CIS 522: Lecture 12

RNNs and LSTMs

2/25/2020

Feedback

HW2 took mean=25h. Will somewhat shorten next year

Some did it in <5. How?

HW3 is, afaik, the last long HW.

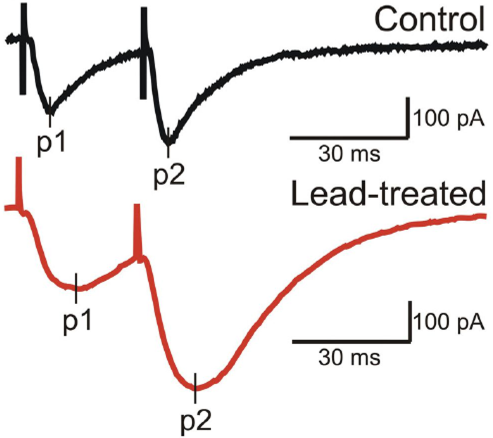
We continue to get "more code", "more examples" and "more theory" requests. Next year fewer guest lectures.

Exam + Projects

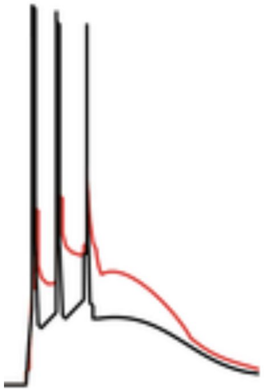
Today's Agenda

- Some neuroscience background
- Review of NLP and variable-length problems
- RNNs
  - Backpropagation through time (BPTT)
- LSTMs
  - Motivation
  - Forget gates
- BiLSTMs
  - Motivation
  - Architecture

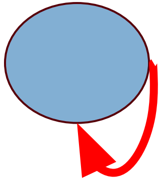
Neuroscience: The ubiquity of memory across timescales



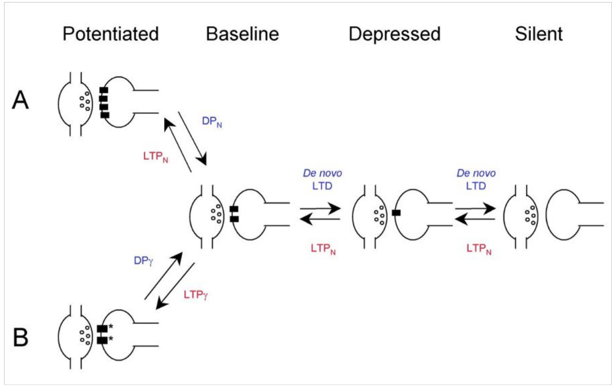
PPF: Zhang et al 2015



Ca2+ spikes: Larkum



Recurrent activity



Plasticity

Working memory

Pencil

Automobile

Evil

Working memory

Graphic

Element

Mediocre

Steel

Plate

Nociception

Memory

Working memory

Random

Bottle

Phone

Overeager

Card

Plant

Chimney

Tree

House

Roof

A multitude of memory elements

Working memory

Episodic memory

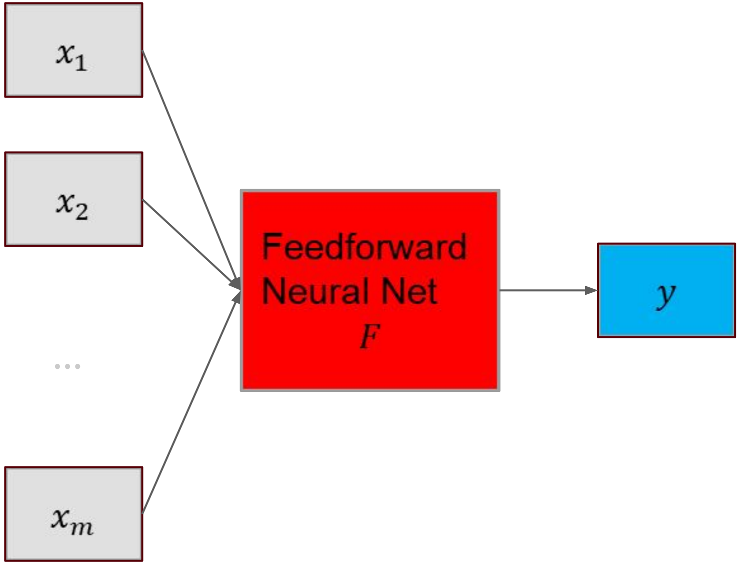
Procedural memory

Declarative memory

Hippocampus

Thoughts have persistence over time. How can we capture this with an architecture?

The API for feedforward nets is too restrictive.



How does the number of parameters scale? PollEV

We increase the relevant time horizon.

Fully connected network

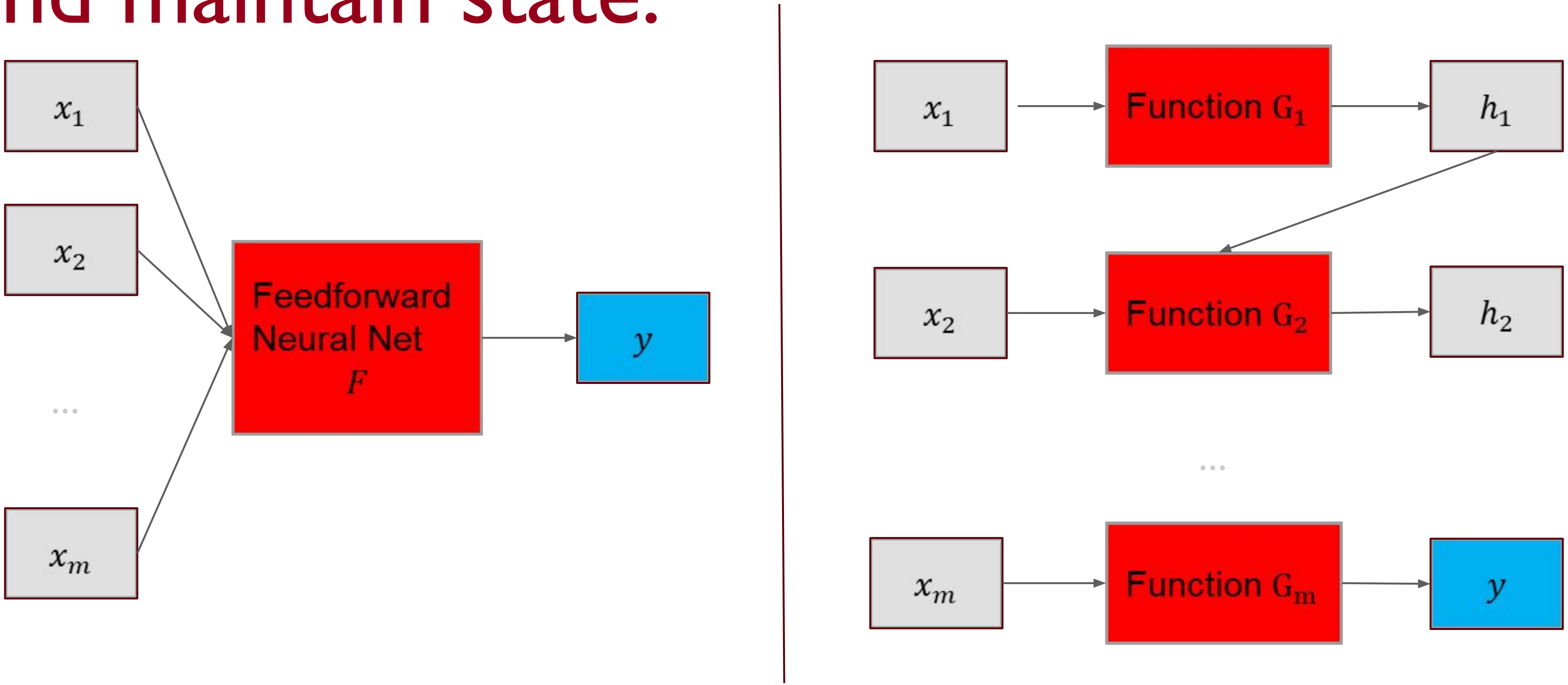
What if we use a deep convnet?

