compositionality in natural languages allows for the creation of intricate semantic meanings by combining simpler semantic elements. This principle states that the meaning of a whole is determined by the meanings of its parts, highlighting how semantic elements combine to form complex structures.

Here we express the composition of two semantic units, which are denoted as u and v, respectively, and the most intuitive way to define the joint representation could be formulated as follows:

$$p = f(u, v)$$

Introduction

However, given the representations of two semantic constituents, it is not enough to derive their joint embeddings with the lack of syntactic information. For instance, although the phrase machine learning and learning machine have the same vocabulary, they contain different meanings: machine learning refers to a research field in artificial intelligence while learning machine means some specific learning algorithms

Therefore, the composition function is redefined to combine the syntactic relationship rule R between the semantic units u and v.

$$p = f(u, v, R)$$

Introduction

Understanding complex meanings involves more than just grasping individual components and their relationships. Context plays a crucial role, as the same sentence can convey different meanings depending on the context. For instance, the sentence "Tom and Jerry is one of the most popular comedies in that style" requires prior knowledge about "Tom and Jerry" as a cartoon comedy and an understanding of the term "style." Therefore, achieving a complete understanding of compositional semantics entails incorporating existing knowledge into the process.

$$p = f(u, v, R, K)$$

Where K represents the background knowledge.

Introduction

Understanding complex meanings involves more than just grasping individual components and their relationships. Context plays a crucial role, as the same sentence can convey different meanings depending on the context. For instance, the sentence "Tom and Jerry is one of the most popular comedies in that style" requires prior knowledge about "Tom and Jerry" as a cartoon comedy and an understanding of the term "style." Therefore, achieving a complete understanding of compositional semantics entails incorporating existing knowledge into the process.

$$p = f(u, v, R, K)$$

Where K represents the background knowledge.

Additive Model

The additive model has a constraint in which it assumes that p, u, and v lie in the same semantic space. This essentially means that all syntactic types have the same dimension. One of the simplest ways is to directly use the sum to represent the joint representation:

$$\mathbf{p} = \mathbf{u} + \mathbf{v}$$

Let the hypothetical vectors for machine and learning be [0, 3, 1, 5, 2] and [1, 4, 2, 2, 0] The sum of the two vectors representing machine and learning would be w(machine) + w(learning) = [1, 7, 3, 7, 2]

Additive Model

The additive model has a constraint in which it assumes that p, u, and v lie in the same semantic space. This essentially means that all syntactic types have the same dimension. One of the simplest ways is to directly use the sum to represent the joint representation:

$$\mathbf{p} = \mathbf{u} + \mathbf{v}$$

Let the hypothetical vectors for machine and learning be [0, 3, 1, 5, 2] and [1, 4, 2, 2, 0]
The sum of the two vectors representing machine and learning would be w(machine) + w(learning) = [1, 7, 3, 7, 2]

Additive Model

To overcome the word order issue, one easy variant is applying a weighted sum instead of uniform weights. This is to say, the composition has the following form:

$$\mathbf{p} = \alpha\mathbf{u} + \beta\mathbf{v}$$

where α and β correspond to different weights for two vectors

As an example, if we set α to 0.3 and β to 0.7, the 0.3 × w(machine) = [0, 0.9, 0.3, 1.5, 0.6] and 0.7 × w(learning) = [0.7,2.8,1.4,1.4,0],and “machine learning” is represented by their addition 0.3 × w(machine) + 0.7 × w(learning) = [0.7, 3.6, 1.7, 2.9, 0.6].

Additive Model

To overcome the word order issue, one easy variant is applying a weighted sum instead of uniform weights. This is to say, the composition has the following form:

$$\mathbf{p} = \alpha \mathbf{u} + \beta \mathbf{v}$$

where α and β correspond to different weights for two vectors

As an example, if we set α to 0.3 and β to 0.7, the 0.3 × w(machine) = [0, 0.9, 0.3, 1.5, 0.6] and 0.7 × w(learning) = [0.7,2.8,1.4,1.4,0],and “machine learning” is represented by their addition 0.3 × w(machine) + 0.7 × w(learning) = [0.7, 3.6, 1.7, 2.9, 0.6].

