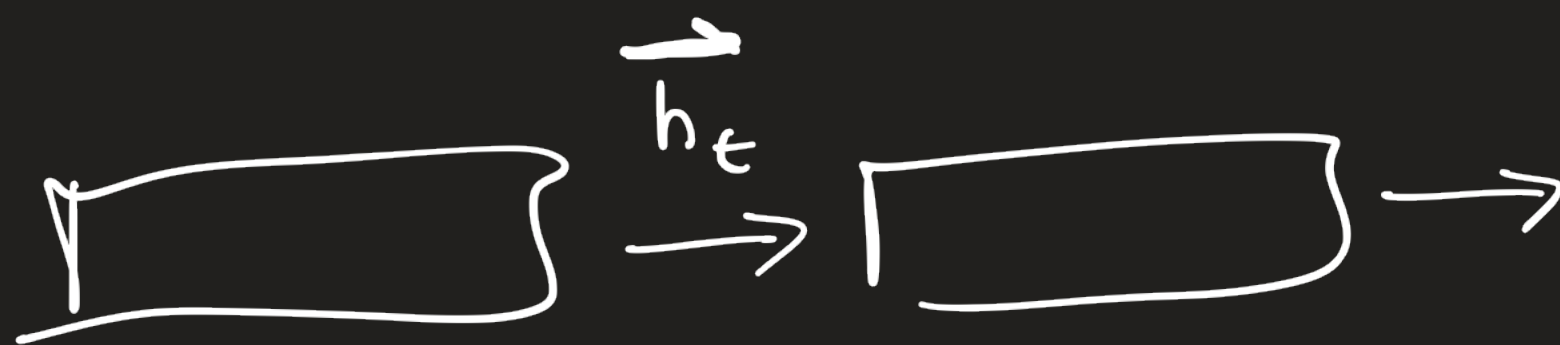


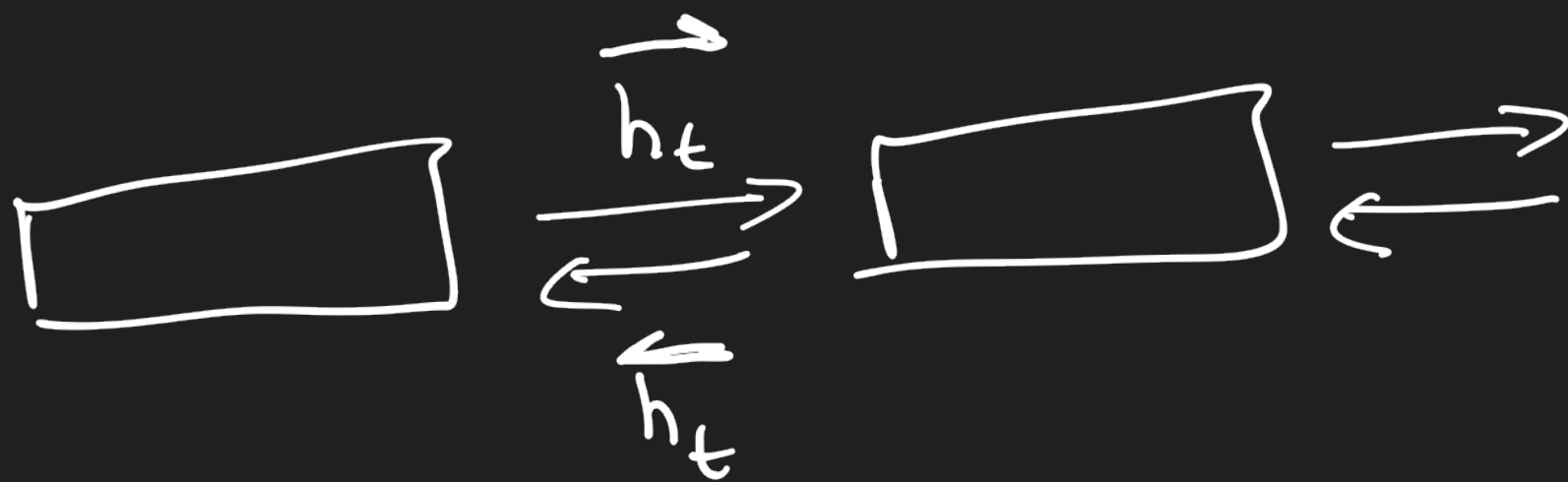
Bidirectional RNN

Very $\rightarrow$ how are you?
 $\uparrow$
well, good, etc.

