Deep Learning

Recurrent Neural Network

Geoffroy Peeters

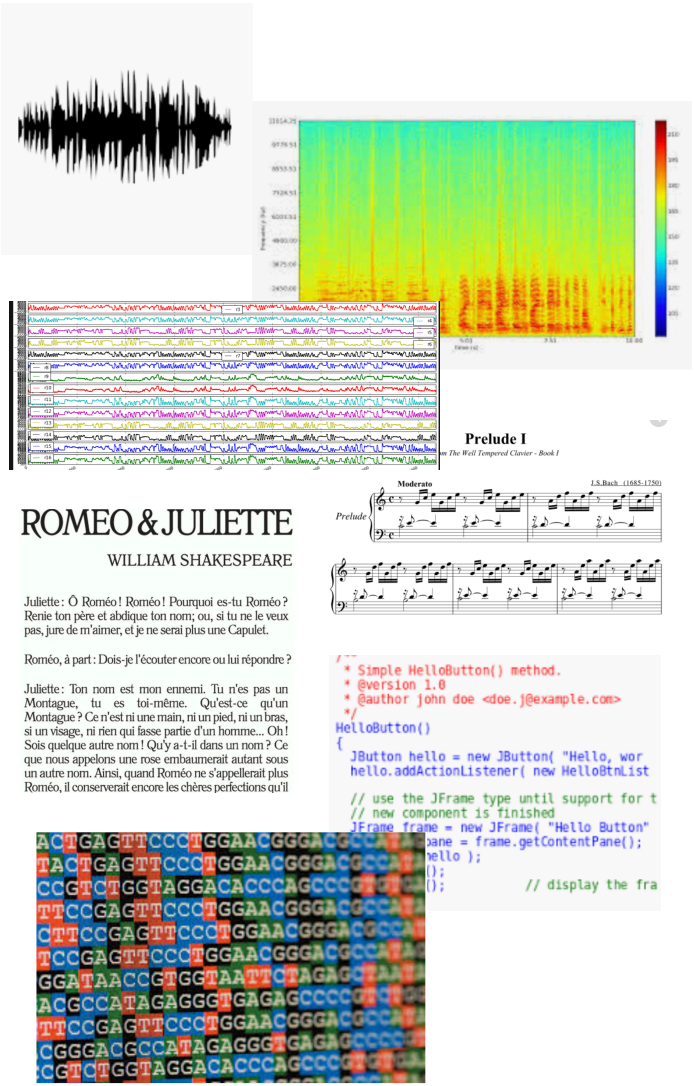
contact: geoffroy.peeters@telecom-paris.fr
Télécom-Paris, IP-Paris, France

Recurrent Neural Network for Sequential Data

What are sequential data ?

- **Sequential data are any type of data for which the order matters**
- Examples**:**
  - **Time series:**
    - Audio (sequence of sound pressures or sequence of Fourier transforms)
    - Sequence of musical notes
    - Video (sequence of images)
    - Sensors (heart-beat, plane altitude)
    - Stock market
  - **Ordered**: (but not with time)
    - Text/Words
    - Genes/ADN

- **Common models:**
  - Vector linear autoregression (VAR)
  - Markov model (Discrete-State HMMs)
  - Kalman filters (Continuous-State HMMs)
  - ...



Recurrent Neural Network for Sequential Data

Notations

- Inputs:
  - $\mathbf{x}^{\langle t \rangle}$: input vector $\mathbf{x}$ at time t (of dimension $n^{[0]}$)
  - $\{\mathbf{x}^{\langle 1 \rangle}, \mathbf{x}^{\langle 2 \rangle}, \ldots, \mathbf{x}^{\langle T_x \rangle}\}$: a sequence of inputs of length T_x
- Outputs:
  - $\mathbf{y}^{\langle t \rangle}$: output vector $\mathbf{y}$ at time t
  - $\{\mathbf{y}^{\langle 1 \rangle}, \mathbf{y}^{\langle 2 \rangle}, \ldots, \mathbf{y}^{\langle T_y \rangle}\}$: a sequence of outputs of length T_y
- T_x and T_y can be of different lengths
- $x^{(i)\langle t \rangle}$ is the i^{th} example in the training-set
  - ${T_x}^{(i)}$ the length of the i^{th} input sequence
  - ${T_y}^{(i)}$ the length of the i^{th} output sequence

Notations

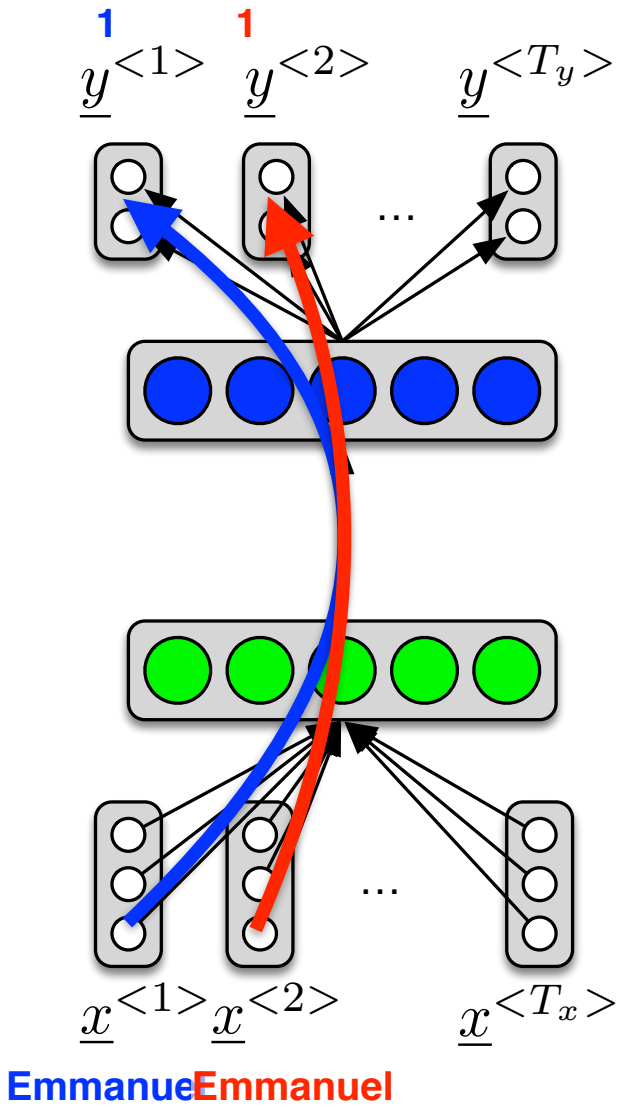
- Different types of output $\mathbf{y}$:
  - Continuous values $\mathbf{y} \in \mathbb{R}^{n_y}$ (Activation: linear, Loss: MSE)
  - Binary-classes $\mathbf{y} \in \{0,1\}$ (Activation: sigmoid, Loss: Binary-Cross-Entropy)
  - Multi-classes $y \in \{1 \ldots K\} \Rightarrow \mathbf{y} \in \{0,1\}^K$ (Activation: softmax, Loss: Cross-Entropy)
- Different types of input $\mathbf{x}$
  - Continuous values $\mathbf{x} \in \mathbb{R}^{n_x}$
  - Categorical values $x \in \{1 \ldots K\} \Rightarrow$ one hot-encoding $\mathbf{x} \in \{0,1\}^K$
  - **How do we process categorical values as input ?**
    - One-hot-encoding
    - Embedding

- **Example**: categorical inputs/ binary outputs
  - "Named Entity Recognition"

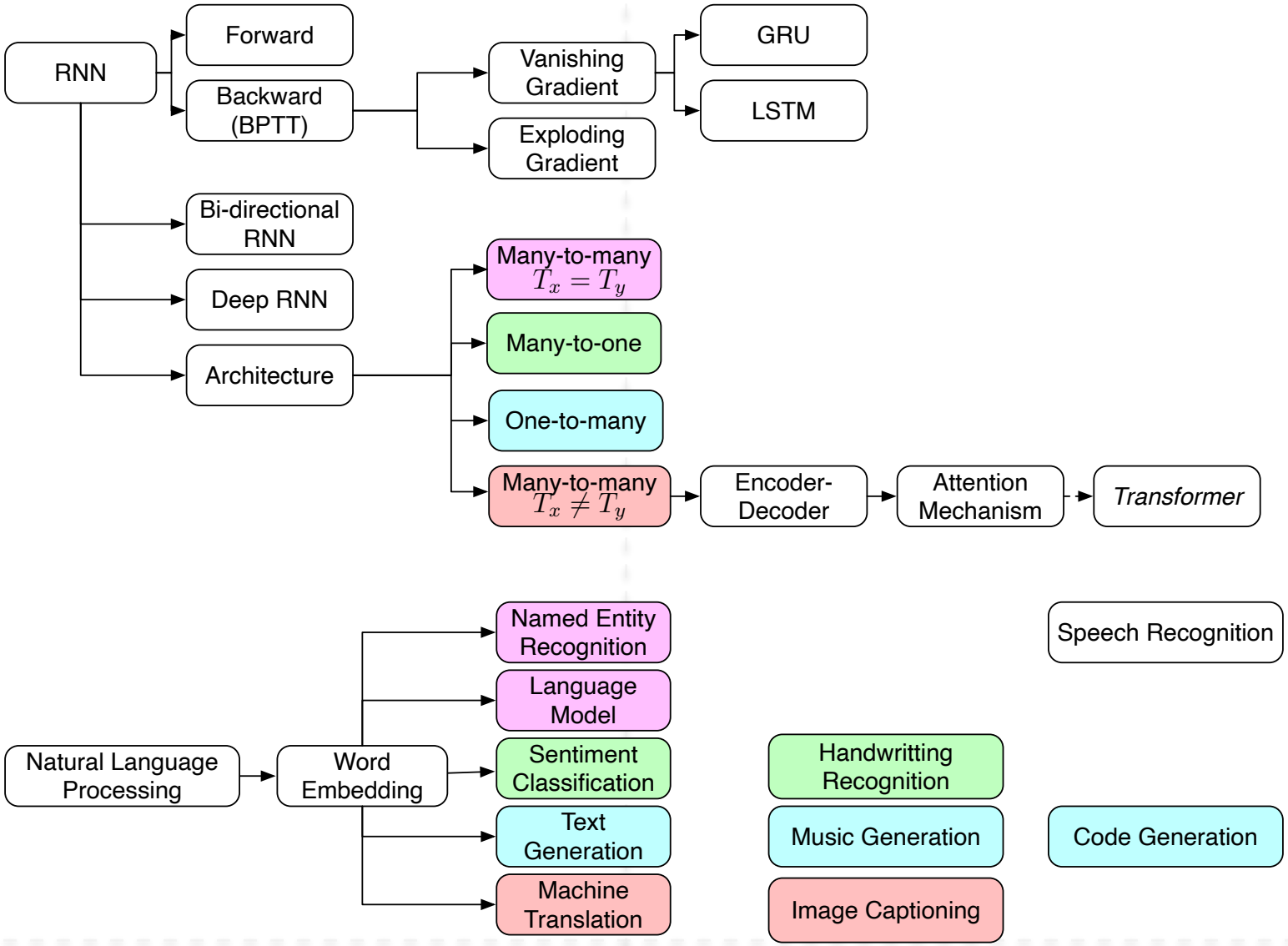
{x}	$x^{<1>}$	$x^{<2>}$	…				$x^{<7>}$
	Emmanuel	Macron	is	the	president	of	France
{y}	$y^{<1>}$	$y^{<2>}$	…				$y^{<7>}$
	1	1	0	0	0	0	1

Recurrent Neural Network for Sequential Data

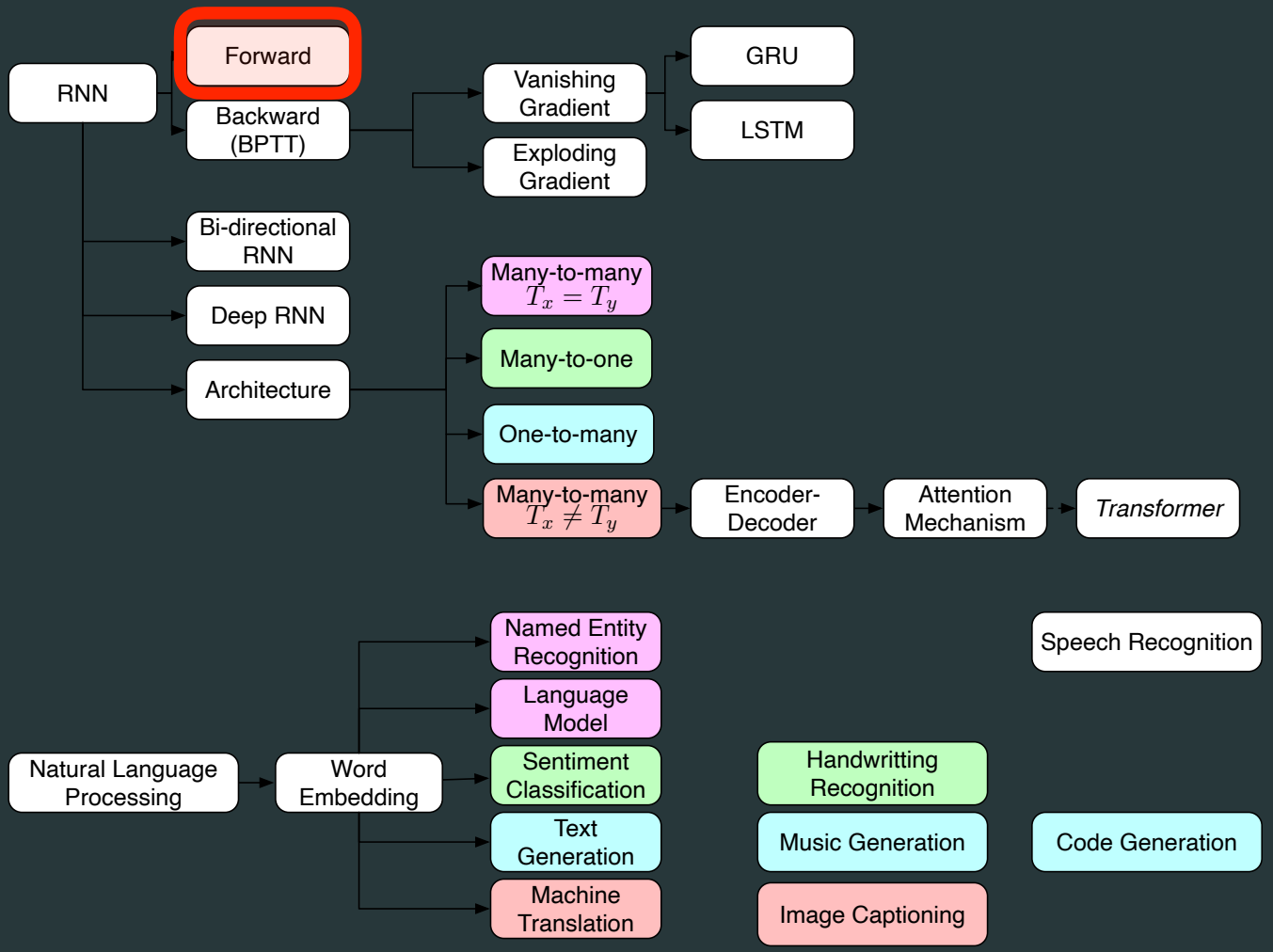
- **Why not using Multi-Layer-Perceptron ?**
  - We could indeed represent an input sequence $\{\mathbf{x}^{\langle 1 \rangle}, \ldots, \mathbf{x}^{\langle T_x \rangle}\}$ as a large-vector of dimension $(n^{[0]} T_x)$
    - but requires a huge input dimension
    - but $T_x^{(i)}$ can be different for each input sequence, $\Rightarrow$ the dimension of the input would be different for each $(n^{[0]} T_x^{(i)})$
  - Solution: zero-padding ? $(n^{[0]} \max_i T_x^{(i)})$
    - still requires a huge input dimension
    - the network will have to learn different weights for the same dimension $n^{[0]}$ at various time $\langle t \rangle$ in the sequence $\Rightarrow$ no weight sharing !
  - **Solutions:**
    - Recurrent Neural Network
    - 1D Convolutional Neural Network (convolution only over time)



Overview

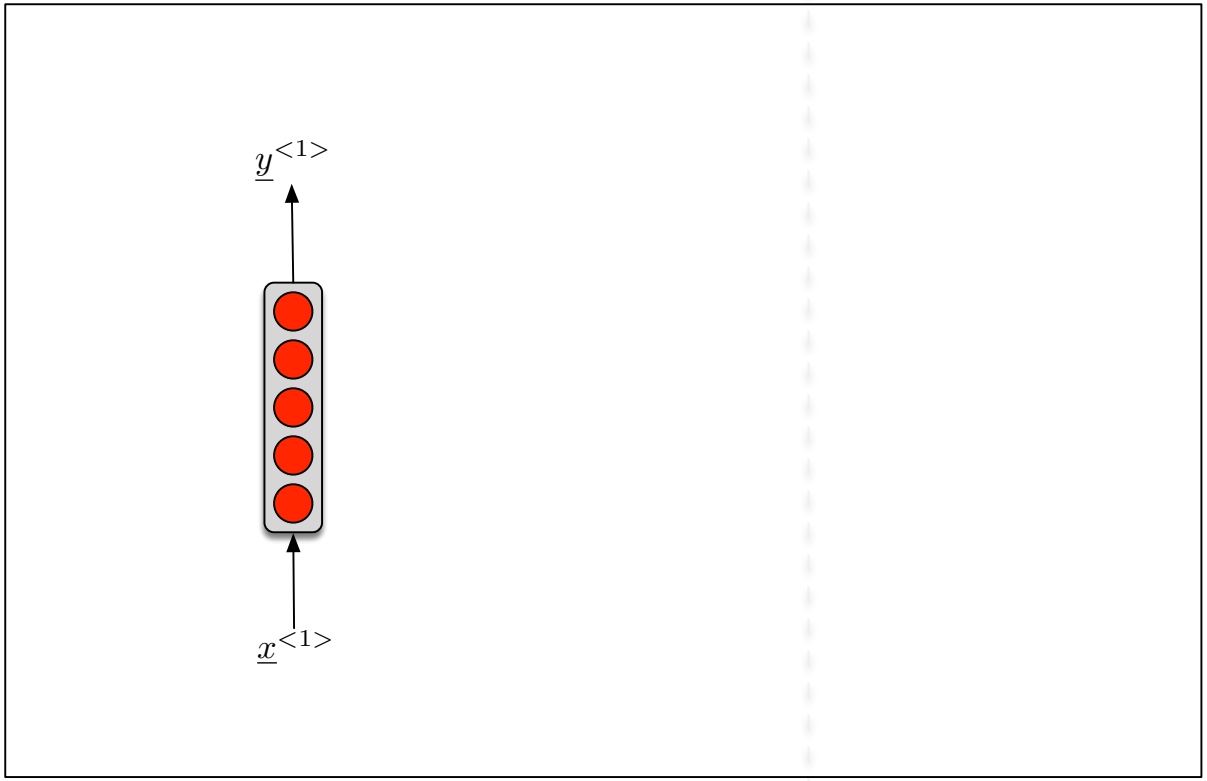

RNN
Forward
Backward (BPTT)
Vanishing Gradient
Exploding Gradient
GRU
LSTM
Bi-directional RNN
Deep RNN
Architecture
Many-to-many $T_x = T_y$
Many-to-one
One-to-many
Many-to-many $T_x \neq T_y$
Encoder-Decoder
Attention Mechanism
Transformer
Natural Language Processing
Word Embedding
Named Entity Recognition
Language Model
Sentiment Classification
Text Generation
Machine Translation
Speech Recognition
Handwritting Recognition
Music Generation
Image Captioning
Code Generation

Recurrent Neural Network for Sequential Data



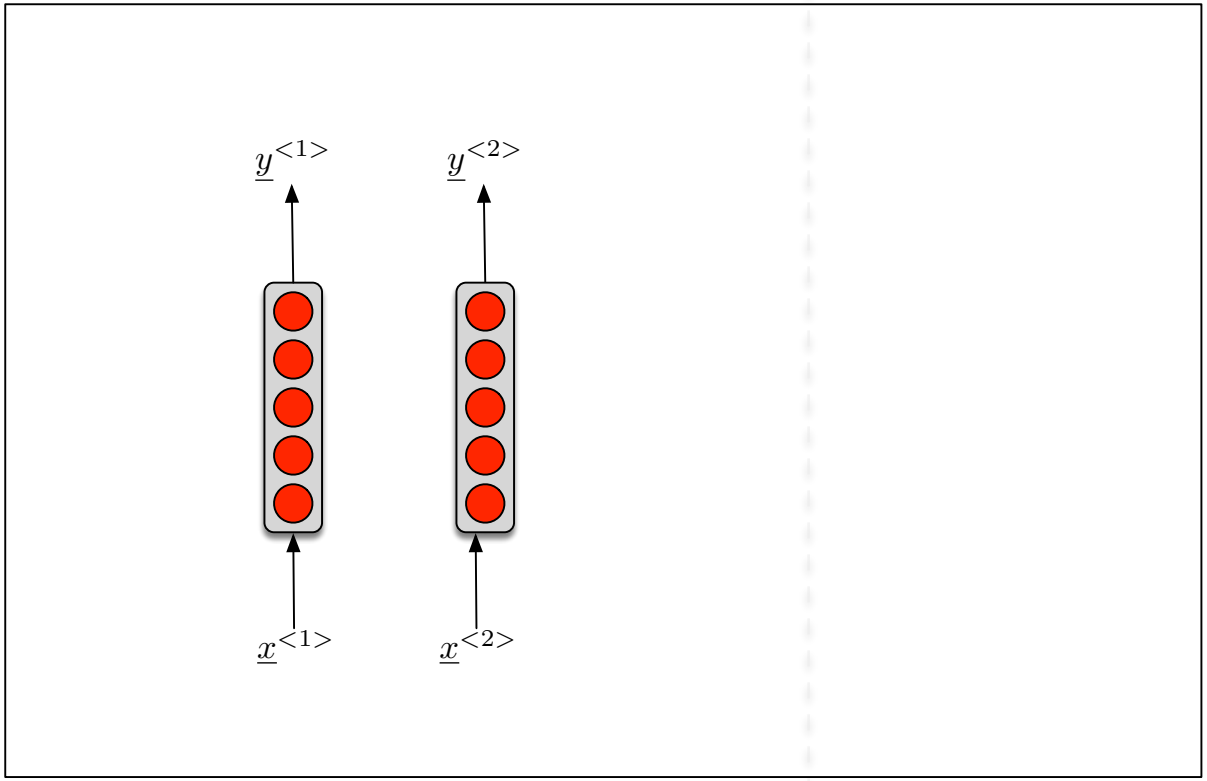
Recurrent Neural Network for Sequential Data

What is an RNN ?



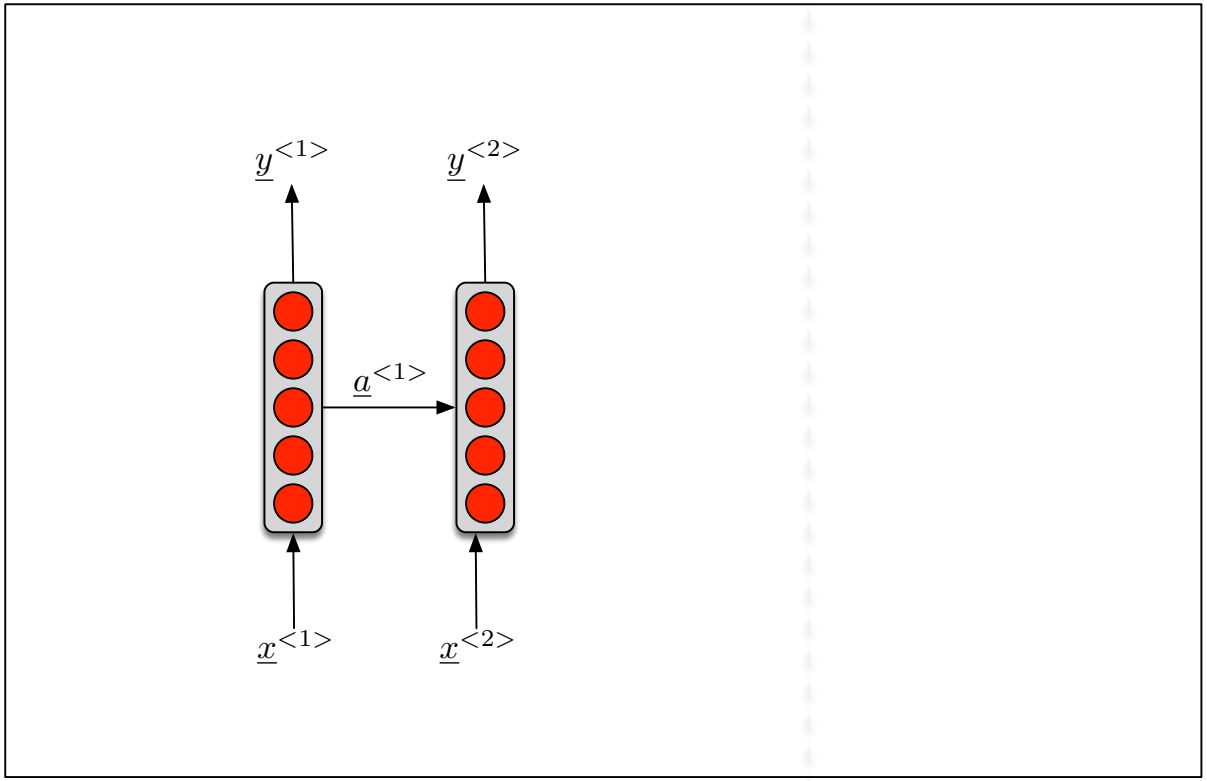
Recurrent Neural Network for Sequential Data

What is an RNN ?



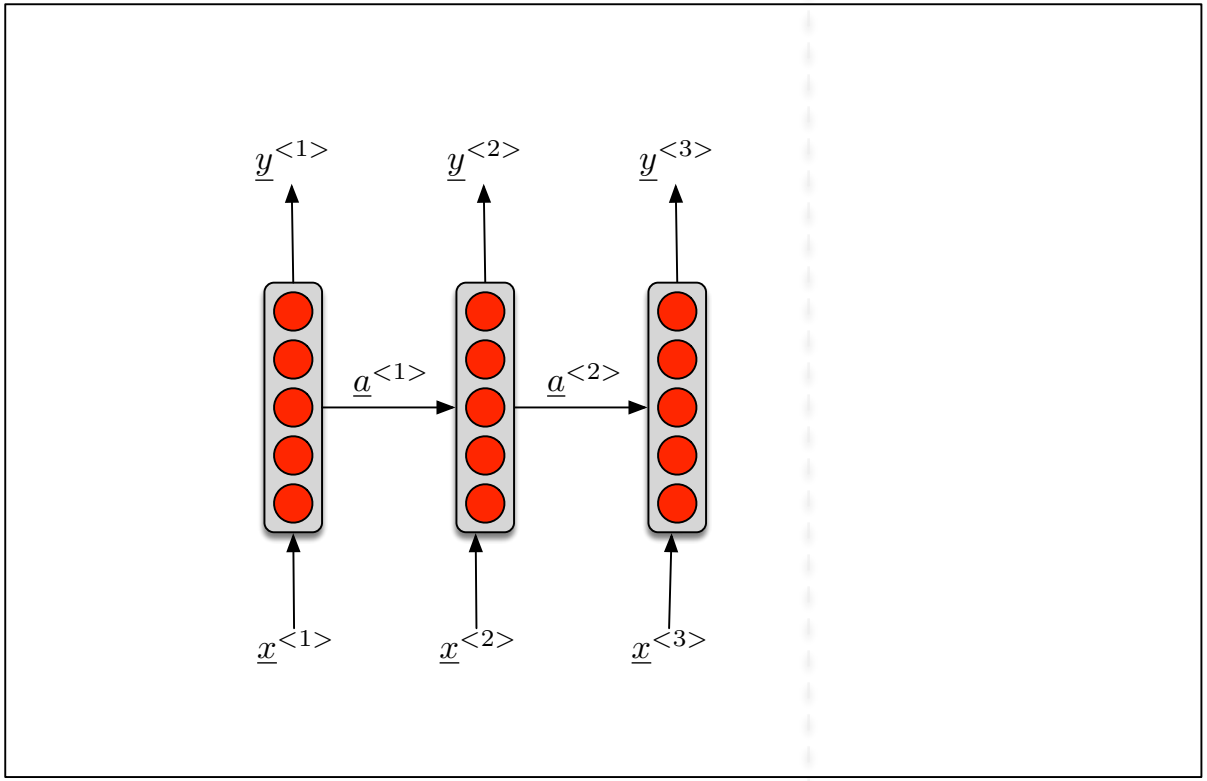
Recurrent Neural Network for Sequential Data

What is an RNN ?



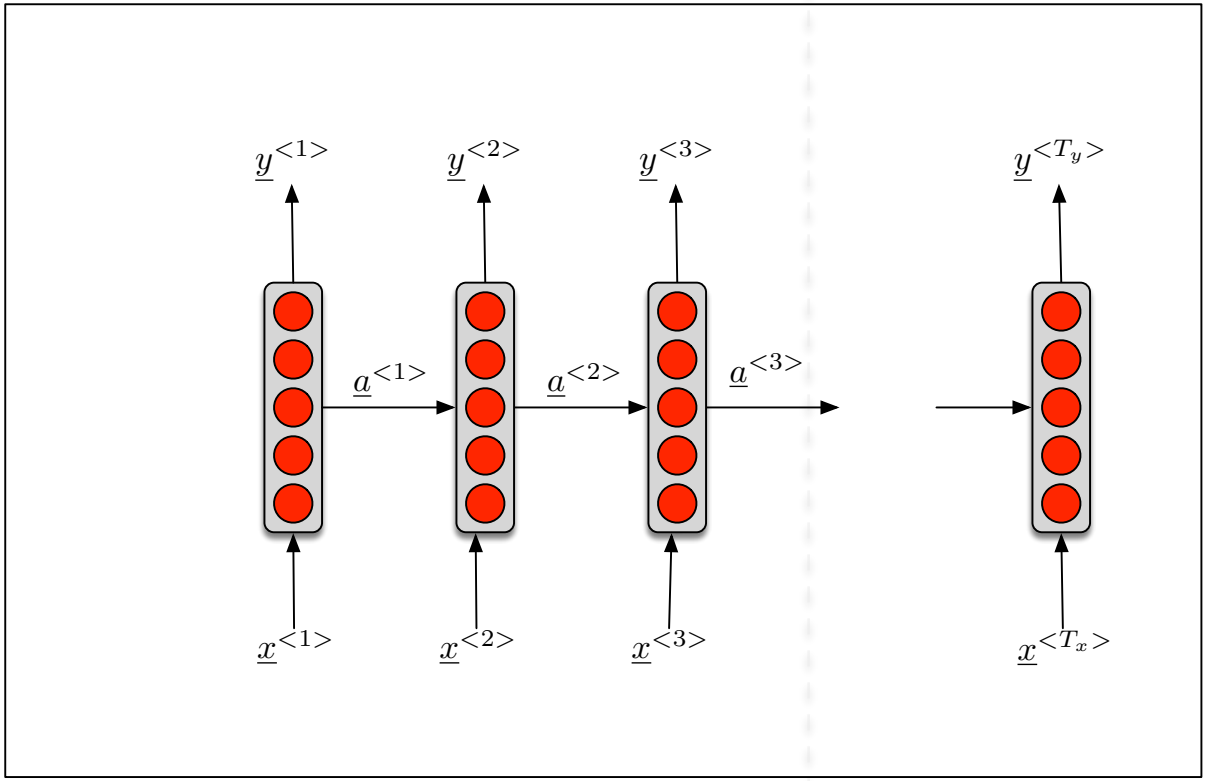
Recurrent Neural Network for Sequential Data

What is an RNN ?



Recurrent Neural Network for Sequential Data

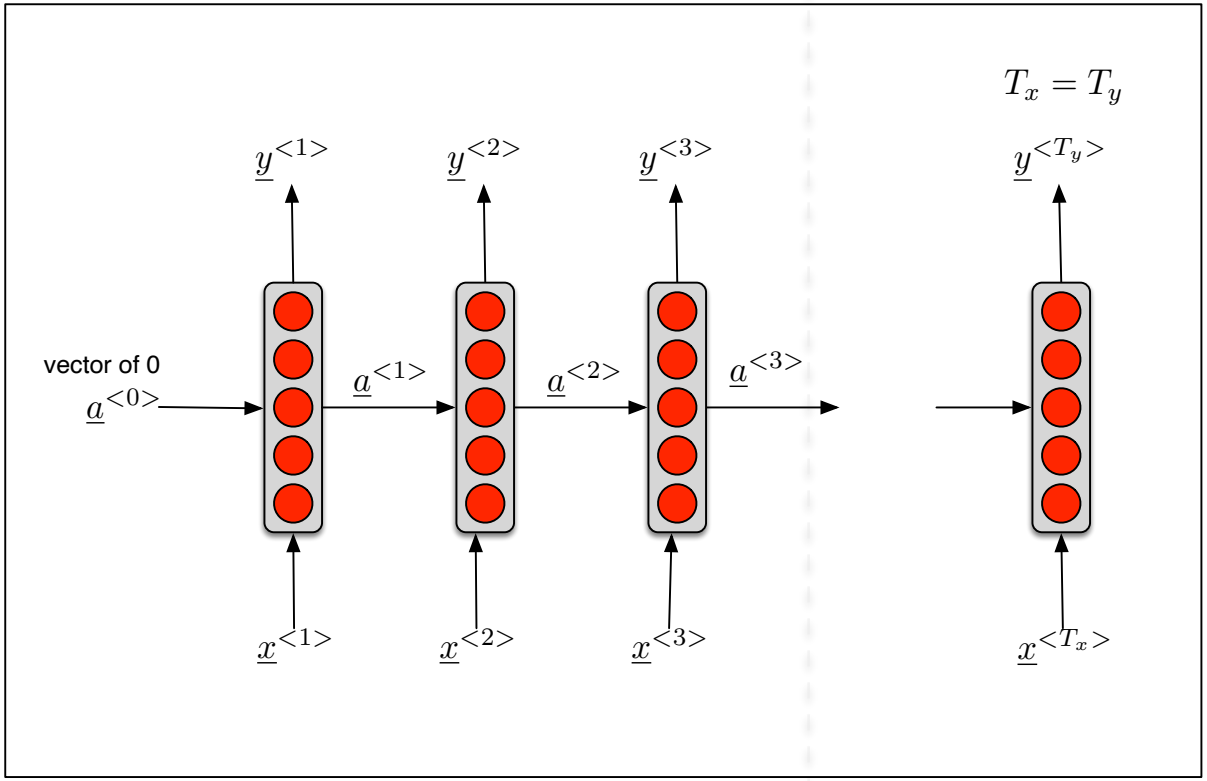
What is an RNN ?



– At each time step $\langle t \rangle$, the RNN gets the activation $a^{\langle t-1 \rangle}$ from the previous time $\langle t-1 \rangle$

Recurrent Neural Network for Sequential Data

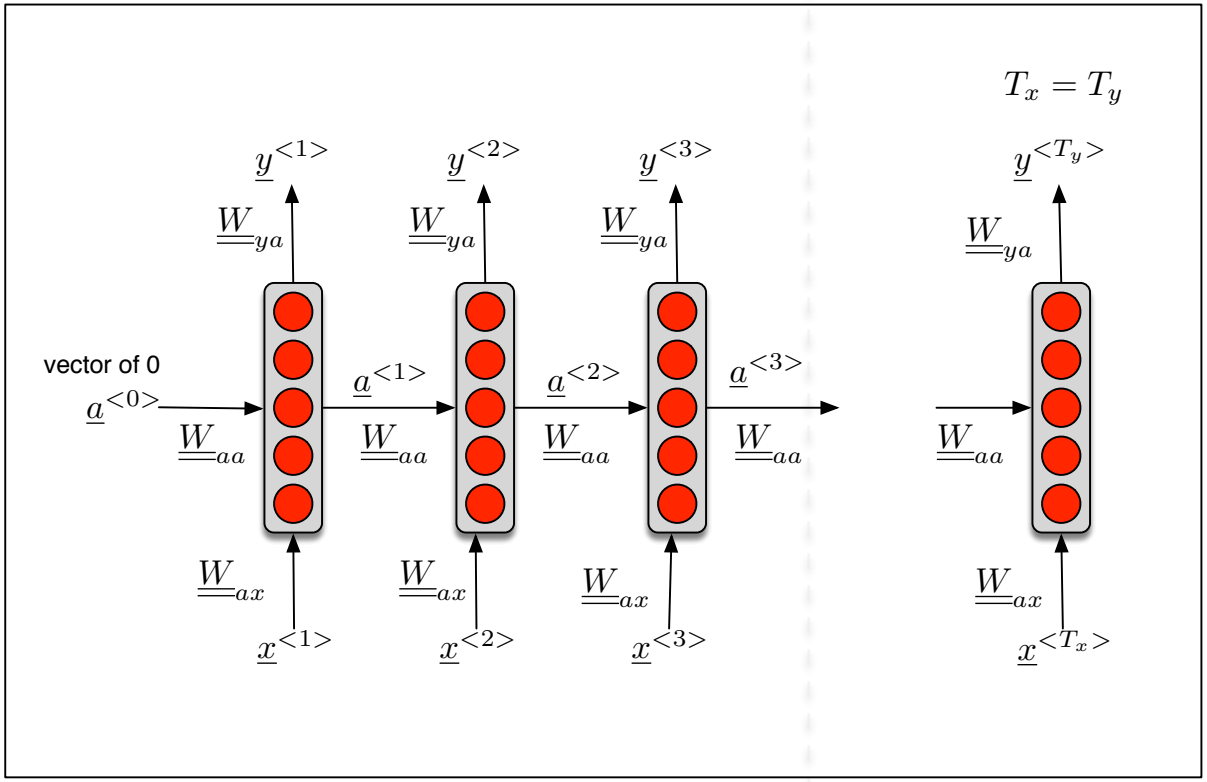
What is an RNN ?



