An introduction to Recurrent Neural Networks

Felipe Salvatore
https://felipessalvatore.github.io/

Thiago Bueno
http://thiagopbueno.github.io/

October 9, 2017

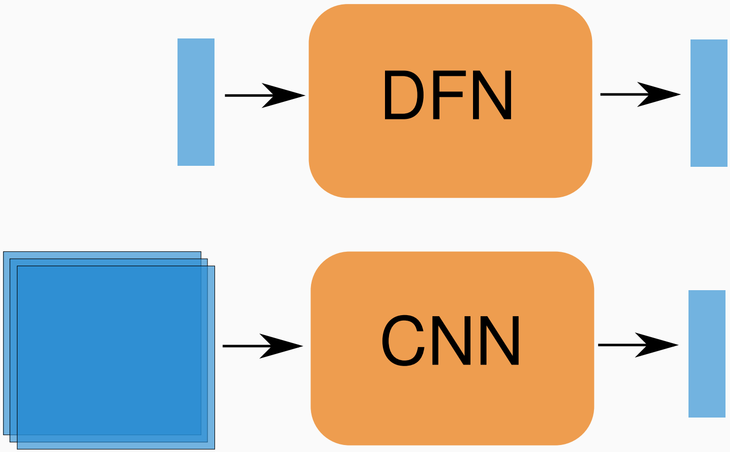
IME-USP: Institute of Mathematics and Statistics, University of São Paulo

Introduction

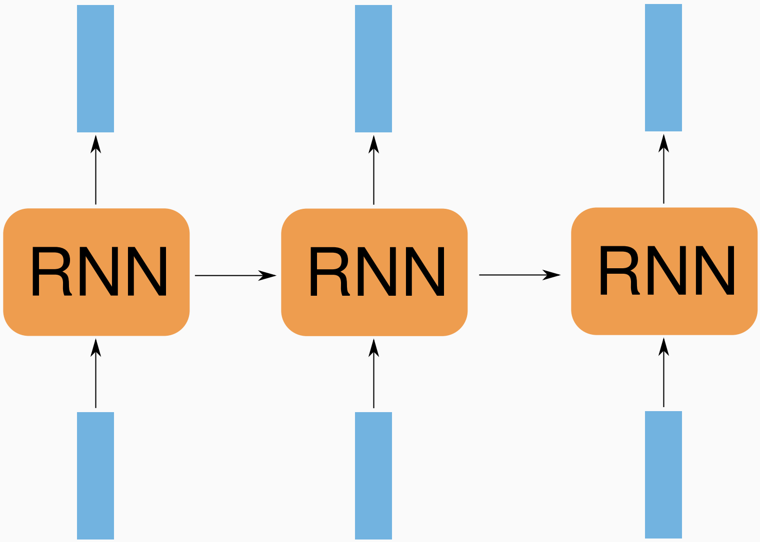
Basic idea

- A Recurrent Neural Network (RNN) allows us to operate over **sequences** of vectors: either sequences in the **input** or the **output**
- This feature differentiate the RNN model from other deep learning architectures such as **Deep Feedforward Network (DFN)** and **Convolutional Neural Network (CNN)**.

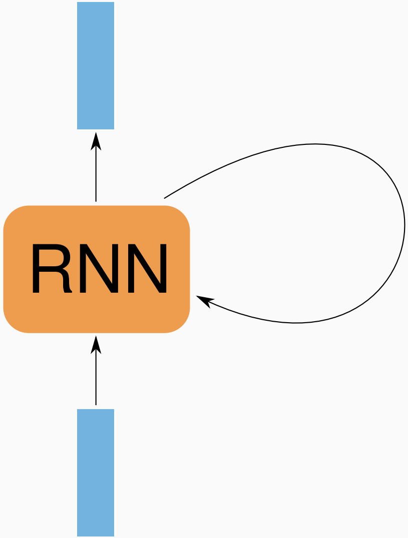
DFN and CNN



RNN: unfold representation

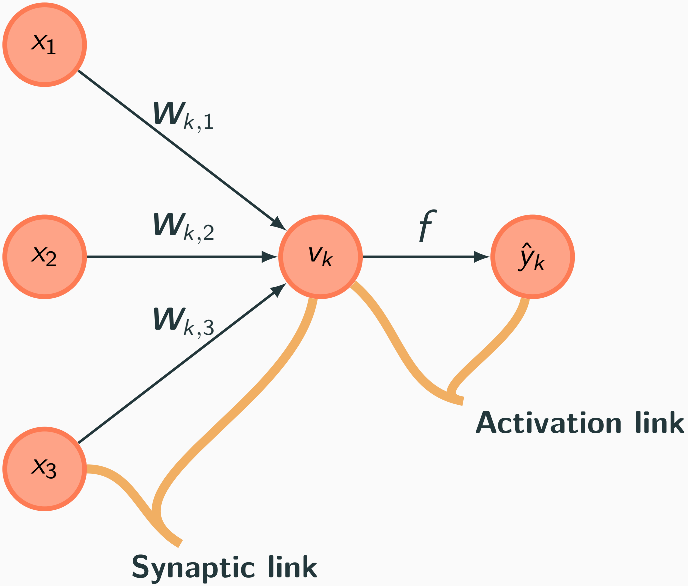


RNN: cyclic representation

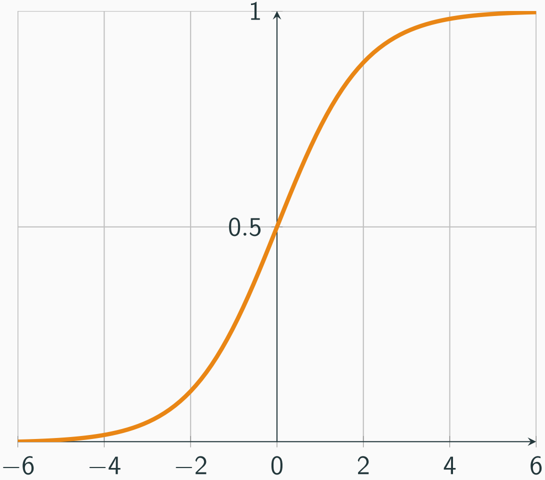


Graph Representation

NN as a directed graph: old version



Recap: sigmoid function



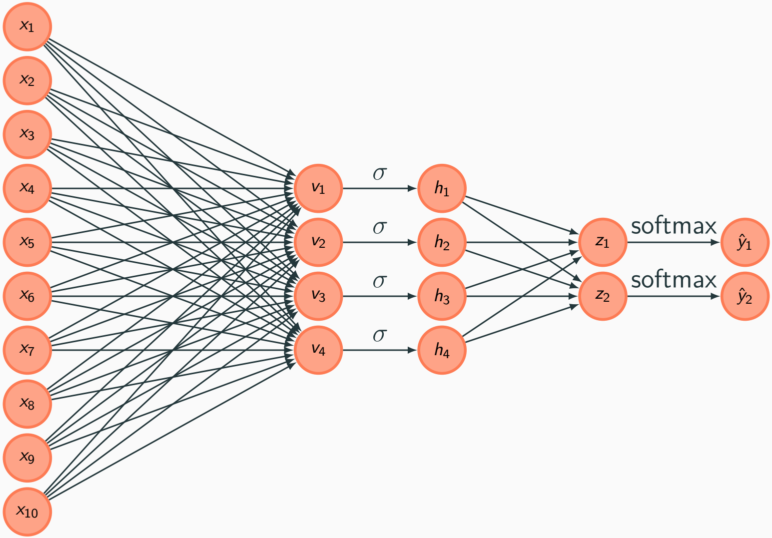
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Recap: softmax function

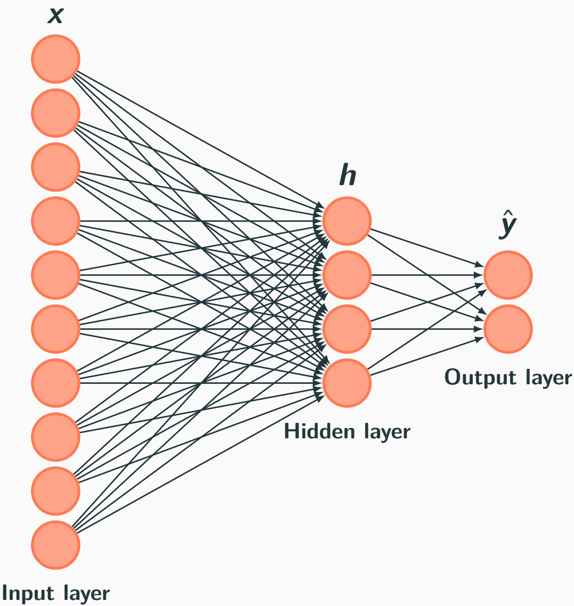
$$\begin{bmatrix} 3.82 \\ 5.35 \\ 1.44 \\ -1.26 \\ 2.71 \\ 1.98 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 0.16115195 \\ 0.74422819 \\ 0.01491471 \\ 0.00100235 \\ 0.05310907 \\ 0.02559374 \end{bmatrix}$$

