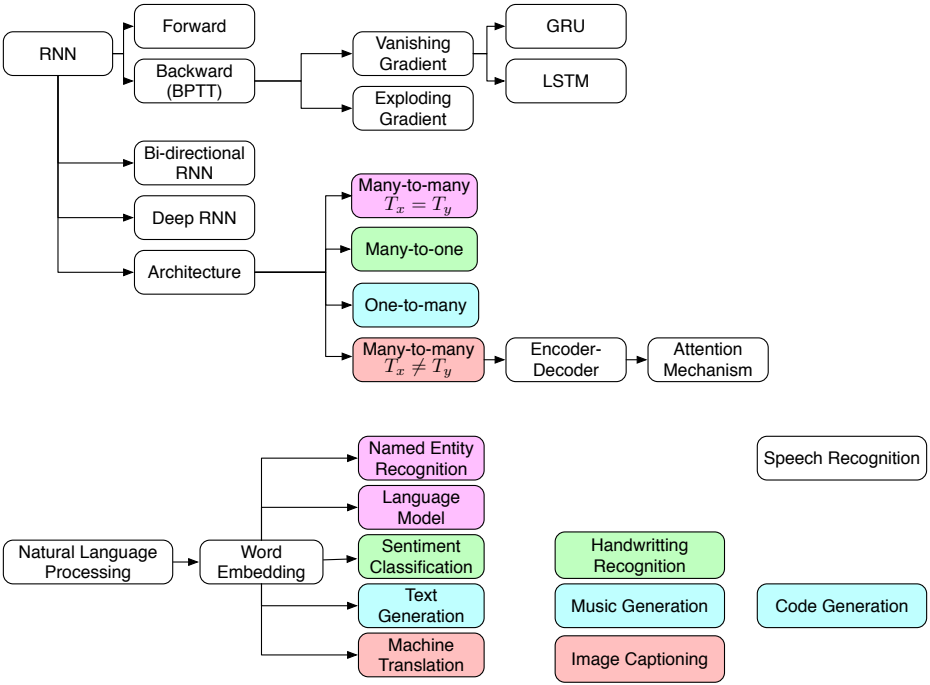
Deep Learning
Topics: Recurrent Neural Networks for Sequential Data

Geoffroy Peeters

LTCI, Télécom Paris, IP Paris

-

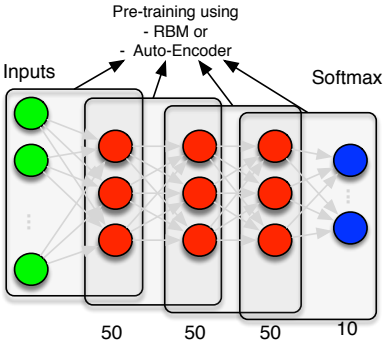
Overview


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

Three main types of Nets

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

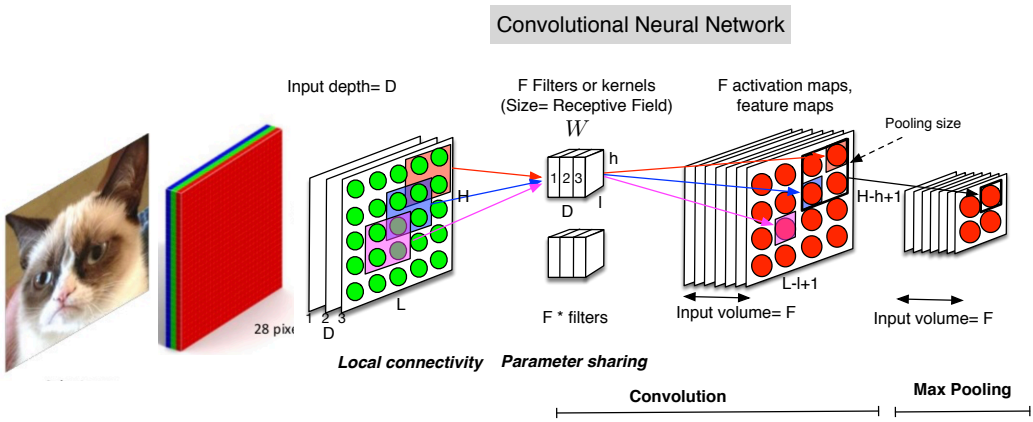
Multi Layers Perceptron (Fully Connected)



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

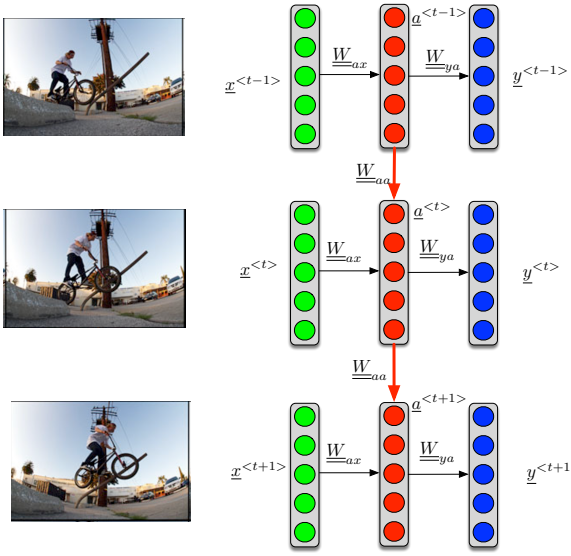
$$\underline{a}^{[l]} = g^{[l]}(\underline{z}^{[l]})$$

Convolutional Neural Networks (CNN)



Recurrent Neural Networks (RNN)

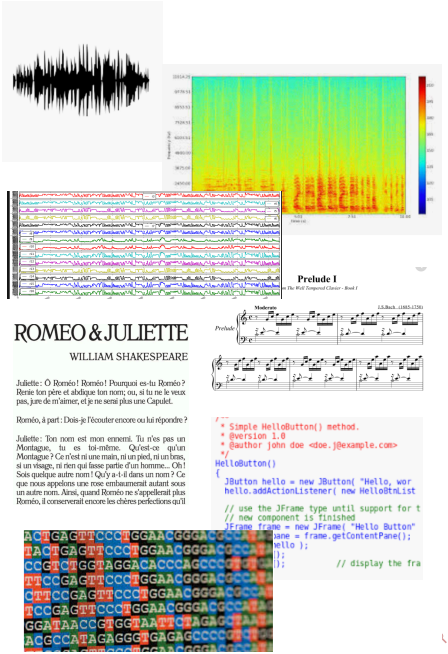
Recurrent Neural Network



Recurrent Neural Networks for Sequencial 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 :
  - $\underline{x}^{\langle t \rangle}$: input vectors $\underline{x}$ at time t (of dimension $n^{[0]}$)
  - $\{\underline{x}^{\langle 1 \rangle}, \underline{x}^{\langle 2 \rangle}, \dots \underline{x}^{\langle T_x \rangle}\}$ the sequence of inputs of length T_x
- Outputs :
  - $\underline{y}^{\langle t \rangle}$: input vectors $\underline{y}$ at time t
  - $\{\underline{y}^{\langle 1 \rangle}, \underline{y}^{\langle 2 \rangle}, \dots \underline{y}^{\langle T_y \rangle}\}$ the sequence of outputs of length T_y
- T_x and T_y can be of different lengths
- $x^{(i)<t>}$ is the i^{th} example in the training-set
  - $T_x{}^{(i)}$ the length of the i^{th} input sequence
  - $T_y{}^{(i)}$ the length of the i^{th} output sequence

Notations

- Different types of output $\underline{y}$:
  - Continuous values $\underline{y} \in \mathbb{R}$ (Linear MSE Loss)
  - Binary-classes $\underline{y} \in \{0, 1\}$ (Sigmoid and Binary-Cross-Entropy)
  - Multi-classes $\underline{y} \in \{1 \dots K\}$ (Softmax and Cross-Entropy)
- Different types of input $\underline{x}$
  - Continuous values $\underline{x} \in \mathbb{R}$
  - Categorical values $\underline{x} \in \{1 \dots K\}$
- **Example** : categorical inputs/ binary outputs
  - "Named Entity Recognition"

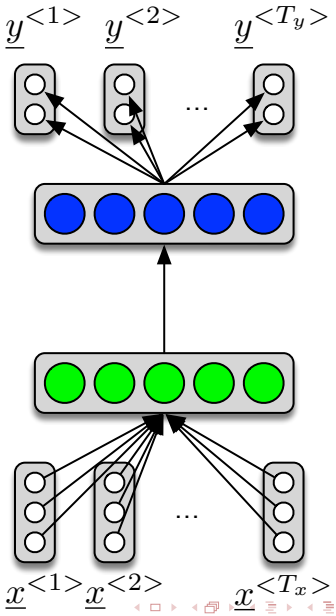
{x}	$x^{<1>}$	$x^{<2>}$	…				$x^{<7>}$
	Emmanuel	Macron	is	the	president	of	France
{y}	$y^{<1>}$	$y^{<2>}$	…				$y^{<7>}$
	1	1	0	0	0	0	1

- **How do we process categorical ©values as input ?**
  - One-hot-encoding
  - Embedding

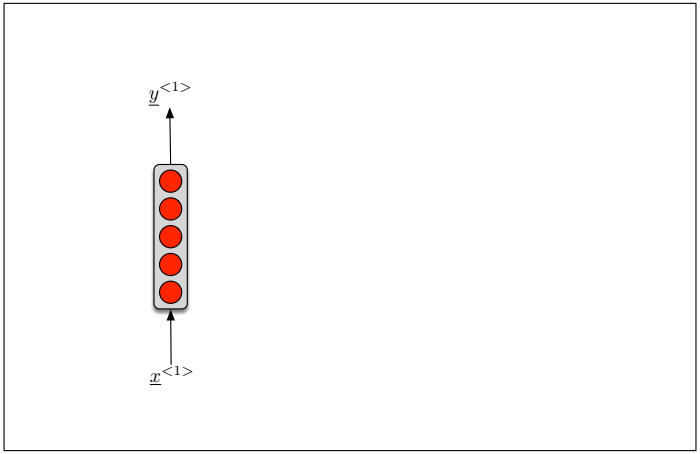
How can we process Sequential data with a Neural Network

- **Why Multi-Layer-Perceptron/Fully-connected networks are not appropriate ?**

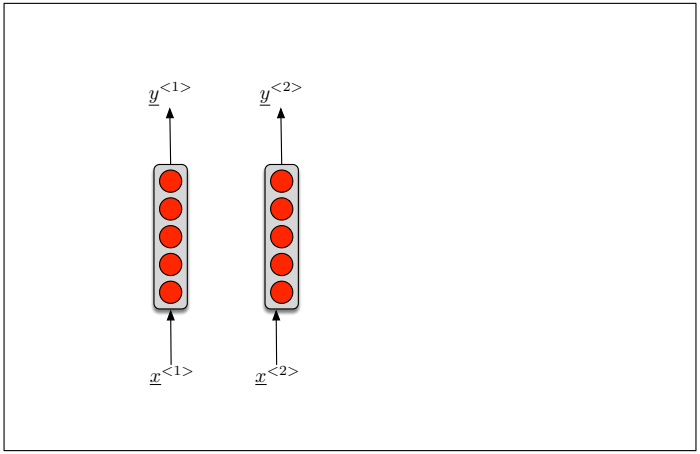
- We could represent an input sequence $\{\underline{x}^{\langle 1\rangle} \cdots \underline{x}^{\langle T_x\rangle}\}$ as a large-vector of dimension $(n^{[0]} T_x)$
- but (because ${T_x}^{(i)}$ can be different for each input sequence) the dimension of this large-vector would change for each
  - zero-padding ?
- still require a huge input dimension $(n^{[0]} T_x)$
- the network will have to learn different weights for the same dimension $n^{[0]}$ at various time $\langle t\rangle$ in the sequence
  - no weight sharing !
- **Solutions :**
  - Recurrent Neural Network
  - 1D Convolutional Neural Network (convolution only over time)



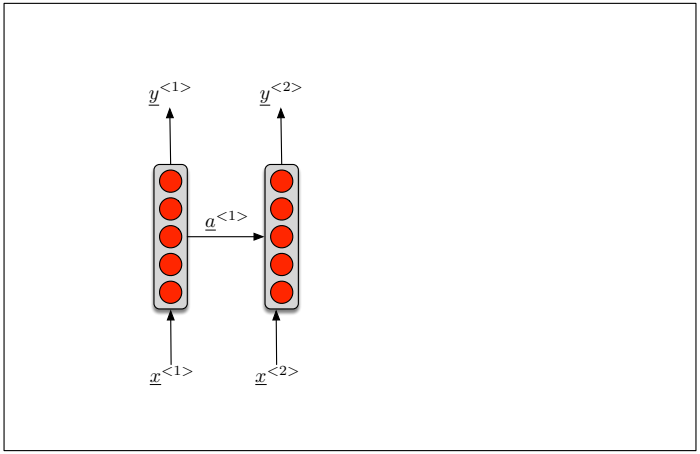
What is an RNN ?



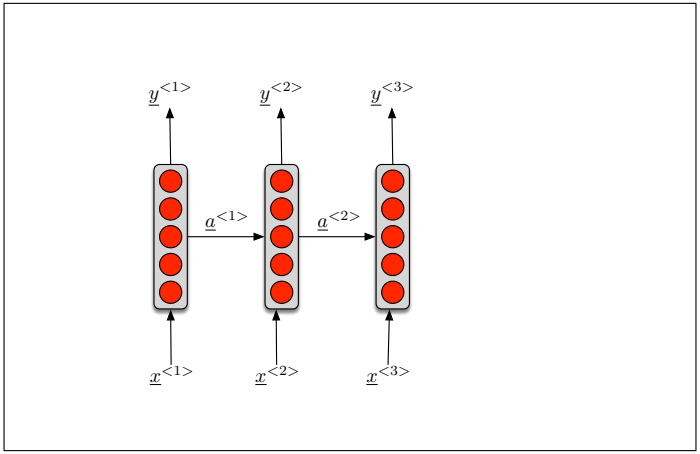
What is an RNN ?



