CAI 4104/6108 – Machine Learning Engineering:

Recurrent Neural Networks 2

Prof. Vincent Bindschaedler

Spring 2024

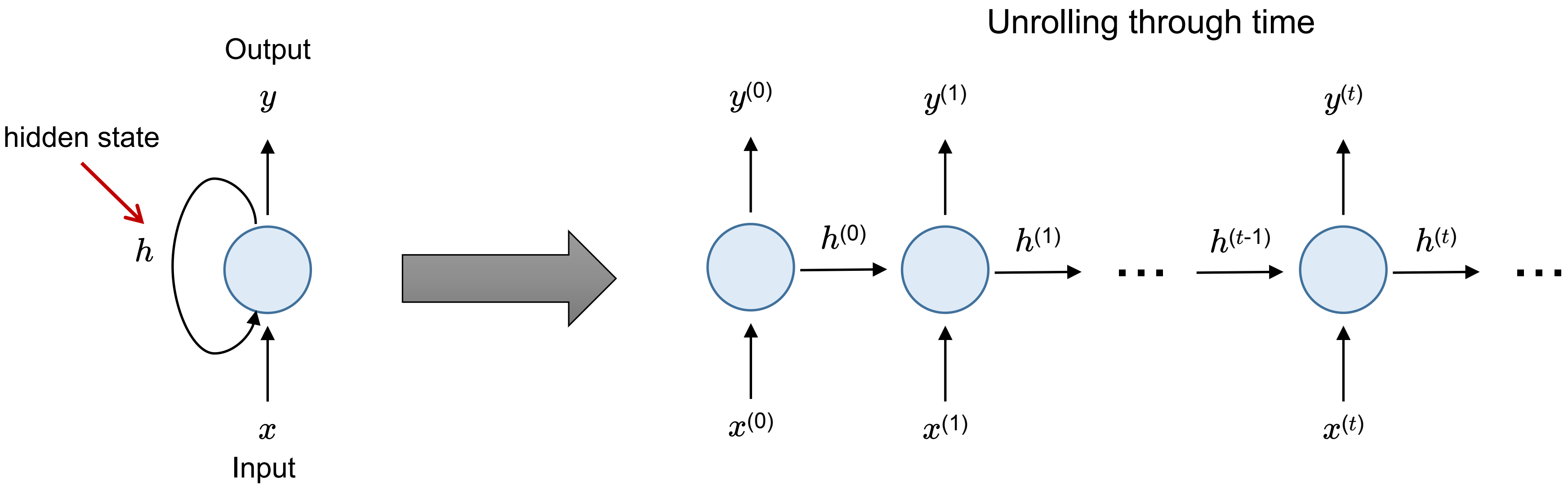
Administrivia

- Project Proposals
  - Due today (3/27) at 11:59pm
  - Please make sure to use the Canvas Project Groups functionality
    - Only one person in the group needs to submit the PDF

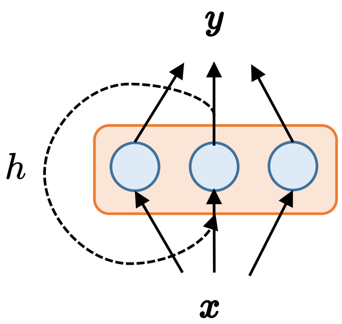
- Homework 4 is out
  - Topic: (debugging) training neural networks (& some CNNs)
  - Due 4/3

Reminder: Recurrent Neurons/Units

- Recurrent Layers:
  - Made up of recurrent neurons/units which keep state
  - State at time t: $h^{(t)}$ is a function g of the previous state $h^{(t\text{-}1)}$ and the current input $x^{(t)}$
    - $h^{(t)} = g(h^{(t\text{-}1)}, x^{(t)})$