Fixed stride

Fixed filter size on each layer

Does it help?

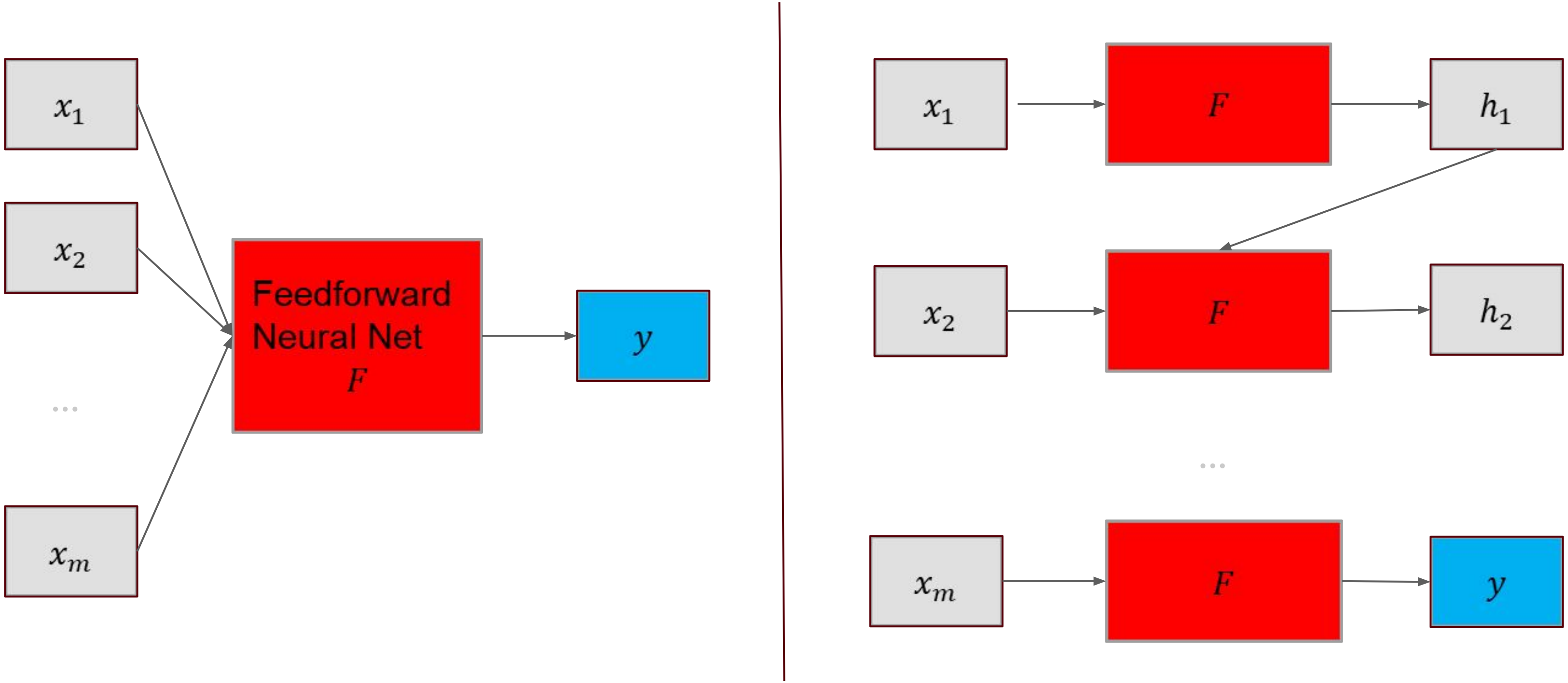
RNNs: We'd like to have a notion of time and maintain state.



We can't learn a new function for each timestep!
How do we simplify our architecture?

We can't learn a new function for each timestep!
How do we simplify our architecture?
The way we think doesn't change from moment to moment.

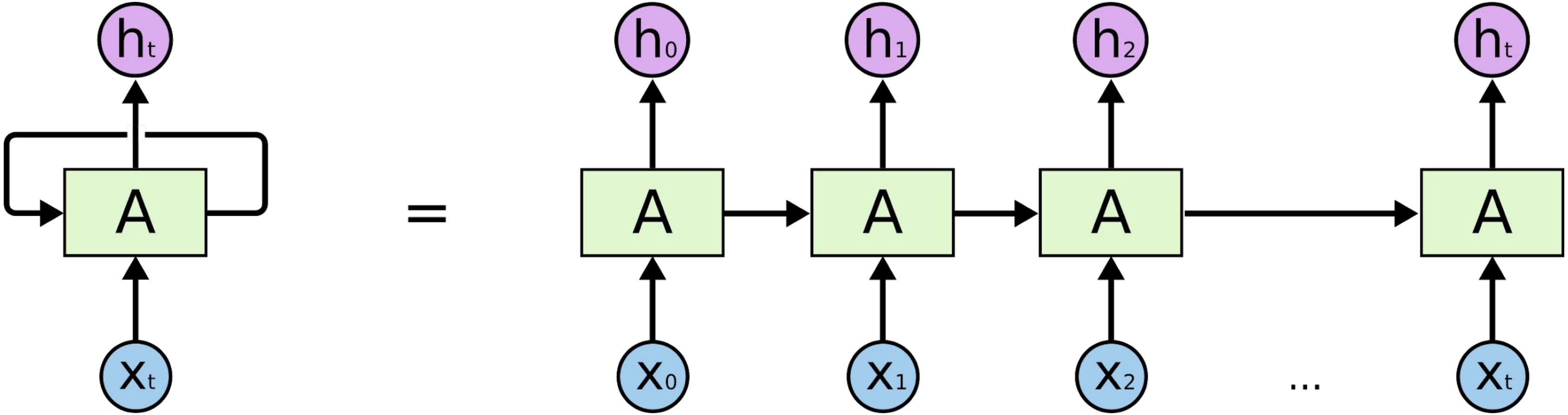
We share weights across time.



Penn Engineering

Recurrent neural networks (RNNs)

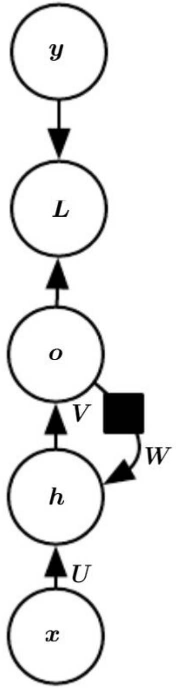
- Prior: the function modifying the state of a thought is invariant to temporal shifts.



PollEV how many parameters?

RNNs typically have (at least) 3 weight tensors (*U, W, V*) and 2 biases.

