Deep Learning

Recurrent Neural Network

Geoffroy Peeters

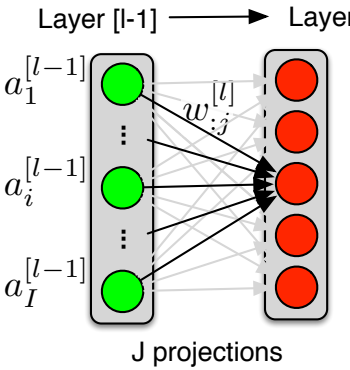
contact: geoffroy.peeters@telecom-paris.fr
Télécom-Paris, IP-Paris, France

Roadmap

Architectures: *distinguish the way neurons are inter-connected*

Lecture 1

Fully Connected Neural Network

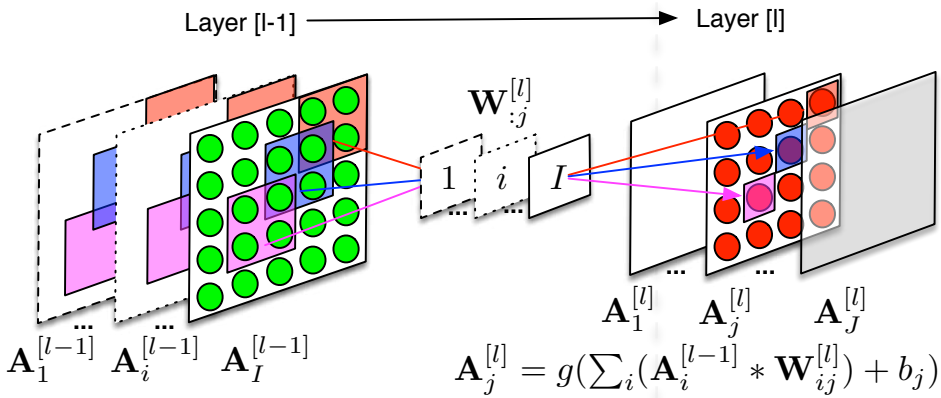


$$a_j^{[l]} = g(\sum_i a_i^{[l-1]} w_{ij}^{[l]} + b_j^{[l]})$$

$$\mathbf{a}^{[l]} = g(\mathbf{a}^{[l-1]}\mathbf{W}^{[l]} + \mathbf{b}^{[l]})$$

Lecture 2

Convolutional Neural Network



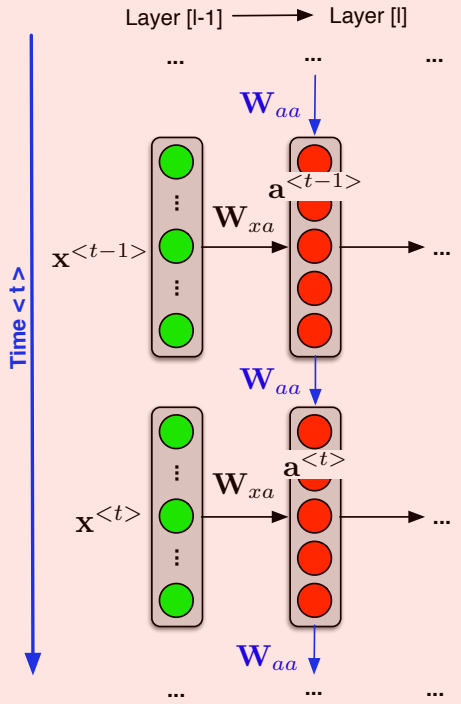
$$\mathbf{A}_j^{[l]} = g(\sum_i (\mathbf{A}_i^{[l-1]} * \mathbf{W}_{ij}^{[l]}) + b_j)$$

I feature maps → J filters or kernels → J feature maps

Lecture 3

Recurrent Neural Network

$$\mathbf{a}^{<t>} = g(\mathbf{x}^{<t>}\mathbf{W}_{xa} + \mathbf{a}^{<t-1>}\mathbf{W}_{aa} + \mathbf{b}_a)$$

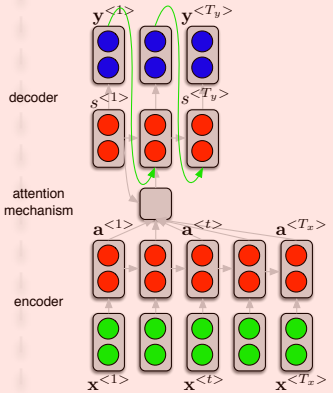


Roadmap

Meta-architectures: *the way architectures are plug to form a system*

Lecture 3

Encoder-Decoder with Attention



Transformer

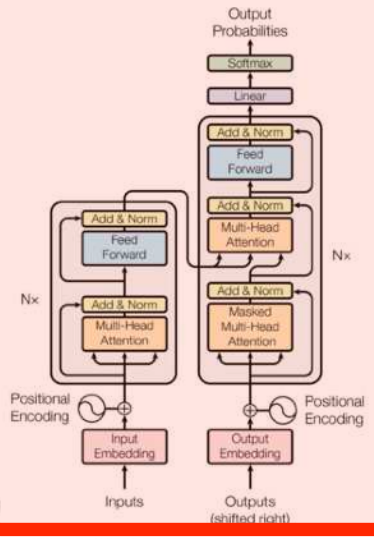
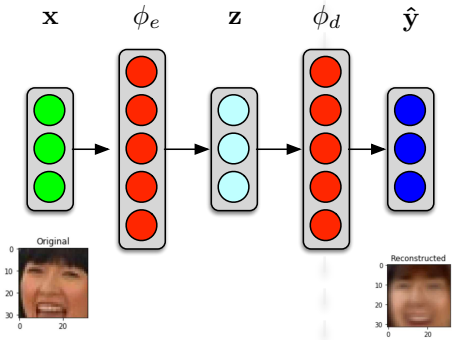


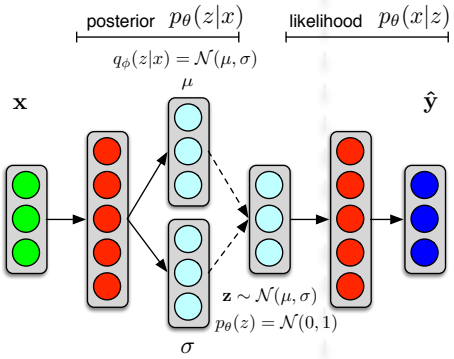
Figure 1: The Transformer - model architecture.

Lecture 4

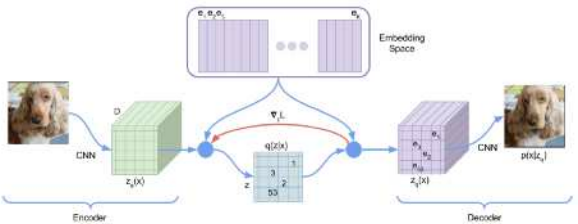
Auto-Encoder



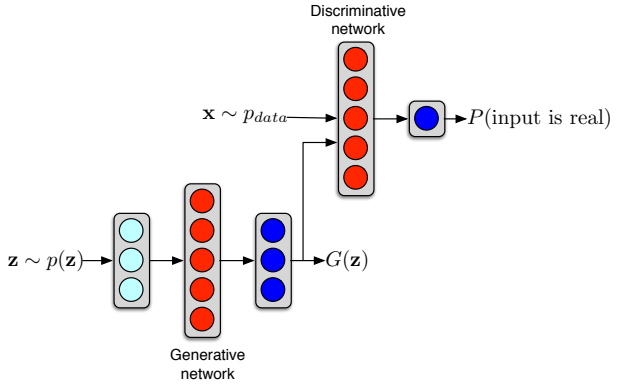
Variational-Auto-Encoder



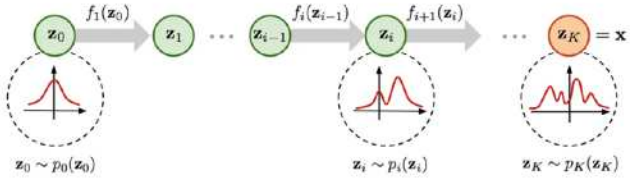
VQ-VAE



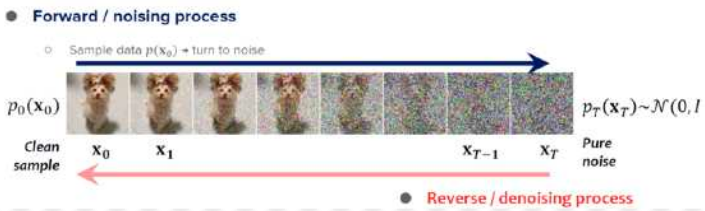
Generative Adversarial Network



Normalizing FLows

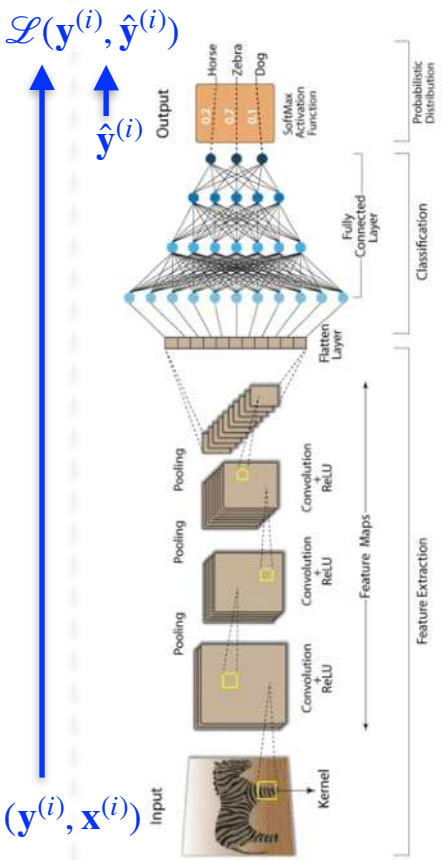


Denoising Diffusion Models



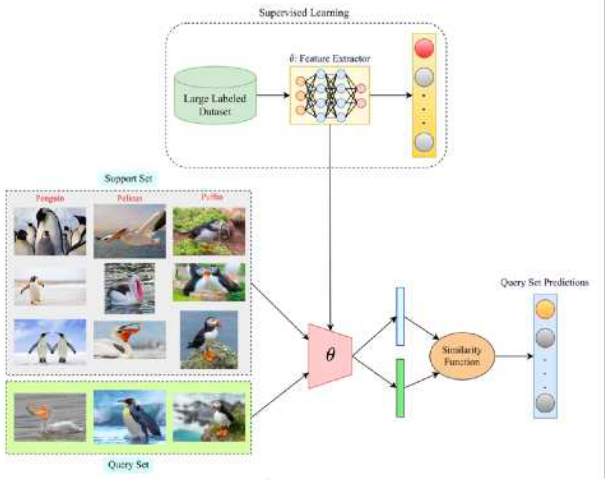
Roadmap
Training-paradigms: *the paradigm we use to train a system*

Supervised learning

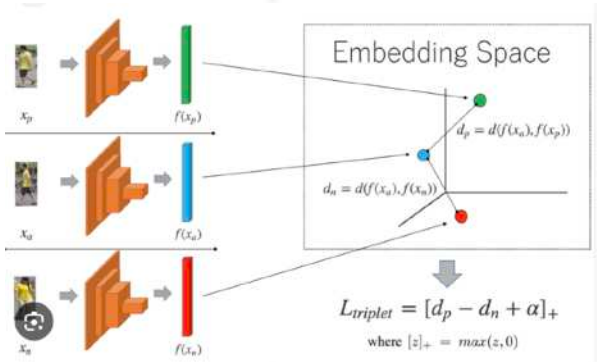


from https://larbifarihi.medium.com/classifying-cats-and-dogs-using-convolutional-neural-networks-cnns-ce6bc7bad64d

Zero/few Shot learning

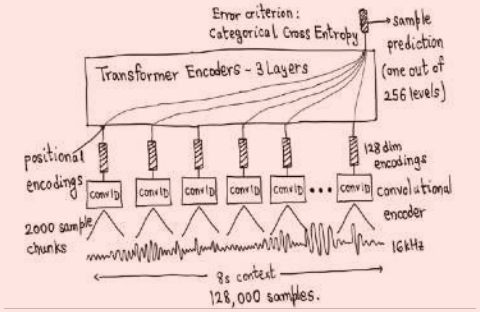


Metric learning

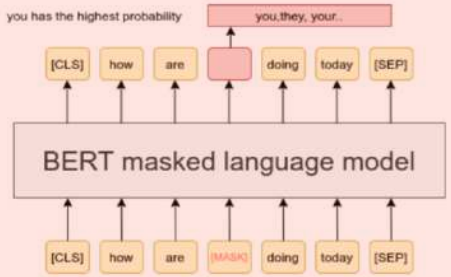


Self Supervised learning

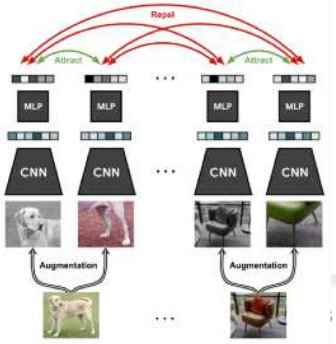
Auto-regressive (WaveNet, ChatGPT-



Masked LLM (BERT)

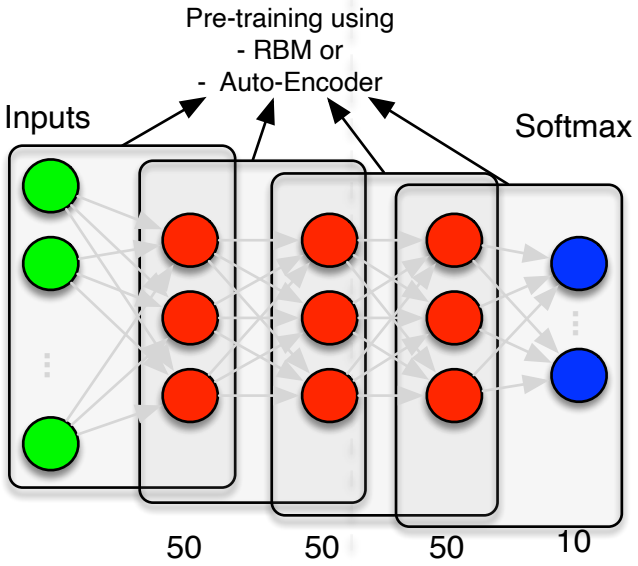


SimCLR



Multi Layers Perceptron (MLP), Fully-Connected (FC), Feed-Forward

Multi Layers Perceptron (Fully Connected)

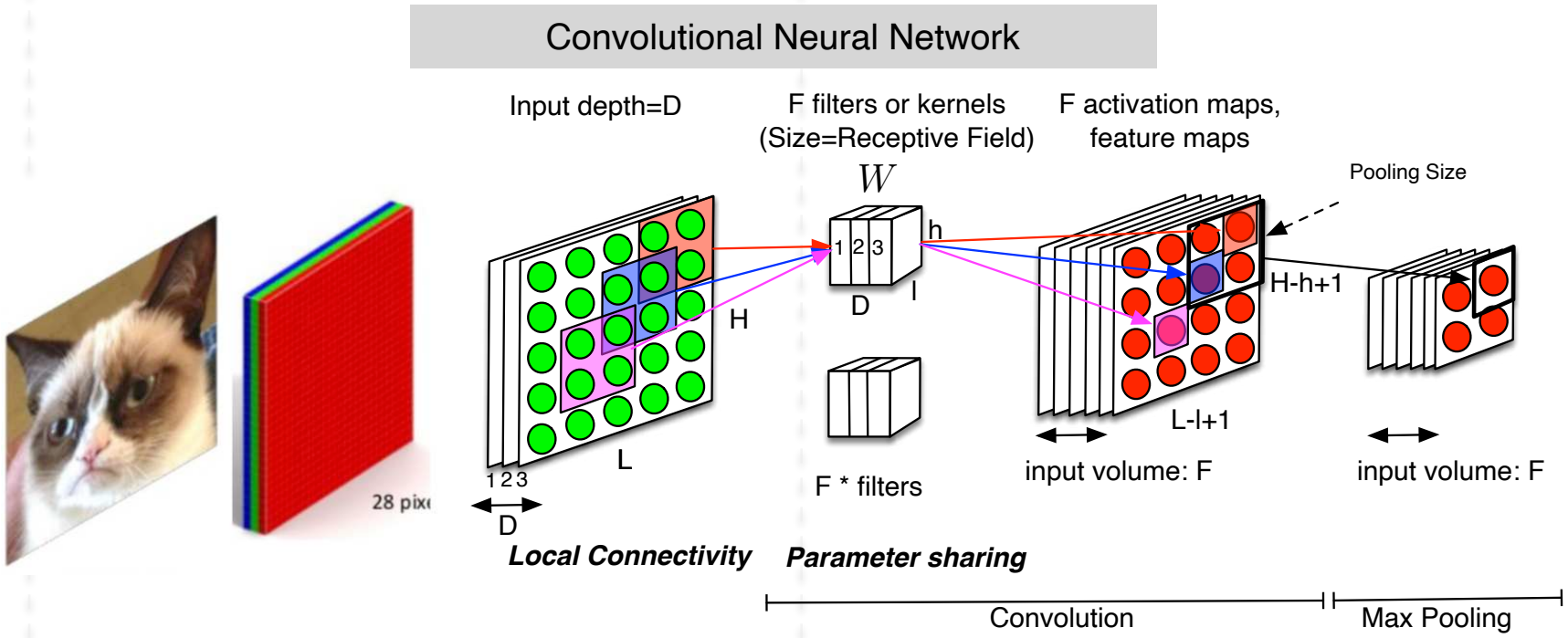


$$\underline{z}^{[l]} = \underline{\underline{W}}^{[l]}\underline{a}^{[l-1]} + \underline{b}^{[l]}$$

$$\underline{a}^{[l]} = g^{[l]}(\underline{z}^{[l]})$$

Three main types of Nets

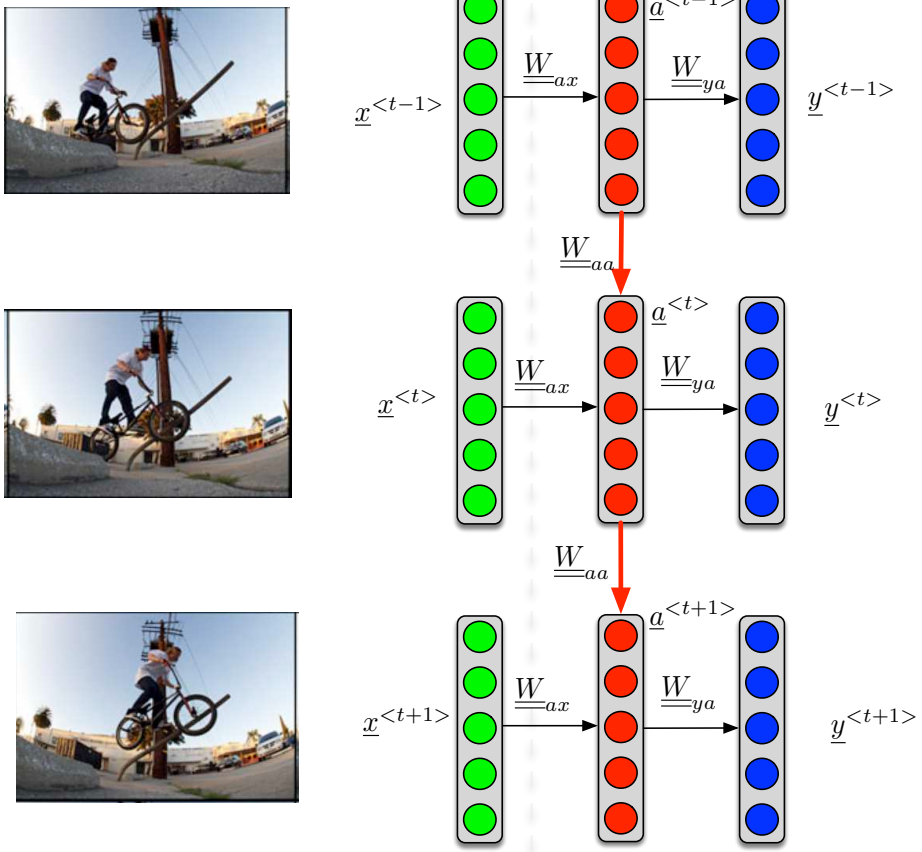
Convolutional Neural Networks (CNN)



Three main types of Nets

Recurrent Neural Networks (RNN)

Recurrent Neural Network

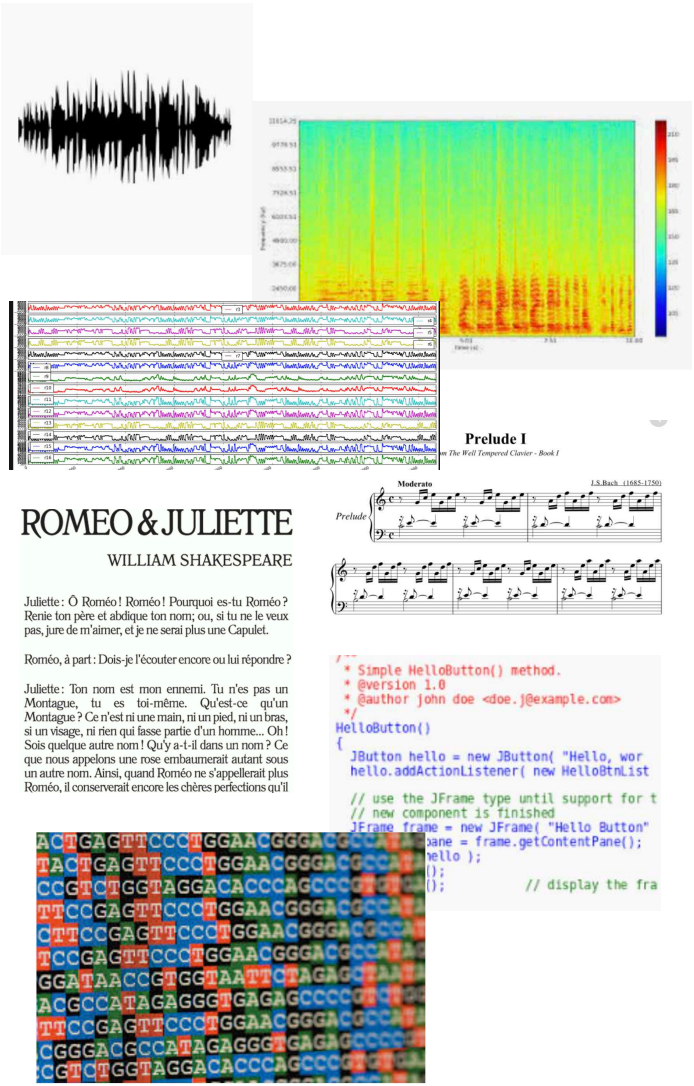


Recurrent Neural Network for Sequential Data

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)
  - ...



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

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

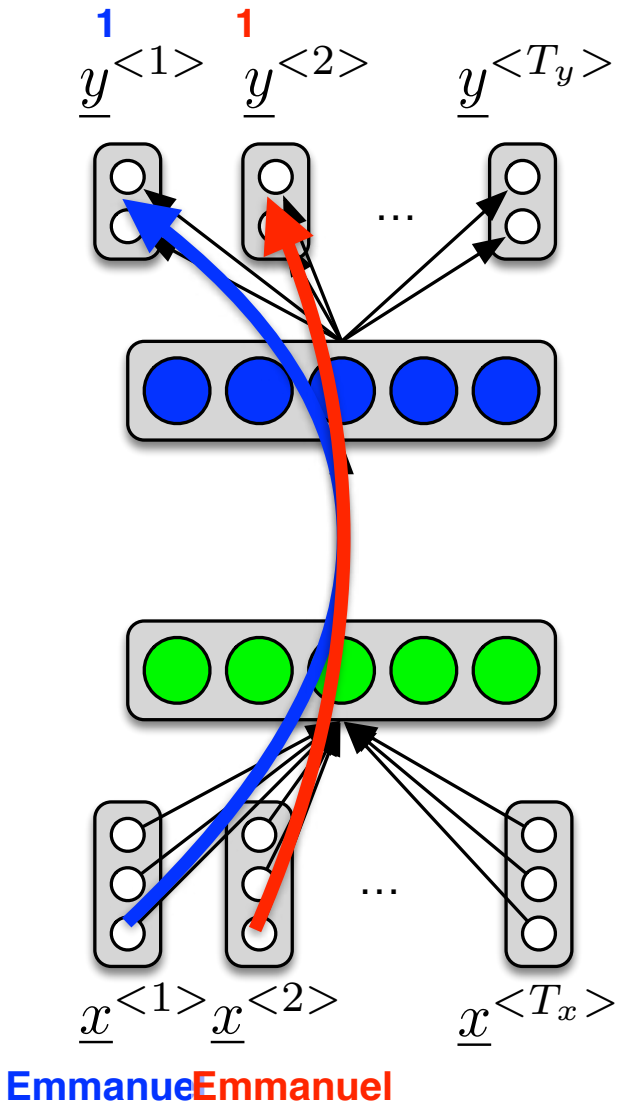
{x}	$x^{<1>}$	$x^{<2>}$	…				$x^{<7>}$
	Emmanuel	Macron	is	the	president	of	France
{y}	$y^{<1>}$	$y^{<2>}$	…				$y^{<7>}$
	1	1	0	0	0	0	1

- Different types of **output** $\mathbf{y}^{\langle t \rangle}$:
  - Continuous values $\mathbf{y}^{\langle t \rangle} \in \mathbb{R}^{n_y}$ (Activation: linear, Loss: MSE)
  - Binary-classes $y^{\langle t \rangle} \in \{0,1\}$ (Activation: sigmoid, Loss: Binary-Cross-Entropy)
  - Multi-classes $y^{\langle t \rangle} \in \{1 \ldots K\} \Rightarrow \mathbf{y}^{\langle t \rangle} \in \{0,1\}^K$ (Activation: softmax, Loss: Cross-Entropy)

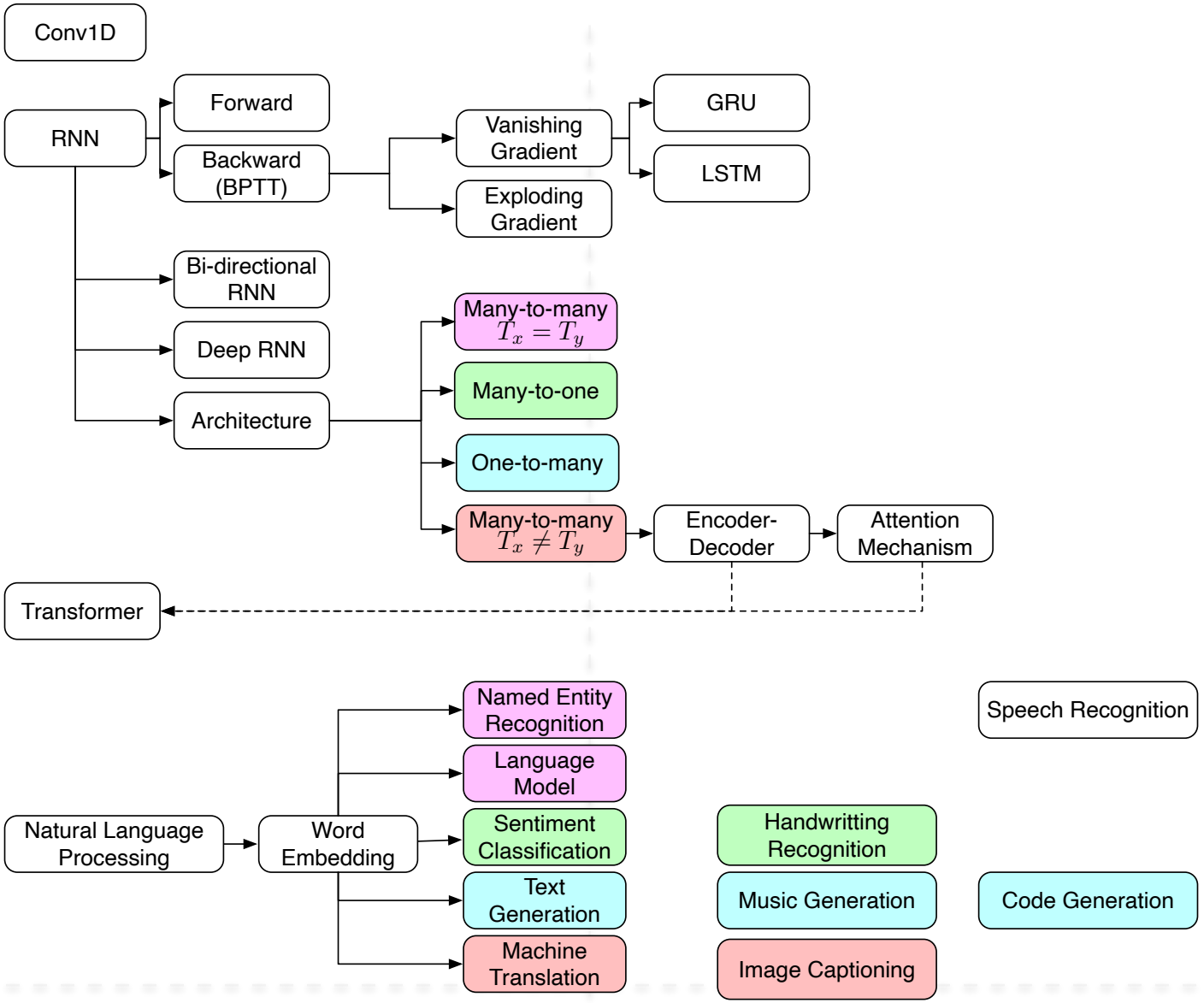
- Different types of **input** $\mathbf{x}^{\langle t \rangle}$
  - Continuous values $\mathbf{x}^{\langle t \rangle} \in \mathbb{R}^{n_x}$
  - Categorical values $x^{\langle t \rangle} \in \{1 \ldots K\} \Rightarrow$ **how do we process categorical values as input ?**
    - One-hot-encoding: $\mathbf{x}^{\langle t \rangle} \in \{0,1\}^K$
    - Embedding: $\mathbf{x}^{\langle t \rangle} \in \mathbb{R}^d$

Recurrent Neural Network for Sequential Data

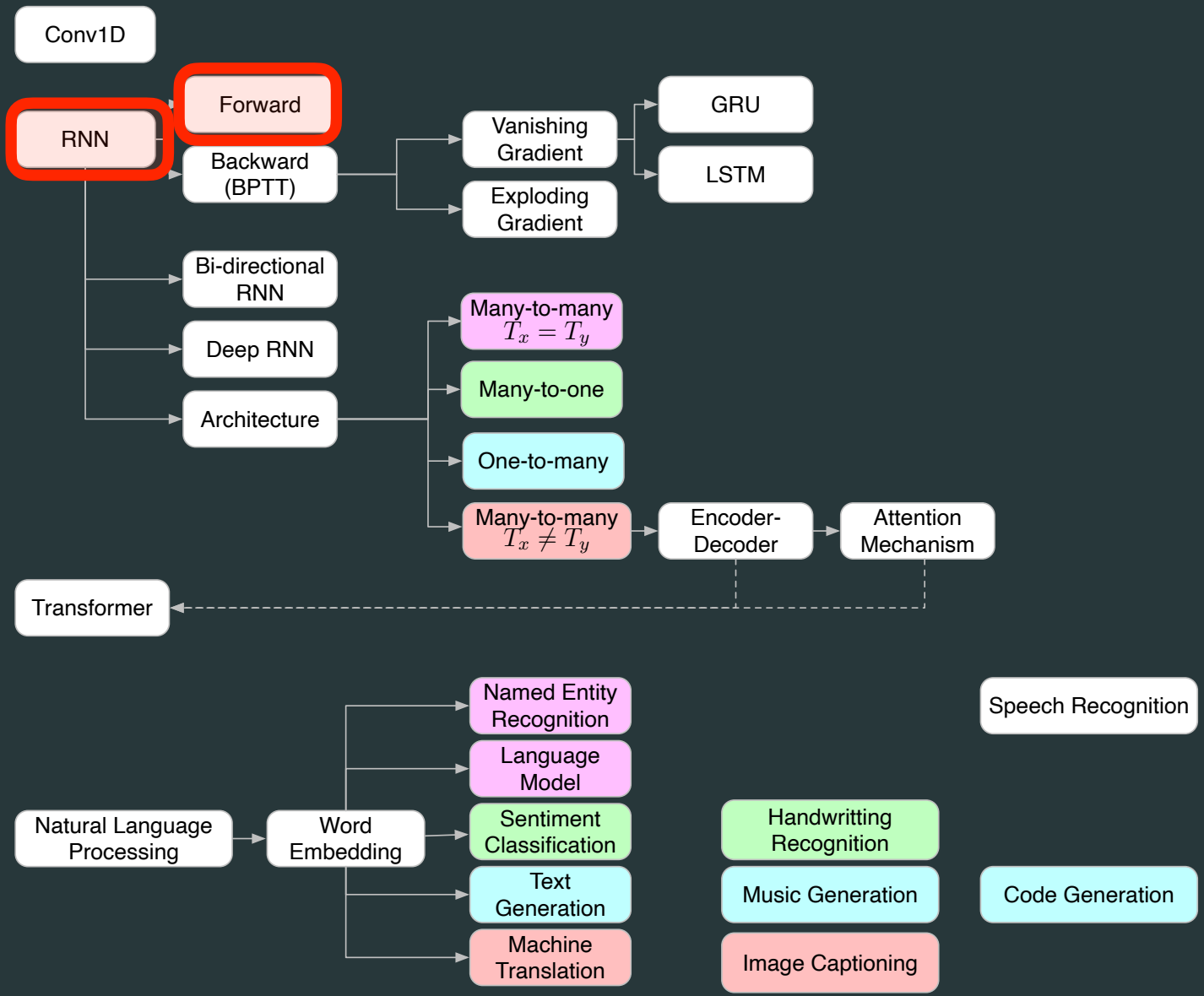
- **Why not using a Multi-Layer-Perceptron ?**
  - We could indeed represent an input sequence $\{\mathbf{x}^{\langle 1 \rangle}, \ldots, \mathbf{x}^{\langle T_x \rangle}\}$ as a large-vector of dimension $(n^{[0]} T_x)$
    - it would requires a huge input dimension !
- Problem:
  - $T_x^{(i)}$ can be different for each i,
    $\Rightarrow$ the dimension $(n^{[0]} T_x^{(i)})$ would be different for each i
  - Solution:
    - zero-padding ? $(n^{[0]} \max_i T_x^{(i)})$
    - it would still requires a huge input dimension !
- Problem:
  - 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 !
  - Solution:
    - Recurrent Neural Network
    - 1D ConvNet (convolution only over time)



Overview

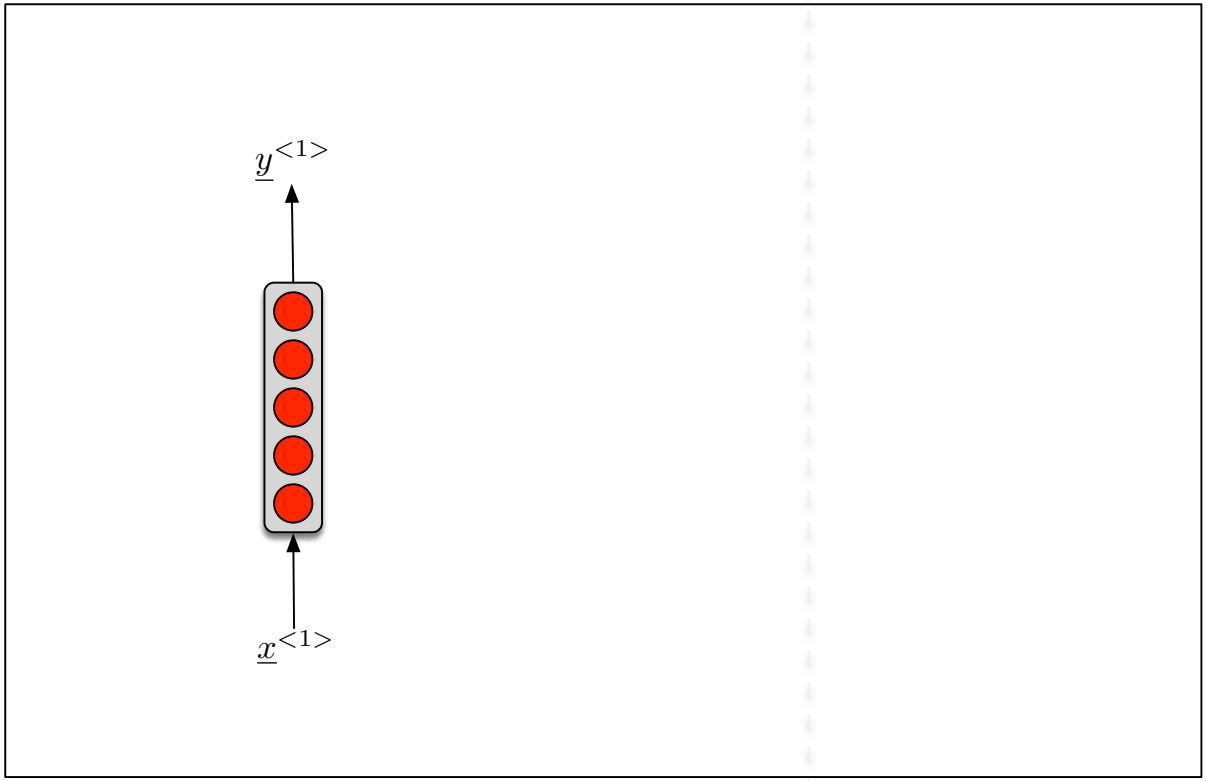

Conv1D
RNN
Forward
Backward (BPTT)
Vanishing Gradient
Exploding Gradient
GRU
LSTM
Bi-directional RNN
Deep RNN
Architecture
Many-to-many $T_x = T_y$
Many-to-one
One-to-many
Many-to-many $T_x \neq T_y$
Encoder-Decoder
Attention Mechanism
Transformer
Natural Language Processing
Word Embedding
Named Entity 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



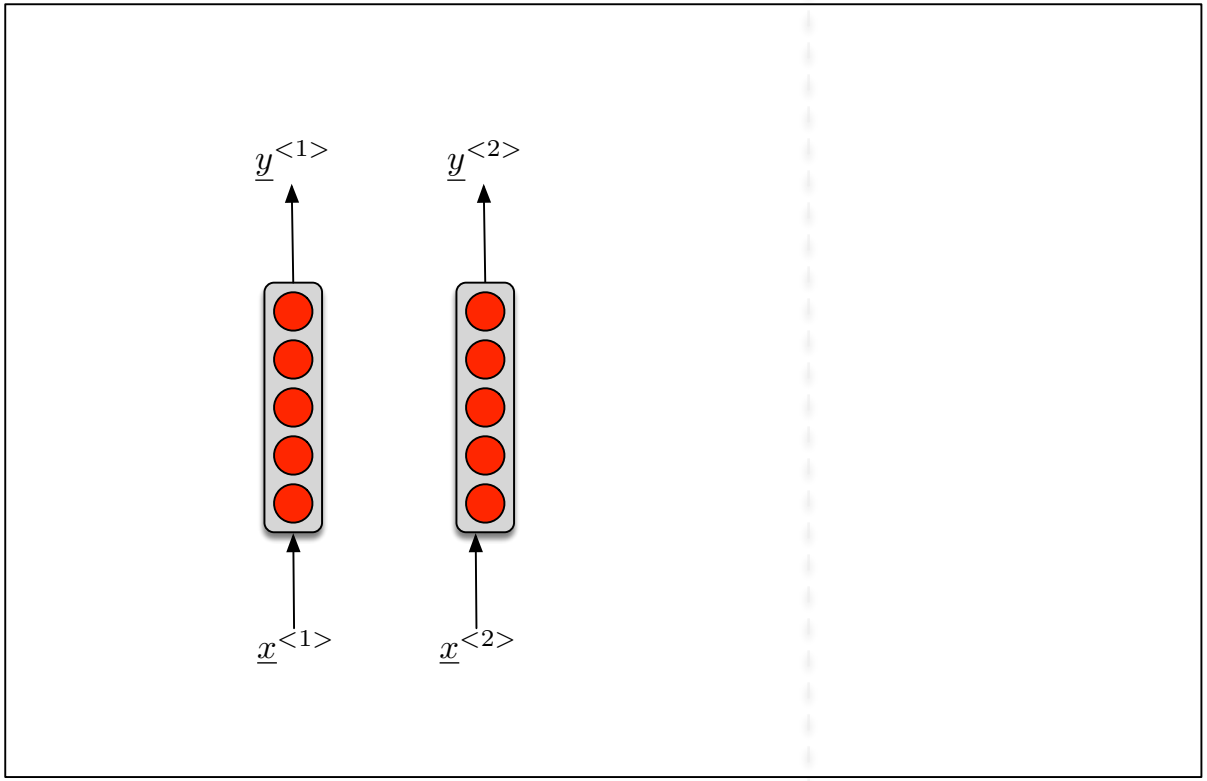
Recurrent Neural Network for Sequential Data

What is an RNN ?



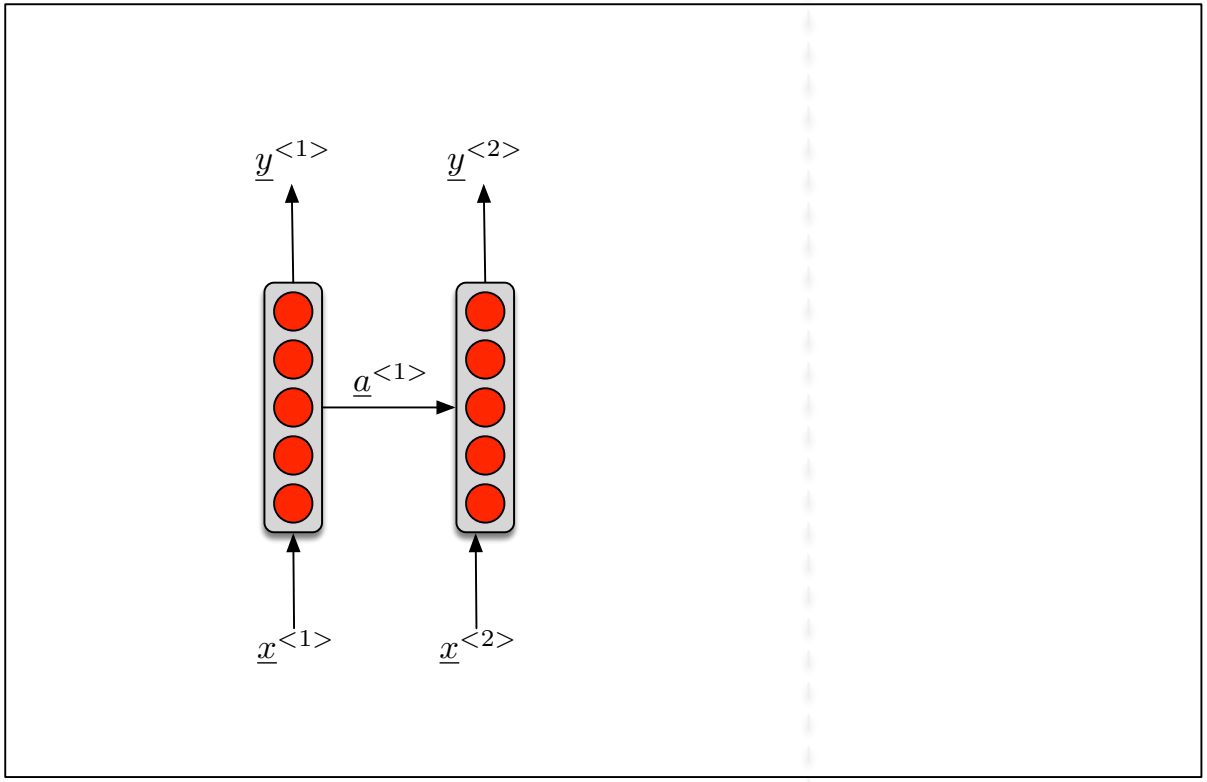
Recurrent Neural Network for Sequential Data

What is an RNN ?



