Charu C. Aggarwal
IBM T J Watson Research Center
Yorktown Heights, NY

Recurrent Neural Networks

Neural Networks and Deep Learning, Springer, 2018
Chapter 7.1–7.2

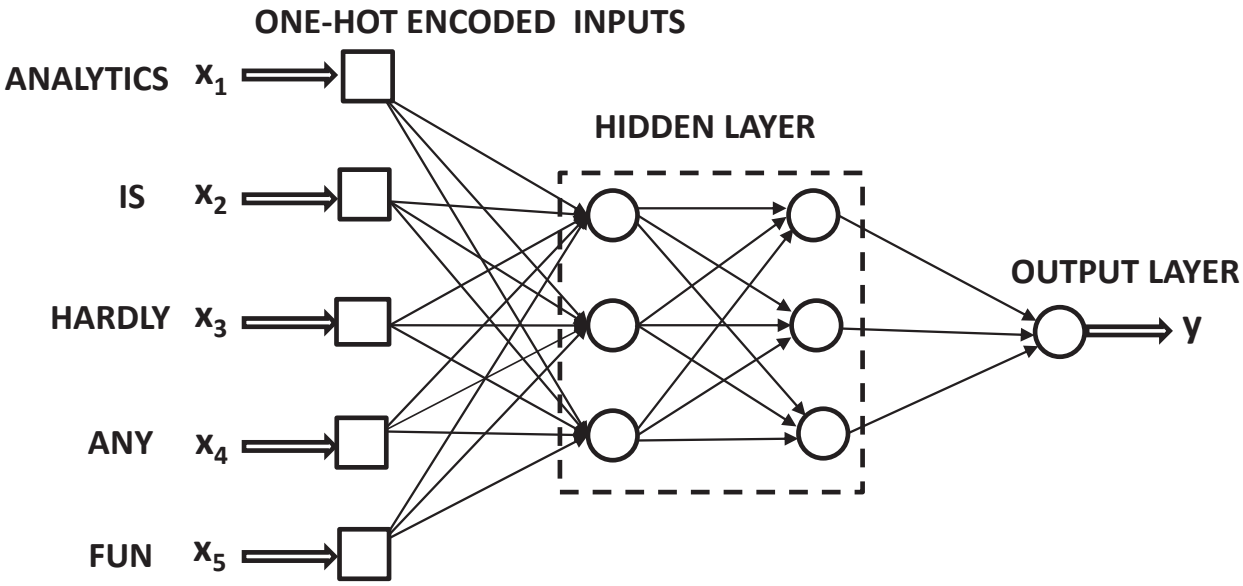
The Challenges of Processing Sequences

- Conventional neural networks have a fixed number of (possibly independent) input dimensions and outputs.

- Sequences often have variable length in which the different items (dimensions) of the sequence are related to one another:
  - Words in a sentence.
  - Values in a time-series
  - Biological sequences

- Recurrent neural networks address such domains.

Problems with Conventional Architecture [Sentiment Analysis]

