Introduction to Deep Learning

Recurrent networks

J. Rynkiewicz

Université Paris 1

2022

Recurrent neural networks

Recurrent neural networks are defined in terms of time (or iteration of the algorithm) t :

- Let us write :
  - X_t, the observation at time t.
  - h_t the hidden state vector of the network.
  - Y_t the output of the network at time t.
  - θ the parameter vector of the network.
- We initialize the hidden state vector h_0.
- The dynamic system implemented by the network will be :

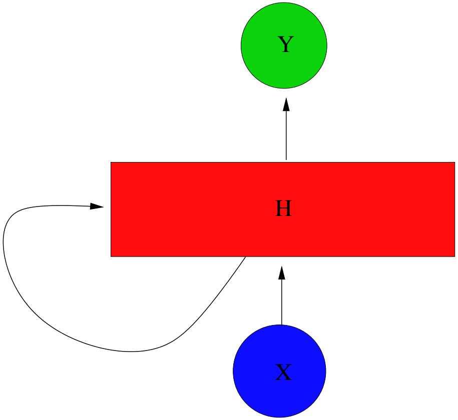
$$\begin{cases} h_t = g_\theta(h_{t-1}, X_t) \\ Y_t = f_\theta(h_t) \end{cases}$$

- We can notice that h_{t-1} is a function of h_{t-2} and X_{t-1} and recursively we can go back to X_1. Thus, Y_t can be a function of all the past observations.

Principle of recurrent networks

These networks allow to take into account the state of the hidden layer at the previous time to influence the hidden layer at the present time :

$$\begin{cases} h_t = g_\theta(h_{t-1}, X_t) \\ Y_t = f_\theta(h_t) \end{cases}$$



Simple recurrent network

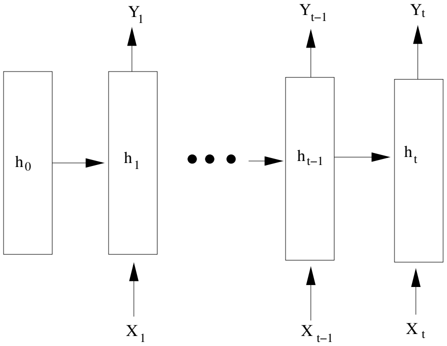
- Suppose that the observations $(X_1, \cdots, X_n)$ are in an alphabet $\mathcal{A}$ of cardinal $d = |\mathcal{A}|$.
- The i-th symbol of the alphabet is coded by a vector of $\mathbb{R}^d$ with only zeros except on 1 in i-th position.
- For $u = (u_1, \cdots, u_d) \in \mathbb{R}^d$, let us write $softmax(u) = \left(\frac{\exp(u_1)}{\sum_{j=1}^{d}\exp(u_j)}, \cdots, \frac{\exp(u_d)}{\sum_{j=1}^{d}\exp(u_j)}\right).$
- In one of its simplest forms, the equation of a recurrent network with a hidden state of dimension H will be :

$$\begin{cases} h_0 = 0_{\mathbb{R}^H} \\ \text{si } t \geq 1 \ : \\ h_t = \tanh(W_{hh}h_{t-1} + W_{hx}X_t + b_h) \\ Y_t = softmax(W_{yh}h_t) \end{cases}$$

where $\theta = (b_h, W_{hx}, W_{hh}, W_{yh}) \in \mathbb{R} \times \mathbb{R}^{H\times d} \times \mathbb{R}^{H\times H} \times \mathbb{R}^{d\times H}$ is the parameter vector of the model.

Diagram of a simple recurrent network

A recurrent network is a “feedforwfard” network that “lengthens” with the sequence of observations and whose weights, from one layer to another, are forced to be equal.



On this diagram, the arrows represent the weights of the network.

