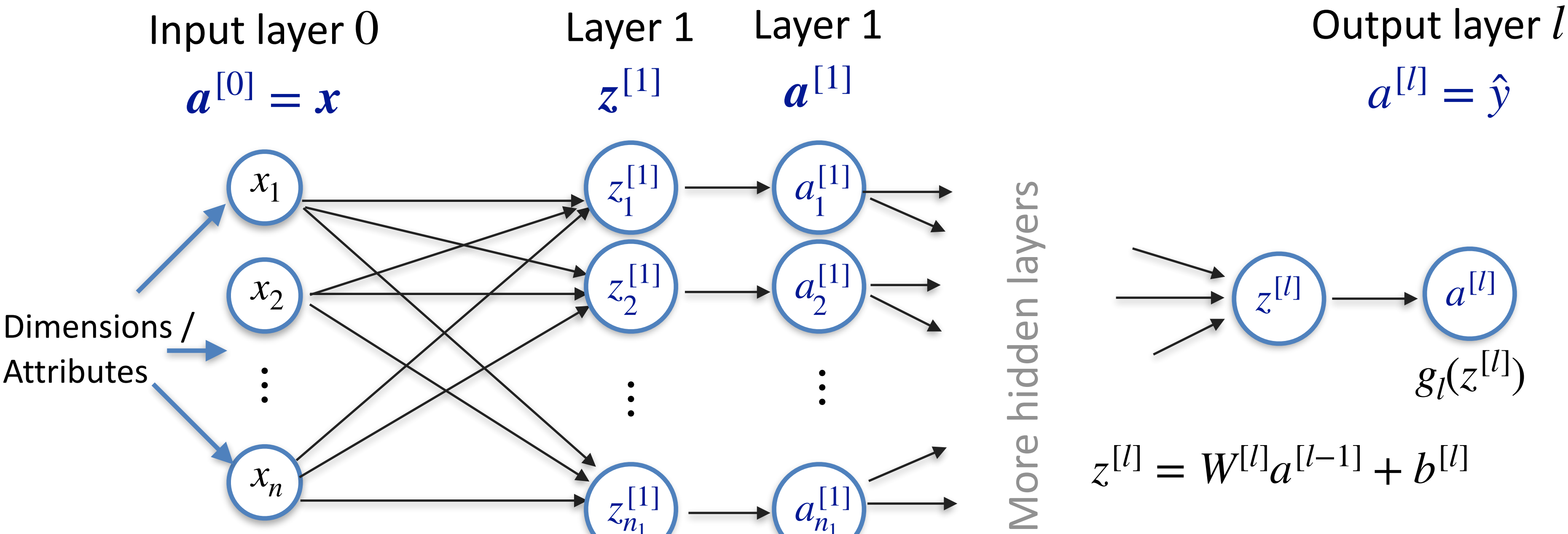
Recurrent Neural Networks

RNN, Types of RNN

Debapriyo Majumdar

debapriyo@isical.ac.in

Remember: deep feedforward networks



$\boldsymbol{z}^{[1]} = W^{[1]}\boldsymbol{a}^{[0]} + \boldsymbol{b}^{[1]}$

affine transformation

$a_i^{[1]} = g_1(z_i^{[1]})$

non-linear activation g_1 (element-wise)

Each layer: an affine transformation, followed by a non-linear activation

Data in sequential form

- Data that are temporal or in the form of sequences
  - Text
  - Speech
  - Music
  - Video
  - DNA sequence, …
- Each data point (and output, sometimes) is a sequence of input units
  - Inputs later in the sequence have dependency on previous inputs
- Early NLP techniques used the bag-of-word model
  - Ignored the sequence
  - Simulated somehow by set of sequences of words

Representing text in deep learning

21 46 34 32 42 7
sequential models are for text data

39 36 10 34 32 21 46
recurrent neural networks are for sequential models

47 36 10 34 48 2 32 28
convolutional neural networks are generally used for images

