

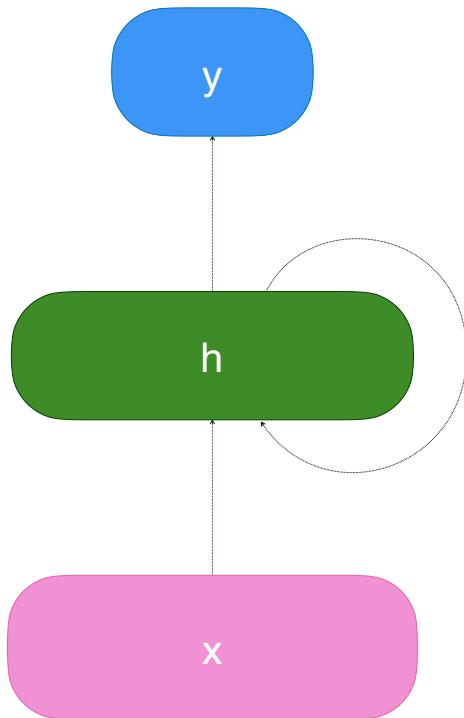
Redes Neuronales Recurrentes

Backpropagation en el Tiempo



Prof. Ricardo Ñanculef - Departamento de Informática UTFSM

Modelo Básico (Elman)



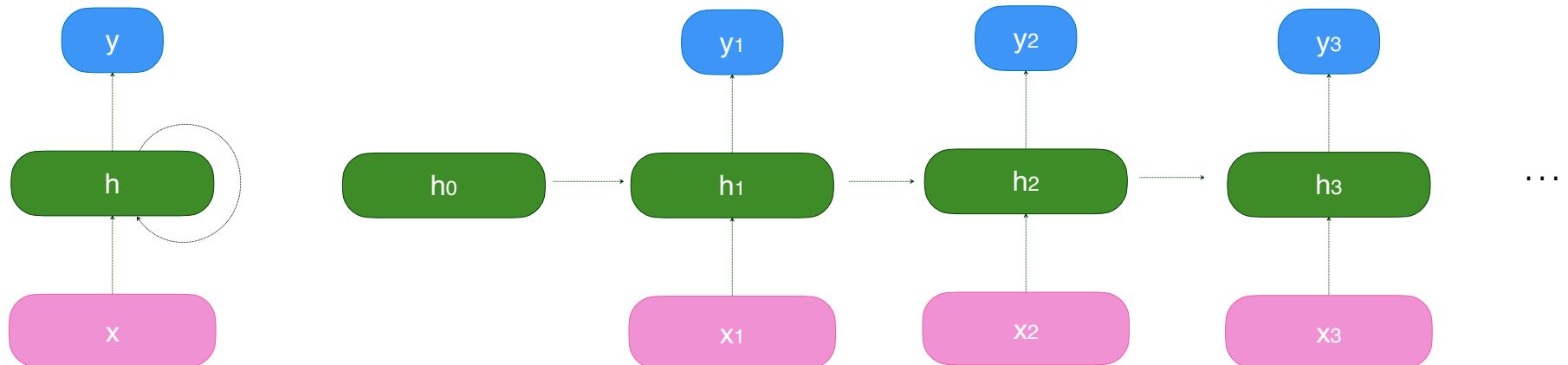
$$y_t = g_o(h_t)$$

$$h_t = g_h(x_t, h_{t-1})$$

- Más formalmente, la red toma como input **secuencias $x_1x_2x_3 \dots$** de entrada y produce como output **secuencias $y_1y_2y_3$** , procesando un elemento a la vez. Partiendo de un estado inicial h_0 , las salidas quedan definidas mediante ecuaciones de la forma:
- Cuando decimos “tiempo” nos referimos entonces al conjunto que indexa las secuencias de entrada y salida $t=1,2,3\dots$

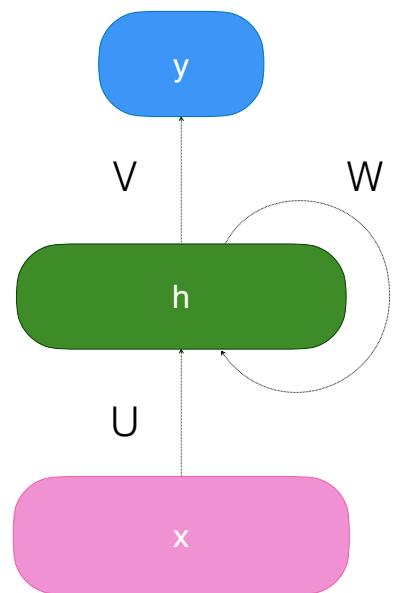
Desenrollado de la Red

- Un modo alternativo de representar la secuencia de cómputo ejecutada por la red consiste en “desenrollar” el grafo anterior (unfolding).



Modelo Básico (Elman)

- Si los elementos de la secuencia son vectores, podemos usar capas tradicionales (densas) de la forma: transformación lineal + no-linealidad elemento a elemento.



$$y_t = g_o(h_t) = \sigma_o(\mathbf{V}h_t + b_v)$$

$$h_t = g_h(x_t, h_{t-1}) = \sigma_h(\mathbf{U}x_t + \mathbf{W}h_{t-1} + b_h)$$

Aprendizaje en Redes Recurrentes

- El aprendizaje en RNNs procede de la misma forma que en redes FNN o CNN. Dada un conjunto de **n secuencias de entrenamiento, ambas de largo T**

$$S = \left\{ (x_t^{(i)}, y_t^{(i)})_{t=1}^T \right\}_{i=1}^n$$

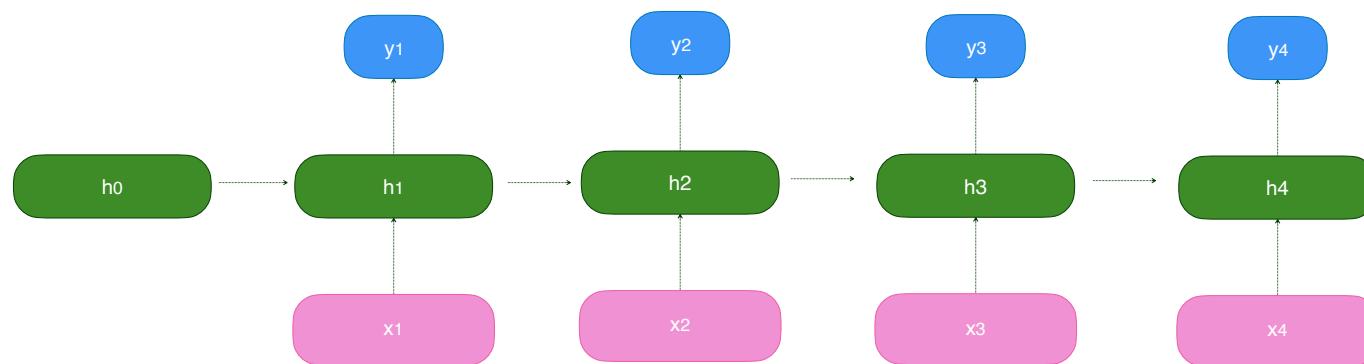
y una **función de costo E** que mide el error de la red en el tiempo t, nos interesa encontrar U,V y W para minimizar:

$$J(\Theta) = \sum_{i=1}^n \sum_{t=1}^T E \left(f \left(x_t^{(i)} \right), y_t^{(i)} \right)$$

- Como en redes anteriores, el método más utilizado en la práctica es back-propagation. La adaptación de este algoritmo al caso recurrente se suele denominar **back-propagation en el tiempo (BPTT)**.