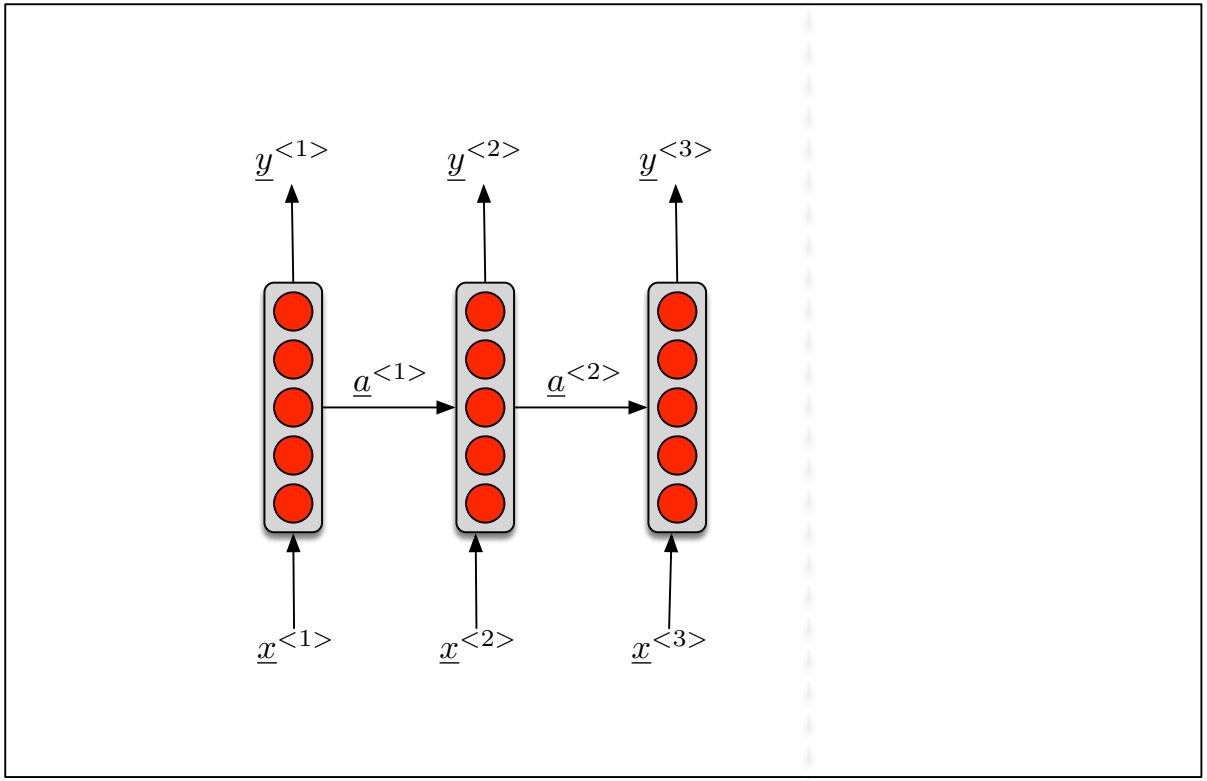
Recurrent Neural Network for Sequential Data

What is an RNN ?



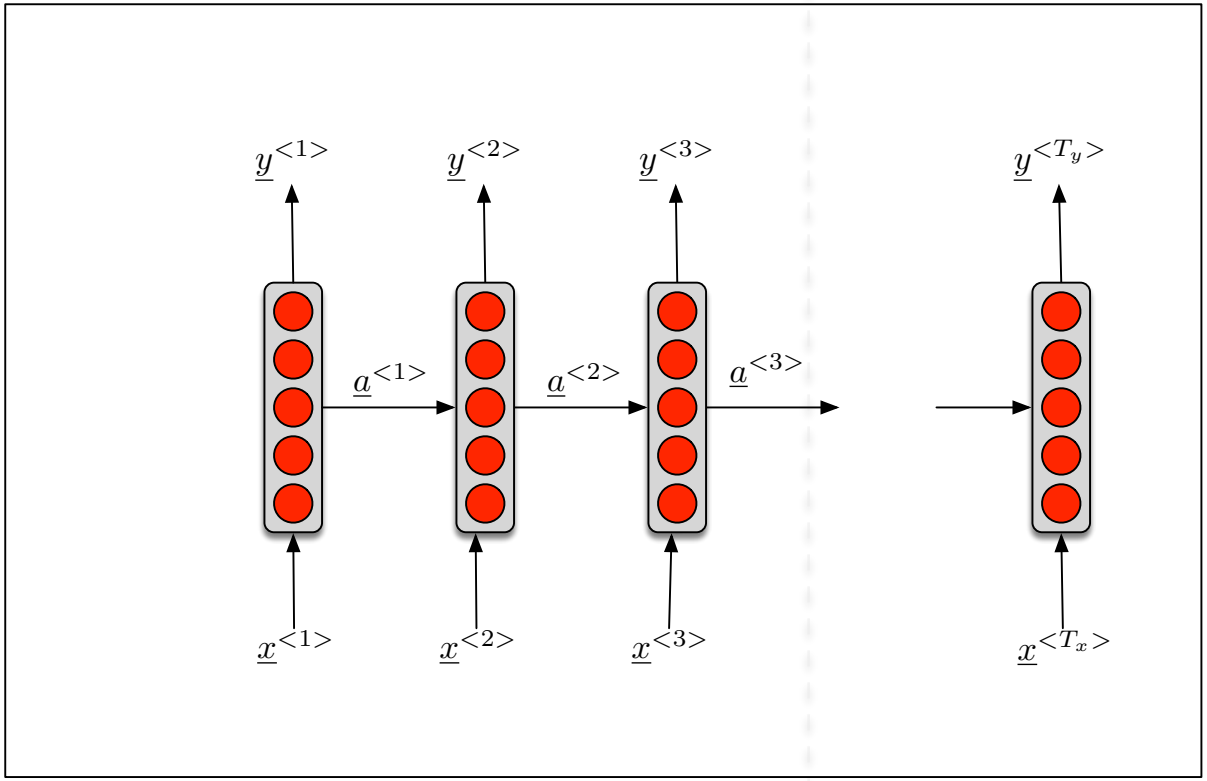
Recurrent Neural Network for Sequential Data

What is an RNN ?



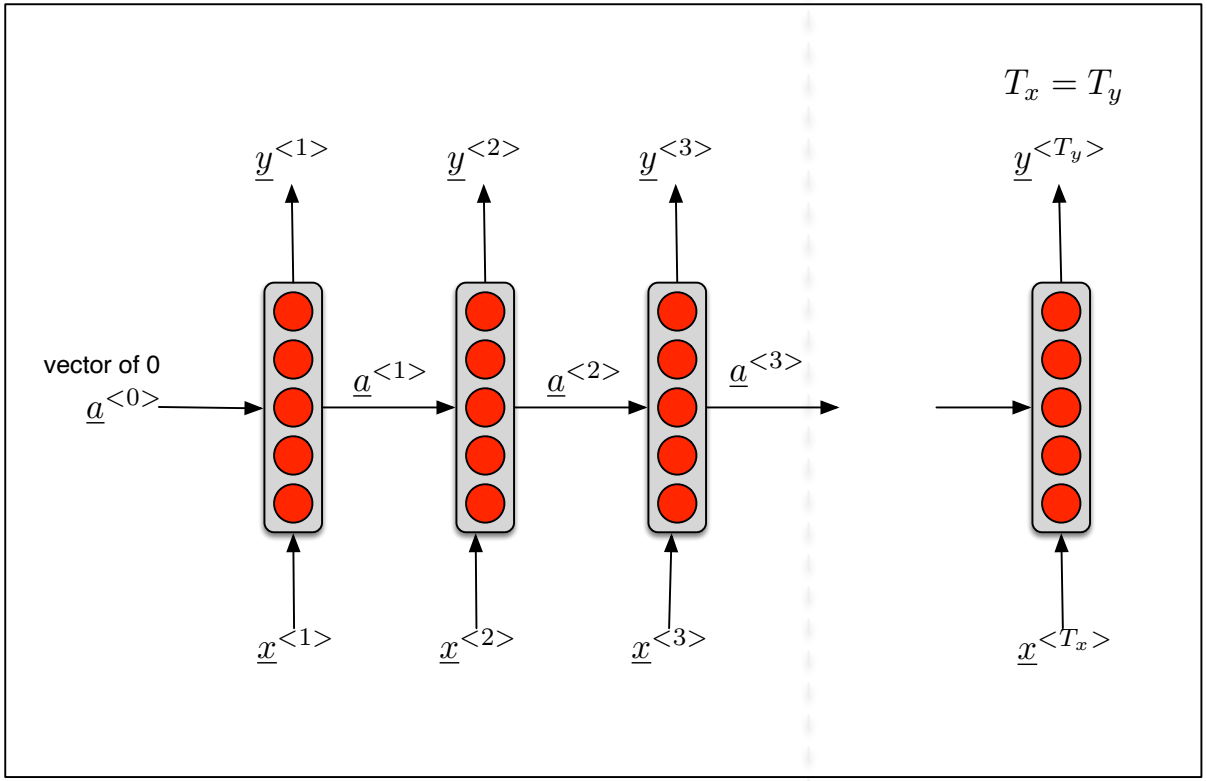
Recurrent Neural Network for Sequential Data

What is an RNN ?



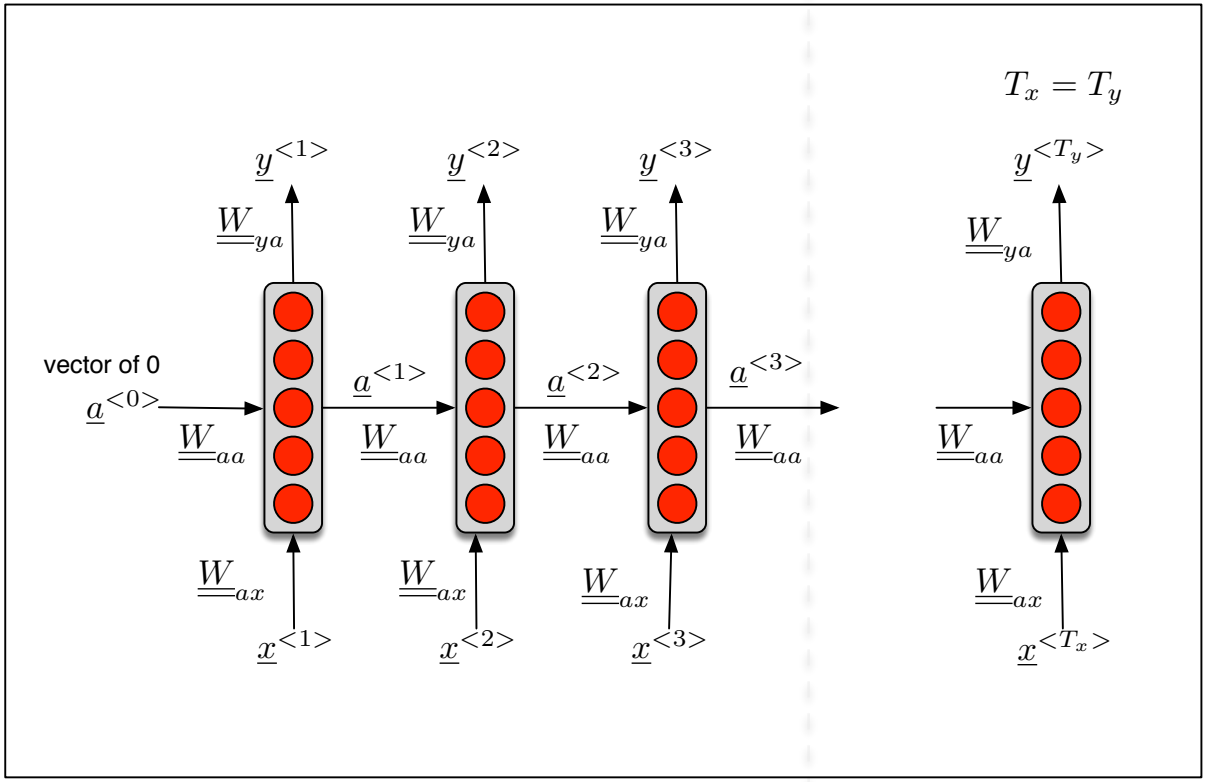
– At each time step $\langle t \rangle$,
  - the RNN gets the activation $a^{\langle t-1 \rangle}$ from the previous time $\langle t-1 \rangle$

What is an RNN ?



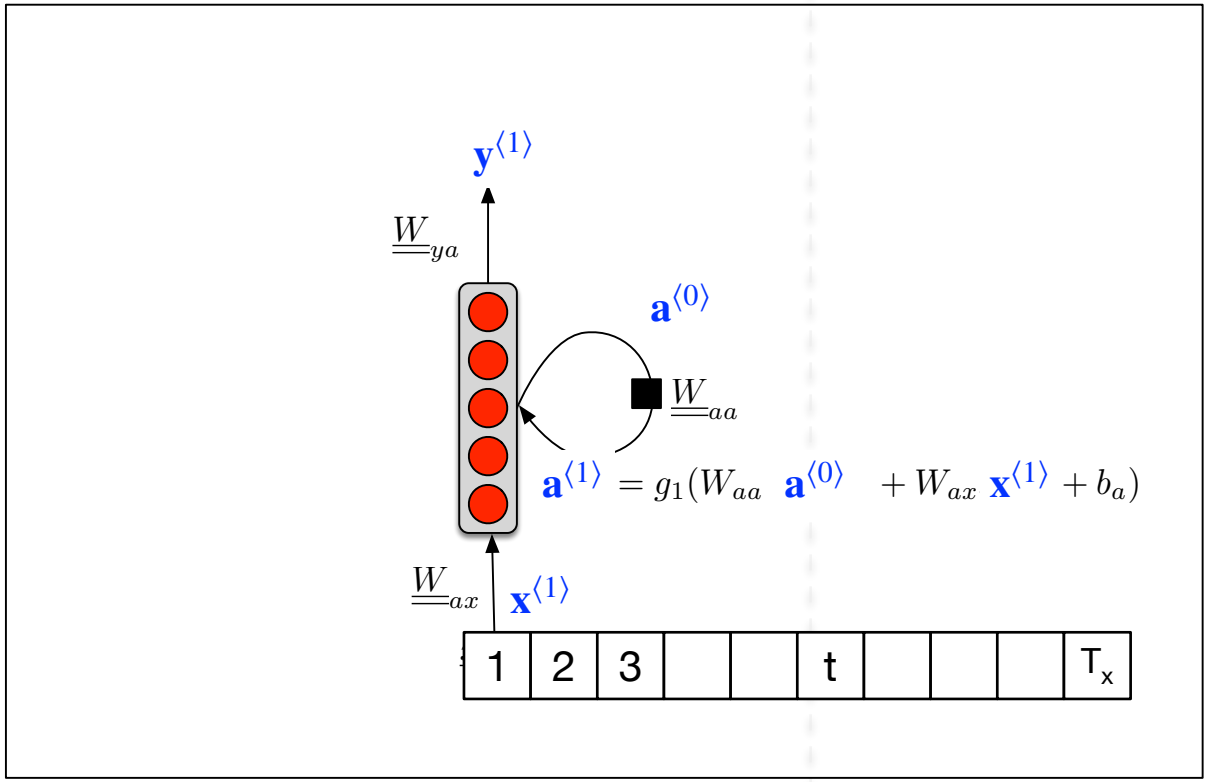
- First activation ?
  - Initialise as $a^{\langle 0 \rangle} = 0$

What is an RNN ?



- Parameters (shared from all time-steps):
  - $\mathbf{W}_{ax}$ from $\mathbf{x}^{\langle t \rangle} \rightarrow \mathbf{a}^{\langle t \rangle}$,
  - $\mathbf{W}_{aa}$ from $\mathbf{a}^{\langle t-1 \rangle} \rightarrow \mathbf{a}^{\langle t \rangle}$,
  - $\mathbf{W}_{ya}$ from $\mathbf{a}^{\langle t \rangle} \rightarrow \mathbf{y}^{\langle t \rangle}$

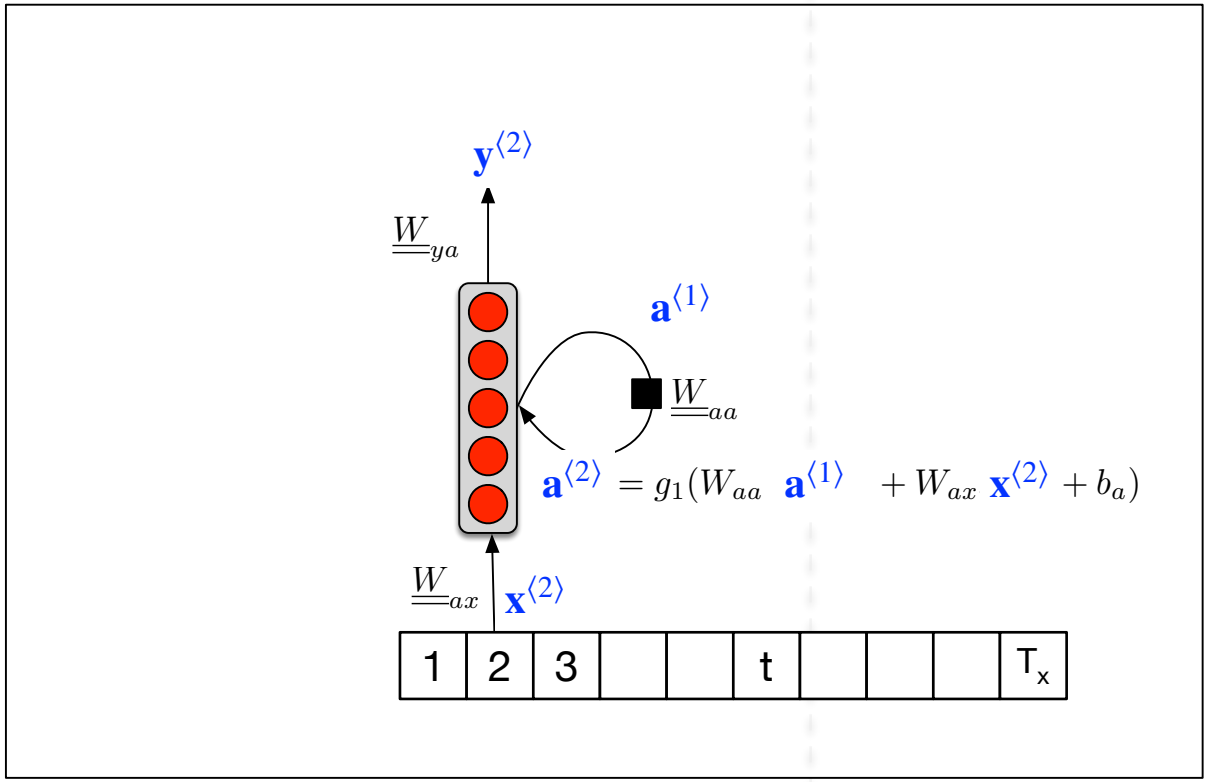
What is an RNN ?



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

Recurrent Neural Network for Sequential Data

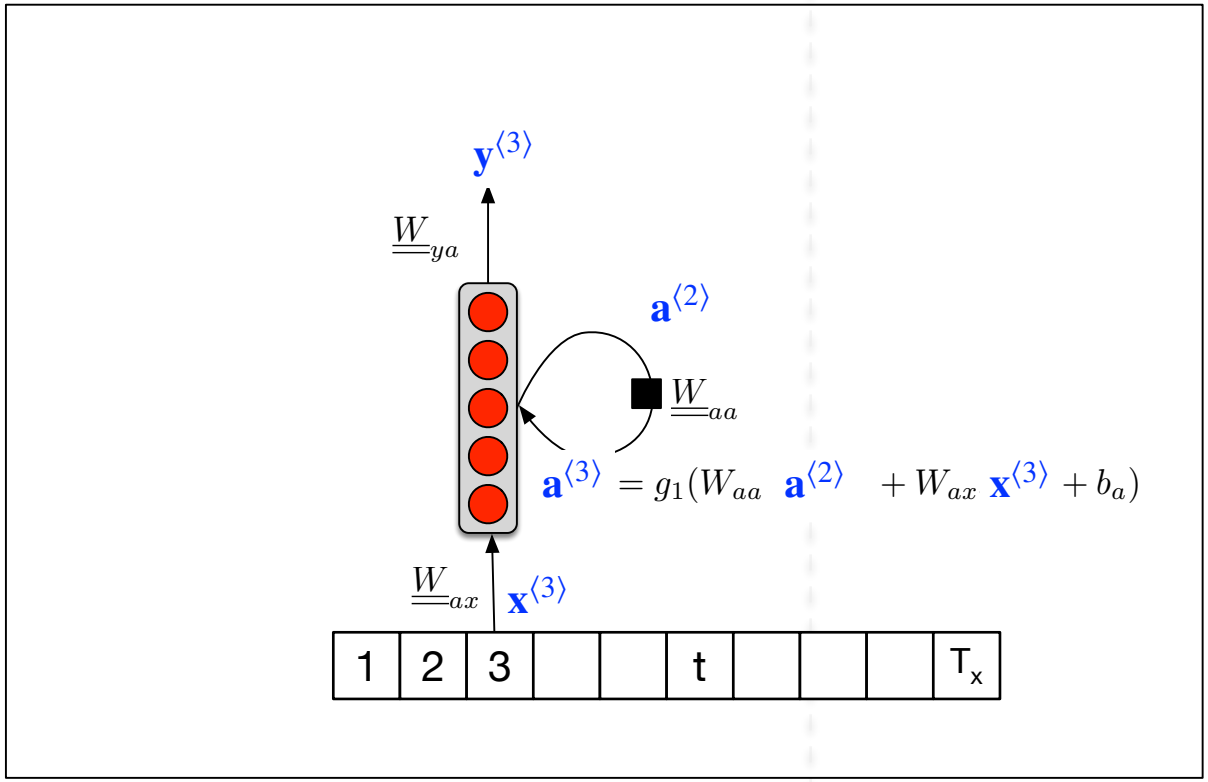
What is an RNN ?



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

Recurrent Neural Network for Sequential Data

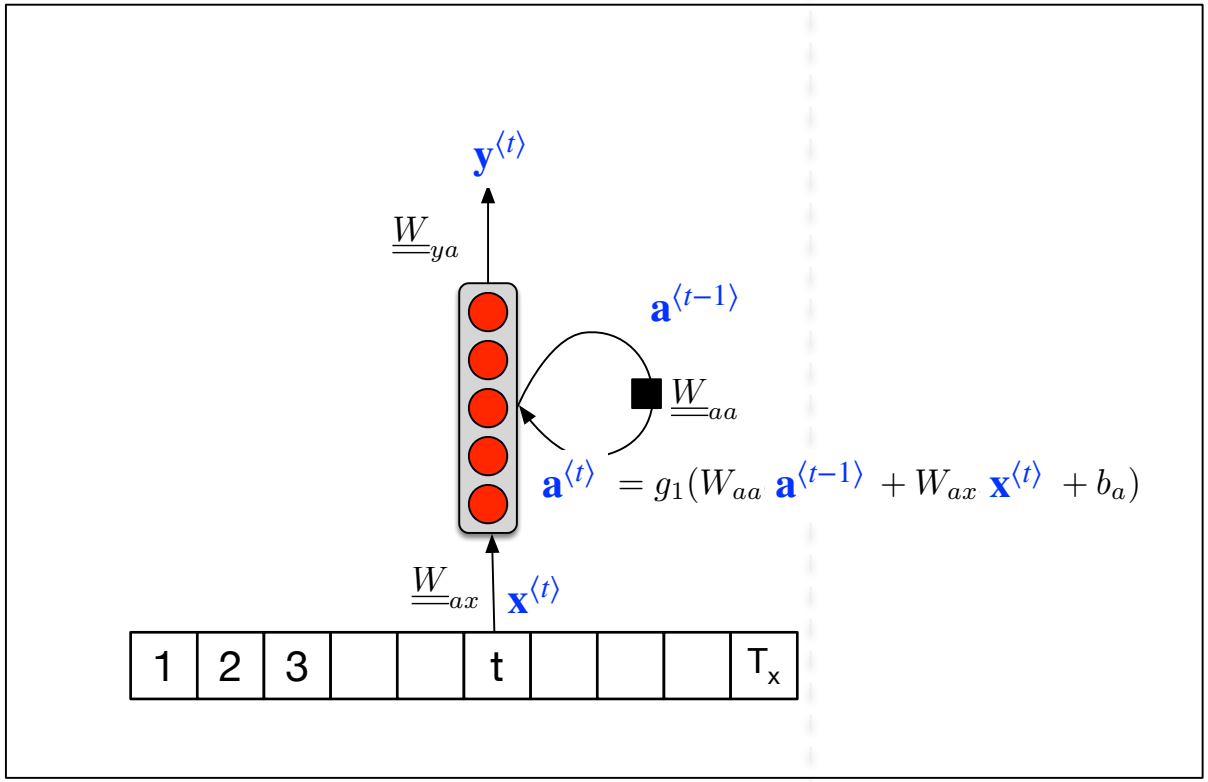
What is an RNN ?



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

Recurrent Neural Network for Sequential Data

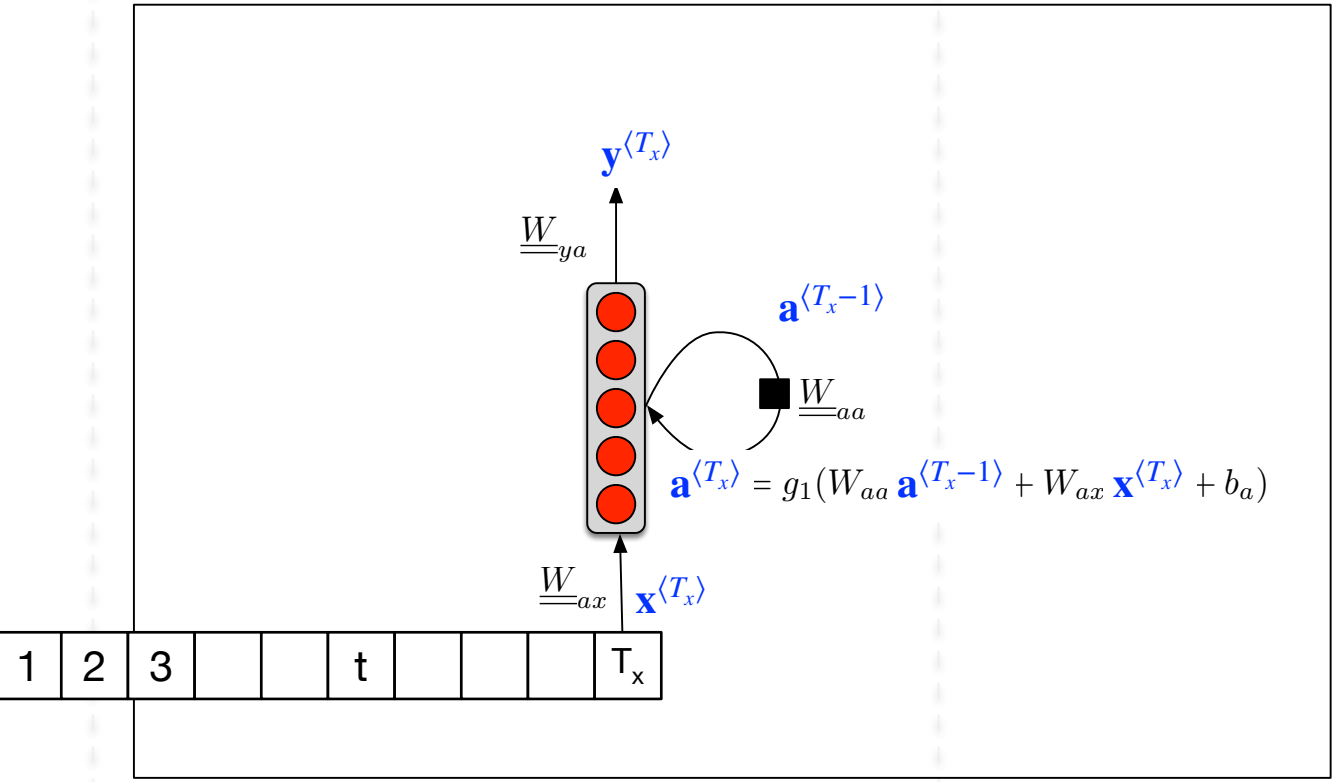
What is an RNN ?



– **"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**

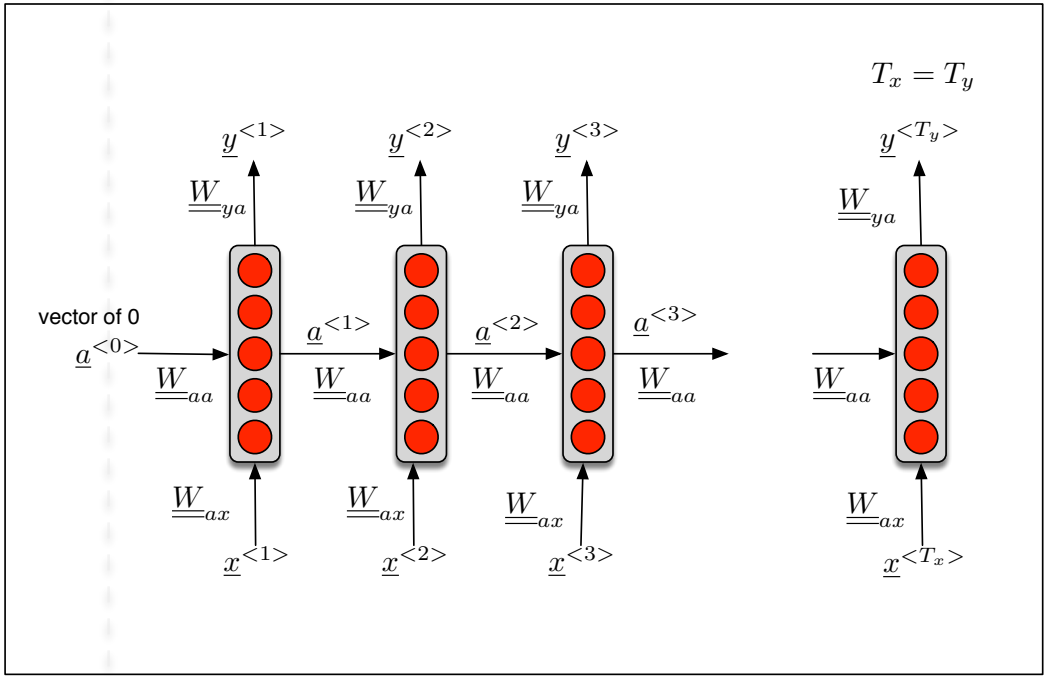
$$\mathbf{a}^{\langle 0 \rangle} = 0$$

$$\dots$$

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

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

$$\dots$$



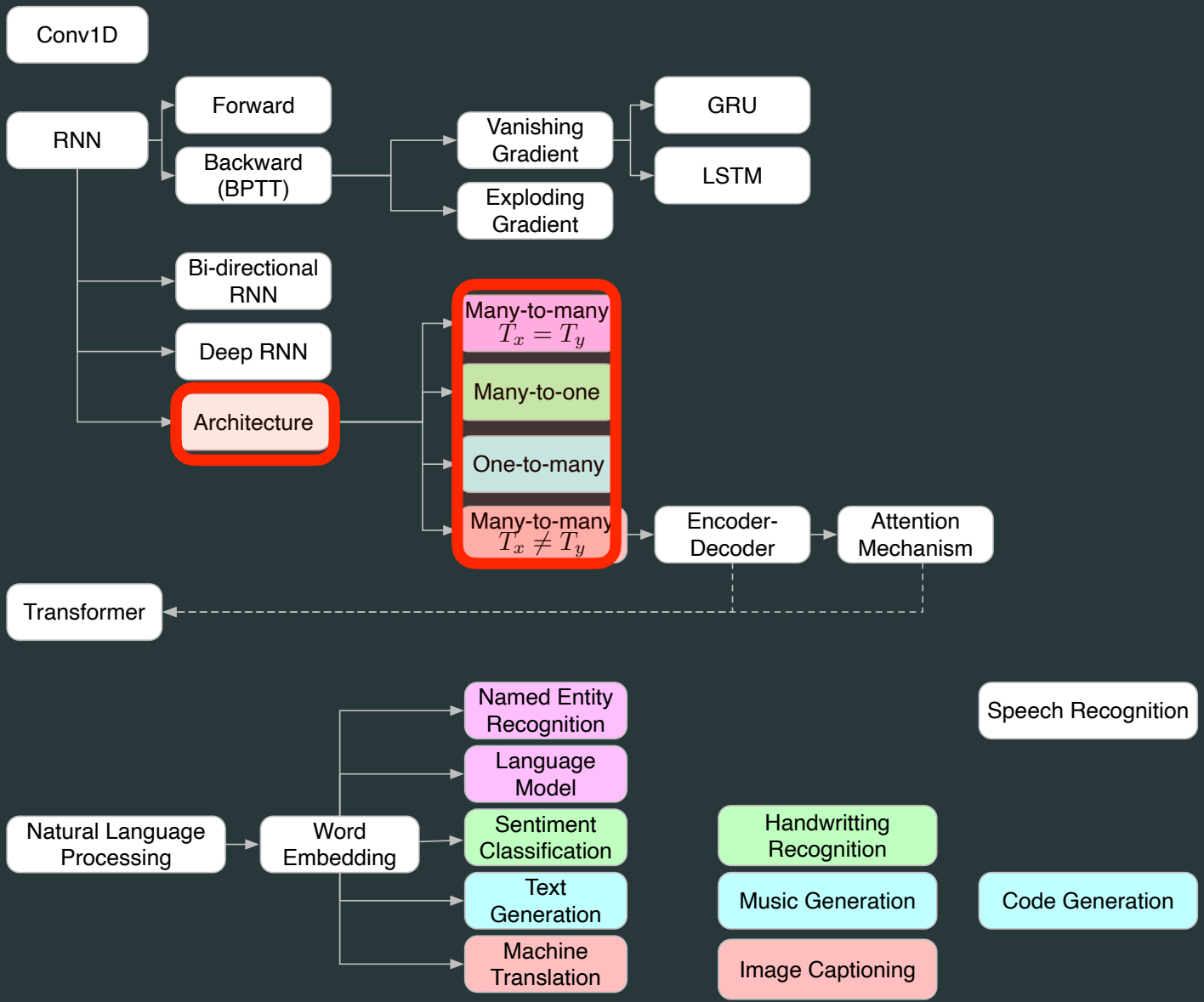
- Other possible notations

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

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

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

Various types of sequential data



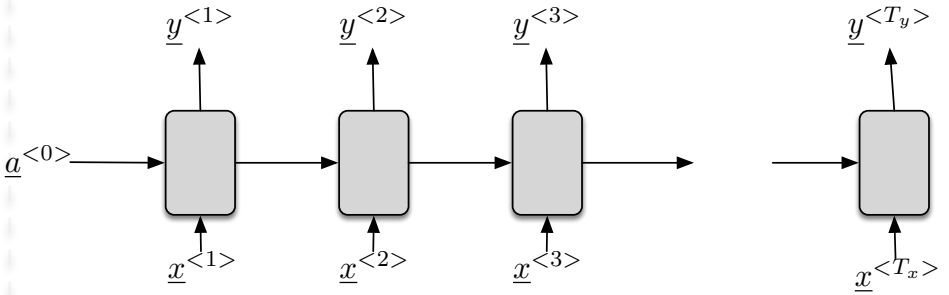
Various types of sequential data

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.

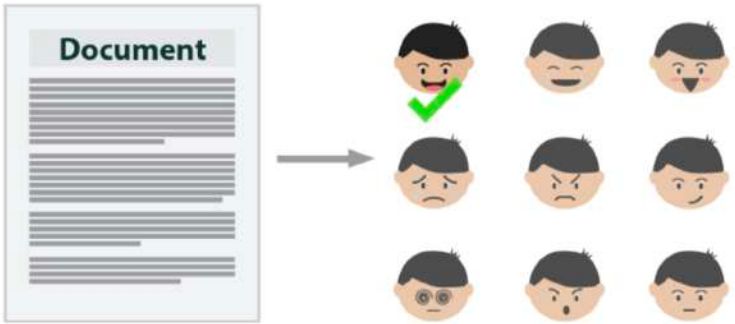
Tag colours:
LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE

