Generative AI

Class 2: Course Overview & Foundations of Deep Learning

Department of CSE
B.Tech, 7th Semester

July 23, 2026

Acknowledgement

Acknowledgement

Some of the material (figures, concepts, and selected slides) presented in this lecture has been adapted from the course materials developed by

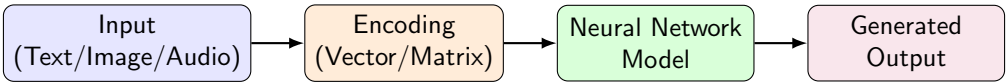
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

The original course materials remain the intellectual property of the respective author and are gratefully acknowledged.

Recap: Why Generative AI?

- Generative AI systems take an **input** — text, image, audio, or a combination — and produce a **new output** in the same or a different modality.
- Examples:
  - Text → Text (ChatGPT-style assistants)
  - Text → Image (DALL-E, Stable Diffusion)
  - Image → Text (captioning, VQA)
  - Audio → Text (speech recognition)
  - Text → Audio (speech synthesis)
- To understand *how* this happens, we need to understand **how machines represent and process each modality** — this is exactly what NLP and Computer Vision give us.

Course Roadmap: The Big Picture



- Every topic in this course fits somewhere in this pipeline.
- Today: what each box means, and why NN is the engine that makes it work.

Topics We Will Cover

1. **Encoding inputs**: representing text, images, and audio as vectors/matrices
   - Text: tokenization, word embeddings, positional encoding
   - Image: pixels, patches, CNN features
   - Audio: waveforms, spectrograms
2. **Feeding into models**: how encoded inputs pass through a network
3. **Generating outputs**: decoding vectors back into text/image/audio
4. **What makes generation "good"**: human-likeness, factual correctness, safety, alignment
5. **Core architectures**: Neural Networks, RNNs, CNNs (foundations)
6. **Modern generative architectures**: Transformers, Autoencoders/VAEs, GANs, Diffusion models (later modules)

Part 1

Why Neural Networks?

Why Not Just Write Rules?

- Language, vision, and sound are extremely **high-dimensional** and **ambiguous**.
- Hand-crafted rules cannot scale to capture all patterns (grammar exceptions, lighting variations, accents, etc.).
- We instead want a system that **learns** the mapping from input to output directly from data.
- Neural Networks are **universal function approximators**: given enough data and parameters, they can approximate very complex input→output mappings.
- This is why NN became the default engine for generative modeling.

What Is a Neural Network, Really?

- Strip away the hype: a neural network is just
  - repeated **matrix multiplications**, and
  - **non-linear activation functions** applied in between.
- A single layer:

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

  where
  - $\mathbf{x}$ = input vector
  - $\mathbf{W}$ = weight matrix (learned)
  - $\mathbf{b}$ = bias vector
  - f = activation function (ReLU, sigmoid, tanh, ...)
- Stack many such layers $\rightarrow$ a *deep* neural network.

The Core Question

If NN is just matrix multiplication...

- Matrix multiplication needs **numbers** — specifically, vectors and matrices.
- But our raw input is a **sentence**, an **image**, or a **sound wave**.
- So the real question becomes:

How do we turn text / image / audio into a matrix?

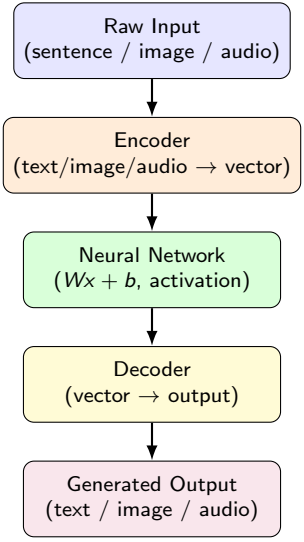
Encoding: Turning Everything Into Numbers

- **Text** $\rightarrow$ tokens $\rightarrow$ embedding vectors (each word/subword becomes a row in a matrix)
- **Image** $\rightarrow$ pixel grid is already numeric (height $\times$ width $\times$ channels), can also be split into patches
- **Audio** $\rightarrow$ waveform samples, or converted into a spectrogram (a 2D matrix of frequency vs. time)
- In every case, the goal is the same:

$$\textbf{raw modality} \longrightarrow \textbf{a matrix/tensor the network can multiply with its weights}$$

(We will study each of these encodings in detail in upcoming classes.)

The Full Pipeline



Part 2

What Do We Actually Want From the Output?

Generating Output Is Not Enough — It Must Be Good

- Once a model can generate *something*, the harder question is: generate *what kind* of output?
- Four key requirements we will study throughout the course:

1. **Human-like generation** — fluent, coherent, natural
2. **Factually correct generation** — grounded in truth, not hallucinated
3. **Safe generation** — avoids harmful, biased, or toxic content
4. **User alignment** — matches user intent and instructions

These Requirements Can Conflict

- A model can be very **fluent** but **factually wrong** (hallucination).
- A model can be **factually correct** but not **aligned** with what the user actually wanted.
- Making a model **safe** sometimes reduces how **helpful** or **creative** it appears.
- Balancing these is an active research problem — this is where techniques like RLHF, instruction tuning, and guardrails come in (later in the course).

Part 3

Preview: Architectures We Will Build On

Neural Network (NN) — Today's Foundation

- Also called a **Feedforward Neural Network** / Multi-Layer Perceptron (MLP).
- Input layer $\rightarrow$ one or more hidden layers $\rightarrow$ output layer.
- Every connection is a learned weight; every layer applies $f(Wx + b)$.
- Good for fixed-size vector inputs, but has no notion of **sequence** or **spatial structure**.
- This limitation motivates RNNs and CNNs.

RNN and CNN — A Quick Preview

RNN (Recurrent Neural Network)

- Designed for **sequential data** (text, audio, time series)
- Maintains a **hidden state** that carries information across time steps
- Useful when order matters

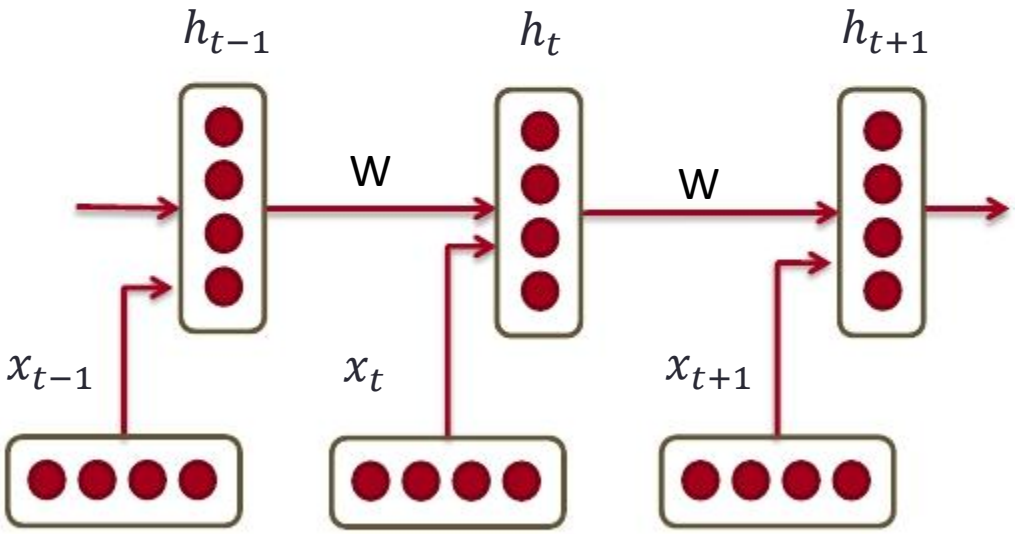
CNN (Convolutional Neural Network)

- Designed for **grid-like data** (images, spectrograms)
- Uses **filters/kernels** to detect local spatial patterns
- Shares weights across the input $\rightarrow$ efficient