Additive Model

However, this model could not consider prior knowledge and syntax information. To incorporate prior information into the additive model, one method combines nearest neighborhood semantics into composition, deriving

$$\mathbf{p} = \mathbf{u} + \mathbf{v} + \sum_{i=1}^{K} \mathbf{n}_i \,,$$

where $n_1, n_2, \ldots n_K$ denote all semantic neighbors of $\boldsymbol{v}$

Multiplicative Model

The multiplicative model aims to make higher order interaction, the most intuitive approach tried to apply the pair-wise product as a composition function approximation

$$\mathbf{p} = \mathbf{u} \odot \mathbf{v}$$

where, $p_i = u_i \cdot \mathrm{v}_\mathrm{i}$, which implies each dimension of the output only depends on the corresponding dimension of two input vectors

Multiplicative Model

In the additive model, we have $\mathbf{p} = \alpha\mathbf{u} + \beta\mathbf{v}$ to alleviate the word order issue. Note that here α and β are two scalars, which could be easily changed to two matrices. Therefore, the composition function could be represented as

$$\mathbf{p} = \mathbf{W}_\alpha \cdot \mathbf{u} + \mathbf{W}_\beta \cdot \mathbf{v}$$

Where $\mathbf{W}_\alpha$ and $\mathbf{W}_\beta$ are matrices which determine the importance of $\mathbf{u}$ and $\mathbf{v}$ to $\mathbf{p}$

Multiplicative Model

Generalizing multiplicative model ahead, another approach is to utilize tensors as multiplicative descriptors and the composition function could be viewed as

$$\mathbf{p} = \overrightarrow{\mathbf{W}} \cdot \mathbf{u}\mathbf{v}$$

Where $\overrightarrow{\mathbf{W}}$ denotes a 3-order tensor, i.e., the formula above could be written as

$$\mathbf{p}_k = \sum_{i,j} \mathbf{W}_{ijk} \cdot \mathbf{u}_i \cdot \mathbf{v}_j$$

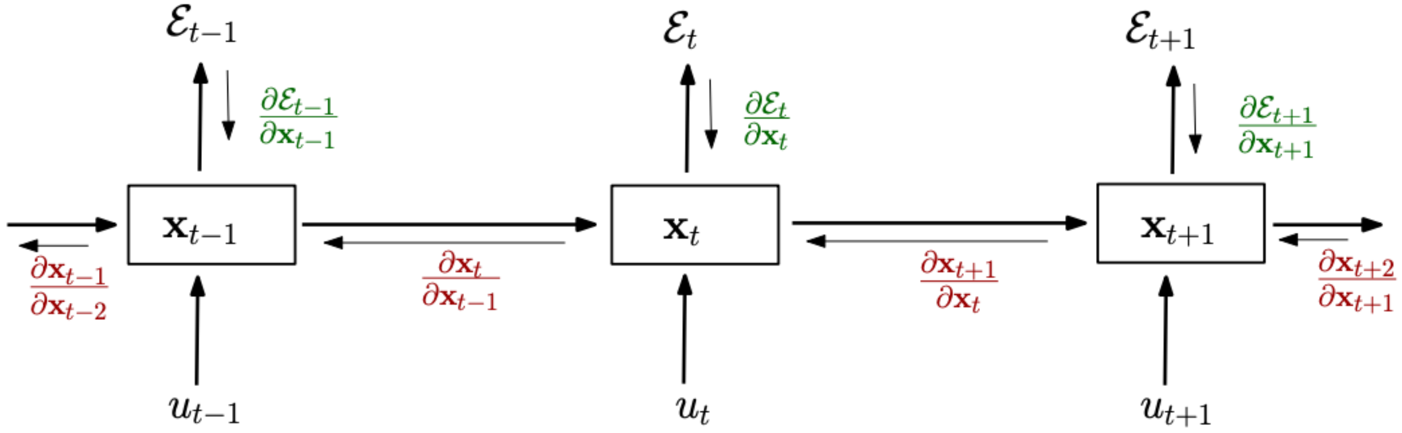
Multiplicative Model

Starting from this simple but general baseline, some researchers proposed to make the function not symmetric to consider word order in the sequence. Paying more attention to the first element, the composition function could be