The problem of optimizing recurrent networks

The gradient is computed by back-propagation of a quantity δ through the layers (all identical), following a linear application of the type :

$$\delta_{l+1} = W_l \delta_l.$$

Since the network "grows" with the number of observations, the number of back-propagations too, different phenomena can happen :

- If the matrix W_l is contracting : $\|X\| < \|W_l X\|$, the quantity δ_l will converge exponentially fast to 0 and the gradient becomes quickly negligible for the layers far from the observation t. The network then forgets exponentially fast the observations far away in time.
- On an other hand, if $\|X\| > \|W_l X\|$, the quantity $delta_l$ will explode and so will the gradient. The observations of the past will have very large weight.

The solution to this problem is to equip the network with parametric functions, which remember or forget the past according to the observations.

The networks GRU (Gated Recurrent Units)

Let us write $\sigma((u_1,\cdots,u_d)^T)=\left(\frac{exp(u_1)}{1+exp(u_1)},\cdots,\frac{exp(u_d)}{1+exp(u_d)}\right)^T$ the sigmoidal function applies to a vector, $\circ$ the Hadamard product (i.e. pointwise) of two matrices. The equations verified by a GRU are as follows :

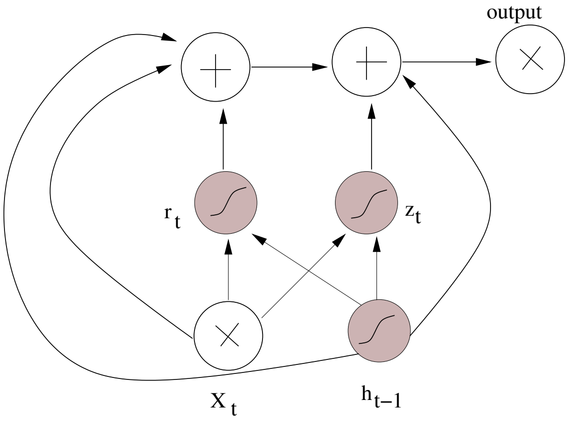
$$\begin{cases} h_0=0\\ z_t=\sigma\left(W_{zh}h_{t-1}+W_{zx}X_t+b_z\right)\\ r_t=\sigma\left(W_{rh}h_{t-1}+W_{rx}X_t+b_r\right)\\ h_t=(1-z_t)\circ h_{t-1}+z_t\circ\sigma\left(W_{hx}X_t+W_{hh}(r_t\circ h_{t-1})+b_h\right)\\ Y_t=softmax(W_{yh}h_t)\end{cases}$$

with :

- z_t the “update gate”, if it is close to 0, h_t will be close to h_{t-1} (we fully remember the past).
- r_t the “reset gate”, if it is close to 0 and z_t is close to 1 (the past will be forgotten).

Illustration of GRU

GRU can be illustrated as follows :



The networks LSTM (Long Short Term Memory)

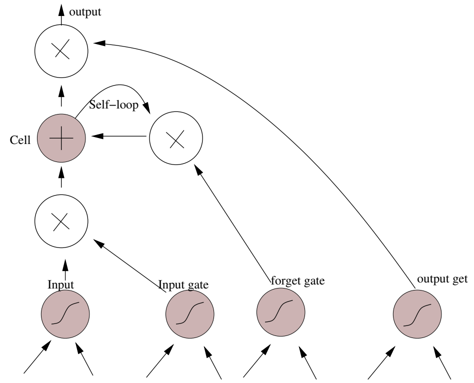
With the same notations as before we have the LSTM equations :

$$\begin{cases} h_0 = 0 \\ f_t = \sigma\left(W_{fh}h_{t-1} + W_{fx}X_t + b_f\right) \\ i_t = \sigma\left(W_{ih}h_{t-1} + W_{ix}X_t + b_i\right) \\ o_t = \sigma\left(W_{oh}h_{t-1} + W_{ox}X_t + b_o\right) \\ c_t = f_t \circ c_{t-1} + i_t \circ \sigma\left(W_{cx}X_t + W_{ch}h_{t-1} + b_c\right) \\ h_t = o_t \circ \sigma(c_t) \\ Y_t = softmax(W_{yh}h_t) \end{cases}$$

- f_t is the forget gate, if the j component of f_t is close to 1, the j component of c_t will be close to the j component of c_{t-1} (we fully remember the past).
- i_t is the input gate, so if the j component of i_t is close to 1, the current input X_t may have a large influence on the j component of c_t.
- o_t is the output gate, if the j component of o_t is close to 1, the j component of h_t will be close to the j component of c_t, otherwise it will be close to 0.
- c_t is the hidden cell (a bit of a precursor to the hidden state h_t).

