

# **Deep Learning Systems: Algorithms and Implementation**

## **Sequence Modeling and Recurrent Networks**

Fall 2022

J. Zico Kolter (this time) and Tianqi Chen  
Carnegie Mellon University

# Outline

Sequence modeling

Recurrent neural networks

LSTMs

Beyond "simple" sequential models

# Outline

Sequence modeling

Recurrent neural networks

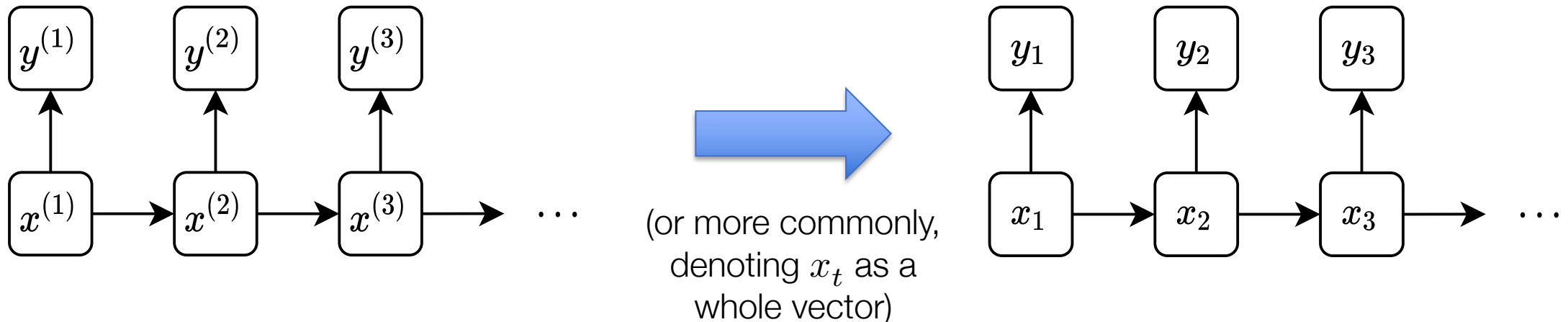
LSTMs

Beyond "simple" sequential models

# Sequence modeling tasks

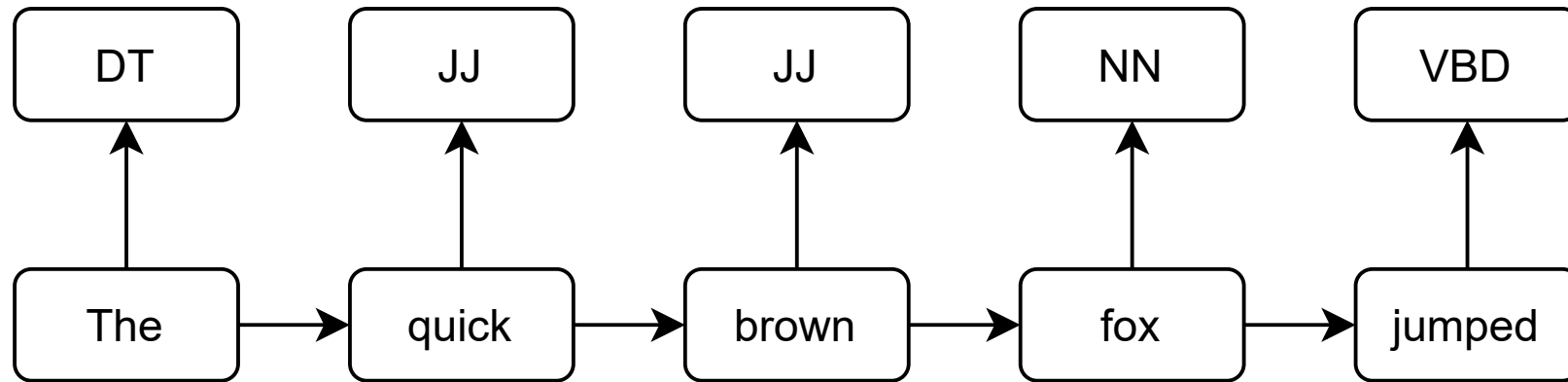
In the examples we have considered so far, we make predictions assuming each input output pair  $x^{(i)}, y^{(i)}$  is independent identically distributed (i.i.d.)

In practice, many cases where the input/output pairs are given in a specific *sequence*, and we need to use the information about this sequence to help us make predictions