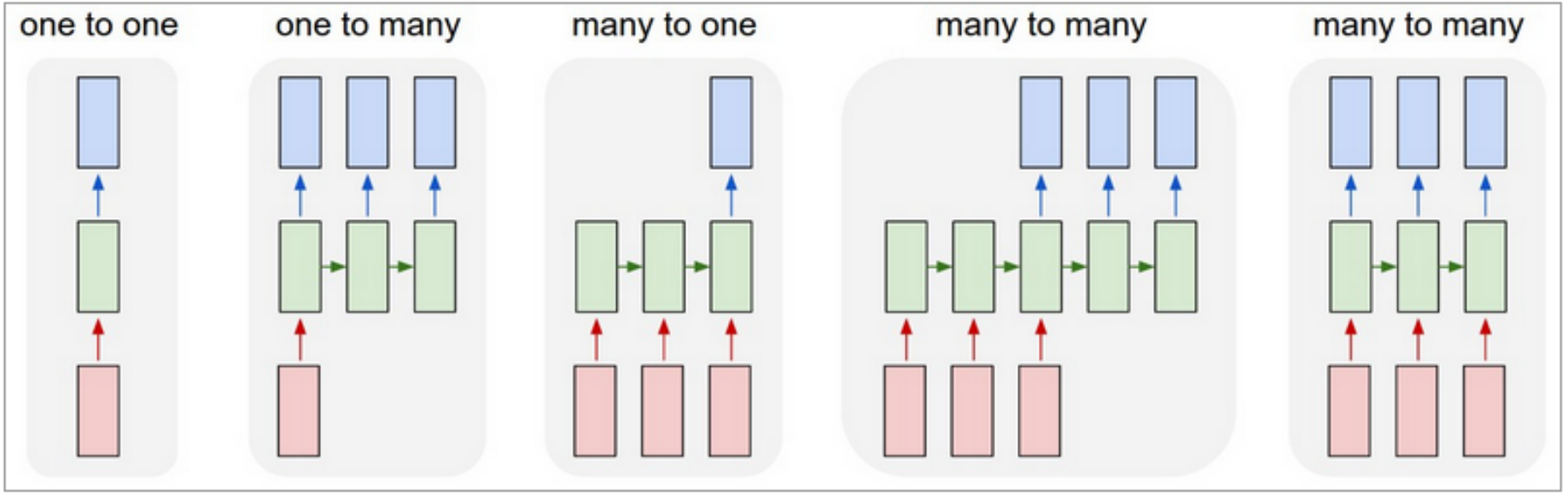
We will cover both in detail, with full derivations, in upcoming classes.

Recurrent Neural Networks

- Condition the neural network on all previous words
- tie the weights at each time step



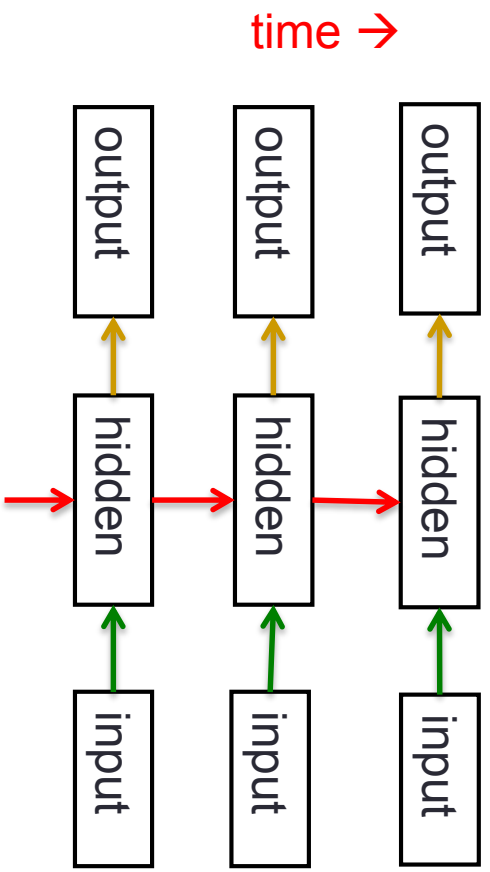
Types of RNN



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Recurrent neural networks

- RNNs combine two properties:
  - Distributed hidden state allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.



Recurrent Neural Network language model

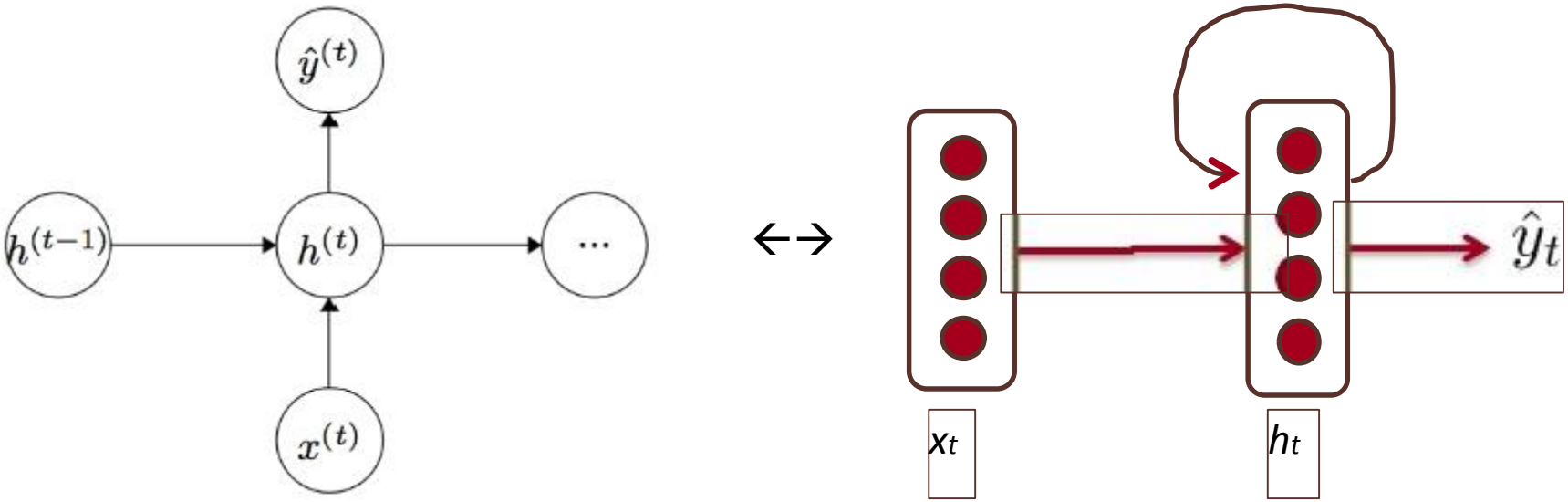
Given list of word **vectors**: $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$

At a single time step:

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

$$\hat{y}_t = \text{softmax}\left(W^{(S)} h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$



Sequence modelling for NLP tasks

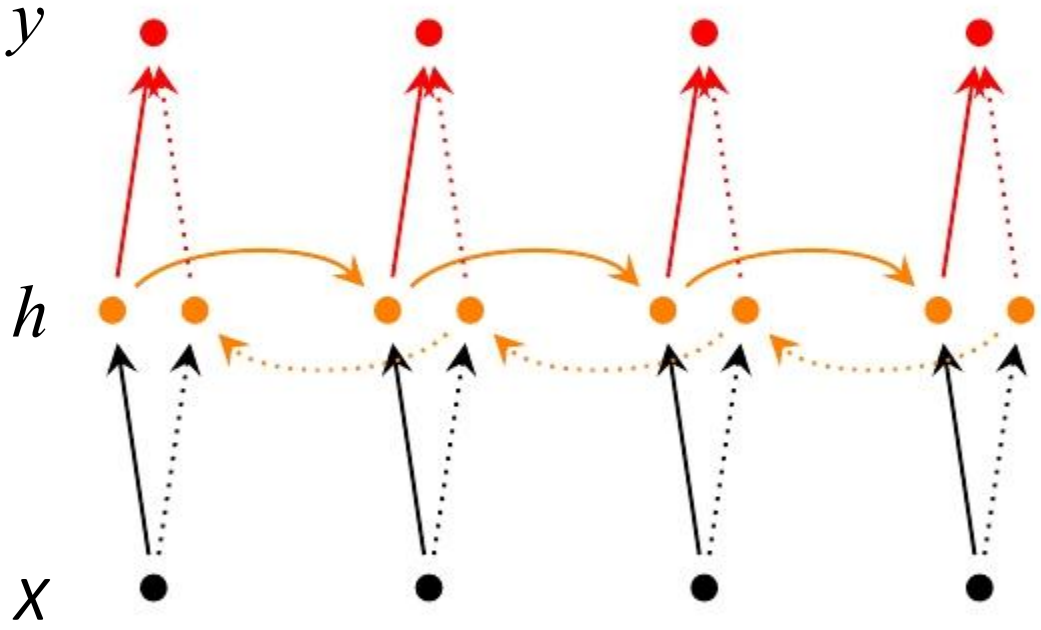
- Language model
- Classification
  - NER
  - the sentiment of each word in its context
  - opinionated expressions