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

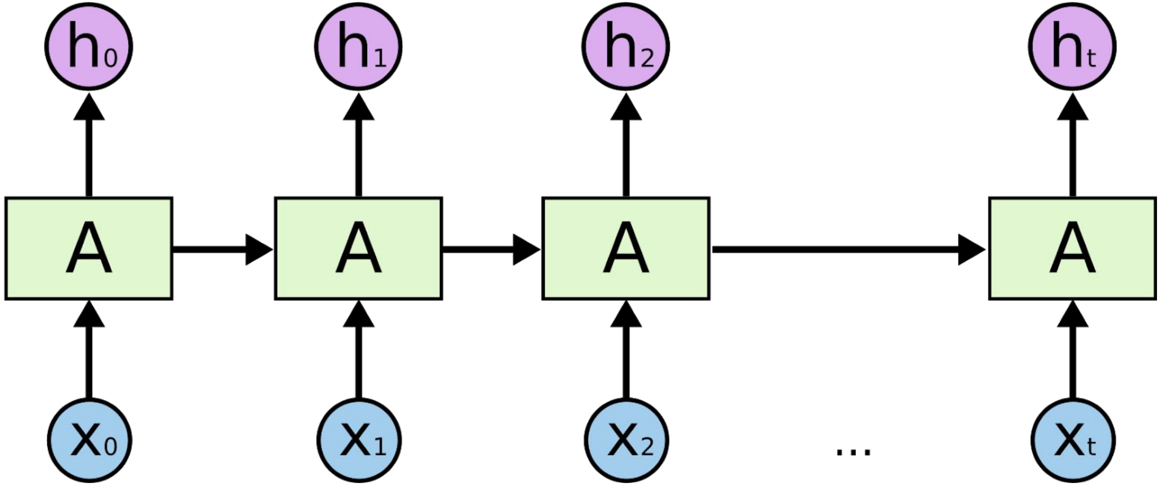


Activation functions

Usually tanh

Want to keep activations from exploding

How does the compute tree look like for an RNN?



What do we want to produce?

A vector (see last lecture)

