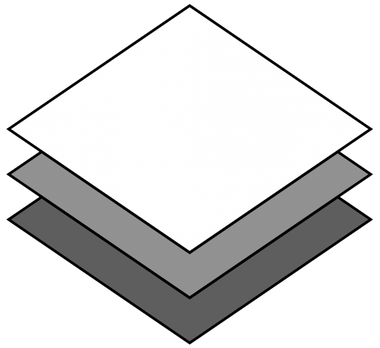


Week 4
Recurrent Neural Networks

Agenda

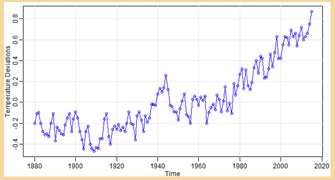
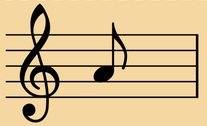
1. Sequence Problems
2. RNNs
3. Vanishing gradients and LSTMs
4. Case study: Machine Translation
   (Bidirectionality and Attention)
5. CTC loss
6. Pros and Cons
7. A preview of non-recurrent sequence models

Agenda

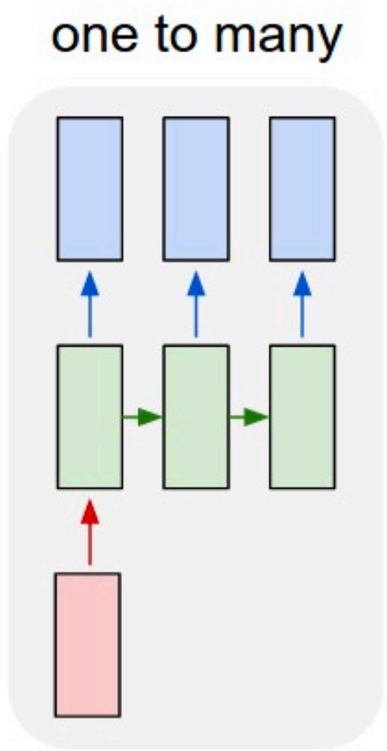
1. **Sequence Problems**
2. RNNs
3. Vanishing gradients and LSTMs
4. Case study: Machine Translation (Bidirectionality and Attention)
5. CTC loss
6. Pros and Cons
7. A preview of non-recurrent sequence models

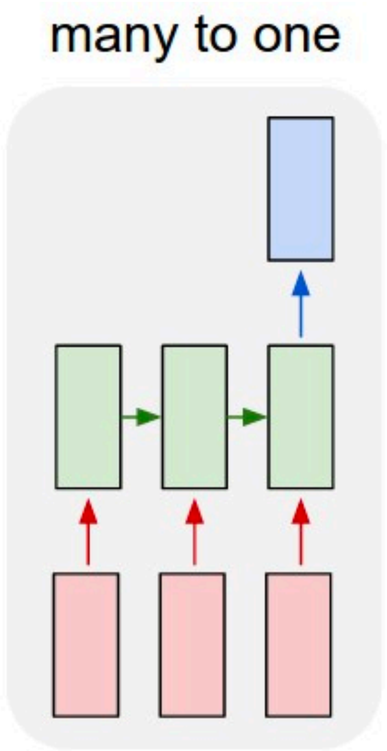
Sequence problems

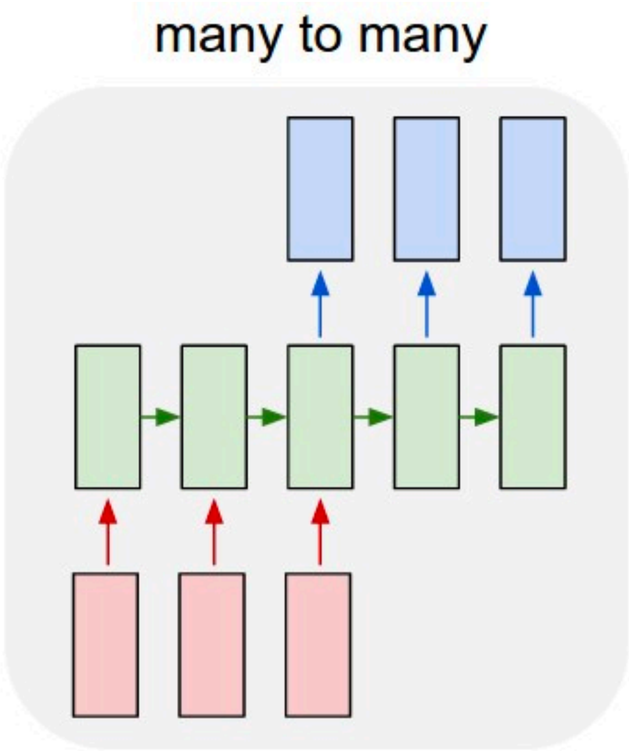
Sequence

Time series forecasting	Time series		→	Predicted next value
Sentiment classification	Review text	6/9/2018 As someone who's taken courses at CSUs, community colleges, and a UC, if you can afford to go to Berkeley, take the chance and go for it and jump in for the full 4 years. It will open up your mind to so many different people from so many different backgrounds, far more than you'd ever get if you stayed home with Ma and Pops.	→	Predicted sentiment 
Translation	English text	welcome to full stack deep learning! 36/5000	↔	French text bienvenue à l'apprentissage en profondeur de la pile! Suggest an edit
Speech recognition and generation	Audio waveform 		↔	Text
Text or music generation	Ø		→	Text or 
Image captioning	Image 		→	Description *“The quick brown fox jumped over the lazy dog”*
Question Answering	Text	**Passage Segment** ...The European Parliament and the Council of the European Union have powers of amendment and veto during the legislative process... **Question** Which governing bodies have veto power?	→	Text ...The European Parliament and the Council of the European Union have

Types of sequence problems





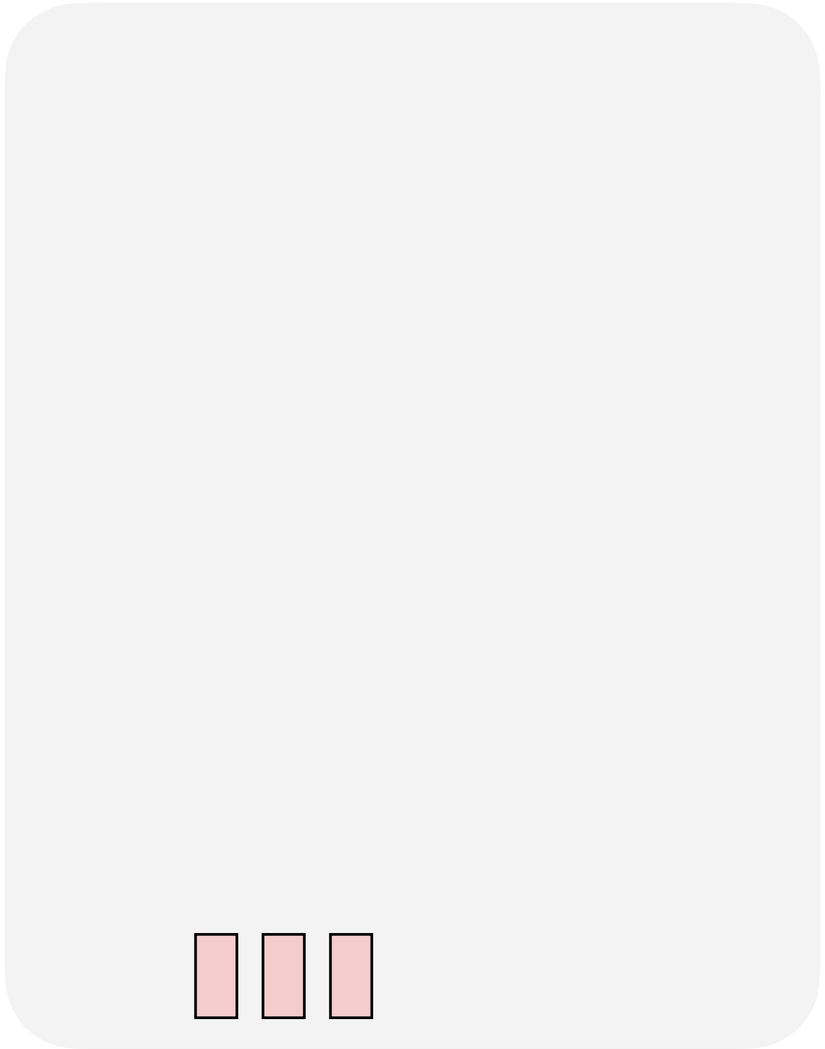


(From http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

Questions?

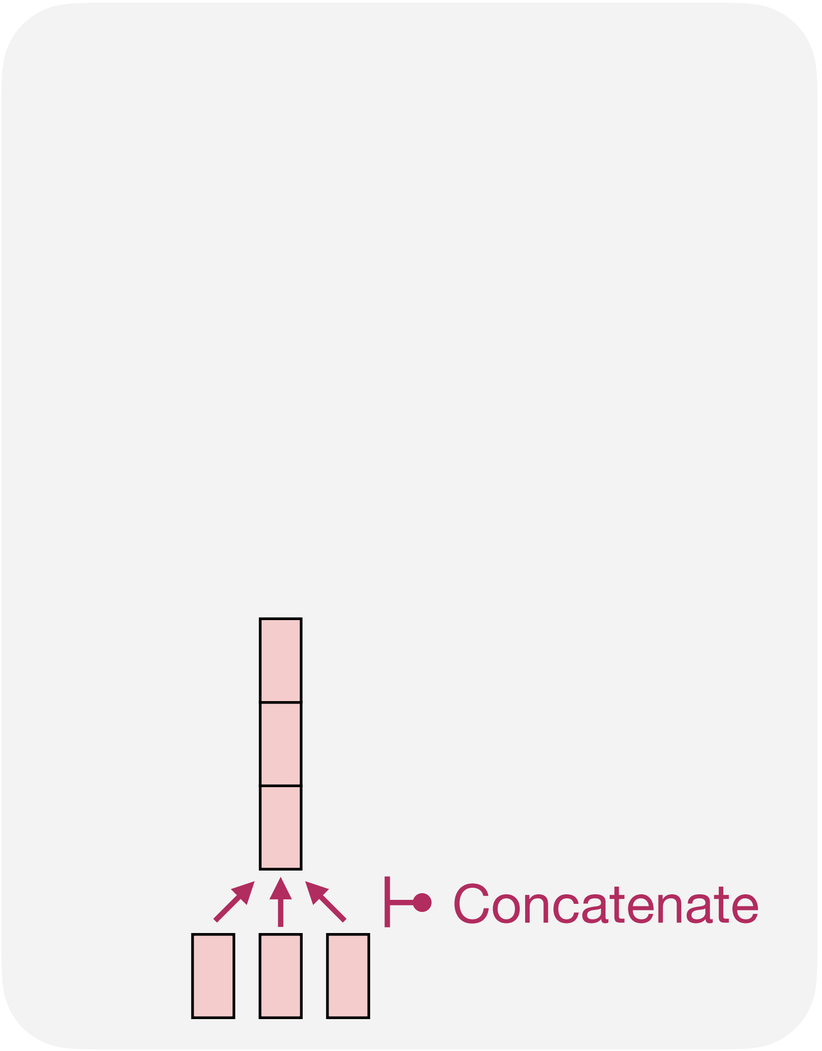
Why not use feedforward networks?

many to many



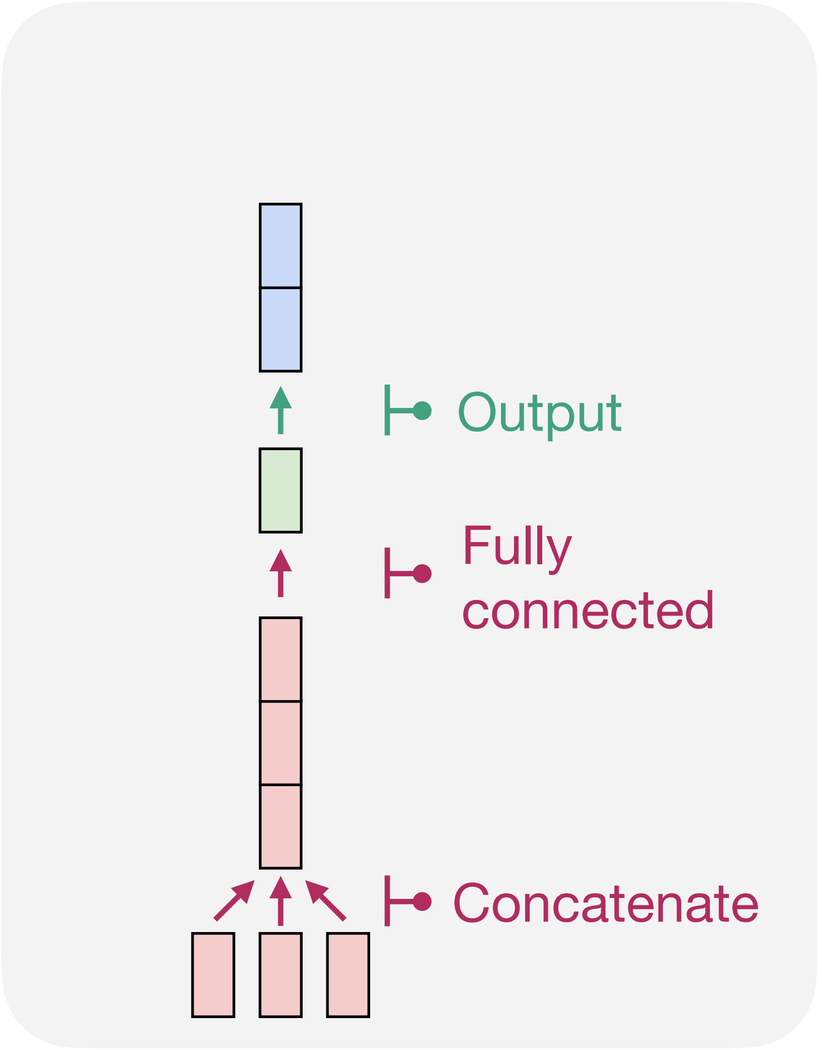
Why not use feedforward networks?

many to many

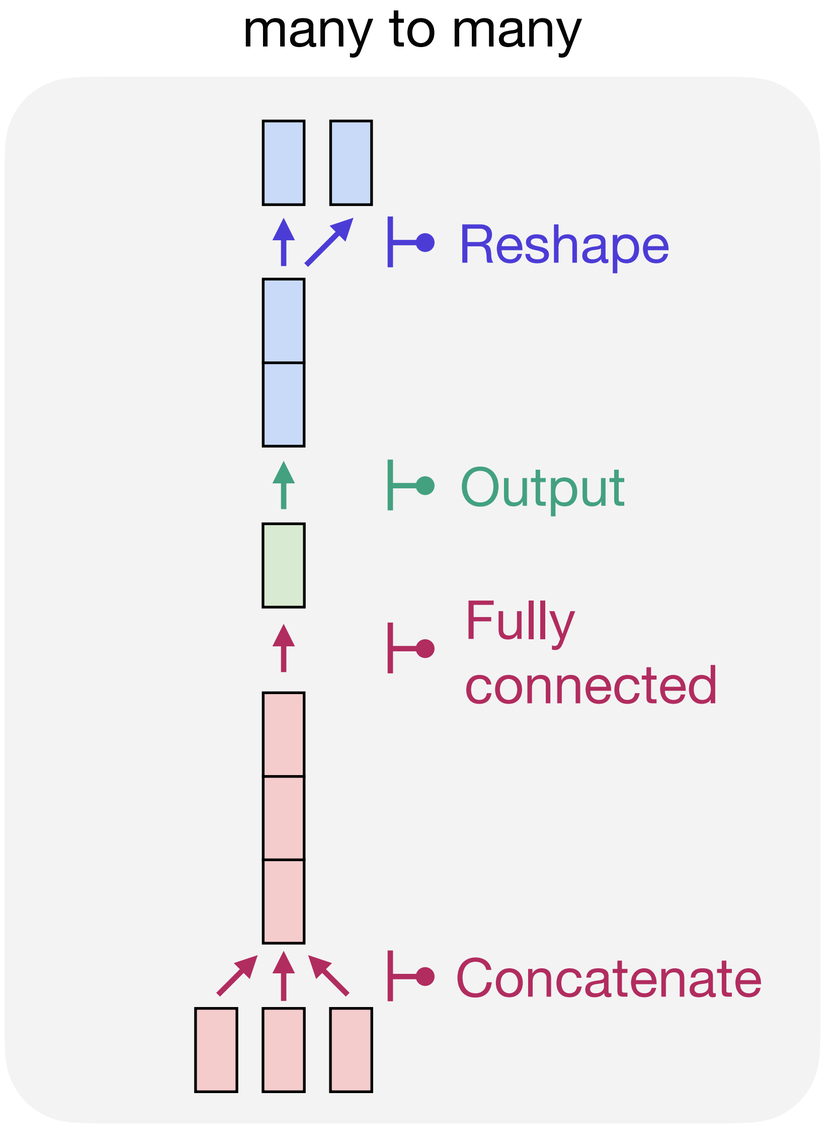


Why not use feedforward networks?

many to many

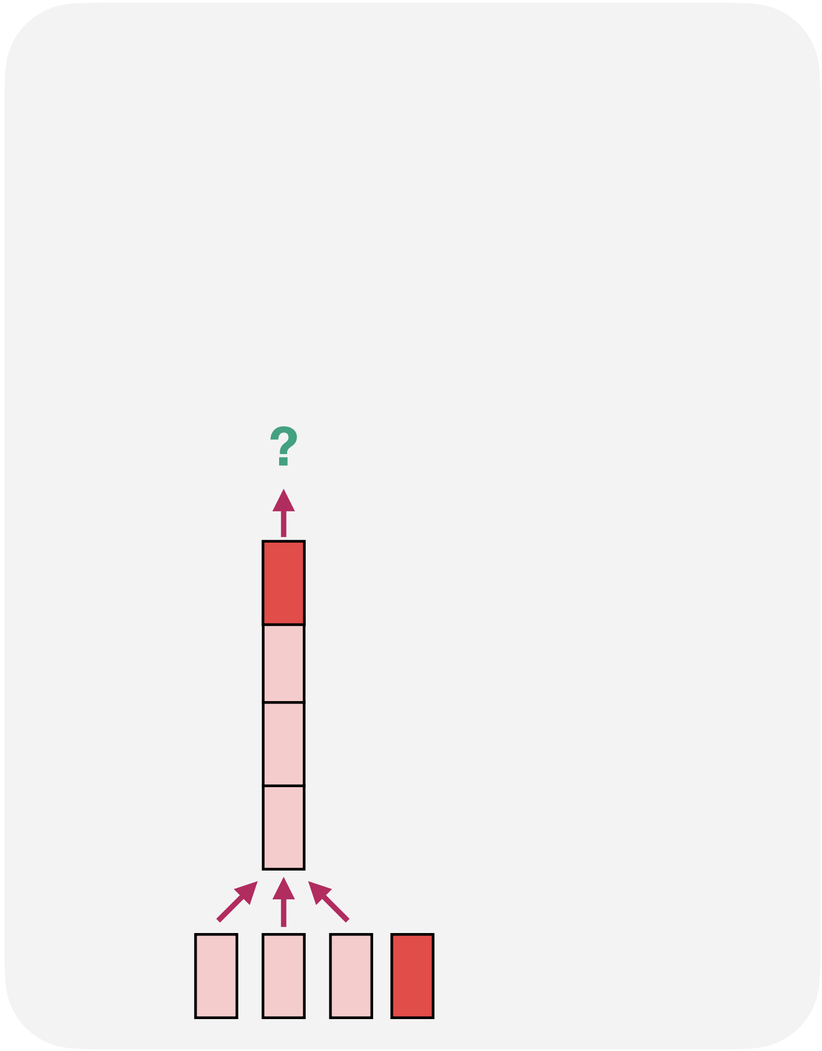


Why not use feedforward networks?



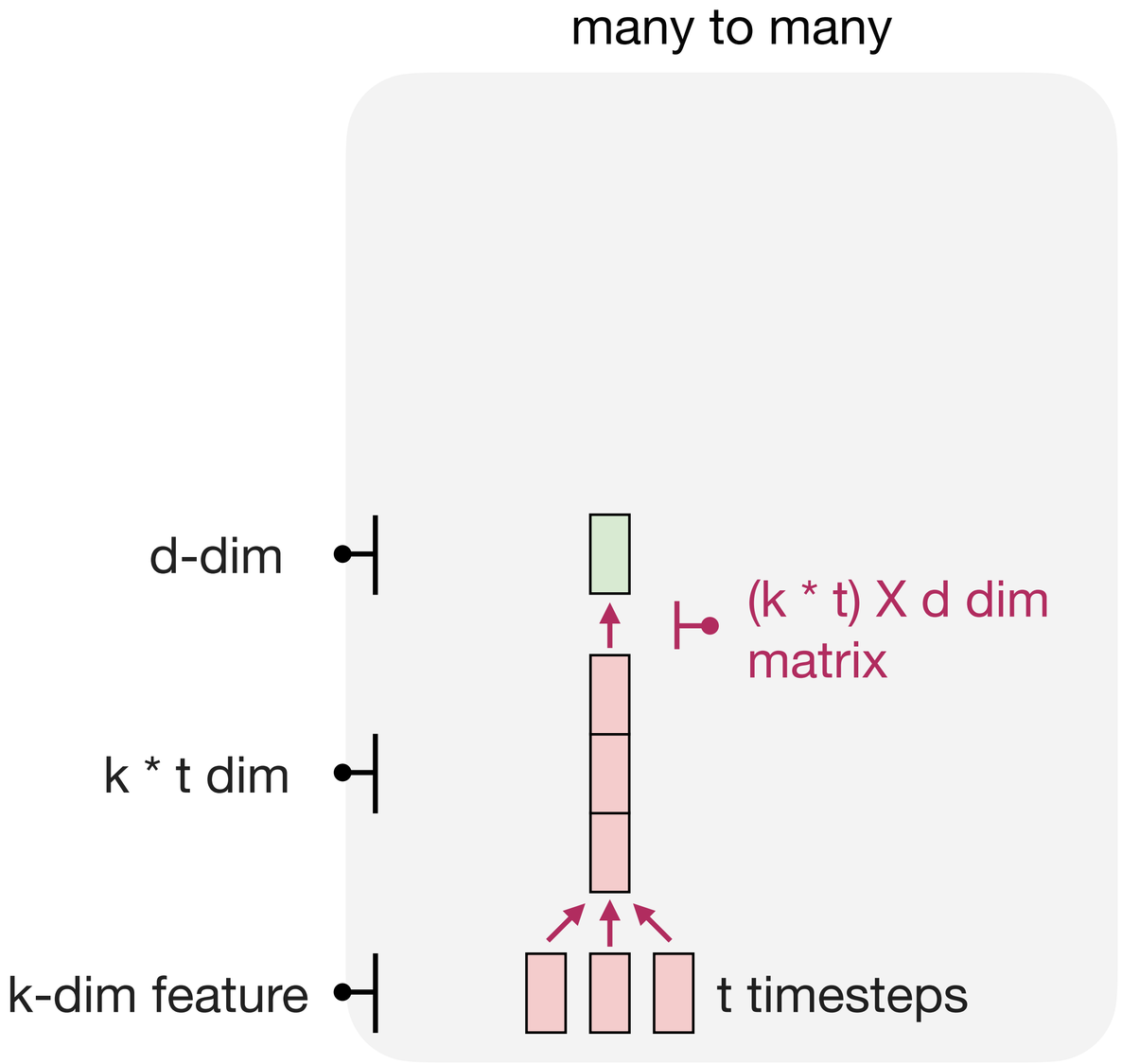
Problem 1: variable length inputs

many to many



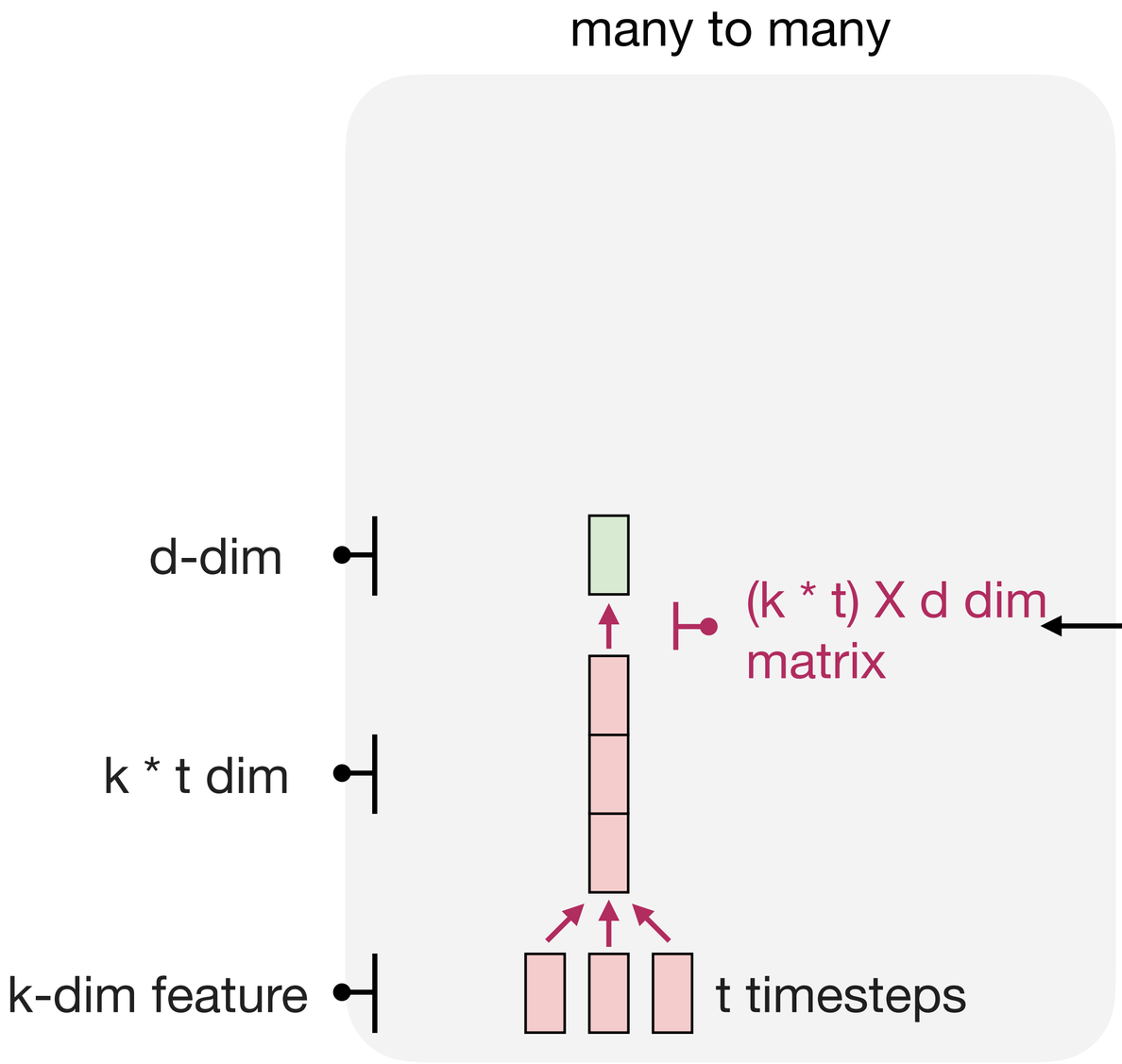
Can deal with this by padding all sequences to the max length, but…

Problem 2: memory scaling



Memory requirement scales linearly in number of timesteps

Problem 3: overkill



This matrix has to learn patterns *everywhere* they may occur in the sequence!

This ignores the nature of the problem: patterns over time.

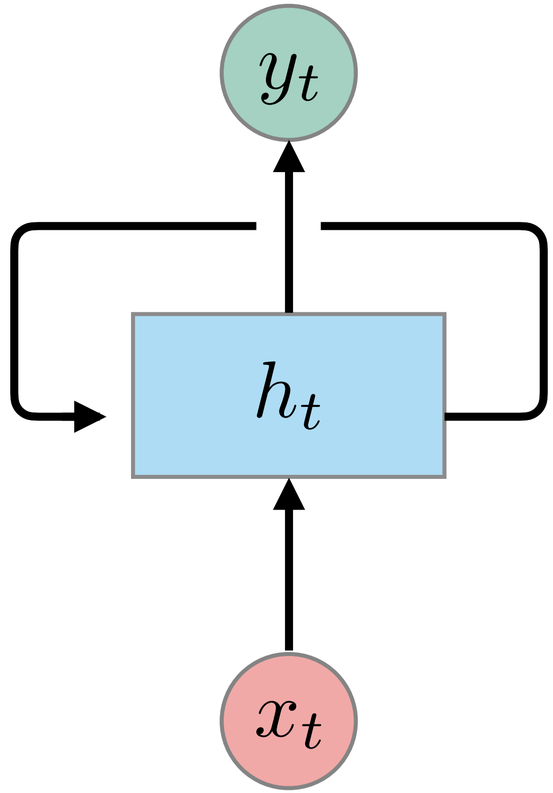
Questions?