$$\mathbf{p} = \overrightarrow{\mathbf{W}} \cdot \mathbf{uuv}$$

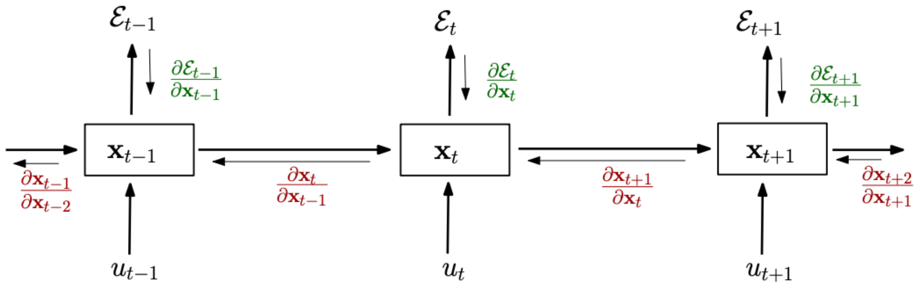
Where $\overrightarrow{\mathbf{W}}$ denotes a 4-order tensor

Vanishing and Exploding Gradients


$\mathcal{E}_{t-1}$
$\mathcal{E}_t$
$\mathcal{E}_{t+1}$
$\frac{\partial \mathcal{E}_{t-1}}{\partial \mathbf{x}_{t-1}}$
$\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}$
$\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{x}_{t+1}}$
$\mathbf{x}_{t-1}$
$\mathbf{x}_t$
$\mathbf{x}_{t+1}$
$\frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_{t-2}}$
$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}}$
$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$
$\frac{\partial \mathbf{x}_{t+2}}{\partial \mathbf{x}_{t+1}}$
u_{t-1}
u_t
u_{t+1}

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b}$$

Vanishing and Exploding Gradients


$\mathcal{E}_{t-1}$
$\mathcal{E}_t$
$\mathcal{E}_{t+1}$
$\frac{\partial \mathcal{E}_{t-1}}{\partial \mathbf{x}_{t-1}}$
$\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}$
$\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{x}_{t+1}}$
$\mathbf{x}_{t-1}$
$\mathbf{x}_t$
$\mathbf{x}_{t+1}$
$\frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_{t-2}}$
$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}}$
$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$
$\frac{\partial \mathbf{x}_{t+2}}{\partial \mathbf{x}_{t+1}}$
u_{t-1}
u_t
u_{t+1}

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial \mathcal{E}_t}{\partial \theta}$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \ge i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1}))$$

RNNs for other NLP tasks

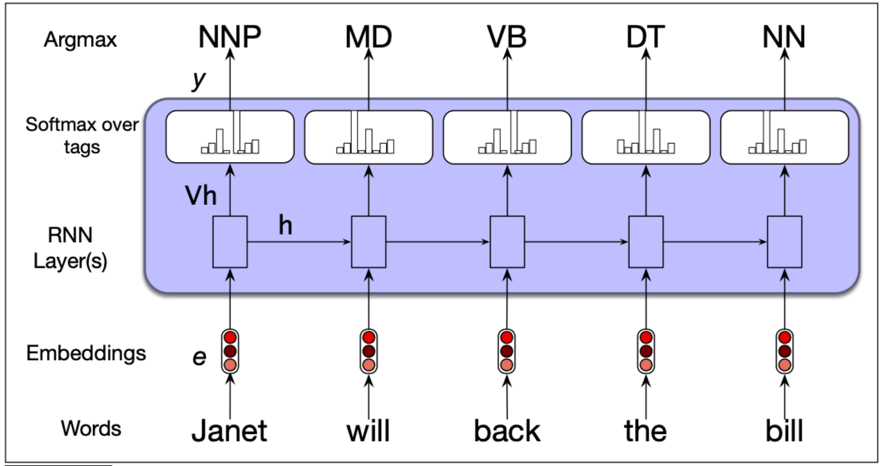
Now that we’ve seen the basic RNN architecture, let’s consider how to apply it to three types of NLP tasks: sequence classification tasks like sentiment analysis and topic classification, sequence labeling tasks like part-of-speech tagging, and text generation tasks, including with a new architecture called the encoder-decoder.