$$softmax(x) = \frac{e^x}{\sum e^x}$$

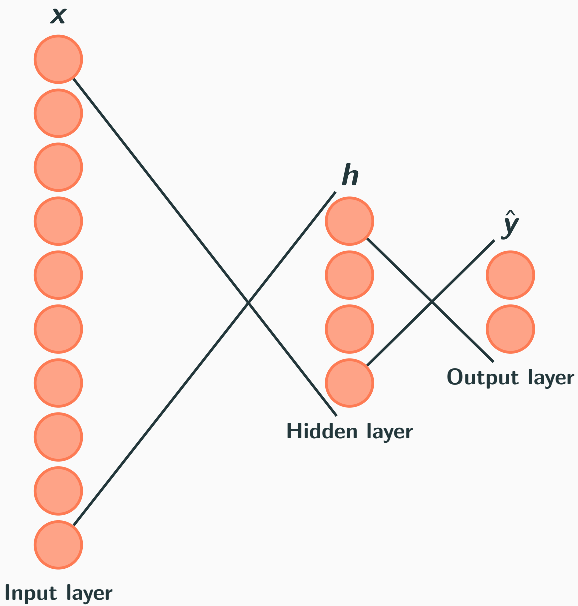
NN as a directed graph: old version



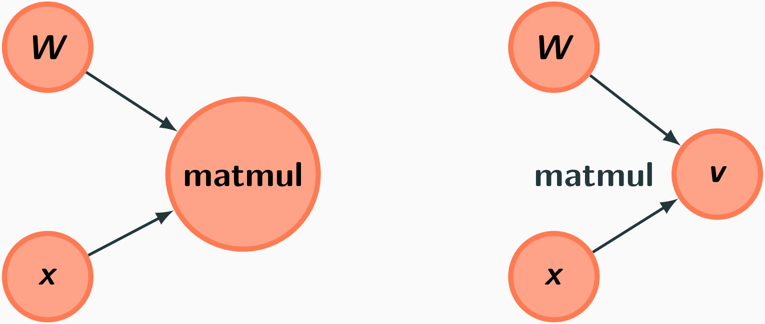
NN as a directed graph: old version



NN as a directed graph: old version

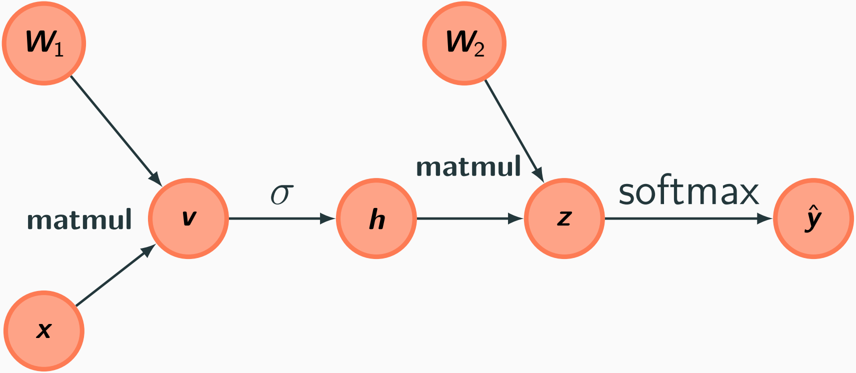


Computational Graphs



$$\boldsymbol{v} = \boldsymbol{W}\boldsymbol{x}$$

Computational Graphs



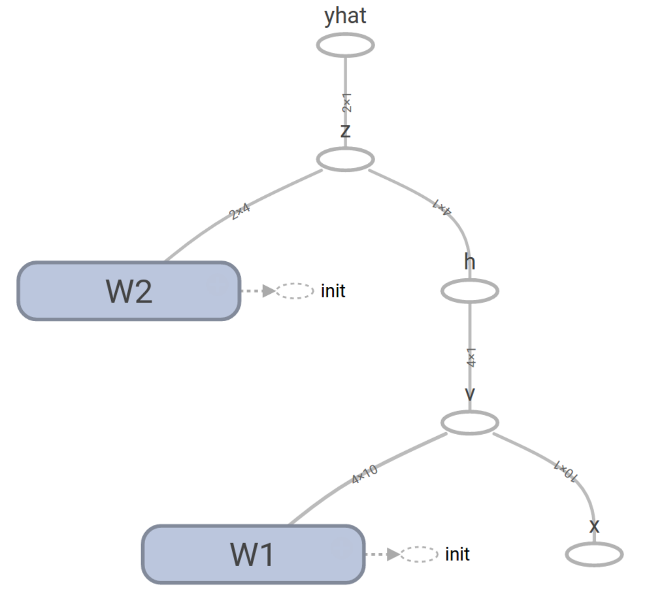
Tensorflow graph

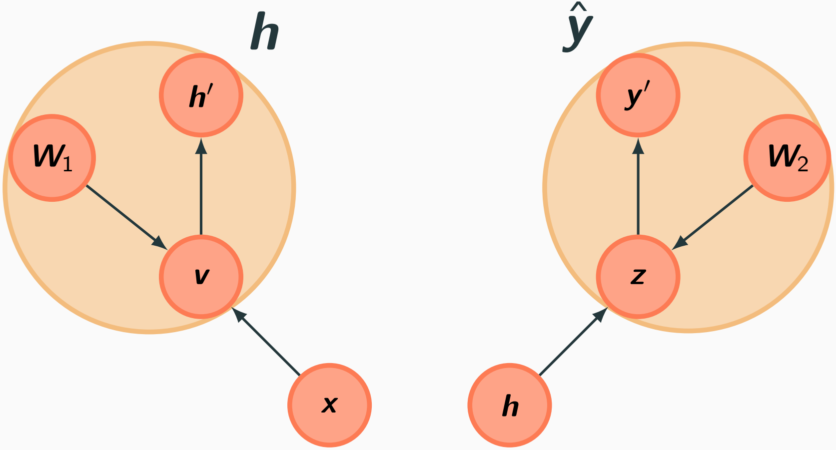
```
import tensorflow as tf
import numpy as np

input_shape = [10,1]
input_to_hidden_shape = [4,10]
hidden_to_output_shape = [2,4]

W1init = np.zeros(input_to_hidden_shape,
          dtype="float32")
W2init = np.zeros(hidden_to_output_shape,
          dtype="float32")
```

Tensorflow graph

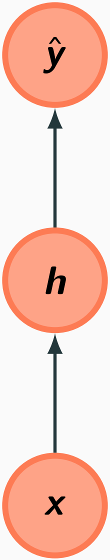
```
graph = tf.Graph()
with graph.as_default():
    x = tf.placeholder(shape=input_shape,
                       dtype="float32")
    W1 = tf.get_variable(initializer=W1init)
    v = tf.matmul(W1, x)
    h = tf.sigmoid(v)
    W2 = tf.get_variable(initializer=W2init)
    z = tf.matmul(W2, h)
    yhat = tf.nn.softmax(z)
```

Tensorboard visualization




$\boldsymbol{h}$
$\boldsymbol{h}'$
$\boldsymbol{W}_1$
$\boldsymbol{v}$
$\boldsymbol{x}$
$\hat{\boldsymbol{y}}$
$\boldsymbol{y}'$
$\boldsymbol{W}_2$
$\boldsymbol{z}$
$\boldsymbol{h}$

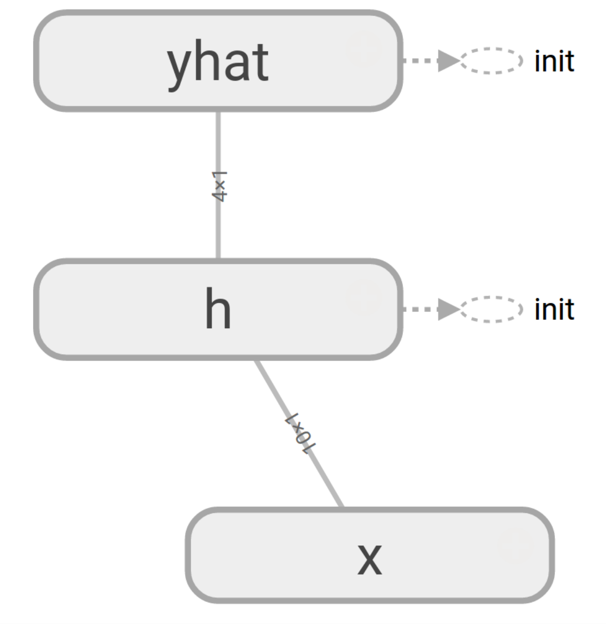
Computational Graphs



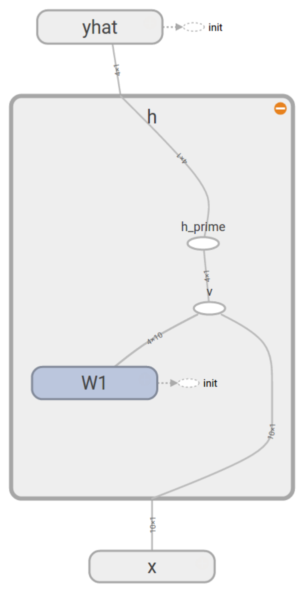
Tensorflow graph

```
graph = tf.Graph()
with graph.as_default():
    with tf.variable_scope("x"):
        x_prime = tf.placeholder(shape=input_shape,
                                 dtype="float32")

    with tf.variable_scope("h"):
        W1 = tf.get_variable(initializer=W1init)
        v = tf.matmul(W1, x_prime)
        h_prime = tf.sigmoid(v)

    with tf.variable_scope("yhat"):
        W2 = tf.get_variable(initializer=W2init)
        z = tf.matmul(W2, h_prime)
        y_prime = tf.nn.softmax(z)
```

Tensorboard visualization

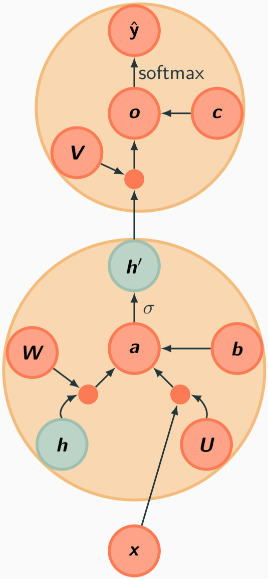


Tensorboard visualization

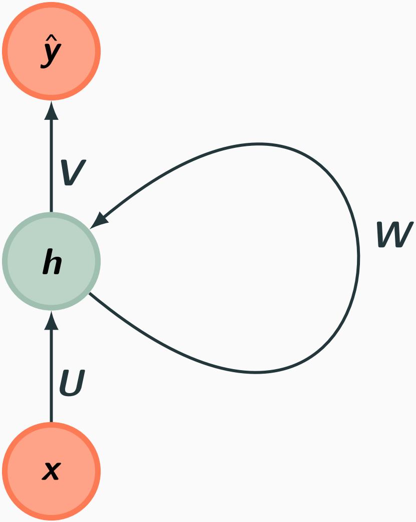


RNN: the model

RNN as a graph



RNN as a graph



Definition

A RNN is a function f with two inputs:

- An input vector $\boldsymbol{x}$.
- A hidden vector $\boldsymbol{h}$ representing a summary of all past inputs, called state or cell state.

Both inputs have a time step index t. The hidden unit has a recurrent definition:

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

Using our example as a concrete case

$$f(\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}; \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{c}, \boldsymbol{b}) = \hat{\boldsymbol{y}}^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c})$$

$$\boldsymbol{h}^{(t)} = g(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b})$$

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b})$$

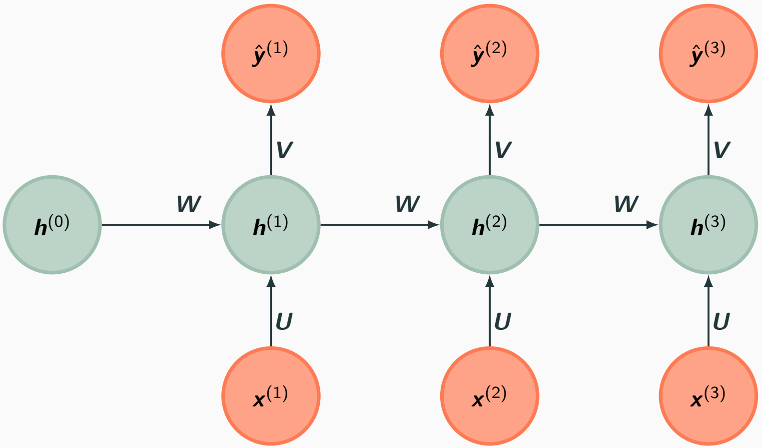
Unfolding the state equation

For a finite number of steps τ, the recurrent definition can be unfolded.

For example when $\tau = 3$:

$$\begin{aligned}
\boldsymbol{h}^{(3)} &= g(\boldsymbol{h}^{(2)}, \boldsymbol{x}^{(3)}; \boldsymbol{\theta}) \\
&= g(g(\boldsymbol{h}^{(1)}, \boldsymbol{x}^{(2)}; \boldsymbol{\theta}), \boldsymbol{x}^{(3)}; \boldsymbol{\theta}) \\
&= g(g(g(\boldsymbol{h}^{(0)}, \boldsymbol{x}^{(1)}; \boldsymbol{\theta}), \boldsymbol{x}^{(2)}; \boldsymbol{\theta}), \boldsymbol{x}^{(3)}; \boldsymbol{\theta})
\end{aligned}$$

Unfolding the graph



Language model

Definition

We call language model a probability distribution over sequences of tokens in a natural language.

$$P(x_1, x_2, x_3, x_4) = p$$

Used for:

- speech recognition
- machine translation
- text auto-completion
- spell correction
- question answering
- summarization

How do we build these probabilities?

Using the chain rule of probability:

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_1x_2)P(x_4|x_1x_2x_3)$$

To make things simple we use a **Markovian assumption**, i.e., for a specific n we assume that:

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t|x_1, \ldots, x_{t-1}) = \prod_{t=1}^{T} P(x_t|x_{t-(n+1)}, \ldots, x_{t-1})$$

Models based on n-gram statistics

The choice of n yields different models.

Unigram language model ($n = 1$):

$$P_{uni}(x_1, x_2, x_3, x_4) = P(x_1)P(x_2)P(x_3)P(x_4)$$

where $P(x_i) = count(x_i)$.

Bigram language model ($n = 2$):

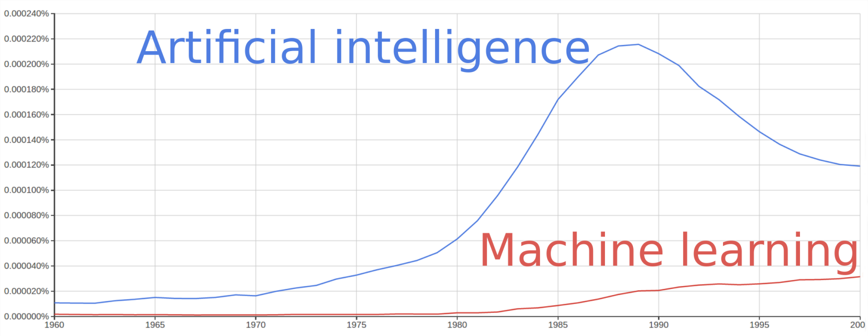
$$P_{bi}(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3)$$

where

$$P(x_i|x_j) = \frac{count(x_i, x_j)}{count(x_j)}$$

n-gram statistics

https://books.google.com/ngrams



Evaluating a language model

- **extrinsic task**: How our model perform in a NLP task such as text auto-completion.
    - Time consuming.
- **intrinsic evaluation**: perplexity.
    - It works only when the test data is very similar to the training data.

Perplexity

Perplexity (PP) can be thought as the weighted average branching factor of a language.

Given $C = x_1, x_2, \ldots, x_T$, we define the perplexity of C as:

$$PP(C) = P(x_1, x_2, \ldots, x_T)^{-\frac{1}{T}}$$

$$= \sqrt[T]{\frac{1}{P(x_1, x_2, \ldots, x_T)}}$$

$$= \sqrt[T]{\prod_{i=1}^{T} \frac{1}{P(x_i | x_1, \ldots, x_{i-1})}}$$

Models based on n-gram statistics

- Higher n-grams yields better performance.
- Higher n-grams requires a lot of memory!

> *"Using one machine **with 140 GB RAM for 2.8 days**, we built an unpruned model on 126 billion tokens."*
>
> *Scalable Modified Kneser-Ney Language Model Estimation* by Heafield et al.

Languagem model as sequential data prediction

Instead of using one approach that is specific for the language domain, we can use a general model for sequential data prediction: a **RNN**.

Our learning task is to estimate the probability distribution

$$P(x_n = \mathrm{word}_{j^*} | x_1, \ldots, x_{n-1})$$

for any $(n-1)$-sequence of words $x_1, \ldots, x_{n-1}$.

Building the dataset

We start with a corpus C with T tokens and a vocabulary $\mathbb{V}$.

