Chapter 10 - Part I

Sequence Modeling: Recurrent and Recursive Nets

César Bragagnini

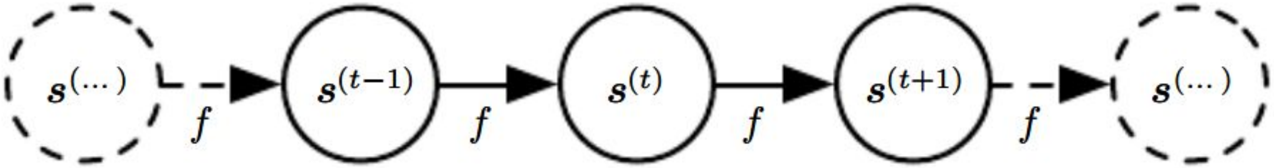
cesarbrma91@gmail.com
@MarchBragagnini

Universidad Católica San Pablo

Outline

1. Unfolding Computational Graphs.
2. Recurrent Neural Networks.
3. Training Recurrent Neural Networks.
4. Bidirectional RNNs.
5. Encoder-Decoder Sequence-to-Sequence.
6. Deep Recurrent Networks.
7. Recursive Neural Networks.
8. References.

Unfolding Computational Graphs



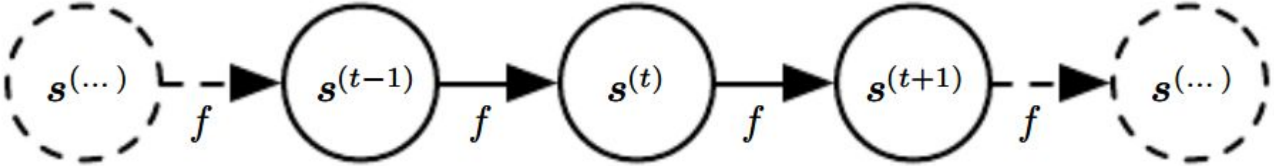
$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$$

Recurrency:

?

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

Unfolding Computational Graphs



$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$$

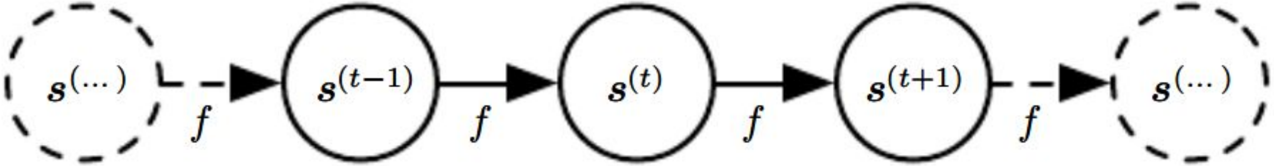
activation function ?

Recurrency:

$$\boxed{\begin{aligned} \boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}) \end{aligned}}$$

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

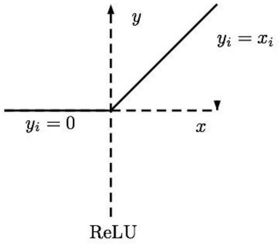
Unfolding Computational Graphs

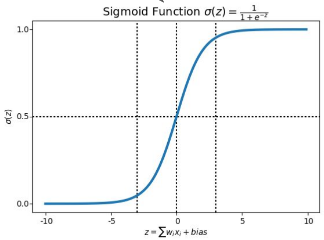


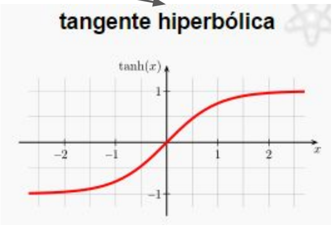
$$s^{(t)} = f(s^{(t-1)}; \theta)$$

Recurrency:

$$\boxed{\begin{aligned} s^{(3)} &= f(s^{(2)}; \theta) \\ &= f(f(s^{(1)}; \theta); \theta) \end{aligned}}$$