Aprendizaje en Redes Recurrentes

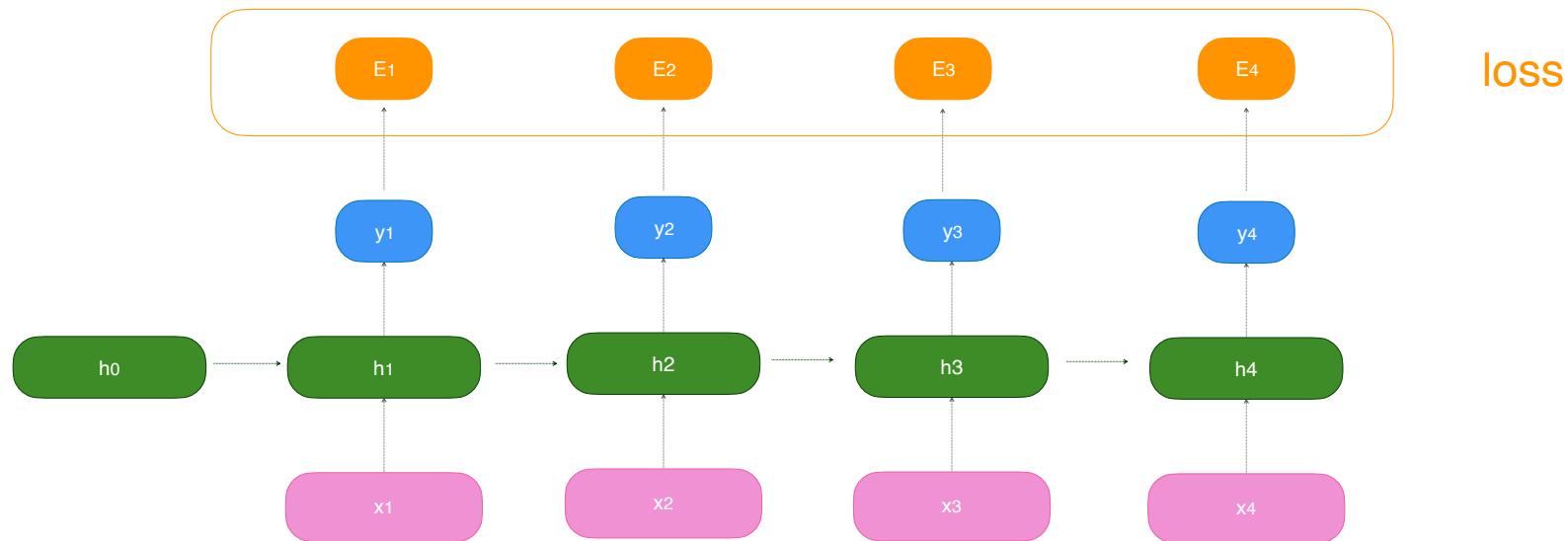
- Esencialmente, el método consiste en desenrollar la red y tratarla como si fuese una red profunda de T capas:



$$J(\Theta) = \sum_{i=1}^n \sum_{t=1}^T E \left(f \left(x_t^{(i)} \right), y_t^{(i)} \right)$$

BP en el Tiempo (BPTT)

- Una diferencia importante es que ahora cada capa emite una salida que es considerada en J .



$$J(\Theta) = \sum_{i=1}^n \sum_{t=1}^T E \left(f \left(x_t^{(i)} \right), y_t^{(i)} \right)$$

Aprendizaje en Redes Recurrentes

- Como la función es aditiva en t y en los ejemplos, las derivadas con respecto a un parámetro θ toman la forma

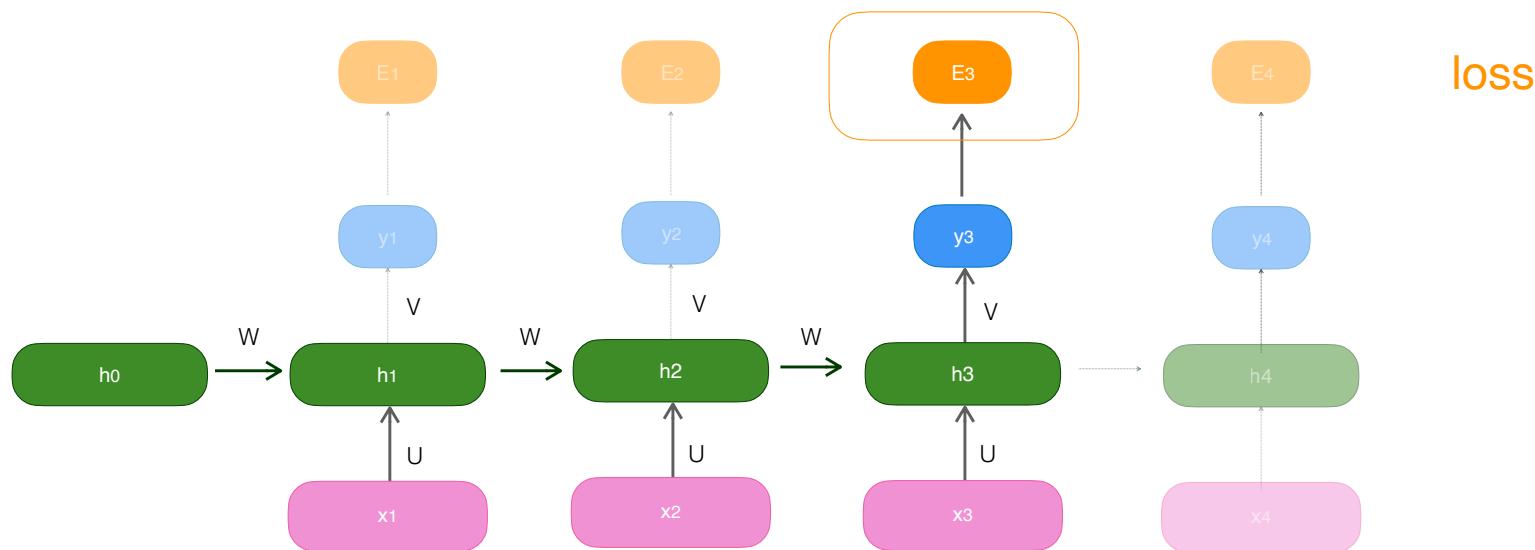
$$J(\Theta) = \sum_{i=1}^n \sum_{t=1}^T E \left(f \left(x_t^{(i)} \right), y_t^{(i)} \right)$$

$$\frac{\partial J}{\partial \theta} = \sum_i \sum_t \frac{\partial E_t^{(i)}}{\partial \theta} \quad \text{con} \quad E_t^{(i)} = E \left(f \left(x_t^{(i)} \right), y_t^{(i)} \right)$$

- Nos podemos concentrar en un tiempo específico
- Por simplicidad notacional omitiremos el super-índice i cuando no cause confusión.