Agenda

1. Sequence Problems
2. **RNNs**
3. Vanishing gradients and LSTMs
4. Case study: Machine Translation (Bidirectionality and Attention)
5. CTC loss
6. Pros and Cons
7. A preview of non-recurrent sequence models

Core idea of RNNs

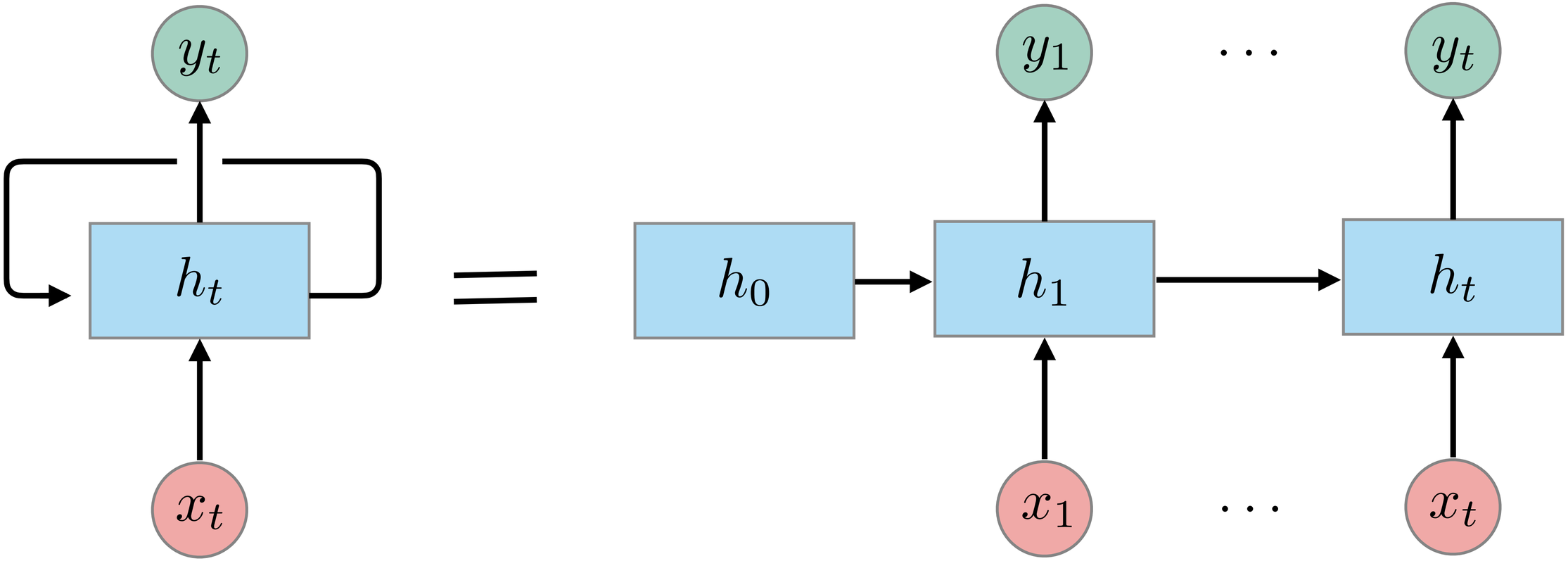
Stateful computation



$$y_t, h_t = f(x_t, h_{t-1})$$

Core idea of RNNs

Stateful computation



$$y_t, h_t = f(x_t, h_{t-1})$$

The RNN in code

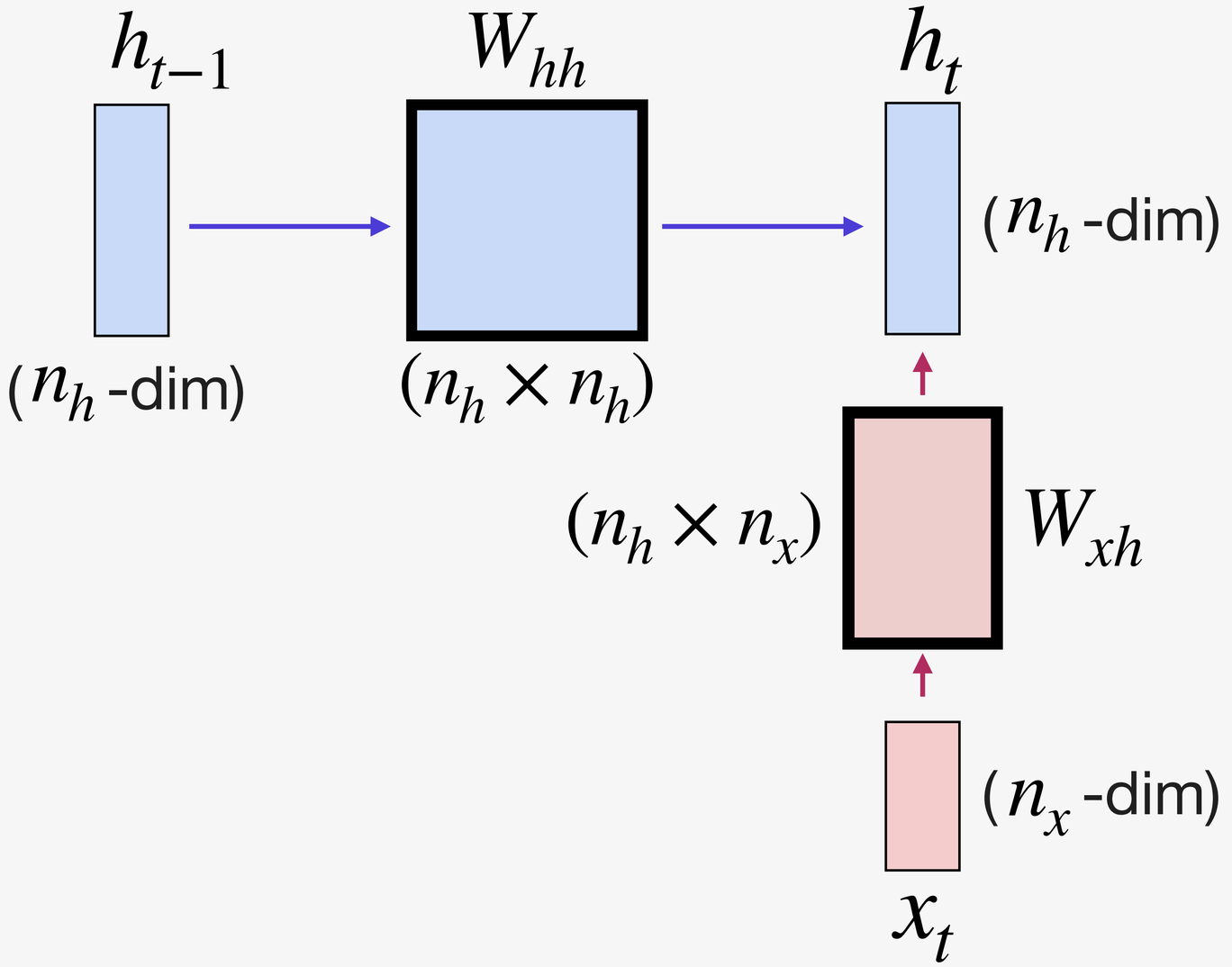
```
class RNN:
    # ...
    def compute_next_h(self, x):
        # Simplest hidden state computation. Will get fancier later.
        h = np.tanh(self.W_hh.dot(self.h) + self.W_xh.dot(x))
        return h

    def step(self, x):
        # Update the hidden state
        self.h = self.compute_next_h(x)
        # Compute the output vector
        y = self.W_hy.dot(self.h)
        return y
```

(*Adapted from* http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

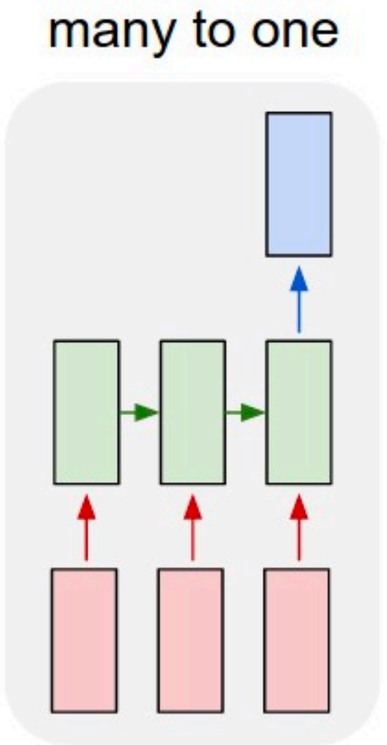
A look at compute_next_h

```
def compute_next_h(self, x):
```



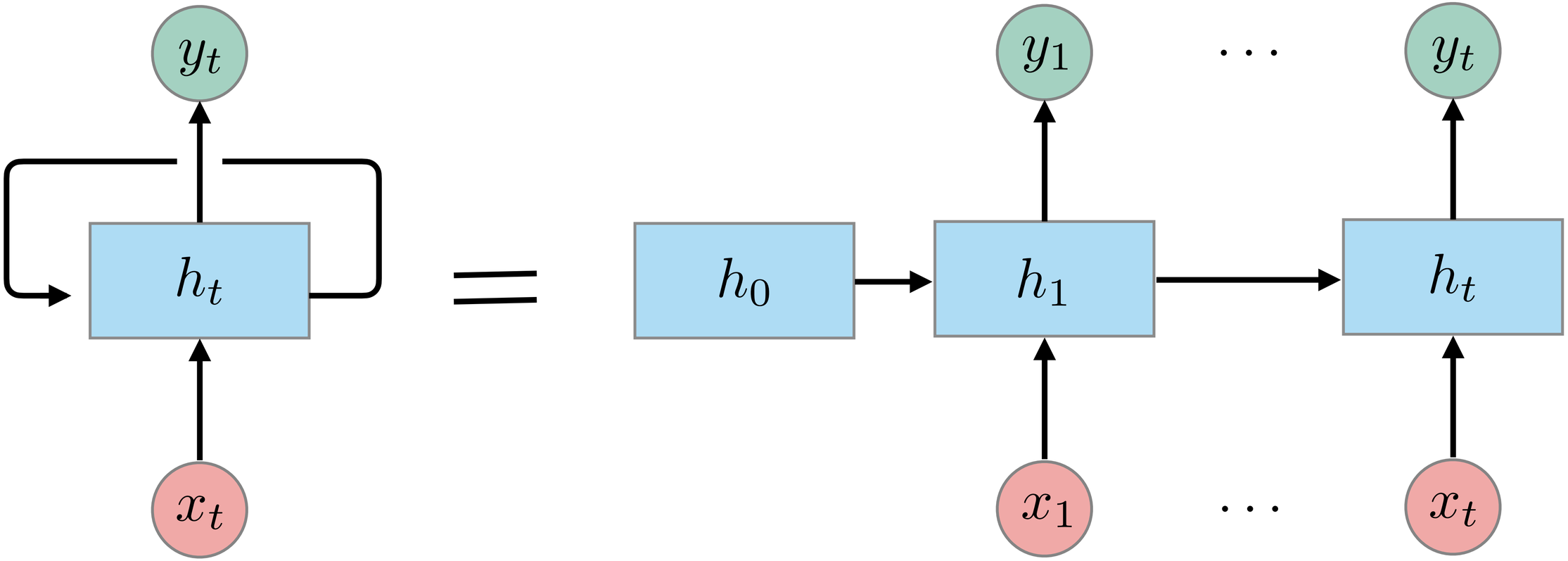
n_h = dim of the RNN

RNNs for **many-to-one** problems



Core idea of RNNs

Stateful computation



$$y_t, h_t = f(x_t, h_{t-1})$$

RNNs for many-to-one problems

Architecture

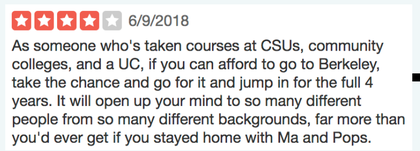
Input

Encoder

State at last timestep

Classifier (Decoder)

Output



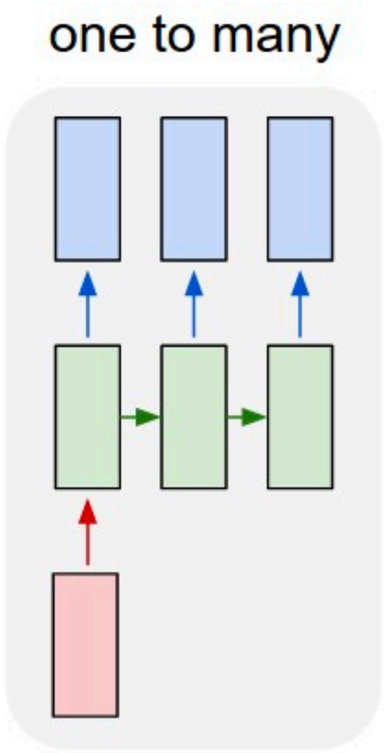
RNN

0.5
0.2
-0.1
-0.3
0.4
1.2

FC

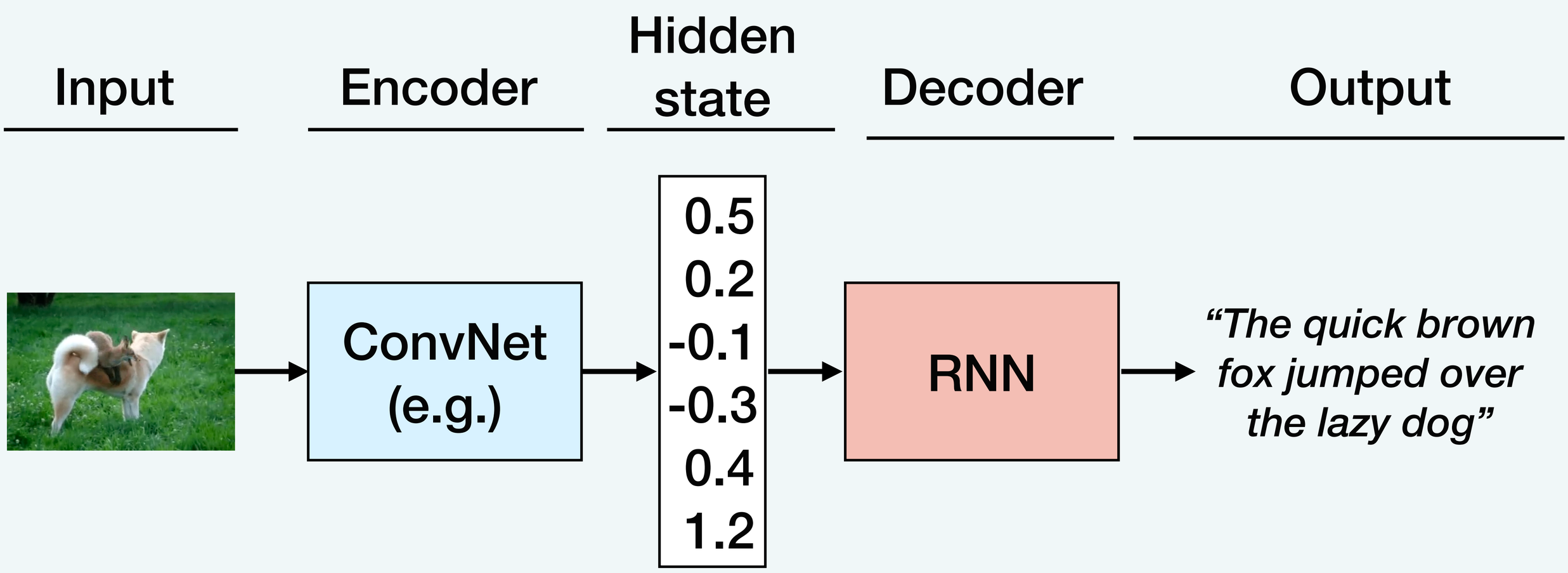
Questions?