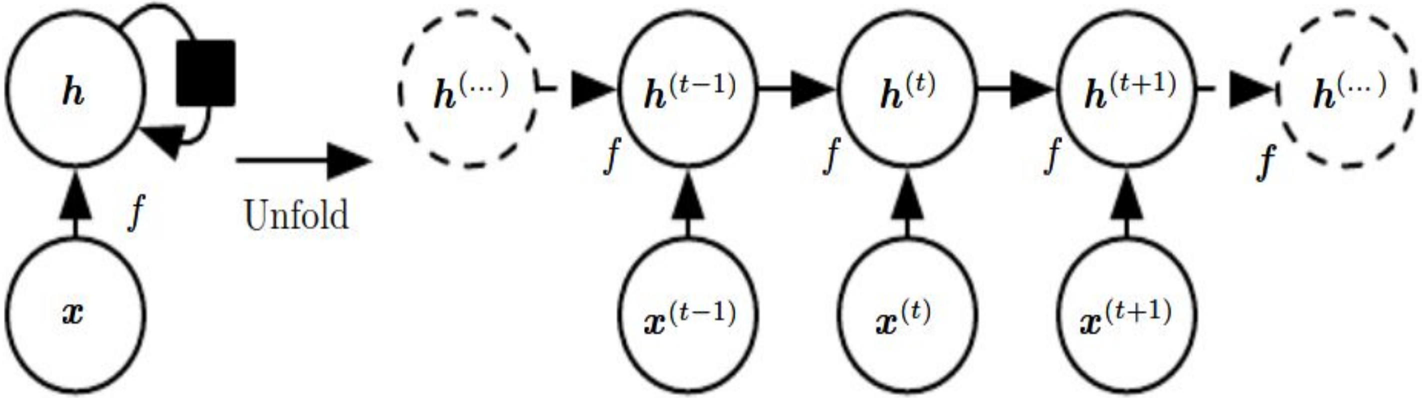




[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

Unfolding Computational Graphs

The black square indicates a delay of a single time step

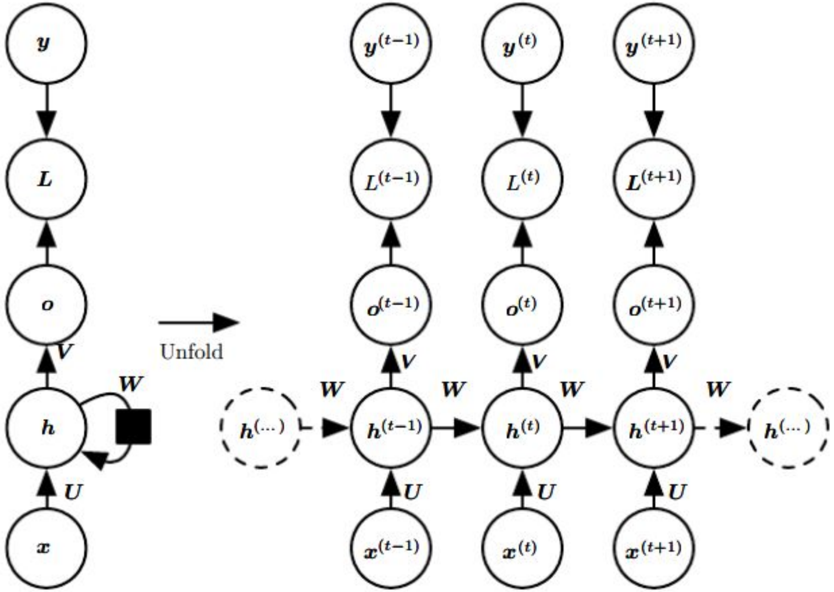


$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

RNN folded and the same RNN as an unfolded computational graph

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

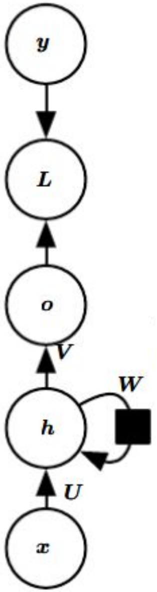
Recurrent Neural Networks



RNN unfolded and folded. This model ***a distribution in which they values are conditionally independent from each other given the x values***.

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

Recurrent Neural Networks



RNN unfolded and folded.

$$\boldsymbol{h}_t = f(\boldsymbol{W}_{hx}\boldsymbol{x}_t + \boldsymbol{W}_{aa}\boldsymbol{h}_{t-1} + \boldsymbol{b}_a)$$
$$\boldsymbol{h}_t = f(\boldsymbol{W}_h[\boldsymbol{x}_t; \boldsymbol{h}_{t-1}] + \boldsymbol{b}_a)$$
$$\boldsymbol{o}_t = f(\boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o)$$
$$\hat{\boldsymbol{y}}_t = \mathrm{Softmax}(\boldsymbol{o}_t)$$

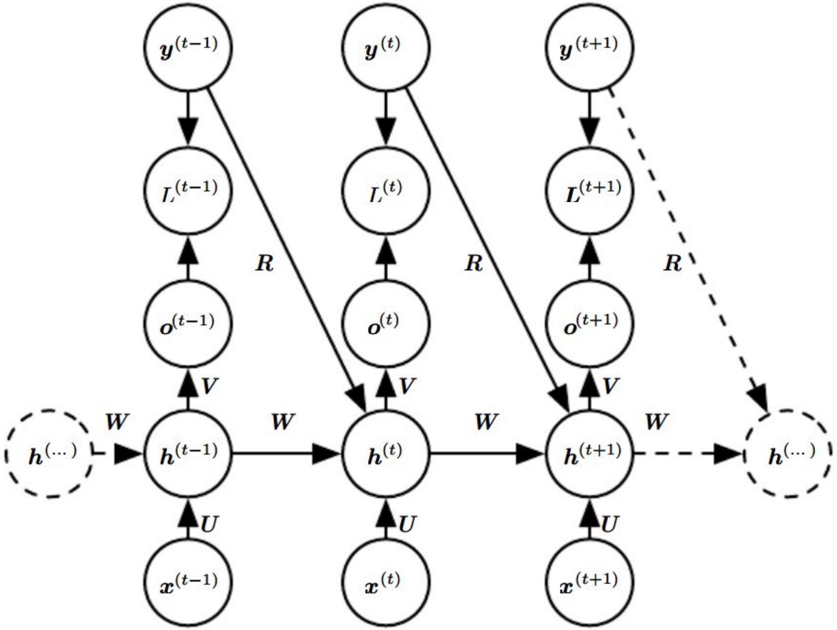
Loss Function and Cost Function

$$\mathcal{L}^t(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t) = -\boldsymbol{y}_t \log \hat{\boldsymbol{y}}_t - (\vec{1} - \boldsymbol{y}_t)\log(\vec{1} - \hat{\boldsymbol{y}}_t)$$
$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{t=1} \mathcal{L}^t(\hat{\boldsymbol{y}}_t, \boldsymbol{y}_t)$$

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

Recurrent Neural Networks

Modeling Sequences Conditioned on Context with RNNs



This RNN ***contains connections from the previous output to the current state***.These connections allow this RNN to model an arbitrary distribution over sequences of y given sequences of x of the same length.