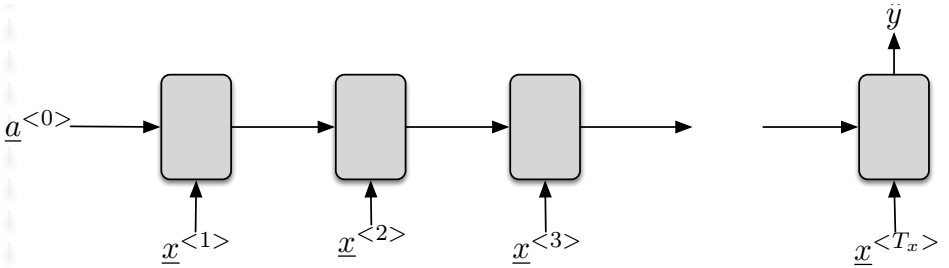


Various types of sequential data

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



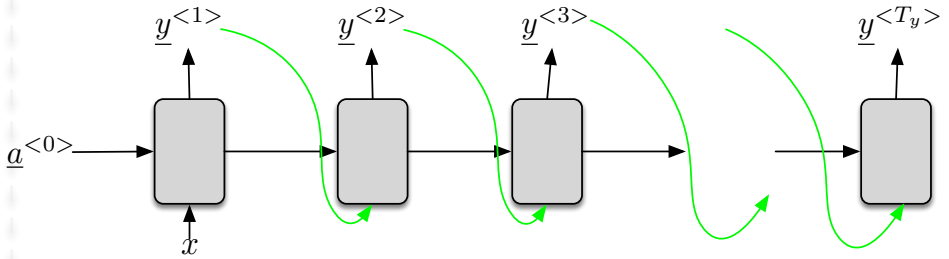


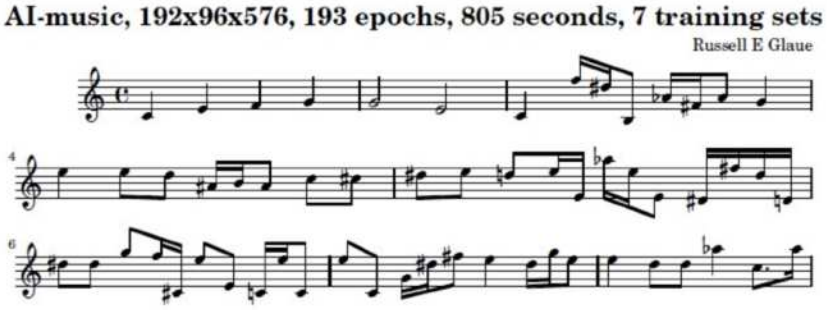
Various types of sequential data

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.]]
```

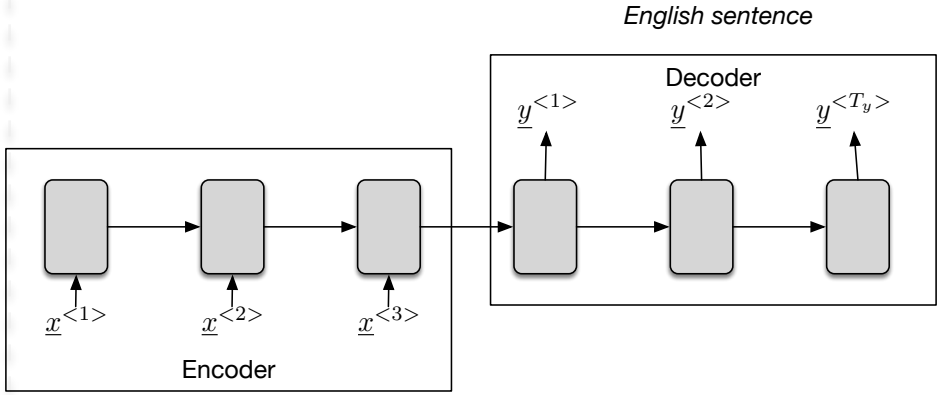




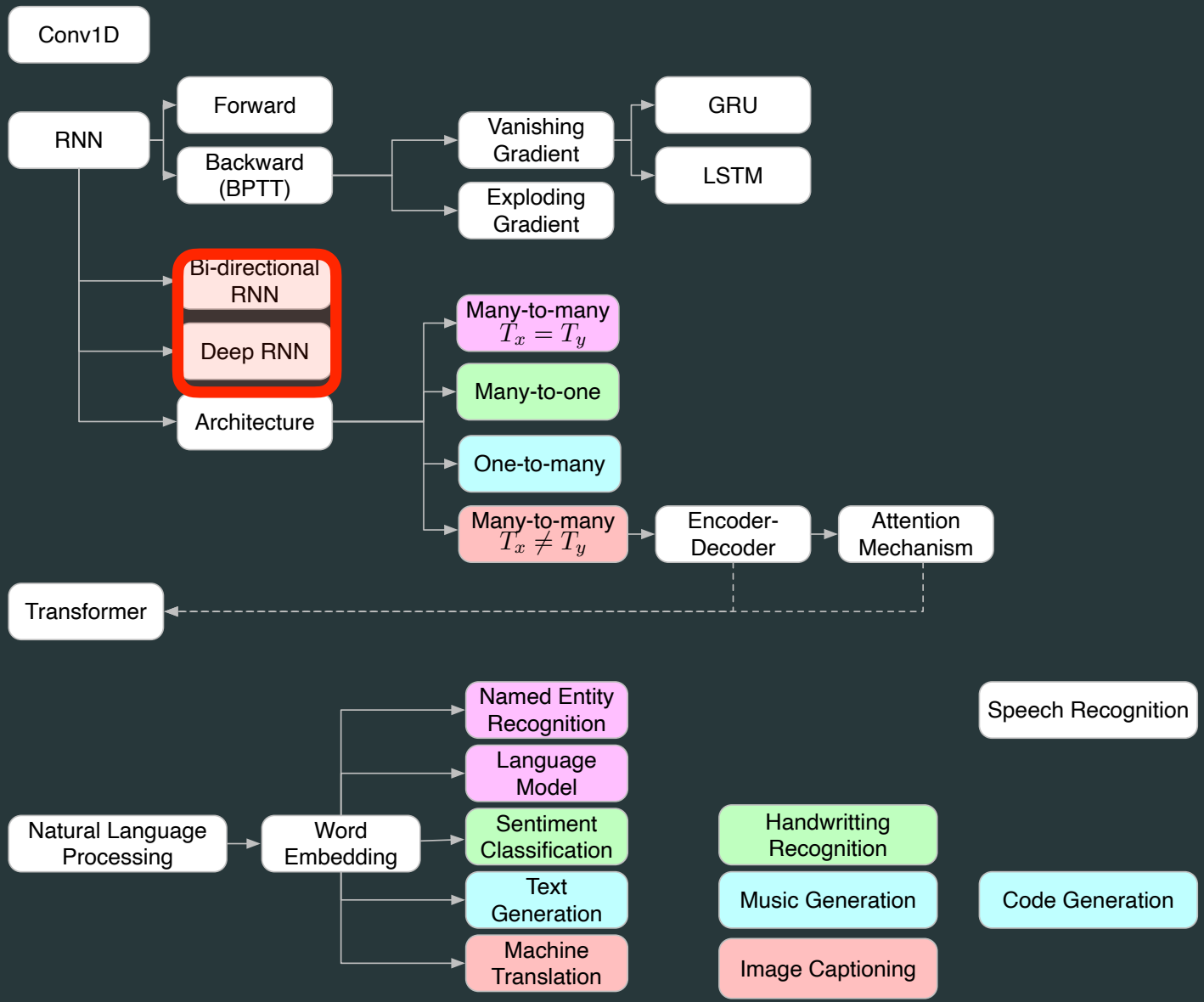
Various types of sequential data

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

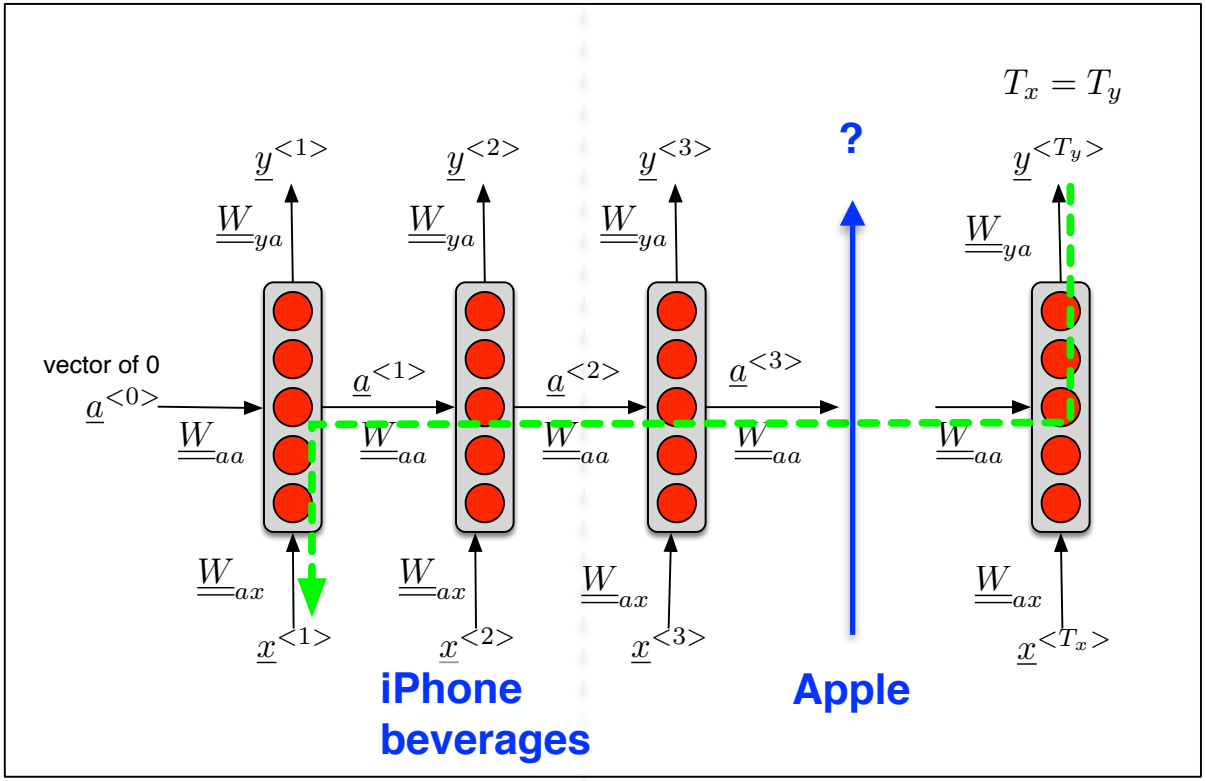


Different architectures



Different architectures

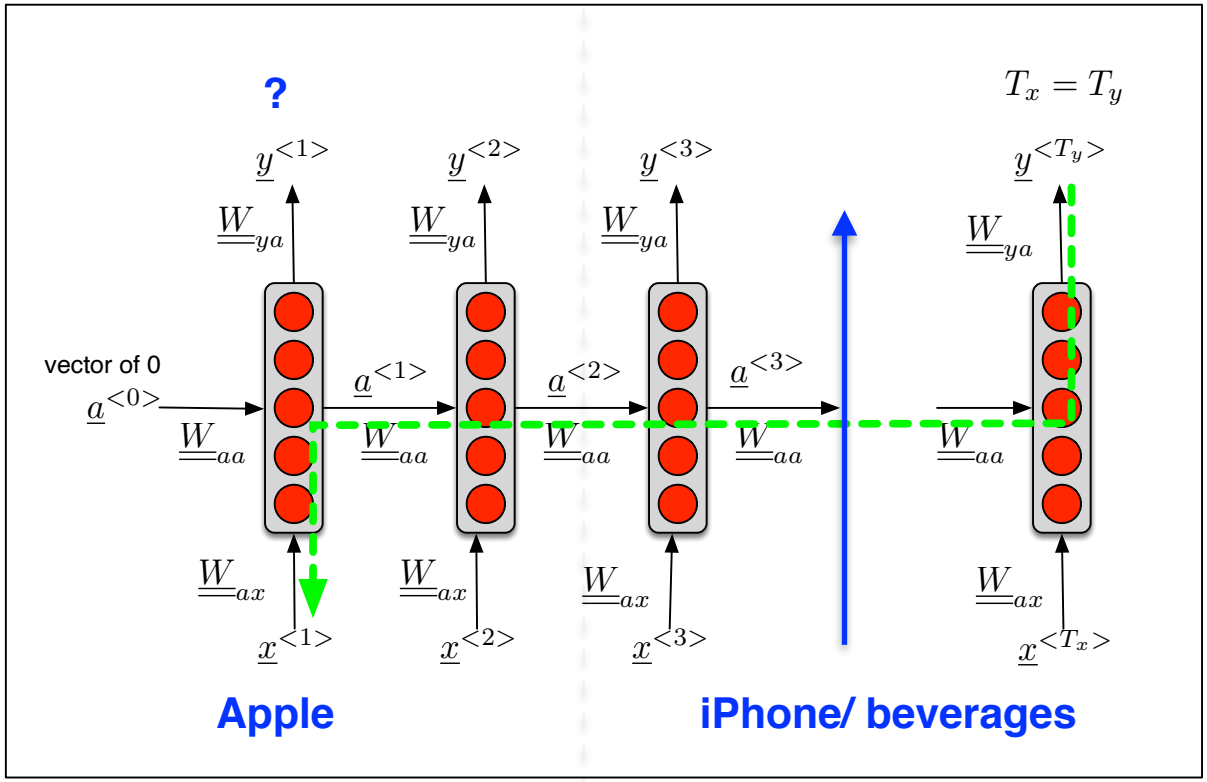
RNN



- 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

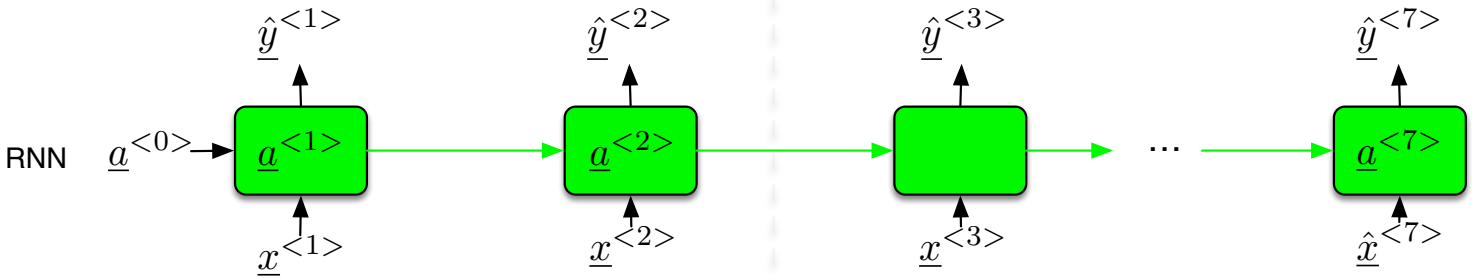
RNN



- 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

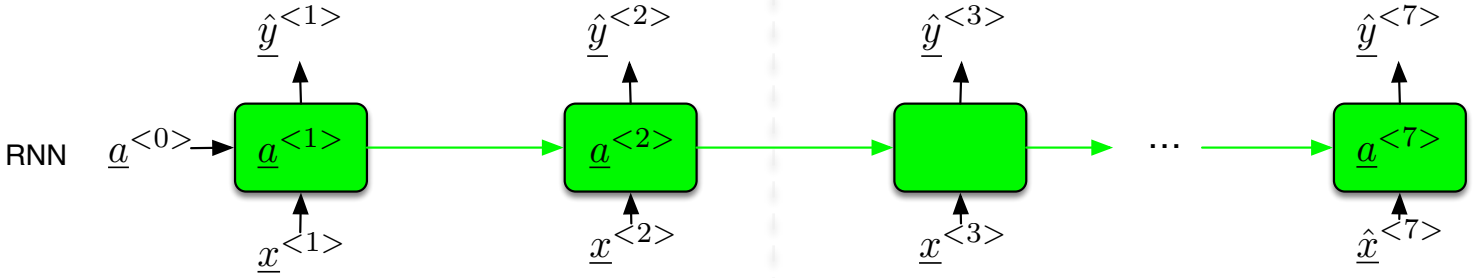
- **Standard RNN**:
  - Left-Right hidden states: $a^{\langle t \rangle}$



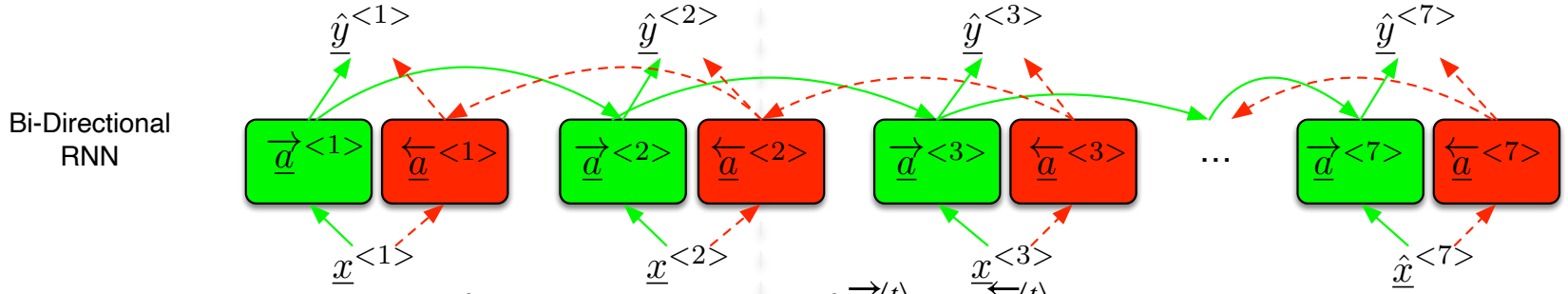
Different architectures

Bi-directional RNN

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



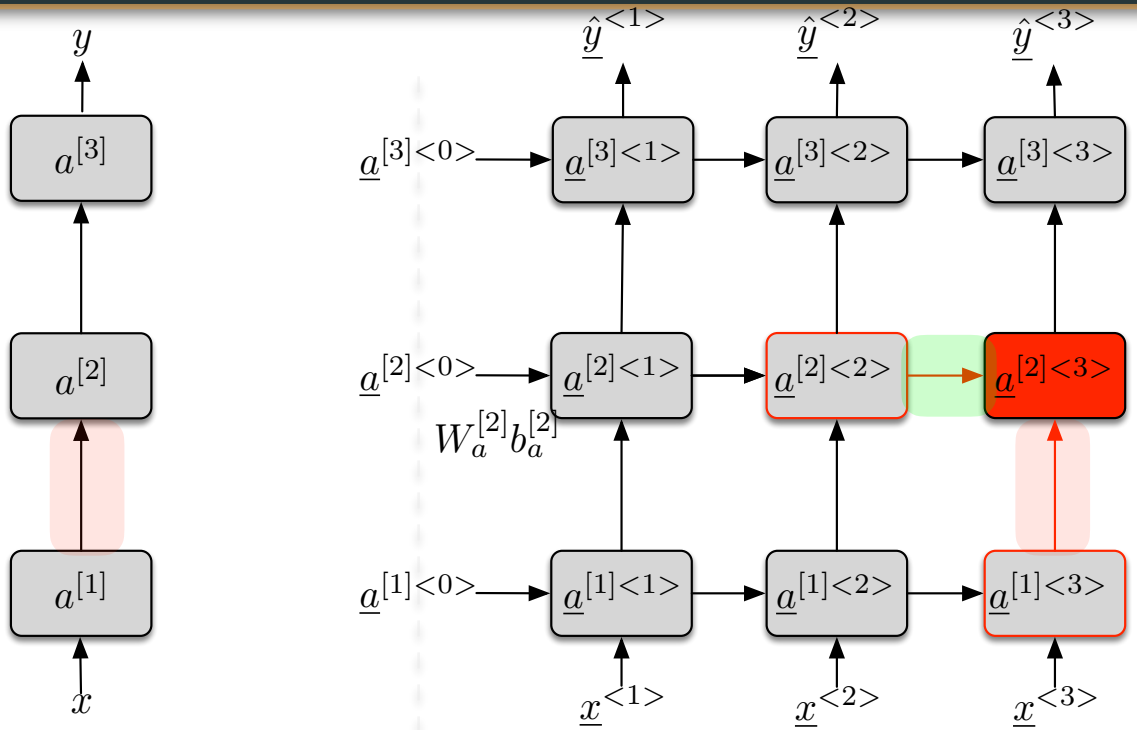
- **Bi-directional RNN:**
  - Left-Right hidden states: $\overrightarrow{a}^{\langle t \rangle}$
  - Right-Left hidden states: $\overleftarrow{a}^{\langle t \rangle}$



  - 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

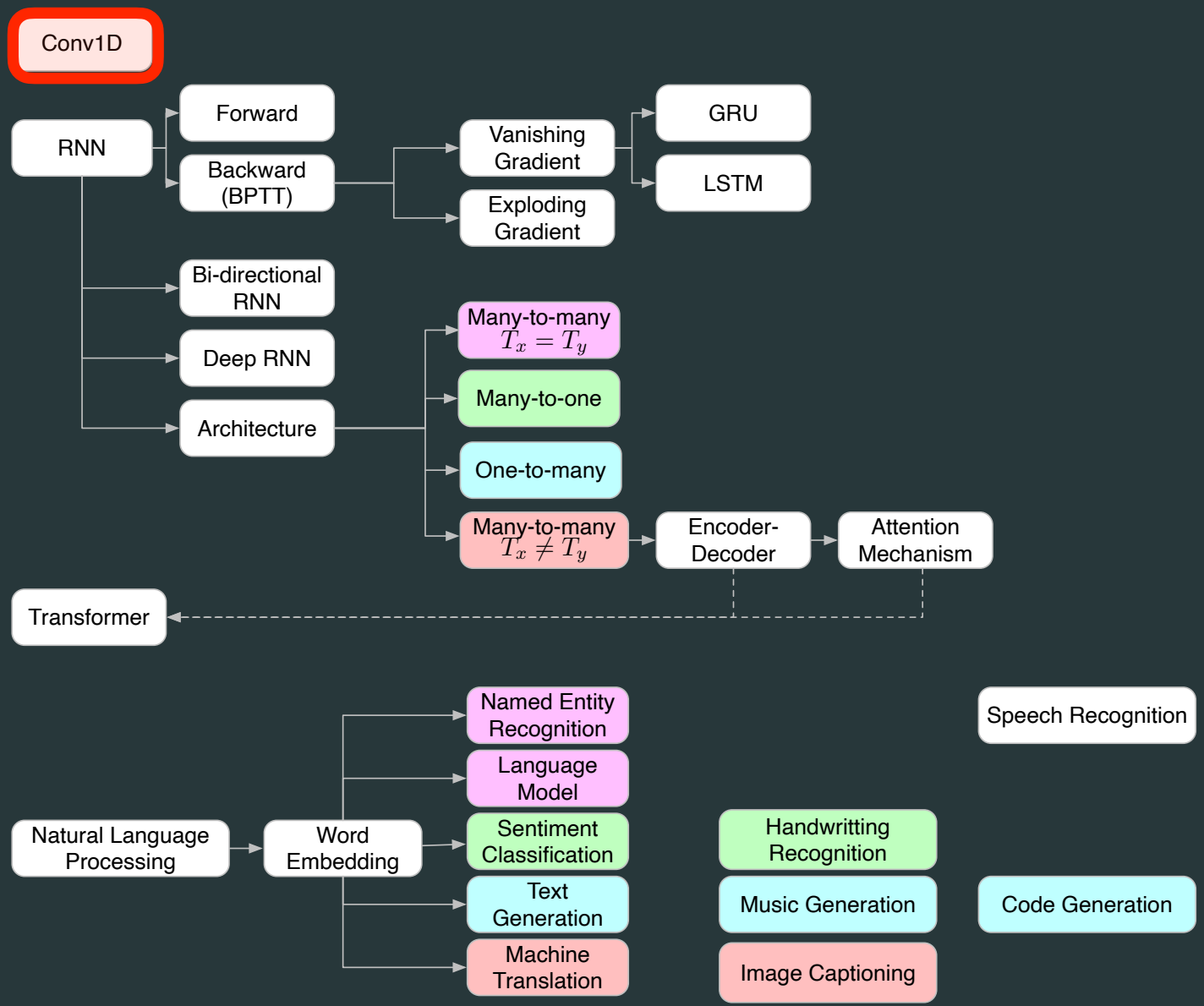
Deep-RNN



- 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 $\langle t-1 \rangle$: $\mathbf{a}^{[l]\langle t-1 \rangle}$

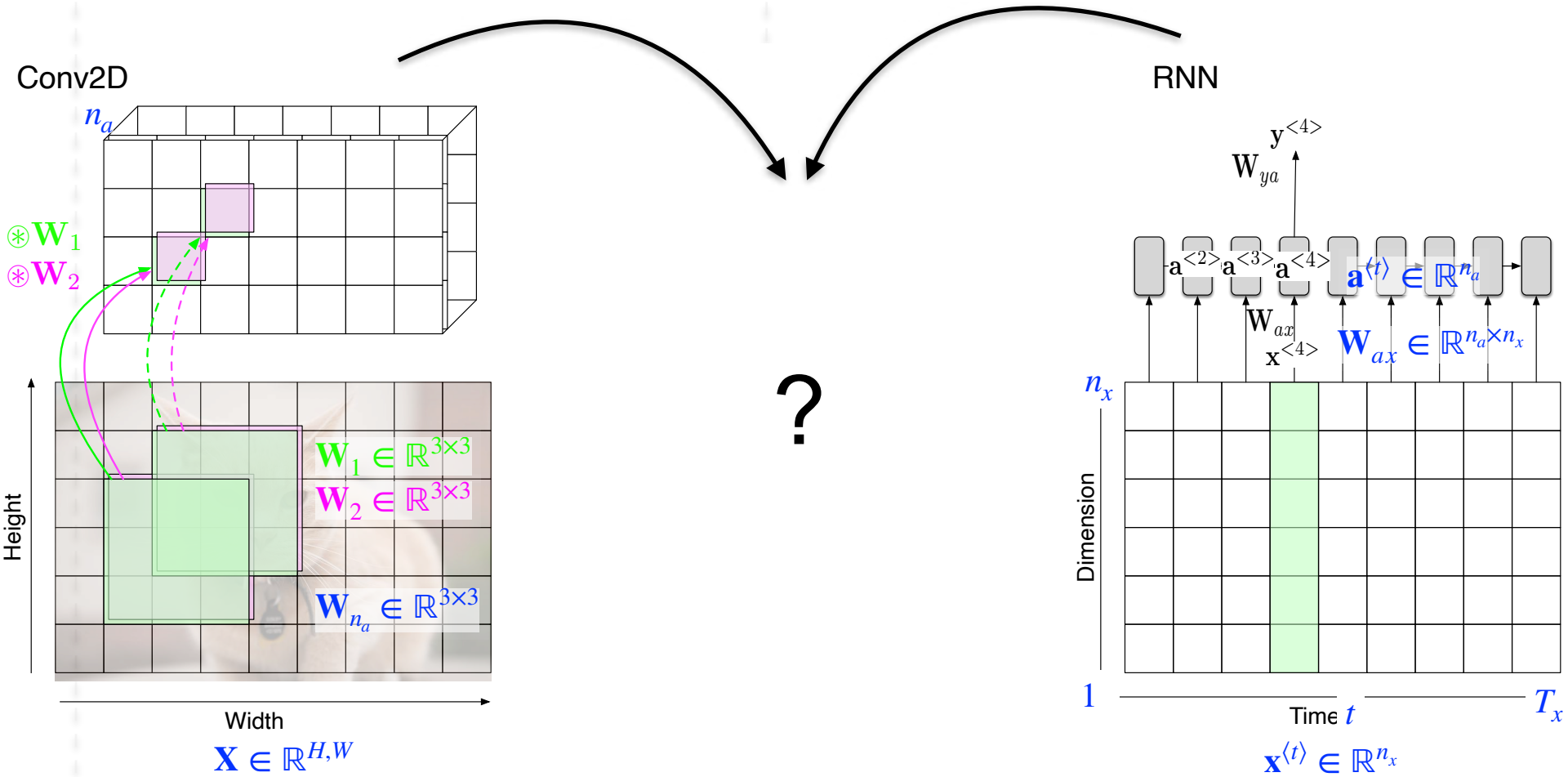
$$a^{[l]<t>} = g\left(W_a^{[l]}[a^{[l]\langle t-1 \rangle}, a^{[l-1]\langle t \rangle}] + b_a^{[l]}\right)$$

Alternative to RNN



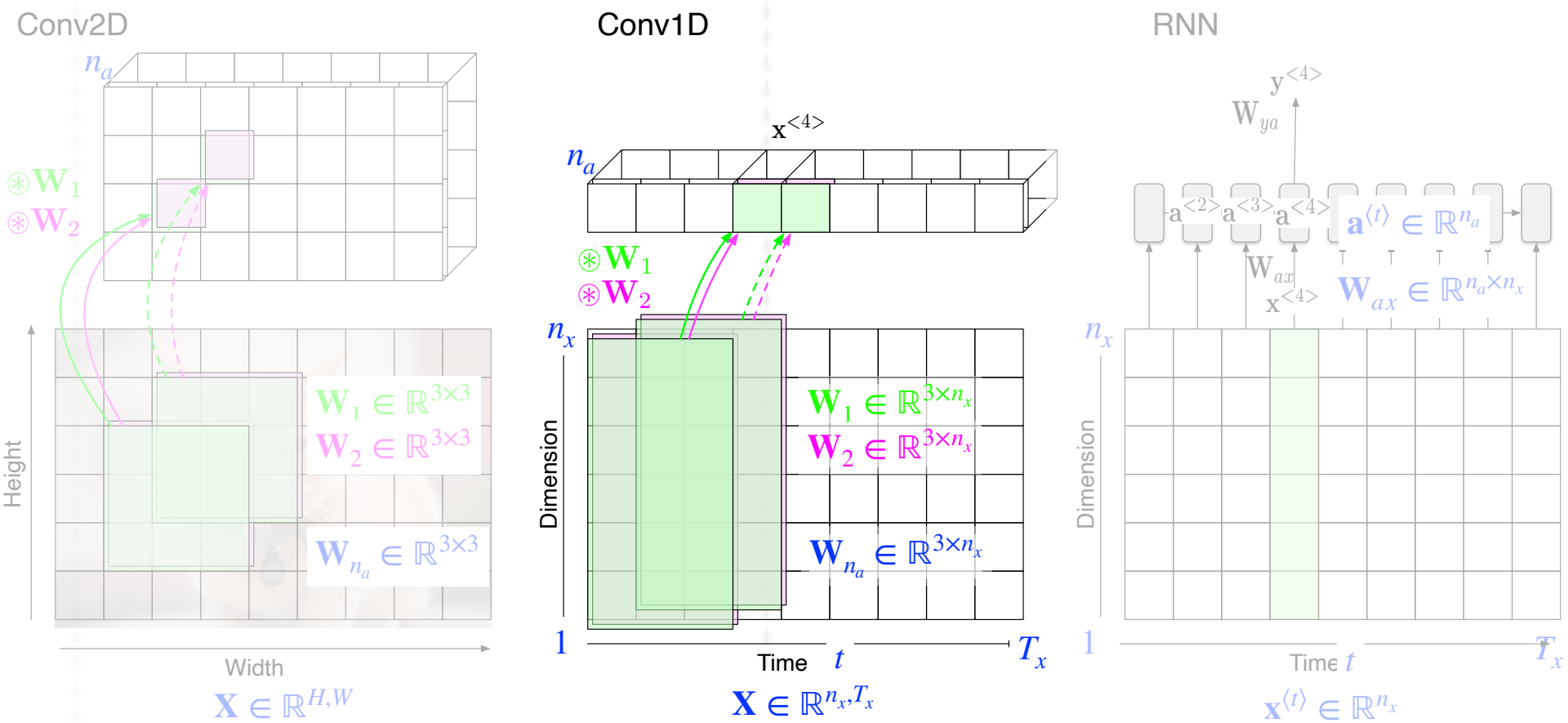
Alternative to RNN

Using 1D convolution instead of RNN

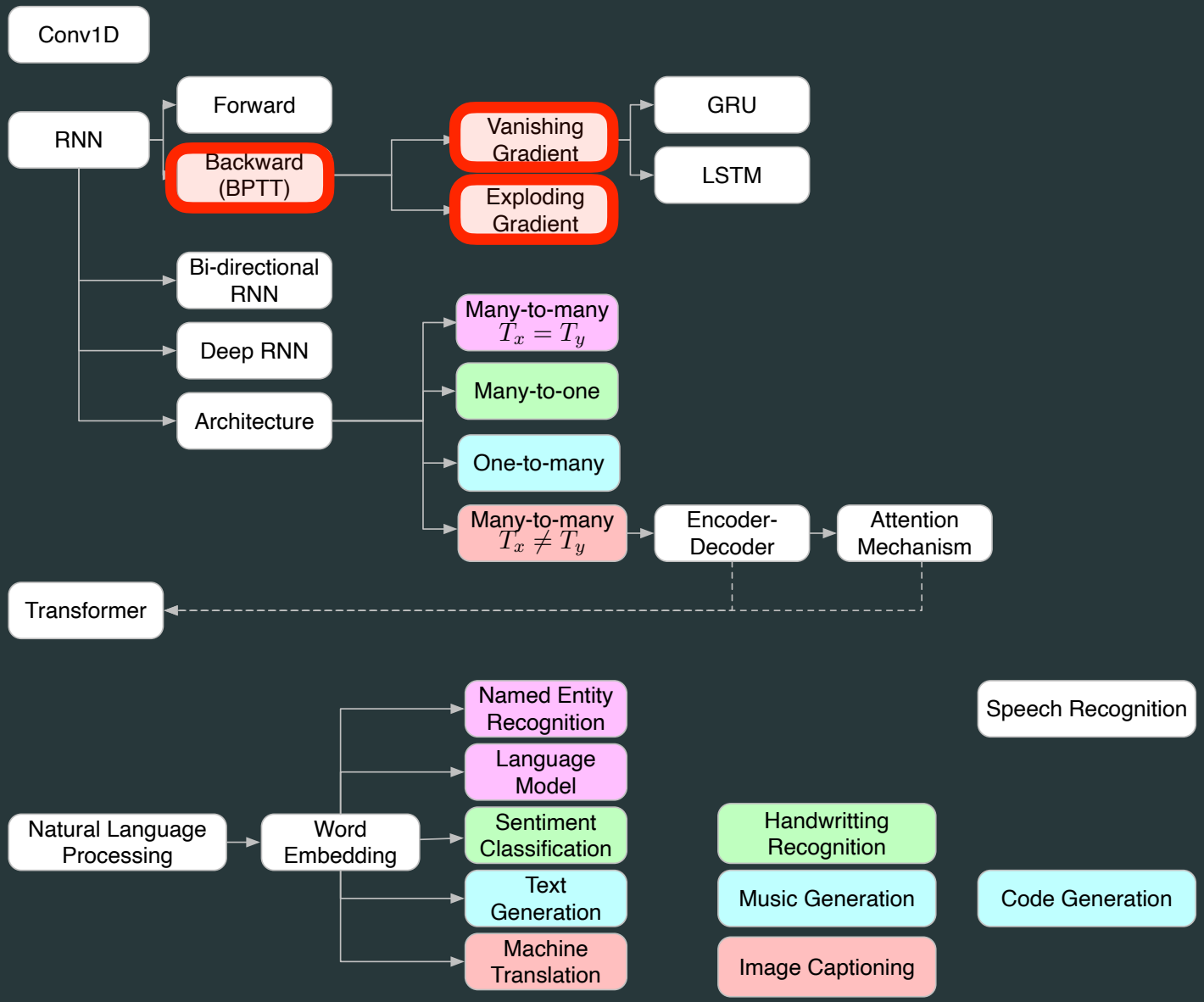


Alternative to RNN

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**

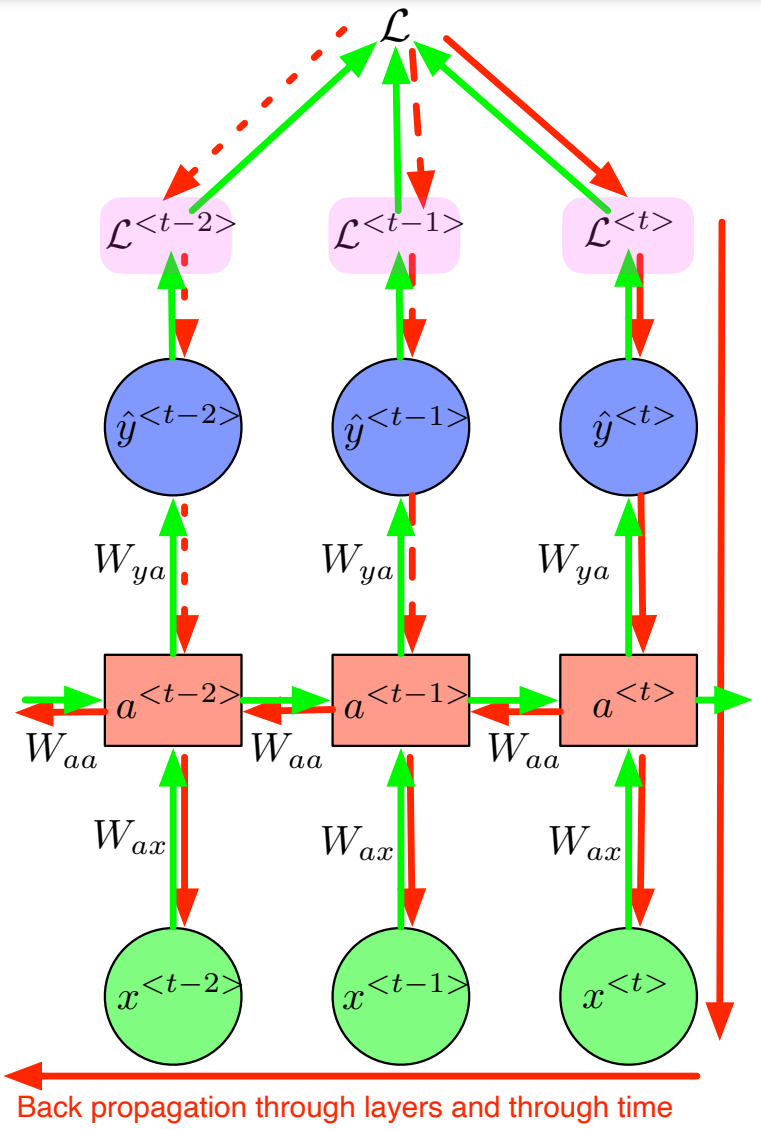
$$\mathbf{a}^{\langle t \rangle} = g_a\left(\mathbf{W}_{ax}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{aa}\mathbf{a}^{\langle t-1 \rangle} + \mathbf{b}_a\right)$$

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

- **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

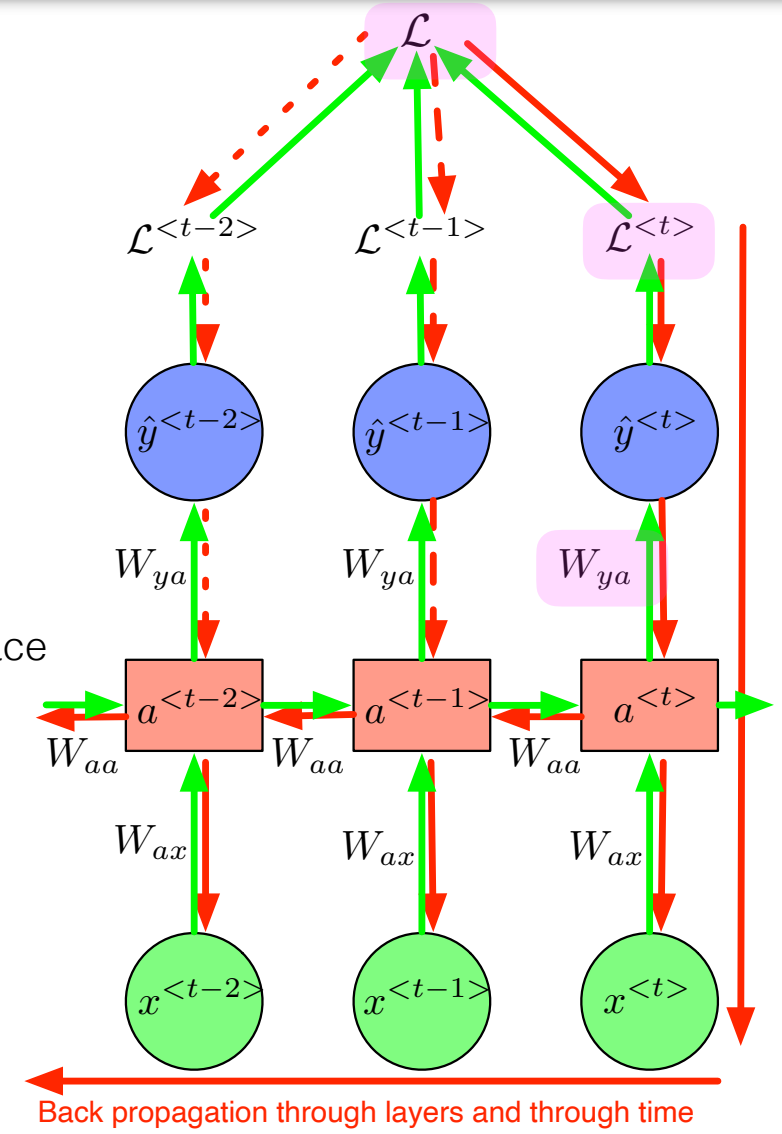
$$\frac{\partial \mathscr{L}}{\partial W_{ya}} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}^{\langle t \rangle}}{\partial W_{ya}}$$

- 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 ...

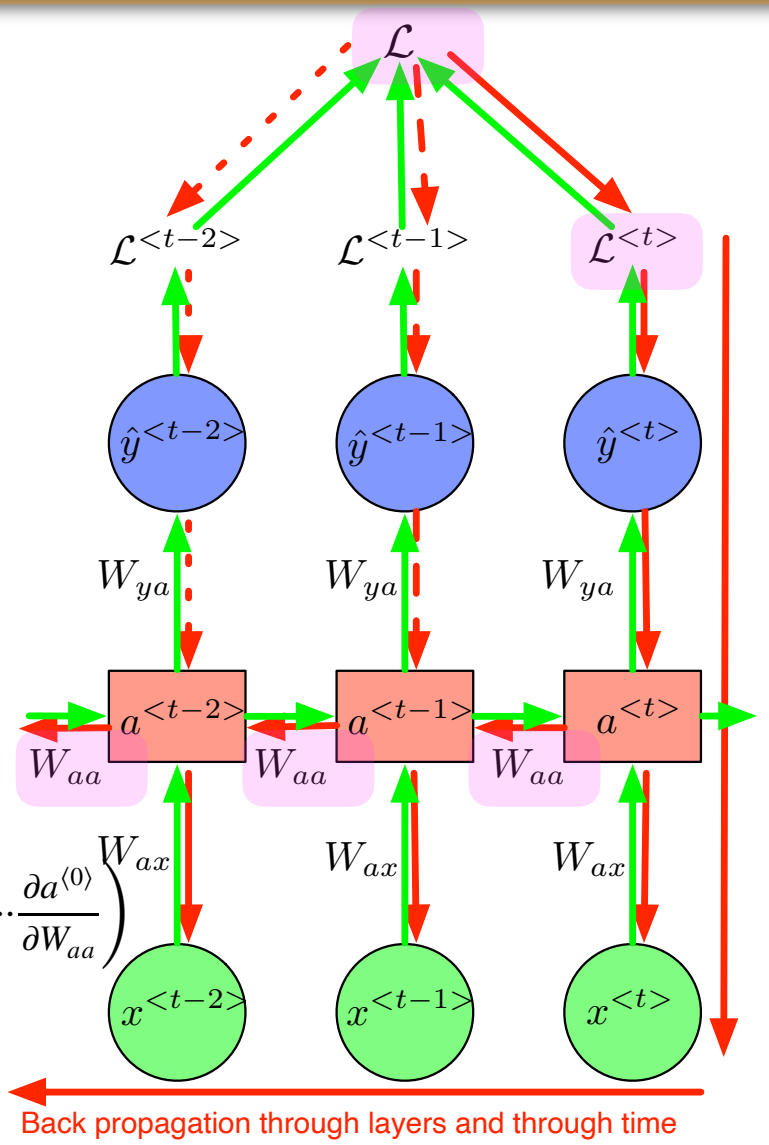
$$\mathbf{a}^{\langle t \rangle} = g_a \left(\mathbf{W}_{ax} \mathbf{x}^{\langle t \rangle} + \mathbf{W}_{aa} \mathbf{a}^{\langle t-1 \rangle} + \mathbf{b}_a \right)$$

- 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 ...

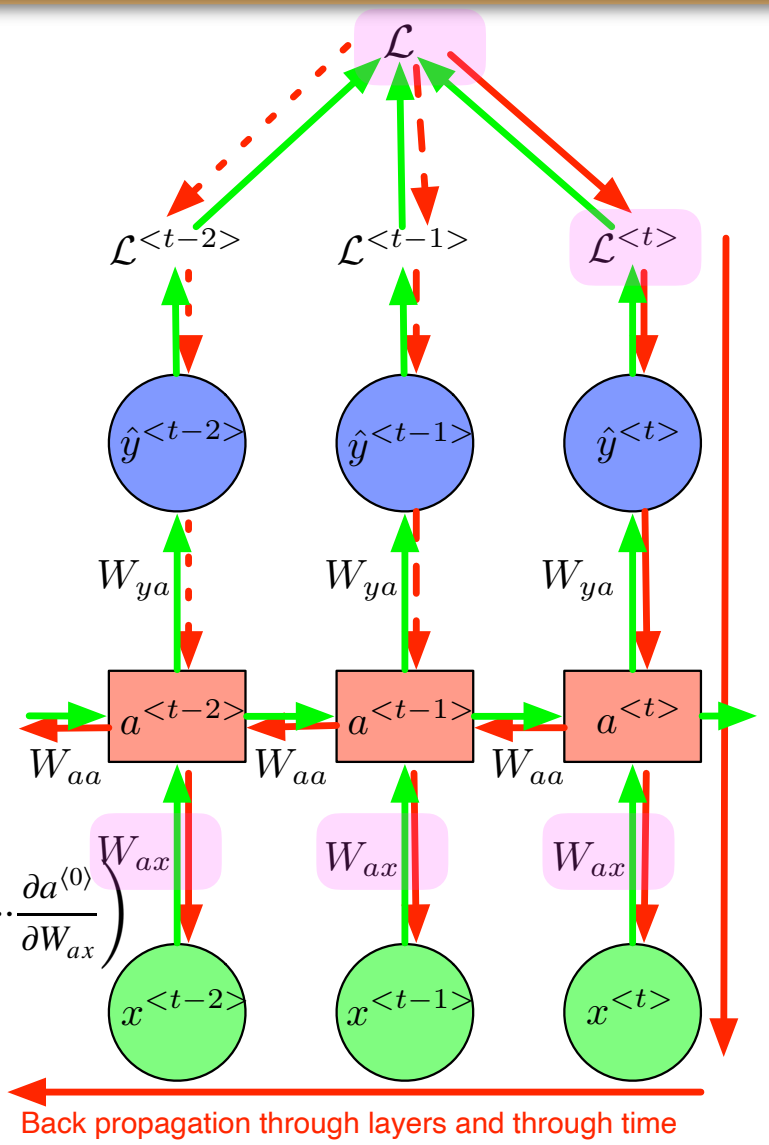
$$\mathbf{a}^{\langle t \rangle} = g_a\left(\mathbf{W}_{ax}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{aa}\mathbf{a}^{\langle t-1 \rangle} + \mathbf{b}_a\right)$$

- We use a formula of total derivative

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

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

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



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)

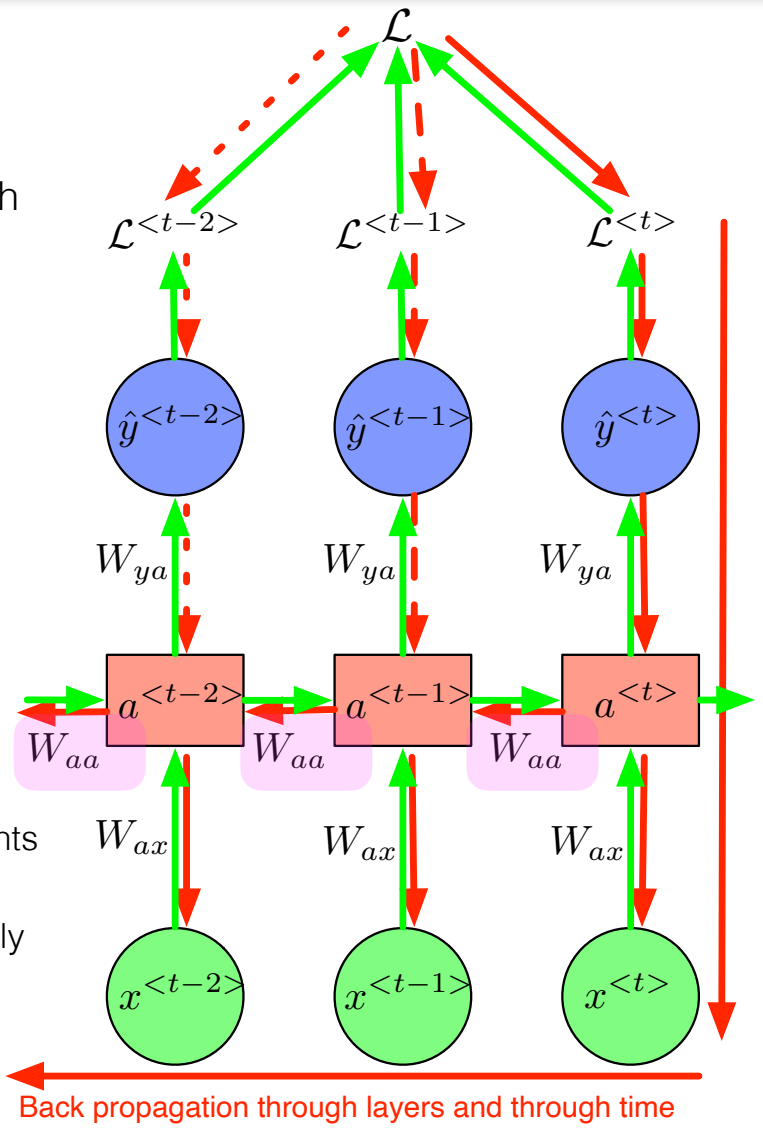
Vanishing and exploding gradients

– 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**

Back Propagation Through Time (BPTT)

Dealing with the exploding gradient

– **Solution 1: gradient clipping**

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

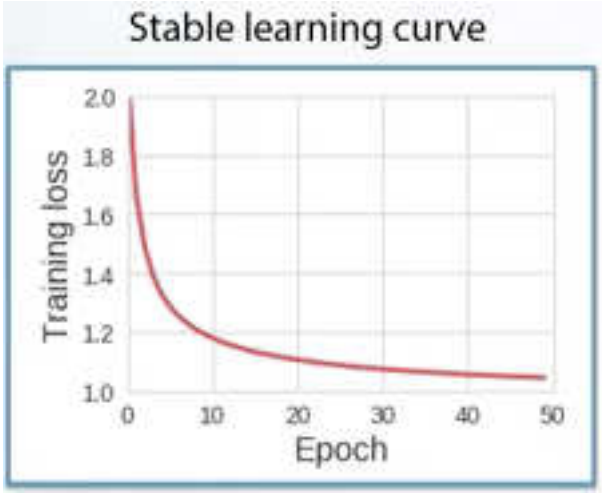
- if $||\theta||$ > threshold

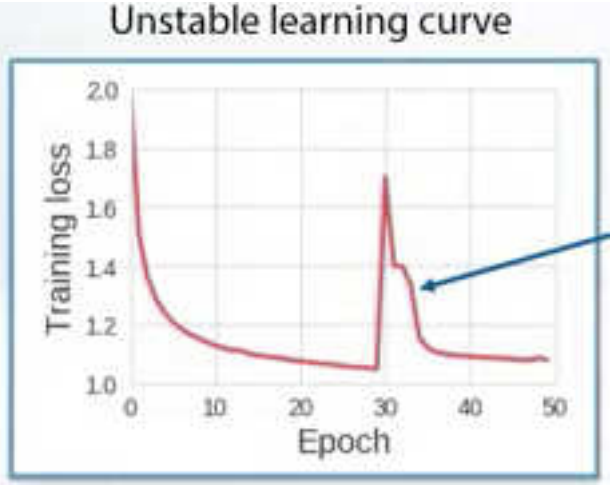
$$d\theta \leftarrow \frac{\text{threshold}}{||\theta||} d\theta$$

– 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**

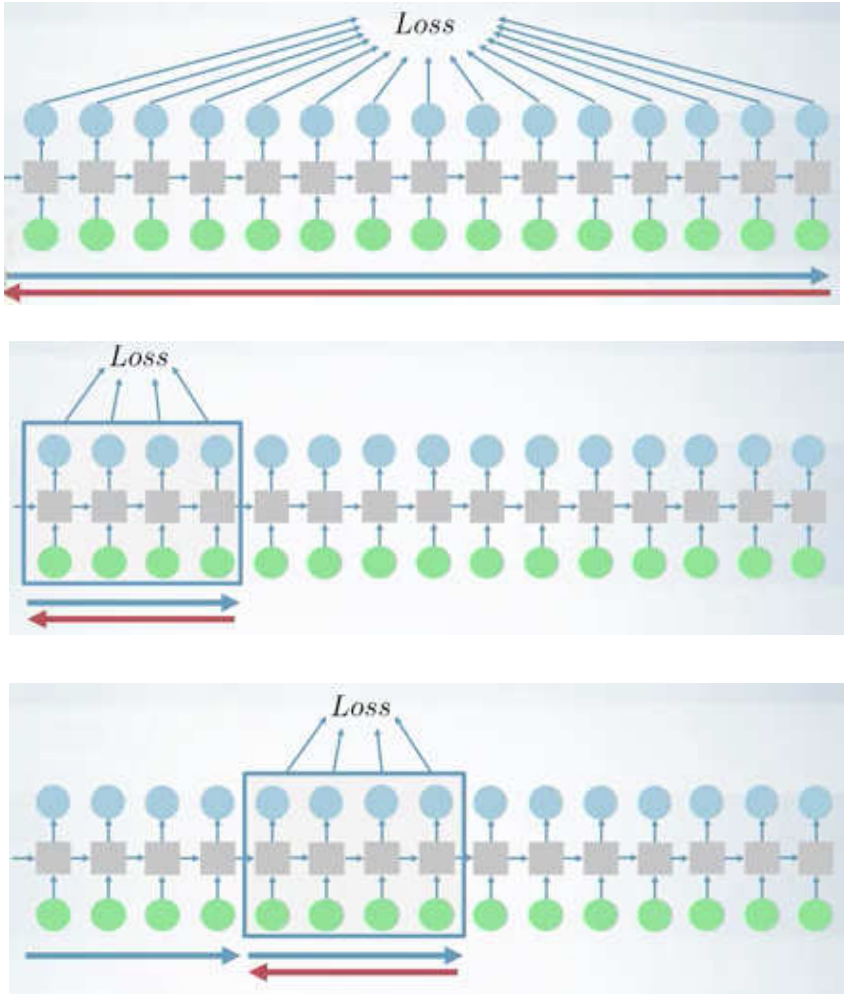
– Training very long sequences
  - time consuming !
  - exploding gradients !

– 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

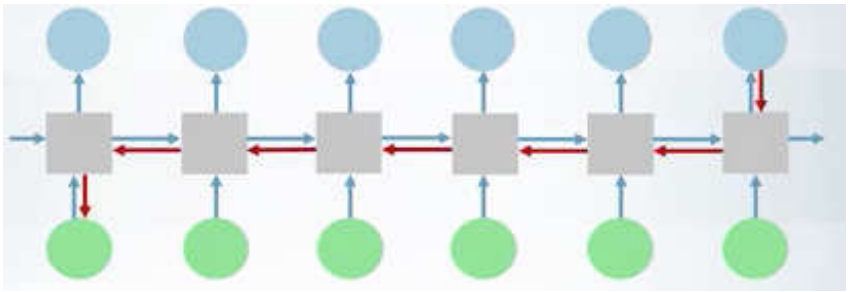
– **Solution 2: use skip-connections**

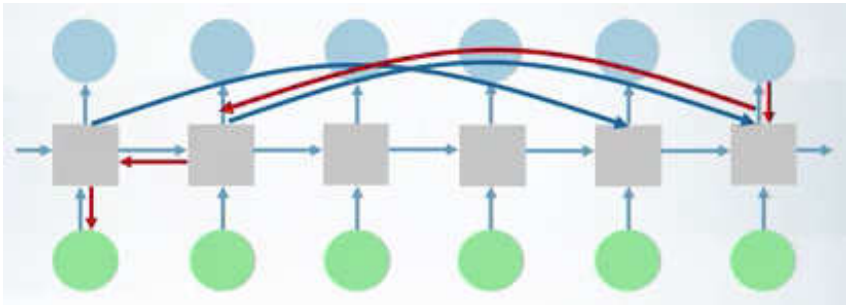
– in RNN: because at each step we multiply the Jacobian matrices → vanishing gradient
  - we cannot learn long-term dependencies

– 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

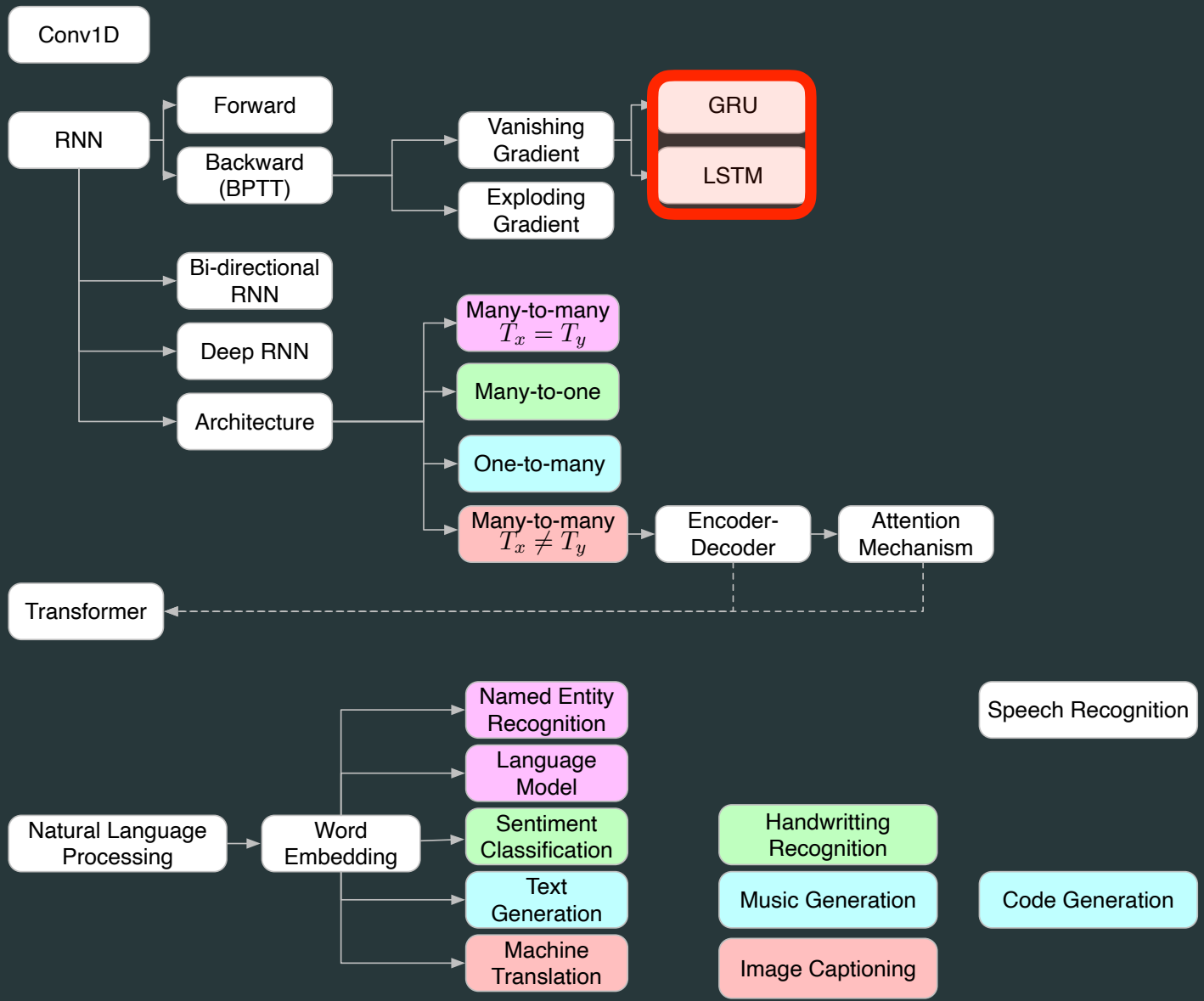
– Back-propagation through the shortcuts: the gradients vanish slower
  - we can learn long-term dependencies

– not exclusive to RNN (see ResNet)





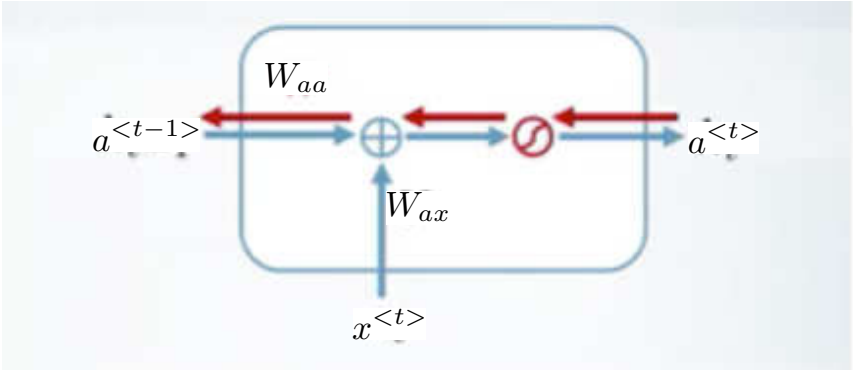
Gated units (LSTM and GRU)



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)

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

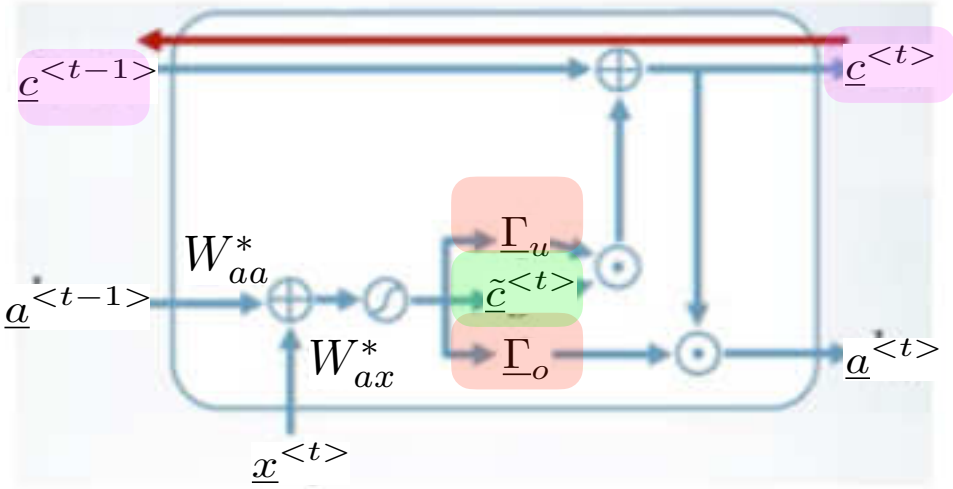
$$\Gamma_o = \sigma\left(W_{ax}^{o} x^{\langle t \rangle} + W_{aa}^{o} a^{\langle t-1 \rangle} + b^{o}\right)$$

  - 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)

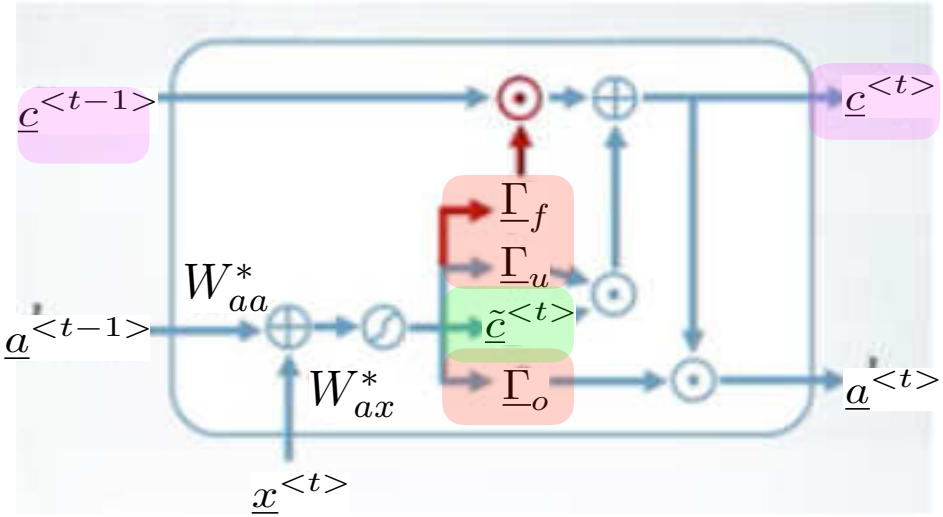
$$\Gamma_f = \sigma\left(W_{ax}^f x^{\langle t\rangle} + W_{aa}^f a^{\langle t-1\rangle} + b^f\right)$$

  - 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

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

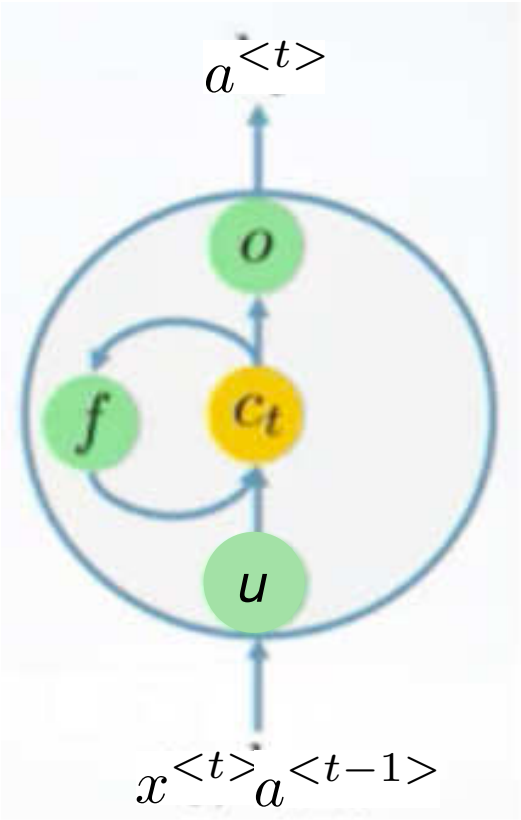


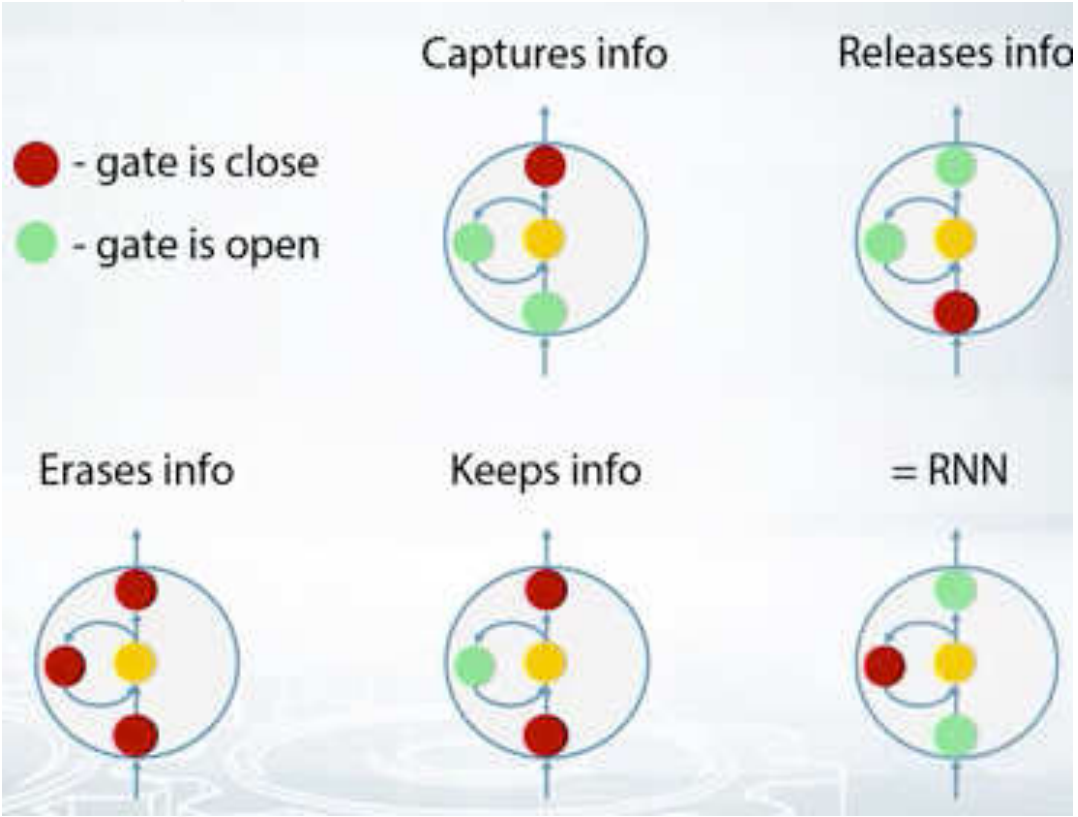
- We now have a multiplication on the short-way
  - $\dfrac{\partial c^{\langle t\rangle}}{\partial c^{\langle t-1\rangle}} = \text{diag}(\Gamma_f)$
  - with $\Gamma_f \in [0,1]$ the forget gate: $\Gamma_f = \sigma\left(W_{ax}^f x^{\langle t\rangle} + W_{aa}^f a^{\langle t-1\rangle} + b^f\right)$
    - 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
  - $W_{ax}^f, W_{aa}^f, W_{ax}^u, W_{aa}^u, W_{ax}^o, W_{aa}^o, W_{ax}^c, W_{aa}^c + b^f, b^u, b^o, b^c$

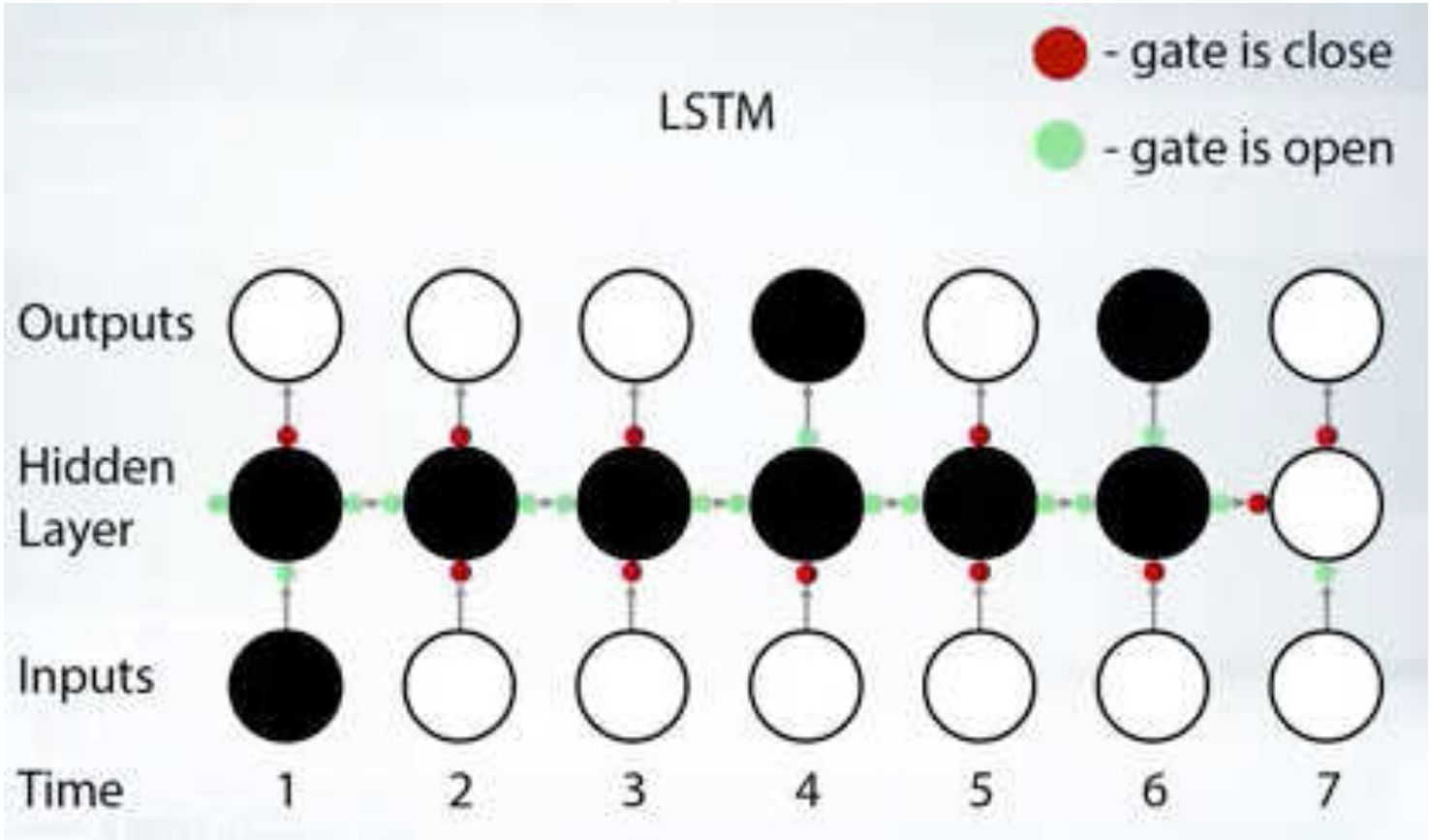
[Hochreiter and Schmidhuber 1997. "Long short-term memory", Neural Computation Volume 9 | Issue 8 | November 15, 1997 p.1735-1780]

LTSM regimes





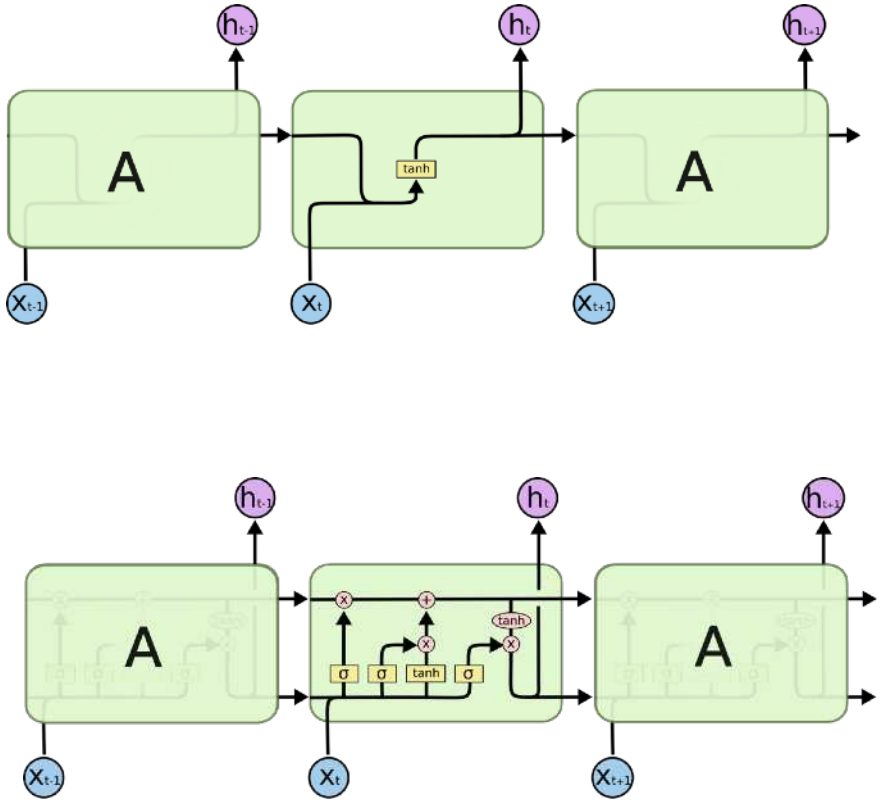
LTSM regimes



Gated units (LSTM and GRU)

LTSM example usage