RNNs for **one-to-many** problems

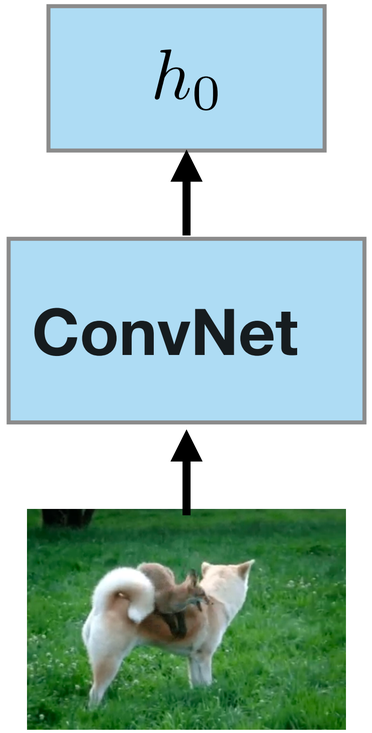


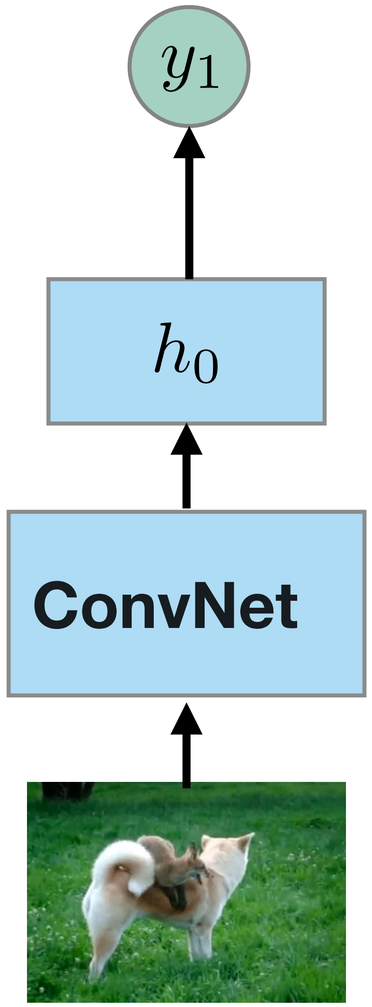
RNNs for one-to-many problems

Encoder-decoder architectures

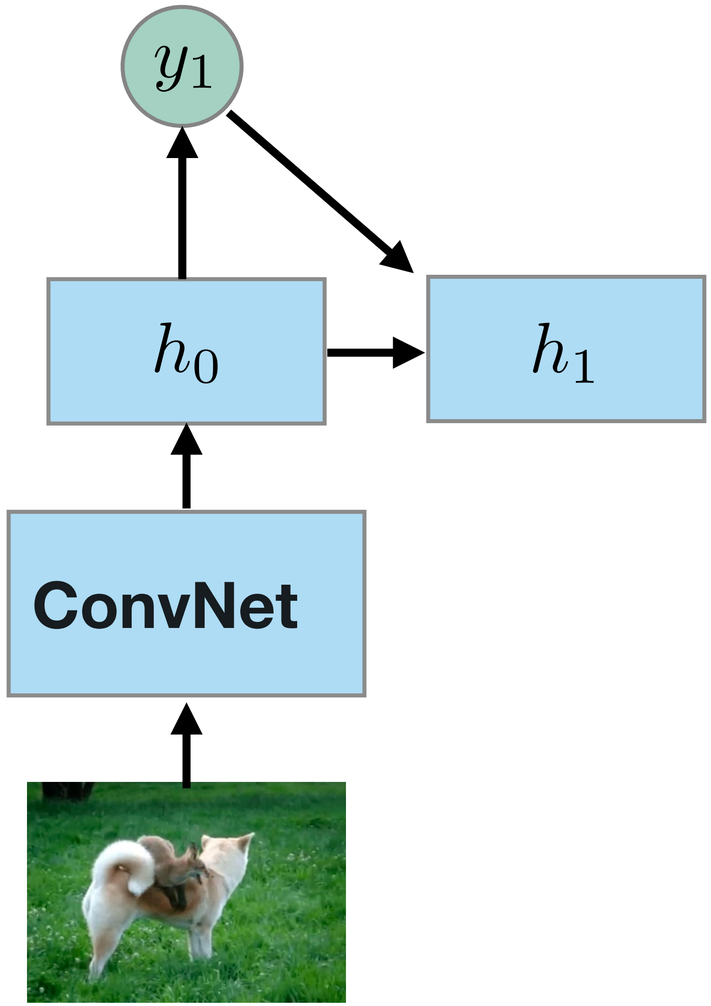


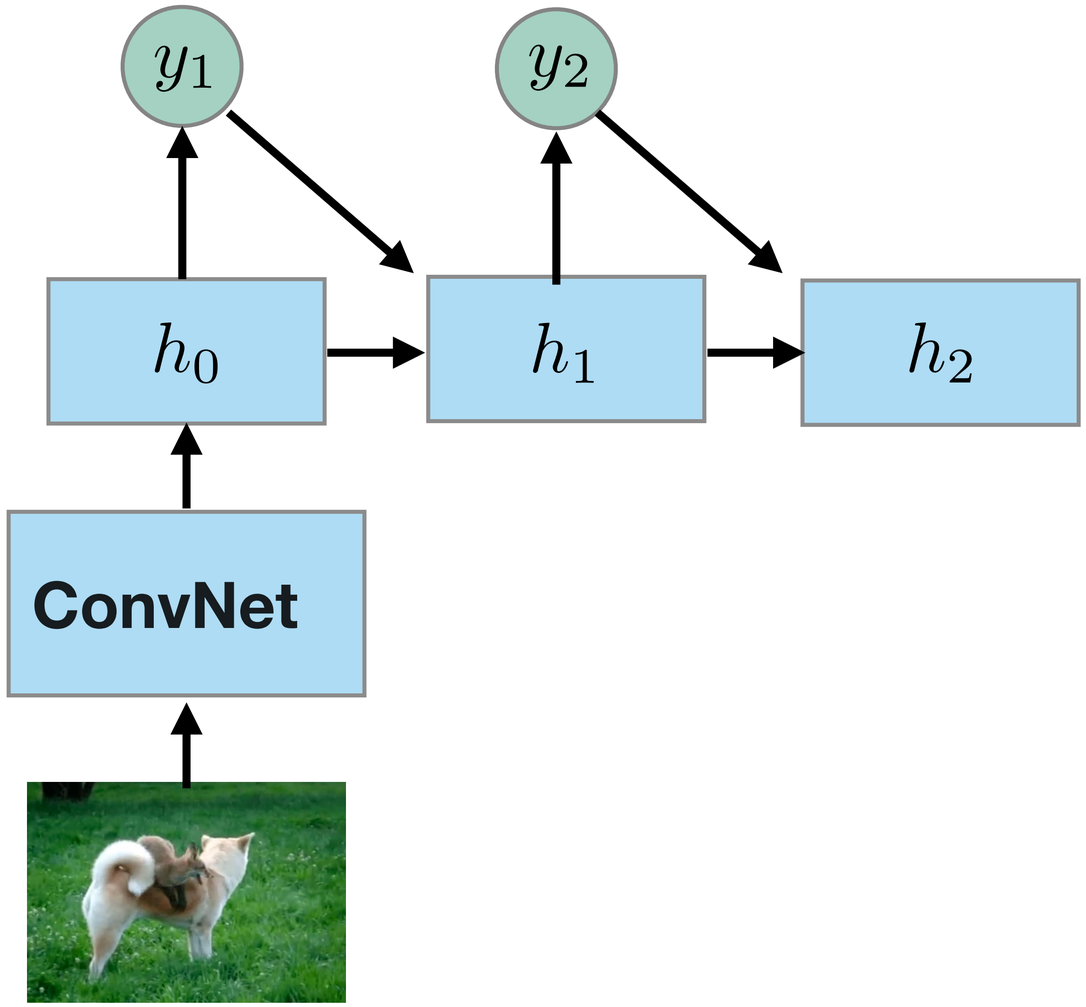
RNNs for **one-to-many** problems



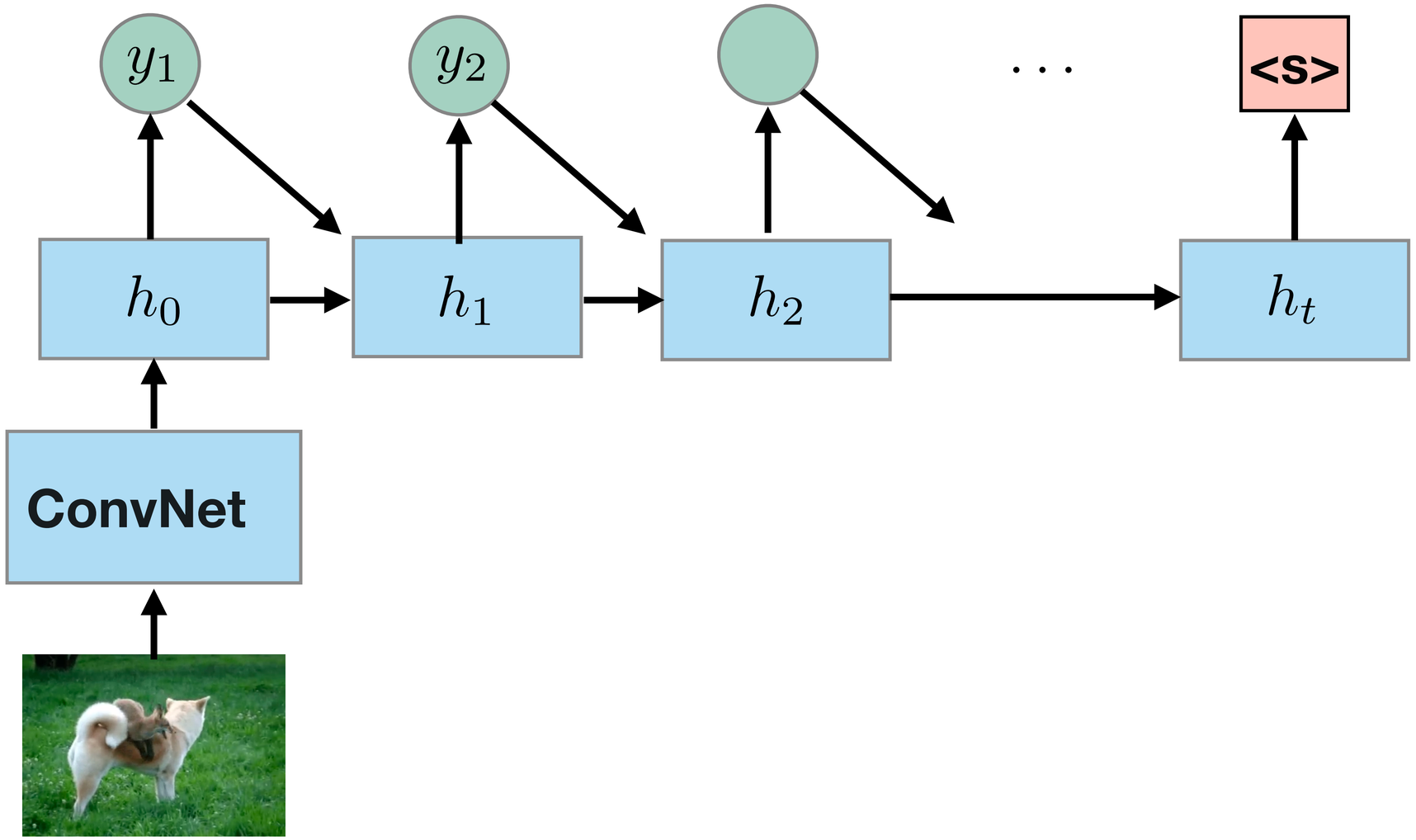

y_1
h_0
ConvNet

Using RNNs for one-to-many problems



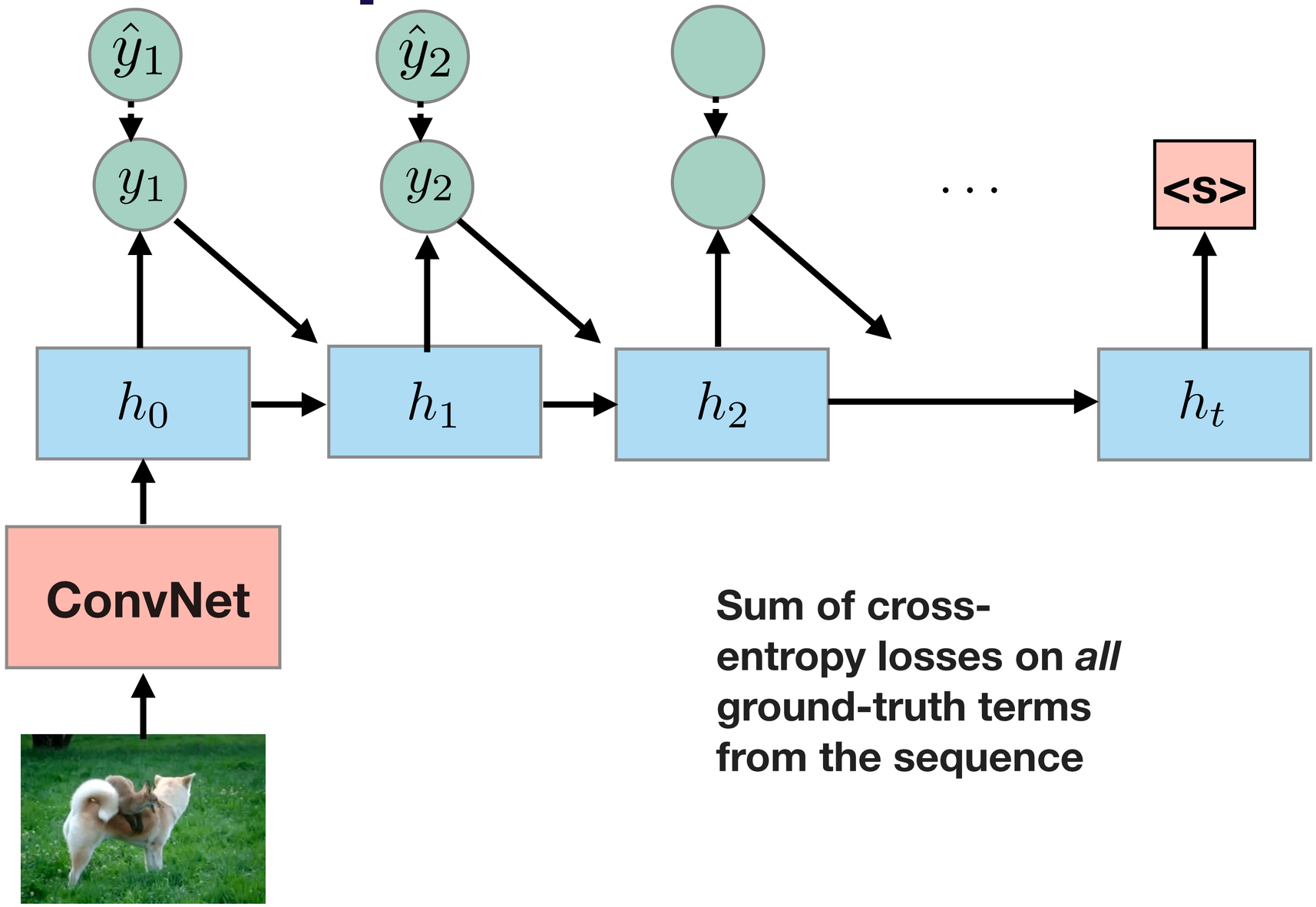

y_1
y_2
h_0
h_1
h_2
ConvNet

Stop characters



Special character that tells the network to stop generating

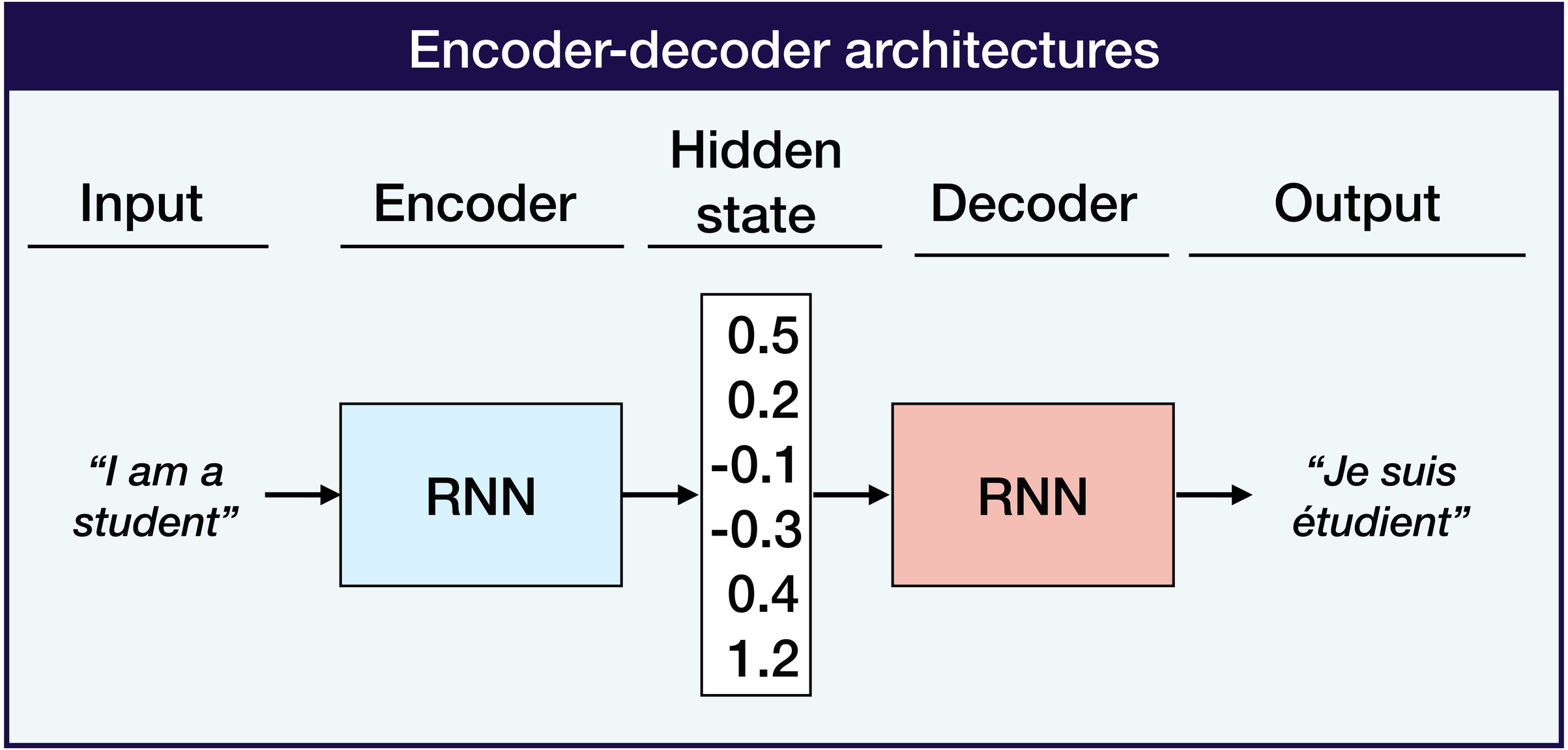
Sequence loss functions



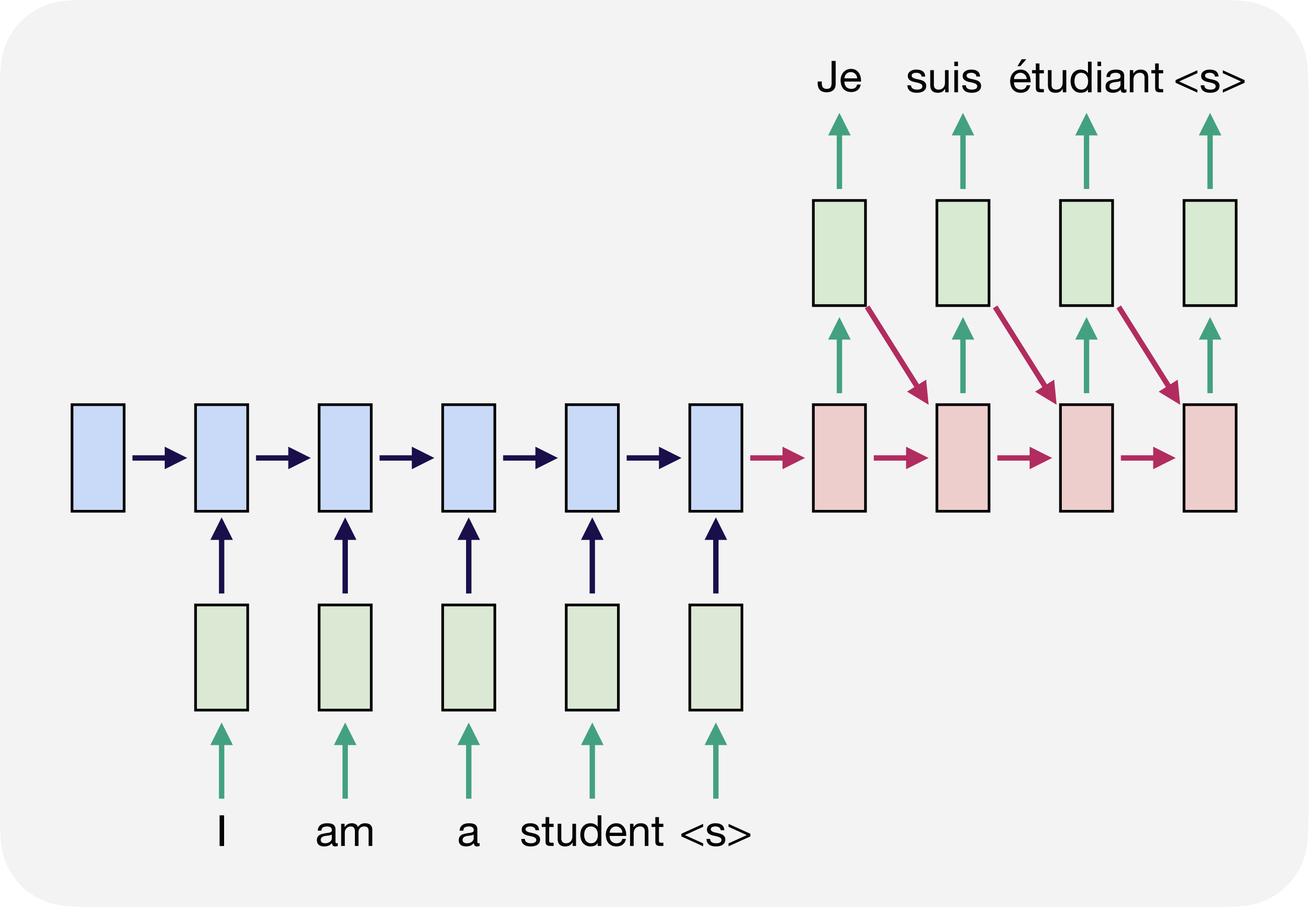
Sum of cross-entropy losses on *all* ground-truth terms from the sequence

Questions?

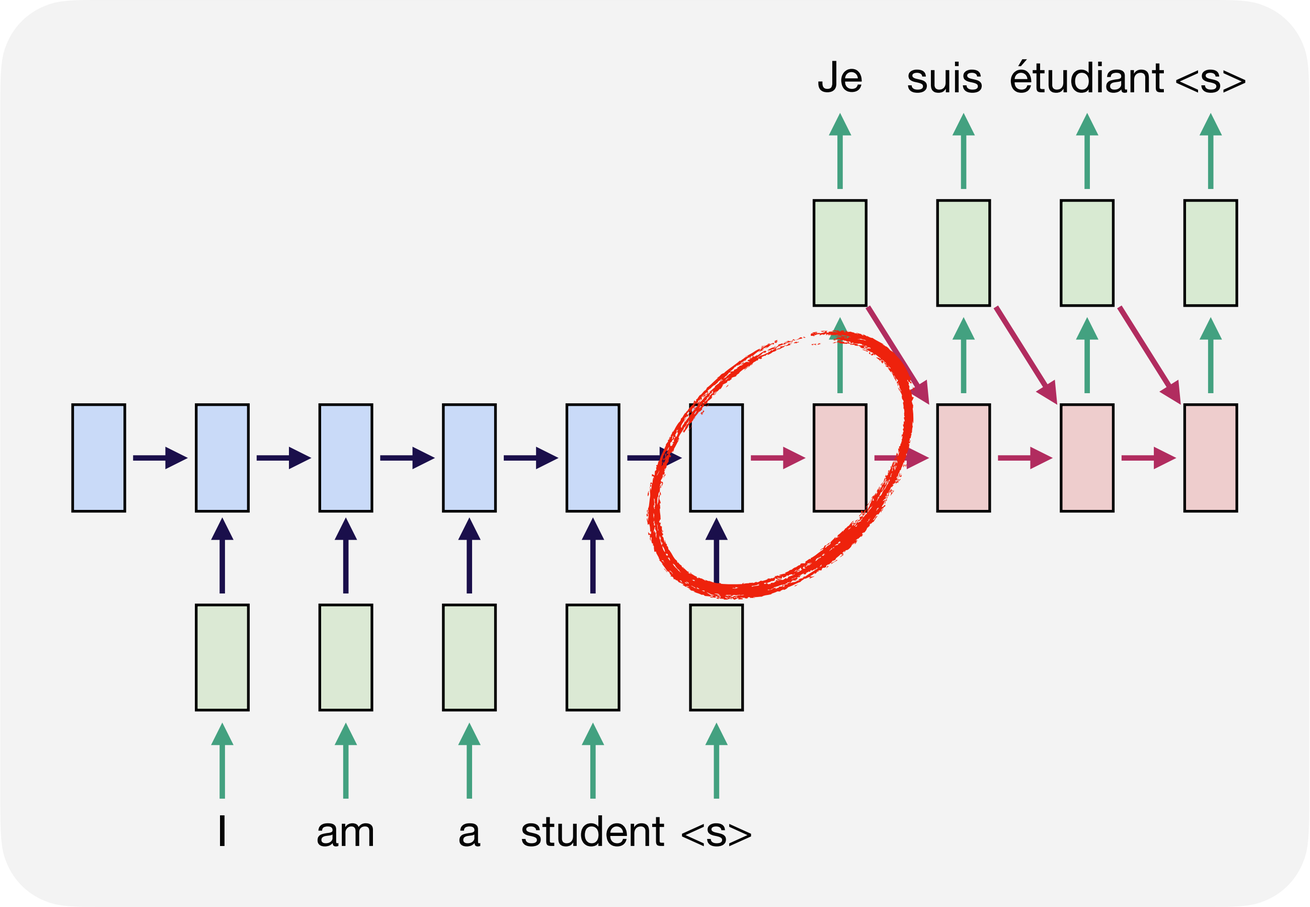
RNNs for many-to-many problems



RNNs for **many-to-many** problems



RNNs for **many-to-many** problems



All the information in the input sentence is condensed into one hidden state vector!

(In practice, we need more tricks for this to work -- explained soon)

Agenda

1. Sequence Problems
2. RNNs
3. **Vanishing gradients and LSTMs**
4. Case study: Machine Translation (Bidirectionality and Attention)
5. CTC loss
6. Pros and Cons
7. A preview of non-recurrent sequence models

RNN Desiderata

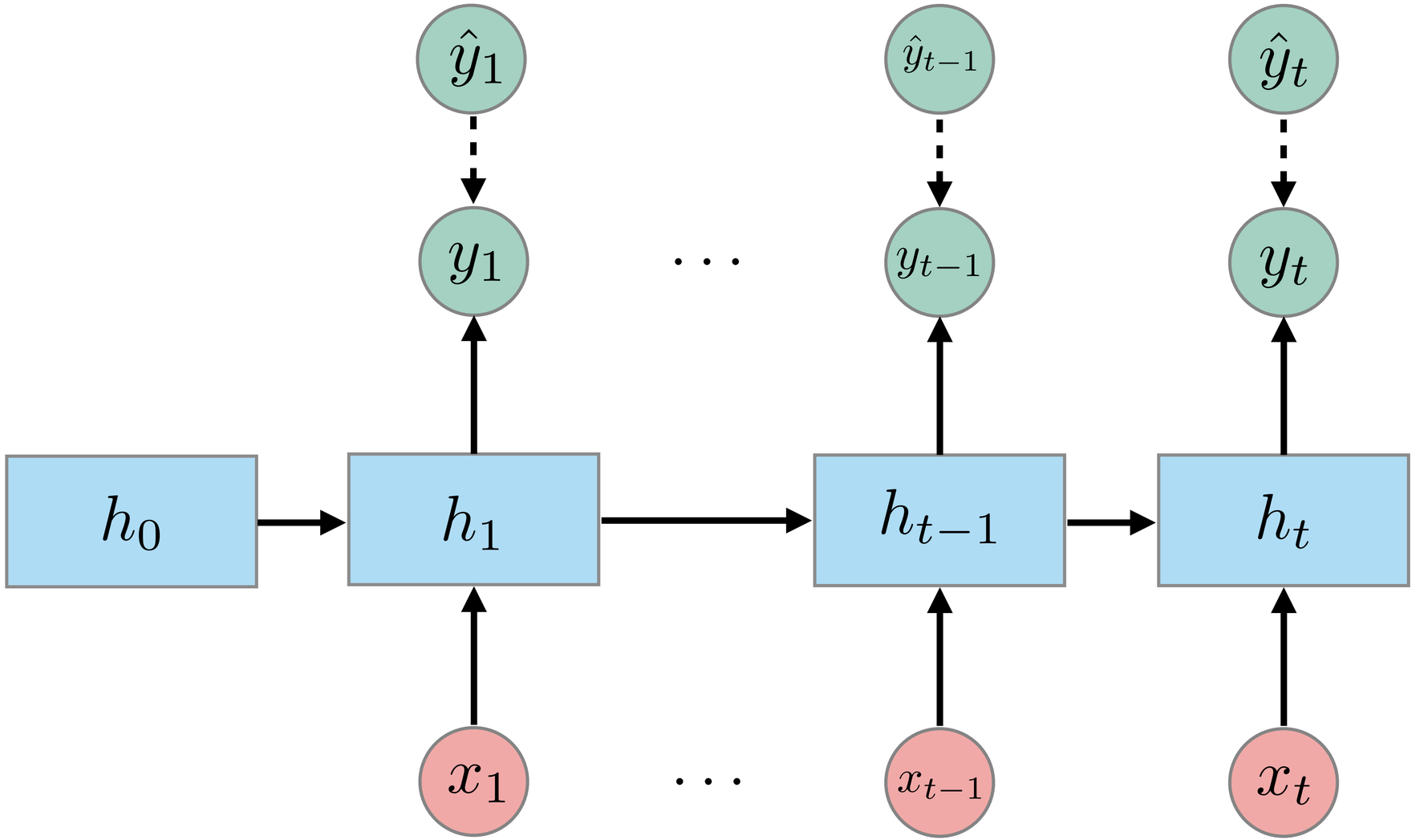
- Goal: handle long sequences
- Connect events from the past to outcomes in the future
  - i.e., *Long-term dependencies*
  - e.g., remember the name of a character from the first sentence

Vanilla RNNs: the reality

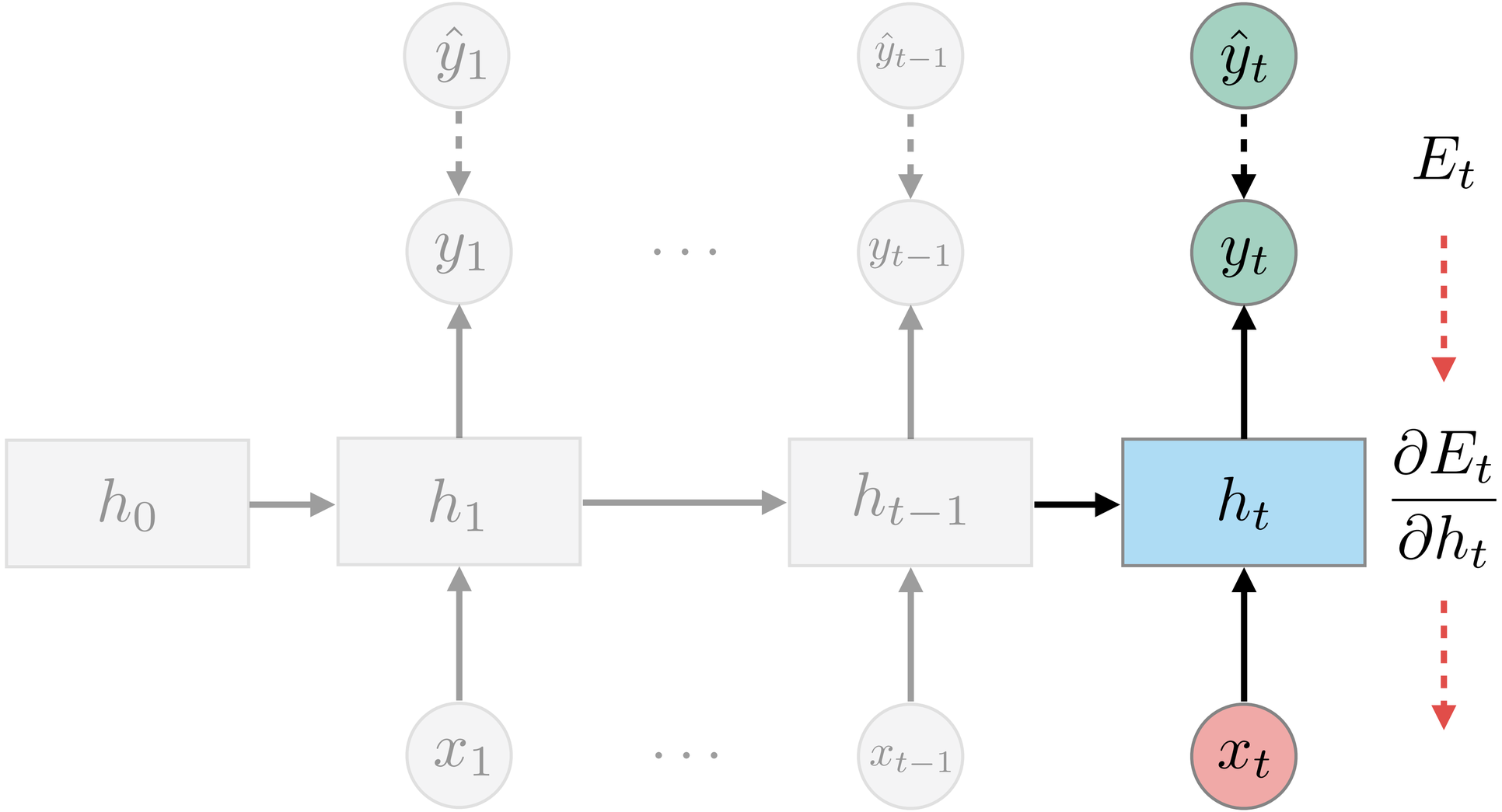
- Can’t handle more than 10-20 timesteps
- Longer-term dependencies get lost
- Why? ***Vanishing gradients***

https://bair.berkeley.edu/blog/2018/08/06/recurrent/

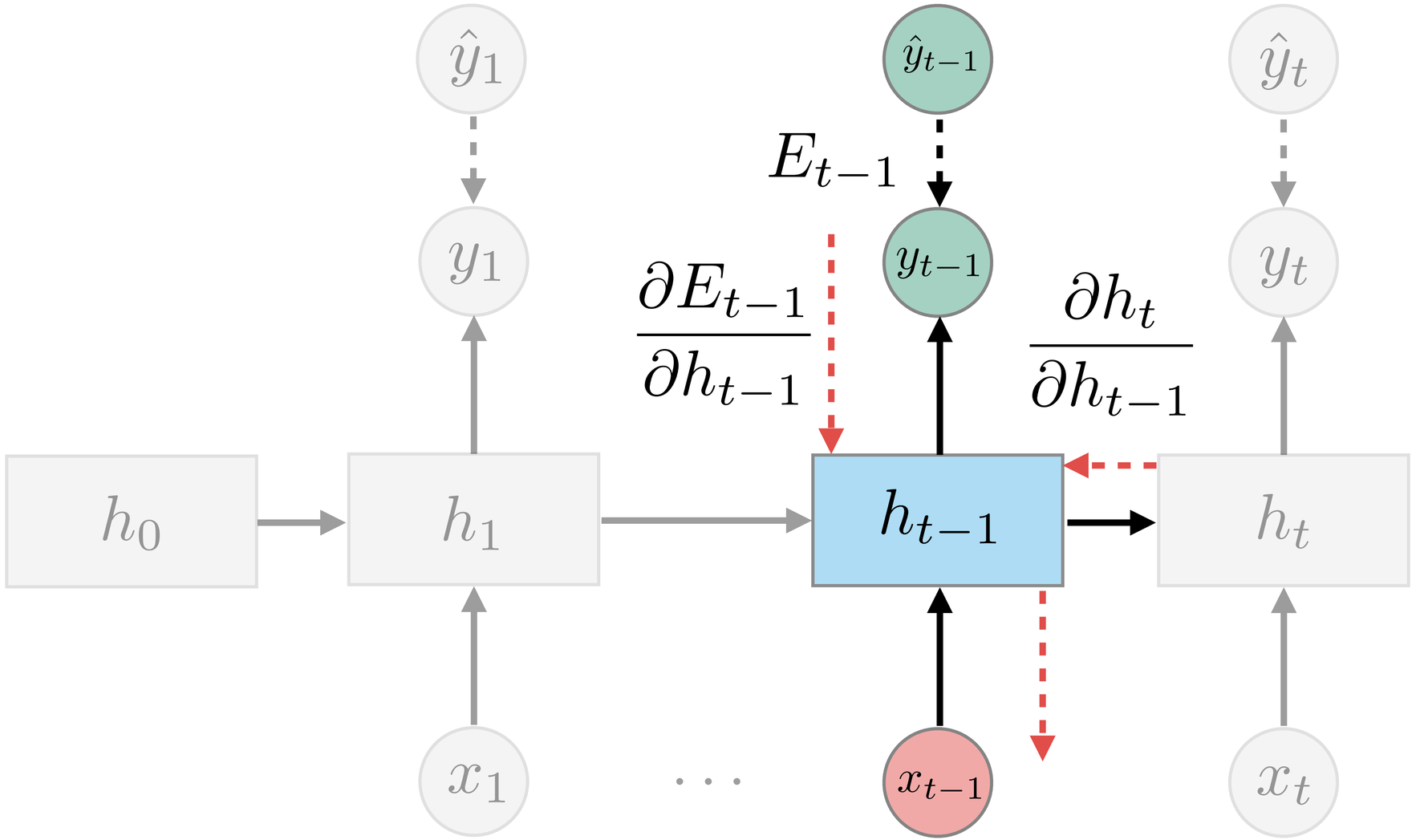
Backpropagation through time



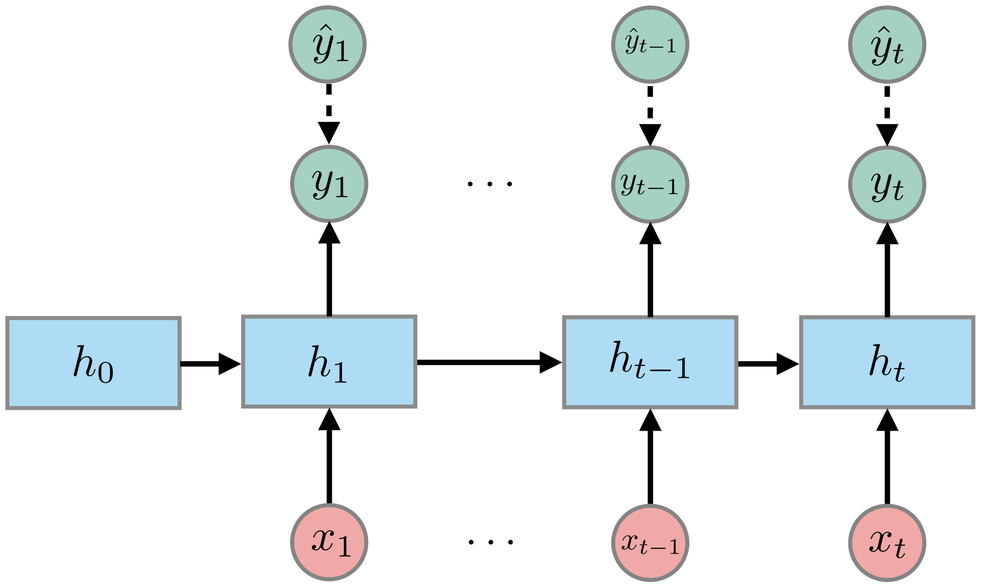
Backpropagation through time



Gradients in RNNs: backpropagation through time



Gradients in RNNs: backprapogation through time



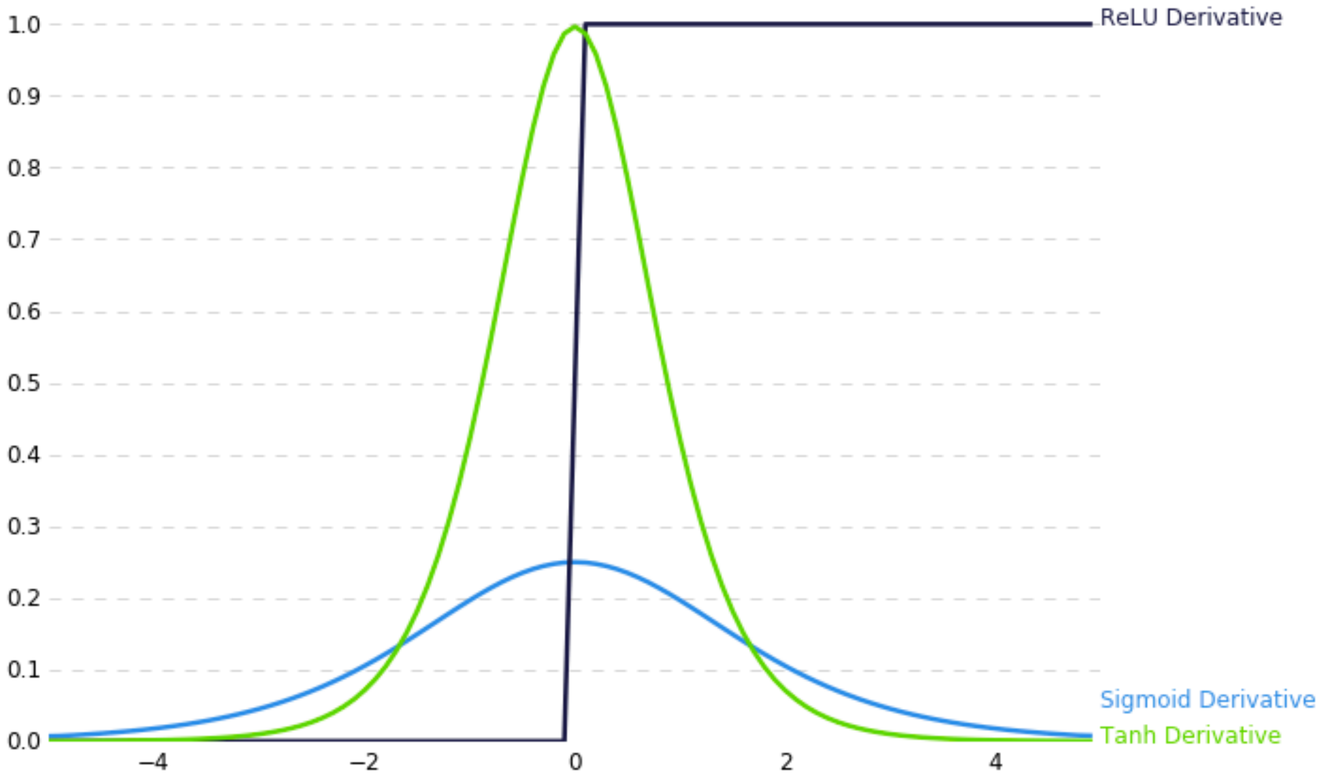
$\frac{\partial E_t}{\partial W_{hh}}$ depends on each $\frac{\partial h_j}{\partial h_{j-1}}$

Recall $h_j = \tanh(W_{hh}h_{j-1} + W_{xh}x_j)$

So each $\frac{\partial h_j}{\partial h_{j-1}}$ introduces a $\tanh'$

Back to vanishing gradients

Derivatives of common activation functions



- sigmoid and tanh have derivatives << 1 near saturation
- At each step magnitude of gradients tends to decrease
- After enough steps gradients are too small
- (ReLU RNNs often have the opposite problem - exploding gradients)

Questions?