Example: **Make Some Noise** by the Beastie Boys.

Yes, here we go again, give you more, nothing lesser
Back on the mic is the anti-depressor
Ad-Rock, the pressure, yes, we need this
The best is yet to come, and yes, believe this
...

- $T = 378$
- $|\mathbb{V}| = 186$

Building the dataset

The dataset is a collection of pairs $(\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{x}$ is one word and $\boldsymbol{y}$ is the immediately next word. For example:

$$(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}) = (\text{Yes, here}).$$
$$(\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}) = (\text{here, we})$$
$$(\boldsymbol{x}^{(3)}, \boldsymbol{y}^{(3)}) = (\text{we, go})$$
$$(\boldsymbol{x}^{(4)}, \boldsymbol{y}^{(4)}) = (\text{go, again})$$
$$(\boldsymbol{x}^{(5)}, \boldsymbol{y}^{(5)}) = (\text{again, give})$$
$$(\boldsymbol{x}^{(6)}, \boldsymbol{y}^{(6)}) = (\text{give, you})$$
$$(\boldsymbol{x}^{(7)}, \boldsymbol{y}^{(7)}) = (\text{you, more})$$
$$\ldots$$

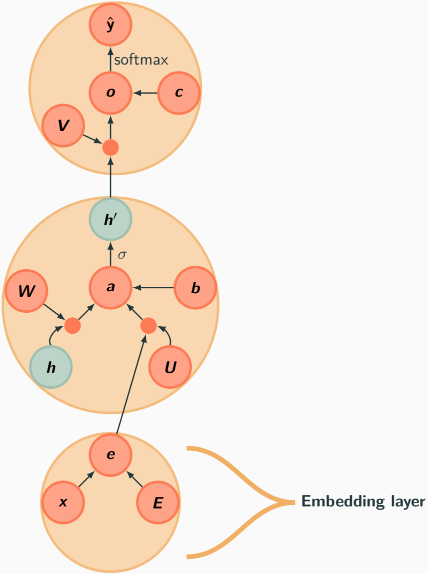
Notation

- $\boldsymbol{E} \in \mathbb{R}^{d,|\mathbb{V}|}$ is the matrix of word embeddings.
- $\boldsymbol{x}^{(t)} \in \mathbb{R}^{|\mathbb{V}|}$ is one-hot word vector at time step t.
- $\boldsymbol{y}^{(t)} \in \mathbb{R}^{|\mathbb{V}|}$ is the ground truth at time step t (also an one-hot word vector).

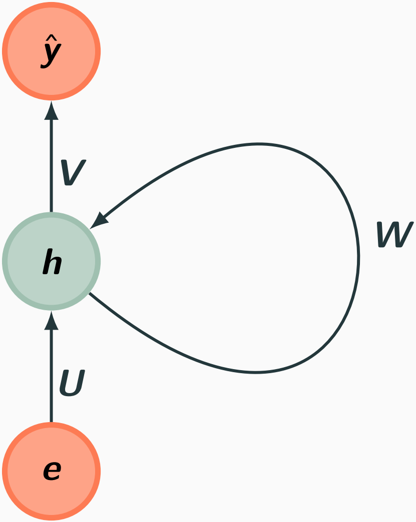
Recap: selecting word embeddings

$$
\begin{aligned}
\boldsymbol{e} &= \boldsymbol{E} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \\
&= \boldsymbol{E}_{:,\, j}
\end{aligned}
$$

The language model: graph



The language model: graph



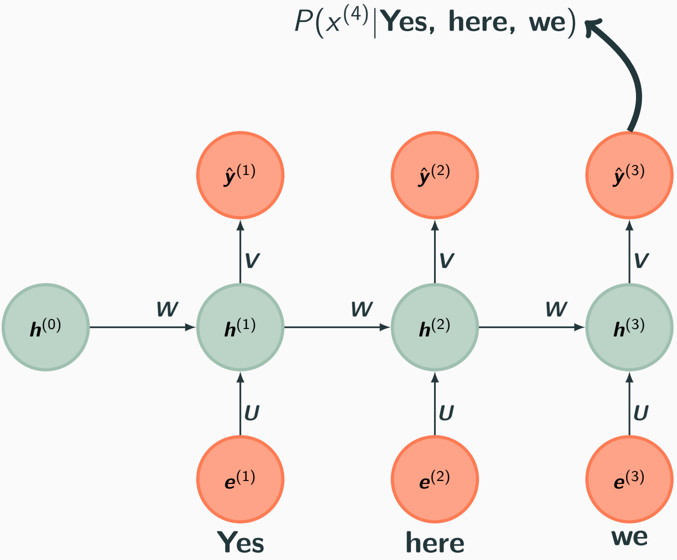
The language model: equations

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \sigma\big(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{e}^{(t)} + \boldsymbol{b}\big)$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax\big(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}\big)$$

The Language model: unfolding example



Recap: Entropy

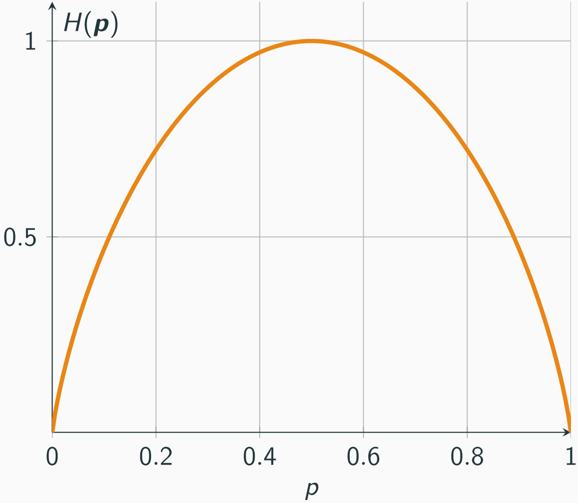
$$\boldsymbol{p} \qquad\qquad \boldsymbol{q}$$

$$\begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} \qquad \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$H(\boldsymbol{p}) = 0.72 \qquad H(\boldsymbol{q}) = 1$$

$$H(\boldsymbol{p}) = \textstyle\sum_i \boldsymbol{p}_i \log \frac{1}{\boldsymbol{p}_i}$$

Recap: Entropy



$$\boldsymbol{p}$$

$$\begin{bmatrix} p \\ 1-p \end{bmatrix}$$

Recap: Kullback-Leibler divergence

$$\boldsymbol{p} = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} \qquad \boldsymbol{q} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad \boldsymbol{p}' = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} \qquad \boldsymbol{q}' = \begin{bmatrix} 0.88 \\ 0.12 \end{bmatrix}$$

$$D_{KL}(\boldsymbol{p}||\boldsymbol{q}) = 0.28 \qquad D_{KL}(\boldsymbol{p}'||\boldsymbol{q}') = 0.04$$

$$D_{KL}(\boldsymbol{p}||\boldsymbol{q}) = \sum_i \boldsymbol{p}_i \log \frac{\boldsymbol{p}_i}{\boldsymbol{q}_i}$$

Recap: Cross entropy

$$
\begin{aligned}
CE(\boldsymbol{p}, \boldsymbol{q}) &= H(\boldsymbol{p}) + D_{KL}(\boldsymbol{p} || \boldsymbol{q}) \\
&= -\sum_i \boldsymbol{p}_i \log(\boldsymbol{q}_i)
\end{aligned}
$$

$$
\arg\min_{\boldsymbol{q}} CE(\boldsymbol{p}, \boldsymbol{q}) = \arg\min_{\boldsymbol{q}} D_{KL}(\boldsymbol{p}, \boldsymbol{q})
$$

Loss function

At each time t the point-wise loss is:

$$\begin{aligned} L^{(t)} &= CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) \\ &= -\log(\hat{\boldsymbol{y}}_{j^*}) \\ &= -\log P(x^{(t+1)} = \text{word}_{j^*} | x^{(1)}, \ldots, x^{(t)}) \end{aligned}$$

For example:

$$L^{(3)} = -\log P(x^{(4)} = \text{go} | \text{Yes}, \text{here}, \text{we})$$

Loss function

The loss L is the mean of all the point-wise losses

$$L = \frac{1}{T} \sum_{t=1}^{T} L^{(t)}$$

To give a concrete example, let's take the first sentence of the lyric as C:

Yes, here we go again, give you more, nothing lesser

- $T = 10$
- $|\mathbb{V}| = 10$

Loss function: example

$$
\begin{aligned}
L = -\frac{1}{10}[&\log P(\text{here}|\text{Yes}) \\
&+ \log P(\text{we}|\text{Yes}, \text{here}) \\
&+ \log P(\text{go}|\text{Yes}, \text{here}, \text{we}) \\
&+ \log P(\text{again}|\text{Yes}, \text{here}, \text{we}, \text{go}) \\
&+ \log P(\text{give}|\text{Yes}, \text{here}, \text{we}, \text{go}, \text{again}) \\
&+ \log P(\text{you}|\text{Yes}, \text{here}, \text{we}, \text{go}, \text{again}, \text{give}) \\
&+ \log P(\text{more}|\text{Yes}, \text{here}, \text{we}, \text{go}, \text{again}, \text{give}, \text{you}) \\
&+ \log P(\text{nothing}|\text{Yes}, \text{here}, \text{we}, \text{go}, \text{again}, \text{give}, \text{you}, \text{more}) \\
&+ \log P(\text{lesser}|\text{Yes}, \text{here}, \text{we}, \text{go}, \text{again}, \text{give}, \text{you}, \text{more}, \text{nothing}) \\
&+ \log P(< \text{eos} > |\text{Yes}, \text{here}, \text{we}, \text{go}, \text{again}, \text{give}, \text{you}, \text{more}, \text{nothing}, \text{lesser})]
\end{aligned}
$$

Loss and Perplexity

Since

$$
\begin{aligned}
L^{(t)} &= -\log P(x^{(t+1)}|x^{(1)},\ldots,x^{(t)}) \\
&= \log(\frac{1}{P(x^{(t+1)}|x^{(1)},\ldots,x^{(t)})})
\end{aligned}
$$

We have that:

$$
\begin{aligned}
L &= \frac{1}{T}\sum_{t=1}^{T} L^{(t)} \\
&= \log\left(\sqrt[T]{\prod_{i=1}^{T}\frac{1}{P(x_i|x_1,\ldots,x_{i-1})}}\right) \\
&= \log(PP(C))
\end{aligned}
$$

Loss and Perplexity

So another definition of perplexity is

$$2^L = PP(C)$$

Back Propagation

Chain rule of Calculus

- $x \in \mathbb{R}$
- $f : \mathbb{R} \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$.
- $y = g(x)$
- $z = f(g(x)) = f(y)$

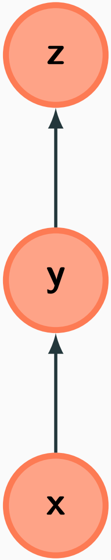
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Chain rule: example

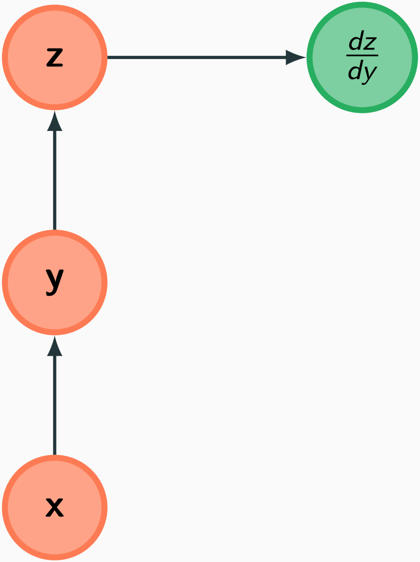
- $y = x^2$
- $z = \log(y)$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = \frac{1}{x^2}2x = \frac{2}{x}$$

Chain rule: graph

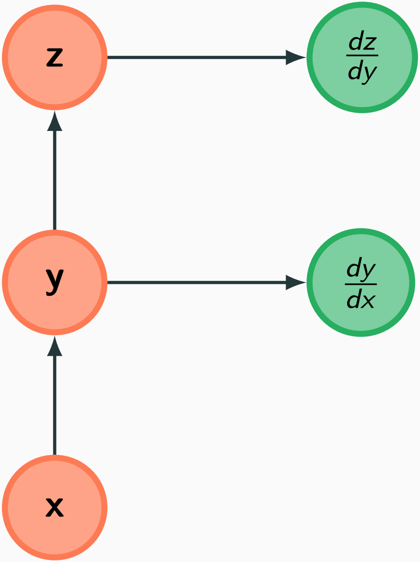


Chain rule: graph



$= \frac{1}{x^2}$

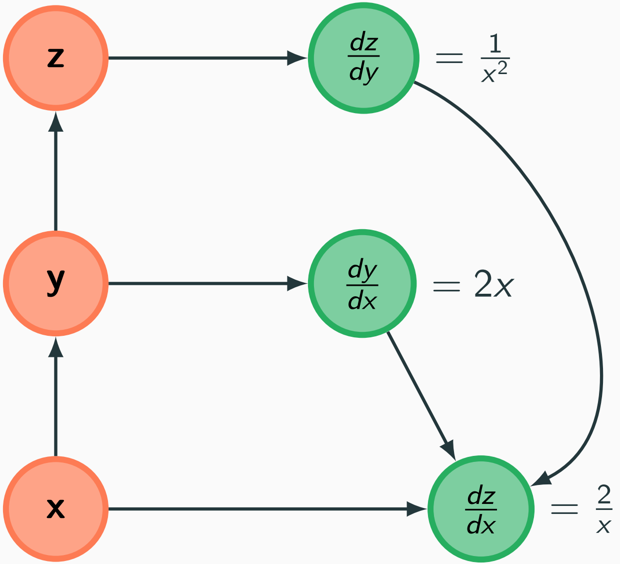
Chain rule: graph



$= \frac{1}{x^2}$

$= 2x$

Chain rule: graph



Chain rule: vector notation

- $\boldsymbol{x} \in \mathbb{R}^m$
- $\boldsymbol{y} \in \mathbb{R}^n$
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$.
- $\boldsymbol{y} = g(\boldsymbol{x})$
- $\boldsymbol{z} = f(g(\boldsymbol{x})) = f(\boldsymbol{y})$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