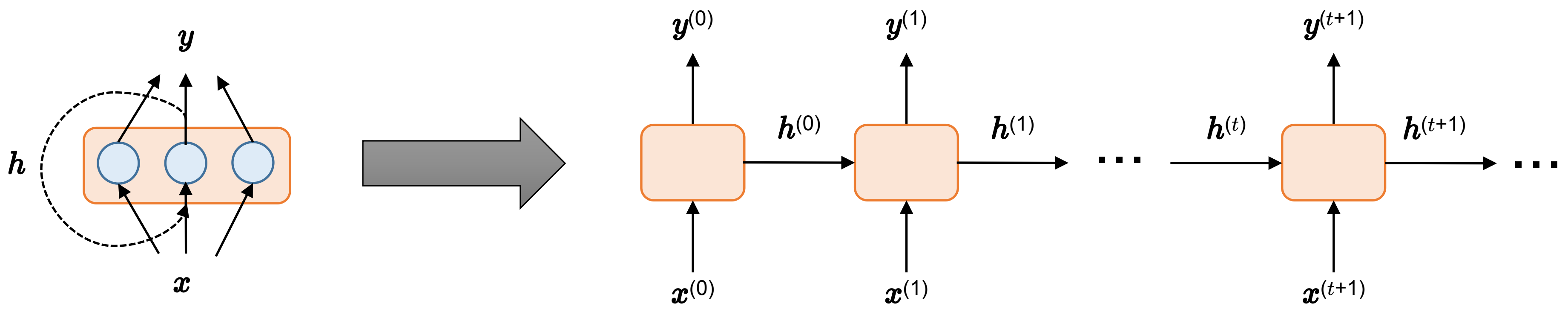
- Recurrent Layers:
  - Weight matrices: $\boldsymbol{W}$ $(m \times k)$ and $\boldsymbol{V}$ $(k \times k)$ Bias vector: $\boldsymbol{b}$ $(k \times 1)$
    - k is the number of units/neurons
  - Activation function: f (e.g., tanh)
  - Hidden state vector: $\boldsymbol{h}^{(t)} = \boldsymbol{V}\, \boldsymbol{y}^{(t\text{-}1)}$ (e.g., we can set $\boldsymbol{h}^{(0)} = 0$)
  - Output vector: $\boldsymbol{y}^{(t)} = f\,(\boldsymbol{W}^T \boldsymbol{x}^{(t)} + \boldsymbol{h}^{(t)} + \boldsymbol{b})$



Reminder: Processing Sequences

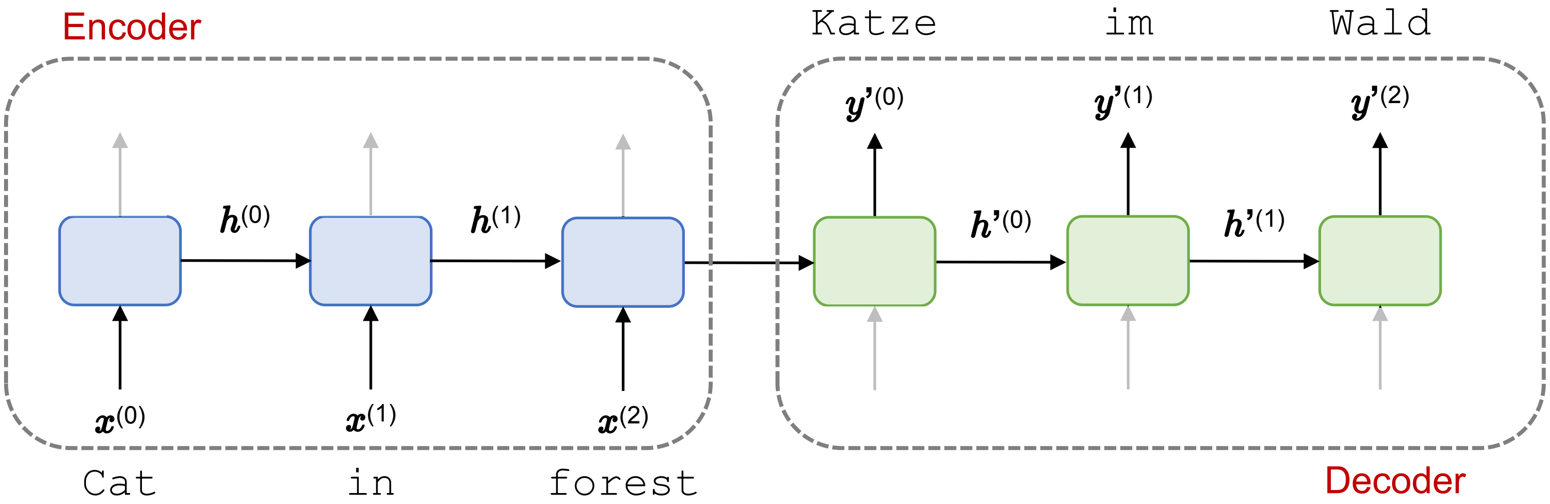
- Architecture & Tasks:
  - Sequence-to-sequence: from an input sequence produce a sequence as output
  - Vector-to-sequence: from a fixed length input produce a sequence as output
  - Sequence-to-vector: from an input sequence produce a fixed length output
  - Encoder-decoder networks: sequence-to-vector followed by vector-to-sequence