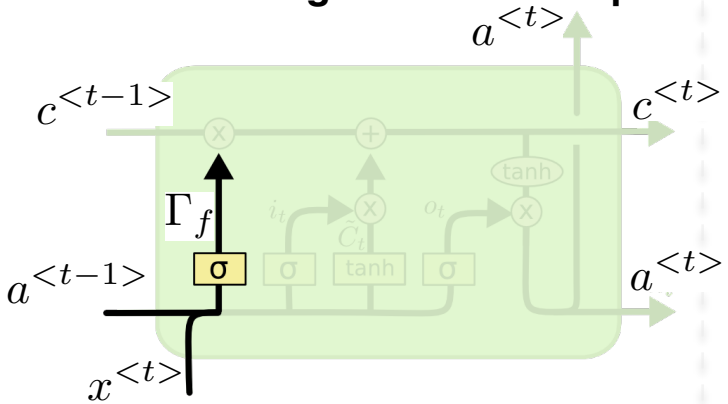
- Instead of having a single neural network layer, there are four, interacting in a very special way.
  - Lines carry a vector
  - A recurrent net transmits its hidden states
  - LSTM introduces a second channel : the cell state
  - It acts as a memory
  - Gates control the memory



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

LTSM example usage

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



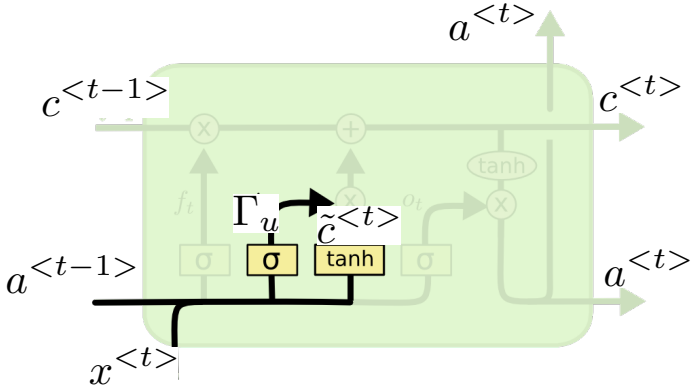
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

- **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)

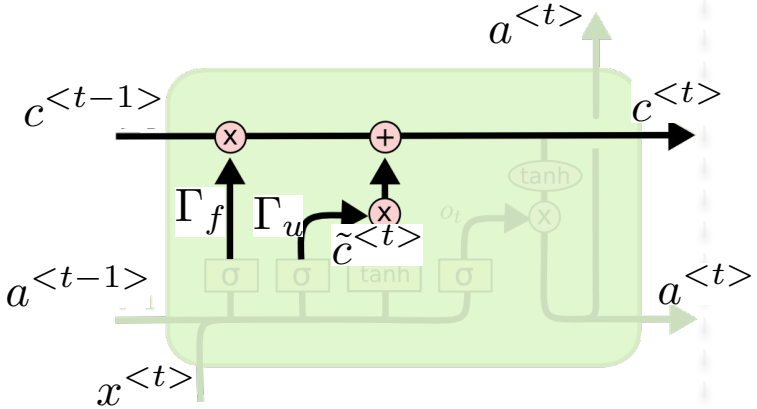
– **Intuition for language modelling**

- 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

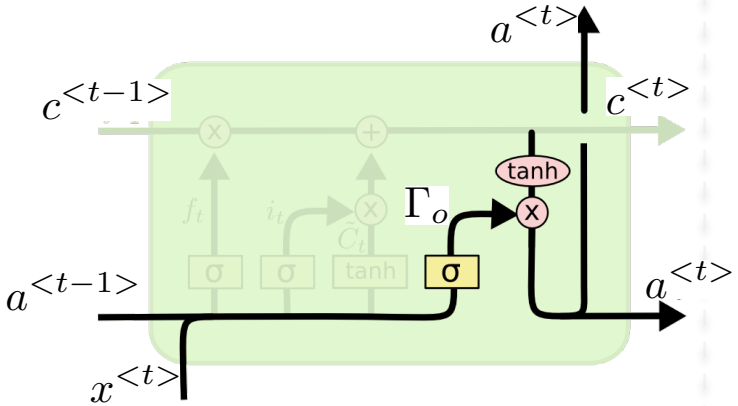
– **Intuition for language modelling**

- 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)

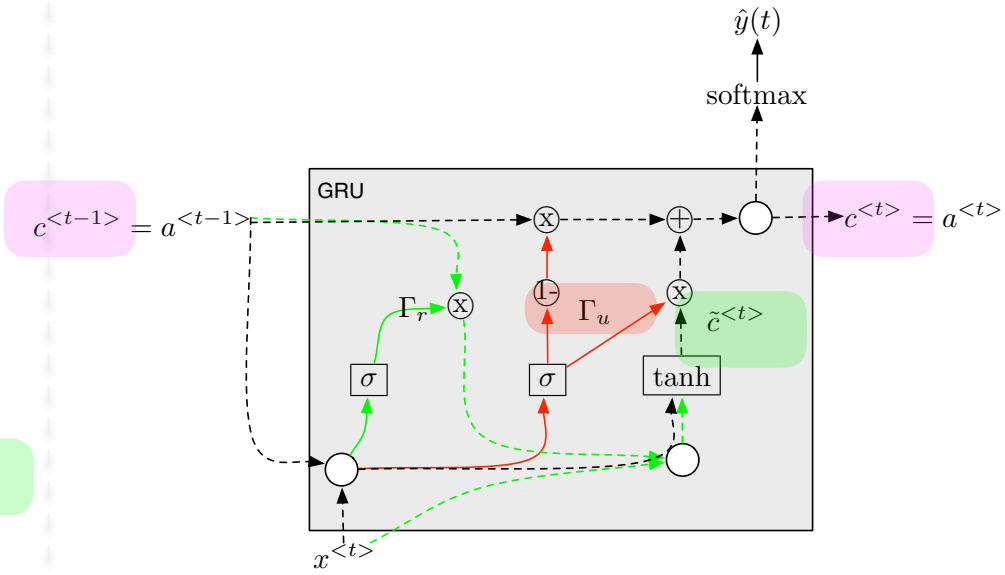
$$\Gamma_u = \sigma\left(W^u \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b^u\right)$$

  - 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)

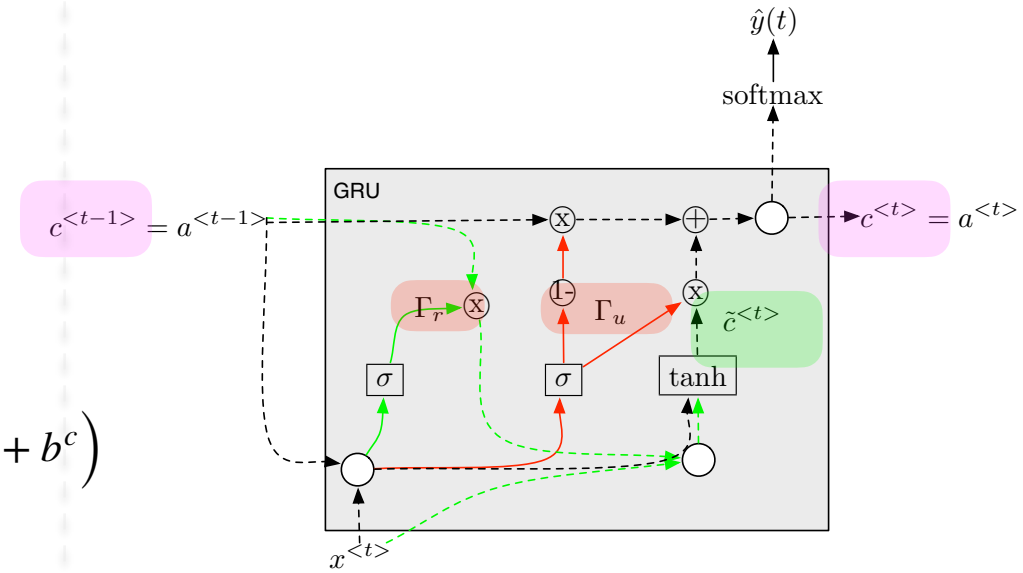
– **Full GRU**

- 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

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

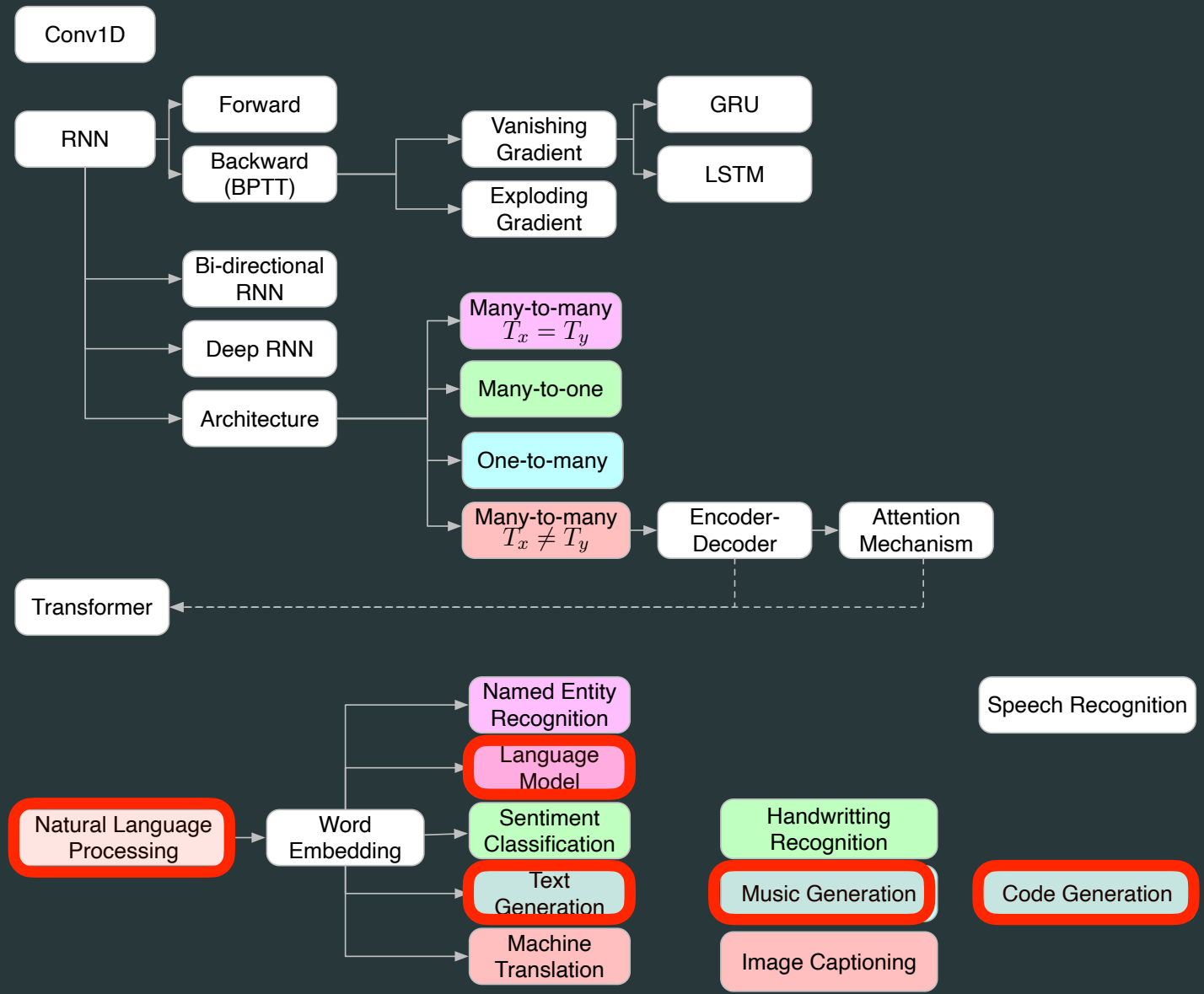
- Candidate cell value

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



[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)**

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

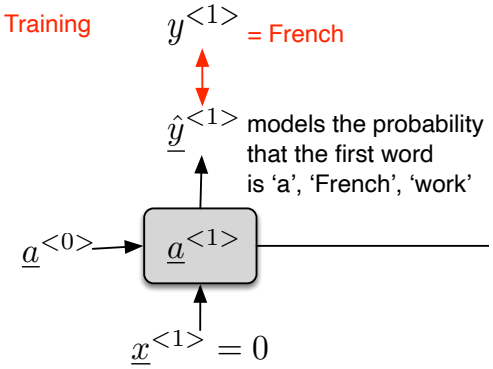
Language model

- **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

- **Objective**:
  - we want to train the parameters θ of the RNN model to maximise the probability of observing the sequence $P(w_1^L) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2))\ldots P(w_L \mid w_1 \ldots w_{L-1})$
  - **Example** for $P(w_1^L) =$ "French people work an average of 35 hours a week."
    - Maximize $P(w_1)$

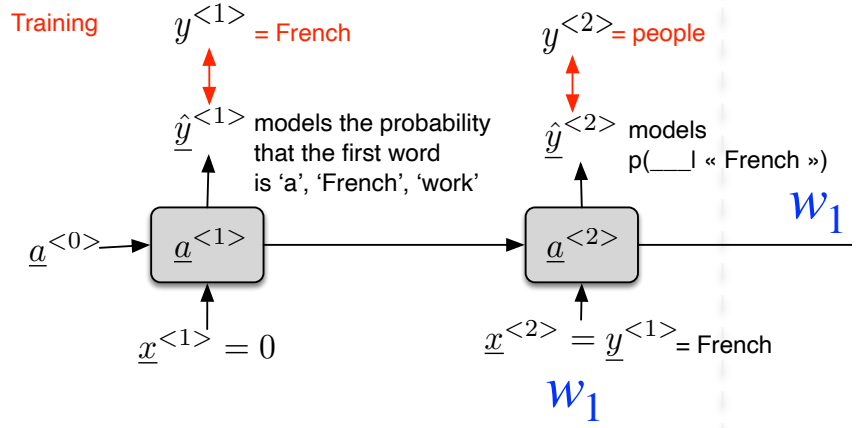
$$P(w_1^L) = P(w_1)$$



Language model: (1) Training using a RNN

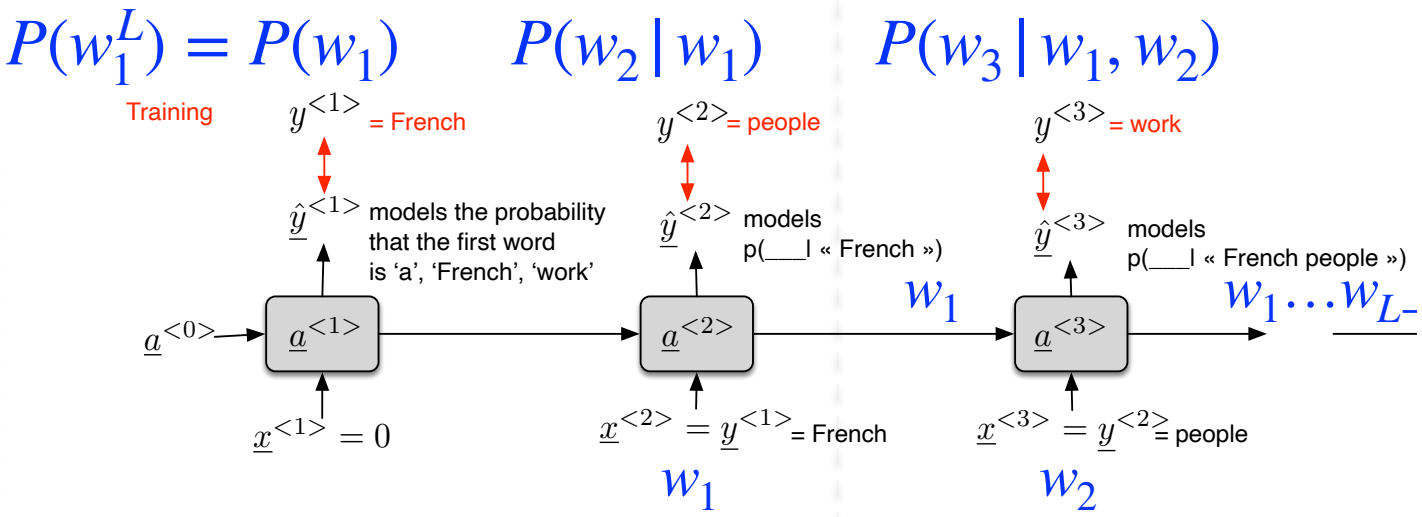
- **Objective**:
  - we want to train the parameters θ of the RNN model to maximise the probability of observing the sequence $P(w_1^L) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2))\ldots P(w_L \mid w_1 \ldots w_{L-1})$
  - **Example** for $P(w_1^L) =$ "French people work an average of 35 hours a week."
    - Maximize $P(w_1)$
    - Maximize $P(w_2 \mid w_1)$

$P(w_1^L) = P(w_1)$ $P(w_2 \mid w_1)$



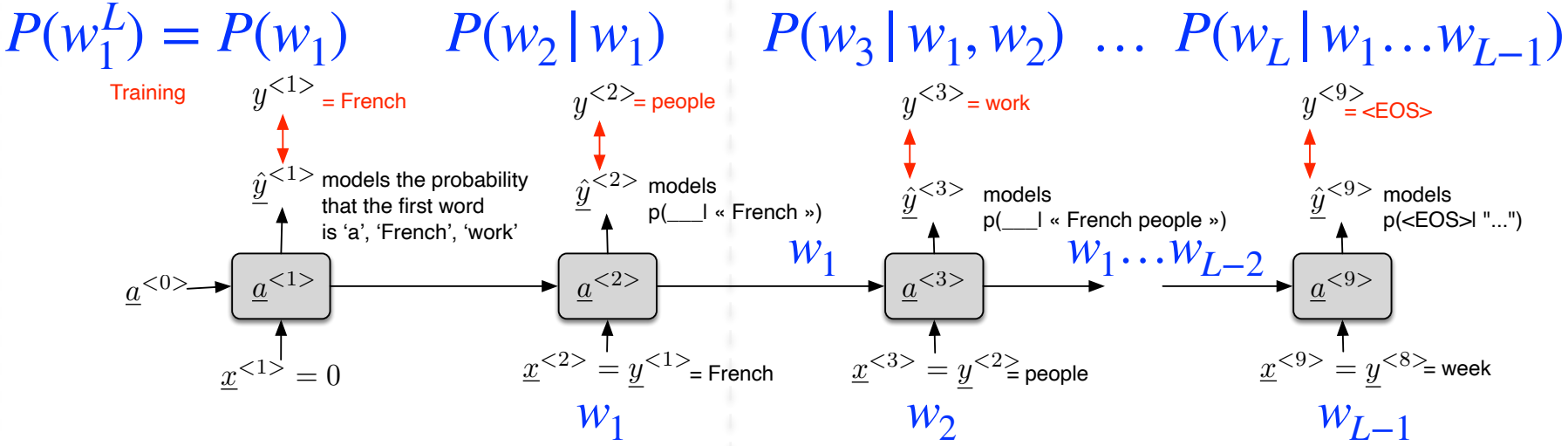
Language model: (1) Training using a RNN

- **Objective**:
  - we want to train the parameters θ of the RNN model to maximise the probability of observing the sequence $P(w_1^L) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2))\ldots P(w_L \mid w_1 \ldots w_{L-1})$
  - **Example** for $P(w_1^L)$ = "French people work an average of 35 hours a week."
    - Maximize $P(w_1)$
    - Maximize $P(w_2 \mid w_1)$
    - Maximise $P(w_3 \mid w_1, w_2)$



Language model: (1) Training using a RNN

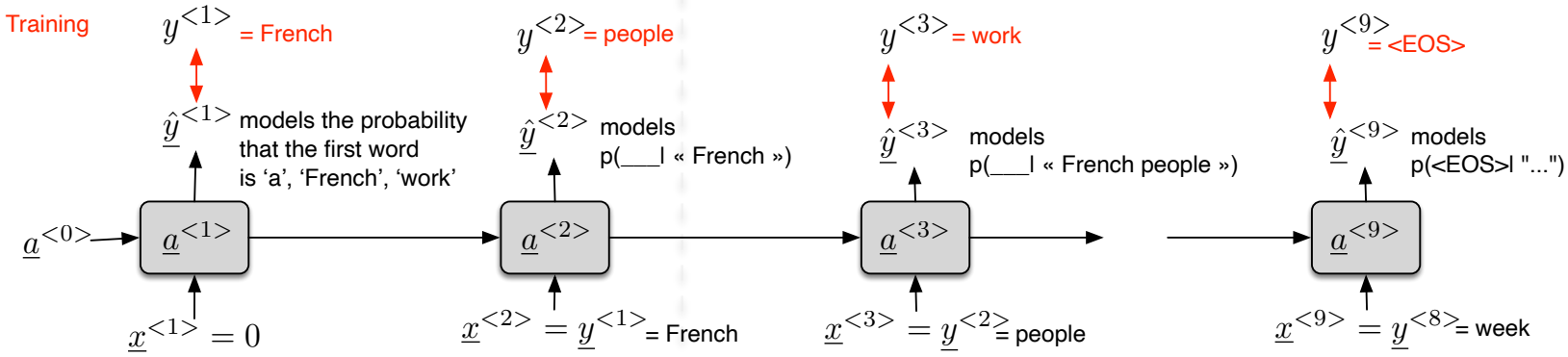
- **Objective**:
  - we want to train the parameters θ of the RNN model to maximise the probability of observing the sequence $P(w_1^L) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2))\ldots P(w_L \mid w_1 \ldots w_{L-1})$
  - **Example** for $P(w_1^L) =$ "French people work an average of 35 hours a week."



Language model: (1) Training using a RNN

– **Algorithm:**

- **given** the previous word as input $x^{\langle t \rangle}$ and the context $x^{\langle 1,\ldots t-1 \rangle}$ (given by the hidden cell $a^{\langle t \rangle}$)
  maximise the probability to observe the next word $x^{\langle t+1 \rangle}$ (we want the output $\hat{y}^{\langle t \rangle}$ to be $y^{\langle t \rangle} = x^{\langle t+1 \rangle}$)
- start with empty context $a^{\langle 0 \rangle} = 0$, empty input word $x^{\langle 1 \rangle} = 0$



– **Training:**

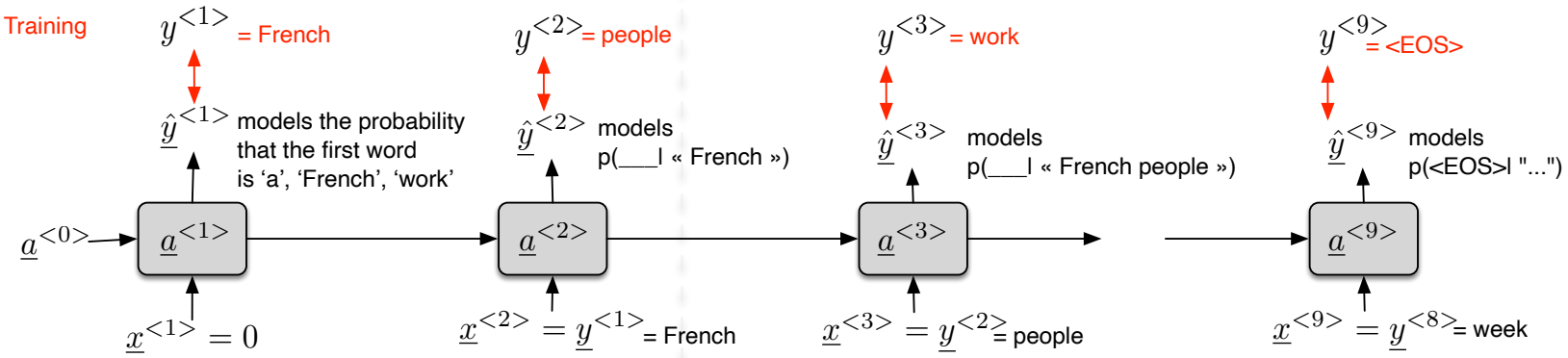
- $\hat{y}^{\langle t \rangle}$: output with a softmax with K neurons, gives the probability of each of the K words in the dictionary
- $y^{\langle t \rangle}$: ground-truth, one-hot-encoding with K classes

Minimise the cross-entropy loss $\mathcal{L} = \sum_t \mathcal{L}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle})$ with $\mathcal{L}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) = -\sum_{c=1}^{K} y_c^{\langle t \rangle} \log(\hat{y}_c^{\langle t \rangle})$

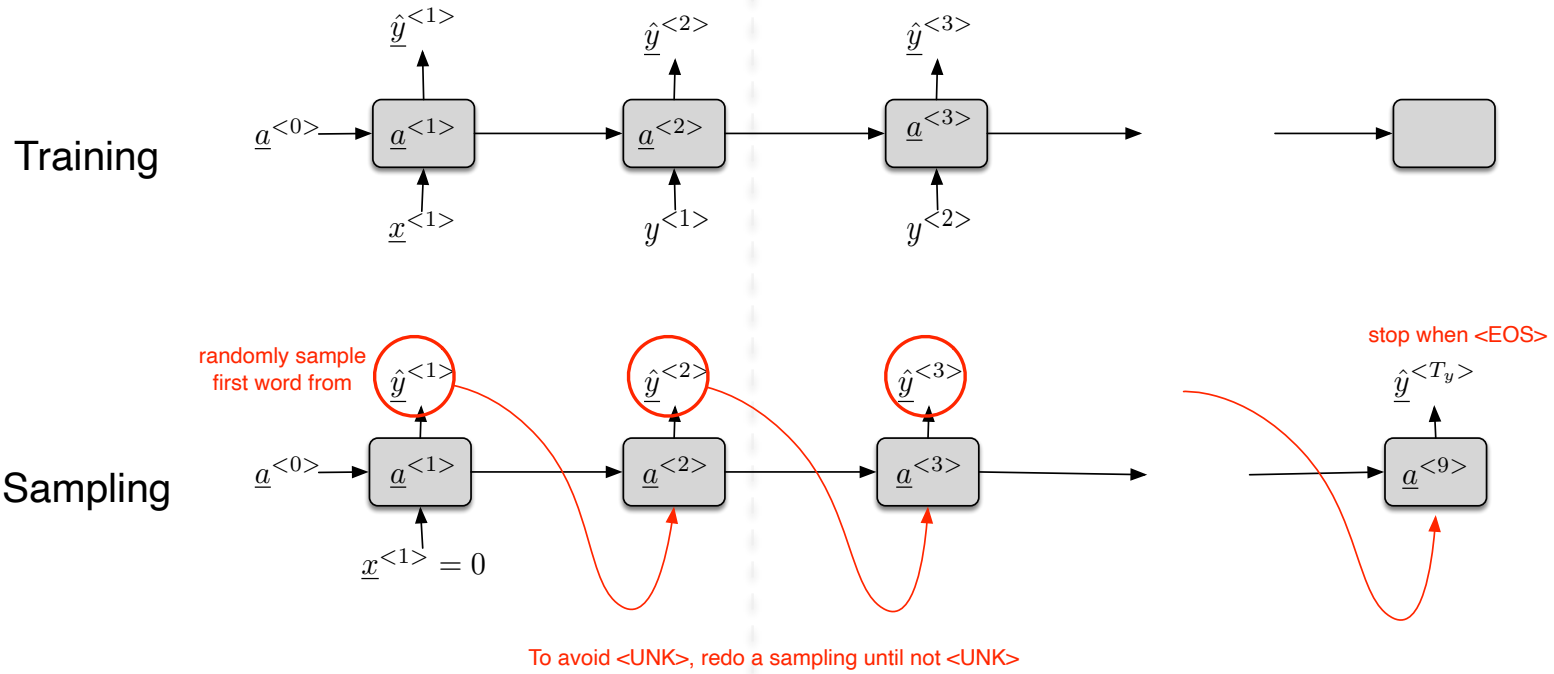
Language model: (2) Test using a RNN

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

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

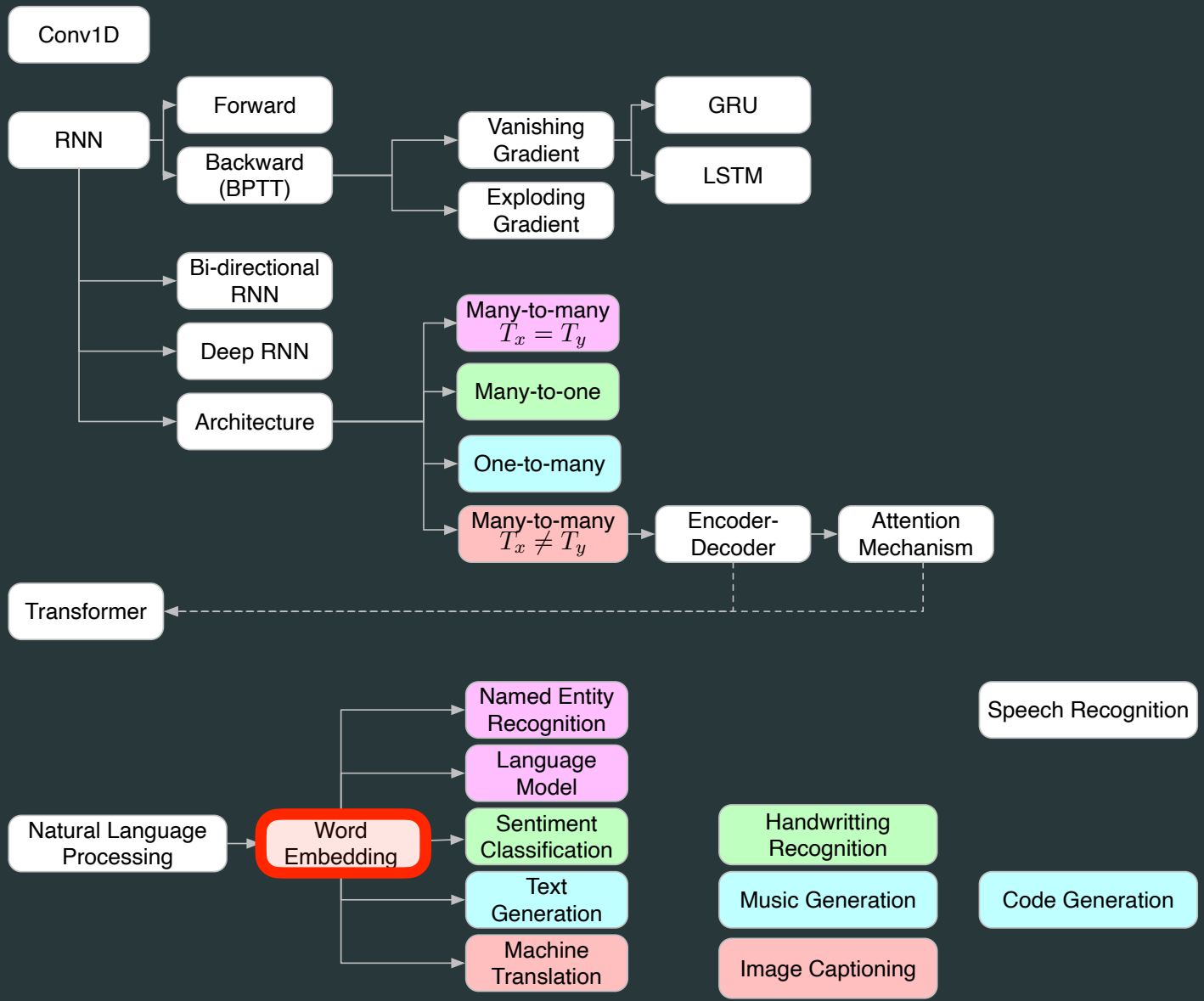


Language model: (3) Sampling novel sequences using RNN



- 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** ___?___ ⇒ ?

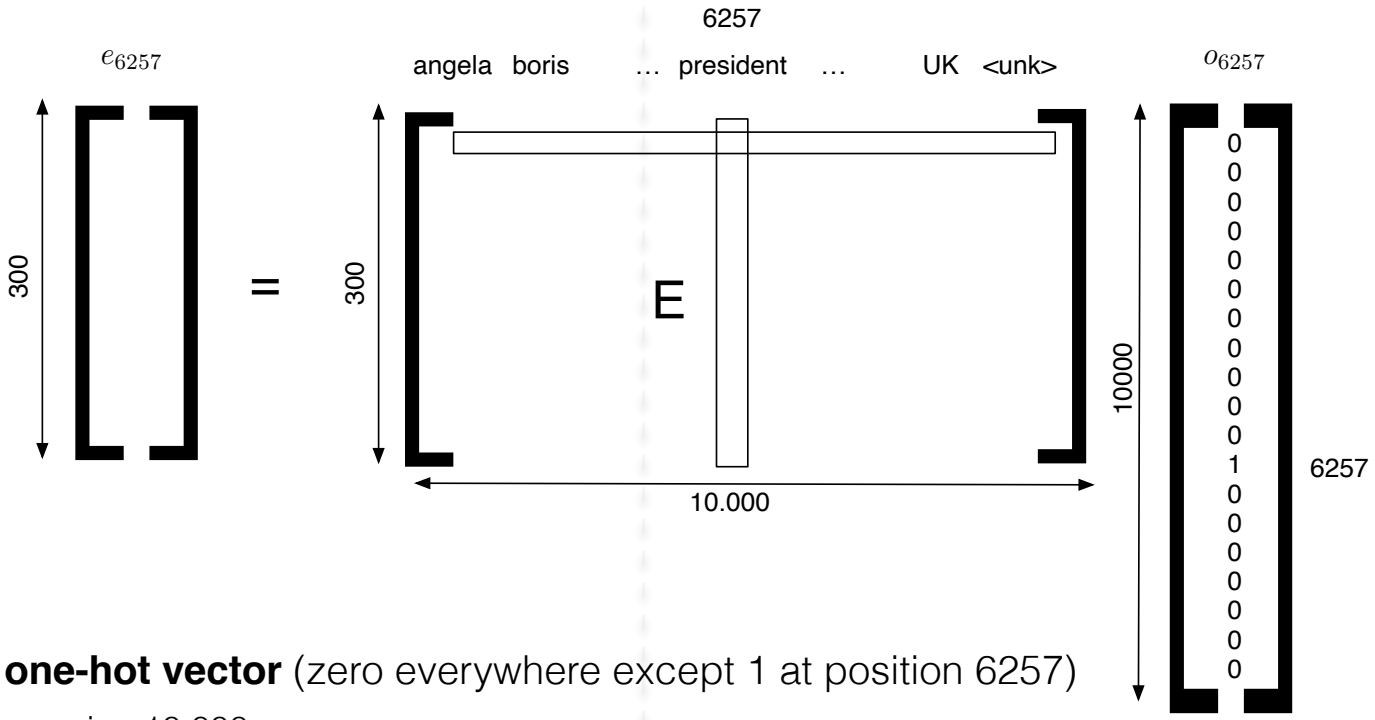
Word embedding

- 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)

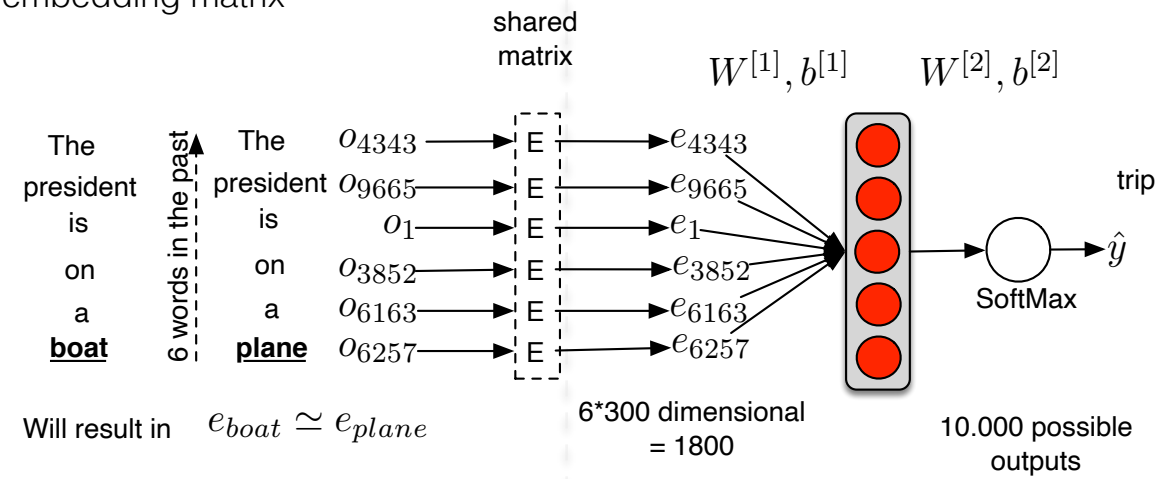
Word embedding/ Learning

- 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

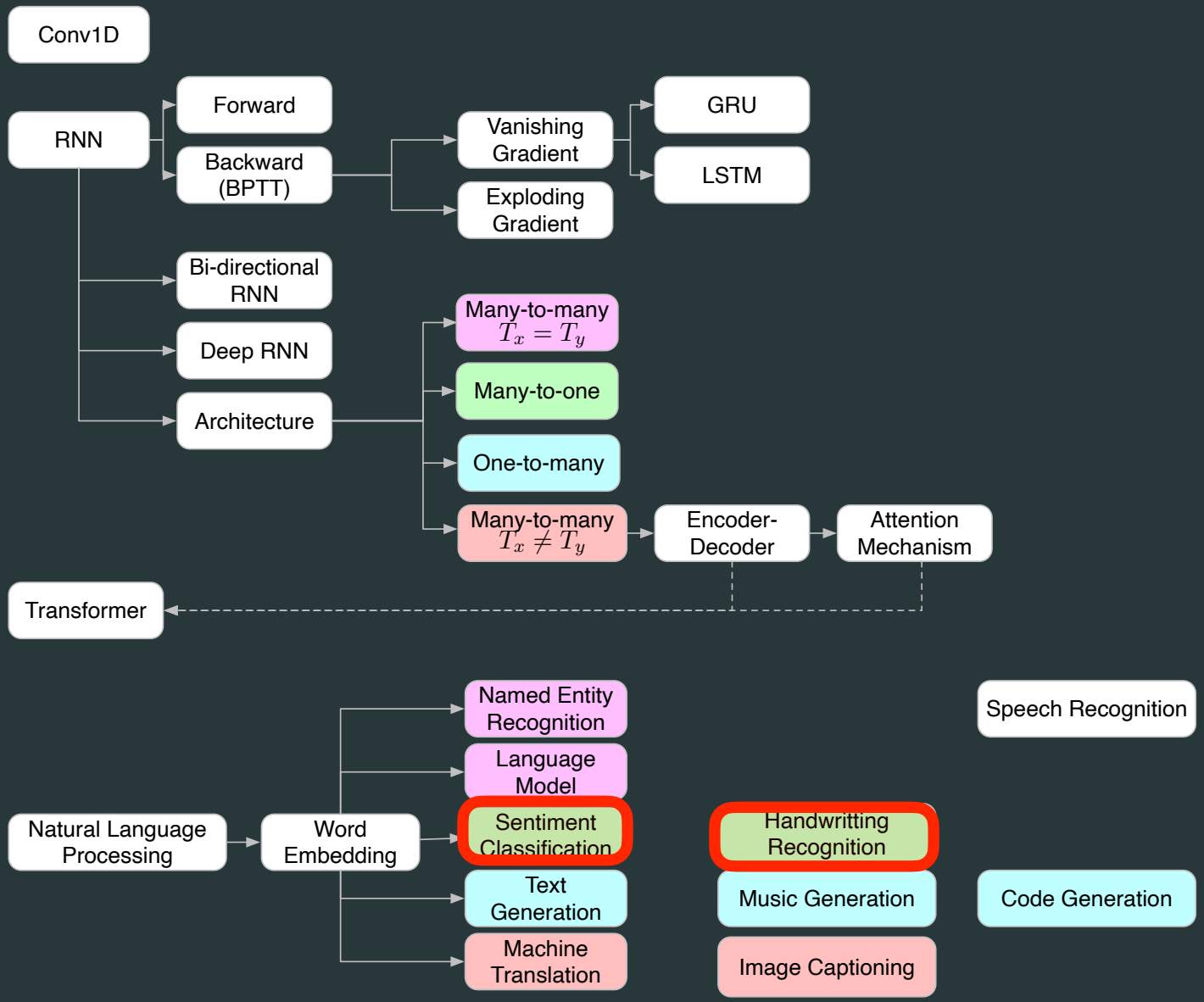


– **Example:**
  - "The president is on a boat $\underset{target}{\text{trip}}$ to the United States."

– **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.]

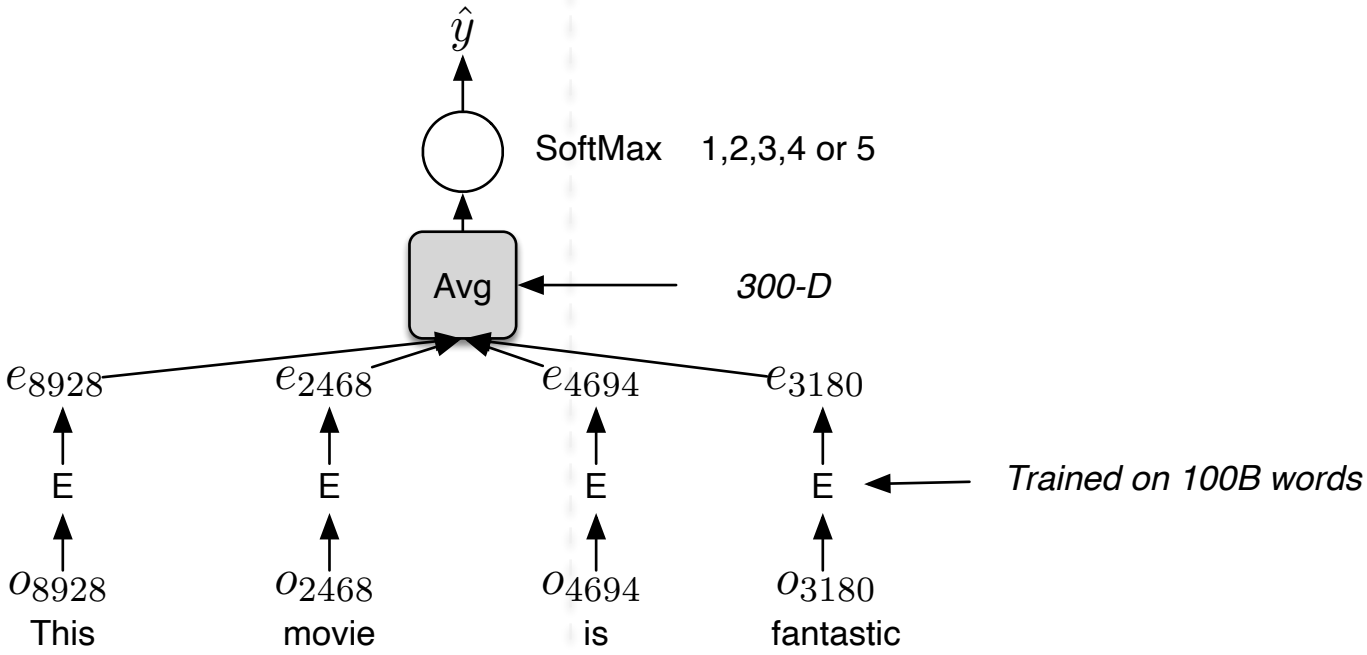
Application: Sentiment Classification



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.	

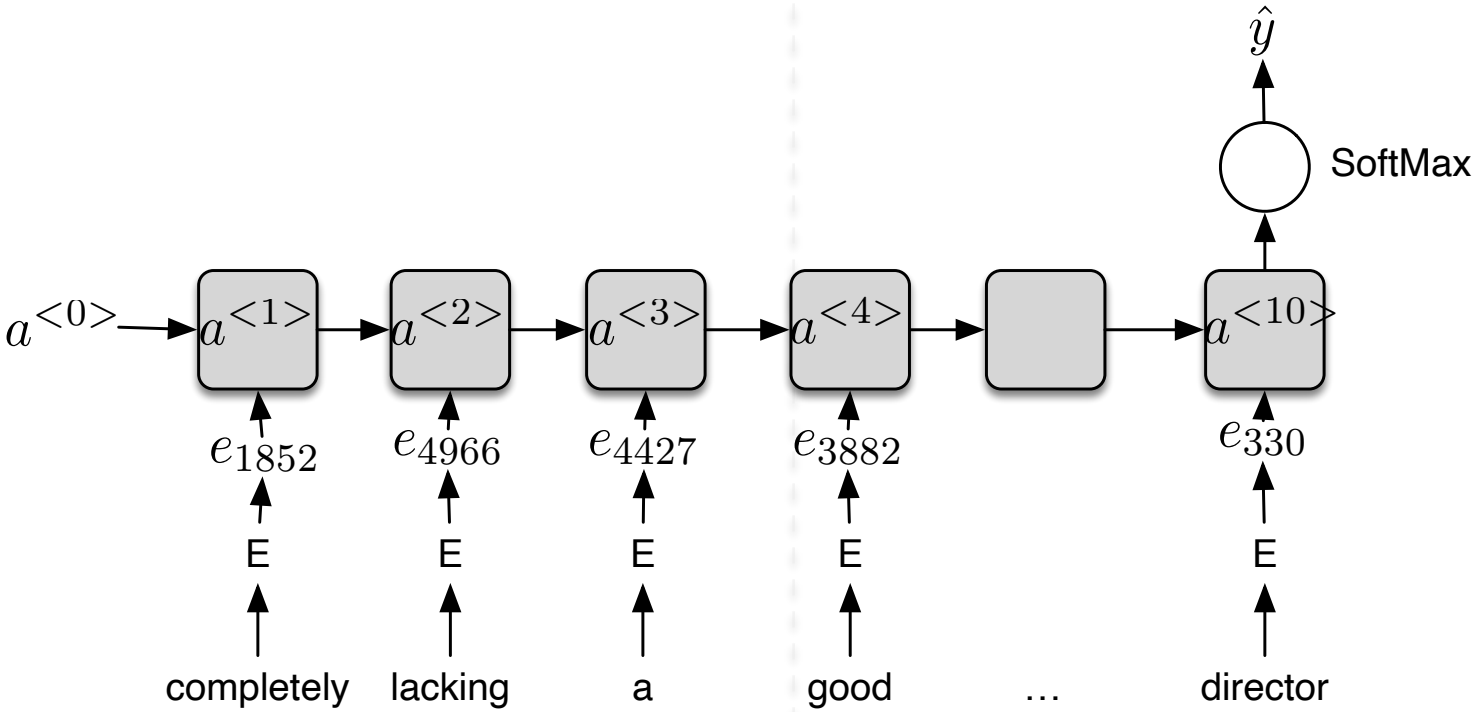
Application: Sentiment Classification / using a simple model



- Does not work for
  *"This movie was completely lacking 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

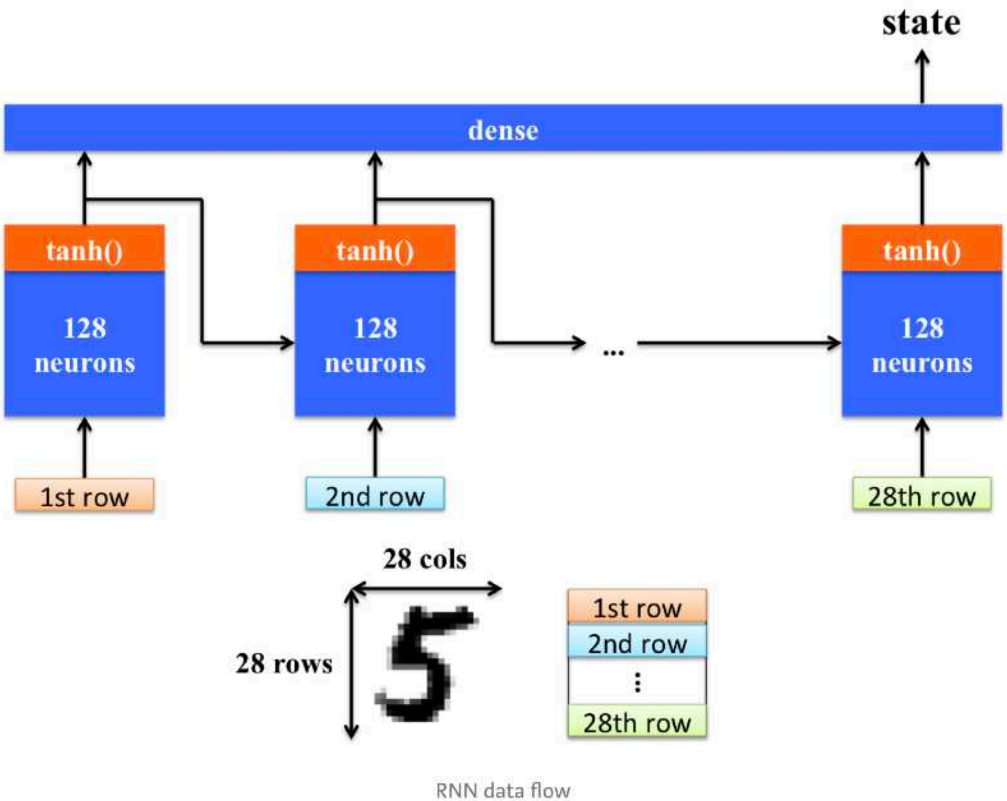
Application: Sentiment Classification / using a RNN

– **Many-to-one** architecture



Application: Handwritten Character Recognition

– **Many-to-one** architecture



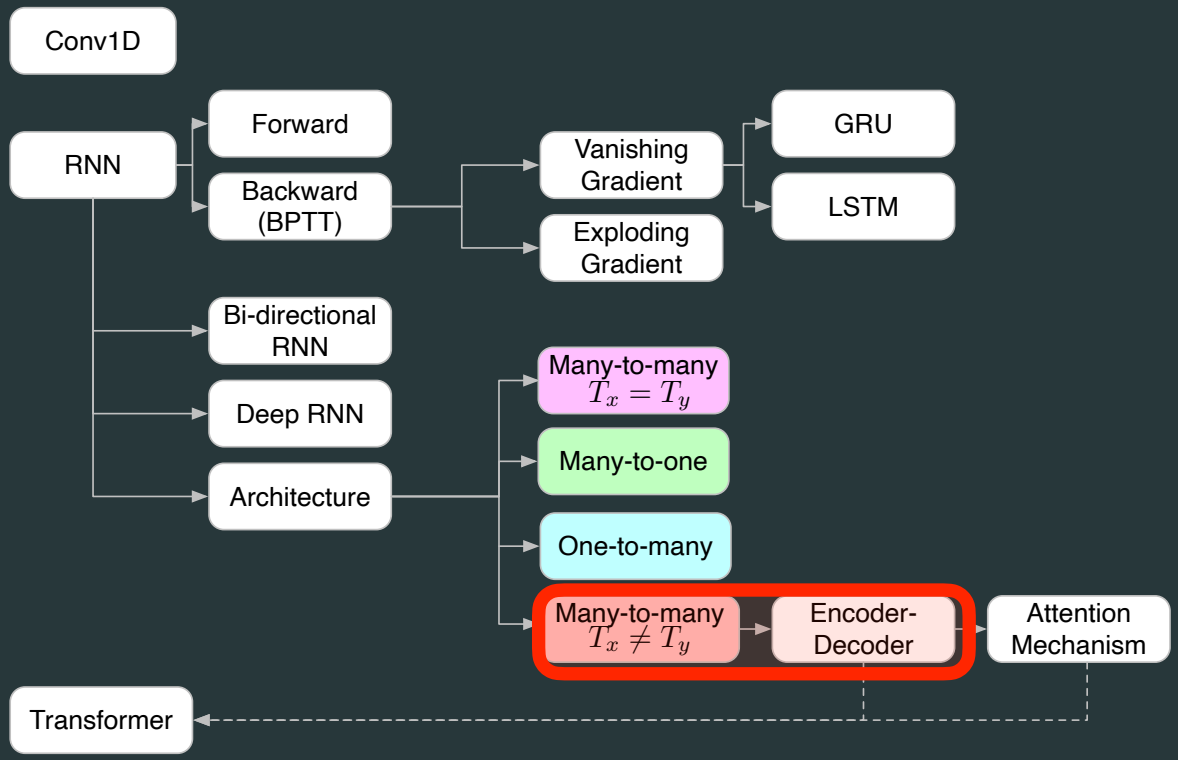
RNN data flow

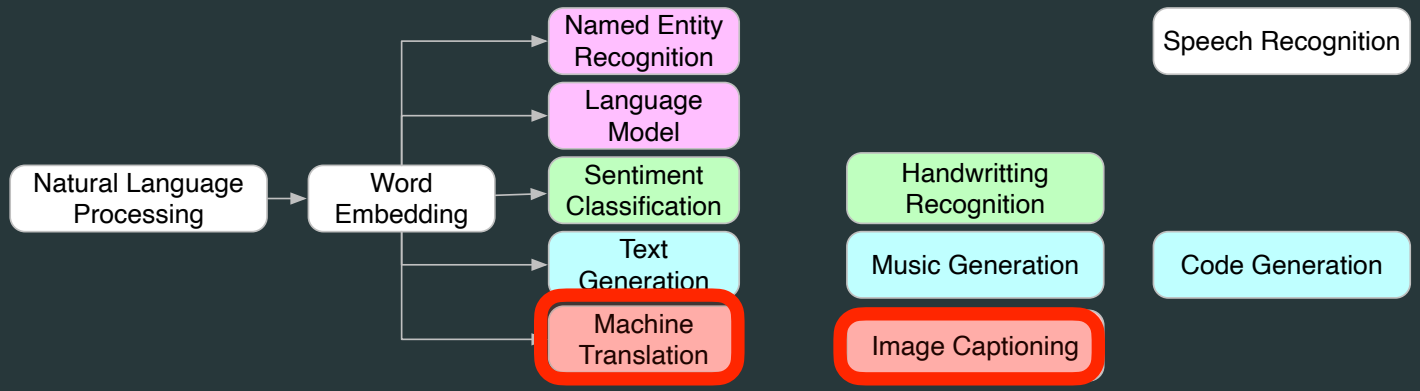
