

An introduction to Recurrent Neural Networks

Felipe Salvatore

<https://felipessalvatore.github.io/>

Thiago Bueno

<http://thiagopbueno.github.io/>

August 31, 2017

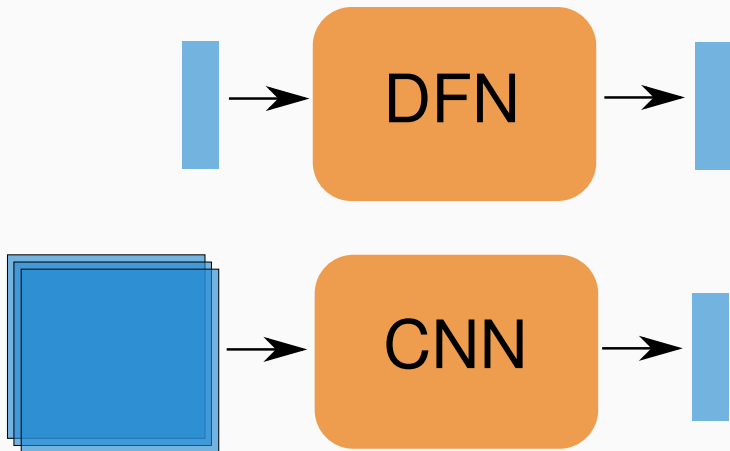
IME-USP: Institute of Mathematics and Statistics, University of São Paulo

Introduction

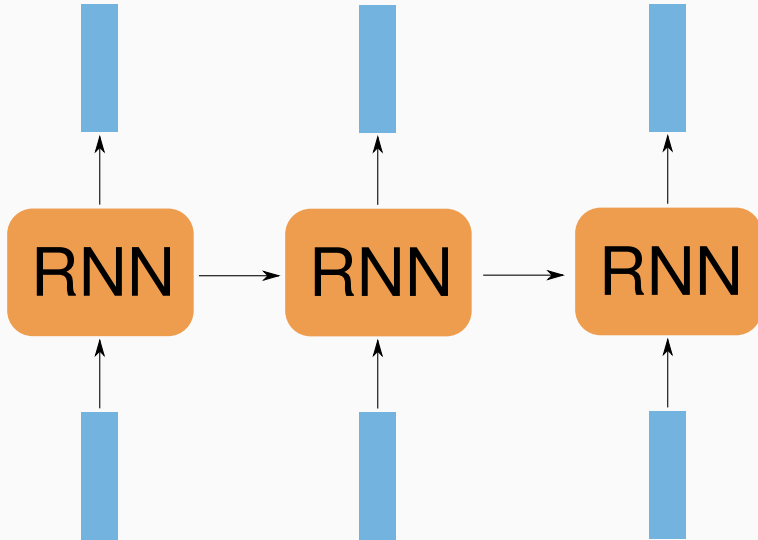
Basic idea

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

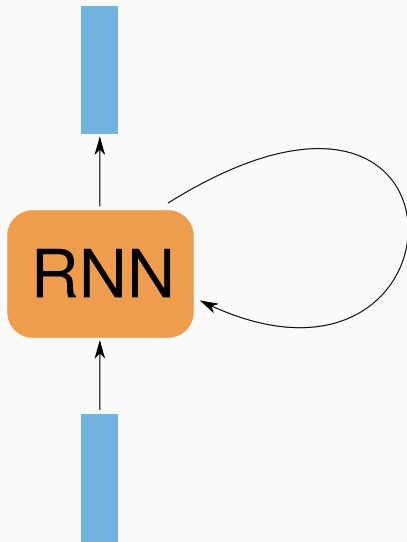
DFN and CNN



RNN: unfold representation

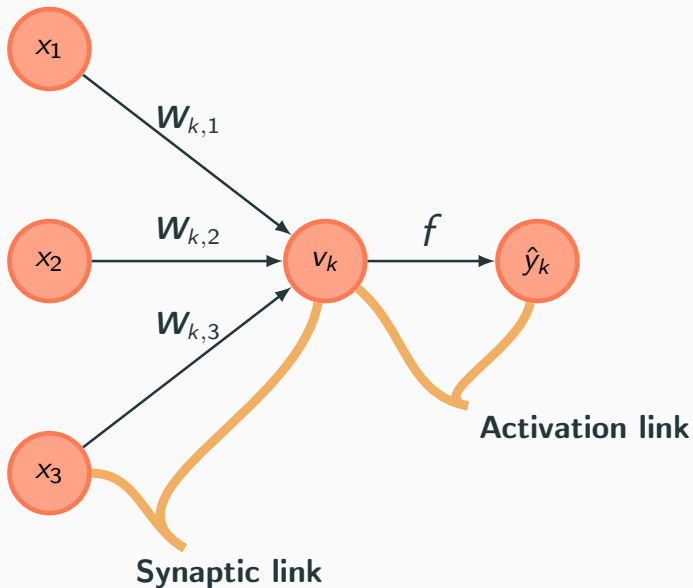


RNN: cyclic representation

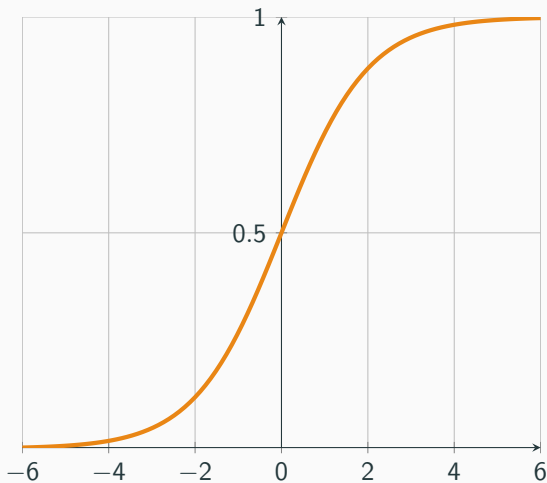


Graph Representation

NN as a directed graph: old version



Recap: sigmoid function



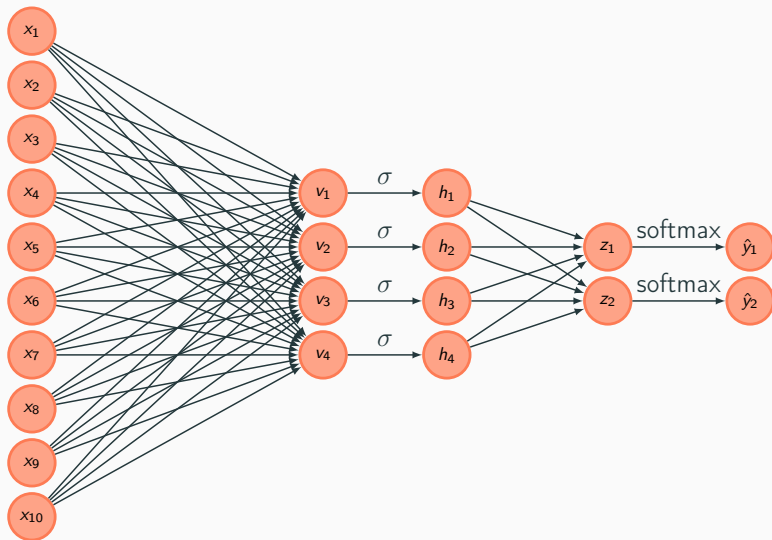
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Recap: softmax function

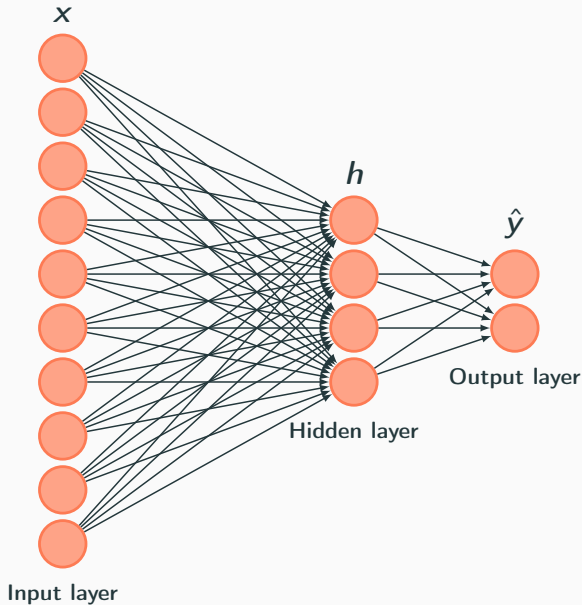
$$\begin{bmatrix} 3.82 \\ 5.35 \\ 1.44 \\ -1.26 \\ 2.71 \\ 1.98 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 0.16115195 \\ 0.74422819 \\ 0.01491471 \\ 0.00100235 \\ 0.05310907 \\ 0.02559374 \end{bmatrix}$$

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

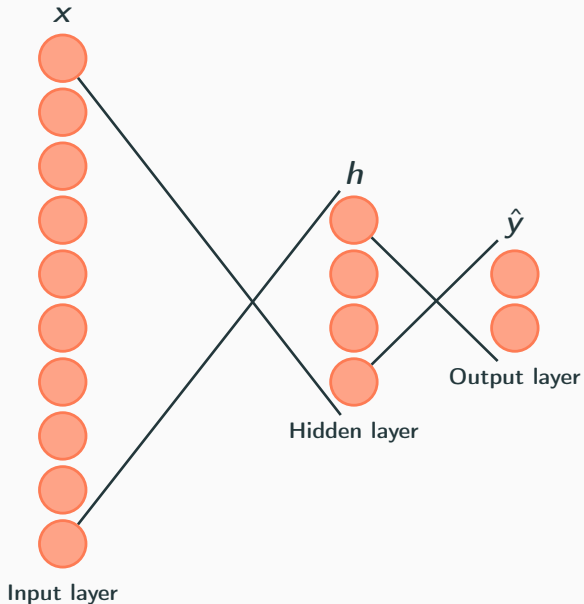
NN as a directed graph: old version



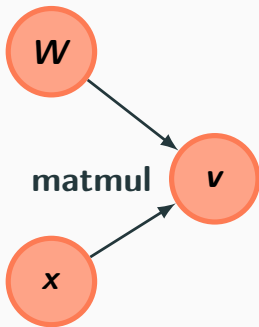
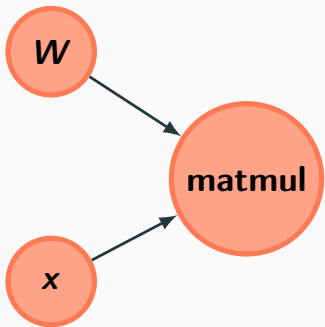
NN as a directed graph: old version