The future can
provide useful
information as well.



standard RNN



bidirectional RNN

Bidirectional RNNs predict the current output $\hat{y}_t$ using the past and the future

$$\vec{h}_t = f(x_t, h_{t-1}, \vec{\omega}_h)$$

$$\overleftarrow{h}_t = f(x_t, h_{t+1}, \overleftarrow{\omega}_h)$$

$$h_t = \begin{bmatrix} \vec{h}_t \\ \overleftarrow{h}_t \end{bmatrix}$$

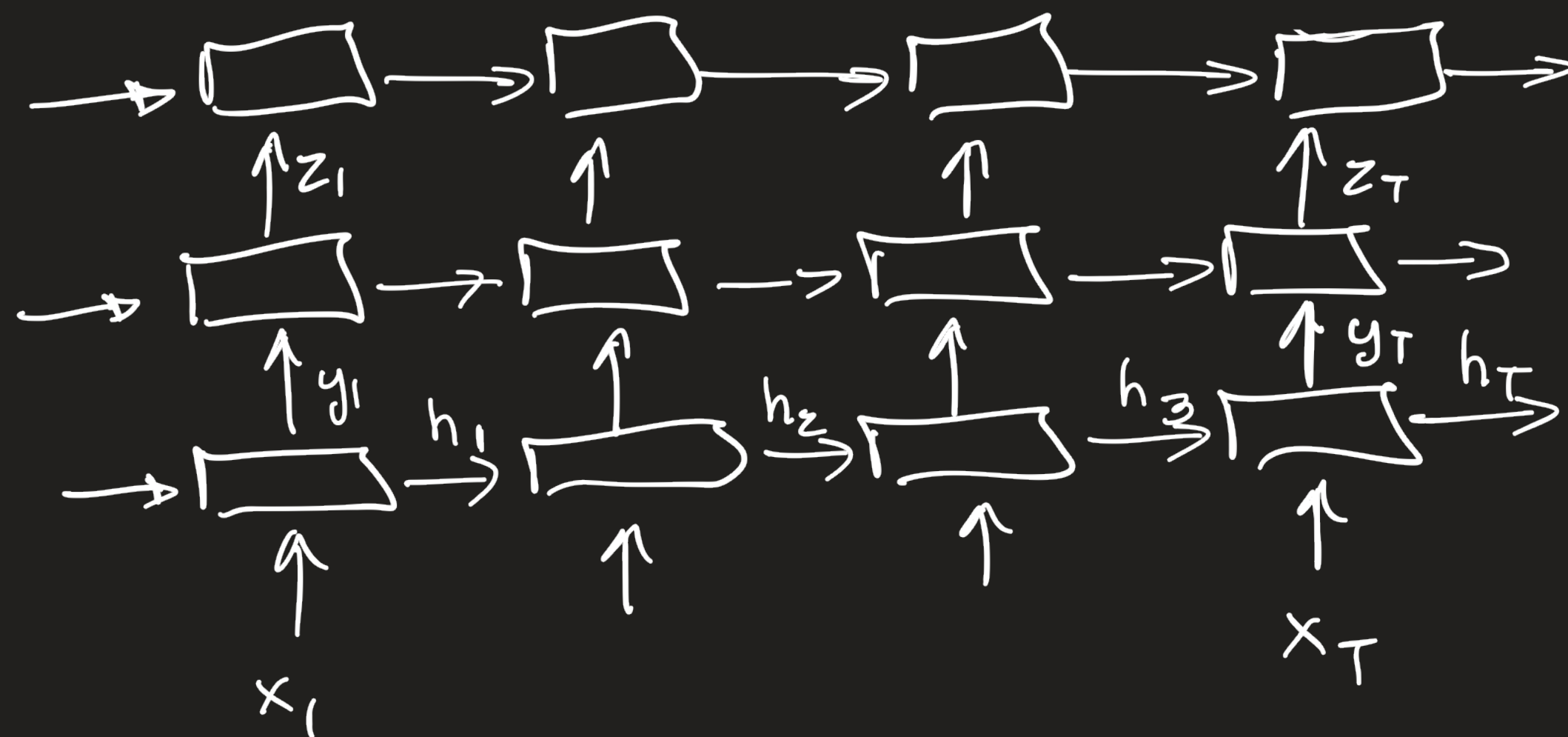
$$\hat{y}_t = g(h_t, \omega_o)$$

Useful for non-causal task where context is useful,
e.g. NLP tagging task (NER, POS, etc.), change point
detection in time series, translation

