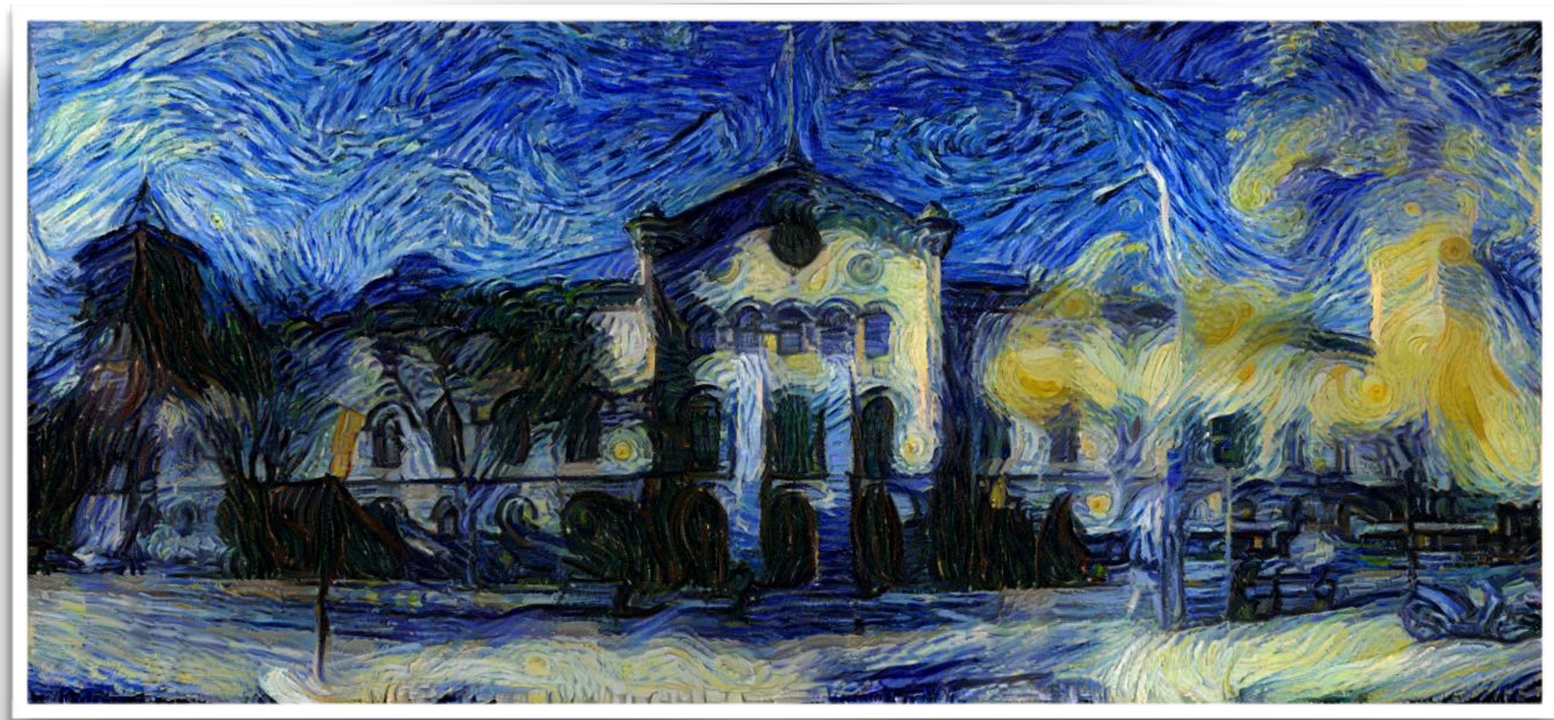
UNIVERSITAT DE BARCELONA

DATASCIENCE@UB



Deep Learning From Scratch

Recurrent Neural Networks

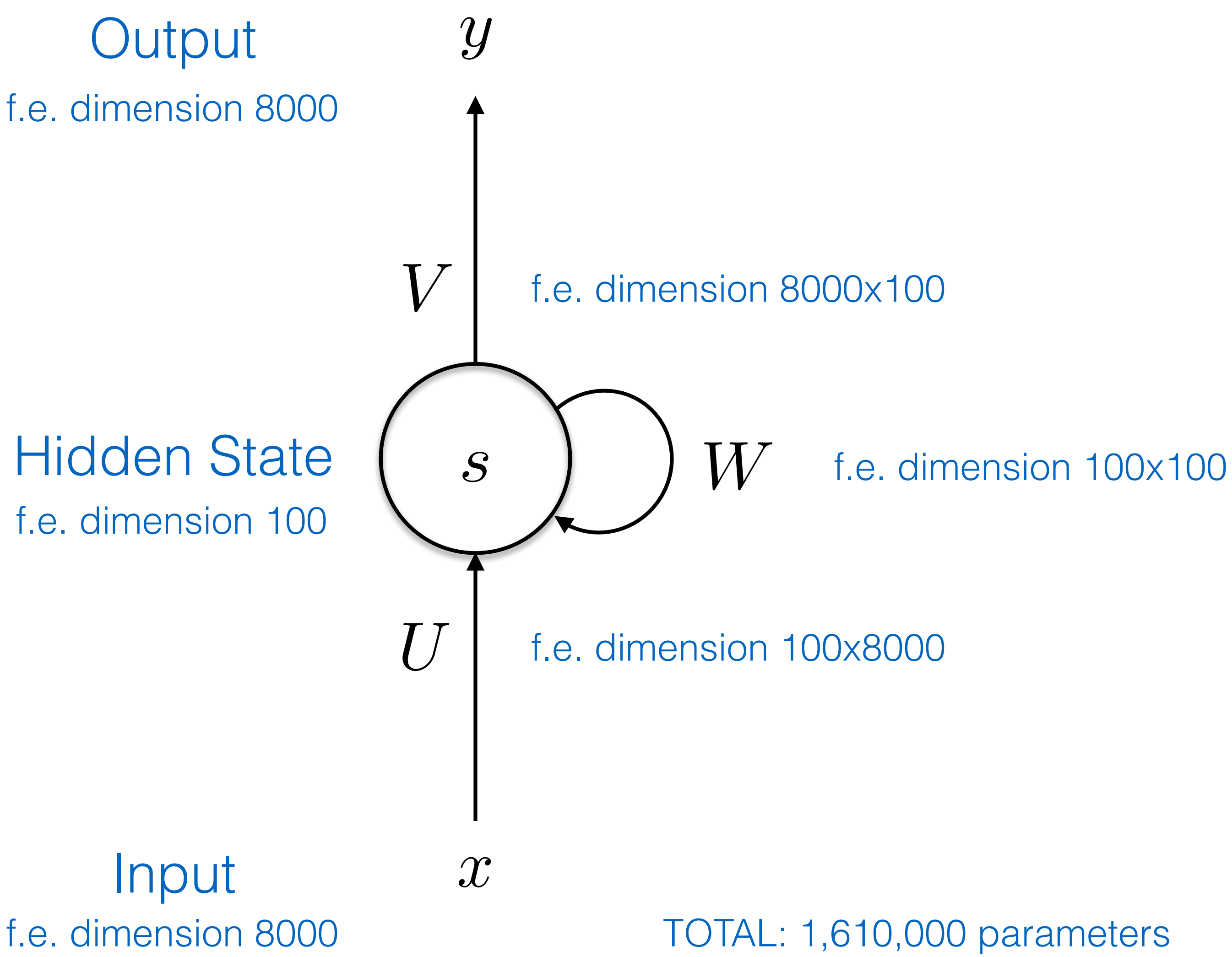
Jordi Vitrià

Classical neural networks, including convolutional ones, suffer from two severe limitations:

- They only accept a fixed-sized vector as input and produce a fixed-sized vector as output.
- They do not consider the sequential nature of some data (language, video frames, time series, etc.)

Recurrent neural networks overcome these limitations by allowing to operate over sequences of vectors (in the input, in the output, or both).