Opinion Mining with Deep Recurrent Nets by Irsoy and Cardie 2014

Bidirectional RNNs

For some tasks you want to incorporate

information from words both preceding and following



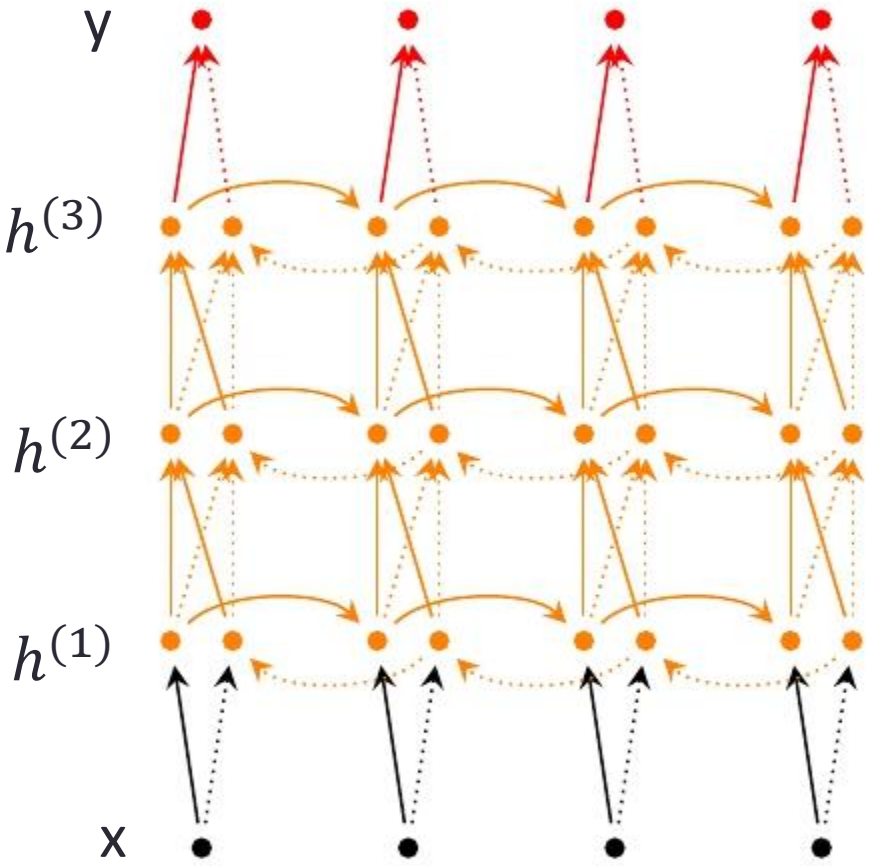
$$\overrightarrow{h_t} = f(\overrightarrow{W x_t} + \overrightarrow{V h_{t-1}} + \vec{b})$$

$$\overleftarrow{h_t} = f(\overleftarrow{W x_t} + \overleftarrow{V h_{t-1}} + \overleftarrow{b}$$

$$y_t = g\left(U\left[\overrightarrow{h_t}; \overleftarrow{h_t}\right] + c\right)$$

$h = \left[\vec{h}; \overleftarrow{h}\right]$ now represents (summarizes) the past and future around a single token.

Deep Bidirectional RNNs



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

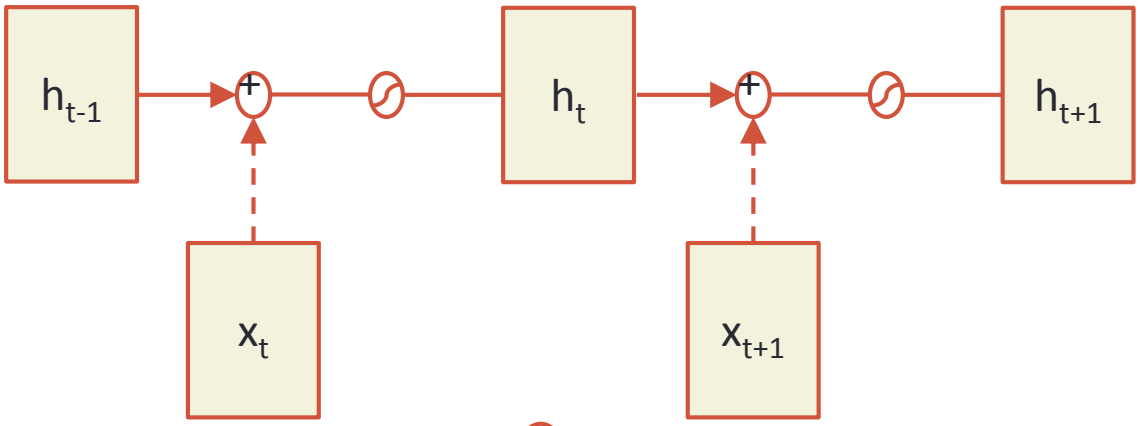
$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Each memory layer passes an intermediate sequential representation to the next.

Complex units: GRU and LSTM

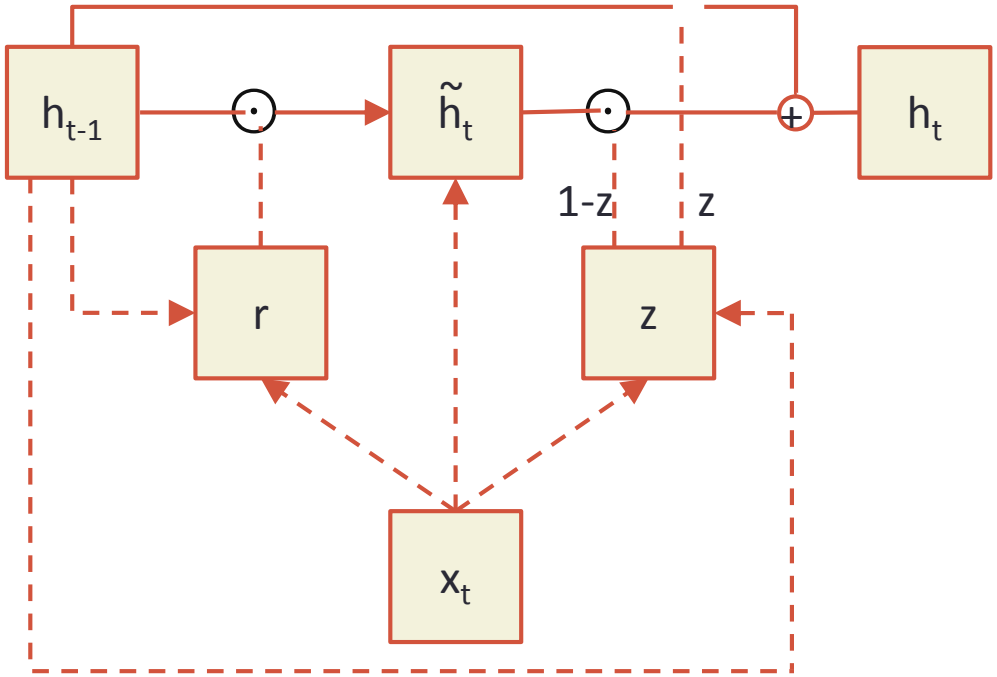
- Make the RNN out of modules that are designed to remember values for a long time
- LSTM: Hochreiter & Schmidhuber (1997)
- GRU: simpler unit proposed by Cho et al (2014)

Simple Recurrent Unit



GRU: Gated recurrent unit

introduced by Cho et al. 2014



- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$

Controls how much of past state should matter now.

- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$

If reset gate unit is ~0, then ignore previous memory and only stores the new word information

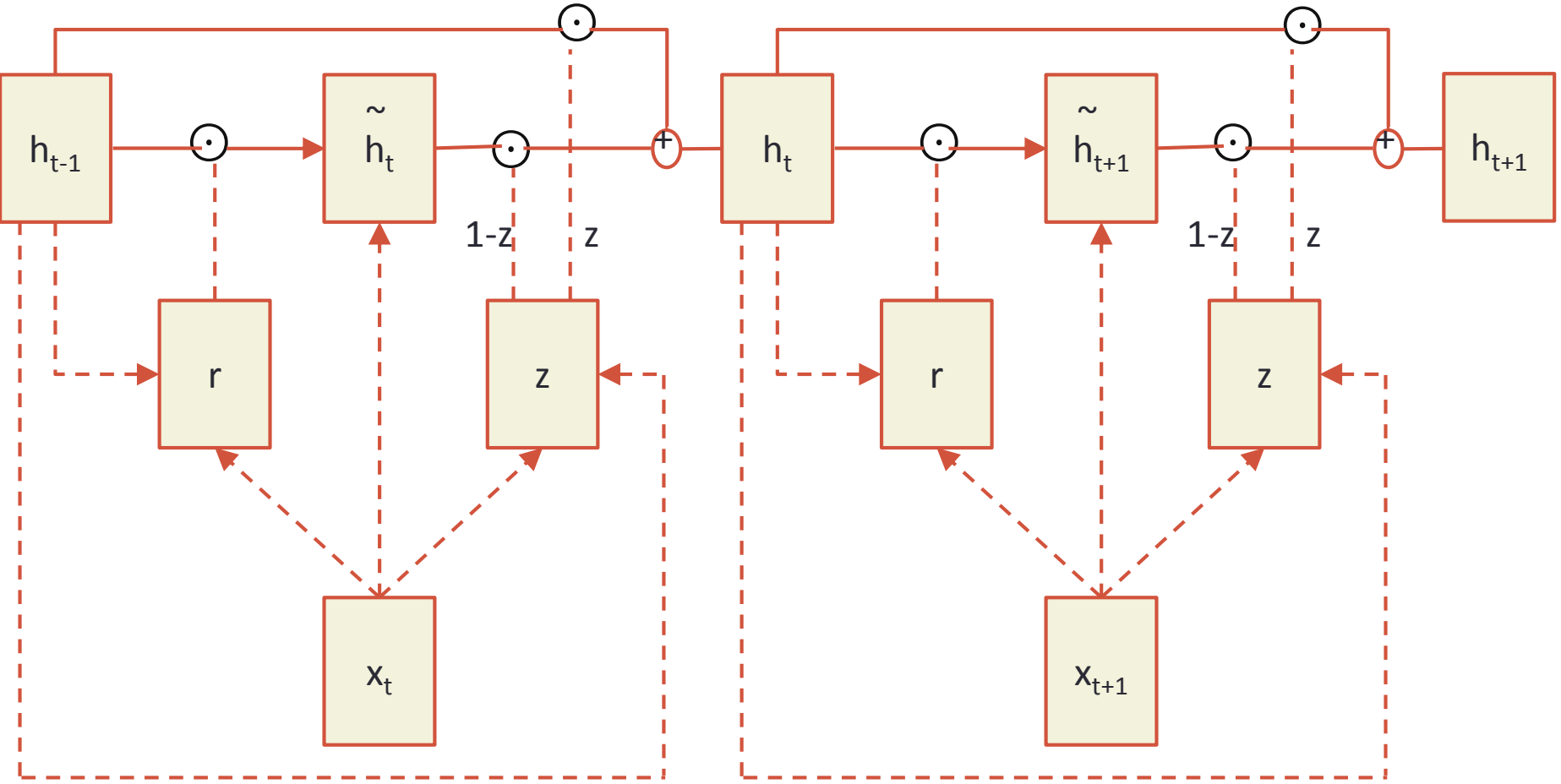
- Final memory at time step combines current and previous time steps:

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

Main ideas:

- keep around memories to capture long distance dependencies
- allow error messages to flow at different strengths depending on the inputs

Gated Recurrent Unit - GRU



Long-short-term-memories (LSTMs)

- We can make the units even more complex
- Allow each time step to modify
  - Input gate (current cell matters)
  - Forget (gate 0, forget past)
  - Output (how much cell is exposed)
  - New memory cell

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$

$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$

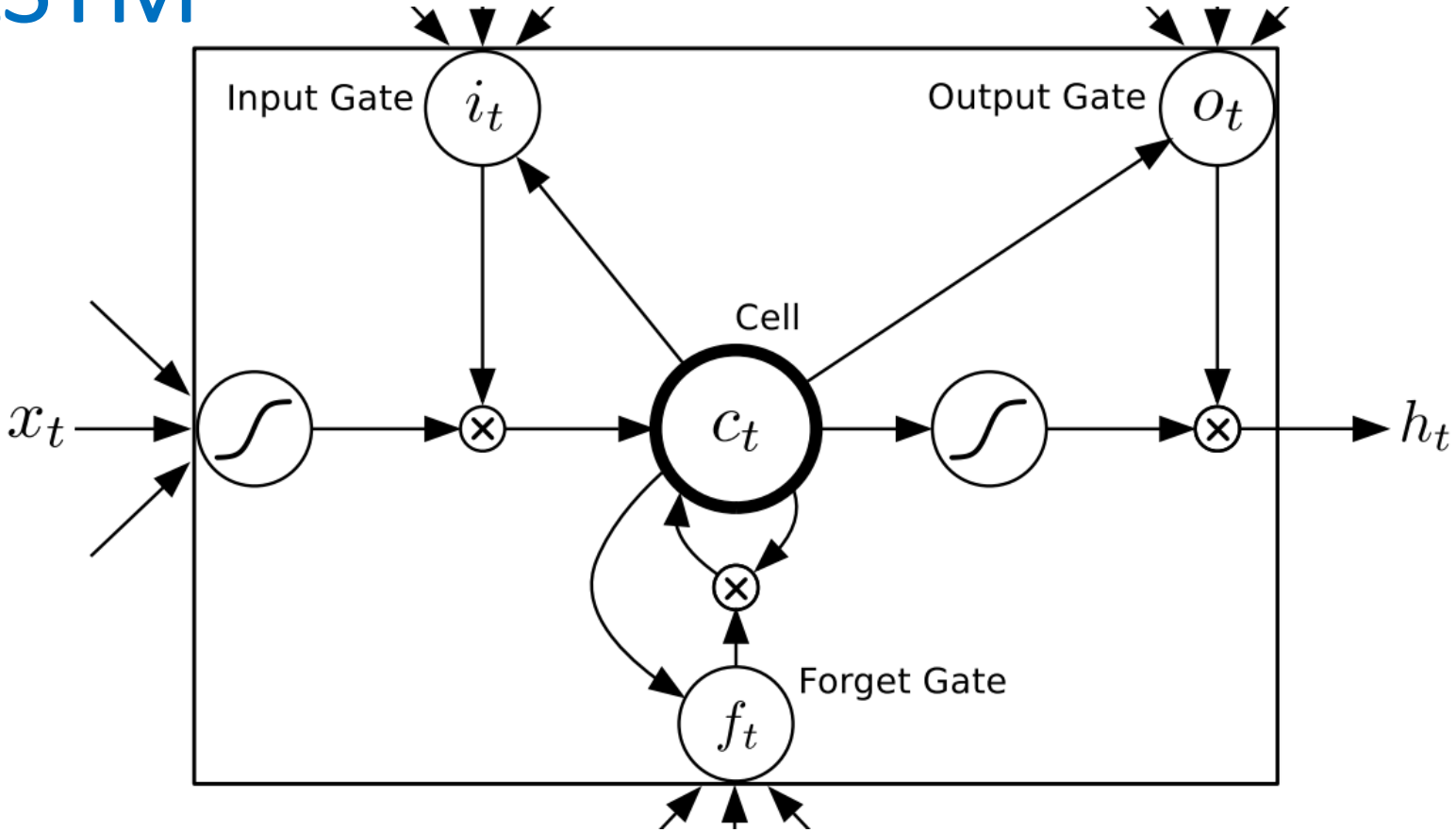
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$