BP en el Tiempo (BPTT)

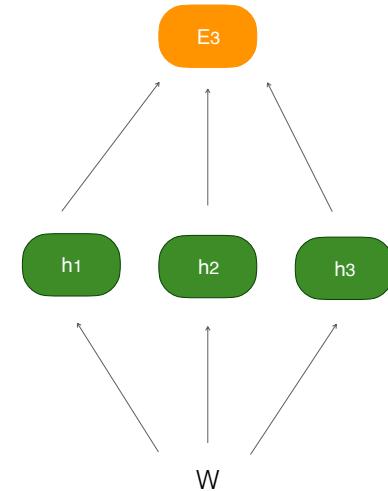
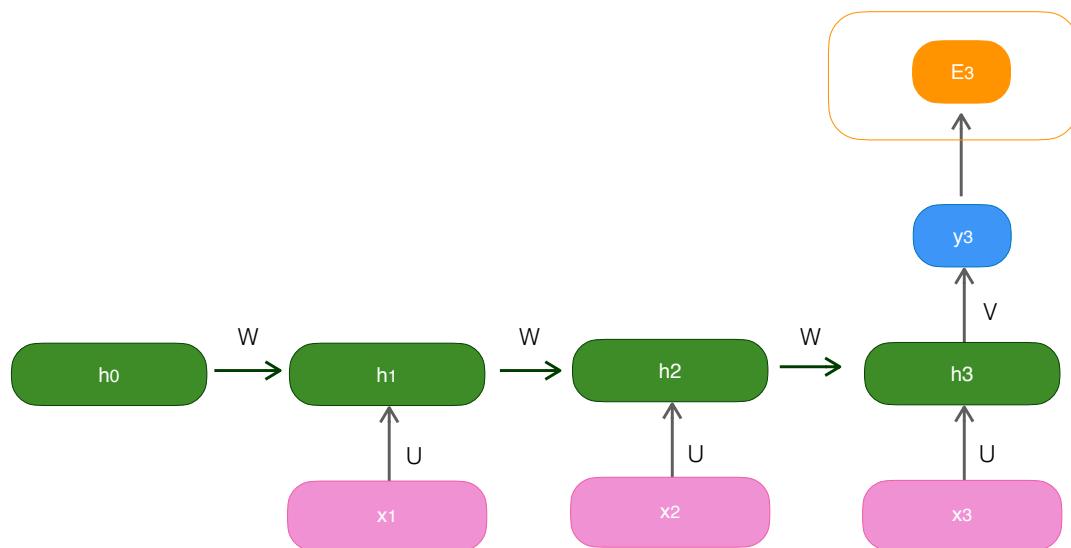
- Por construcción, la salida en el tiempo t depende del pasado.
- Como los pesos son compartidos, la influencia de un parámetro (como W o U) se produce a través de múltiples puntos.



BP en el Tiempo (BPTT)

- Por la regla de la cadena, las derivadas corresponderán a la suma de estas derivadas parciales.

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_s} \frac{\partial h_s}{\partial W}$$



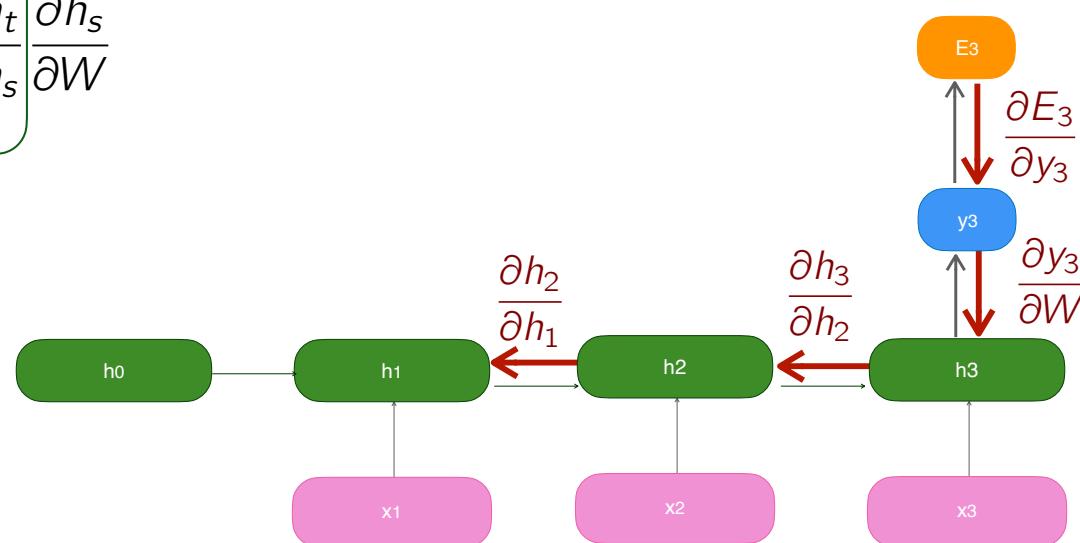
BP en el Tiempo (BPTT)

- Desarrollando obtenemos, para la **matriz de recurrencias W**

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_s} \frac{\partial h_s}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_s} \frac{\partial h_s}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\frac{\partial h_t}{\partial h_s} \right) \frac{\partial h_s}{\partial W}$$



BP en el Tiempo (BPTT)

- Desarrollando obtenemos, para la **matriz de recurrencias W**

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_s} \frac{\partial h_s}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_s} \frac{\partial h_s}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\frac{\partial h_t}{\partial h_s} \right) \frac{\partial h_s}{\partial W}$$

Jacobiano estado-estado

determina la influencia del tiempo s en el error obtenido al tiempo t

- Invocando las definiciones, obtenemos fácilmente cada uno de los términos en la expresión anterior.

$$y_t = g_o(h_t) = \sigma_o(\mathbf{V}h_t + b_v)$$

$$h_t = g_h(x_t, h_{t-1}) = \sigma_h(\mathbf{U}x_t + \mathbf{W}h_{t-1} + b_h)$$



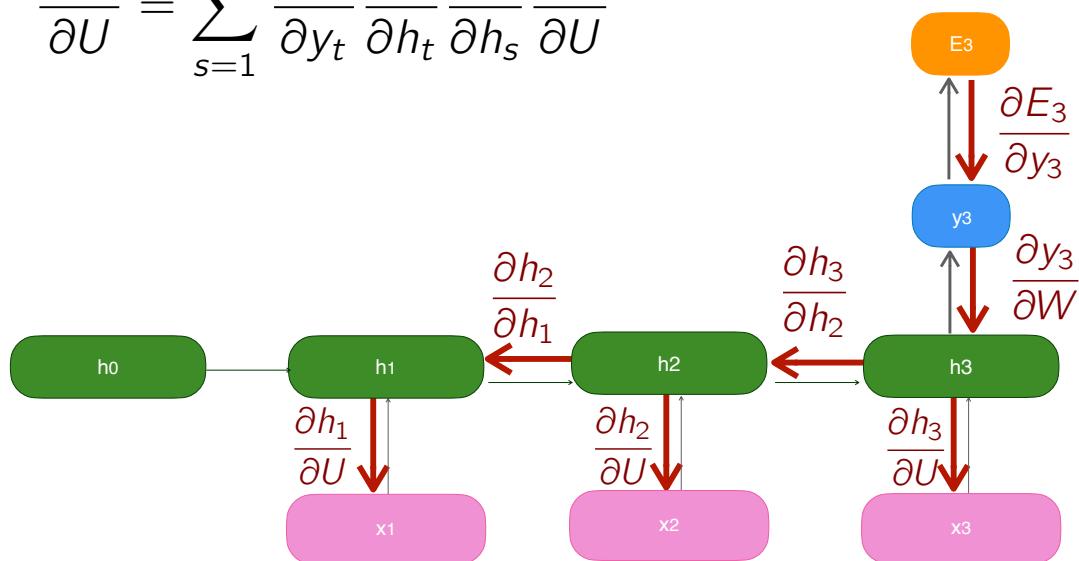
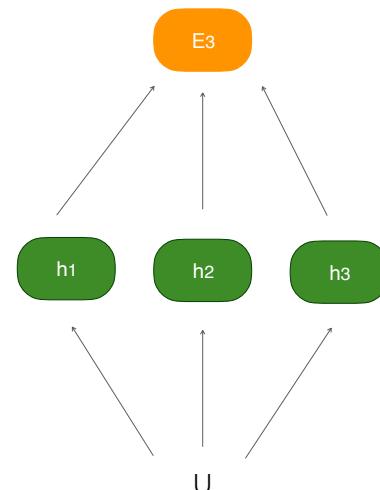
BP en el Tiempo (BPTT)