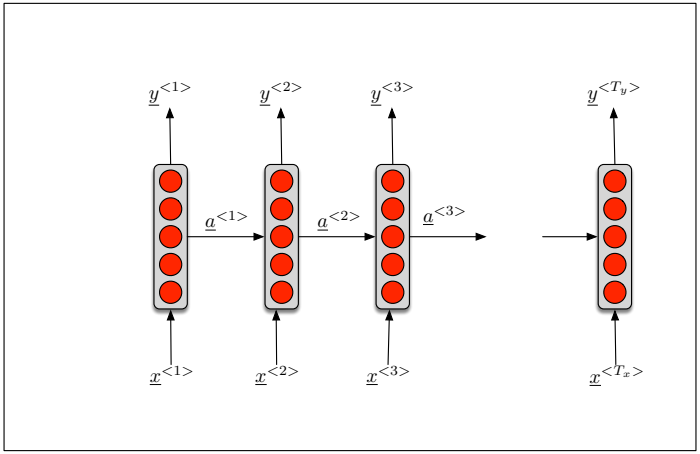
What is an RNN ?



What is an RNN ?

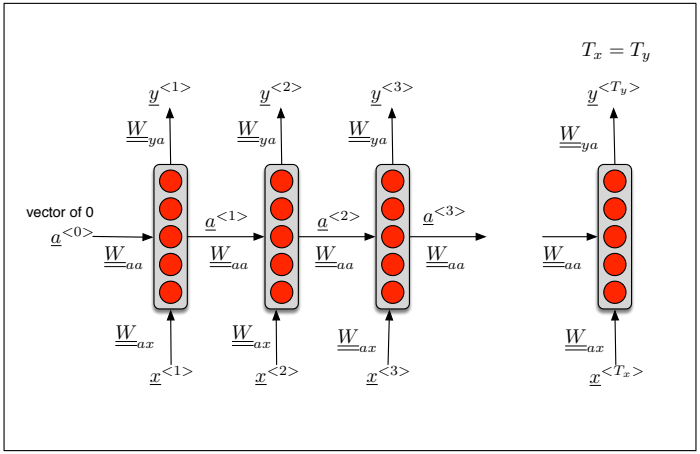


What is an RNN ?



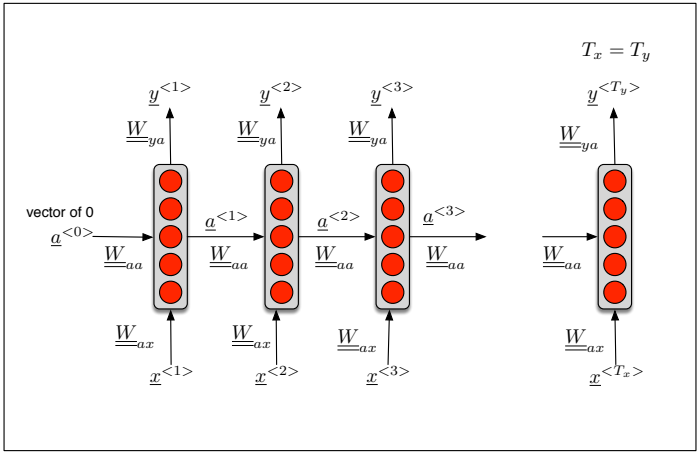
- At each time step $< t >$, the RNN passes its activation $a^{\langle t \rangle}$ to the next time step $< t + 1 >$

What is an RNN ?



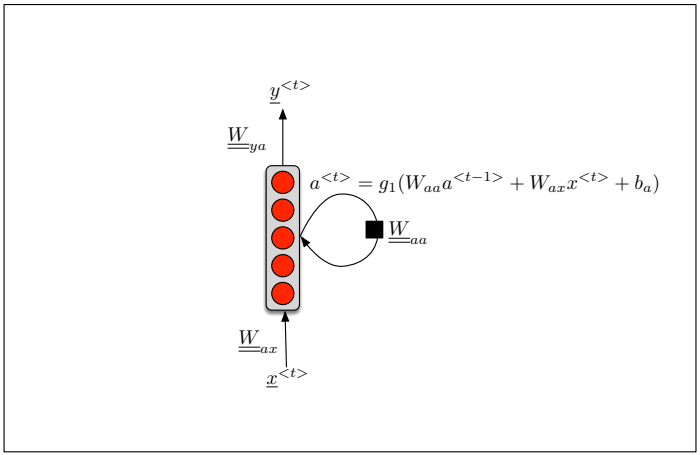
- Initialize the first activation $a^{\langle 0 \rangle}$ as zero

What is an RNN ?



- $\underline{\underline{W}}_{ax}$ ($x^{\langle t \rangle} \rightarrow a^{\langle t \rangle}$) is the same for every time step
- $\underline{\underline{W}}_{aa}$ ($a^{\langle t-1 \rangle} \rightarrow a^{\langle t \rangle}$) is the same for every time step
- $\underline{\underline{W}}_{ya}$ ($a^{\langle t \rangle} \rightarrow y^{\langle t \rangle}$) is the same for every time step

Other possible representation



- $\underline{\underline{W}}_{ax}$ ($x^{\langle t \rangle} \rightarrow a^{\langle t \rangle}$) is the same for every time step
- $\underline{\underline{W}}_{aa}$ ($a^{\langle t-1 \rangle} \rightarrow a^{\langle t \rangle}$) is the same for every time step
- $\underline{\underline{W}}_{ya}$ ($a^{\langle t \rangle} \rightarrow y^{\langle t \rangle}$) is the same for every time step

- **Forward propagation** :

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

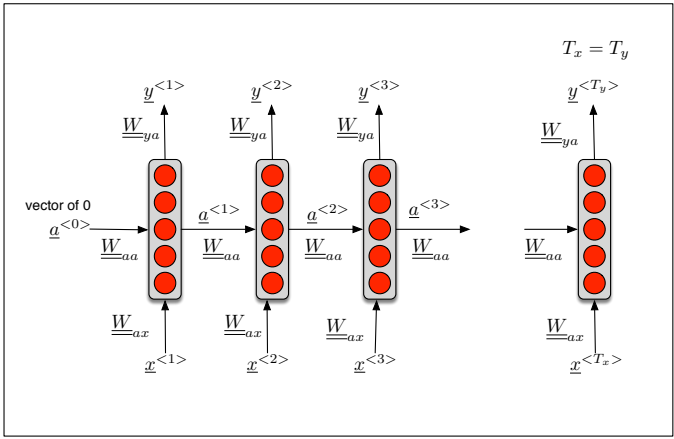
$$a^{\langle 1 \rangle} = g_1\left(W_{aa}a^{\langle 0 \rangle} + W_{ax}x^{\langle 1 \rangle} + b_a\right)$$

$$\hat{y}^{\langle 1 \rangle} = g_2\left(W_{ya}a^{\langle 1 \rangle} + b_y\right)$$

$$\dots$$

$$a^{\langle t \rangle} = g_1\left(W_{aa}a^{\langle t-1 \rangle} + W_{ax}x^{\langle t \rangle} + b_a\right)$$

$$\hat{y}^{\langle t \rangle} = g_2\left(W_{ya}a^{\langle t \rangle} + b_y\right)$$



- **Compact notations :**

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

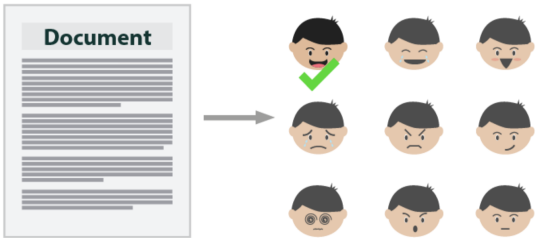
$$a^{\langle t \rangle} = g_1\left(W_a[a^{\langle t-1 \rangle}; x^{\langle t \rangle}] + b_a\right)$$

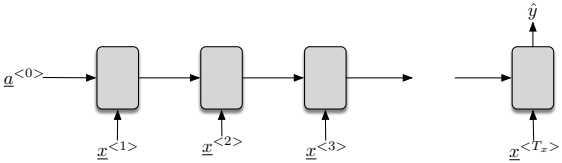
$$\hat{y}^{\langle t \rangle} = g_2\left(W_y a^{\langle t \rangle} + b_y\right)$$

Various types of sequential data

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





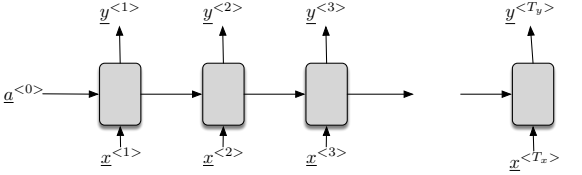
- **Many-to-one** : sentiment classification
  - x = text (movie review)
  - $y = 0/1$ or $1 \cdots 5$ rating

Various types of sequential data/ Many-To-Many $T_y = T_x$

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.

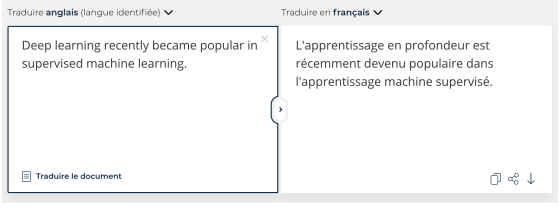
Tag colours:
LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE

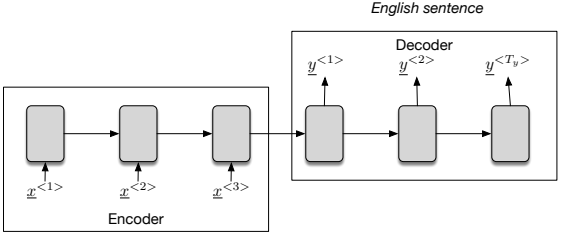


- **Many-to-many** : $T_x = T_y$

Various types of sequential data/ Many-To-Many $T_y \neq T_x$

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



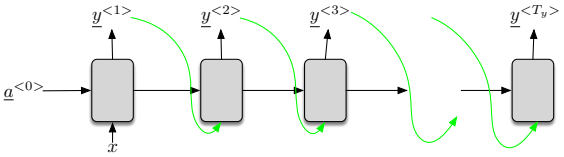


- **Many-to-many** : $T_x \neq T_y$: Machine translation
  - Encoder/Decoder
  - **Attention** mechansim
  - **Transformer** architecture

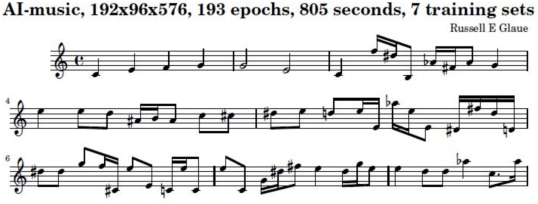
Various types of sequential data/ One-To-Many $T_x = 1, T_y > 1$