Final memory cell: $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$

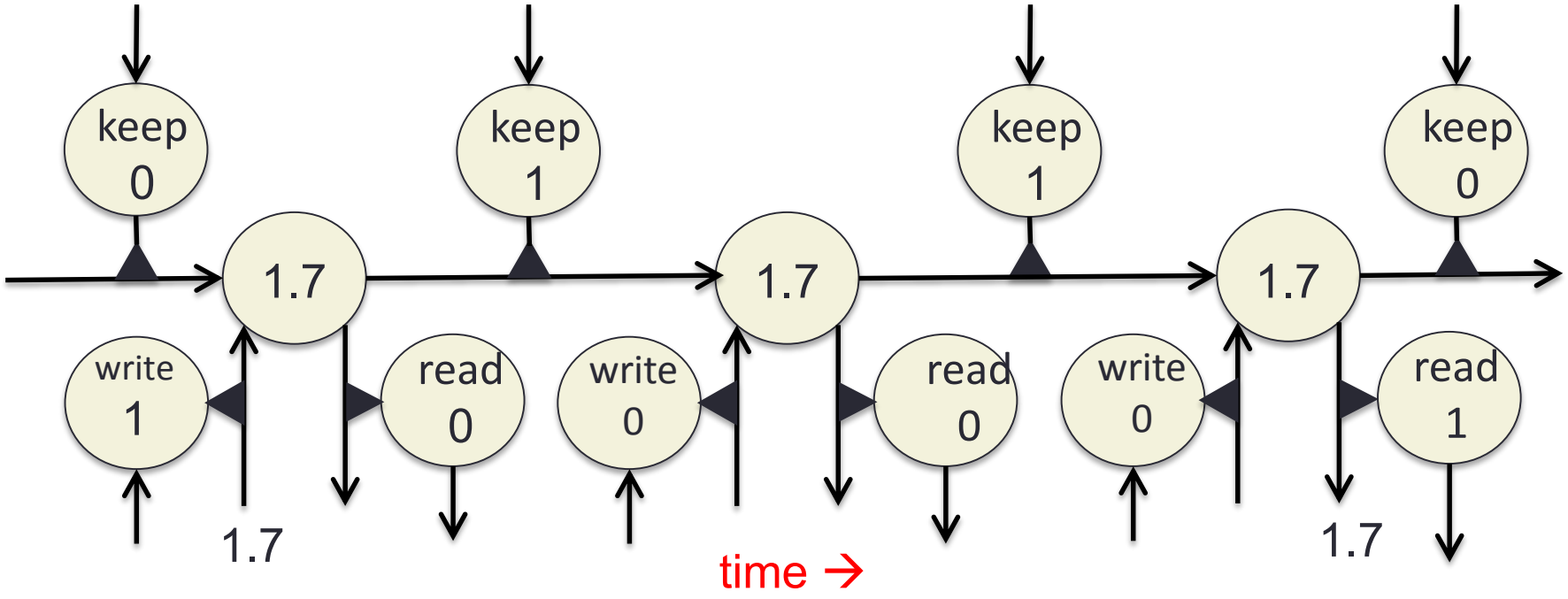
Final hidden state: $h_t = o_t \circ \tanh(c_t)$

LSTM



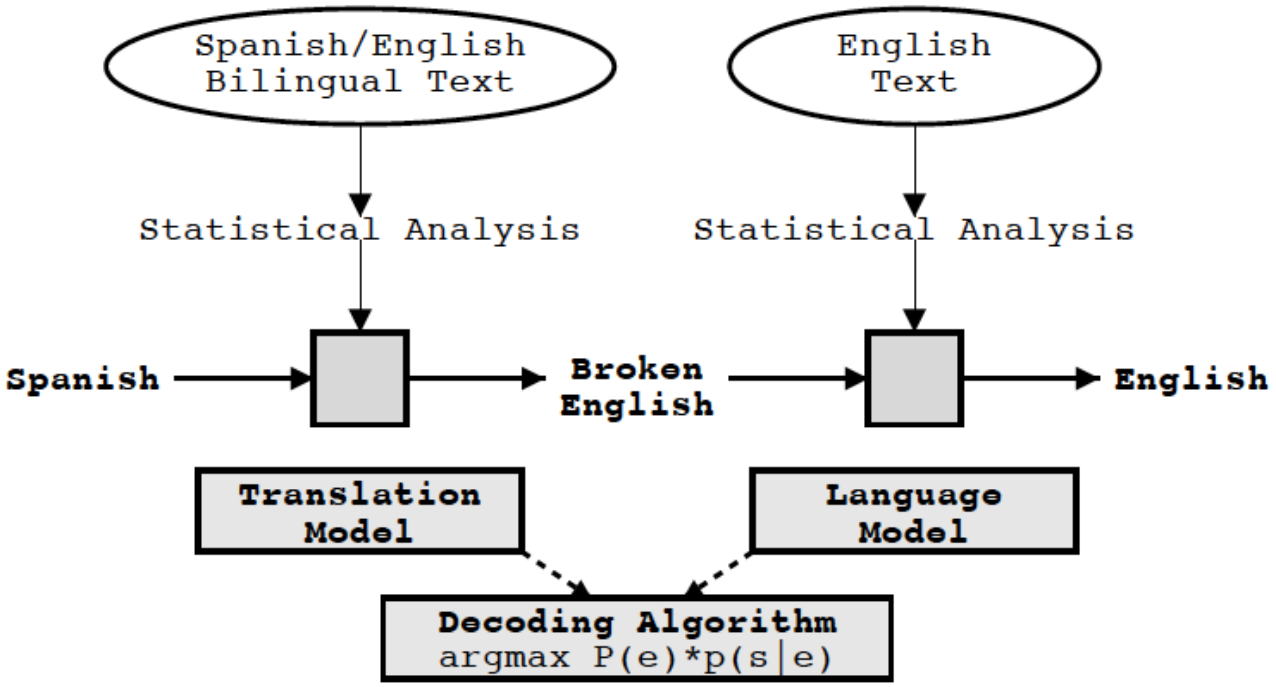
- Intuition: memory cells can keep information intact, unless inputs makes them forget it or overwrite it with new input.
- Cell can decide to output this information or just store it