Unrolling through time



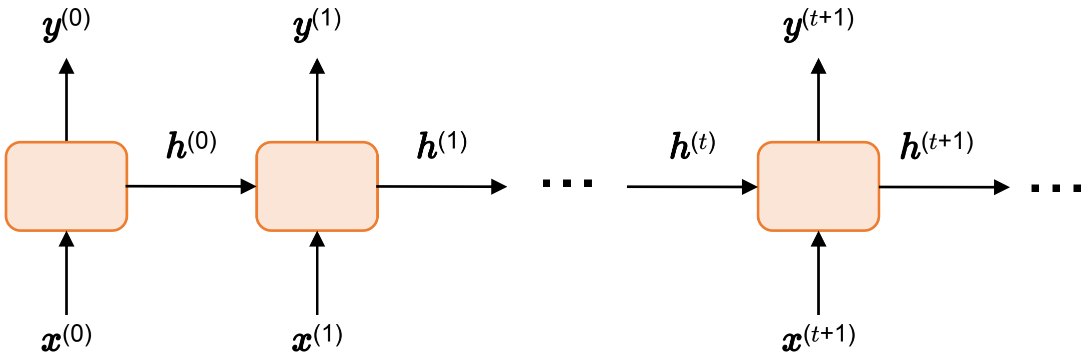
Reminder: Processing Sequences

- Architecture & Tasks:
  - Encoder-decoder networks: sequence-to-vector followed by vector-to-sequence
  - Example: Language translation
    - Translate a sentence from one language to another



Reminder: Training RNNs

- How does training work?
  - Backpropagation through time
  - Note:
    - Loss is typically averaged over the entire output
    - The weights are shared across time
    - Training is slow



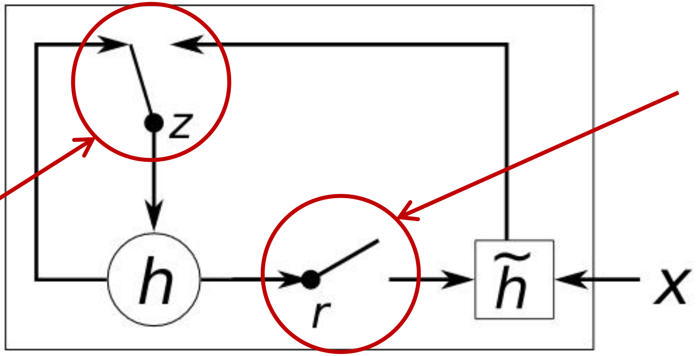
Reminder: Gradients & Short-Term Memory

- Unstable gradients problem
  - Activation functions that do not saturate (e.g., ReLU) can make things worse
    - Typically, we use activation functions such as tanh or sigmoid
  - We cannot use batch normalization across time steps
    - But we can use gradient clipping

- Short-term memory problem
  - RNNs cannot remember long-term dependencies well
    - Intuition: information is lost at each time step
  - Mitigation
    - Use a different type of cell (e.g., LSTM, GRU, etc.)

- Types of cells
  - Simple/traditional RNN cell
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)
    - Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.” In EMNLP, 2014.