BTT needs to be modified to account for future dependencies

Deep RNNs

(typically used when "LSTMs" or "RNNs" are mentioned)



- train layer-wise, by computing all outputs from layer l so have inputs for layer $l+1$
- modify BPTT to account for dependencies between layers

Sequence-to-Sequence Modeling (2014 used for very good machine translation, popularized deep LSTM)

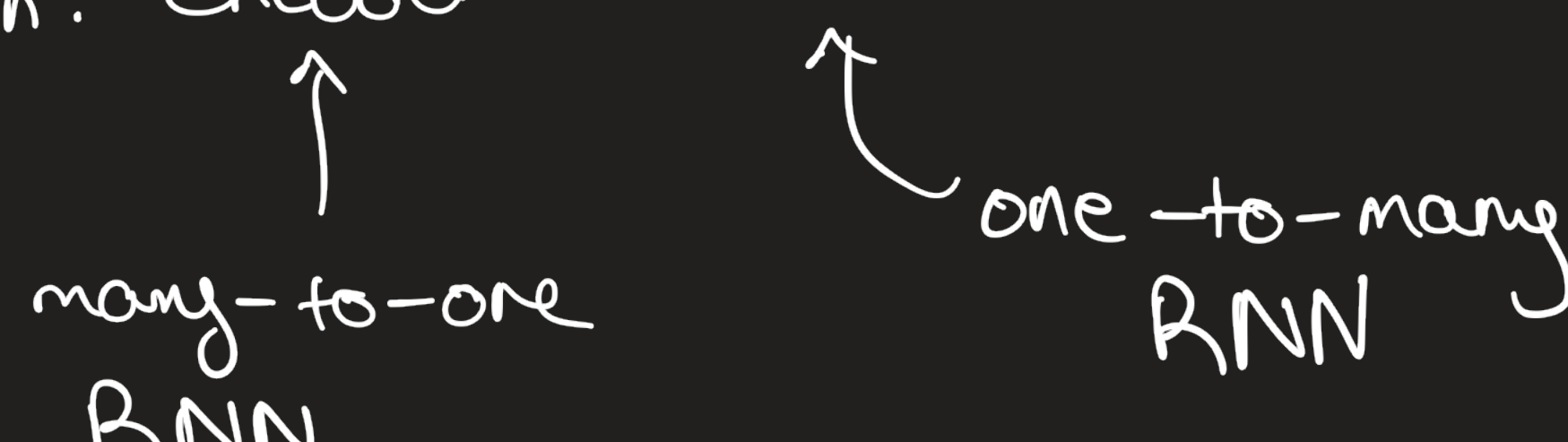
Sutskever et al. "Sequence to Sequence Learning w/ NNS" (seq-to-seq model)

Goal: given input sequence of tokens $(x_1, \dots, x_T)$
predict output sequence of tokens $(y_1, \dots, y_{T'})$

Canonical example: Machine Translation

Issue: T may be different than T' so the many-to-many design paradigm for RNNs may not fit