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.]]
```

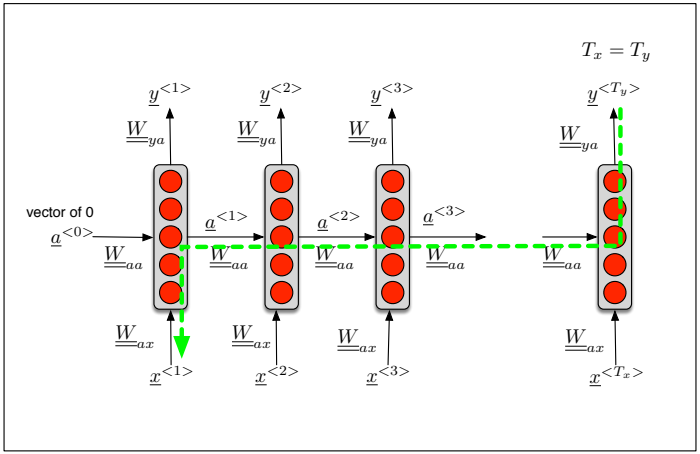


- **One-to-many :** text, code, music generation



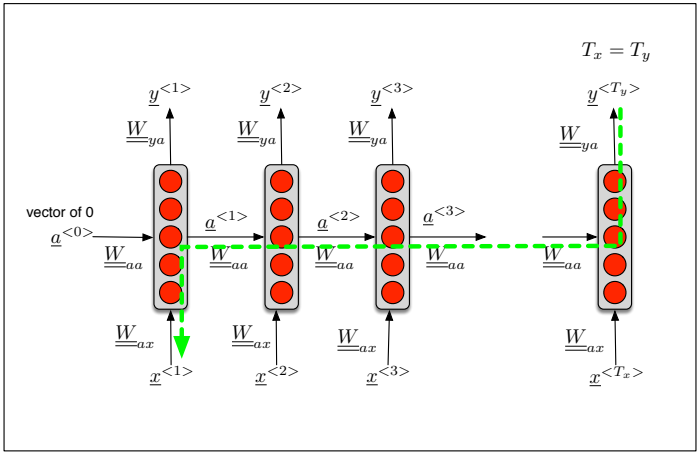
Different architectures

Bi-directional RNN



- RNN scans through the data from left to right ⇒ can do skip connections 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."
- ⇒ we can predict (since RNN depends on the past)

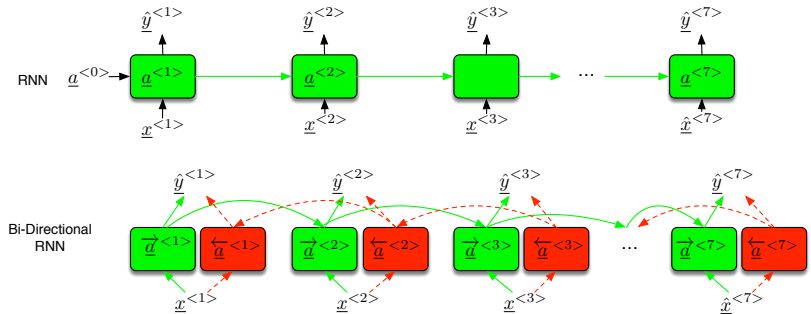
Bi-directional RNN



- RNN scans through the data from left to right ⇒ can do skip connections 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

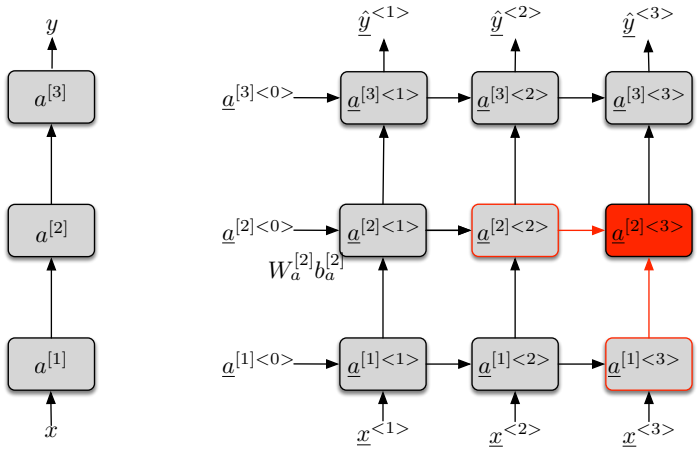
- Solution ?
  - Bi-directional RNN (BRNN)
- Question : is "Apple" a named entity ?
  - "Apple stores sale the new **iPhone**."
  - "Apple juice is a refreshing and healthy **beverages**."
- $\Rightarrow$ we can predict since we now depend on the past and the future



- The prediction $\hat{y}^{\langle t \rangle}$ is done from the concatenation of
  - the **left-to-right** $\overrightarrow{a}^{\langle t \rangle}$ and
  - the **right-to-left** $\overleftarrow{a}^{\langle t \rangle}$ hidden states of RNN

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

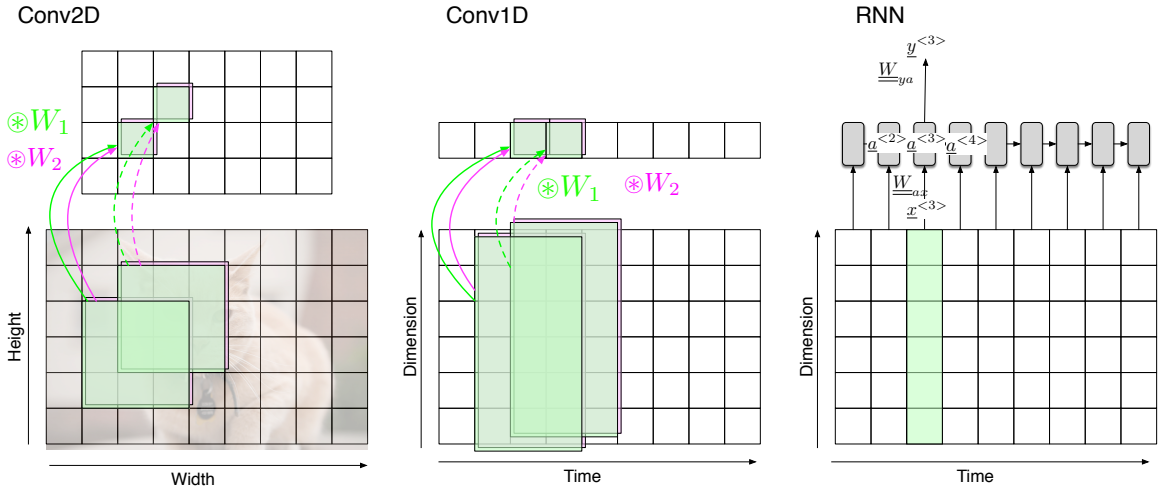
Deep RNN



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

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

Using 1D Convolution instead of RNN



Forward pass

- Compute $a^{\langle t \rangle}$, $\hat{y}^{\langle t \rangle}$, $\mathcal{L}^{\langle t \rangle}$, $\mathcal{L}$

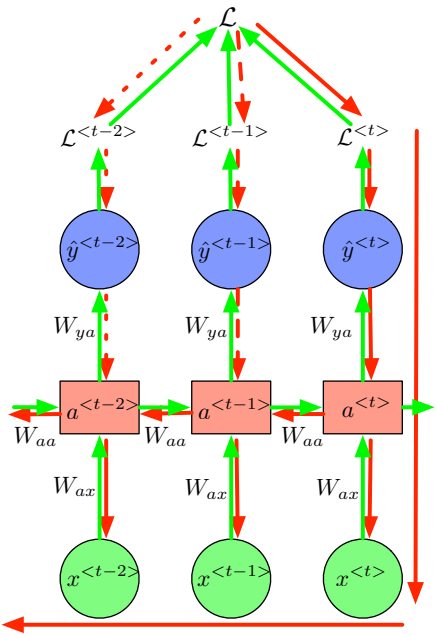
$$a^{\langle t \rangle} = g_a(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}_t = g_y(W_{ya}a^{\langle t \rangle} + b_y)$$

- Compute the Loss
  - Loss for time $< t >$: $\mathcal{L}^{\langle t \rangle}(y^{\langle t \rangle}, \hat{y}^{\langle t \rangle})$
    - $y^{\langle t \rangle}$: true label
    - $\hat{y}^{\langle t \rangle}$: predicted label
  - Total loss :

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

- **Backward pass**
  - compute $\frac{\partial \mathcal{L}}{\partial W_{ya}}$, $\frac{\partial \mathcal{L}}{\partial W_{ax}}$, $\frac{\partial \mathcal{L}}{\partial W_{aa}}$, $\frac{\partial \mathcal{L}}{\partial b_x}$, $\frac{\partial \mathcal{L}}{\partial b_a}$
  - All weights are shared across time steps ! ! !



Backward pass : 1) compute $\frac{\partial L}{\partial W_{ya}}$

- We measure how much varying W_{ya} affect $\mathcal{L}^{\langle t \rangle}$.

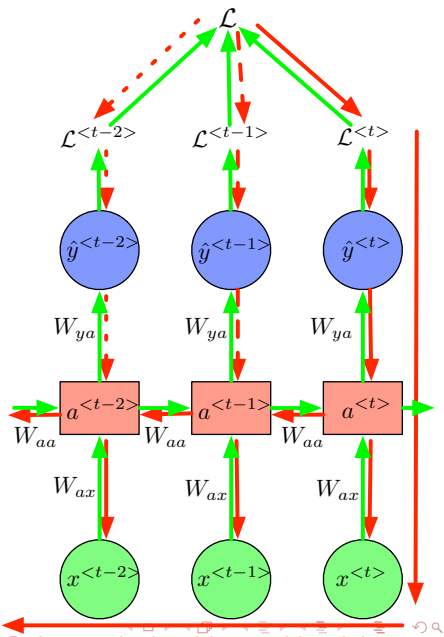
$$\frac{\partial \mathcal{L}}{\partial W_{ya}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial W_{ya}}$$

- We can use the chain rule.

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

- This is very simple since $\hat{y}_t$ depends on W_{ya} on one place (indeed $a^{\langle t \rangle}$ does not depend on W_{ya}).

$$\hat{y}^{\langle t \rangle} = g_y(W_{ya} a^{\langle t \rangle} + b_y)$$



Backward pass : 2) compute $\frac{\partial \mathcal{L}}{\partial W_{aa}}$

- We measure how much varying W_{aa} affect $\mathcal{L}^{\langle t \rangle}$.

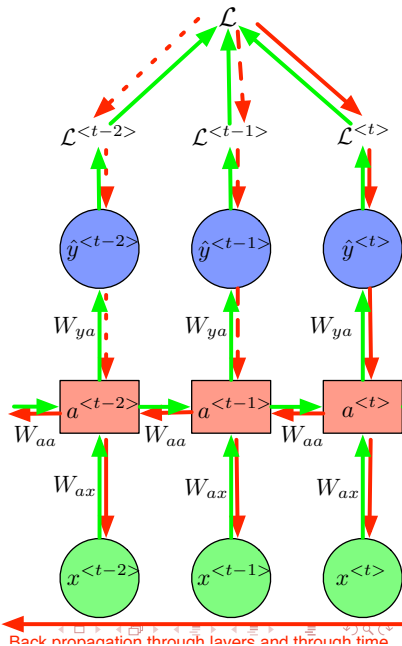
$$\frac{\partial \mathcal{L}}{\partial W_{aa}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial W_{aa}}$$

- We cannot use the chain rule ($\frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial W_{aa}} = \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\partial \hat{y}^{\langle t \rangle}}{\partial a^{\langle t \rangle}} \frac{\partial a^{\langle t \rangle}}{\partial W_{aa}}$) because a_t depends on W_{aa} also through $a^{\langle t-1 \rangle}$ which itself ...

$$a^{\langle t \rangle} = g_a(W_{ax} x^{\langle t \rangle} + W_{aa} a^{\langle t-1 \rangle} + b_a)$$

- We use a formula of total derivative

$$\begin{aligned}
&\frac{d\mathcal{L}^{\langle t \rangle}}{dW_{aa}} \\
&= \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\partial \hat{y}^{\langle t \rangle}}{\partial a^{\langle t \rangle}} \left(\frac{\partial a^{\langle t \rangle}}{\partial W_{aa}} + \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \frac{\partial a^{\langle t-1 \rangle}}{\partial W_{aa}} + \ldots + \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \cdots \frac{\partial a^{\langle 0 \rangle}}{\partial W_{aa}} \right) \\
&= \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\hat{y}^{\langle t \rangle}}{a^{\langle t \rangle}} \sum_{k=0}^{t} \frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} \cdots \frac{\partial a^{\langle k+1 \rangle}}{\partial a^{\langle k \rangle}} \frac{\partial a^{\langle k \rangle}}{\partial W_{aa}} \\
&= \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial \hat{y}^{\langle t \rangle}} \frac{\partial \hat{y}^{\langle t \rangle}}{\partial a^{\langle t \rangle}} \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-1 \rangle}} \right) \frac{\partial a^{\langle k \rangle}}{\partial W_{aa}}
\end{aligned}$$



Reminder : total derivative

- **Total derivative**
  - different from its corresponding partial derivative (∂).
  - calculation of the total derivative of f with respect to t
    - does not assume that the other arguments are constant while t varies ;
    - instead, it assumes that the other arguments too depend on t
    - includes these indirect dependencies to find the overall dependency of f on t.
- Total derivative of $f(t, x(t), y(t))$ with respect to t is
$$\frac{df}{dt} = \frac{\partial f}{\partial t}\frac{dt}{dt} + \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$
- **Example :** $f(x, y) = xy$
  - Partial derivative of f with respect to x :
$$\frac{\partial f}{\partial x} = y$$
    - if y depends on x (suppose we are constrained to the line $y = x$), the **partial derivative** does not give the true rate of change of f as x changes because it holds y fixed.
  - Total derivative
$$\frac{df}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} = y + x \cdot 1 = x + y = 2x$$
  - Notice that this is not equal to the partial derivative :
$$\frac{df}{dx} = 2x \neq \frac{\partial f}{\partial x} = y = x$$

Backward pass : 3) compute $\frac{\partial \mathcal{L}}{\partial W_{ax}}$

- We measure how much varying W_{ax} affect $\mathcal{L}^{\langle t \rangle}$.

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

- It is also necessary to go backwards in time to calculate $\frac{\partial \mathcal{L}}{\partial W_{ax}}$
  - same situation as with W_{aa} :
  - $a^{\langle t \rangle}$ depends on W_{ax} also through $a^{\langle t-1 \rangle}$ which itself ...