NN as a directed graph: old version

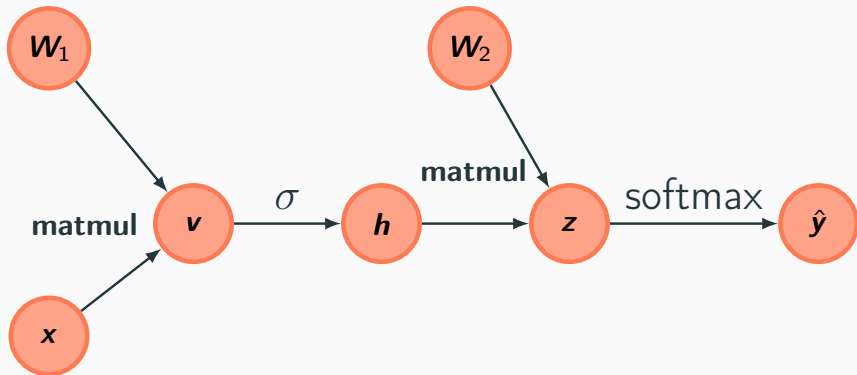


Computational Graphs



$$v = Wx$$

Computational Graphs



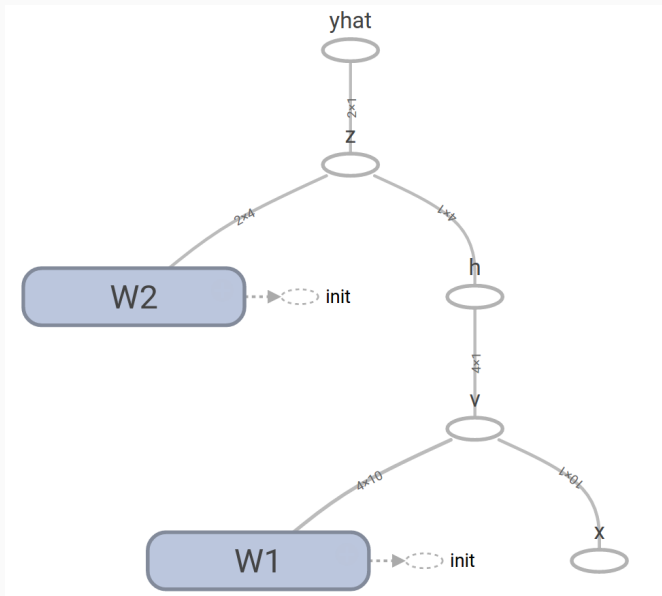
Tensorflow graph

```
1  import tensorflow as tf
2  import numpy as np
3
4  input_shape = [10,1]
5  input_to_hidden_shape = [4,10]
6  hidden_to_output_shape = [2,4]
7
8  W1init = np.zeros(input_to_hidden_shape,
9                    dtype="float32")
10 W2init = np.zeros(hidden_to_output_shape,
11                   dtype="float32")
```

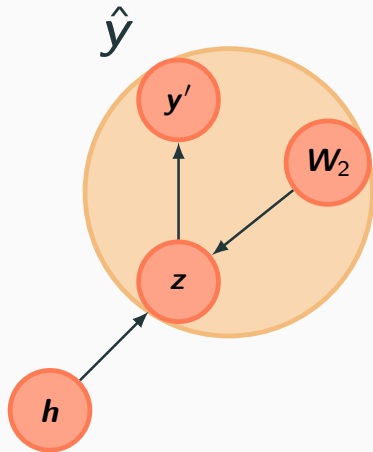
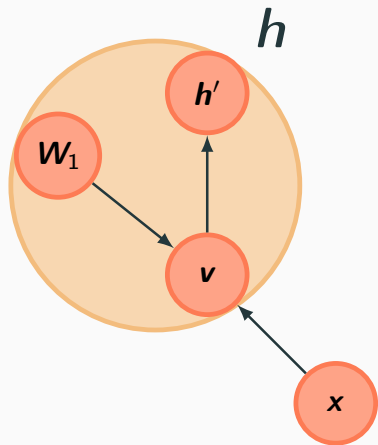

Tensorflow graph

```
1 graph = tf.Graph()
2 with graph.as_default():
3     x = tf.placeholder(shape=input_shape,
4                        dtype="float32")
5     W1 = tf.get_variable(initializer=W1init)
6     v = tf.matmul(W1, x)
7     h = tf.sigmoid(v)
8     W2 = tf.get_variable(initializer=W2init)
9     z = tf.matmul(W2, h)
10    yhat = tf.nn.softmax(z)
```

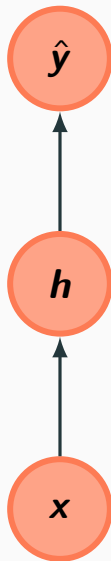
Tensorboard visualization



Computational Graphs



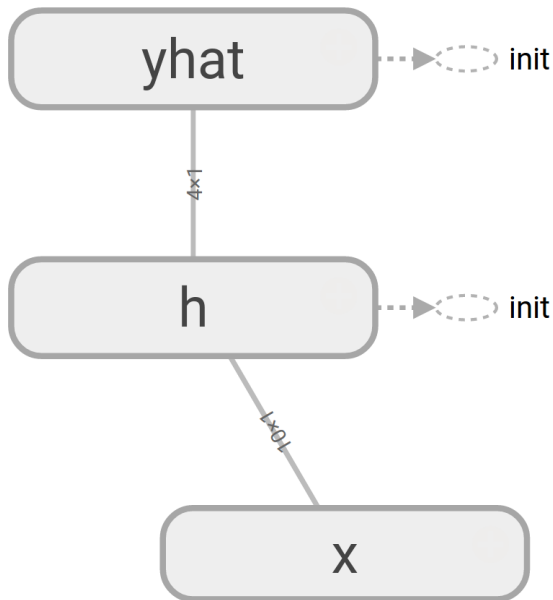
Computational Graphs



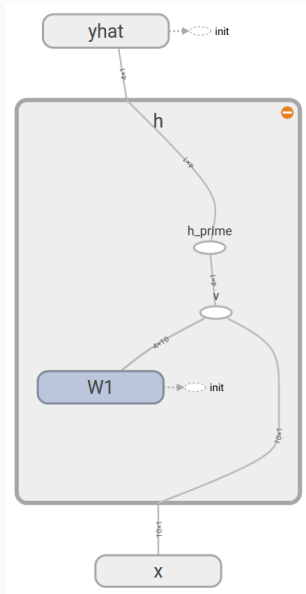
Tensorflow graph

```
1  graph = tf.Graph()
2  with graph.as_default():
3      with tf.variable_scope("x"):
4          x_prime = tf.placeholder(shape=input_shape,
5                                   dtype="float32")
6
7      with tf.variable_scope("h"):
8          W1 = tf.get_variable(initializer=W1init)
9          v = tf.matmul(W1, x_prime)
10         h_prime = tf.sigmoid(v)
11
12     with tf.variable_scope("yhat"):
13         W2 = tf.get_variable(initializer=W2init)
14         z = tf.matmul(W2, h_prime)
15         y_prime = tf.nn.softmax(z)
```

Tensorboard visualization

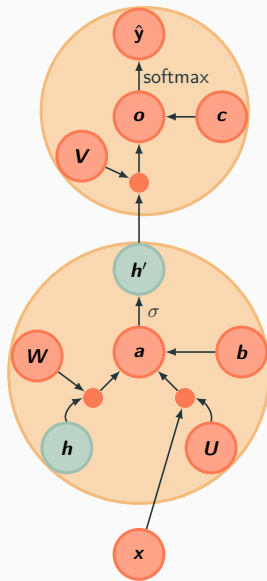


Tensorboard visualization

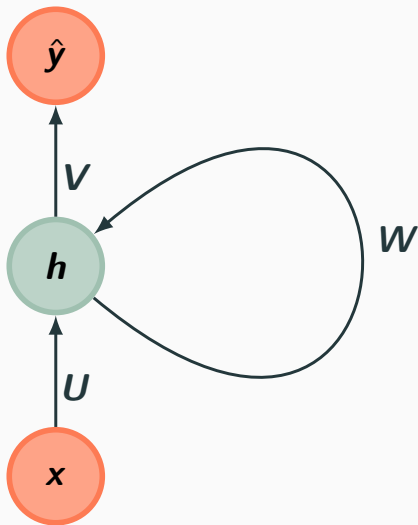


RNN: the model

RNN as a graph



RNN as a graph



Definition

A RNN is a function f with two inputs:

- An input vector \mathbf{x} .
- A hidden vector \mathbf{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:

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

Using our example as a concrete case

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

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

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

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

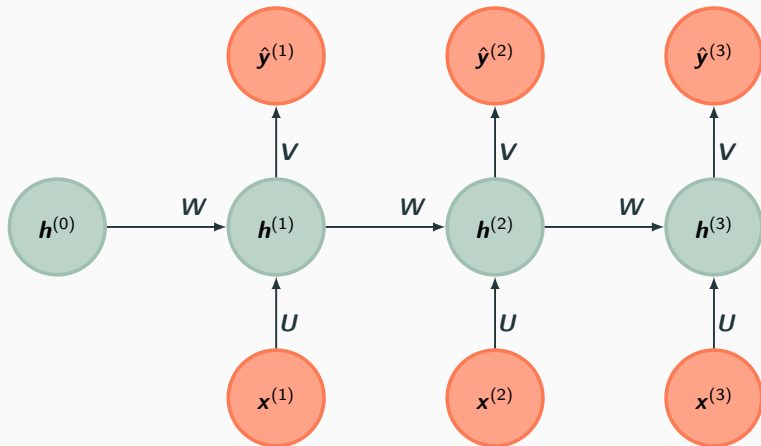
Unfolding the state equation

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

For example when $\tau = 3$:

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

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:

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

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 $P(x_i) = \text{count}(x_i)$.