Soln: encoder-decoder architecture

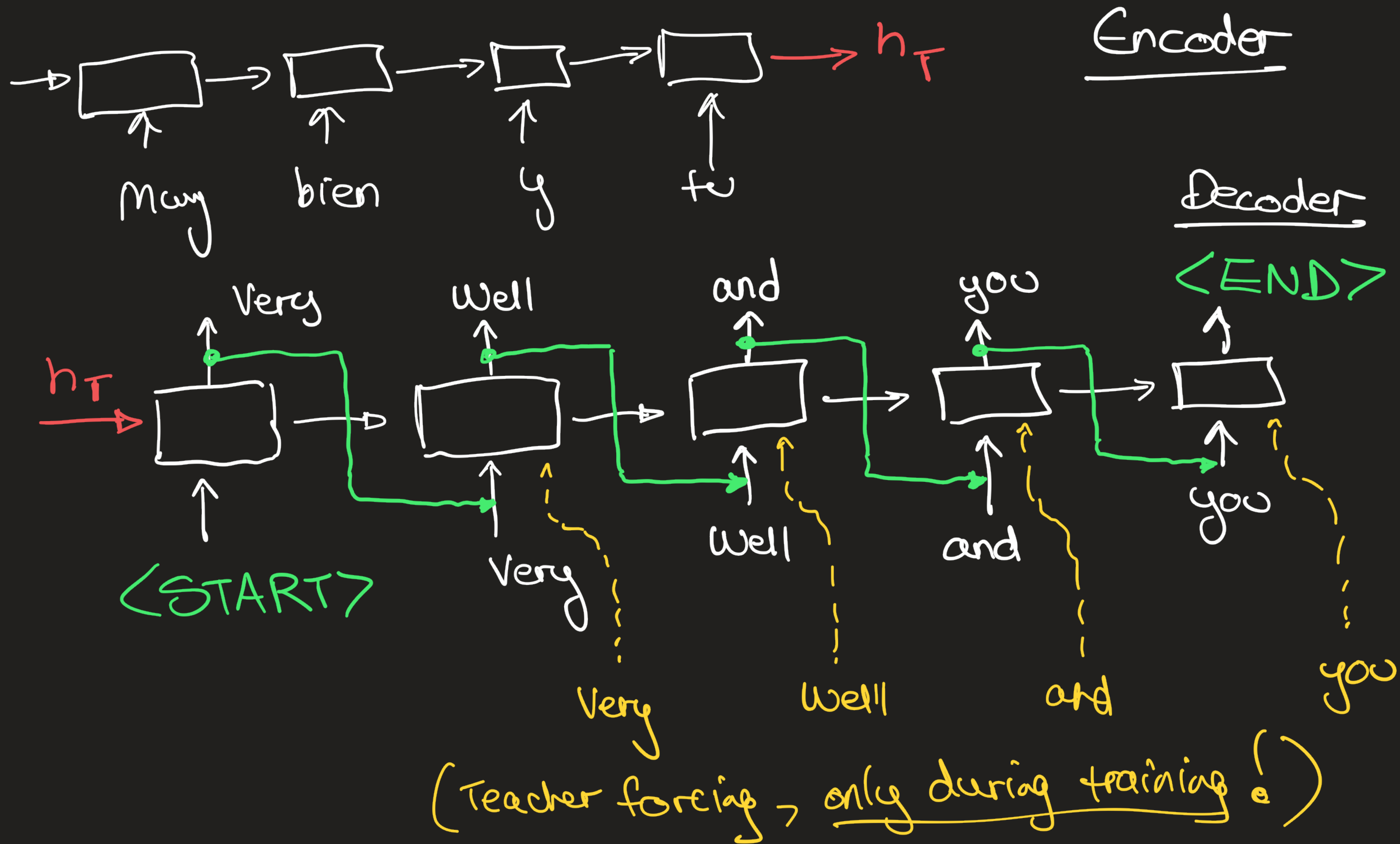


The diagram shows the encoder-decoder architecture. An upward arrow from the text 'many-to-one RNN' points to the 'encoder' part of 'encoder-decoder architecture'. A curved upward arrow from the text 'one-to-many RNN' points to the 'decoder' part of 'encoder-decoder architecture'.