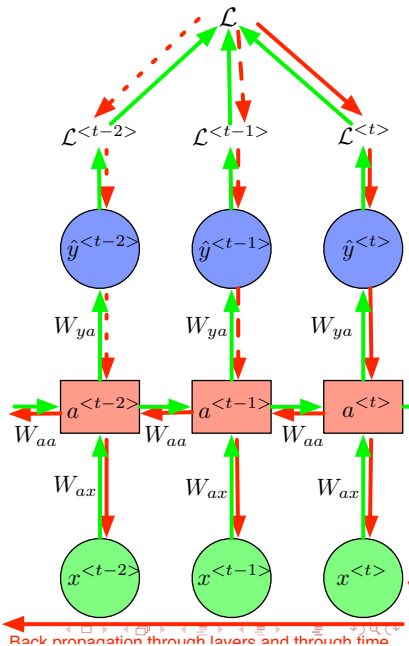
$$a^{\langle t \rangle} = g_a(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

- We use a formula of total derivative

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

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

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



Vanishing and exploding gradients

Vanishing gradient

- 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.
- Very deep NN
  - very difficult to propagate back the gradient to affect the weights of the early Layers

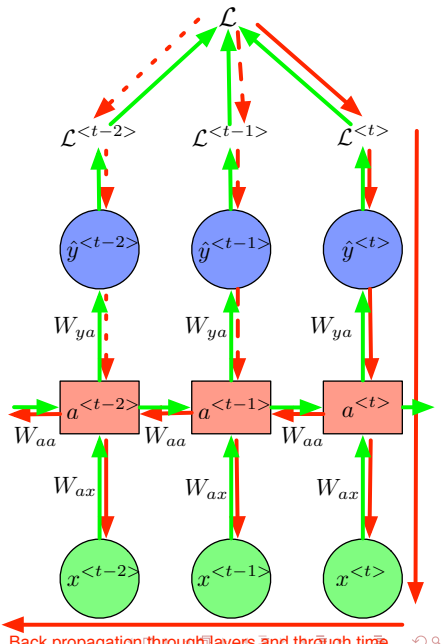
Vanishing and exploding gradients

- To train an RNN we need to backpropagate through layers and through time

$$\frac{\partial \mathcal{L}}{\partial W_{aa}} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial W_{aa}}$$

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

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



Back propagation through layers and through time

Vanishing and exploding gradients

$$\frac{\partial \mathcal{L}}{\partial W_{aa}} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial W_{aa}}$$

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

- Suppose only one hidden units, then a_i is a scalar and consequently $\frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-\mathbf{1} \rangle}}$ is also a scalar
  - If $\left| \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-\mathbf{1} \rangle}} \right| < 1$ then the product goes to 0 exponentially fast
  - If $\left| \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-\mathbf{1} \rangle}} \right| > 1$ then the product goes to infinity exponentially fast
- Vanishing gradients
  - $\left| \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-\mathbf{1} \rangle}} \right| < 1$
    - contributions from faraway steps vanish and don't affect the training
    - difficult to learn long-range dependencies
- Exploding gradients
  - $\left| \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-\mathbf{1} \rangle}} \right| > 1$
    - make the learning process unstable
    - gradient could even become NaN

Vanishing and exploding gradients

- For more than one neuron, a is a matrix, Jacobian
  - use spectral matrix norm, largest singular value of the matrix
- $\left\| \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-\mathbf{1} \rangle}} \right\|_2 < 1$
  - the product goes to zero-norm exponentially fast
- $\left\| \frac{\partial a^{\langle i \rangle}}{\partial a^{\langle i-\mathbf{1} \rangle}} \right\|_2 > 1$
  - the product goes to a matrix infinite norm exponentially fast

Vanishing and exploding gradients

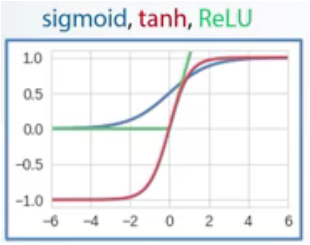
Is it really a problem in practice ?

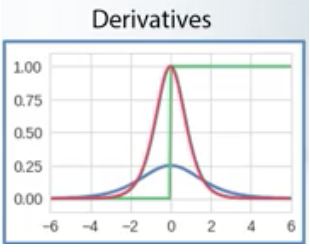
$$a^{\langle t \rangle} = g_a(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_h)$$
$$= g_a(inp^{\langle t \rangle})$$

where $inp^{\langle t \rangle}$ is pre-activation at time step t

$$\frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} = \frac{\partial a^{\langle t \rangle}}{\partial inp^{\langle t \rangle}} \frac{\partial inp^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}}$$
$$= \text{diag}(g_a'(inp^{\langle t \rangle})) \cdot W_{aa}$$

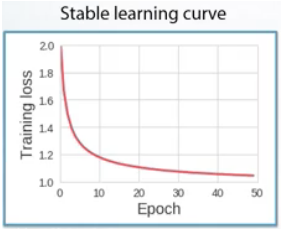
- $\text{diag}(g_a'(inp^{\langle t \rangle}))$ depends on the type of non-linearity used
  - sigmoid and tanh :
    - vanishing gradients are very likely
  - ReLU :
    - OK for the positive part, but the gradient can vanish for the negative part
- $\|W\|$ may be either small or large
  - Small $\|W\|$ could increase the vanishing gradient problem
  - Large $\|W\|$ could increase the exploding gradient problem (especially with ReLu)

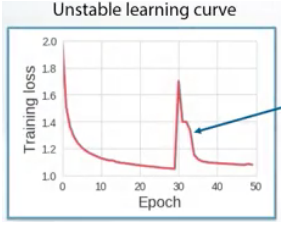




Dealing with the exploding gradient problem

- **Detection ?**



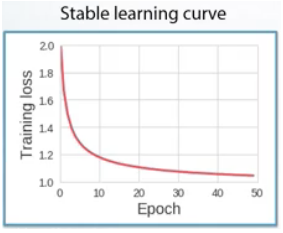


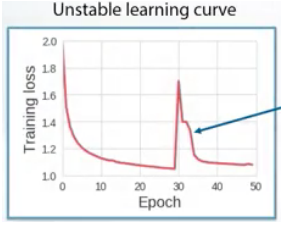
Dealing with the exploding gradient problem

- **Solution 1 : gradient clipping**
  - gradient $d\theta = \frac{\partial \mathcal{L}}{\partial \theta}$, where θ are all the parameters of the network
  - if $\|d\theta\| > threshold$

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

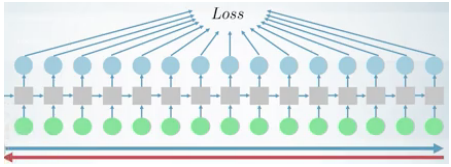
  - clipping doesn't change the direction of the gradient but change its length
  - we can clip only the norm of the part which causes the problem
  - we choose the threshold manually : start with a large threshold and then reduce it

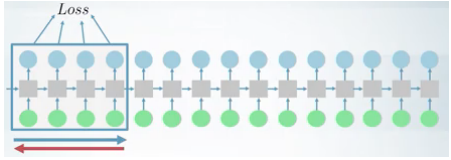


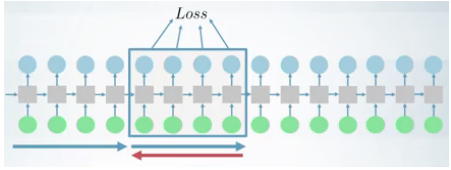


Dealing with the exploding gradient problem

- **Solution 2 : 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 backpropagate 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 problem

- Contributions from faraway steps vanish and don't affect the training
- It is difficult to learn long-range dependencies
- **Detection ?** : difficult to detect
  - the learning curve looks OK but the loss stops decreasing
    - Is it due
      - to the vanishing gradient problem ?
      - or the task is difficult to be solved ?
    - The gradient norm is small $\| \frac{\partial \mathcal{L}^{\langle t \rangle}}{\partial a^{\langle t-\mathbf{100} \rangle}} \|_2$ is small
      - but this maybe due to the fact that there is no long term dependencies
- **Solution ?**
  - use ReLU activation function
  - initialization of the recurrent weight Matrix
  - skip connections
    - gated units (LSTM, GRU)

Dealing with the vanishing gradient problem

- The Jacobian matrix depends on the

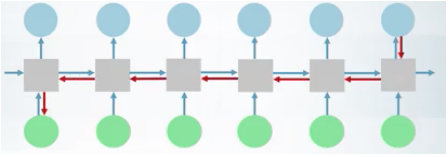
$$\frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} = \frac{\partial a^{\langle t \rangle}}{\partial inp^{\langle t \rangle}} \frac{\partial inp^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}}$$
$$= \text{diag}(g_a'(inp_t)) \cdot W_{aa}$$

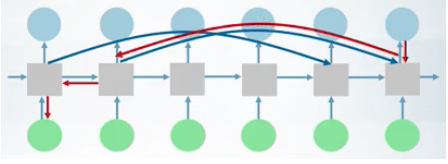
  - choice of the activation function : use ReLu
  - values of the recurrent weights W_{aa}

- **Solution 1 : orthogonalization**
  -Q is orthogonal if $Q^T = Q^{-1}$
  - then $\prod_i Q_i$ doesn't explode or vanish
  - initialize W_{aa} with an orthogonal matrix
  - Some approach use orthogonalization of W_{aa} through the whole training

Dealing with the vanishing gradient problem

- **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
  - Backpropagation through the shortcuts : the gradients vanish slower
    - we can learn long-term dependencies
  - not exclusive to RNN (see ResNet)