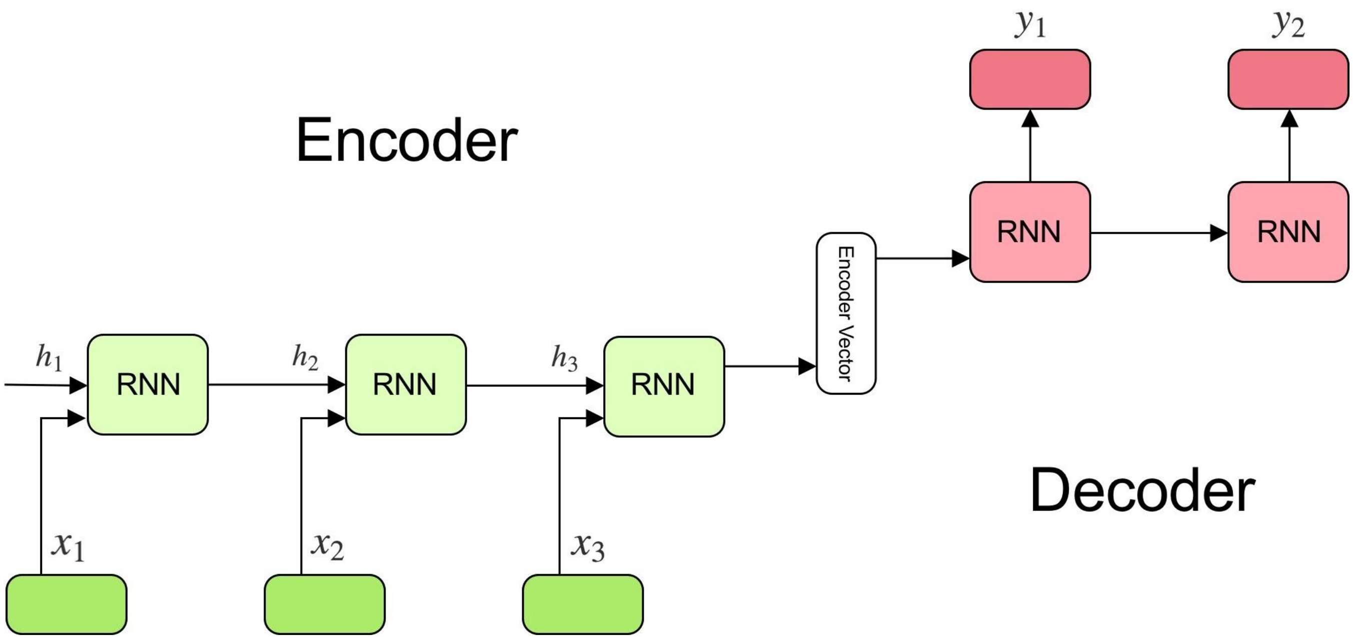
Text output

Text translations

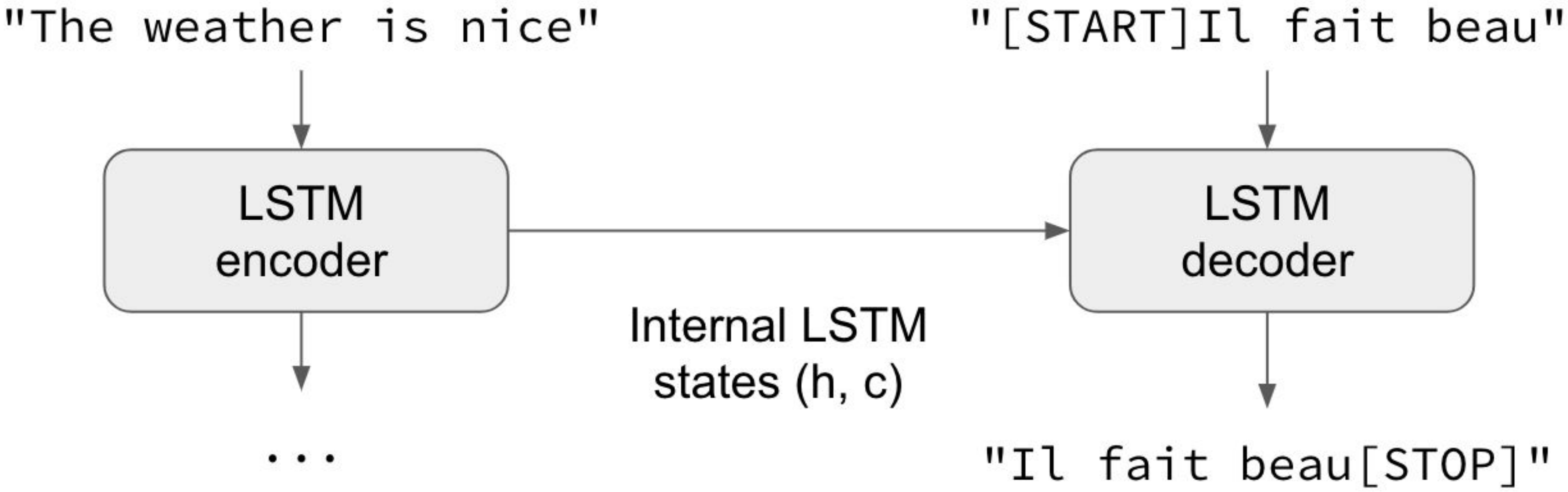
Video translations

Pose tracking etc

Sequence-to-Sequence (Seq2Seq) Learning

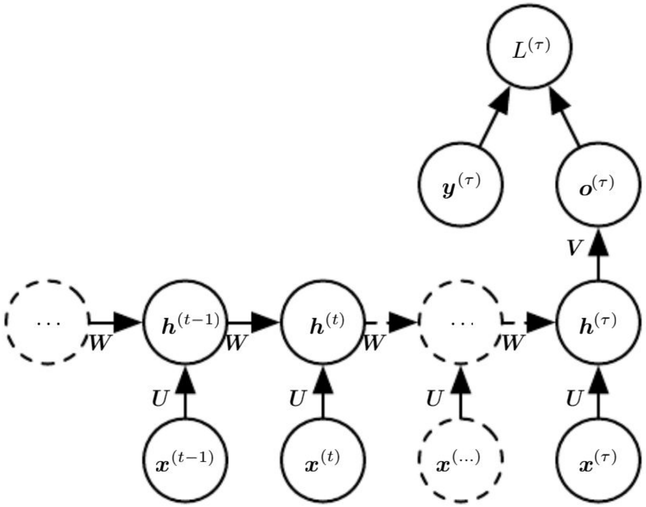


Example application

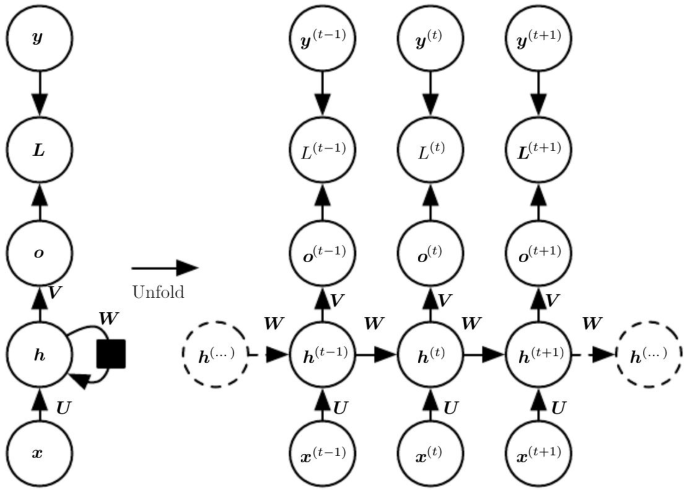


Towards data science

The exact unrolled architecture depends on the learning problem.

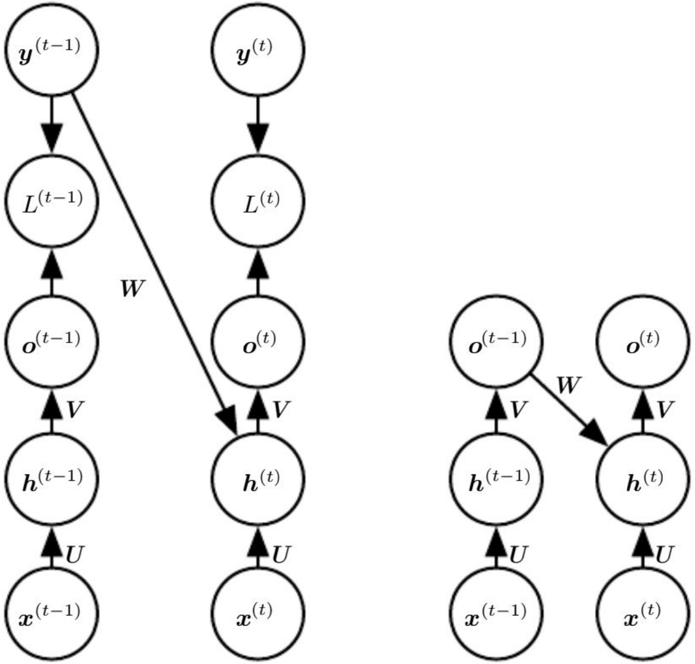


Variable-length input, single output at the end.



Variable-length input, simultaneous outputs.

Teacher forcing helps learn complex patterns concurrently



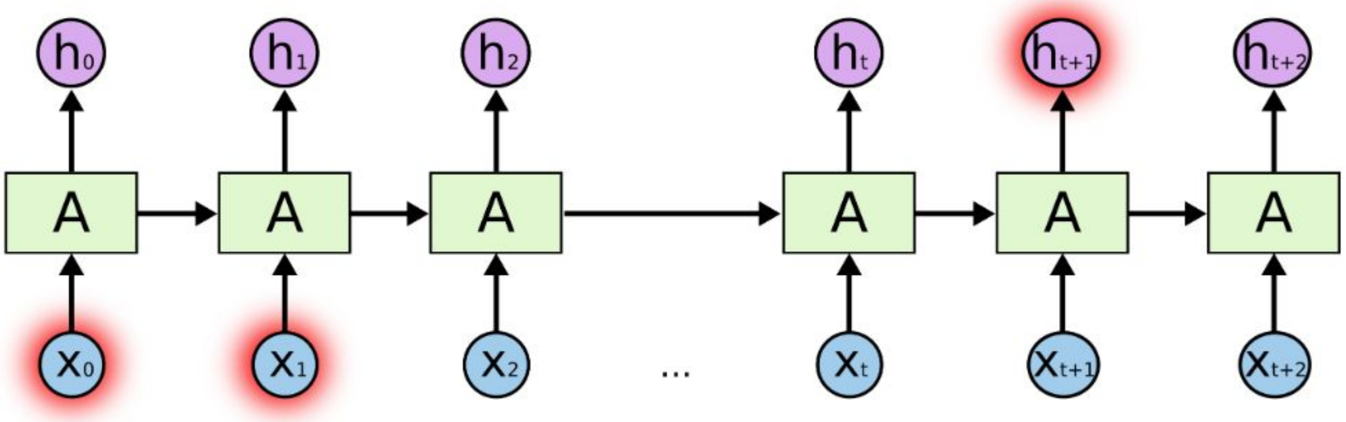
Train time

Test time

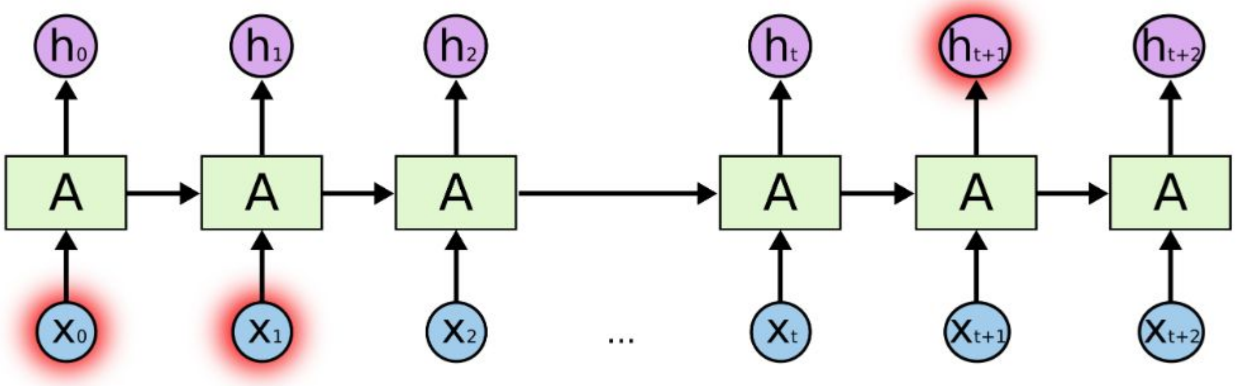
Backpropagation through time (BPTT)

$$\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, \\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L, \\
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}^{(t)}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)^\top}, \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)^\top}, \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)^\top},
\end{aligned}$$

RNNs are awful at learning long-term patterns.



RNNs are awful at learning long-term patterns.



What will the gradients do?

Which properties do we want memory in an RNN to have

Store information for arbitrary duration

Be resistant to noise

Be trainable

Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio, Patrice Simard, and Paolo Frasconi, *Student Member, IEEE*

A hard task for this

A relevant sequence (length L)

Followed by an irrelevant sequence (length T>>L)

Answer at end

Intuition: figure out relevant information. Then keep it for L steps.