28 31 30 23 3 49 16
images may be colored with three channels

16 5 1 47 36 12 4
ResNet is a convolutional neural network architecture

21 19 6 39 36 10
sequential inputs need recurrent neural networks

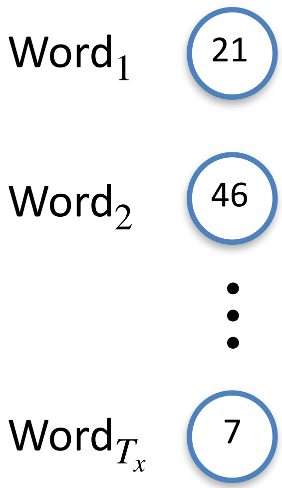
Assign an ID to each word

Each sentence (input) becomes a sequence of integers

Representing text for deep learning

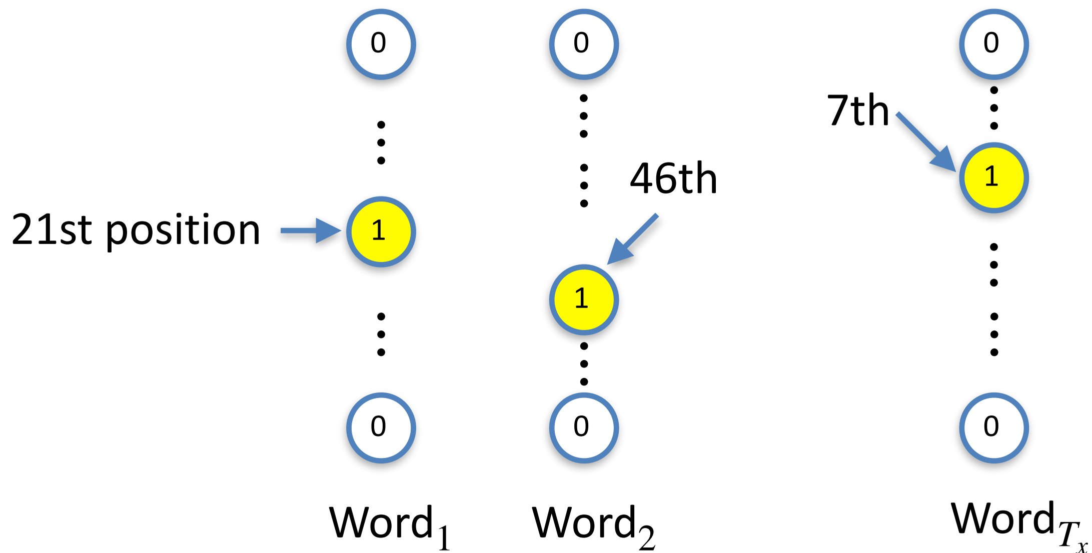
Consider a sentence of length T_x

The integer sequence
as input vector



Problem: the numbers have an order
that does not represent the words.

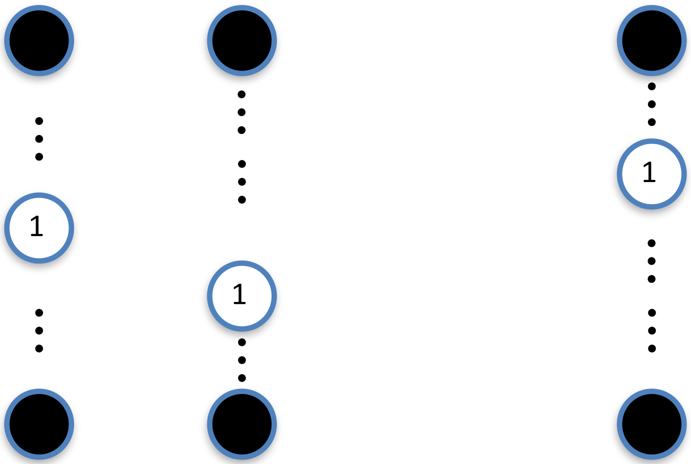
One-hot encoding
Each integer (word) as a vector



Each word is a vector with all zeros except one
Orthogonal to each other

Word embeddings

White = 1; Black = 0, Grey $\in (0,1)$.