- Initialise the first activation with $a^{\langle 0 \rangle} = 0$

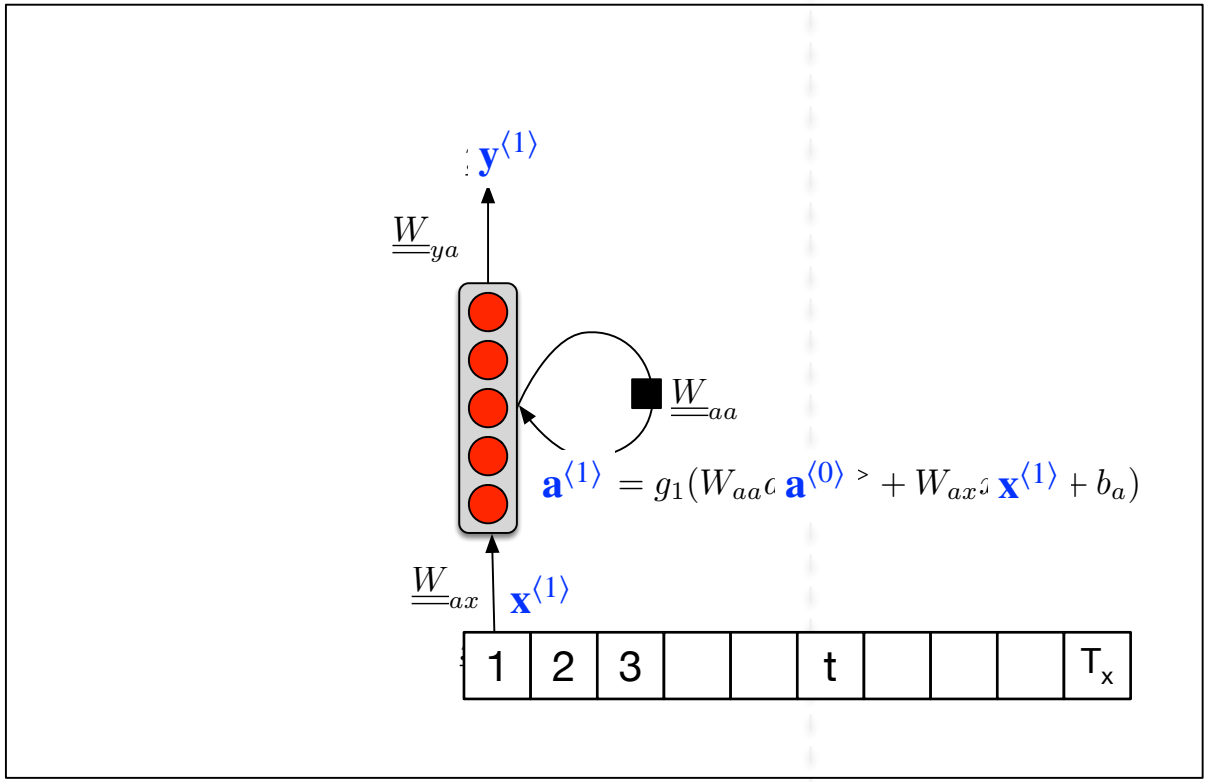
Recurrent Neural Network for Sequential Data

What is an RNN ?



- Weight matrices: $\mathbf{W}_{ax}$ from $\mathbf{x}^{\langle t \rangle} \rightarrow \mathbf{a}^{\langle t \rangle}$, $\mathbf{W}_{aa}$ from $\mathbf{a}^{\langle t-1 \rangle} \rightarrow \mathbf{a}^{\langle t \rangle}$, $\mathbf{W}_{ay}$ from $\mathbf{a}^{\langle t \rangle} \rightarrow \mathbf{y}^{\langle t \rangle}$
- They are shared from all time-steps

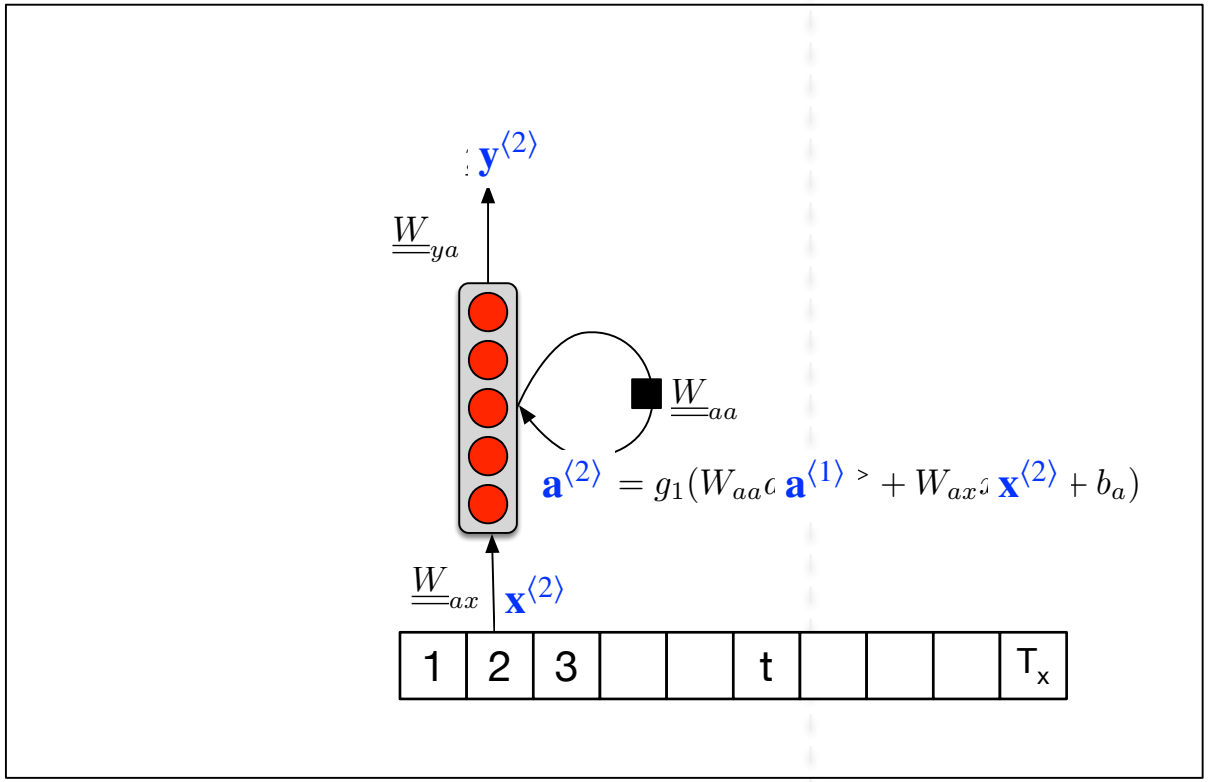
What is an RNN ?



– **"Rolled"** version of RNN (the real one, only one network)

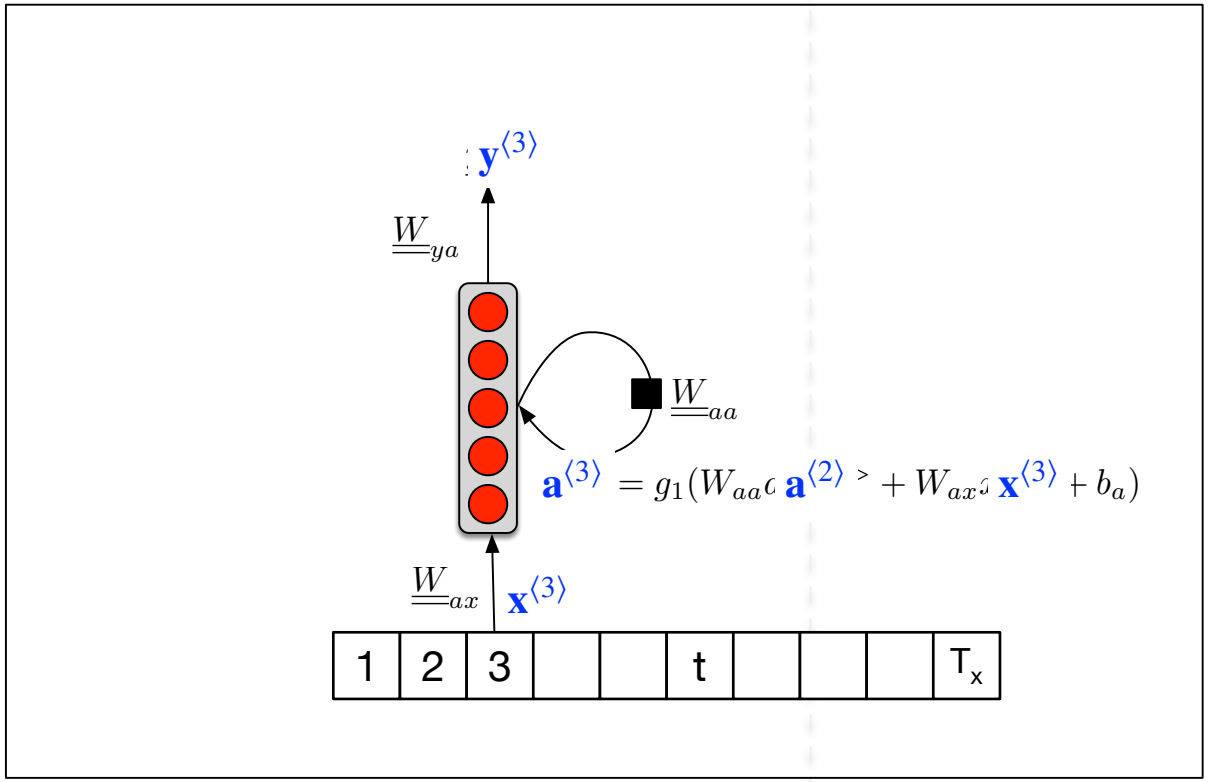
Recurrent Neural Network for Sequential Data

What is an RNN ?



– **"Rolled"** version of RNN (the real one, only one network)

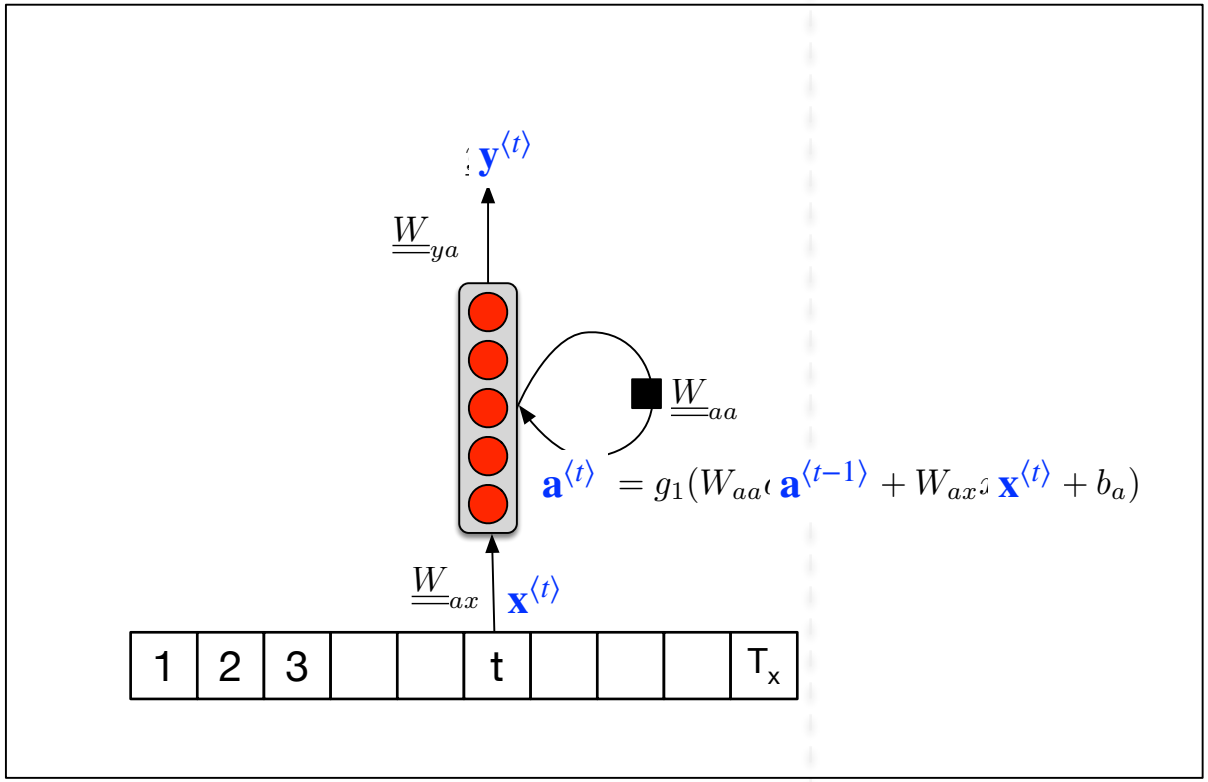
What is an RNN ?



– **"Rolled"** version of RNN (the real one, only one network)

Recurrent Neural Network for Sequential Data

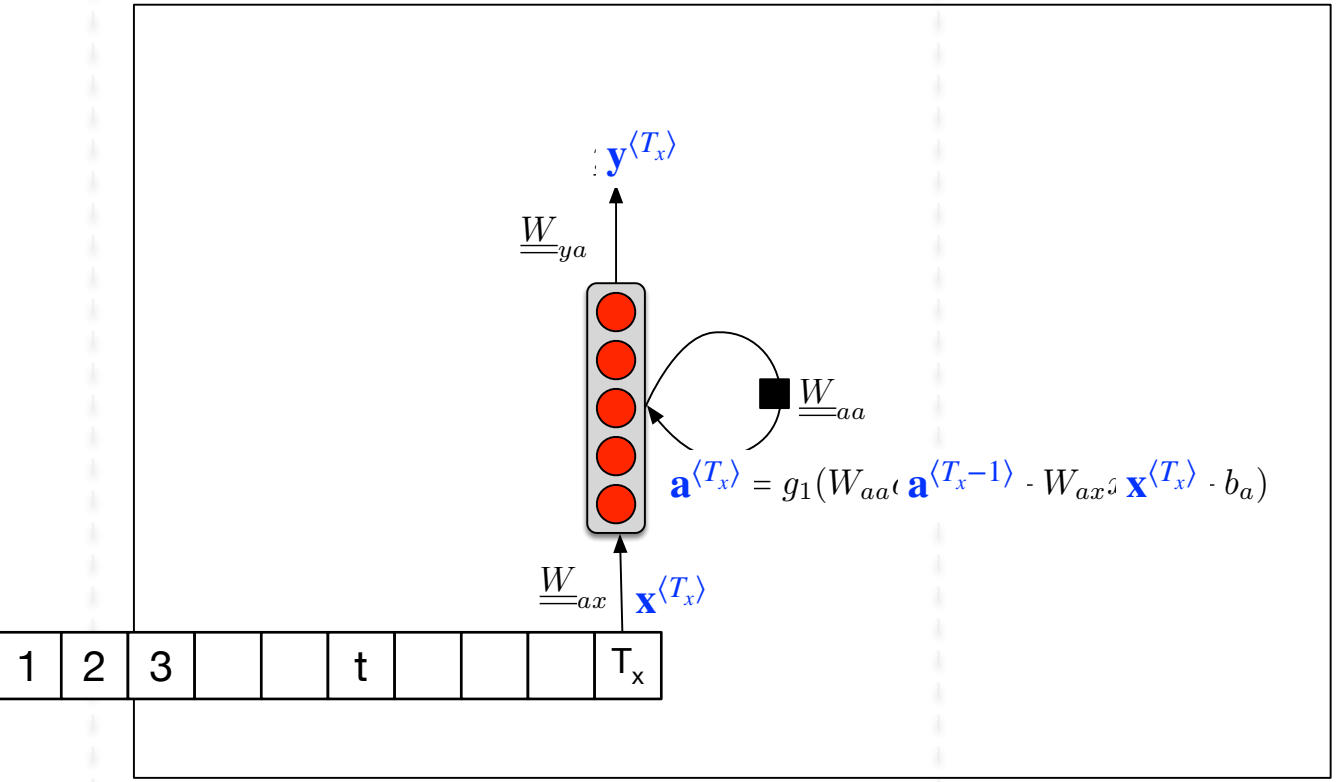
What is an RNN ?



– **"Rolled"** version of RNN (the real one, only one network)

Recurrent Neural Network for Sequential Data

What is an RNN ?



– **"Rolled"** version of RNN (the real one, only one network)

Recurrent Neural Network for Sequential Data

What is an RNN ?

- **Forward propagation**

$$\mathbf{a}^{\langle 0\rangle} = 0$$

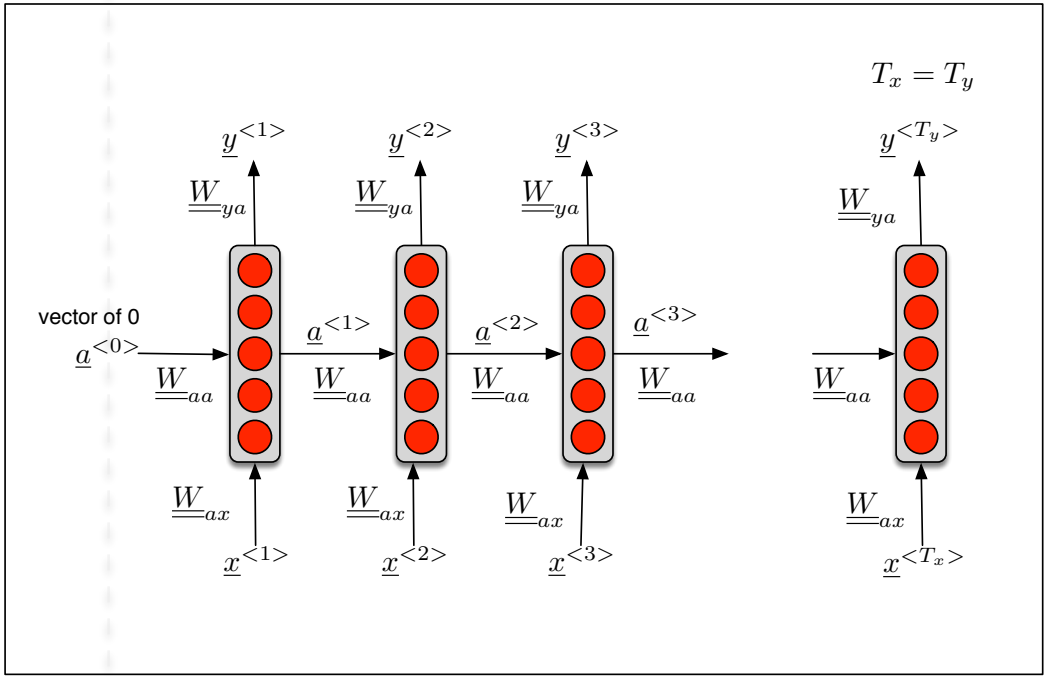
$$\mathbf{a}^{\langle 1\rangle} = g_1\left(\mathbf{W}_{aa}a^{\langle 0\rangle} + \mathbf{W}_{ax}x^{\langle 1\rangle} + \mathbf{b}_a\right)$$

$$\hat{\mathbf{y}}^{\langle 1\rangle} = g_2\left(\mathbf{W}_{ya}\mathbf{a}^{\langle 1\rangle} + \mathbf{b}_y\right)$$

$$\cdots$$

$$\mathbf{a}^{\langle t\rangle} = g_1\left(\mathbf{W}_{aa}\mathbf{a}^{\langle t-1\rangle} + \mathbf{W}_{ax}\mathbf{x}^{\langle t\rangle} + \mathbf{b}_a\right)$$

$$\hat{\mathbf{y}}^{\langle t\rangle} = g_2\left(\mathbf{W}_{ya}\mathbf{a}^{\langle t\rangle} + \mathbf{b}_y\right)$$



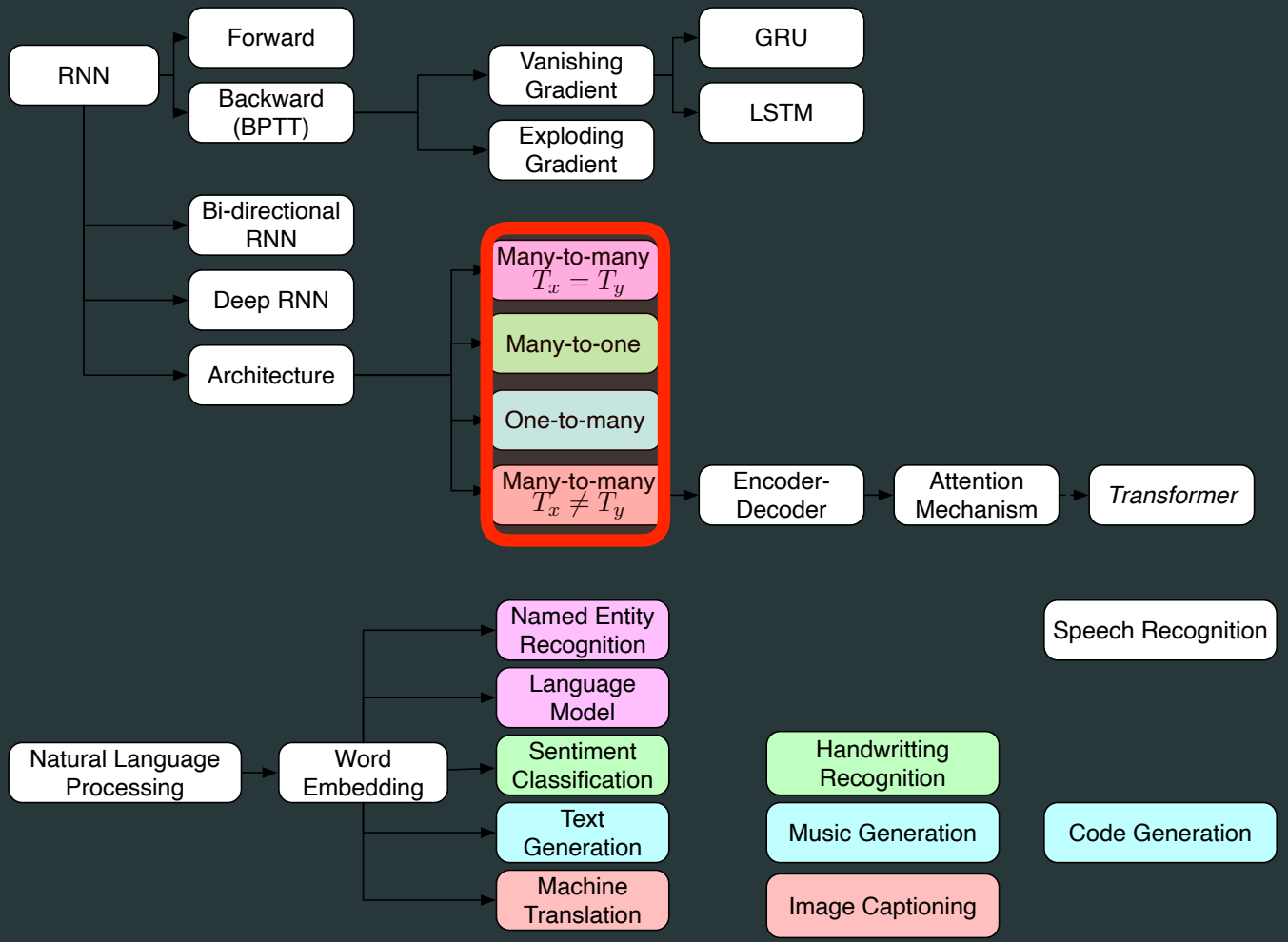
- **Compact notations**

$$\underset{(n_a,n_a)}{\mathbf{W}_{aa}} \mid \underset{(n_a,n_x))}{\mathbf{W}_{ax}} \begin{bmatrix} \mathbf{a}^{\langle t-1\rangle}_{(n_a,m)} \\ \mathbf{x}^{\langle t\rangle}_{(n_x,m)} \end{bmatrix}$$

$$\mathbf{a}^{\langle t\rangle} = g_1\left(\mathbf{W}_a[\mathbf{a}^{\langle t-1\rangle}; \mathbf{x}^{\langle t\rangle}] + \mathbf{b}_a\right)$$

$$\hat{\mathbf{y}}^{\langle t\rangle} = g_2\left(\mathbf{W}_y\mathbf{a}^{\langle t\rangle} + \mathbf{b}_y\right)$$

Various types of sequential data



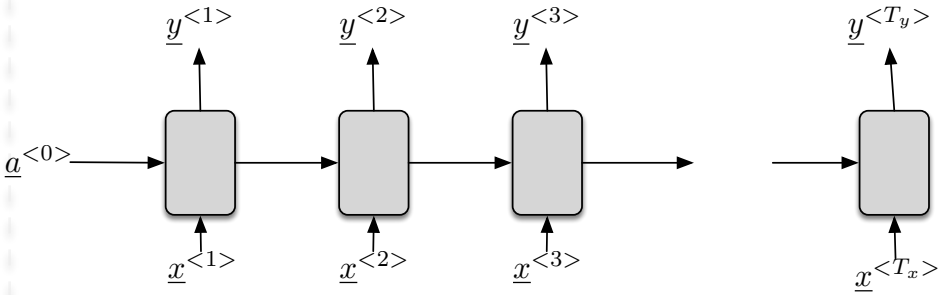
Various types of sequential data

Many-to-Many : $T_x = T_y$.

Input $\underline{x}$	Output $\underline{y}$	Type	Examples
sequence $T_x > 1$	sequence $T_y = T_x$	Many-To-Many	Named entity recognition

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell–Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

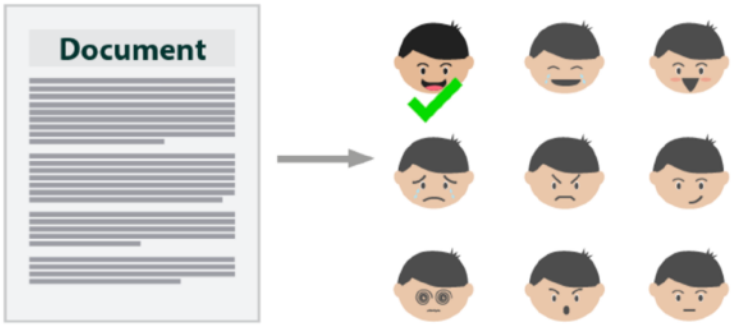
Tag colours:
LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE

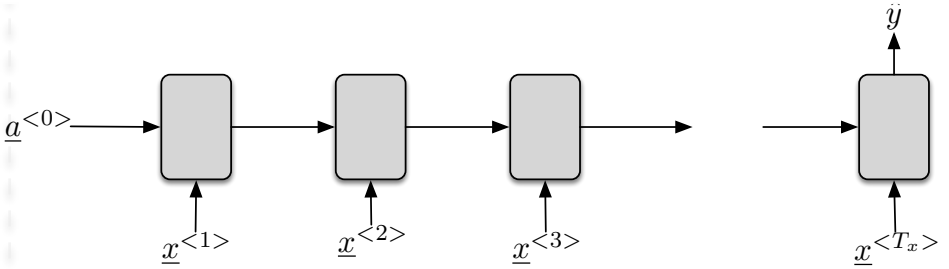


Various types of sequential data

Many-to-One : $T_x > 1, T_y = 1$.

Input $\underline{x}$	Output $\underline{y}$	Type	Examples
sequence $T_x > 1$	single $T_y = 1$	Many-To-One	Sentiment analysis, Video activity detection



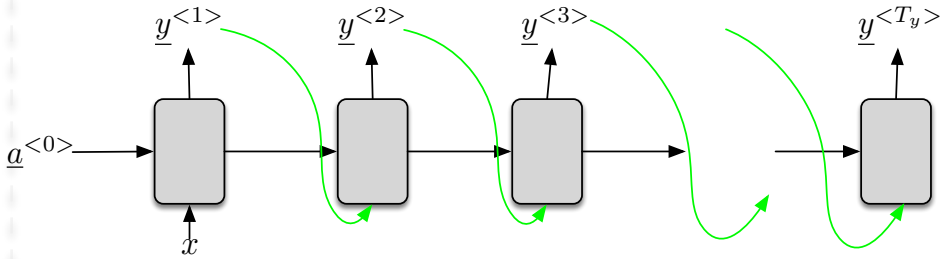


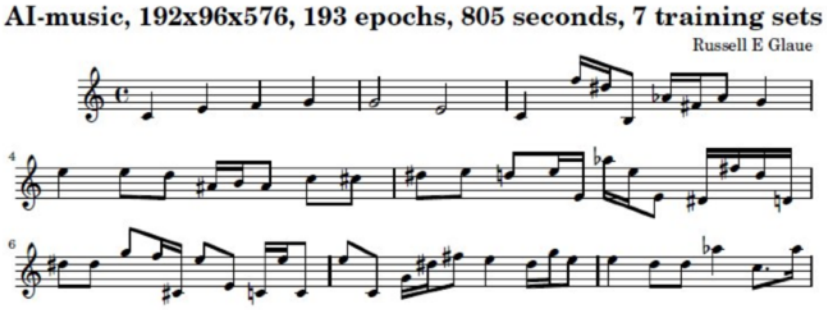
Various types of sequential data

One-to-Many : $T_x = 1, T_y > 1$.

Input $\underline{x}$	Output $\underline{y}$	Type	Examples
single $T_x = 1$	sequence, $T_y > 1$	One-To-Many	Text generation, music generation

```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be said to be directly in Cantonese
Communication, which followed a ceremony and set inspired prison, training. The
emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest of India with the conflict.
Copyright was the succession of independence in the slop of Syrian influence that
was a famous German movement based on a more popular servicious, non-doctrinal
and sexual power post. Many governments recognize the military housing of the
[[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those Socialism's rule,
was starting to signing a major tripad of aid exile.]]
```

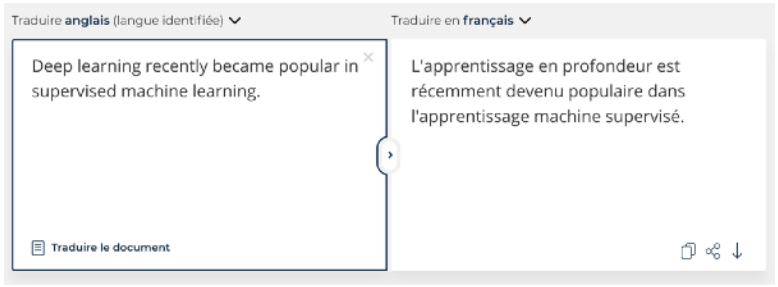


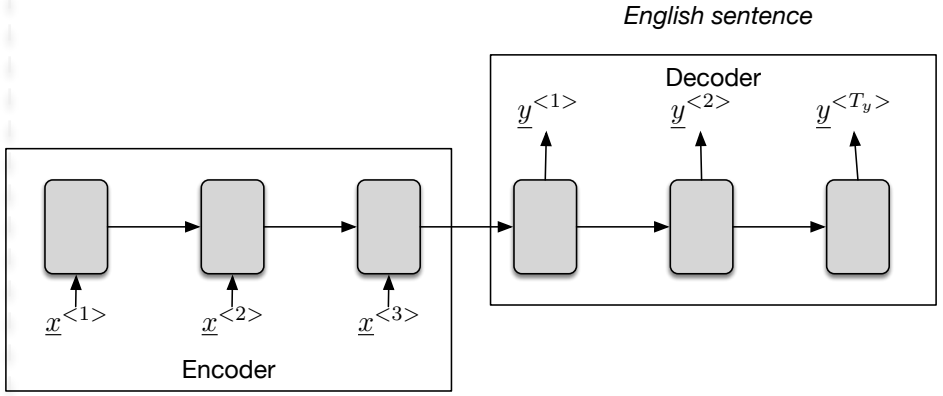


Various types of sequential data

Many-to-Many : $T_x \neq T_y$.