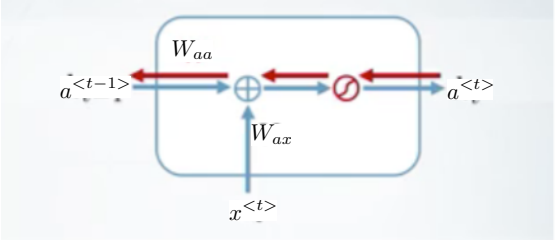


Gated units (LSTM and GRU)

Main ideas : Recurrent Neural Network (RNN)

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

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



Main ideas : Long Short Term Memory Units (LSTM)

- Add a **memory cell**
  - the same dimension as hidden layer
- **Simplified LSTM** (no possibility to erase anything) :

Candidate cell value

$$\tilde{c}^{\langle t \rangle} = g\left(V_g x^{\langle t \rangle} + W_g a^{\langle t-1 \rangle} + b_g\right)$$

Update gate

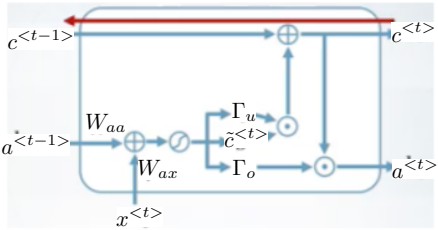
$$\Gamma_u = \sigma\left(V_u x^{\langle t \rangle} + W_u a^{\langle t-1 \rangle} + b_u\right)$$

Output gate

$$\Gamma_o = \sigma\left(V_o x^{\langle t \rangle} + W_o a^{\langle t-1 \rangle} + b_o\right)$$

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

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



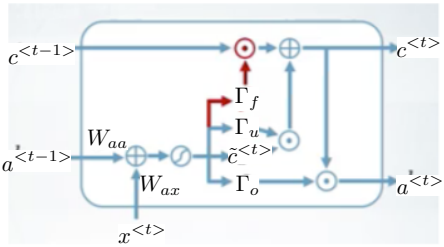
- no non-linearity or multiplication in the memory cell-path
- $\frac{\partial c^{\langle t \rangle}}{\partial c^{\langle t-1 \rangle}} = diag(1)$
- no vanishing gradients ! ! !

Main ideas : Long Short Term Memory Units (LSTM)

- **Forget gate LSTM** :

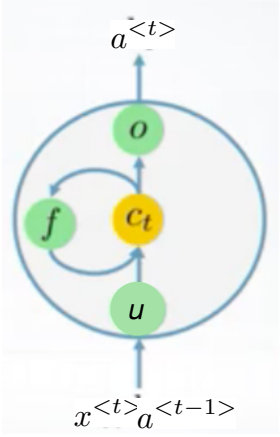
  Forget gate

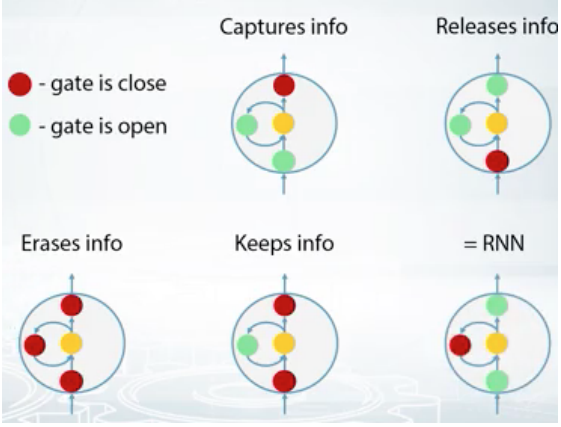
$$\Gamma_f = \sigma\left(V_f x^{\langle t\rangle} + W_f a^{\langle t-1\rangle} + b_f\right)$$
$$c^{\langle t\rangle} = \Gamma_f \odot c^{\langle t-1\rangle} + \Gamma_u \odot \tilde{c}^{\langle t\rangle}$$
$$a^{\langle t\rangle} = \Gamma_o \odot f(c^{\langle t\rangle})$$



- We now have a multiplication on the short-way
  - $\frac{\partial c^{\langle t\rangle}}{\partial c^{\langle t-\mathbf{1}\rangle}} = \text{diag}(\Gamma_f)$
  - with Γ_f the forget gate : $\Gamma_f = \sigma(V_f x^{\langle t\rangle} + W_f a^{\langle t-1\rangle} + b_f)$
  - Γ_f takes value between 0 and 1
    - $\Rightarrow$ set a high initial value for b_f
    - at the beginning of the training, the LSTM doesn't forget and can find long-term dependencies in the data
- LSTM are very nice but has four times more parameters than RNN

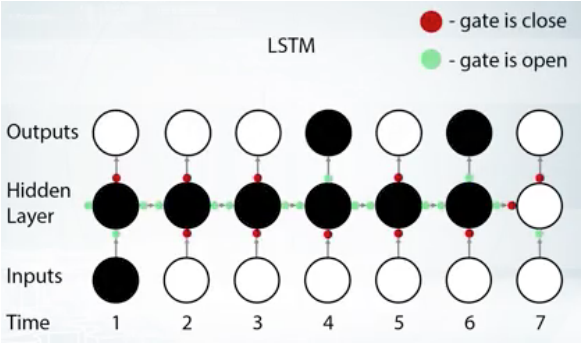
LSTM regimes





Main ideas : Long Short Term Memory Units (LSTM)

- LSTM regimes



Main ideas : Gated Recurrent Unit (GRU)

- **Fewer gates**

Relevance gate

$$\Gamma_r = \sigma\left(V_r x^{\langle t \rangle} + W_r a^{\langle t-1 \rangle} + b_r\right)$$

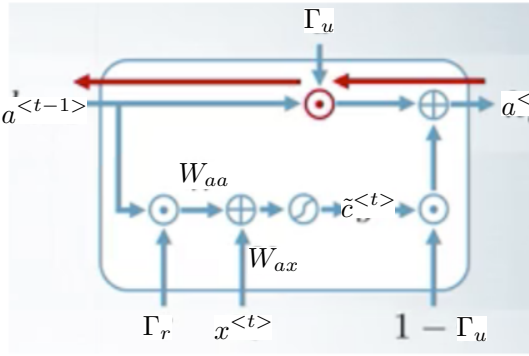
Candidate cell value

$$\tilde{c}^{\langle t \rangle} = g(V_g x^{\langle t \rangle} + W_g(a^{\langle t-1 \rangle} \cdot \Gamma_r) + b_g)$$

Update gate

$$\Gamma_u = \sigma(V_u x^{\langle t \rangle} + W_u a^{\langle t-1 \rangle} + b_u)$$

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



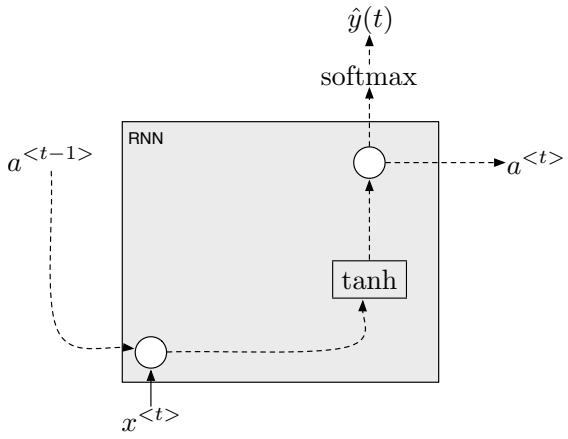
- Vanishing gradient problem ?
  - similar as for LSTM

$$\frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} = \text{diag}(1 - \Gamma_u) \cdot \frac{\partial \tilde{c}^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} + \text{diag}(\Gamma_u)$$

  - $\Rightarrow$ high initial value for b_u

Recurrent Neural Network (RNN)

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



Long Short Term Memory Units (LSTM)

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

Candidate cell value

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

Forget gate

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

Update gate

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

Cell update

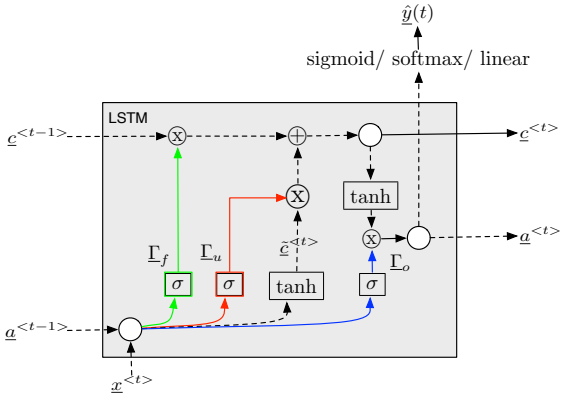
$$c^{<t>} = \Gamma_f \odot c^{<t-1>} + \Gamma_u \odot \tilde{c}^{<t>}$$

Output gate

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

Output

$$a^{<t>} = \Gamma_o \odot \tanh(c^{<t>})$$



Gated Recurrent Unit (GRU)

[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"]

- **Simple GRU**

Candidate cell value

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

Update gate

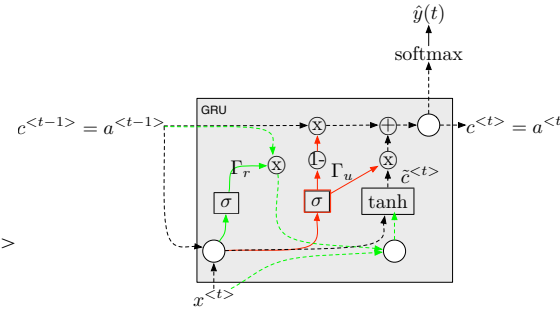
$$\Gamma_u = \sigma(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

Cell update

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

Output

$$c^{<t>} = a^{<t>}$$



Gated Recurrent Unit (GRU)

[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"]

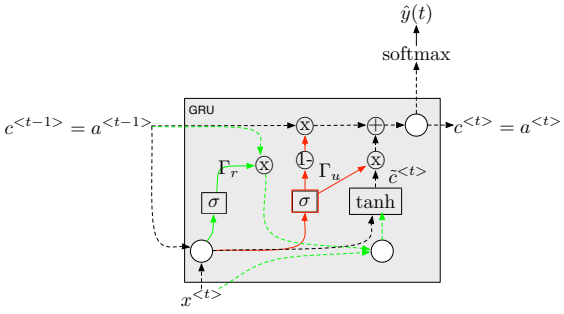
- **Full GRU**
  - adding a relevance gate Γ_r
  - instead of using all $c^{<t-1>}$ to compute $\tilde{c}^{<t>}$, we will first filter it using a "Relevant gate" Γ_r.
  - The "Relevant gate" Γ_r is computed using $c^{<t-1>}$ and $x^{<t>}$.

Relevant gate

$$\Gamma_r = \sigma\left(W_r[c^{<t-1>}, x^{<t>}] + b_c\right)$$

Cell update

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r \odot c^{<t-1>}, x^{<t>}] + b_c)$$

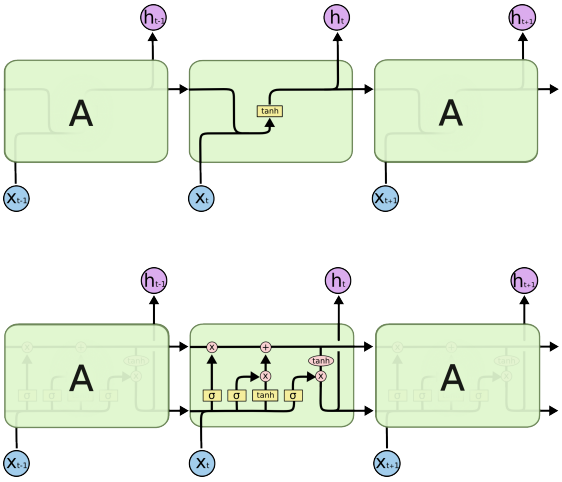


LSTM Example usage

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

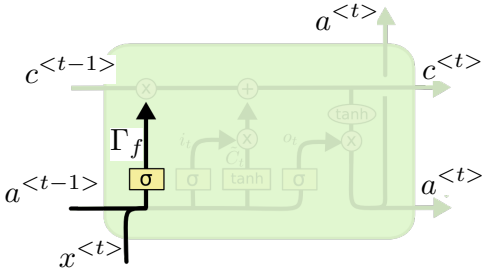
Instead of having a single neural network layer, there are four, interacting in a very special way.

- 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



LSTM Example usage

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



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

Action

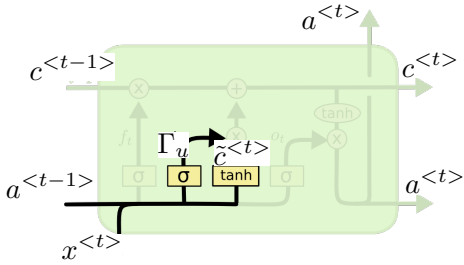
- The sigmoid (forget gate) answers for each component :
  - 1 : to keep it,
  - 0 to forget it, or
  - a value in-between to mitigate its influence

Intuition for language modeling

- 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

LSTM 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

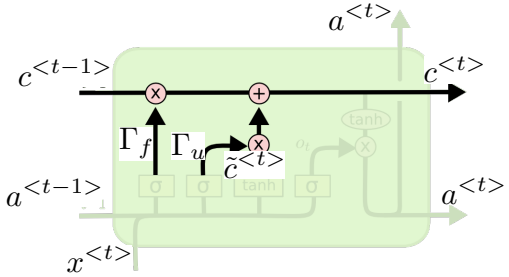
- Create the update $\tilde{c}^{\langle t \rangle}$ of the cell state
- and its contribution Γ_u (the update gate with a sigmoid activation)

Intuition for language modeling

- Add the gender of the new subject to the cell state,
- to replace the old one we're forgetting. appears

LSTM Example usage

- **Write the new state**



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

Action

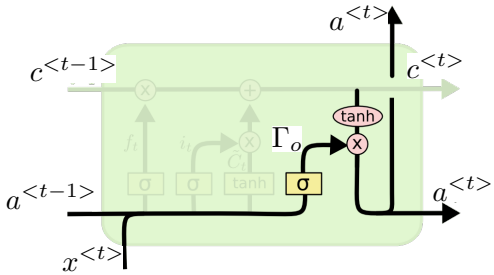
- Merge the old cell state modified by the forget gate
- with the new input

Intuition for language modeling

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

LSTM 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 modeling

- Since we just saw a subject,
- output the relevant information for the future (gender, numnber) appears

Natural Language Processing

Language model

- **Automatic Speech Recognition** :
  - "This is an example of an homophone." or "This is an egg-sample of an homophone."
