The Mathemmatical Engineering of Deep Learning

Chapter 8 - Lecture 8

B. Liquet[1,2] and S. Moka[3] and Y. Nazarathy[3]

[1] Macquarie University [2] LMAP, Université de Pau et des Pays de L'Adour' [3] The University of Queensland

Outline of Lecture

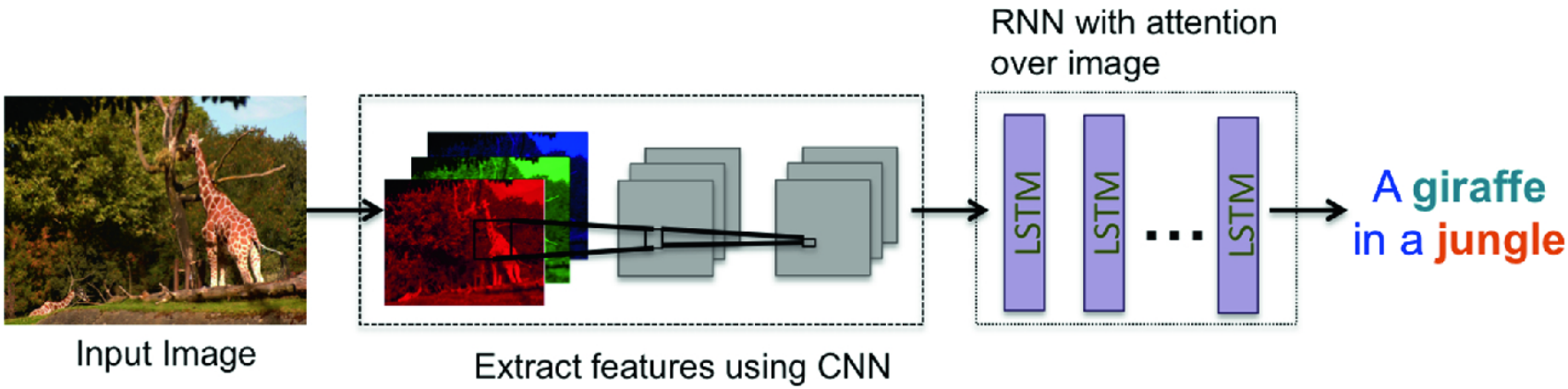
- Sequence data
- Recurrent Neural Network (RNN)
- Long Short Term Memory (LSTM)
- Transformers
- Example: Usage of auto-encoders for language translation

Introduction of Sequence Models

- Analysis of sequential data: text sentences, time-series and other discrete sequences data.
- Design to handle **sequential information** while **Convolutional Neural Network** process spatial information.
- **Key point**: data we are processing are not anymore independently and identically distributed (i.i.d.) samples and the data carry some dependency due to the sequential order of the data.
- **Sequence Models** is very popular:
  - speech recognition
  - voice recognition
  - time series prediction
  - natural language processing.

Application

- **Image Captioning:** caption an image by analyzing the present action
- Exploit a **CNN** to generate a set of candidate words and use a **RNN** or **LSTM** to construct a coherent sentence from the words (see details here)

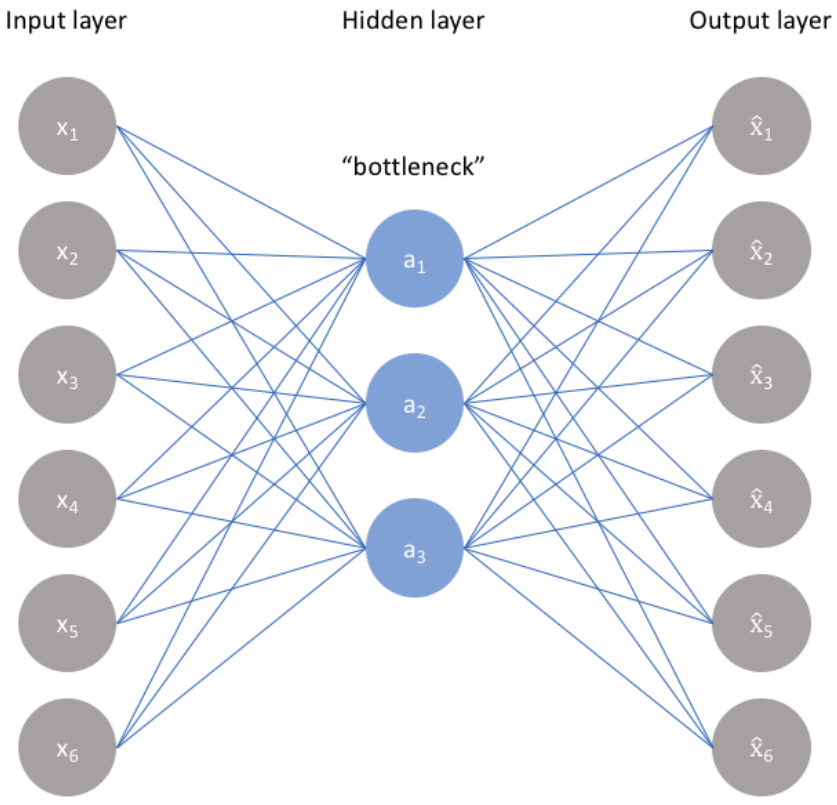


Application

- **Time Series:** prediction time series problem, e.g., stock market predictions

Autoencoders

Autoencoders



Autoencoder model balances:

- **Sensitive** to the inputs enough to accurately build a reconstruction.
- **Insensitive** enough to the inputs that the model doesn't simply memorize or overfit the training data.

Singular Value Decomposition (SVD)

Singular Value Decomposition, or SVD, is a computational method often employed to calculate principal components for a dataset. Using SVD to perform PCA is efficient and numerically robust. (see https://intoli.com/blog/pca-and-svd/).

Let a matrix $M : p \times q$ of rank r:

$$M = U\Delta V^T = \sum_{l=1}^{r} \delta_l u_l v_l^T,$$

- $U = (u_l) : p \times p$ and $V = (v_l) : q \times q$ are two orthogonal matrices which contain the normalised left (resp. right) singular vectors
- $\Delta = \mathrm{diag}(\delta_1, \ldots, \delta_r, 0, \ldots, 0)$: the ordered singular values $\delta_1 \geqslant \delta_2 \geqslant \cdots \geqslant \delta_r > 0$.

Reduction Dimension using SVD

Eckart-Young (1936) states that the (truncated) SVD of a given matrix M (of rank r) provides the best reconstitution (in a least squares sense) of M by a matrix with a lower rank k:

$$\min_{A \text{ of rank } k} \|M - A\|_F^2 = \left\| M - \sum_{\ell=1}^{k} \delta_\ell u_\ell v_\ell^T \right\|_F^2 = \sum_{\ell=k+1}^{r} \delta_\ell^2.$$

SVD as a Compression/Dimension Reduction Tool

We start by reading an image and we perform SVDs on this image.

```
if (!"jpeg" %in% installed.packages())
  install.packages("jpeg")
# Read image file into an array with three channels
# (Red-Green-Blue, RGB)
liquet <- jpeg::readJPEG("liquet.jpeg")
r <- liquet[, , 1] ; g <- liquet[, , 2] ; b <- liquet[, , 3]
# Performs full SVD of each channel
liquet.r.svd <- svd(r) ; liquet.g.svd <- svd(g) ;
liquet.b.svd <- svd(b)
rgb.svds <- list(liquet.r.svd, liquet.g.svd, liquet.b.svd)
```

SVD as a Compression/Dimension Reduction Tool

These two functions will be needed to display an image stored in an RGB array:

```
# Function to display an image stored in an RGB array
plot.image <- function(pic, main = "") {
 h <- dim(pic)[1] ; w <- dim(pic)[2]
 plot(x = c(0, h), y = c(0, w), type = "n", xlab = "",
      ylab = "", main = main)
 rasterImage(pic, 0, 0, h, w)
}
```

Function to compress an image via SVD of each channel

```
compress.image <- function(rgb.svds, nb.comp) {
 # nb.comp (number of components) should be less than min(dim(img[,,1])),
 # i.e., 170 here
 svd.lower.dim <- lapply(rgb.svds, function(i)
   list(d = i$d[1:nb.comp],
   u = i$u[, 1:nb.comp],
   v = i$v[, 1:nb.comp]))

 img <- sapply(svd.lower.dim, function(i) {
    img.compressed <- i$u %*% diag(i$d) %*% t(i$v)
  }, simplify = 'array')
 img[img < 0] <- 0
 img[img > 1] <- 1
 return(list(img = img, svd.reduced = svd.lower.dim))
}
```

plot side-by-side the original and compressed images now.

```
par(mfrow = c(1, 2))
plot.image(liquet, "Original image")
p <- 20 ; plot.image(compress.image(rgb.svds, p)$img,
                     paste("SVD with", p, "components"))
```

compression ?

As you can see, with 20 components (over 170 maximum), we can still recognize Benoit!

How much compression did we achieve with 20 components?

```
object.size(rgb.svds) # Original image
```

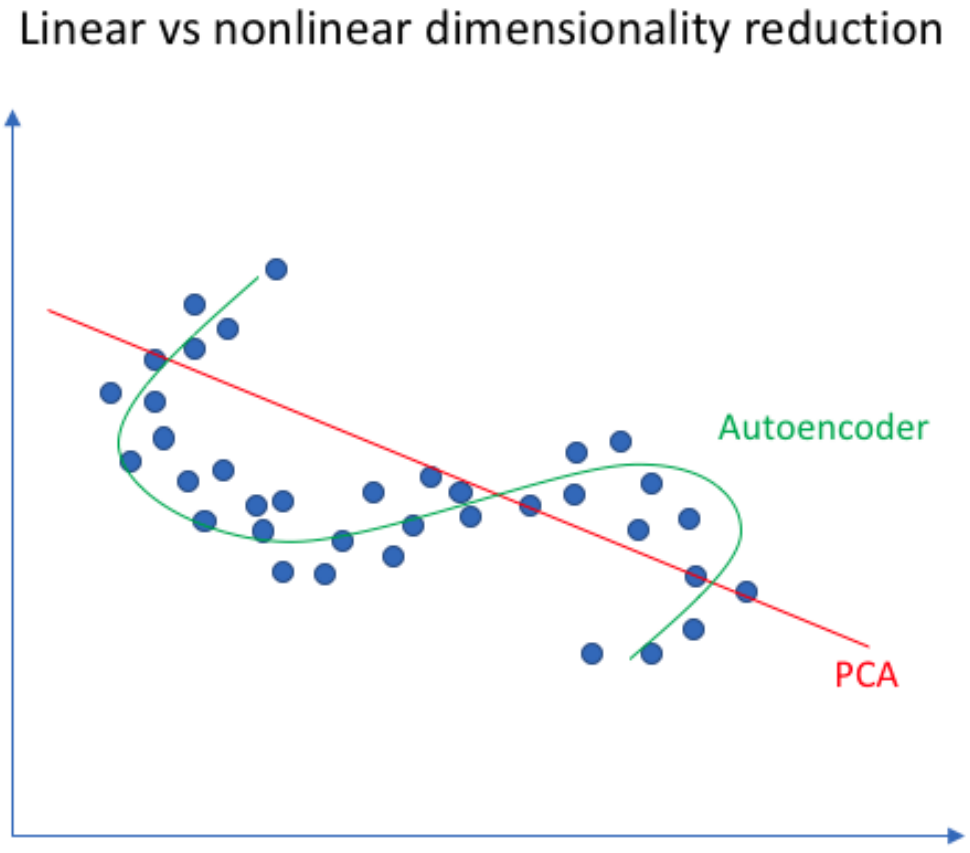
```
## 1740920 bytes
```

```
object.size(compress.image(rgb.svds, p)$svd.reduced)
```

```
## 207320 bytes
```

```
# Compressed image
```