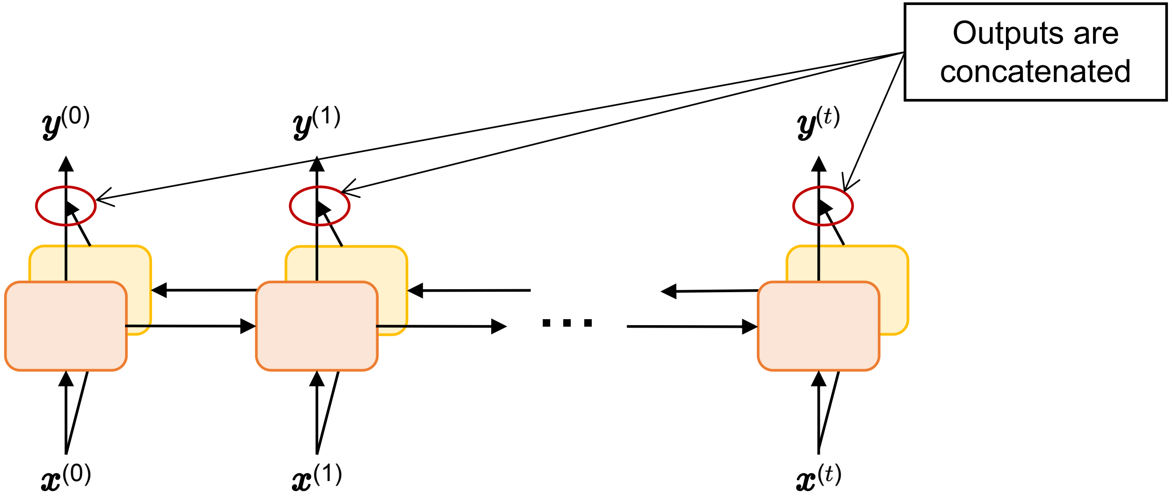
update gate *z* — allows information from previous hidden state to carry over to current hidden state



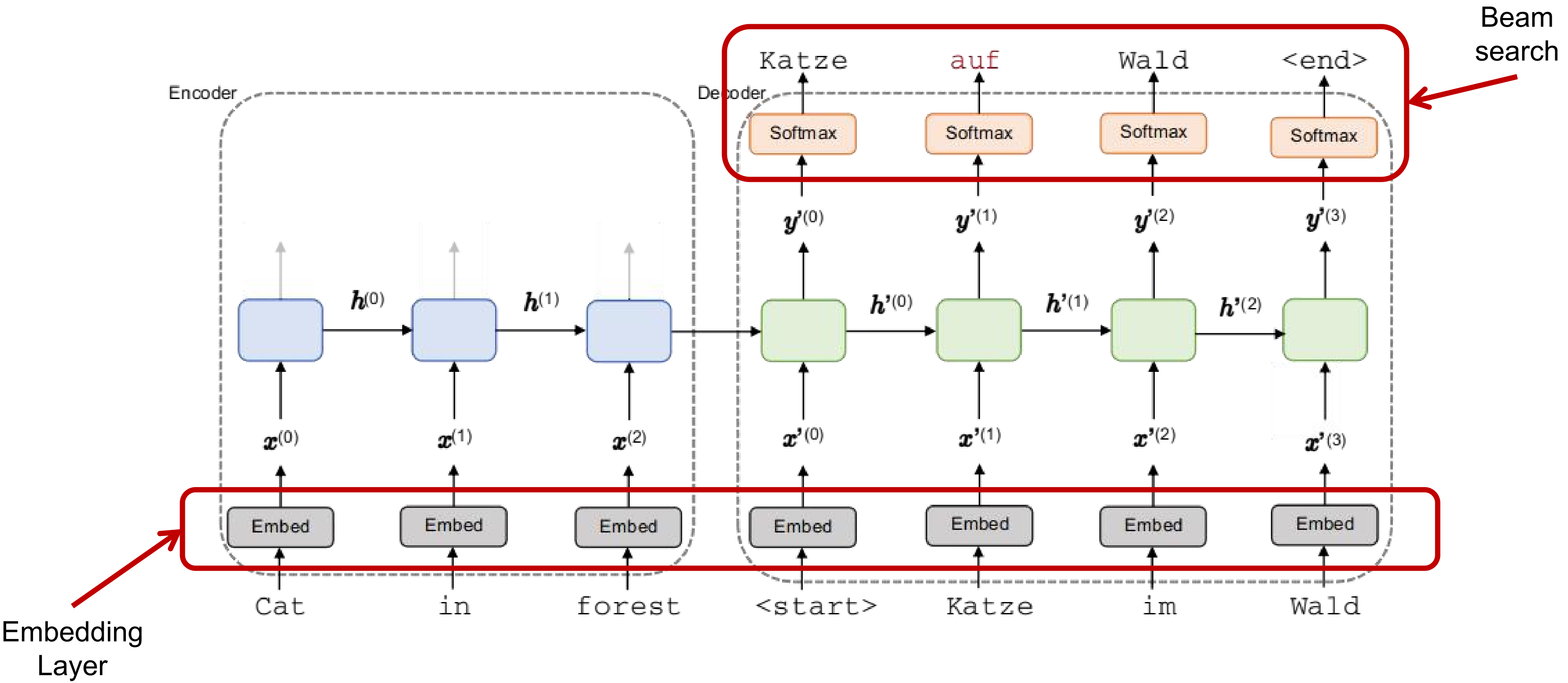
Reset gate *r* — decides if previous hidden state is ignored (reset to current input)