    - $p(\text{"}example\text{"}) = 5.7 \cdot 10^{-10}$
    - $p(\text{"}egg - sample\text{"}) = 3.2 \cdot 10^{-13}$
- **Language model :** $p(sentence)$
  - $p(y^{\langle 1 \rangle}, y^{\langle 2 \rangle}, \ldots, y^{\langle T_y \rangle})$
- **How to build a language model ?**
  - need a large corpus of English text
  - tokenize the text
    - convert to one-hot-vector
    - \+ <EOS> (end of sentence)
    - \+ <UNK> (unknown word, very uncommon vocalular) : Gif-sur-Yvette

Language model

- **Notation :**

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

- **Applications**
  - Automatic Speech Recognition, Machine Translation, OCR, ...
- **Goal**
  - **Estimate the (non-zero) probability of a word sequence for a given vocabulary**

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

  - with the **N-gram assumption** :

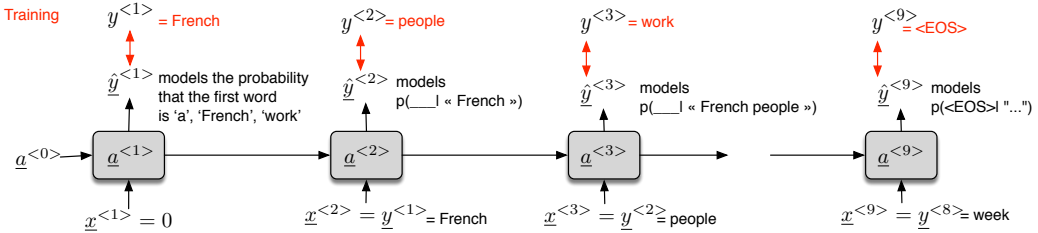
    $$P(w_1^L) = \prod_{i=1}^{L} P(w_i | w_{i-n+1}^{i-1}), \forall i, w_i \in V$$

  - in the **recurrent way** :

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

Language model/ ① Training using a RNN

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



- **Training :**
  - output of the network (softmax) is $\hat{y}^{\langle t \rangle}$ (probability of each word)
  - ground-truth is $y^{\langle t \rangle}$ (one-hot-encoding)
  - minimize the loss

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

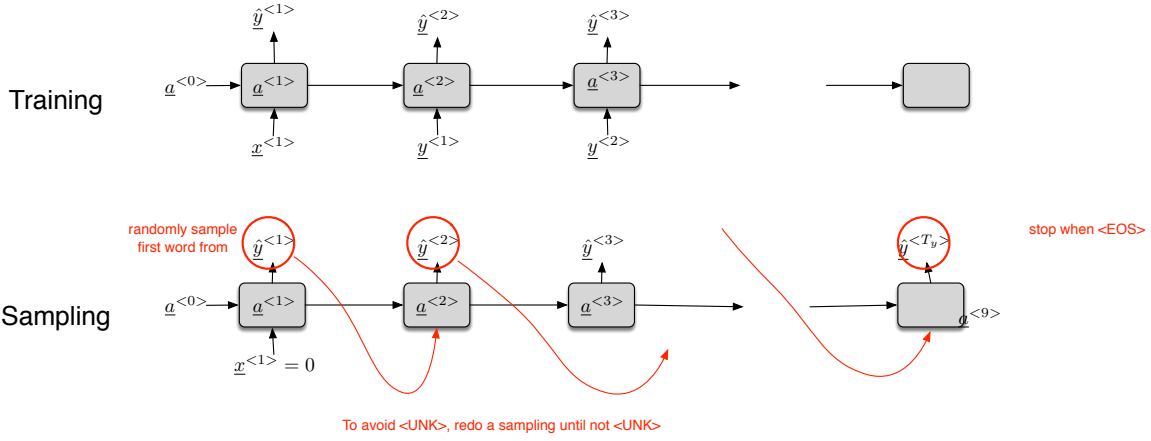
$$\mathcal{L} = \sum_t \mathcal{L}_t(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle})$$

Language model/ ② Testing using a RNN

- Given a new sentence of three words $(y^{(1)}, y^{(2)}, y^{(3)})$
  - what is the probability of this sentence ?

$$
\begin{aligned}
P(y^{(1)}, y^{(2)}, y^{(3)}) = & p(y^{(1)}) && \text{given by the first softmax} \\
& p(y^{(2)}|y^{(1)}) && \text{given by the second softmax} \\
& p(y^{(3)}|y^{(1)}, y^{(2)}) && \text{given by the third softmax}
\end{aligned}
$$

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 vector o_{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 generalize
- Example :
  - we have a trained a language model to complete the sentence
    - The president is on a **boat** __?__" $\Rightarrow$ "trip"
  - since there is no specific relationship between "boat" and "plane", the algorithm cannot complete the sentence
    - The president is on a **plane** __?__" $\Rightarrow$?

Word embedding

Word embedding

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

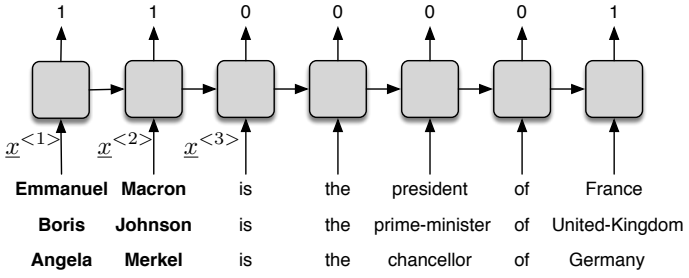
	Man	Woman	King	Queen	Uncle	Aunt	President	Chancelor	France	Germany	Boat	Plane
	5391	9853	4914	7157	456	6257	7124	3212	6789	1234	5923	8871
Gender	-1	1	-1	1	-1	1	-0,7	-0,3	0	0	0	0
Royal	0	0	1	1	0	0	0,5	0,5	0	0	0	0
Age	0	0	0,7	0,7	0,5	0,5	0,7	0,7	0	0	0	0
Country	0	0	0,5	0,5	0	0	0,5	0,2	1	1	0	0
Transportation	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 generalization) that
  - The president is on a **boat** __?__ " $\Rightarrow$ "trip"
  - The president is on a **plane** __?__ " $\Rightarrow$ "trip"

Word embedding/ How to get Word Embeddings

- ① Train from scratch a Word Embedding matrix
  - require a lot of data (1 to 100 Billion words)
- ② Download from the internet
  - https://fasttext.cc/docs/en/crawl-vectors.html
  - has been already trained on 1 to 100 Billion words
  - perform **transfer learning** :
    - use directly for a task (possibly with a much smaller training set)
    - use feter some fine-tuning the word embeddings with the new training set

Word embedding/ Use for Named Entity Recognition



Word embedding/ Properties

[Mikolov et al, 2013 "Linguistic regularities in continuous space word representations."]

	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
...	...	...	...	...	...	...	...	...	...	...	...	...

- Man $\Rightarrow$ (is to) Woman
- King $\Rightarrow$ (is to) ? $\Rightarrow$ Queen
  - $e_{man} - e_{woman} \simeq e_{king} - e_?$

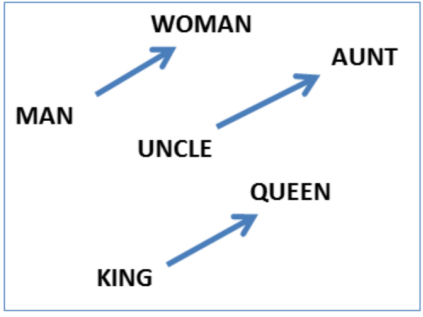
$$e_{man} - e_{woman} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

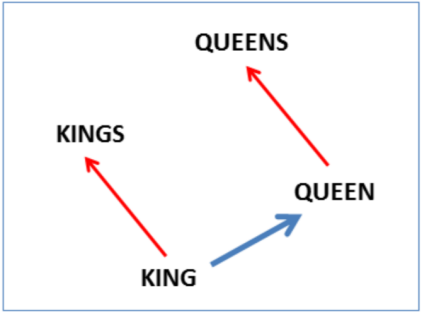
$$e_{king} - e_{queen} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- find w such that $\arg\max_w sim(e_w, e_{king} - e_{man} + e_{woman})$
  - a) cosine similarity :
    - $sim(u, v) = \frac{u^T v}{||u||_2 ||v||_2} = \cos(\phi)$
  - b) square distance, euclidean distance
    - $||u - v||^2$

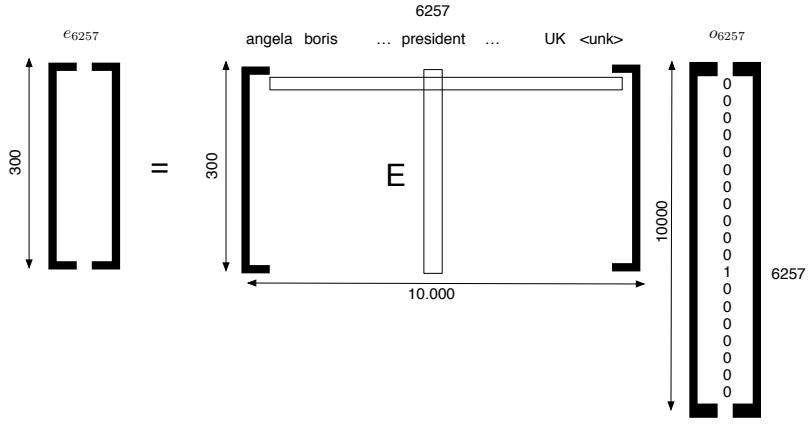
Word embedding/ Properties

[Mikolov et al, 2013 "Linguistic regularities in continuous space word representations."]





Word embedding/ Matrix



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

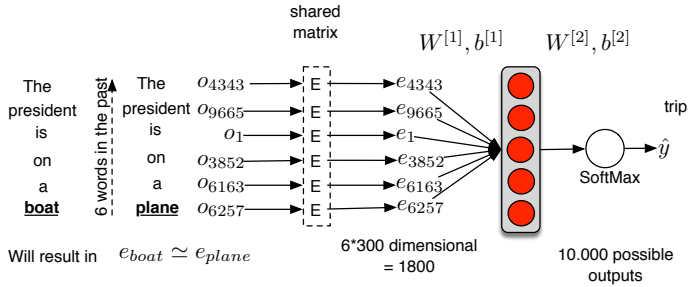
Word embedding/ Learning using

- Various existing methods to learn word embedding :
  - Classic neural language model (Bengio)
  - Collobert and Weston model
  - Word2Vec (Mikolov)
    - Continuous bag-of-words (CBOW)
    - Skip-gram
  - Glove model (Pennington)

Word embedding/ Classic neural language model

[Bengio et al., 2003 "A Neural probabilistic model"]

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



- **Example :**
  - "The president is on a boat *trip*$_{target}$ to the United States."
- **Other possibile 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 __?__ "

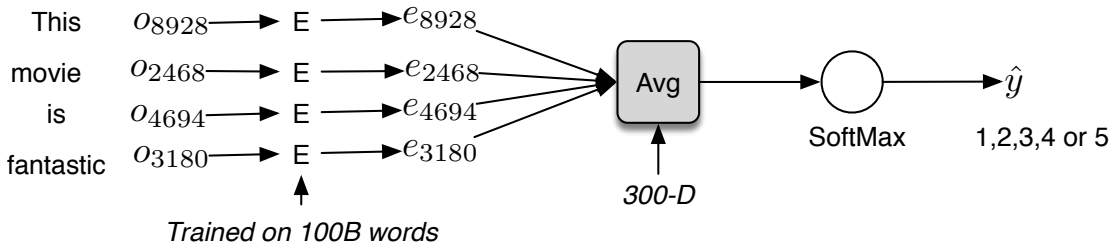
Application : Sentiment Classification

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

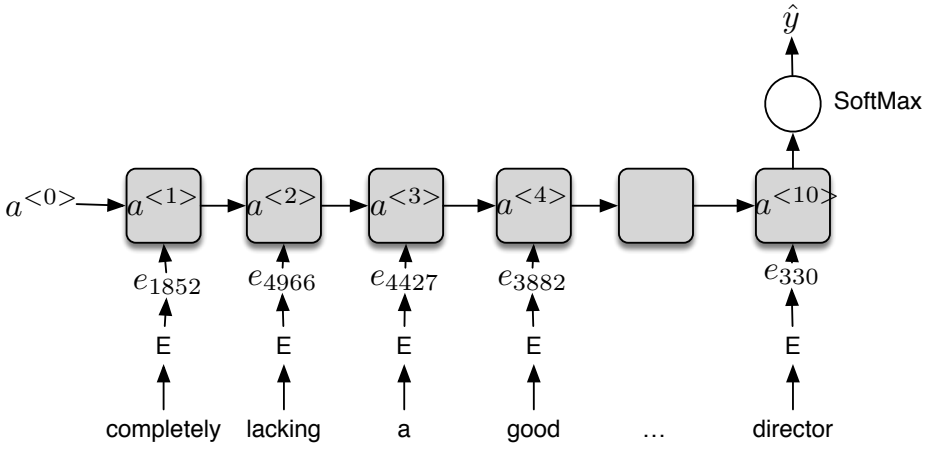
The	movie	is	fantastic
8928	2468	4694	3180



- Does not work for "This movie was completely lacking a **good** script, **good** actors and a **good** director.".
  - Avg= good $\Rightarrow$ does not understand

Application : Sentiment Classification/ using RNN

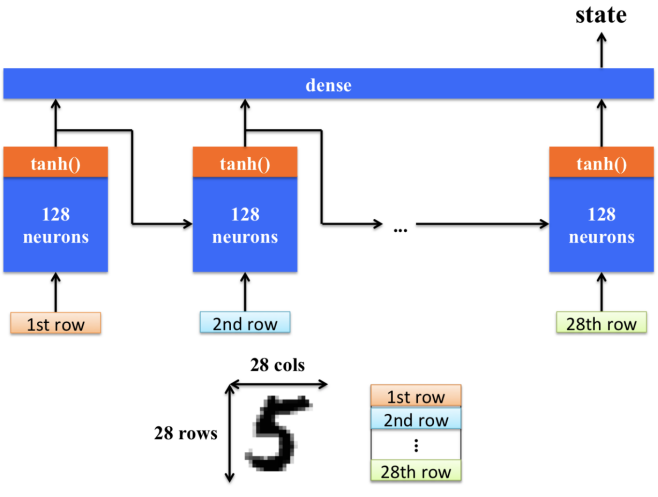
- **Many-to-one** architecture



Application : Handwritten Character Recognition

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

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



RNN data flow

MNIST dataset

Sequence to sequence

Introduction

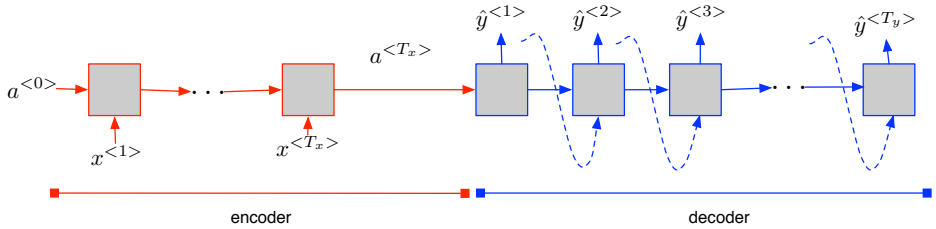
[Sutskever et al. 2014, "Sequence to sequence learning with neural networks"]

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

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

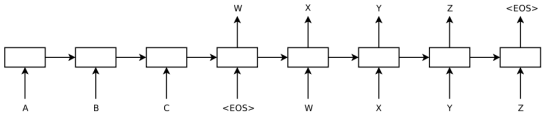
{x}	$x^{<1>}$	$x^{<2>}$	...			$x^{<6>}$		
	Macron	voyage	aux	Etats-Unis	à	Noël		
{y}	$y^{<1>}$	$y^{<2>}$	...					$y^{<8>}$
	Macron	is	travelling	to	the	United-Stat	for	Christmas