Chain rule: vector notation

- $\boldsymbol{x} \in \mathbb{R}^m$
- $\boldsymbol{y} \in \mathbb{R}^n$
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$.
- $\boldsymbol{y} = g(\boldsymbol{x})$
- $\boldsymbol{z} = f(g(\boldsymbol{x})) = f(\boldsymbol{y})$

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} z$$

Recap: gradient

$$\nabla_{\boldsymbol{y}} z = \begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \frac{\partial z}{\partial y_2} \\ \vdots \\ \frac{\partial z}{\partial y_n} \end{bmatrix}$$

Recap: Jacobian matrix

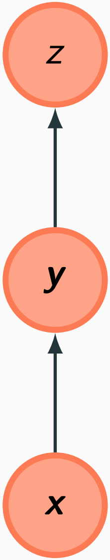
$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \ldots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}$$

Chain rule: example

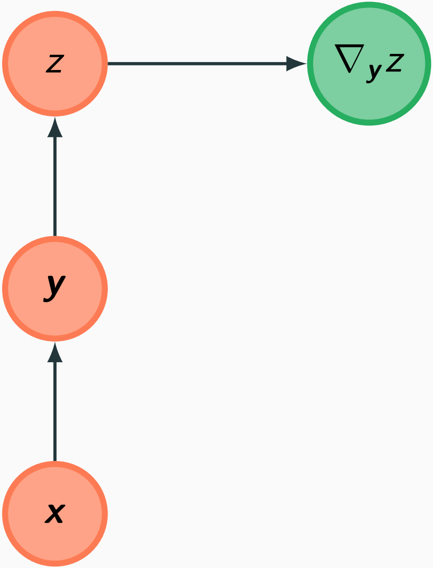
- $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$
- $z = \boldsymbol{b}^T \boldsymbol{y}$

$$\nabla_{\boldsymbol{x}} z = \boldsymbol{W}^T \boldsymbol{b}$$

Chain rule: graph

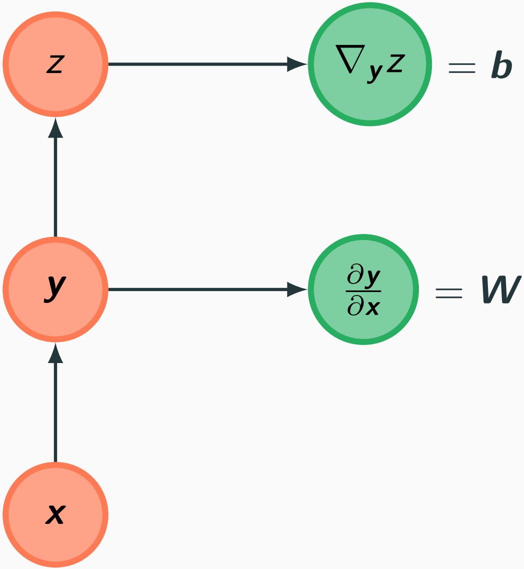


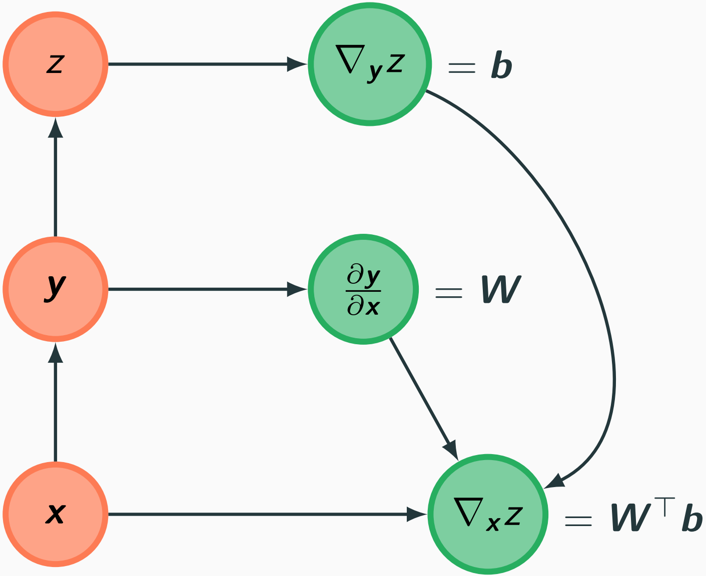
Chain rule: graph



$= \boldsymbol{b}$

Chain rule: graph




z
$\nabla_{\boldsymbol{y}} z$
$= \boldsymbol{b}$
$\boldsymbol{y}$
$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$
$= \boldsymbol{W}$
$\boldsymbol{x}$
$\nabla_{\boldsymbol{x}} z$
$= \boldsymbol{W}^\top \boldsymbol{b}$

Computing the gradient in a RNN

- We simple apply the back-propagation algorithm to the unrolled computational graph.

- Since each subgraph represents a time step, the application of back-propagation in this model is also called Back-Propagation Through Time.

A very simple RNN

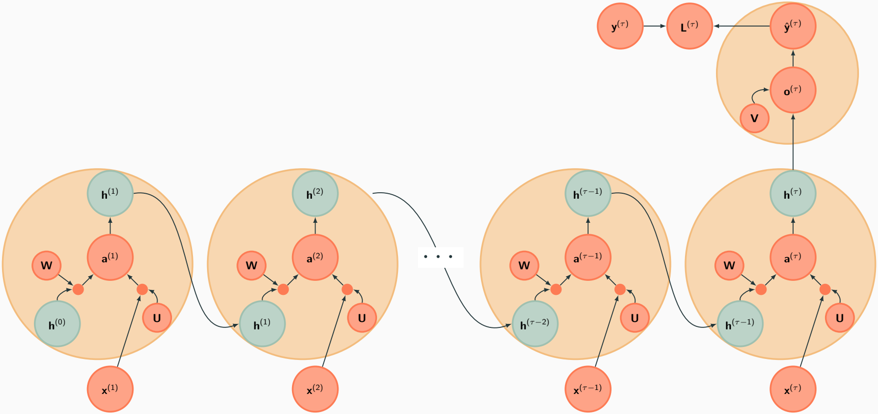
$$\boldsymbol{a}^{(t)} = \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{a}^{(t)})$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{V}\boldsymbol{h}^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{o}^{(t)})$$

RNN time unfolding

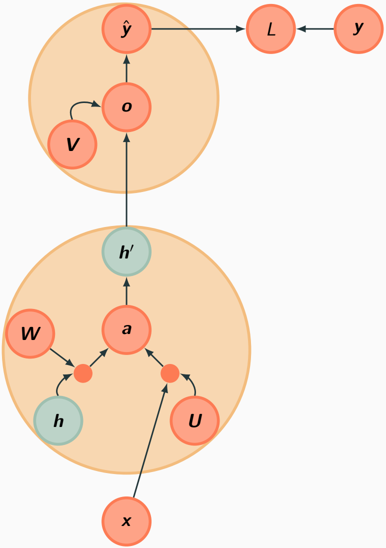


Recap: Hadamard product

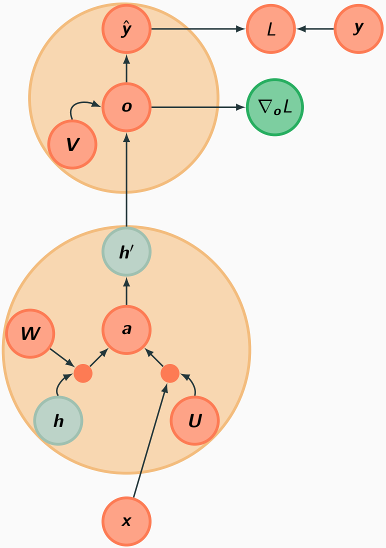
$$(\boldsymbol{x} \circ \boldsymbol{y})_i = \boldsymbol{x}_i \boldsymbol{y}_i$$