Backpropagation through a memory cell



MACHINE TRANSLATION

Statistical Machine Translation

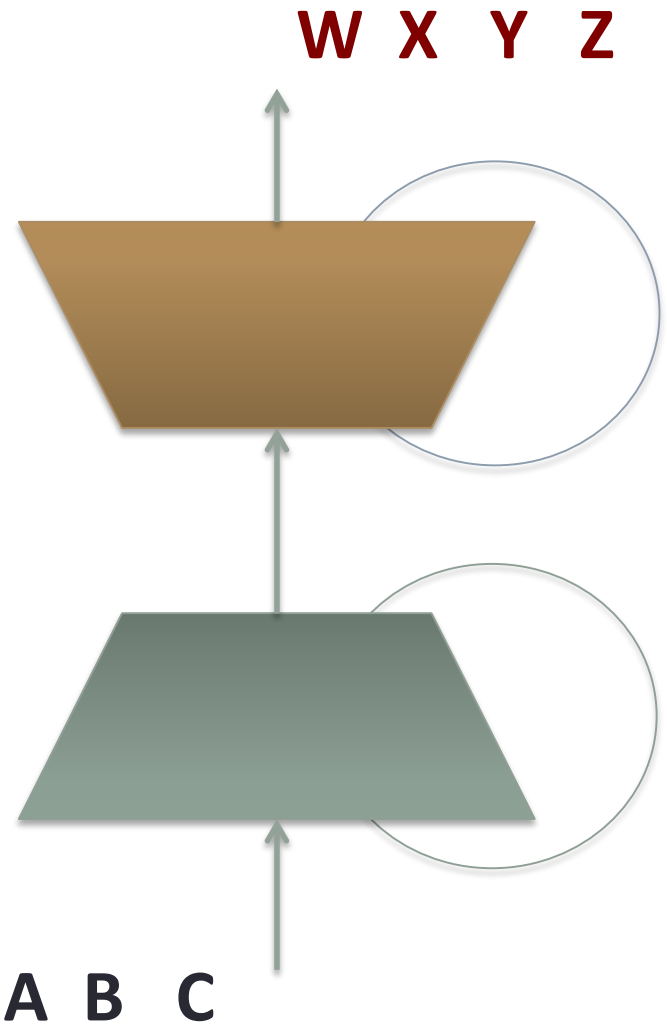


Components

- Translation model
- Language Model
- Decoding

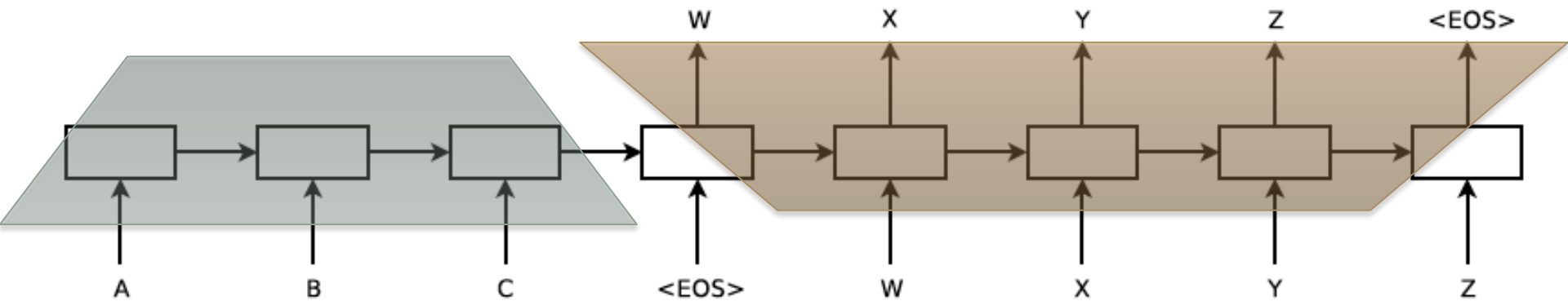
Neural Machine Translation

Model



Neural Machine Translation

Model



Sutskever et al. 2014

MT with RNNs – Simplest Model

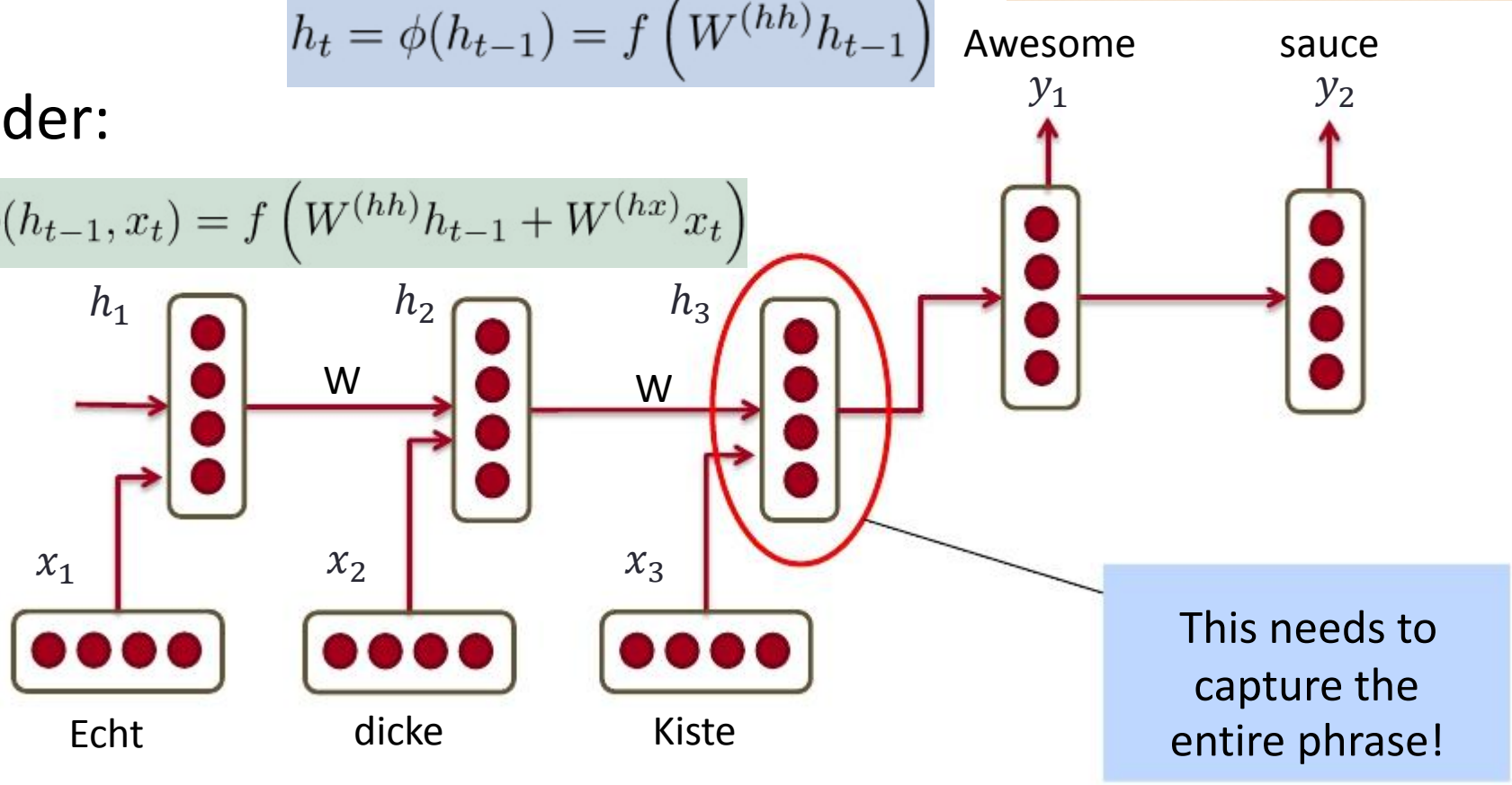
Decoder:

$$h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$$

$$y_t = softmax\left(W^{(S)} h_t\right)$$

Encoder:

$$h_t = \phi(h_{t-1}, x_t) = f\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$$



MT with RNNs – Simplest Model

Encoder: $h_t = \phi(h_{t-1}, x_t) = f\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$

Decoder: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$

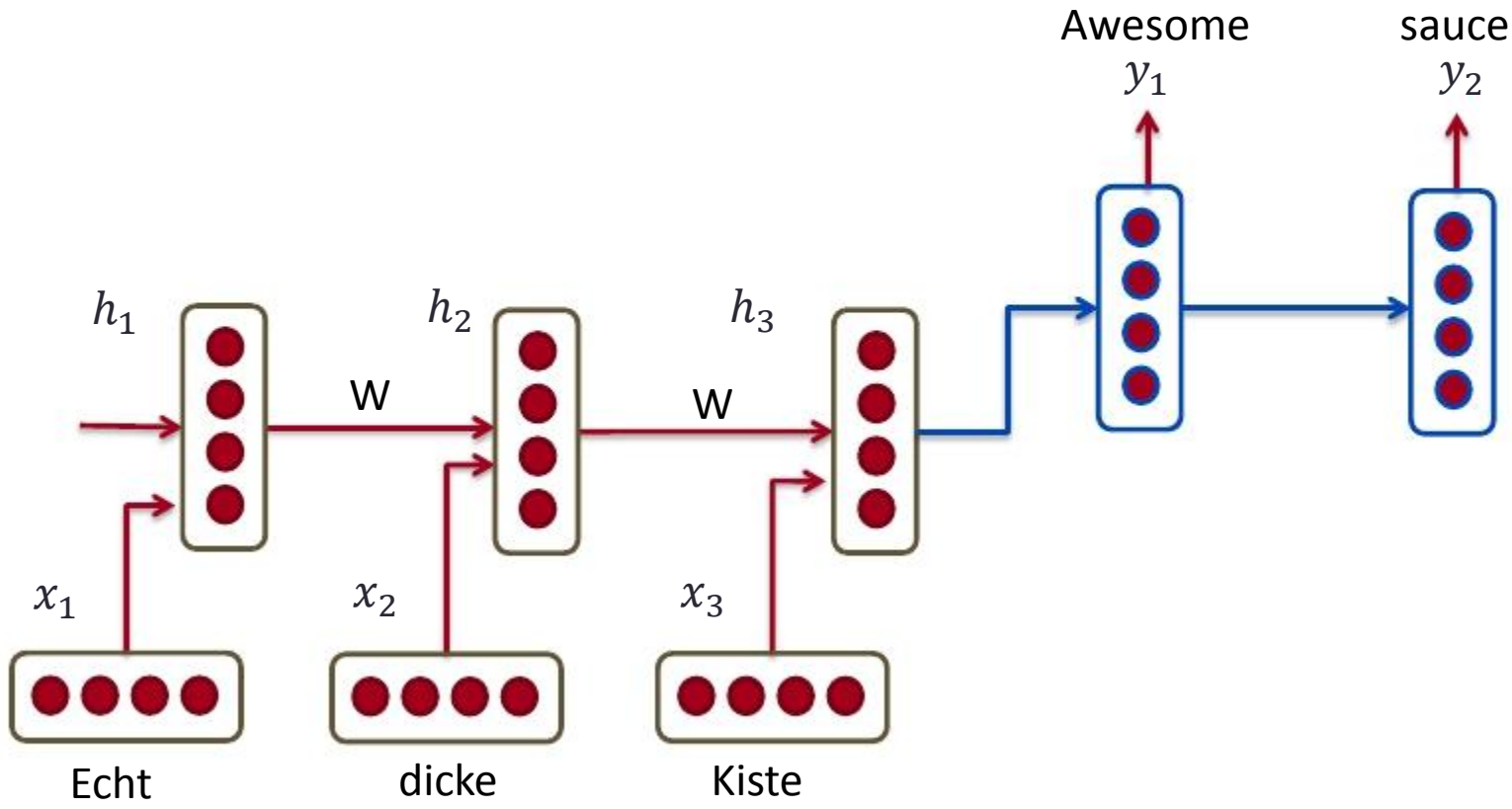
Output: $y_t = softmax\left(W^{(S)} h_t\right)$