Illustration of the hidden cell equations c_t

The equations for the hidden cell c_t can be illustrated as follows :



Estimation of parameters

- $\theta_{n+1} = \theta_n - \gamma \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ln(L_\theta(x_{t+i}, y_{t+i}))}{\partial \theta}$. During the algorithm, γ decreases regularly.
- We use Adagrad or Adam to speed up the gradient descent. These algorithms improve the stochastic gradient method by automatically determining a learning rate for each parameter.
- We can use regularization techniques such as weight decay and drop-out.
- We split the learning set in two :
  - *n* first data for the learning set.
  - *m* data for the validation set (hold out).
- The proportion of data in the training set and data in the validation set depends on the number of available data.

An example : “characteres” models for texts (1)

- We consider that a text is a sequence of characters.
- The characters belong to an alphabet $\mathcal{A} = \{a_1, \cdots, a_d\}$ (letters of the alphabet, punctuation, spaces, line breaks).
- We will try to predict the next character according to the past characters.
- Markov chains seem to be a reasonable model : For $k \in \mathcal{A}$

$$P(X_t = k | X_{t-1}, \cdots, X_1) = P(X_t = k | X_{t-1}, \cdots, X_{t-p})$$

- Recurrent networks, especially with gates, allow to implement such models by keeping a reasonable number of free parameters and by allowing the order of the Markov chain p to be relatively large.
- A canonical encoding of characters $a_1, \cdots, a_d$ is on the simplex (one hot encoding) :

$$a_i = (0, \cdots, 0, 1, 0, \cdots 0) \text{ with 1 in i-th position.}$$

An example : “characteres” models for texts (2)

- The output of the recurrent network will be the probability of the next character : $Y_t = (P(X_{t+1} = a_1), \cdots, P(X_{t+1} = a_d))^T$.
- The weights are estimated by minimizing the opposite of the log-likelihood.
- The selected network will be the one that minimizes the classification error on the validation set.
- Once the network is trained, we will generate texts by randomly drawing the next character using the probabilities calculated by the network.
- The generated characters will be used by the network to generate the following characters.
- To generate highly probable characters, we will use a probability vector of the form : $softmax(\frac{1}{T} W_{yh} h_t)$, where T is the temperature.
- The smaller T is, the more likely characters will be drawn.

An example : “characteres” models for texts (3)

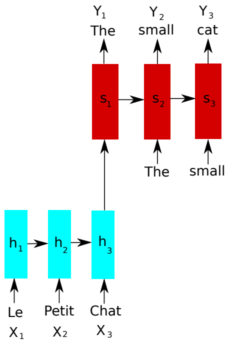
- We estimated an LSTM with 3 hidden layers and 512 hidden units on each layer on the complete work of Shakespeare (1.4 million of characters).
- The step size of the gradient was $\gamma = 0.001$, we went over the training data 10 times.
- The last 200,000 letters were used for the validation set.
- We then generated a text with a temperature $T = 0.3$, here is an extract :
  - CASSANDRA : The world is strong and sick, but that I would The sun that strikes to come to be the season.
  - FERDINALUS : What would you think the bright and sound of men That is a second strength to strike the sun ?
  - CASSIUS : What is this the devil shall I see the state ?
  - CASSIUS : The duke and honest country with the state Which the most father shall be subject’d with him.
  - KING JOHN : A little sword and the most part of death, And then the sun to her that makes the fool The state of soldiers and the little gates And the conflict of this command and mouth The way to see the true and subjects forth.