Input $\underline{x}$	Output $\underline{y}$	Type	Examples
sequence $T_x > 1$	sequence $T_y > 1, T_y \neq T_x$	Many-To-Many	Automatic Speech Recognition, Machine translation

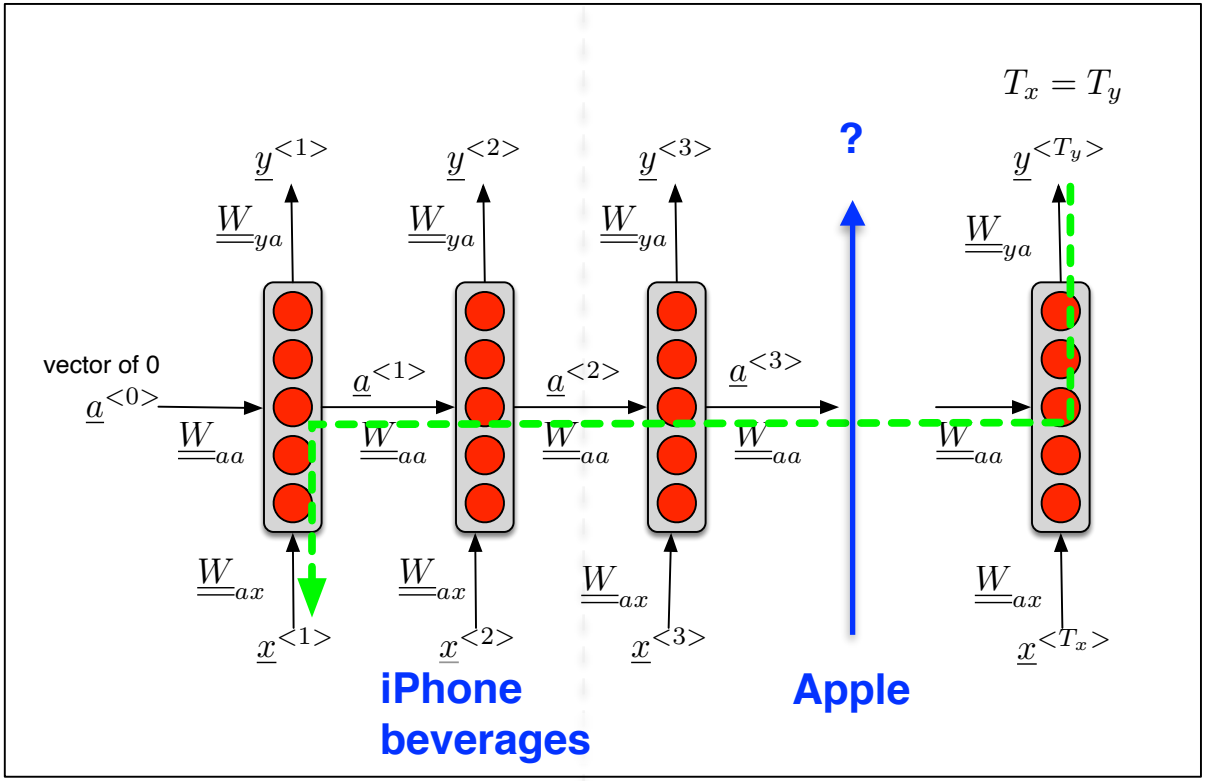




Different architectures

Different architectures

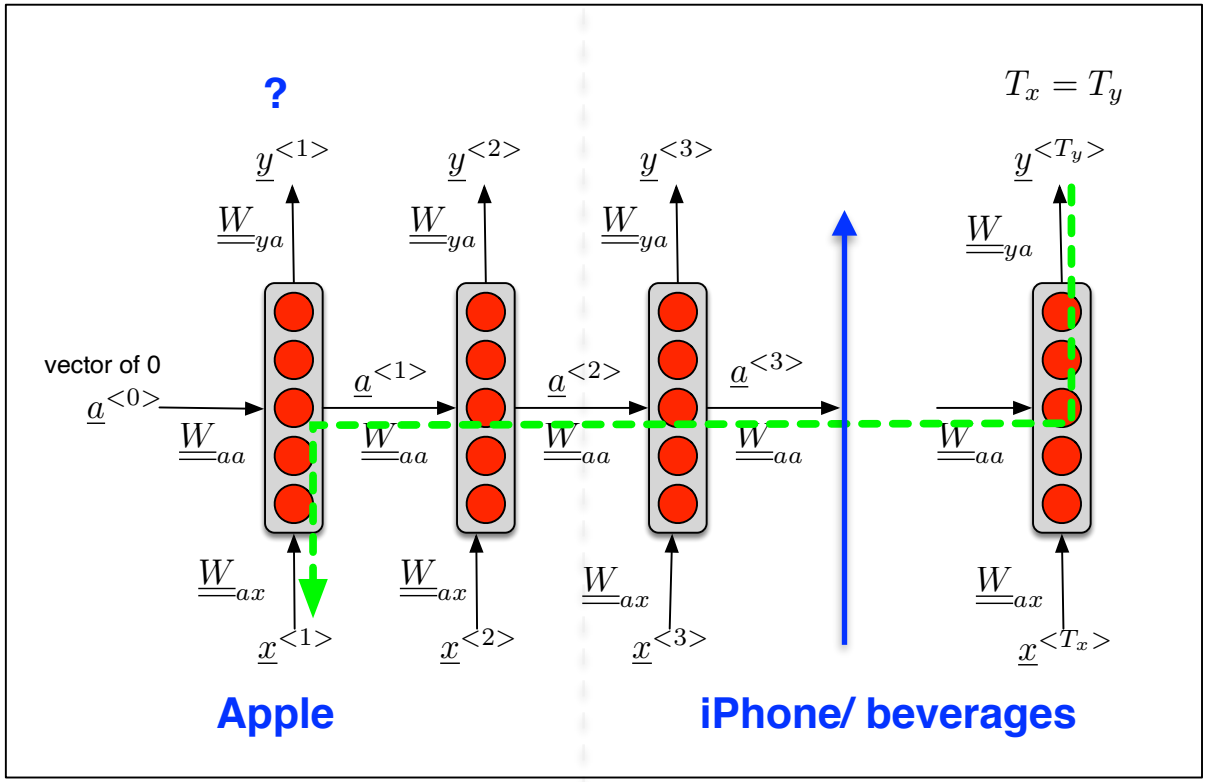
RNN



- RNN scans data from left to right ⇒ can do skip connections but only in the past
- Question: is "**Apple**" a named entity ?
  - "The new **iPhone** is sold in **Apple** stores."
  - "A refreshing and healthy **beverages** is an **Apple** juice."
- ⇒ we can predict (since RNN depends on the past)

Different architectures

RNN

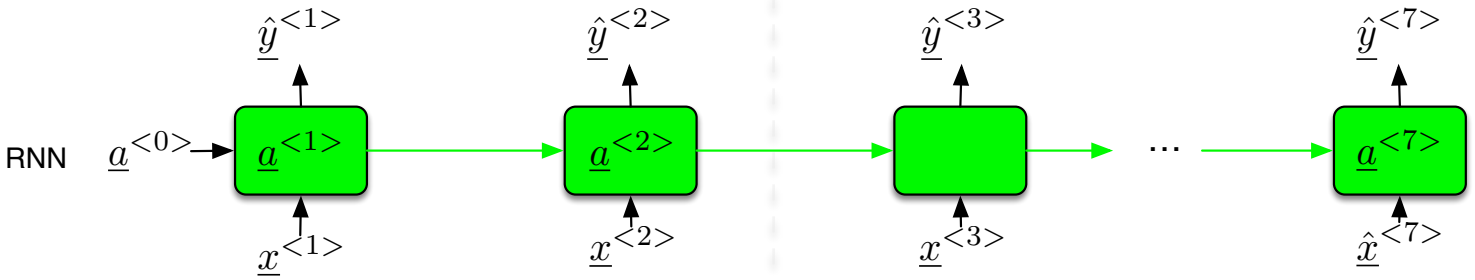


- RNN scans data from left to right ⇒ can do skip connections but only in the past
- Question: is "**Apple**" a named entity ?
  - "**Apple** stores sale the new **iPhone**."
  - "**Apple** juice is a refreshing and healthy **beverages**."
- ⇒ we cannot predict (since RNN does not depend on the future)

Bi-directional RNN

– **Standard RNN**:

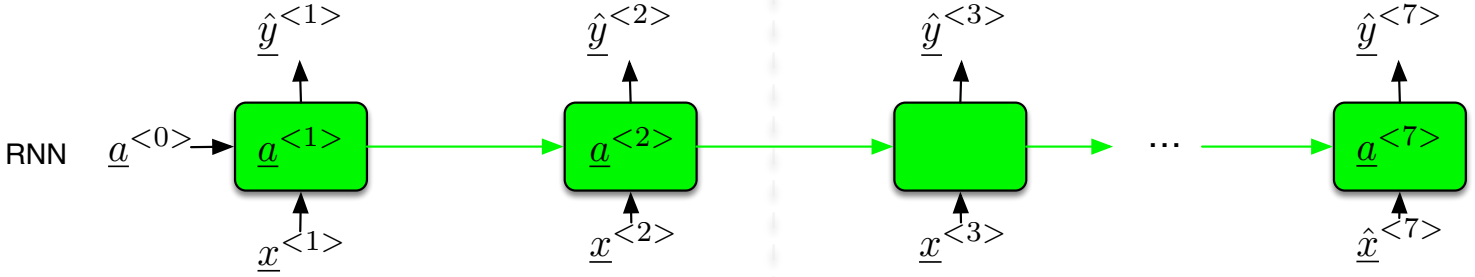
- Left-Right hidden states: $a^{\langle t \rangle}$



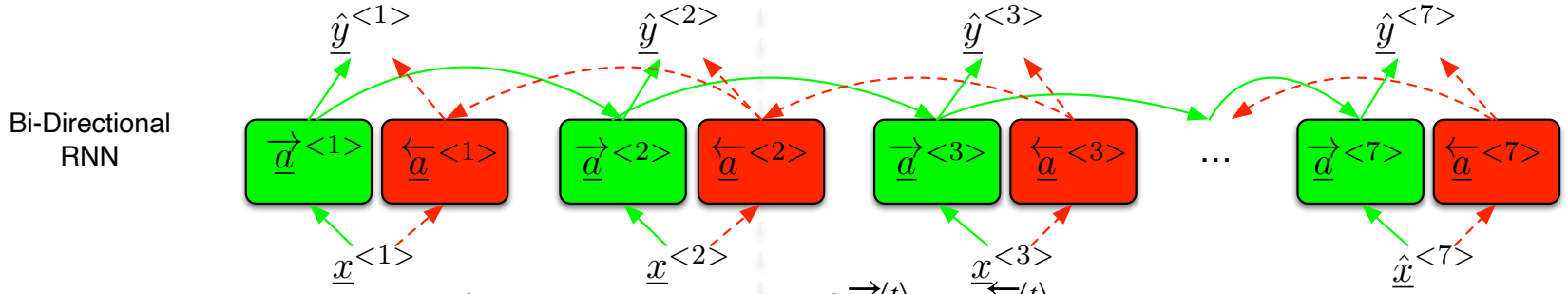
Different architectures

Bi-directional RNN

- **Standard RNN**:
  - Left-Right hidden states: $a^{\langle t \rangle}$



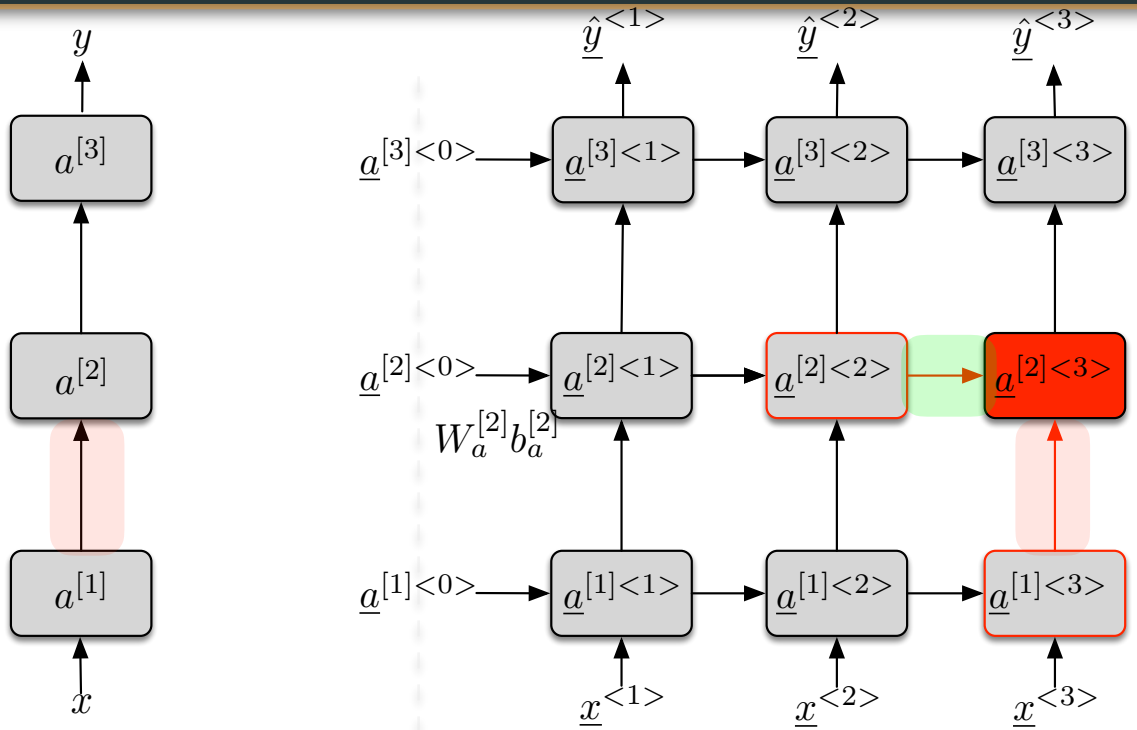
- **Bi-directional RNN:**
  - Left-Right hidden states: $\overrightarrow{a}^{\langle t \rangle}$
  - Right-Left hidden states: $\overleftarrow{a}^{\langle t \rangle}$



  - The prediction is done from the concatenation of $\overrightarrow{a}^{\langle t \rangle}$ and $\overleftarrow{a}^{\langle t \rangle}$
  - $\hat{y}^{\langle t \rangle} = g(W_y[\overrightarrow{a}^{\langle t \rangle}, \overleftarrow{a}^{\langle t \rangle}] + b_y)$

Different architectures

Deep-RNN

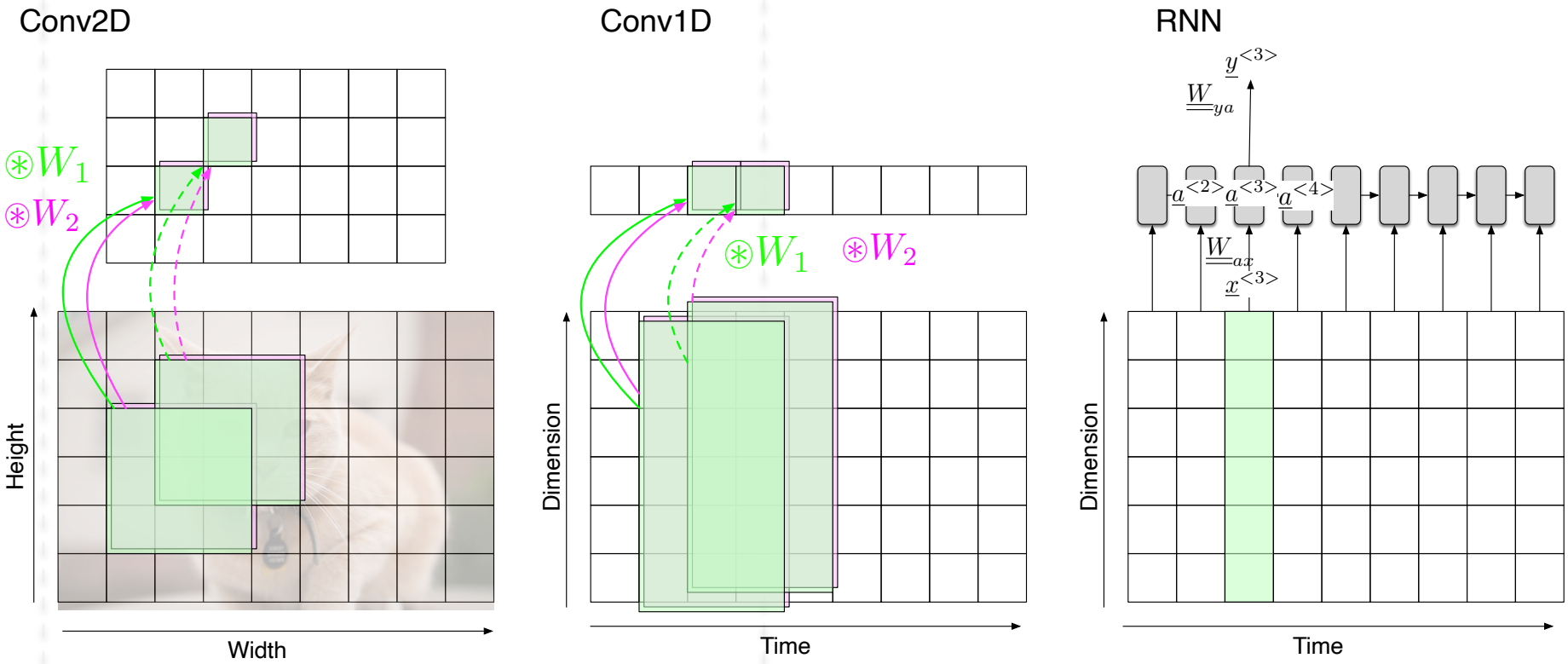


- For the layer $[1]$, the inputs of the cell at time $\langle t \rangle$ are
  - the input at the current time $\langle t \rangle$: $x^{\langle t \rangle} = \mathbf{a}^{[0]\langle t \rangle}$
  - the value of the cell at the previous time $\langle t-1 \rangle$: $\mathbf{a}^{[1]\langle t-1 \rangle}$
- For the layer $[l]$, the inputs of the cell at time $\langle t \rangle$ are
  - the value of the cell of the previous layer $[l-1]$ at the current time $\langle t \rangle$: $\mathbf{a}^{[l-1]\langle t \rangle}$
  - the value of the cell of the current layer $[l]$ at the previous time $\langle t-1 \rangle$: $\mathbf{a}^{[l]\langle t-1 \rangle}$

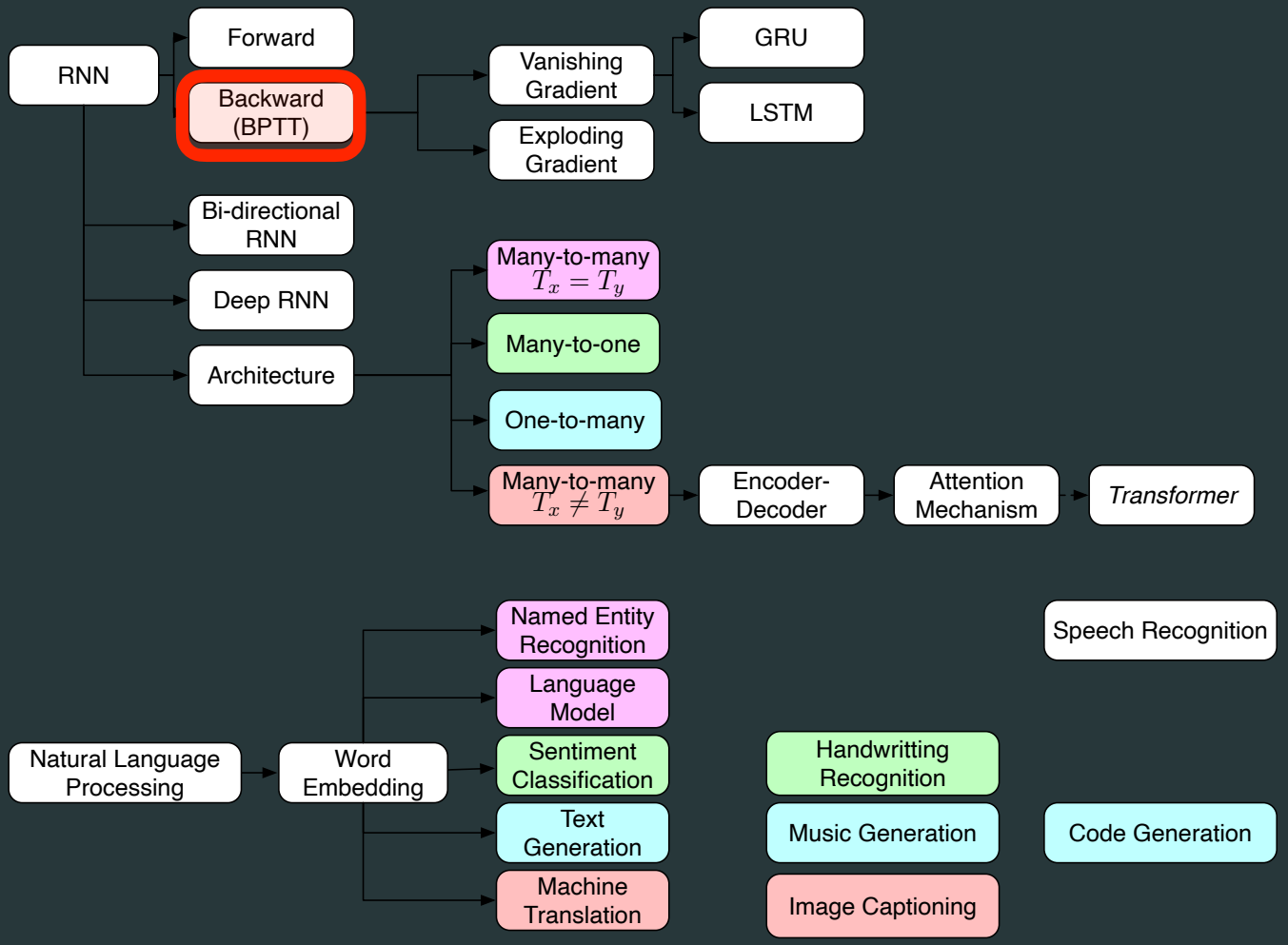
$$a^{[l]<t>} = g\left(W_a^{[l]}[a^{[l]\langle t-1 \rangle}, a^{[l-1]\langle t \rangle}] + b_a^{[l]}\right)$$

Different architectures

Using 1D convolution instead of RNN



Back Propagation Through Time (BPTT)



Back Propagation Through Time (BPTT)

Forward pass

- Compute $\mathbf{a}^{\langle t \rangle}, \hat{y}^{\langle t \rangle}, \mathscr{L}^{\langle t \rangle}, \mathscr{L}$
- **Forward**

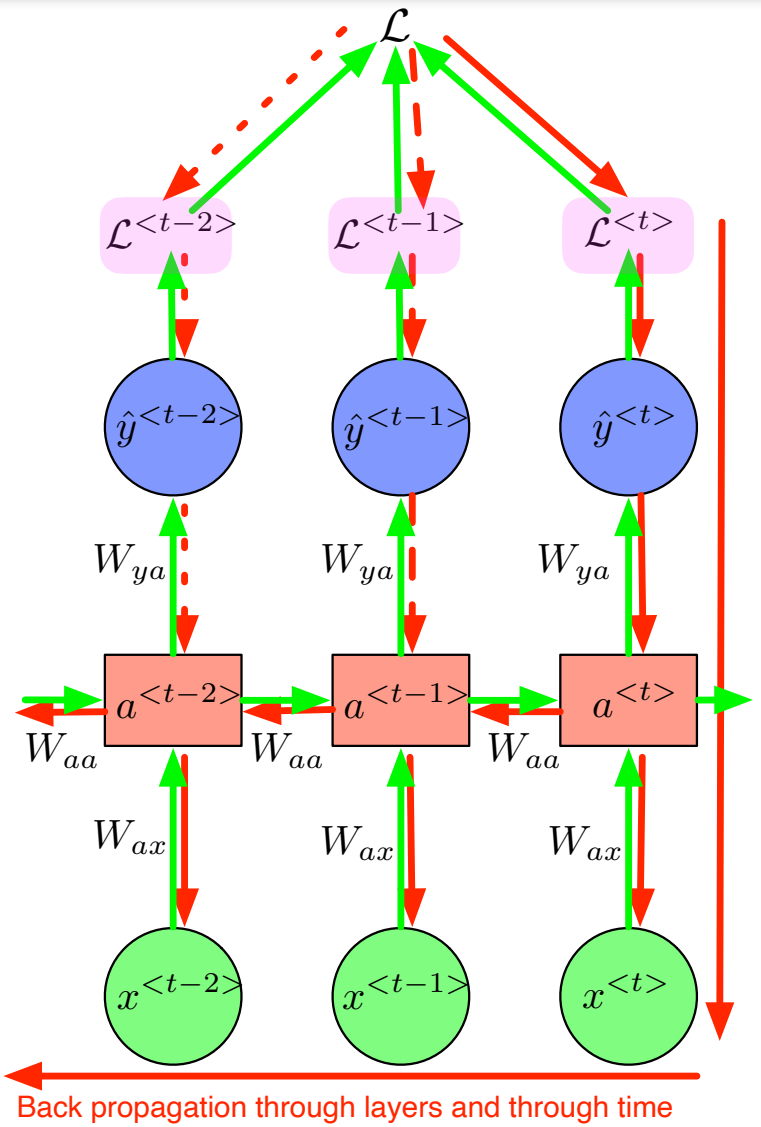
$$\mathbf{a}^{\langle t \rangle} = g_a\left(\mathbf{W}_{ax}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{aa}\mathbf{a}^{\langle t-1 \rangle} + \mathbf{b}_a\right)$$

$$\hat{y}^{\langle t \rangle} = g_y\left(\mathbf{W}_{ya}\mathbf{a}^{\langle t \rangle} + \mathbf{b}_y\right)$$

- **Loss**
  - Loss for time $\langle t \rangle$: $\mathscr{L}^{\langle t \rangle}(y^{\langle t \rangle}, \hat{y}^{\langle t \rangle})$
    - $y^{\langle t \rangle}$: ground-truth
    - $\hat{y}^{\langle t \rangle}$: prediction
  - Total loss

$$\mathscr{L} = \sum_t \mathscr{L}^{\langle t \rangle}(y^{\langle t \rangle}, \hat{y}^{\langle t \rangle})$$

- **Gradients ?**
  - compute $\frac{\partial \mathscr{L}}{\partial W_{ya}}, \frac{\partial \mathscr{L}}{\partial W_{ax}}, \frac{\partial \mathscr{L}}{\partial W_{aa}}, \frac{\partial \mathscr{L}}{\partial b_y}, \frac{\partial \mathscr{L}}{\partial b_a}$
  - All weights are shared across time steps !!!



Back Propagation Through Time (BPTT)

Backward: $\frac{\partial \mathscr{L}}{\partial W_{ya}}$.

- Linearity

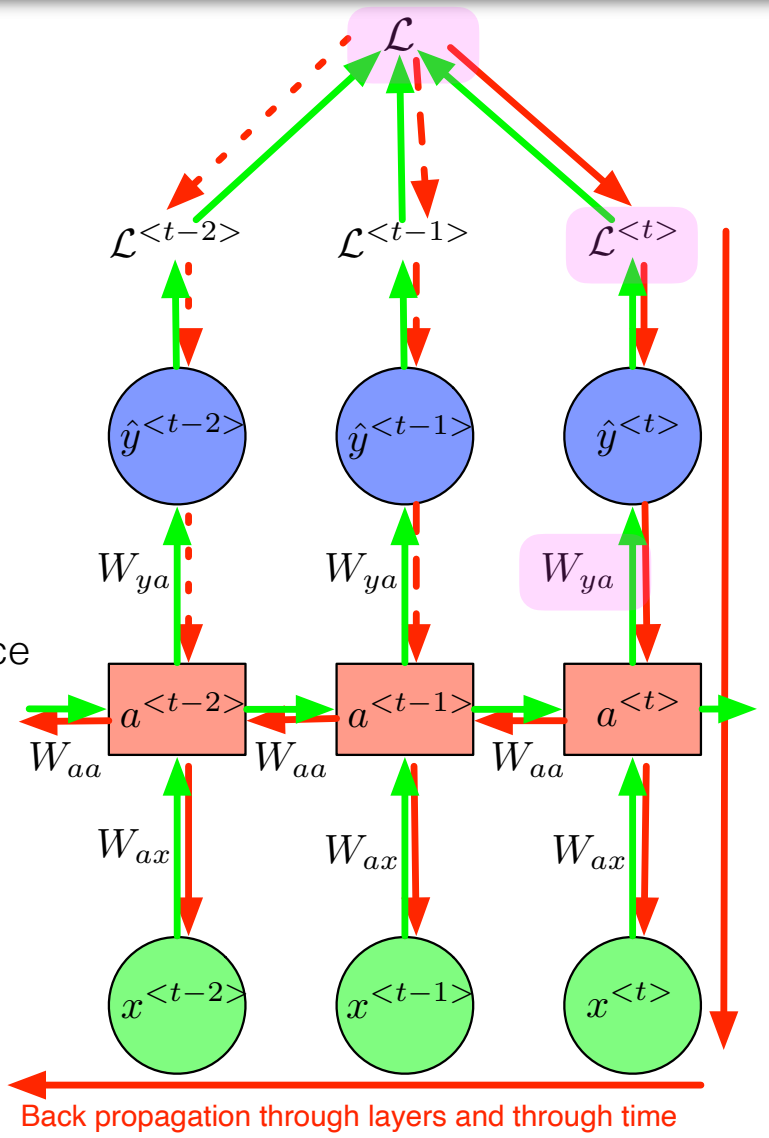
$$\frac{\partial \mathscr{L}}{\partial W_{ya}} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{ya}}$$

- How much varying W_{ya} affect $\mathscr{L}^{\langle t \rangle}$? : chain rule

$$\frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{ya}} = \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\partial \hat{y}^{\langle t \rangle}}{\partial W_{ya}}$$

- Very simple since $\hat{y}^{\langle t \rangle}$ depends on W_{ya} on only place (indeed $a^{\langle t \rangle}$ does not depend on W_{ya})

$$\hat{y}^{\langle t \rangle} = g_y \left(\mathbf{W}_{ya} \mathbf{a}^{\langle t \rangle} + \mathbf{b}_y \right)$$



Back Propagation Through Time (BPTT)

Backward: $\frac{\partial \mathscr{L}}{\partial W_{aa}}$.

- Linearity

$$\frac{\partial \mathscr{L}}{\partial W_{aa}} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{aa}}$$

- How much varying W_{aa} affect $\mathscr{L}^{\langle t \rangle}$? no chain rule

$$\frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{aa}} \neq \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\partial \hat{y}^{\langle t \rangle}}{\partial a^{\langle t \rangle}} \frac{\partial a^{\langle t \rangle}}{\partial W_{aa}}$$

  - because $a^{\langle t \rangle}$ also depends on W_{aa} through $a^{\langle t-1 \rangle}$ which itself depends on ...

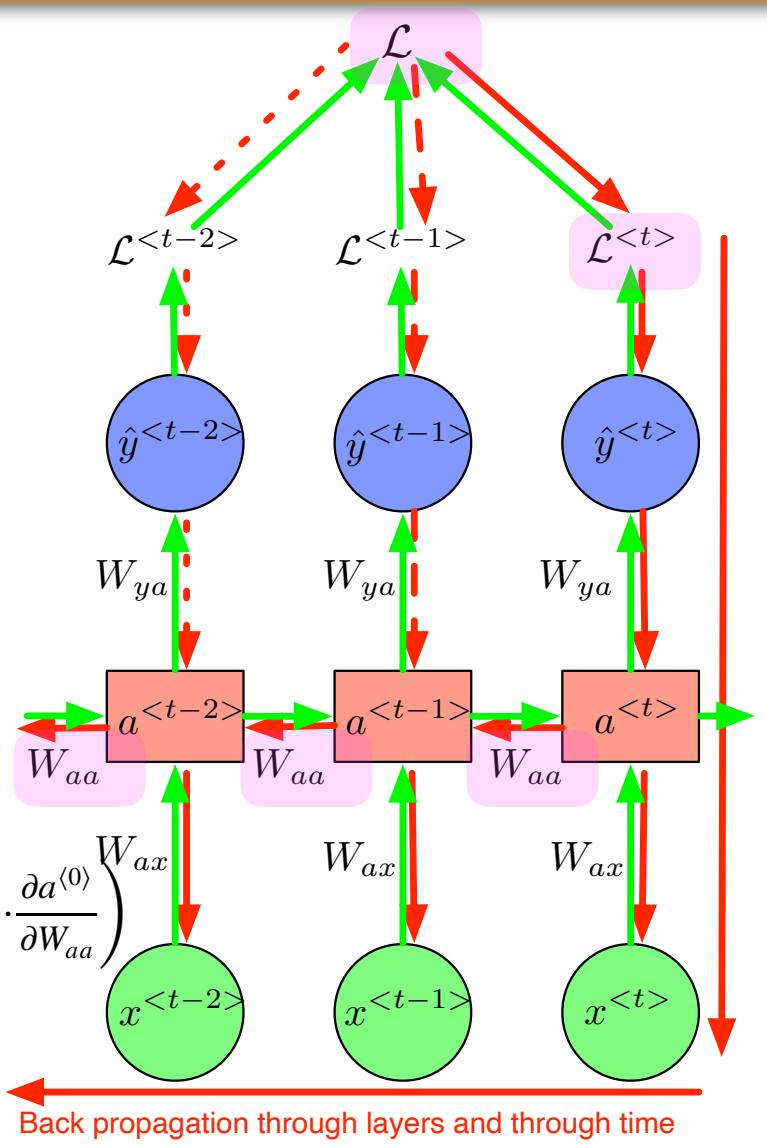
$$\mathbf{a}^{\langle t \rangle} = g_a \left(\mathbf{W}_{ax} \mathbf{x}^{\langle t \rangle} + \mathbf{W}_{aa} \mathbf{a}^{\langle t-1 \rangle} + \mathbf{b}_a \right)$$

- We use a formula of total derivative

$$\frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{aa}} = \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\partial \hat{y}^{\langle t \rangle}}{\partial a^{\langle t \rangle}} \left(\frac{\partial a^{\langle t \rangle}}{\partial W_{aa}} + \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \frac{\partial a^{\langle t-1 \rangle}}{\partial W_{aa}} + \cdots + \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \cdots \frac{\partial a^{\langle 0 \rangle}}{\partial W_{aa}} \right)$$

$$\sum_{k=0}^{t} \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \cdots \frac{\partial a^{\langle k+1 \rangle}}{\partial a^{\langle k \rangle}} \frac{\partial a^{\langle k \rangle}}{\partial W_{aa}}$$

$$\sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-1 \rangle}} \right) \frac{\partial a^{\langle k \rangle}}{\partial W_{aa}}$$



Back Propagation Through Time (BPTT)

Backward: $\frac{\partial \mathscr{L}}{\partial W_{ax}}$.

- Linearity

$$\frac{\partial \mathscr{L}}{\partial W_{ax}} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{ax}}$$

- How much varying W_{ax} affect $\mathscr{L}^{\langle t \rangle}$? no chain rule

$$\frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{ax}} \neq \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\partial \hat{y}^{\langle t \rangle}}{\partial a^{\langle t \rangle}} \frac{\partial a^{\langle t \rangle}}{\partial W_{ax}}$$

  - because $a^{\langle t \rangle}$ also depends on W_{ax} through $a^{\langle t-1 \rangle}$ which itself depends on ...

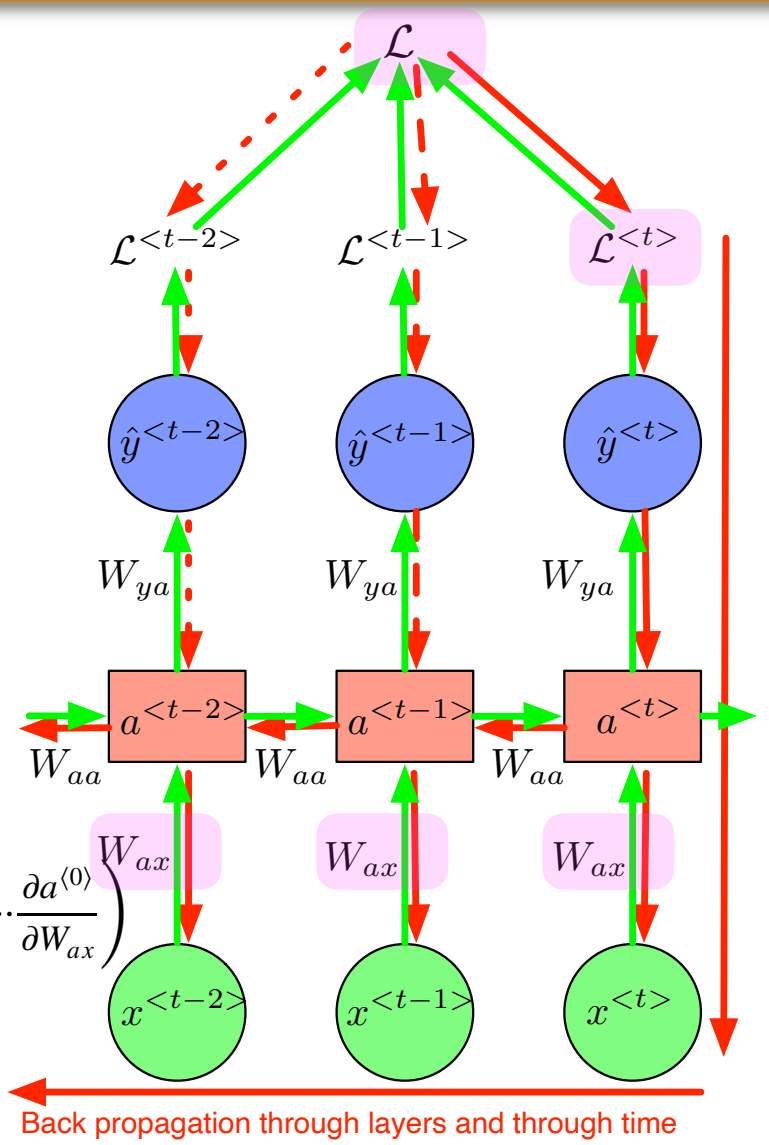
$$\mathbf{a}^{\langle t \rangle} = g_a\left(\mathbf{W}_{ax}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{aa}\mathbf{a}^{\langle t-1 \rangle} + \mathbf{b}_a\right)$$

- We use a formula of total derivative

$$\frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{ax}} = \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\partial \hat{y}^{\langle t \rangle}}{\partial a^{\langle t \rangle}} \left(\frac{\partial a^{\langle t \rangle}}{\partial W_{ax}} + \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \frac{\partial a^{\langle t-1 \rangle}}{\partial W_{ax}} + \cdots + \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \cdots \frac{\partial a^{\langle 0 \rangle}}{\partial W_{ax}} \right)$$

$$\sum_{k=0}^{t} \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \cdots \frac{\partial a^{\langle k+1 \rangle}}{\partial a^{\langle k \rangle}} \frac{\partial a^{\langle k \rangle}}{\partial W_{ax}}$$

$$\sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-1 \rangle}} \right) \frac{\partial a^{\langle k \rangle}}{\partial W_{ax}}$$



Back Propagation Through Time (BPTT)

Vanishing and exploding gradients

- Vanishing gradient
  - **In MLP**: very deep NN
    - very difficult to propagate back the gradient to affect the weights of the early Layers
  - **In RNN: very difficult to learn long-term dependency**
    - The student, which already followed 6 hours of lessons, was tired.
    - The students, which already followed 6 hours of lessons, were tired.