Bigram language model ($n = 2$):

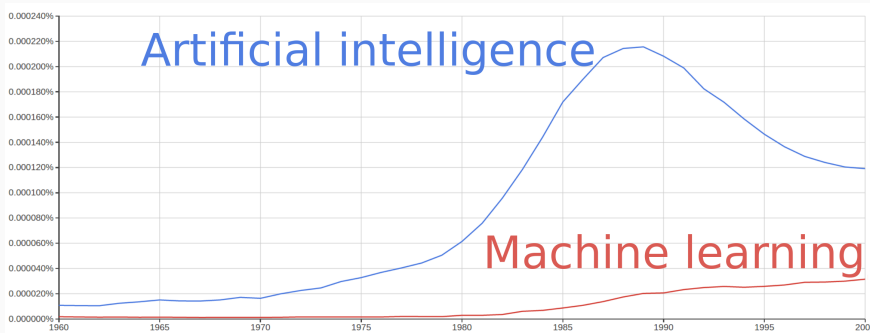
$$P_{bi}(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3)$$

where

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

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

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

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

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

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.

Language 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

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

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

Building the dataset

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

Example: **Meditations in an Emergency** by Frank O'Hara.

Am I to become profligate as if I were a blonde? Or religious as if I were French?

Each time my heart is broken it makes me feel more adventurous ...

- $T = 645$
- $|\mathbb{V}| = 347$

Building the dataset

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

$$(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}) = (\text{Am}, \text{I}).$$

$$(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}) = (\text{I}, \text{to})$$

$$(\mathbf{x}^{(3)}, \mathbf{y}^{(3)}) = (\text{to}, \text{become})$$

$$(\mathbf{x}^{(3)}, \mathbf{y}^{(3)}) = (\text{become}, \text{profligate})$$

$$(\mathbf{x}^{(4)}, \mathbf{y}^{(4)}) = (\text{profligate}, \text{as})$$

$$(\mathbf{x}^{(5)}, \mathbf{y}^{(5)}) = (\text{as}, \text{if})$$

$$(\mathbf{x}^{(6)}, \mathbf{y}^{(6)}) = (\text{if}, \text{I})$$

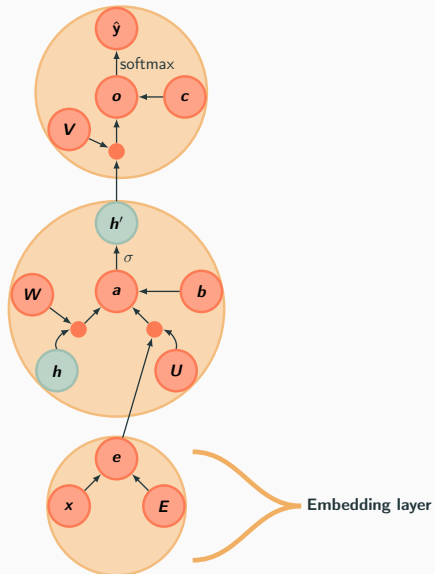
...

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

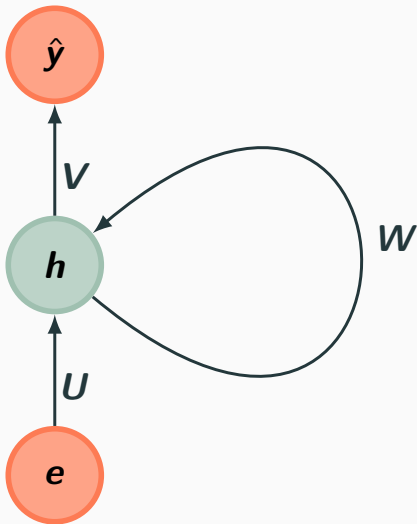
Recap: selecting word embeddings

$$e = E \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \\ = E_{:,j}$$

The language model: graph



The language model: graph



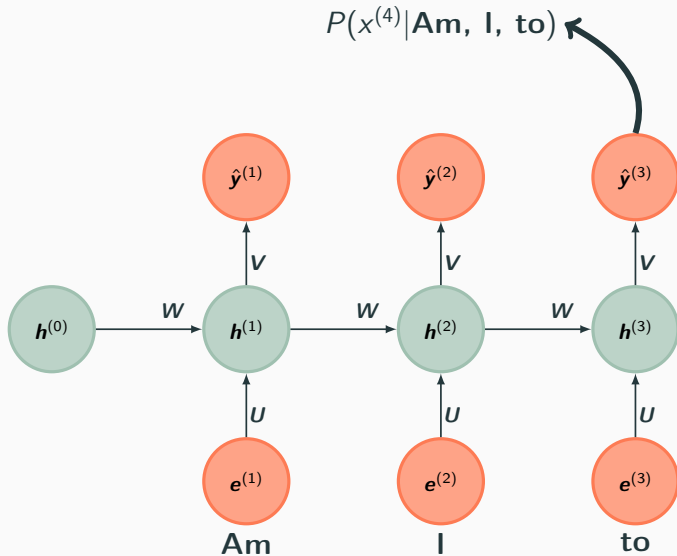
The language model: equations

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

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

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

The Language model: unfolding example



$$CE(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_i y_i \log(\hat{y}_i)$$

Loss function

At each time t the point-wise loss is:

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

For example:

$$L^{(3)} = -\log P(x^{(4)} = \text{become} | \text{Am, I, to})$$

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 poem as C :

Am I to become profligate as if I were a blonde?

- $T = 12$
- $|\mathbb{V}| = 11$

Loss function: example

$$\begin{aligned} L = & -\frac{1}{12} [\log P(\text{Am} | < \text{eos} >) \\ & + \log P(\text{I} | \text{Am}) \\ & + \log P(\text{to} | \text{Am}, \text{I}) \\ & + \log P(\text{become} | \text{Am}, \text{I}, \text{to}) \\ & + \log P(\text{profligate} | \text{Am}, \text{I}, \text{to}, \text{become}) \\ & + \log P(\text{as} | \text{Am}, \text{I}, \text{to}, \text{become}, \text{profligate}) \\ & + \log P(\text{if} | \text{Am}, \text{I}, \text{to}, \text{become}, \text{profligate}, \text{as}) \\ & + \log P(\text{I} | \text{Am}, \text{I}, \text{to}, \text{become}, \text{profligate}, \text{as}, \text{if}) \\ & + \log P(\text{were} | \text{Am}, \text{I}, \text{to}, \text{become}, \text{profligate}, \text{as}, \text{if}, \text{I}) \\ & + \log P(\text{a} | \text{Am}, \text{I}, \text{to}, \text{become}, \text{profligate}, \text{as}, \text{if}, \text{I}, \text{were}) \\ & + \log P(\text{blonde} | \text{Am}, \text{I}, \text{to}, \text{become}, \text{profligate}, \text{as}, \text{if}, \text{I}, \text{were}, \text{a}) \\ & + \log P(< \text{eos} > | \text{Am}, \text{I}, \text{to}, \text{become}, \text{profligate}, \text{as}, \text{if}, \text{I}, \text{were}, \text{a}, \text{blonde})] \end{aligned}$$

Loss and Perplexity

Since

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

We have that:

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

So another definition of perplexity is

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

Back Propagation

Chain rule of Calculus

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

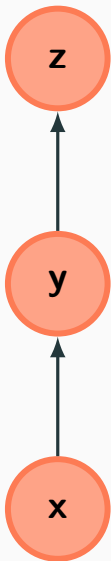
$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Chain rule: example