# Example: Part of speech tagging

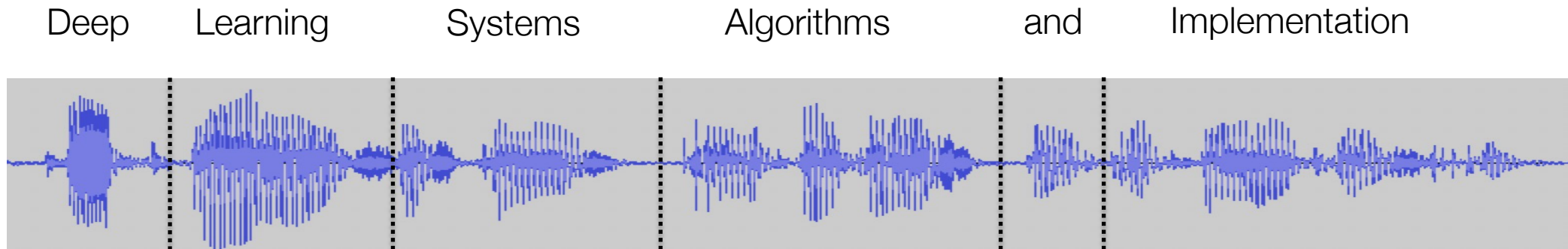
Given a sequence of words, determine the part of speech of each word



A word's part of speech depends on the context in which it is being used, not just on the word itself

# Example: speech to text

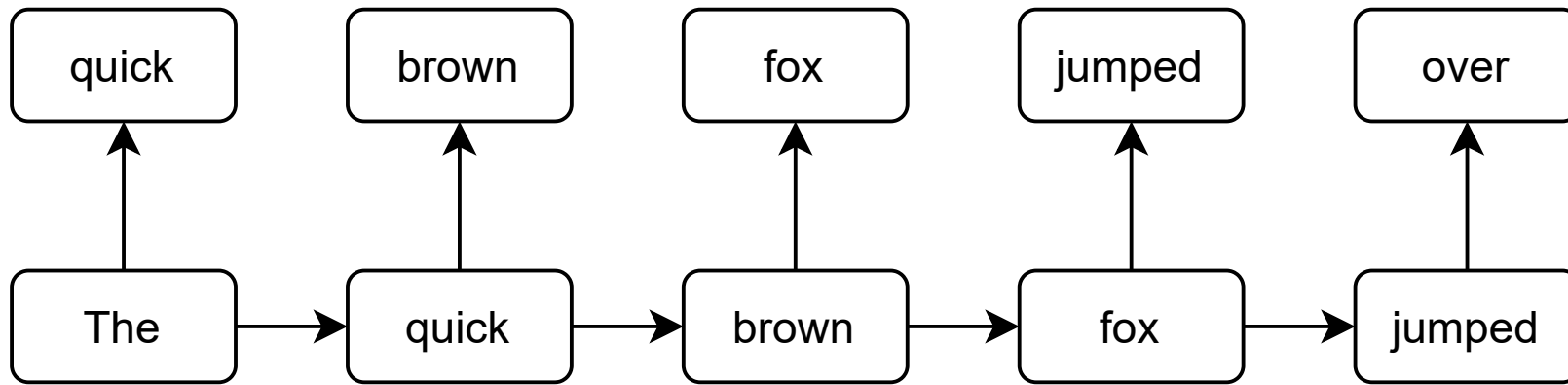
Given a audio signal (assume we even know the word boundaries, and map each segment to a fix-sized vector descriptor), determine the corresponding transcription



Again, context of the words is extremely important (see e.g., any bad speech recognition system that attempts to “wreck a nice beach”)

# Example: autoregressive prediction

A special case of sequential prediction where the elements to predict is the next element in the sequence