Vanilla Recurrent Neural Network

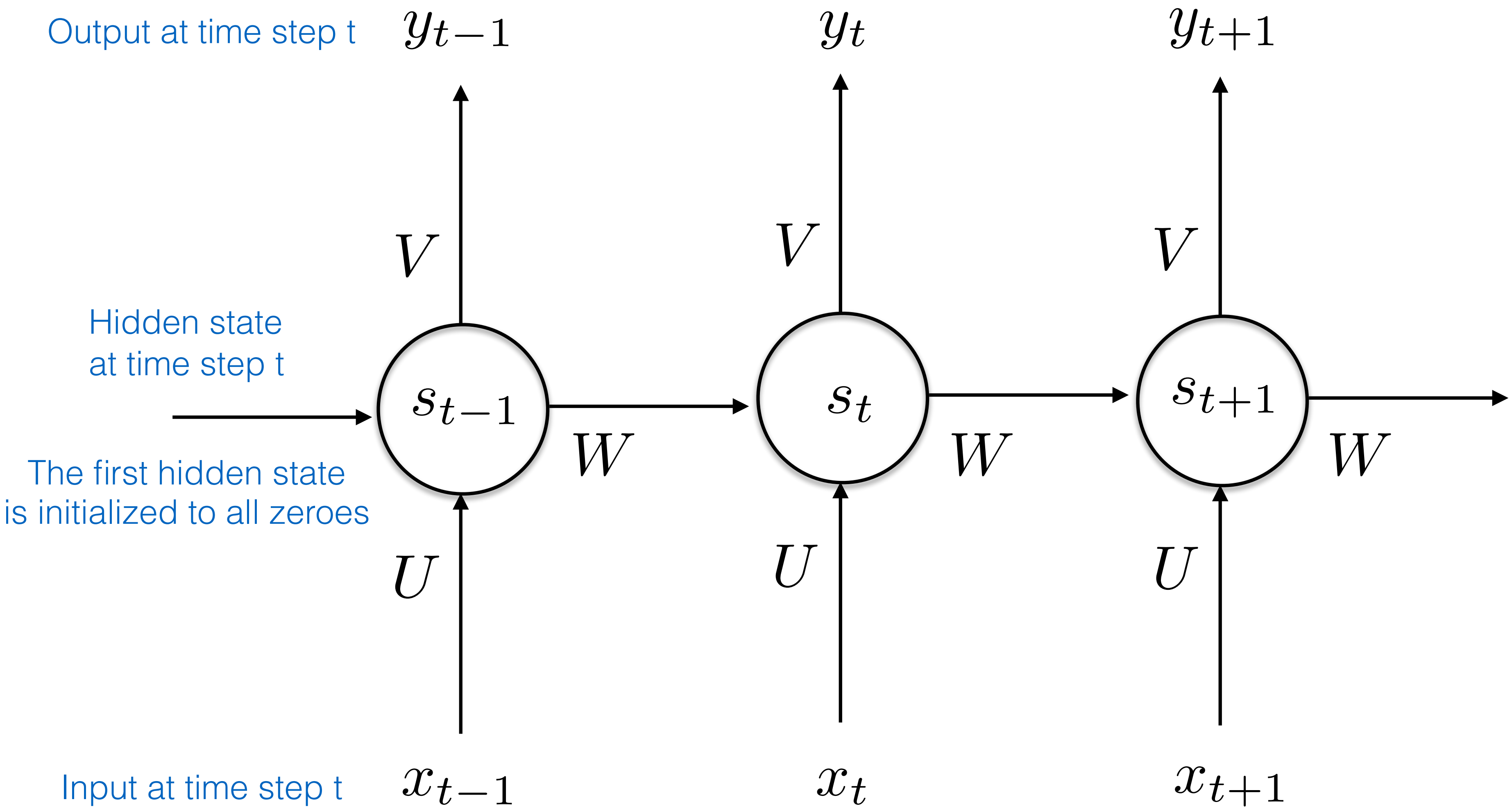


Unfolding in time of a RNN

By unrolling we mean that we write out the network for the complete sequence.