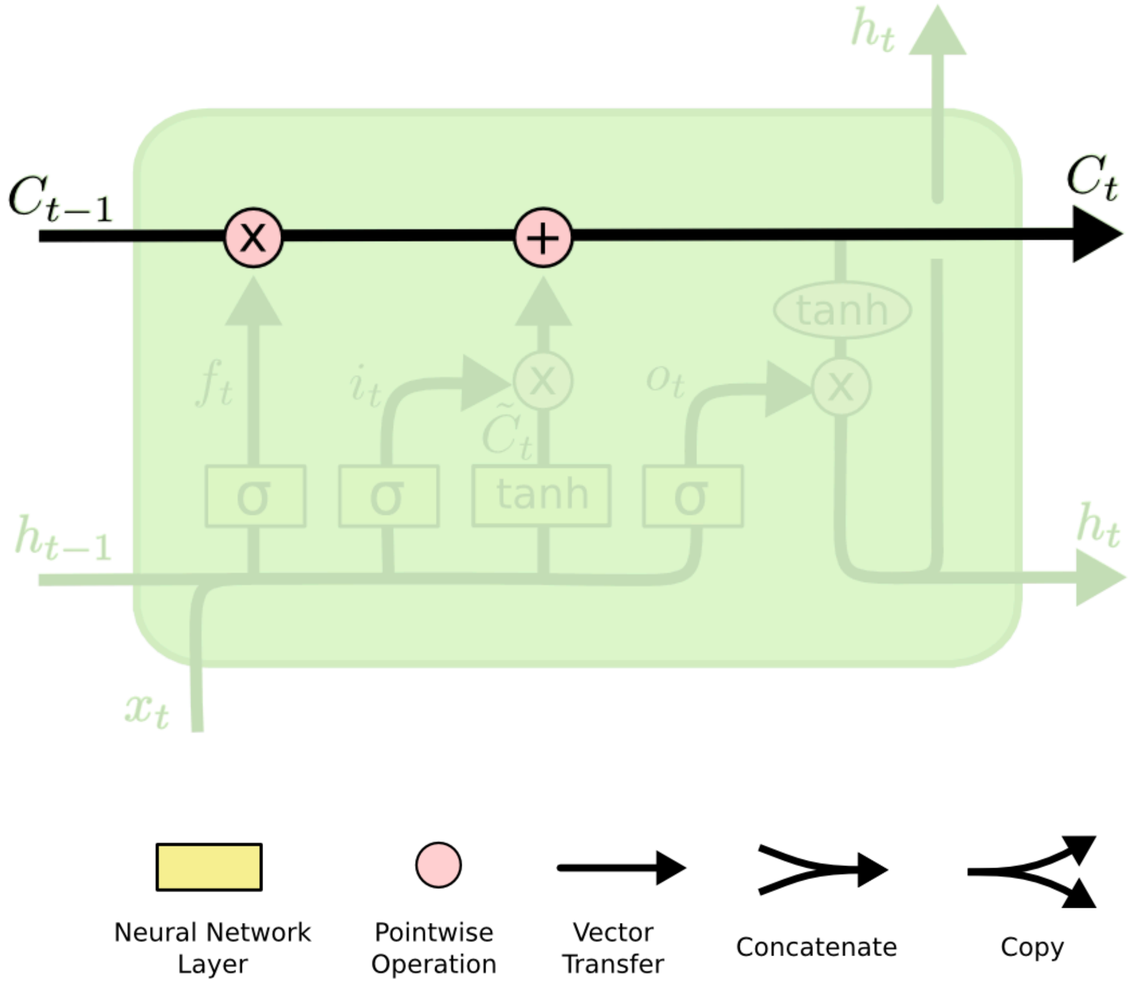
Intro to LSTMs

Idea: use a compute_next_h that preserves gradients

```
class LSTM(RNN):
    # ...
    def compute_next_h(self, x):
        h = lstm(x, self.h)
        # or gru(x, self.h), etc.
        return h
    # ...
```

For more info, see http://colah.github.io/posts/2015-08-Understanding-LSTMs/

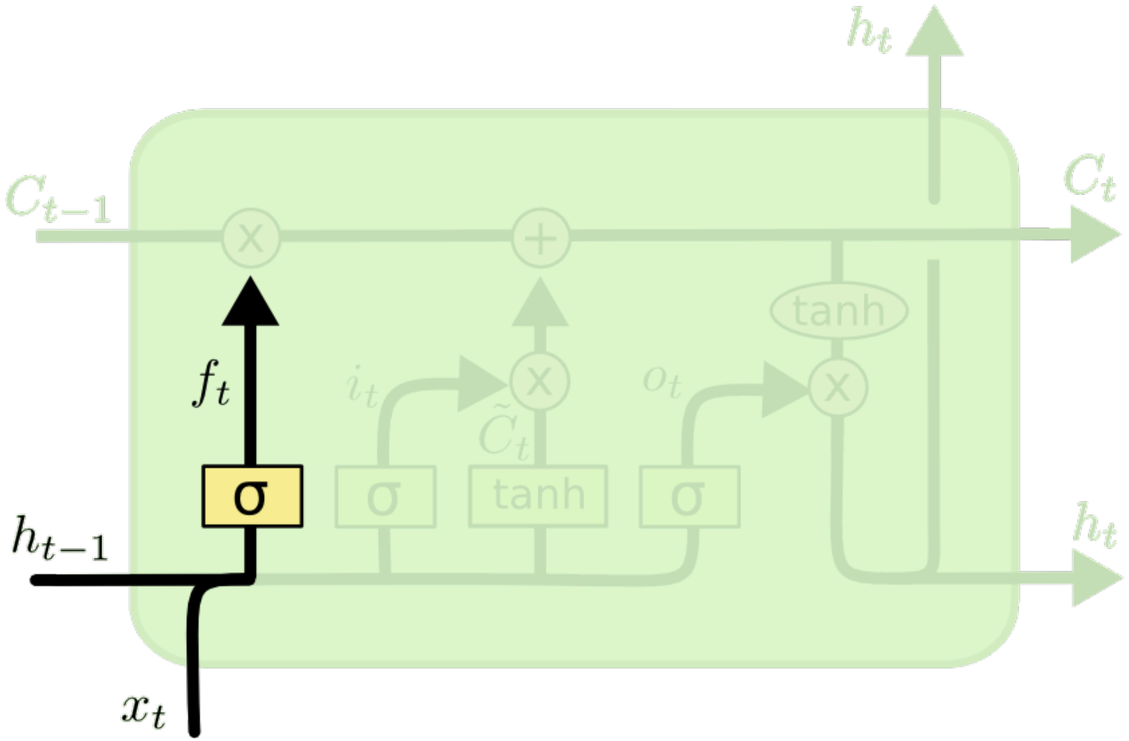
Building up to LSTMs



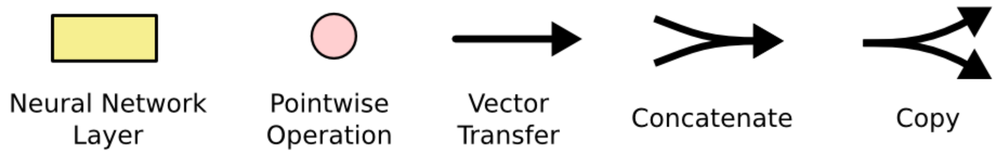
Main idea: introduce a new **"cell state"** channel

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

Building up to LSTMs

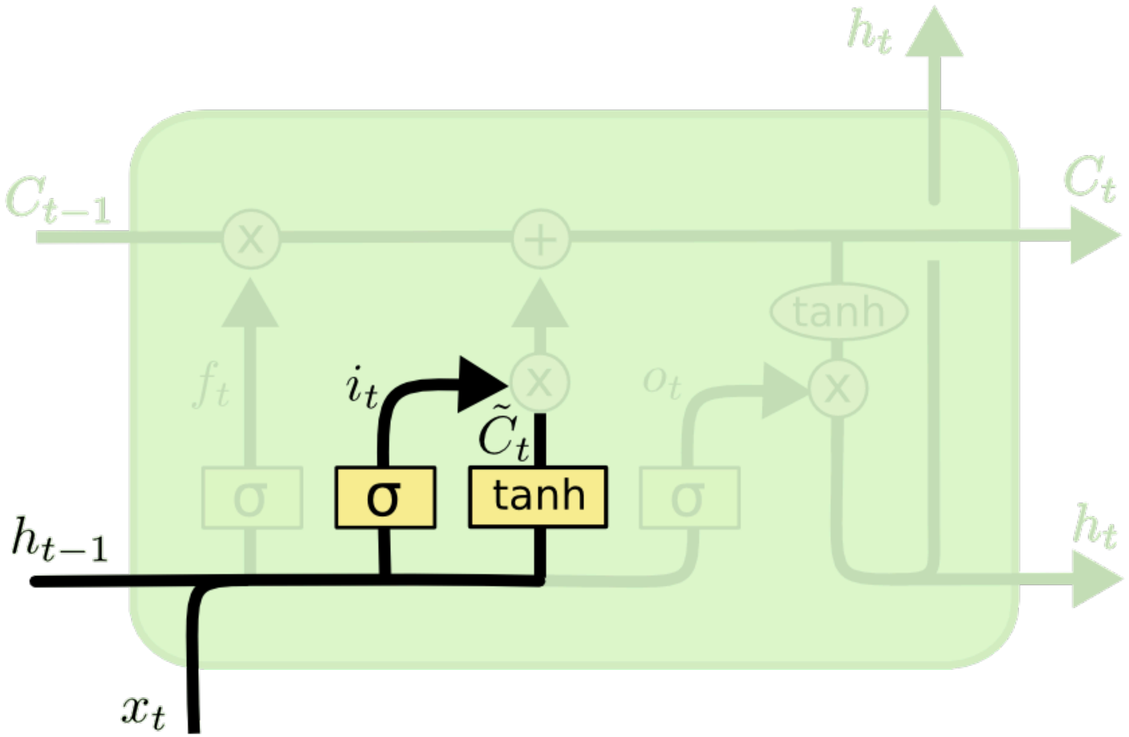


Forget gate:
decide what parts of old state to forget

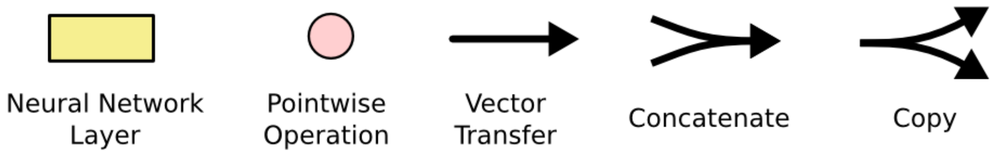


From http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Building up to LSTMs

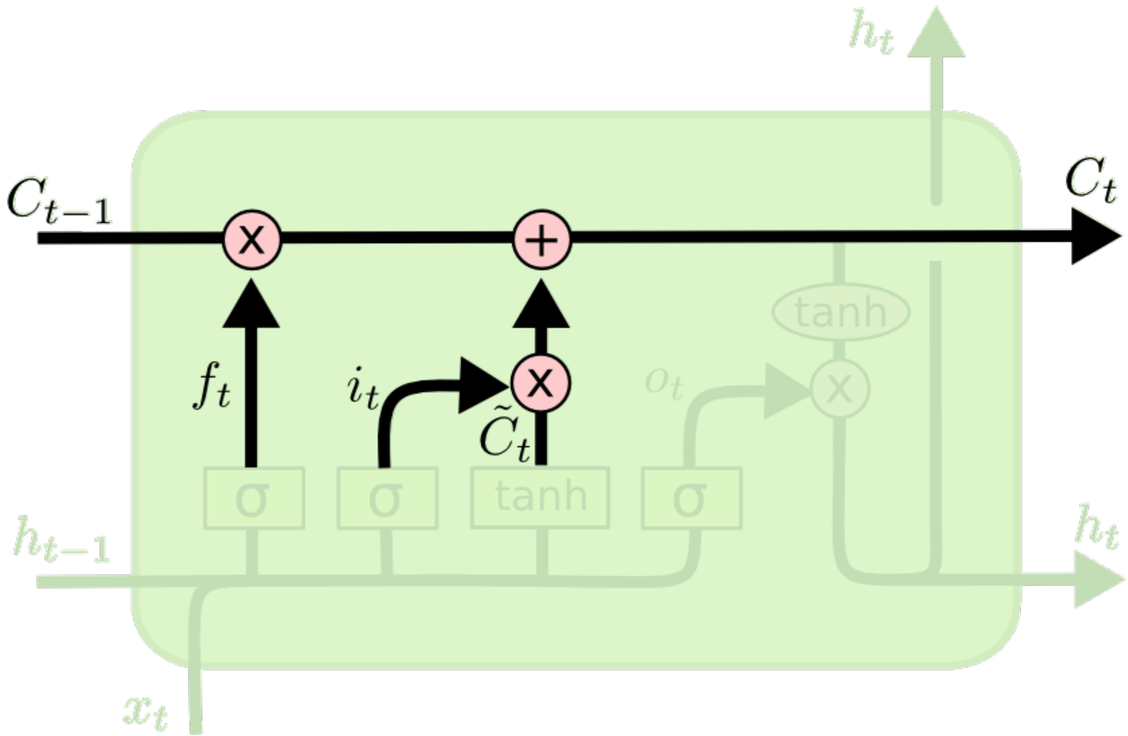


Input gate:
decide how to update the cell state

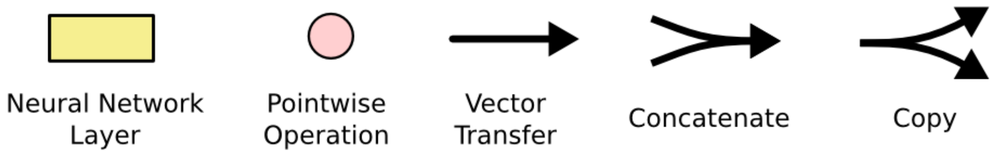


From http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Building up to LSTMs

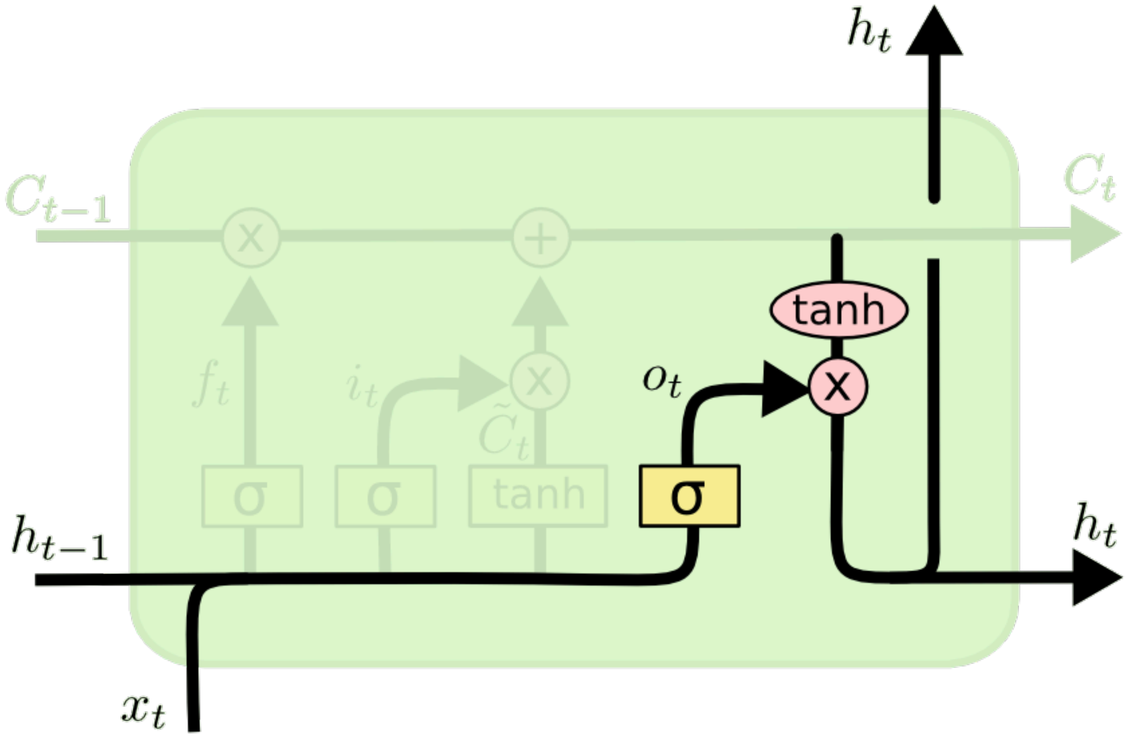


Now actually update the cell state

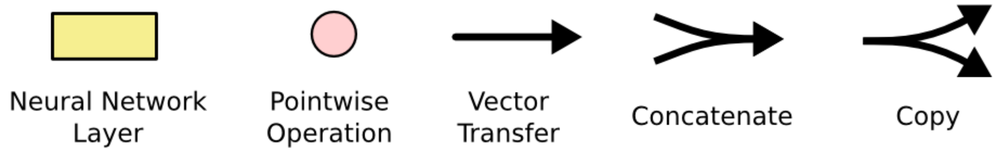


From http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Building up to LSTMs



Finally, decide what to output as hidden state



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

What about GRUs, etc?

LSTM: A Search Space Odyssey
Greff, et al.

- 9 LSTM variants (including GRU)
- 3 datasets
- **Conclusion:** Hard to beat regular LSTM

An Empirical Exploration of Recurrent Neural Network Architectures
(Jozefowicz, Zaremba, Sutskever)

- 10,000 architectures
- 3 datasets
- **Conclusion:** GRUs > LSTMs. Some architectures better than GRU (but only slightly)

What about GRUs, etc?

Our advice

- LSTMs work well for most tasks
- Try GRUs if LSTMs are not performing well

Questions?

Agenda

1. Sequence Problems
2. RNNs
3. Vanishing gradients and LSTMs
4. **Case study: Machine Translation (Bidirectionality and Attention)**
5. CTC loss
6. Pros and Cons
7. A preview of non-recurrent sequence models

Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

(Wu et al., 2016)

Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?

Description of the problem

Input

“I am a student”

Output

“Je suis étudiant(e)” (French)

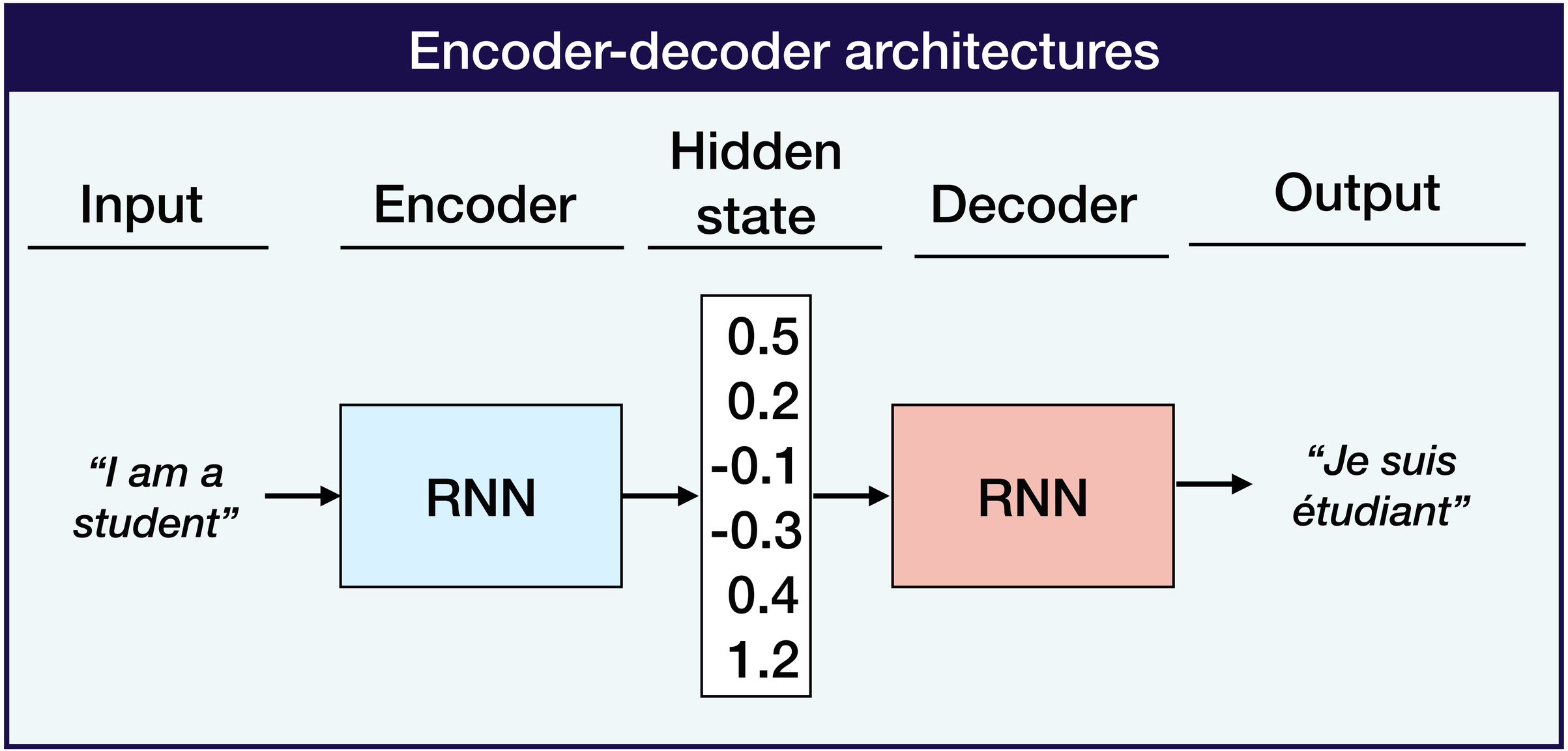
“Soy un estudiante” (Spanish)

我是学生 (Chinese)

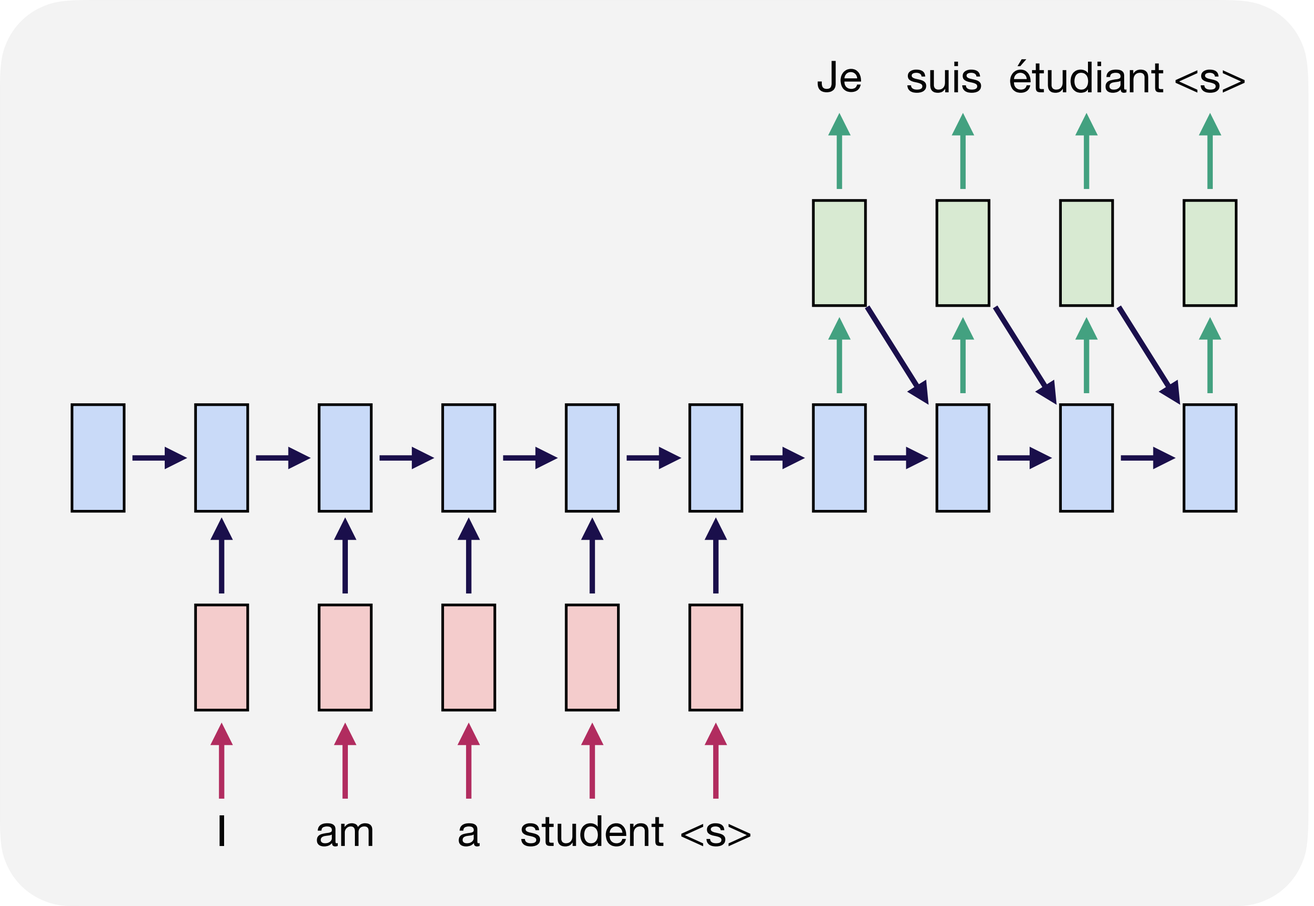
Я студент (Russian)

Key questions for applications

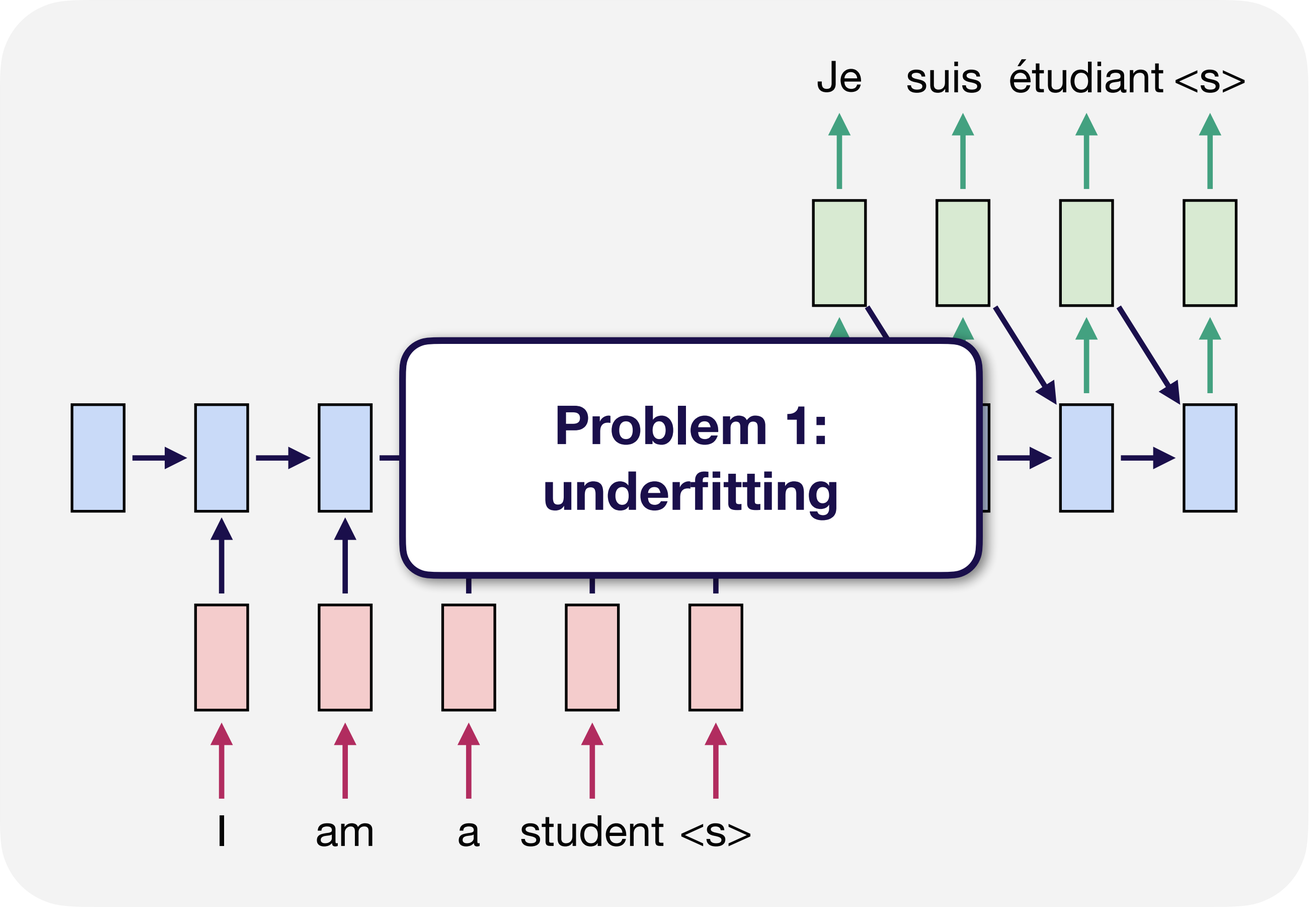
- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?