Common e.g., in time series forecasting, language modeling, and other use cases

# Outline

Sequence modeling

Recurrent neural networks

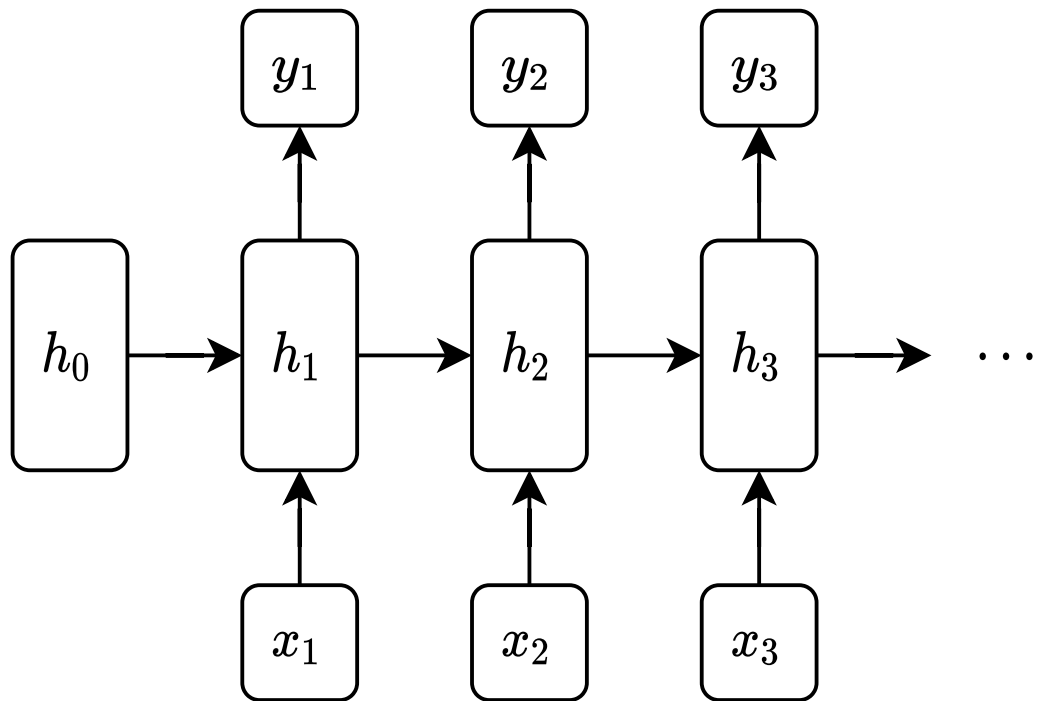
LSTMs

Beyond "simple" sequential models



# Recurrent neural networks

Recurrent neural networks (RNNs) maintain a hidden state over time, which is a function of the current input and previous hidden state



$$h_t = f(W_{hh}h_{t-1} + W_{hx}x_t + b_h)$$
$$y_t = g(W_{yh}h_t + b_y)$$

where  $f$  and  $g$  are activation functions,  $W_{hh}$ ,  $W_{hx}$ ,  $W_{yh}$  are weights and  $b_h$ ,  $b_y$  are bias terms

# How to train your RNN

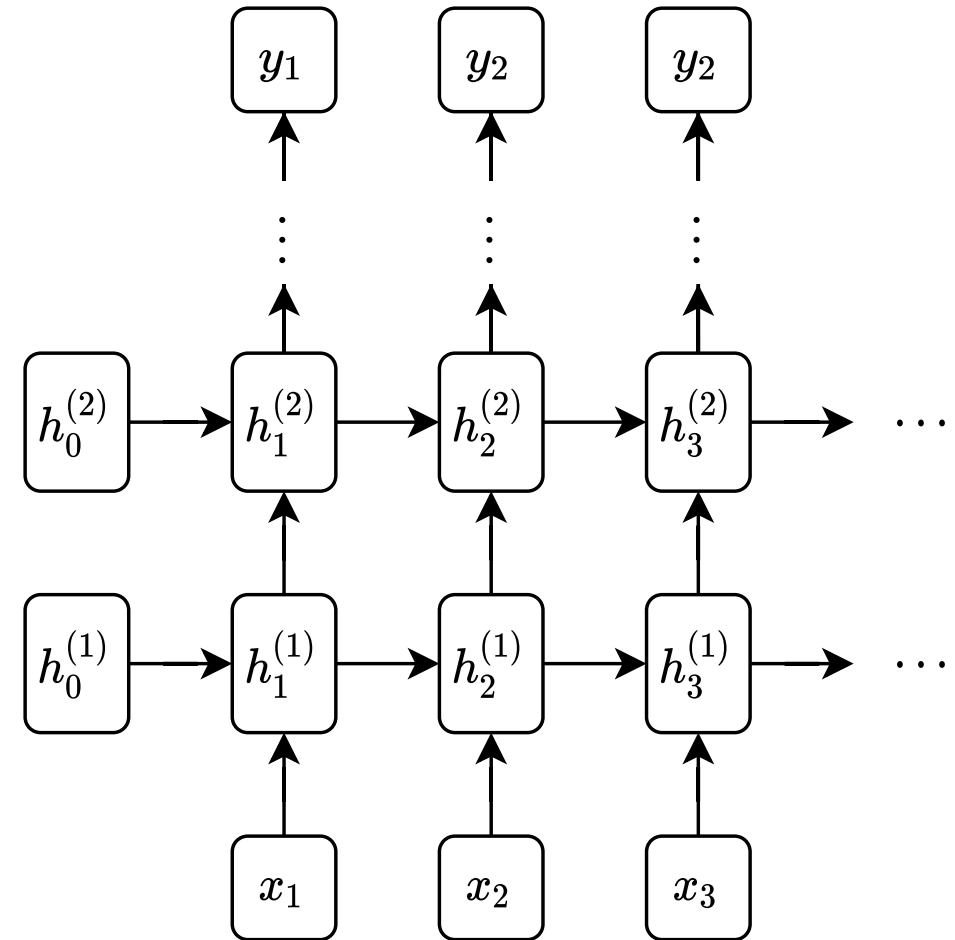
Given a sequence of inputs and target outputs  $(x_1, \dots, x_T, y_1^*, \dots, y_T^*)$ , we can train an RNN using backpropagation through time, which just involves “unrolling” the RNN over the length of the sequence, then relying mostly on autodiff

```
opt = Optimizer(params = (w_hh, w_hx, w_yh, b_h, b_y))
h[0] = 0
l = 0
for t = 1, ..., T:
    h[t] = f(w_hh * h[t-1] + w_hx * x[t] + b_h)
    y[t] = g(w_yh * h[t] + b_y)
    l += Loss(y[t], y_star[t])
l.backward()
opt.step()
```

# Stacking RNNs

Just like normal neural networks, RNNs can be stacked together, treating the hidden unit of one layer as the input to the next layer, to form “deep” RNNs