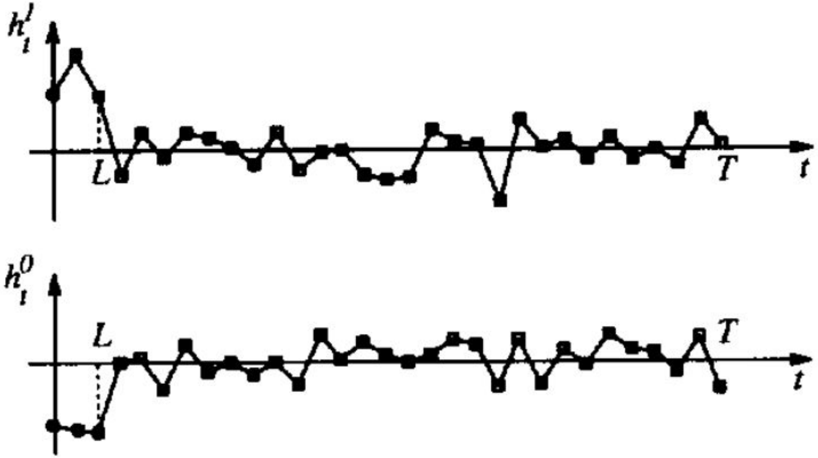
Trivial solution: using only one tanh neuron!

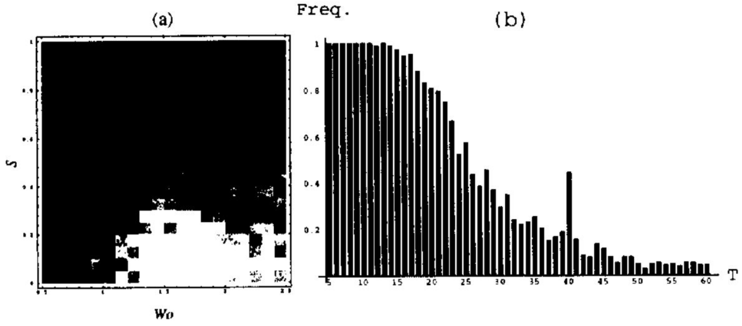
: Latch to +/- 1

Dynamics and failure

$$x_t = f(a_t) = tanh(a_t)$$

$$a_t = wx_{t-1} + h_t$$





Let us think about the gradients

We want that the irrelevant input has little input

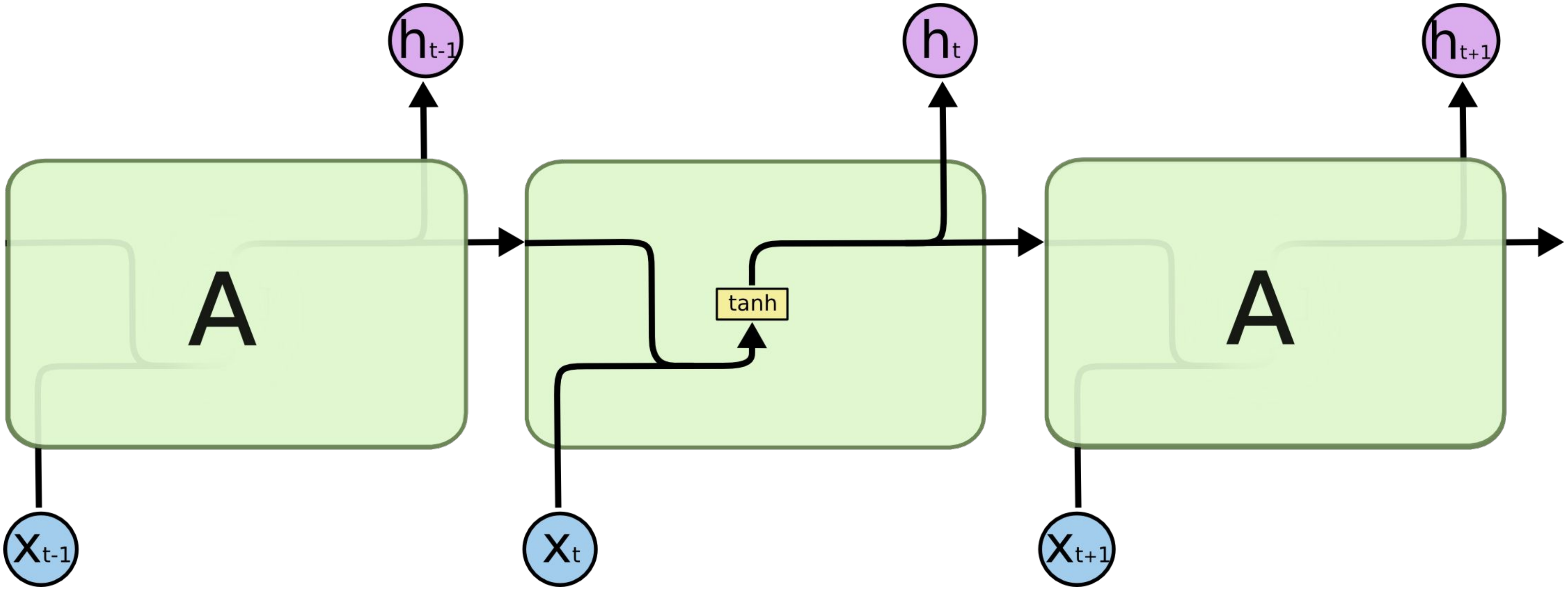
So the input has little influence

What does that mean for the gradient?