Back Propagation Through Time (BPTT)

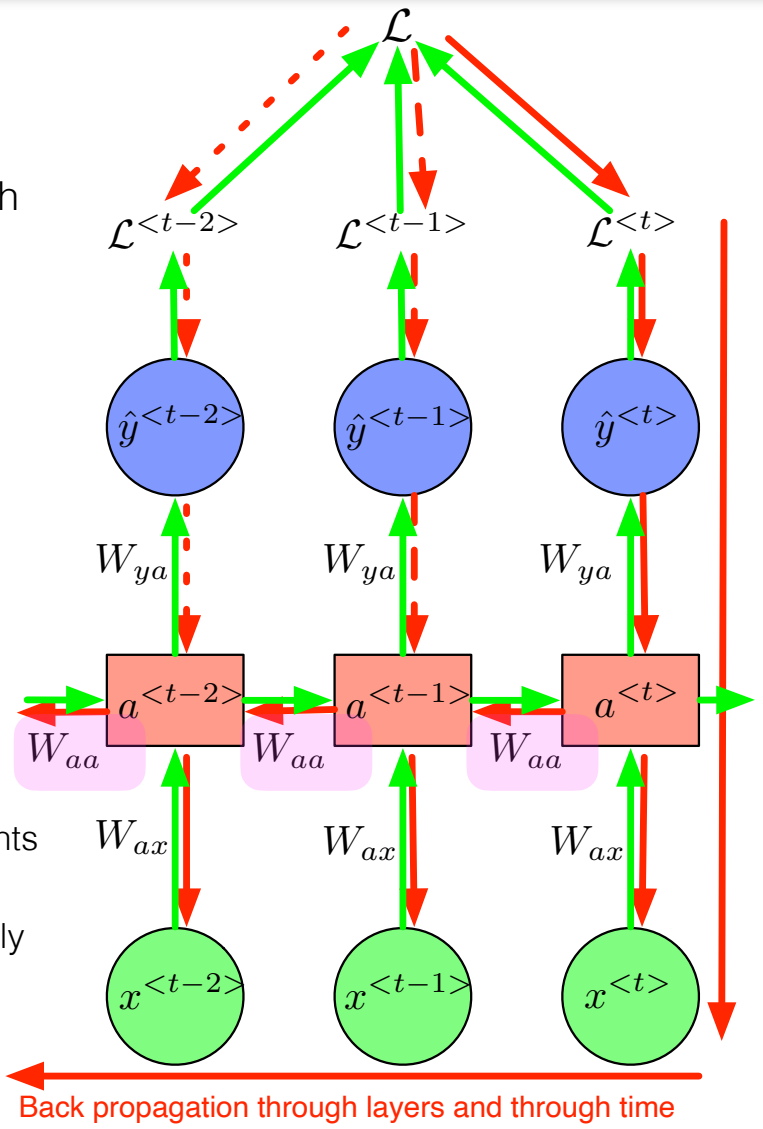
Vanishing and exploding gradients

– To train an RNN we need to back-propagate through layers and through time

$$\frac{\partial \mathscr{L}}{\partial W_{aa}} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{aa}}$$

$$\frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{aa}} = \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-1 \rangle}} \right) \frac{\partial a^{\langle k \rangle}}{\partial W_{aa}}$$

- each term in the sum is the contribution of
  - a state at time $\langle k \rangle$
  - to the gradient of the loss at time step $\langle t \rangle$
- the more steps between $\langle k \rangle$ and $\langle t \rangle$, the more elements in this product
- the values of these Jacobian matrices have particularly severe impact on the contributions of faraway steps



Back Propagation Through Time (BPTT)

Vanishing and exploding gradients

$$\frac{\partial \mathcal{L}}{\partial W_{aa}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial W_{aa}}$$

$$\frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial W_{aa}} = \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-1 \rangle}} \right) \frac{\partial a^{\langle k \rangle}}{\partial W_{aa}}$$

_ Suppose only one hidden units, then $a^{\langle t \rangle}$ is a scalar and consequently $\frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}}$ is also a scalar

- if $\left| \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \right| < 1$, then the product goes to **0** exponentially fast
- if $\left| \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \right| > 1$, then the product goes to ∞ exponentially fast

_ Vanishing gradients: $\left| \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \right| < 1$

  – contributions from faraway steps vanish and don't affect the training
  – difficult to learn long-range dependencies

_ Exploding gradients: $\left| \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \right| > 1$

  – make the learning process unstable
  – gradient could even become **NaN**

Dealing with the exploding gradient

– **Solution 1: gradient clipping**

_ gradient $d\theta = \frac{\partial \mathcal{L}}{\partial \theta}$ where θ are all the parameters

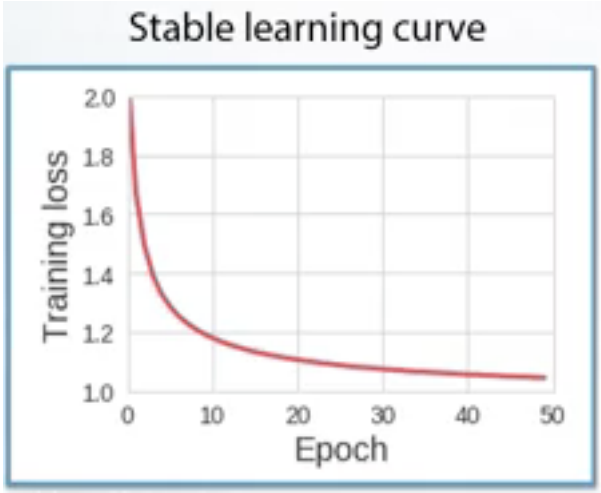
- if $\|\theta\|$ > threshold

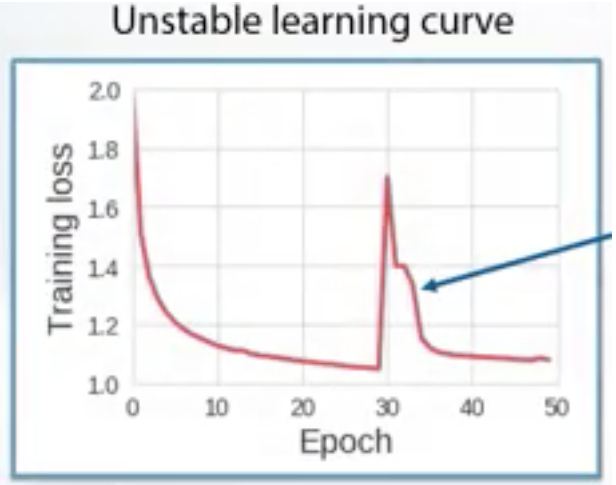
$$d\theta \leftarrow \frac{\text{threshold}}{\|\theta\|} d\theta$$

– clipping doesn't change the direction of the gradient but change its length

– we can clip only the norm of the part which causes the problem

– we choose the threshold manually: start with a large threshold and then reduce it



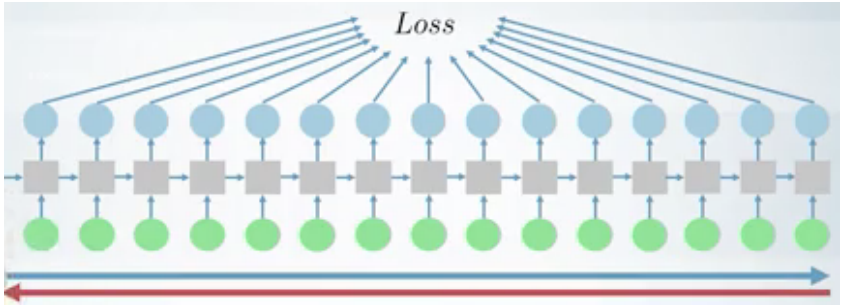


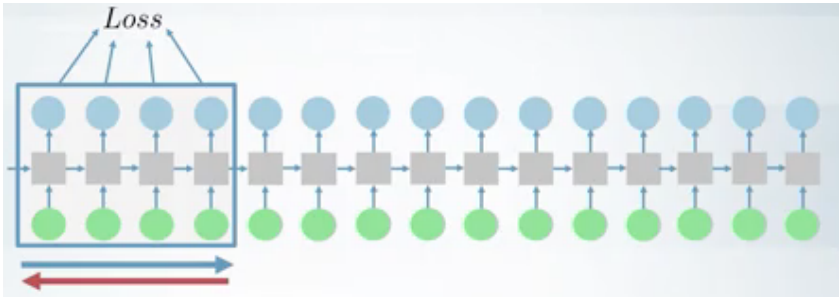
Back Propagation Through Time (BPTT)

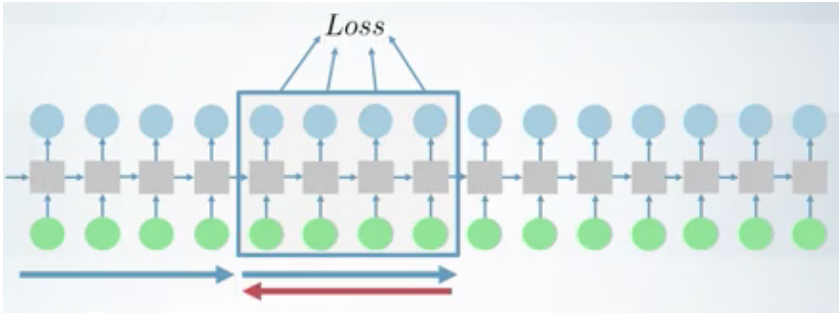
Dealing with the exploding gradient

– **Solution 1: truncated BPTT**

– Training very long sequences
  - time consuming !
  - exploding gradients !
– Truncated BPTT:
  - run forward and backward passes through the chunks of the sequence (instead of the whole sequence)
– Forward
  - We carry hidden states forward in time forever
– Backward
  - only back-propagate in the chunks (small number of steps)
– Much faster but dependencies longer than the chunk size don't affect the training (at least they still work at Forward pass)







Dealing with the vanishing gradient

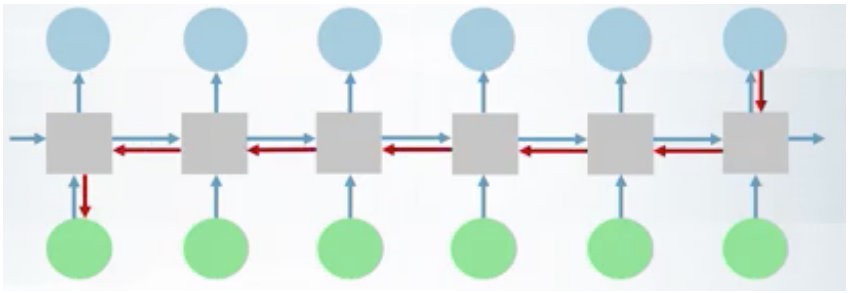
– **Solution 2: use skip-connections**

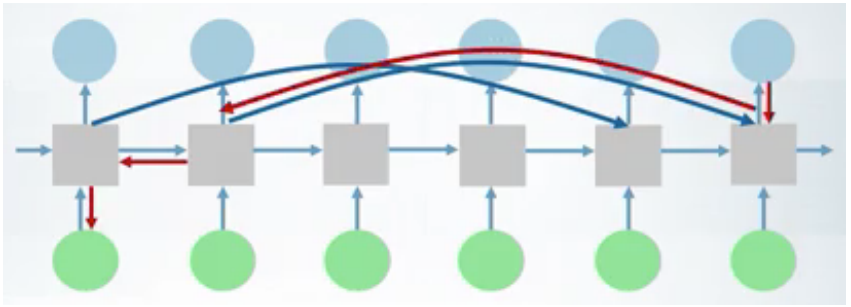
– in RNN: because at each step we multiply the Jacobian matrices → vanishing gradient
  - we cannot learn long-term dependencies

– Solution: add **short-cuts** between hidden states that are separated by more than one time step
  - are usual connections (with their own parameter matrices)
  - create much shorter paths between far away time-steps

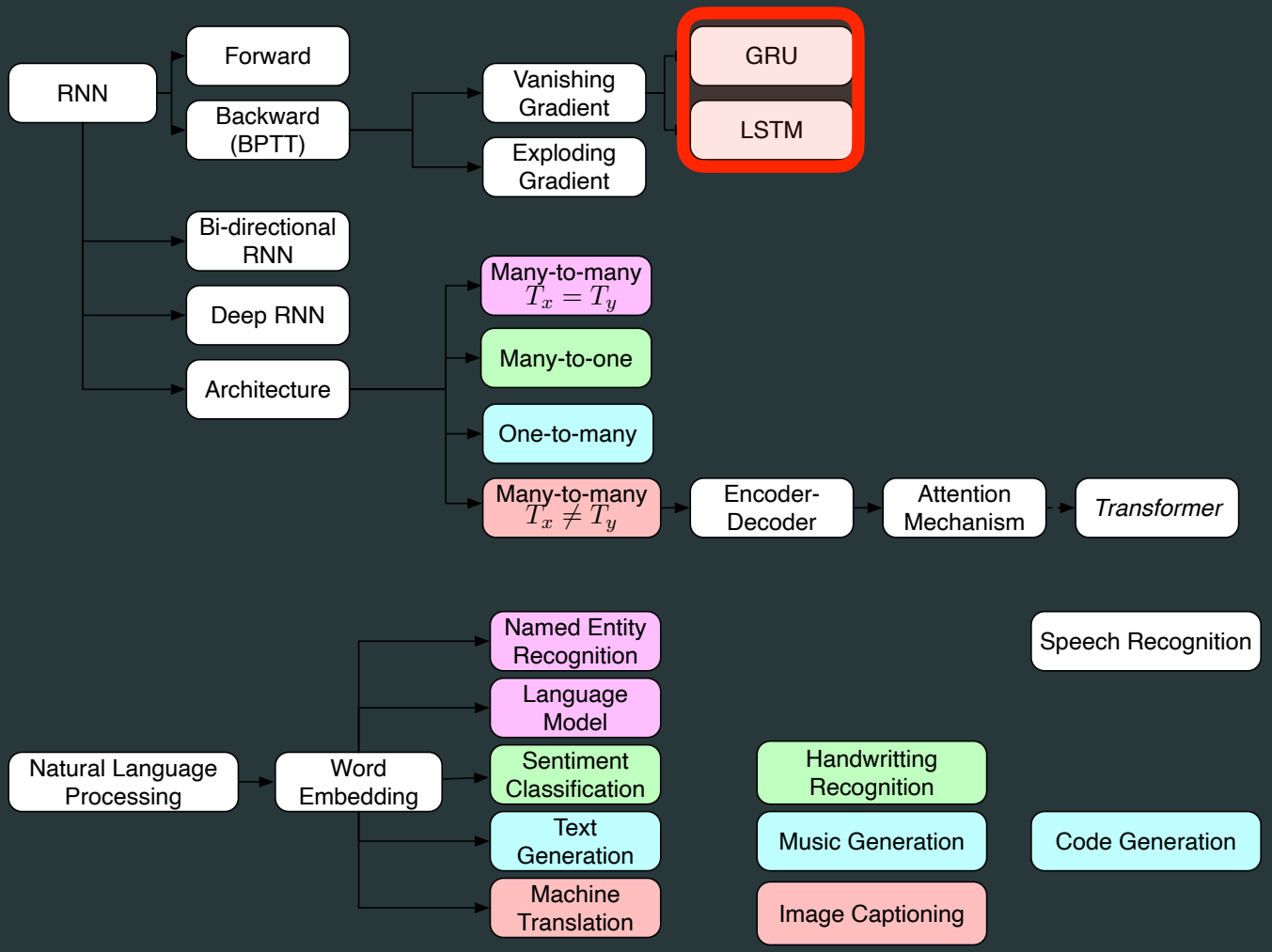
– Back-propagation through the shortcuts: the gradients vanish slower
  - we can learn long-term dependencies

– not exclusive to RNN (see ResNet)





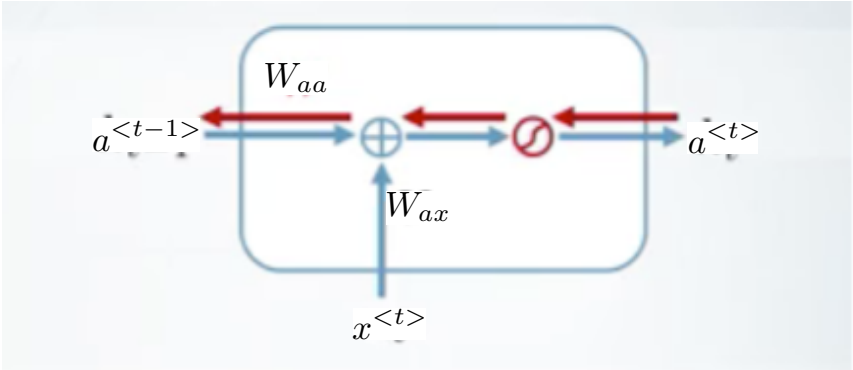
Gated units (LSTM and GRU)



Main ideas : Recurrent Neural Network (RNN)

$$\mathbf{a}^{\langle t \rangle} = g_a\left(\mathbf{W}_{ax}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{aa}\mathbf{a}^{\langle t-1 \rangle} + \mathbf{b}_a\right)$$

- non-linearities and/or multiplication creates the vanishing gradient problem
- Solution ?
  - create a short-way for back-propagation without any non-linearities or multiplication



Main ideas : Long Short Term Memory Units (LSTM)

- Add a new path: the **memory cell** $c^{\langle t \rangle}$
  - same dimension as the hidden layer $a^{\langle t \rangle}$
- **Simplified LSTM** (no possibility to erase)
  - Candidate cell value

$$\tilde{c}^{\langle t \rangle} = g\left(W_{ax}^{c} x^{\langle t \rangle} + W_{aa}^{c} a^{\langle t-1 \rangle} + b^{c}\right)$$

  - Gates (update, output)

$$\Gamma_u = \sigma\left(W_{ax}^{u} x^{\langle t \rangle} + W_{aa}^{u} a^{\langle t-1 \rangle} + b^{u}\right)$$

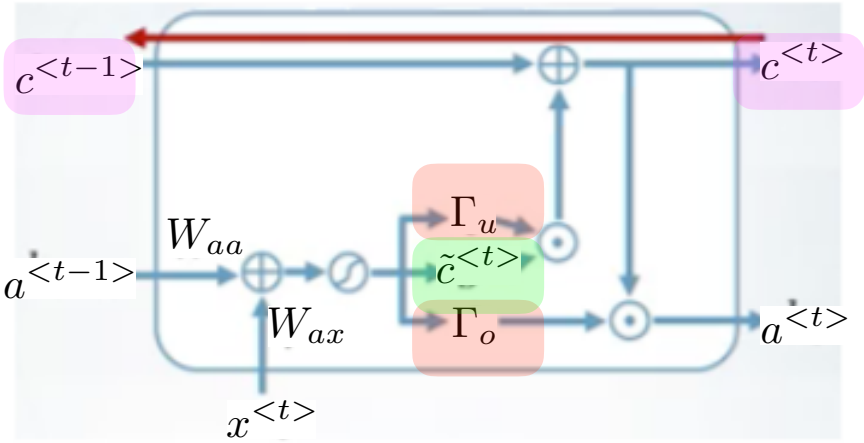
$$\Gamma_o = \sigma\left(W_{ax}^{o} x^{\langle t \rangle} + W_{aa}^{o} a^{\langle t-1 \rangle} + b^{o}\right)$$

  - Output memory cell

$$c^{\langle t \rangle} = c^{\langle t-1 \rangle} + \Gamma_u \odot \tilde{c}^{\langle t \rangle}$$

  - Output hidden state

$$a^{\langle t \rangle} = \Gamma_o \odot f\left(c^{\langle t \rangle}\right)$$



- no non-linearity or multiplication in the memory cell-path
  - $\Rightarrow \dfrac{\partial c^{\langle t \rangle}}{\partial c^{\langle t-1 \rangle}} = \text{diag}(1) \Rightarrow$ no vanishing gradients

[Hochreiter and Schmidhuber 1997. "Long short-term memory", Neural Computation Volume 9 | Issue 8 | November 15, 1997 p.1735-1780]

Main ideas : Long Short Term Memory Units (LSTM)

- **Forget Gate LSTM** (no possibility to erase)
  - Gates (forget)

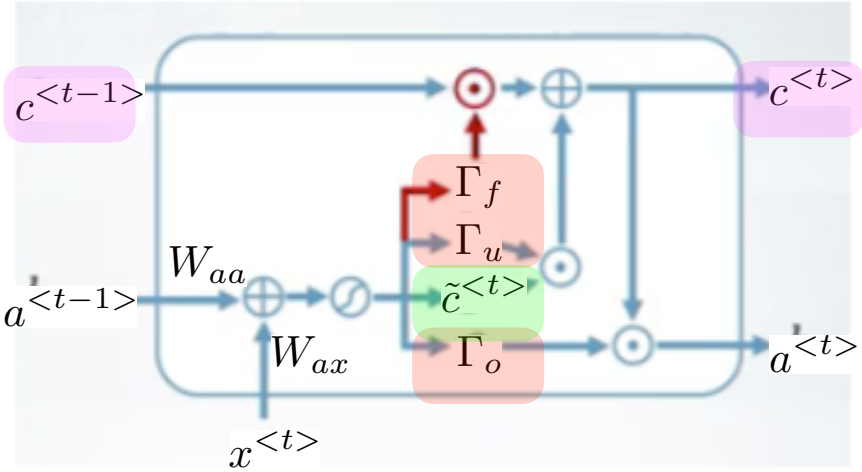
$$\Gamma_f = \sigma\left(W_{ax}^f x^{\langle t\rangle} + W_{aa}^f a^{\langle t-1\rangle} + b^f\right)$$

  - Output memory cell

$$c^{\langle t\rangle} = \Gamma_f \odot c^{\langle t-1\rangle} + \Gamma_u \odot \tilde{c}^{\langle t\rangle}$$

  - Output hidden state

$$a^{\langle t\rangle} = \Gamma_o \odot f\left(c^{\langle t\rangle}\right)$$

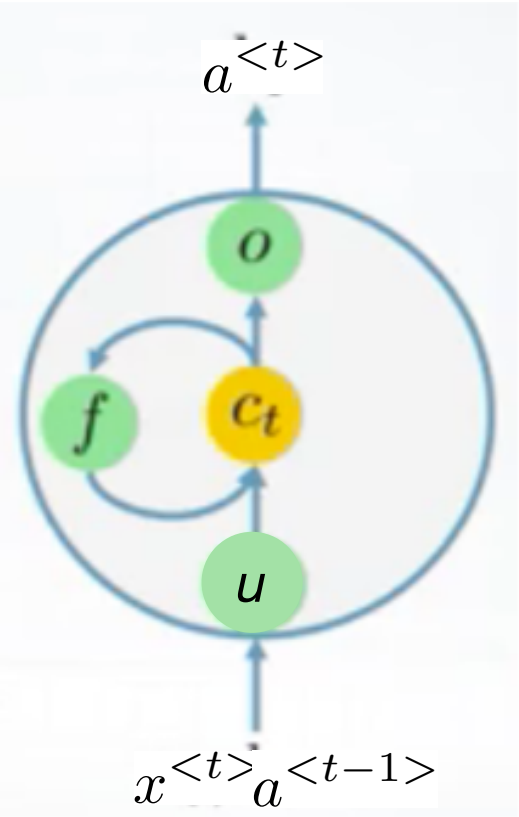


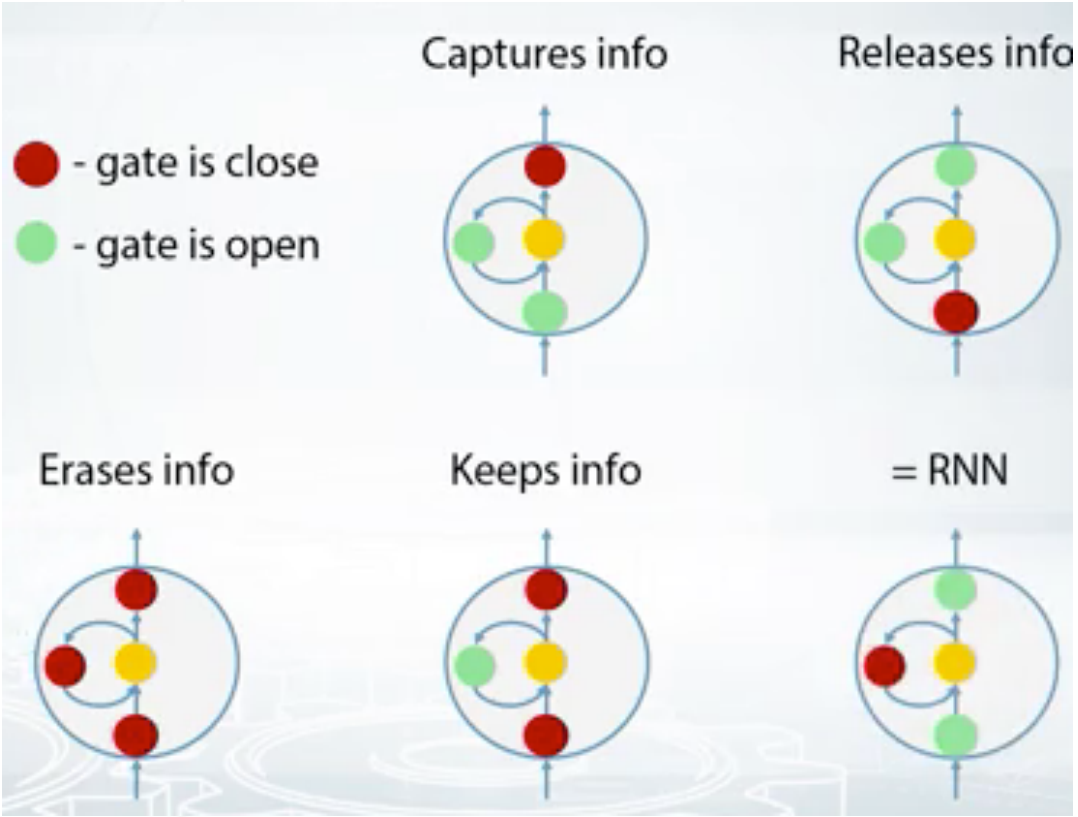
- We now have a multiplication on the short-way
  - $\dfrac{\partial c^{\langle t\rangle}}{\partial c^{\langle t-1\rangle}} = \text{diag}(\Gamma_f)$
  - with $\Gamma_f \in [0,1]$ the forget gate: $\Gamma_f = \sigma\left(W_{ax}^f x^{\langle t\rangle} + W_{aa}^f a^{\langle t-1\rangle} + b^f\right)$
    - set a high initial value for $b^f \Rightarrow$ at the beginning of training the gate is open, the LSTM doesn't forget and can learn long-term dependencies in the data

- **Note**: LSTM has four times more parameters to be trained than RNN
  - $W_{ax}^f, W_{aa}^f, W_{ax}^u, W_{aa}^u, W_{ax}^o, W_{aa}^o, W_{ax}^c, W_{aa} + b^f, b^u, b^o, b^c$

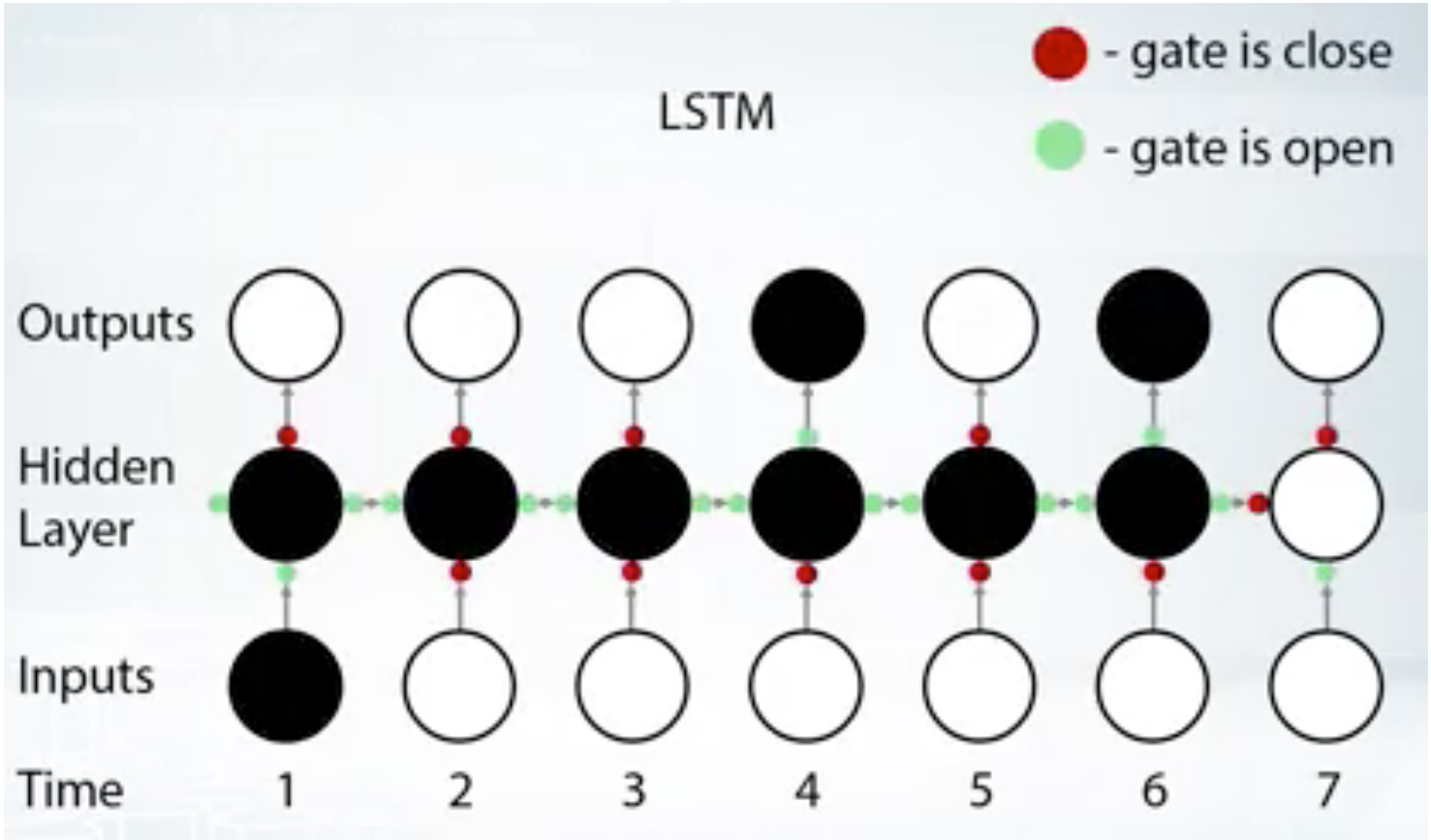
[Hochreiter and Schmidhuber 1997. "Long short-term memory", Neural Computation Volume 9 | Issue 8 | November 15, 1997 p.1735-1780]

LTSM regimes





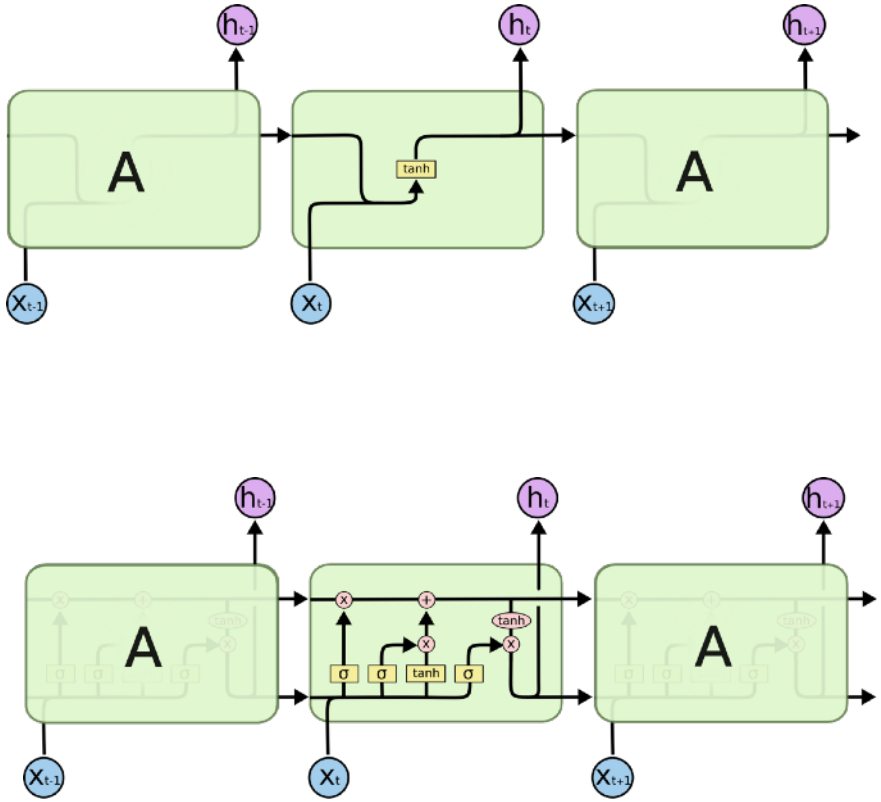
LTSM regimes



Gated units (LSTM and GRU)

LTSM example usage

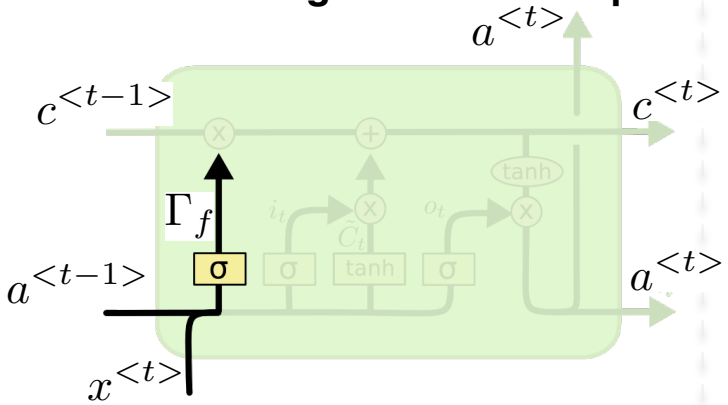
– Instead of having a single neural network layer, there are four, interacting in a very special way.
  - Lines carry a vector
  - A recurrent net transmits its hidden states
  - LSTM introduces a second channel : the cell state
  - It acts as a memory
  - Gates control the memory



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

LTSM example usage

- **What should be forgotten from the previous cell state ?**



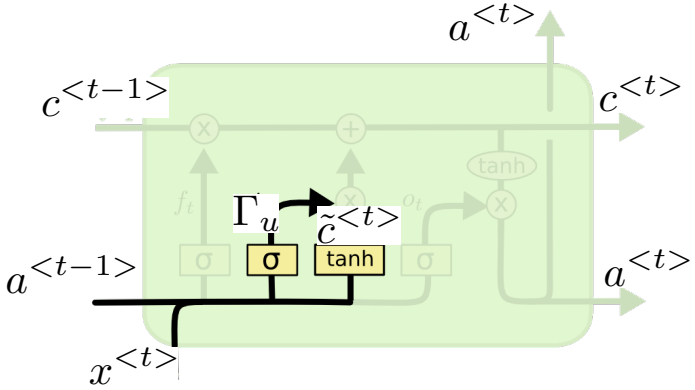
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

- **Action**
  - The sigmoid (forget gate) answers for each component :
    - 1 : to keep it,
    - 0 to forget it, or
    - a value in-between to mitigate its influence
- **Intuition for language modelling**
  - The cell state might embed the gender of the present subject :
    - keep it to predict the correct pronouns,
    - or forget it, when a new subject appears

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

LTSM example usage

- **What should be taken into account ?**



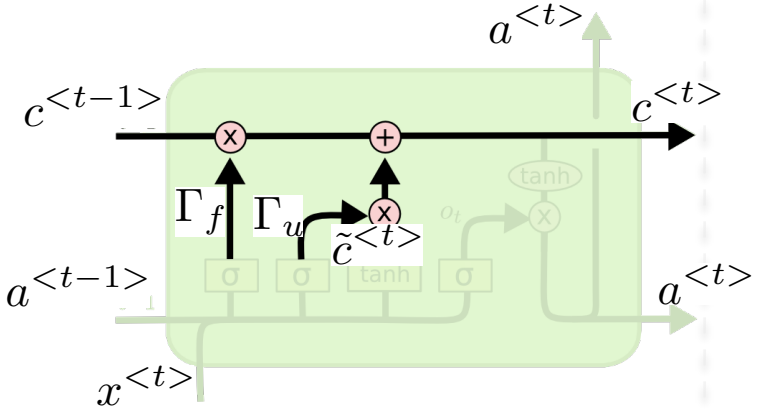
$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

- **Action**
  - Create the update $\tilde{c}^{\langle t \rangle}$ of the cell state
  - and its contribution Γ_u (the update gate with a sigmoid activation)
- **Intuition for language modelling**
  - Add the gender of the new subject to the cell state,
  - to replace the old one we're forgetting.