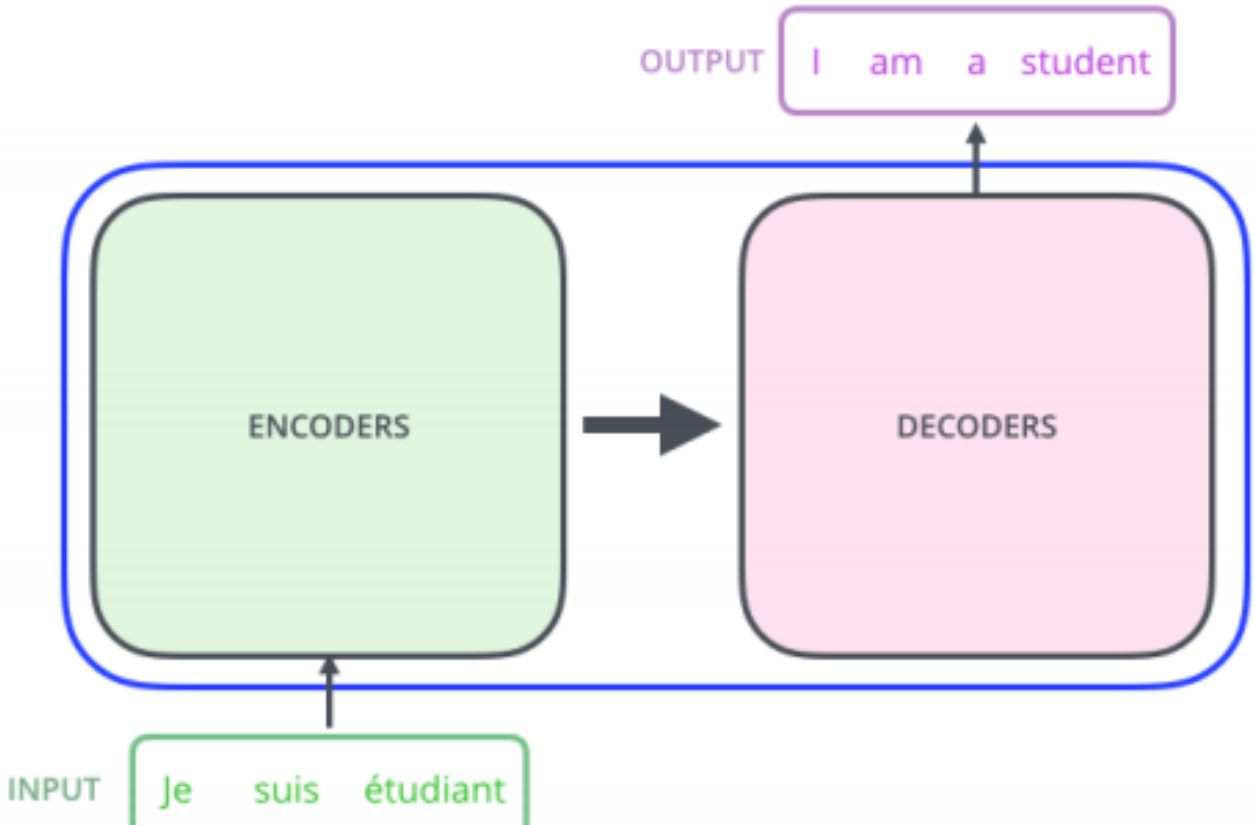
Autoencoders and PCA



Link between PCA and Autoencoders

Application

- **Natural Language Processing:** Text mining and Sentiment (e.g., Learning word vectors for sentiment analysis)
- **Machine Translation:** Given an input in one language use sequence models to translate the input into different languages as output. Here a recent survey



Applications

- **Speech recognition**: [Deep Recurrent Neural network for speech recognition]
- **DNA sequence analysis**: [Recurrent Neural Network for Predicting Transcription Factor Binding Sites]
- **RNN Generated TED Talks** [YouTube Link]
- **RNN Generated Eminem rapper** [RNN Shady}]
- **RNN Generated Music** [Music Link]
- **Music generation** [generating classical music using recurrent neural networks]

Recurrent Neural network

Recurrent Neural Network

RNN is especially designed to deal with sequential data which is not i.i.d.

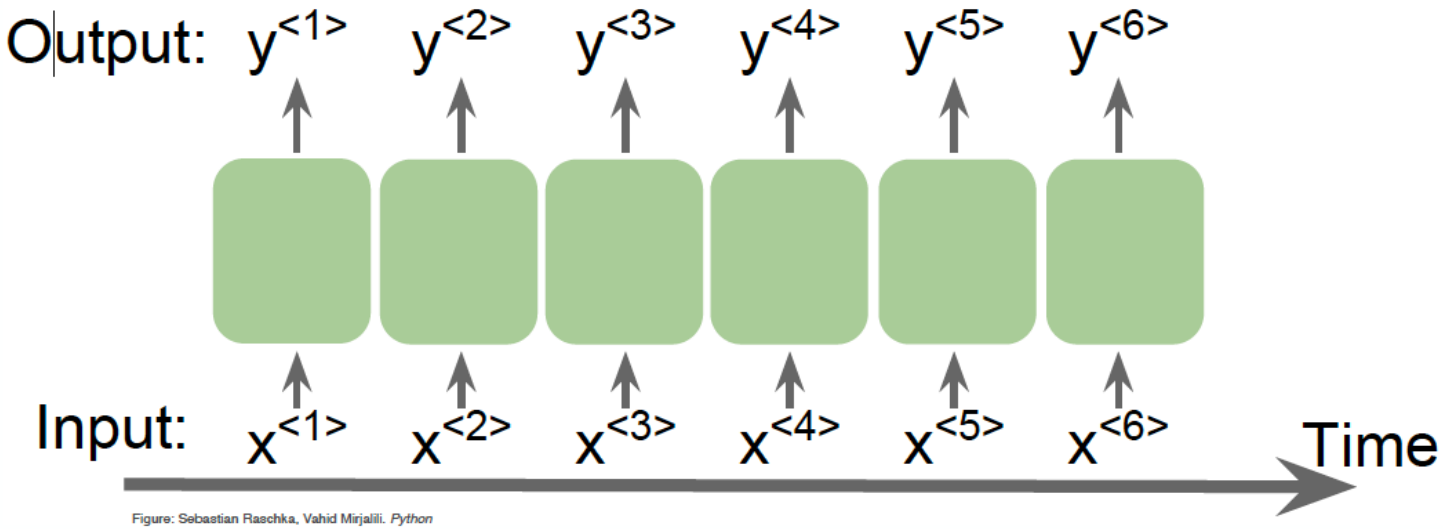


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

RNN can tackle the following challenges from sequence data:

- to deal with variable-length sequences
- to maintain sequence order
- to keep track of long-term dependencies
- to share parameters across the sequence

Graphical Representation of RNN

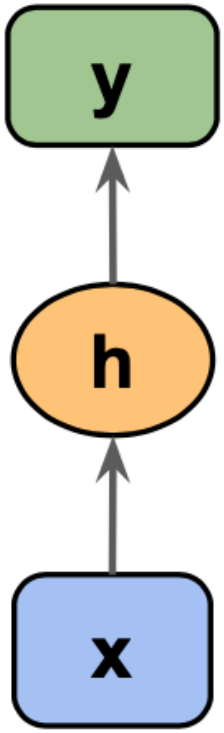
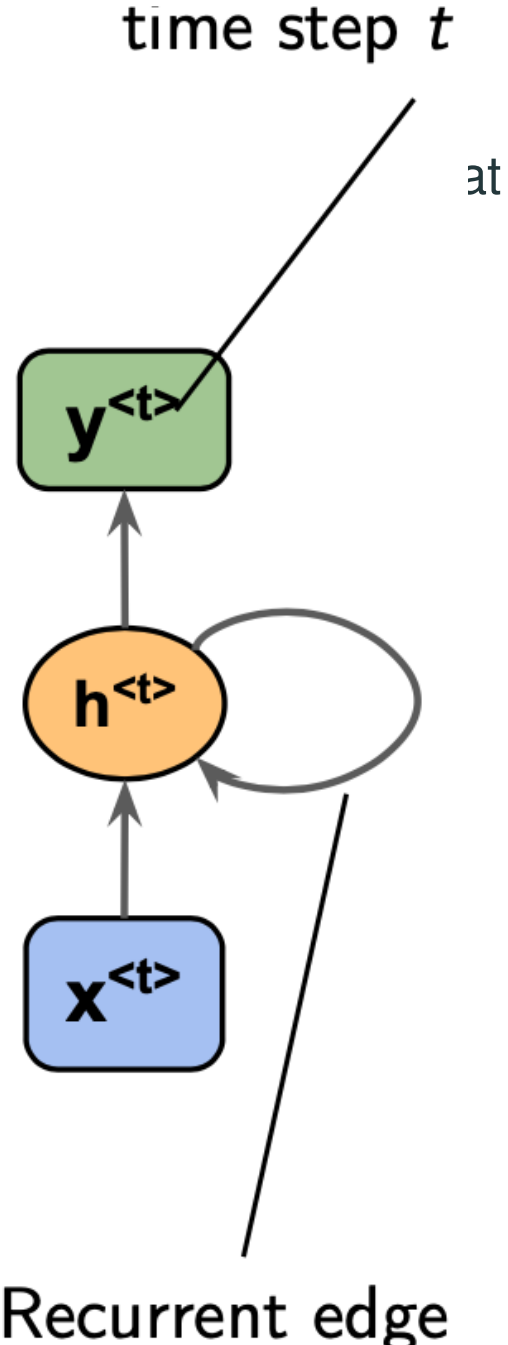
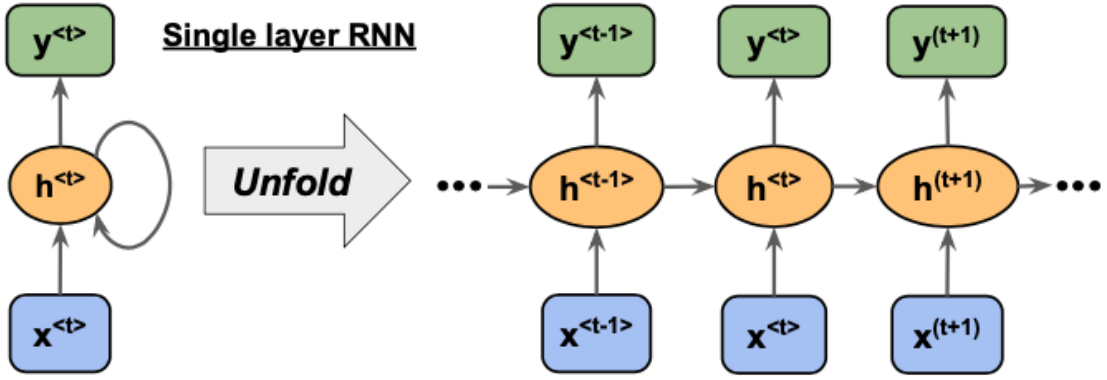


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019

Recurrent Neural Network (RNN)



Unfold Representation



$$\underbrace{h^{<t>}}_{\text{cell state}} = f_W(\underbrace{h^{<t-1>}}_{\text{old state}}, \underbrace{x^{<t>}}_{\text{input vector}})$$

- f_W is a function parameterized by the weight W.
- At every time step t the same function f_W is used and same set of weight parameters.