- Análogamente, para la matriz de entrada U obtenemos

$$\frac{\partial E_t}{\partial U} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_s} \frac{\partial h_s}{\partial U}$$

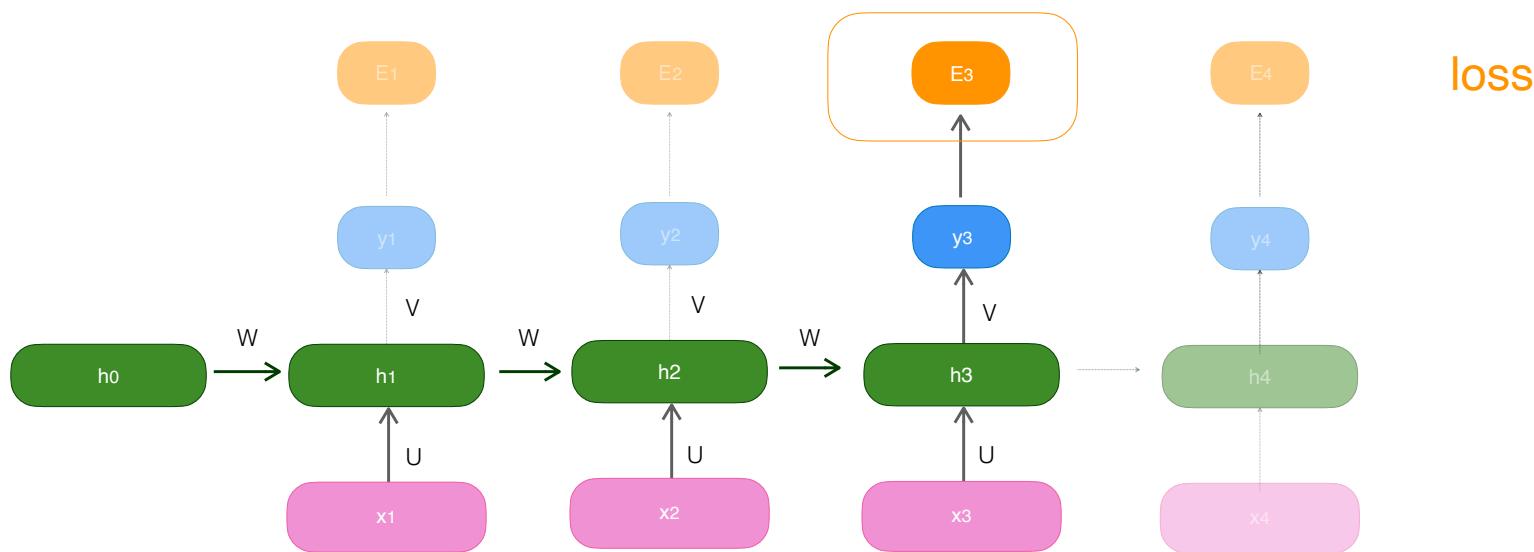
$$\frac{\partial E_t}{\partial U} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_s} \frac{\partial h_s}{\partial U}$$

$$\frac{\partial E_t}{\partial U} = \sum_{s=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_s} \frac{\partial h_s}{\partial U}$$



BP en el Tiempo (BPTT)

- La matriz de salida V en cambio, influye en E_t sólo a través de y_t



Dependencias de largo plazo

- Consideremos el siguiente problema. Dadas secuencias de largo $p=100$ de un alfabeto de tamaño $p+1$ queremos entrenar una red neuronal que aprenda a memorizar el primer número y lo devuelva al final.

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
<http://www7.informatik.tu-muenchen.de/~hochreit>

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
<http://www.idsia.ch/~juergen>

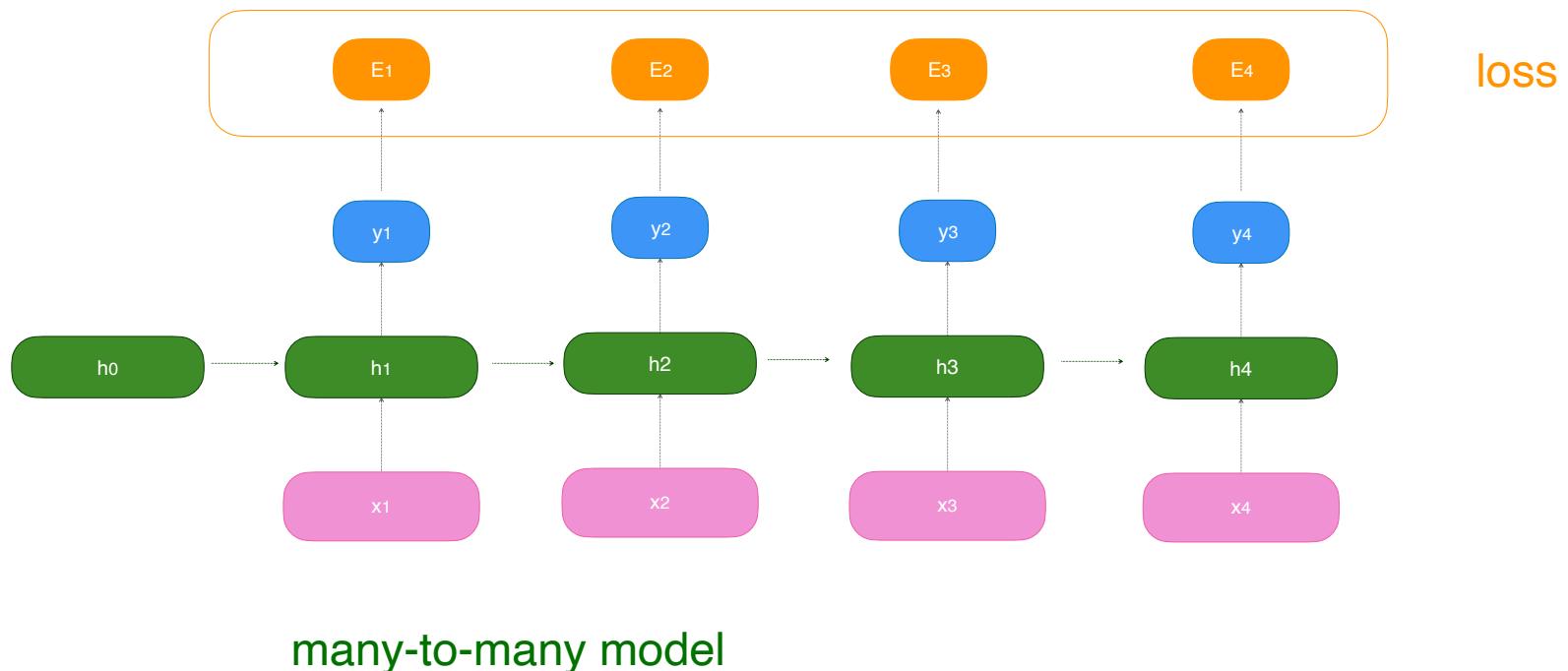
Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.