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

LTSM example usage

– **Write the new state**



$$c^{<t>} = \Gamma_u \odot \tilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>}$$

– **Action**

- Merge the old cell state modified by the forget gate with the new input

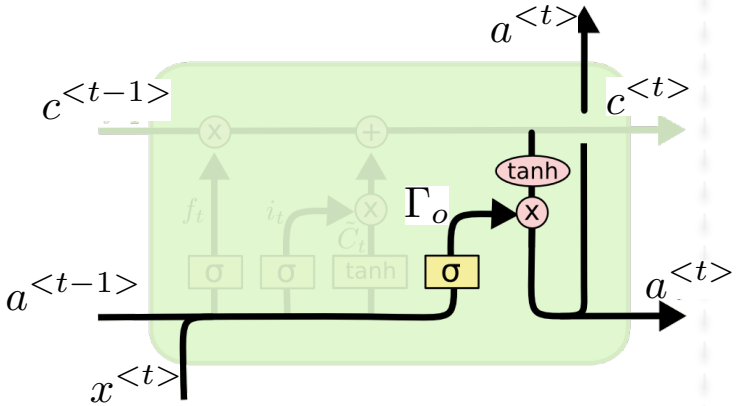
– **Intuition for language modelling**

- Decide to drop the information about the old subject
- Refresh the memory

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

LTSM example usage

- **Write the new hidden state**



$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$
$$a^{<t>} = \Gamma_o \odot \tanh(c^{<t>})$$

- **Action**
  - Decide what parts of the (filtered) cell state to output $a^{\langle t \rangle}$
  - Compute the hidden state
- **Intuition for language modelling**
  - Since we just saw a subject,
  - output the relevant information for the future (gender, number)

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

Gated Recurrent Unit (GRU)

- **Simple GRU**
  - Candidate cell value

$$\tilde{c}^{\langle t \rangle} = \tanh\left(W^c \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b^c\right)$$

  - Gates (update)

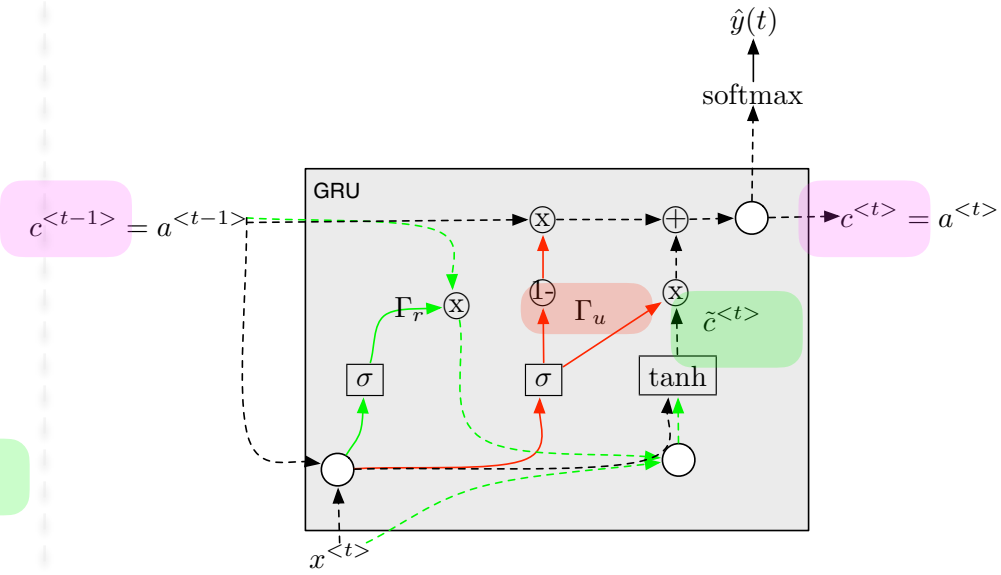
$$\Gamma_u = \sigma\left(W^u \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b^u\right)$$

  - Output memory cell

$$c^{\langle t \rangle} = (1 - \Gamma_u) \odot c^{\langle t-1 \rangle} + \Gamma_u \odot \tilde{c}^{\langle t \rangle}$$

  - Output hidden state

$$a^{\langle t \rangle} = c^{\langle t \rangle}$$



[Cho et al. 2014. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches"]
[Chung et al. 2014 "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling"]

Gated Recurrent Unit (GRU)

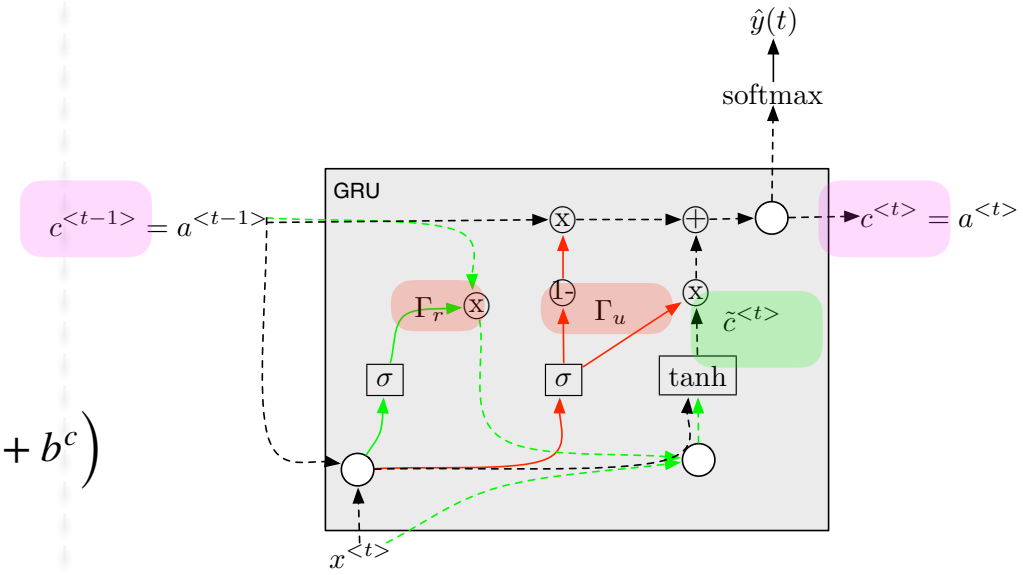
– **Full GRU**

- add a relevance gate Γ_r
- instead of using $c^{\langle t-1 \rangle}$ to compute $\tilde{c}^{\langle t \rangle}$ we first filter it using a relevance gate Γ_r

$$\Gamma_r = \sigma\left(W^r\left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b^r\right)$$

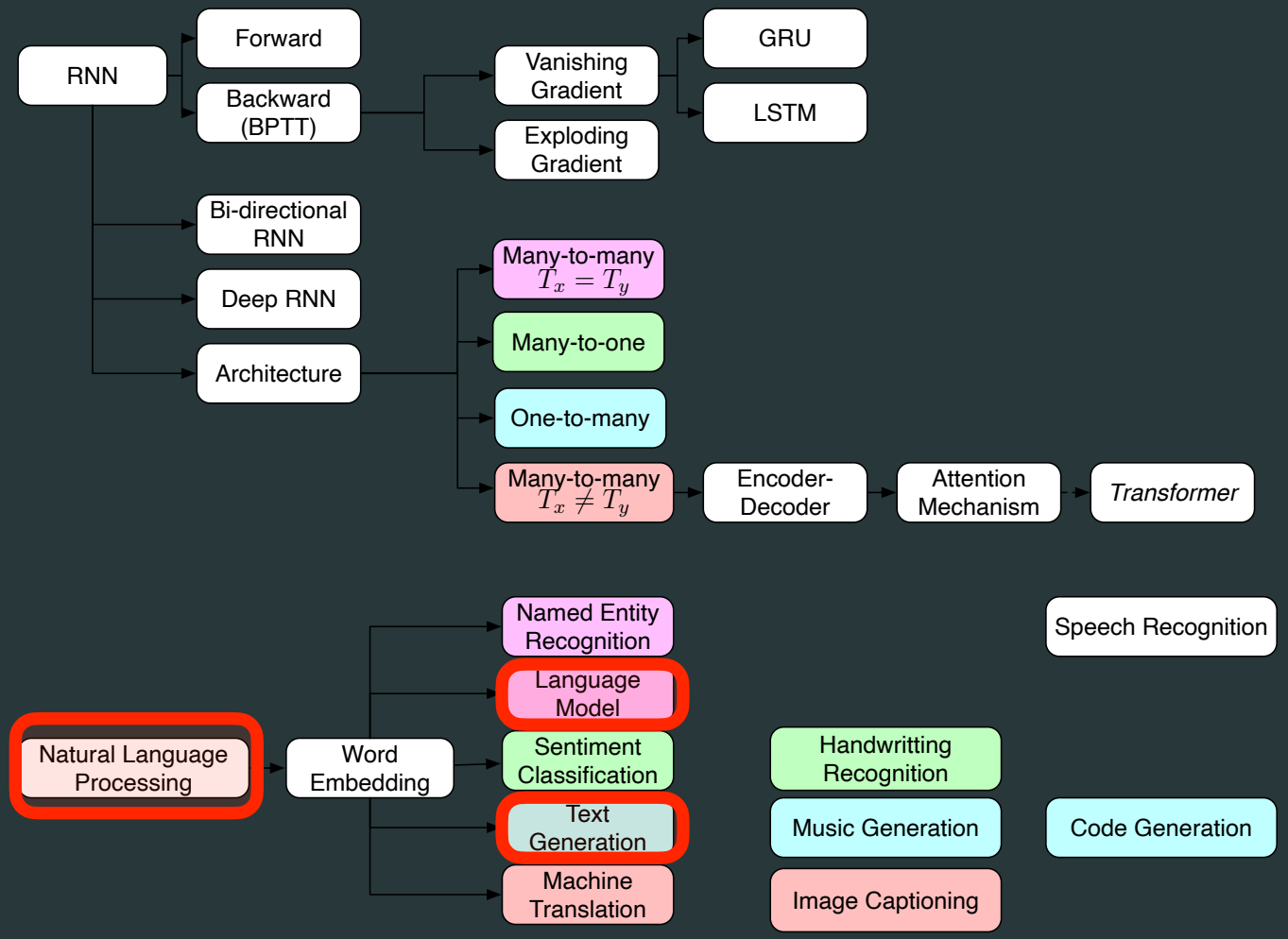
- Candidate cell value

$$\tilde{c}^{\langle t \rangle} = \tanh\left(W^c\left[\Gamma_r \odot c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b^c\right)$$



[Cho et al. 2014. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches"]
[Chung et al. 2014 "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling"]

Natural Language Processing



Language model

– **Applications**
  - Automatic Speech Recognition, Machine Translation, Optical Character Recognition, ...

– **Automatic Speech Recognition**
  - We need a language model to disambiguate what the acoustic model has eared
  - Choice between "This is an **example** of an homophone" or "This is an **egg-sample** of an homophone"
    – p("example") =$5.7 \ \ 10^{-10}$ $\Rightarrow$ is more likely in this context
    – p("egg-sample")=$3.2 \ \ 10^{-13}$

– **Language model**: p(sentence)
  - $p(y^{\langle 1 \rangle}, y^{\langle 2 \rangle}, \ldots, y^{\langle T_y \rangle})$

Language model

– **Notation:**

- $\mathbf{w} := w_1^L := w_1, w_2, \ldots w_L$

– **Goal:**

- **Estimate the (non-zero) probability of a word sequence for a given vocabulary**

$$P(w_1^L) = P(w_1, w_2, \ldots, w_L)$$
$$= \prod_{i=1}^{L} P(w_i \,|\, w_1^{i-1}) \qquad \forall i, w_i \in V$$
$$= P(w_1)P(w_2 \,|\, w_1)P(w_3 \,|\, w_1, w_2)P(w_4 \,|\, w_1, w_2, w_3)\ldots P(w_L \,|\, w_1 \ldots w_{L-1})$$

- Simplification: the **N-gram assumption :**

$$P(w_1^L) = \prod_{i=1}^{L} P(w_i \,|\, w_{i-N+1}^{i-1}), \quad \forall i, w_i \in V$$

– **Bi-gram (N=2) assumption**

$$P(w_1^L) = P(w_1)P(w_2 \,|\, w_1)P(w_3 \,|\, w_2)P(w_4 \,|\, w_3)\ldots P(w_L \,|\, w_{L-1})$$

- using **Recurrent Neural Network (RNN)**

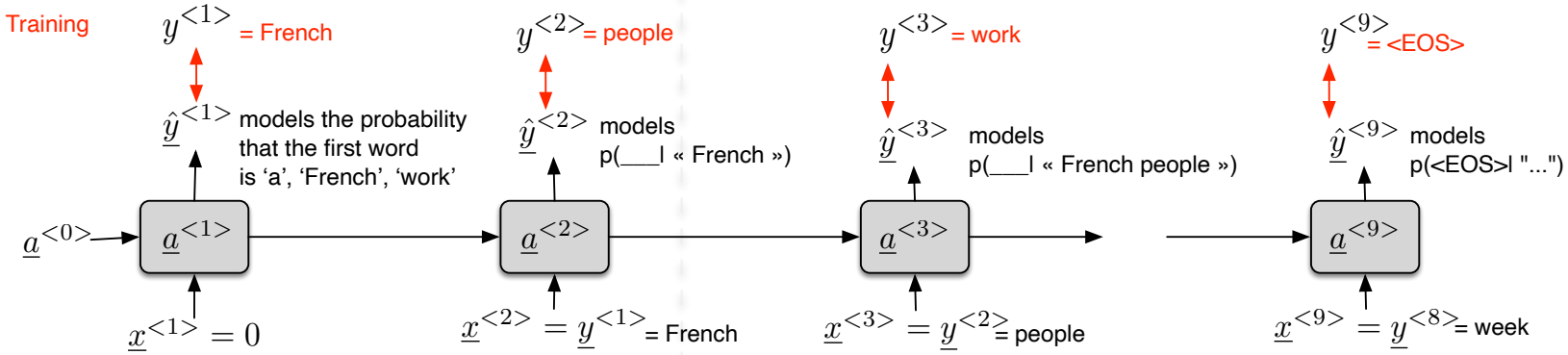
$$P(w_i \,|\, w_1^{i-1})$$

Language model

- **How to build a language model ?**
  - need a large corpus of English text
  - define a dictionary
  - tokenise the text
  - convert to one-hot-vector
    - + **<EOS>** (end of sentence)
    - + **<UNK>** (unknown word, very uncommon vocabulary): Gif-sur-Yvette

Language model: (1) Training using a RNN

- **Algorithm:**
  - start with empty context, empty input word
  - given the previous word as input $x^{\langle t \rangle}$ and the context (given by the hidden cell $a^{\langle t \rangle}$) predict (maximise the probability to observe) the next word $y^{\langle t \rangle}$ as output $\hat{y}^{\langle t \rangle}$
- **Example**: "French people work an average of 35 hours a week."



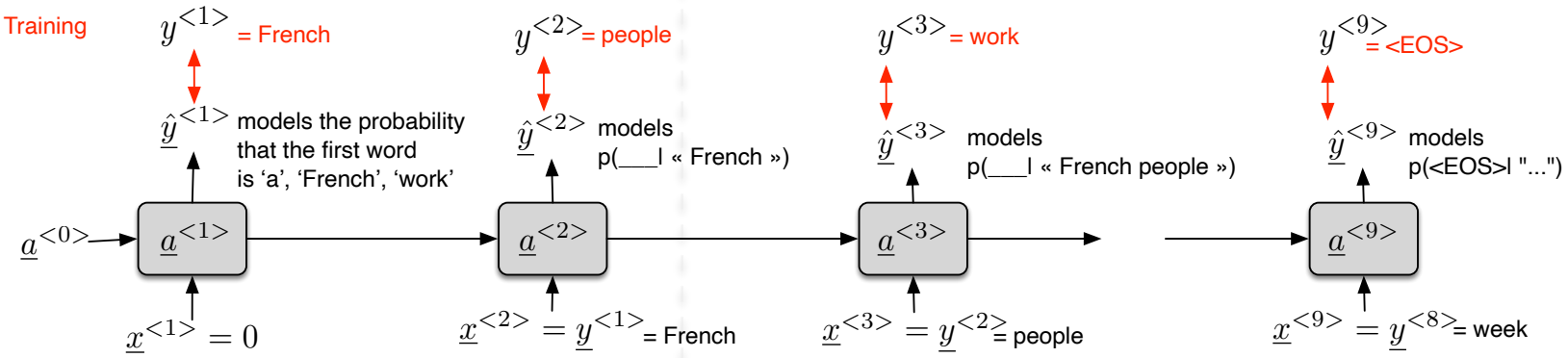
- **Training:**
  - $\hat{y}^{\langle t \rangle}$: output with a softmax with K neurons, gives the probability of each of the K words in the dictionary
  - $y^{\langle t \rangle}$: ground-truth, one-hot-encoding with K classes
  - Minimise the cross-entropy loss

$$\mathcal{L} = \sum_t \mathcal{L}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) \quad \text{with} \quad \mathcal{L}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) = -\sum_{c=1}^{K} y_c^{\langle t \rangle} \log(\hat{y}_c^{\langle t \rangle})$$

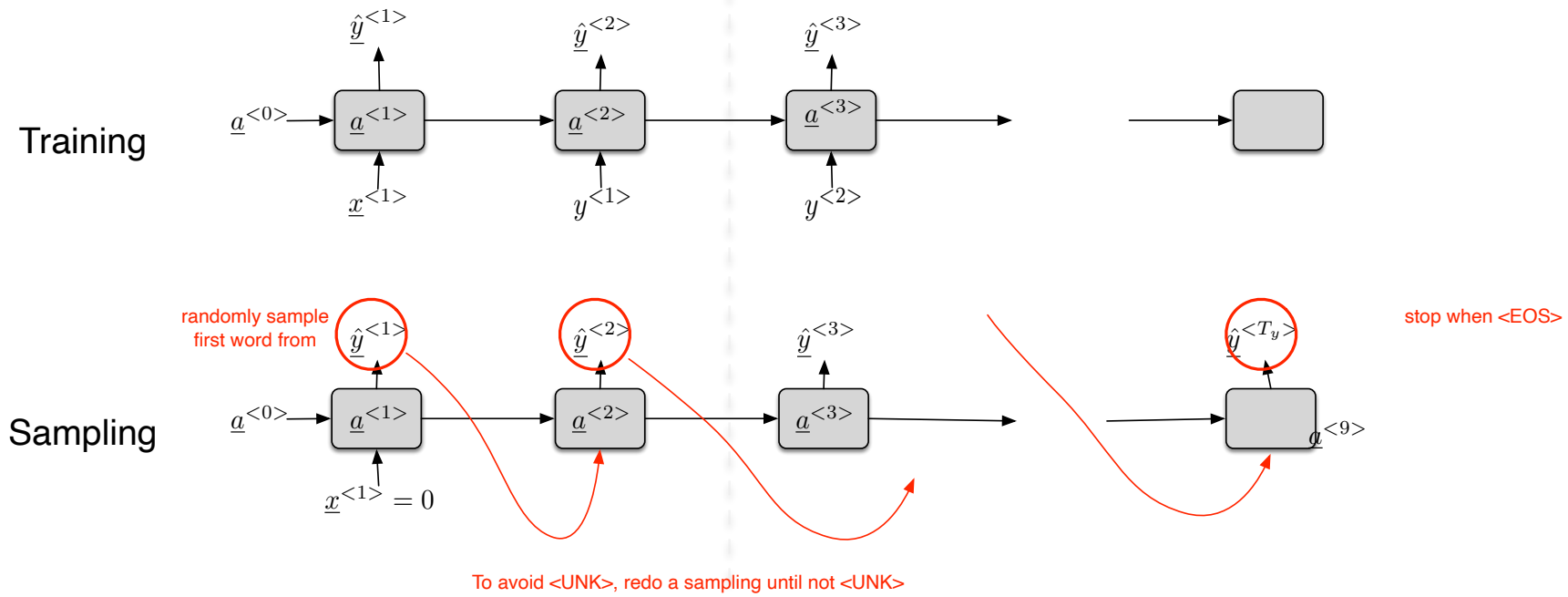
Language model: (2) Test using a RNN

– Given a new sentence of three words $\{y^{\langle 1\rangle}, y^{\langle 2\rangle}, y^{\langle 3\rangle}\}$ what is its probability ?

$$p(y^{\langle 1\rangle}, y^{\langle 2\rangle}, y^{\langle 3\rangle}) = p(y^{\langle 1\rangle}) \quad \text{given by the first softmax}$$
$$p(y^{\langle 2\rangle} | y^{\langle 1\rangle}) \quad \text{given by the second softmax}$$
$$p(y^{\langle 3\rangle} | y^{\langle 1\rangle}, y^{\langle 2\rangle}) \quad \text{given by the third softmax}$$

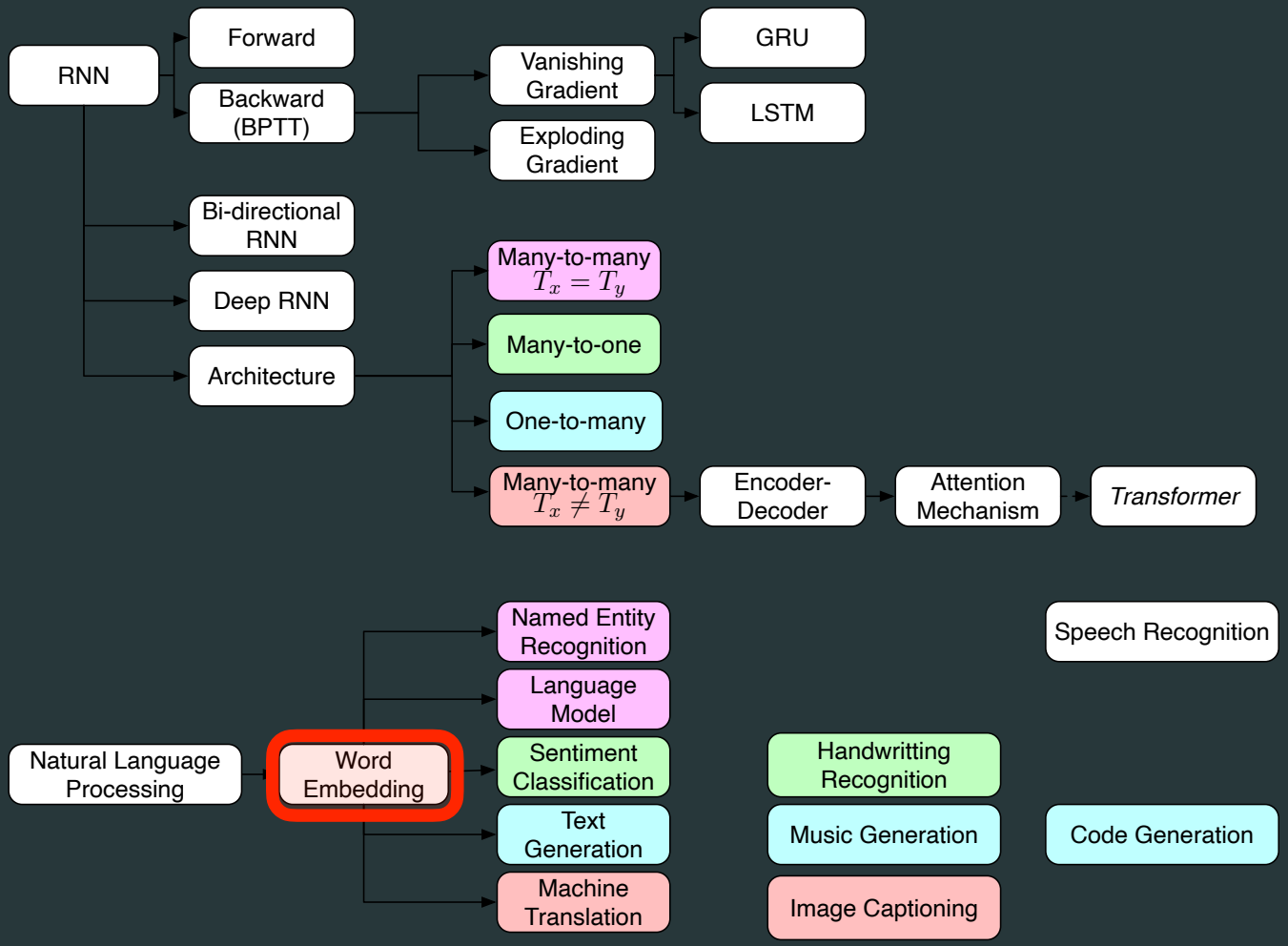


Language model: (3) Sampling novel sequences using RNN



- Word-level RNN
- Character-level RNN:
  - Vocabulary= ['a', 'b', 'c', ..., '.', ',', ';', 'A', 'B', 'C', ...]
- Examples:
  - Shakespeare, wikipedia, source-code generation
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/

One-hot-encoding



One-hot-encoding

- How to represent the individual words in a sentence ?
  - What will be $x^{\langle t \rangle}$?
- Construct the list of the N most used categories (words) in the corpus
  - **"vocabulary/dictionary"**
  - N is usually in the range 10.000 words - 50.000.
  - add a <UNK> (unknown word) for the words that are not in vocabulary/dictionary
- Each word is now associated with an ID

1		Angela
…		Boris
…		chancelor
367		Emmanuel
…		France
…		Germany
4075		is
…		Johnson
6830		Macron
…		Merkel
…		of
…		president
…		prime-minister
…		the
10.000		UK

One-hot-encoding

- **One-hot-encoding:**
  - the one-hot vector of an ID is a vector filled with 0s, except for a 1 at the position associated with the ID
  - each word w is associated with a one-hot-vector vectoro_{ID}
  - If $N = 10.000$, $x^{\langle t \rangle}$ has dimension 10.000
- a one-hot encoding makes no assumption about word similarity
  - all words are equally different from each other

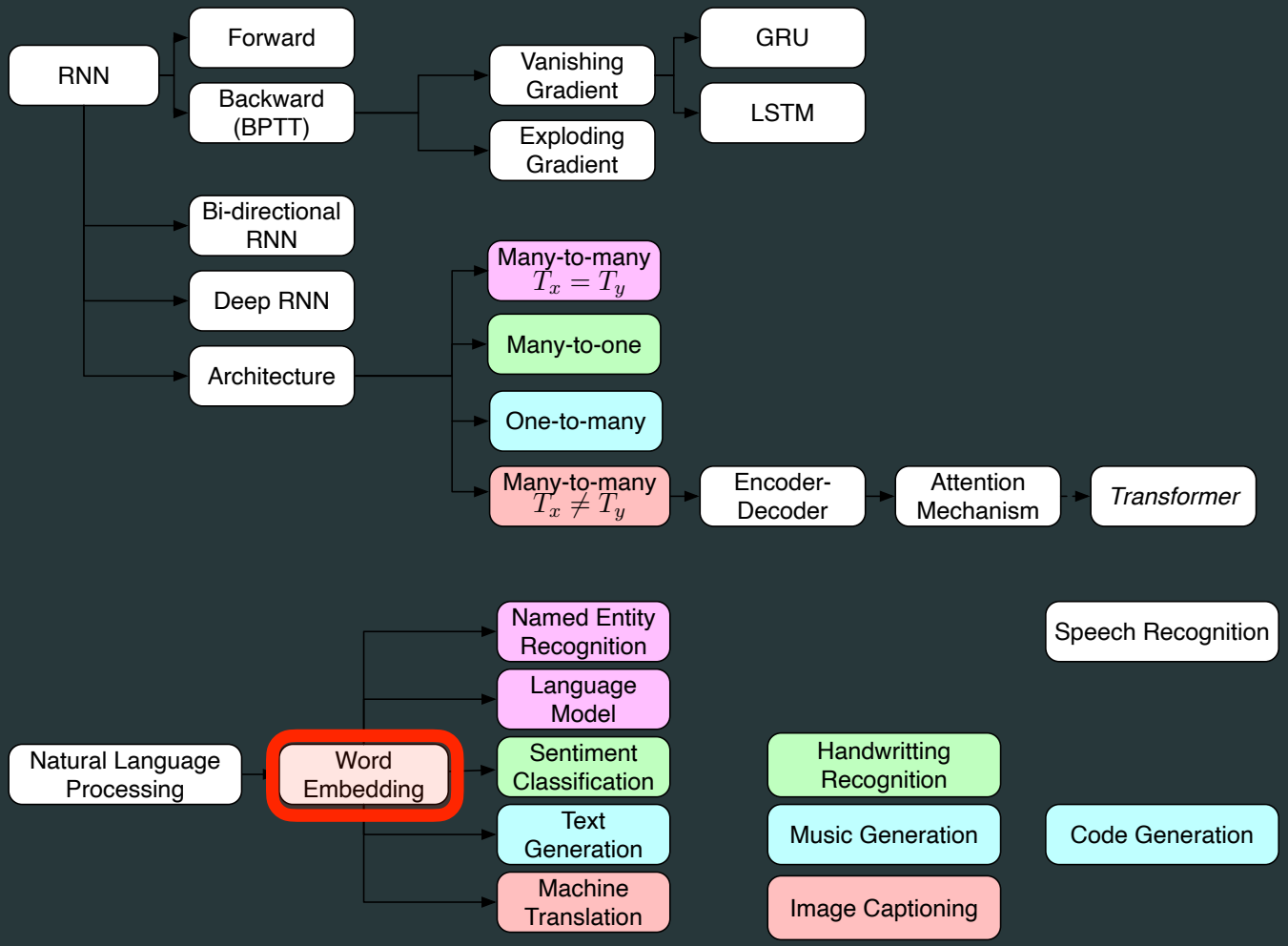
{x}	$x^{<1>}$	$x^{<2>}$	$x^{<3>}$	$x^{<4>}$	$x^{<5>}$	$x^{<6>}$	$x^{<7>}$
	Emmanuel	Macron	is	the	president	of	France
1	0	0	0				
2	0	0	0				
…	…	…	…				
367	1	0	0				
…	…	…	…				
4075	0	0	1				
…	…	…	…				
6830	0	1	0				
…	…	…	…				
10.000	0	0	0				

One-hot-encoding

- **Weaknesses of one-hot-encoding**
  - it is very high-dimensional
    - vulnerability to overfitting, computationally expensive
  - all words are equally different from each other
    - the inner product between any two one-hot vectors is zero
    - as a consequence, we cannot generalise

- **Example:**
  - we have a trained a language model to complete the sentence
    - The president is on a **boat**" ___?___ ⇒"trip"
  - since there is no specific relationship between "boat" and "plane", the algorithm cannot complete the sentence
    - The president is on a **plane** ___?___ ⇒ ?

Word embedding



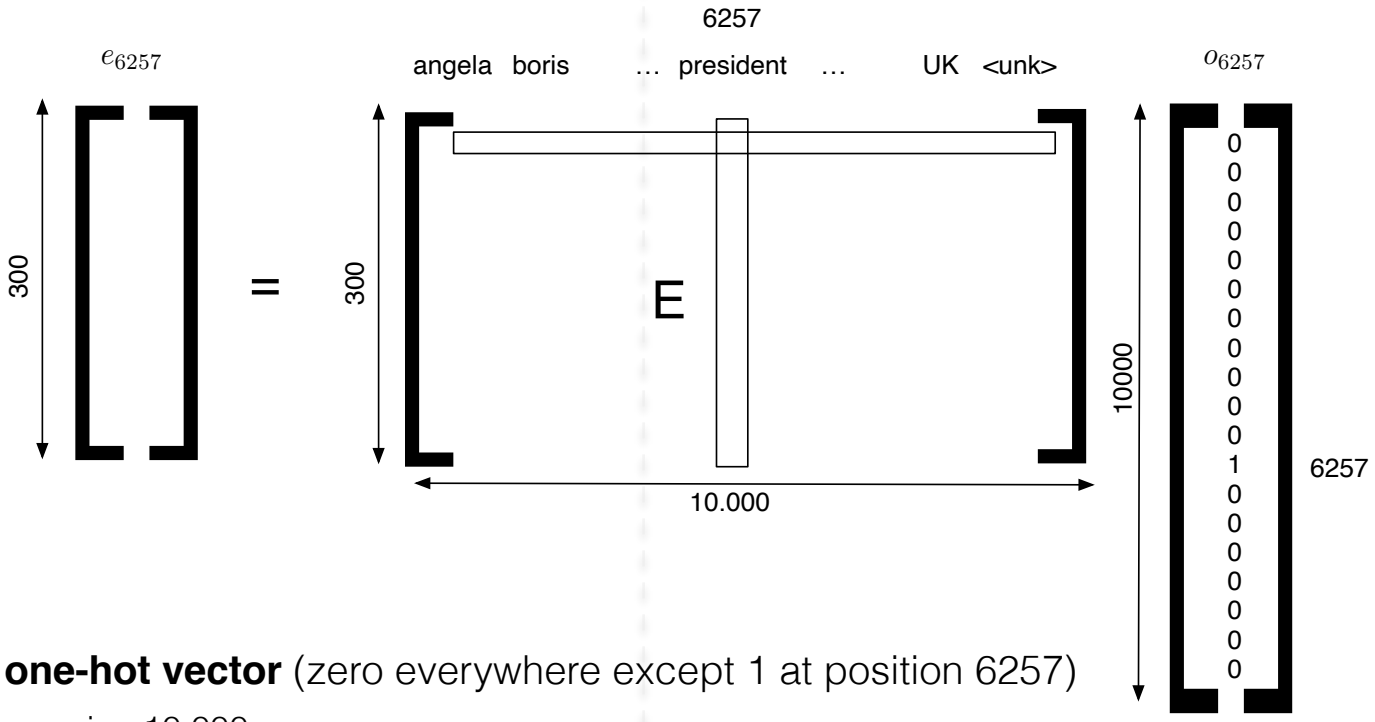
Word embedding

- learn a continuous representation of words
- each word w is associated with a real-valued vector e_{ID}
- if embedding size is 300, e_{5391} is a 300 dimensional vector
- we would like the distance $\|e_{ID1} - e_{ID2}\|$ to reflect the meaningful similarities between words

	Man	Woman	King	Queen	Uncle	Aunt	President	Chancelor	France	Germany	Boat	Plane
	5391	9853	4914	7157	456	6257	7124	3212	6789	1234	5923	8871
Gender	-1	1	-1	1	-1	1	-0,7	-0,3	0	0	0	0
Royal	0	0	1	1	0	0	0,5	0,5	0	0	0	0
Age	0	0	0,7	0,7	0,5	0,5	0,7	0,7	0	0	0	0
Country	0	0	0,5	0,5	0	0	0,5	0,2	1	1	0	0
Transportation	0	0	0	0	0	0	0	0	0	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...

- In this representation "boat" and "plane" are quiet similar
  - some of the features may differ but most features are similar
- Therefore the algorithm can find (by generalisation) that
  - The president is on a **boat** ___?___ ⇒ "trip"
  - The president is on a **plane** ___?___ ⇒ "trip"

Word embedding/ Matrix



- $\mathbf{o}_{6257}$= **one-hot vector** (zero everywhere except 1 at position 6257)
  - dimension 10.000
- $\underset{(300)}{\mathbf{e}_{6257}} = \underset{(300,10.000)}{\mathbf{E}} \cdot \underset{(10.000,1)}{\mathbf{o}_{6257}}$ = **embedding** of $\mathbf{o}_{6257}$
- $\mathbf{e}_j = \mathbf{E} \cdot \mathbf{o}_j$= embedding for word j
- In practice, use **look-up table** (take the column vector of $\mathbf{E}$ corresponding to 6257)

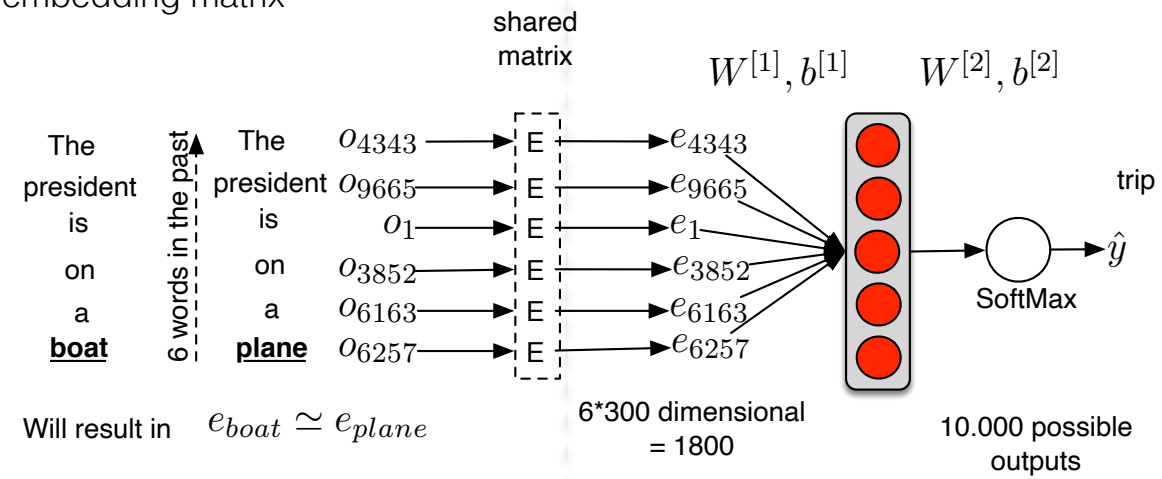
Word embedding/ Learning

- Various existing methods to learn word embedding:
  - Classic neural language model [Bengio et al., 2003]
  - Collobert and Weston model [Collobert, Weston, 2008]
  - Word2Vec [Mikolov et al. 2013]
    - Continuous bag-of-words (CBOW)
    - Skip-gram
  - Glove model [Pennington et al. 2014]

[Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. Journal of machine learning research, 3(Feb):1137–1155, 2003.]
[R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, pages 160–167. ACM, 2008.]
[T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.]
[T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013.]
[J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.]

Word embedding/ Classic neural language model

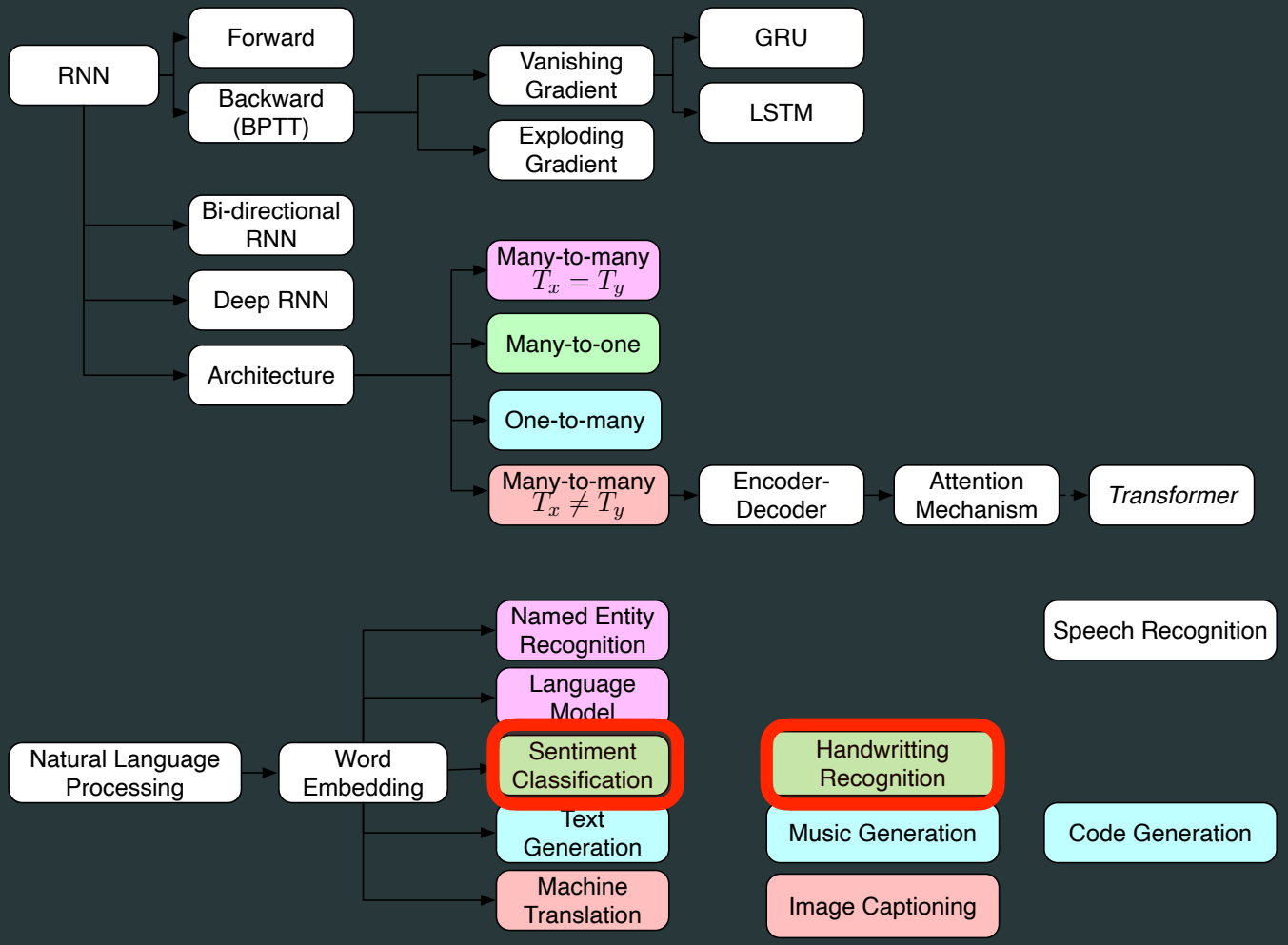
- **Method**: build a language model (learn to predict the following word) using a MLP
  - Use back-propagation to learn the parameters: $\mathbf{E}, \mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{W}^{[2]}, \mathbf{b}^{[2]}$
  - $\mathbf{E}$ is the embedding matrix



- **Example:**
  - "The president is on a boat $\underset{target}{\text{trip}}$ to the United States."
- **Other possible definition of the context**
  - last 4 words
  - 4 words on the left, 4 words on the right: "is on a boat ___?___ to the United States"
  - last 1 words: "boat ___?___"
  - nearby 1 word (skip-gram): "president ___?___"

[Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. Journal of machine learning research, 3(Feb):1137–1155, 2003.]