Minimize cross entropy error for all target words conditioned on source words

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(y^{(n)}|x^{(n)})$$

RNN Translation Model Extension

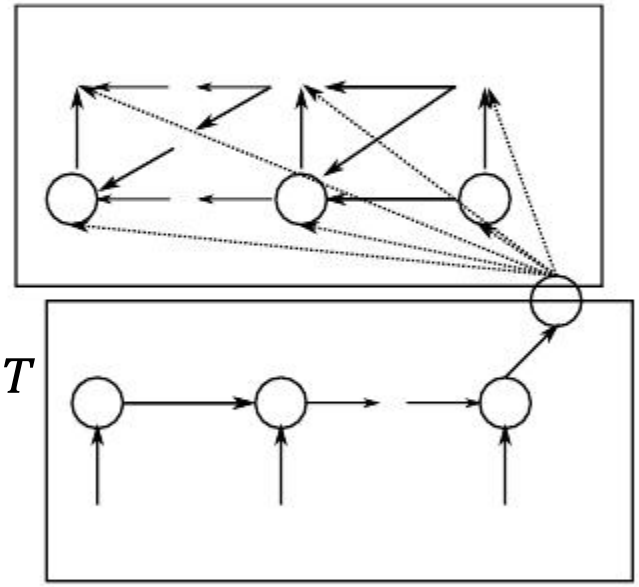
1. Train different RNN weights for encoding and decoding



RNN Translation Model Extension

Notation: Each input of ϕ has its own linear transformation matrix.

$$h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$$



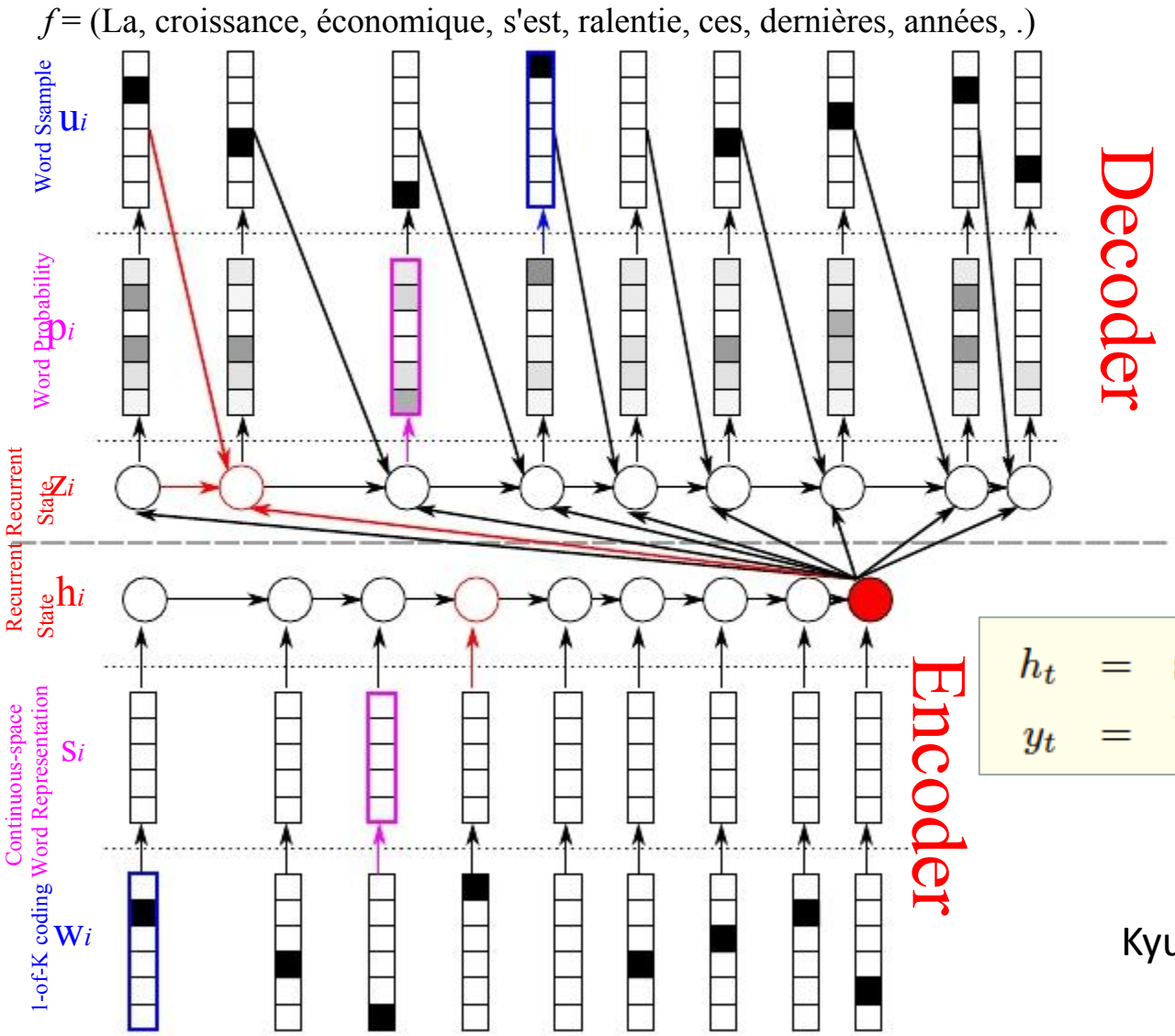
2. Compute every hidden state in decoder from

- Previous hidden state
- Last hidden Vector of encoder $c = h_T$
- Previous predicted output word y_{t-1}

$$h_{D,t} = \phi_D(h_{t-1}, c, y_{t-1})$$

Cho et al 2014

Different picture for same idea

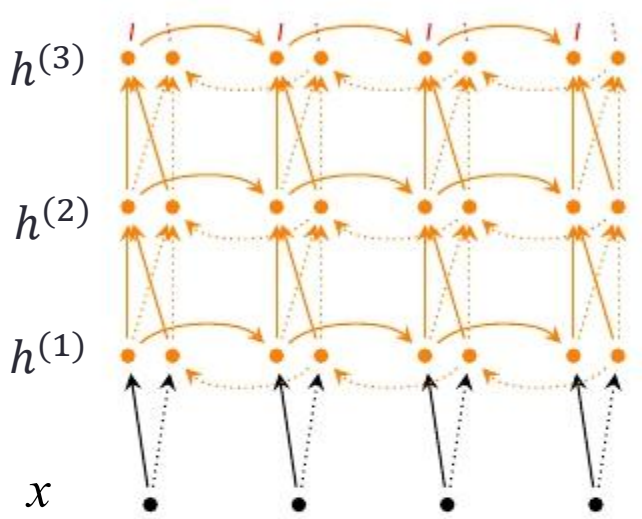


$$h_t = \text{sigm}\left(W^{\text{hx}} x_t + W^{\text{hh}} h_{t-1}\right)$$
$$y_t = W^{\text{yh}} h_t$$