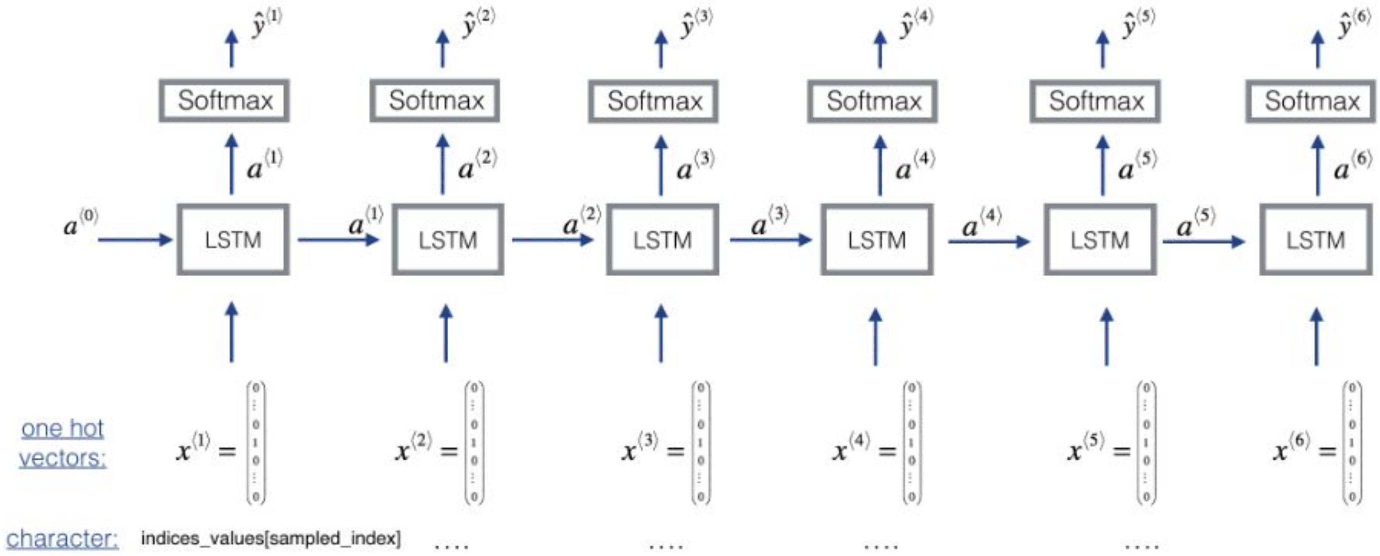
[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

Recurrent Neural Networks

Application

Language Modeling Train

$$x^{\langle i+1 \rangle} = y^{\langle i \rangle}$$

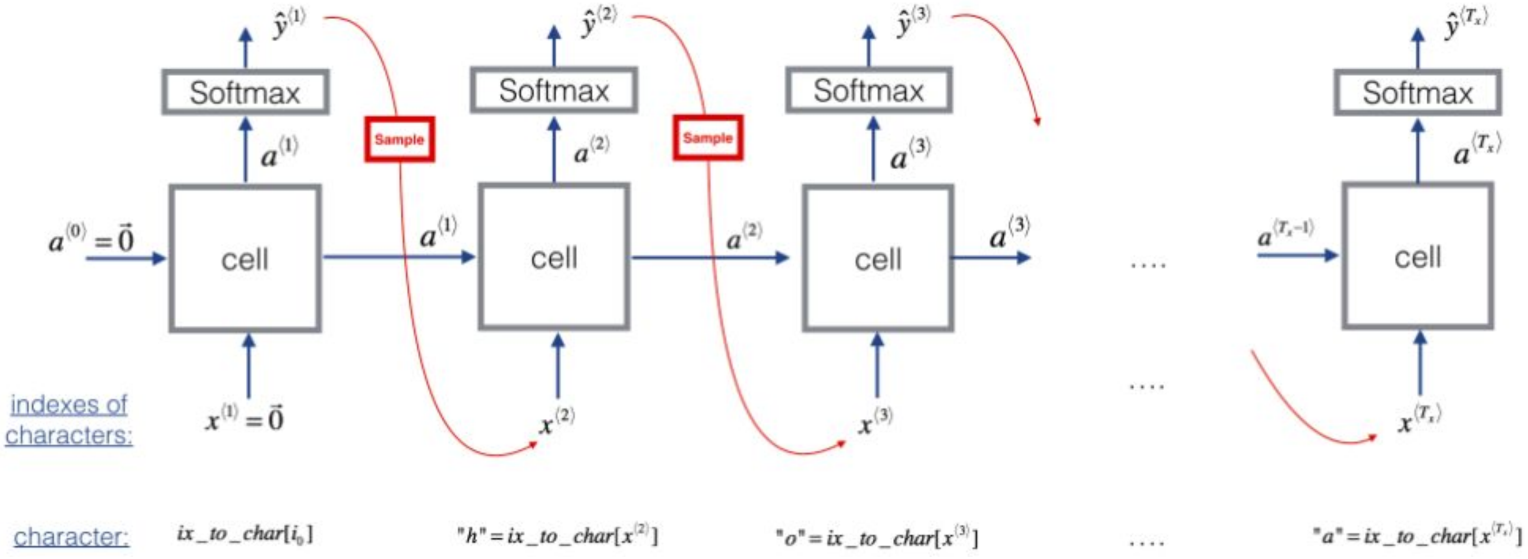


[ ] Deep Learning Specialization, Sequence Models, Andrew Ng, 2016.

Recurrent Neural Networks

Application

Language Modeling Testing



[ ] Deep Learning Specialization, Sequence Models, Andrew Ng, 2016.