Multilayer Layer RNN

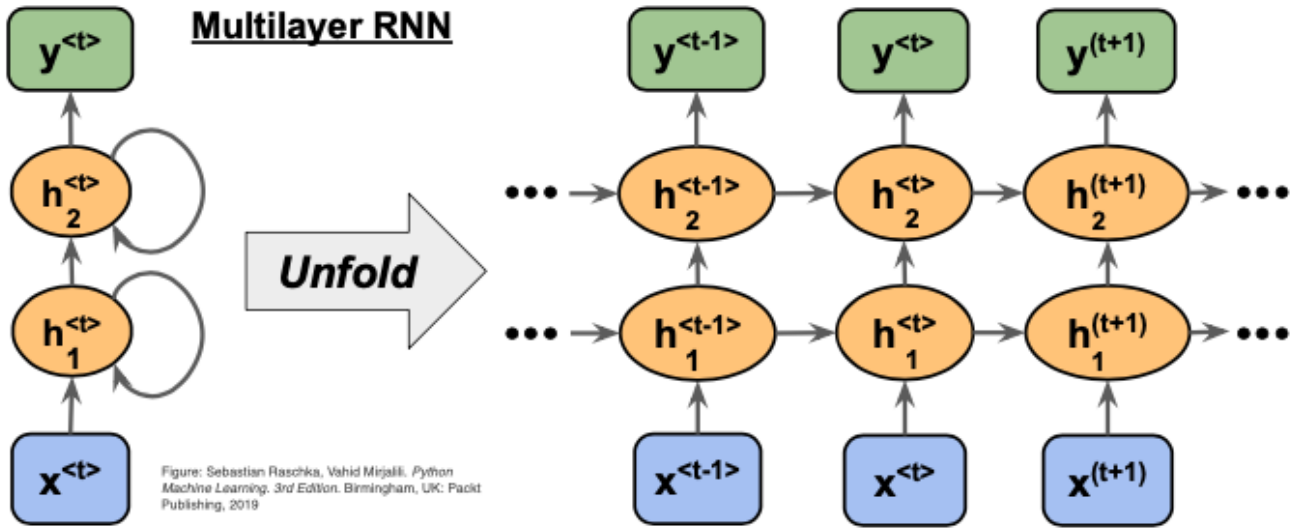


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

Weight Matrices parameters in RNN Sequence Modeling Tasks

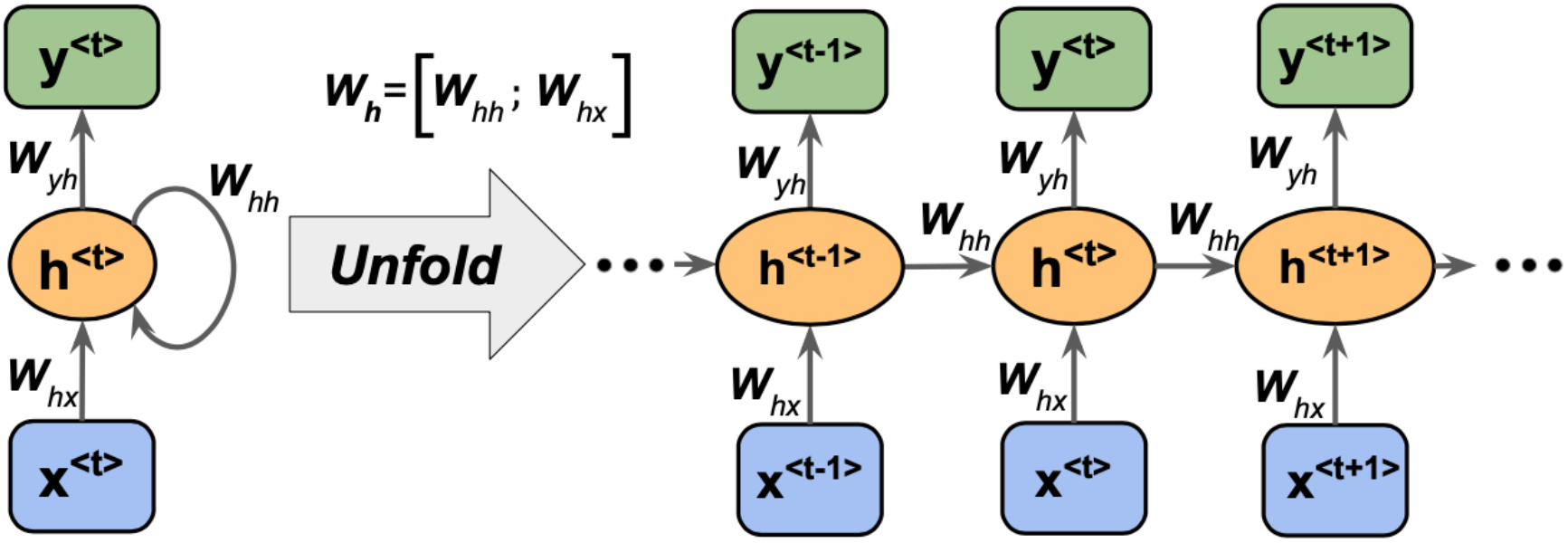


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019

- $x^{<1>}, \ldots, x^{<t-1>}, x^{<t>}, x^{<t+1>}, \ldots x^{<m>}$: the input data. In NLP, for example, the sequence input is a **sentence** of m words $x^{<1>}, \ldots, x^{<t-1>}, x^{<t>}, x^{<t+1>}, \ldots x^{<m>}$.
- $x^{<t>} \in \mathbb{R}^{|\mathcal{V}|}$: input vector at time t. For example each word $x^{<t>}$ in the sentence will be input as 1-hot vector of size $|\mathcal{V}|$ (where $\mathcal{V}$ is the vocabulary, $|\mathcal{V}|$ the size).

Weight Matrices parameters in RNN Sequence Modeling Tasks

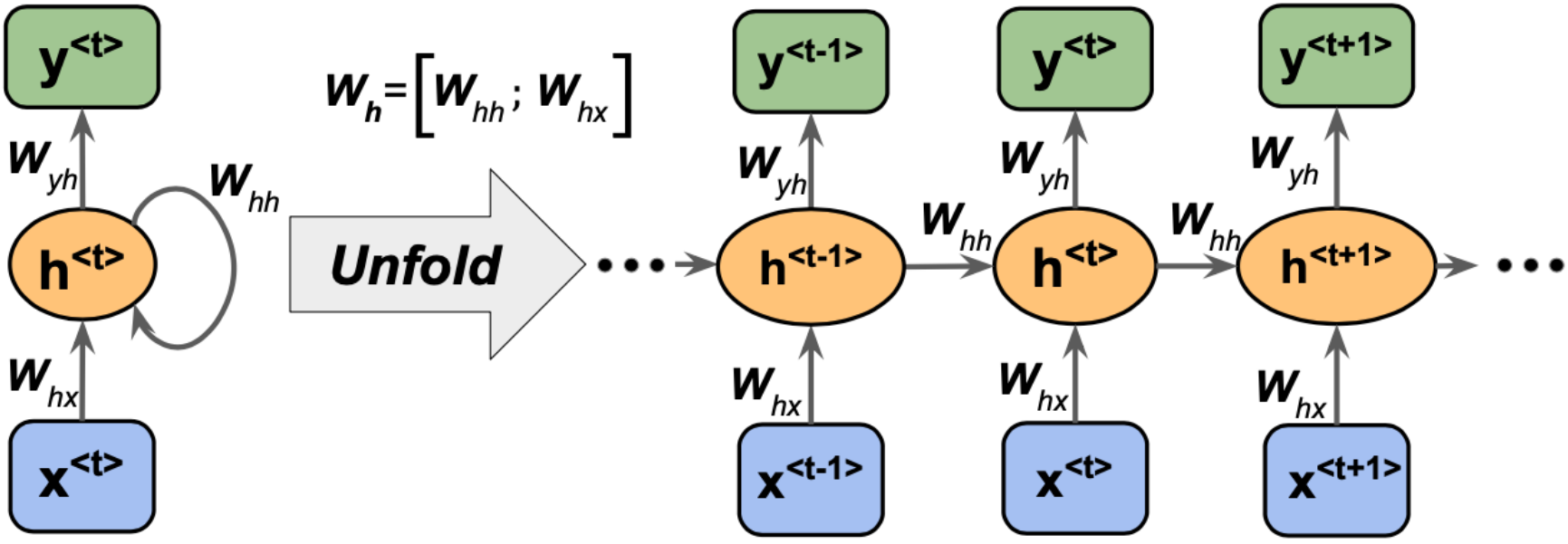
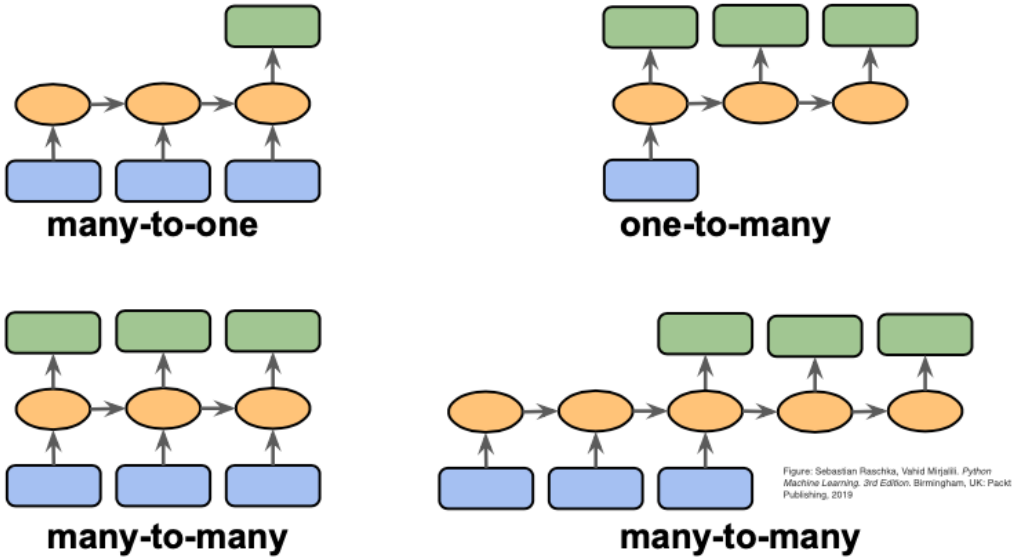


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019

Different types of Sequence Modeling Tasks



- **many-to-one**: Sentiment Classification, action prediction (sequence of video -> action class)
- **one-to-many**: Image captioning (image -> sequence of words)
- **many-to-many**: Video Captioning (Sequance of video frames -> Caption)
- **many-to-many**: Video Classification on frame level

How to train a RNN ?