Encoder-decoder architectures
Input
Encoder
Hidden state
Decoder
Output
“I am a student”
RNN
0.5
0.2
-0.1
-0.3
0.4
1.2
RNN
“Je suis étudiant”

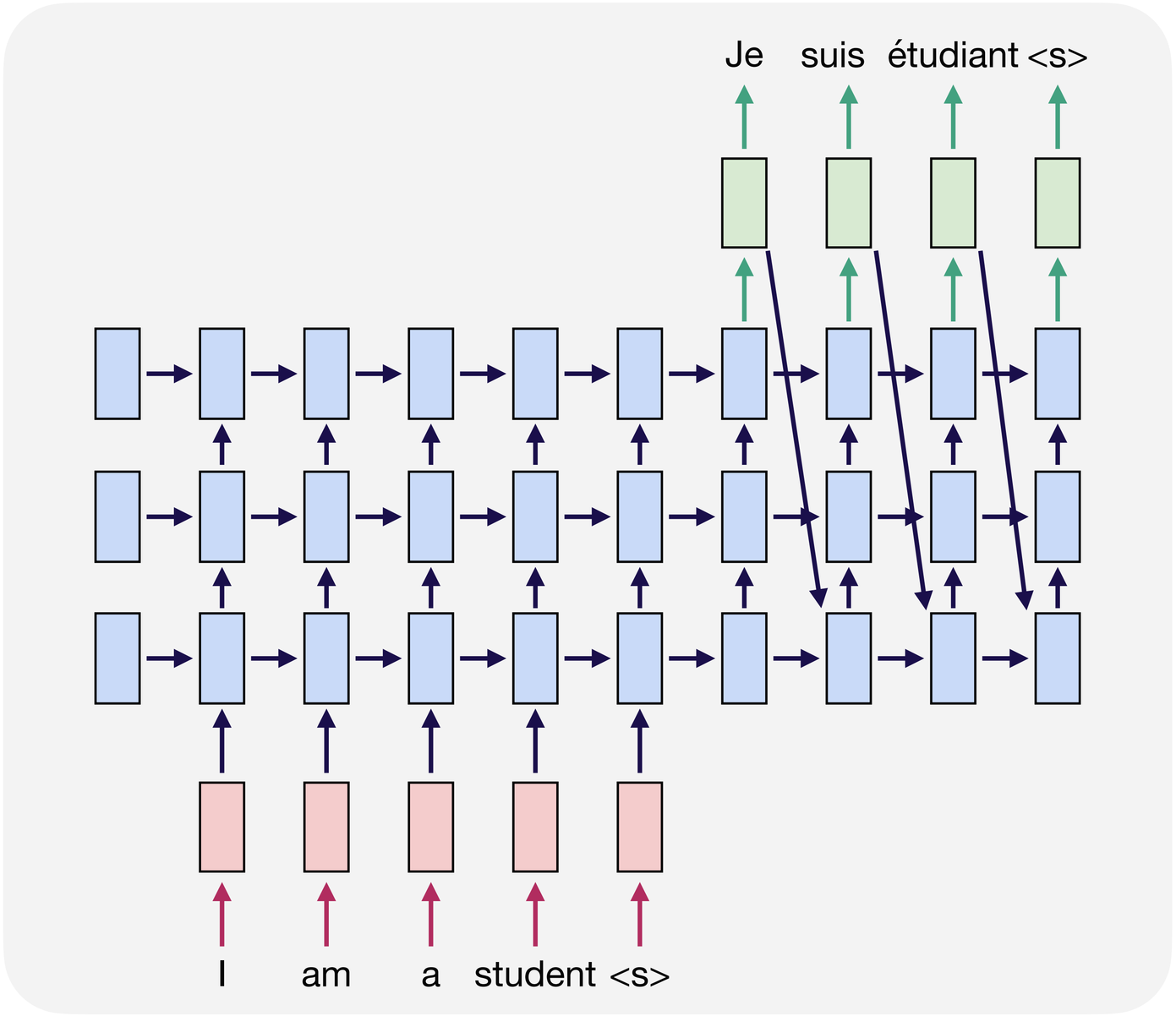
Normal LSTM



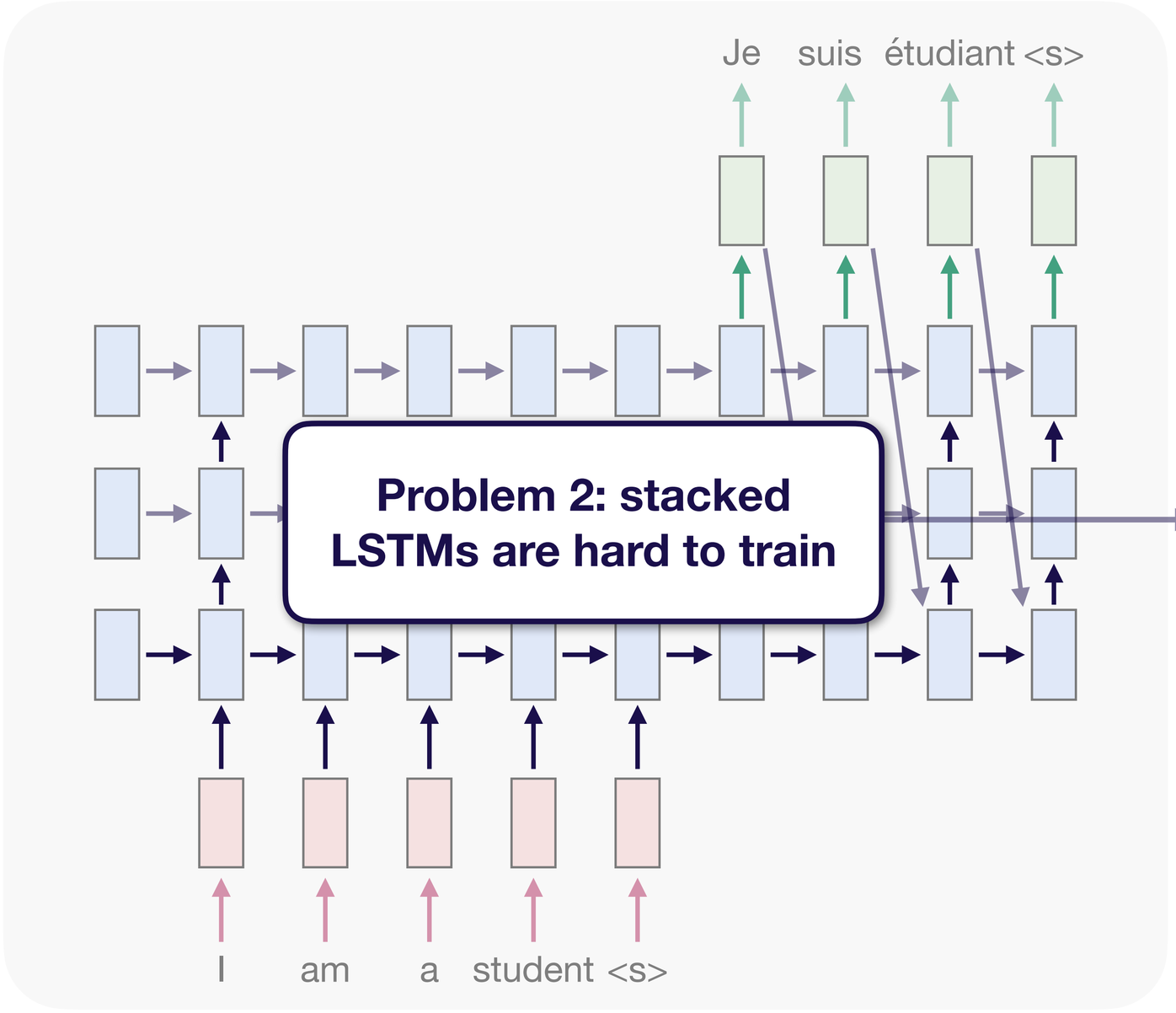
Normal LSTM



Solution: stacked LSTM layers

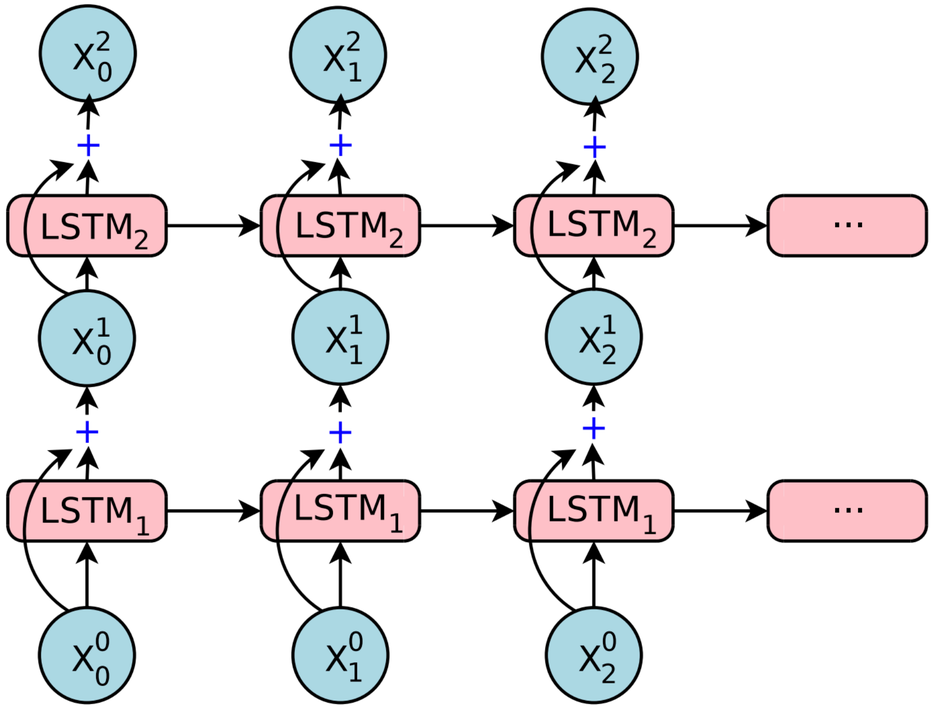


Stacked LSTMs

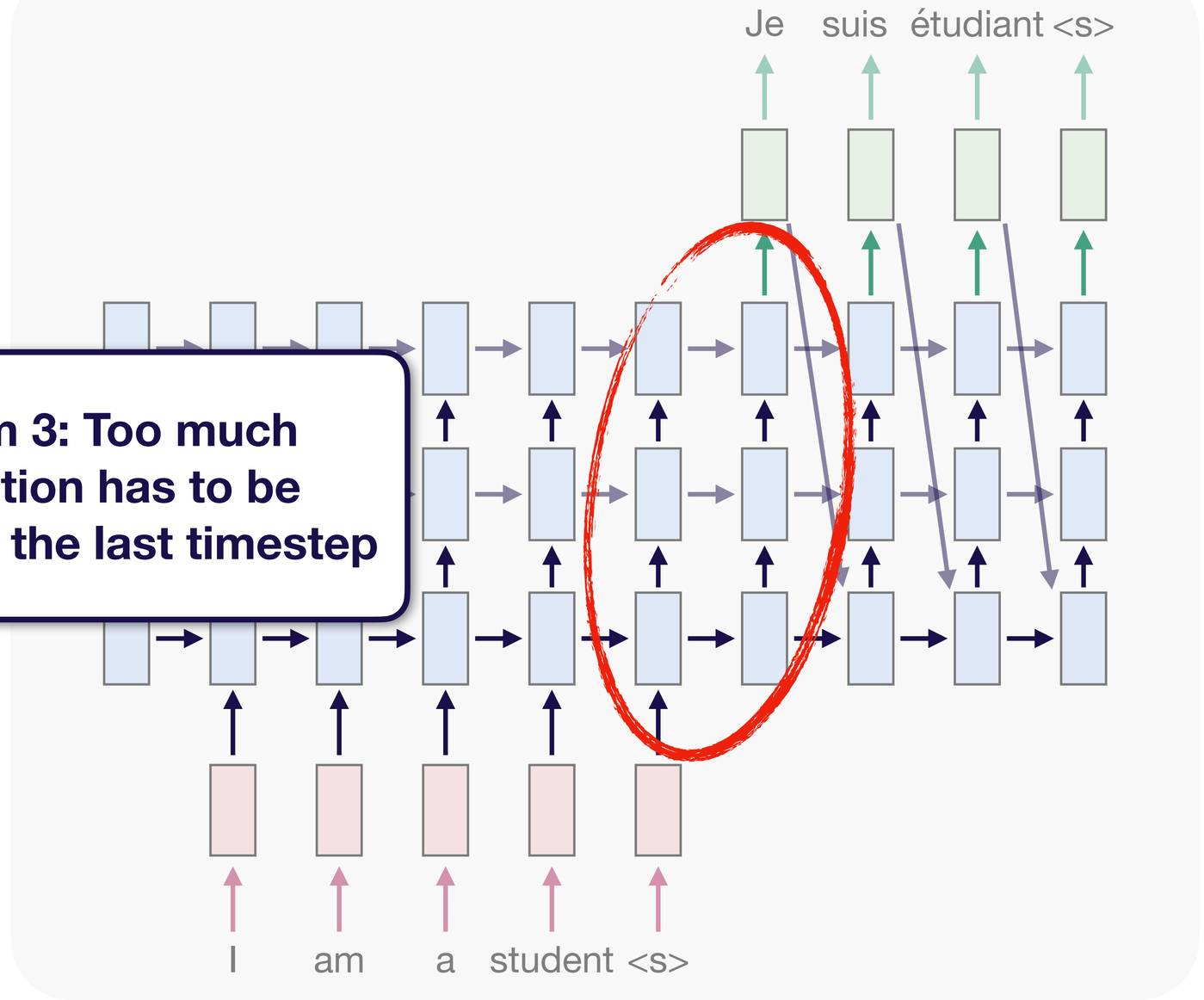


"In our experience with large-scale translation tasks, simple stacked LSTM layers work well up to 4 layers, barely with 6 layers, and very poorly with 8 layers"

Solution: residual connections



Stacked LSTMs



Problem 3: Too much information has to be encoded in the last timestep

Solution: attention

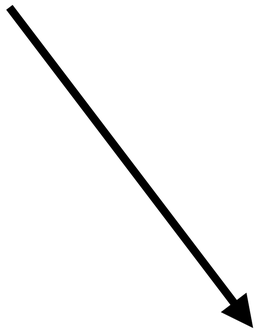
Key idea: Instead of compressing all past time steps into a single hidden state, give the neural network access to the entire history

Pay **attention** to only a subset of past vectors

le renard brun rapide a sauté sur le chien paresseux

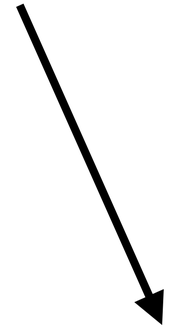
The quick brown fox jumped over the lazy dog

le renard brun rapide a sauté sur le chien paresseux



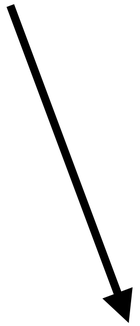
The quick brown fox jumped over the lazy dog

le **renard** brun rapide a sauté sur le chien paresseux



The quick brown fox jumped over the lazy dog

le renard **brun** rapide a sauté sur le chien paresseux



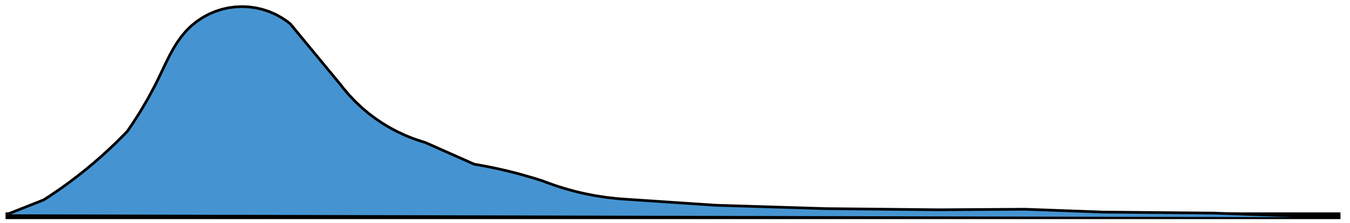
The quick **brown** fox jumped over the lazy dog

le renard brun **rapide** a sauté sur le chien paresseux

The **quick** brown fox jumped over the lazy dog

Relevance to this word (p_i)

0.01 **0.9** 0.03 0.05 0.01 0.0 0.0 0.0 0.0

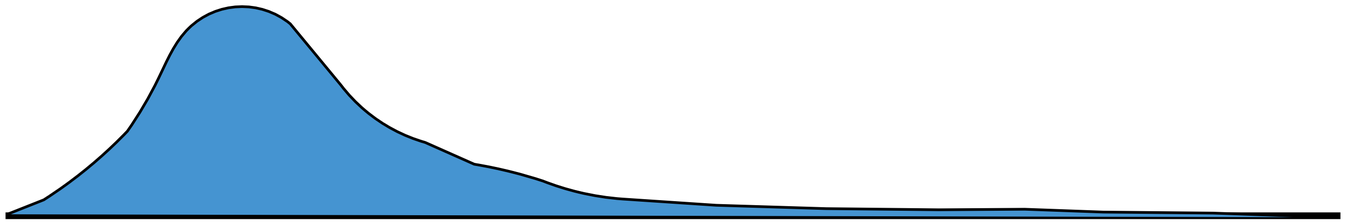


le renard brun **rapide** a sauté sur le chien paresseux

The **quick** brown fox jumped over the lazy dog

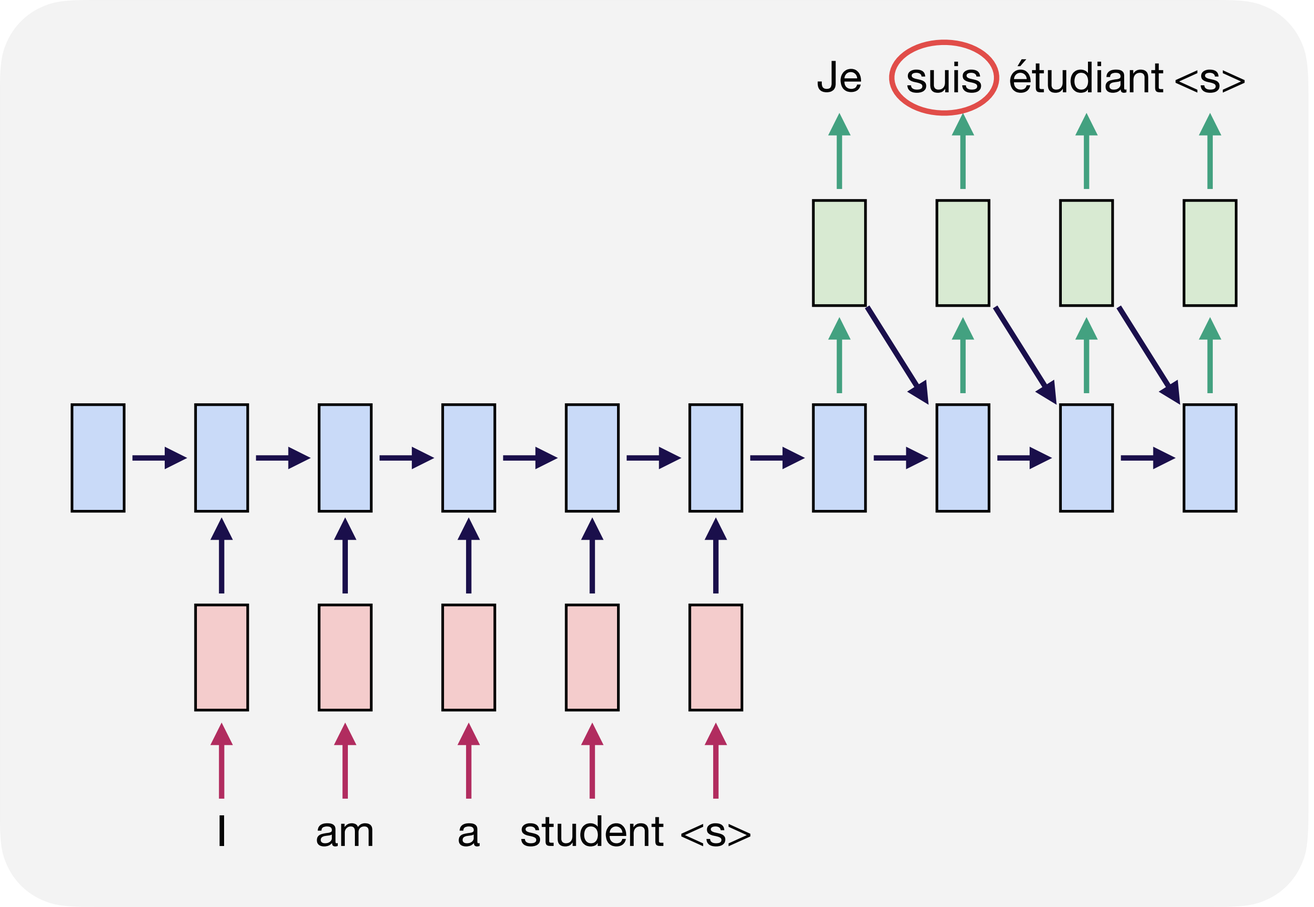
Relevance to this word (p_i)

0.01 **0.9** 0.03 0.05 0.01 0.0 0.0 0.0 0.0

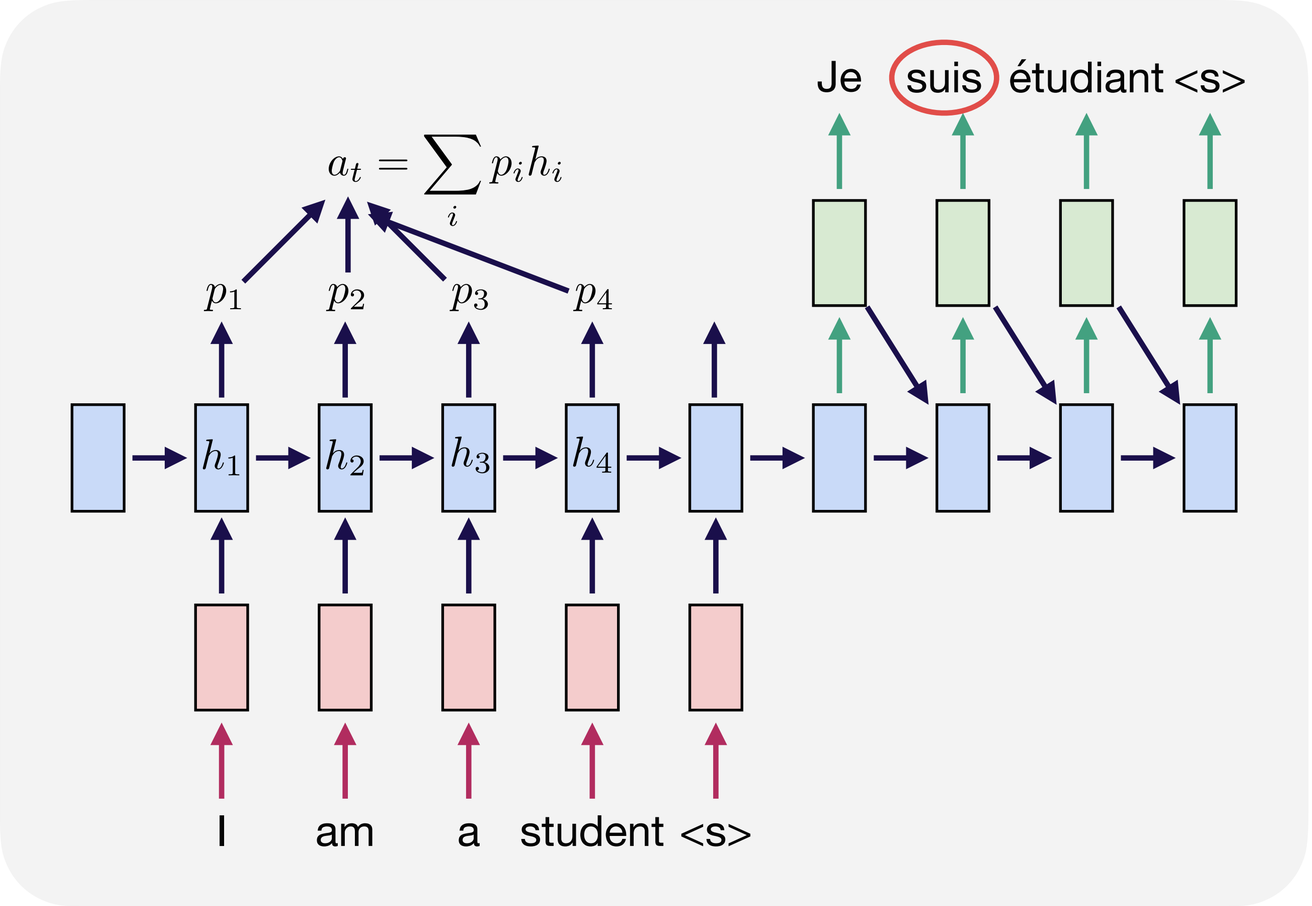


How to use relevance scores?

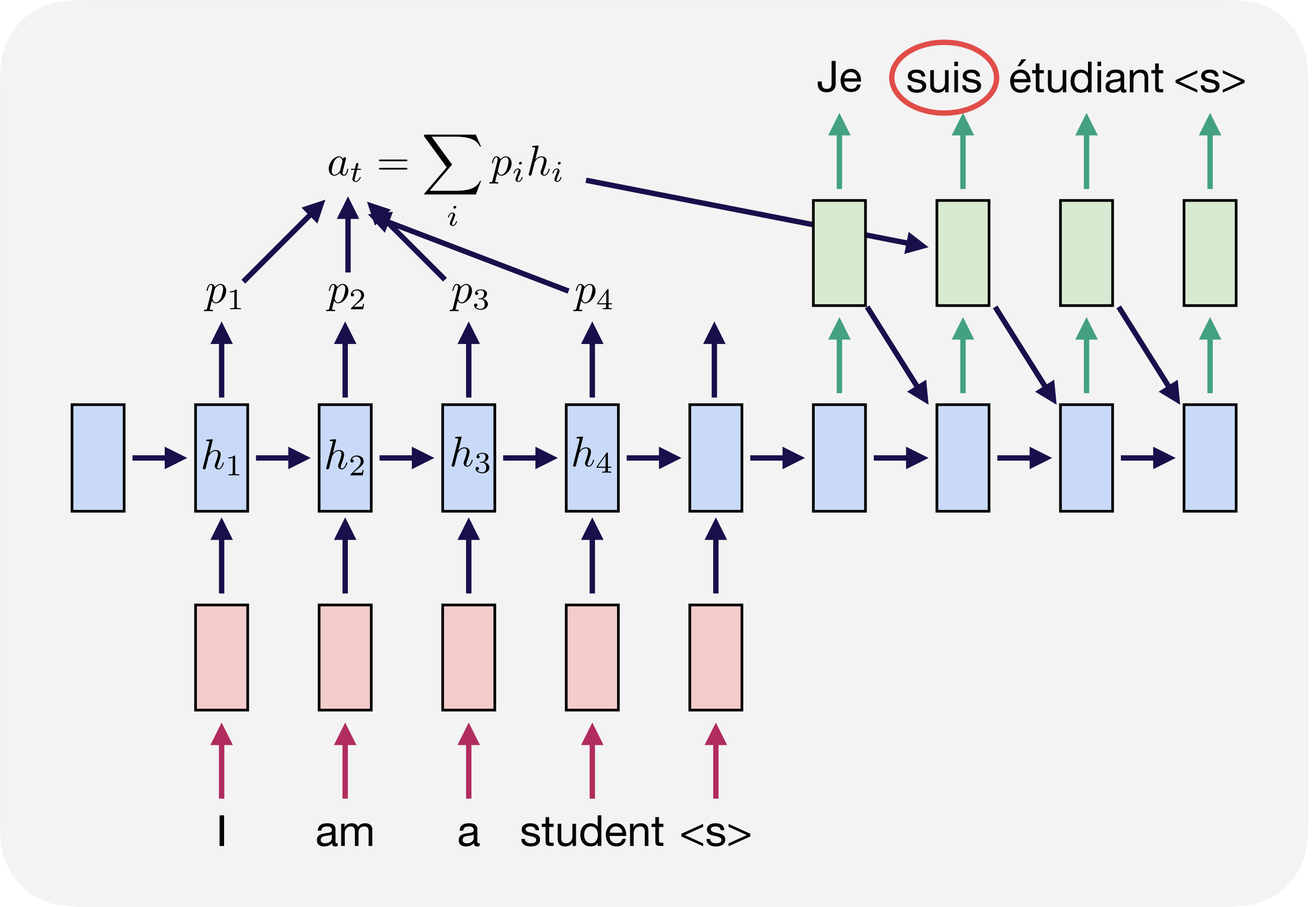
Attention: how to use relevance scores



Attention: how to use relevance scores



Attention: how to use relevance scores



Attention Visualization

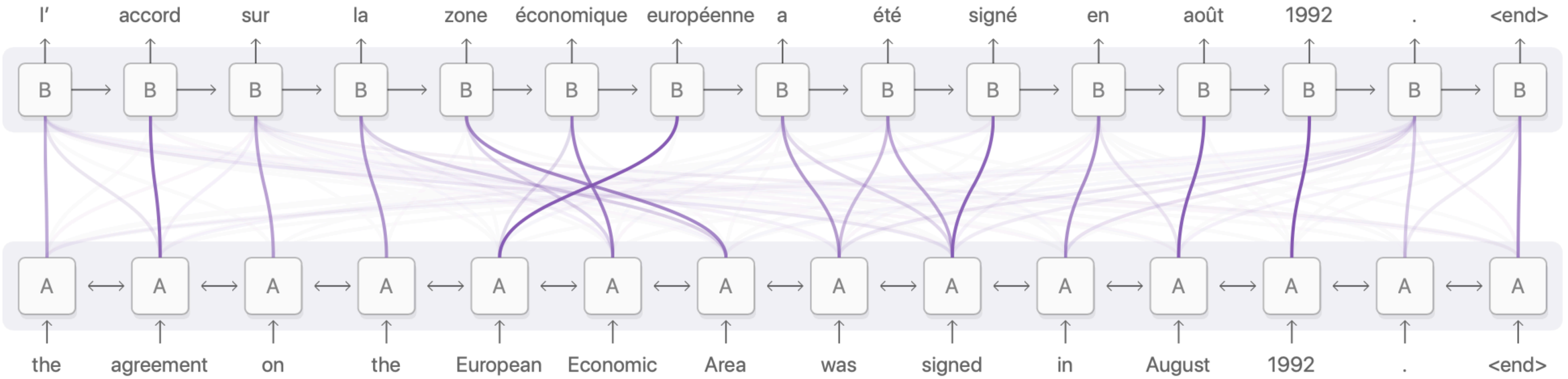
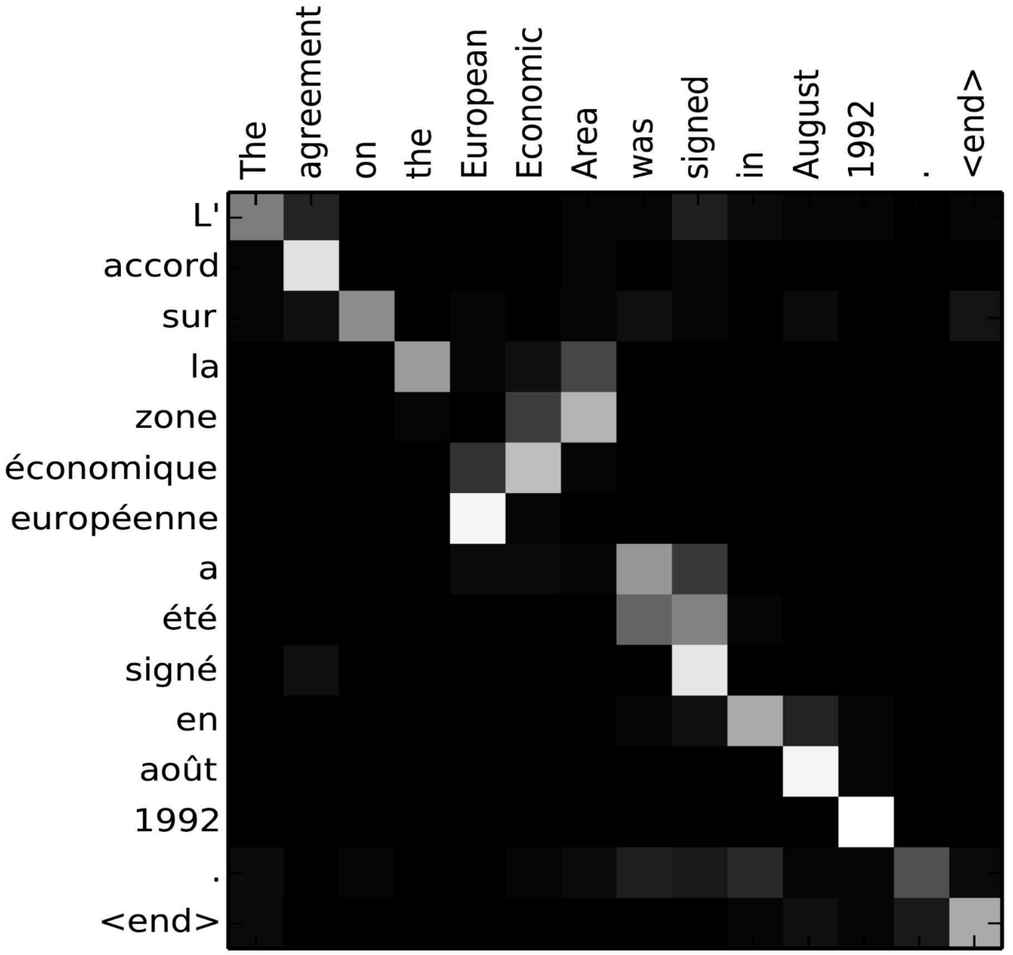


Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

https://distill.pub/2016/augmented-rnns/#attentional-interfaces

Attention Visualization



https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention: not just for translation

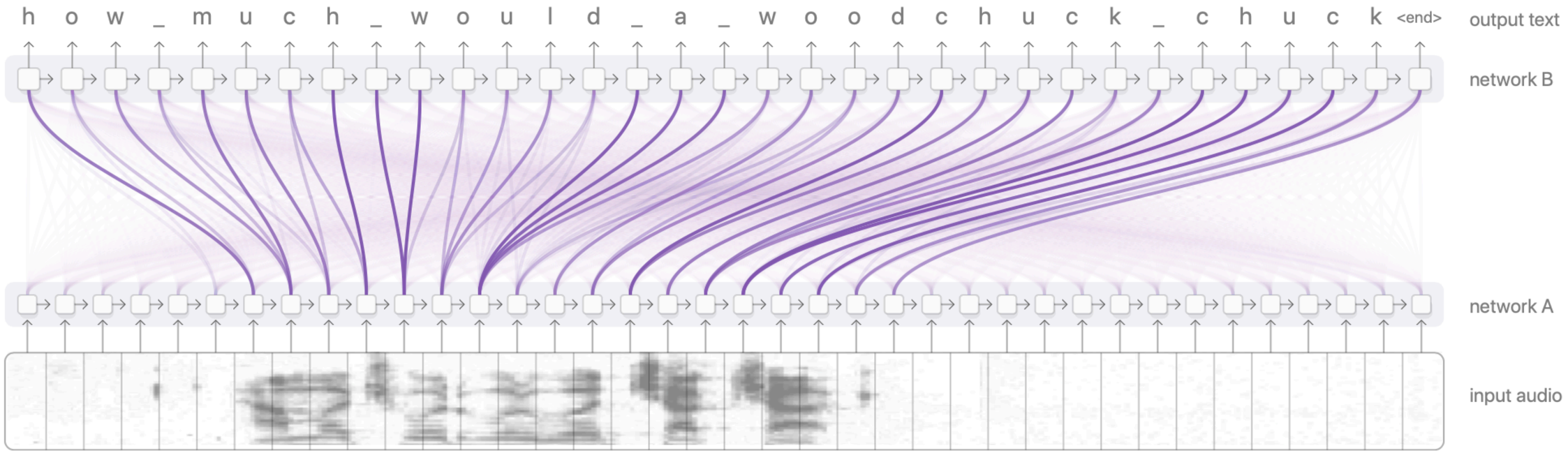


Figure derived from Chan, *et al.* 2015

https://distill.pub/2016/augmented-rnns/#attentional-interfaces

Attention: not just for translation

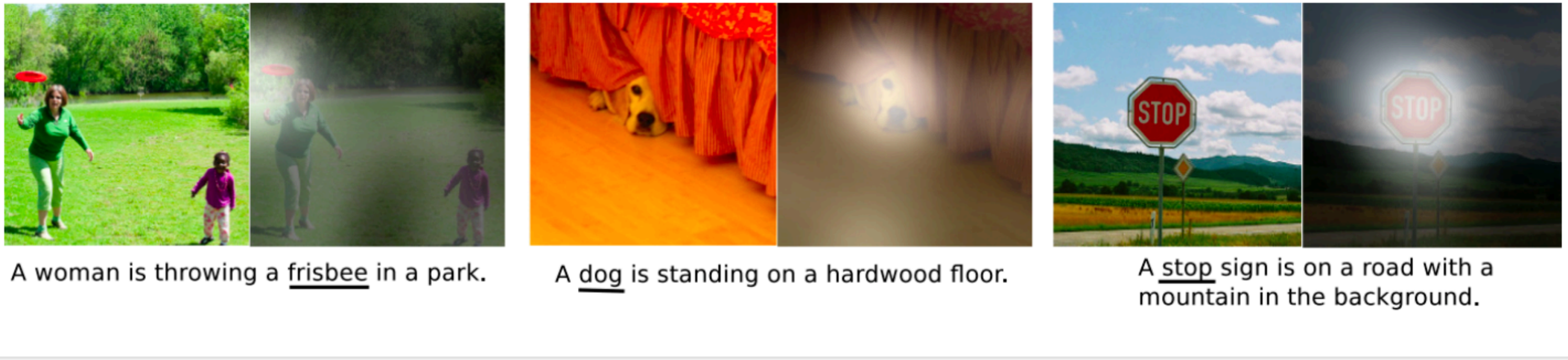
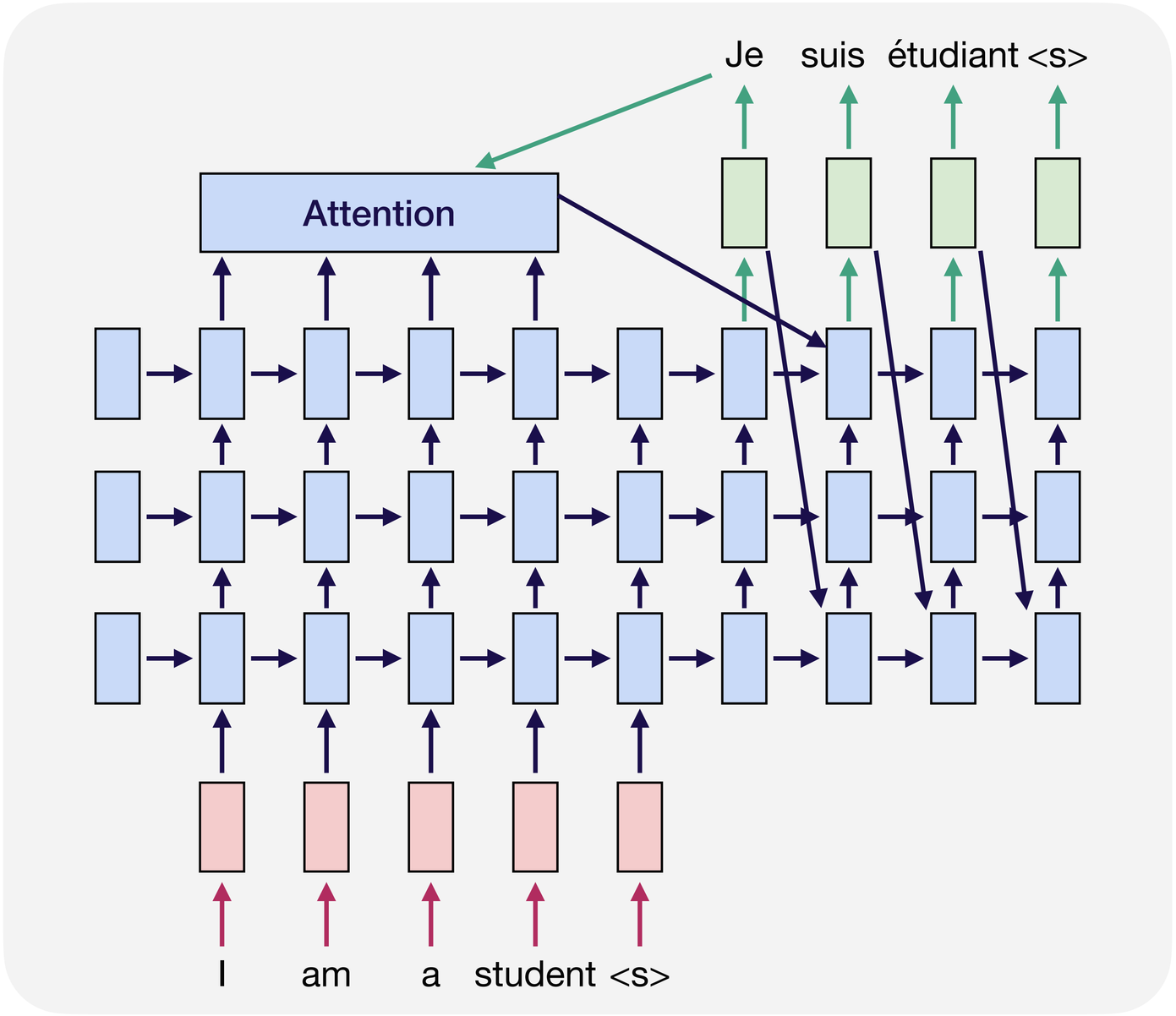


Figure from [3]

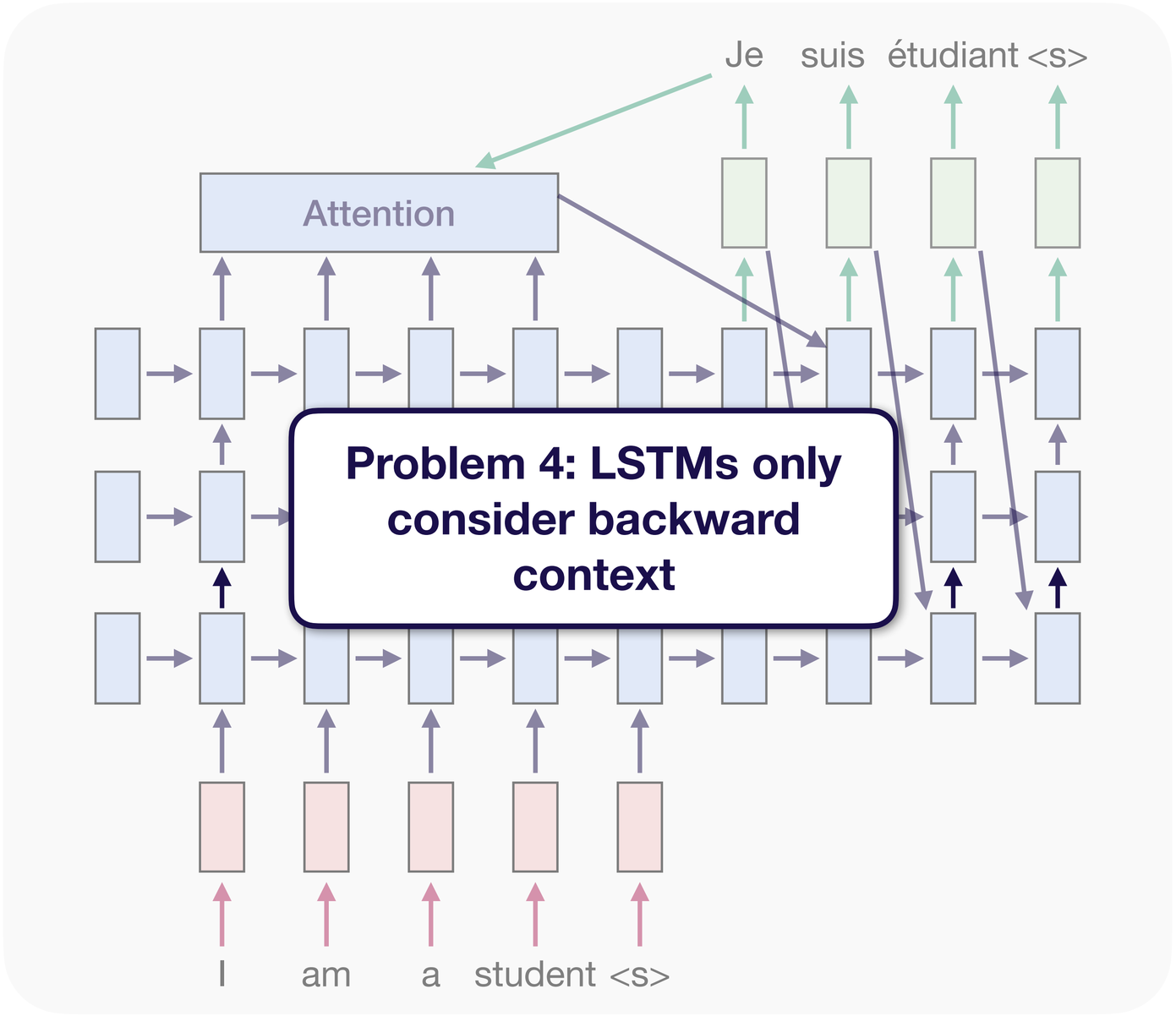
https://distill.pub/2016/augmented-rnns/#attentional-interfaces

Questions?

Stacked LSTM with attention

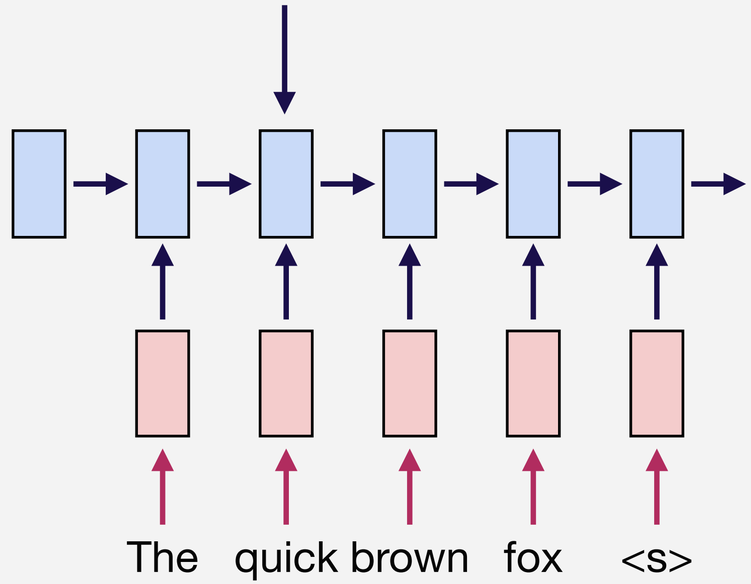


Stacked LSTM with attention



Why is future information important?

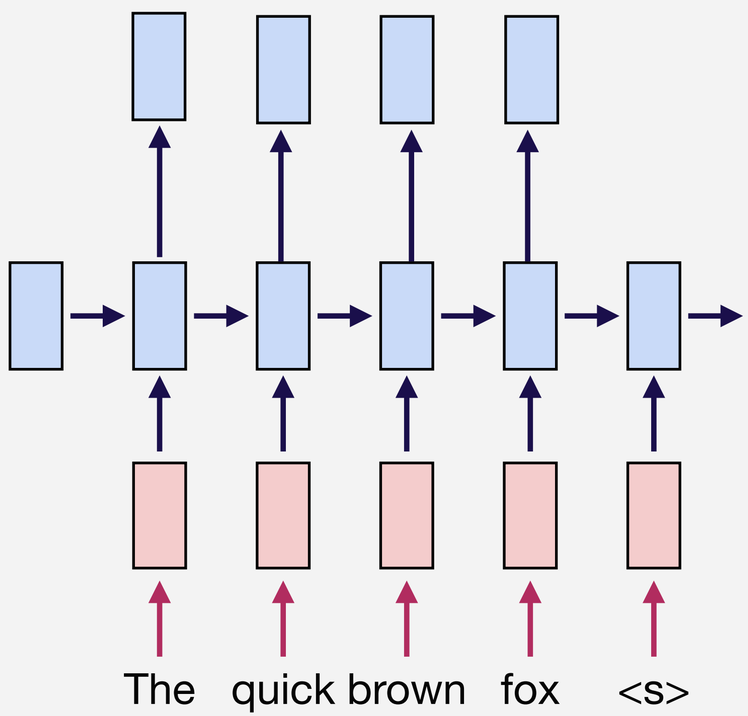
“quick” could mean a lot of things: fast, intelligent, etc. To compress it optimally, need to know what comes after!



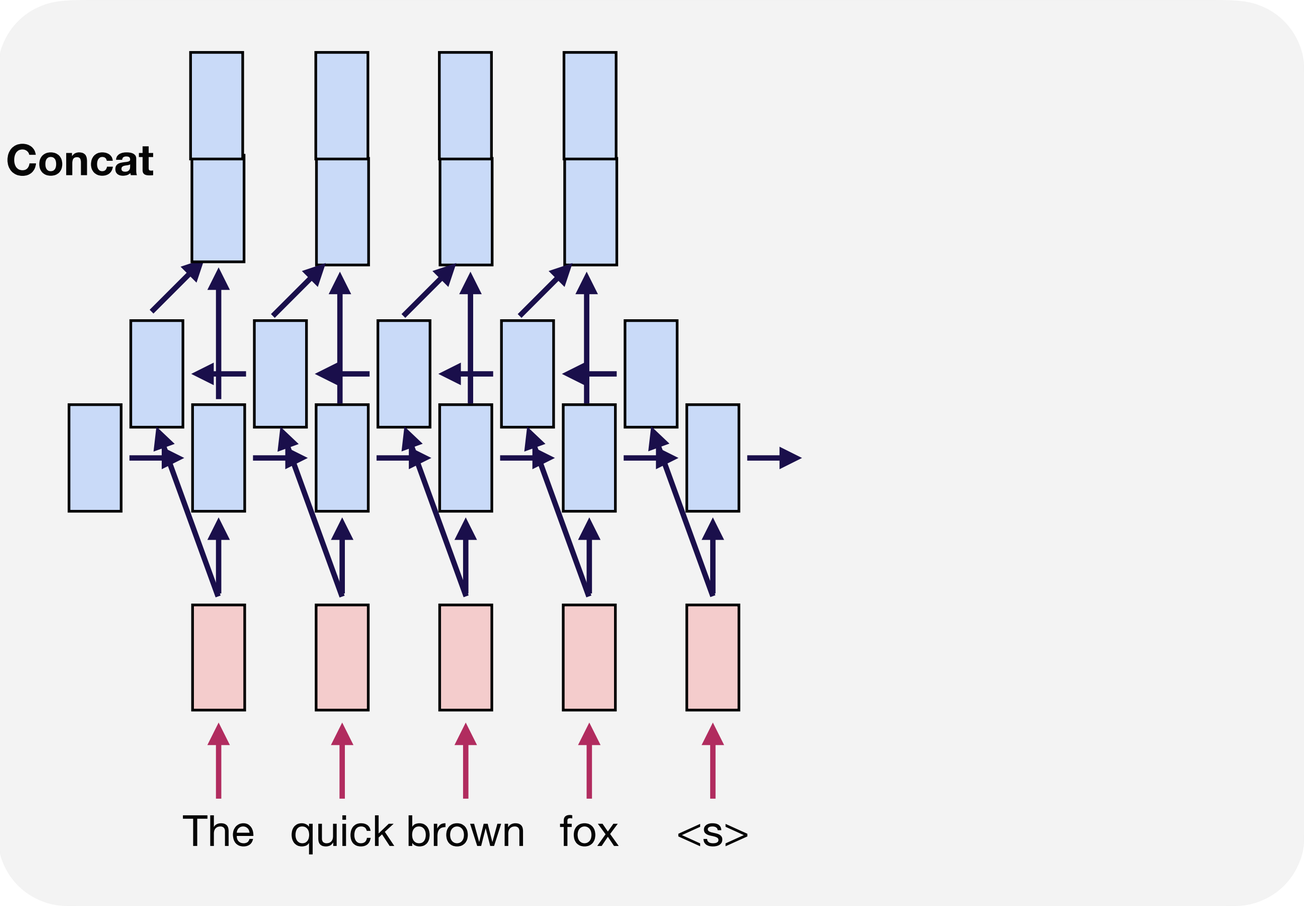
Solution: bidirectionality

Key idea: Use one LSTM to process the sequence in forward order, the other in backward order

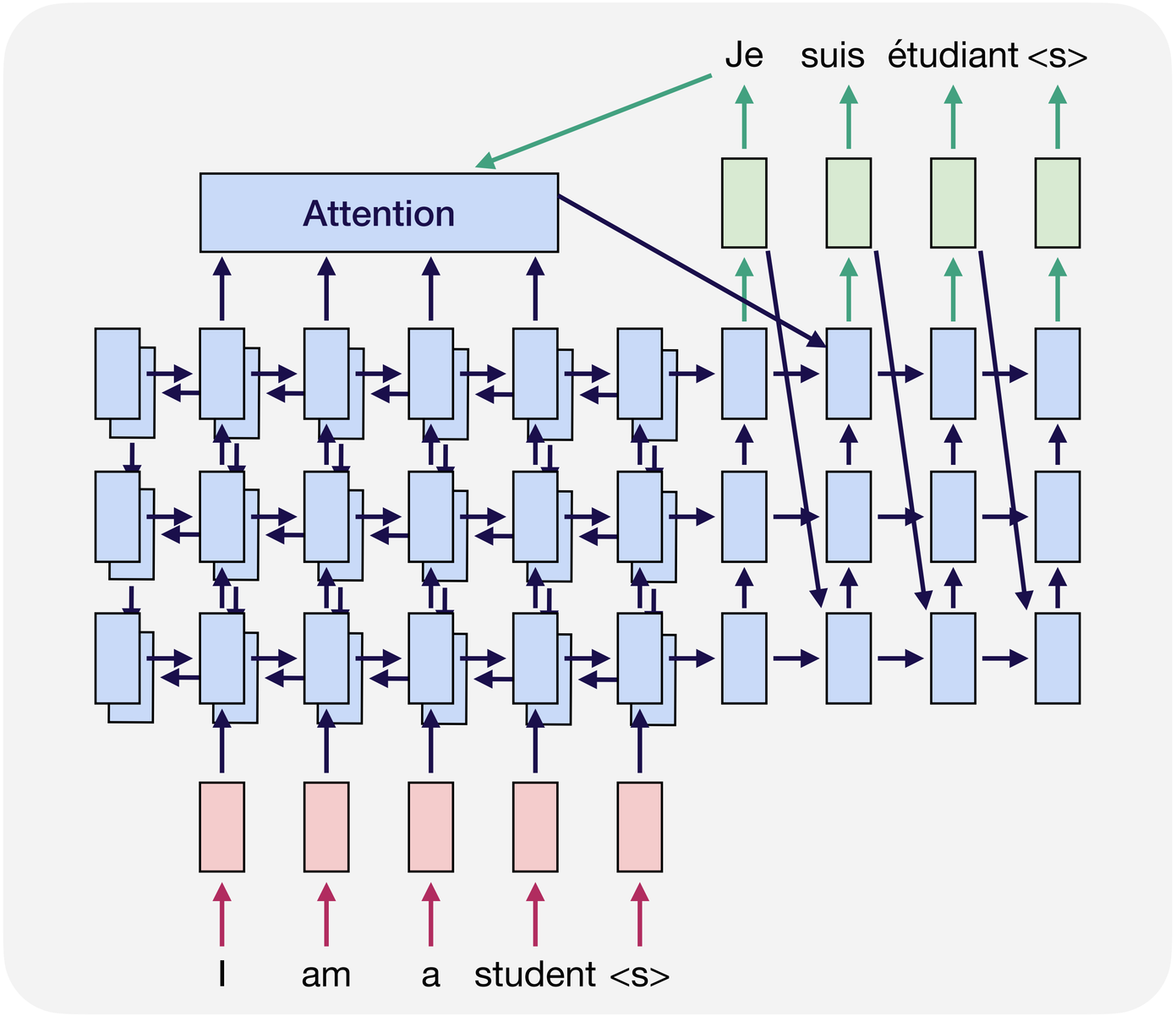
Solution: bidirectionality



Solution: bidirectionality



Stacked biderictional LSTMs with attention



Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?

Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trai on?
- How did the ?
- Wha were needed for **inference** in deployment?

Skipping!

Summary of GNMT approach

- **Stacked LSTM encoder-decoder** architecture with residual connections
- **Attention** enables longer-term connections
- To encode future information in the source sentence use a **bidirectional LSTM**
- Train using **standard cross-entropy** on a **large** dataset
- **Speed up** inference with quantization of weights

Questions?

Agenda

1. Sequence Problems
2. RNNs
3. Vanishing gradients and LSTMs
4. Case study: Machine Translation
   (Bidirectionality and Attention)
5. **CTC loss**
6. Pros and Cons
7. A preview of non-recurrent sequence models

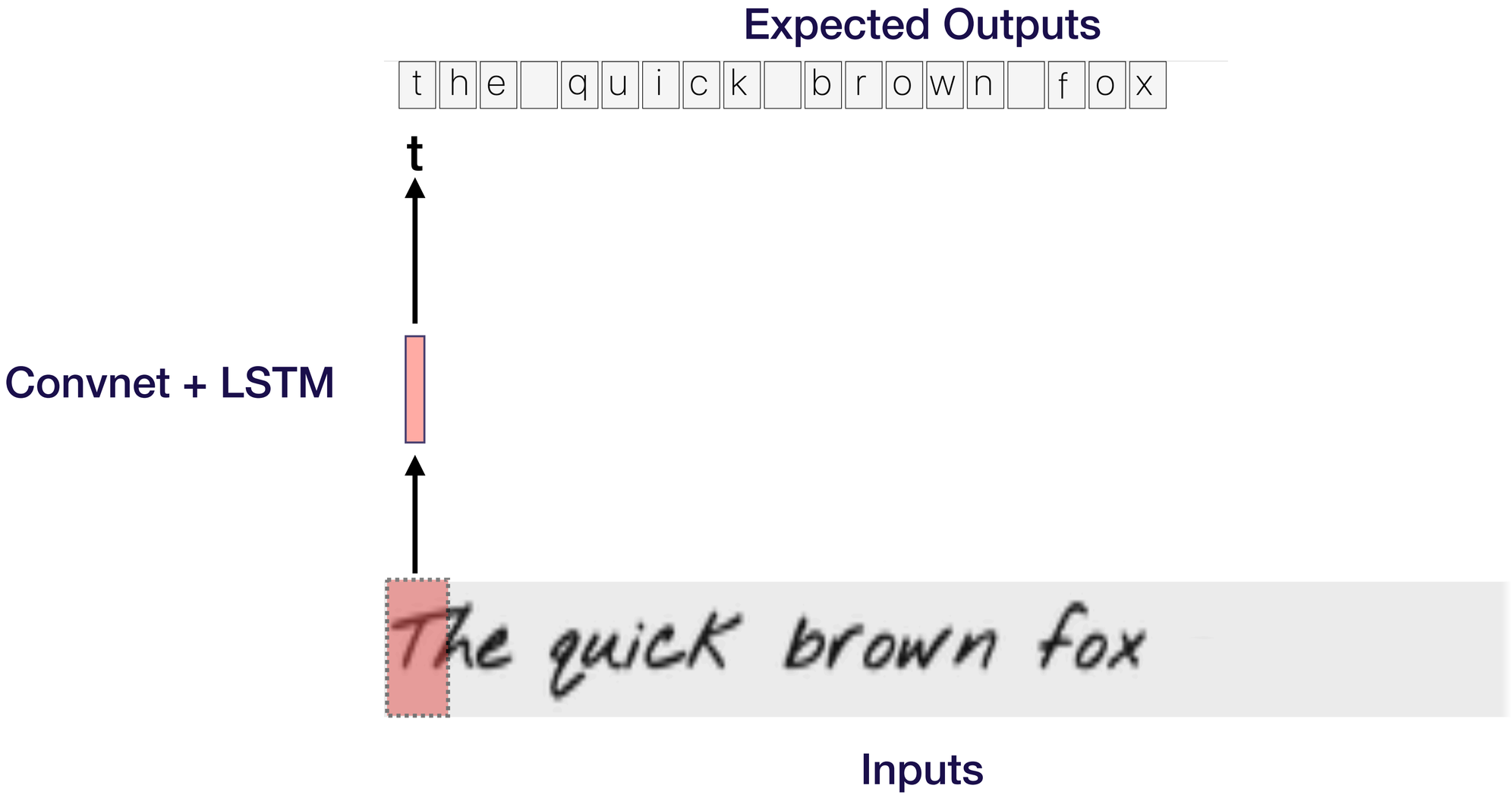
The Goal

Expected Outputs

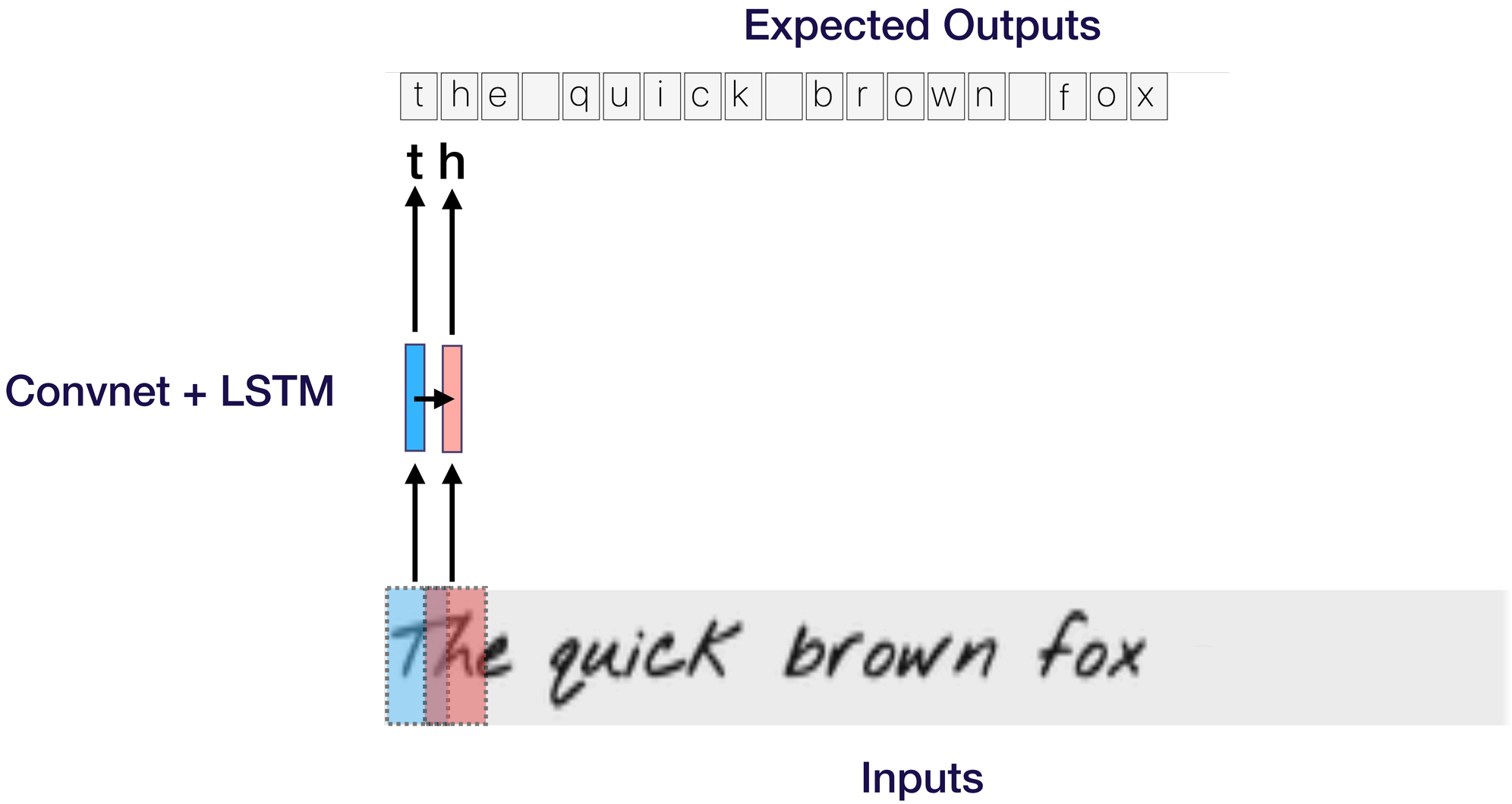
t h e _ q u i c k _ b r o w n _ f o x

Inputs

The Idea


Expected Outputs
t h e q u i c k b r o w n f o x
t
Convnet + LSTM
The quick brown fox
Inputs