Training Recurrent Neural Networks

Teaching forcing

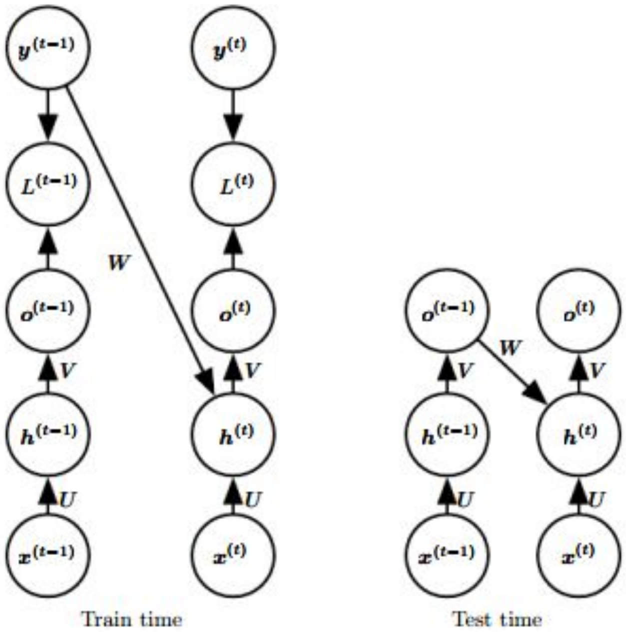


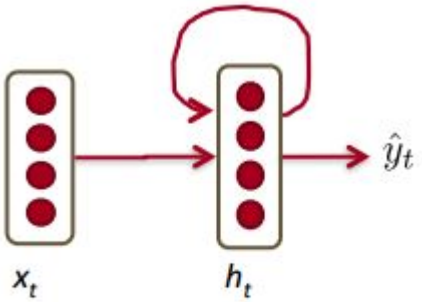
Illustration of teacher forcing. Teacher forcing is a training technique that is applicable to RNNs that have connections from their output to their hidden states at the next time step.

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

Training Recurrent Neural Networks

Vanishing/Exploding gradient

$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$



$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \boxed{\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}}$$

[ ] CS224N/Ling284, Lecture 8, Stanford University, Manning , 2016.

Training Recurrent Neural Networks

Vanishing/Exploding gradient

Each partial is a Jacobian

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Analyzing the norms of the Jacobians, yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \le \|W^T\| \|\mathrm{diag}[f'(h_{j-1})]\| \le \beta_W \beta_h$$

Where we defined β's as upper bounds of the norms

The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \le (\beta_W \beta_h)^{t-k}$$

This can become very small or very large quickly

[ ] CS224N/Ling284, Lecture 8, Stanford University, Manning , 2016.
[ ] Learning Long-Term Dependencies with Gradient Descent is Difficult, Bengio, 1994.

Training Recurrent Neural Networks

BackPropagation Through Time

for $t = T \rightarrow 1$ **do**

`// Output backprop`

$\quad ds_t \leftarrow \mathbf{1}_{y_t} - p_t$

$\quad dW_{hy} \leftarrow dW_{hy} + ds_t \cdot h_t^\top$

$\quad dh_t \leftarrow dh_t + W_{hy}^\top \cdot ds_t$

`// RNN backprop`

$\quad dW_{xh} \leftarrow dW_{xh} + (\sigma'(T_{\text{rnn}} z_t) \circ dh_t) \cdot x_t^\top$

$\quad dW_{hh} \leftarrow dW_{hh} + (\sigma'(T_{\text{rnn}} z_t) \circ dh_t) \cdot h_{t-1}^\top$

`// Input backprop`

$\quad dx_t \leftarrow W_{xh}^\top \cdot (\sigma'(T_{\text{rnn}} z_t) \circ dh_t)$

$\quad dh_{t-1} \leftarrow W_{hh}^\top \cdot (\sigma'(T_{\text{rnn}} z_t) \circ dh_t)$

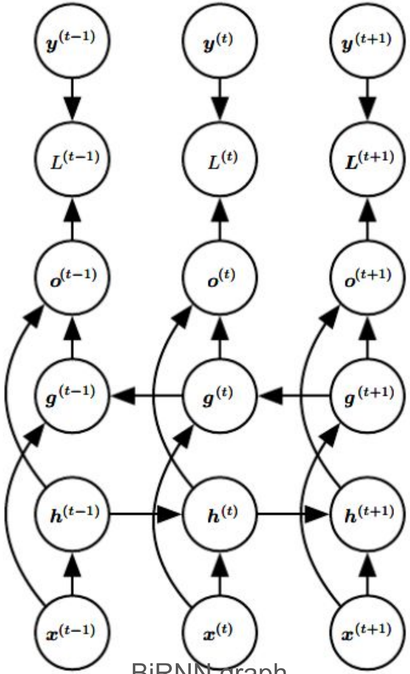
end

BPTT algorithm for vanilla RNNs

[ ] Neural Machine Translation, Minh-Thang Luong, 2016.

Bidirectional RNNs

Based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements. For example, to predict a missing word in a sequence you want to look at both the left and the right context



BiRNN graph.

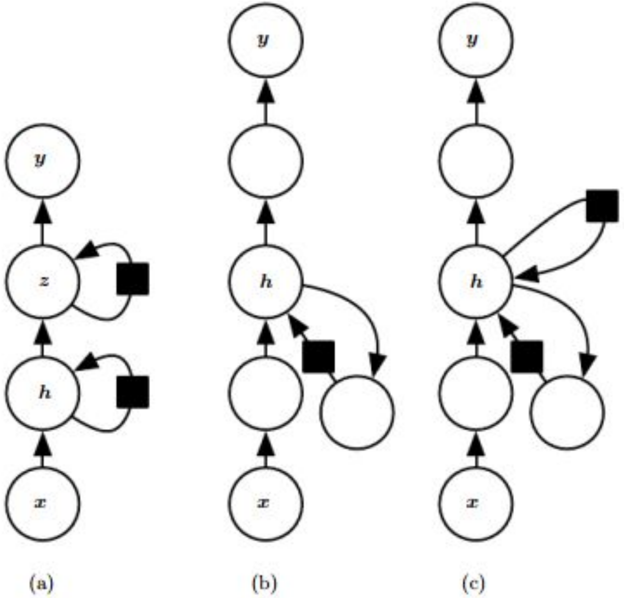
$$
\begin{aligned}
\boldsymbol{h}^{(t-1)} &= f(\boldsymbol{W}_h[\boldsymbol{h}^{(t-2)}, \boldsymbol{x}^{(t-1)}] + \boldsymbol{b}_h) \\
\boldsymbol{h}^{(t)} &= f(\boldsymbol{W}_h[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_h) \\
\boldsymbol{g}^{(t+1)} &= f(\boldsymbol{W}_g[\boldsymbol{g}^{(t+2)}, \boldsymbol{x}^{(t+1)}] + \boldsymbol{b}_g) \\
\boldsymbol{g}^{(t)} &= f(\boldsymbol{W}_g[\boldsymbol{g}^{(t+1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_g)
\end{aligned}
$$

$$\boldsymbol{o}^{(t)} = \hat{\boldsymbol{y}}_{(t)} = \mathrm{Softmax}(\boldsymbol{W}_o[\boldsymbol{h}^{(t)}, \boldsymbol{g}^{(t)}] + \boldsymbol{b}_o)$$

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

Deep Recurrent Networks

Stacking multiple layers on top of each other is useful because they are able to progressively extract more abstract features of the current words or sentences.