- Solution :
  - **Sequence to sequence** [Sutskever] or **Encoder-Decoder** [Cho] architecture

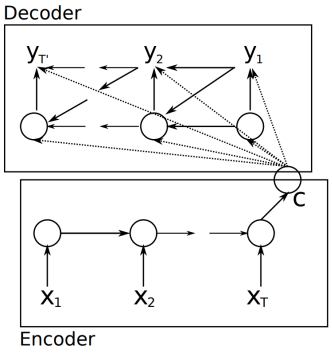


Introduction

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

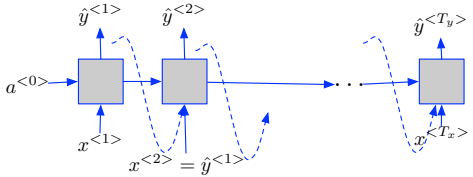


[Cho et al. 2014 "Learning phrase representations using rnn encoder-decoder for statistical machine translation"]

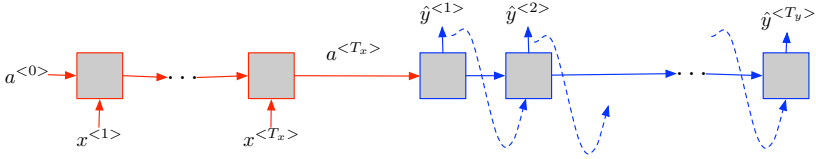


Machine Translation

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



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



Machine Translation/ ④ Decoding (finding the most likely sequence)

- $x^{\langle t \rangle}$= "Macron voyage aux Etats-Unis à Noël"
- $y^{\langle t \rangle}$? possible translations
  - "Macron travels to the United States for Christmas"
  - "Macron is travelling to the United States for Christmas"
  - "Macron is going to travel to the United States for Christmas"
  - "Macron is going to be travelling to the United States for Christmas"
  - "For Christmas, Macron will travel to the United States"
- **Choose the most likely Sequence** :

$$\underset{y^{\langle 1 \rangle} \ldots y^{\langle T_y \rangle}}{\text{argmax}} \; p(y^{\langle 1 \rangle} \ldots y^{\langle T_y \rangle} | x)$$

Machine Translation/ ④ Decoding (finding the most likely sequence)

- **Choose the most likely Sequence** :

$$\underset{y^{\langle 1 \rangle} \ldots y^{\langle T_y \rangle}}{\text{argmax}} \; p(y^{\langle 1 \rangle} \ldots y^{\langle T_y \rangle} | x)$$

- **① Greedy search**
  - at each time, pick the most likely word $\Rightarrow$ not good because

$p(\text{"}\textit{Macron is travelling America}\text{"}|x) > p(\text{"}\textit{Macron is going to be travelling America}\text{"}|x)$

but $p(\text{"}\textit{Macron is travelling}\text{"}|x) < p(\text{"}\textit{Macron is travelling}\text{"}|x)$ (because "going" is very common

- **② Exhaustive search**
  - track all the possibilities
  - intractable
- **③ Approximate search : beam search**
  - only track (keep tracking) the B best possible path at each time

Attention model

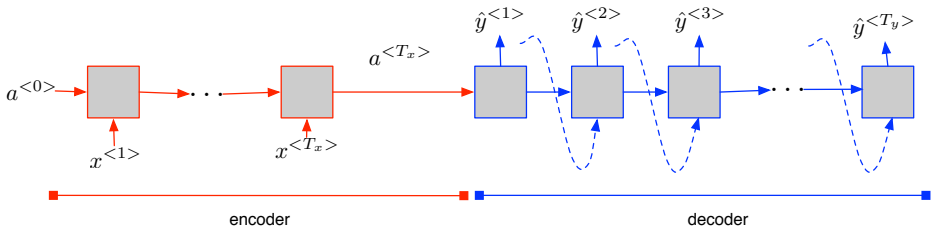
Attention model

[Bahdanau et al. 2014 "Neural machine translation by jointly learning to align and translate"]

- **The problem of long sequences**

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.

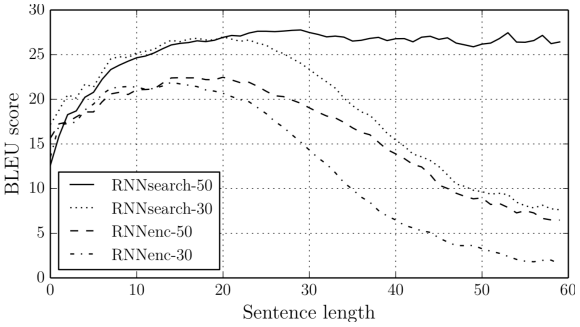
This kind of experience is part of Disney's efforts to "extend the lifetime of its series and build new relationships with audiences via digital platforms that are becoming ever more important," he added.



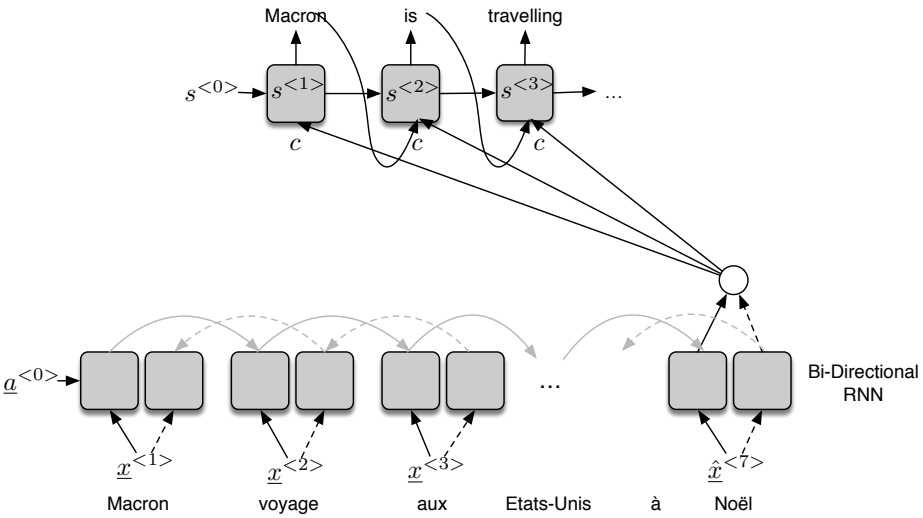
- Encoder/Decoder :
  - $a^{\langle T_x \rangle}$ is supposed to memorize the whole sentence then translate it ;
    - human way : translate each part of a sentence at a time
- **Attention model ?** (modification of the Encoder/Decoder)
  - $a^{\langle T_x \rangle}$ is replaced by a local version in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

Attention model

- Bleu score for long sentences

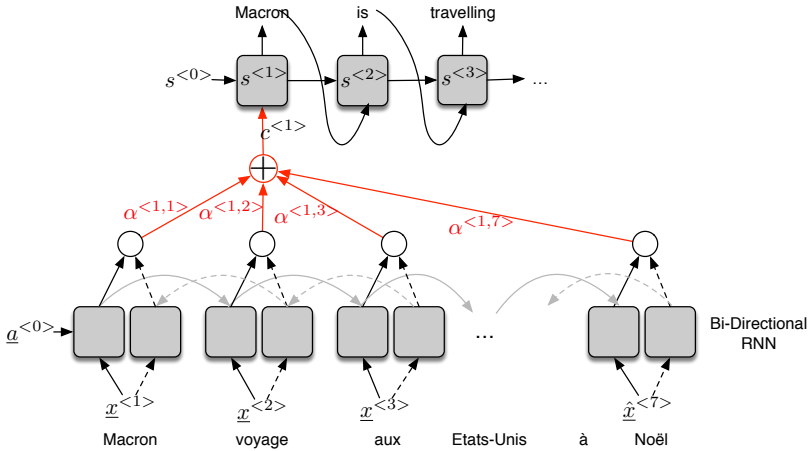


Attention model



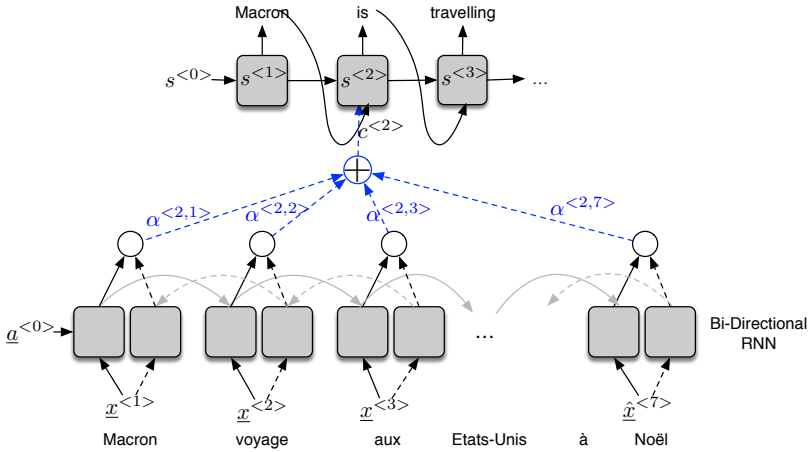
- In usual encoder-decoder,
  - information used for the decoding corresponds to the encoding at T_x : $c = a^{\langle T_x \rangle}$

Attention model



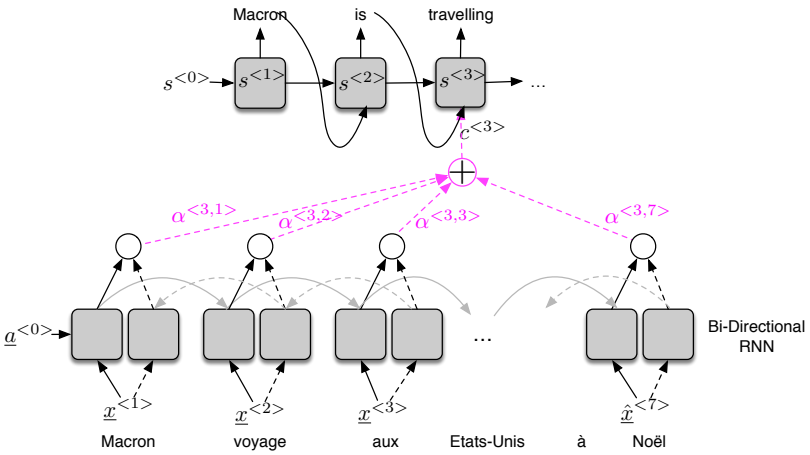
- Attention model
  - replace $c = a^{\langle T_x \rangle}$ by a local version $c^{\langle t \rangle}$
  - $c^{\langle t \rangle}$ is computed as a weighted sum of the encoding hidden states $a^{\langle t \rangle}$
- **Attention weights** $\alpha^{\langle \tau, t \rangle}$:
  - when generating information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

Attention model



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

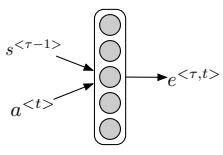
Attention model

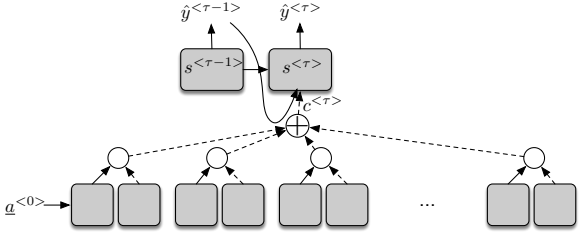


- **Attention weights** $\alpha^{\langle 3,t \rangle}$
  - describe an "alignment" between information at
    - **encoding time** t : computed using $\overrightarrow{a}^{\langle t \rangle}$ and $\overleftarrow{a}^{\langle t \rangle}$
      - we note $a^{\langle t \rangle} = [\overrightarrow{a}^{\langle t \rangle}, \overleftarrow{a}^{\langle t \rangle}]$
    - **decoding time** $\tau = 3$: computed using $s^{\langle 2 \rangle}$

Attention model

- We define each conditional probability as
  - $p(y^{\langle\tau\rangle}|y^{\langle 1\rangle},\dots y^{\langle\tau-1\rangle},x) = g(y^{\langle\tau-1\rangle}, s^{\langle\tau\rangle}, c^{\langle\tau\rangle})$
    - where $s^{\langle\tau\rangle}$ is an RNN hidden state for time τ
    - $s^{\langle\tau\rangle} = f(s^{\langle\tau-1\rangle}, y^{\langle\tau-1\rangle}, c^{\langle\tau\rangle})$
- For each target word $y^{\langle\tau\rangle}$, the probability is conditioned on a distinct **context vector** $c^{\langle\tau\rangle}$
  - $c^{\langle\tau\rangle}$ depends on a sequence of annotations $(a^{\langle 1\rangle},\dots,a^{\langle T_x\rangle})$ to which an encoder maps the input sentence
    - Each $a^{\langle t\rangle}$ contains information about the whole input sequence with a strong focus on the parts surrounding the t-th word of the input sequence





Attention model

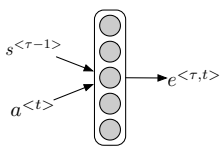
- The **context vector** $c^{\langle \tau \rangle}$ at decoding time τ
  - computed as the sum of the annotations $a^{\langle t \rangle}$ weighted by their **attention weights** $\alpha^{\langle \tau, t \rangle}$

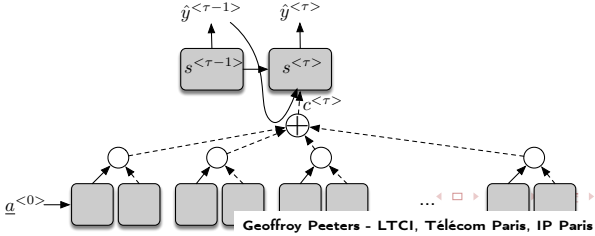
$$c^{\langle \tau \rangle} = \sum_t \alpha^{\langle \tau, t \rangle} a^{\langle t \rangle}$$

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

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

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





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.

- Visualization of $\alpha^{\langle \tau, t \rangle}$

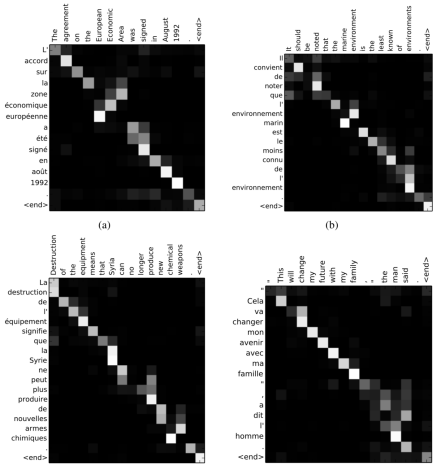
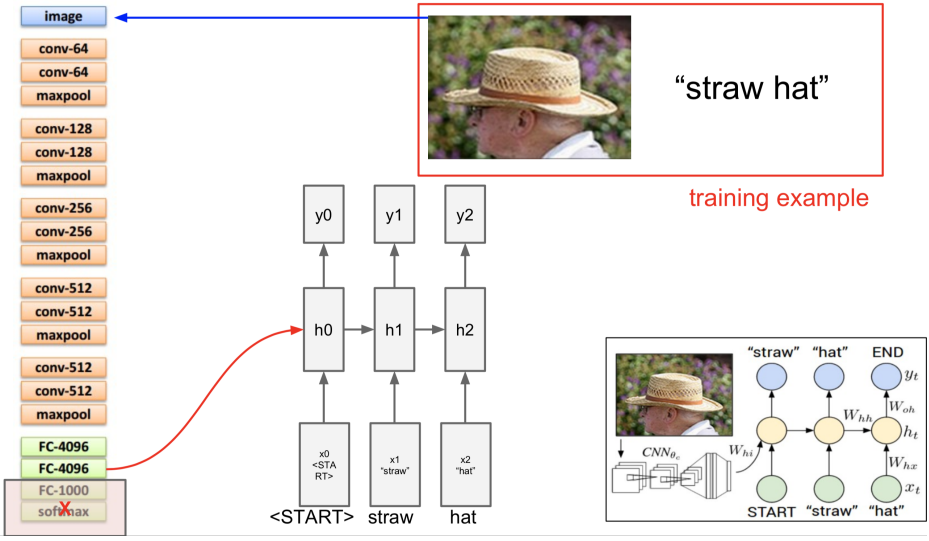


Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word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

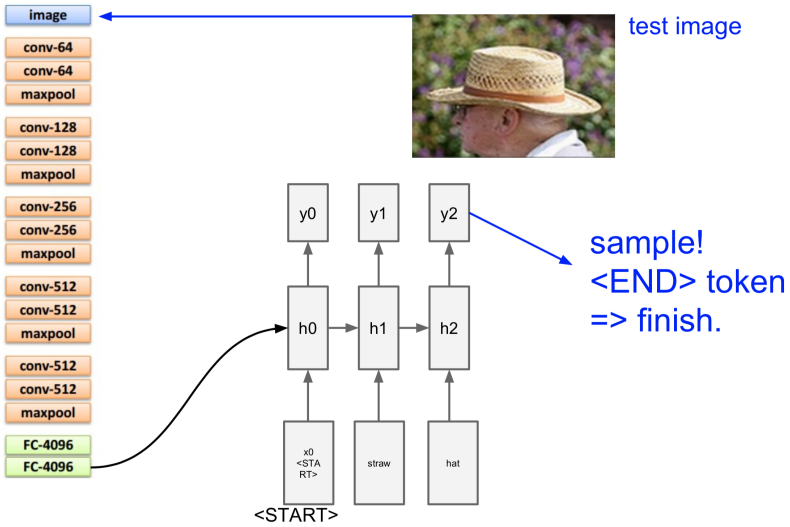
Application : Image captioning

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

- http://cs231n.stanford.edu

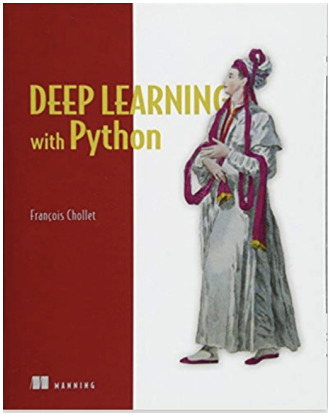


Application : Image captioning



Implementations in keras

- Readings :
  - https://keras.io
  - François Chollet : "Deep Learning with Python"
  - keras Cheat Sheet
  - Attilio Fiandrotti keras slides



MLP in keras