Sequence Labeling

In sequence labeling, the network’s task is to assign a label chosen from a small fixed set of labels to each element of a sequence, like the part-of-speech tagging and named entity recognition tasks

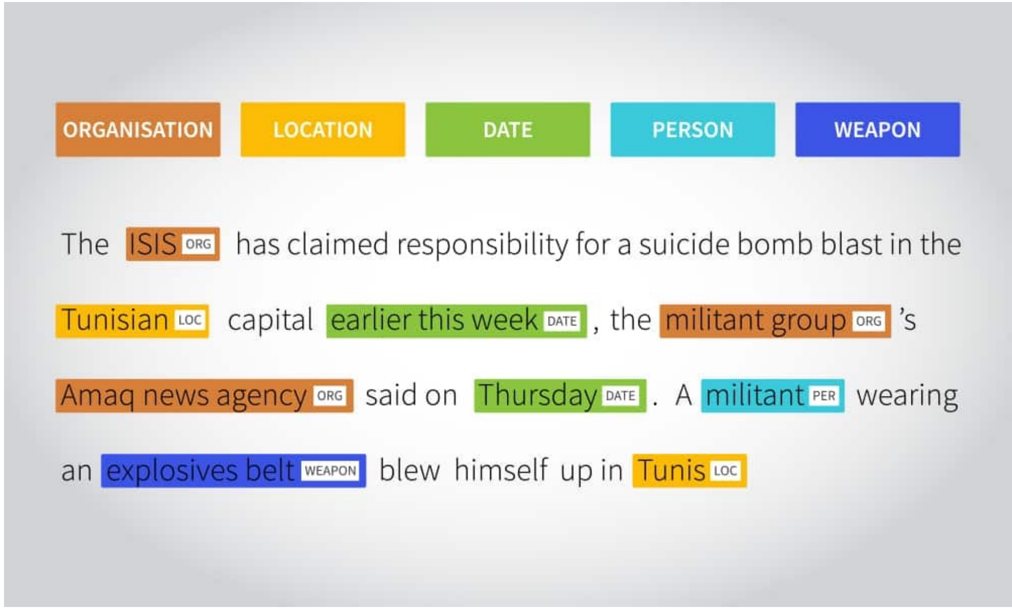
Sequence Labeling

In this figure, the inputs at each time step are pretrained word embeddings corresponding to the input tokens. The RNN block is an abstraction that represents an unrolled simple recurrent network consisting of an input layer, hidden layer, and output layer at each time step, as well as the shared U, V and W weight matrices that comprise the network. The outputs of the network at each time step represent the distribution over the POS tagset generated by a softmax layer.



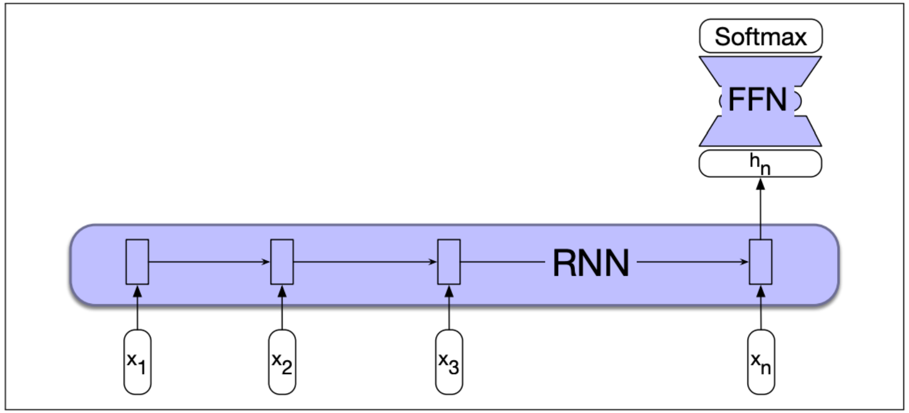
Sequence Labeling

Named Entity Recognition is a part of Natural Language Processing. The primary objective of NER is to process structured and unstructured data and classify these named entities into predefined categories. Some common categories include name, location, company, time, monetary values, events, and more.