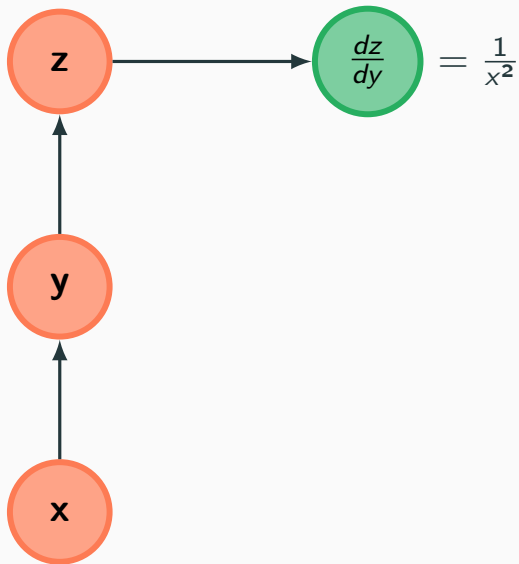
- $y = x^2$
- $z = \log(y)$

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

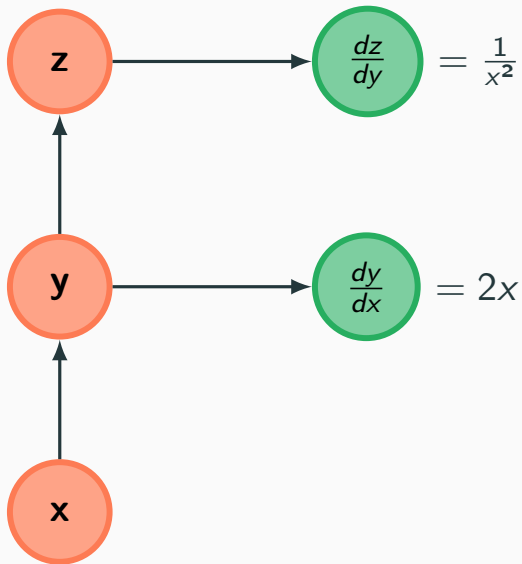
Chain rule: graph



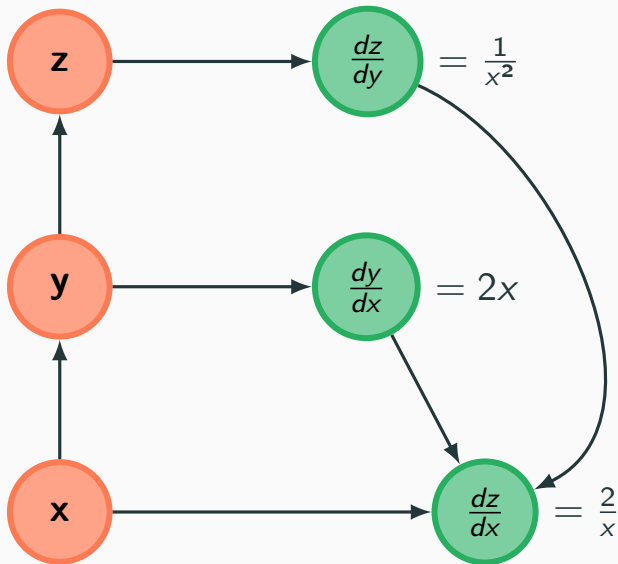
Chain rule: graph



Chain rule: graph



Chain rule: graph



Chain rule: vector notation

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

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

Chain rule: vector notation

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

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

Recap: gradient

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

Recap: Jacobian matrix

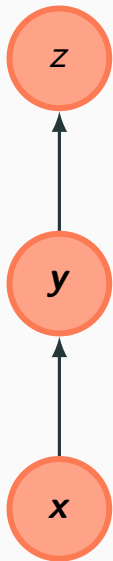
$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \dots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \cdots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

Chain rule: example

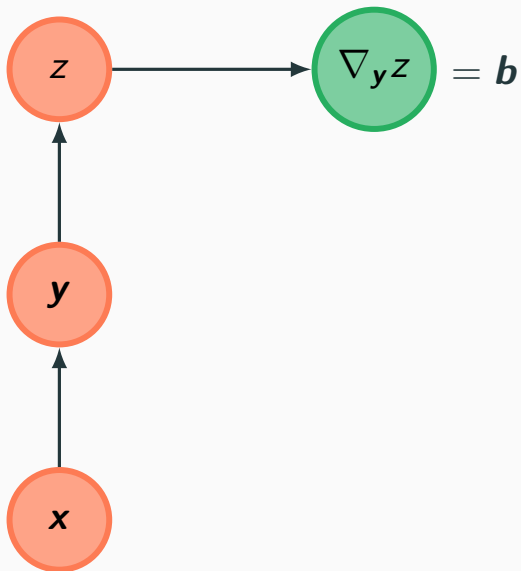
- $y = Wx$
- $z = b^T y$

$$\nabla_x z = W^T b$$

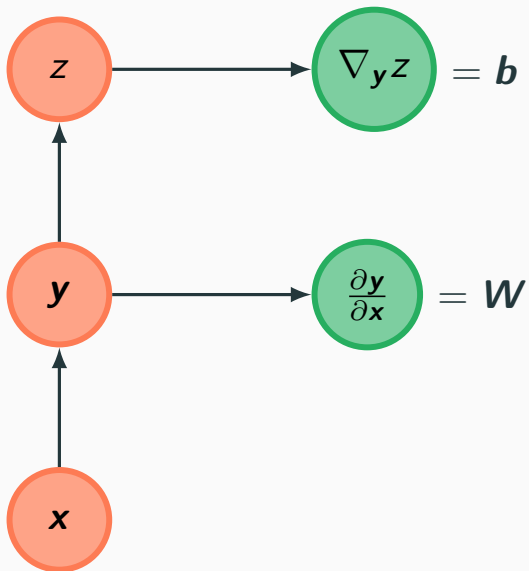
Chain rule: graph



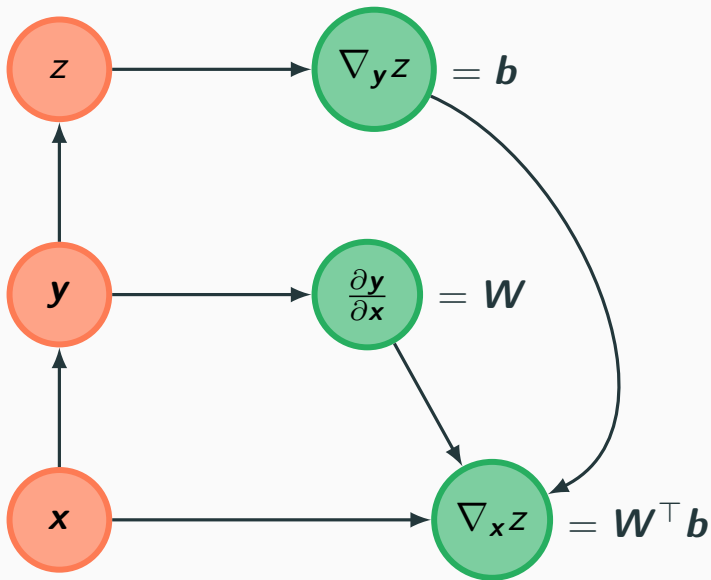
Chain rule: graph



Chain rule: graph



Chain rule: graph



Computing the gradient in a RNN

- We simply 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

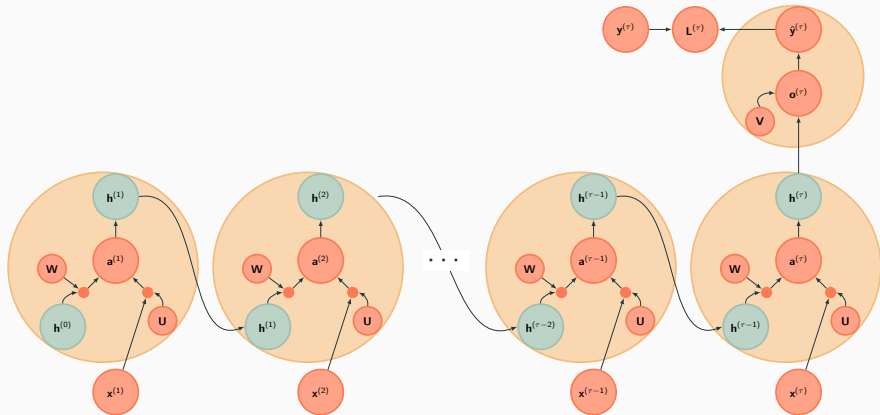
$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$$

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

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

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

RNN time unfolding

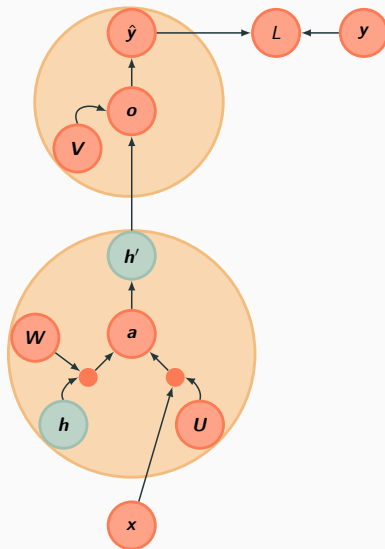


Recap: Hadamard product

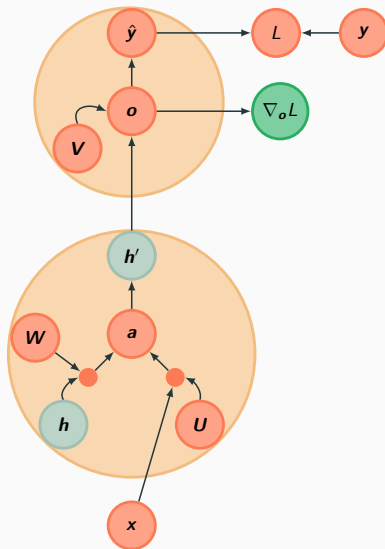
$$(\mathbf{x} \circ \mathbf{y})_i = x_i y_i$$