RNN a clearly some nice features:

- can process any length input
- In theory for each time t can use information from many steps back
- Same weights applied on every timestep

However RNN could be very slow to train

- In practice it is difficult to access information from many steps back.

Let consider training a RNN for **language model example**:

- We first have to get access to a big corpus of text which is a sequence of words $x^{<1>}, \ldots, x^{<t-1>}, x^{<t>}, x^{<t+1>}, ...x^{<T>}$
- **Forward pass** and compute the output distribution $\widehat{y}^{<t>}$ for every time t: predicted probability distribution of every word, given words so far
- Compute the **Loss fuction** on each step t. Here, **cross-entropy** between predicted probability distribution $\widehat{y}^{<t>}$ and the true next word $y^{<t>}$ (one-hot for $\widehat{x}^{<t+1>}$):

$$L^{<t>}(\theta) = CE(y^{<t>}, \widehat{y}^{<t>}) = -\sum_{j=1}^{|V|} y_j^{<t>} \times log(\widehat{y}_j^{<t>})$$

- Average this to get overall loss for entire training set:

$$L = \frac{1}{T}\sum_{t=1}^{T} L^{<t>}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_j^{<t>} \times log(\widehat{y}_j^{<t>})$$

- Computing the loss and the gradients across **entire corpus** is too expensive.
- In practice we use a batch of sentences to compute the loss and Stochastic Gradient Descent is exploited to compute the gradients for small chunk of data, and then update.

Backpropagation through time

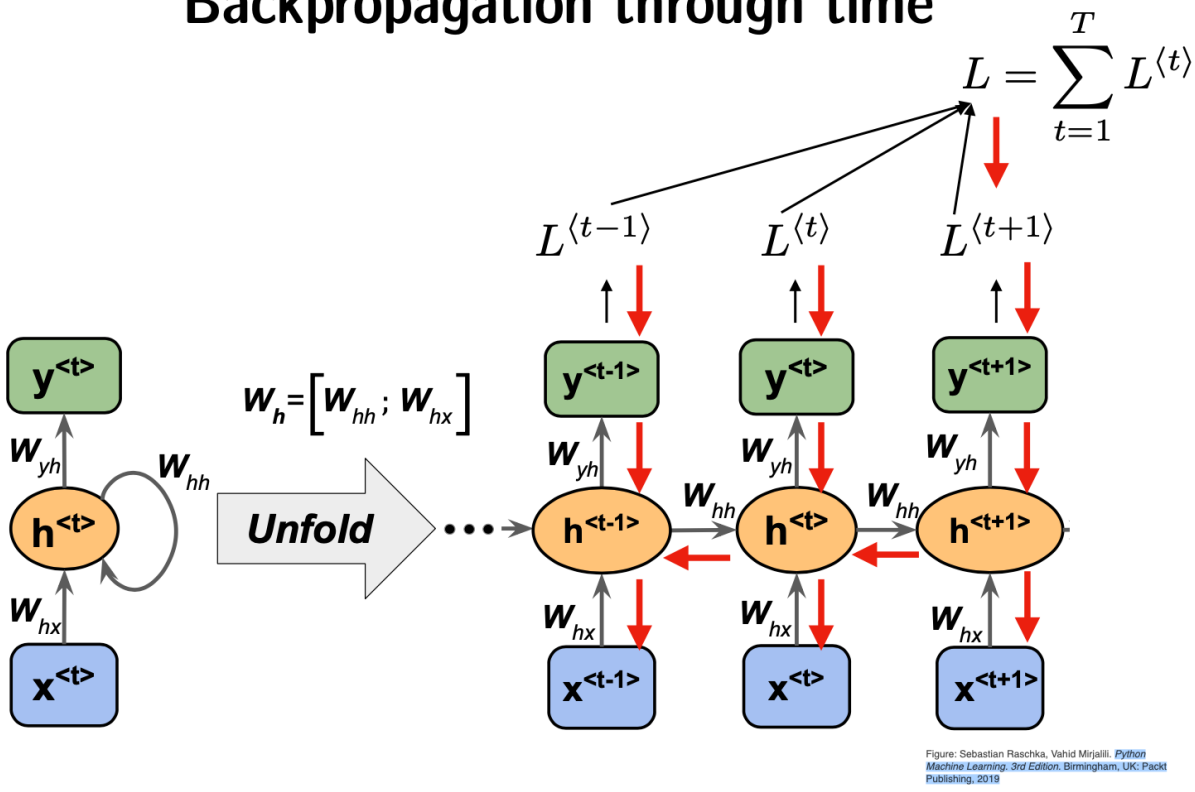


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning, 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

The backpropagation over time as two summations

- First summation over L

$$\frac{\partial L}{\partial W_{hh}} = \sum_{j=1}^{T} \frac{\partial L^{<j>}}{\partial W_{hh}}$$

- Second summations showing that each $L^{<t>}$ depends on the weight matrices before it:

$$\frac{\partial L^{<t>}}{\partial W_{hh}} = \sum_{k=1}^{t} \frac{\partial L^{<k>}}{\partial W_{hh}}$$

Using the multivariate chain-rule:

$$\boxed{\frac{\partial L^{<t>}}{\partial W_{hh}} = \sum_{k=1}^{t} \frac{\partial L^{<t>}}{\partial \widehat{y}^{<t>}} \frac{\partial \widehat{y}^{<t>}}{\partial h^{<t>}} \frac{\partial h^{<t>}}{\partial h^{<k>}} \frac{\partial h^{<k>}}{\partial W_{hh}}}$$

- Chain rule differentiation over all hidden layers within $[k, t]$:

$$\frac{\partial h^{<t>}}{\partial h^{<k>}} = \prod_{j=k+1}^{t} \frac{\partial h^{<j>}}{\partial h^{<j-1>}} = \prod_{j=k+1}^{t} W_{hh}^{T} \times diag[f'(W_{hh} h^{<j-1>} + W_{hx} x^{<j>})]$$

$$\frac{\partial h^{<j>}}{\partial h^{<j-1>}} = \left[\frac{\partial h^{<j>}}{\partial h_1^{<j-1>}}, \ldots, \frac{\partial h^{<j>}}{\partial h_{D_h}^{<j-1>}} \right] = \begin{bmatrix} \frac{\partial h_1^{<j>}}{\partial h_1^{<j-1>}} & \cdots & \frac{\partial h_1^{<j>}}{\partial h_{D_h}^{<j-1>}} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \frac{\partial h_{D_h}^{<j>}}{\partial h_1^{<j-1>}} & \cdots & \frac{\partial h_{D_h}^{<j>}}{\partial h_{D_h}^{<j-1>}} \end{bmatrix}$$

Vanishing and Exploding gradient

- Remind us that the RNN is based on this recursive relation

$$h^{<t>} = \sigma(W_{hh} h^{<t-1>} + W_{hx} x^{<t>}).$$

Let consider the identity function $\sigma(x) = x$:

$$\begin{aligned} \frac{\partial h^{<t>}}{\partial h^{<t-1>}} &= diag(\sigma'(W_{hh} h^{<t-1>} + W_{hx} x^{<t>})) W_{hh} \\ &= I W_{hh} = W_{hh} \end{aligned}$$

The gradient of the loss $L^{<i>}$ on time i, with respect to the hidden state $h^{<j>}$ on some previous time j (with $\ell = i - j$):

$$\begin{aligned} \frac{\partial L^{<i>}}{\partial h^{<j>}} &= \frac{\partial L^{<i>}}{\partial h^{<i>}} \prod_{j<t\leqslant i} \frac{\partial h^{<t>}}{\partial h^{<t-1>}} \\ &= \frac{\partial L^{<i>}}{\partial h^{<i>}} \prod_{j<t\leqslant i} W_{hh} = \frac{\partial L^{<i>}}{\partial h^{<i>}} W_{hh}^{\ell} \end{aligned}$$

Then if W_{hh} is "small", then this term gets exponentially problematic as ℓ becomes large.

Indeed, if the weight matrix W_{hh} is diagonalizable:

$$W_{hh} = Q^{-1} \times \Delta \times Q,$$

where Q is composed of the eigenvectors and Δ is a diagonal matrix with the eigenvalues on the diagonal. Computing the power of W_{hh} is then given by:

$$W_{hh}^{\ell} = Q^{-1} \times \Delta^{\ell} \times Q.$$

Thus eigenvalues lower than 1 will lead to vanishing gradient while eigenvalues greater than 1 will lead to exploding gradient.

Some Conclusion

- **Gradient signal** from faraway is lost because it's much smaller than gradient signal from close-by.
- **weights** are updated only with respect to near effects, not long-term effects.
- Example in **language modeling**, the contribution of faraway words to predicting the next word at time-step t diminishes when the gradient vanishes early on.
- The gradient can be viewed as a measure of the effect of the past on the future. Thus if gradient is small, the model cannot learn this dependency and the model unable to predict similar long-distance dependencies at test time.
- **Exploding gradient** is also a big issue for updating the weight and can cause too big step during the stochastic gradient descent. Moreover, once the gradient value grows extremely large, it causes an overflow (i.e. NaN).
- A solution to solve the problem of **exploding gradients** has been first introduced by Thomas Mikolov who proposed the **Gradient clipping**: scale down the gradient before applying an update when the norm of the gradient is greater than some threshold.