ORGANISATION
LOCATION
DATE
PERSON
WEAPON
The ISIS ORG has claimed responsibility for a suicide bomb blast in the Tunisian LOC capital earlier this week DATE, the militant group ORG 's Amaq news agency ORG said on Thursday DATE. A militant PER wearing an explosives belt WEAPON blew himself up in Tunis LOC

RNNs for Sequence Classification

To apply RNNs in this setting, we pass the text to be classified through the RNN a word at a time generating a new hidden layer at each time step. We can then take the hidden layer for the last token of the text, h_n, to constitute a compressed representation of the entire sequence. We can pass this representation h_n to a feedforward network that chooses a class via a softmax over the possible classes



RNNs for Sequence Classification

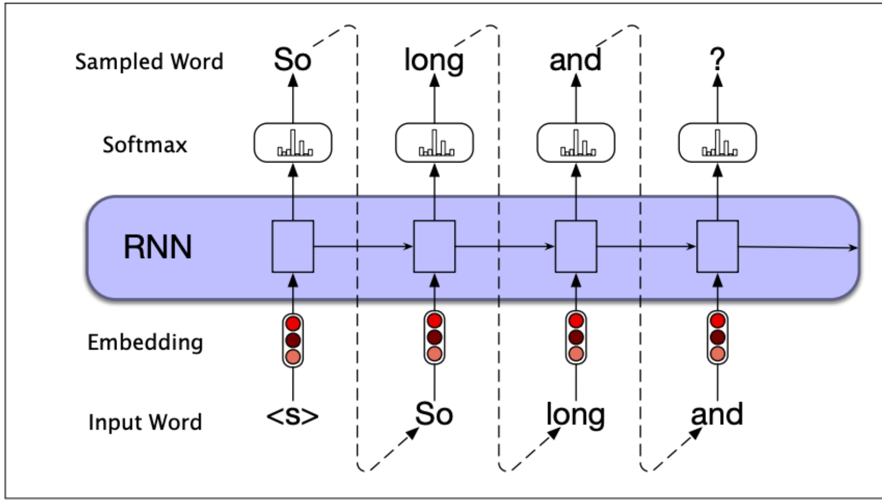
- Another option, instead of using just the last token hn to represent the whole sequence, is to use some sort of pooling function of all the hidden states h_i for each word i in the sequence. For example, we can create a representation that pools all the n hidden states by taking their element-wise mean:

$$h_{mean} = \frac{1}{n}\sum_{i=1}^{n} h_i$$

- Or we can take the element-wise max; the element-wise max of a set of n vectors is a new vector whose kth element is the max of the kth elements of all the n vectors.

Generation with RNN-Based Language Models

RNN-based language models can also be used to generate text. Text generation is of enormous practical importance, part of tasks like question answering, machine translation, text summarization, grammar correction, story generation, and conversational dialogue; any task where a system needs to produce text, conditioned on some other text

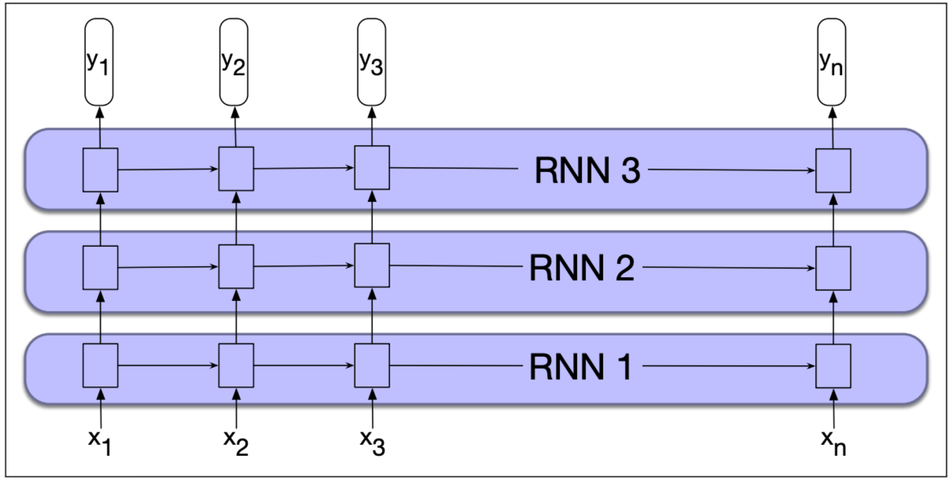


Stacked and Bidirectional RNN architectures

Recurrent networks are quite flexible. By combining the feedforward nature of unrolled computational graphs with vectors as common inputs and outputs, complex networks can be treated as modules that can be combined in creative ways. This section introduces two of the more common network architectures used in language processing with RNNs