MNIST dataset

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"] LINK
[Graves et al. 2009 "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks"] LINK

Sequence to Sequence

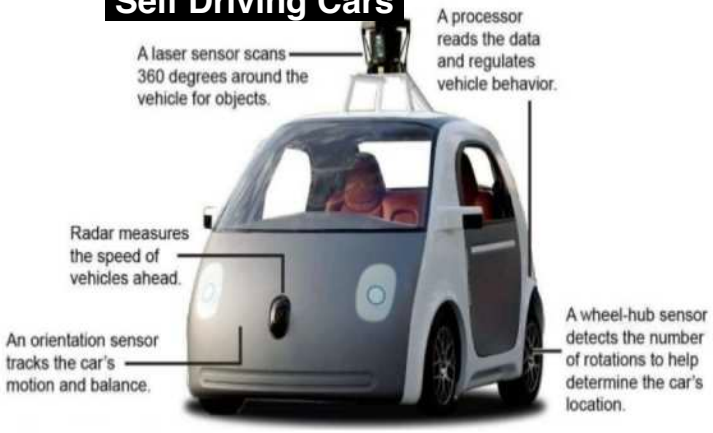




Applications

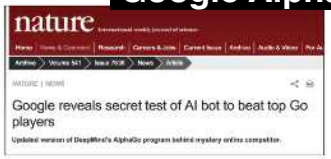
Automatic Speech Recognition

Self Driving Cars



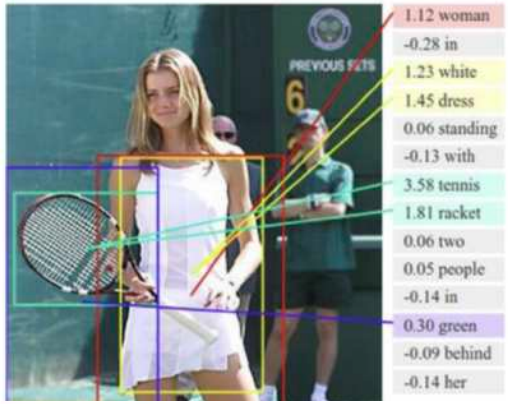
DALL-E

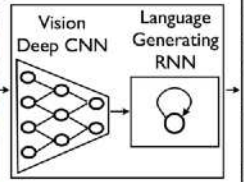
Google Alpha Go



DEEP LEARNING HAS MASTERED GO

Automatic picture captioning

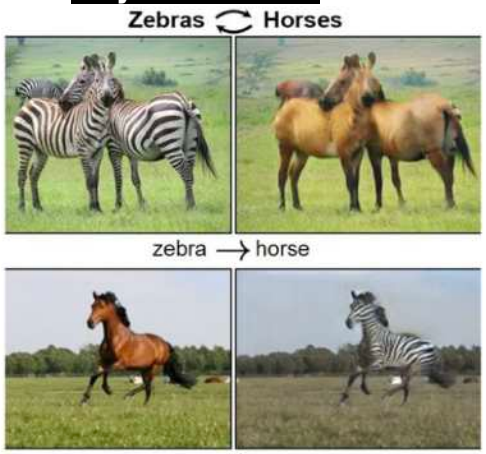




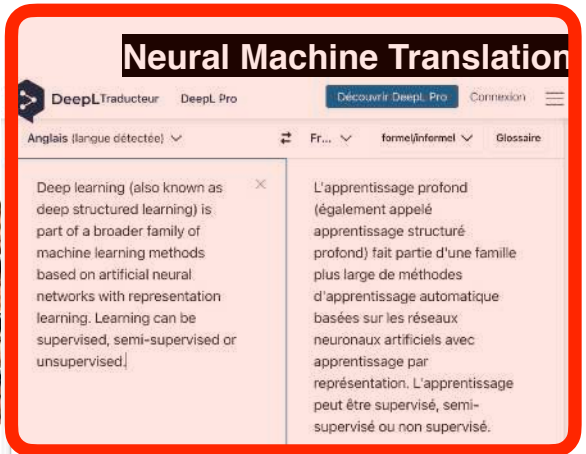
A group shopping at an outdoor market.

There are many vegetables at the fruit stand.

Style Transfer



Neural Machine Translation

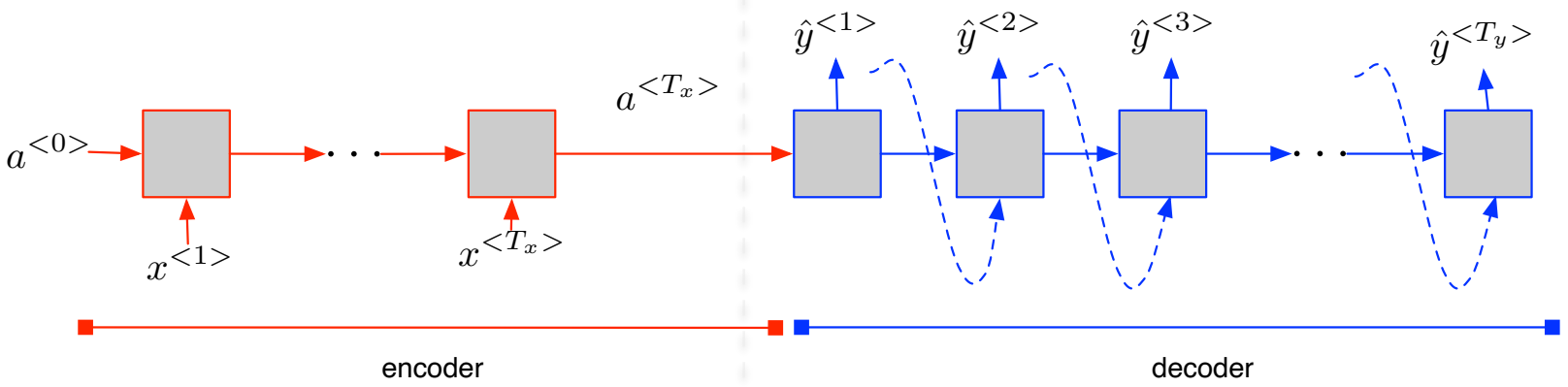


Sequence to Sequence

- 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

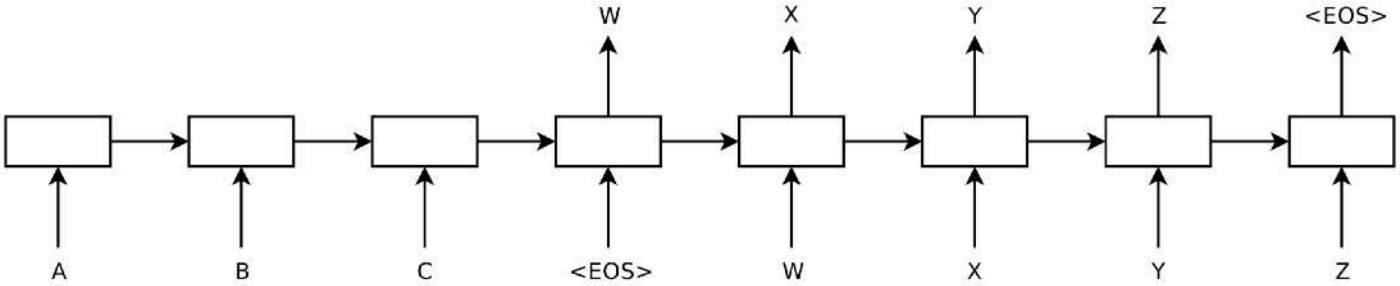
- Solution:
  - **Sequence to sequence** [Sutskever et al., 2014] or **Encoder-Decoder** [Cho et al., 2014] architecture



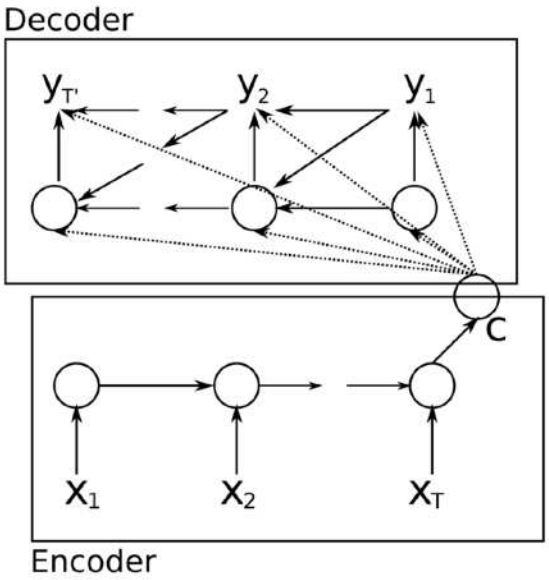
[Sutskever et al. 2014, "Sequence to sequence learning with neural networks"] LINK
[Cho et al. 2014 "Learning phrase representations using RNN encoder-decoder for statistical machine translation"] LINK

Introduction

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

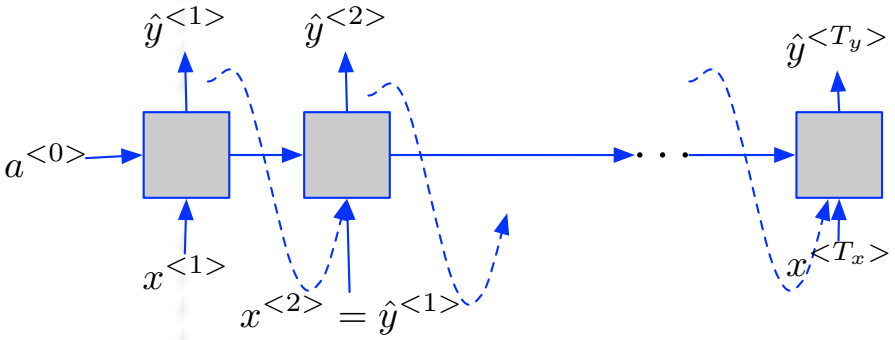


[Cho et al. 2014 "Learning phrase representations using RNN encoder-decoder for statistical machine translation"] LINK

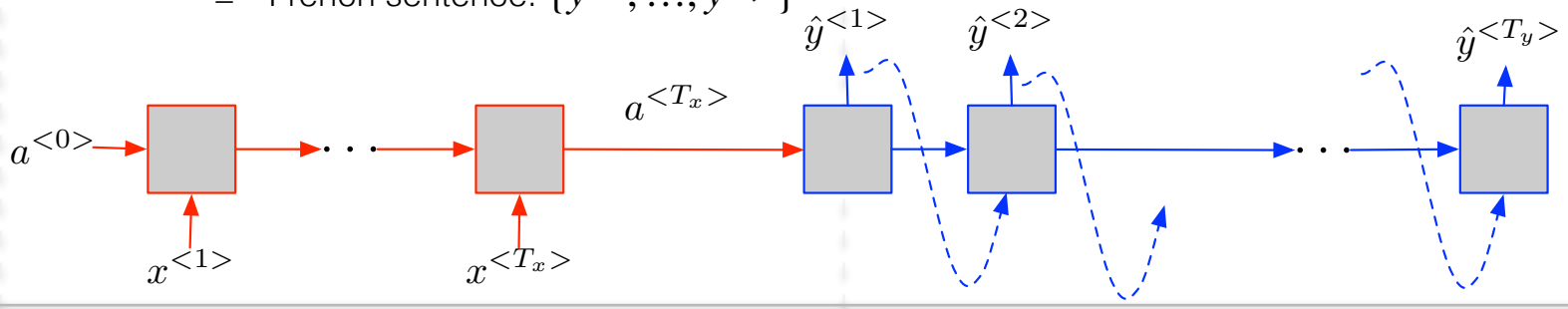


Machine Translation

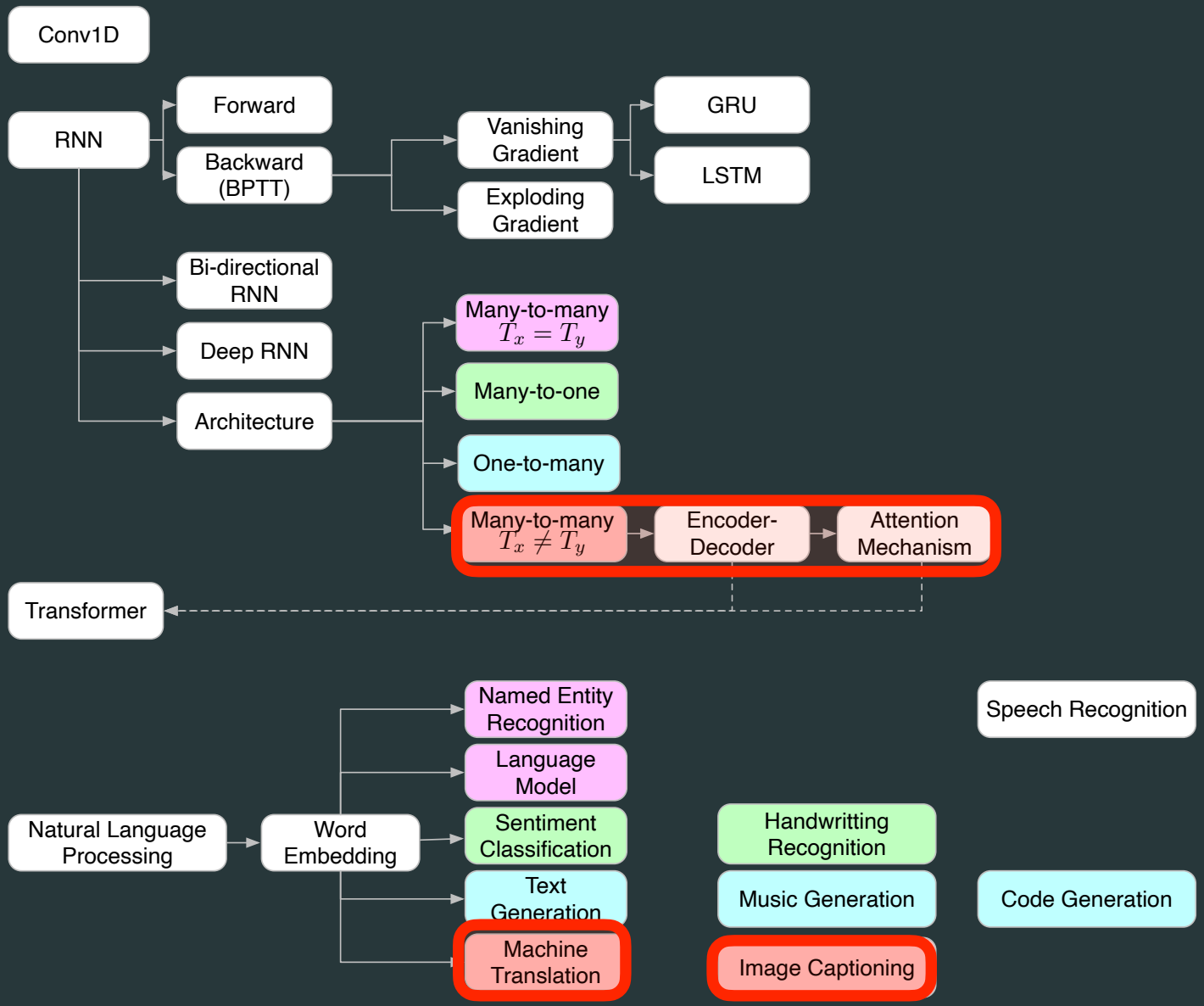
- **Language model**
  - $p(y^{\langle 1 \rangle}, y^{\langle 2 \rangle}, \ldots, y^{\langle T_y \rangle})$



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

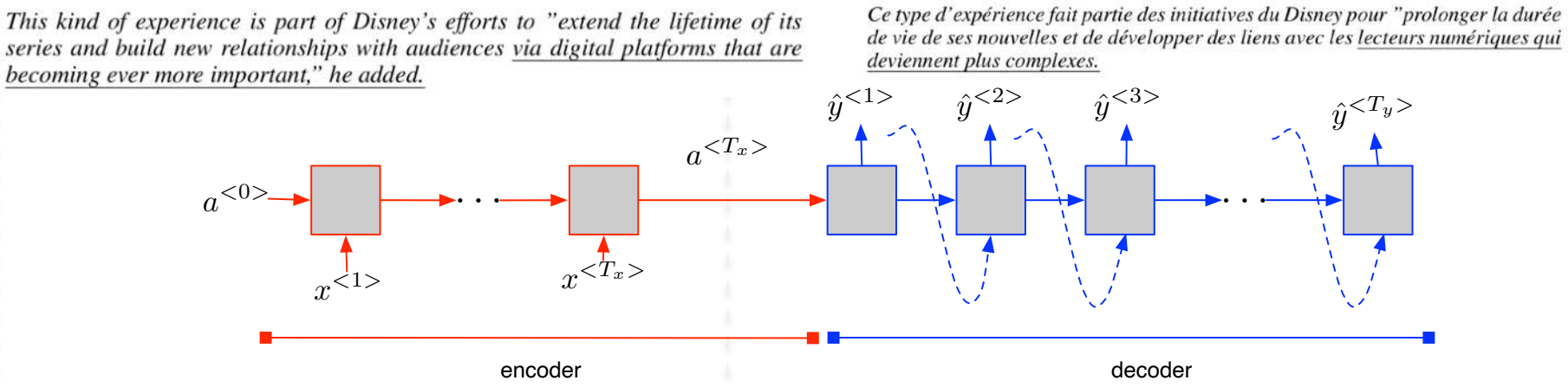


Attention model



Attention model

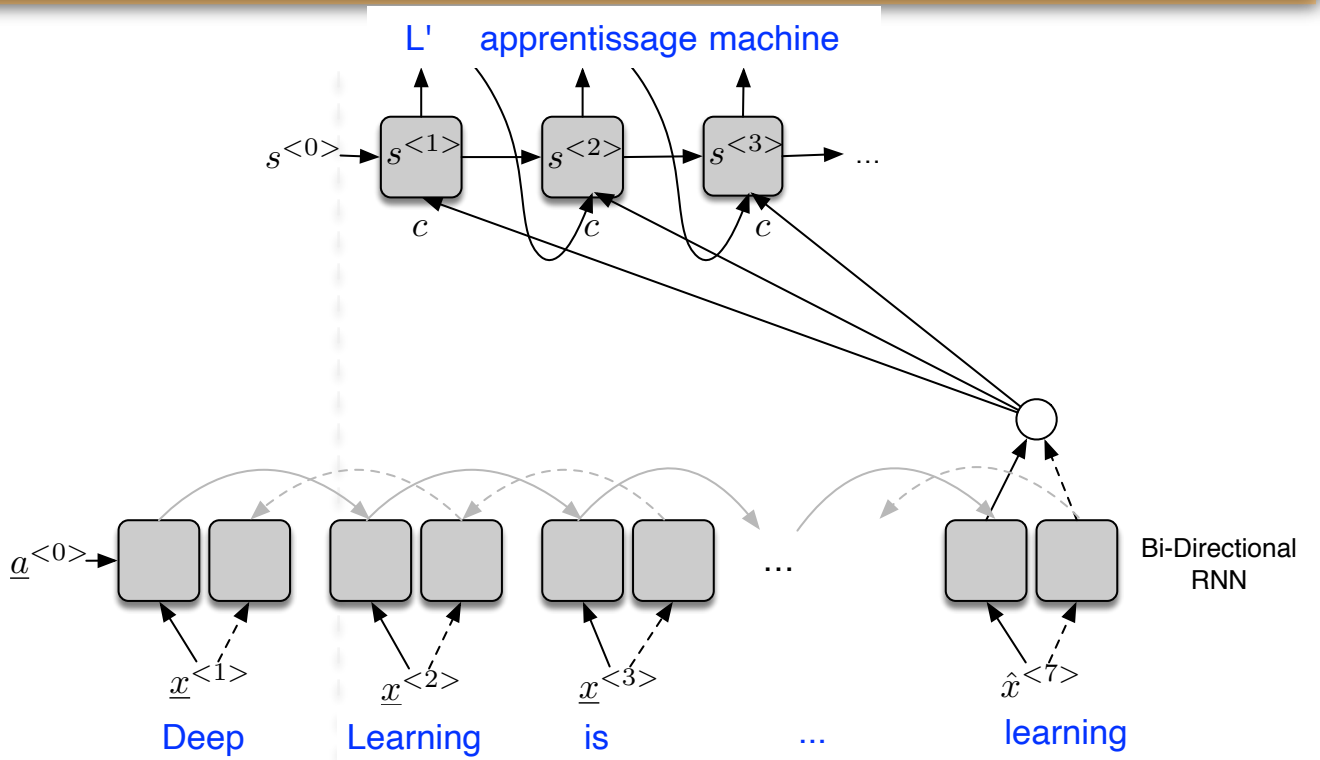
– **The problem of long sequences**



– 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", ICLR] LINK

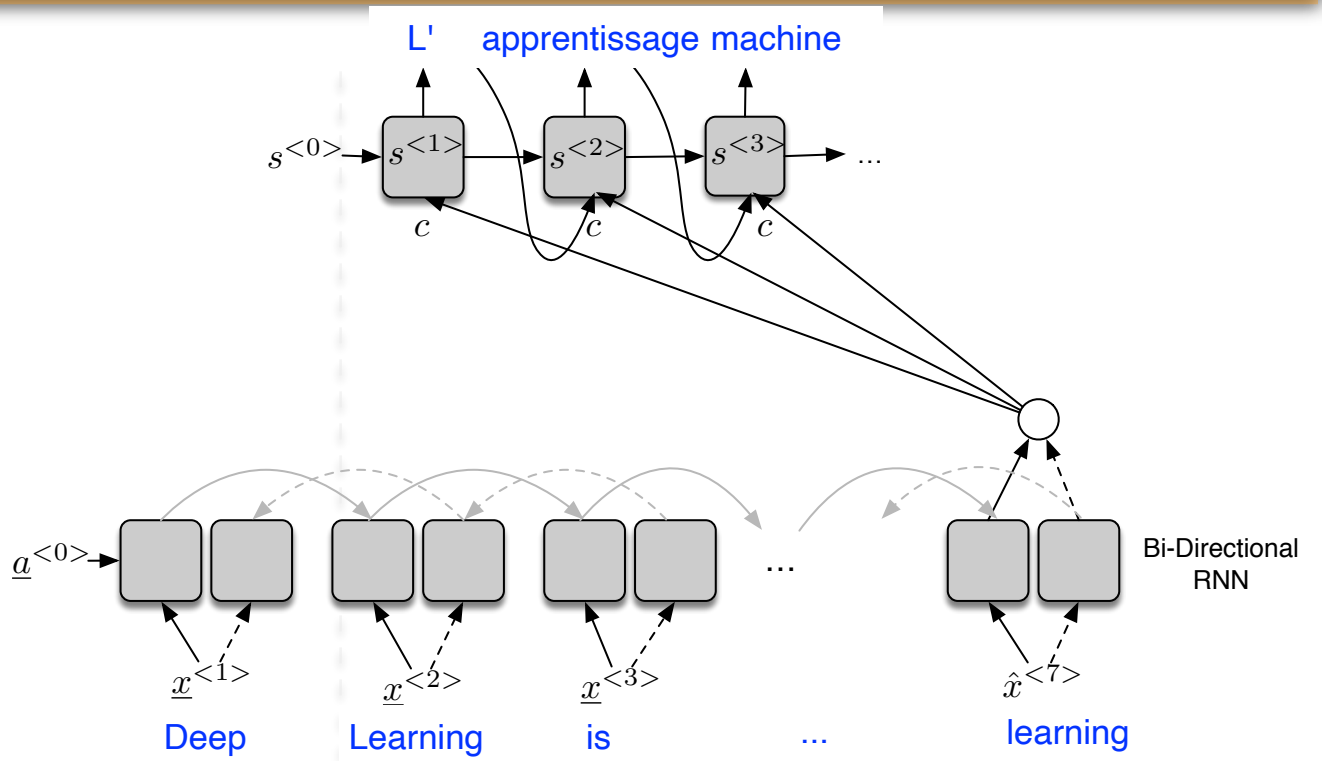
Attention model



– Notation:
  - $a^{\langle t \rangle}$: encoder hidden states
  - $s^{\langle \tau \rangle}$: decoder hidden states

Sequence to sequence

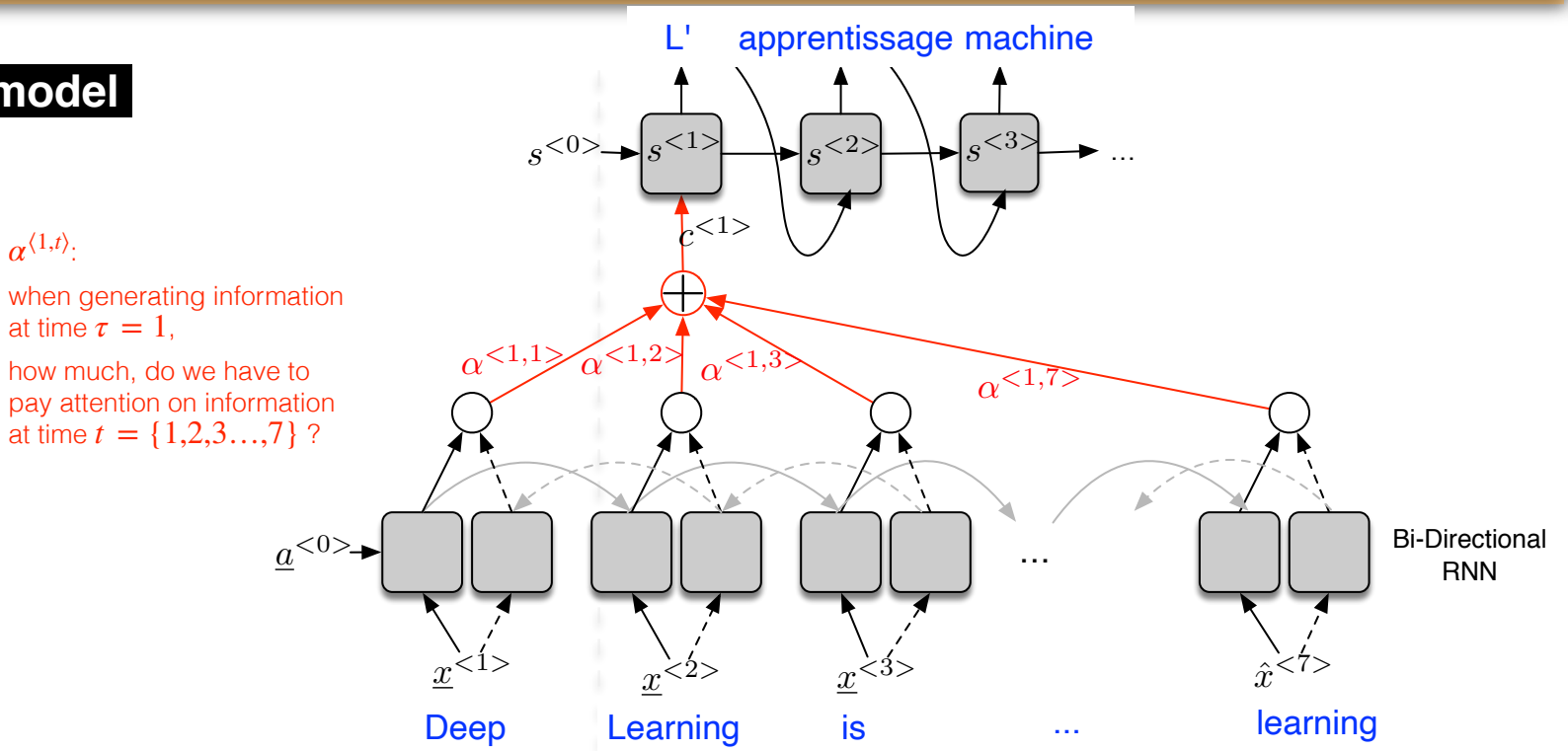
Attention model



- In **usual encoder-decoder:**
  - we use the same context for all decoding time τ: $c^{\langle \tau \rangle} = c$
  - the context c corresponds to the encoding hidden state at time T_x: $c = a^{\langle T_x \rangle}$

Sequence to sequence

Attention model



– **In attention model:**

- each decoding time τ has its own context: $c^{\langle\tau\rangle}$
- the context $c^{\langle\tau\rangle}$ is computed as a weighted sum of the encoding hidden states $a^{\langle t\rangle}$

– weights $\alpha^{\langle\tau,t\rangle}$:

- attention weights
- when generating information at time τ, how much, do we have to pay attention on information at time $t = \{1,2,3\ldots,7\}$

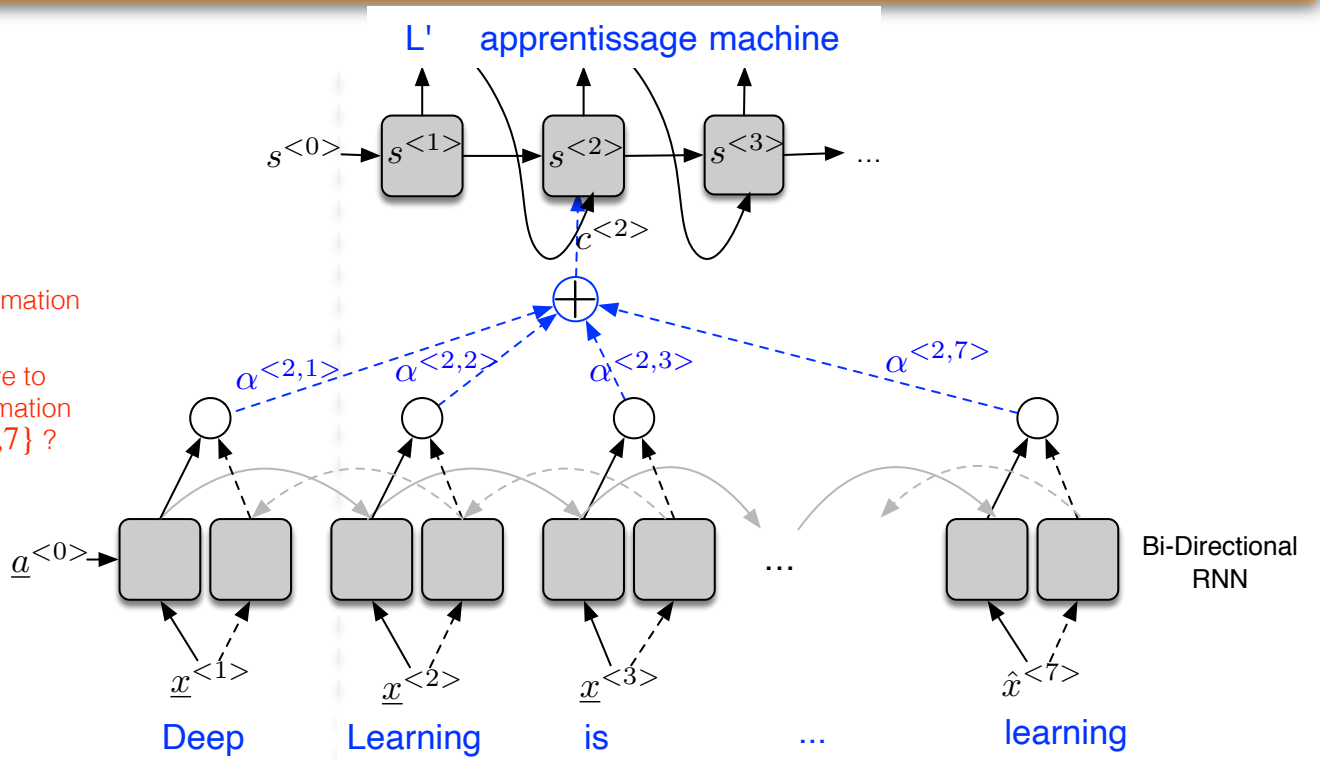
Sequence to sequence

Attention model

$\alpha^{\langle 2,t \rangle}$:

when generating information at time $\tau = 2$,

how much, do we have to pay attention on information at time $t = \{1,2,3\ldots,7\}$?



– **In attention model:**

- each decoding time τ has its own context: $c^{\langle \tau \rangle}$
- the context $c^{\langle \tau \rangle}$ is computed as a weighted sum of the encoding hidden states $a^{\langle t \rangle}$

– weights $\alpha^{\langle \tau,t \rangle}$:

- attention weights
- when generating information at time τ, how much, do we have to pay attention on information at time $t = \{1,2,3\ldots,7\}$

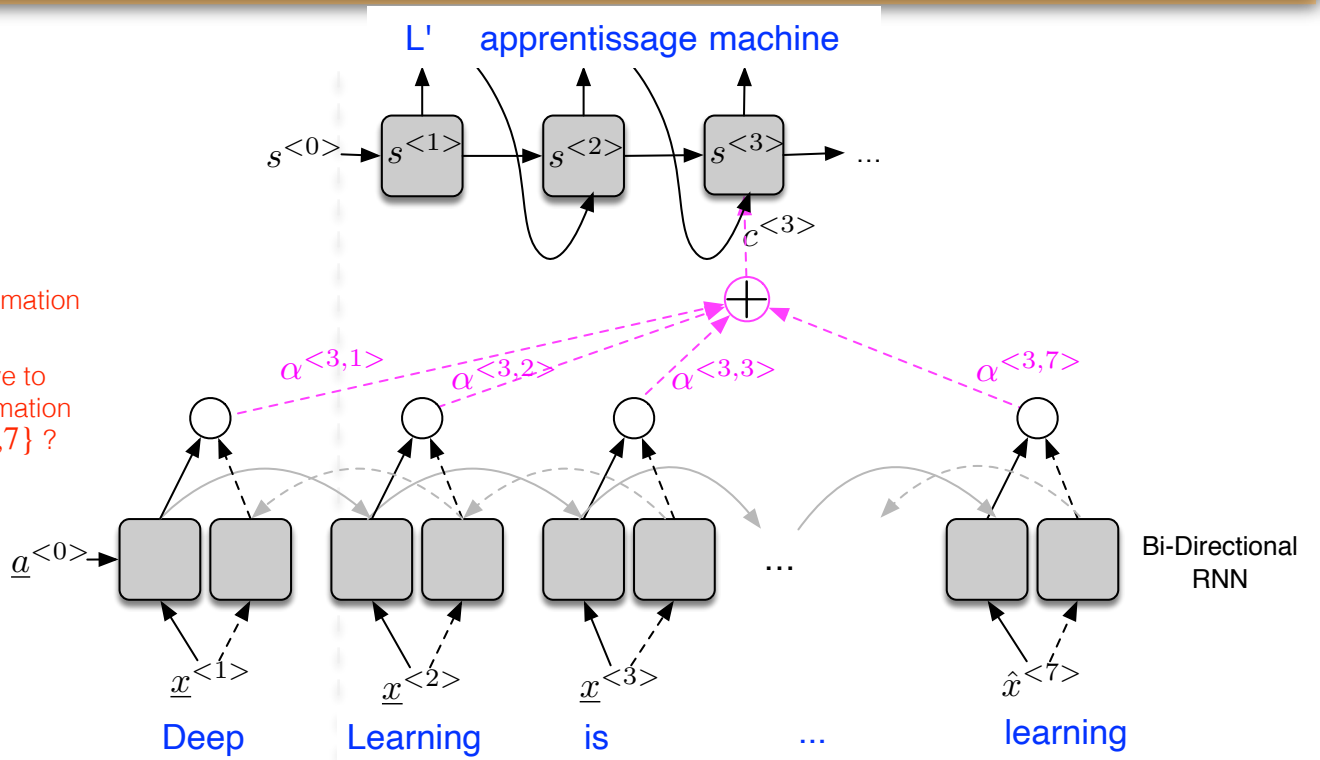
Sequence to sequence

Attention model

$\alpha^{\langle 3,t \rangle}$:

when generating information at time $\tau = 3$,

how much, do we have to pay attention on information at time $t = \{1,2,3\ldots,7\}$?



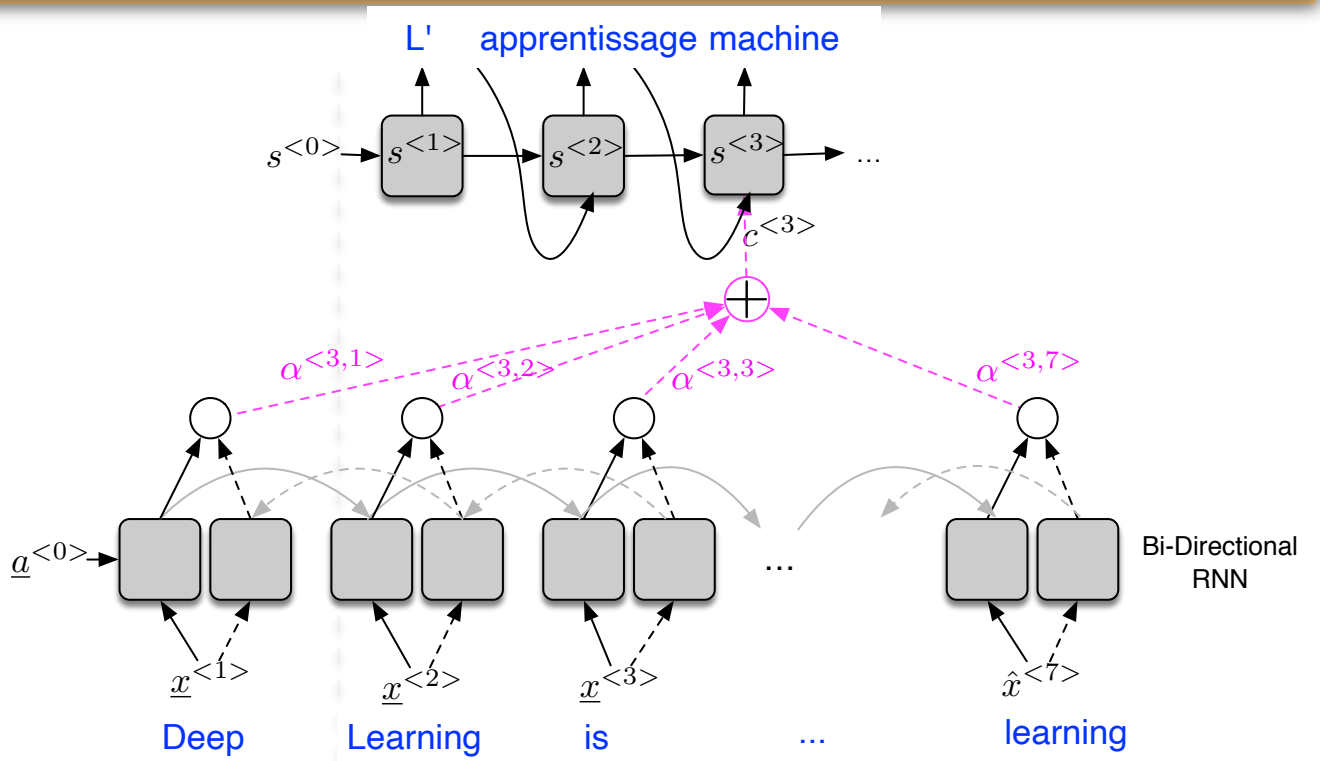
– **In attention model:**

- each decoding time τ has its own context: $c^{\langle \tau \rangle}$
- the context $c^{\langle \tau \rangle}$ is computed as a weighted sum of the encoding hidden states $a^{\langle t \rangle}$

– weights $\alpha^{\langle \tau,t \rangle}$:

- attention weights
- when generating information at time τ, how much, do we have to pay attention on information at time $t = \{1,2,3\ldots,7\}$

Attention model



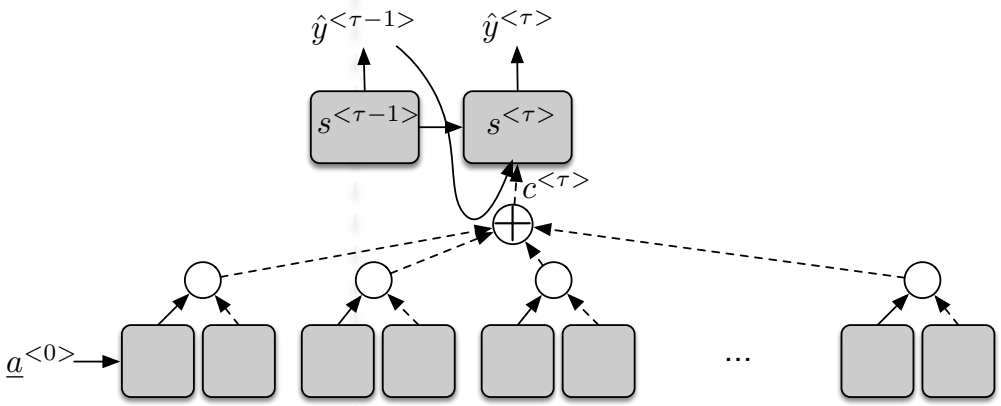
- Attention weights $\alpha^{\langle \tau=3,t \rangle}$ describe an "alignment" between information at
  - **encoding time** t:
    - computed using $\overrightarrow{a}^{\langle t \rangle}$ and $\overleftarrow{a}^{\langle t \rangle}$; we note $a^{\langle t \rangle} = [\overrightarrow{a}^{\langle t \rangle},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

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** $\alpha^{\langle \tau, t \rangle}$

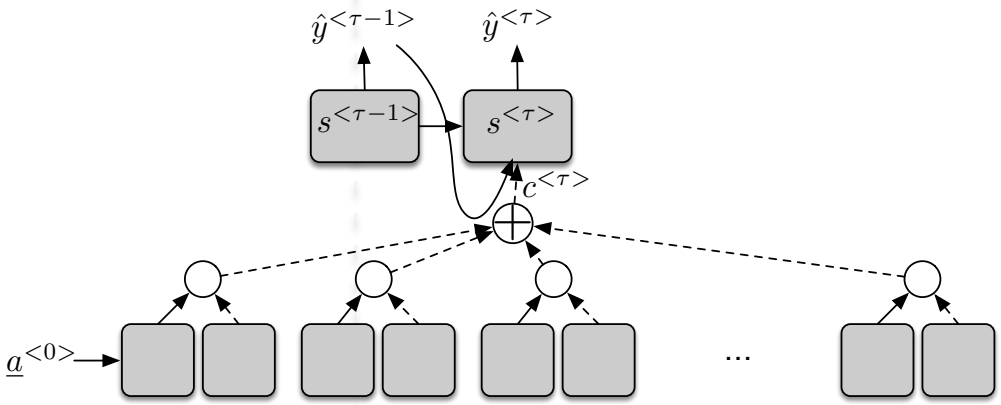
$$c^{\langle \tau \rangle} = \sum_t \alpha^{\langle \tau, t \rangle} a^{\langle t \rangle}$$



Attention model

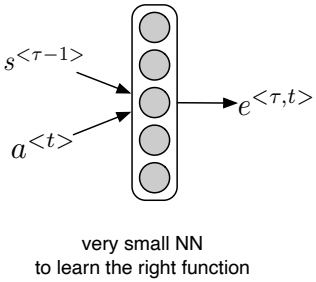
- The **attention weight** $\alpha^{\langle \tau,t \rangle}$
  - $\alpha^{\langle \tau,t \rangle}$= represents the amount of attention $y^{\langle \tau \rangle}$ should pay to $a^{\langle t \rangle}$
  - computed using a softmax on the **alignment model** $e^{\langle \tau,t \rangle}$

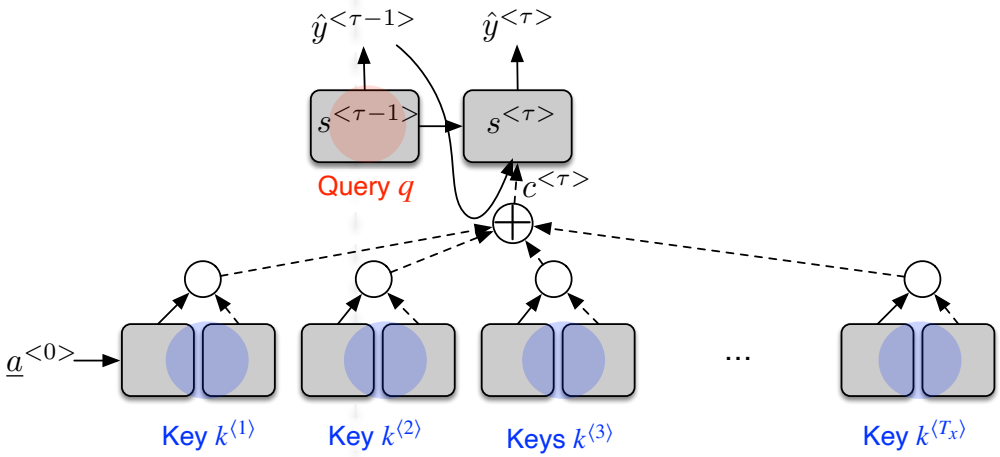
$$\alpha^{\langle \tau,t \rangle} = \frac{\exp(e^{\langle \tau,t \rangle})}{\sum_{t'=1}^{T_x} \exp(e^{\langle \tau,t' \rangle})} \text{ with } \sum_t \alpha^{\langle \tau,t \rangle} = 1$$



Attention model

- The **alignment model** $e^{\langle \tau,t \rangle}$ scores how well the inputs around position t match the output at position τ
  - is computed using two inputs
    - **query**: the RNN hidden state $s^{\langle \tau-1 \rangle}$ of the output sequence (just before emitting $y^{\langle \tau \rangle}$)
    - **keys**: the RNN hidden states $a^{\langle t \rangle}$ of the input sentence.