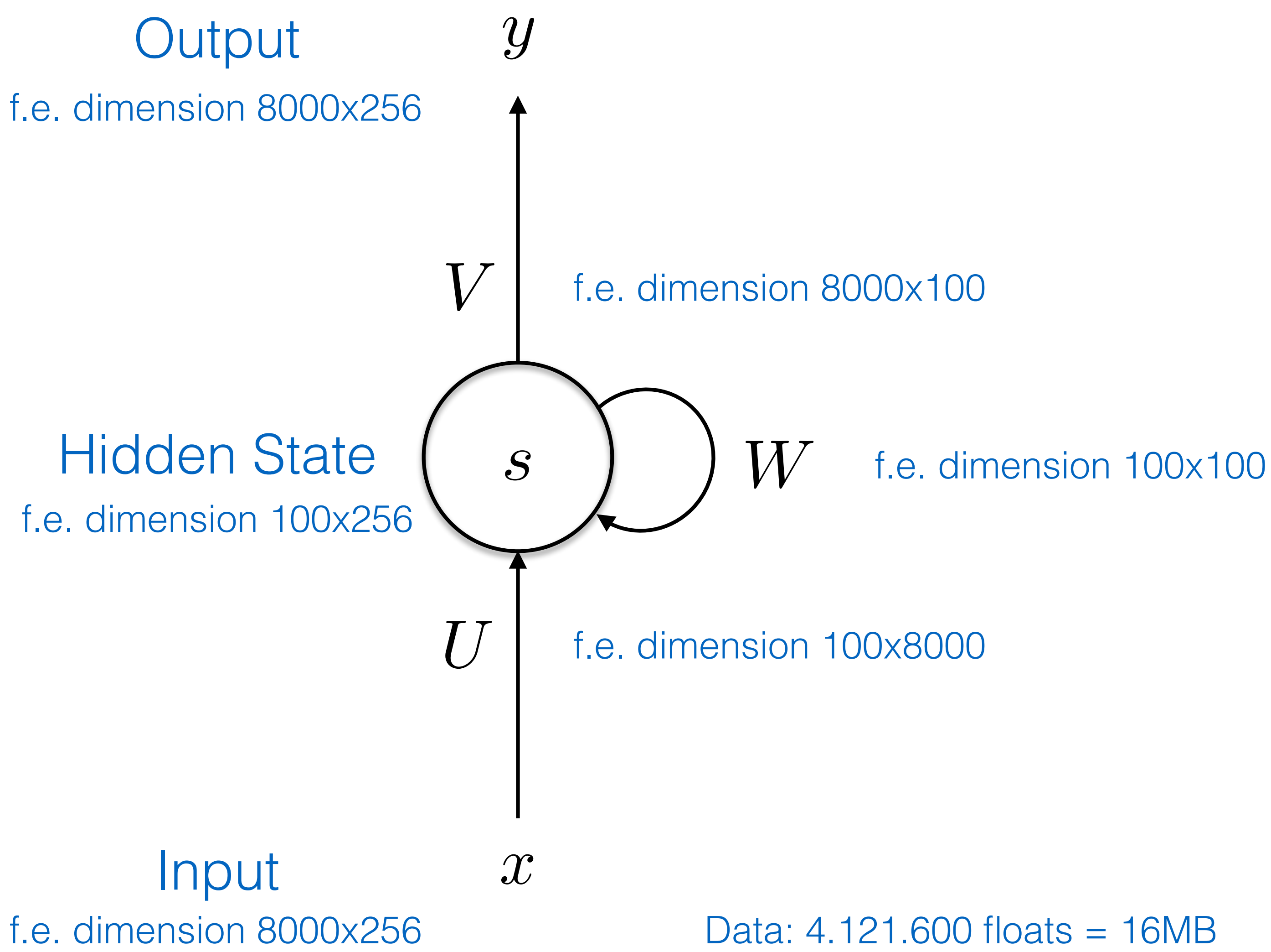
Basic equations of the RNN

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$y_t = \text{softmax}(Vs_t)$$



Vanilla Recurrent Neural Network

minibatch version



$$s_t$$

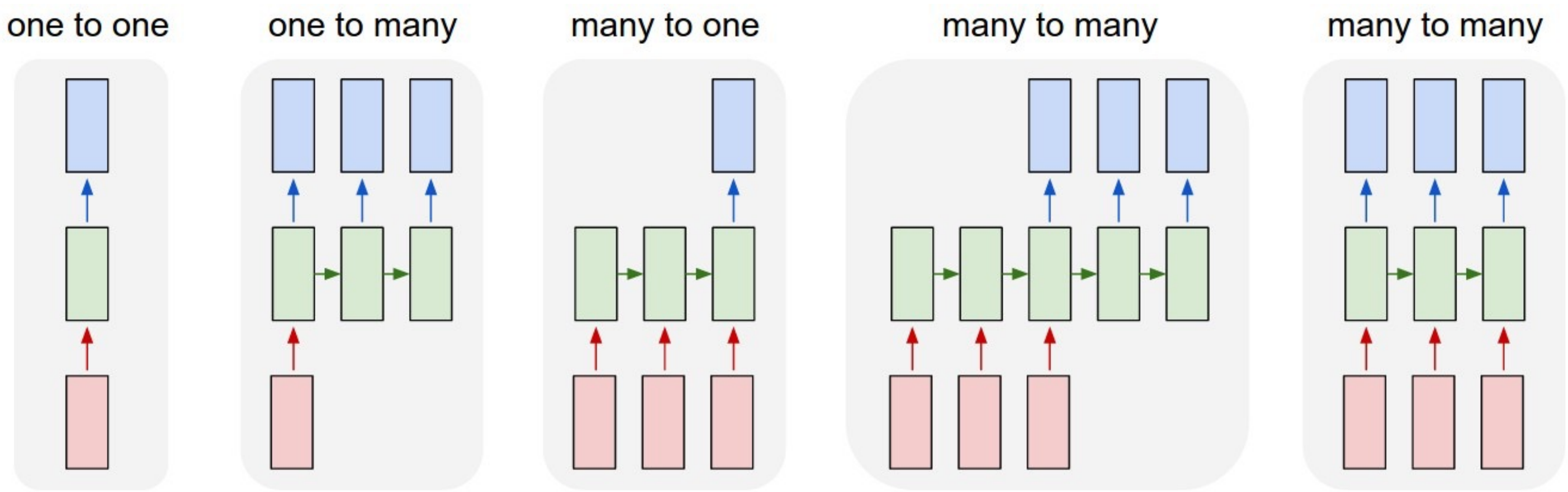
- We can think of the **hidden state** as a memory of the network that captures information about the previous steps.

$$U, V, W$$

- The RNN **shares the parameters** across all time steps.

$$y_t$$

- It is not necessary to have **outputs** at each time step.



Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN have shown success in:

- Language modeling and generation.
- Machine Translation.
- Speech Recognition.
- Image Description.
- Question Answering.
- Etc.

RNN Training

Training a RNN is similar to training a traditional NN, but some modifications.

The main reason is that parameters are shared by all time steps: in order to compute the gradient at t=4, we need to propagate 3 steps and sum up the gradients.

This is called **Backpropagation through time** (BPTT).

RNN Computation

```python
class RNN:
    # ...
    def step(self, x):
        self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
        y = np.dot(self.W_hy, self.h)
        return y
    # ...
```

`self.h` is initialized with the zero vector.

`Matrices` are initialized with random numbers.

We can go deep by stacking RNN:

```python
y1 = rnn1.step(x)
y2 = rnn2.step(y1)
```

RNN Models

Vanilla RNNs trained with SGD are unstable/difficult to learn. Bit various **tricks** make our life easier:

- Gating Units
- Gradient Clipping
- Steeper gates
- Better initialization

Gated Units

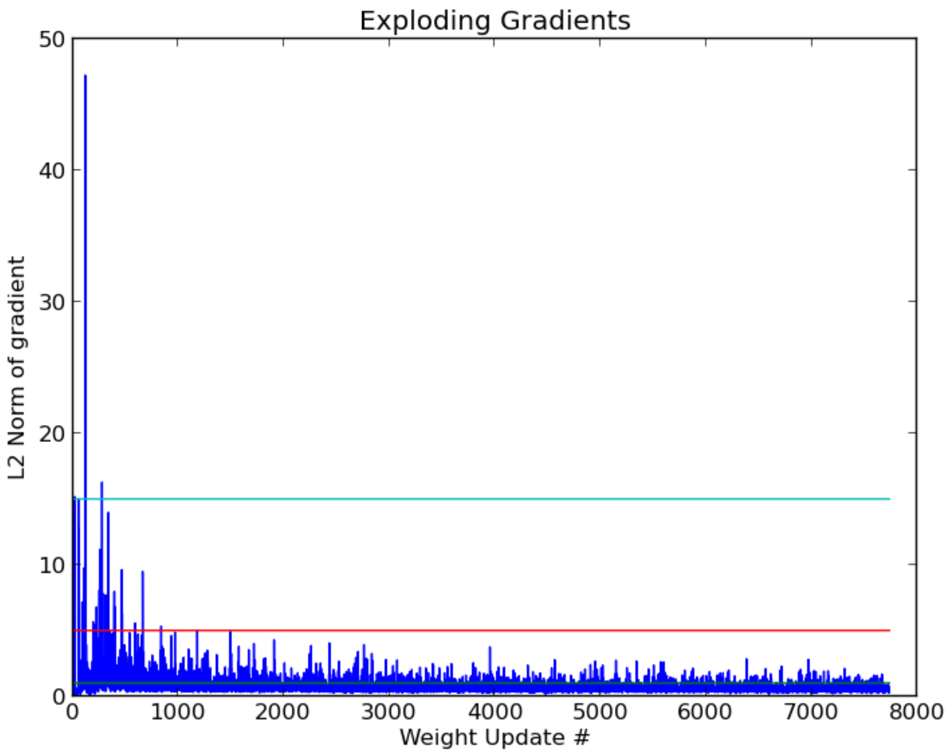
There are two types of gated RNNs:

- **Gated Recurrent Units** (GRU) by recently introduced K. Cho. GRU is simpler, faster, and optimizes quicker.

- **Long short term memory** (LSTM) by S. Hochreiter and J.Schmidhuber has been around since 1997 and has been used far more. LSTM may be better in the long run due to its greater complexity.