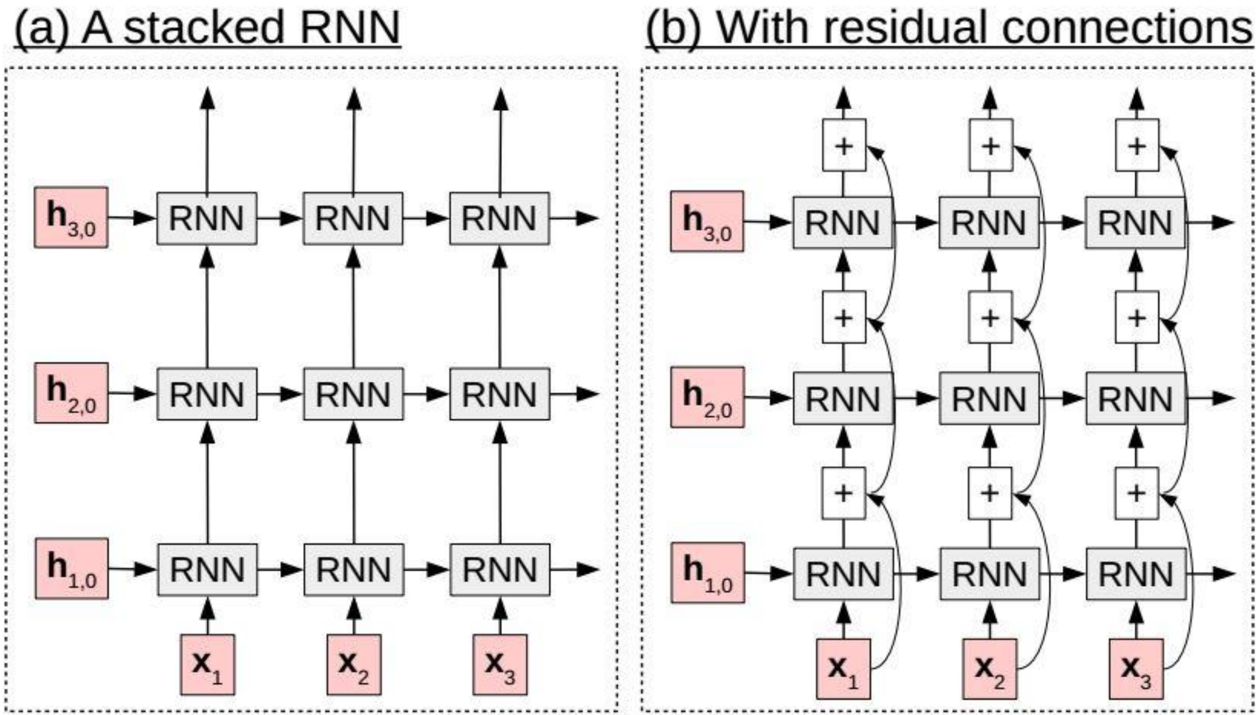


A recurrent neural network can be made deep in many ways. (a)The hidden recurrent state can be broken down into groups organized hierarchically. (b)Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden and hidden-to-output parts. This may lengthen the shortest path linking different time steps. (c)The path-lengthening effect can be mitigated by introducing skip connections.

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

Deep Recurrent Networks

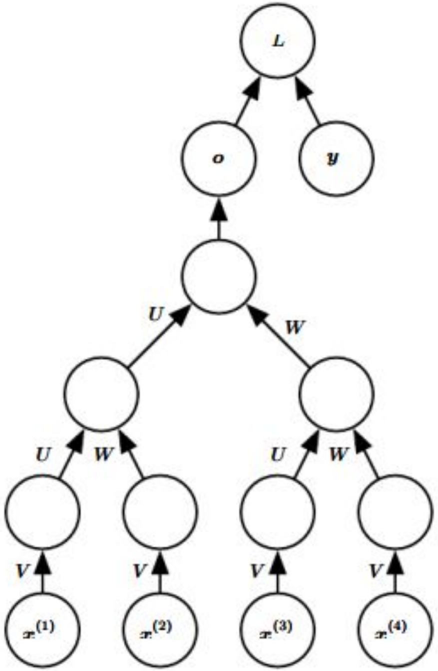
Maybe appear gradient vanish problem in vertical



Solution for gradient vanish problem is **residual connections**

[ ] Neural Machine Translation and Sequence-to-sequence Models: A Tutorial, Graham Neubig, 2017.