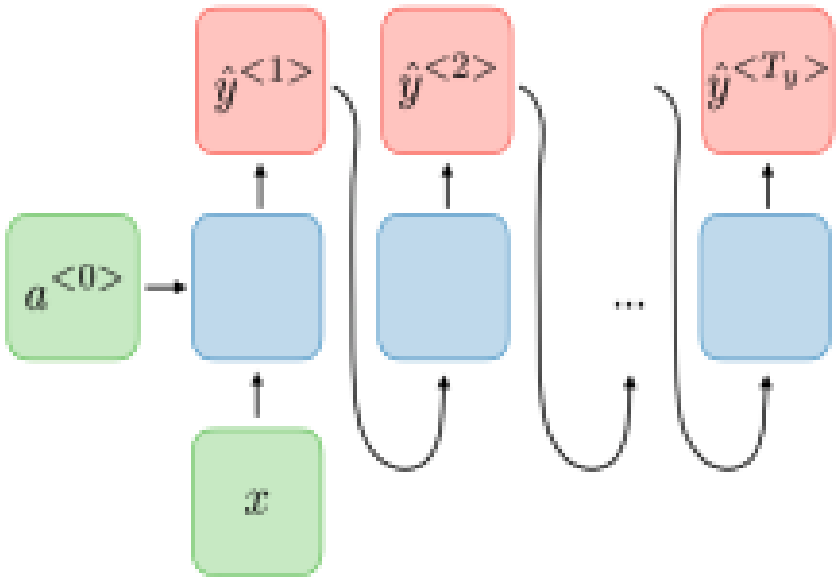
Pros and cons

- **Advantages**
  - Possibility of processing input of any length
  - Model size not increasing with size of input
  - Computation takes into account historical information
  - Weights are shared across time

- **Drawbacks**
  - Computation being slow
  - Difficulty of accessing information from a long time ago
  - Cannot consider any future input for the current state

Different acrchitectures

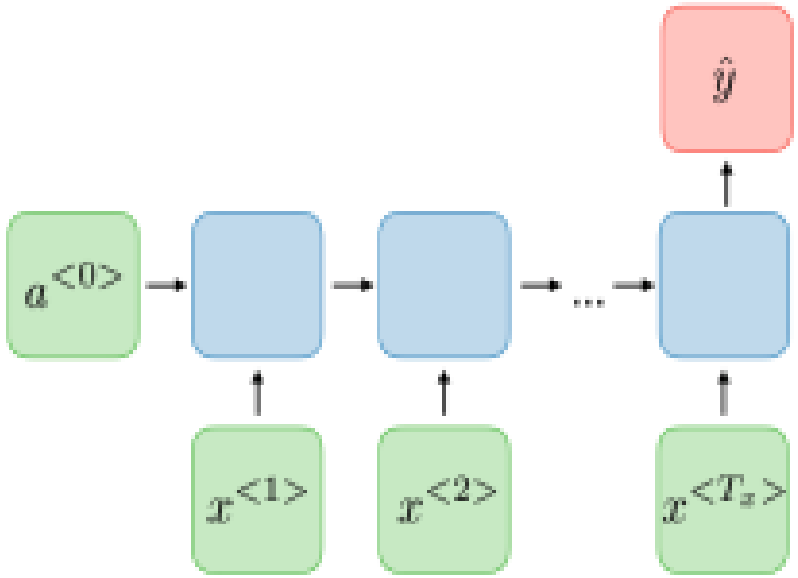
- One-to-many: $T_y > 1$ and T_x =?