Kyunghyun Cho et al. 2014

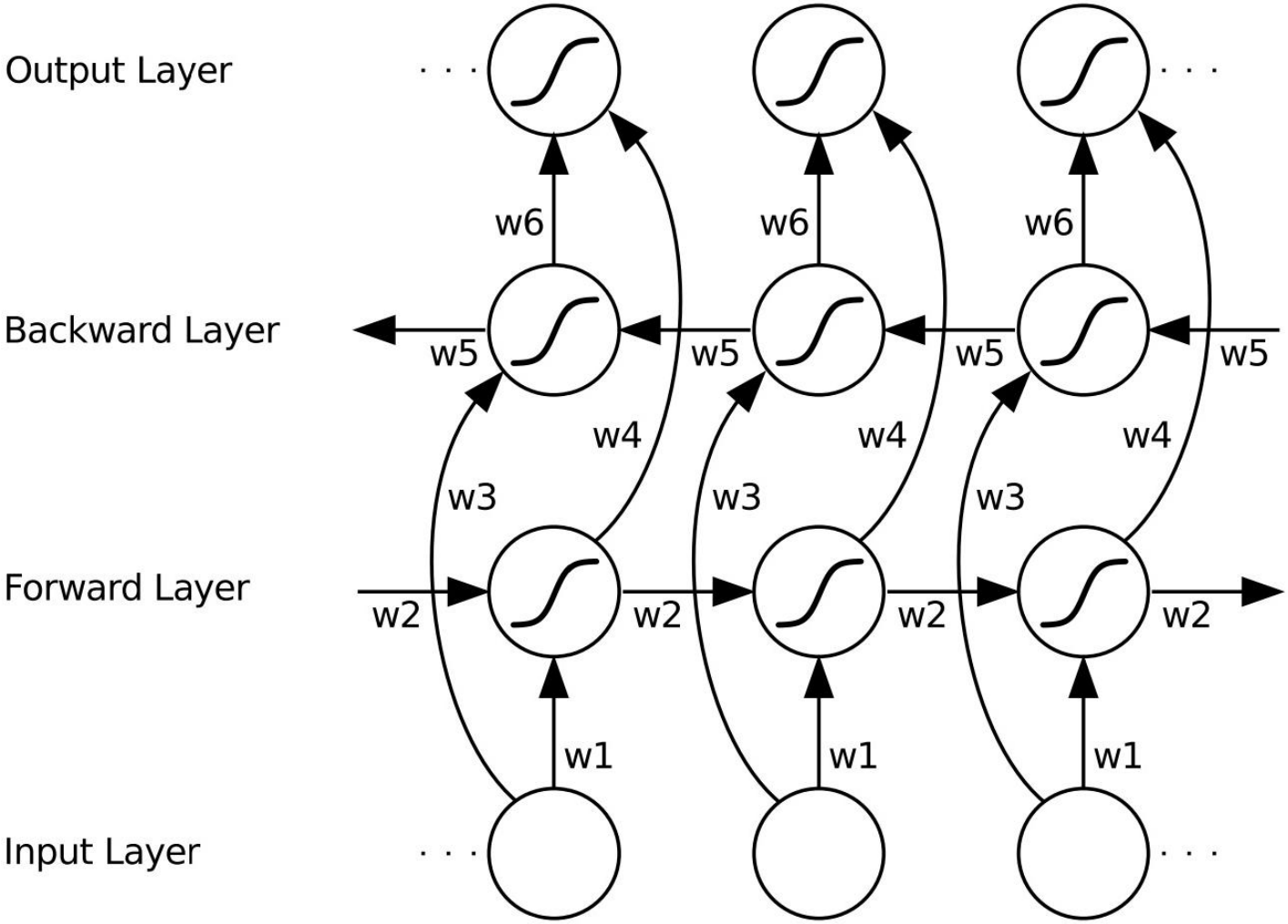
RNN Translation Model Extension

3. Train stacked/deep RNNs with multiple layers
4. Train bidirectional encoder

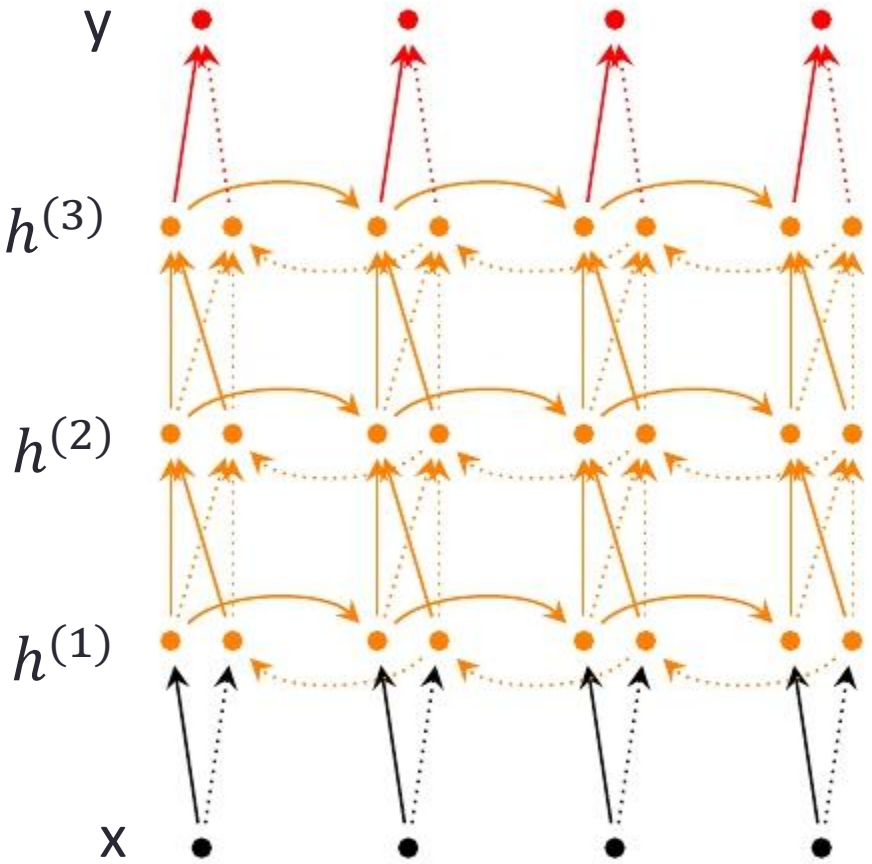


5. Train input sequence in reverse order for simple optimization problem: Instead of A B C → X Y, train with C B A → X Y

Bi-Directional RNNs



Deep Bi-Directional RNNs



Each memory layer passes an intermediate sequential representation to the next.

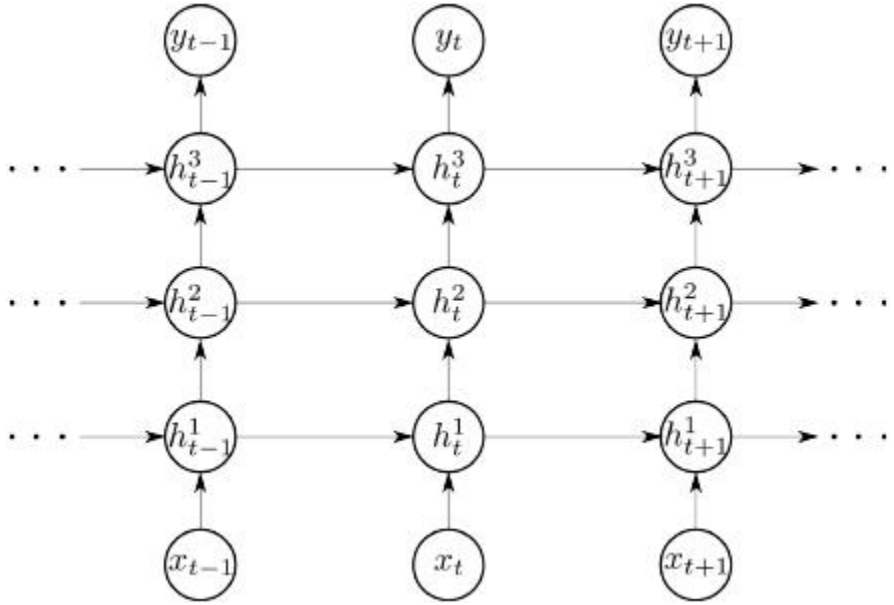
Big Improvement: Better Units

- Use GRU's or LSTM's

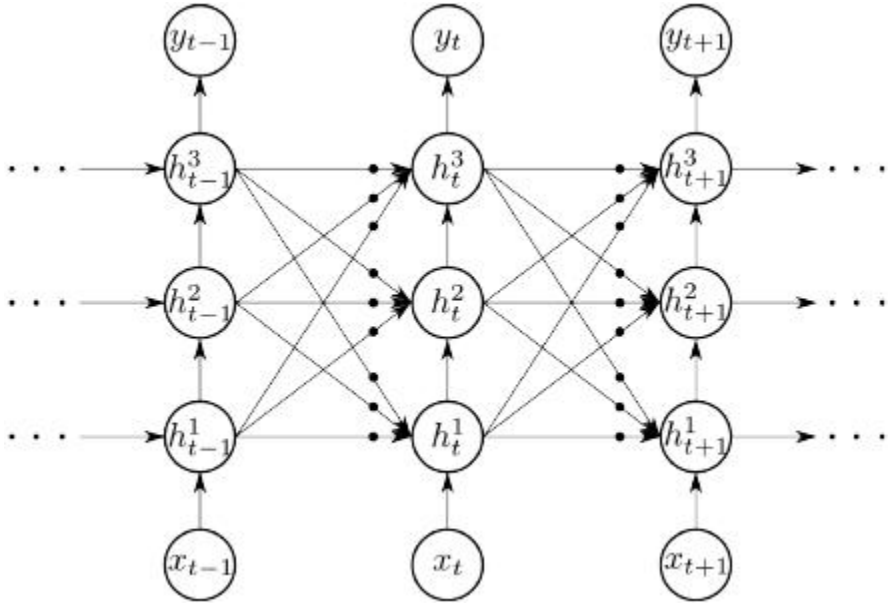
- Main ideas:
  - Keep around memories to capture long distance dependencies
  - Allow error messages to flow at different strengths depending on the inputs

Further Improvements: More Gates!

Gated Feedback Recurrent Neural Networks, Chung et al. 2015



(a) Conventional stacked RNN



(b) Gated Feedback RNN

Neural MT – unrealistic?

- This model is quite unrealistic?

"You can't cram the meaning of a whole %&!$# sentence into a single $&!# vector!"* Ray Mooney

- Learn to align and translate jointly.

Questions?