$$(\mathbf{x} \circ \mathbf{y}) = \text{diag}(\mathbf{y})\mathbf{x}$$

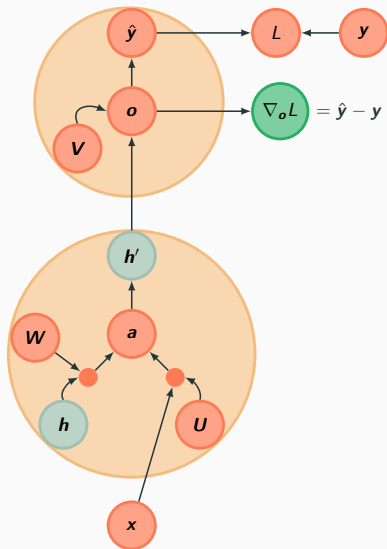
Chain rule: RNN graph



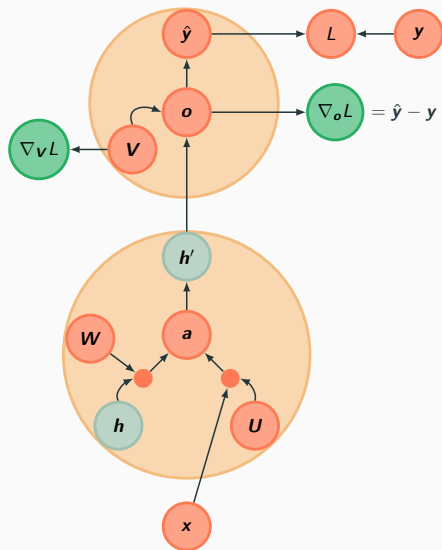
Chain rule: RNN graph



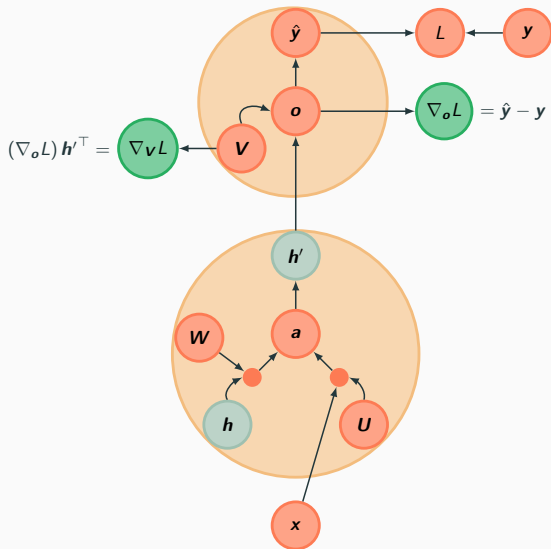
Chain rule: RNN graph



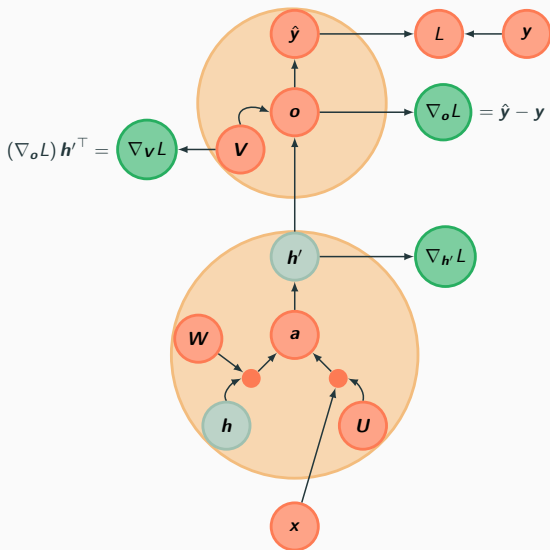
Chain rule: RNN graph



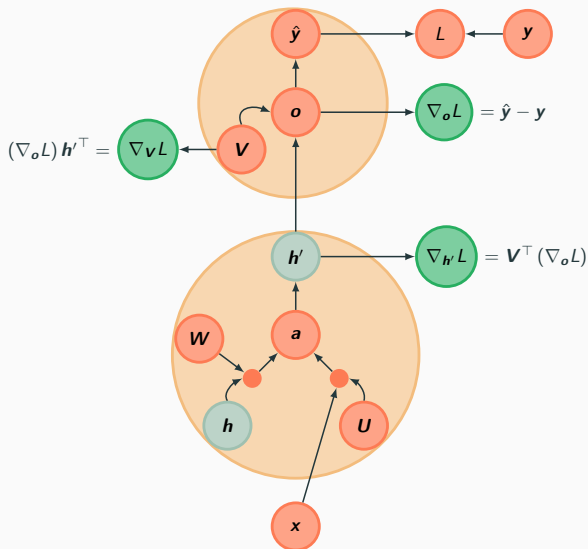
Chain rule: RNN graph



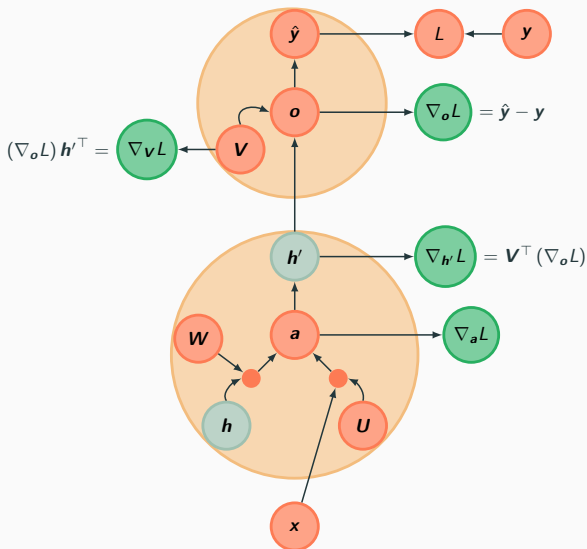
Chain rule: RNN graph



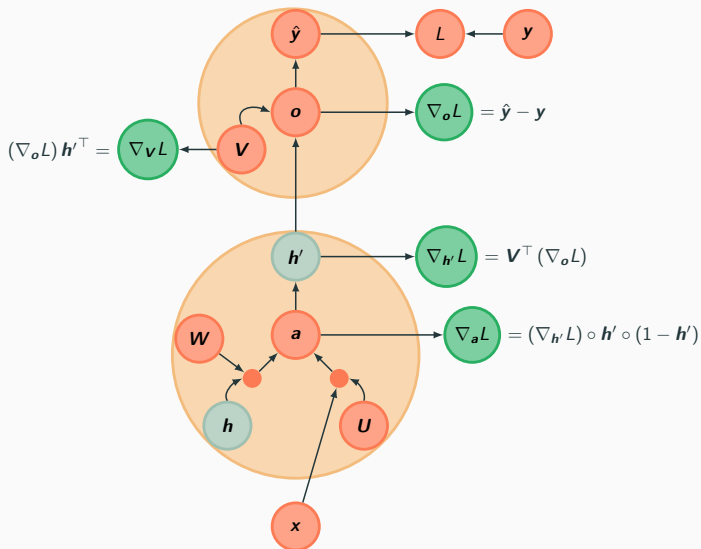
Chain rule: RNN graph



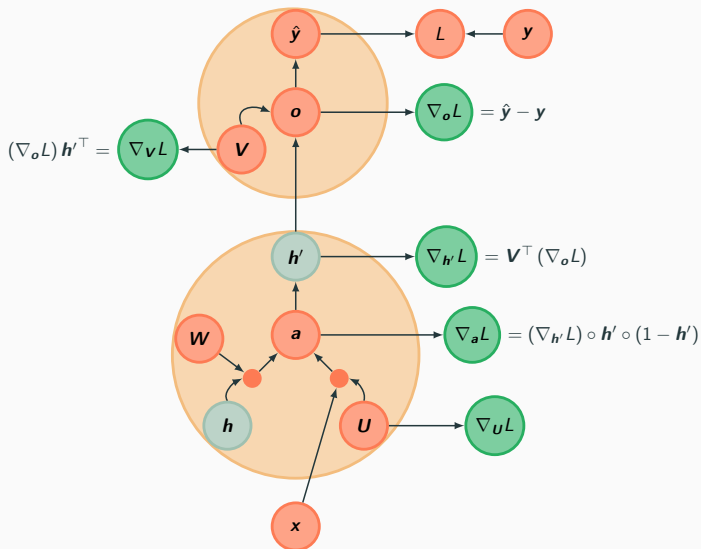
Chain rule: RNN graph



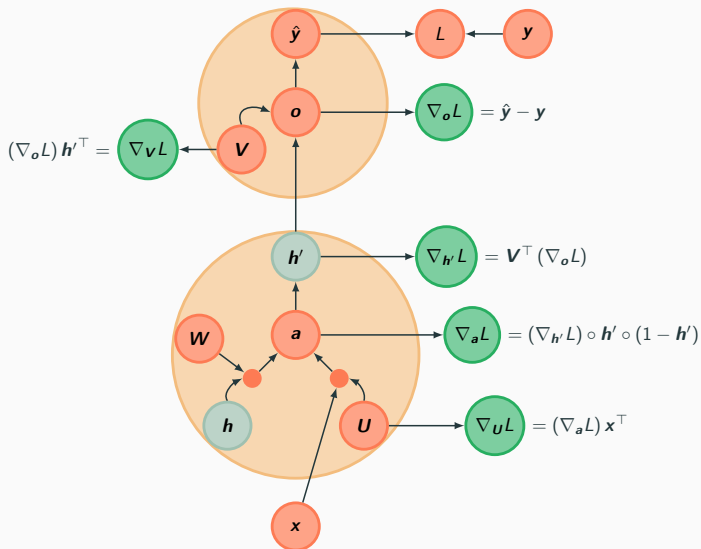
Chain rule: RNN graph



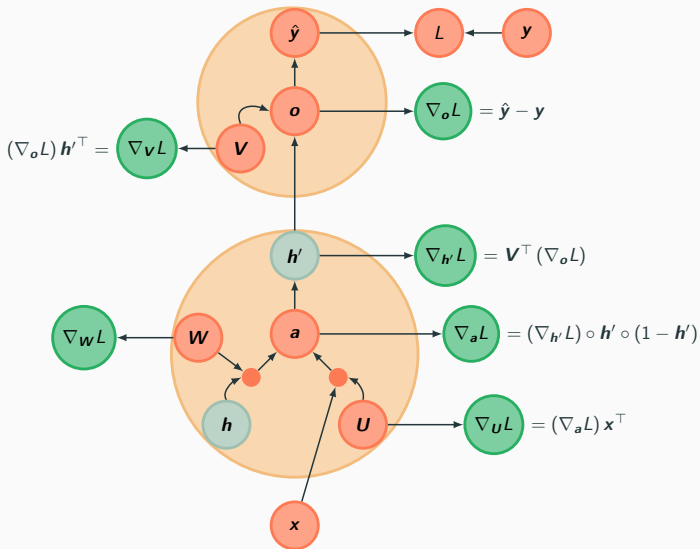
Chain rule: RNN graph



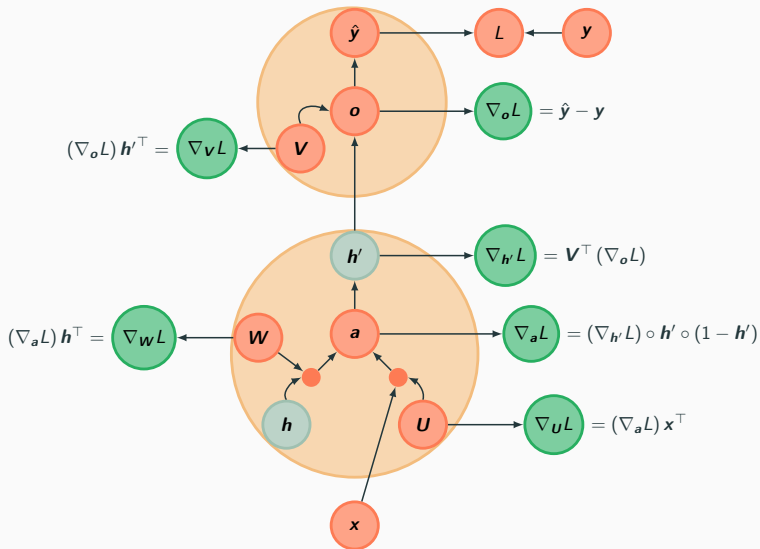
Chain rule: RNN graph



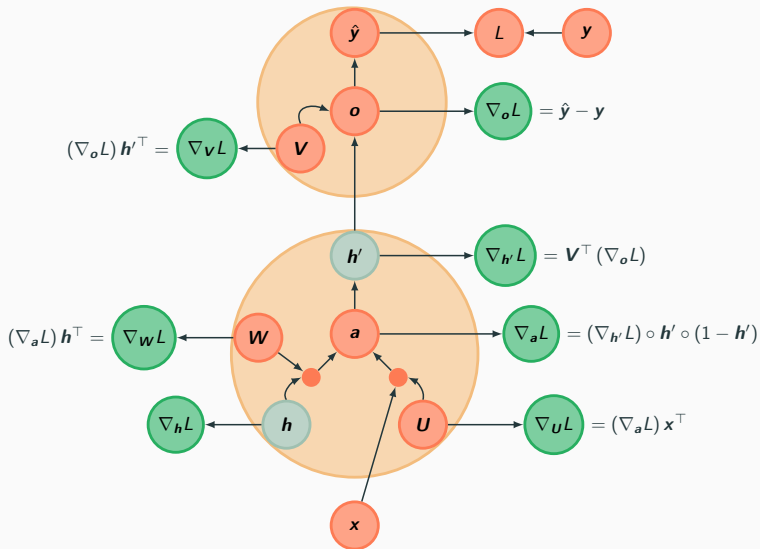
Chain rule: RNN graph



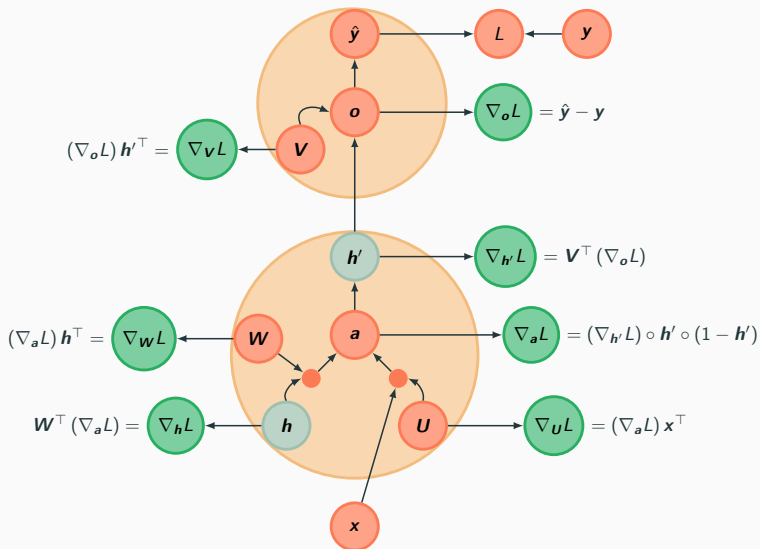
Chain rule: RNN graph



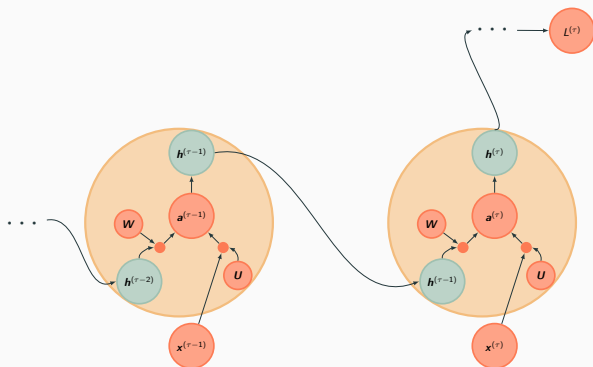
Chain rule: RNN graph



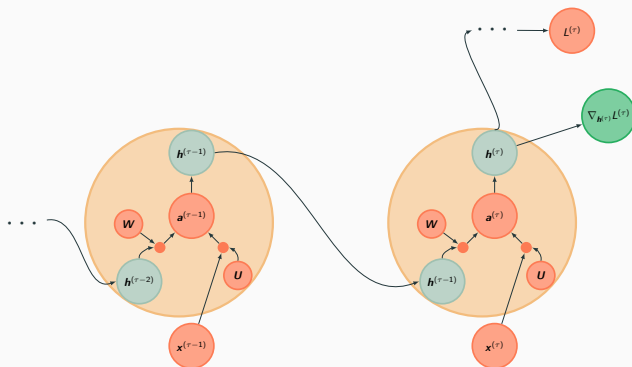
Chain rule: RNN graph



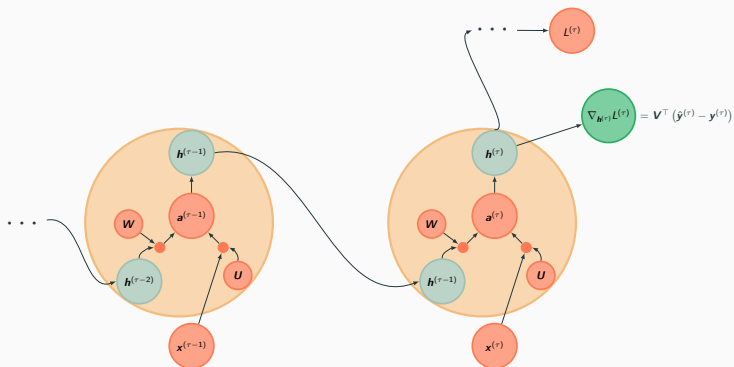
Back Propagation Through Time



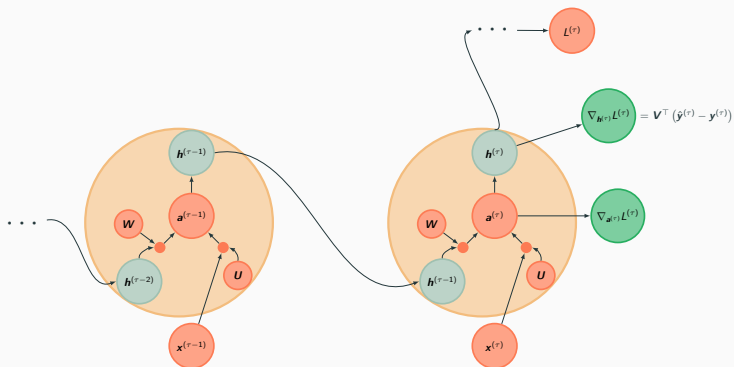
Back Propagation Through Time



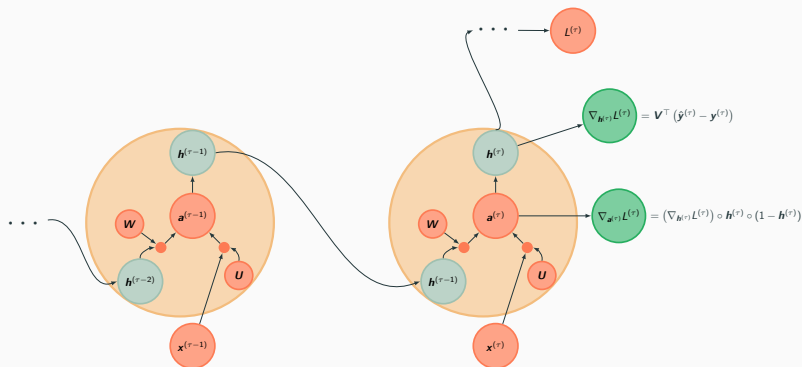
Back Propagation Through Time



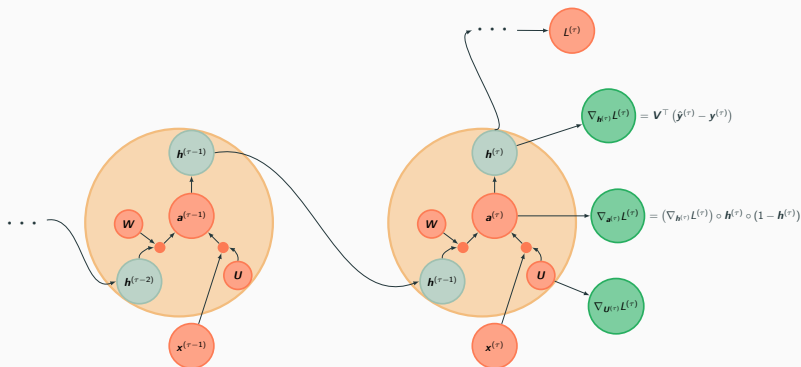
Back Propagation Through Time



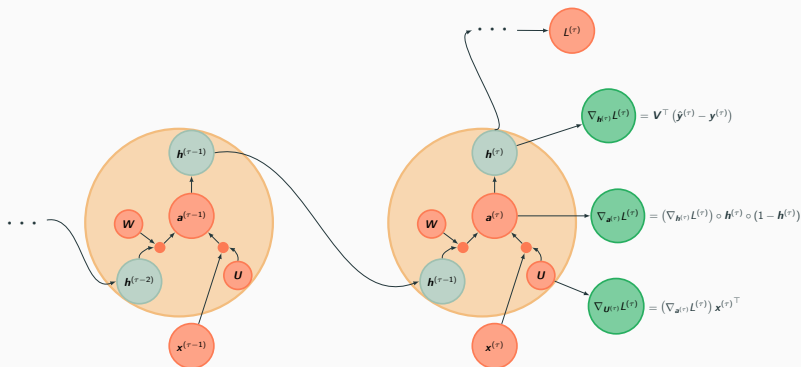
Back Propagation Through Time



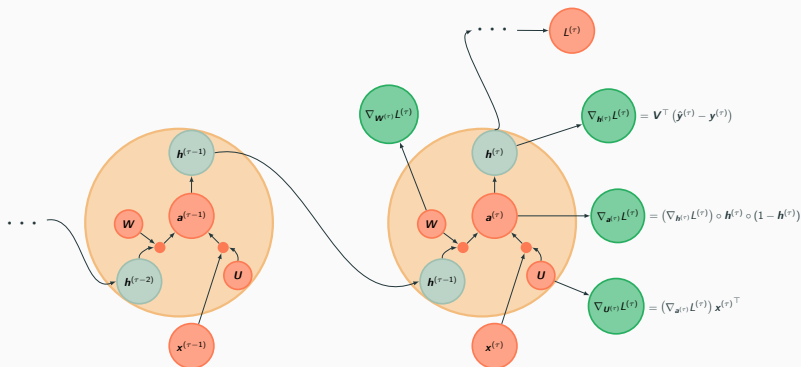
Back Propagation Through Time



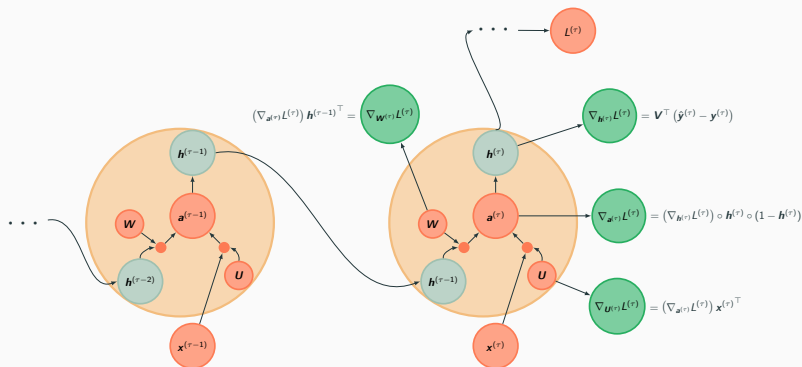
Back Propagation Through Time



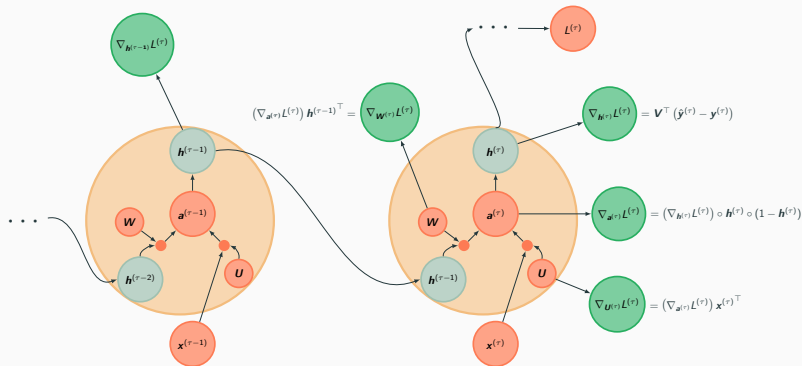
Back Propagation Through Time



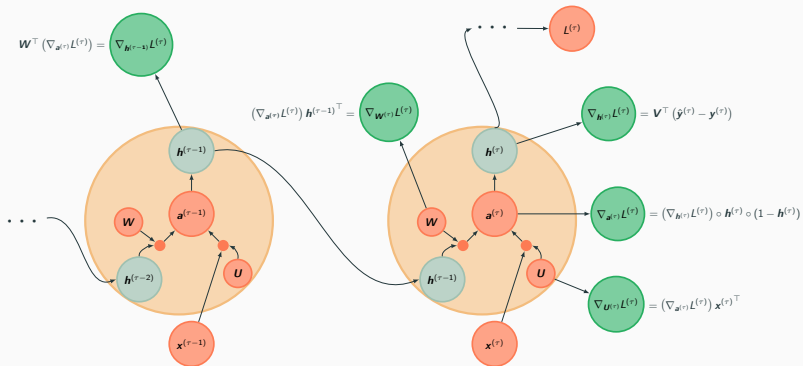
Back Propagation Through Time



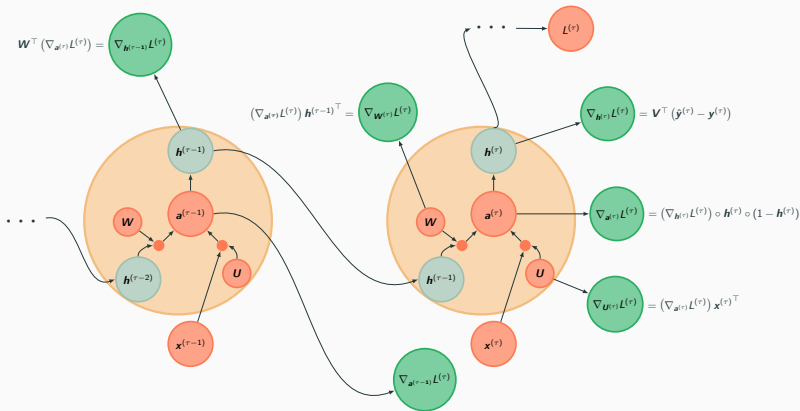