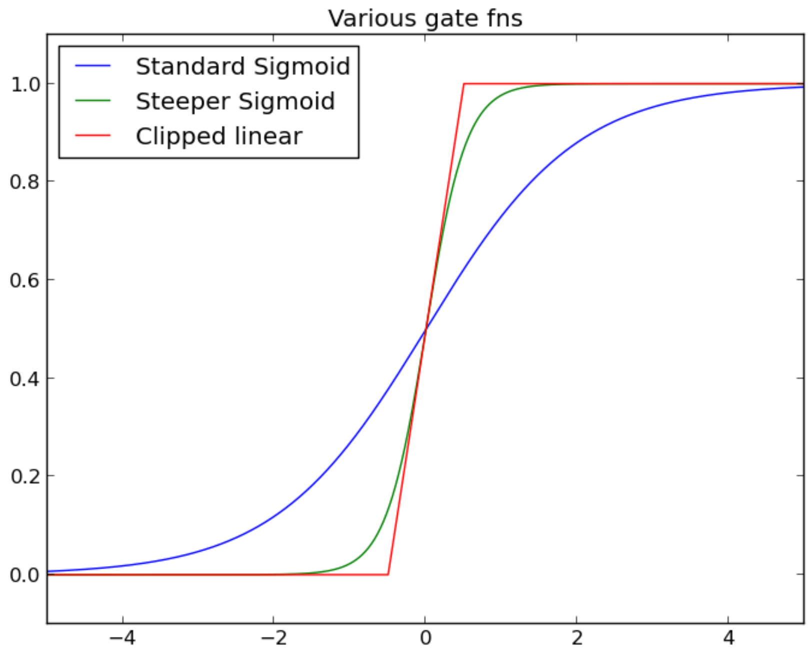
Exploding gradients

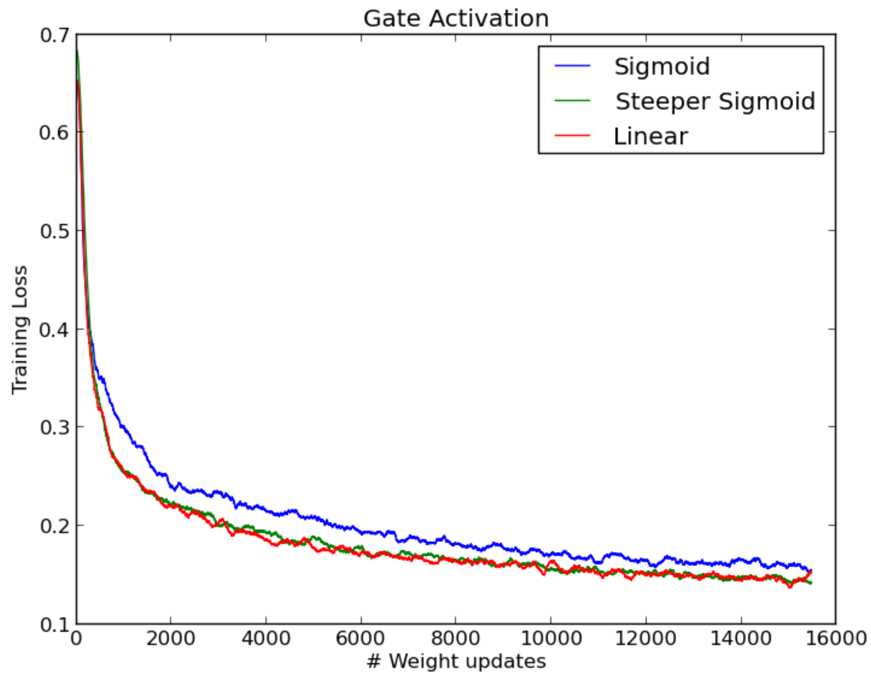
Exploding gradients may be a major problem for traditional RNNs trained with SGD. In 2012, R Pascanu and T. Mikolov proposed clipping the norm of the gradient to alleviate this.



Steeper Gates

We can make the gates “steeper” so they change more rapidly from “off” to “on” so model learns to use them quicker.

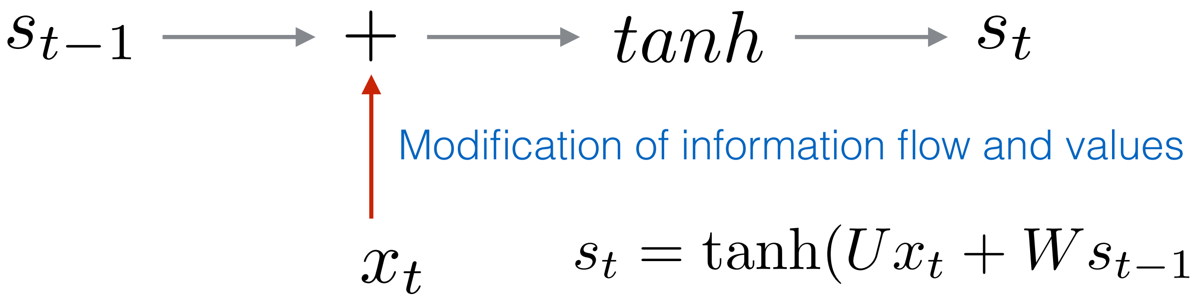




Better initialization

It has been showed that initializing weight matrices with random orthogonal matrices works better than random gaussian (or uniform) matrices.