$$(\boldsymbol{x} \circ \boldsymbol{y}) = diag(\boldsymbol{y})\boldsymbol{x}$$

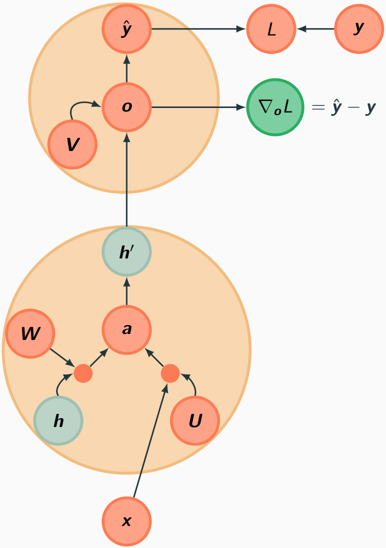
Chain rule: RNN graph



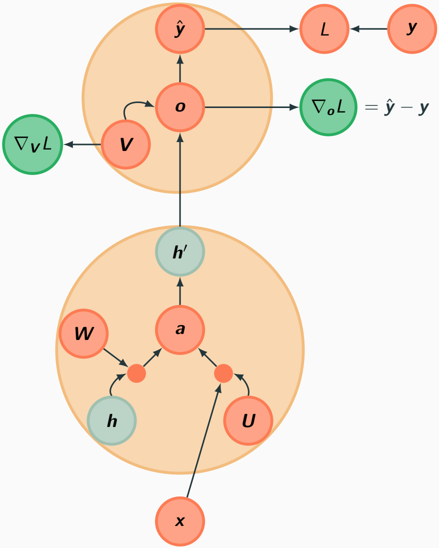
Chain rule: RNN graph



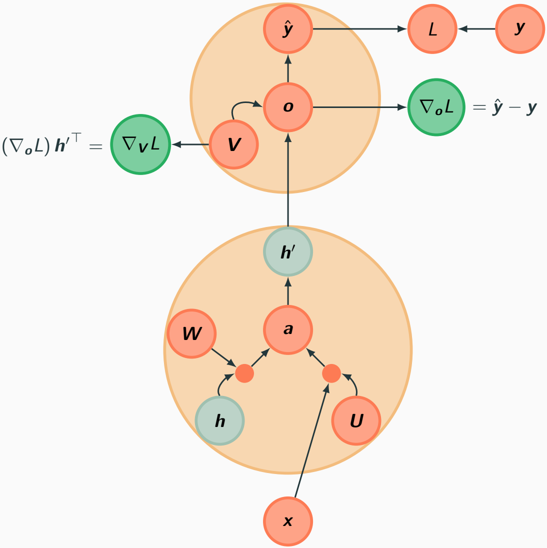
Chain rule: RNN graph



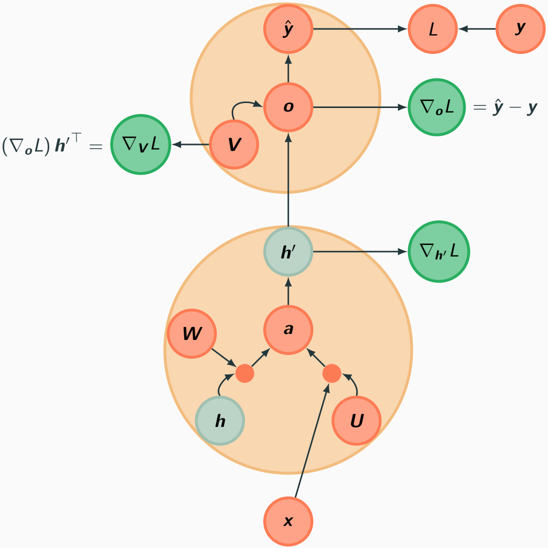
Chain rule: RNN graph



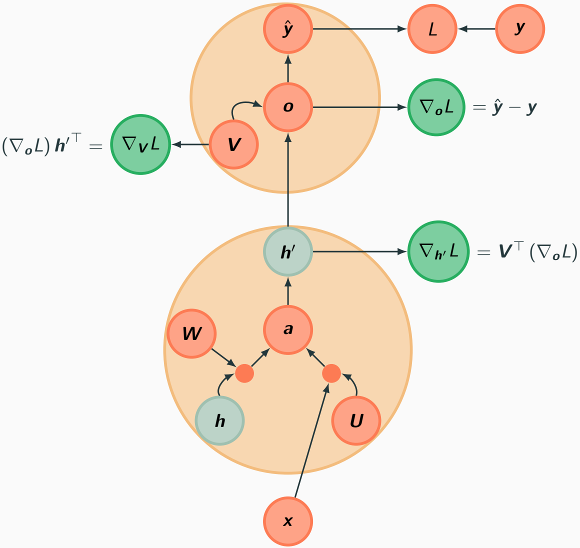
Chain rule: RNN graph



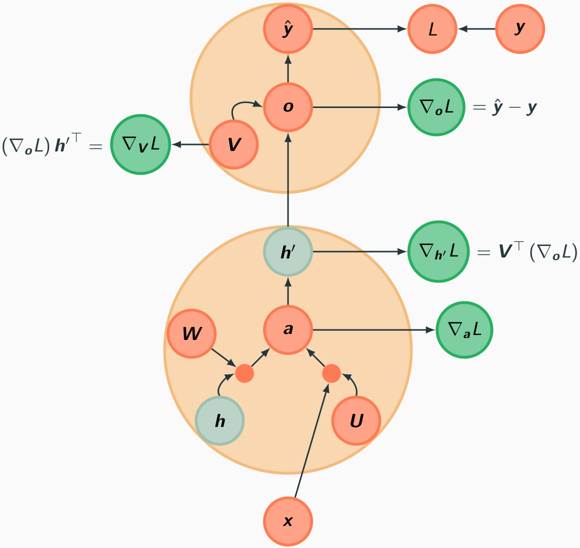
Chain rule: RNN graph



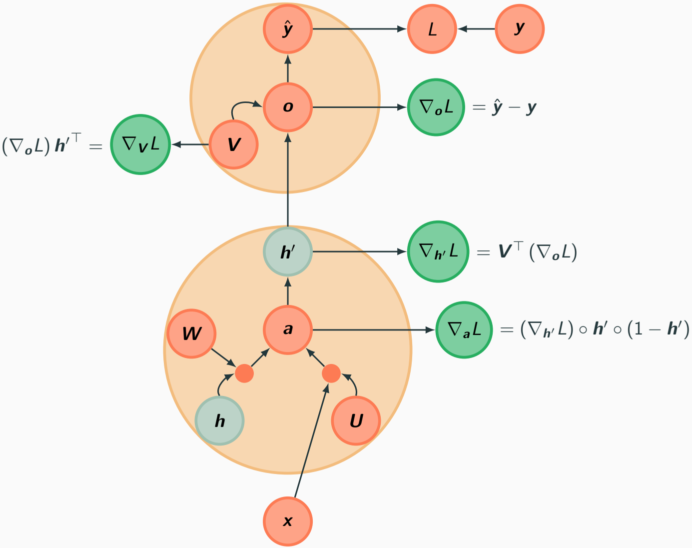
Chain rule: RNN graph



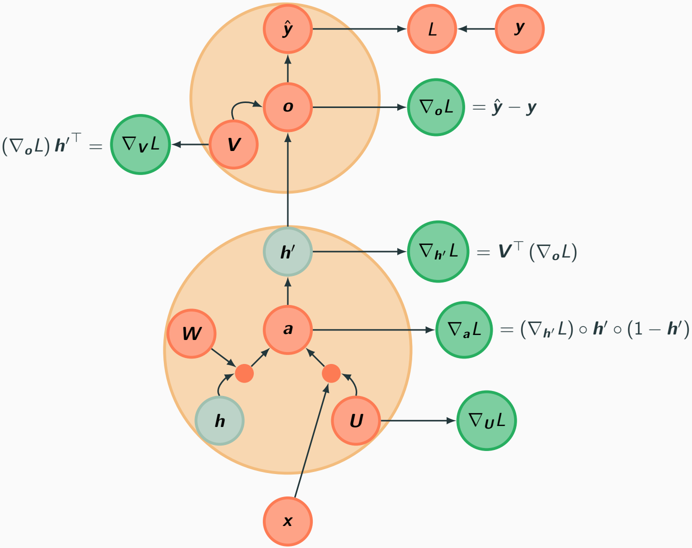
Chain rule: RNN graph



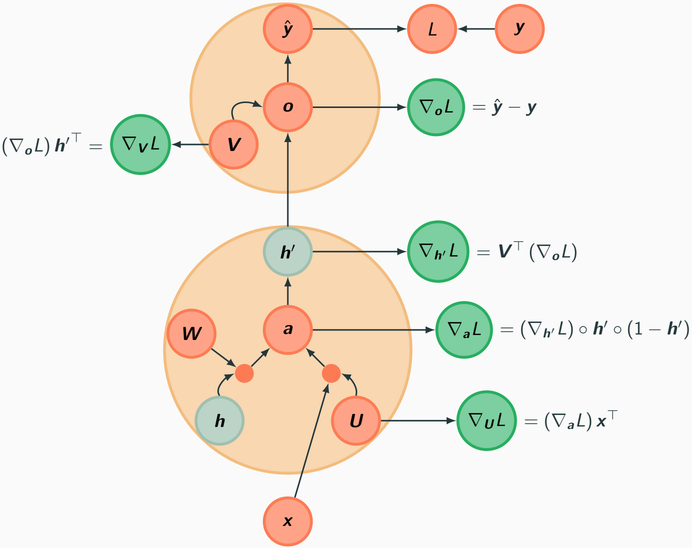
Chain rule: RNN graph



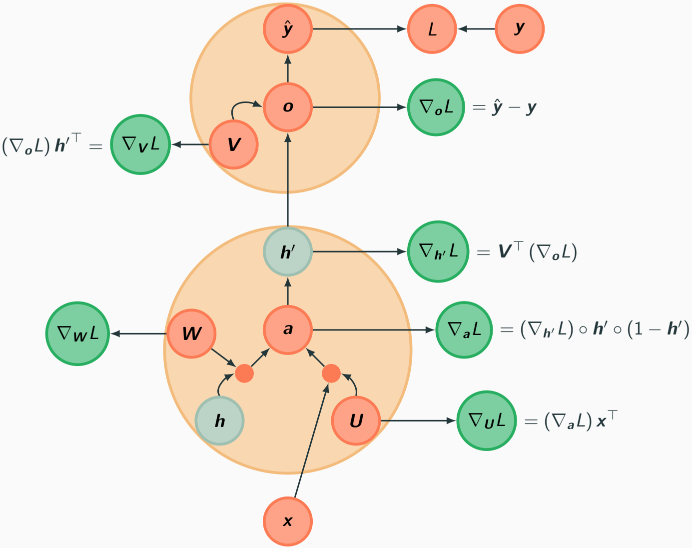
Chain rule: RNN graph



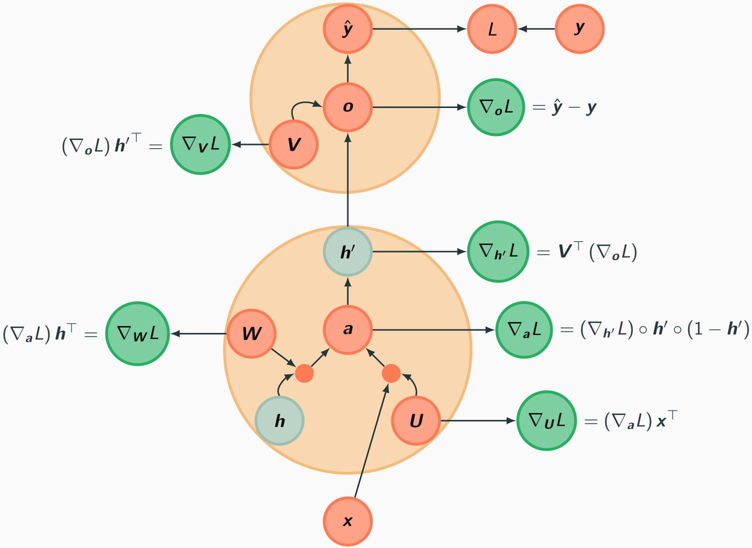
Chain rule: RNN graph



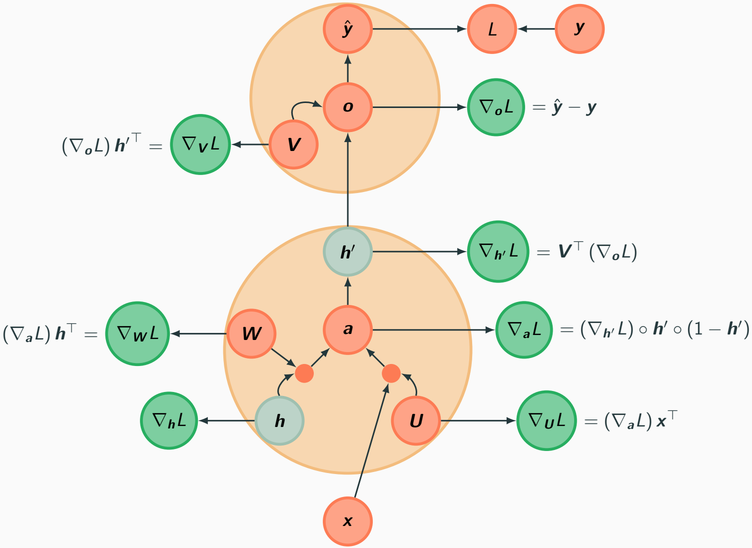
Chain rule: RNN graph



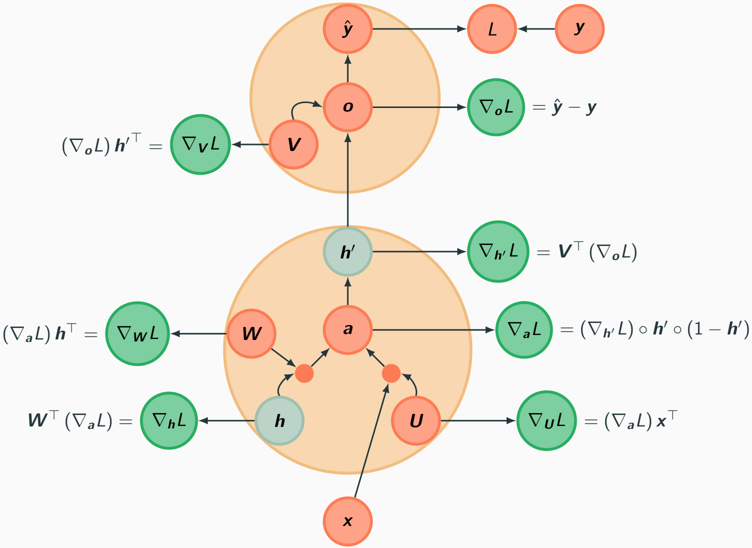
Chain rule: RNN graph



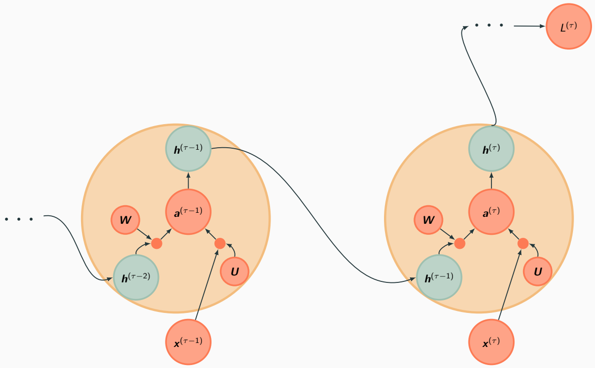
Chain rule: RNN graph



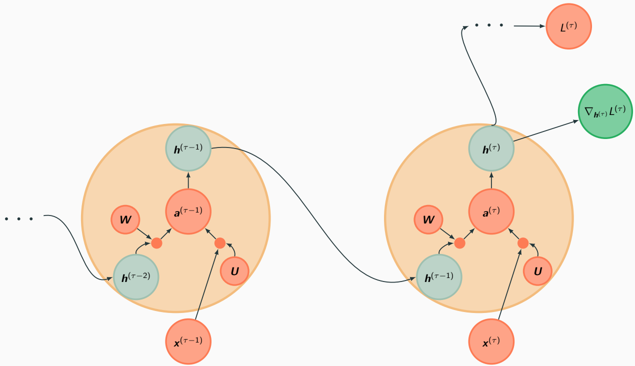
Chain rule: RNN graph



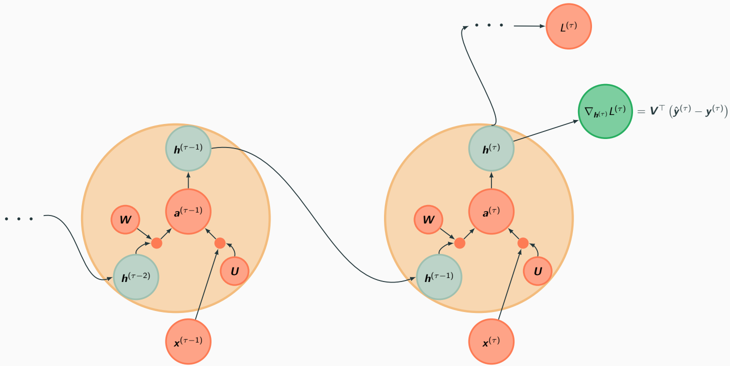
Back Propagation Throught Time



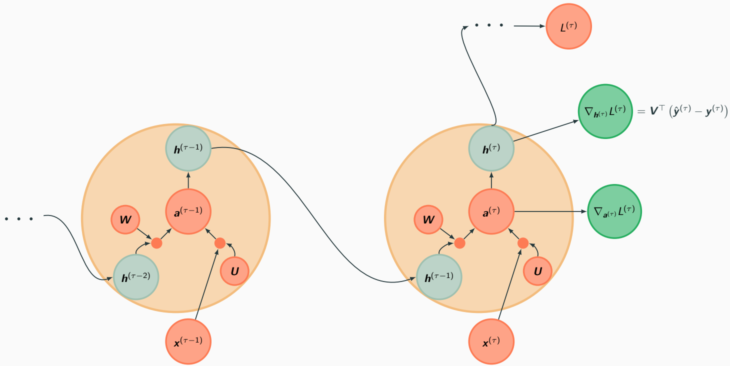
Back Propagation Throught Time



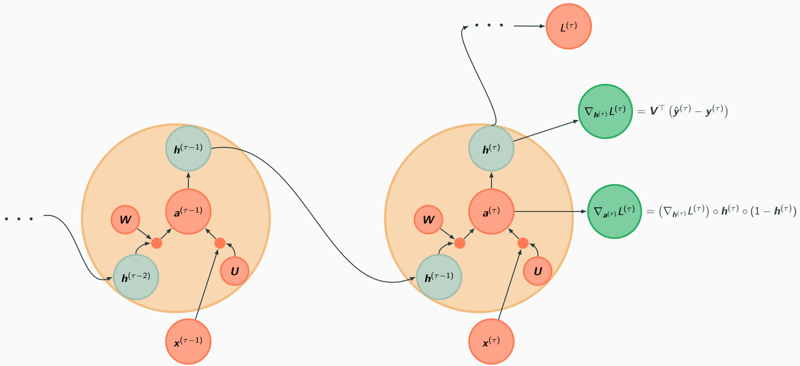
Back Propagation Throught Time



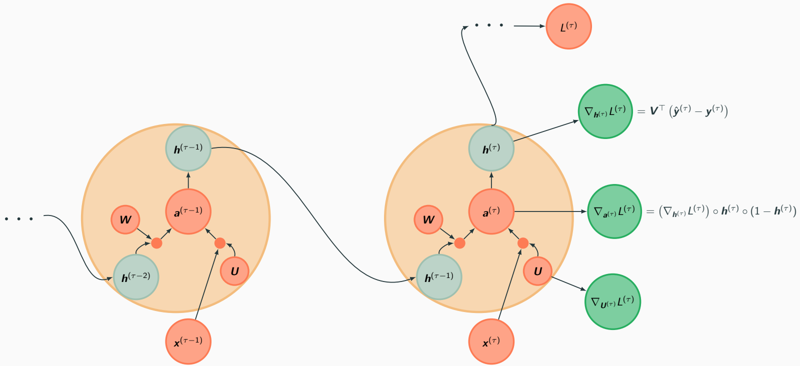
Back Propagation Throught Time



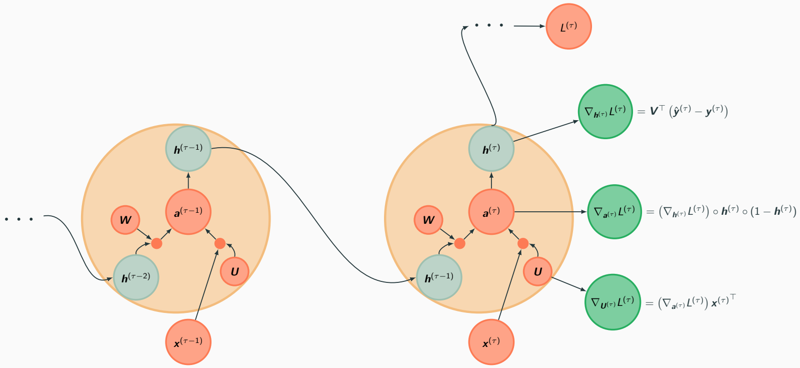
Back Propagation Throught Time



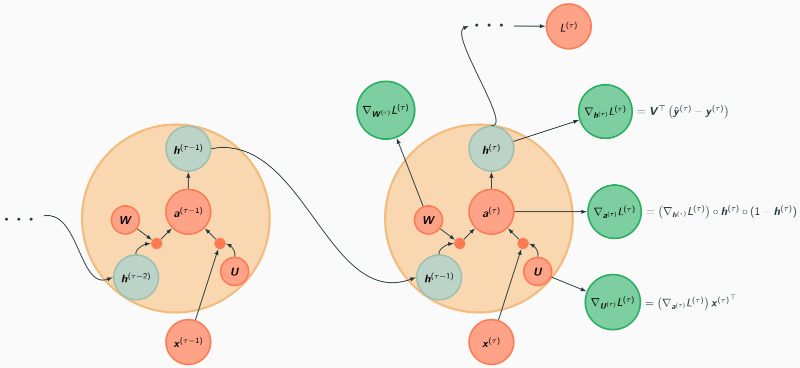
Back Propagation Throught Time



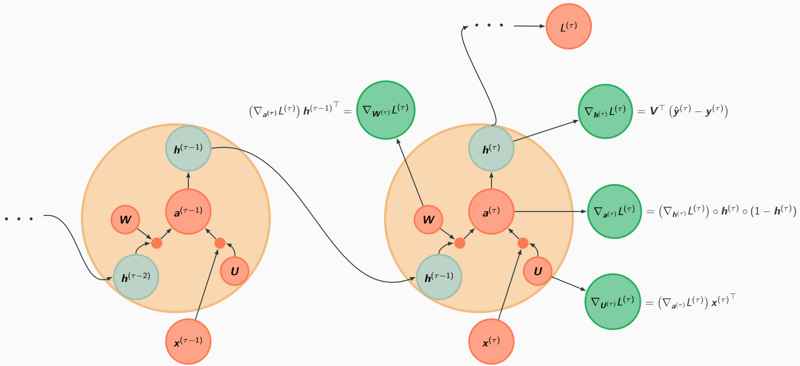
Back Propagation Throught Time



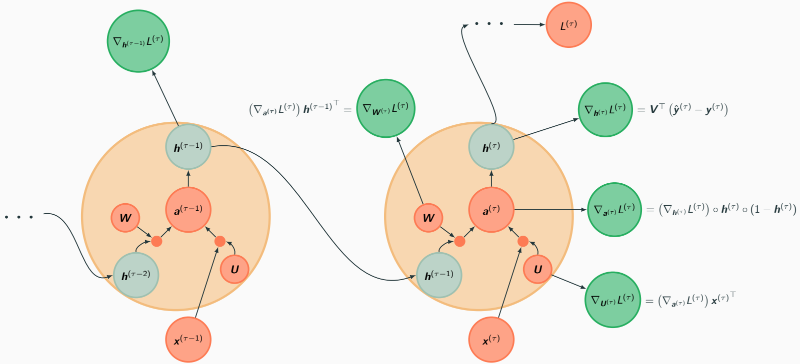
Back Propagation Throught Time



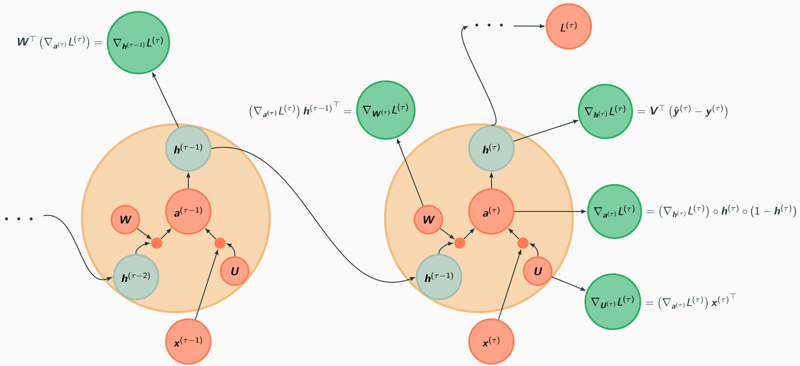
Back Propagation Throught Time



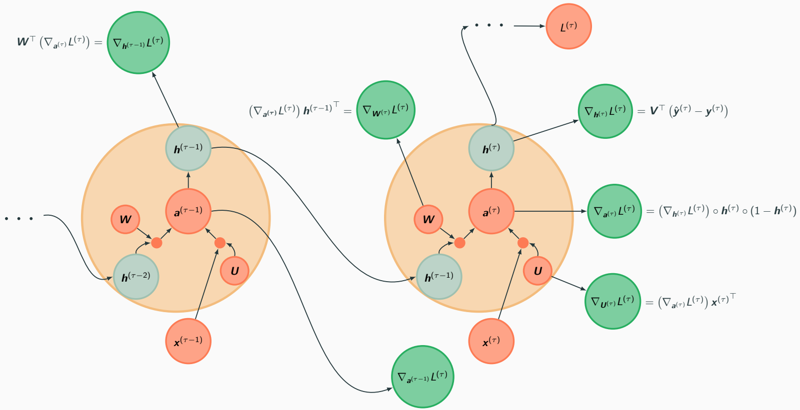
Back Propagation Throught Time



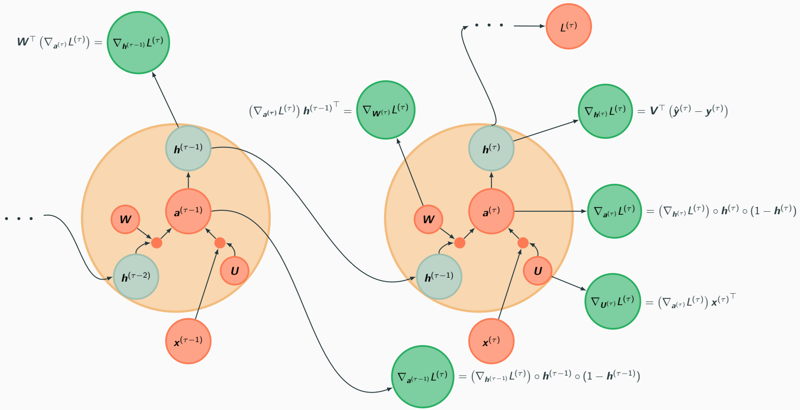
Back Propagation Throught Time



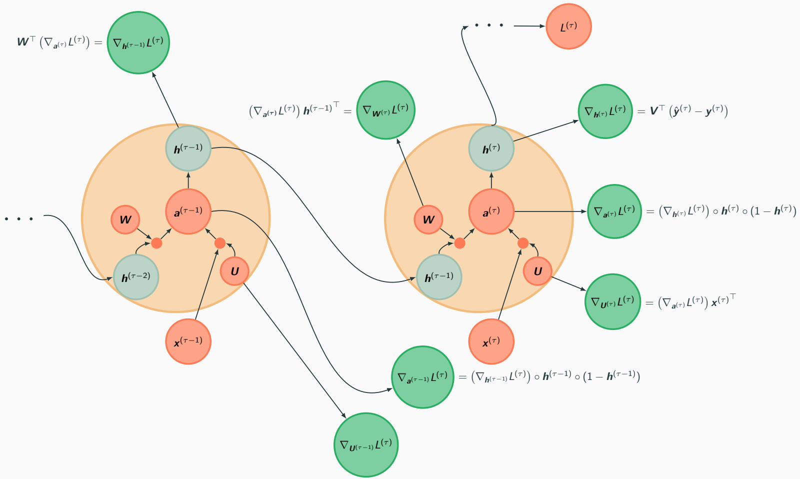
Back Propagation Throught Time



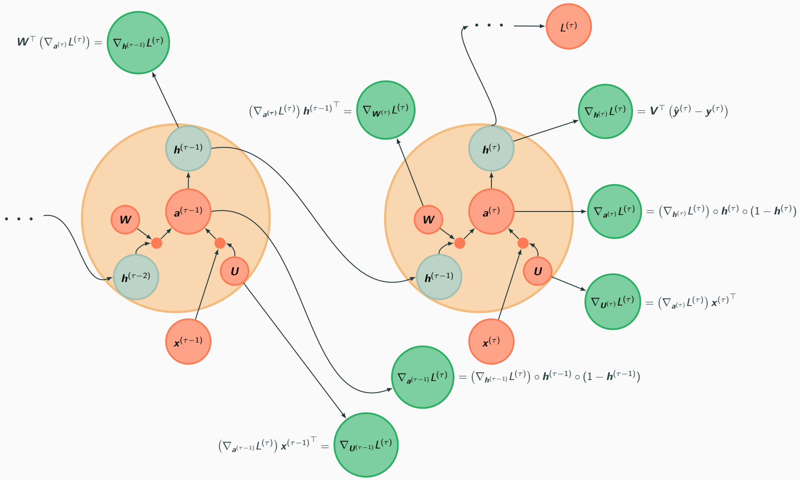
Back Propagation Throught Time



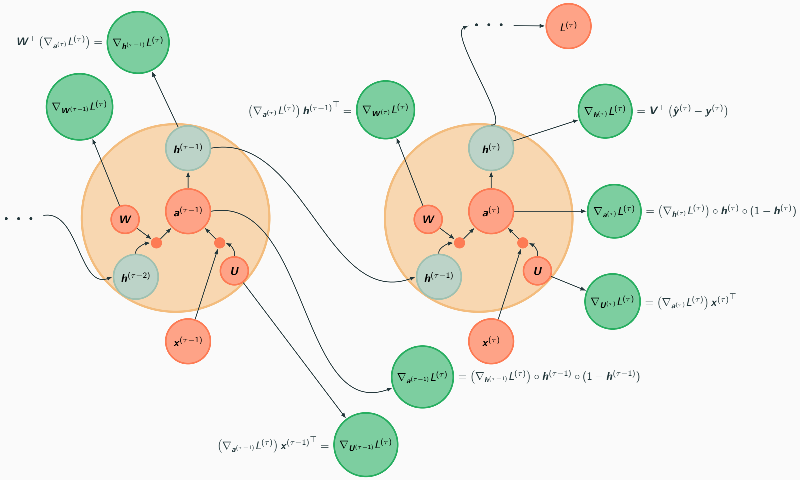
Back Propagation Throught Time



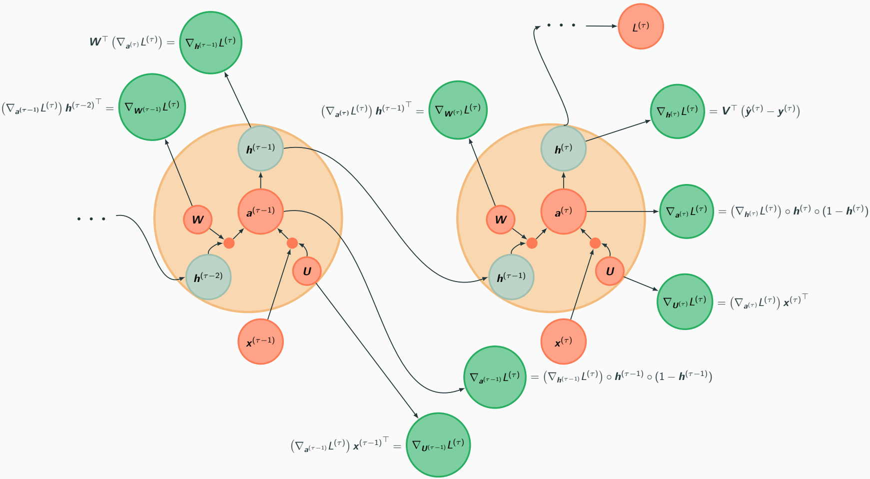
Back Propagation Throught Time



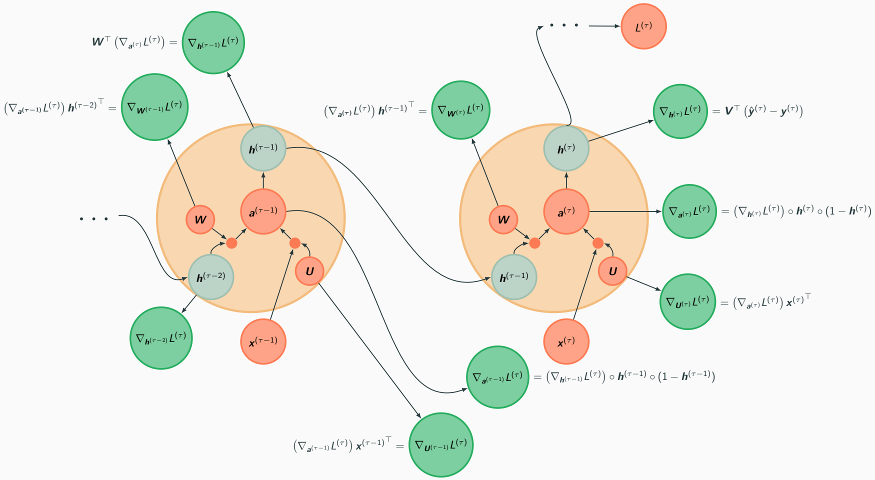
Back Propagation Throught Time



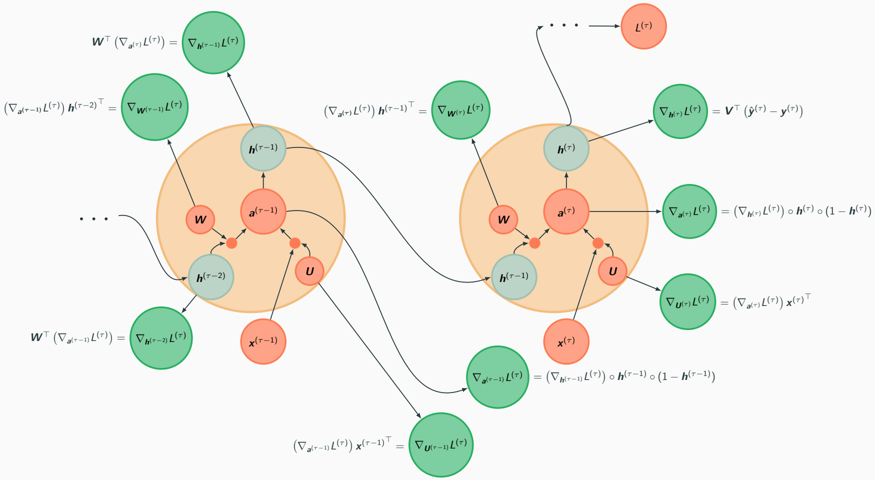
Back Propagation Throught Time



Back Propagation Throught Time



Back Propagation Throught Time



Back Propagation Through Time

The gradients on $\boldsymbol{V}$, $\boldsymbol{W}$ and $\boldsymbol{U}$ are:

$$\nabla_{\boldsymbol{V}} L^{(\tau)} = \left(\nabla_{\boldsymbol{o}^{(\tau)}} L^{(\tau)}\right) \boldsymbol{h}^{(\tau)\top}$$

$$\nabla_{\boldsymbol{W}} L^{(\tau)} = \sum_{t=1}^{\tau} \nabla_{\boldsymbol{W}^{(t)}} L^{(\tau)}$$

$$\nabla_{\boldsymbol{U}} L^{(\tau)} = \sum_{t=1}^{\tau} \nabla_{\boldsymbol{U}^{(t)}} L^{(\tau)}$$