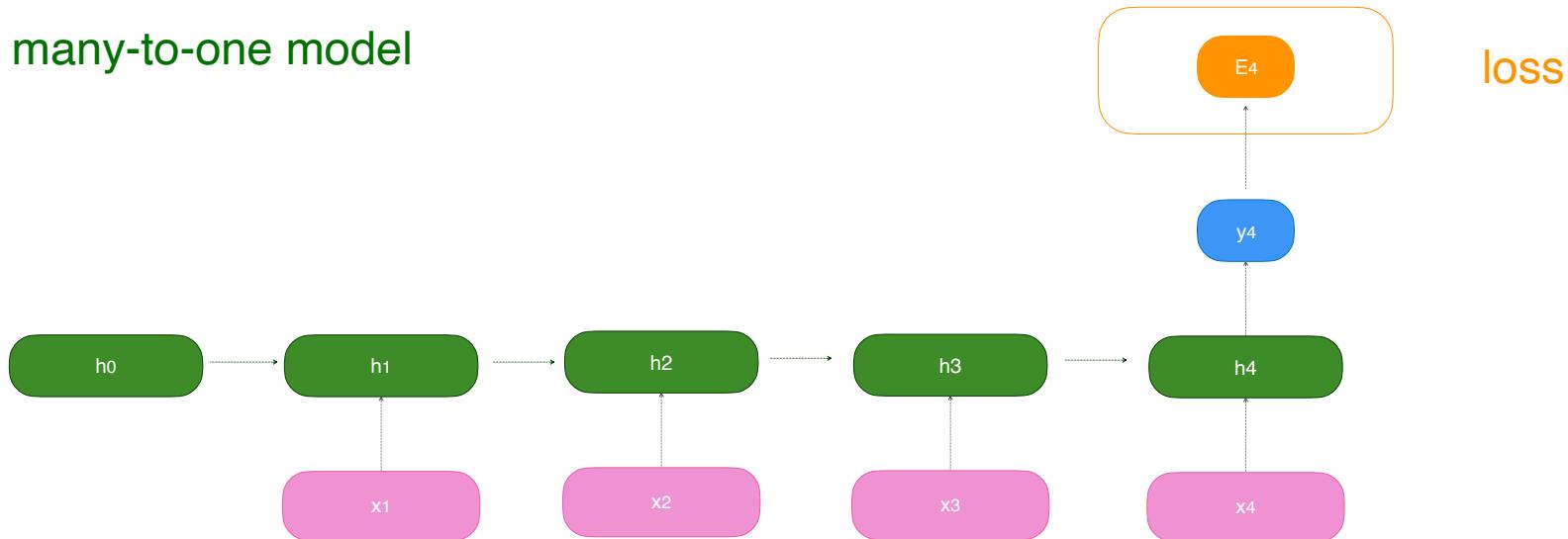
Dependencias de largo plazo

- En teoría bastaría preparar secuencias de entrenamiento con la respuesta deseada y hacer una pequeña modificación a nuestra arquitectura anterior.



Dependencias de largo plazo

many-to-one model



- Este es un ejemplo de problema many-to-one en que nos interesa producir un único output para toda la secuencia (otro ejemplo sería la clasificación de frases en una determinada clase de sentimiento).

Keras Implementation

The screenshot shows the Keras API reference page for the SimpleRNN layer. The left sidebar has a red 'Keras' logo and navigation links: About Keras, Getting started, Developer guides, Keras API reference (which is selected), Models API, Layers API (which is highlighted in red), Callbacks API, Data preprocessing, Optimizers, Metrics, Losses, Built-in small datasets, Keras Applications, Utilities, Code examples, Why choose Keras?, Community & governance, and Contributing to Keras. The main content area has a search bar and a breadcrumb trail: » Keras API reference / Layers API / Recurrent layers / SimpleRNN layer. The title is 'SimpleRNN layer'. Below it is a section titled 'SimpleRNN class' with a code snippet:

```
tf.keras.layers.SimpleRNN(  
    units,  
    activation="tanh",  
    use_bias=True,  
    kernel_initializer="glorot_uniform",  
    recurrent_initializer="orthogonal",  
    bias_initializer="zeros",  
    kernel_regularizer=None,  
    recurrent_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    recurrent_constraint=None,  
    bias_constraint=None,  
    dropout=0.0,  
    recurrent_dropout=0.0,  
    return_sequences=False,  
    return_state=False,  
    go_backwards=False,  
    stateful=False,  
    unroll=False,  
    **kwargs  
)
```

Fully-connected RNN where the output is to be fed back to input.



K Keras

```
[7] from keras.models import Sequential
    from keras.layers import Dense, Activation, TimeDistributed
    from keras.layers.recurrent import GRU, SimpleRNN, LSTM
    import numpy as np

    InputDim = 15
    MaxSeqLen = 64
    HiddenSize = 16
    OutputSize = 8
    n_samples = 1000
    model = Sequential()
    model.add(SimpleRNN(HiddenSize, return_sequences=False, input_shape=(MaxSeqLen, InputDim)))
    model.add(Dense(OutputSize))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

    model.summary()
```

↳ Model: "sequential_7"

Layer (type)	Output Shape	Param #
=====		
simple_rnn_6 (SimpleRNN)	(None, 16)	512
=====		
dense_6 (Dense)	(None, 8)	136
=====		
activation_5 (Activation)	(None, 8)	0
=====		
Total params:	648	
Trainable params:	648	
Non-trainable params:	0	



K Keras

```
▶ from keras.models import Sequential
  from keras.layers import Dense, Activation, TimeDistributed
  from keras.layers.recurrent import GRU, SimpleRNN, LSTM
  import numpy as np

  InputDim = 15
  MaxSeqLen = 64
  HiddenSize = 16
  OutputSize = 8
  n_samples = 1000
  model = Sequential()
  model.add(SimpleRNN(HiddenSize, return_sequences=True, input_shape=(MaxSeqLen, InputDim)))
  #model.add(TimeDistributed(Dense(OutputSize)))
  model.add(Dense(OutputSize))
  model.add(Activation('softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

  model.summary()
```

```
⇨ Model: "sequential_5"

Layer (type)          Output Shape         Param #
=====
simple_rnn_4 (SimpleRNN)    (None, 64, 16)      512
time_distributed_3 (TimeDist (None, 64, 8)       136
activation_4 (Activation)   (None, 64, 8)        0
=====
Total params: 648
Trainable params: 648
Non-trainable params: 0
```



Dependencias de largo plazo

- La red que hemos definido es un aproximador universal en el sentido de que cualquier función computable por una máquina de Turing puede ser realizada/implementada por una RNN.

On The Computational Power Of Neural Nets

Hava T. Siegelmann
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
siegelma@paul.rutgers.edu

Eduardo D. Sontag
Department of Mathematics
Rutgers University
New Brunswick, NJ 08903
sontag@hilbert.rutgers.edu

Abstract

This paper deals with finite networks which consist of interconnections of synchronously evolving processors. Each processor updates its state by applying a “sigmoidal” scalar non-linearity to a linear combination of the previous states of all units. We prove that one may simulate all Turing Machines by rational nets. In particular, one can do this in linear time, and there is a net made up of about 1,000 processors which computes a universal partial-recursive function. Products (high order nets) are not required, contrary to what had been stated in the literature. Furthermore, we assert a similar theorem about non-deterministic Turing Machines. Consequences for undecidability and complexity issues about nets are discussed too.

but other nonlinearities are of course of interest as well. One of the processors is singled out as the “output node” of the net. Another processor, called the “validation node,” signals when the output occurs.

The use of sigmoidal functions —as opposed to hard thresholds— is what distinguishes this area from older work that dealt only with finite automata. Indeed, it has long been known, at least since the classical papers by McCulloch and Pitts ([12], [9]), how to implement logic gates by threshold networks, and therefore how to simulate finite automata by such nets. For us, however, nets are essentially *analog* computational devices, in accordance with models currently used in neural net practice.

In [14], Pollack argued that a certain recurrent net model, which he called a “neuring machine,” is universal. The model in [14] consisted of a finite number of neurons of two different kinds, having identity and threshold responses, respectively. The machine was *high-order*, that is, the activations were combined using



Siegelmann, Hava T., and Eduardo D. Sontag. "On the computational power of neural nets." *Proceedings of the fifth annual workshop on Computational learning theory*. 1992.



Dependencias de largo plazo

- Como el problema es muy sencillo esperaríamos que una RNN aprenda rápidamente el patrón contenido en las secuencias de ejemplo.
- Sin embargo, como mostró Hochreiter en 1997, nuestra RNN es incapaz de aprender esta tarea cuando T es modestamente grande (mayor a 15 o 20).