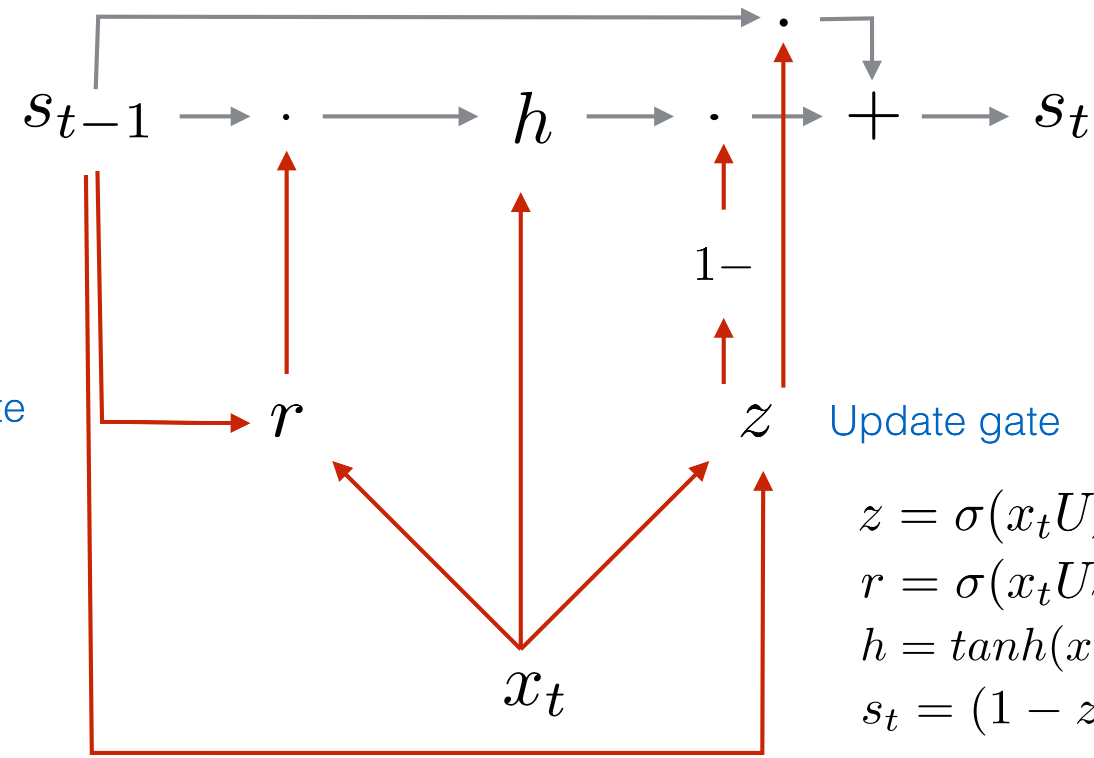
Gated Recurrent Unit (GRU)