Vanishing or Exploding Gradient

The problem of training RNN

Let's calculate $\nabla_{\boldsymbol{U}} L^{(3)}$:

$$\begin{aligned}\nabla_{\boldsymbol{U}} L^{(3)} &= \sum_{t=1}^{3} \nabla_{\boldsymbol{U}^{(t)}} L^{(3)} \\ &= \nabla_{\boldsymbol{U}^{(1)}} L^{(3)} + \nabla_{\boldsymbol{U}^{(2)}} L^{(3)} + \nabla_{\boldsymbol{U}^{(3)}} L^{(3)}\end{aligned}$$

Calculating $\nabla_{\boldsymbol{U}^{(3)}} L^{(3)}$

$$\nabla_{\boldsymbol{U}^{(3)}} L^{(3)} = \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right) \boldsymbol{x}^{(3)\top}$$

Calculating $\nabla_{\boldsymbol{U}^{(3)}} L^{(3)}$

$$\begin{aligned}
\nabla_{\boldsymbol{U}^{(3)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right) \boldsymbol{x}^{(3)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(3)}} L^{(3)}\right) \circ \boldsymbol{h}^{(3)} \circ \left(1 - \boldsymbol{h}^{(3)}\right)\right) \boldsymbol{x}^{(3)\top}
\end{aligned}$$

Calculating $\nabla_{\boldsymbol{U}^{(3)}} L^{(3)}$

$$\begin{aligned}
\nabla_{\boldsymbol{U}^{(3)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right) \boldsymbol{x}^{(3)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(3)}} L^{(3)}\right) \circ \boldsymbol{h}^{(3)} \circ (1 - \boldsymbol{h}^{(3)})\right) \boldsymbol{x}^{(3)\top} \\
&= \left(\left(\boldsymbol{V}^\top \left(\hat{\boldsymbol{y}}^{(3)} - \boldsymbol{y}^{(3)}\right)\right) \circ \boldsymbol{h}^{(3)} \circ (1 - \boldsymbol{h}^{(3)})\right) \boldsymbol{x}^{(3)\top}
\end{aligned}$$

Calculating $\nabla_{\boldsymbol{U}^{(2)}} L^{(3)}$

$$\nabla_{\boldsymbol{U}^{(2)}} L^{(3)} = \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right) \boldsymbol{x}^{(2)\top}$$

Calculating $\nabla_{\boldsymbol{U}^{(2)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(2)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(2)}} L^{(3)}\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top}
\end{aligned}
$$

Calculating $\nabla_{\boldsymbol{U}^{(2)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(2)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(2)}} L^{(3)}\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top}
\end{aligned}
$$

Calculating $\nabla_{\boldsymbol{U}^{(2)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(2)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(2)}} L^{(3)}\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\boldsymbol{W}^{\top} \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\boldsymbol{W}^{\top} \left(\left(\nabla_{\boldsymbol{h}^{(3)}} L^{(3)}\right) \circ \boldsymbol{h}^{(3)} \circ (1 - \boldsymbol{h}^{(3)})\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top}
\end{aligned}
$$