Word_1 Word_2 Word_{T_x}

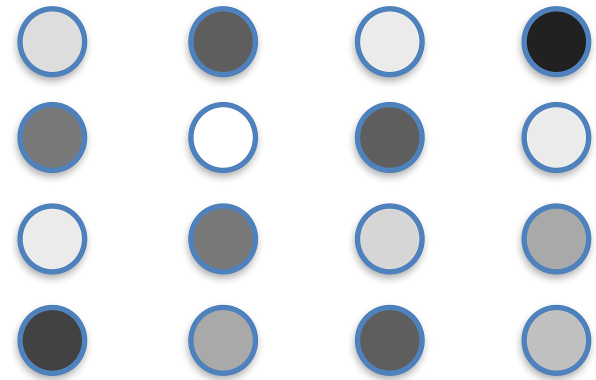
One-hot encoding

Each word is a vector with all zeros except one

Orthogonal to each other

But words have meanings

Some words are similar to each other



car biscuit vehicle tea

Better idea

More compact vector representation

retaining semantics of words

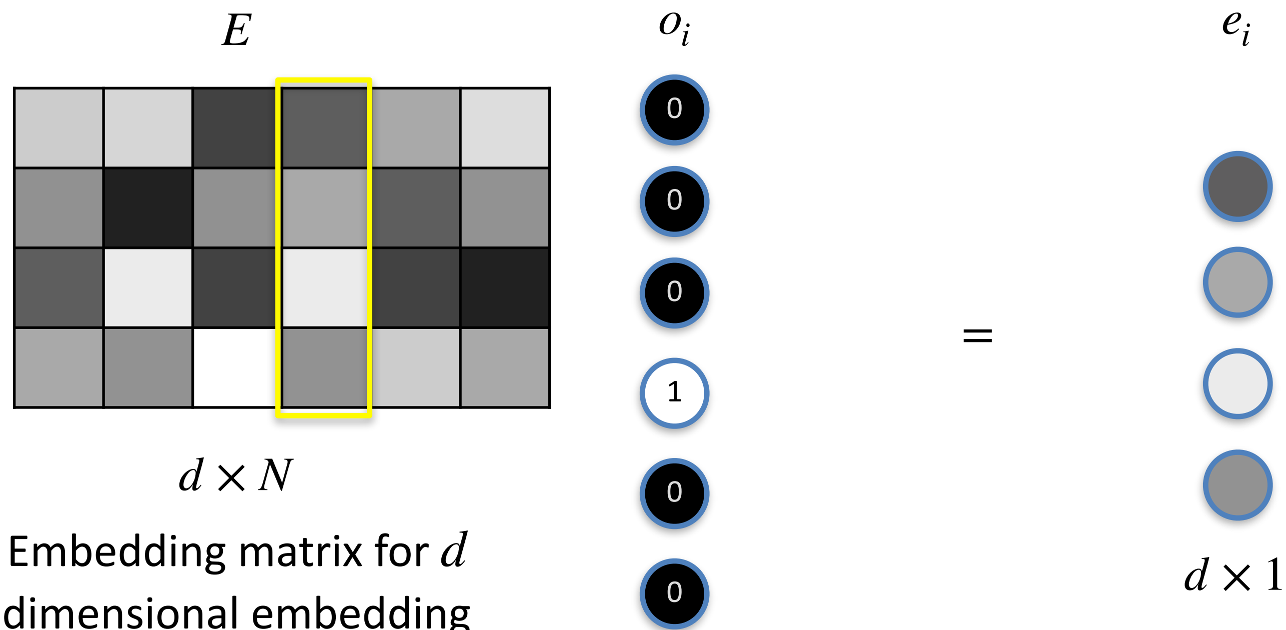
Similar words $\Longrightarrow$ similar vectors

Learn word embeddings

The embedding matrix

White = 1; Black = 0, Grey $\in (0,1)$.