$$s_t = \tanh(Ux_t + Ws_{t-1})$$

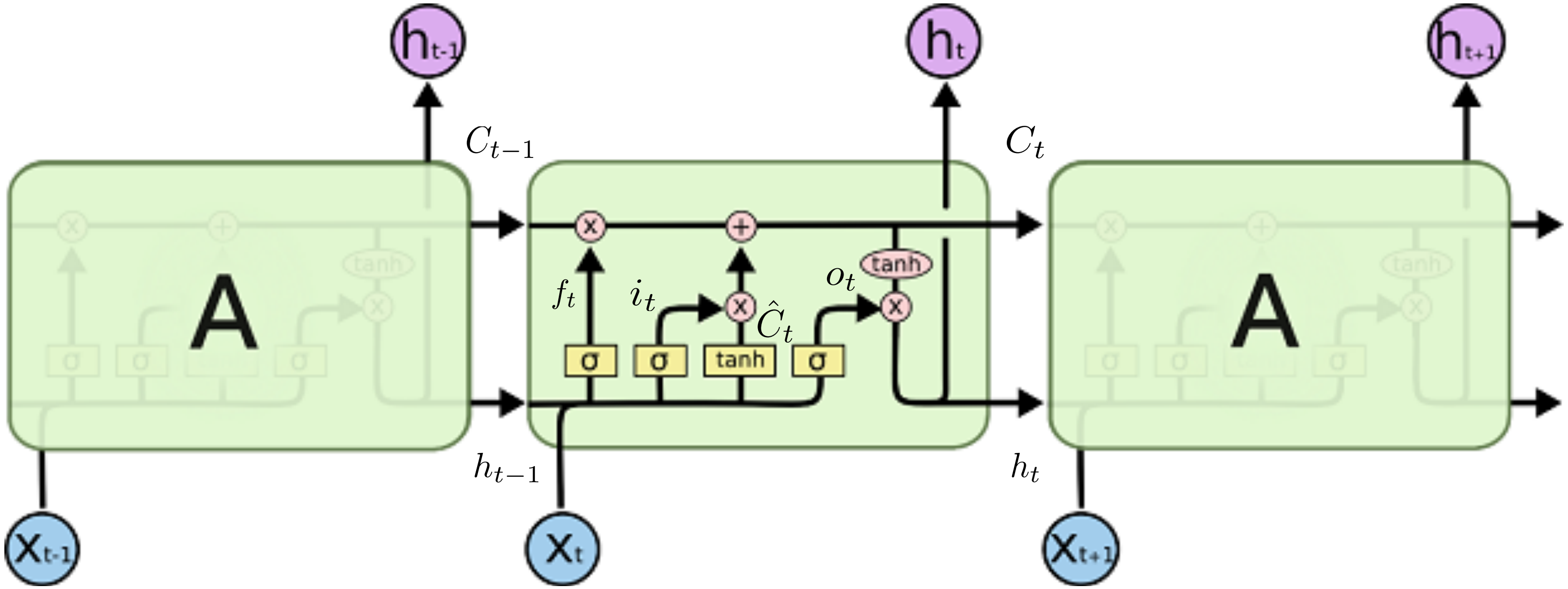
Vanilla RNN

GRU



$$z = \sigma(x_t U_z + s_{t-1} W_z)$$
$$r = \sigma(x_t U_r + s_{t-1} W_r)$$
$$h = tanh(x_t U_h + (s_{t-1} \cdot r) W_h)$$
$$s_t = (1 - z) \cdot h + z \cdot s_{t-1}$$

Long Short Term Memory Unit (LSTM)



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$f_t = \sigma(W_f[h_{t-1} \cdot x_t] + b_f)$$
$$i_t = \sigma(W_i[h_{t-1} \cdot x_t] + b_i)$$
$$\hat{C}_t = \tanh(W_C[h_{t-1} \cdot x_t] + b_C)$$
$$C_t = f_t C_{t-1} + i_t \hat{C}_t$$
$$o_t = \sigma(W_o[h_{t-1} \cdot x_t] + b_o)$$
$$h_t = o_t \tanh(C_t)$$

From text to RNN input

String Input

The cat sat on the mat.

Tokenize

The | **cat** | **sat** | **on** | **the** | **mat** | **.**

Indexing

0 **1** **2** **3** **0** **4** **5**

Embedding

2.5 0.3 -1.2 | **0.2 -3.3 0.7** | **-4.1 1.6 2.8** | **1.1 5.7 -0.2** | **2.5 0.3 -1.2** | **1.4 0.6 -3.9** | **-3.8 1.5 0.1**

Example: Name Modeling

Let's build a sequential (Name) model with a Recurrent Neural Network. Let's say we have name of m chars.

A name model allows us to predict the probability of observing the name as:

$$P(c_1 \dots c_m) = \prod_{i=1}^{m} P(c_i | c_1 \dots c_{i-1})$$

Note that in the equation the probability of each char is conditioned on all previous chars.

Example: Name Modeling

To train our model we need text to learn from a large dataset of names. Fortunately we don't need any labels to train a language model, just raw text.

I downloaded 52,700 Catalan names from a dataset available on

http://territori.gencat.cat/ca/01_departament/11_normativa_i_documentacio/03_documentacio/02_territori_i_mobilitat/cartografia/nomenclator_oficial_de_toponimia_de_catalunya/

Example: Name Modeling

Results

Alzinetes, torrent de les	Regueret, lo
Alzinetes, vall de les	Regueret, lo
Alzinó, Mas d'	**Regueró**
Alzinosa, collada de l'	Reguerols, els
Alzinosa, font de l'	Reguerons, els
Benavent, roc de	Vallverdú, Mas de
Benaviure, Cal	Vallverdú, serrat de
Benca	**Vallvicamanyà**
Bendiners, pla de	Vallvidrera
Benedi, roc del	Vallvidrera, riera de
Fiola, la	Terraubella, Corral de
Fiola, puig de la	Terraubes
Fiper, Granja del	**Terravanca**
Firassa, Finca	Terrer Nou, Can
Firell	Terrer Roig, lo