Application: Music generation

Different acrchitectures

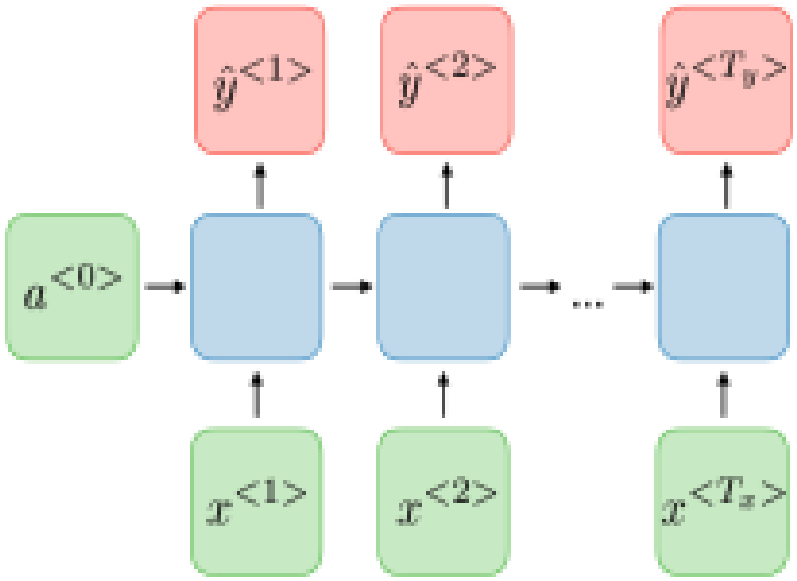
- Many-to-one: T_x and T_y



Application: Sentiment classification

Different acrchitectures

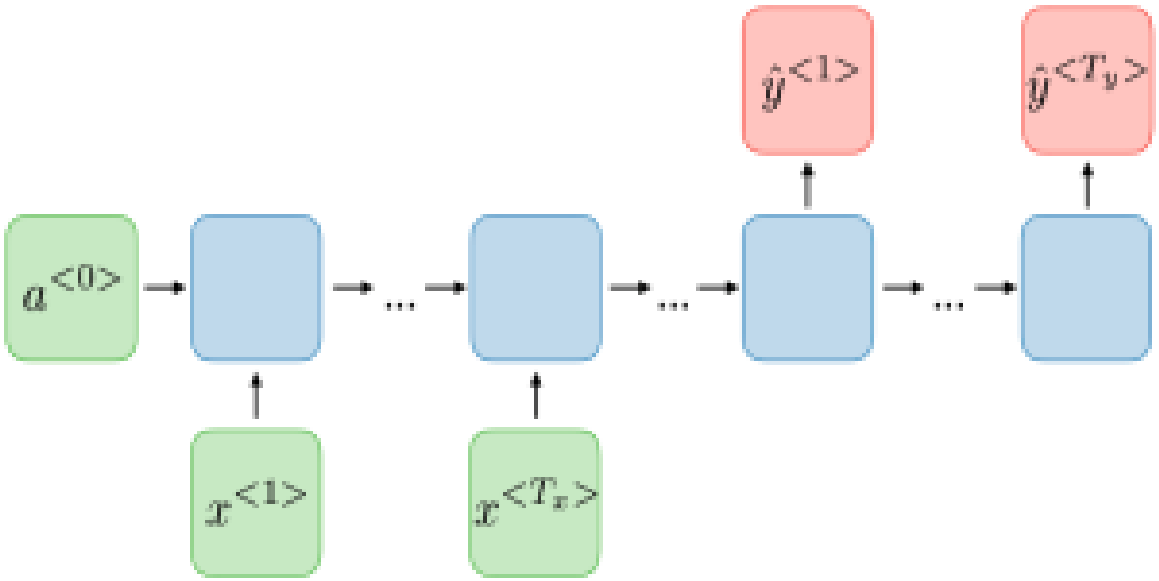
- Many-to-many: $T_{x} $ and $T_y $



Application: Name entity recognition

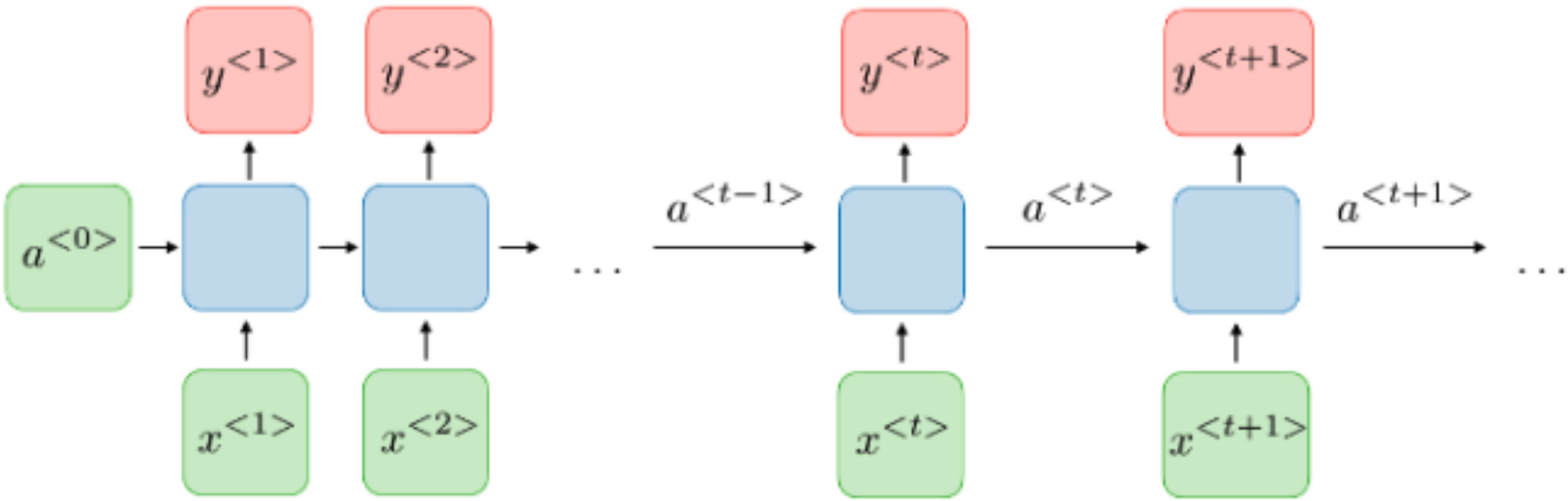
Different acrchitectures

- Many-to-many: T_x and T_y



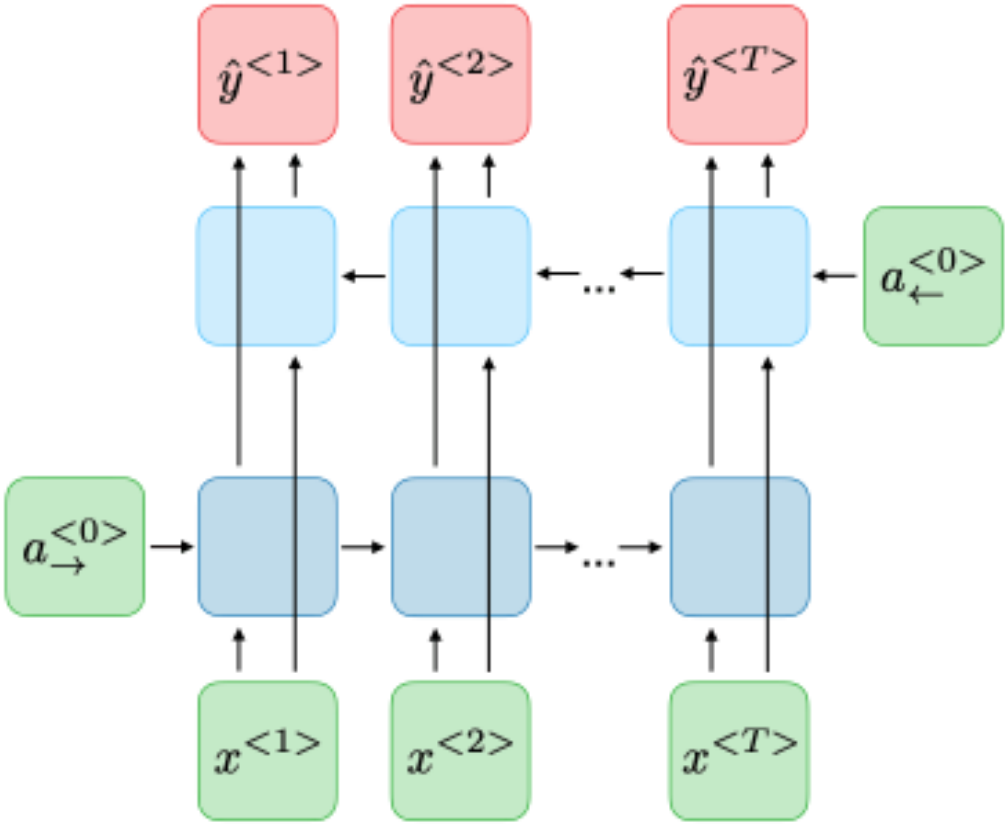
Application: Machine translation

Overview RNN



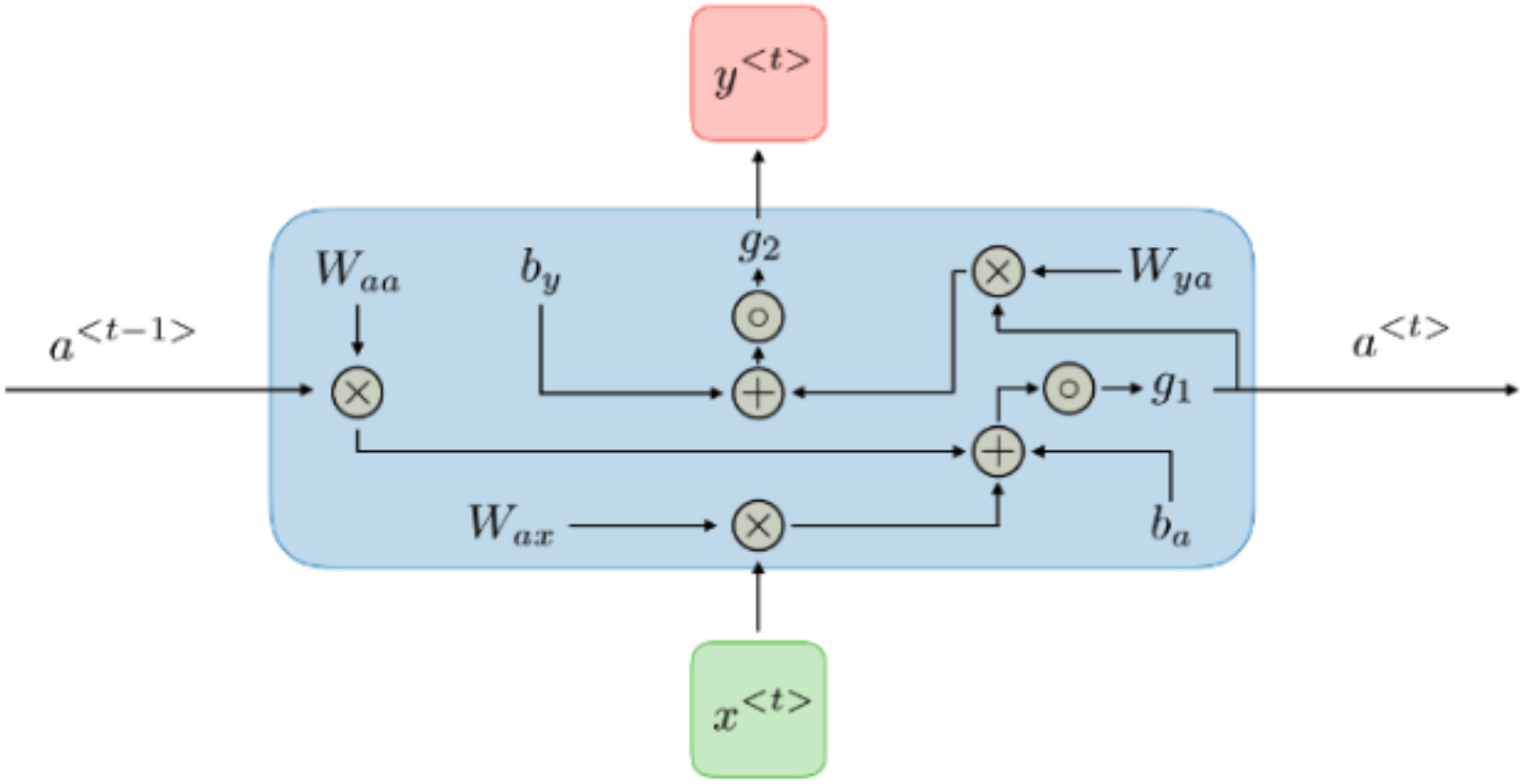
Variant of RNN

- Bidirectional (BRNN)



Popular in speech recognition

RNN Gate



$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Long Short Term Memory (LSTM)

Why ?

- *The AMSI summer school is ??*

- *Yony grew up in Israel. He loves playing with his kids. He is aslo a great teacher and colleagues. He is fluent in ???*}

Long Short Term Memory (LSTM)

- What makes LSTM cell special ?
- How do LSTM cell achieve long term dependency ?
- How does it know what information to keepp ?
- What information to discard from mermory ?

Solution exploiting gates: input, forget and output

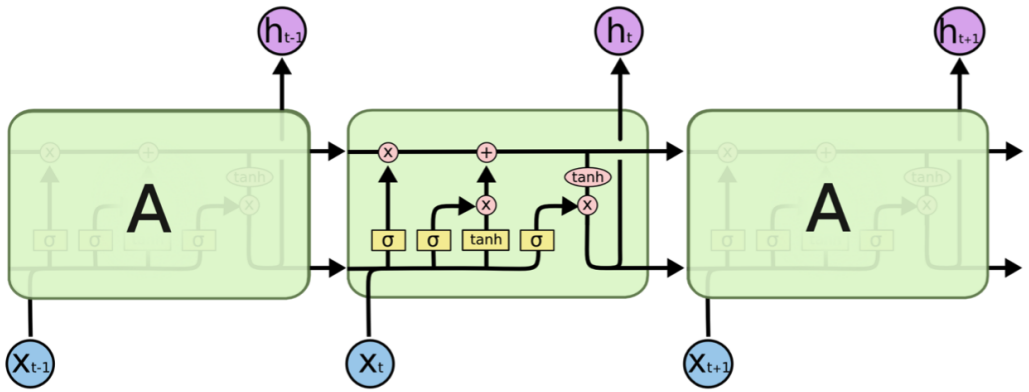
Difference between RNN and LSTM

- RNN cell: Hidden state h_t
  - for storing information
  - making prediction

- LSTM: Hidden state is broken into two states
  - (1) Cell state: called internal memory where all information will be stored
  - (2) Hidden state: used for computing the output

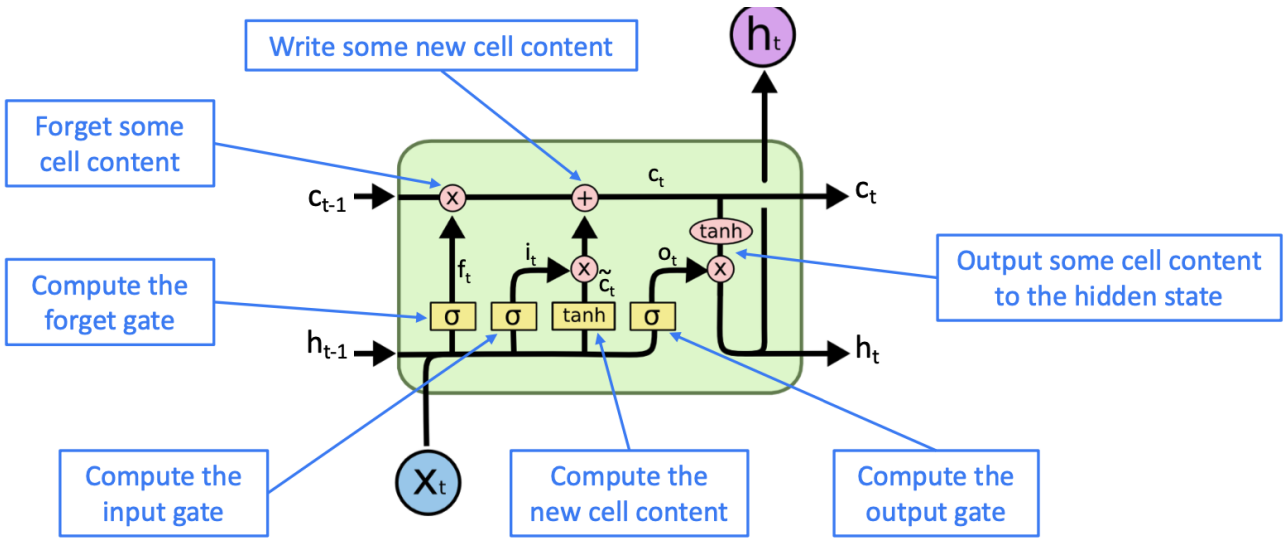
Long Short Term Memory (LSTM)

- LSTM is used to take into account long-term dependencies and was introduced by Hochreiter and Schmidhuber in 1997 to offer a solution to the vanishing gradients problem.
- LSTM models make each node a more complex unit with gates controlling what information is passed through rather each node being just a simple RNN cell.



Main ideas

- At each time t, LSTM provides a **hidden state** ($h^{(t)}$) and a **cell state** ($c^{(t)}$) which are both vectors of length n. The cell has the ability to stores *long-term information*.
- Further, the LSTM model can *erase*, *write* and *read* information from the **cell**.
- Gates are defined to get the ability to selection which information to either erased, written or read. Gates are vectors of length n. At each step t the gates can be *open* (1), *closed* (0) or somewhere in-between.



From a sequence of inputs $x^{(t)}$, LSTM computes a sequence of hidden states $h^{(t)}$, and cell state $c^{(t)}$:

- **Forget gate:** controls what is kept versus forgotten, from previous cell state

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$

- **Input gate:** controls what parts of the new cell content are written to cell

$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$

- **Output gate:** controls what parts of cell are output to hidden state

$$o^{(t)} = \sigma\left(W_o h^{(t-1)}) + U_o x^{(t)} + b_o\right)$$

- **New cell content:** this is the new content to be written to the cell

$$\widetilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^t + b_c\right)$$

- **cell state:** erase (“forget”) some content from last cell state, and write (“input”) some new cell content

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \widetilde{c}^{(t)}$$

- **Hidden state:** read (“output”) some content from the cell

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

Some Remarks

- LSTM have been largely exploited for handwritting recognition, speech recognition, machine translation, parsing, image captioning.
- The architecture of LSTM model is especially designed to preserve information over many timesteps.
- LSTM uses **gates** to control the *flow of information*