Recursive Neural Networks



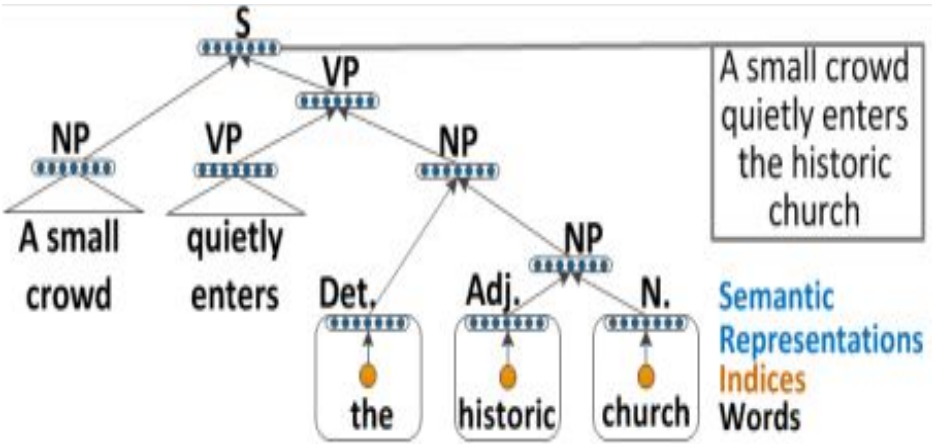
A recursive network has a computational graph that generalizes that of the recurrent network from a chain to a tree. A variable-size sequence x(1), x (2), . . . , x(t) can be mapped to a fixed-size representation (the output o), with a fixed set of parameters (the weight matrices U, V , W).

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016.

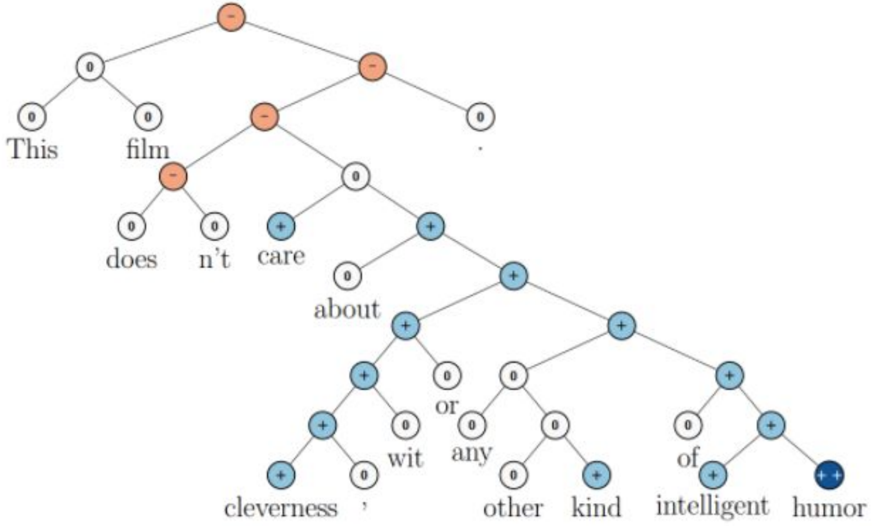
Recursive Neural Networks

Applications:

Parsing Natural Language Sentences



Sentiment Analysis

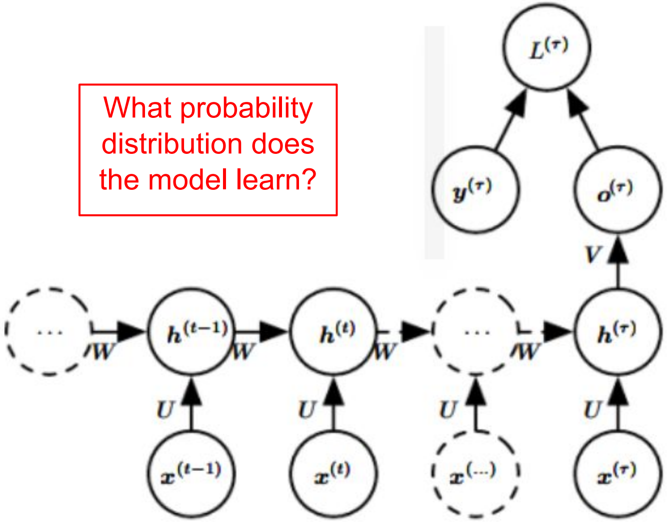


[ ] Parsing Natural Scenes and Natural Language with Recursive Neural Networks, Socher, 2011.
[ ] Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, Socher, 2013.

Encoder-Decoder Sequence-to-Sequence

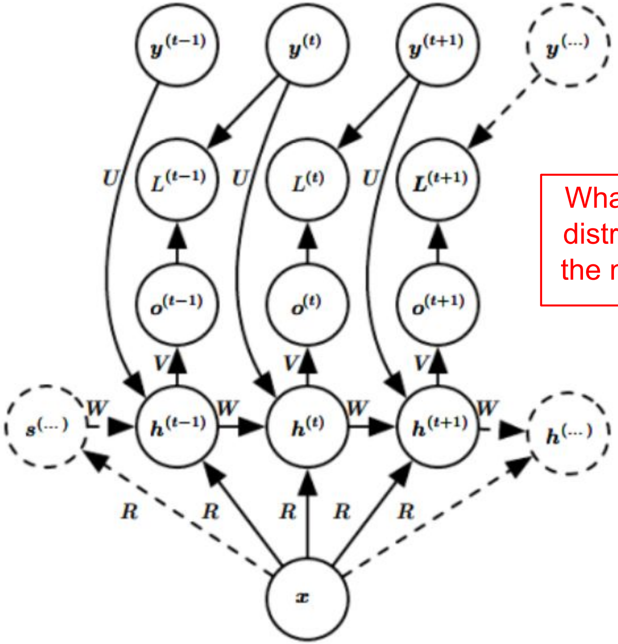
RNN for many-to-one

What probability distribution does the model learn?



RNN for one-to-many

What probability distribution does the model learn?



(a) RNN model to mapping an input sequence $(\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(\tau)})$ to a fixed vector $\boldsymbol{o}^{\tau}$.
(b) RNN model to mapping a fixed vector $\boldsymbol{x}$ to a sequence $(\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{n_y})$.