  - $e^{\langle \tau,t \rangle}$ is computed using $f(s^{\langle \tau-1 \rangle}, a^{\langle t \rangle})$
    - can be a feedforward neural network (jointly trained with all the other components)
    - can be a dot-product: $e^{\langle t \rangle} = q^T k^{\langle t \rangle}$
    - can be multiplicative attention: $e^{\langle t \rangle} = q^T W k^{\langle t \rangle}$





https://lilianweng.github.io/posts/2018-06-24-attention/

Attention model

– Different **alignment models**

Here
$\mathbf{s}_t$ corresponds to $\mathbf{s}^{\langle \tau \rangle}$
$\mathbf{h}_i$ corresponds to $\mathbf{a}^{\langle t \rangle}$

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

https://lilianweng.github.io/posts/2018-06-24-attention/

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)

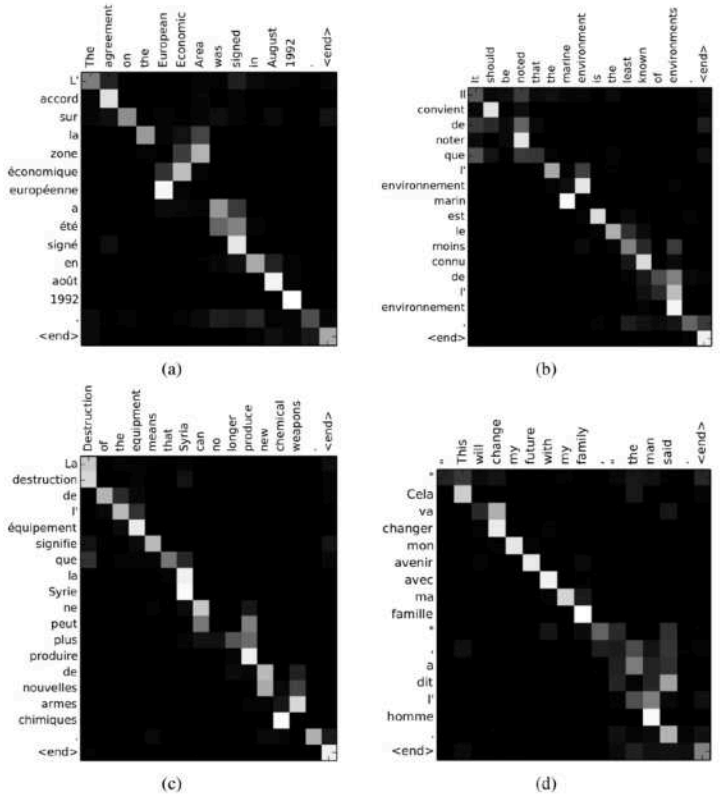
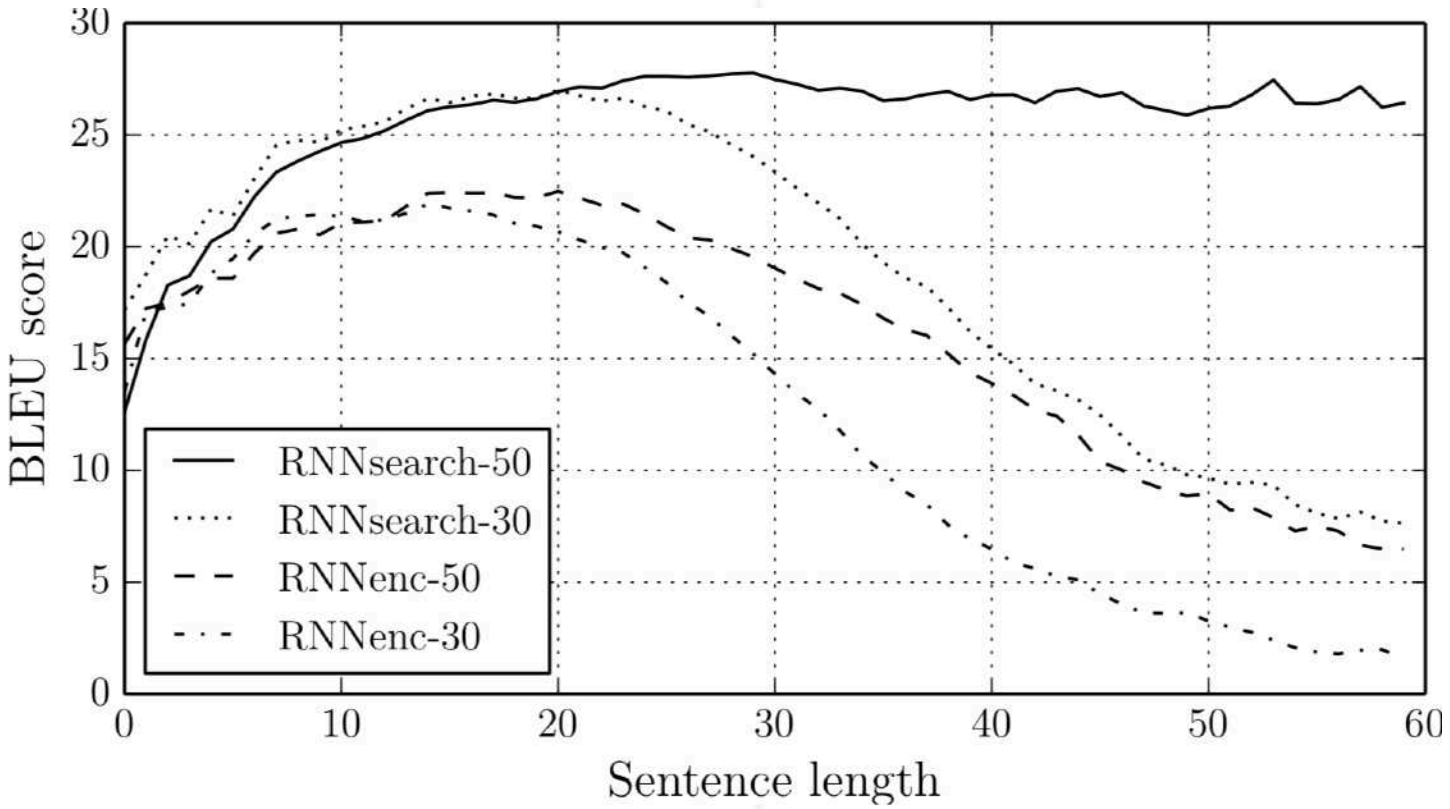


Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word (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

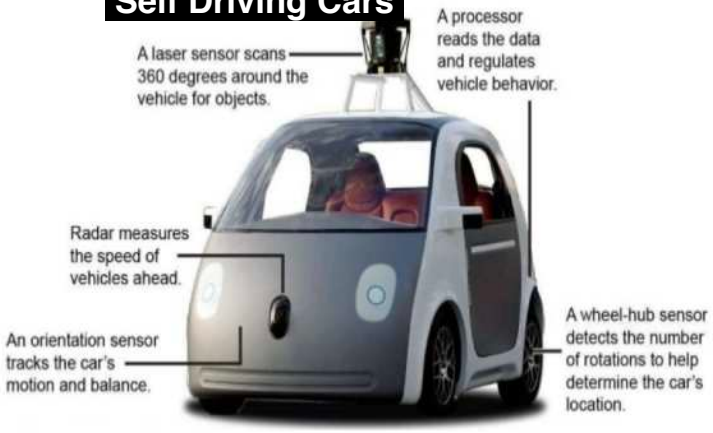


[Bahdanau et al. 2015 "Neural machine translation by jointly learning to align and translate"]

Applications

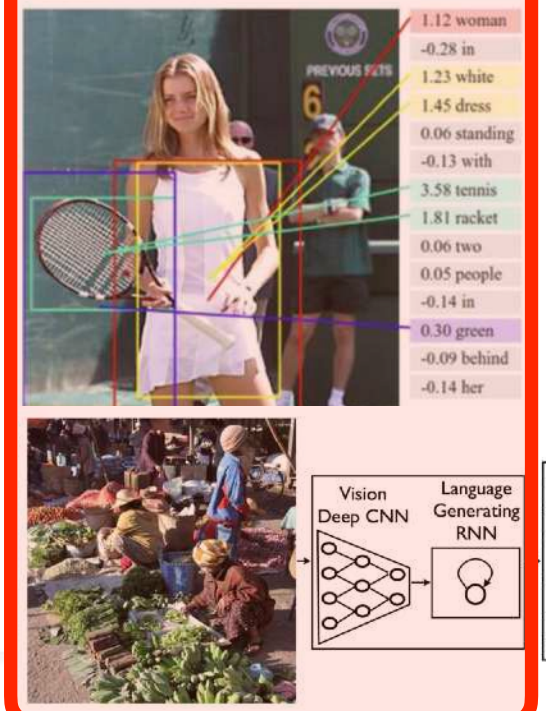
Automatic Speech Recognition

Self Driving Cars

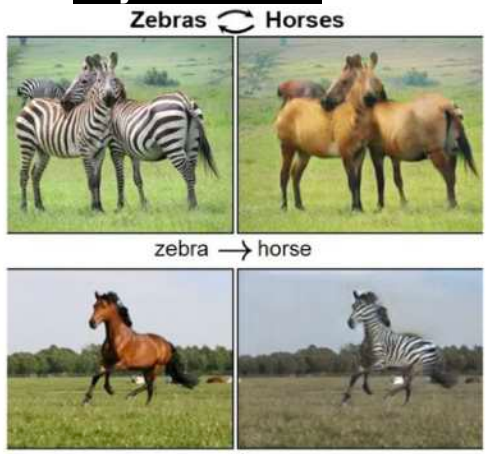


DALL-E

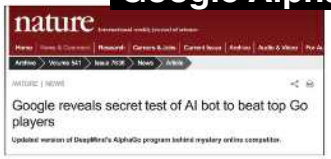
Automatic picture captioning



Style Transfer

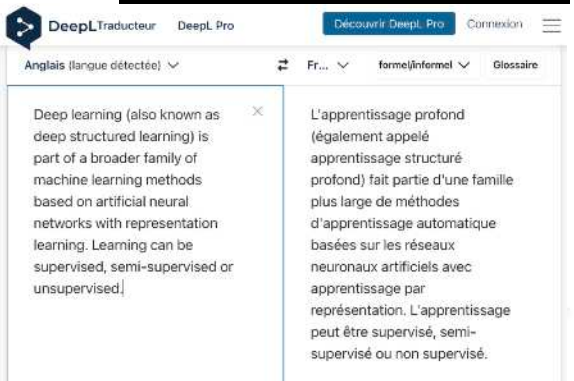


Google Alpha Go



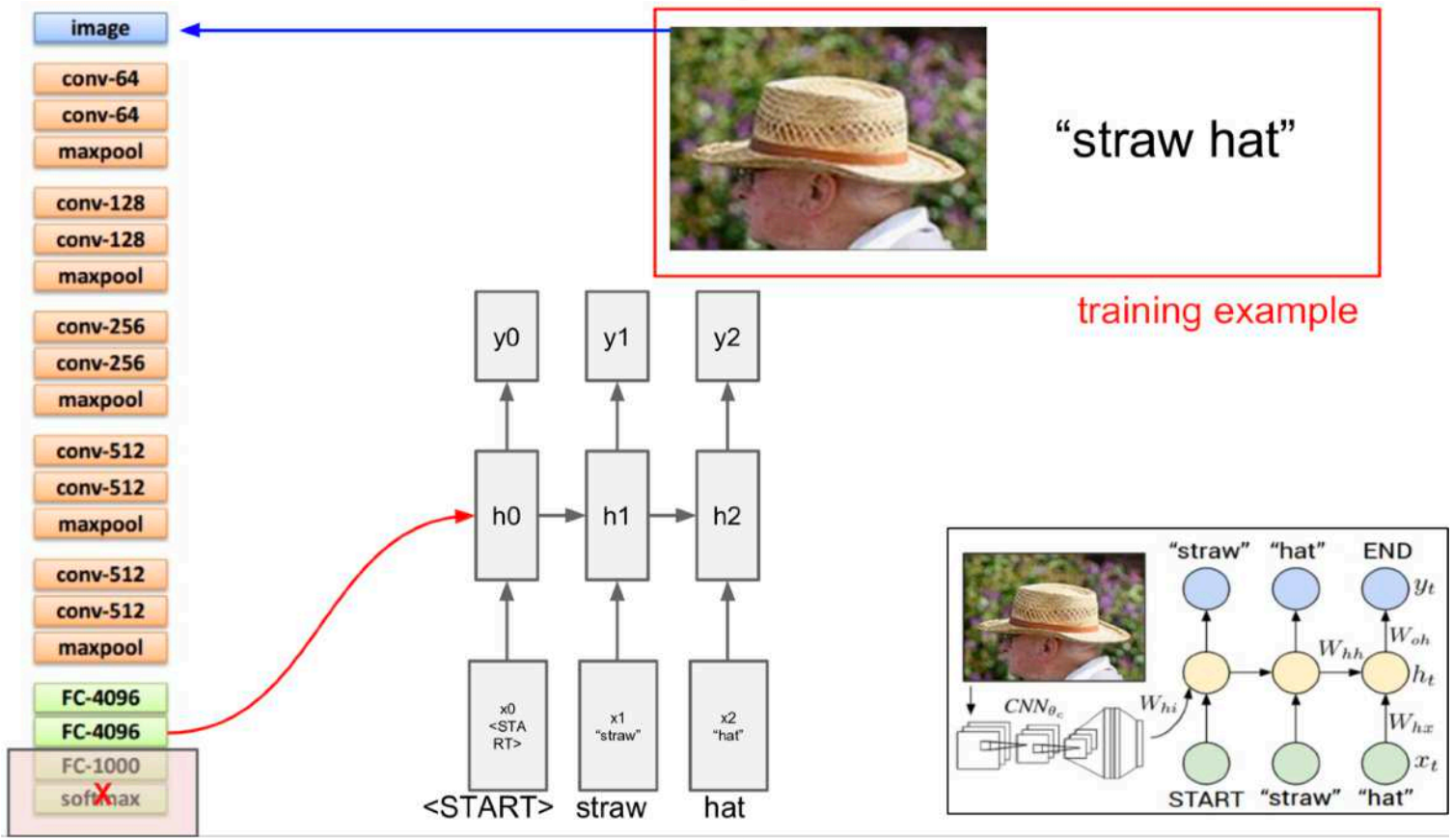
DEEP LEARNING HAS MASTERED GO

Neural Machine Translation



Sequence to sequence

Application: Image captioning

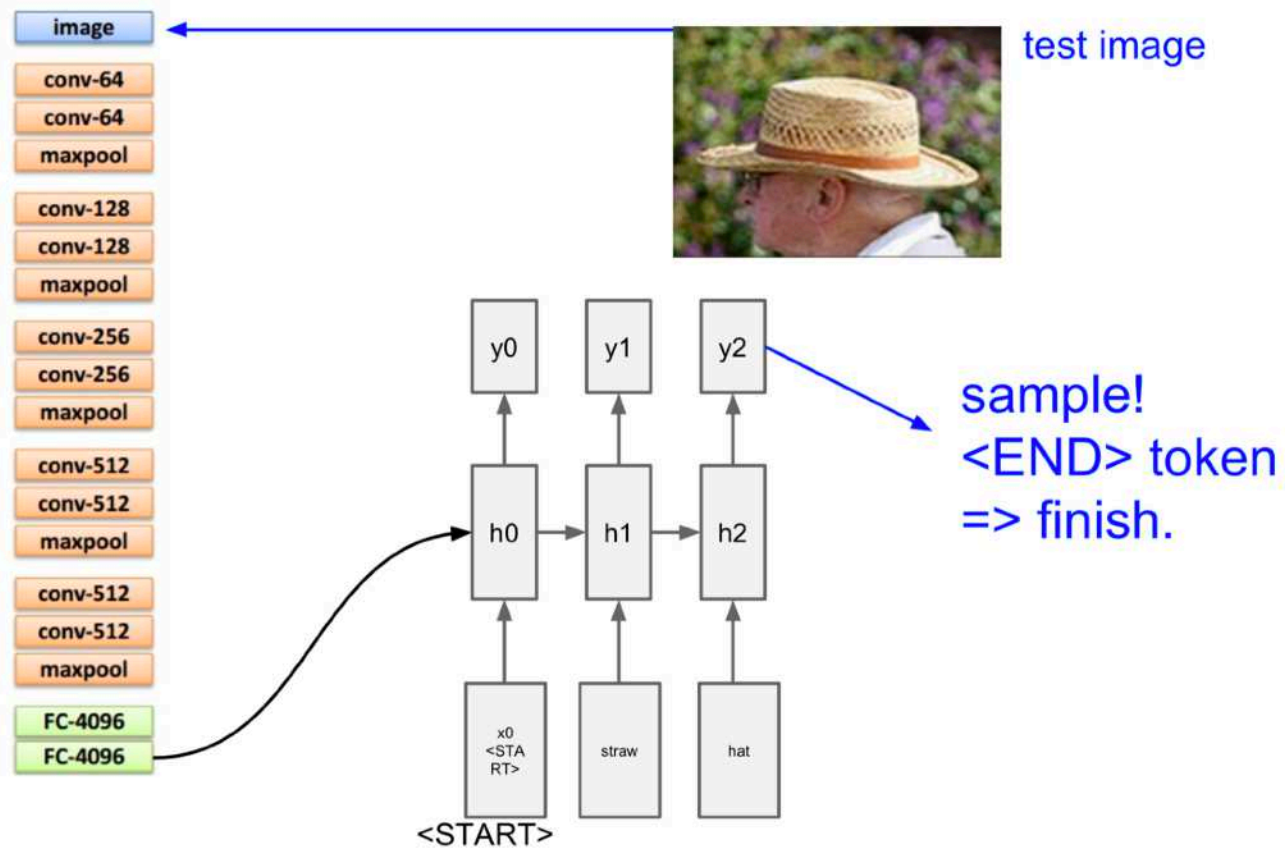


SOURCE: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

[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

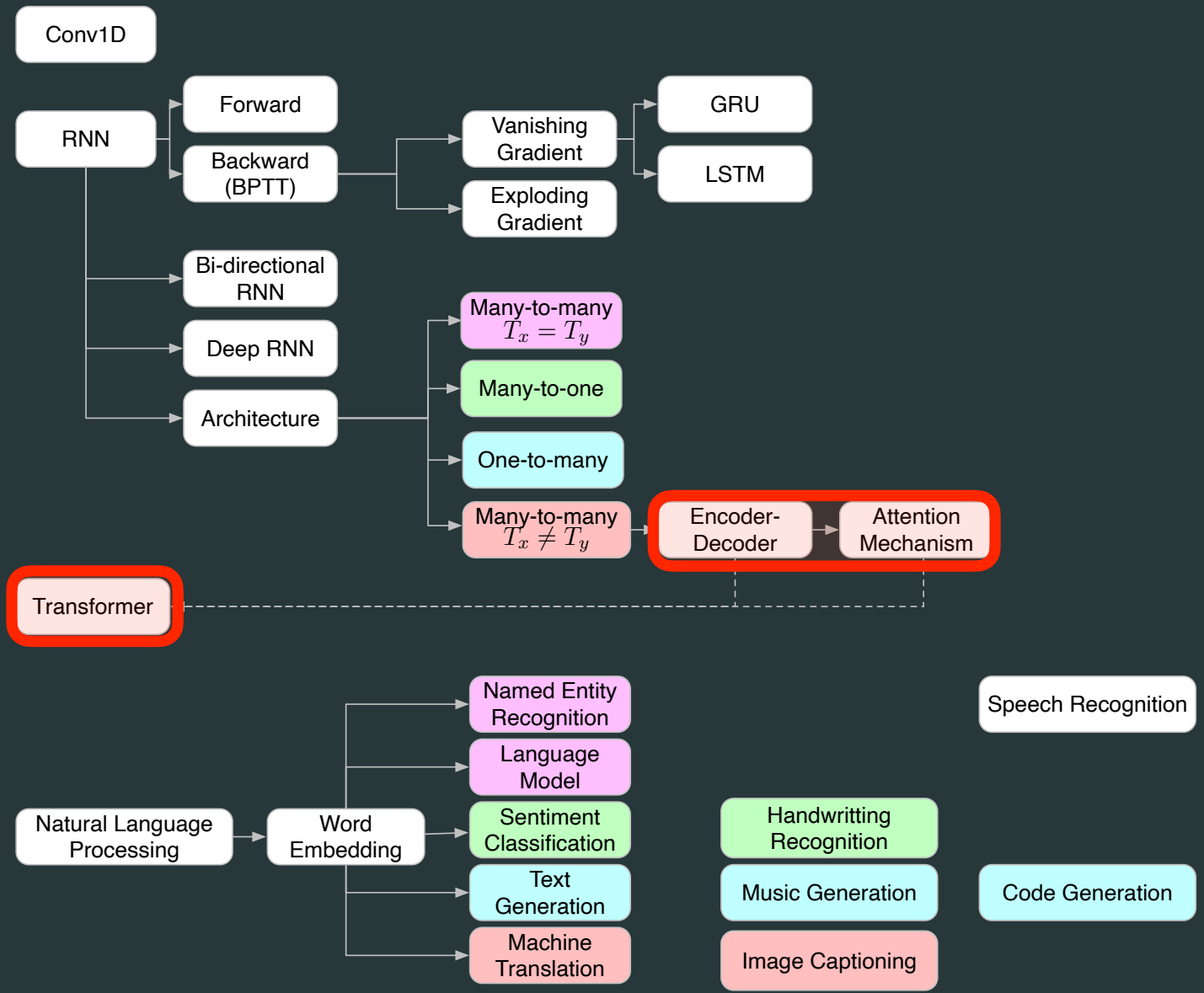
Application: Image captioning



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

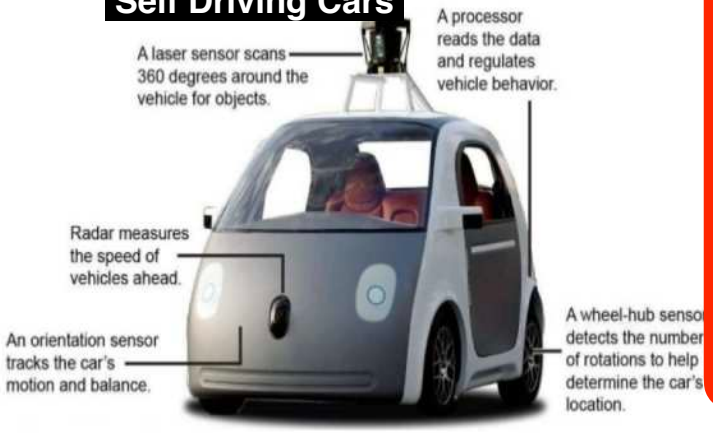
Transformer



Applications

Automatic Speech Recognition

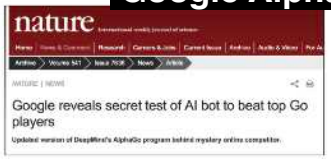
Self Driving Cars



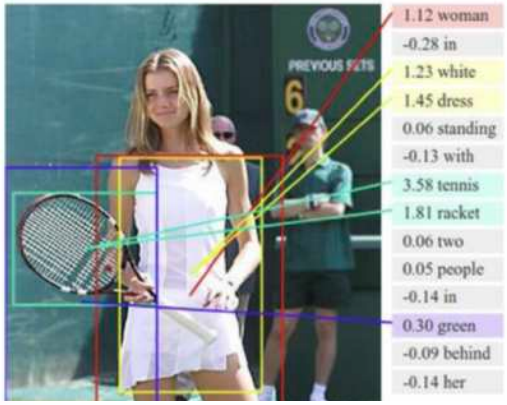
DALL-E

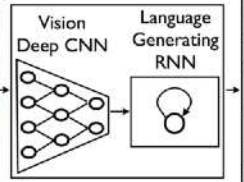
DEEP LEARNING HAS MASTERED GO

Google AlphaGo



Automatic picture captioning

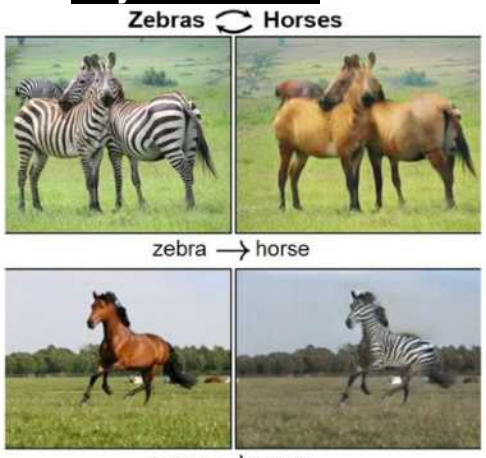




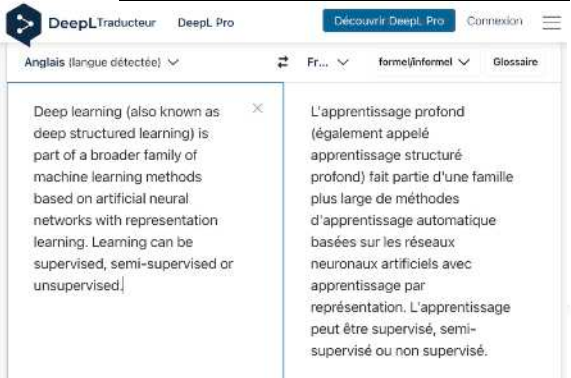
A group shopping at an outdoor market.

There are many vegetables at the fruit stand.

Style Transfer



Neural Machine Translation



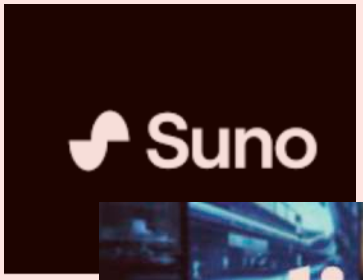
Applications
Generative AI (GenAI)

NLP

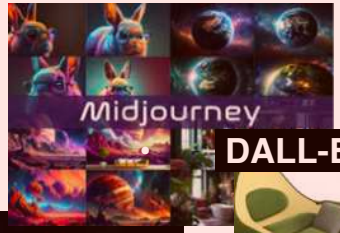
Video

https://www.youtube.com/watch?v=8LjfxFQ5iNk

Music



Computer Vision



DALL-E

Speech (deep fake)

Voice dubbing

Transformer

- **Goal ?**
  - get rid of complex recurrent or convolutional neural networks
- **How ?**
  - simple network architecture (still an encoder decoder) based on
    - (1) attention mechanisms ⇒ self-attention
    - (2) encoder blocks
    - (3) decoder blocks
    - (4) positional encoding

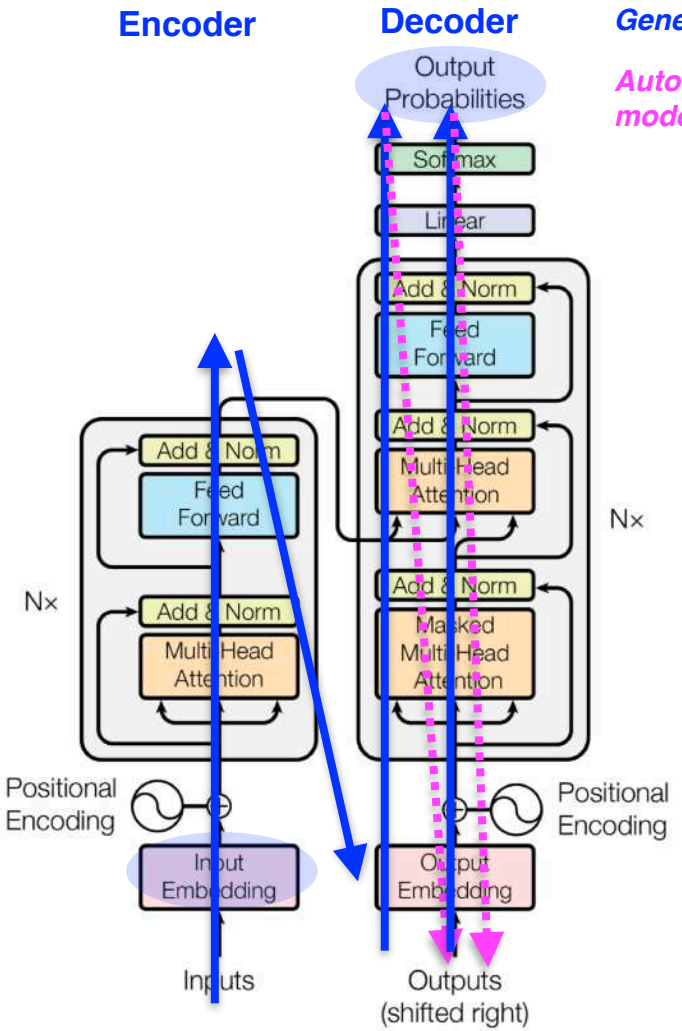


Figure 1: The Transformer - model architecture.

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

Transformer
(1) Self-Attention

Self-Attention ?

– **Example**: we want to translate the following input sequence

- "*The animal didn't cross the street because it was too tired*"

- **Problem**: What does "*it*" in this sentence refer to ? Is it referring to "*street*" or "*animal*" ?

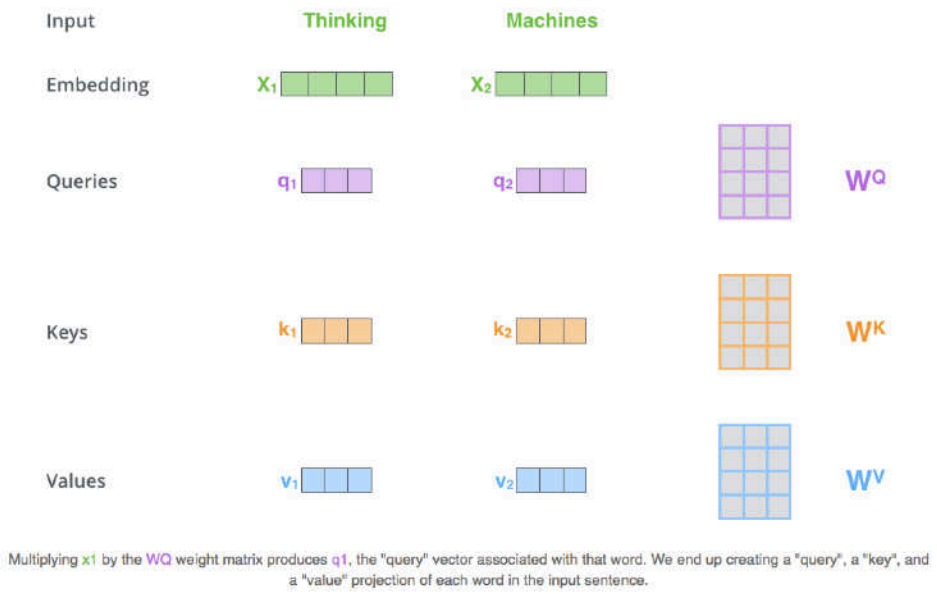
– **Self-Attention ?**

- As the model processes each word (each position in the input sequence), self attention allows the process to look at other positions in the sequence for clues that can help lead to a better encoding for this word
- **Example**: when the model is processing the word "*it*", self-attention allows to associate "*it*" with "*animal*"

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

Implementation

- **First step** :
  - for each input vectors $\mathbf{x}_i \in \mathbb{R}^d$: create three vectors
    - a Query vector $\mathbf{q}_i \in \mathbb{R}^{d'}$
    - a Key vector $\mathbf{k}_i \in \mathbb{R}^{d'}$
    - a Value vector $\mathbf{v}_i \in \mathbb{R}^{d'}$
  - the vectors are created by multiplying $\mathbf{x}_i$ by three matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$
    - $\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q$ $\quad (1,d') = (1,d)(d,d')$
    - $\mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K$
    - $\mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$



- $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ are trained during the training process
- $d' < d$: the new vectors are smaller in dimension than the embedding vector

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

Implementation

– Second step : calculate a score, an "attention"

- α_{ij}= while encoding position i (query), how much should we pay attention on position j (key)

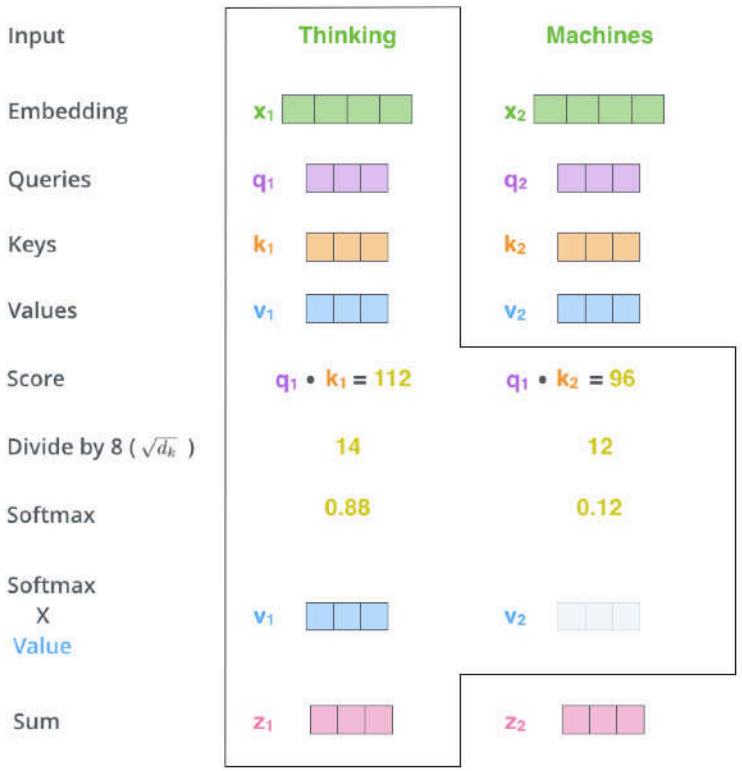
- **Example** : calculate self-attention for "Thinking"

- α_{ij} ?
  - dot product of query q_i with key k_j
  - divide by $\sqrt{d'}$ (square root of the dimension of the key)

$$e_{ij} = \frac{\mathbf{q_i k_j}^T}{\sqrt{d'}} \quad (1,d')(d',1)$$

  - normalise using softmax

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=0}^{T_k} \exp(e_{ik})}$$



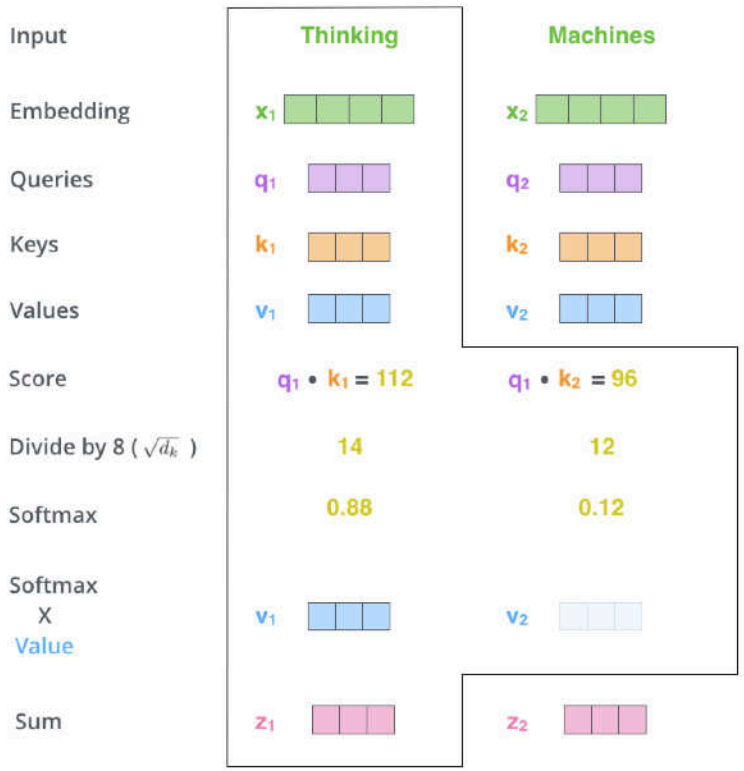
[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Implementation

– **Third step :**

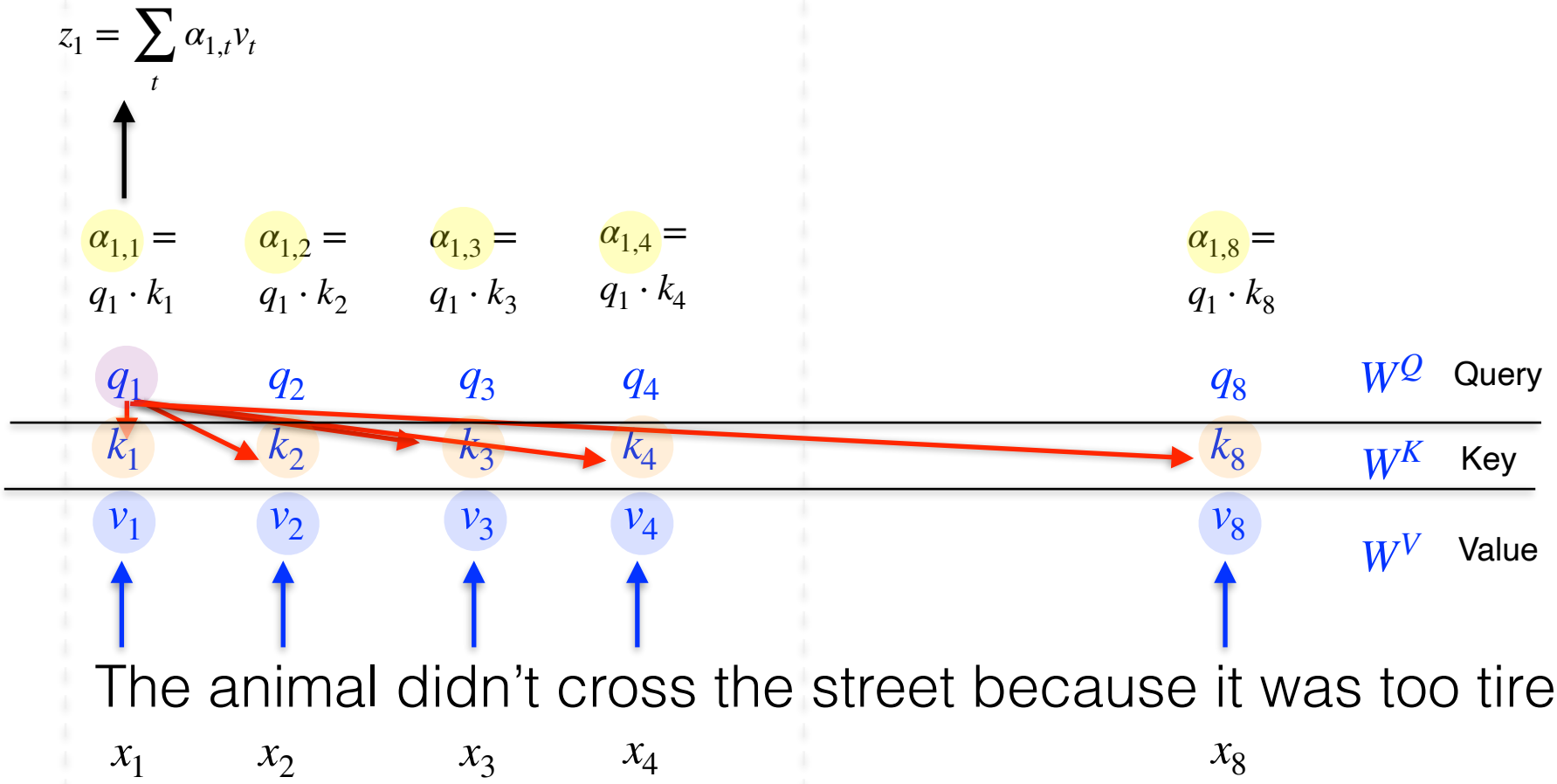
- weighted value: multiply each **value** by the **score**
  - keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words
- final: sum up the weighted value vectors
  - produces the output $\mathbf{z}_i$ of the self-attention layer at this position

$$\mathbf{z}_i = \sum_{j=0}^{T_k} \alpha_{ij} \mathbf{v}_j$$

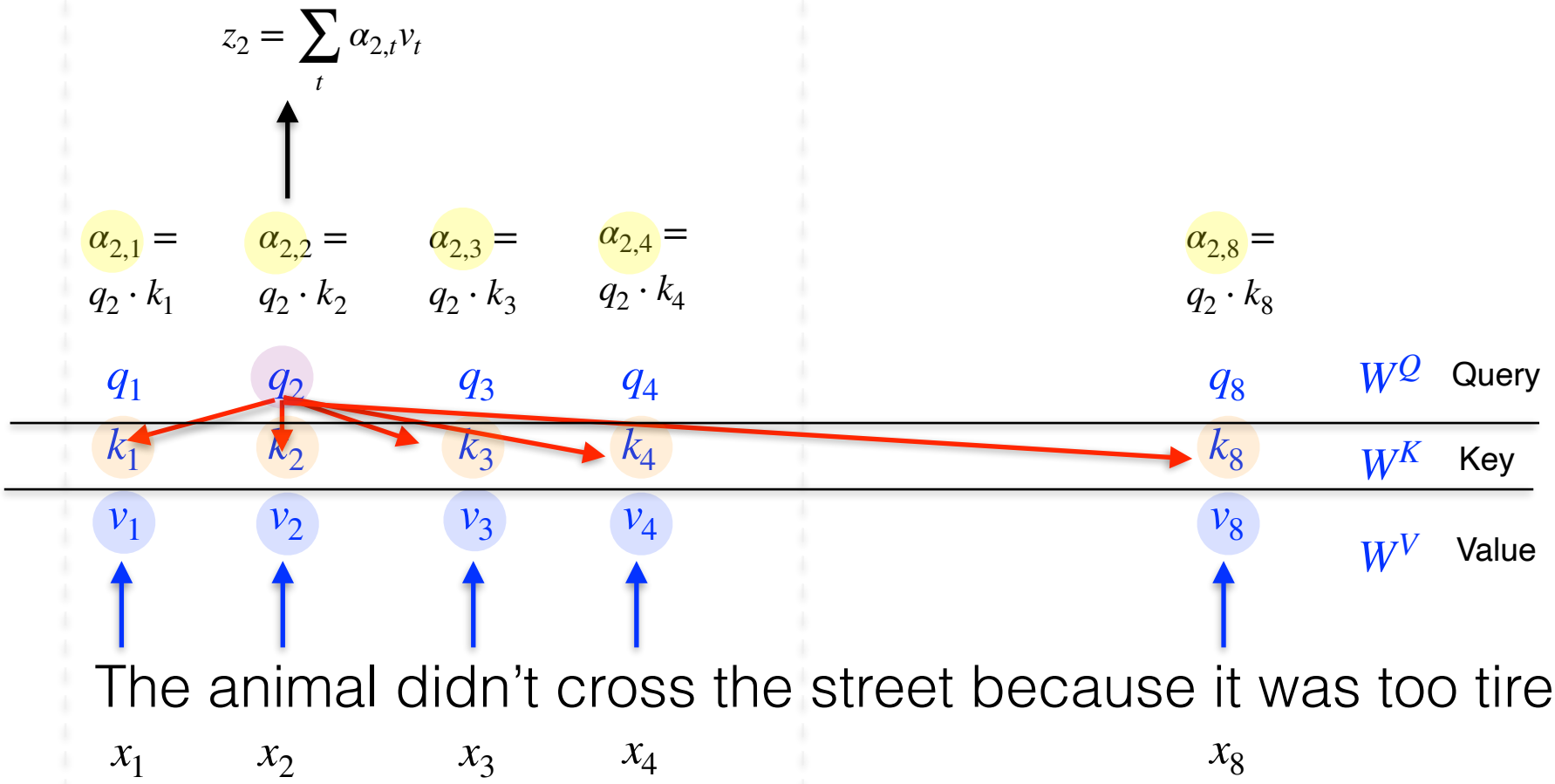


[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

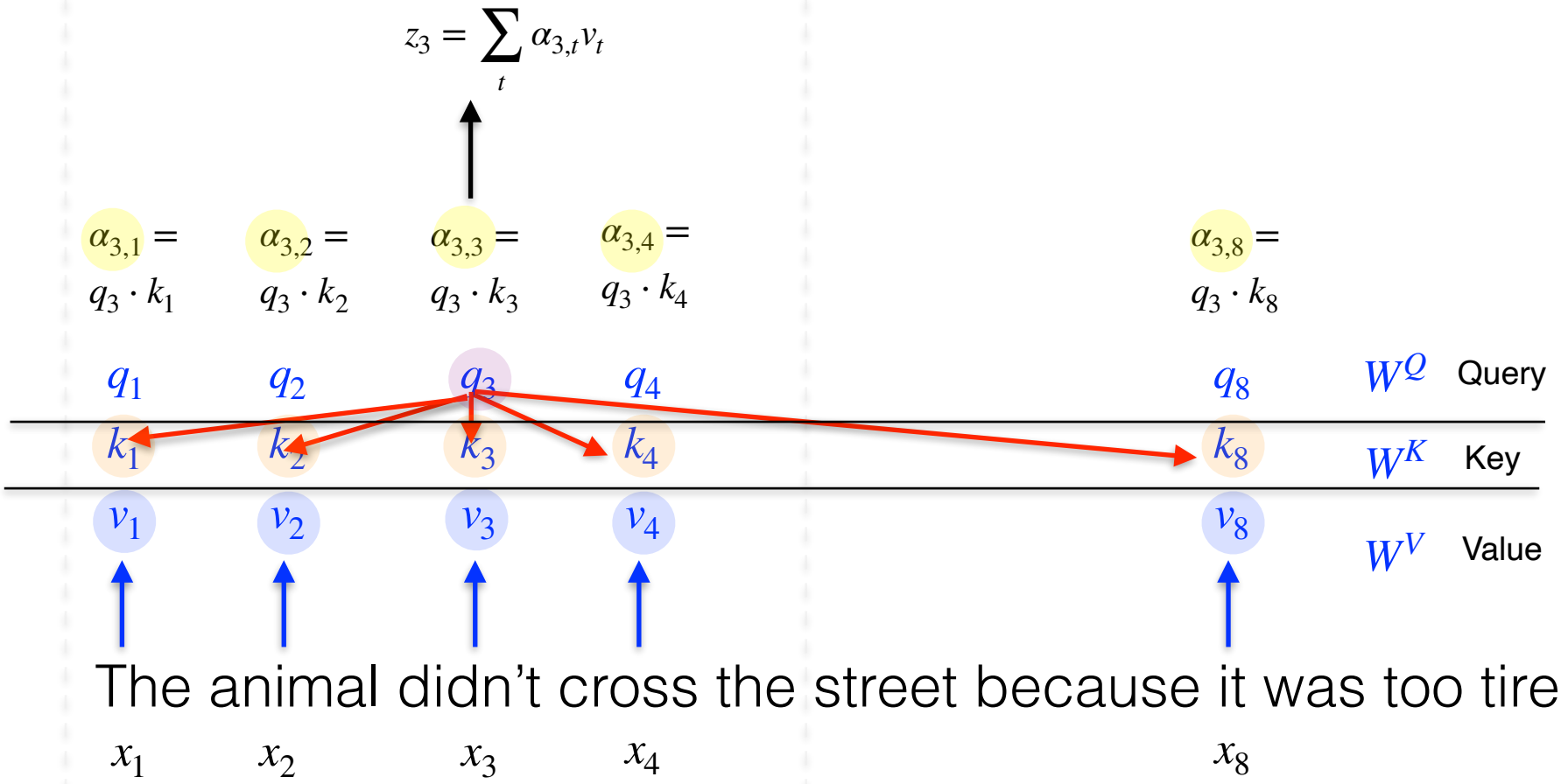