Some Remarks

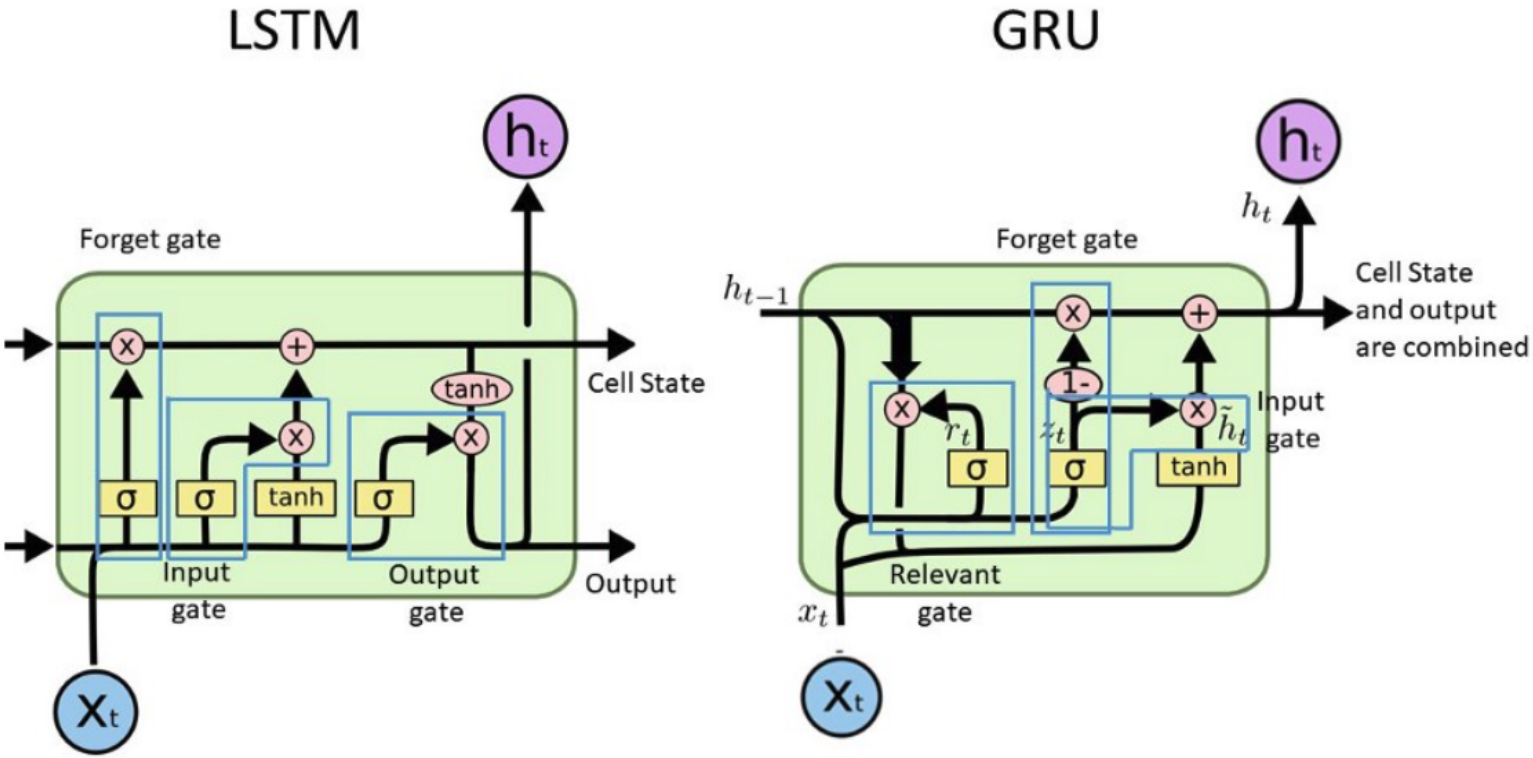
- Backpropagating from $c^{(t)}$ to $c^{(t-1)}$ is only element-wise multiplication by the f gate, and there is no matrix multiplication by W.
- The f gate is different at every time step, ranged between 0 and 1 due to **sigmoid property**, thus it overcome the issue of multiplying the same thing over and over again.
- The Backpropagation through time with uninterrupted gradient flow helps for the vanishing gradient issue even if LSTM does not **guarantee** vanishing/exploding gradient issues.

LSTM Forward and Backward Pass

A nice visual explanation here

Gated Recurrent Units (GRU)

- **GRU** has been proposed by Cho et al. in 2014 as an alternative to the LSTM.
- There is no cell state and at each times step t. There are an input $x^{(t)}$ and hidden state $h^{(t)}$.



- **Update gate:** controls what parts of hidden state are updated vs preserved

$$u^{(t)} = \sigma\left(W_u h^{(t-1)} + U_u x^{(t)} + b_u\right)$$

- **Reset gate:** controls what parts of previous hidden state are used to compute new content

$$r^{(t)} = \sigma\left(W_r h^{(t-1)} + U_r x^{(t)} + b_u\right)$$

- **New hidden state content:** reset gate selects useful parts of previous hidden state. Use this and current input to compute new hidden content.

$$\widetilde{h}^{(t)} = \tanh\left(W_r(r^t \circ h^{(t-1)}) + U_h x^{(t)} + b_h\right)$$

- **Hidden state:** update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$h^{(t)} = (1 - u^{(t)}) \circ h^{(t-1)} + u^{(t)} \circ \widetilde{h}^{(t)}$$

- Comparison of LSTM and GRU models (Jozefowicz et al 2015)

Auto-encoders for language translation

Machine translation (MT) is one of the main active research area in Natural Language Processing (NLP).

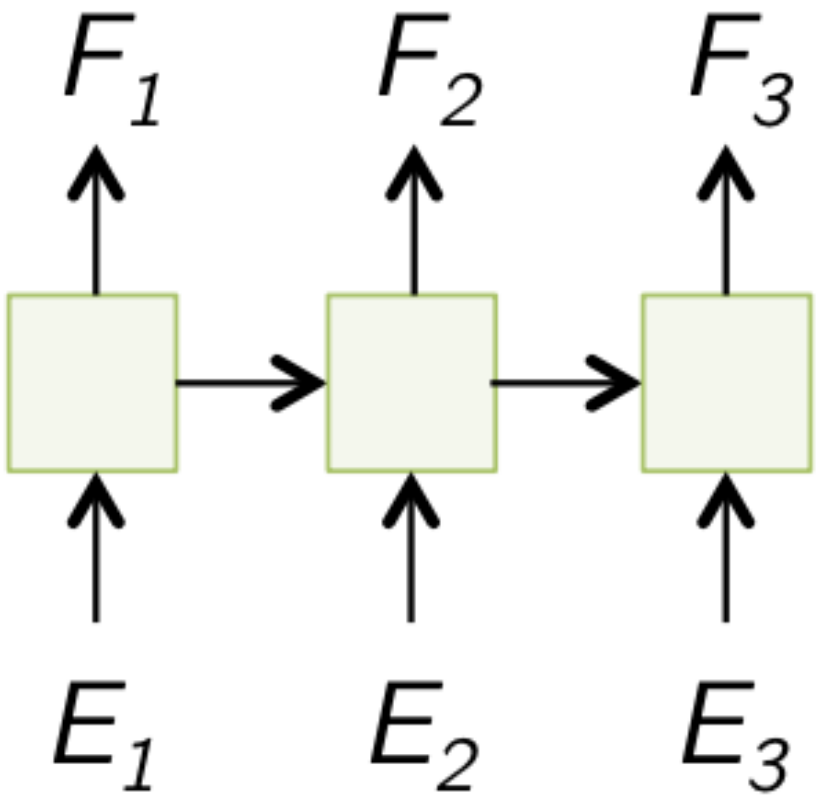
- The **goal** is to provide a fast and reliable computer program that translates a text in one language (**source**) into another language (**the target**)

Using neural network model, the main architecture used for MT is the **encoder–decoder** model:

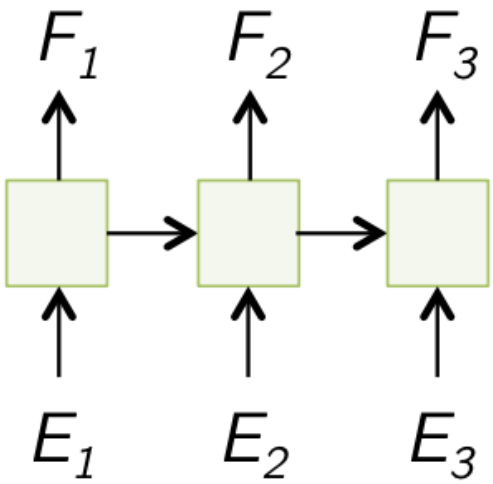
- **encoder part** which summarizes the information in the source sentence
- **decoder part** based on the encoding, generate the target-language output in a step-by-step fashion

Question for you

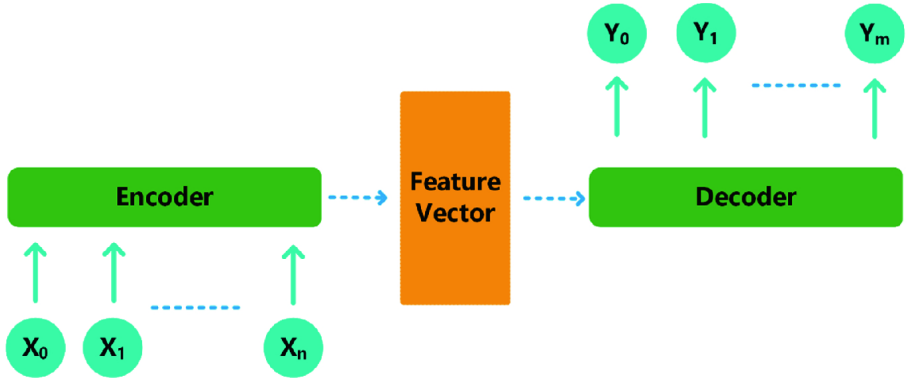
- Consider the problem of translation of English to French
- E.g. *What is your name = Commet tu t'appelle*
- Is this archiceture suitable for this problem ?



Tentative solution



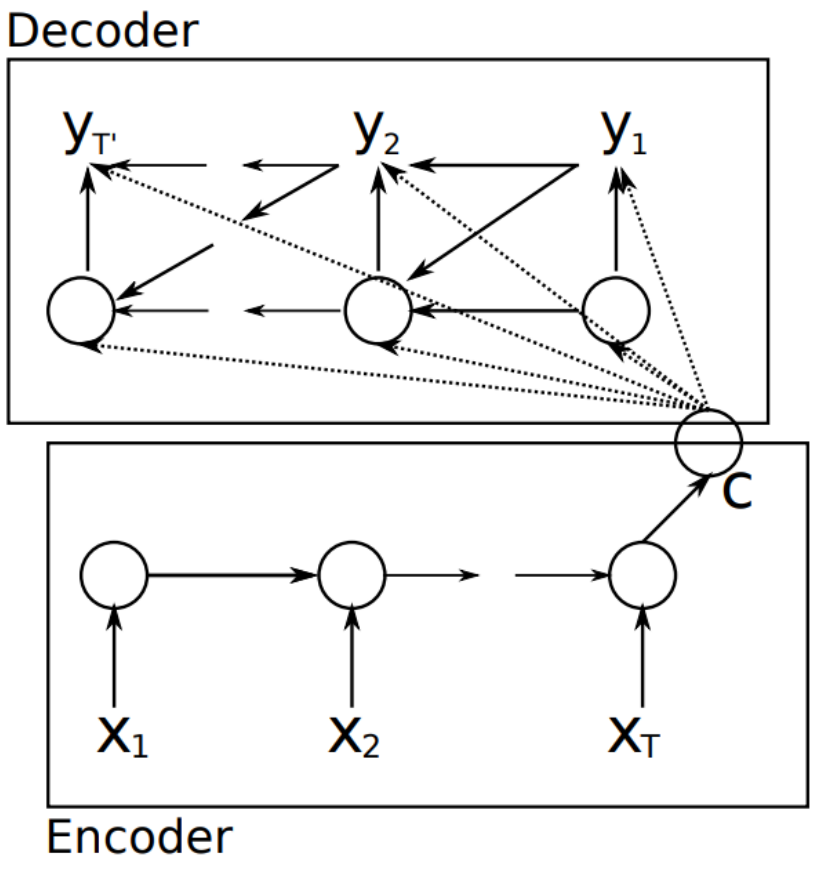
- **encoder part** which summarizes the information in the source sentence
- **decoder part** based on the encoding, generate the target-language output in a step-by-step fashion