Stacked RNN

Stacked RNNs generally outperform single-layer networks. One reason for this success seems to be that the network induces representations at differing levels of abstraction across layers. Just as the early stages of the human visual system detect edges that are then used for finding larger regions and shapes, the initial layers of stacked networks can induce representations that serve as useful abstractions for further layers—representations that might prove difficult to induce in a single RNN.



Bidirectional RNNs

the inputs $x_1, \dots, x_t$, and represents the context of the network to the left of the current time.

$$\mathbf{h}_t^f = \text{RNN}_{\text{forward}}(\mathbf{x}_1, \dots, \mathbf{x}_t)$$

the hidden state at time t represents information about the sequence to the right of the current input.

$$\mathbf{h}_t^b = \text{RNN}_{\text{backward}}(\mathbf{x}_t, \dots \mathbf{x}_n)$$

A bidirectional RNN combines two independent RNNs, one where the input is processed from the start to the end, and the other from the end to the start.

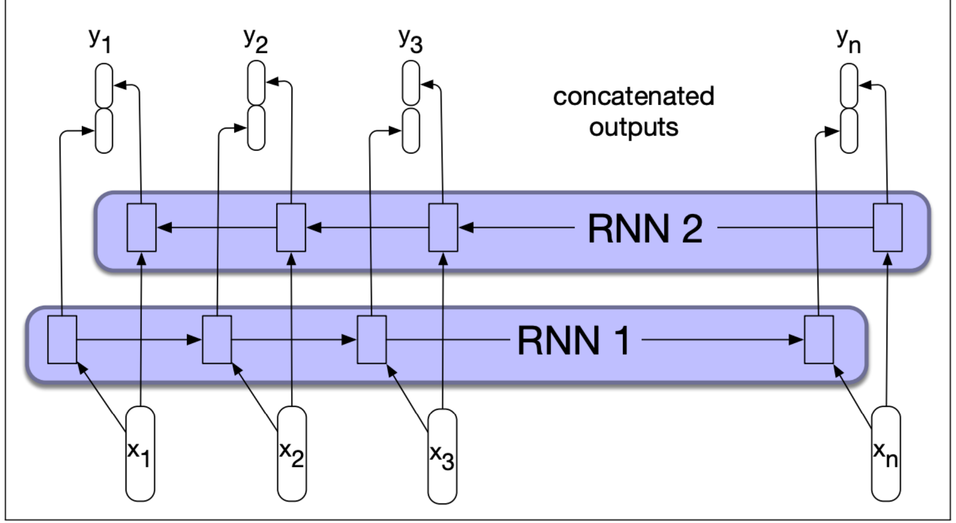
$$\begin{aligned} \mathbf{h}_t &= [\mathbf{h}_t^f \,;\, \mathbf{h}_t^b] \\ &= \mathbf{h}_t^f \oplus \mathbf{h}_t^b \end{aligned}$$