$$e_i = E o_i$$



$d \times N$

Embedding matrix for d dimensional embedding

$N \times 1$

One-hot vector for the i-th word in the vocabulary

$d \times 1$

Embedding for the i-th word in the vocabulary

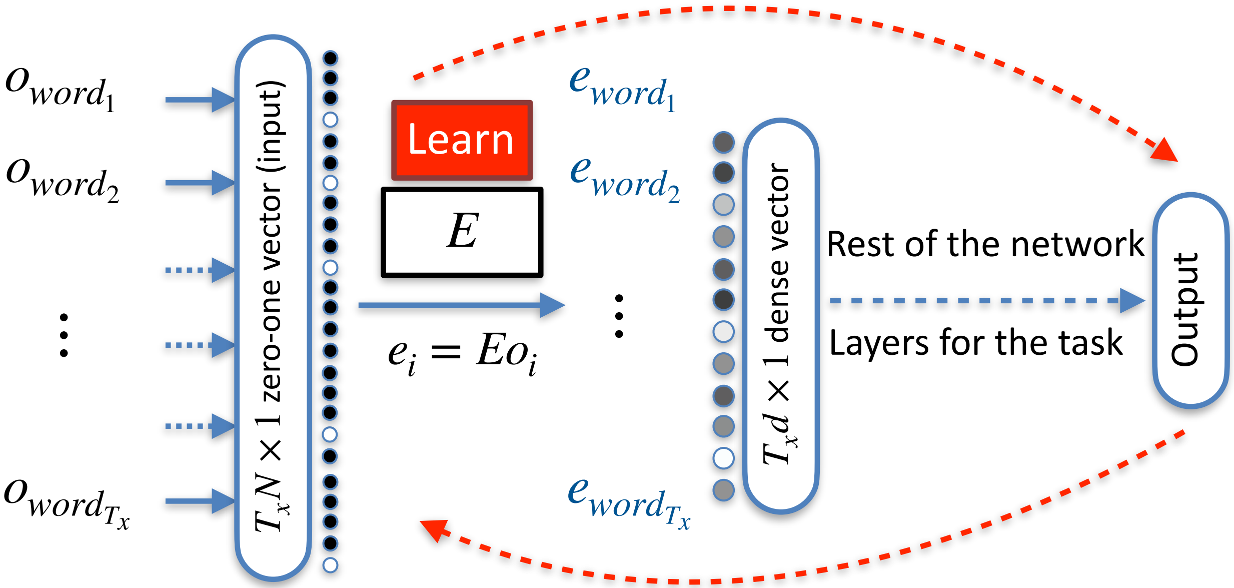
Learning word embeddings

White = 1; Black = 0, Grey $\in (0,1)$.

Add an embedding layer in the beginning

```
inputs = keras.Input(shape=(None,), dtype="int32")

# Embed each integer in a d-dimensional vector
x = layers.Embedding(N, d)(inputs)
```