The idea


Expected Outputs
t h e q u i c k b r o w n f o x
t h
Convnet + LSTM
Inputs

The idea

Expected Outputs

t h e _ q u i c k _ b r o w n _ f o x

t h e

Convnet + LSTM

Inputs

The idea

Expected Outputs

t h e _ q u i c k _ b r o w n _ f o x

Convnet + LSTM

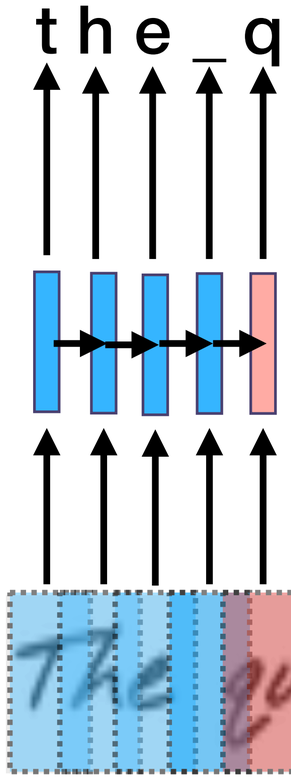
Inputs

The idea

Expected Outputs

t h e _ q u i c k _ b r o w n _ f o x

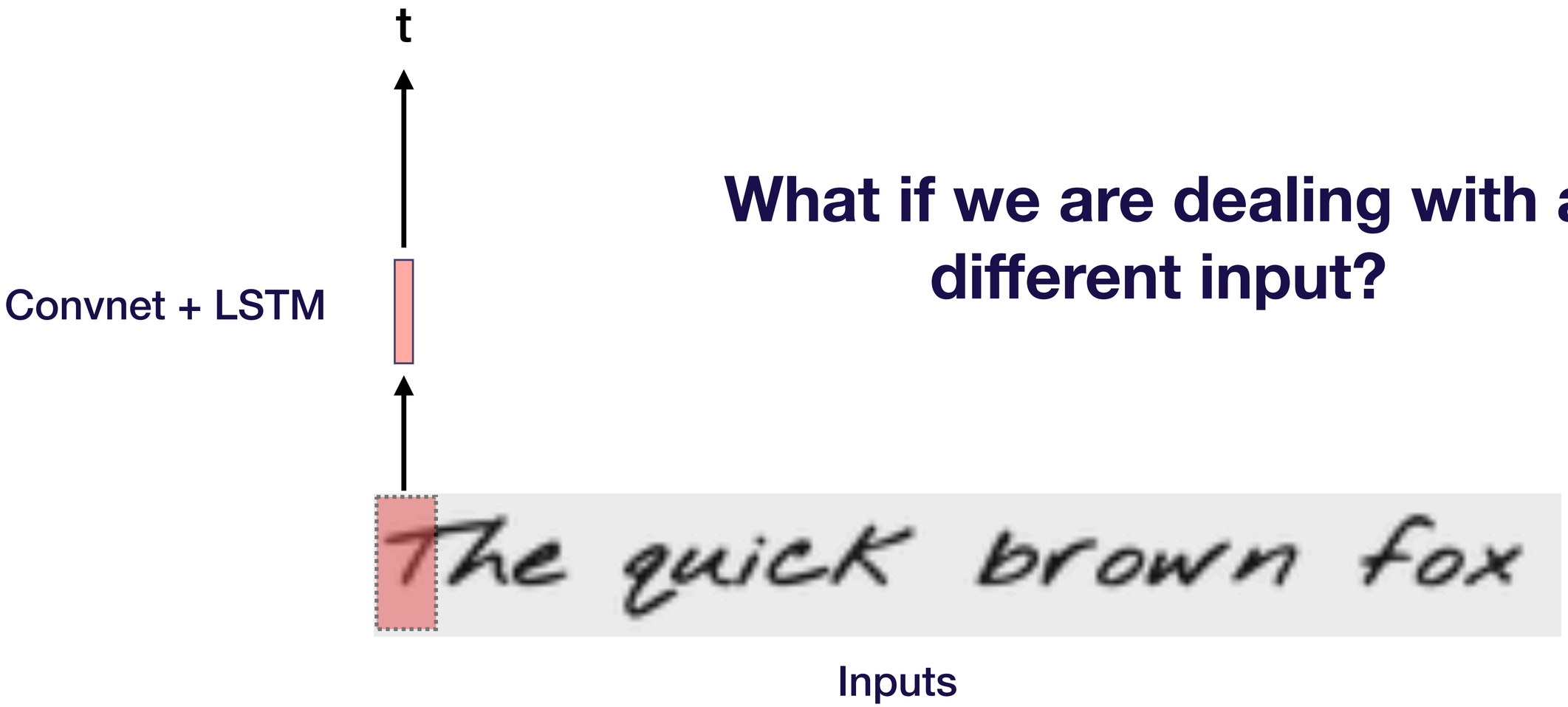
Convnet + LSTM



Inputs

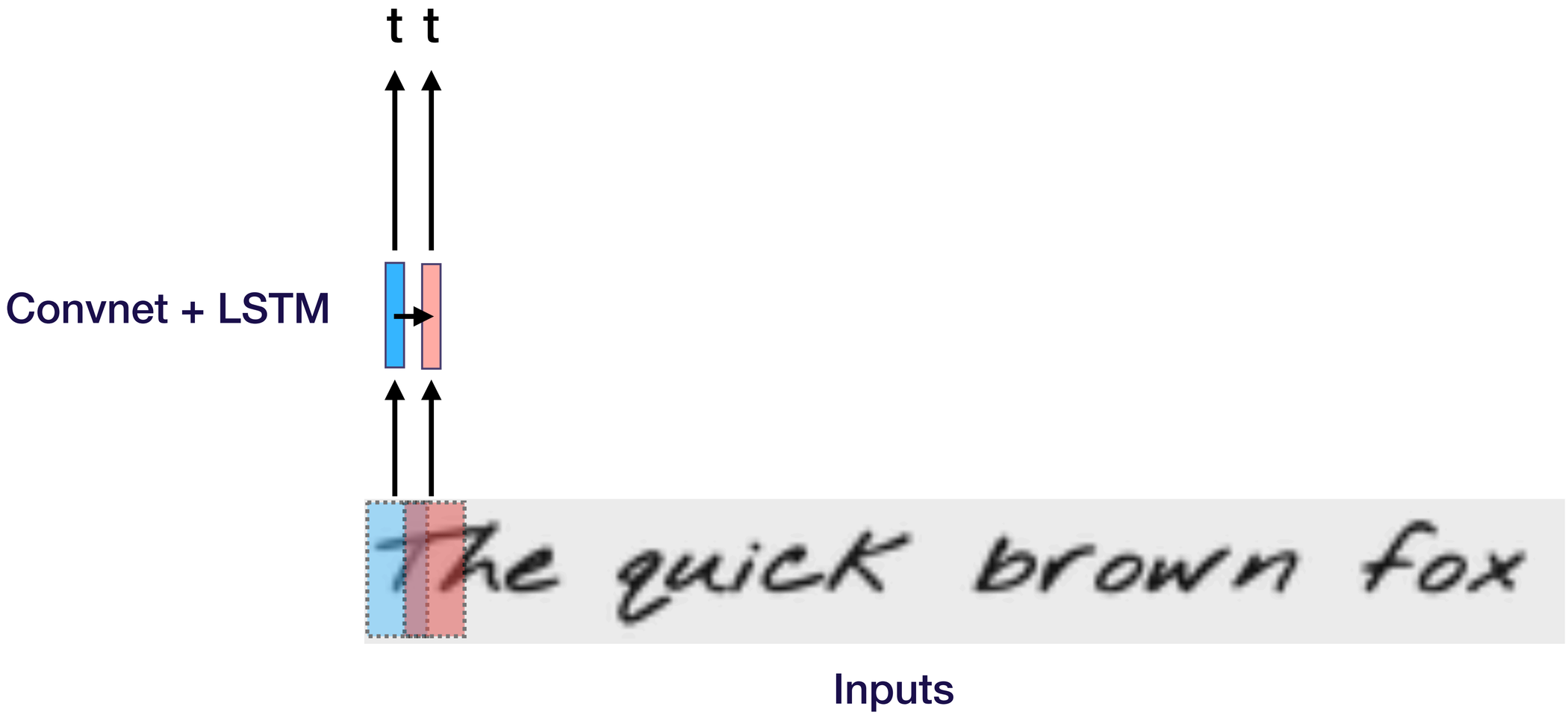
Works great!

The problem

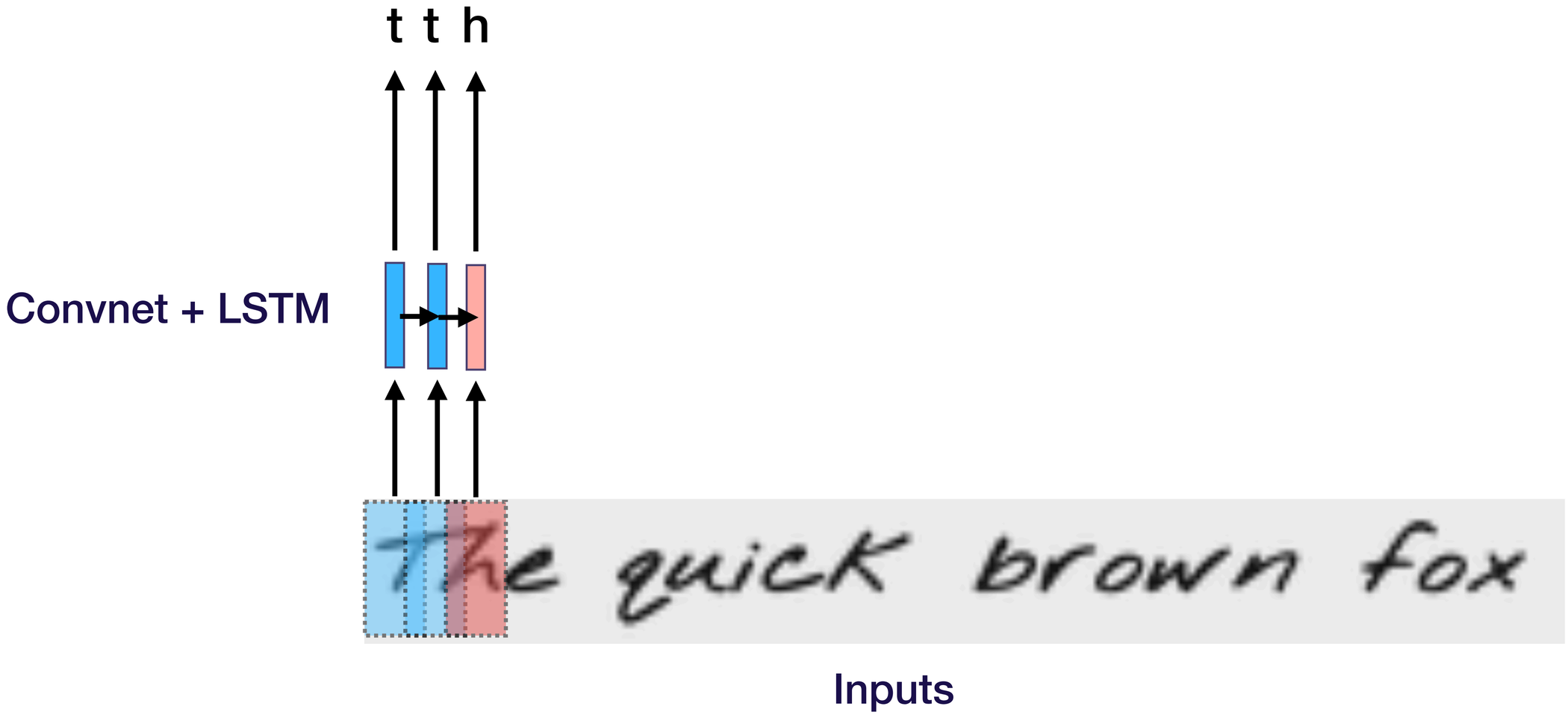


What if we are dealing with a different input?

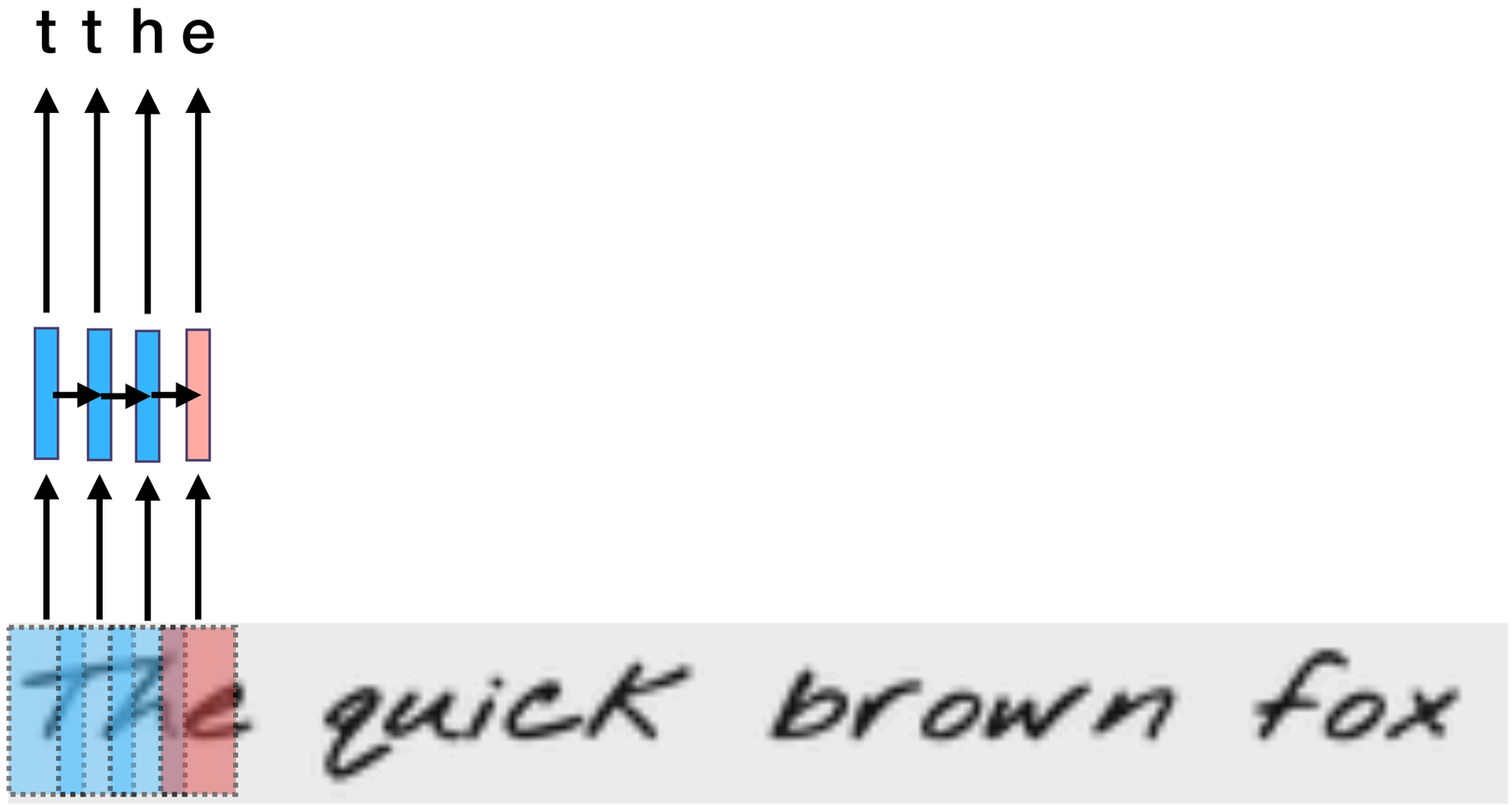
The problem



The problem

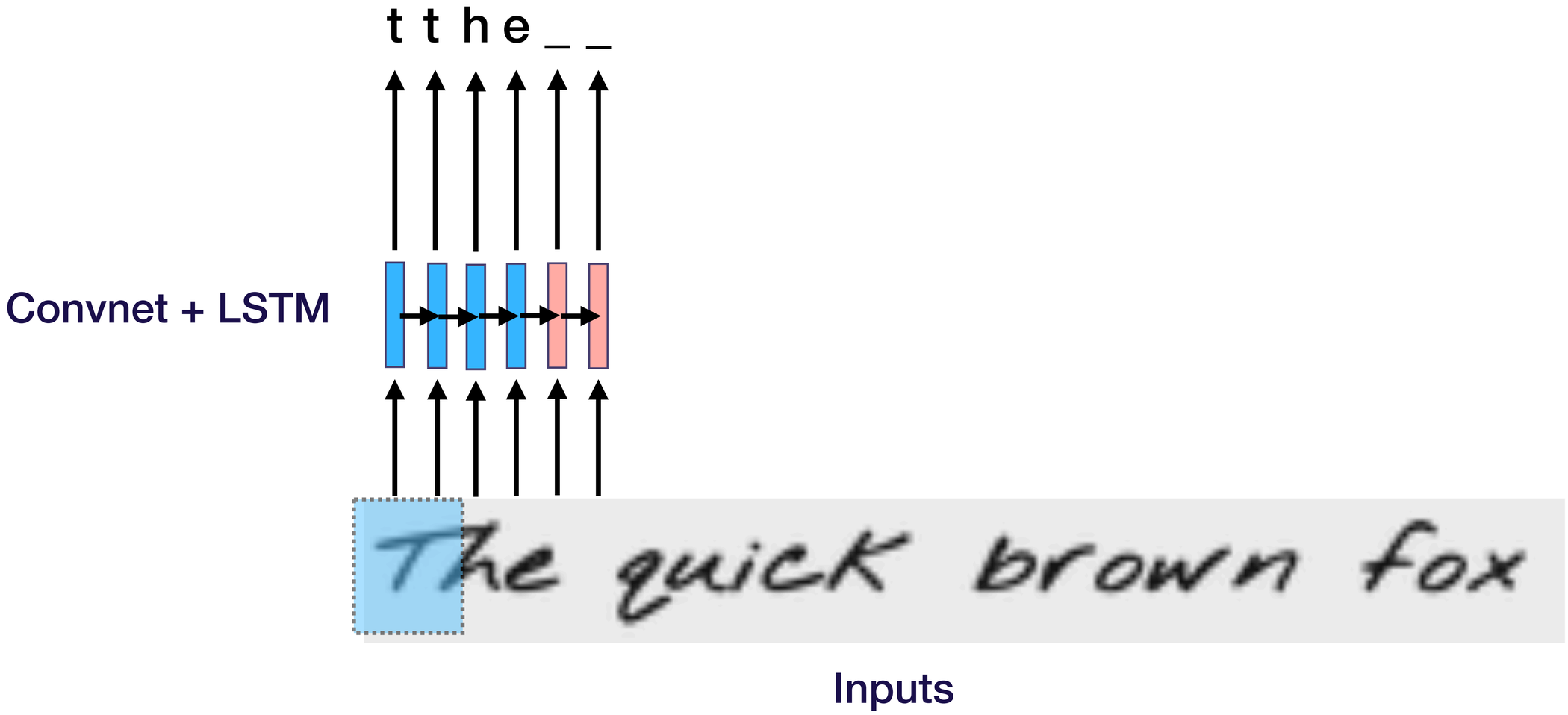


The problem

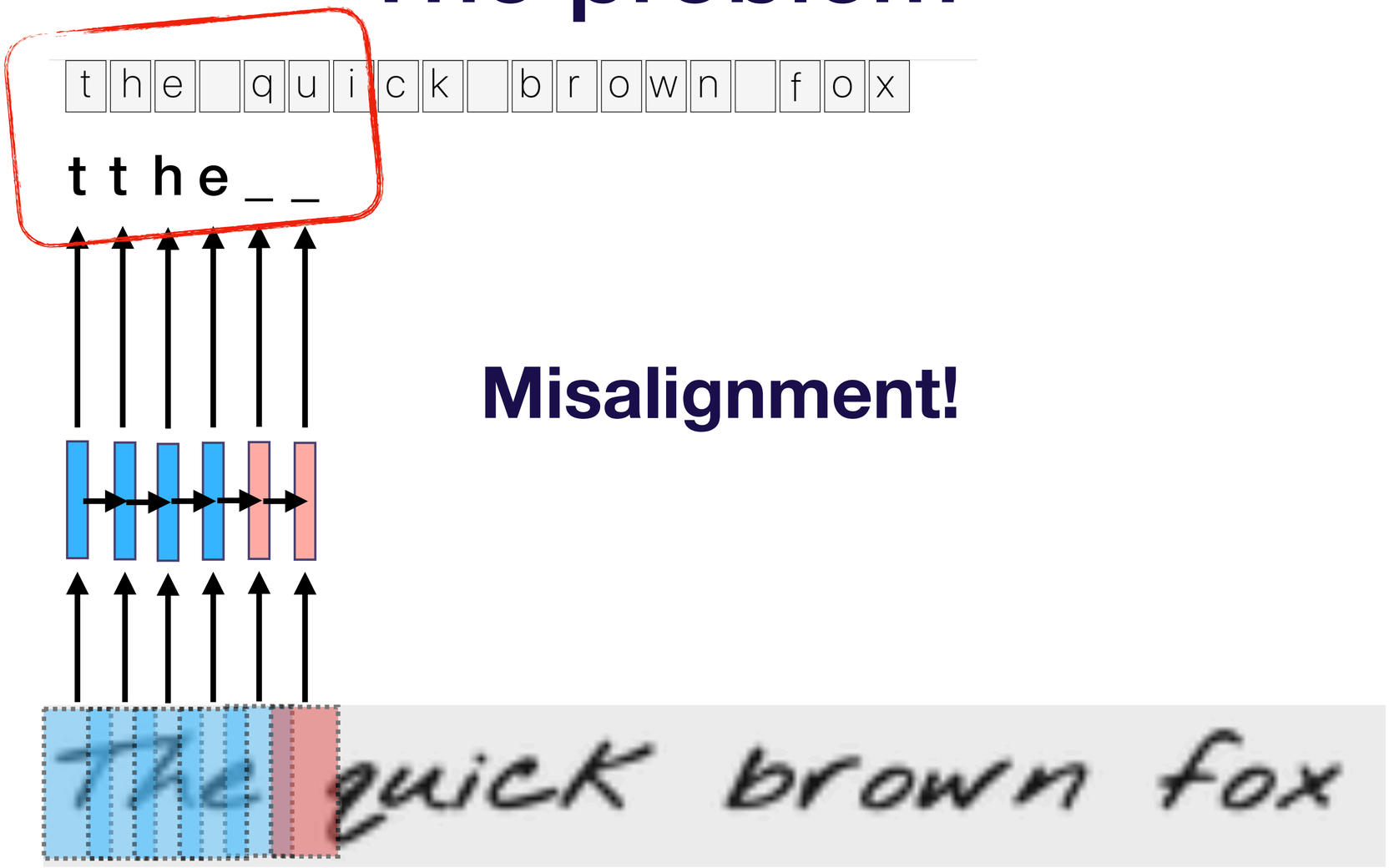


The problem

The problem

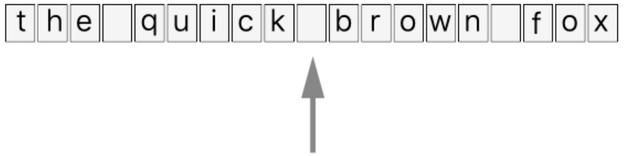


The problem



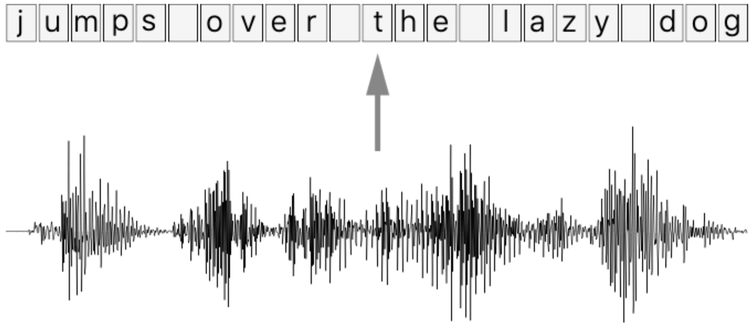
More generally

Y



X

Handwriting recognition: The input can be (x, y) coordinates of a pen stroke or pixels in an image.

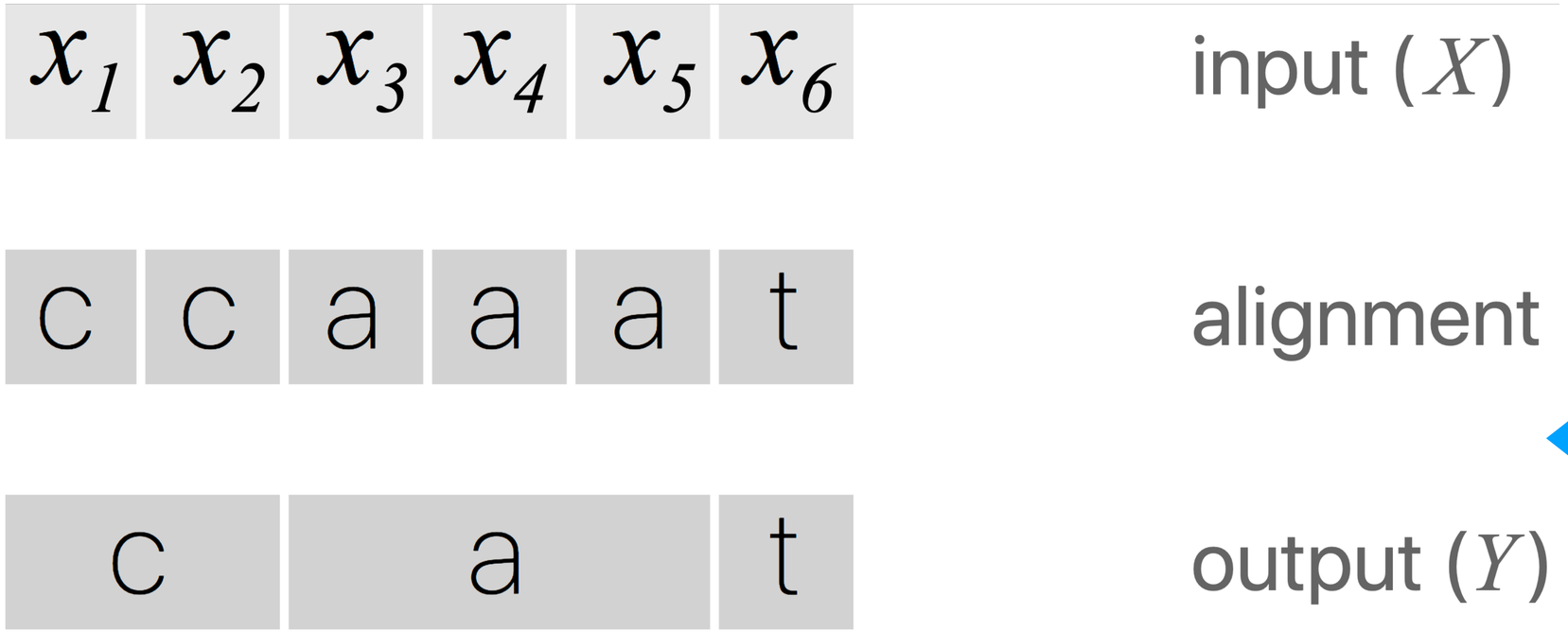


Speech recognition: The input can be a spectrogram or some other frequency based feature extractor.

- Both X and Y can vary in length.
- The ratio of the lengths of X and Y can vary.
- We don't have an accurate alignment (correspondence of the elements) of X and Y.