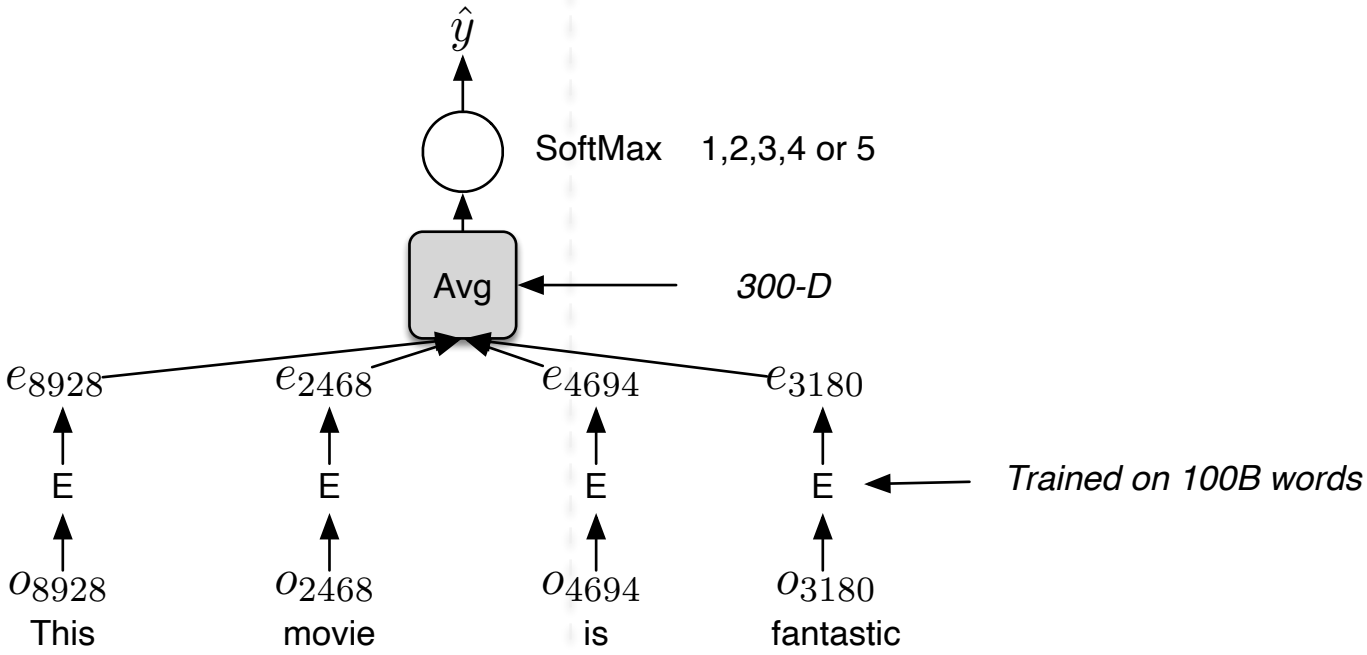
Application: Sentiment Classification



Application: Sentiment Classification

x	y
This movie is fantastic ! I really like it because it is so good !	
Not to my taste, will skip and watch another movie.	
This movie was completely lacking a good script, good actors and a good director.	

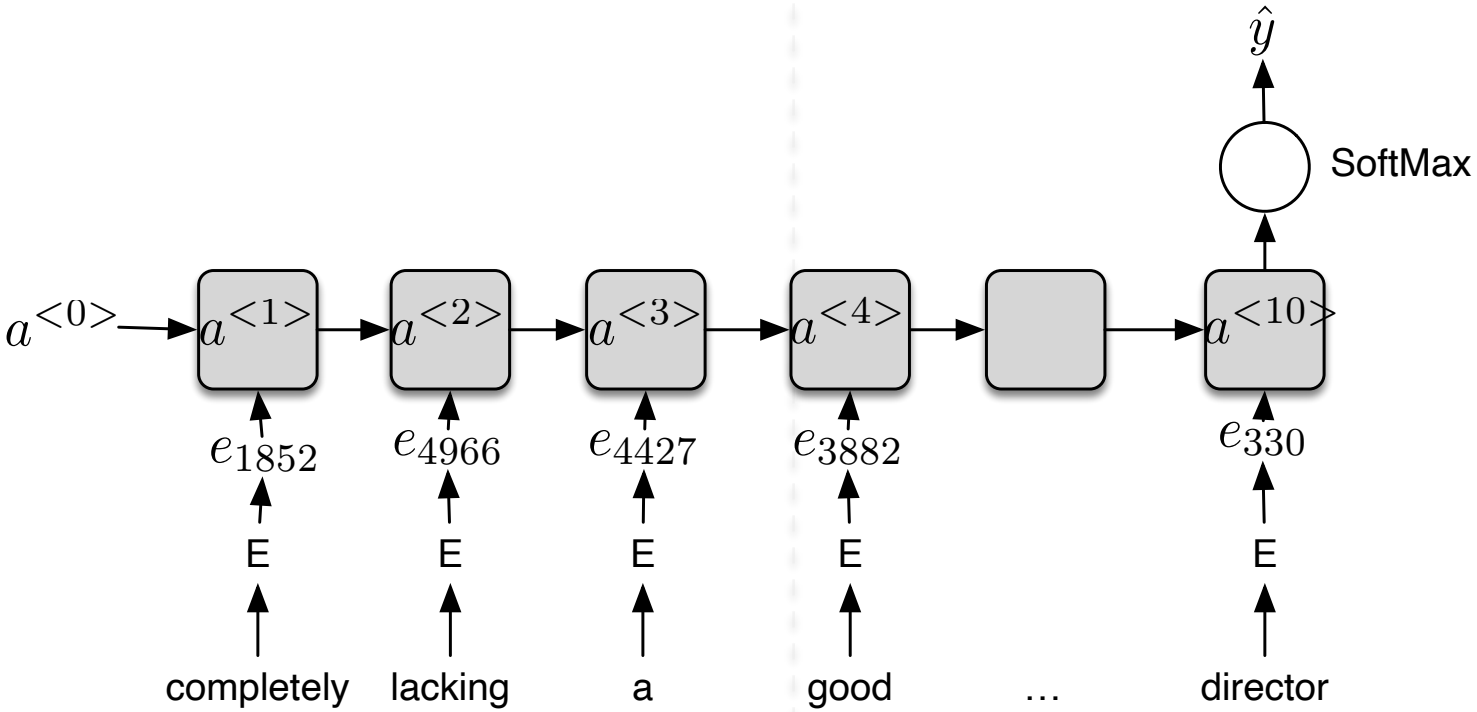
Application: Sentiment Classification / using a simple model



- Does not work for
  *"This movie was completely <u>lacking</u> a **good** script, **good** actors and a **good** director"*
  - This is because the Avg of "good" is positive
  - The model can not not understand that we means the opposite ("lacking")
  - We need a Sequence model ⇒RNN

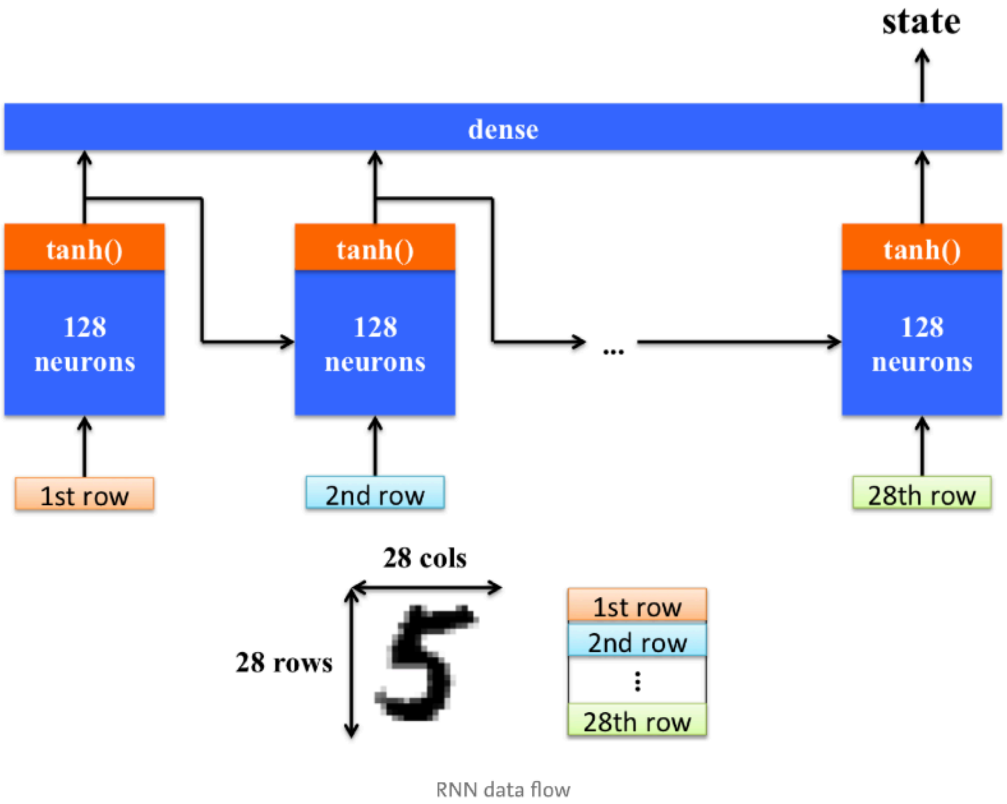
Application: Sentiment Classification / using a RNN

– **Many-to-one** architecture



Application: Handwritten Character Recognition

– **Many-to-one** architecture



RNN data flow

MNIST dataset

source https://medium.com/machine-learning-algorithms/mnist-using-recurrent-neural-network-2d070a5915a2

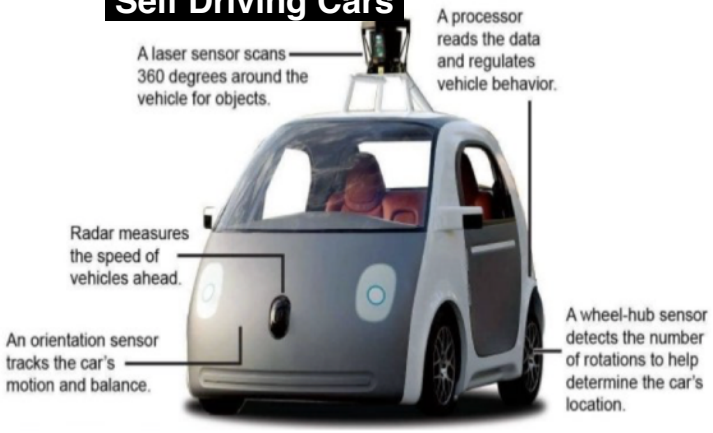
[Graves et al. 2008 "Unconstrained online handwriting recognition with recurrent neural networks"]\\
[Graves et al. 2009 "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks"]

Sequence to Sequence

Applications

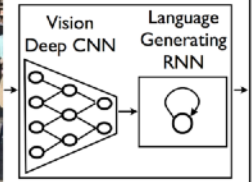
Automatic Speech Recognition

Self Driving Cars



DALL-E

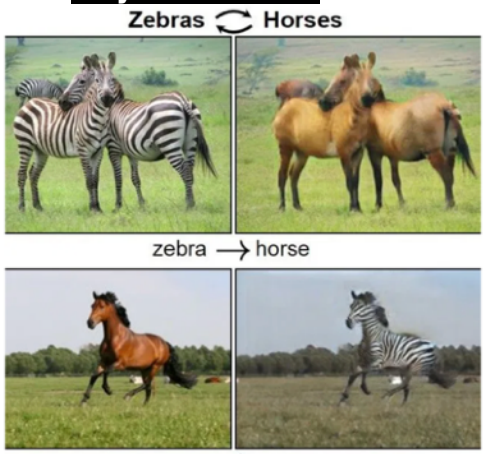
Automatic picture captioning



A group shopping at an outdoor market.

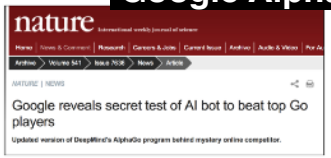
There are many vegetables at the fruit stand.

Style Transfer

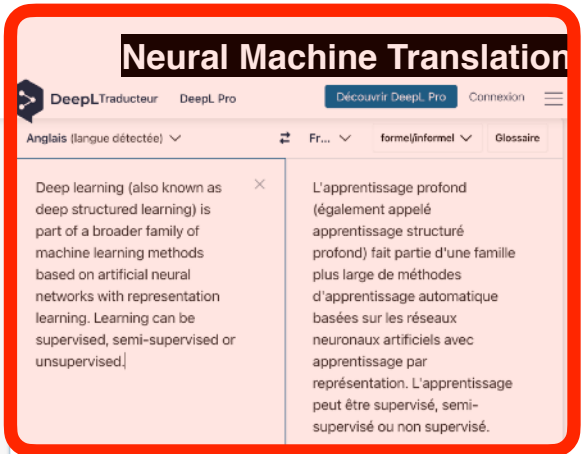


Google Alpha Go

DEEP LEARNING HAS MASTERED GO



Neural Machine Translation

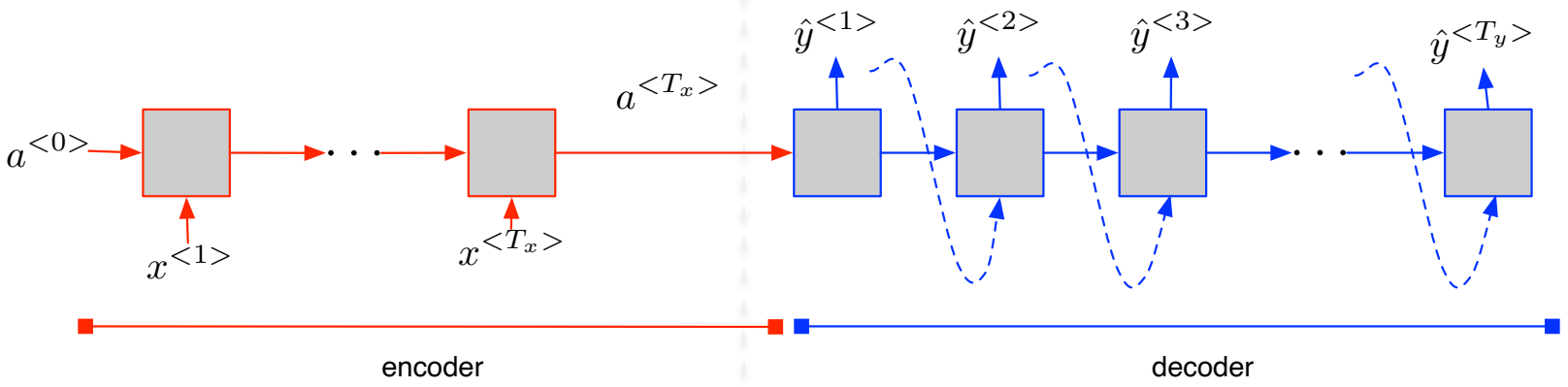


Sequence to Sequence

- What happens if $T_x \neq T_y$ (many-to-many architecture but with different lengths)
  - Example: Machine Translation, Automatic Speech Recognition

{x}	$x^{<1>}$	$x^{<2>}$	…				$x^{<7>}$		
	Deep	Learning	is	part	of	machine	learning		
{y}	$y^{<1>}$	$y^{<2>}$	…						$y^{<9>}$
	L'	apprentiss:	profond	fait	partie	de	l'	apprentiss:	machine

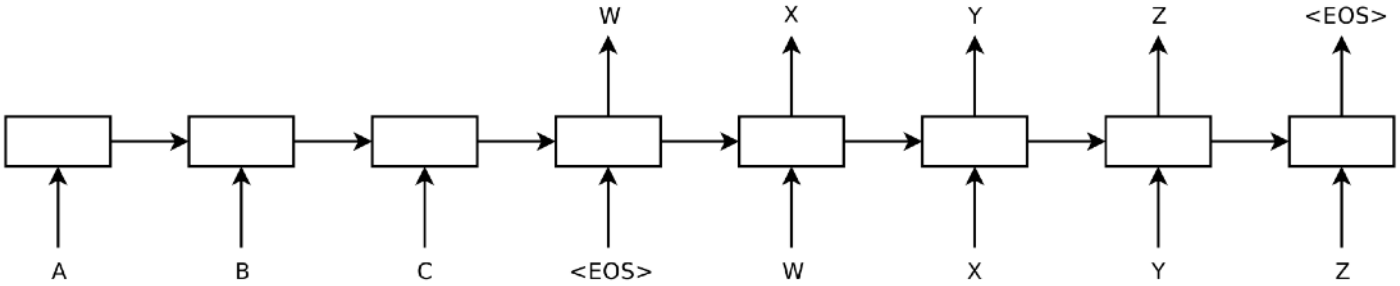
- Solution:
  - **Sequence to sequence** [Sutskever et al., 2014] or **Encoder-Decoder** [Cho et al., 2014] architecture



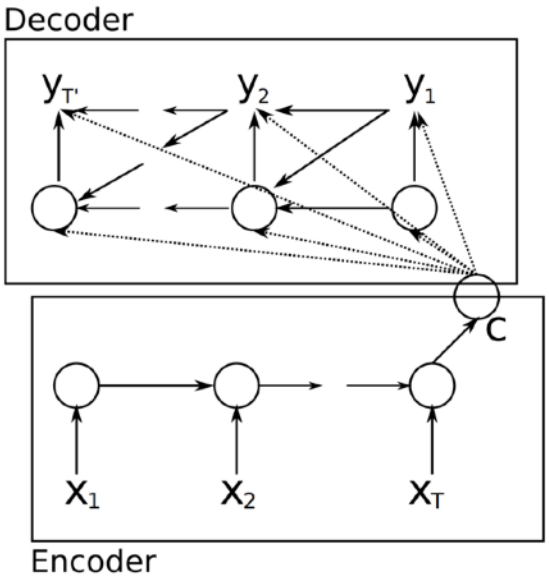
[Sutskever et al. 2014, "Sequence to sequence learning with neural networks"] LINK
[Cho et al. 2014 "Learning phrase representations using RNN encoder-decoder for statistical machine translation"] LINK

Introduction

[Sutskever et al. 2014, "Sequence to sequence learning with neural networks"] LINK

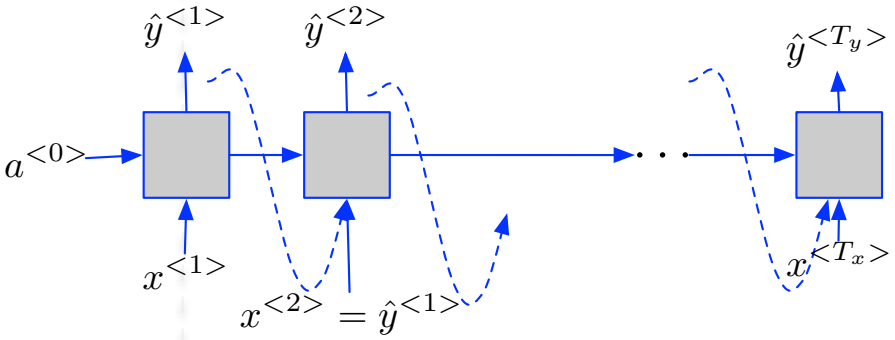


[Cho et al. 2014 "Learning phrase representations using RNN encoder-decoder for statistical machine translation"] LINK

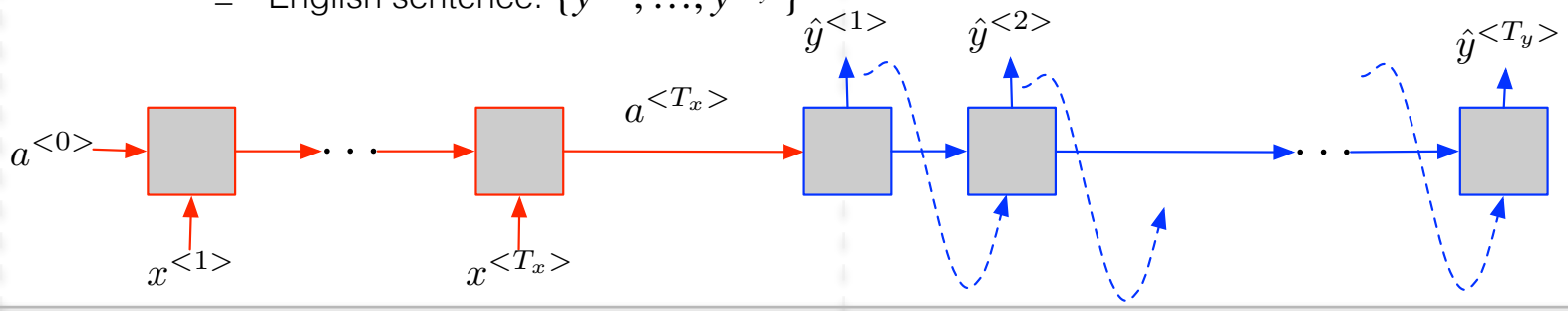


Machine Translation

- **Language model**
  - $p(y^{\langle 1 \rangle}, y^{\langle 2 \rangle}, \ldots, y^{\langle T_y \rangle})$



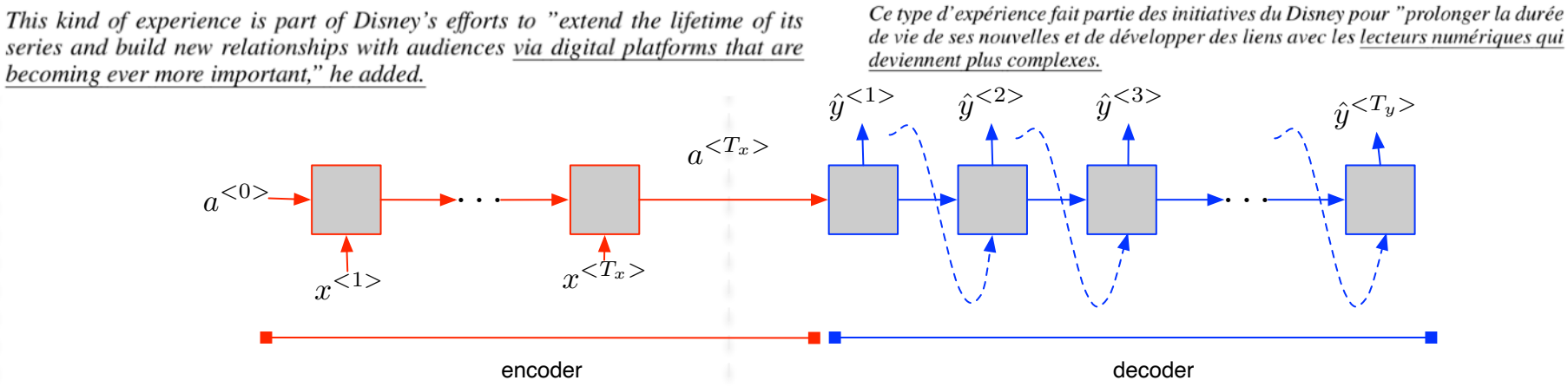
- **Machine Translation:**
  - Conditional language model: $p(y^{\langle 1 \rangle}, \ldots, y^{\langle T_y \rangle} | x^{\langle 1 \rangle}, \ldots, x^{\langle T_x \rangle})$
    - French sentence: $\{x^{\langle 1 \rangle}, \ldots, x^{\langle T_x \rangle}\}$
    - English sentence: $\{y^{\langle 1 \rangle}, \ldots, y^{\langle T_y \rangle}\}$



Attention model

Attention model

– **The problem of long sequences**

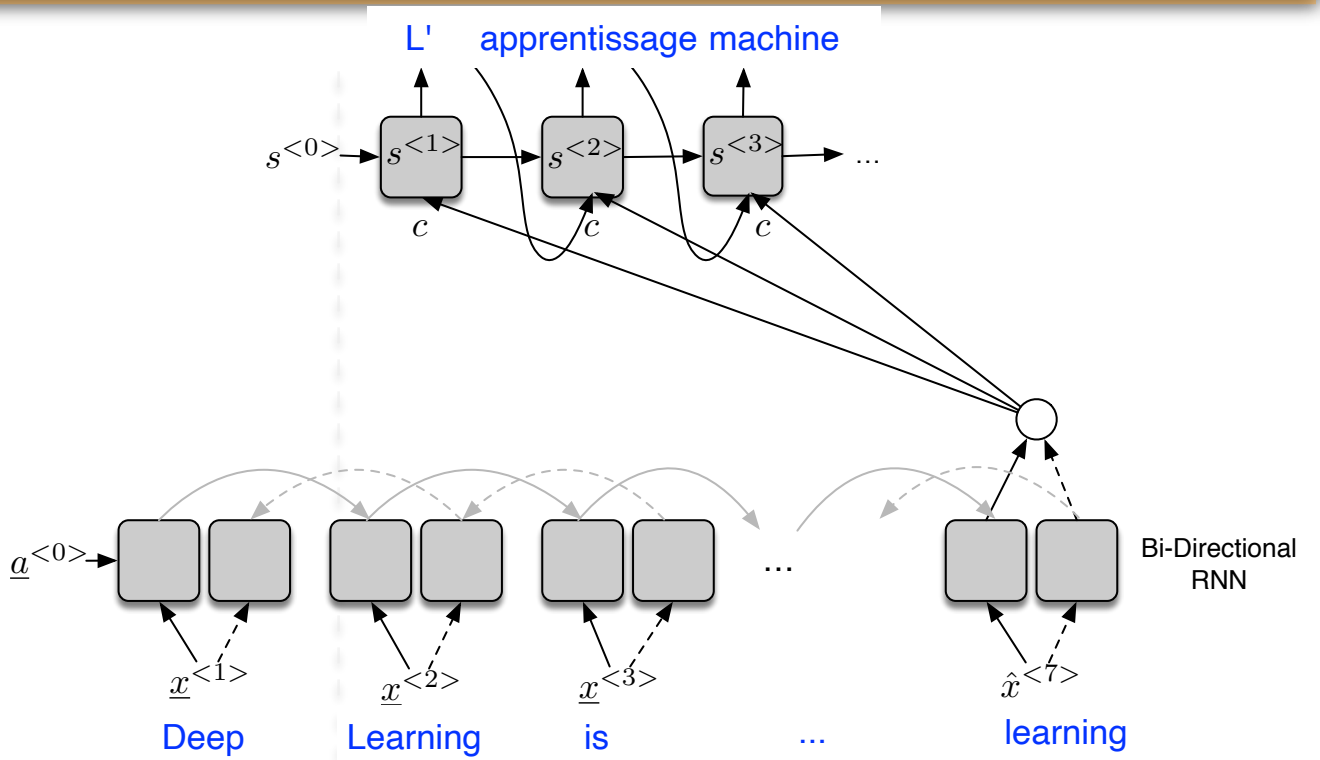


– Encoder/Decoder:
  - $a^{\langle T_x \rangle}$ is supposed to memorise the whole sentence then translate it;
    - human way: translate each part of a sentence at a time
– **Attention model?** (modification of the Encoder/Decoder)
  - $a^{\langle T_x \rangle}$ is replaced by a local version in the encoder ⇒ we pay attention on specific encoding

[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Sequence to sequence

Attention model

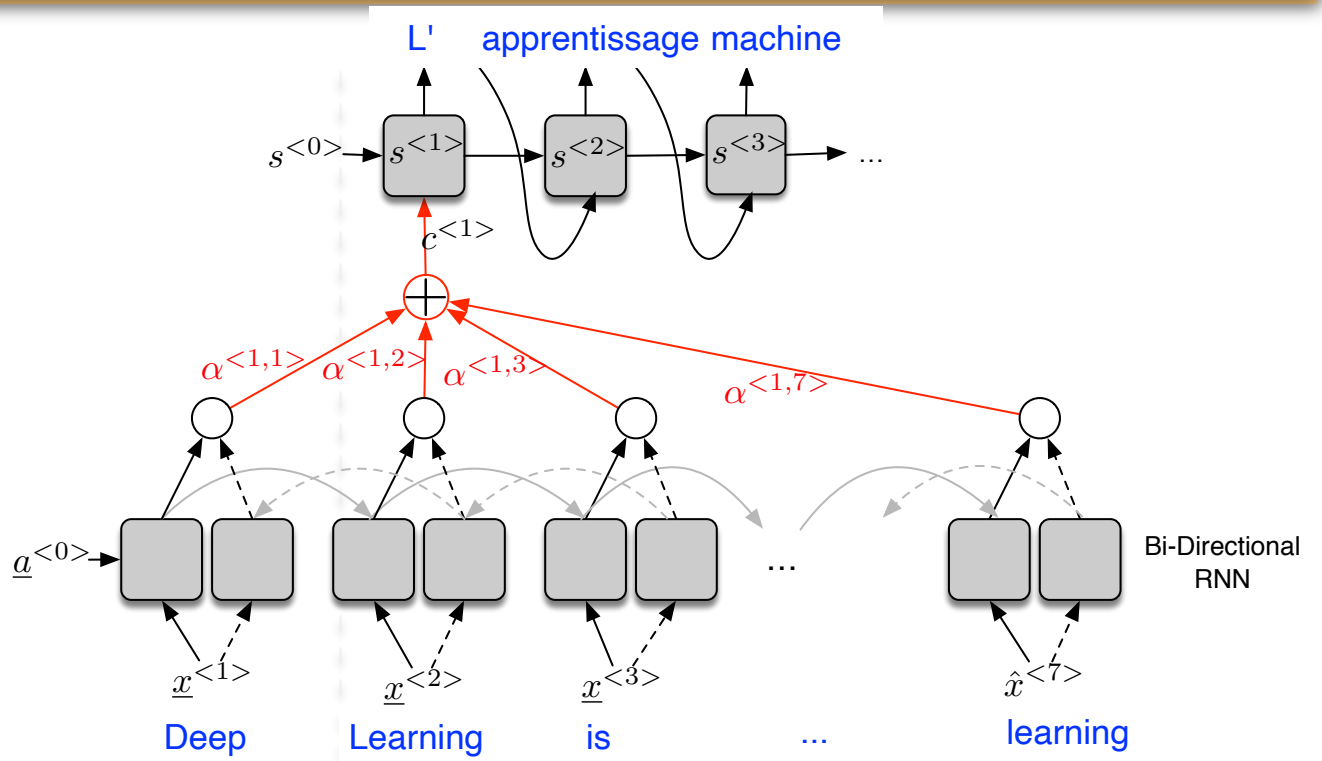


- In usual encoder-decoder,
  - information used for the decoding corresponds to the encoding at T_x: $c = a^{\langle T_x \rangle}$

[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Sequence to sequence

Attention model

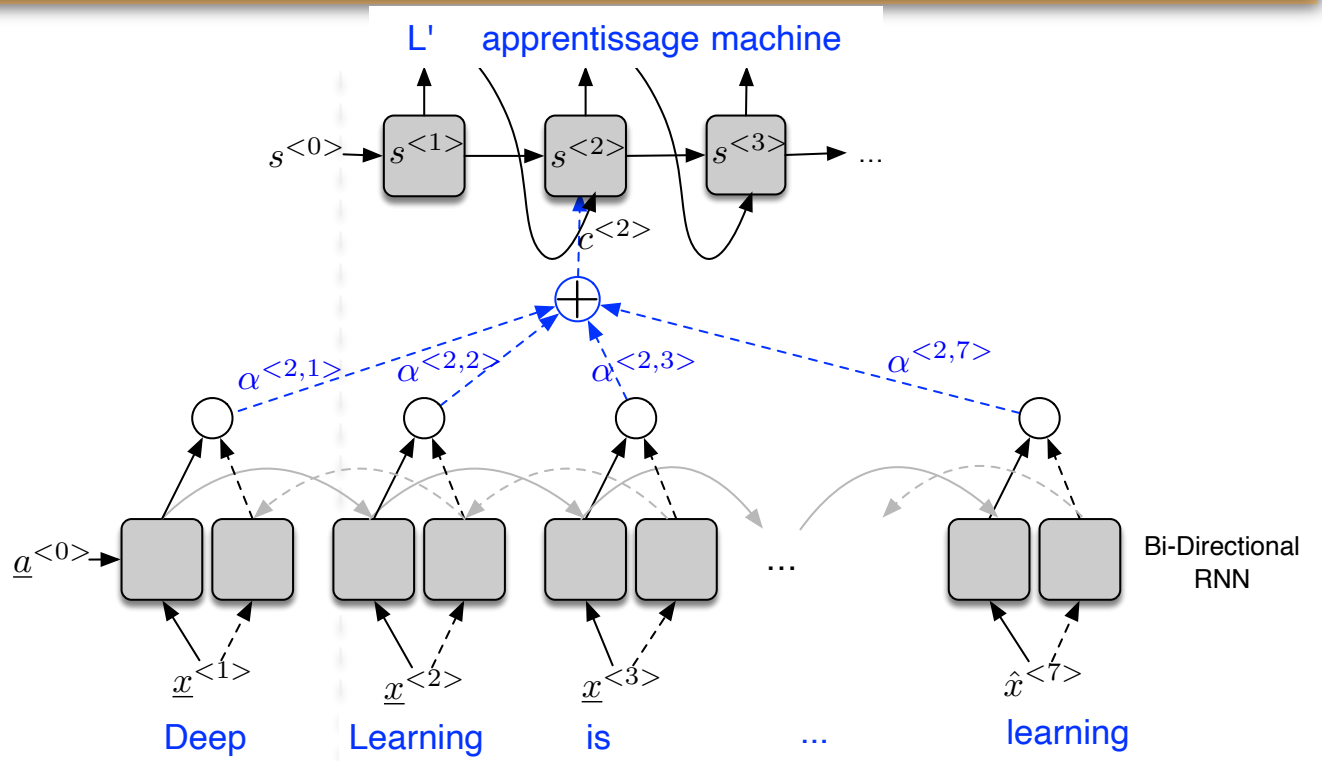


- Attention model
  - replace $c = a^{\langle T_x \rangle}$ by a local version $c^{\langle \tau \rangle}$
  - $c^{\langle \tau \rangle}$ is computed as a weighted sum of the encoding hidden states $a^{\langle t \rangle}$
- **Attention weights $\alpha^{\langle \tau,t \rangle}$:**
  - when generating information at time $\tau = 1$, how much, do we have to pay attention on information at time $t = \{1,2,3\ldots,7\}$
- Notation:
  - $a^{\langle t \rangle}, s^{\langle \tau \rangle}$: hidden states of encoder/ decoder