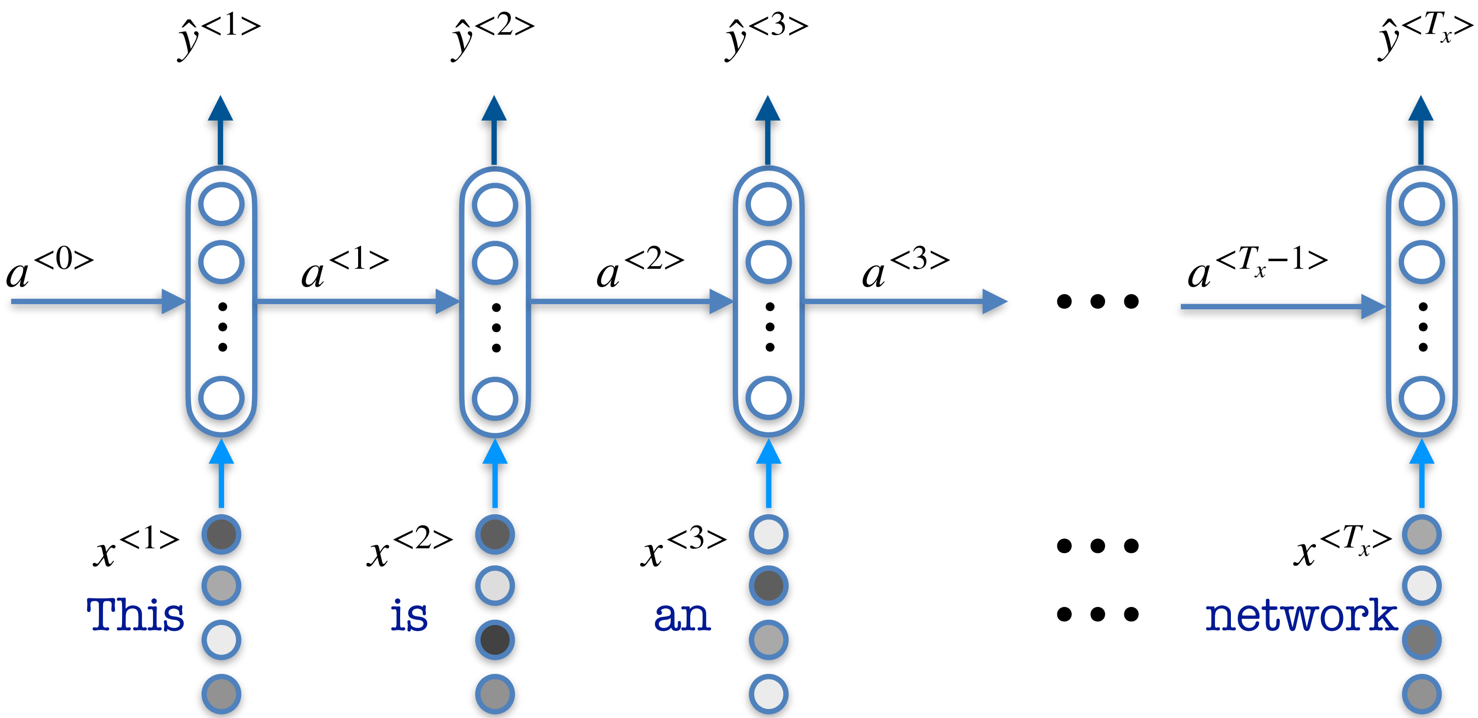
The embedding can be learnt as a byproduct of any task for which the model is trained

Input as one-hot vectors

We will learn more about word embeddings later.

The Recurrent Neural Network (RNN)

Output $\hat{y}$ = the sequence $\hat{y}^{<1>}, \cdots, \hat{y}^{<T_x>}$



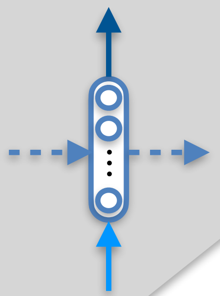
Input x : This is an example of a recurrent neural network.

In this example, length of $\hat{y}$ = length of x = T_x

But it can be different, as well as the architecture also may be somewhat different

An RNN Cell

What happens inside an RNN cell?



Parameters $W_{aa}, W_{ax}, W_{ya}, W_{yx}, b_a, b_y$ are same for all cells for $t = 1, \cdots, T_x$

$\hat{y}^{<t>}$

Output

$$\hat{y}^{<t>} = g_y(W_{ya}a^{<t-1>} + W_{yx}x^{<t>} + b_y)$$

Activation to the next cell

$$a^{<t>} = g_a(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$a^{<t-1>}$ $(n_a \times 1)$

$a^{<t>}$ $(n_a \times 1)$

$x^{<t>}$ $(n_x \times 1)$

- Input $x^{<t>}$: some vector representation (say, one-hot) of the input element (e.g. a word)

An RNN Cell: Details of the Operations

$$\overset{n_a \times 1}{a^{<t>}} = g_a(\underset{n_a \times n_a}{W_{aa}}\,\overset{n_a \times 1}{a^{<t-1>}} + \underset{n_a \times n_x}{W_{ax}}\,\overset{n_x \times 1}{x^{<t>}} + \underset{n_a \times 1}{b_a}) \equiv a^{<t>} = g_a\left(\underbrace{\overbrace{\left[W_{aa} \,\middle|\, W_{ax} \right]}^{W_a}}_{n_a \times (n_a + n_x)} \underbrace{\begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}}_{(n_a + n_x) \times 1} + b_a \right)$$