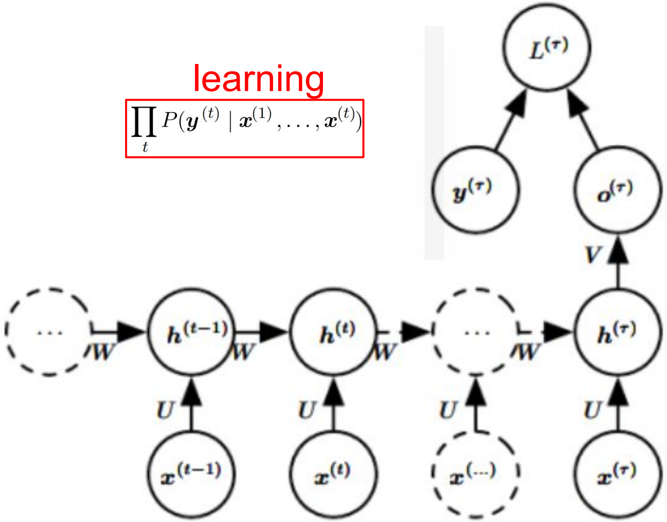
[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016.

Encoder-Decoder Sequence-to-Sequence

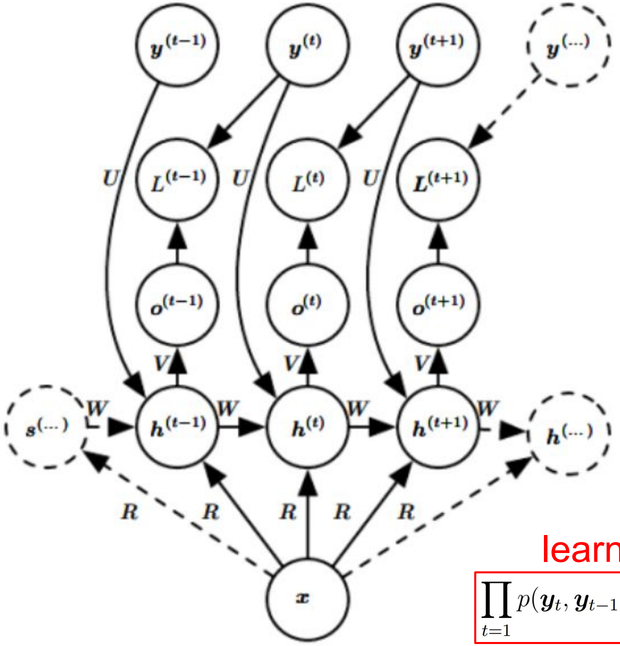
RNN for many-to-one

learning

$$\boxed{\prod_t P(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)})}$$



RNN for one-to-many



learning

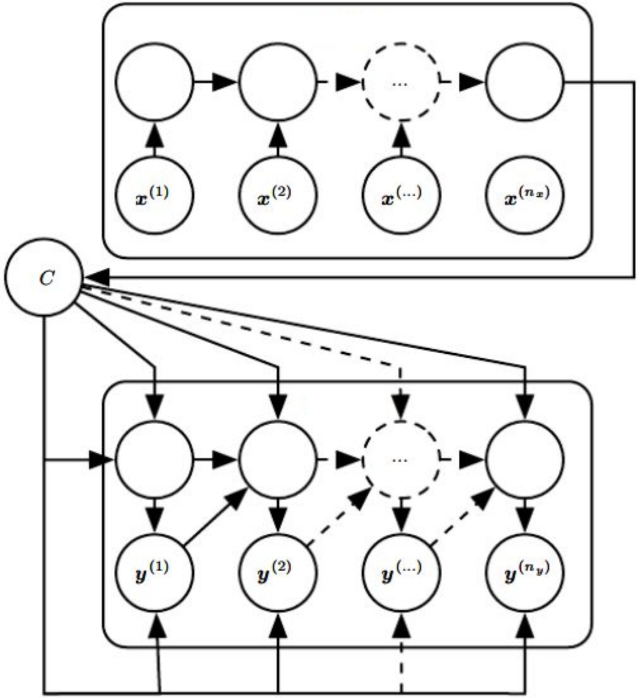
$$\boxed{\prod_{t=1} p(\boldsymbol{y}_t, \boldsymbol{y}_{t-1}, \ldots, \boldsymbol{y}_1 \mid \boldsymbol{x})}$$

(a) RNN model to mapping an input sequence $(\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(\tau)})$ to a fixed vector $\boldsymbol{o}^{\tau}$.
(b) RNN model to mapping a fixed vector $\boldsymbol{x}$ to a sequence $(\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{n_y})$.

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016.

Encoder-Decoder Sequence-to-Sequence

Encoder or decoder ?



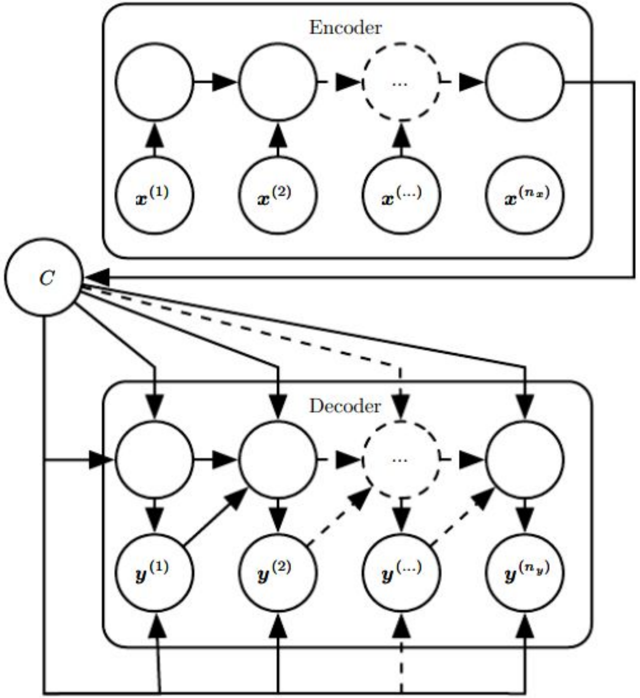
Encoder or decoder ?

Encoder-Decoder model for sequence-to-sequence

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016.