Transformer
(1) Self-Attention



Transformer
(1) Self-Attention



Transformer
(1) Self-Attention



Implementation

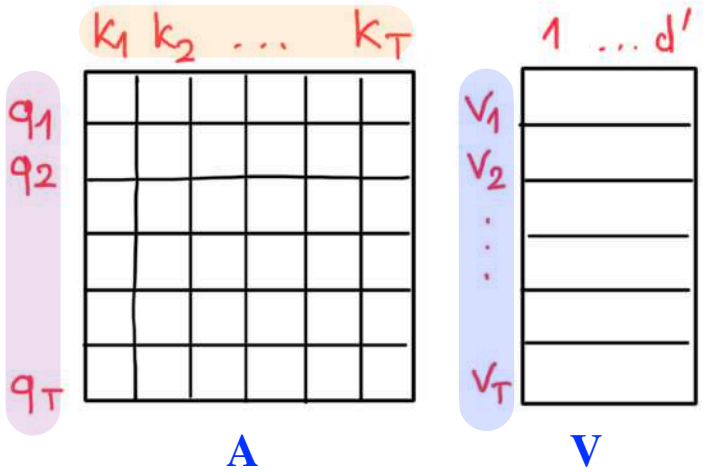
– We can express all computation in **Matrix form** (for all time steps i jointly)

- $\mathbf{Q} = \mathbf{X}\,\mathbf{W}^Q$ $\quad (T, d') = (T, d)(d, d')$
- $\mathbf{K} = \mathbf{X}\,\mathbf{W}^K$
- $\mathbf{V} = \mathbf{X}\,\mathbf{W}^V$

$$\mathbf{E} = \frac{\mathbf{Q}\,\mathbf{K}^T}{\sqrt{d'}} \qquad (T_q, T_k) = (T_q, d')(d', T_k)$$

$$\alpha_{ij} = \frac{\exp(\mathbf{E}_{ij})}{\sum_k \exp(\mathbf{E}_{ik})}$$

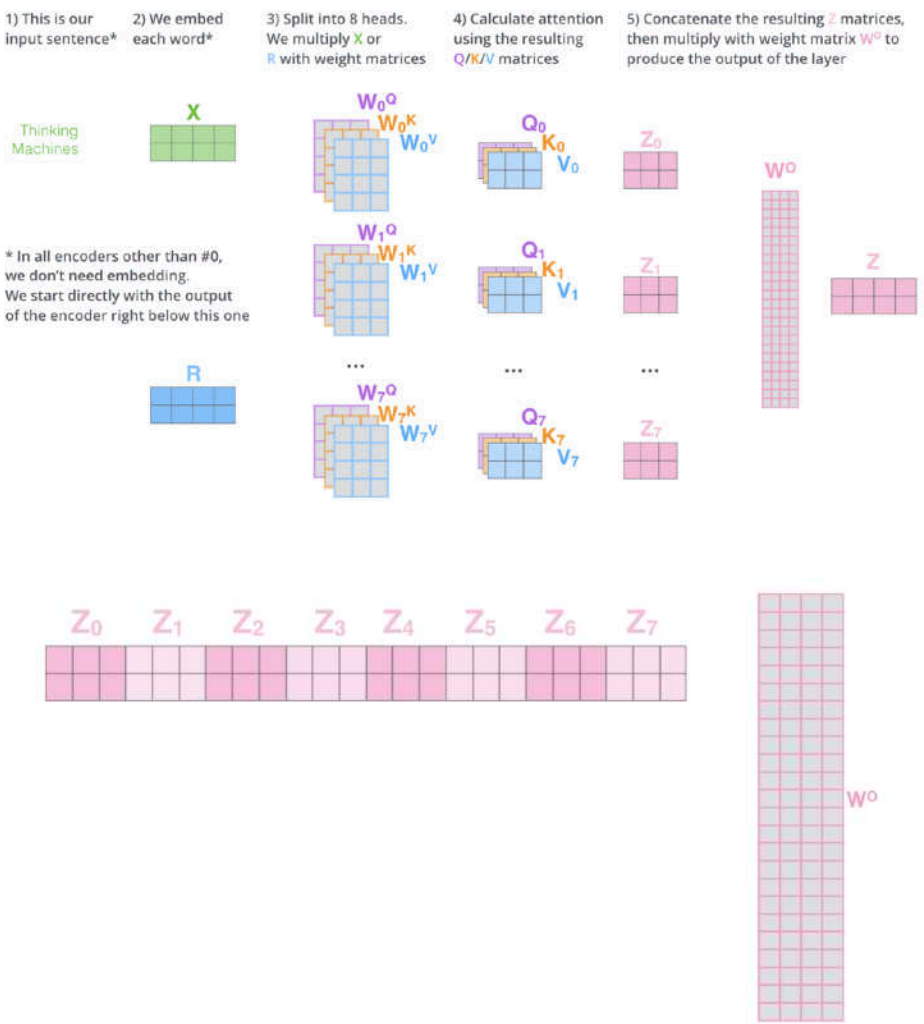
$$\mathbf{Z} = \mathbf{A}\,\mathbf{V} \qquad (T_q, d') = (T_q, T_k)(T_k, d')$$



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

Multi-head Self-Attention

- Several attention heads (often 8)
  - $\mathbf{Q}_0 = \mathbf{X}\mathbf{W}_0^Q, \mathbf{K}_0 = \mathbf{X}\mathbf{W}_0^K, \mathbf{V}_0 = \mathbf{X}\mathbf{W}_0^V$
  - $\mathbf{Q}_1 = \mathbf{X}\mathbf{W}_1^Q, \mathbf{K}_1 = \mathbf{X}\mathbf{W}_1^K, \mathbf{V}_1 = \mathbf{X}\mathbf{W}_1^V$
  - ...
- Leads to several values
  - $\mathbf{Z}_0 \quad (T_q, d')$
  - $\mathbf{Z}_1 \quad (T_q, d')$
  - ...
- Which are concatenated
  - $\mathbf{Z} \quad (T_q, 8 * d')$
- Then projected again using $\mathbf{W}^O \quad (8 * d', d)$

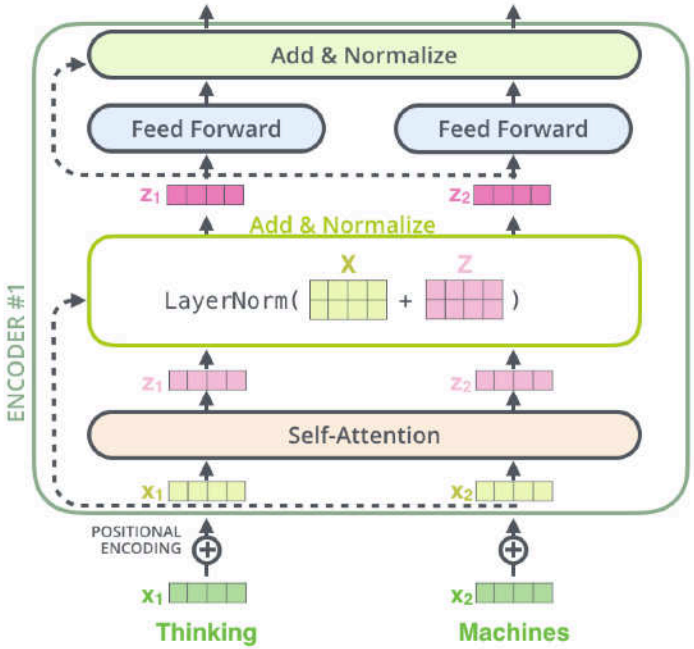


[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

One encoder block

- **Input** $\{\mathbf{x}_i\}_{i\in\{0,\ldots,T-1\}}$:
  - words turned into vectors using an embedding algorithm (only for bottom-most encoder)
- **Two sub-layers**
  - **(1) Self-Attention** layer,
    - compute dependencies between the inputs
    - output: $\{\mathbf{z}_i\}_{i\in\{0,\ldots,T-1\}}$
  - **(2) Feed Forward** neural network, dim_feedforward=2048
    - $FFN(\mathbf{z}_i) = \max(0, \mathbf{z}_i W_1 + b_1) W_2 + b_2$
    - process each element $\mathbf{z}_i$ **independently**
    - the exact same network is applied to each position: can be process in parallel
- Each sub-layer has
  - a **residual** connection around it
  - a **layer-normalisation** step

    normalises the inputs across the features dimension (instead of the batch dimension)

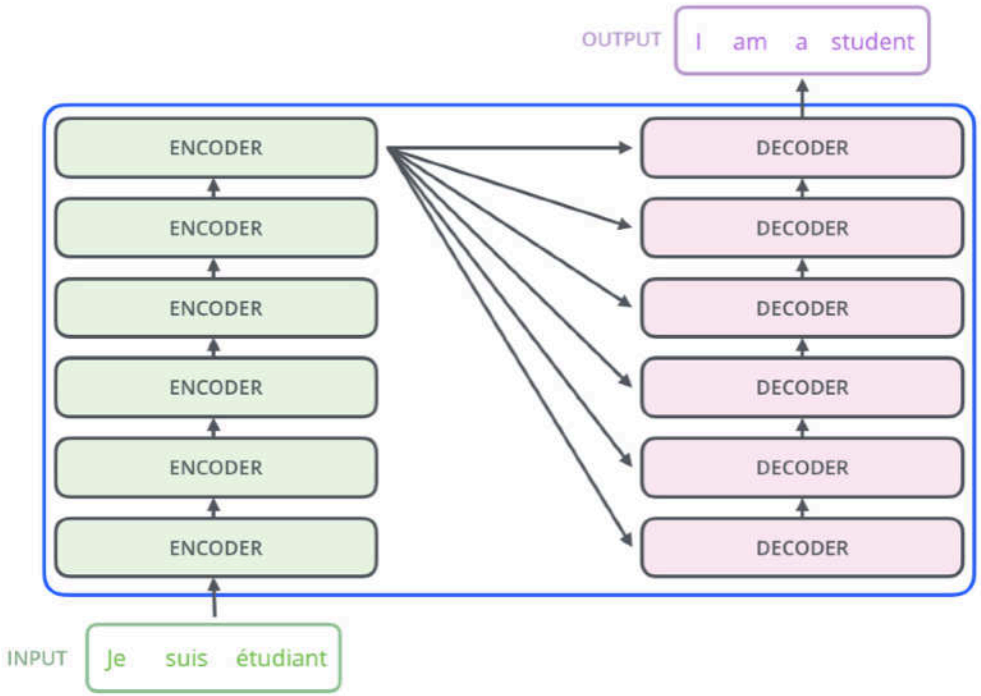


[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

The whole encoder

– **Encoding** component
  - stack of 6 encoder blocks on top of each other

– **Decoding** component
  - stack of 6 decoders blocks on top of each other

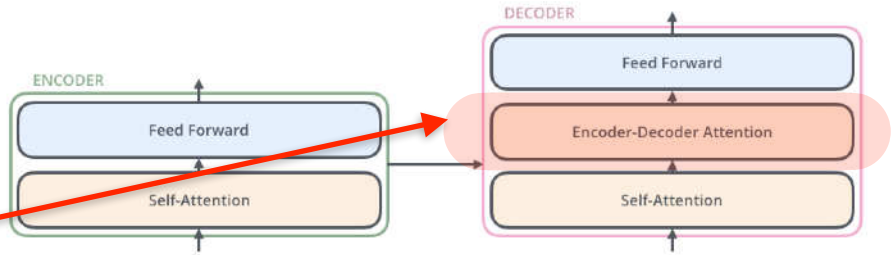


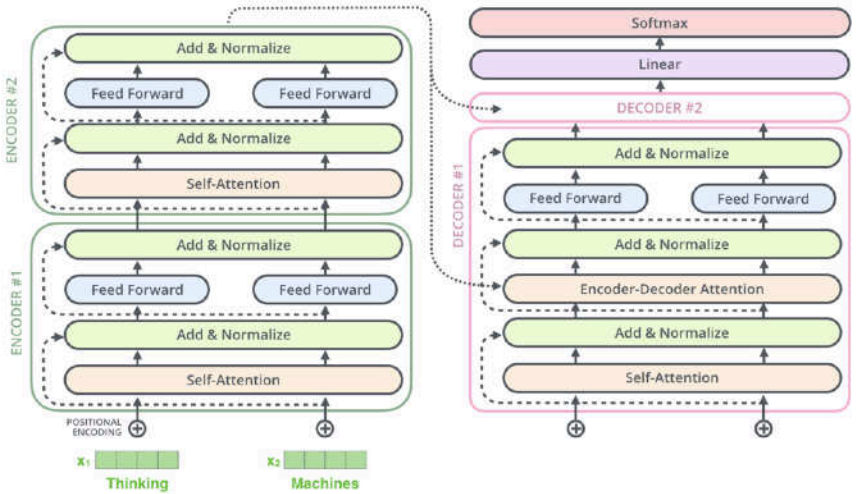
[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Transformer
(3) Decoder block

One decoder block

- **Three sub-layers**
  - **(1) Self-Attention (as the encoder)**
    - but auto-regressive (masking)
  - **(2) Encoder-Decoder Attention**
    - helps the decoder focus on relevant parts of the input sentence
    - similar to what attention does in Seq2Seq models
  - **(3) Feed Forward** layers (as the encoder)
- Each sub-layer has
  - a **residual** connection around it
  - a **layer-normalisation** step

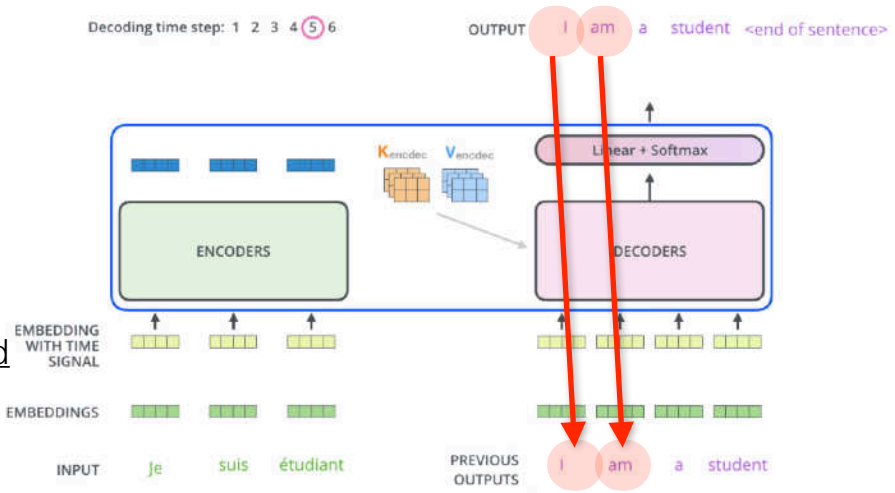




[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

One decoder block

- **Decoder Self-Attention:**
  - is auto-regressive !!!
    - output of the decoder of each step is fed to the bottom in the next step
    - the self-attention layer only allowed to attend to earlier positions in the output sequence
    - masking (see next slide)
- **Encoder-Decoder Attention ?**
  - works just like multi-head self-attention
  - helps the decoder focus on appropriate places in the input sequence
  - K_{encdec} and V_{encdec} comes from transformed output of the top encoder
    - used by each decoder in its EDA layer
  - Queries come from the layer below it
- Embed and add **positional encoding** to the decoder inputs (as for the encoder)



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

Masked Self-Attention (decoder)

– **Matrix form** (for all time steps i)

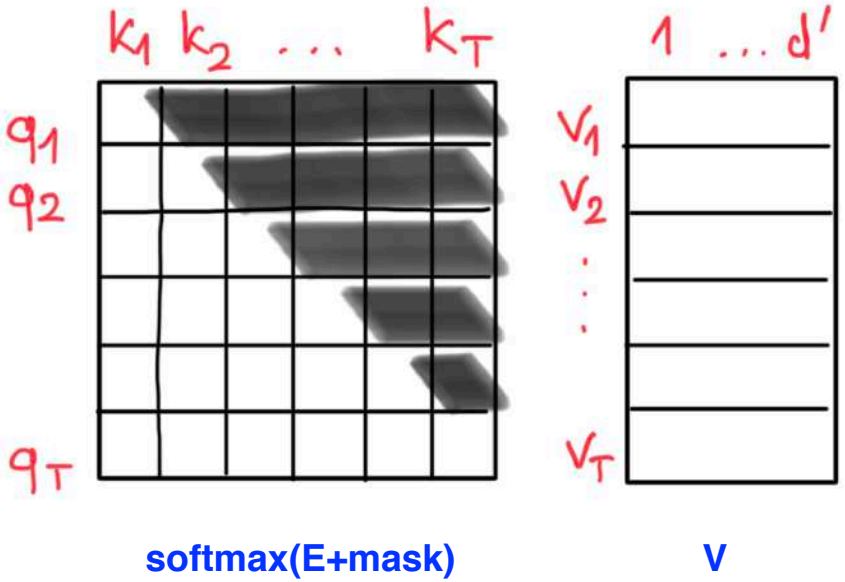
- the decoder is an auto-regressive model
- it cannot see the future
- use masking

$$\mathbf{E} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d'}} \qquad [(T_q, d')(d', T_k)](T, d')$$

$$\mathbf{Z} = \textbf{softmax}\,(\mathbf{E} + \textbf{mask})\,\mathbf{V}$$

$$\text{mask} = \begin{pmatrix} 0 & -\inf & -\inf & \dots & -\inf \\ 0 & 0 & -\inf & \dots & -\inf \\ 0 & 0 & 0 & \dots & -\inf \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

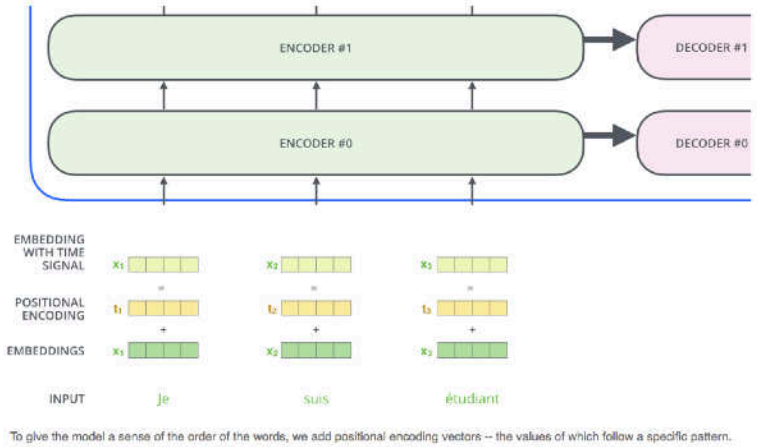
$$e^{-\inf} = 0$$



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

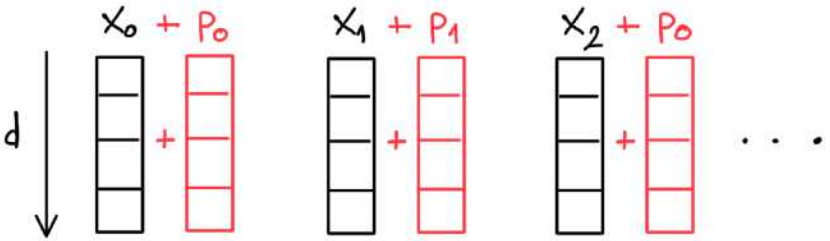
Transformer
(4) Positional encoding

- **Problem:**
  - Transformer doesn't have an inherent sense of word order or position;
  - It does not know that $\mathbf{x}_i$ is before or after $\mathbf{x}_j$



- **Solution:**
  - We encode the order/position of each element of the sequence by **adding** a **Positional Encoding**

- **Which Positional Encoding ?**
  - Define
    - $p_0 = [0,0,\ldots,0], p_1 = [1,1,\ldots,1], p_1 = [2,2,\ldots,2]$
    - PE add too much distortion to $\mathbf{x}_i$!
  - Normalise by the length of the sequence
    - $p_0 = [0,0,\ldots,0], p_1 = [1/3,\ldots,1/3], p_2 = [2/3,\ldots,2/3]$
    - PE will change depending on the length !



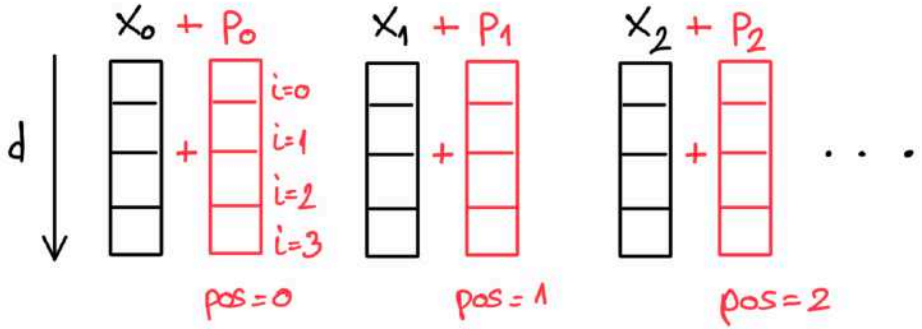
[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

Which positional encoding ?

– **[Vaswani et al 2017]**

$$\text{PE}(pos,2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$\text{PE}(pos,2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$



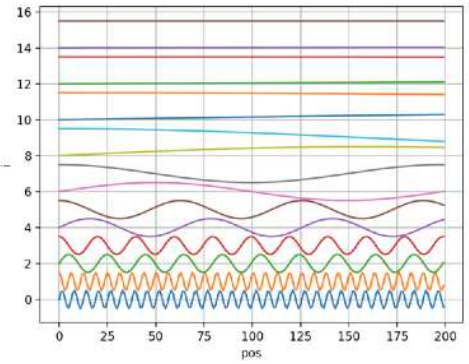
– **Example**:

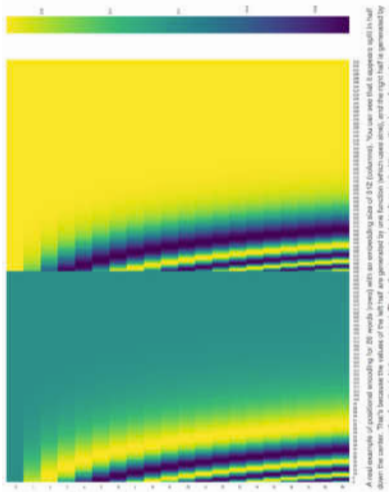
$i = 0 \rightarrow 10000^{0/4} = 1$ $\rightarrow PE(pos,0) = \sin(pos)$

$\rightarrow PE(pos,1) = \cos(pos)$

$i = 1 \rightarrow 10000^{2/4} = 100$ $\rightarrow PE(pos,2) = \sin(pos/100)$

$\rightarrow PE(pos,3) = \cos(pos/100)$