Deep RNNs & Bidirectionality

- Deep RNNs:
  - RNNs with multiple recurrent layers
- Bi-directional RNNs:
  - Takes in sequence forwards and also backwards

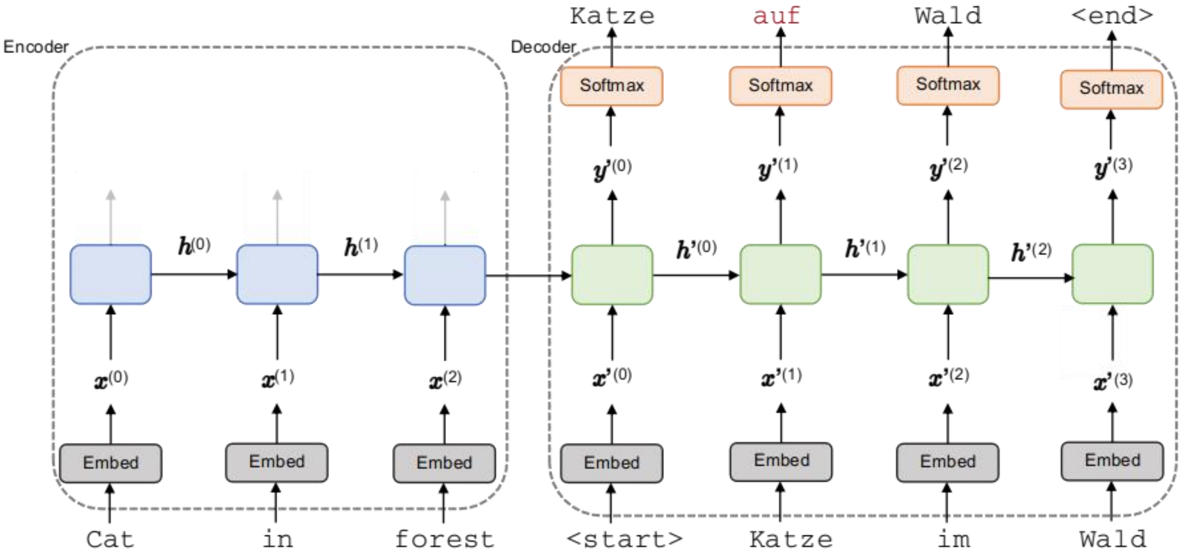


Training Encoder-Decoder for Sequences

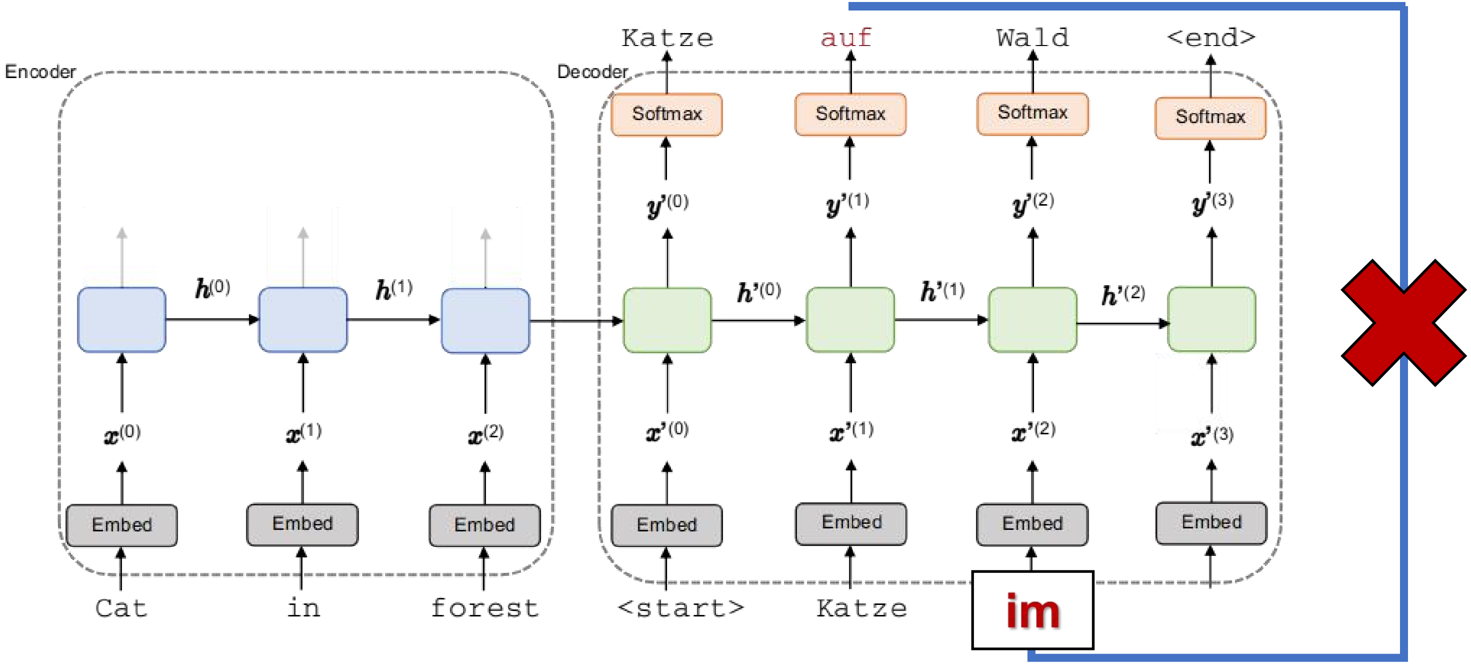


Outputs from Probabilities