many-to-one RNN

one-to-many RNN

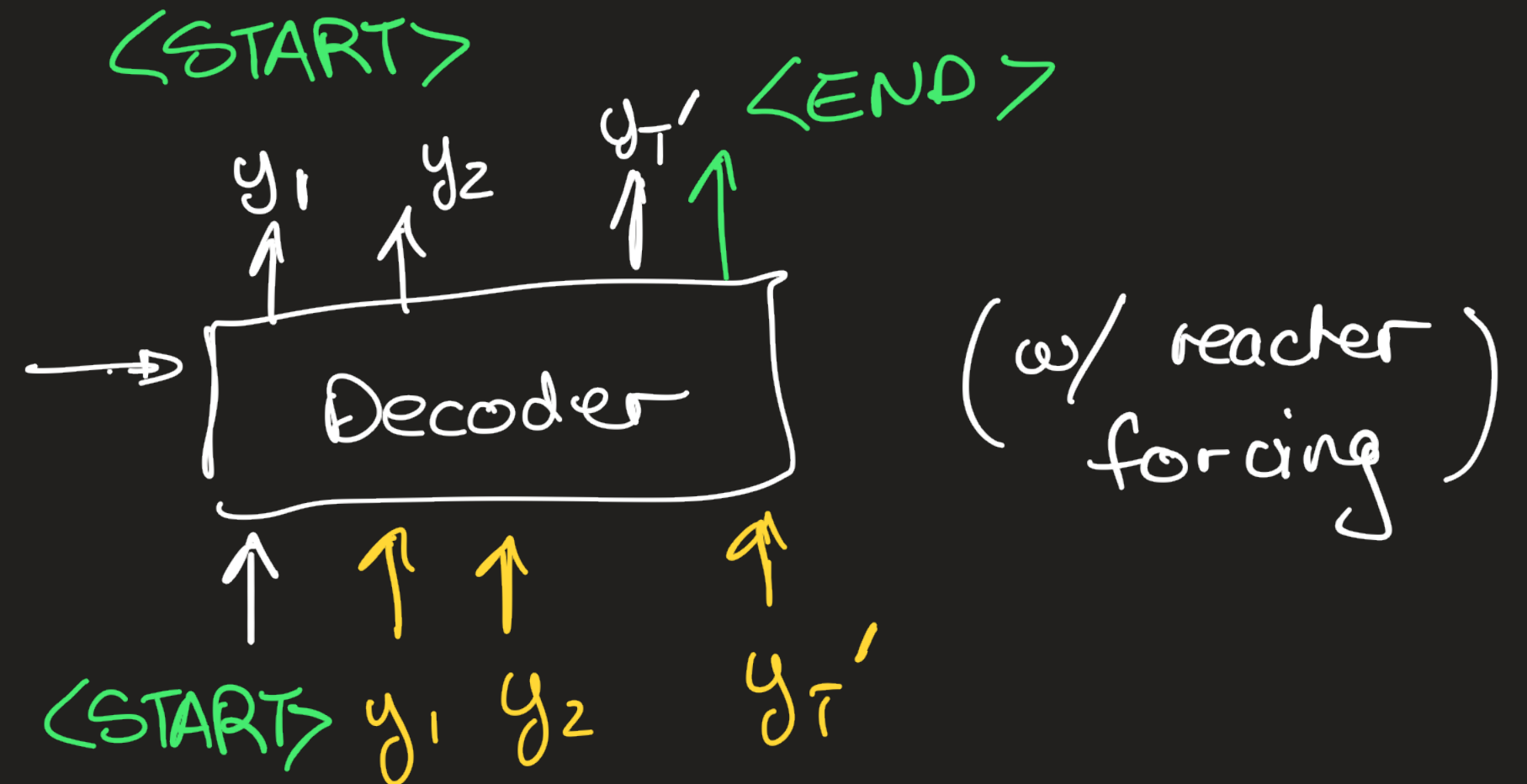
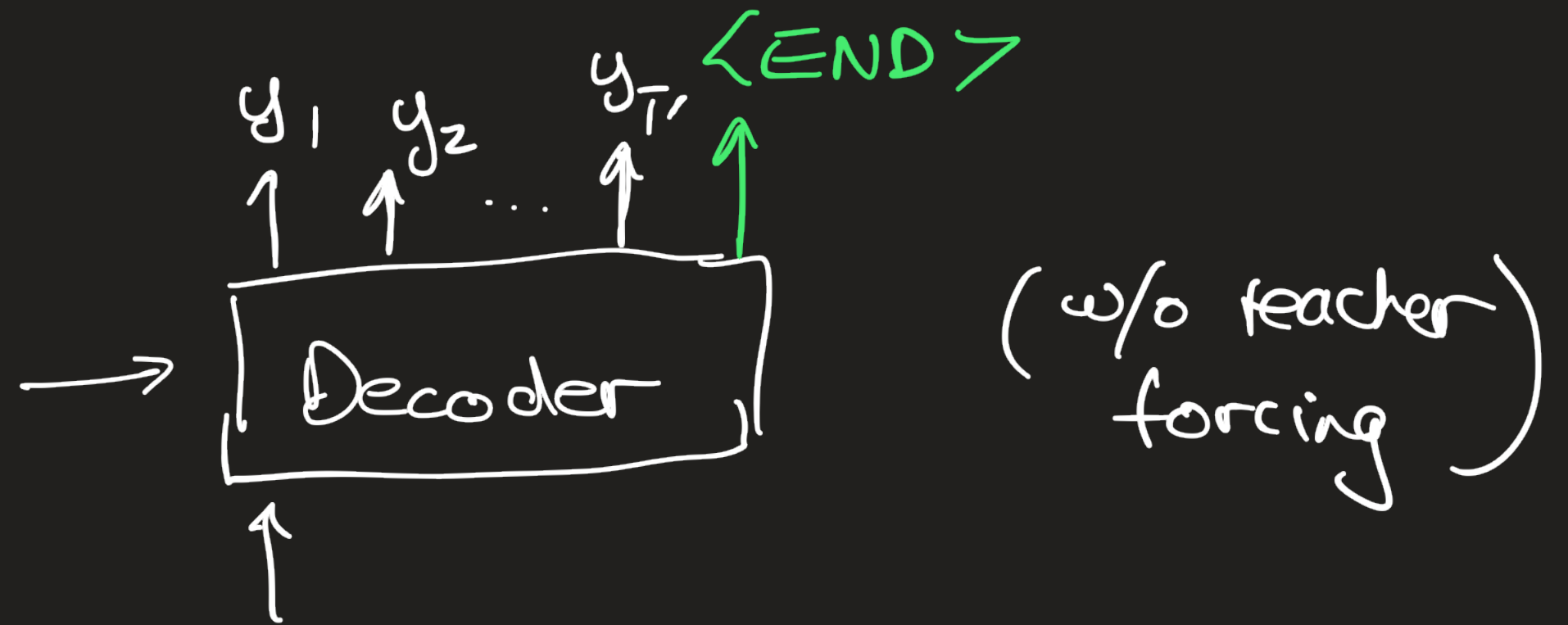