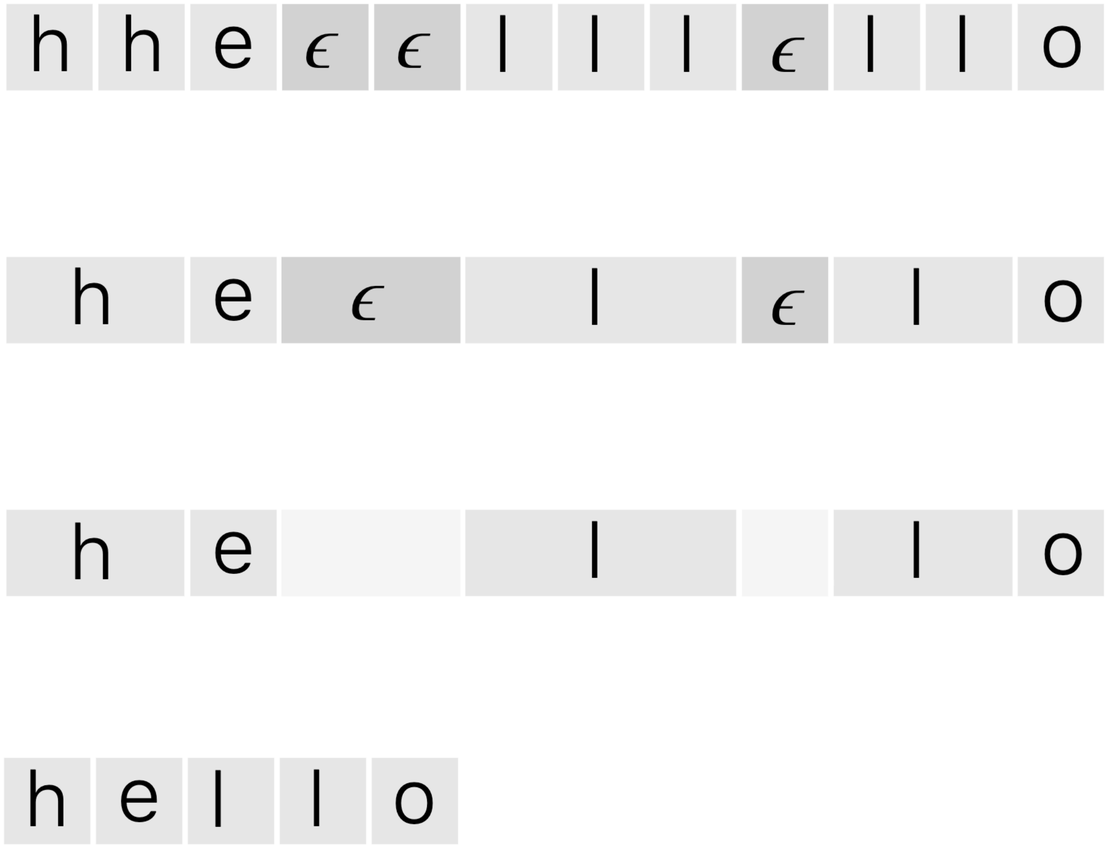
https://distill.pub/2017/ctc/

Intuition for the solution



https://distill.pub/2017/ctc/

CTC Loss



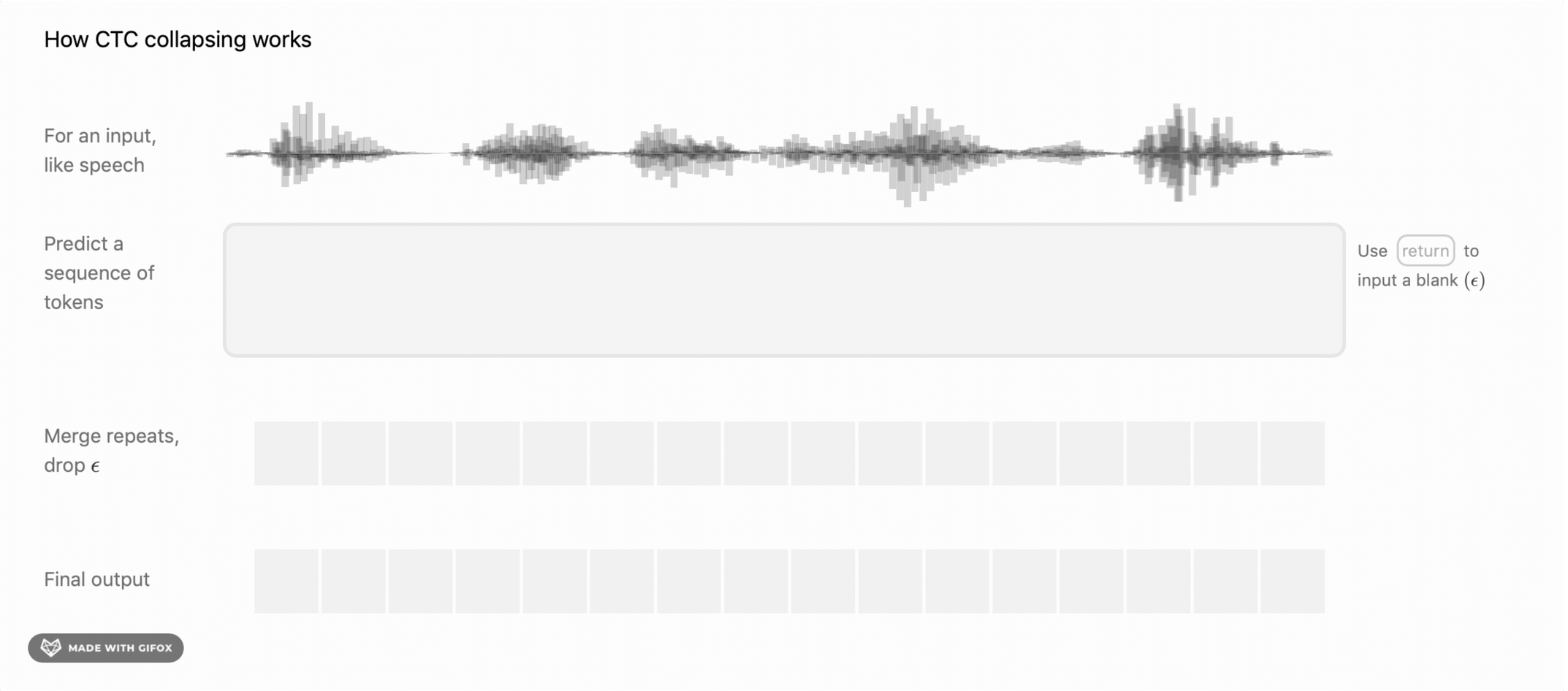
First, merge repeat characters.

Then, remove any ϵ tokens.

The remaining characters are the output.

https://distill.pub/2017/ctc/

For more of the technical details



https://distill.pub/2017/ctc/

We implement this in Lab 3!

Questions?

Agenda

1. Sequence Problems
2. RNNs
3. Vanishing gradients and LSTMs
4. Case study: Machine Translation
   (Bidirectionality and Attention)
5. CTC loss
6. **Pros and Cons**
7. A preview of non-recurrent sequence models

Pros

- Encoder / decoder LSTM architectures can model arbitrary (one-to-many, many-to-one, and many-to-many) sequence problems
- Many successes in NLP and other applications

Cons

- Recurrent network training is not as parallelizable as FC or CNN, due to the need to go in sequence
- Therefore much slower!
- Also can be finicky to train

Questions?

Agenda

1. Sequence Problems
2. RNNs
3. Vanishing gradients and LSTMs
4. Case study: Machine Translation
   (Bidirectionality and Attention)
5. CTC loss
6. Pros and Cons
7. **A preview of non-recurrent sequence models**

Sequence data does not require recurrent models!

- Today we will see a **convolutional** approach to sequence data modeling
- Next week we will expand this idea to all-fully-connected **Transformer** models

Approach we'll discuss

WaveNet: A Generative Model for Raw Audio
(van den Oord et al, 2016)

Used in Google Assistant and Google Cloud Text to Speech (e.g., for call centers)

Key ideas introduced

- Convolutional sequence models

Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?

Description of the problem

Input

“I am a student”

Output

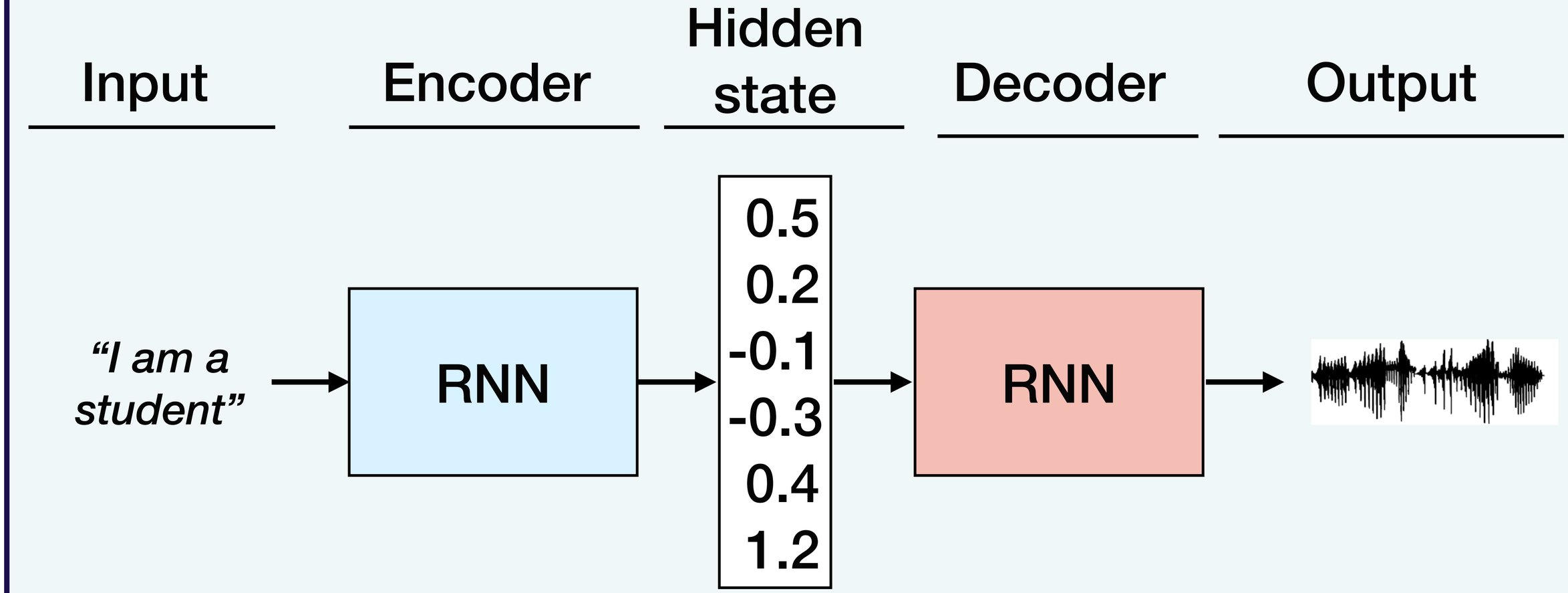
Audio waveform of speaking the text

Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?

Baseline model architecture

Encoder-decoder architectures



Baseline model architecture

Encoder-decoder architectures

Input

Encoder

Hidden state

Decoder

Output

“I am a student”

RN

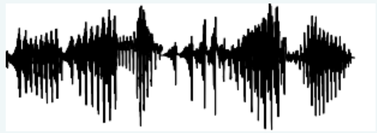
0.4

1.2

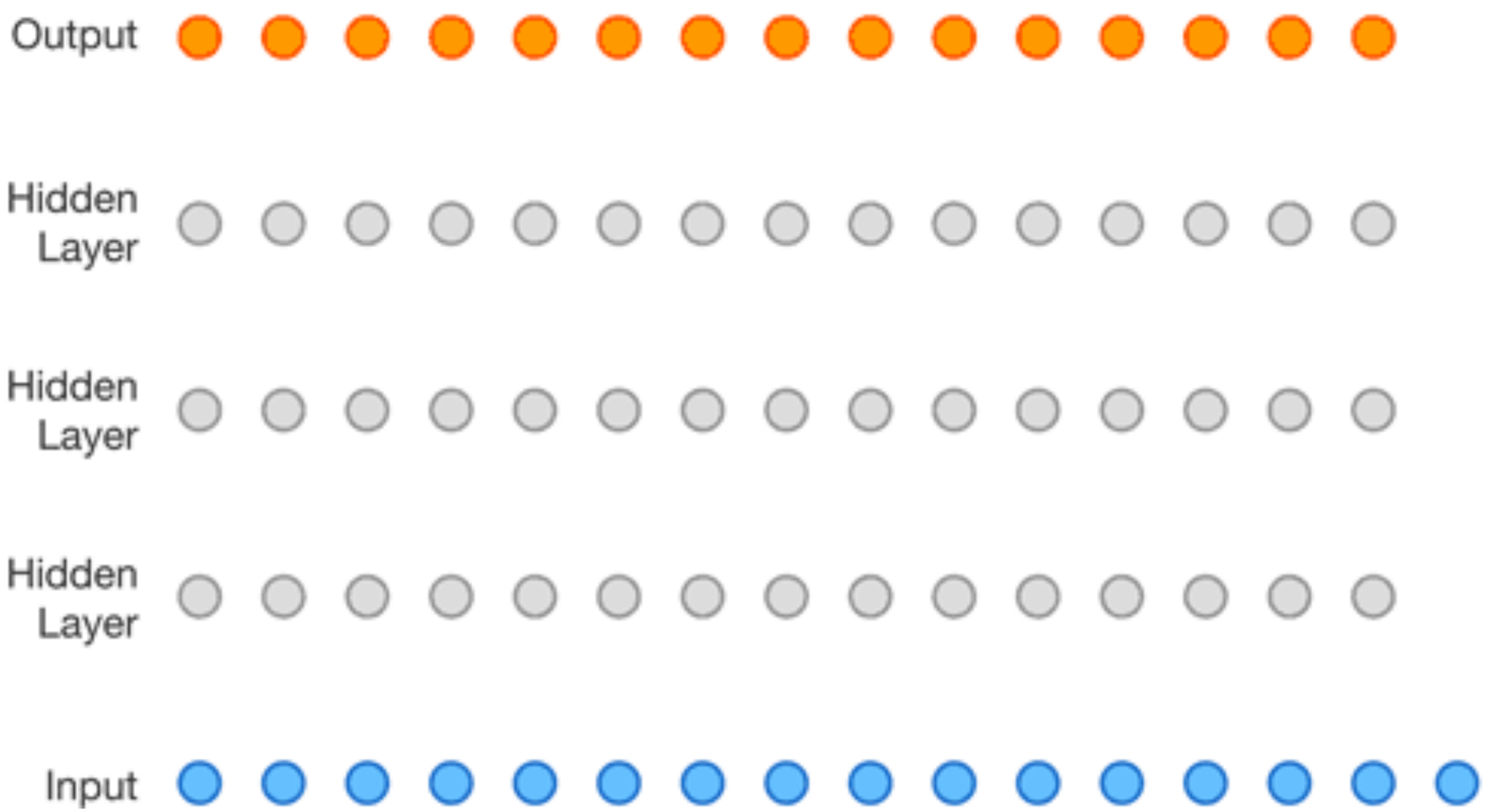
LSTM limitations that make audio modeling difficult:

(1) Long-term dependencies

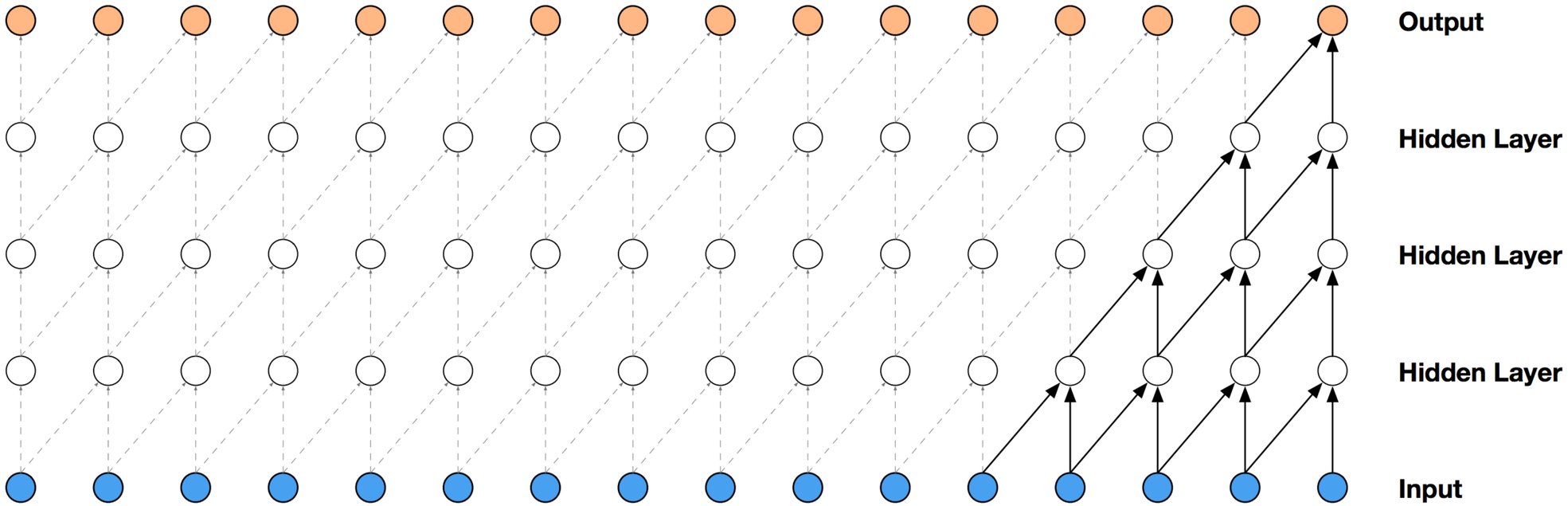
(2) Sequential training required



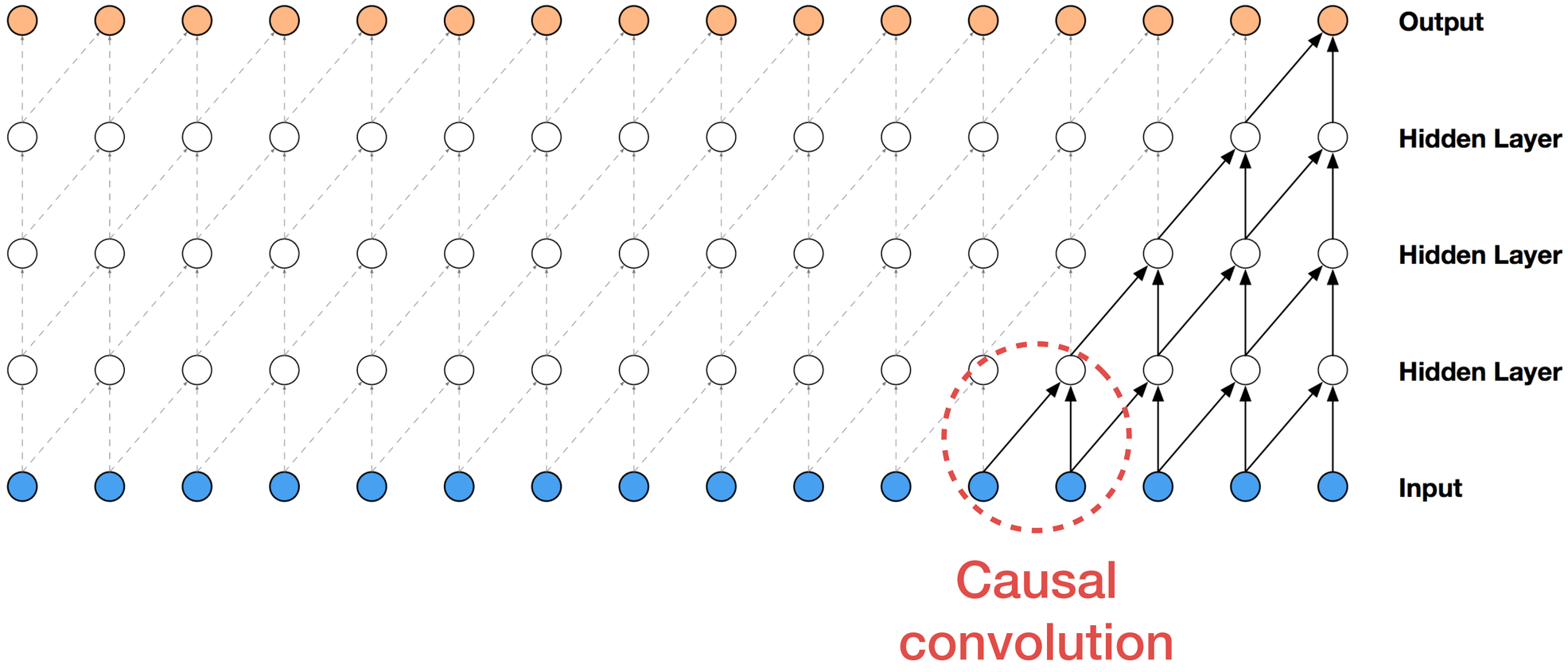
Insight: convolutions can be used for sequence data



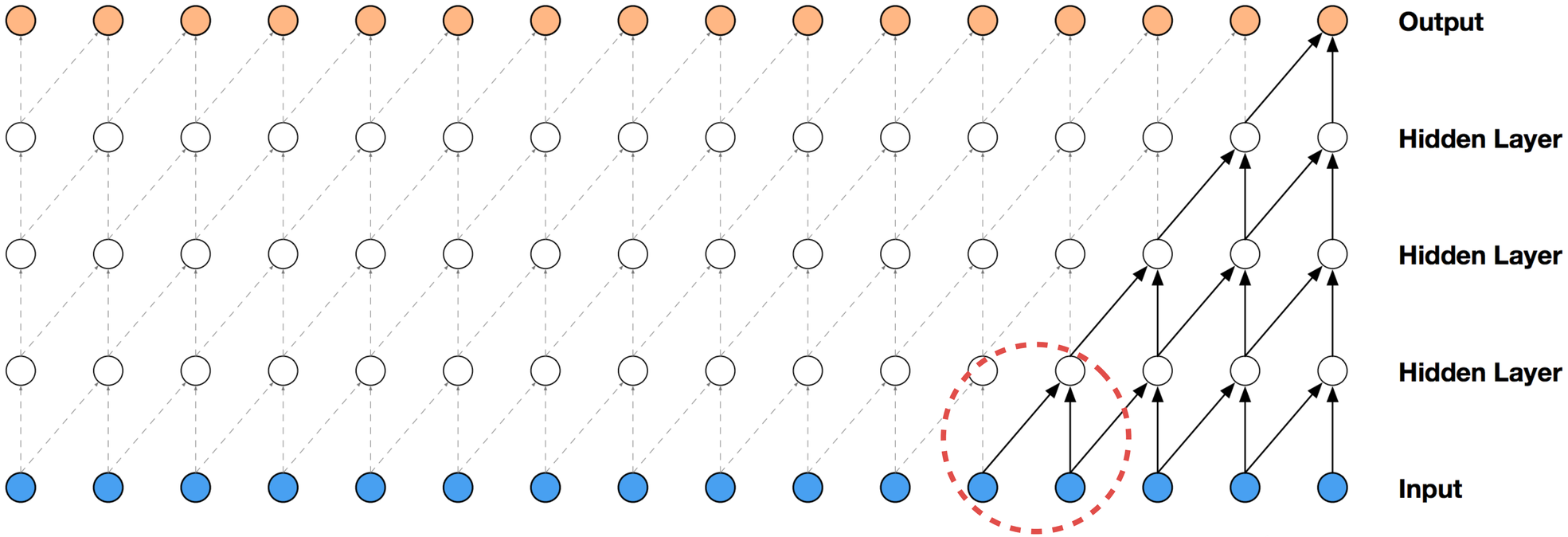
Insight: convolutions can be used for sequence data



Insight: convolutions can be used for sequence data



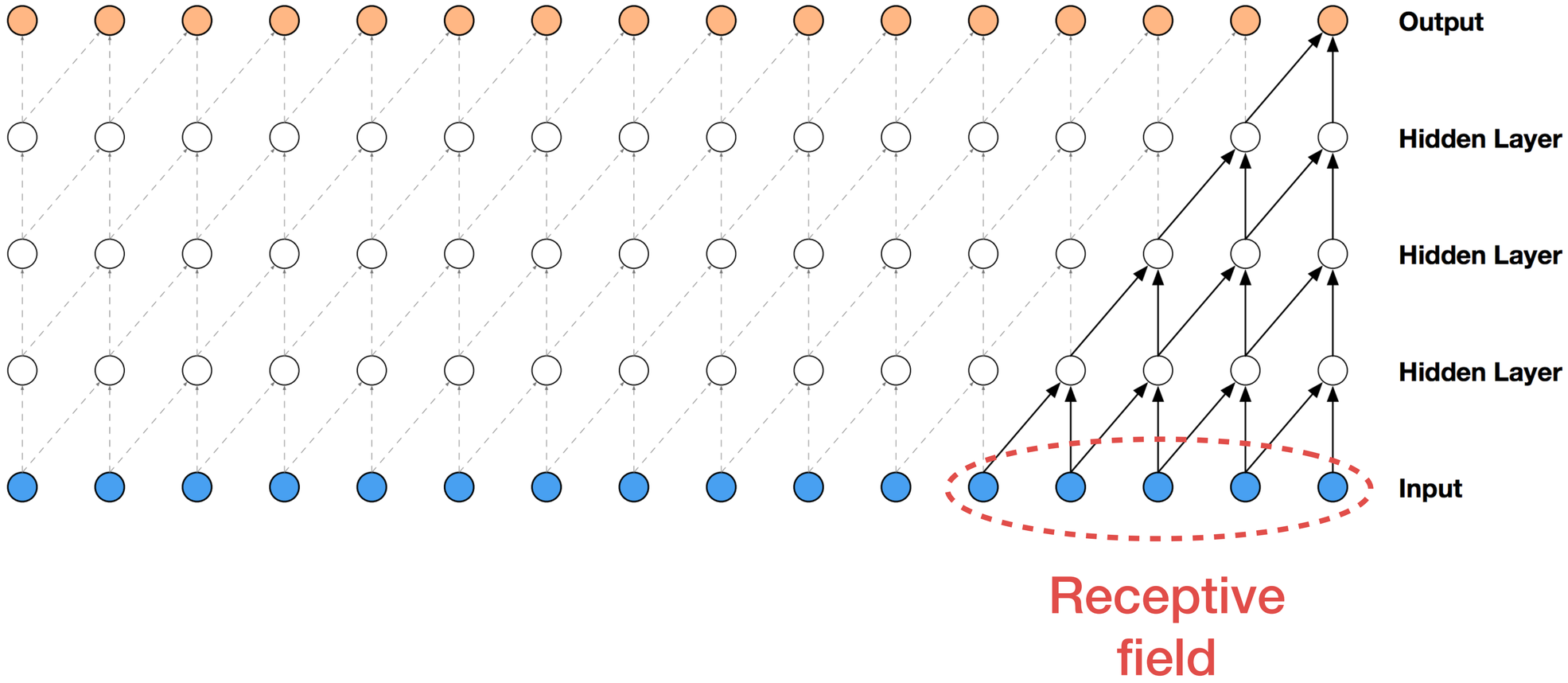
Insight: convolutions can be used for sequence data



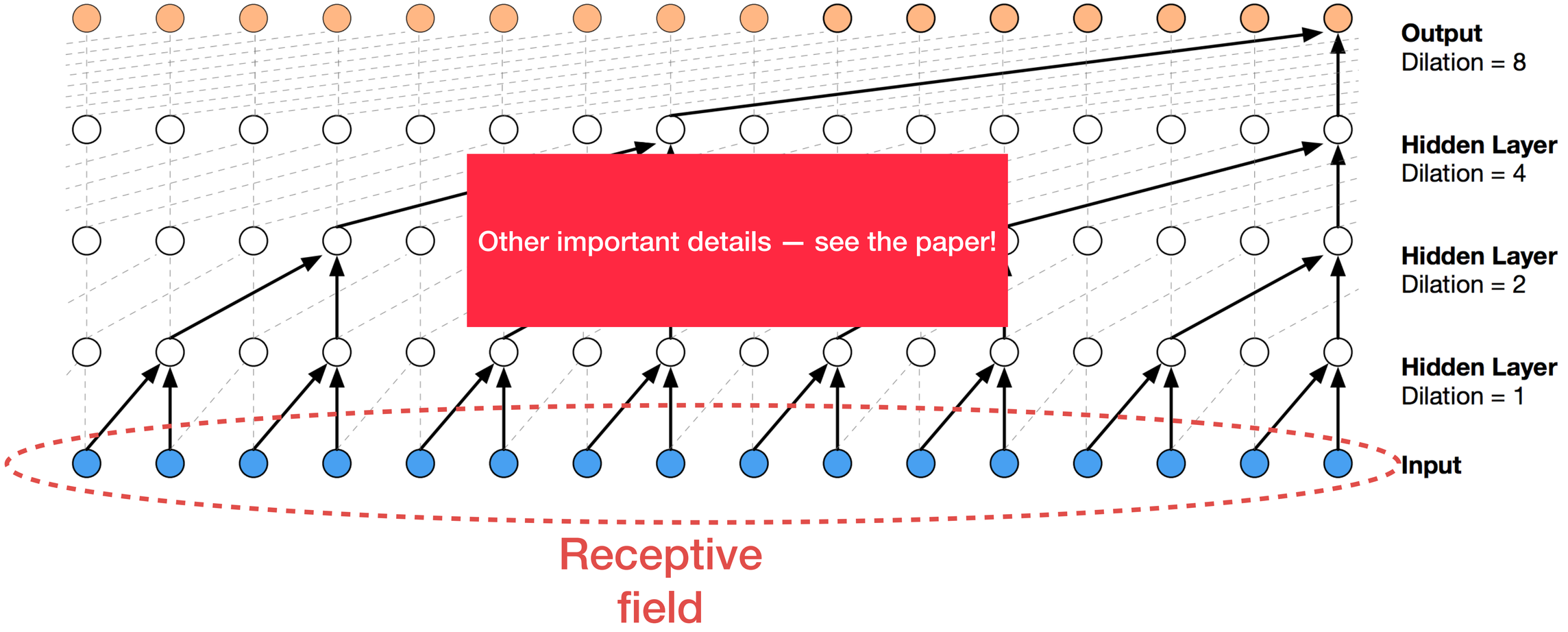
Causal convolution

$$\sum_{i=t-k}^{t} x_i w_i$$

Challenge: getting a large receptive field



Solution: dilated convolutions



Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?

WaveNet loss function

Minimize negative log likelihood

$$\mathcal{L} = \sum_{i=1}^{N} \log P_\theta(Y^i \mid X^i)$$

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{w_j \in Y^i} \log P_\theta(w_j \mid w_1, \cdots w_{j-1}, X^i)$$

Each log P(…) is a standard cross-entropy loss

Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?

Datasets for WaveNet TTS

- Internal Google datasets
  - North American English: 24.6 hours of speech
  - Mandarin Chinese: 34.8 hours
  - ~16,000 samples per second

Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?

Training WaveNet

- WaveNet training can be done in parallel, and no fancy tricks were needed to train it

Key questions for applications

- What **problem** are they trying to solve?
- What **model architecture** was used?
- What **loss function** was used?
- What **dataset** was it trained on?
- How did they do **training**?
- What tricks were needed for **inference** in deployment?

WaveNet inference

- Primary drawback of WaveNet: although training is parallel, inference is serial
- Followup paper introduced fast, parallel synthesis version of WaveNet: Parallel WaveNet

"Parallel WaveNet: Fast High-Fidelity Speech Synthesis", Van den Oord et al.

Summary of WaveNet approach

- Another way of dealing with long-term dependencies is to do away with recurrent networks altogether
- Instead, you can use a form of 1d convolutions called ***causal convolutions***
- To increase the receptive field of causal convolutions, can used ***dilated causal convolutions***
- In addition to long-term dependencies, main advantage of WaveNet is ***fast parallel training***
- Tradeoff is ***slow inference time***, which can be mitigated through the parallel WaveNet approach

Questions?