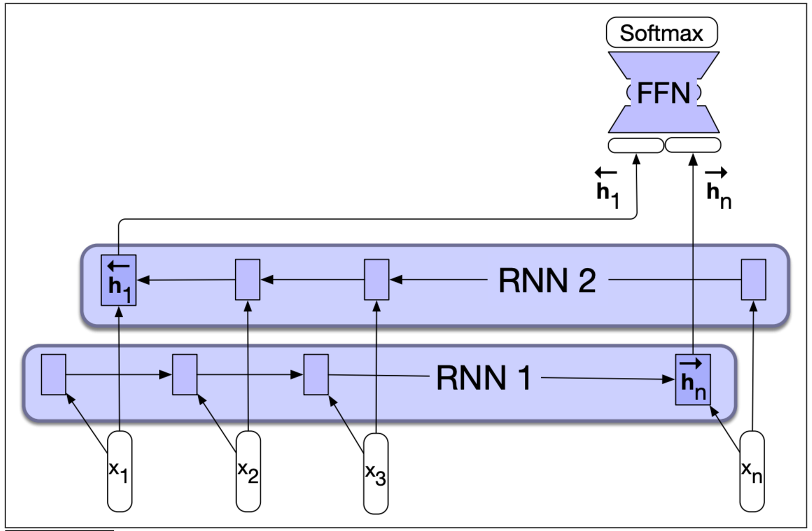
Bidirectional RNNs

$$\mathbf{h}_t^f = \mathrm{RNN}_{\text{forward}}(\mathbf{x}_1,\ldots,\mathbf{x}_t)$$

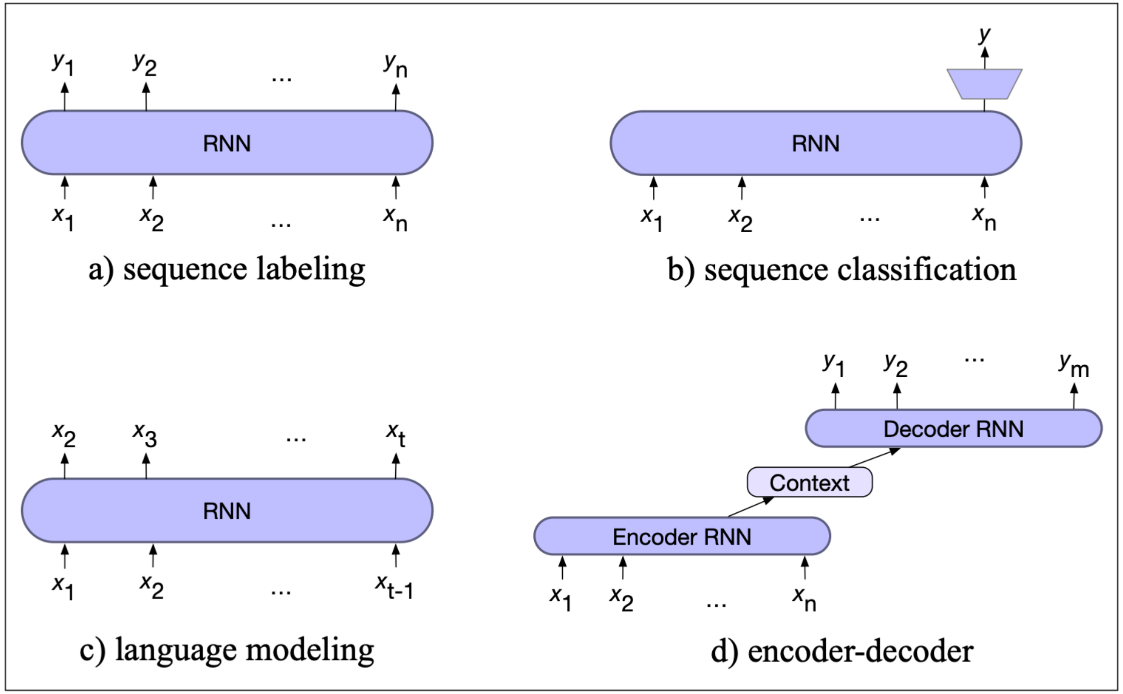
$$\mathbf{h}_t^b = \mathrm{RNN}_{\text{backward}}(\mathbf{x}_t,\ldots\mathbf{x}_n)$$

$$\begin{aligned}\mathbf{h}_t &= [\mathbf{h}_t^f\,;\,\mathbf{h}_t^b] \\ &= \mathbf{h}_t^f \oplus \mathbf{h}_t^b\end{aligned}$$





Summary



y_1 y_2 … y_n
RNN
x_1 x_2 … x_n
a) sequence labeling
y
RNN
x_1 x_2 … x_n
b) sequence classification
x_2 x_3 … x_t
RNN
x_1 x_2 … x_{t-1}
c) language modeling
y_1 y_2 … y_m
Decoder RNN
Context
Encoder RNN
x_1 x_2 … x_n
d) encoder-decoder