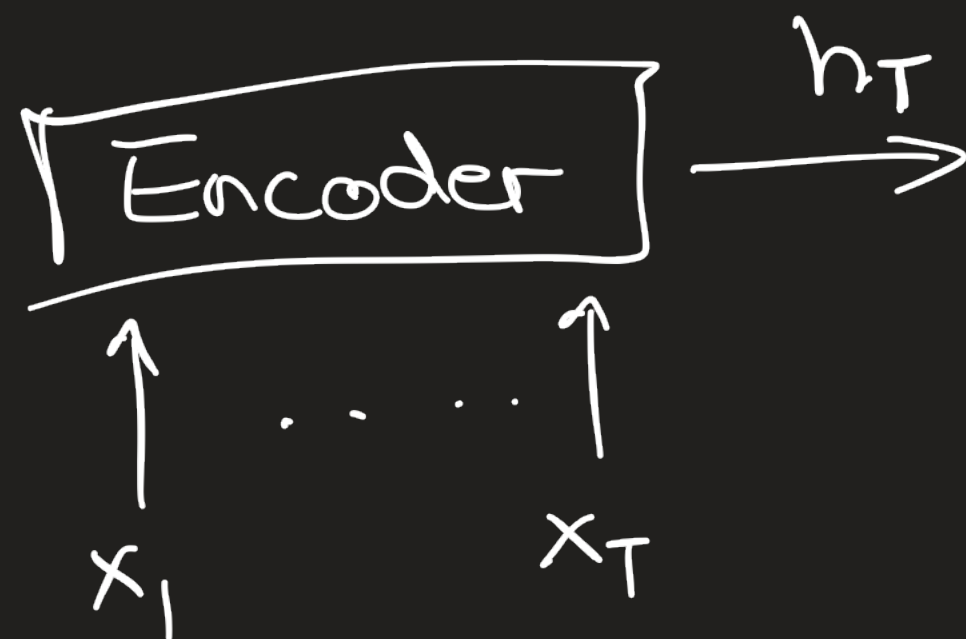
Sequence-to-Sequence architecture