Calculating $\nabla_{\boldsymbol{U}^{(2)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(2)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(2)}} L^{(3)}\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\nabla_{\boldsymbol{h}^{(3)}} L^{(3)}\right) \circ \boldsymbol{h}^{(3)} \circ (1 - \boldsymbol{h}^{(3)})\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\boldsymbol{V}^\top \left(\hat{\boldsymbol{y}}^{(3)} - \boldsymbol{y}^{(3)}\right)\right) \circ \boldsymbol{h}^{(3)} \circ (1 - \boldsymbol{h}^{(3)})\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right) \boldsymbol{x}^{(2)\top}
\end{aligned}
$$

Calculating $\nabla_{\boldsymbol{U}^{(1)}} L^{(3)}$

$$\nabla_{\boldsymbol{U}^{(1)}} L^{(3)} = \left(\nabla_{\boldsymbol{a}^{(1)}} L^{(3)}\right) \boldsymbol{x}^{(1)\top}$$

Calculating $\nabla_{\boldsymbol{U}^{(1)}} L^{(3)}$

$$\begin{aligned}
\nabla_{\boldsymbol{U}^{(1)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(1)}} L^{(3)}\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(1)}} L^{(3)}\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top}
\end{aligned}$$

Calculating $\nabla_{\boldsymbol{U}^{(1)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(1)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(1)}} L^{(3)}\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(1)}} L^{(3)}\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^{\top} \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top}
\end{aligned}
$$

Calculating $\nabla_{\boldsymbol{U}^{(1)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(1)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(1)}} L^{(3)}\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(1)}} L^{(3)}\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\nabla_{\boldsymbol{h}^{(2)}} L^{(3)}\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top}
\end{aligned}
$$

Calculating $\nabla_{\boldsymbol{U}^{(1)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(1)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(1)}} L^{(3)}\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(1)}} L^{(3)}\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\nabla_{\boldsymbol{h}^{(2)}} L^{(3)}\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top}
\end{aligned}
$$

Calculating $\nabla_{\boldsymbol{U}^{(1)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(1)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(1)}} L^{(3)}\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(1)}} L^{(3)}\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\nabla_{\boldsymbol{h}^{(2)}} L^{(3)}\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\boldsymbol{W}^\top \left(\left(\nabla_{\boldsymbol{h}^{(3)}} L^{(3)}\right) \circ \boldsymbol{h}^{(3)} \circ (1 - \boldsymbol{h}^{(3)})\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1 - \boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1 - \boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top}
\end{aligned}
$$