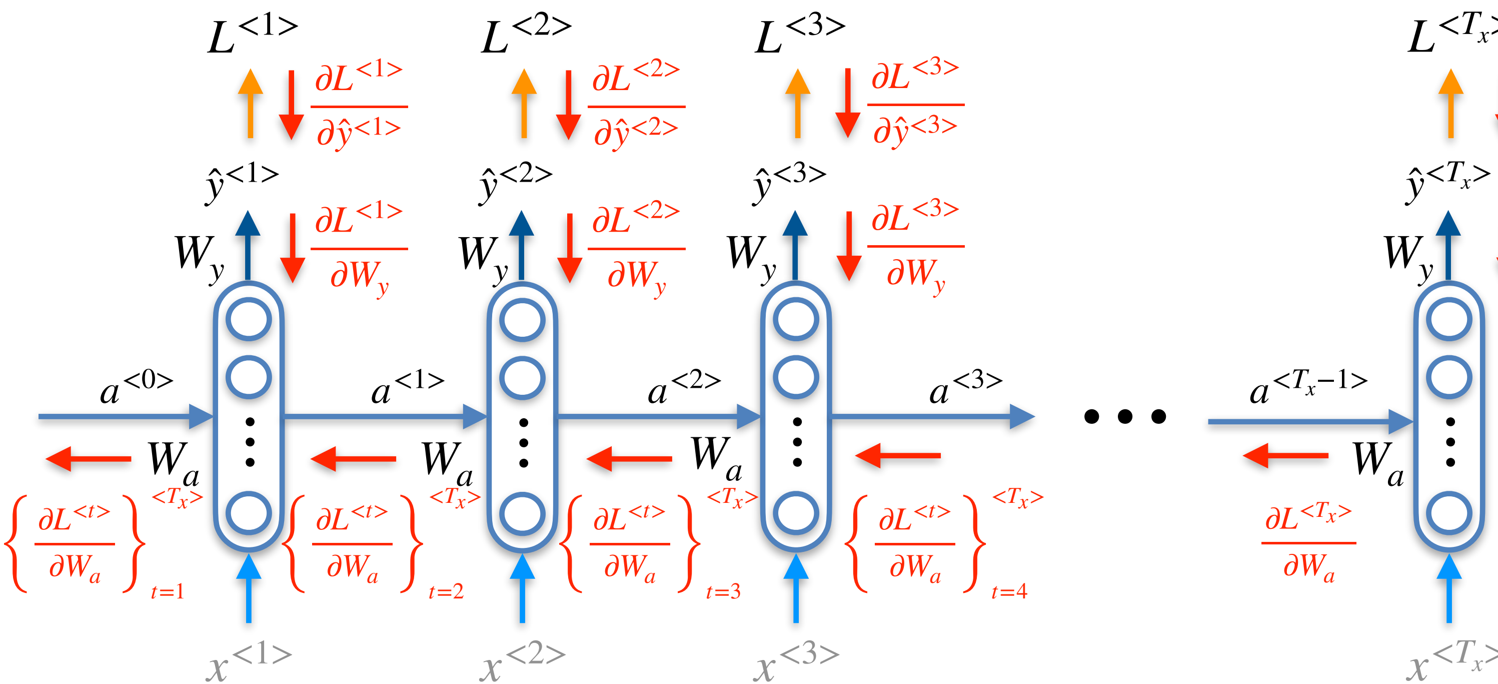
We can write

$$a^{<t>} = g_a(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

and, similarly

$$\hat{y}^{<t>} = g_y(W_y[a^{<t-1>}, x^{<t>}] + b_y)$$

Backpropagation through time



Note: this formulation will vary with changes in the architecture

Total loss: $L = \sum_{t=1}^{T_x} L^{<t>} = \sum_{t=1}^{T_x} L(\hat{y}^{<t>}, y^{<t>})$

$$\frac{\partial L}{\partial W_y} = \sum_{t=1}^{T_x} \frac{\partial L^{<t>}}{\partial W_y}$$

$$\frac{\partial L}{\partial W_a} = \sum_{t=1}^{T_x} \frac{\partial L^{<t>}}{\partial W_a}$$