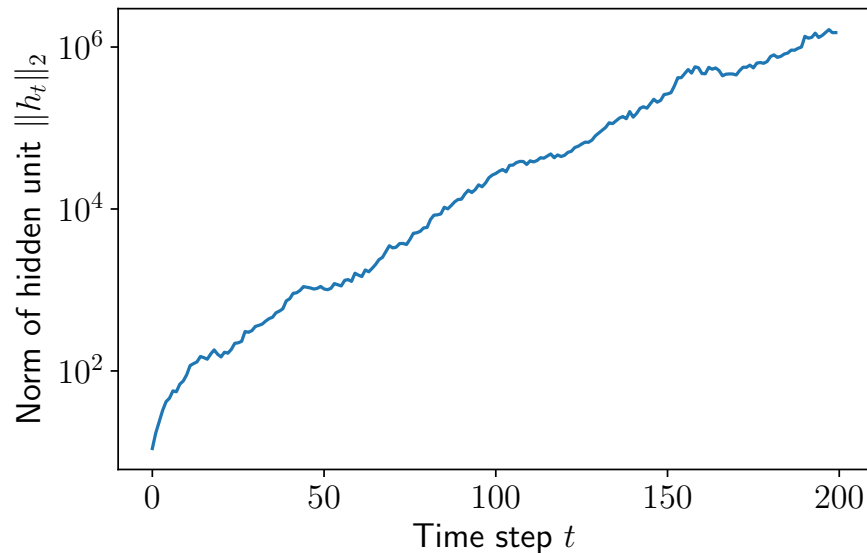
Practically speaking, tends to be less value in “very deep” RNNs than for other architectures



# Exploding activations/gradients

The challenge for training RNNs is similar to that of training deep MLP networks

Because we train RNNs on long sequences, if the weights/activation of the RNN are scaled poorly, the hidden activations (and therefore also the gradients) will grow unboundedly with sequence length



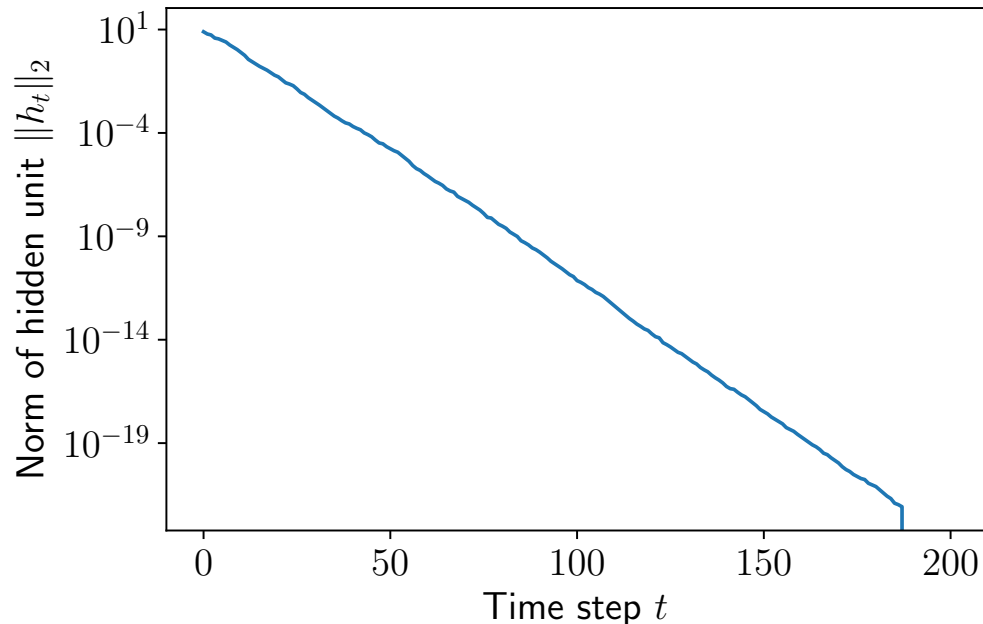
Single layer RNN with ReLU activations, using weight initialization

$$W_{hh} \sim \mathcal{N}(0, 3/n)$$

Recall that  $\sigma^2 = 2/n$  was the “proper” initialization for ReLU activations

# Vanishing activation/gradients

Similarly, if weights are too small then information from the inputs will quickly decay with time (and it is precisely the “long range” dependencies that we would often like to model with sequence models)



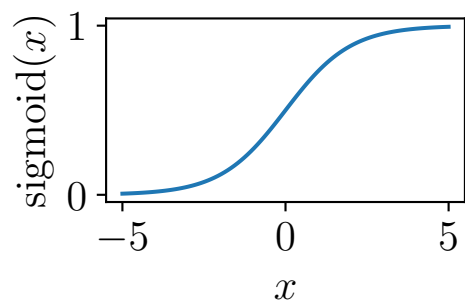
Single layer RNN with ReLU activations, using weight initialization

$$W_{hh} \sim \mathcal{N}(0, 1.5/n)$$

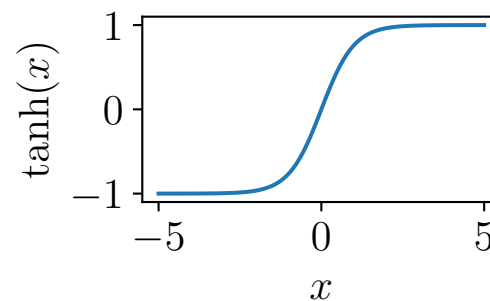
Non-zero input only provided here for time 1, showing decay of information about this input over time

# Alternative Activations

One obvious problem with the ReLU is that it can grow unboundedly; does using bounded activations “fix” this problem?