[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Sequence to sequence

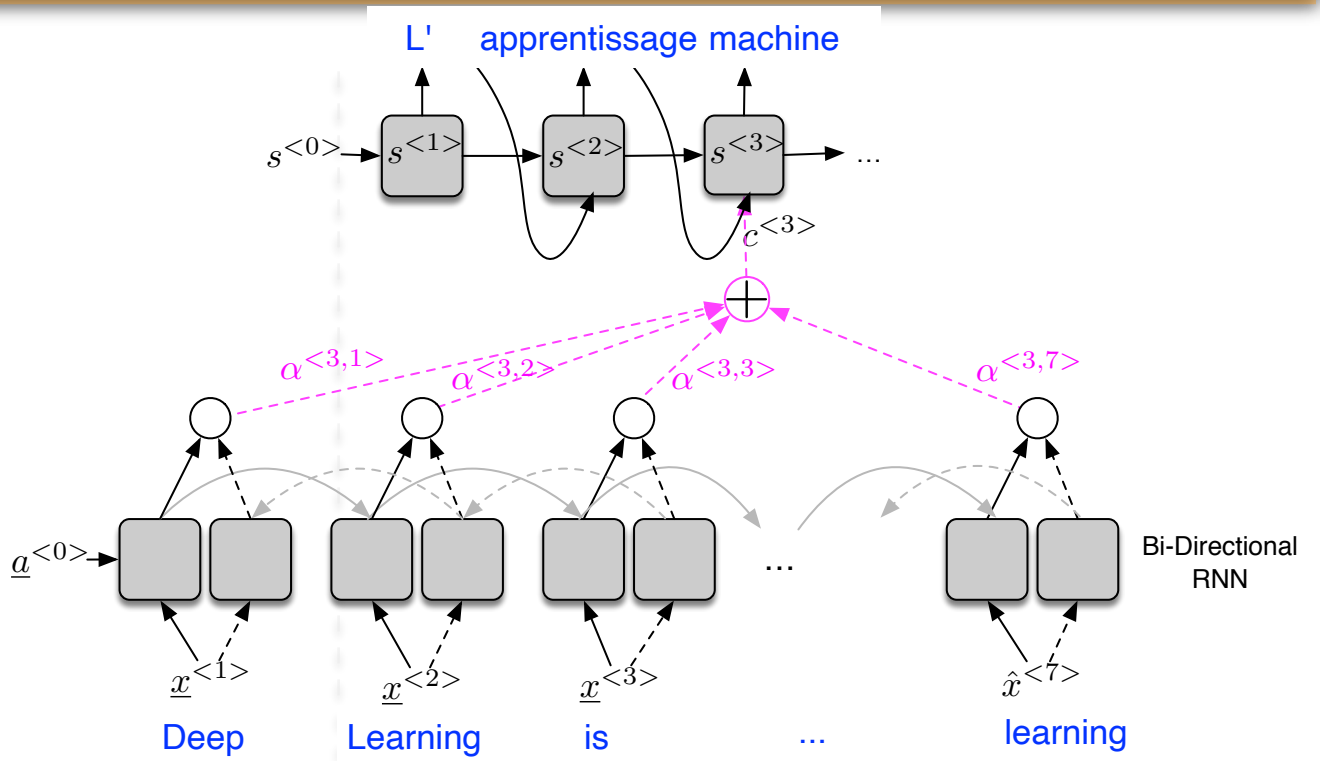
Attention model



- Attention model
  - replace $c = a^{\langle T_x \rangle}$ by a local version $c^{\langle \tau \rangle}$
  - $c^{\langle \tau \rangle}$ is computed as a weighted sum of the encoding hidden states $a^{\langle t \rangle}$
- **Attention weights $\alpha^{\langle \tau,t \rangle}$:**
  - when generating information at time $\tau = 1$, how much, do we have to pay attention on information at time $t = \{1,2,3\ldots,7\}$
- Notation:
  - $a^{\langle t \rangle}, s^{\langle \tau \rangle}$: hidden states of encoder/ decoder

[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Attention model



- Attention weights $\alpha^{\langle \tau=3,t \rangle}$
  - describe an "alignment" between information at
  - **encoding time** t: computed using $\overrightarrow{a}^{\langle t \rangle}$ and $\overleftarrow{a}^{\langle t \rangle}$
    - we note $a^{\langle t \rangle} = [\overrightarrow{a}^{\langle t \rangle}, \overleftarrow{a}^{\langle t \rangle}]$
  - **decoding time** $\tau = 3$: computed using $s^{\langle \tau=2 \rangle}$ (we do not yet observe $\tau = 3$)

[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

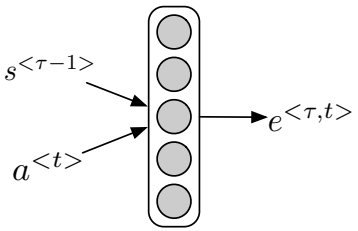
Attention model

- The **context vector** $c^{\langle \tau \rangle}$ at decoding time τ
  - computed as the sum of the annotations $a^{\langle t \rangle}$ weighted by their **attention weights** $\alpha^{\langle \tau,t \rangle}$

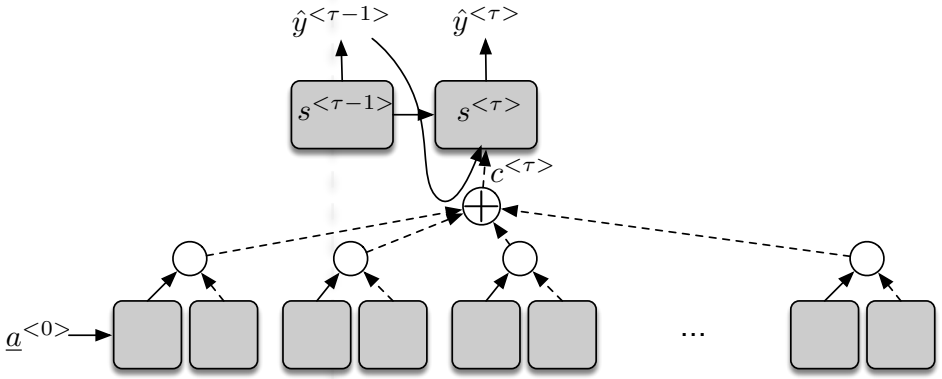
$$c^{\langle \tau \rangle} = \sum_t \alpha^{\langle \tau,t \rangle} a^{\langle t \rangle}$$

- The **attention weight** $\alpha^{\langle \tau,t \rangle}$
  - $\alpha^{\langle \tau,t \rangle}$= among of attention $y^{\langle \tau \rangle}$ should pay to $a^{\langle t \rangle}$
  - computed using a softmax on the **alignment model** $e^{\langle \tau,t \rangle}$

$$\alpha^{\langle \tau,t \rangle} = \frac{\exp(e^{\langle \tau,t \rangle})}{\sum_{t'=1}^{T_x} \exp(e^{\langle \tau,t' \rangle})} \text{ with } \sum_t \alpha^{\langle \tau,t \rangle} = 1$$



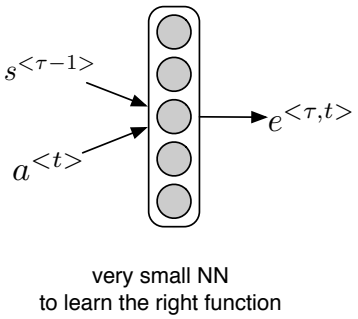
very small NN
to learn the right function

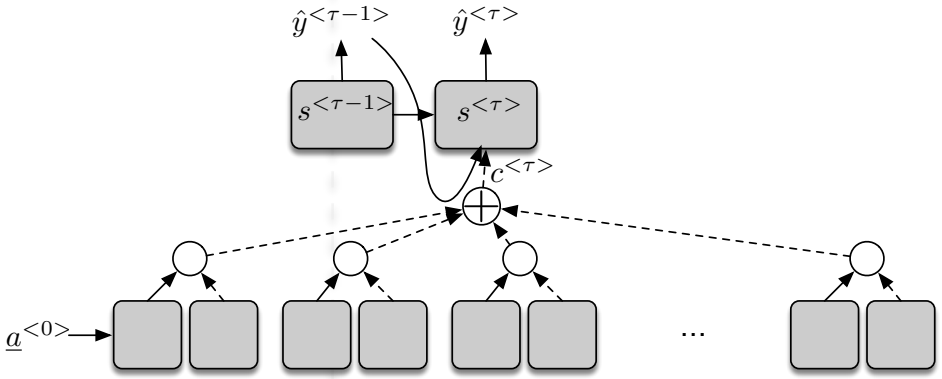


[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Attention model

- The **alignment model** $e^{\langle \tau,t \rangle}$
  - scores how well the inputs around position t match the output at position τ
  - computed using two inputs
    - the RNN hidden state $s^{\langle \tau-1 \rangle}$ (just before emitting $y^{\langle \tau \rangle}$)
    - the t-th annotation $a^{\langle t \rangle}$ of the input sentence.
  - computed using a feedforward neural network (jointly with all the other components)





[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Attention model

– Original sentence

This kind of experience is part of Disney's efforts to "extend the lifetime of its series and build new relationships with audiences via digital platforms that are becoming ever more important," he added.

– Encoder/Decoder without attention model

Ce type d'expérience fait partie des initiatives du Disney pour "prolonger la durée de vie de ses nouvelles et de développer des liens avec les lecteurs numériques qui deviennent plus complexes.

– Encoder/Decoder with attention model

Ce genre d'expérience fait partie des efforts de Disney pour "prolonger la durée de vie de ses séries et créer de nouvelles relations avec des publics via des plateformes numériques de plus en plus importantes", a-t-il ajouté.

Visualisation of $\alpha^{\langle \tau, t \rangle}$ (English to French)

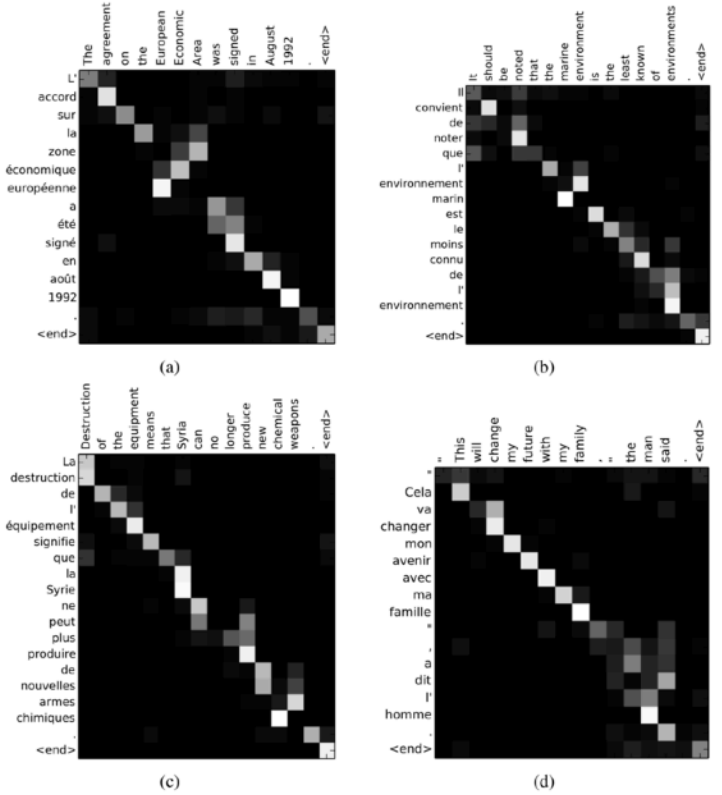
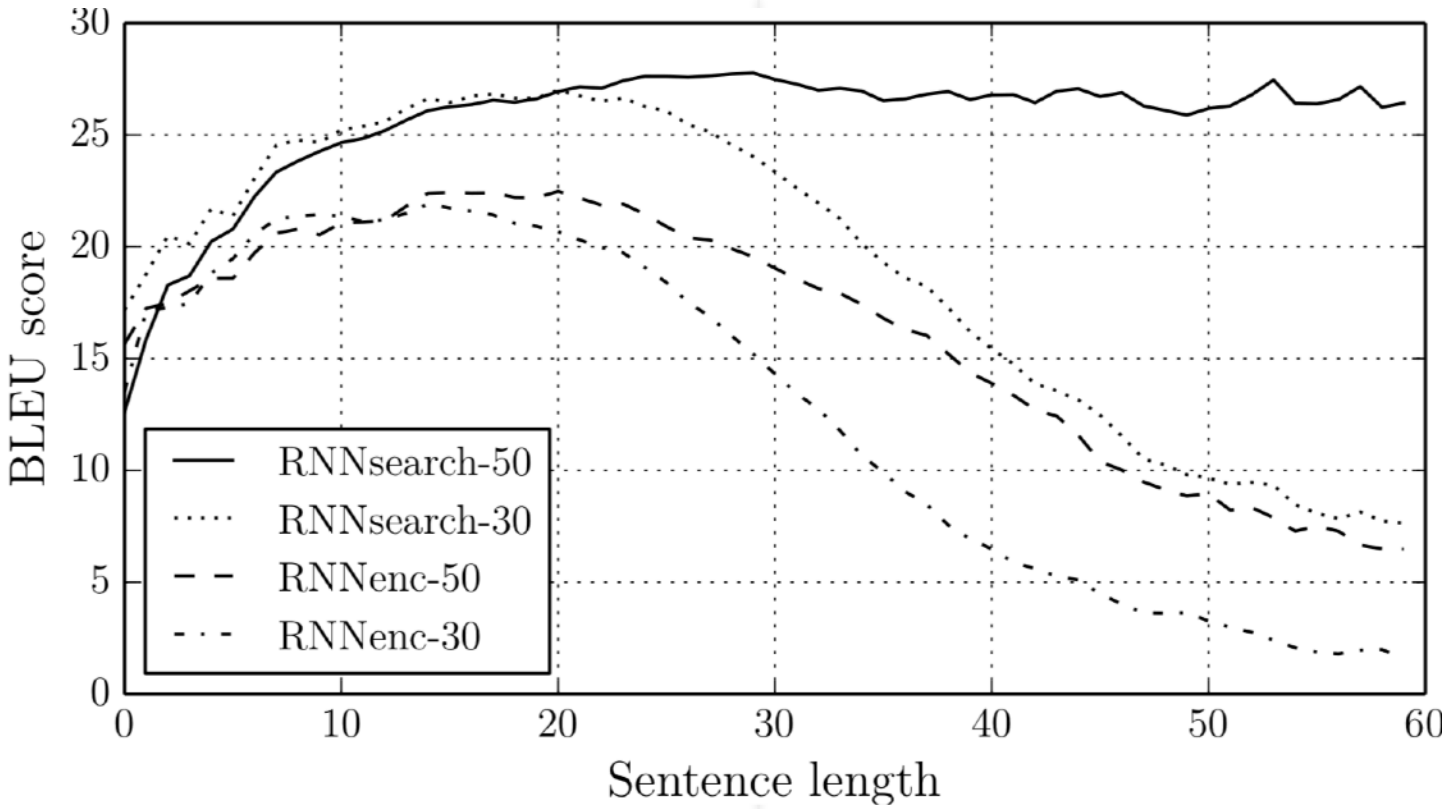


Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word (see Eq. (6)), in grayscale (0: black, 1: white). (a) an arbitrary sentence. (b–d) three randomly selected samples among the sentences without any unknown words and of length between 10 and 20 words from the test set.

[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Attention model

– Bleu score for long sentences

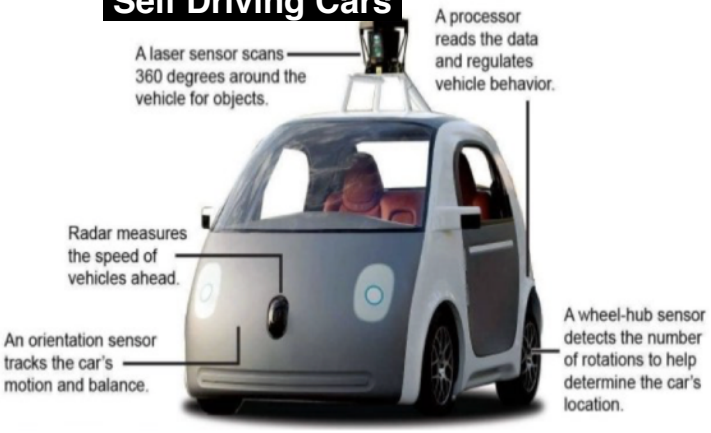


[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Applications

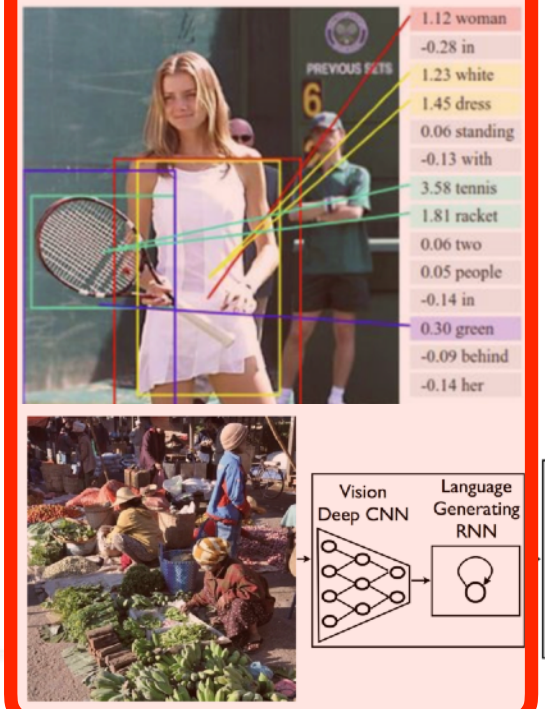
Automatic Speech Recognition

Self Driving Cars

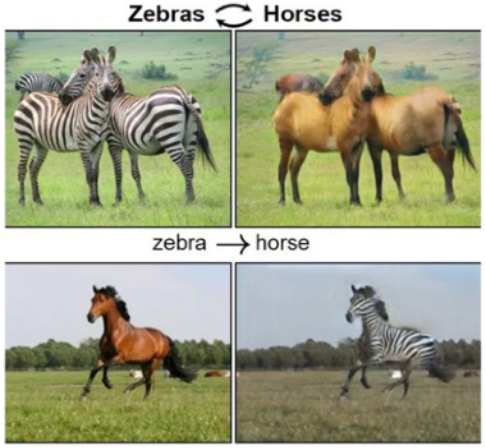


DALL-E

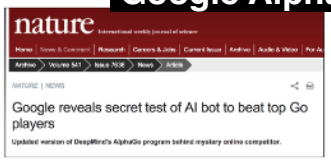
Automatic picture captioning



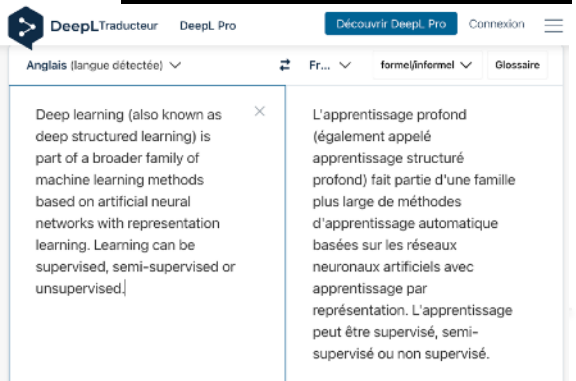
Style Transfer



Google Alpha Go

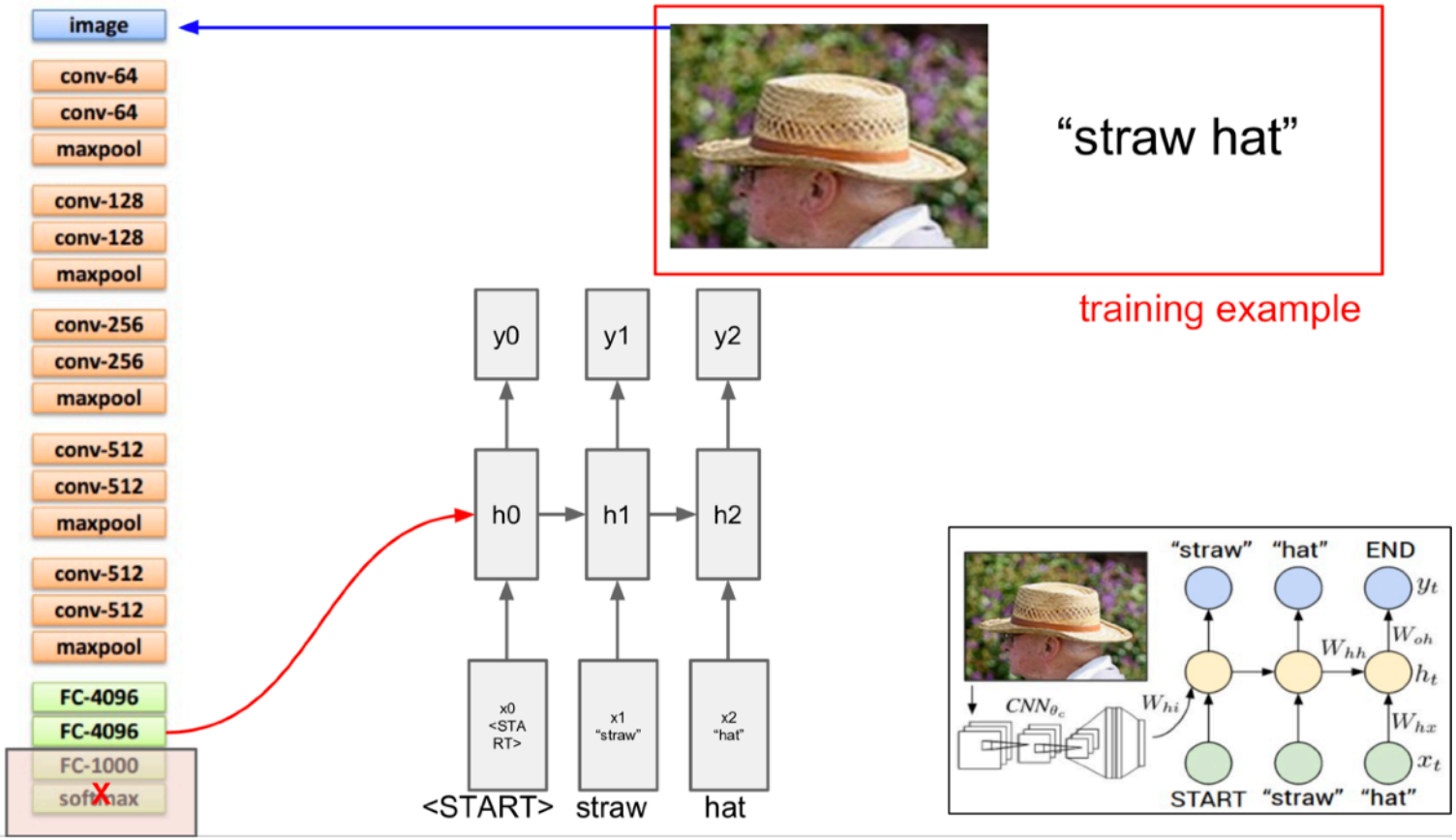


Neural Machine Translation



Sequence to sequence

Application: Image captioning

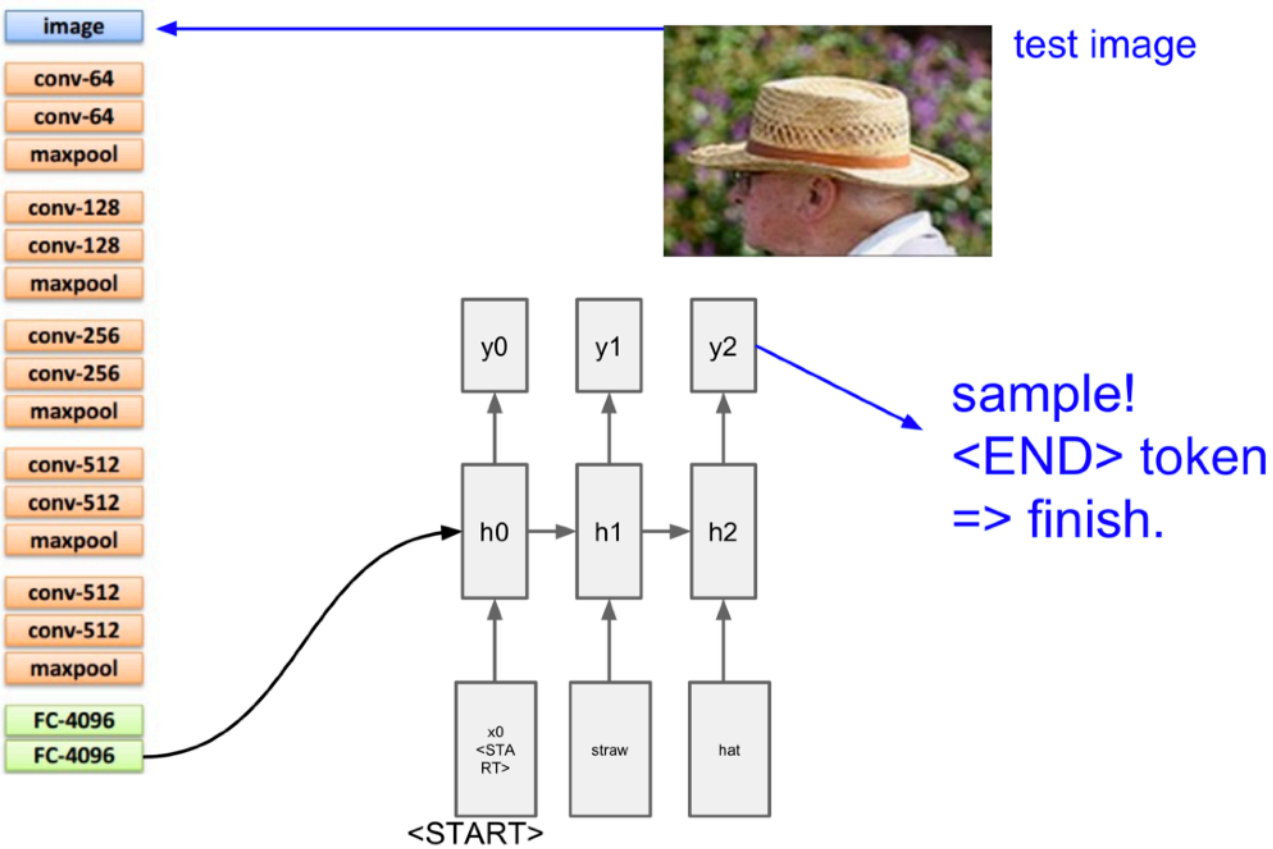


SOURCE: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

[Mao et al. 2014 "Deep captioning with multimodal recurrent neural networks (m-rnn)"] LINK
[Vinyals et al. 2015 "Show and tell: A neural image caption generator"] LINK
[Karpathy et al. 2015 "Deep visual-semantic alignments for generating image descriptions"] LINK

Sequence to sequence

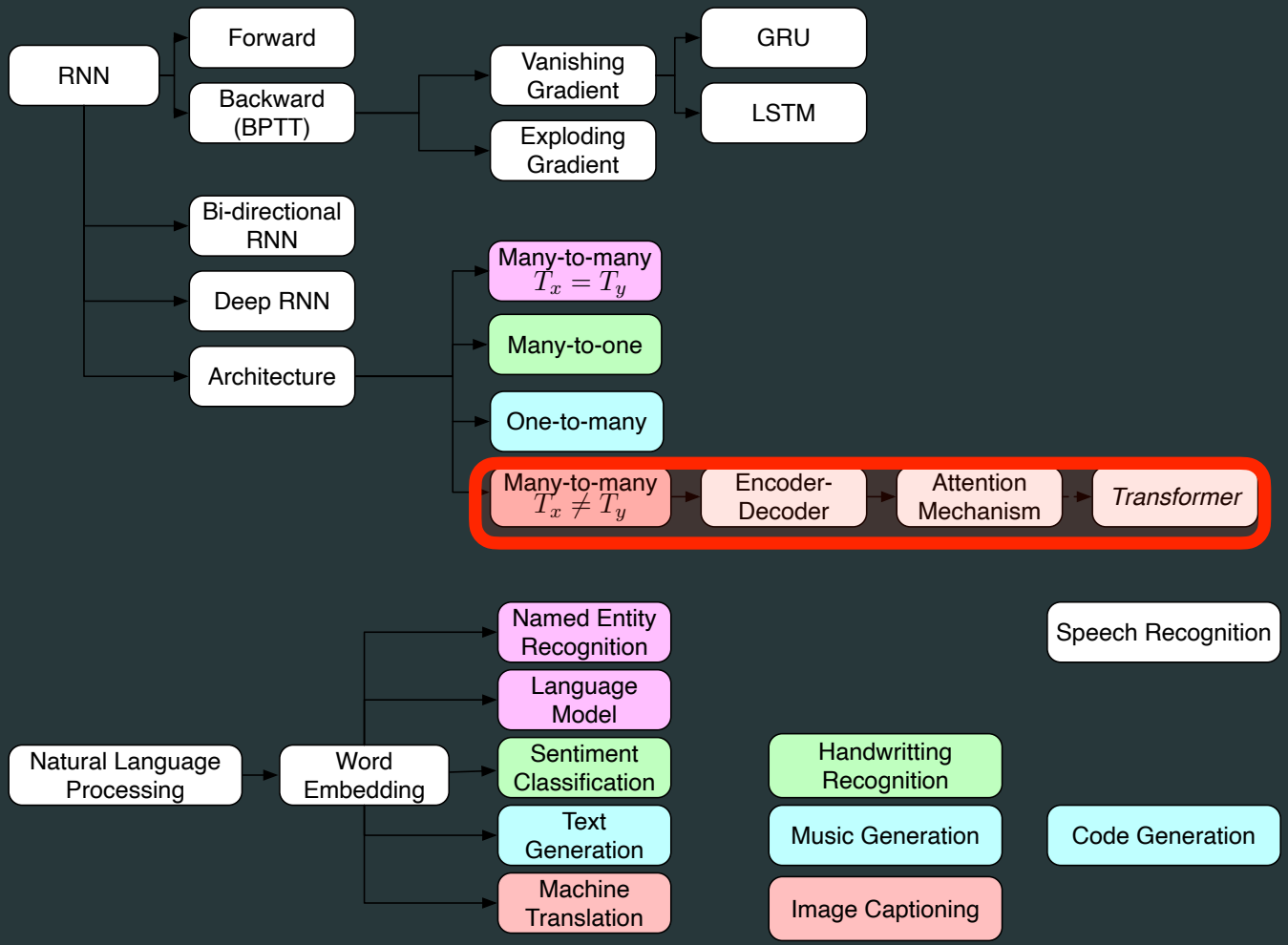
Application: Image captioning



SOURCE: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

[Mao et al. 2014 "Deep captioning with multimodal recurrent neural networks (m-rnn)"] LINK
[Vinyals et al. 2015 "Show and tell: A neural image caption generator"] LINK
[Karpathy et al. 2015 "Deep visual-semantic alignments for generating image descriptions"] LINK

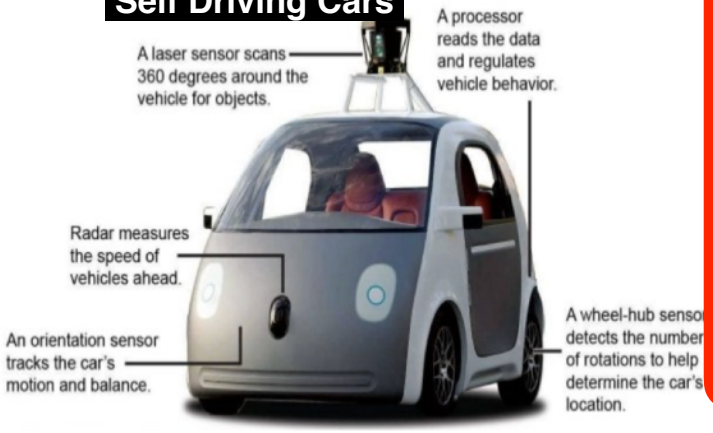
Transformer



Applications

Automatic Speech Recognition

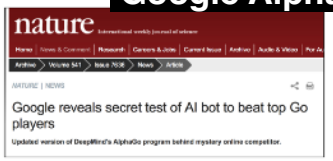
Self Driving Cars



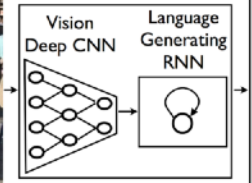
DALL-E

DEEP LEARNING HAS MASTERED GO

Google AlphaGo



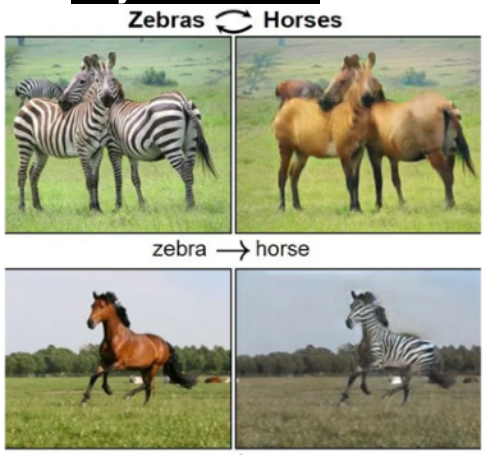
Automatic picture captioning



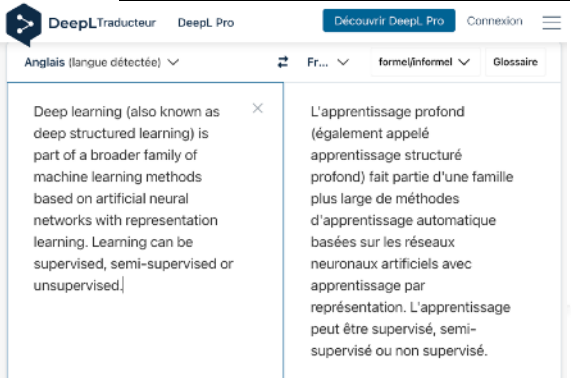
A group shopping at an outdoor market.

There are many vegetables at the fruit stand.

Style Transfer



Neural Machine Translation



Transformer

- **Global idea:**
  - get rid of complex recurrent or convolutional neural networks that include an encoder and a decoder
  - a new simple network architecture based solely on attention mechanisms, dispensing with recurrence and convolutions entirely
  - still an encoder decoder

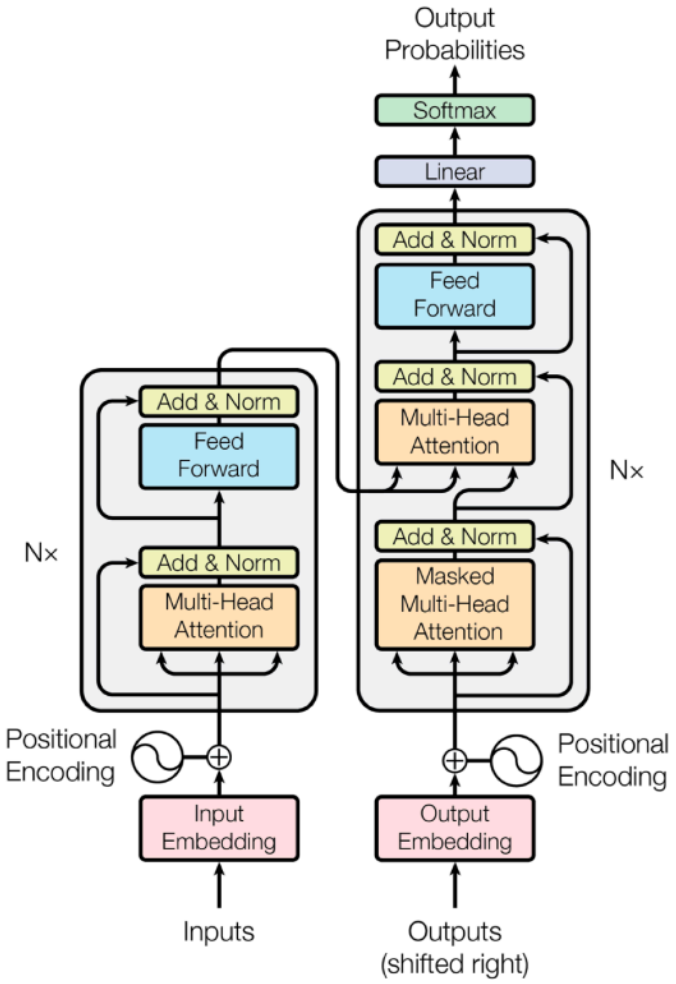


Figure 1: The Transformer - model architecture.