Back Propagation Through Time



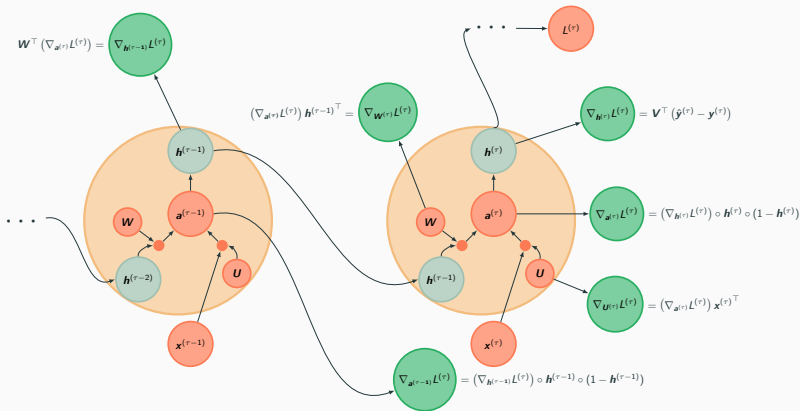
Back Propagation Through Time



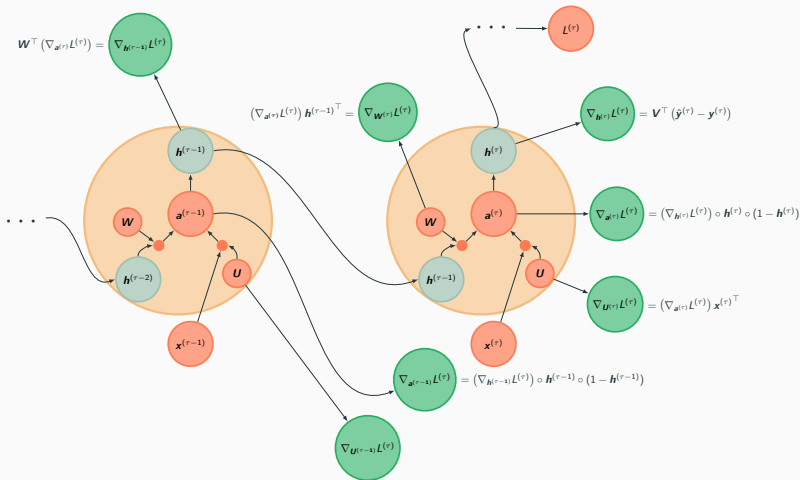
Back Propagation Through Time



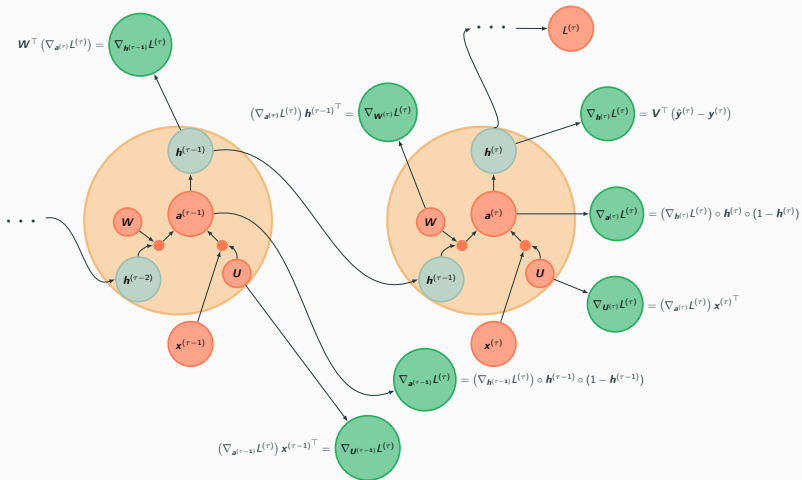
Back Propagation Through Time



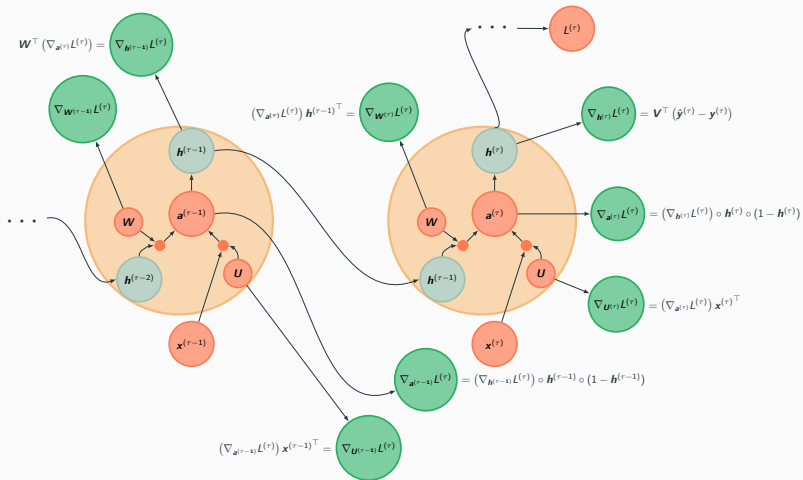
Back Propagation Through Time



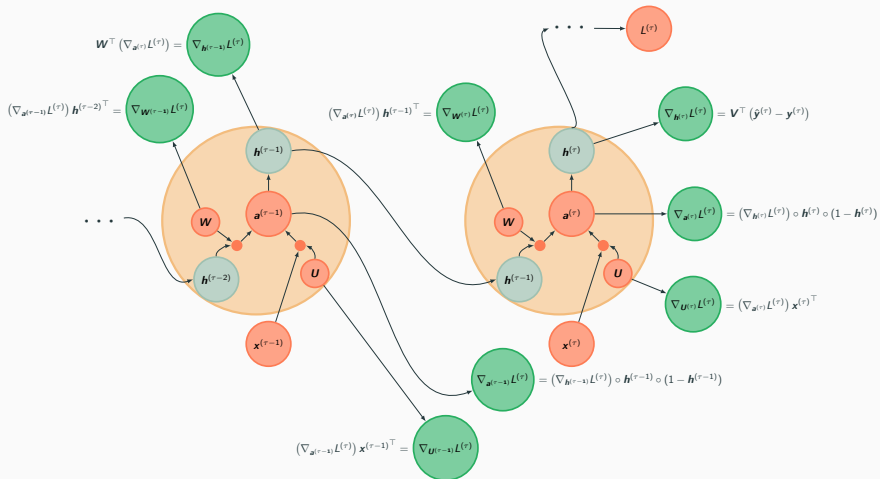
Back Propagation Through Time



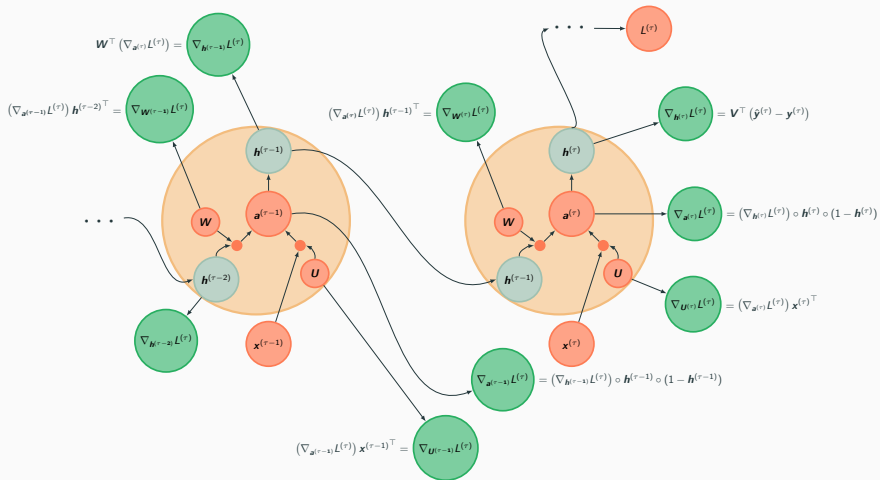
Back Propagation Through Time



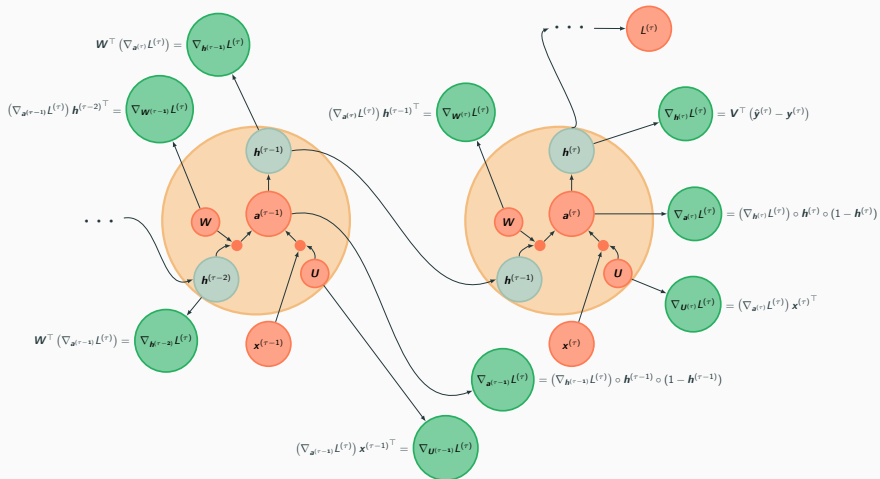
Back Propagation Through Time



Back Propagation Through Time



Back Propagation Through Time



Back Propagation Through Time

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

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

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

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

Vanishing or Exploding Gradient

The problem of training RNN

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

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

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

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

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

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

Calculating $\nabla_{\mathbf{u}^{(3)}} L^{(3)}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Taking a closer look

With some notation we can simplify these gradients as follows:

$$\nabla_{\mathbf{U}^{(3)}} L^{(3)} = \mathbf{c} \mathbf{x}^{(3)\top}$$

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

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

Taking a closer look