Calculating $\nabla_{\boldsymbol{U}^{(1)}} L^{(3)}$

$$
\begin{aligned}
\nabla_{\boldsymbol{U}^{(1)}} L^{(3)} &= \left(\nabla_{\boldsymbol{a}^{(1)}} L^{(3)}\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\nabla_{\boldsymbol{h}^{(1)}} L^{(3)}\right) \circ \boldsymbol{h}^{(1)} \circ (1-\boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(2)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1-\boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\nabla_{\boldsymbol{h}^{(2)}} L^{(3)}\right) \circ \boldsymbol{h}^{(2)} \circ (1-\boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1-\boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\boldsymbol{W}^\top \left(\nabla_{\boldsymbol{a}^{(3)}} L^{(3)}\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1-\boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1-\boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\boldsymbol{W}^\top \left(\left(\nabla_{\boldsymbol{h}^{(3)}} L^{(3)}\right) \circ \boldsymbol{h}^{(3)} \circ (1-\boldsymbol{h}^{(3)})\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1-\boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1-\boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top} \\
&= \left(\left(\boldsymbol{W}^\top \left(\left(\boldsymbol{W}^\top \left(\left(\boldsymbol{V}^\top \left(\hat{\boldsymbol{y}}^{(3)} - \boldsymbol{y}^{(3)}\right)\right) \circ \boldsymbol{h}^{(3)} \circ (1-\boldsymbol{h}^{(3)})\right)\right) \circ \boldsymbol{h}^{(2)} \circ (1-\boldsymbol{h}^{(2)})\right)\right) \circ \boldsymbol{h}^{(1)} \circ (1-\boldsymbol{h}^{(1)})\right) \boldsymbol{x}^{(1)\top}
\end{aligned}
$$

Taking a closer look

With some notation we can simplify these gradients as follows:

$$\nabla_{\boldsymbol{U}^{(3)}} L^{(3)} = \boldsymbol{c}\boldsymbol{x}^{(3)^{T}}$$

$$\nabla_{\boldsymbol{U}^{(2)}} L^{(3)} = \left(diag(\boldsymbol{b}^{(2)})\boldsymbol{W}^{T}\boldsymbol{c}\right)\boldsymbol{x}^{(2)^{\top}}$$

$$\nabla_{\boldsymbol{U}^{(1)}} L^{(3)} = \left(\left(diag(\boldsymbol{b}^{(1)})\boldsymbol{W}^{T}\right)\left(diag(\boldsymbol{b}^{(2)})\boldsymbol{W}^{T}\right)\boldsymbol{c}\right)\boldsymbol{x}^{(1)^{\top}}$$

Vanishing

If we initialize $\boldsymbol{W}$ such that $||\boldsymbol{W}|| < 1$, the gradient for further time steps will be very small (vanishing problem).

https://www.youtube.com/watch?v=xAl8fu8myW0

Exploding

If $||\boldsymbol{W}|| > 1$, the gradient for further time steps will be larger and larger (exploding problem).

https://www.youtube.com/watch?v=dqW-jw5qKK8

The vanishing problem

The gradients from the steps closed to τ (the last step) have more influence than the ones very far back.

This is bad for capturing long-term dependecies.

Possible solutions (hacks)

- Clip gradients to a maximum value.
- Choosing the right activation functions, e.g. ReLU.
- Initialize weights to the identity matrix.
- LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), etc

Implementation

Truncated Back Propagation

www.tensorflow.org/versions/master/tutorials/recurrent

"By design, the output of a recurrent neural network (RNN) depends on arbitrarily distant inputs. Unfortunately, this makes backpropagation computation difficult. In order to make the learning process tractable, it is common practice to create an 'unrolled' version of the network, which contains a fixed number (num_steps) of LSTM inputs and outputs."

Tensorflow implementation

```python
self.rnn_outputs = []

initialshape = (self.batch_size, self.hidden_size)
Wshape = (self.hidden_size, self.hidden_size)
Ushape = (self.embed_size, self.hidden_size)
Vshape = (self.config.hidden_size, self.vocab_size)

with tf.variable_scope("memory"):
    self.initial_state = tf.zeros(initialshape)

with tf.variable_scope("hidden"):
    self.W = tf.get_variable("W", shape=Wshape)
    self.input_weights = init_wb(Ushape, "input_weights")
```

Tensorflow implementation

```
previous_h = self.initial_state
for i, tensor in enumerate(self.inputs):
          # len(self.inputs) = num_steps
    with tf.variable_scope("RNN", reuse=True):
        drop_tensor = tf.nn.dropout(tensor,
                                    self.dropout_placeholder)
        h = (tf.matmul(previous_h, self.W) +
             affine_transformation(drop_tensor,
                                   self.input_weights))
        h = tf.nn.dropout(tf.sigmoid(h),
                          self.dropout_placeholder)
        self.rnn_outputs.append(h)
        previous_h = h
        if i == (len(self.inputs) - 1):
            self.final_state = h
```

Tensorflow implementation

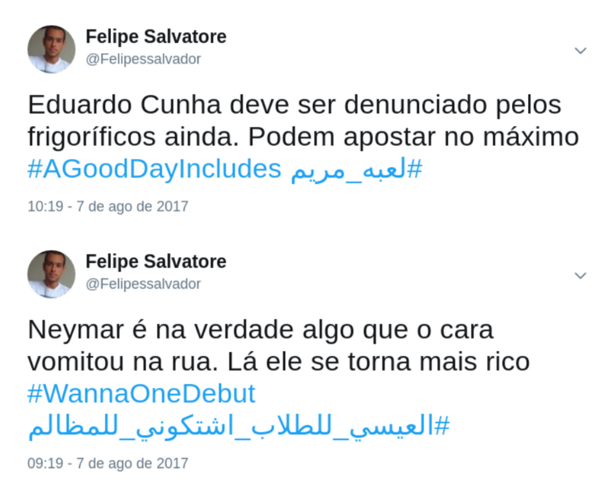
```python
with tf.variable_scope("Projection_layer"):
    self.output_weights = init_wb(Vshape, "output_weights")
    self.logits = [affine_transformation(tensor,
                                         self.output_weights)
                   for tensor in self.rnn_outputs]
```

MytwitterBot: TrumpBot

https://github.com/felipessalvatore/MyTwitterBot

MytwitterBot: SakaBot

https://github.com/felipessalvatore/MyTwitterBot



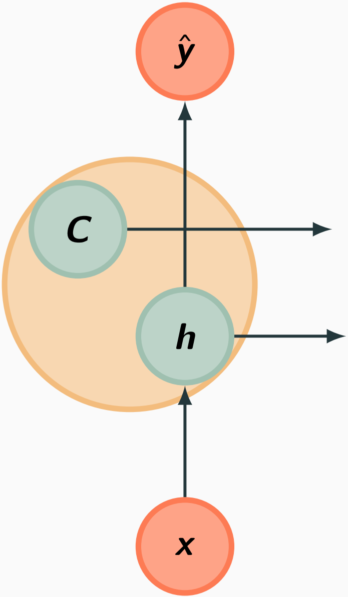
Conclusion

What's next?

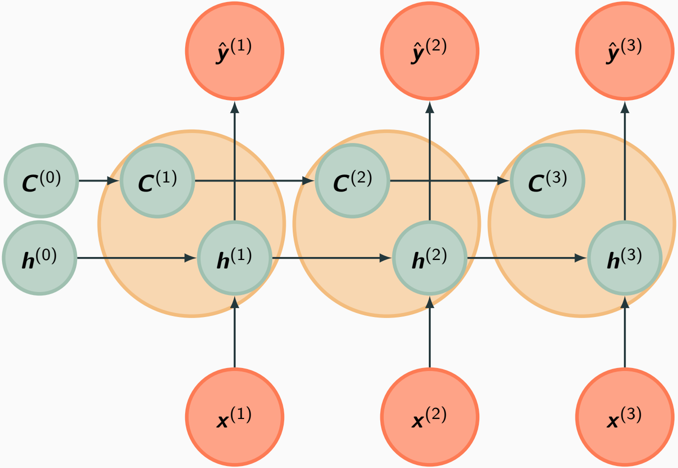
After some experiments with the hyper parameters my best result on the Penn Treebank (PTB) corpus was

Model	Val	Test
Mikolov et al (2011)[2]	163.2	149.9

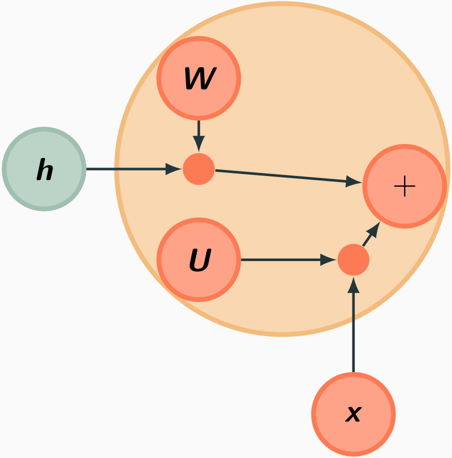
LSTM



LSTM: unfolded model

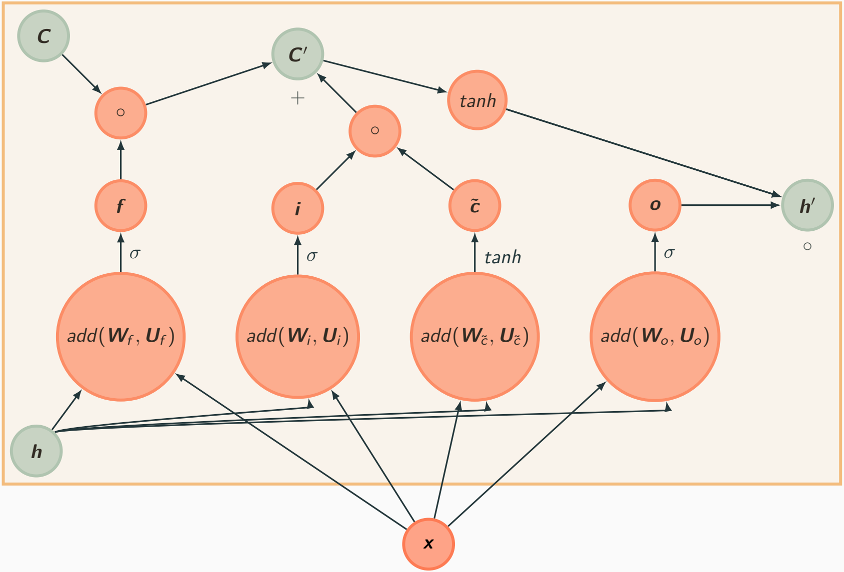


Simplification



$add(\boldsymbol{W}, \boldsymbol{U})$

LSTM: recurrence



LSTM: equations

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{U}_i \boldsymbol{x}_t + \boldsymbol{W}_i \boldsymbol{h}_{t-1}\right)$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{U}_f \boldsymbol{x}_t + \boldsymbol{W}_f \boldsymbol{h}_{t-1}\right)$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{U}_o \boldsymbol{x}_t + \boldsymbol{W}_o \boldsymbol{h}_{t-1}\right)$$

$$\tilde{\boldsymbol{c}}_t = \tanh\left(\boldsymbol{U}_c \boldsymbol{x}_t + \boldsymbol{W}_c \boldsymbol{h}_{t-1}\right)$$

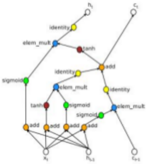
$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tilde{\boldsymbol{c}}_t$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)$$

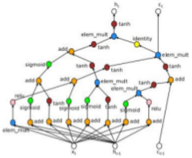
LSTM: changing cell

http://blog.ycombinator.com/jeff-deans-lecture-for-yc-ai/

“Normal” LSTM cell



Cell discovered by architecture search



Penn Tree Bank Language Modeling Task

Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	24M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	$\boxed{66.0}$
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	$\boxed{62.4}$

Table 2: Single model perplexity on the test set of the Penn Treebank language modeling task. Parameter numbers with ‡ are estimates with reference to Merity et al. (2016).

LSTM: https://arxiv.org/abs/1708.02182

Richard
@RichardSocher

Seguindo

When Zoph & Le at Google got 62 perplexity on PTB, I thought it'd be impossible to beat. Amazing progress in AI atm. arxiv.org/abs/1708.02182

Traduzir do inglês

Model results over Penn Treebank (PTB)	Params	Val	Test
Grave et al. (2016) - LSTM	–	–	82.3
Grave et al. (2016) - LSTM + continuous cache pointer	–	–	72.1
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	24M	75.7	73.2
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	51M	71.1	68.5
Zilly et al. (2016) - Variational RHN (tied)	23M	67.9	65.4
Zoph & Le (2016) - NAS Cell (tied)	25M	–	64.0
Zoph & Le (2016) - NAS Cell (tied)	54M	–	62.4
Melis et al. (2017) - 4-layer skip connection LSTM (tied)	24M	60.9	58.3
AWD-LSTM - 3-layer LSTM (tied)	24M	**60.0**	**57.3**
AWD-LSTM - 3-layer LSTM (tied) + continuous cache pointer	24M	**53.9**	**52.8**

01:47 - 8 de ago de 2017

References I

I. Goodfellow, Y. Bengio, and A. Courville.
***Deep Learning*.**
MIT Press, 2017.

T. Mikolov, S. Kombrink, L. Burget, J. Cernocký, and S. Khudanpur.

Extensions of recurrent neural network language.
IEEE, pages 5528–5531, 2011.