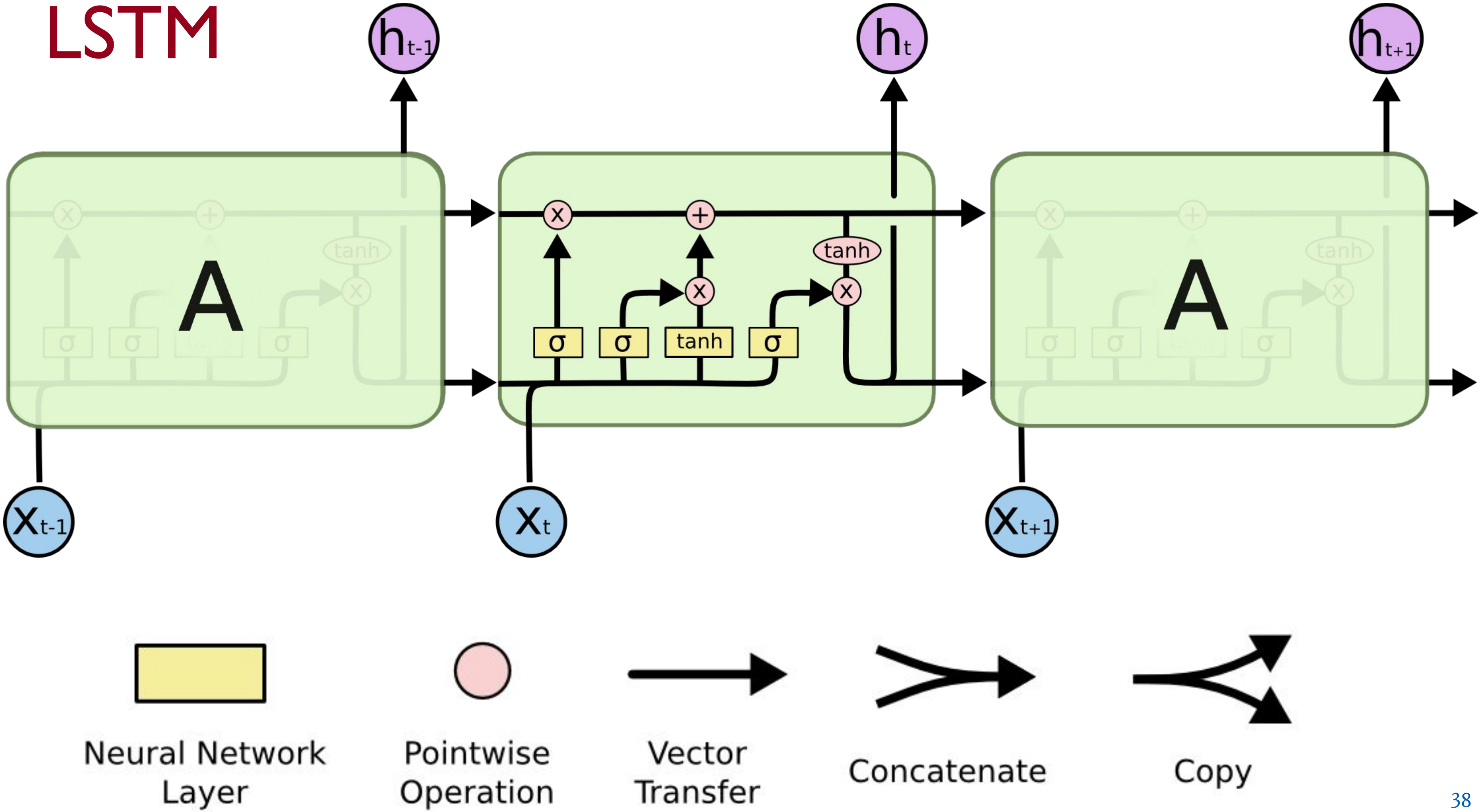
LSTMs

Leaning on Colah's blog:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN



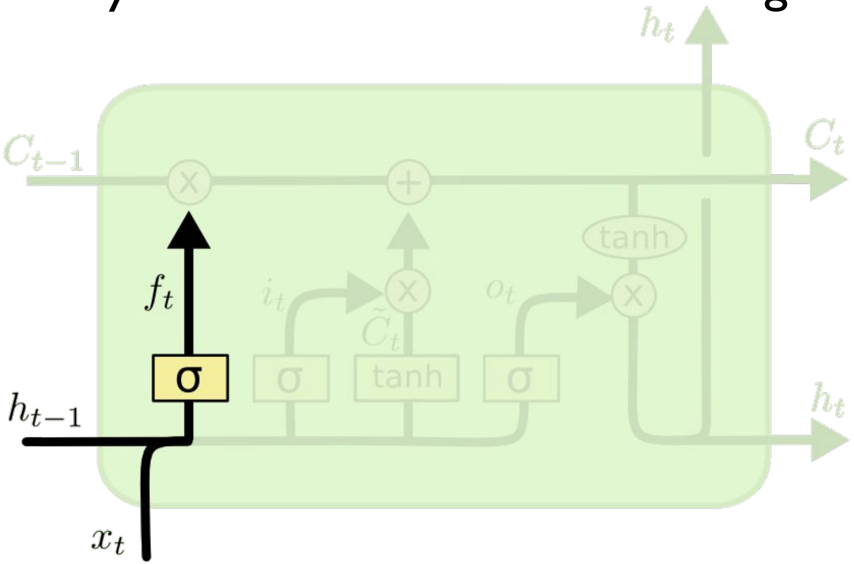
LSTM



ht-1
ht
ht+1
A
x
+
tanh
σ
σ
tanh
σ
A
Xt-1
Xt
Xt+1
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

Stepping through the LSTM: delete if makes sense

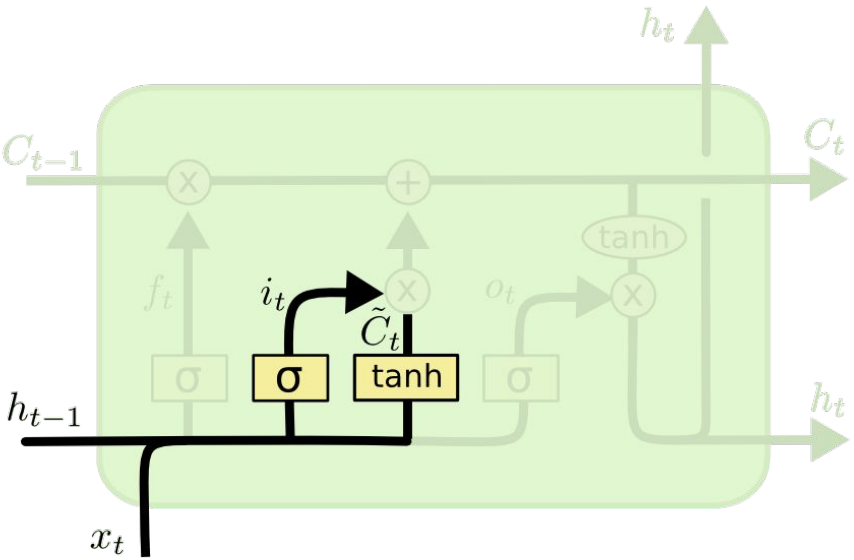
Say cell should contain if last image I saw is a puppy



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Step 1: forget if new subject

Write new value if makes sense



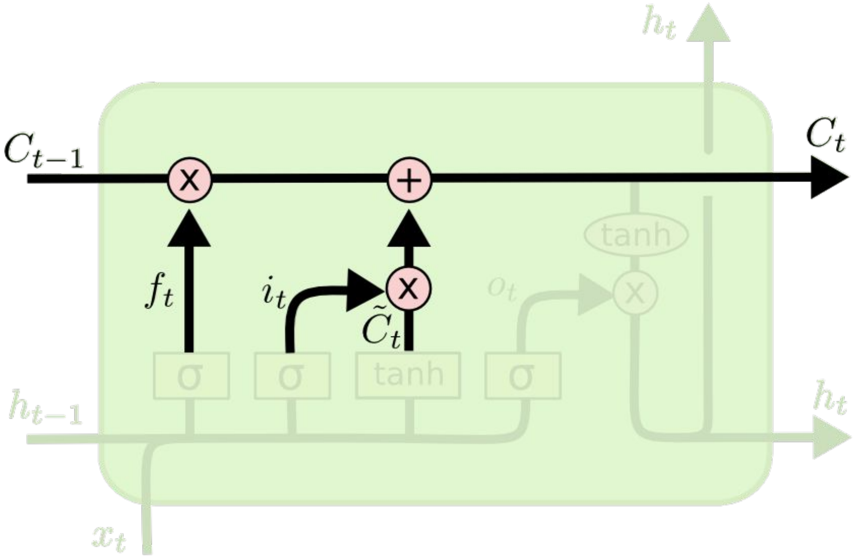
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Tanh kinda binarizes the to-be-stored values.
Even more resilient.

Step 2: write if new subject

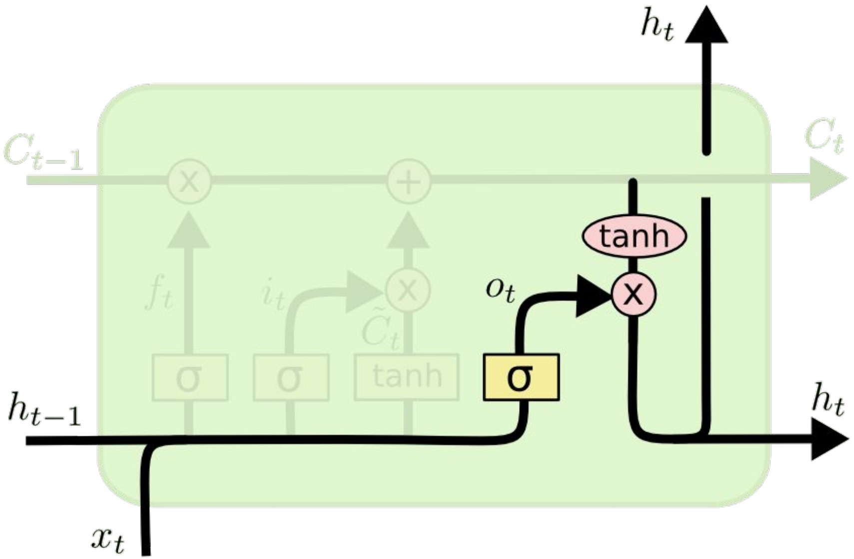
Commit to memory



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step 3: commit to hidden state