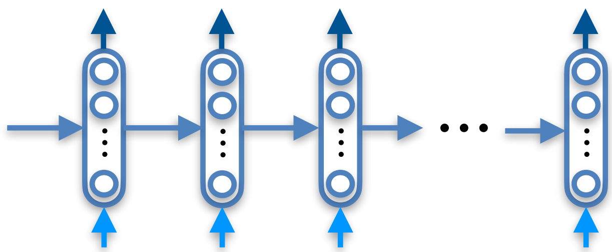
Similarly for b_y and b_a

Input x is a single training example. Update W_y, b_y, W_a, b_a happens after the complete backpropagation (possibly for all examples of a mini-batch)

Different RNN Architectures

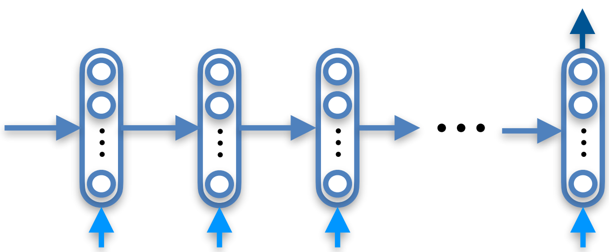
Many to Many

Named entity recognition, POS tagging, …



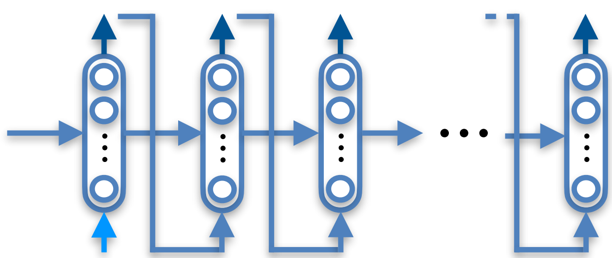
Many to One

Sentiment classification, …



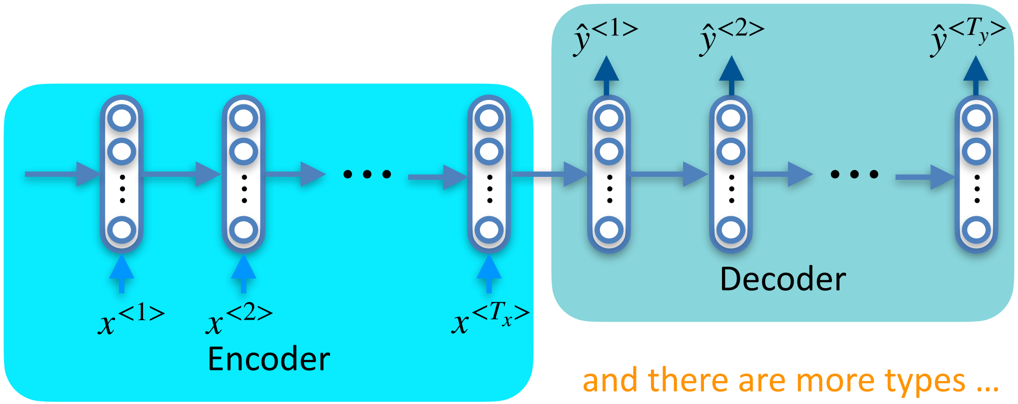
One to Many

Music generation, text generation, …



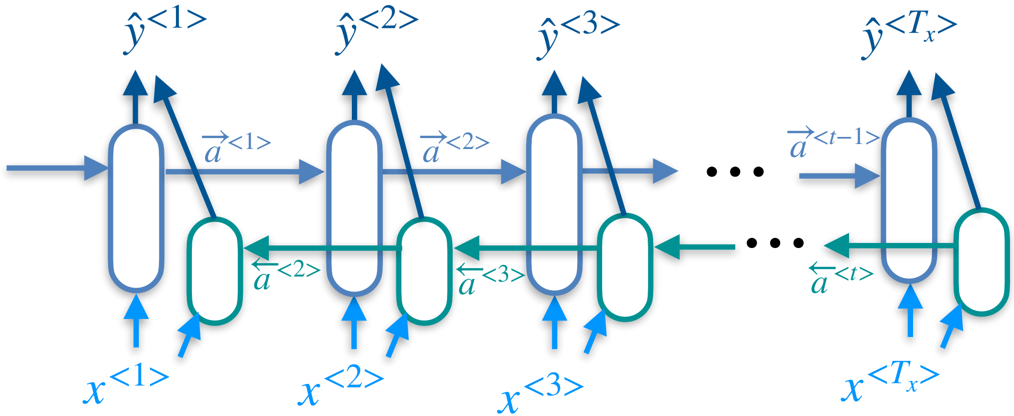
Many to Many (encoder - decoder)