The encoder-decoder architecture

Until 2017, recurrent networks were used for machine translation.

- An encoder "reads" the sentence to be translated. The last state of the hidden layer encoded this sentence.
- A decoder was initialized with this hidden layer of the encoder, it then generated the translated sentence recursively.
- This model can be summarized by the equation :

$$P\left(y_t|y_1,\cdots,y_{t-1},x\right) = g\left(y_{t-1},s_t,h_T\right)$$



The attention mechanism

In the years 2015 some authors have proposed an attention mechanism (Neural Machine translation by jointly learning to align and translate, Bahdanau et al. (ICLR 2015)).

- Let us write $x = \left(x_1, \cdots, x_{T_x}\right)$ the inputs of the encoder and $c = q\left(h_1, \cdots, h_{T_x}\right)$ the context, where $h_t \in \mathbb{R}^d$ is the hidden state at time t.
- For the previous "encoder-decoder", we have $q\left(h_1, \cdots, h_{T_x}\right) = h_{T_x}$. The authors therefore propose a richer context.
- If we write $\mathbf{y} = (y_1, \cdots, y_T)$ the predicted words, we can summarize this model by the equations :

$$P(\mathbf{y}) = \prod_{t=1}^{T} P\left(y_t \,|y_1, \cdots, y_{t-1}, \mathbf{x}\right)$$

$$P\left(y_t|y_1, \cdots, y_{t-1}, \mathbf{x}\right) = g\left(y_{t-1}, s_t, c_t\right) \text{ et } s_t = f\left(s_{t-1}, y_{t-1}, c_t\right)$$

where s_t is the hidden layer of the decoder at time t and

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j \text{ avec } \alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp e_{tk}} \text{ et } e_{tj} = a\left(s_{t-1}, h_j\right)$$

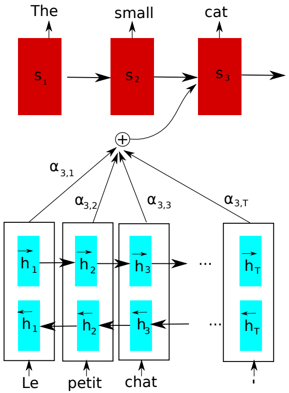
- $a\left(s_{t-1}, h_j\right)$ is a parametrical function, for example, a small neural network.

Illustration of the attention mechanism

$$P\left(y_t|y_1,\cdots,y_{t-1},\mathbf{x}\right)=g\left(y_{t-1},s_t,c_t\right)\text{ et } s_t=f\left(s_{t-1},y_{t-1},c_t\right)$$

where s_t is the hidden layer of the decoder at time t and

$$c_t=\sum_{j=1}^{T}\alpha_{tj}h_j\text{ avec }\alpha_{tj}=\frac{\exp(e_{tj})}{\sum_{k=1}^{T}\exp e_{tk}}\text{ et } e_{tj}=a\left(s_{t-1},h_j\right)$$



Evolution of the attention mechanism

- The attention mechanism is based on the alignment of the hidden layer of the decoder at time t and that of the encoder at time j : $\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp e_{tk}}$ where $e_{tj} = a\left(s_t, h_j\right)$.
- We can consider two alternatives for $a\left(s_t, h_j\right)$:

$$\begin{cases} a\left(s_t, h_j\right) = v_a \tanh(L_a(s_t, h_j)) \text{ “mini” réseau de neurones} \\ \text{ou bien} \\ a\left(s_t, h_j\right) = s_t^T W_a h_j \text{ produit scalaire généralisé} \end{cases}$$

- If we choose W_a well, the generalized scalar product is as efficient and simpler than the "mini" neural network.
- It turned out that the attention mechanism is the key element of natural language processing models. (Attention is all you need, Vaswani et al. NIPS 2017).
- Attention-only models (without recurrent neural networks) are now the state of the art for natural language processing.