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



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

No ... creating large enough weights to not cause activations/gradients to vanish requires being in the “saturating” regions of the activations, where gradients are very small  $\implies$  still have vanishing gradients

# Outline

Sequence modeling

Recurrent neural networks

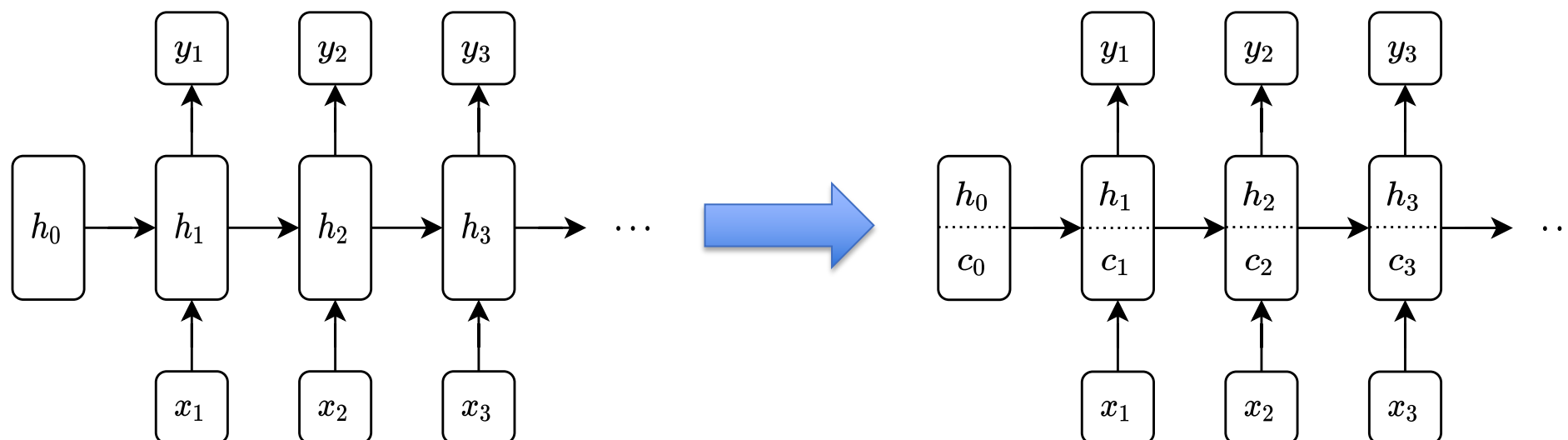
LSTMs

Beyond "simple" sequential models

# Long short term memory RNNs

Long short term memory (LSTM) cells are a particular form of hidden unit update that avoids (some of) the problems of vanilla LSTMs

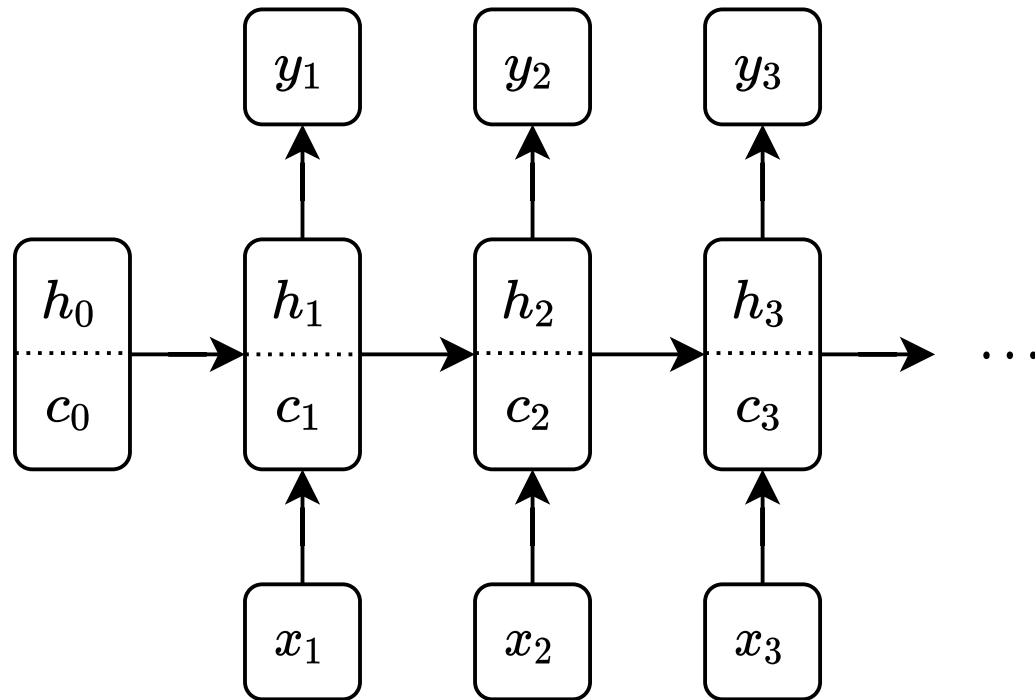
Step 1: Divide the hidden unit into two components, called (confusingly) the *hidden state* and the *cell state*