```
[94] # --- Input dimensions are (m, n_in)
     # --- Output dimensions are (m, n_out)
     X_dummy = np.random.randn(20, 100)

     input_shape=(100,)

     X_input  = Input(shape = input_shape, name='Input')
     X        = Dense(128, name='FC1')(X_input)
     X        = Activation('relu', name='Activation1')(X)
     X        = Dense(1, name='FC2')(X)
     X_output = Activation('sigmoid', name='Activation2')(X)

     model = Model(inputs= X_input, outputs=X_output)

     model.summary()
     print('input dimension: ', X_dummy.shape)
     print('output dimension: ', model.predict(X_dummy).shape)
```

```
Model: "model_56"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 100)               0
_________________________________________________________________
FC1 (Dense)                  (None, 128)               12928
_________________________________________________________________
Activation1 (Activation)     (None, 128)               0
_________________________________________________________________
FC2 (Dense)                  (None, 1)                 129
_________________________________________________________________
Activation2 (Activation)     (None, 1)                 0
=================================================================
Total params: 13,057
Trainable params: 13,057
Non-trainable params: 0
_________________________________________________________________
input dimension:  (20, 100)
output dimension:  (20, 1)
```

RNN in keras

```
[101] # --- Input dimensions are
      # ---       (m, seq_len1, n_in)
      X_dummy = np.random.randn(20, 15, 100)
      input_shape=(15, 100,)

      #       (m, seq_len)    if use of embbeding
      #X_dummy =  np.random.randint(0, 10000, (20,15))
      #input_shape=(15,)

      # --- Output dimensions are
      # ---       (m, seq_len1, n_out)    if many-to-many Tx=Ty
      # ---       (m, seq_len2, n_out)    if many-to-many Tx \neq Ty
      # ---       (m, n_out)              if many-to-one


      X_input = Input(shape = input_shape, name='Input')
      #X       = Embedding(10000, 300, name='Embedding')(X_input)   # --- 10000 is the size of the vocabulary / 300 is the size of the embedding
      X, H, C = LSTM(32, return_sequences=True, return_state=True, name='RNN1')(X_input)
      X, H, C = LSTM(32, return_sequences=True, return_state=True, name='RNN2')(X)
      X       = Dense(1, name='FC')(X)
      X_output= Activation('sigmoid', name='Activation')(X)

      model = Model(inputs= X_input, outputs=X_output)

      model.summary()
      print('input dimension: ', X_dummy.shape)
      print('output dimension: ', model.predict(X_dummy).shape)
```

```
Model: "model_61"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 15, 100)           0
_________________________________________________________________
RNN1 (LSTM)                  [(None, 15, 32), (None, 3 17024
_________________________________________________________________
RNN2 (LSTM)                  [(None, 15, 32), (None, 3 8320
_________________________________________________________________
FC (Dense)                   (None, 15, 1)             33
_________________________________________________________________
Activation (Activation)      (None, 15, 1)             0
=================================================================
Total params: 25,377
Trainable params: 25,377
Non-trainable params: 0
_________________________________________________________________
input dimension:  (20, 15, 100)
output dimension:  (20, 15, 1)
```

RNN in keras

```
[103] # --- Input dimensions are
      # ---      (m, seq_len1, n_in)
      X_dummy = np.random.randn(20, 15, 100)
      input_shape=(15, 100,)

      #       (m, seq_len)    if use of embbeding
      #X_dummy =  np.random.randint(0, 10000, (20,15))
      #input_shape=(15,)

      # --- Output dimensions are
      # ---       (m, seq_len1, n_out)    if many-to-many Tx=Ty
      # ---       (m, seq_len2, n_out)    if many-to-many Tx \neq Ty
      # ---       (m, n_out)              if many-to-one


      X_input = Input(shape = input_shape, name='Input')
      #X       = Embedding(10000, 300, name='Embedding')(X_input)   # --- 10000 is the size of the vocabulary / 300 is the size of the embedding
      X, H, C = LSTM(32, return_sequences=True, return_state=True, name='RNN1')(X_input)
      X, H, C = LSTM(32, return_sequences=False, return_state=True, name='RNN2')(X)
      X       = Dense(1, name='FC')(X)
      X_output= Activation('sigmoid', name='Activation')(X)

      model = Model(inputs= X_input, outputs=X_output)

      model.summary()
      print('input dimension: ', X_dummy.shape)
      print('output dimension: ', model.predict(X_dummy).shape)
```

```
Model: "model_62"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 15, 100)           0
_________________________________________________________________
RNN1 (LSTM)                  [(None, 15, 32), (None, 3 17024
_________________________________________________________________
RNN2 (LSTM)                  [(None, 32), (None, 32),  8320
_________________________________________________________________
FC (Dense)                   (None, 1)                 33
_________________________________________________________________
Activation (Activation)      (None, 1)                 0
=================================================================
Total params: 25,377
Trainable params: 25,377
Non-trainable params: 0
_________________________________________________________________
input dimension:  (20, 15, 100)
output dimension:  (20, 1)
```

Dimensions : RNN many-to-one with embedding

RNN in keras

```
[104] # --- Input dimensions are
      # ---       (m, seq_len1, n_in)
      #X_dummy = np.random.randn(20, 15, 100)
      #input_shape=(15, 100,)

      #       (m, seq_len)    if use of embbeding
      X_dummy =  np.random.randint(0, 10000, (20,15))
      input_shape=(15,)

      # --- Output dimensions are
      # ---       (m, seq_len1, n_out)    if many-to-many Tx=Ty
      # ---       (m, seq_len2, n_out)    if many-to-many Tx \neq Ty
      # ---       (m, n_out)              if many-to-one


      X_input = Input(shape = input_shape, name='Input')
      X       = Embedding(10000, 300, name='Embedding')(X_input)   # --- 10000 is the size of the vocabulary / 300 is the size of the embedding
      X, H, C = LSTM(32, return_sequences=True, return_state=True, name='RNN1')(X)
      X, H, C = LSTM(32, return_sequences=False, return_state=True, name='RNN2')(X)
      X       = Dense(1, name='FC')(X)
      X_output= Activation('sigmoid', name='Activation')(X)

      model = Model(inputs= X_input, outputs=X_output)

      model.summary()
      print('input dimension: ', X_dummy.shape)
      print('output dimension: ', model.predict(X_dummy).shape)
```

```
Model: "model_63"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 15)                0
_________________________________________________________________
Embedding (Embedding)        (None, 15, 300)           3000000
_________________________________________________________________
RNN1 (LSTM)                  [(None, 15, 32), (None, 3 42624
_________________________________________________________________
RNN2 (LSTM)                  [(None, 32), (None, 32),  8320
_________________________________________________________________
FC (Dense)                   (None, 1)                 33
_________________________________________________________________
Activation (Activation)      (None, 1)                 0
=================================================================
Total params: 3,050,977
Trainable params: 3,050,977
Non-trainable params: 0
_________________________________________________________________
input dimension:  (20, 15)
output dimension:  (20, 1)
```

Dimensions : RNN padding sequences of different lengths

Keras Padding Sequences

```
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer

reviews = ['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.']

data = [review.split() for review in reviews]
print('*** Sequences have different lenghts'); print(len(data[0])); print(len(data[1])); print(len(data[2]))

# --- First tokenize all the texts
tokenizer_obj = Tokenizer()
tokenizer_obj.fit_on_texts(reviews)
print('*** word to index'); print(tokenizer_obj.word_index)
print('*** index to word'); print(tokenizer_obj.index_word)

X_train_tokens = tokenizer_obj.texts_to_sequences(reviews)
print('*** Tokenized texts')
print(X_train_tokens)

# --- Second pad sequence length
max_length = 40
X_train_pad = pad_sequences(X_train_tokens, maxlen=max_length, value=0.0, padding='post')
print('*** Post-padding'); print(X_train_pad)
X_train_pad = pad_sequences(X_train_tokens, maxlen=max_length, value=0.0, padding='pre')
print('*** Pre-padding'); print(X_train_pad)
```

```
*** Sequences have different lenghts
15
10
14
*** word to index
{'good': 1, 'movie': 2, 'this': 3, 'is': 4, 'it': 5, 'and': 6, 'a': 7, 'fantastic': 8, 'i': 9, 'really': 10, 'like': 11, 'beca
*** index to word
{1: 'good', 2: 'movie', 3: 'this', 4: 'is', 5: 'it', 6: 'and', 7: 'a', 8: 'fantastic', 9: 'i', 10: 'really', 11: 'like', 12: '
*** Tokenized texts
[[3, 2, 4, 8, 9, 10, 11, 5, 12, 5, 4, 13, 1], [14, 15, 16, 17, 18, 19, 6, 20, 21, 2], [3, 2, 22, 23, 24, 7, 1, 25, 1, 26, 6, 7
*** Post-padding
[[ 3  2  4  8  9 10 11  5 12  5  4 13  1  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [14 15 16 17 18 19  6 20 21  2  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 3  2 22 23 24  7  1 25  1 26  6  7  1 27  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]]
*** Pre-padding
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  3  2  4  8  9 10 11  5 12  5  4 13  1]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0 14 15 16 17 18 19  6 20 21  2]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  3  2 22 23 24  7  1 25  1 26  6  7  1 27]]
```

1. Example many to many ($T_x = T_y$)

```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(26, 5511, 101)
(26, 1375, 1)
(25, 5511, 101)
(25, 1375, 1)
```

```
def model(input_shape):
    X_input = Input(shape = input_shape)

    X = Conv1D(196, kernel_size=15, strides=4)(X_input)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)
    X = Dropout(0.8)(X)

    X = GRU(units = 128, return_sequences = True)(X)
    X = Dropout(0.8)(X)
    X = BatchNormalization()(X)

    X = GRU(units = 128, return_sequences = True)(X)
    X = Dropout(0.8)(X)
    X = BatchNormalization()(X)
    X = Dropout(0.8)(X)

    X = TimeDistributed(Dense(1, activation = "sigmoid"))(X)
    #X = Dense(1, activation = "sigmoid")(X)

    model = Model(inputs = X_input, outputs = X)

    return model
```

```
Tx = 5511
n_freq = 101
Ty = 1375
model = model(input_shape = (Tx, n_freq))
opt = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, decay=0.01)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=["accuracy"])
model.fit(X_train, Y_train, batch_size = 5, epochs=10)
```

2. Example many to one ($T_x > 1, T_y = 1$) : with embedding matrix

```
def F_pretrained_embedding_layer(word_to_vec_map, word_to_index):
    vocab_len = len(word_to_index) + 1
    emb_dim = word_to_vec_map["cucumber"].shape[0]
    emb_matrix = np.zeros((vocab_len, emb_dim))
    for word, index in word_to_index.items():
        emb_matrix[index, :] = word_to_vec_map[word]
    embedding_layer = Embedding(vocab_len, emb_dim, trainable=False)
    embedding_layer.build((None,))
    embedding_layer.set_weights([emb_matrix])
    return embedding_layer
```

```
def myModel(input_shape, word_to_vec_map, word_to_index):

    sentence_indices = Input(input_shape, dtype='int32')
    embedding_layer = F_pretrained_embedding_layer(word_to_vec_map, word_to_index)
    embeddings = embedding_layer(sentence_indices)

    X = LSTM(128, return_sequences=True)(embeddings)
    X = Dropout(0.5)(X)
    X = LSTM(128, return_sequences=False)(X)
    X = Dropout(0.5)(X)
    X = Dense(5)(X)
    X = Activation('softmax')(X)

    model = Model(inputs=sentence_indices, outputs=X)
    return model
```

```
model = myModel((maxLen,), word_to_vec_map, word_to_index)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

3. Example one to many ($T_x = 1, T_y > 1$) : training language model

```
n_a = 64
n_values = 78
reshapor = Reshape((1, 78))
LSTM_cell = LSTM(n_a, return_state = True)
densor = Dense(n_values, activation='softmax')
```

```
def F_language_model(Tx, n_a, n_values):

    X = Input(shape=(Tx, n_values))

    a0 = Input(shape=(n_a,), name='a0')
    c0 = Input(shape=(n_a,), name='c0')
    a = a0
    c = c0

    outputs = []

    for t in range(Tx):
        x = Lambda(lambda x: x[:,t,:])(X)
        x = reshapor(x)
        a, _, c = LSTM_cell(x, initial_state=[a, c])
        out = densor(a)
        outputs.append(out)

    model = Model(inputs=[X, a0, c0], outputs=outputs)

    return model
```

```
m = 60
a0 = np.zeros((m, n_a))
c0 = np.zeros((m, n_a))

model = F_language_model(Tx = 30 , n_a = 64, n_values = 78)
opt = Adam(lr=0.01, beta_1=0.9, beta_2=0.999, decay=0.01)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.fit([X, a0, c0], list(Y), epochs=100)
```

3. Example one to many ($T_x = 1, T_y > 1$) : inference

```
def one_hot(x):
    x = K.argmax(x)
    x = tf.one_hot(x, 78)
    x = RepeatVector(1)(x)
    return x

def F_inference_model(LSTM_cell, densor, n_values = 78, n_a = 64, Ty = 100):
    x0 = Input(shape=(1, n_values))
    a0 = Input(shape=(n_a,), name='a0')
    c0 = Input(shape=(n_a,), name='c0')
    a = a0
    c = c0
    x = x0

    outputs = []

    for t in range(Ty):

        a, _, c = LSTM_cell(x, initial_state=[a, c])
        out = densor(a)
        outputs.append(out)
        x = Lambda(one_hot)(out)

    inference_model = Model(inputs=[x0, a0, c0], outputs=outputs)
    return inference_model
```

```
inference_model = F_inference_model(LSTM_cell, densor, n_values = 78, n_a = 64, Ty = 50)

x_initializer = np.zeros((1, 1, 78))
a_initializer = np.zeros((1, n_a))
c_initializer = np.zeros((1, n_a))

pred = inference_model.predict([x_initializer, a_initializer, c_initializer])

indices = np.argmax(pred, axis=-1)
results = to_categorical(indices, num_classes=78)
```

4. Example many to many ($T_x \neq T_y$) : Encoder/Decoder training

```
# Define an input sequence and process it.
encoder_inputs = Input(shape=(None, num_encoder_tokens))

encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))

# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)

decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

```
# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

# Run training
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)
```

4. Example many to many ($T_x \neq T_y$) : Encoder/Decoder inference

```
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model([decoder_inputs] + decoder_states_inputs, [decoder_outputs] + decoder_states)
```

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1, num_decoder_tokens))
    # Populate the first character of target sequence with the start character.
    target_seq[0, 0, target_token_index['\t']] = 1.

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char

        # Exit condition: either hit max length
        # or find stop character.
        if (sampled_char == '\n' or
           len(decoded_sentence) > max_decoder_seq_length):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.

        # Update states
        states_value = [h, c]

    return decoded_sentence
```