- Introduced by Cho et al. (2014).
- A variant called **sequence-to-sequence** model has been introduced by Sutskever et al., (2014)

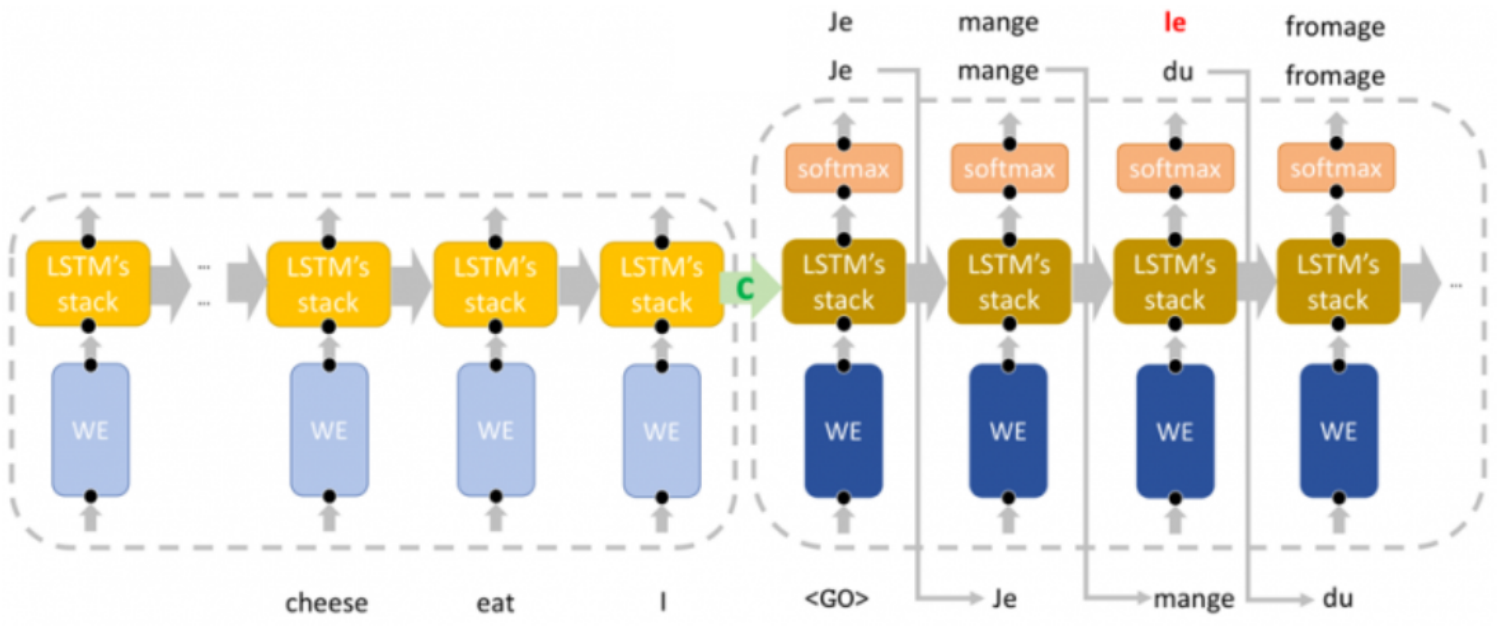
Cho's model

Principle: the encoder and decoder are both **GRUs**. The final state of the encoder is used as the *summary c* and then this summary is accessed by all steps in the decoder



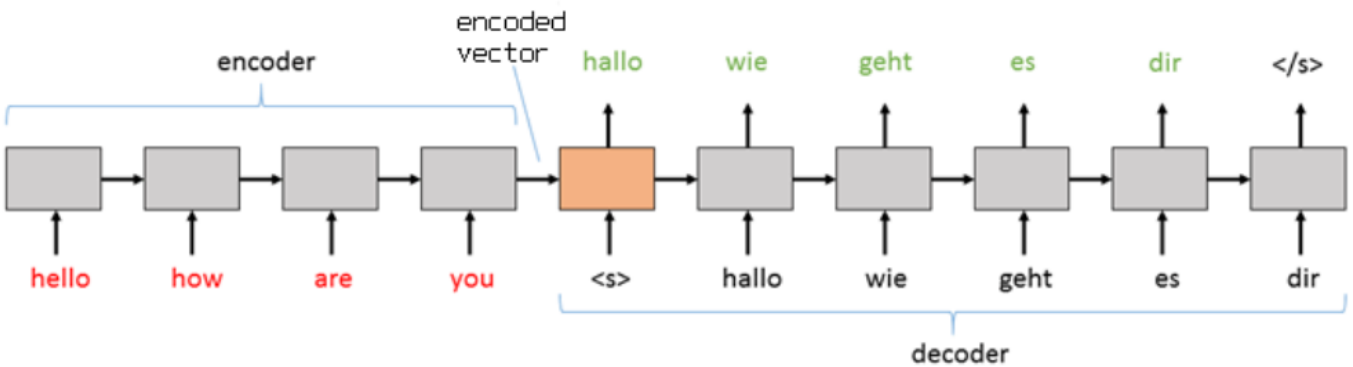
Sutskever's model

Principle. In this model the encoder and decoder are **multilayered LSTMs**. The final state of the encoder becomes the initial state of the decoder. So the source sentence has to be **reversed**.

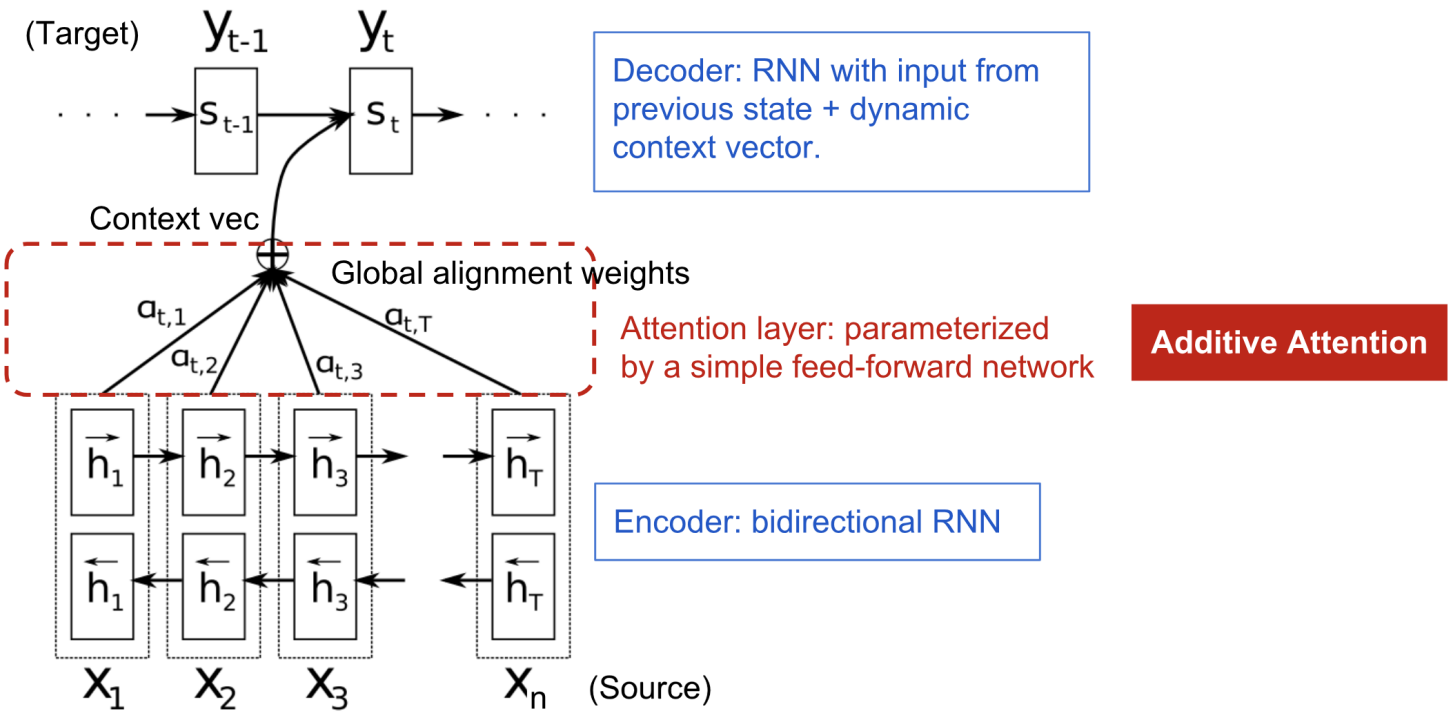


Improvement with Attention layer

- Limitation of these models are exposed in the paper "On the Properties of Neural MachineTranslation...". They showed that the performance of the encoder-decoder network degrades rapidly as the length of the input sentence increases.
- **Main drawback** of the previous models is that the encoded vector need to capture the entire phrase (sentence) and might skipped many important details. Moreover the information needs to “flow” through many RNN steps which is quite difficult for long sentence.



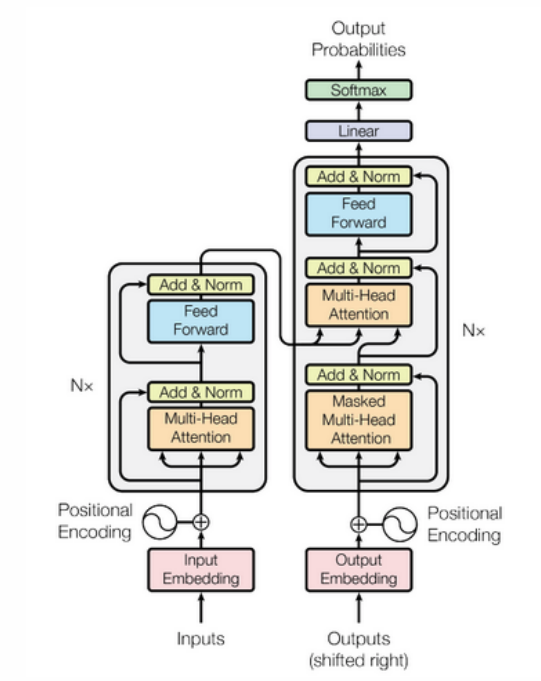
- Bahdanau et al. (2015) have proposed to include **attention layer** which consist to include attention mechanisms to give more importance to some of the input words compared to others while translating the sentence.



- A survey of the different implementations of attention model is presented by Galassi et al. (2019)

Transformer

Recently, Vaswani et al., 2017) offered a kind of revolution for Machine translation architecture: "Attention is all you need".



This architecture is called the transformer which offers **encoder-decoder structure** that uses **attention** for information flow.

Further reading for MT

- Understand **attention concept**
- Understand **attention interface**
- A comprehensive illustration of the transformer
- Chen et al. (2018) who compared several types of recent MT models
- A comprehensive PyTorch-based seq2seq tutorial by Joost Bastings:The Annotated Encoder–Decoder with Attention

Take Home Message

- Why RNN ?
- LSTM
- Backpropagation through time
- Attention layer
- Auto-Encoder
- GRU