$$\begin{aligned} & \nabla_{\mathbf{U}^{(1)}} L^{(\tau)} \\ &= \\ & \left(\left(\text{diag}(\mathbf{b}^{(1)}) \mathbf{W}^T \right) \left(\text{diag}(\mathbf{b}^{(2)}) \mathbf{W}^T \right) \dots \left(\text{diag}(\mathbf{b}^{(\tau-1)}) \mathbf{W}^T \right) \mathbf{c} \right) \mathbf{x}^{(1)\top} \end{aligned}$$

Gradient for further time steps will have more matrix products using $\text{diag}(\mathbf{b}) \mathbf{W}^T$.

Vanishing

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

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

Exploding

If $\|W\| > 1$, 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 dependencies.

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

```
1  self.rnn_outputs = []
2
3  initialshape = (self.batch_size, self.hidden_size)
4  Wshape = (self.hidden_size, self.hidden_size)
5  Ushape = (self.embed_size, self.hidden_size)
6  Vshape = (self.config.hidden_size, self.vocab_size)
7
8  with tf.variable_scope("memory"):
9      self.initial_state = tf.zeros(initialshape)
10
11  with tf.variable_scope("hidden"):
12      self.W = tf.get_variable("W", shape=Wshape)
13      self.input_weights = init_wb(Ushape, "input_weights")
```

Tensorflow implementation

```
1  previous_h = self.initial_state
2  for i, tensor in enumerate(self.inputs):
3      # len(self.inputs) = num_steps
4      with tf.variable_scope("RNN", reuse=True):
5          drop_tensor = tf.nn.dropout(tensor,
6                                      self.dropout_placeholder)
7          h = (tf.matmul(previous_h, self.W) +
8              affine_transformation(drop_tensor,
9                                   self.input_weights))
10         h = tf.nn.dropout(tf.sigmoid(h),
11                           self.dropout_placeholder)
12         self.rnn_outputs.append(h)
13         previous_h = h