# Long short term memory RNNs

Step 2: Use a very specific formula to update the hidden state and cell state (throwing in some other names, like “forget gate”, “input gate”, “output gate” for good measure)



$$\begin{bmatrix} i_t \\ f_t \\ g_t \\ o_t \end{bmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \tanh \\ \text{sigmoid} \end{pmatrix} (W_{hh}h_{t-1} + W_{hx}x_t + b_h)$$
$$c_t = c_{t-1} \circ f_t + i_t \circ g_t$$
$$h_t = \tanh(c_t) \circ o_t$$

?????


# Why do LSTMs work?

There have been a seemingly infinite number of papers / blog posts about “understanding how LSTMs work” (I find most of them rather unhelpful)

$$\begin{bmatrix} i_t \\ f_t \\ g_t \\ o_t \end{bmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \tanh \\ \text{sigmoid} \end{pmatrix} (W_{hh}h_{t-1} + W_{hx}x_t + b_h)$$

$c_t = c_{t-1} \circ f_t + i_t \circ g_t$

$h_t = \tanh(c_t) \circ o_t$



The key is this line here:

- We form  $c_t$  by scaling down  $c_{t-1}$  (remember,  $f_t$  is in  $[0,1]^n$ ), then adding a term to it
- Importantly, “saturating” sigmoid activation for  $f_t$  at 1 would just pass through  $c_{t-1}$  untouched
- $\implies$  For a wide(r) range of weights, LSTMs don’t suffer vanishing gradients

# Some famous LSTMs

A notably famous blog post in the history of LSTMs:  
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>



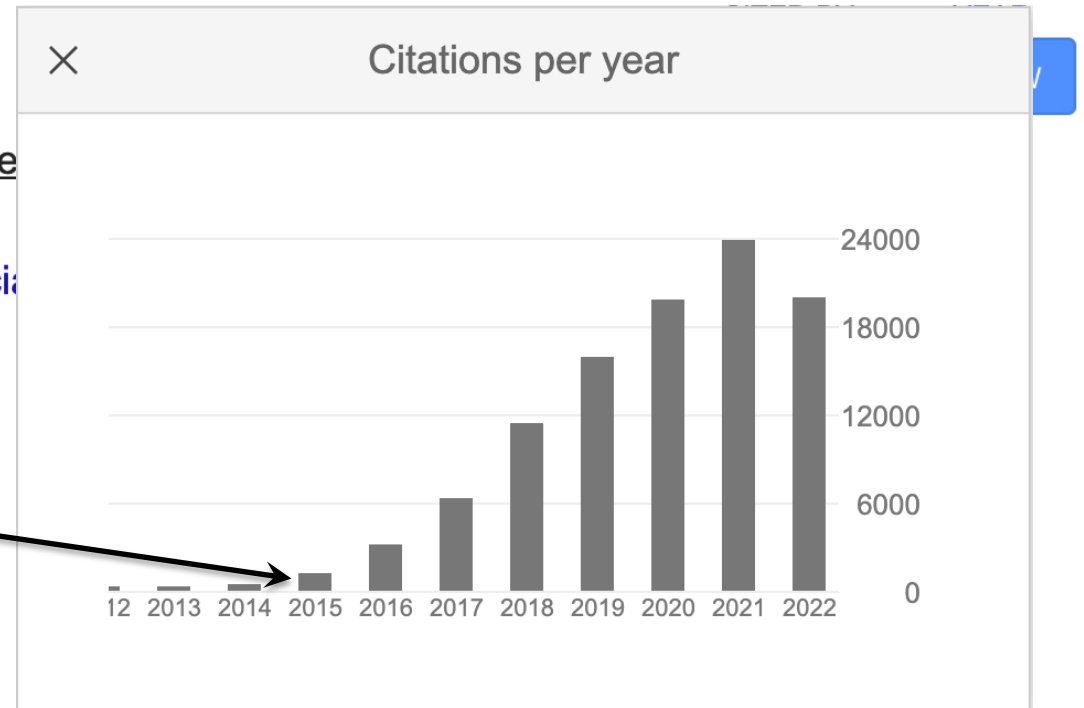
**Sepp Hochreiter**

Institute for Machine Learning, Johannes Kepler University Linz

Verified email at ml.jku.at - Homepage

[Machine Learning](#) [Deep Learning](#) [Artificial Intelligence](#)

Andrej's blog post



# Some famous LSTMs

```

/*
 * Increment the size file of the new incorrect
UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    ...