Machine Translation, …



and there are more types …

Bidirectional RNN



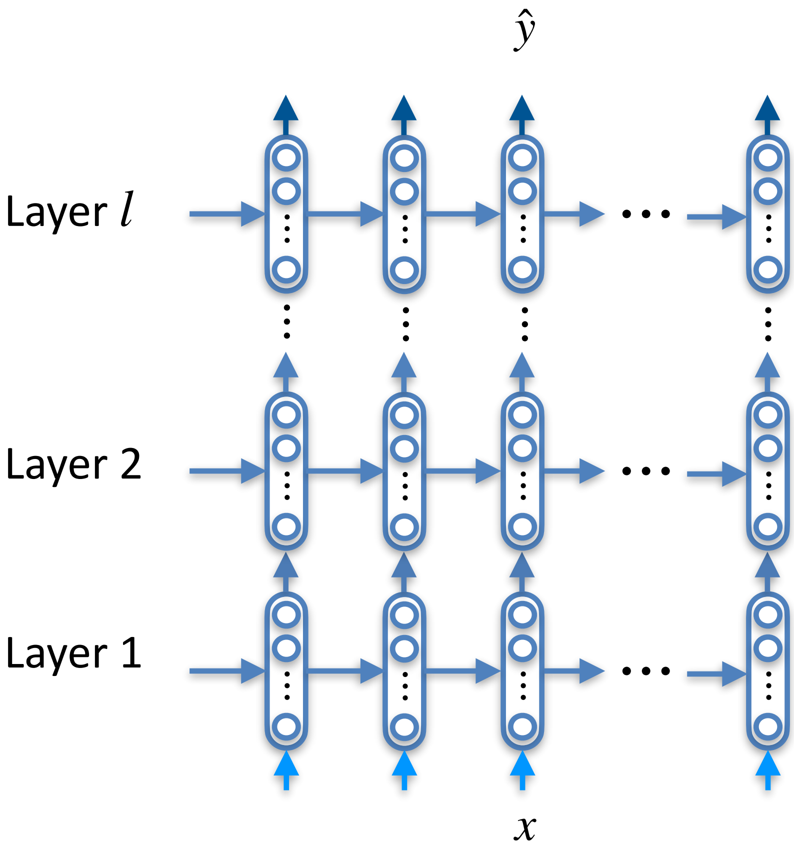
An acyclic graph

- Sometimes, earlier cells may require information from later cells to make sense
- Example: “Teddy bears are lovely gifts.” and “Teddy Roosevelt was a president of the United States of America.”
- Each output $\hat{y}^{<t>}$ can be obtained by both forward and backward cells

$$\hat{y}^{<t>} = g(W_y[\overrightarrow{a}^{<t>}, \overleftarrow{a}^{<t>}, x^{<t>}] + b_y)$$

- Usually the forward-RNN and backward-RNN are trained together by stacking the hidden state activations $\overrightarrow{a}^{<t>}$ and $\overleftarrow{a}^{<t>}$
- Bidirectional RNNs are widely used in practice

Deep RNN with multiple layers



- Multiple layers of RNN can be stacked to make a deep RNN
- Also called *stacked RNNs*
- Intuition: lower-level layers capture lower-level features, higher-level layers capture higher level features
- In practice, number of layers for multi-layer RNNs (2-6) are not as much as ConvNets (can be 150 - 200)
- Transformer based networks (e.g. BERT) can be up to 24 layers

References

- Andrew Ng's lectures on *Sequence Models*: www.coursera.org/learn/nlp-sequence-models
- Chris Manning, Abigail See and other TAs. *Natural Language Processing with Deep Learning.* Stanford University Course (CS224n), Winter 2019. web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. www.deeplearningbook.org