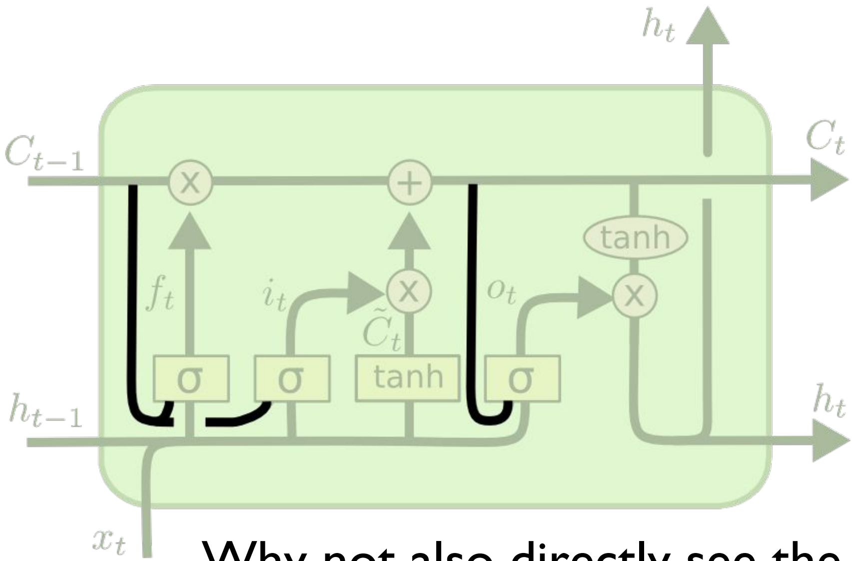
Get output hidden state



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Variant: Look at memory



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$

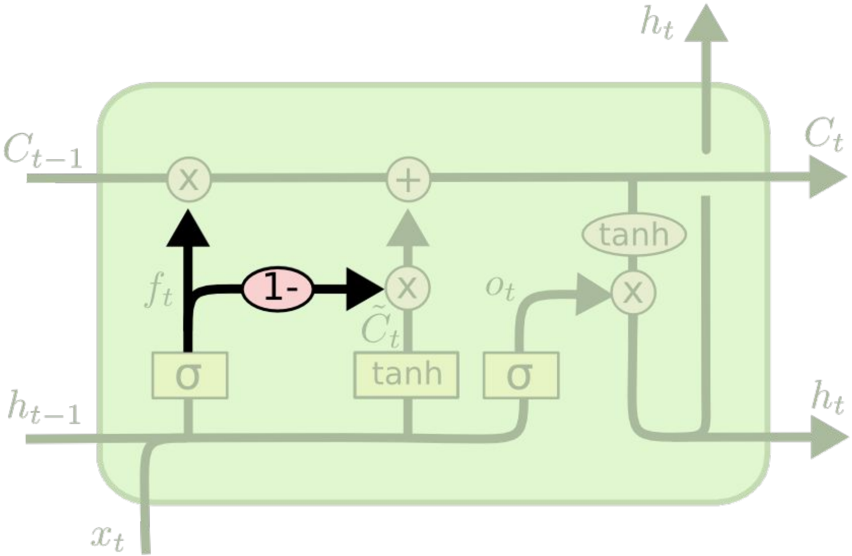
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$

$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Why not also directly see the memory?

Gers and Schmidthuber 2000

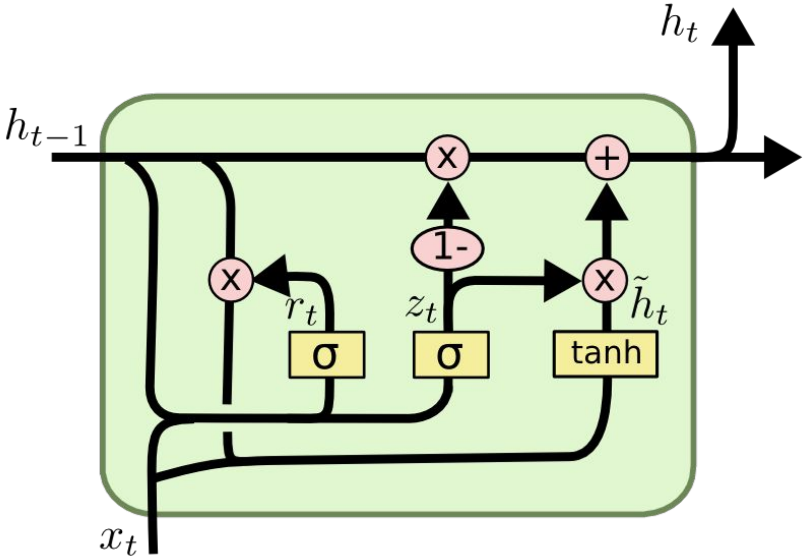
Variant: Couple read and write



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

Couple delete and write, but can not do decay anymore!

Gated Recurrent Unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Success stories of LSTMs

SOTA in Precipitation nowcasting (2016)

SOTA named entity (2016)

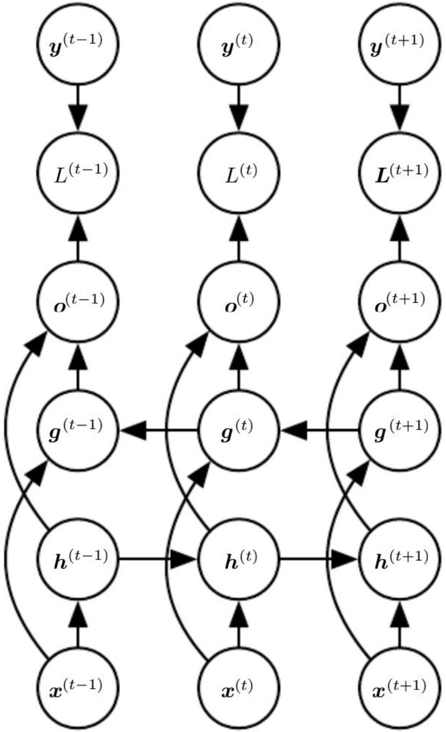
Neural decoding (spykesML)

Countless others

Above all: easy to train. Crazy useful for just about anything with time

BiLSTMs

Architecture of a biLSTM



BiLSTM performance

- In practice, performs better and trains faster than vanilla LSTM
- Shorter paths to much of the information. Shorter path = better gradients
- Speculation as to why exactly it is better - i.e ensures past and future data are equally weighted - but no one really knows.