Method	Delay p	Learning rate	# weights	% Successful trials	Success after
RTRL	4	1.0	36	78	1,043,000
RTRL	4	4.0	36	56	892,000
RTRL	4	10.0	36	22	254,000
RTRL	10	1.0-10.0	144	0	> 5,000,000
RTRL	100	1.0-10.0	10404	0	> 5,000,000
BPTT	100	1.0-10.0	10404	0	> 5,000,000
CH	100	1.0	10506	33	32,400
LSTM	100	1.0	10504	100	5,040



Gradientes desvanecientes (y explosivos)

- Recordemos la derivada para la **matriz de recurrencias W**

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_s} \frac{\partial h_s}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_s} \frac{\partial h_s}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{s=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\frac{\partial h_t}{\partial h_s} \right) \frac{\partial h_s}{\partial W}$$

Jacobiano estado-estado

determina la influencia del tiempo s en el error obtenido al tiempo t

- Usando nuevamente la regla de la cadena obtenemos

$$\frac{\partial h_t}{\partial h_s} = \prod_{k=s+1}^t \frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{s+1}}{\partial h_s}$$



Gradientes desvanecientes (y explosivos)

- Por lo tanto la contribución del tiempo t a la señal que permite actualizar W (gradiente) queda acotada de la siguiente forma

$$\left\| \frac{\partial h_t}{\partial h_s} \right\| = \left\| \prod_{k=s+1}^t \frac{\partial h_t}{\partial h_k} \right\|$$

- Si asumimos que el Jacobiano estado-estado no cambia en el tiempo (por ejemplo nuestra red es lineal) obtenemos

$$\left\| \frac{\partial h_t}{\partial h_s} \right\| = \|J\|^t$$

- Si la norma de J es menor a 1, los gradientes se desvanecen, mientras si esta norma es mayor a 1 explotan.

Gradientes desvanecientes (y explosivos)

- Por lo tanto la contribución del tiempo t a la señal que permite actualizar W (gradiente) queda acotada de la siguiente forma

$$\left\| \frac{\partial h_t}{\partial h_s} \right\| = \left\| \prod_{k=s+1}^t \frac{\partial h_t}{\partial h_k} \right\|$$

- Si asumimos que el Jacobiano estado-estado no cambia en el tiempo (por ejemplo nuestra red es lineal) obtenemos

$$\left\| \frac{\partial h_t}{\partial h_s} \right\| = \|J\|^{t-s}$$

- Si la norma de J es mucho menor a 1, los gradientes se desvanecen, mientras si esta norma es mucho mayor a 1 explotan.
- Si la norma de J es 1, los gradientes se propagan sin problemas, pero esto ocurre de igual manera para todos los tiempos entre t y s (memoria no selectiva).



Gradientes desvanecientes (y explosivos)

On the difficulty of training recurrent neural networks

Razvan Pascanu

Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

PASCANUR@IRO.UMONTREAL.CA

Tomas Mikolov

Speech@FIT, Brno University of Technology, Brno, Czech Republic

T.MIKOLOV@GMAIL.COM



En el caso de redes no lineales ...

$$\frac{\partial h_t}{\partial h_s} = \prod_{k=s+1}^t \frac{\partial h_k}{\partial h_{k-1}} \prod_{k=s+1}^t W^T \text{diag}(g'(\cdot))$$

$$\left\| \frac{\partial h_k}{\partial h_{k-1}} \right\| \leq \|W\| \|\text{diag}(g'(\cdot))\|$$

$$\begin{aligned} \text{tanh: } & \gamma = 1 \\ \text{sigmoid: } & \gamma = 1/4 \end{aligned}$$

Si asumimos que $\|\text{diag}(g'(\cdot))\| \leq \gamma$

$\|W\| < 1/\gamma$ sufficient for vanishing gradients

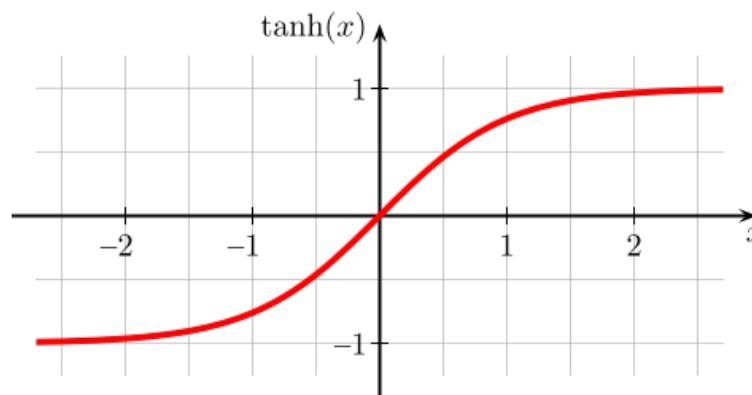
$\|W\| > 1/\gamma$ necessary for exploding gradients

Gradientes desvanecientes (y explosivos)

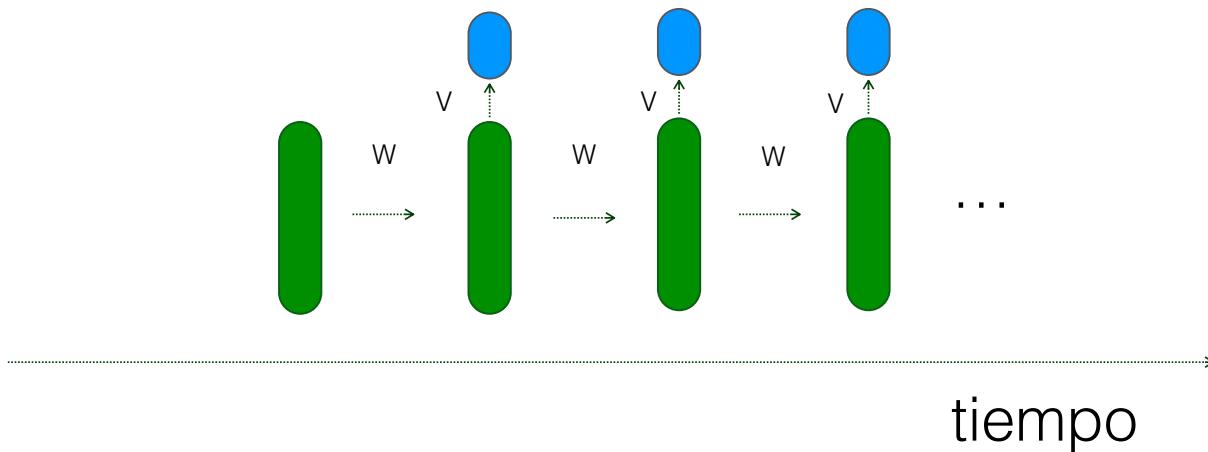
- La “explosión” de las señales que fluyen por la red puede manifestarse también en el forward pass. En efecto, para redes lineales tenemos que

$$\|h_t\| \propto \|J\|^t \|h_0\|$$

- El uso de no-linealidades tipo squashing evita este problema en el forward pass.