```

Generation from character-by-character  
autoregressive model trained on Linux source code

For  $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m_n} = 0$ , hence we can find a closed subset  $\mathcal{H}$  in  $\mathcal{H}$  and any sets  $\mathcal{F}$  on  $X$ ,  $U$  is a closed immersion of  $S$ , then  $U \rightarrow T$  is a separated algebraic space.

*Proof.* Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparico in the fibre product covering we have to prove the lemma generated by  $\coprod Z \times_U U \rightarrow V$ . Consider the maps  $M$  along the set of points  $Sch_{ppf}$  and  $U \rightarrow U$  is the fibre category of  $S$  in  $U$  in Section, ?? and the fact that any  $U$  affine, see Morphisms, Lemma ?? . Hence we obtain a scheme  $S$  and any open subset  $W \subset U$  in  $Sh(G)$  such that  $\text{Spec}(R') \rightarrow S$  is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that  $f_i$  is of finite presentation over  $S$ . We claim that  $\mathcal{O}_{X,x}$  is a scheme where  $x, x', s'' \in S'$  such that  $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}_{X',x'}$  is separated. By Algebra, Lemma ?? we can define a map of complexes  $\text{GL}_{S'}(x'/S'')$  and we win.  $\square$

To prove study we see that  $\mathcal{F}_U$  is a covering of  $\mathcal{X}'$ , and  $\mathcal{T}_i$  is an object of  $\mathcal{F}_{X/S}$  for  $i > 0$  and  $\mathcal{F}_p$  exists and let  $\mathcal{F}_i$  be a presheaf of  $\mathcal{O}_X$ -modules on  $\mathcal{C}$  as a  $\mathcal{F}$ -module. In particular  $\mathcal{F} = U/\mathcal{F}$  we have to show that

$$\widehat{M}^* = \mathcal{I}^* \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (Sch/S)_{ppf}^{opp}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \rightarrow (U, \text{Spec}(A))$$

is an open subset of  $X$ . Thus  $U$  is affine. This is a continuous map of  $X$  is the inverse, the groupoid scheme  $S$ .

*Proof.* See discussion of sheaves of sets.  $\square$

The result for prove any open covering follows from the less of Example ?? . It may replace  $S$  by  $X_{spaces, \acute{e}tale}$  which gives an open subspace of  $X$  and  $T$  equal to  $S_{Zar}$ , see Descent, Lemma ?? . Namely, by Lemma ?? we see that  $R$  is geometrically regular over  $S$ .

**Lemma 0.1.** Assume (3) and (3) by the construction in the description.

Suppose  $X = \lim |X|$  (by the formal open covering  $X$  and a single map  $\text{Proj}_X(\mathcal{A}) = \text{Spec}(B)$  over  $U$  compatible with the complex

$$\text{Set}(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X, \mathcal{O}_X}).$$

When in this case of to show that  $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$  is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If  $T$  is surjective we may assume that  $T$  is connected with residue fields of  $S$ . Moreover there exists a closed subspace  $Z \subset X$  of  $X$  where  $U$  in  $X'$  is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1)  $f$  is locally of finite type. Since  $S = \text{Spec}(R)$  and  $Y = \text{Spec}(R)$ .

*Proof.* This is form all sheaves of sheaves on  $X$ . But given a scheme  $U$  and a surjective étale morphism  $U \rightarrow X$ . Let  $U \cap U = \coprod_{i=1,\dots,n} U_i$  be the scheme  $X$  over  $S$  at the schemes  $X_i \rightarrow X$  and  $U = \lim_i X_i$ .  $\square$

The following lemma surjective restrocomposes of this implies that  $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\dots,0}$ .

**Lemma 0.2.** Let  $X$  be a locally Noetherian scheme over  $S$ ,  $E = \mathcal{F}_{X/S}$ . Set  $\mathcal{I} = \mathcal{I}_1 \subset \mathcal{I}'_n$ . Since  $\mathcal{I}^n \subset \mathcal{I}^n$  are nonzero over  $i_0 \leq \mathfrak{p}$  is a subset of  $\mathcal{I}_{n,0} \circ \mathcal{A}_2$  works.

**Lemma 0.3.** In Situation ?? . Hence we may assume  $\mathfrak{q}' = 0$ .

*Proof.* We will use the property we see that  $\mathfrak{p}$  is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where  $K$  is an  $F$ -algebra where  $\delta_{n+1}$  is a scheme over  $S$ .  $\square$