- Greedy search: take the token/word with the highest probability
- Beam search: select the best k sequences so far based on joint probabilities
  - k is called the beam width (hyperparameter)

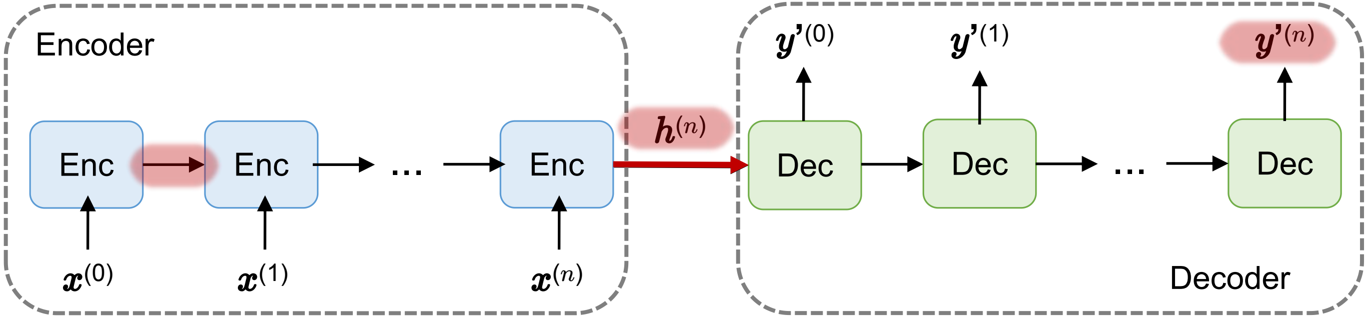


- Teacher forcing (training):
  - Use the actual input as ground truth and **not** the model output of earlier step