Training:

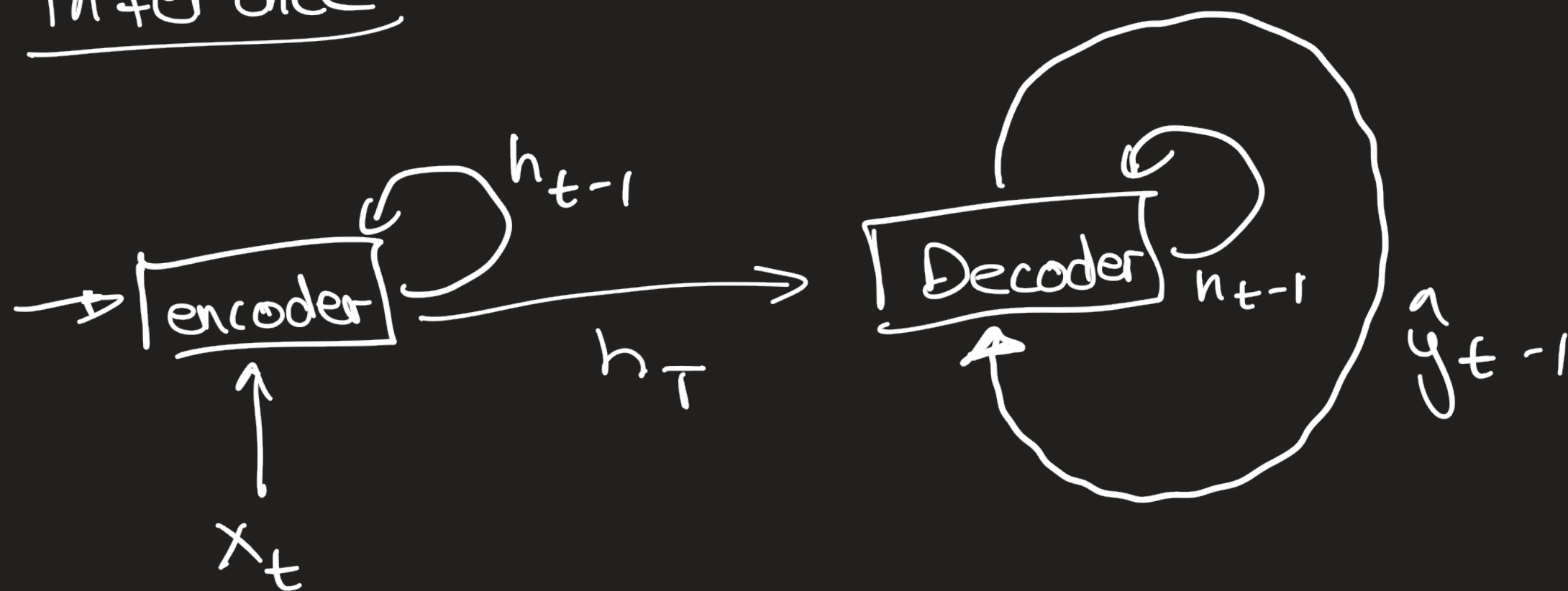
- train the encoder & decoder simultaneously (using BTT) so that given the input sequence $(x_1, \dots, x_T)$ and output sequence $(y_1, \dots, y_{T-1})$:
 - encoder learns an informative h_T for decoding
 - decoder learns how to, given h_T and a special **<START>** token, generate the correct output followed by an **<END>** token.
- Since the output $\hat{y}_t$ depends on $\hat{y}_{t-1}$ we can have very slow training if $\hat{y}_t$ is inaccurate at the start of the sequence, because the errors propagate. Instead, use **Teacher forcing**: use y_{t-1} (the ground truth) as the input to predict $\hat{y}_t$ at training time

Training



As our train inputs: $(x_1, \dots, x_T)$ and $(\langle \text{START} \rangle, y_1, \dots, y_{T'})$
and as training outputs: $(y_1, \dots, y_{T'}, \langle \text{END} \rangle)$