Encoder-Decoder Sequence-to-Sequence

$\boxed{C = \boldsymbol{h}^{(n_x)}}$



encoder

decoder

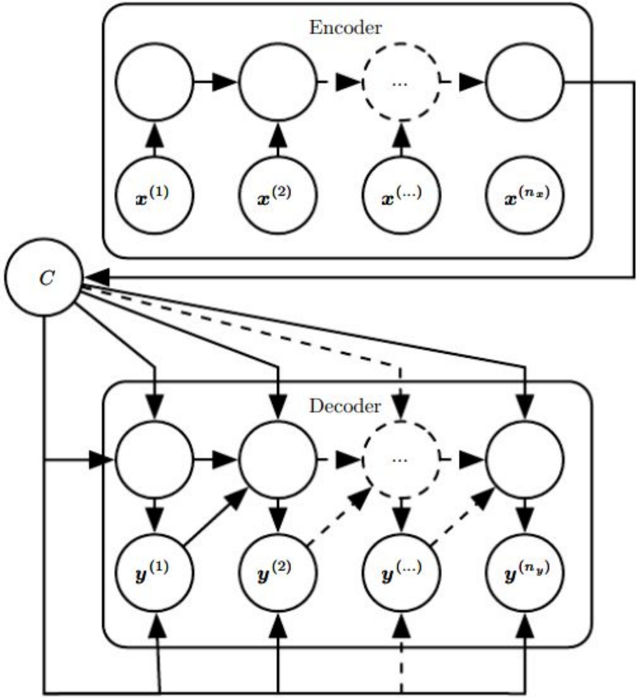
What probability distribution does the model learn?

Encoder-Decoder model for sequence-to-sequence

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016.

Encoder-Decoder Sequence-to-Sequence

$\boxed{C = \boldsymbol{h}^{(n_x)}}$



encoder

decoder

learning

$$\boxed{\prod_{t=1} p(\boldsymbol{y}_t \mid \boldsymbol{c}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1})}$$

Encoder-Decoder model for sequence-to-sequence

[ ] Deep Learning, Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016.

Encoder-Decoder Sequence-to-Sequence

encoder

$$\boldsymbol{h}_t = f(\boldsymbol{W}[e(\boldsymbol{x}_t), \boldsymbol{h}_{t-1}] + \boldsymbol{b}_e)$$
$$\vdots = \qquad \vdots$$
$$\boldsymbol{h}_{n_x} = f(\boldsymbol{W}[e(\boldsymbol{x}_{n_x}), \boldsymbol{h}_{n_x-1}] + \boldsymbol{b}_e)$$
$$\boldsymbol{c} = g(\boldsymbol{V}\boldsymbol{h}_{n_x})$$

decoder

$$\boldsymbol{h}_0' = \tanh(\boldsymbol{V}'\boldsymbol{c})$$
$$\boldsymbol{h}_{t'}' = f(\boldsymbol{W}'\boldsymbol{h}_{t'-1}' + \boldsymbol{U}e(\boldsymbol{y}_{t'-1}) + \boldsymbol{C}\boldsymbol{c} + \boldsymbol{b}_{dh})$$
$$\boldsymbol{s}_{t'} = o(\boldsymbol{O}_h\boldsymbol{h}_{t'}' + \boldsymbol{O}_y e(\boldsymbol{y}_{t'-1}) + \boldsymbol{O}_c\boldsymbol{c} + \boldsymbol{b}_{do})$$

[ ] Learning phrase representations using RNN encoder-decoder for statistical machine translation, Cho, 2014.

Encoder-Decoder Sequence-to-Sequence

Donde $\boldsymbol{g}_j$ es la j-fila de la matriz $\boldsymbol{G} \in \mathbb{R}^{k_t \times m}$, y m es la dimensión del token predecido.

$$p(\boldsymbol{y}_{t',j} = 1 | \boldsymbol{y}_{t'-1}, \cdots, \boldsymbol{y}_1, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{n_x}) = \frac{\exp(\boldsymbol{g}_j \boldsymbol{s}_{t'})}{\sum_{j'}^{k_t} \exp(\boldsymbol{g}_{j'} \boldsymbol{s}_{t'})} \tag{4.3}$$

Probability Distribution

$$p(\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{T'} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = \prod_{t=1}^{T'} p(\boldsymbol{y}_t \mid \boldsymbol{c}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1})$$

[ ] Learning phrase representations using RNN encoder-decoder for statistical machine translation, Cho, 2014.

Encoder-Decoder Sequence-to-Sequence

Problem with encoder-decoder model:

Is that it attempts to store information sentences of any arbitrary length in a hidden vector of fixed size .

- *Small sentences-big network*
- *Huge sentence- small network*

Encoder-Decoder Sequence-to-Sequence

Suggested Papers for sequence-to-sequence:

- Learning phrase representations using RNN encoder-decoder for statistical machine translation, Cho, 2014.
- Sequence to sequence learning with neural network, Sutskever, 2014.
- Recurrent continuous translation models, Kalchbrenner, 2013. (*encoder CNN*)

References

- Deep Learning, Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016.
- Learning phrase representations using RNN encoder-decoder for statistical machine translation, Cho, 2014.
- Neural Machine Translation and Sequence-to-sequence Models: A Tutorial, Graham Neubig, 2017.
- Parsing Natural Scenes and Natural Language with Recursive Neural Networks, Socher, 2011.
- Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, Socher, 2013.
- Learning Long-Term Dependencies with Gradient Descent is Difficult, Bengio, 1994.
- Deep Learning Specialization, Sequence Models, Andrew Ng, 2016.
- CS224N/Ling284, Lecture 8, Stanford University, Manning , 2016.
- Learning Long-Term Dependencies with Gradient Descent is Difficult, Bengio, 1994.
- Neural Machine Translation, Minh-Thang Luong, 2016.

Chapter 10 - Part I

Sequence Modeling: Recurrent and Recursive Nets

César Bragagnini

cesarbrma91@gmail.com
@MarchBragagnini

Universidad Católica San Pablo