[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

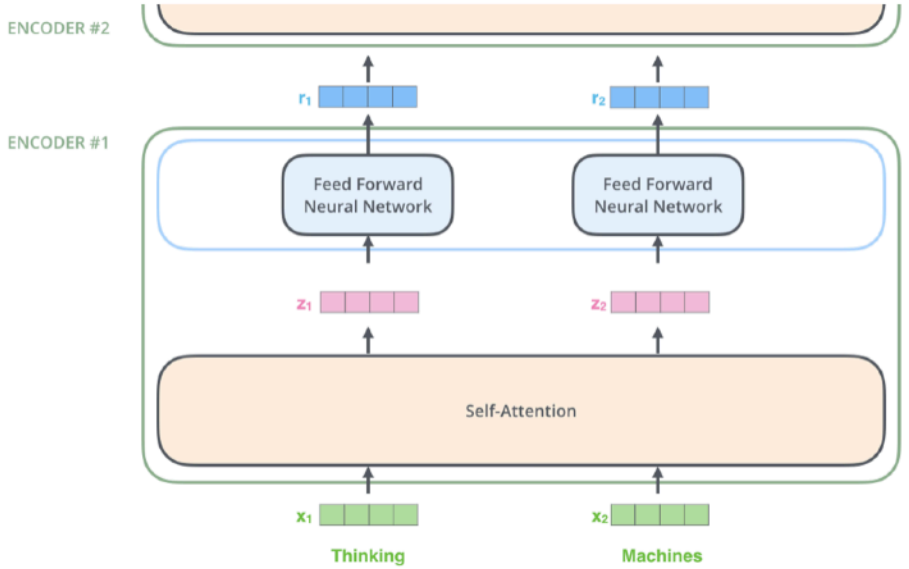
Transformer

- **Self-Attention :**
  - Input sentence we want to translate : "*The animal didn't cross the street because it was too tired*"
    - What does "*it*" in this sentence refer to ? Is it referring to the "*street*" or to the "*animal*" ?
  - When the model is processing the word "*it*", self-attention allows it to associate "*it*" with "*animal*"
  - As the model processes each word (each position in the input sequence), self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word

[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

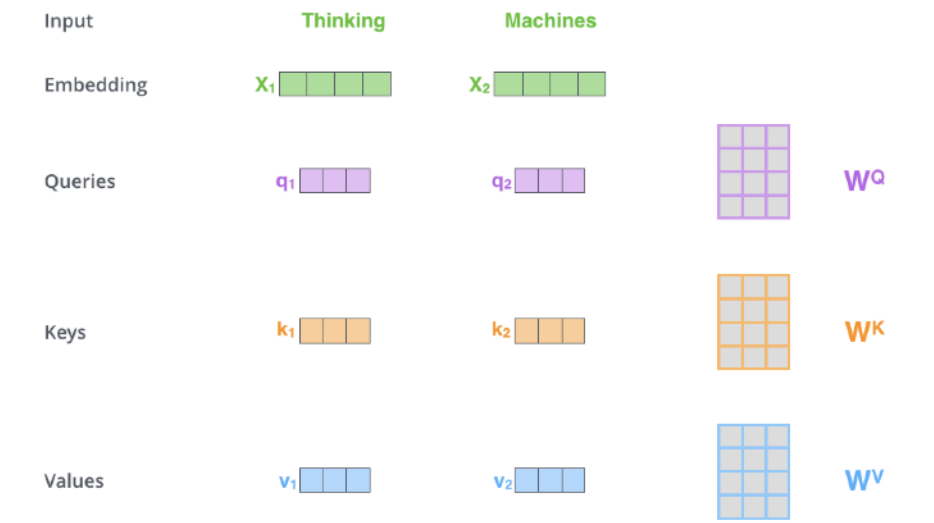
- **Now We're Encoding !**
  - An encoder receives a list of vectors as input
  - It processes this list by passing these vectors into a **self-attention** layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

- **First step** :
  - create three vectors from each of the encoder's input vectors x_i:
    - a Query vector q_i,
    - a Key vector k_i
    - a Value vector v_i
  - the vectors are created by multiplying the input by three matrices W^Q, W^K, W^V that are trained during the training process
  - the new vectors are smaller in dimension than the embedding vector

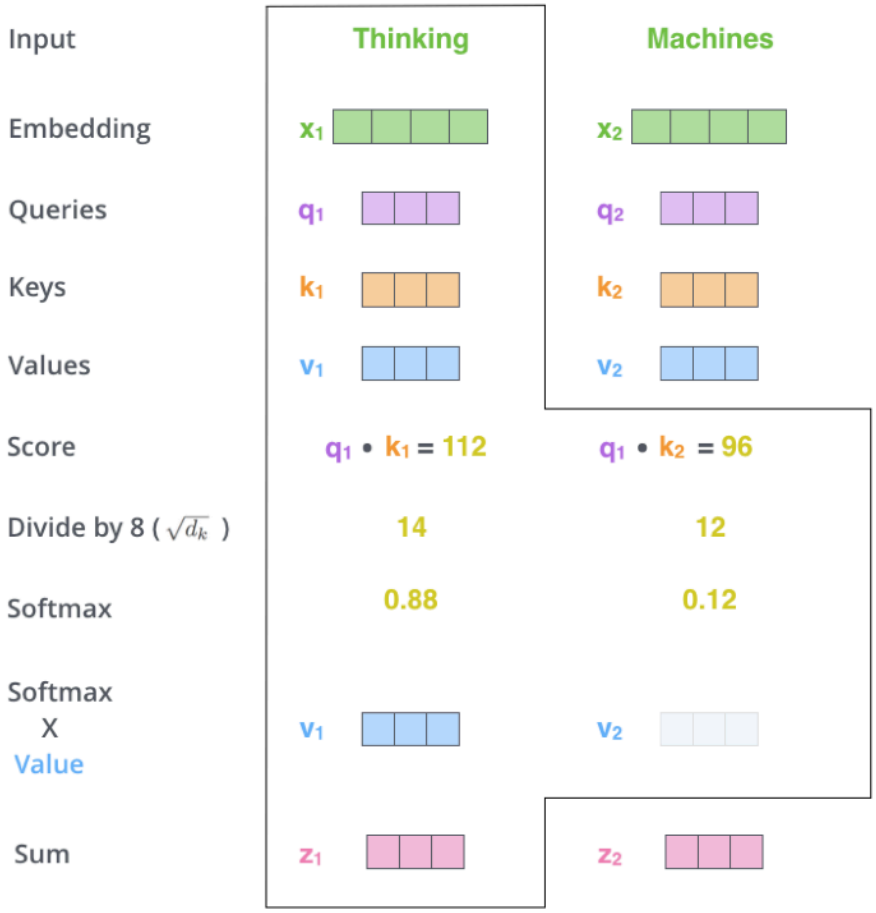


Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

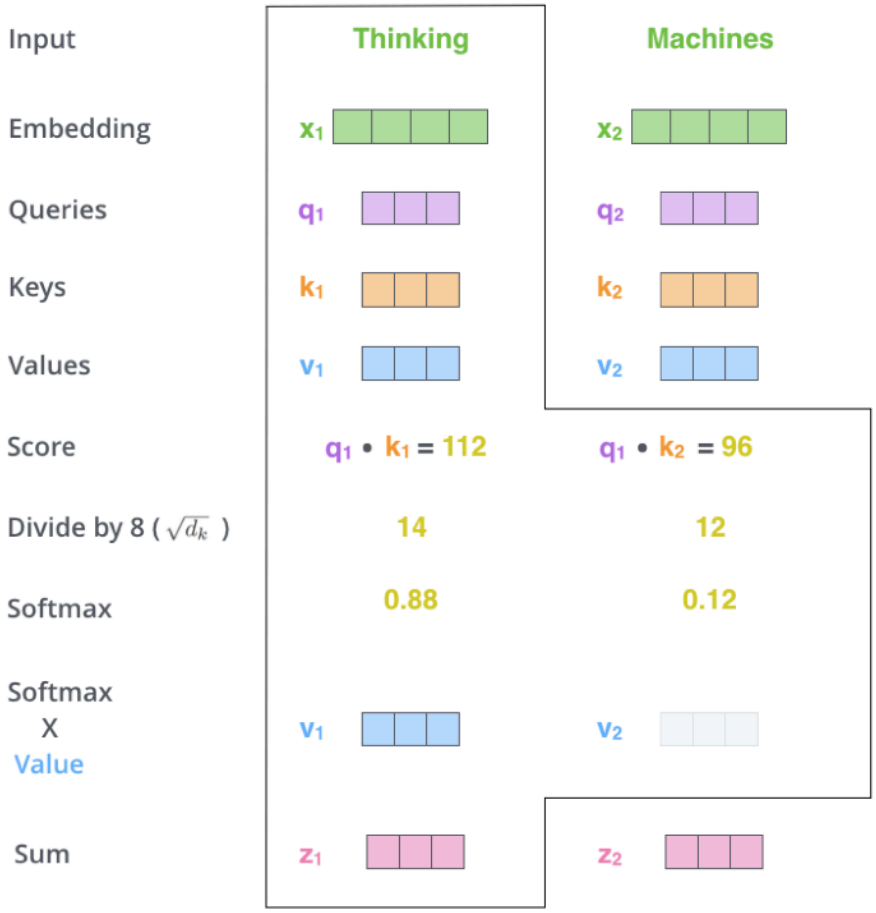
- **Second step : calculate a score**
  - The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.
  - Example : calculate self-attention for the first word "Thinking" ⇒ we score each word of the input sentence against this word
  - Score?
    - dot product of the query vector q_i with the key vector k_i of the respective word we're scoring
    - divide by 8 (the square root of the dimension of the key vectors – 64)
    - pass the result through a softmax operation
      - softmax score determines how much each word will be expressed at this position



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

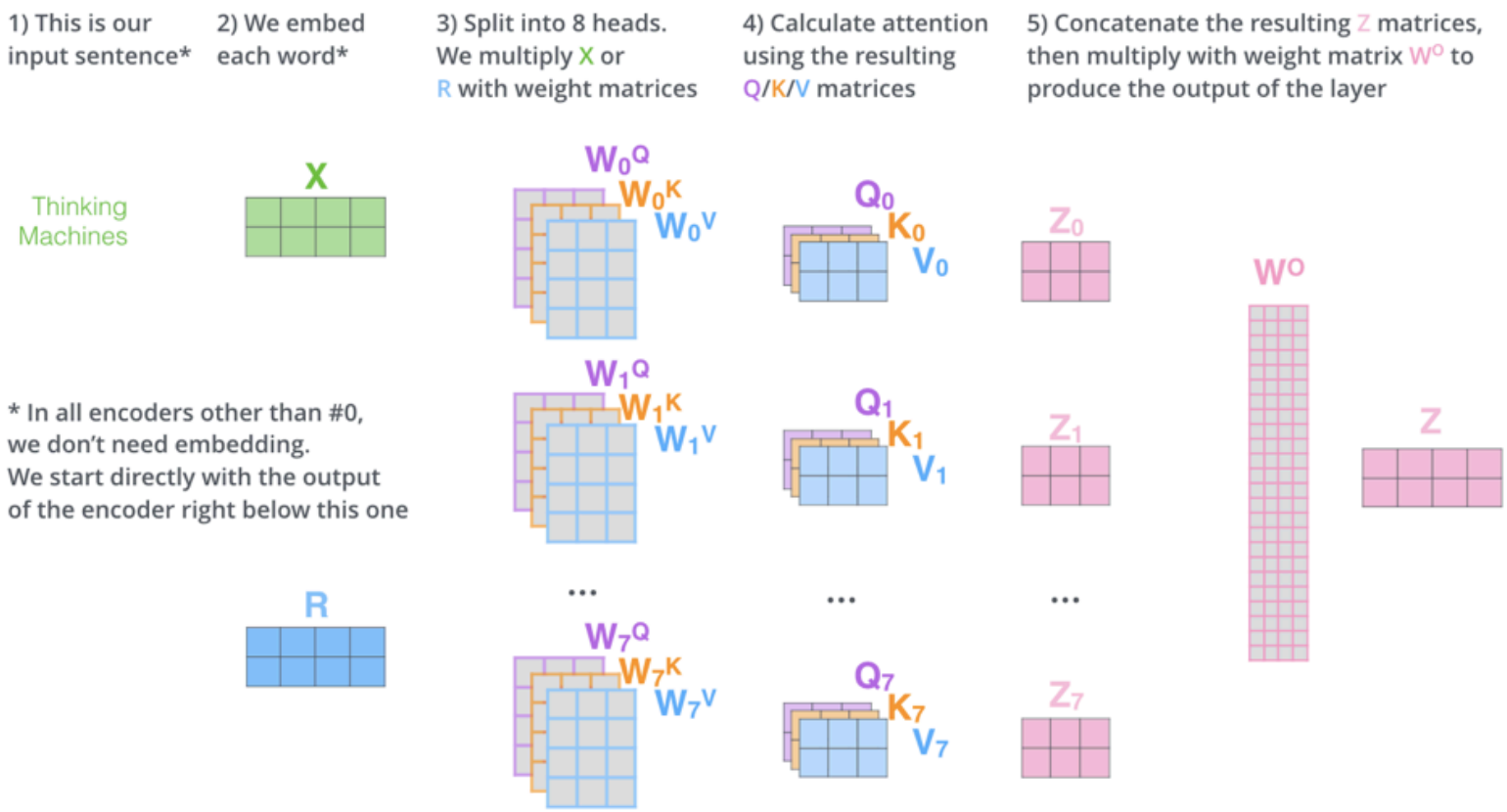
- **Third step :**
  - multiply each **value** v_i vector by the **softmax** score
    - keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words
  - sum up the weighted value vectors
    - produces the output z_i of the self-attention layer at this position (for the first word).



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

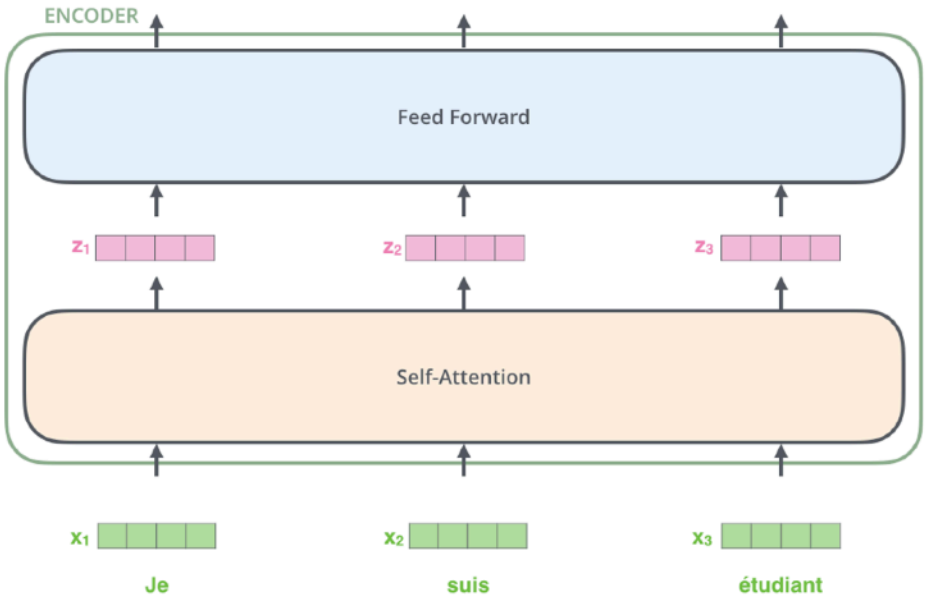
- **In summary**



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

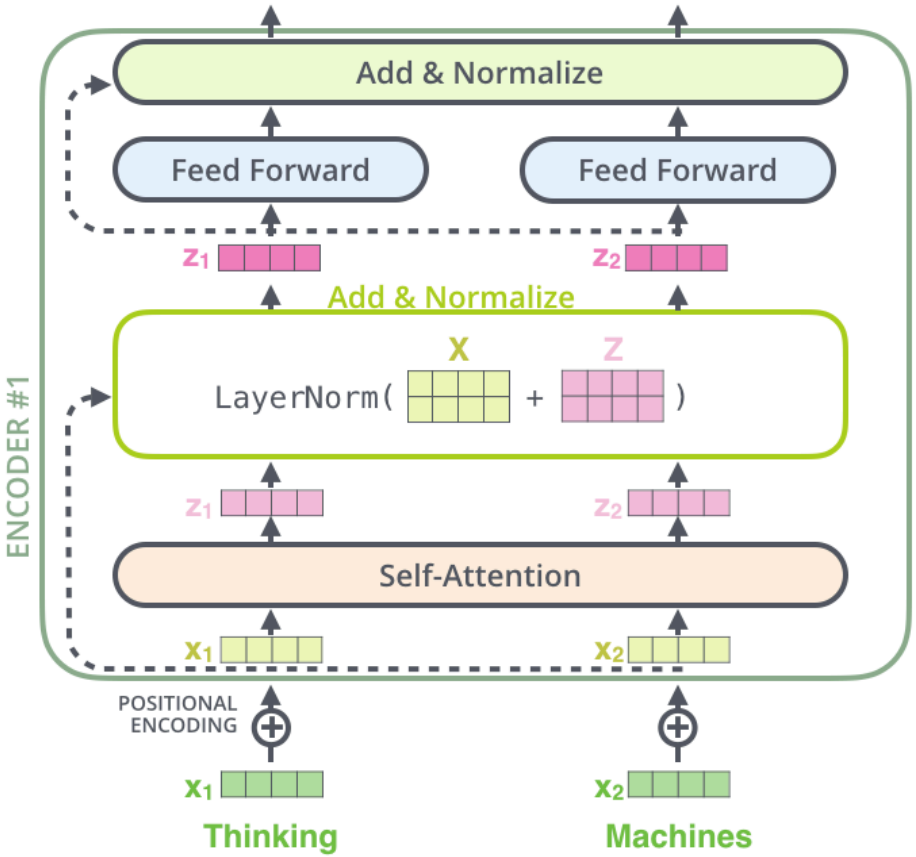
- **Bringing The Tensors Into The Picture**
  - first turn each input word into a vector using an embedding algorithm (only happens in the bottom-most encoder)
  - after embedding the words in our input sequence, each of them flows through each of the two layers of the encoder
  - word in each position flows through its own path in the encoder
  - There are dependencies between these paths in the self-attention layer. The feed-forward layer does not have those dependencies, however, and thus the various paths can be executed in parallel while flowing through the feed-forward layer



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

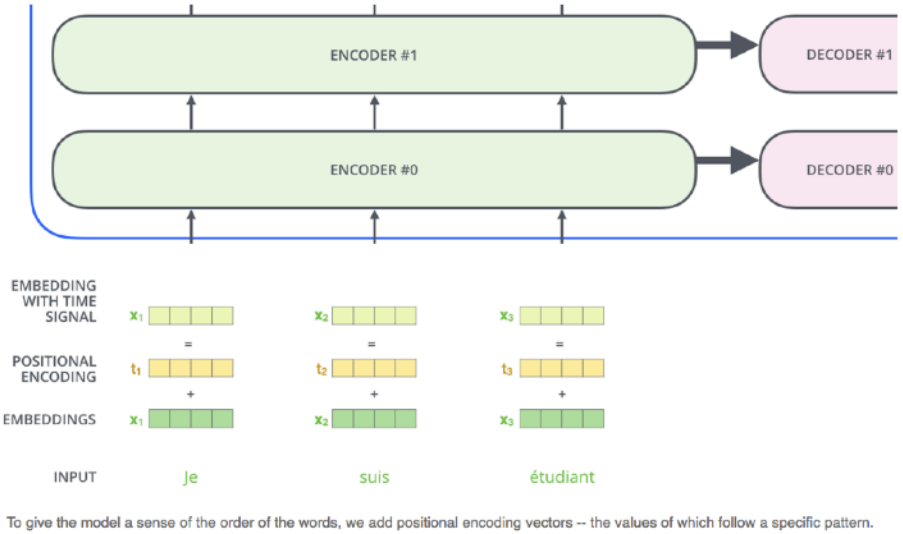
- **Residuals + Normalisation**
  - each sub-layer (self-attention, feed-forward) in each encoder has
    - a residual connection around it
    - is followed by a layer-normalisation step



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

- **Representing The Order of The Sequence Using Positional Encoding**
  - Find a way to account for the order of the words in the input sequence
    - Transformer adds a vector to each input embedding
    - These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence

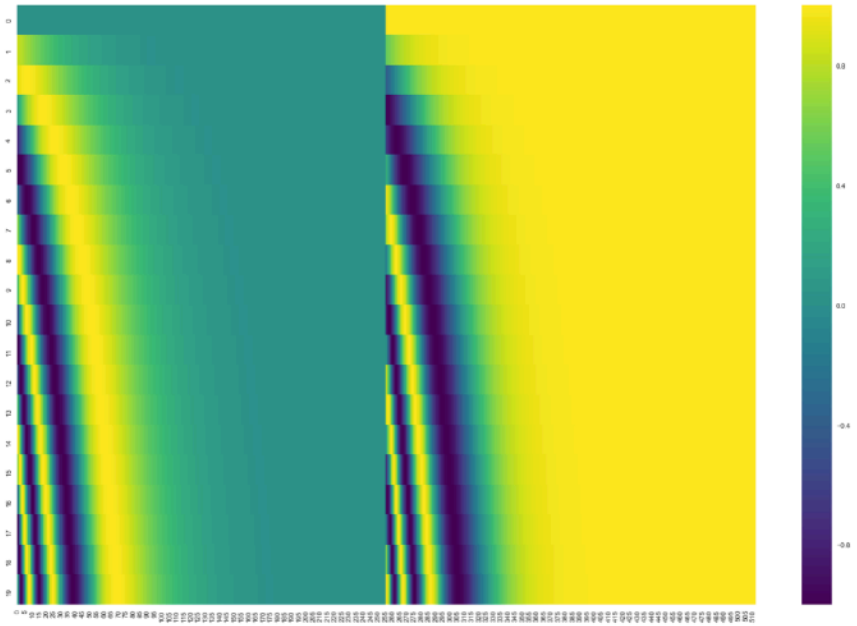


To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.

[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

- **Representing The Order of The Sequence Using Positional Encoding**
  - What might this pattern look like ?

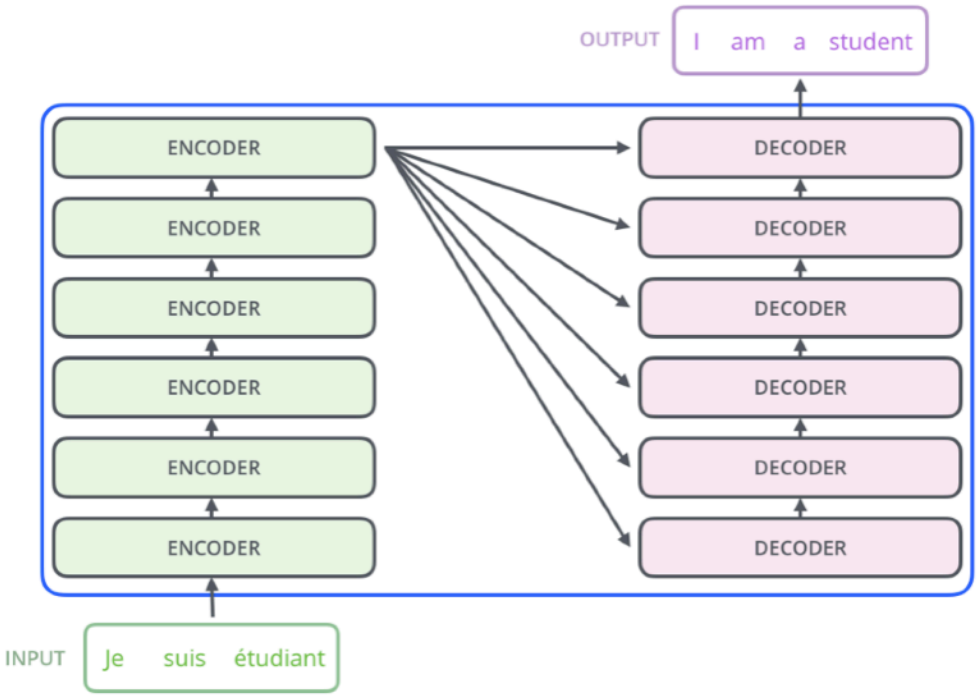


A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.

[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

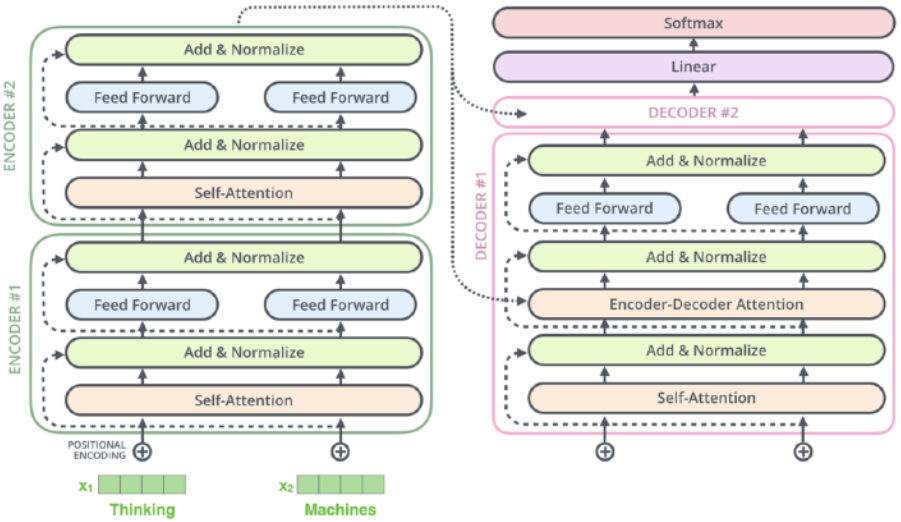
- The encoding component is a stack of 6 encoders on top of each other
- The decoding component is a stack of decoders of the same number



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

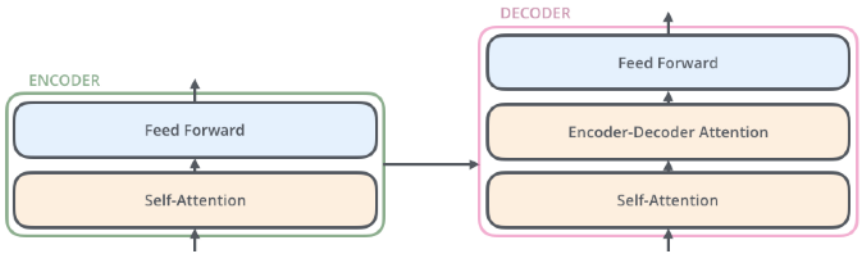
- **Plugin the encoder to the decoder**



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer

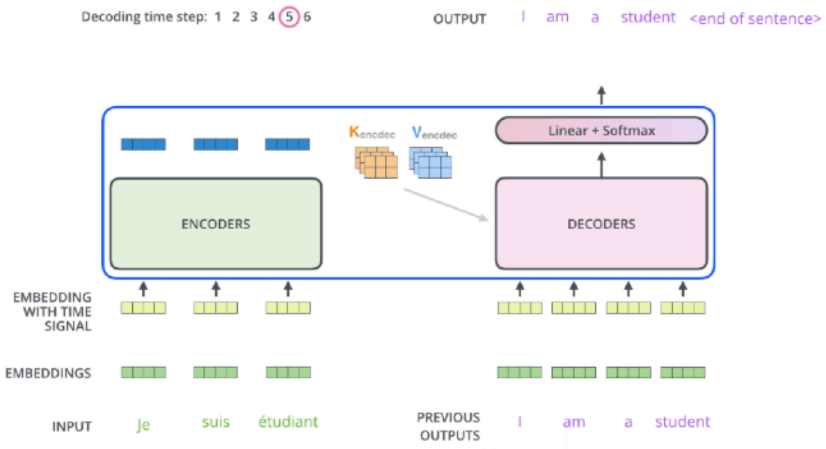
- The decoder has both those layers
  - but between them is an **attention layer** that helps the decoder focus on relevant parts of the input sentence (similar what attention does in Seq2Seq models)
-



[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

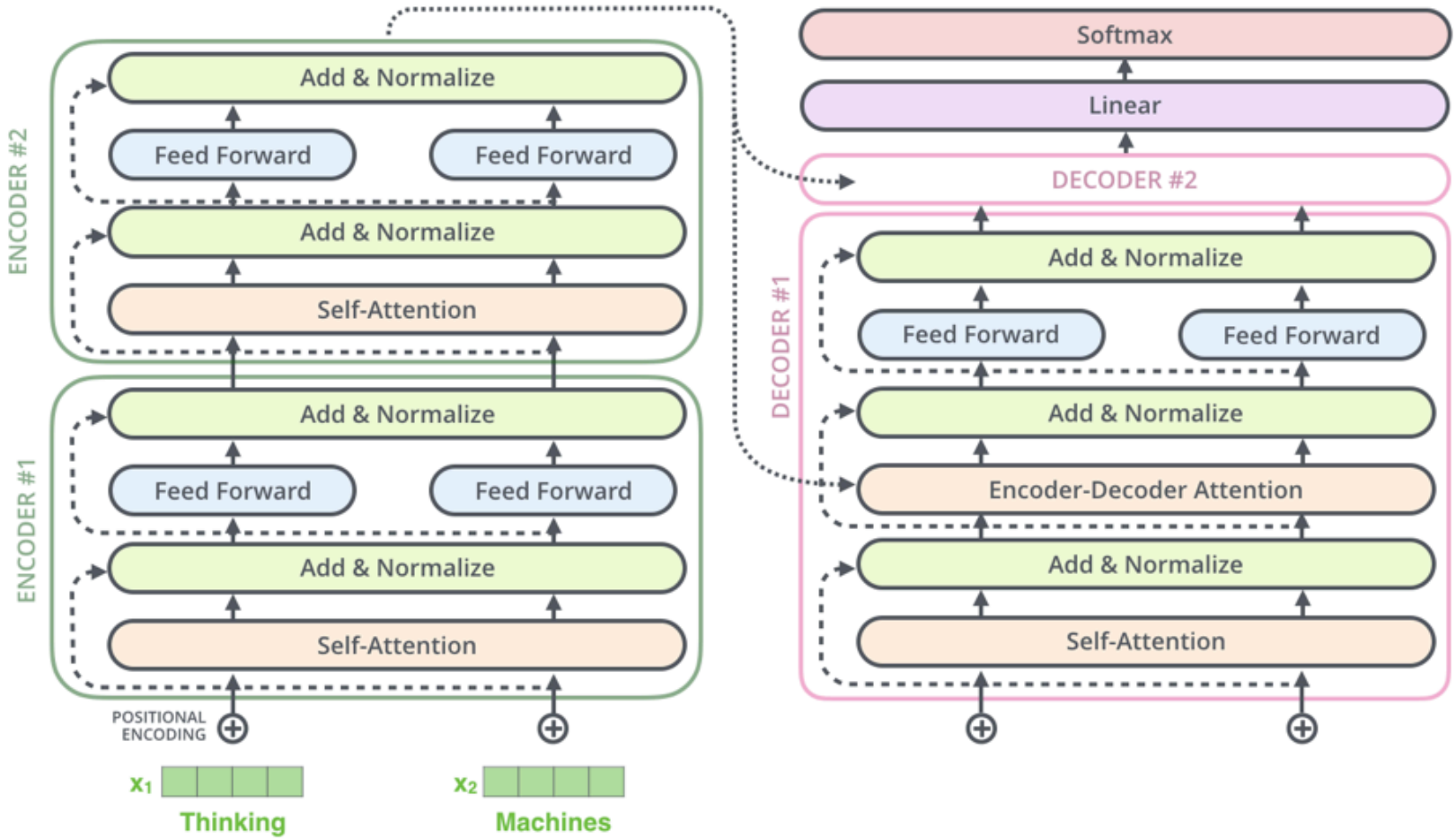
Transformer

- **Decoder side**
  - The output of the top encoder is then transformed into a set of attention vectors K and V
    - to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence
  - The output of the decoder of each step is fed to the bottom in the next step
  - As for the encoder, we embed and add positional encoding to those decoder inputs
  - In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence
  - The "Encoder-Decoder Attention" layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.



- [Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
  [J. Alammar "The Illustrated Transformer"] LINK

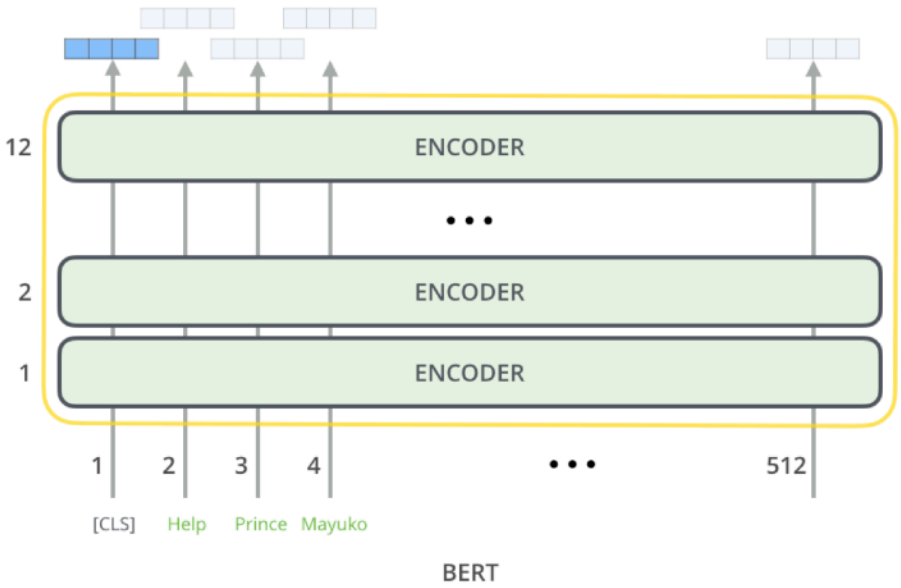
Transformer

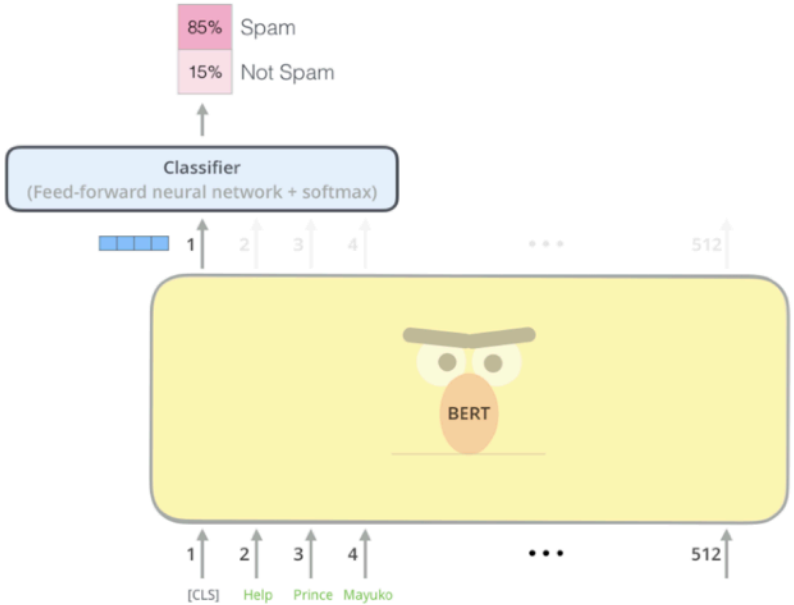


[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

BERT

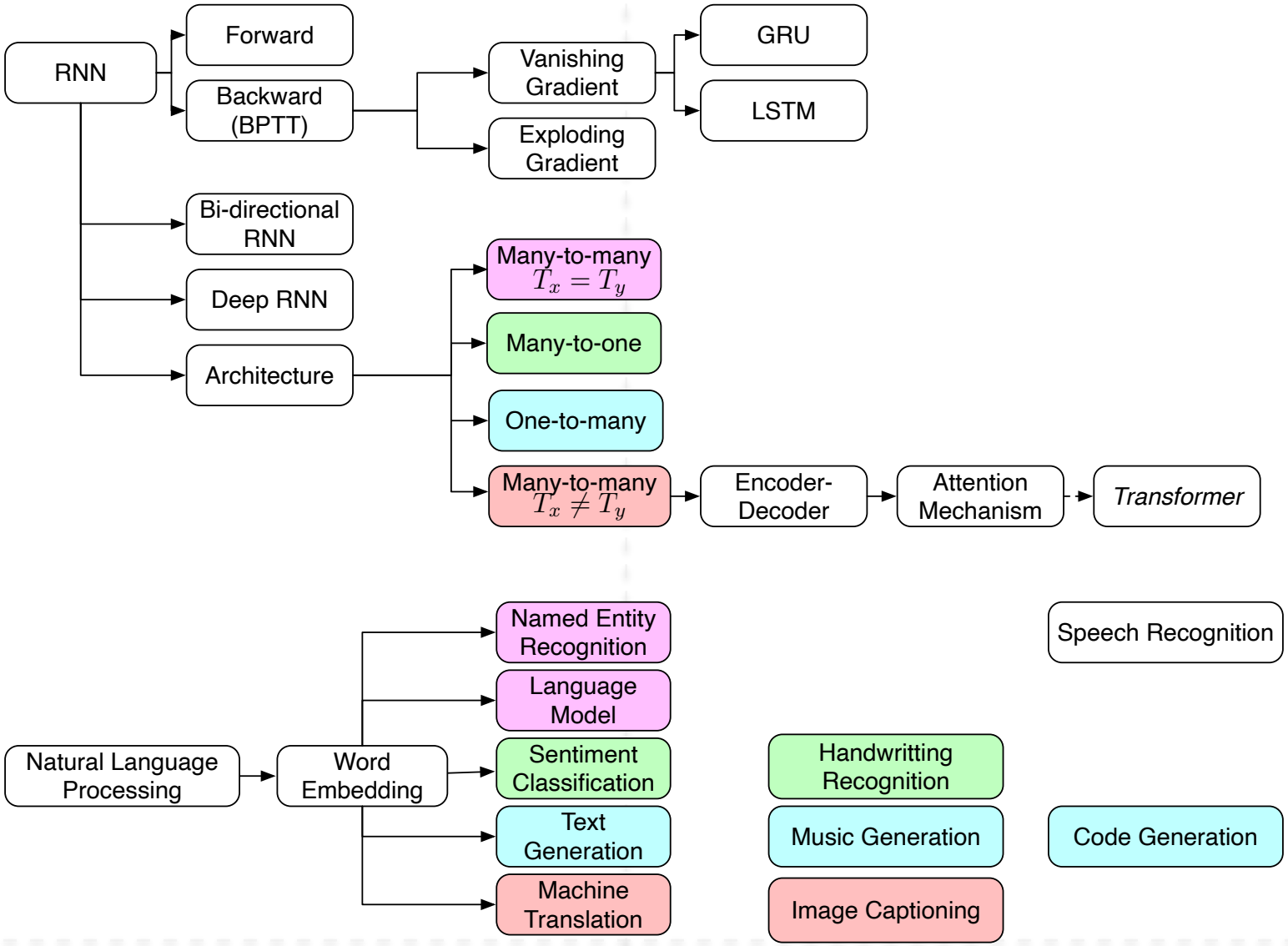
- a trained Transformer Encoder stack
- the first input token is supplied with a special [CLS] token. CLS here stands for Classification





[J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2019] LINK
[J. Alammar "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)"] LINK

Overview


RNN
Forward
Backward (BPTT)
Vanishing Gradient
Exploding Gradient
GRU
LSTM
Bi-directional RNN
Deep RNN
Architecture
Many-to-many $T_x = T_y$
Many-to-one
One-to-many
Many-to-many $T_x \neq T_y$
Encoder-Decoder
Attention Mechanism
Transformer
Natural Language Processing
Word Embedding
Named Entity Recognition
Language Model
Sentiment Classification
Text Generation
Machine Translation
Speech Recognition
Handwritting Recognition
Music Generation
Image Captioning
Code Generation