Success stories

SOTA in POS tagging 2015 (*Bidirectional LSTM-CRF Models for Sequence Tagging*)

SOTA in speech recognition 2013

Above all: it is reasonably fast to train. Workhorse for countless research applications

InferSent

- Achieved good results in:
  - Semantic Textual Similarity
  - Paraphrase detection
  - Caption-image retrieval
- Outperforms newer, more sophisticated models like BERT in tasks such as Semantic Textual Similarity

Supervised Learning of Universal Sentence Representations from Natural Language Inference Data

Alexis Conneau
Facebook AI Research
aconneau@fb.com

Douwe Kiela
Facebook AI Research
dkiela@fb.com

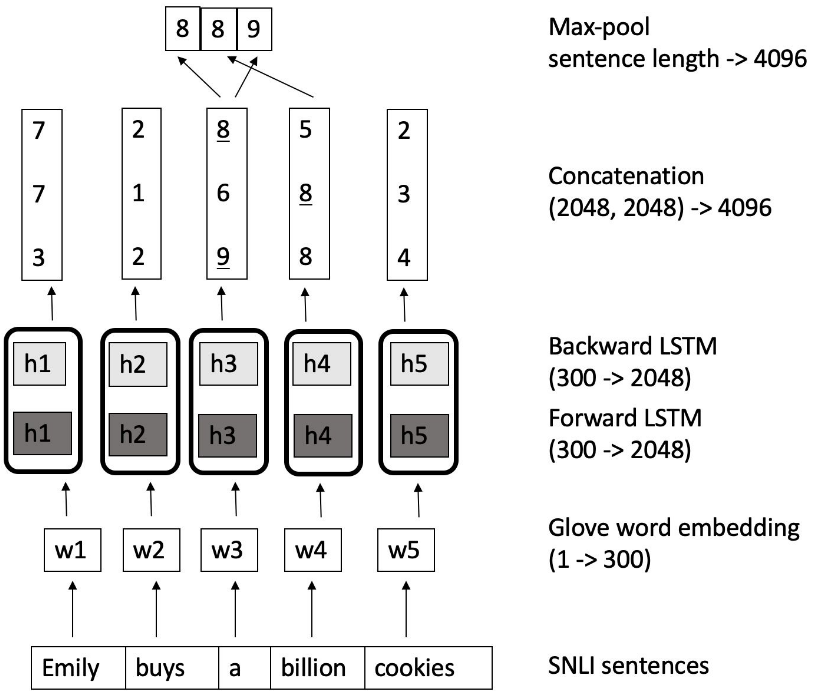
Holger Schwenk
Facebook AI Research
schwenk@fb.com

Loïc Barrault
LIUM, Université Le Mans
loic.barrault@univ-lemans.fr

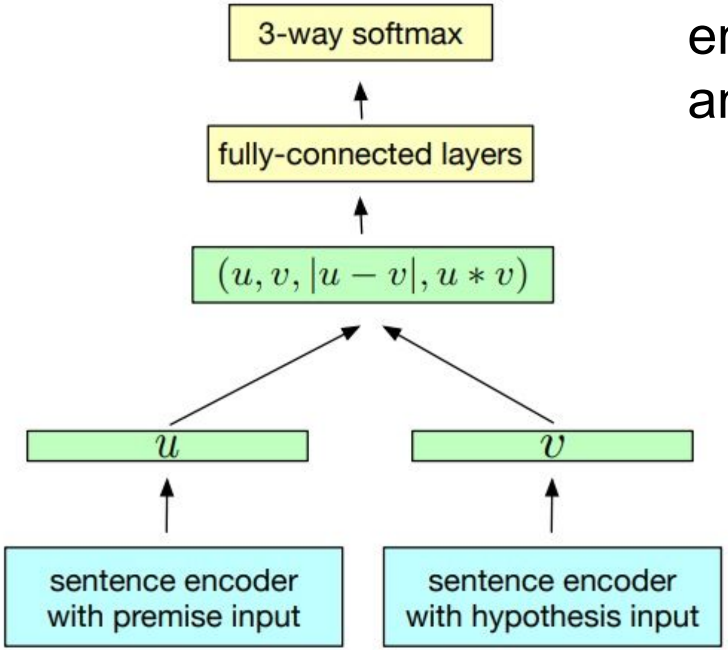
Antoine Bordes
Facebook AI Research
abordes@fb.com

InferSent architecture

Model Architecture



Finetuning InferSent fro Natural Language Inference

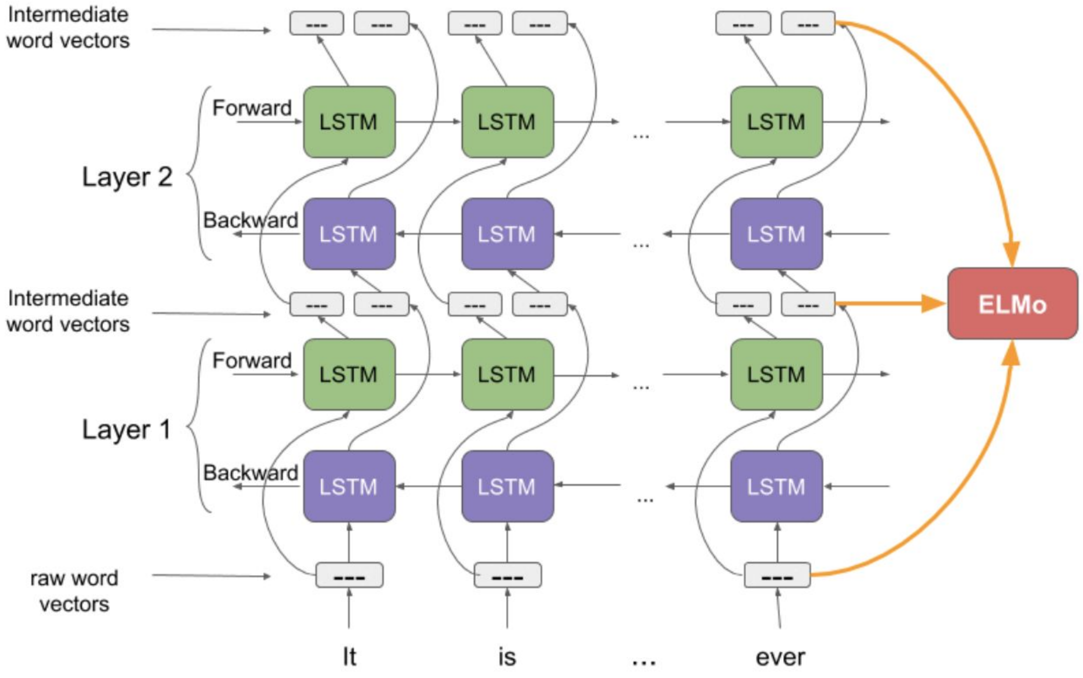


entailment, contradiction and neutral

Embeddings from Language Models (ELMo)

- BiLSTM developed in 2018 that generates word embeddings based on the context it appears in
- Achieved state of the art in many NLP tasks including:
  - Question Answering
  - Sentiment Analysis for movie reviews
  - Sentiment Analysis
- Has since been surpassed by BERT-like architectures which we will cover soon!

ELMo architecture



What we learned today

- Discussed modern NLP for variable-length problems
- RNNs
  - Architecture
  - Backpropagation through time (BPTT)
- LSTMs
  - Motivation
  - Forget gates
- BiLSTMs
  - Motivation
  - Architecture

PyTorch implementation

LSTM

```
# Recurrent neural network (many-to-one)
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, num_classes):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        # Set initial hidden and cell states
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

        # Forward propagate LSTM
        out, _ = self.lstm(x, (h0, c0))  # out: tensor of shape (batch_size, seq_length, hidden_size)

        # Decode the hidden state of the last time step
        out = self.fc(out[:, -1, :])
        return out

model = RNN(input_size, hidden_size, num_layers, num_classes).to(device)
```

biLSTM

```
# Bidirectional recurrent neural network (many-to-one)
class BiRNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, num_classes):
        super(BiRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, bidirectional=True)
        self.fc = nn.Linear(hidden_size*2, num_classes)  # 2 for bidirection

    def forward(self, x):
        # Set initial states
        h0 = torch.zeros(self.num_layers*2, x.size(0), self.hidden_size).to(device) # 2 for bidirection
        c0 = torch.zeros(self.num_layers*2, x.size(0), self.hidden_size).to(device)

        # Forward propagate LSTM
        out, _ = self.lstm(x, (h0, c0))  # out: tensor of shape (batch_size, seq_length, hidden_size*2)

        # Decode the hidden state of the last time step
        out = self.fc(out[:, -1, :])
        return out
```