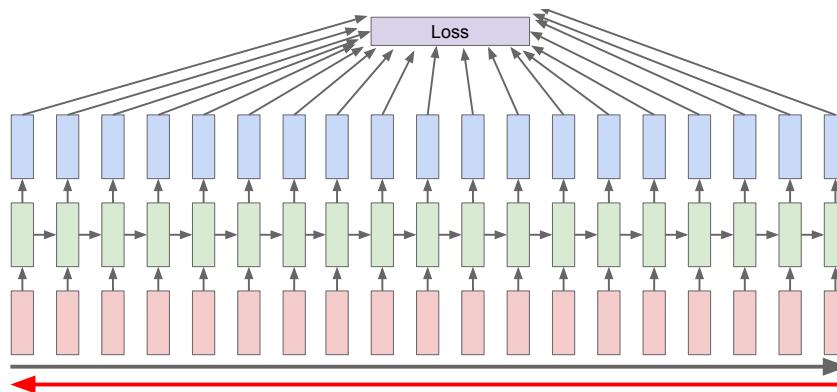
Gradientes desvanecientes (y explosivos)



Soluciones:

- BTTP truncado
- Leaky units (Error Carrusel)
- Echo State Networks
- LSTM/GRU

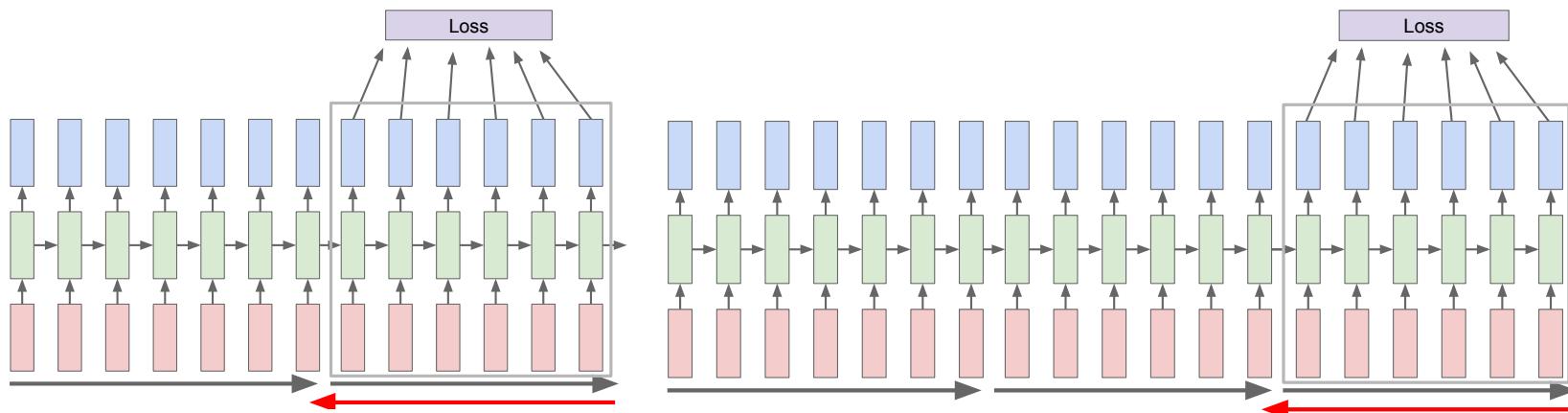
BPTT Truncado



- En el BPTT estándar, la propagación de las señales de error se produce después de procesar una secuencia completa.

BPTT Truncado

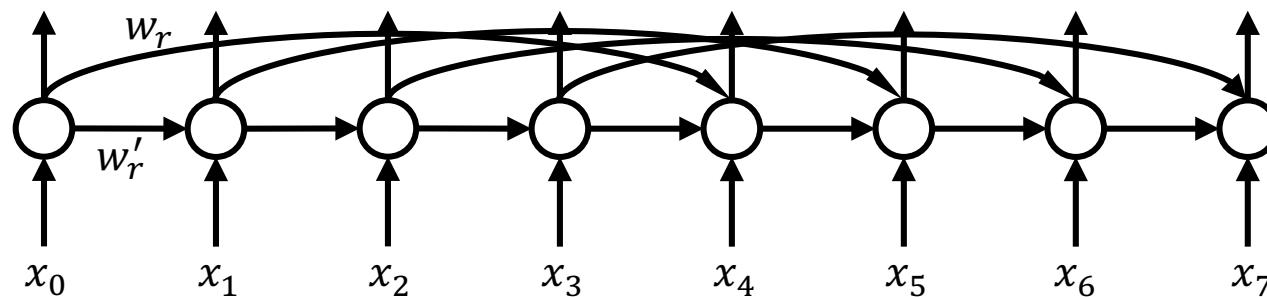
- El BPTT truncado consiste en “adelantar” el backward pass de modo que las dependencias por aprender sean de más corto plazo.



- Más que una “solución” al problema de aprender dependencias largas/distantes en el tiempo, esta técnica consiste en evitarlas.

Múltiples Escalas Temporales

- En una red recurrente estándar, las conexiones recurrentes van desde el tiempo t al tiempo $t+1$. La escala temporal está fija.
- Una idea común a muchos métodos diseñados para aprender dependencias largas en el tiempo, consiste en operar sobre **múltiples escalas temporales simultáneamente**.
- La forma más simple de obtener esto es introducir conexiones denominadas “skip connections” es decir recurrencias directas desde el tiempo t al tiempo s con $s > t+1$.



Múltiples Escalas Temporales

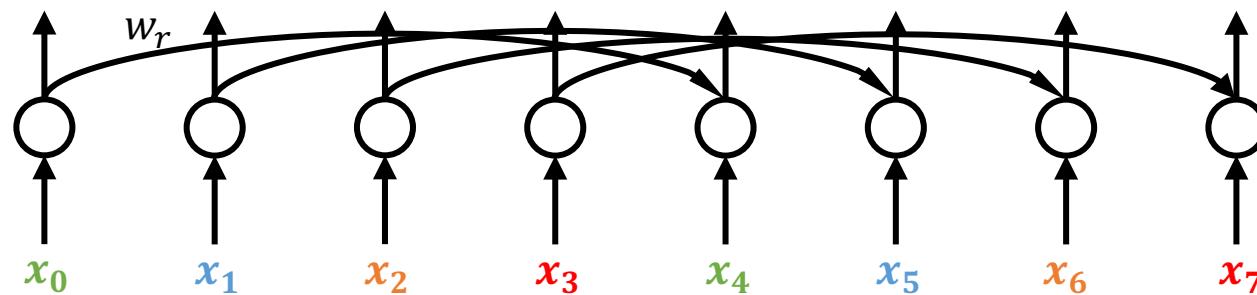
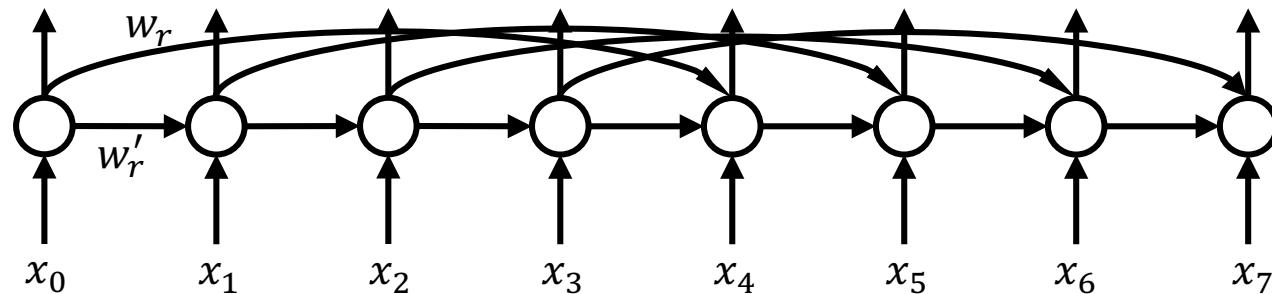
- Otro método históricamente relevante consiste en introducir unidades con recurrencias lineales denominadas “leaky units”

$$h_t = (1 - \alpha)h_{t-1} + \alpha\sigma_h(Ux_t)$$

- Estas unidades mantienen una media móvil parametrizada de las activaciones pasadas, de modo que pueden “olvidar” más lentamente (α cercano a 0) o más agresivamente (α cercano a 1) el pasado según sea necesario.
- Es posible verlas como una implementación continua de la idea de las “skip connections”.