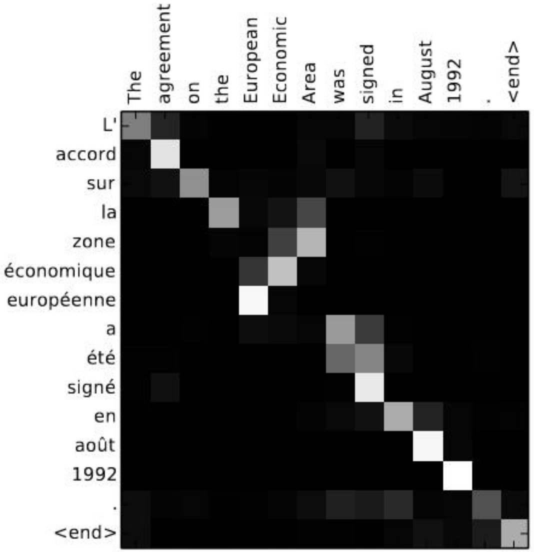
Long Paths & Attention

- *What if the decoder could **focus** on the **most important words** at each time step?*
  - Seminal paper introducing attention (called Bahdanau attention)
    - Bahdanau, Dzmitry, Kyung Hyun Cho, and Yoshua Bengio. "*Neural machine translation by jointly learning to align and translate*." In 3rd International Conference on Learning Representations, ICLR 2015.
  - Note: there are multiple types of attention (e.g., Dot-product attention, multiplicative attention, self-attention, visual attention etc.)

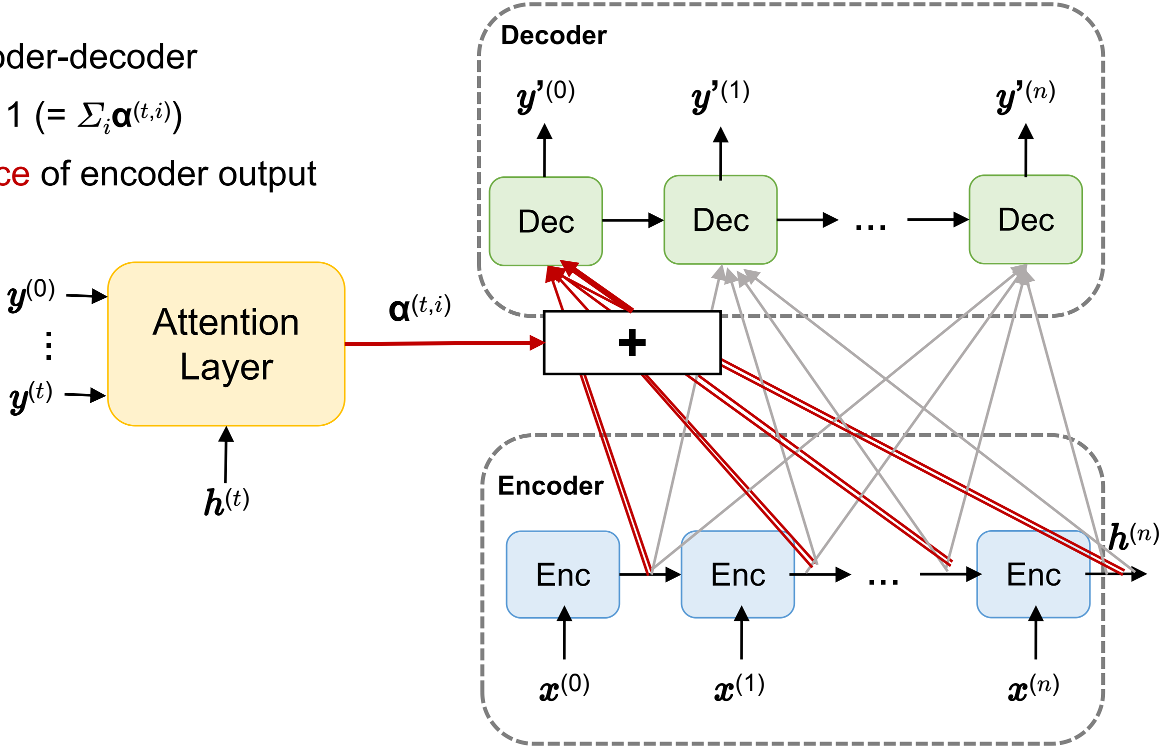


Attention

- **Attention** "network" (or layer)
  - Trained at the same time at the encoder-decoder
  - Produces weights $\boldsymbol{\alpha}^{(t,i)}$ which sum to 1 ($= \Sigma_i \boldsymbol{\alpha}^{(t,i)}$)
  - Weight $\boldsymbol{\alpha}^{(t,i)}$ represents the importance of encoder output



Source: Bahdanau et al. ICLR 2015.



Next Time

- Friday (3/29): Exercise

- Upcoming:
  - Project Proposals due 3/27
  - Homework 4 due 4/3