```
pos=np.arange(0,200)
d=16
clf()
for i in range(int(d/2)):
                    plot(pos, 2*i+0.5*sin(pos/(10000**(2*i/d))));
                    plot(pos, 2*i+1++0.5*cos(pos/(10000**(2*i/d))));
plt.xlabel('pos'), plt.ylabel('i'); plt.grid(True)
```

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

Which positional encoding ?

– **Absolute positional encoding**

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}$$

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d}}$$

$$x_i = x_i + p_i$$

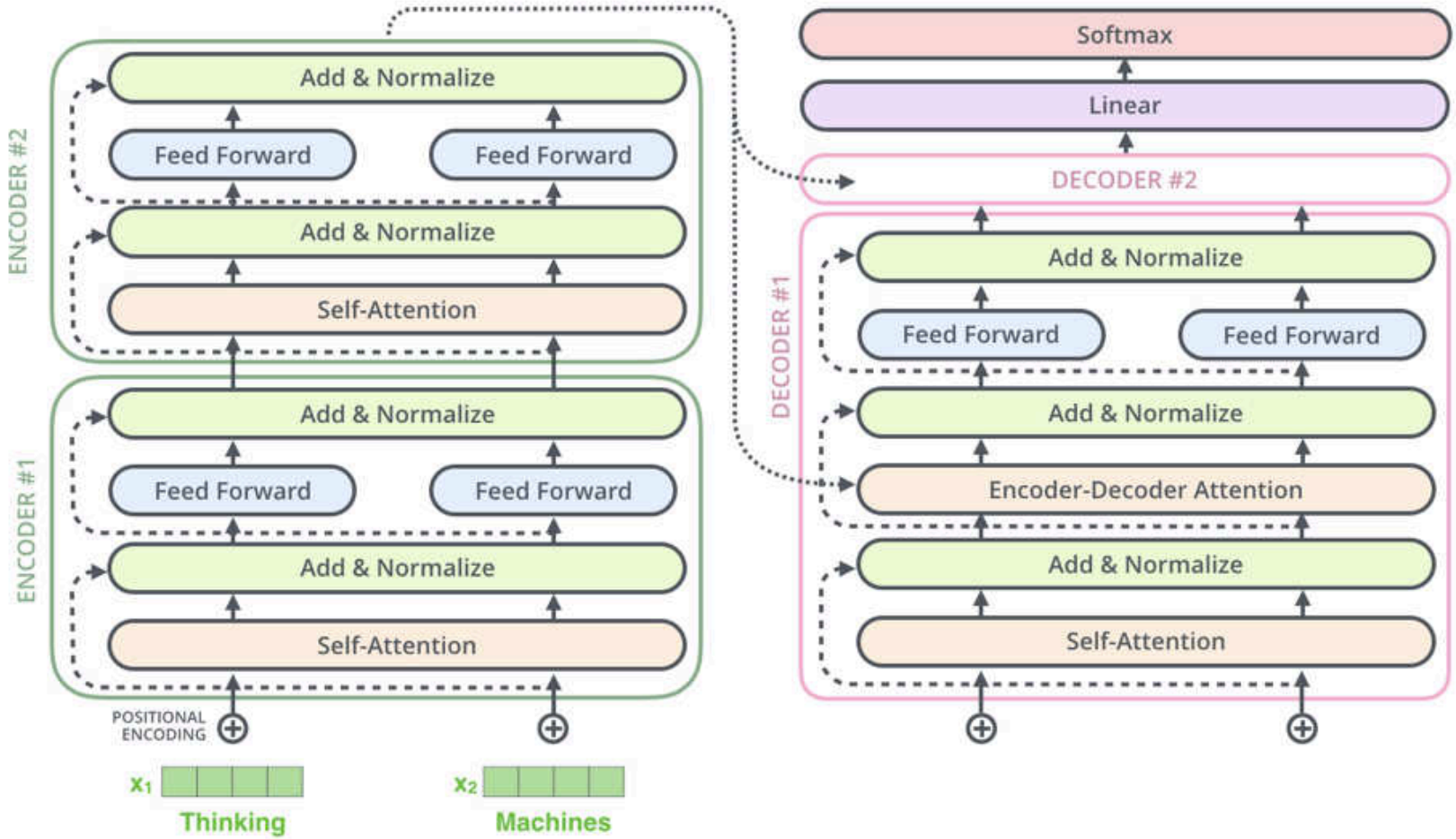
– **Relative positional encoding**

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V + p_{ij}^V)$$

$$e_{ij} = \frac{(x_i W^Q + p_{ij}^Q)(x_j W^K + p_{ij}^K)^T}{\sqrt{d}}$$

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

Transformer



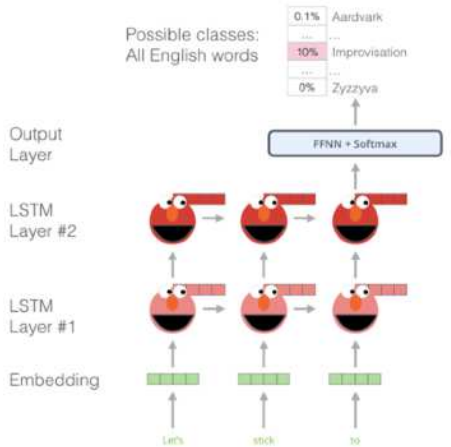
[Vaswani et al. "Attention Is All You Need", NIPS, 2017] LINK
[J. Alammar "The Illustrated Transformer"] LINK

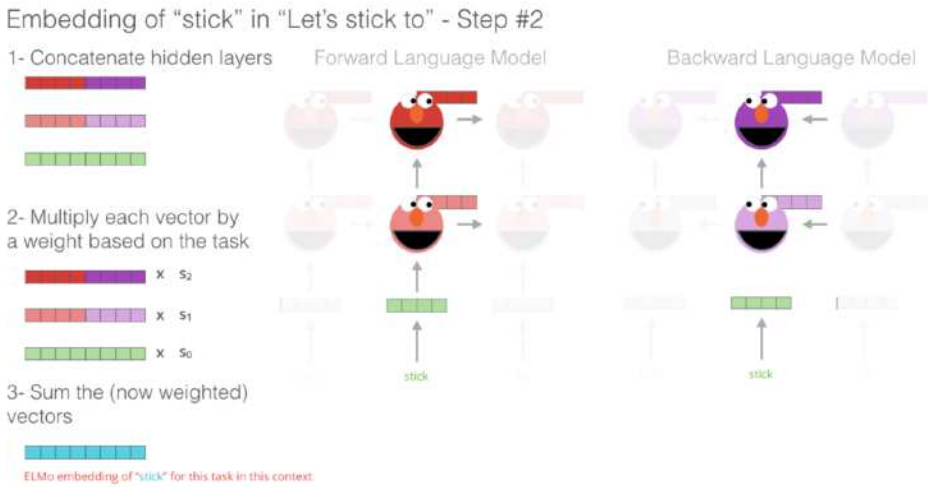
Transformer usage

Transformer usage in NLP

ELMo (Embeddings from Language Models)

- Context Matters:
  - takes into the context when computing the embedding
- How ?
  - bi-directional LSTM
- Contextualised embedding :
  - grouping together the hidden states (and initial embedding) in a certain way (concatenation followed by weighted summation)



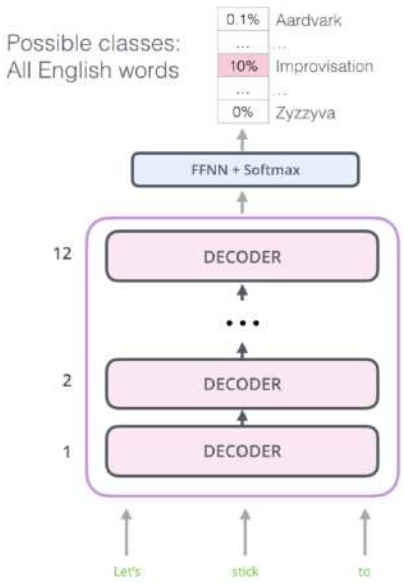


[Peters et al. "Deep contextualized word representations", 2018] LINK

Transformer usage in NLP

OpenAI GPT

- Use Transformer (instead of LSTM)
- Only use the decoder (predicting the next word) since it's built to mask future tokens
- Pre-training:
  - predict the next word using massive (unlabelled) datasets: 7,000 books
- Transfer-learning (downstream tasks)



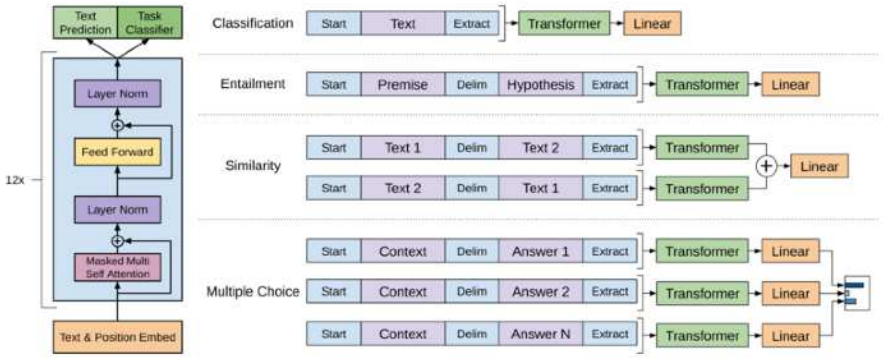


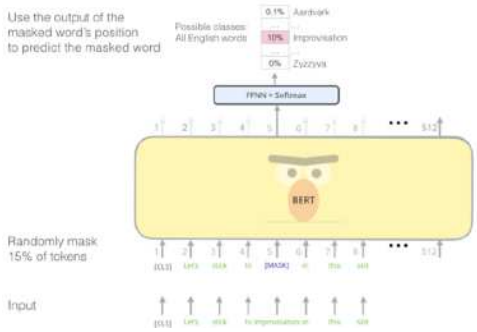
Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

[Radford et al. "Improving Language Understanding by Generative Pre-Training", 2018] LINK

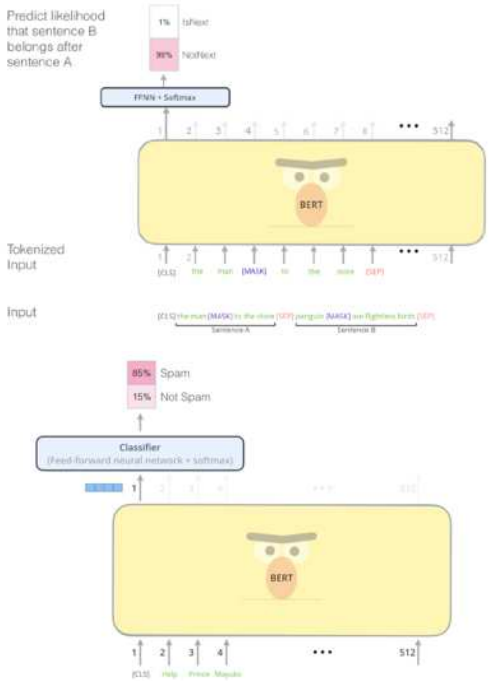
Transformer usage in NLP

Google BERT

- OpenAI GPT is Transformer Decoder
  - this is great but Decoder is not bi-directional (LSTM ELMo was !)
- BERT
  - Use the Transformer Encoder
  - the first input token is supplied with a special [CLS] token. CLS here stands for Classification
- Training on pretext tasks:
  - Masked Language Model
  - Two-sentence classification task
- Use directly the CLS output for downstream tasks (Spam, not Spam)



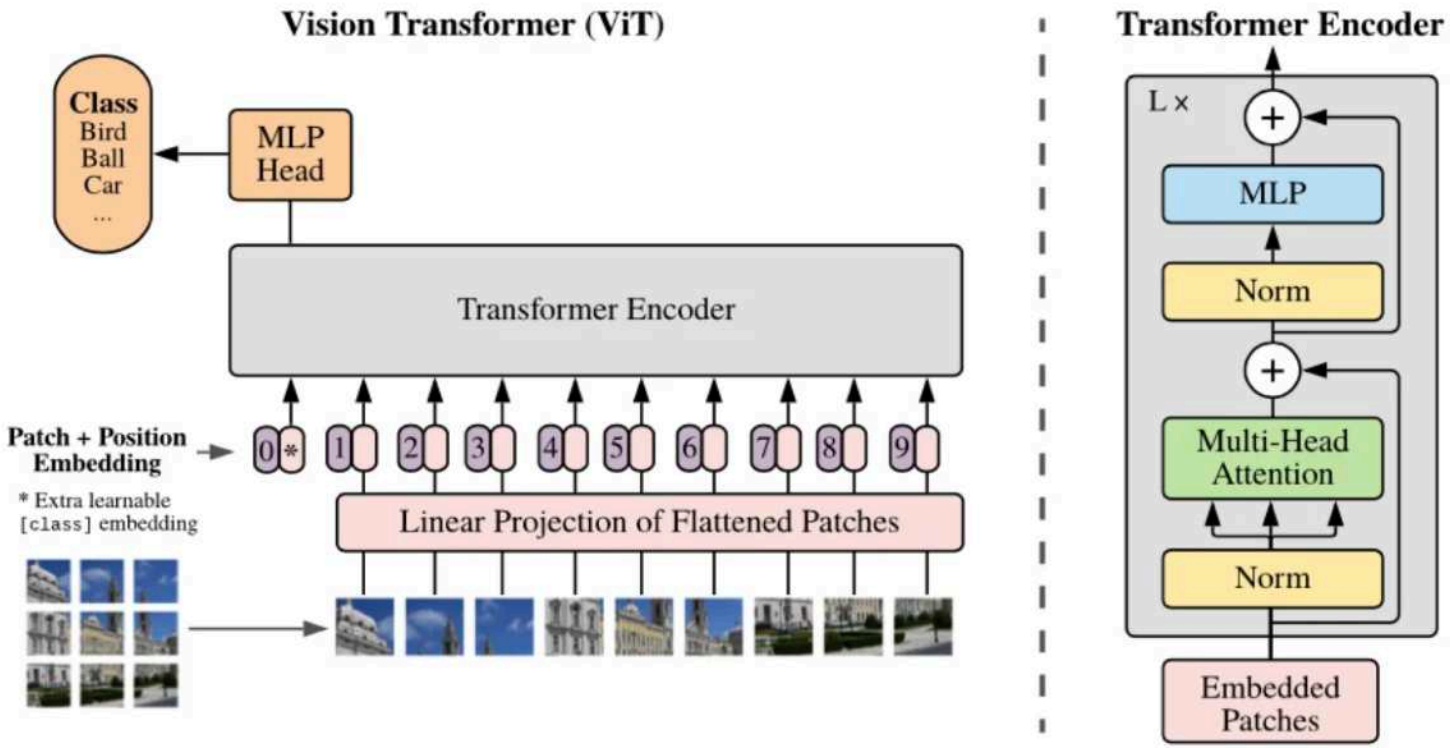
Masked Language Model



Two-sentence classification task

[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

Vision Transformer

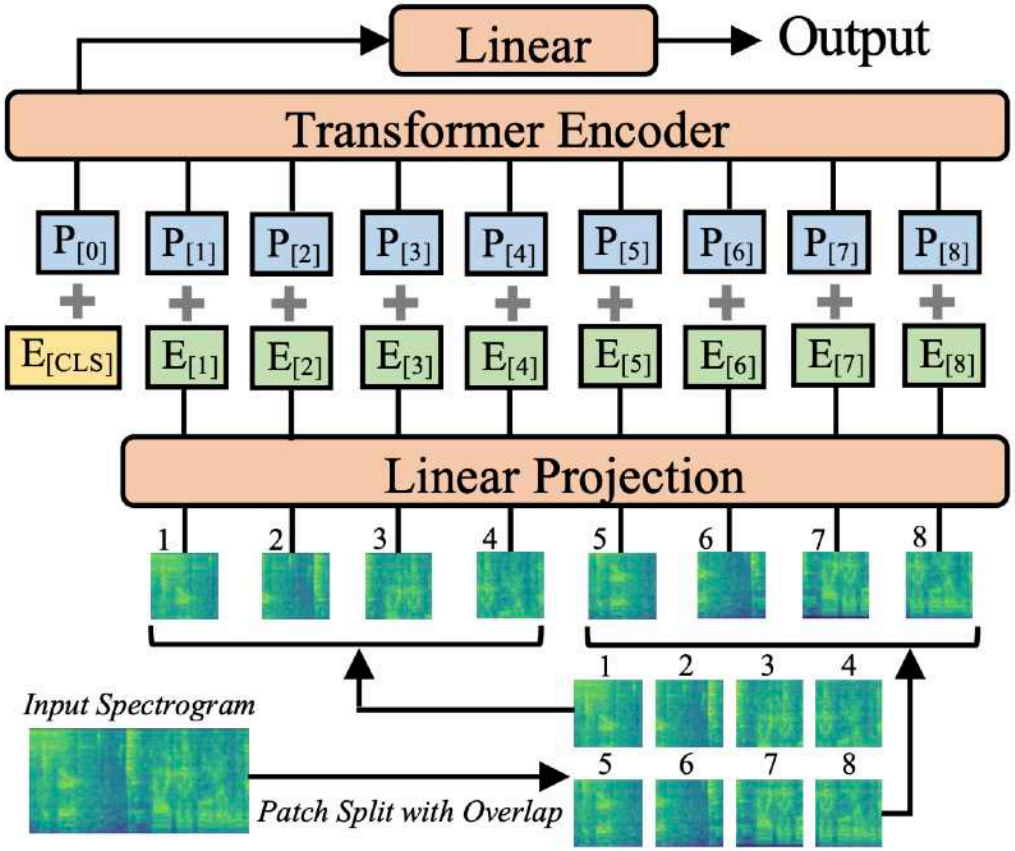


Vision Transformer ViT Architecture – Source

[Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale"] LINK

Transformer usage in Audio

Audio Spectrogram Transformer



[Gong et al. "AST: Audio Spectrogram Transformer"] LINK

Evolutionary Tree

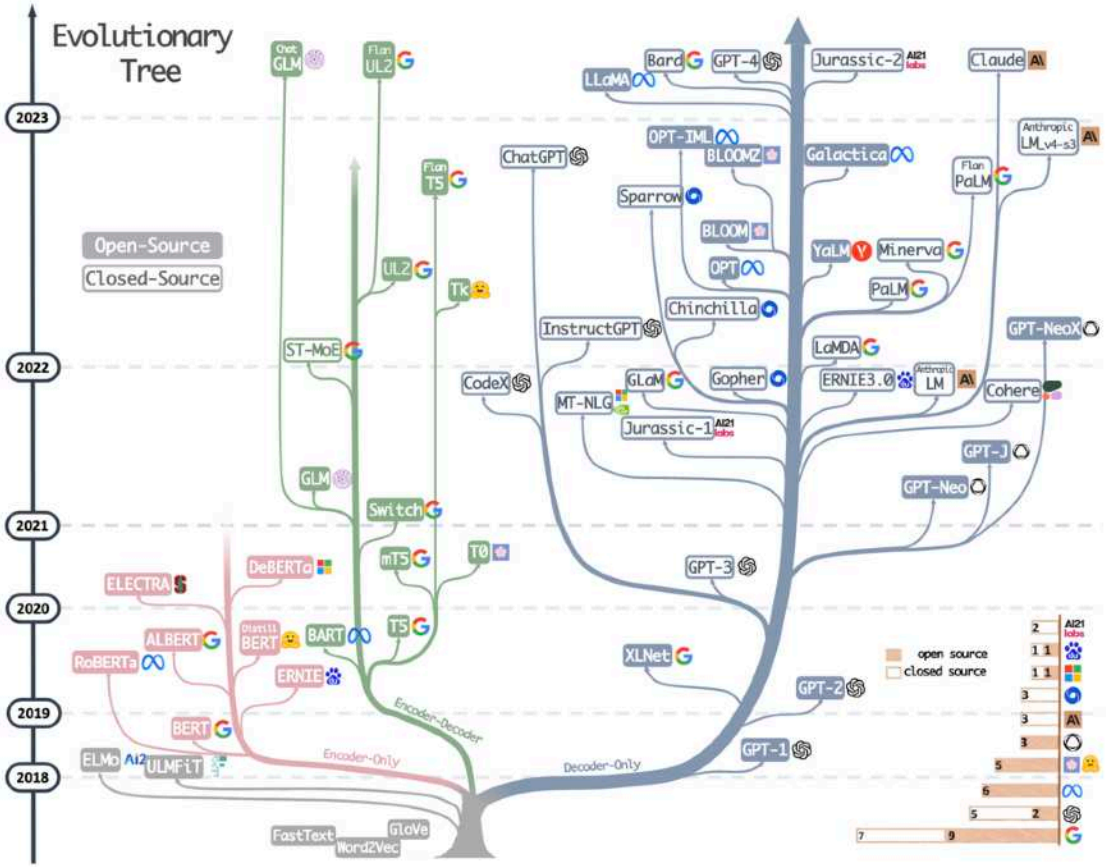
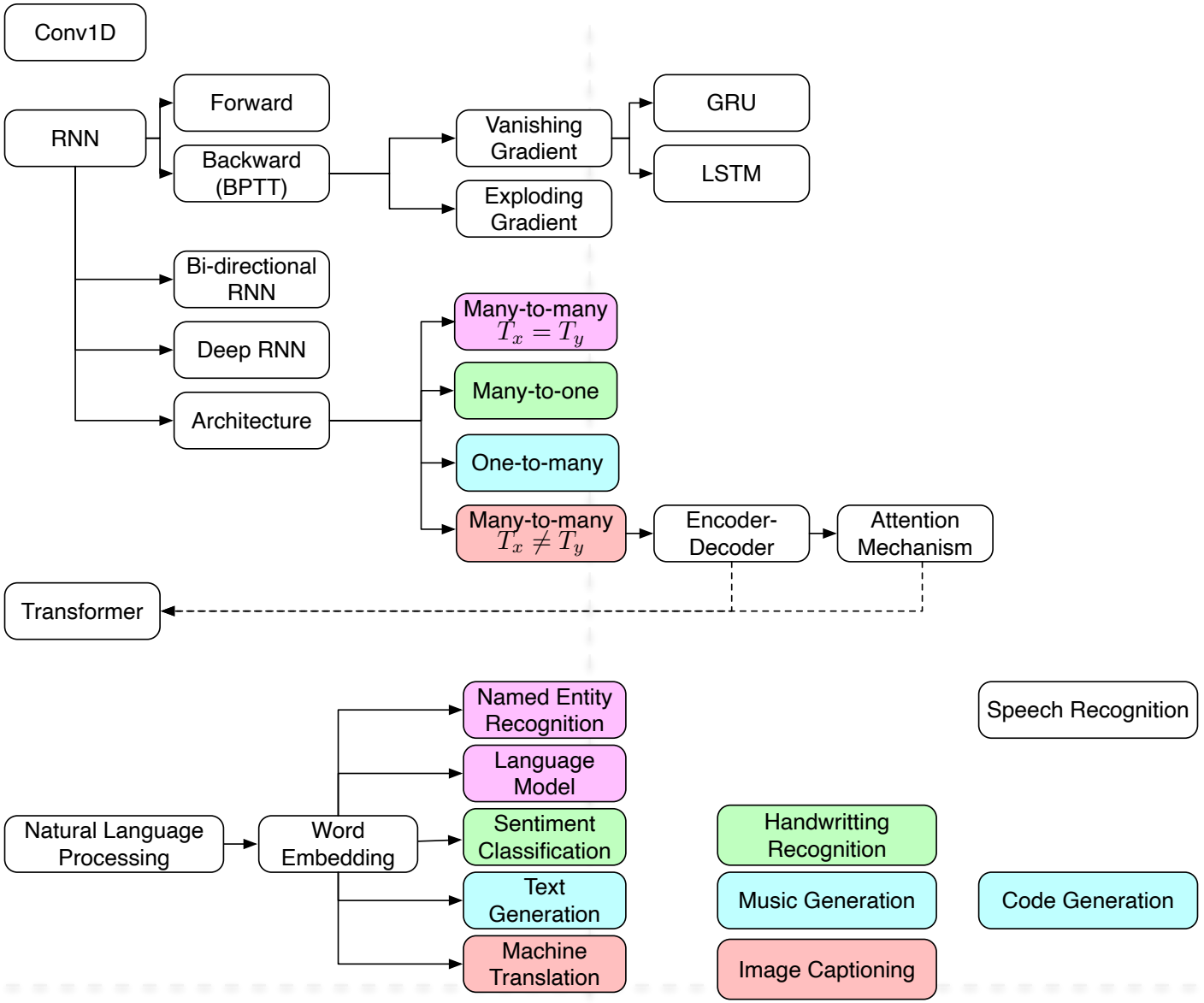


Fig. 1. The evolutionary tree of modern LLMs traces the development of language models in recent years and highlights some of the most well-known models. Models on the same branch have closer relationships. Transformer-based models are shown in non-grey colors: decoder-only models in the blue branch, encoder-only models in the pink branch, and encoder-decoder models in the green branch. The vertical position of the models on the timeline represents their release dates. Open-source models are represented by solid squares, while closed-source models are represented by hollow ones. The stacked bar plot in the bottom right corner shows the number of models from various companies and institutions.

[Yang et al. "Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond", 2023] LINK

Overview


Conv1D
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