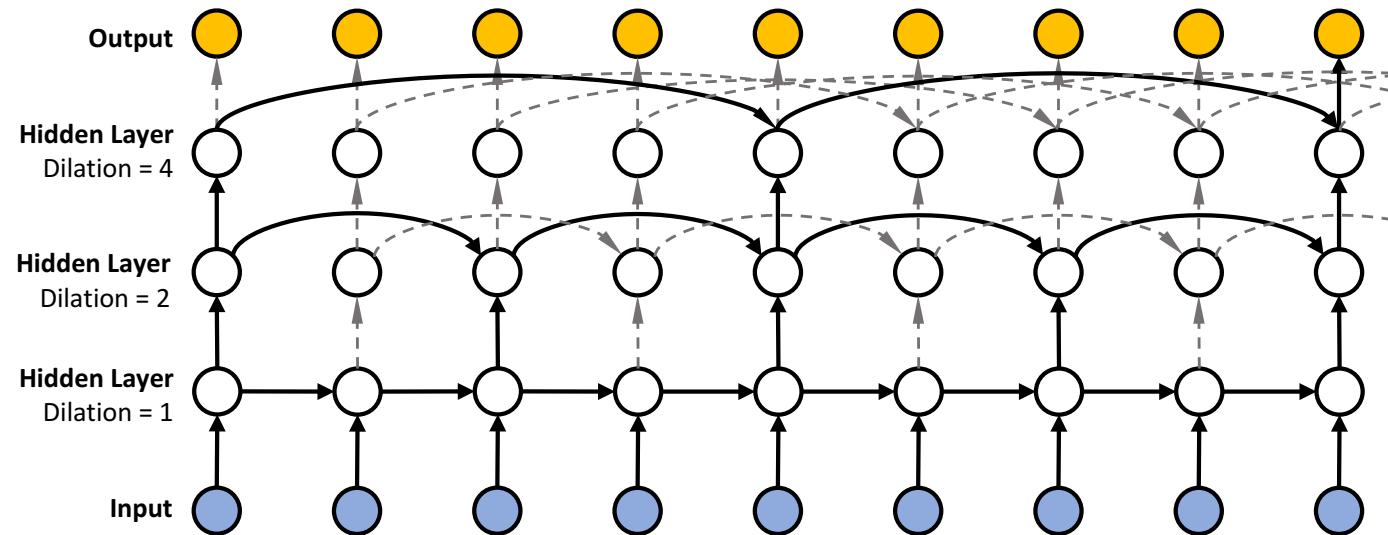
Múltiples Escalas Temporales

- Otra forma de manejar múltiples escalas temporales en la red consiste en suprimir a la vez que se agregan nuevas.



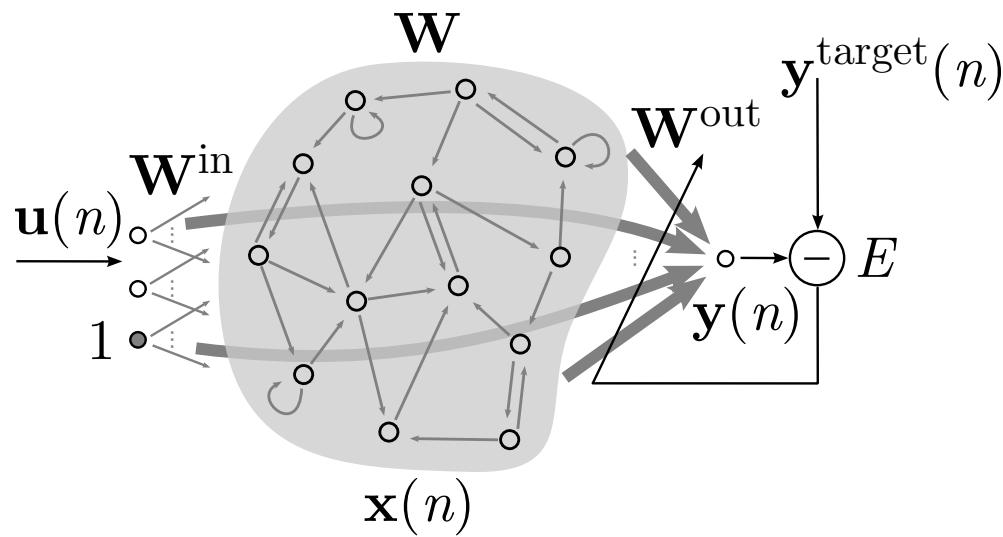
Múltiples Escalas Temporales

- Esta idea se ha mostrado efectiva en publicaciones recientes cuando se implementa superponiendo capas con distintos grados de “dilatación temporal”



Echo State Networks

- Una idea extrema para evitar el problema de los gradientes desvanecientes consiste en hacer no-entrenables los pesos de la red, excepto los de salida.

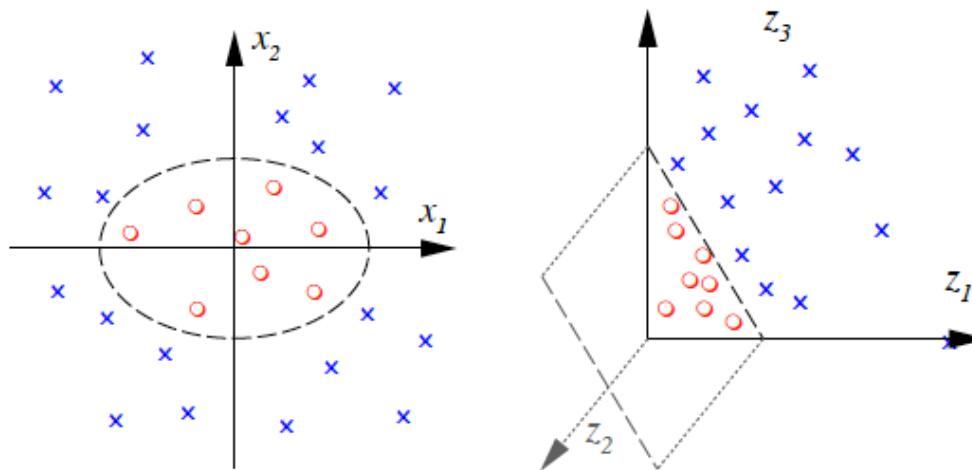


- La explosión en el otro sentido (forward pass) se maneja usando matrices de pesos con normas controladas (1 en la versión tradicional, pero mayores en la actualidad) y funciones de activación tipo squashing.

Echo State Networks

- La red actúa como un gigantesco extractor de características que la capa de salida aprende a combinar apropiadamente.

$$\Phi : R^2 \rightarrow R^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



Entonces ...

- Una red de Elman permite abordar fácilmente problemas donde el input es una secuencia y se desea obtener “etiquetar” cada elemento de la secuencia dando como resultado otra secuencia (modelo many-to-many). También es muy simple abordar problemas donde se desea predecir un elemento asociado a toda la secuencia (modelo many-to-one).
- En ambos casos, es posible entrenar la red desenrollándola en el tiempo y aplicando back-propagation estándar. Producto de los pesos compartidos, los gradientes correspondientes a las capas recurrentes y de entrada, resultarán ser una suma de gradientes en el tiempo. Este método se denomina BPTT.
- En la práctica, este método no funciona muy bien para secuencias largas. La razón es, nuevamente, la inestabilidad de los gradientes que se propagan en el backward pass.
- Soluciones históricamente relevantes para abordar el problema consisten en el uso BPTT truncado, skip connections, leaky units, múltiples escalas temporales y echo state networks.