14         if i == (len(self.inputs) - 1):
15             self.final_state = h
```

Tensorflow implementation

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


MyTwitterBot: TrumpBot

<https://github.com/felipessalvatore/MyTwitterBot>



Felipe Salvatore

@Felipessalvador

Hillary can make america great again.

[@greta](#) [@MarkBurnettTV](#)

[#DinheiroNãoCompra](#) [#SecretBallot](#)

[#خسوف_القمر](#)

Traduzir do inglês

15:10 - 7 de ago de 2017



Felipe Salvatore

@Felipessalvador

Obama is all beautiful. I agree with people attacking me. Amazing. [@CLewandowski_](#)

[#SecretBallot](#) [@garyplayer](#) [@greta](#)

Traduzir do inglês

14:40 - 7 de ago de 2017

MytwitterBot: SakaBot

<https://github.com/felipessalvatore/MyTwitterBot>



Felipe Salvatore

@Felipessalvador



Eduardo Cunha deve ser denunciado pelos frigoríficos ainda. Podem apostar no máximo
[#AGoodDayIncludes](#) [#لعبه_مریم](#)

10:19 - 7 de ago de 2017



Felipe Salvatore

@Felipessalvador



Neymar é na verdade algo que o cara vomitou na rua. Lá ele se torna mais rico
[#WannaOneDebut](#)
[#العيسي_للطلاب_اشتكوني_للمظالم](#)

09:19 - 7 de ago de 2017

Conclusion

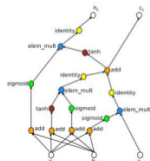
What's next?

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

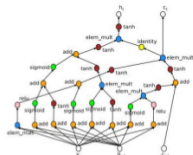
Model	Val	Test
Mikolov et al (2011)[2]	163.2	149.9

<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	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	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).



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



I. G. Y. B. A. Courville.

Deep Learning.

MIT Press, 2017.



T. M. S. K. L. B. J. C. S. Khudanpur.

Extensions of recurrent neural network language.

IEEE, pages 5528–5531, 2011.