Inference

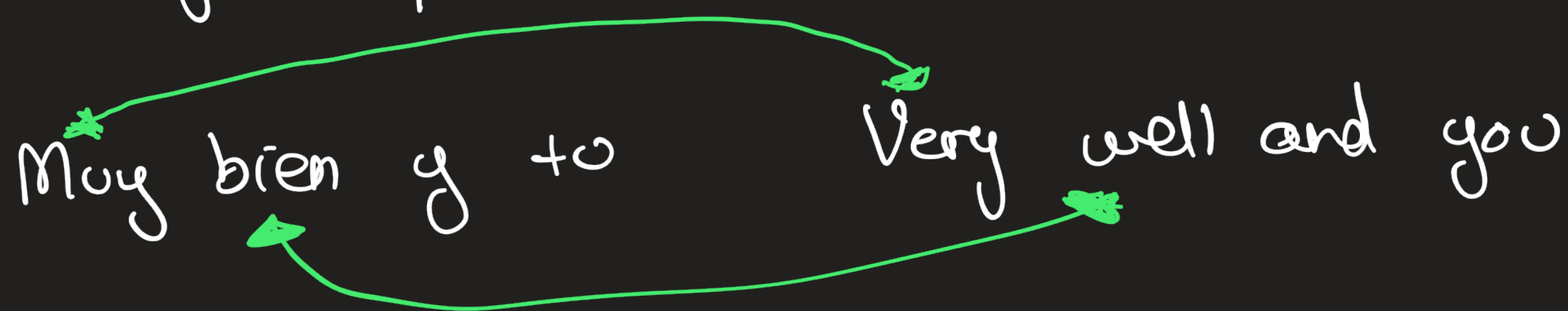


- Given input sequence $(x_1, \dots, x_T)$ use encoder to obtain h_T
- initialize the decoder w/ hidden state h_T and feed it the **<START>** token
- At each time, feed $\hat{y}_{t-1}$ into the decoder to predict the next token and take $\hat{y}_t$ to be the most likely token

Details

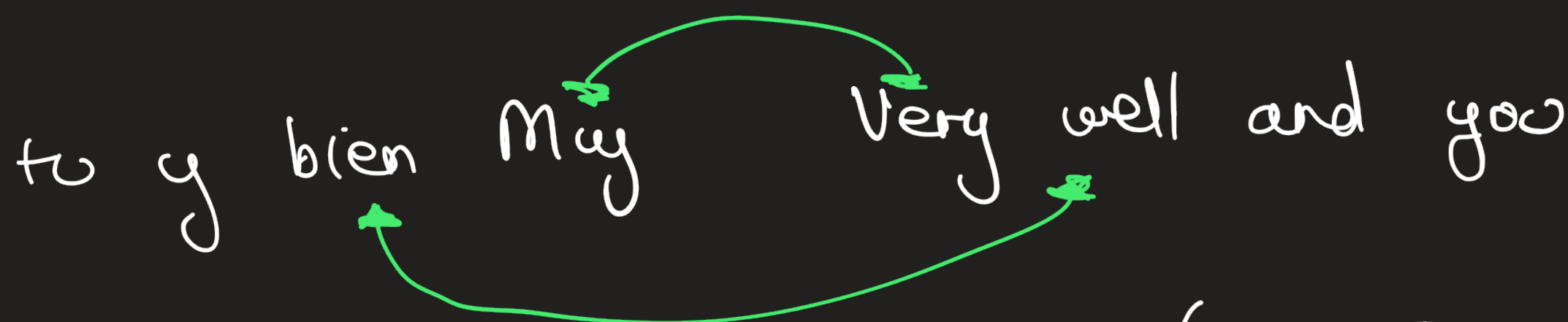
- use deep LSTMs (GRUs) in the encoder and decoder
- reverse the order of the source sequence to introduce shorter length dependencies w/ the target sequence

Muy bien y to Very well and you



A diagram illustrating a long dependency in a source sequence. The text "Muy bien y to" is on the left and "Very well and you" is on the right. A green arrow starts from the word "to" and points to the word "Very", indicating a long-distance dependency.

to y bien Muy Very well and you



A diagram illustrating a shorter dependency in a source sequence. The text "to y bien Muy" is on the left and "Very well and you" is on the right. A green arrow starts from the word "Muy" and points to the word "Very", indicating a shorter-distance dependency compared to the previous example.

- use "beam decoding" in practice (see Sutskever et al.)

Recap course:

- basic prob theory & optim theory needed for ML
- several stochastic first-order algs for ML
(SGD, AdaGrad, Adam, Rmsprop)
- basic convex machine learning models (esp. logistic regression)
- practice w/ implementing these algs & seeing impact of hyperparam choices
- kernel learning for nonlinear ML w/ potentially infinite dim features
- fully nonlinear ML using NNs (tradeoffs inherent in NNs:
no global minimum, hyperparam sxn is larger issue, training time)

- different NN architectures for different problem classes:
 - Fully connected (general prob) ; autoencoders
 - CNNs for vision; several popular architectures
 - RNNs for sequential data
- Mitigation strategies for exploding & vanishing gradients

Topics we didn't cover:

- Gaussian Processes
- attention & transformers (canonical architecture for NLP)
- graph NNs
- generative models (GANs and VAEs)
- deep RL
- online learning > bandits