... trained on Latex source code

# Outline

Sequence modeling

Recurrent neural networks

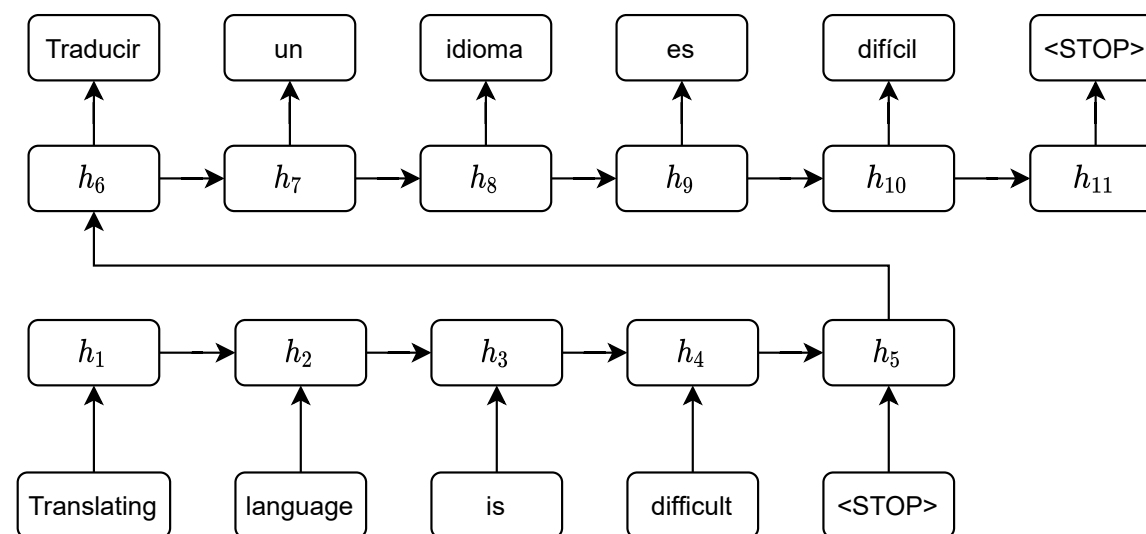
LSTMs

Beyond "simple" sequential models

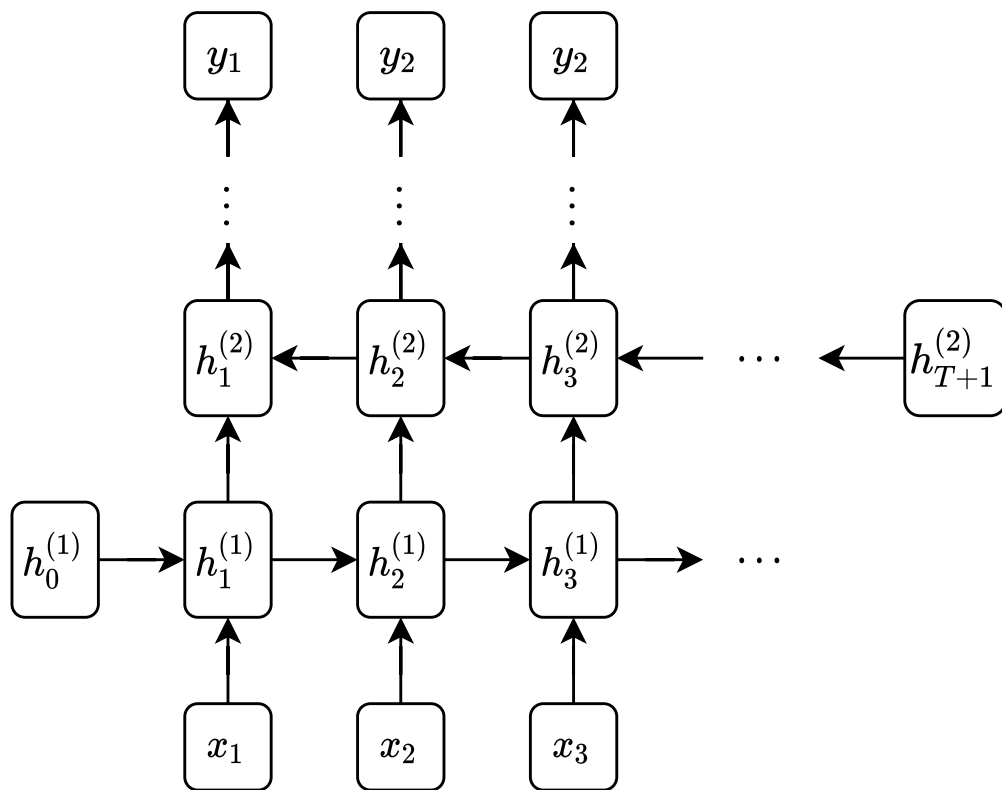
# Sequence-to-sequence models

To give you a short glimpse of the kind of things you can do with RNNs/LSTMs beyond “simple” sequence prediction, consider the task of trying to translate between languages

Can concatenate two RNNs together, one that “only” processes the sequence to create a final hidden state (i.e., no loss function); then a section that takes in this initial hidden state, and “only” generates a sequence



# Bidirectional RNNs



RNNs can use *only* the sequence information up until time  $t$  to predict  $y_t$

- This is sometimes desirable (e.g., autoregressive models)
- But sometime undesirable (e.g., language translation where we want to use “whole” input sequence)

Bi-directional RNNs: stack a forward-running RNN with a backward-running RNN: information from the entire sequence to propagates to the hidden state