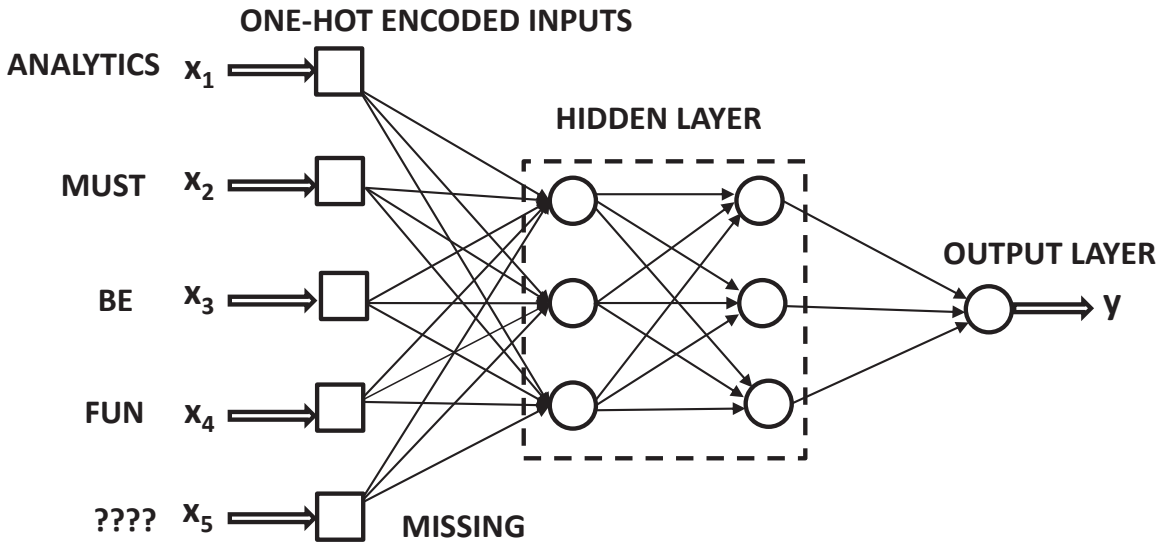


- The architecture above cannot handle sequences of length larger than 5.

- The sequential relationship among input elements is not encoded well.

  – Distinguish "*cat chased mouse*" and "*mouse chased cat*"

Problems with Conventional Architecture

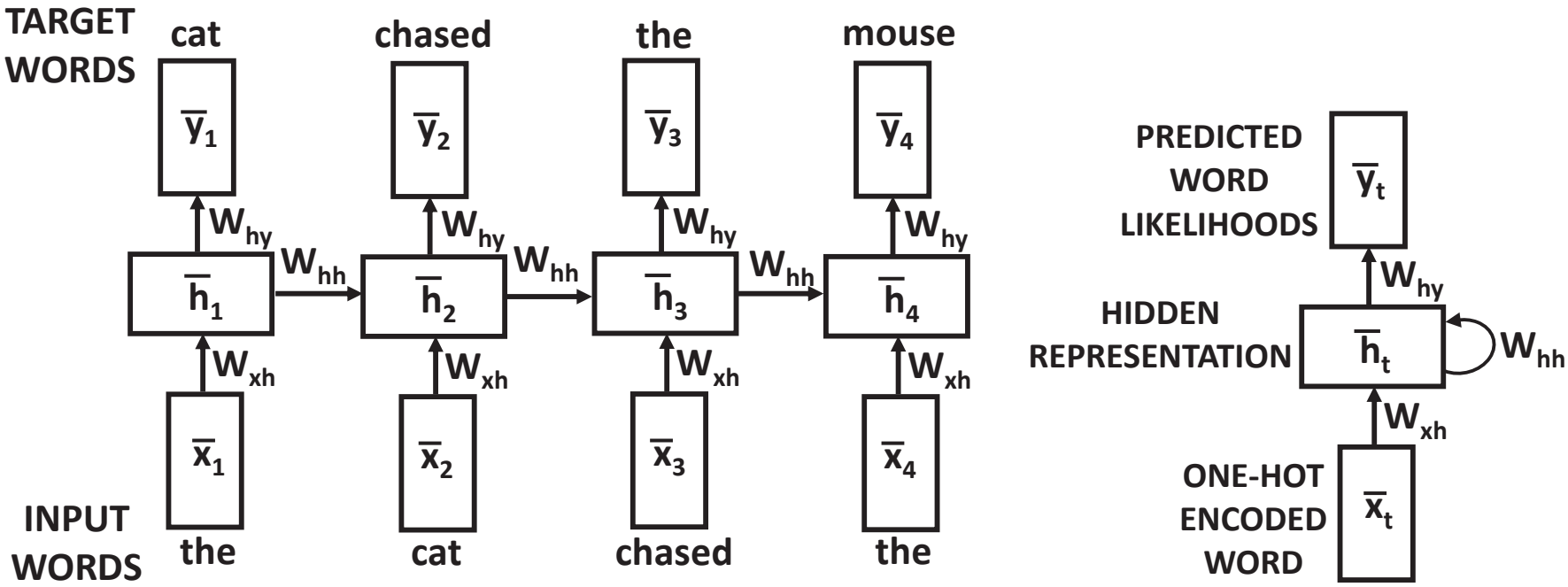


- Small changes in word ordering change sentiment.
- Missing inputs if sequence is too small.
- How do you create an architecture with variable number of inputs but fixed number of parameters?
  - Machine translation has variable number of *outputs*!

Desiderata for Architecture Processing Sequences

- The ith element of the sequence should be fed into the neural network after the network has had a chance to see what has come before it.
  - Can be enforced by feeding ith element of sequence into layer i.
  - *Each layer associated with a time-stamp* (variable number of layers).
- **Markov Property:** Any element of the sequence can be predicted from its preceding elements using a fixed model.
  - Can be (approximately) enforced by using shared weights across layers.
  - Number of weight parameters independent of number of layers!

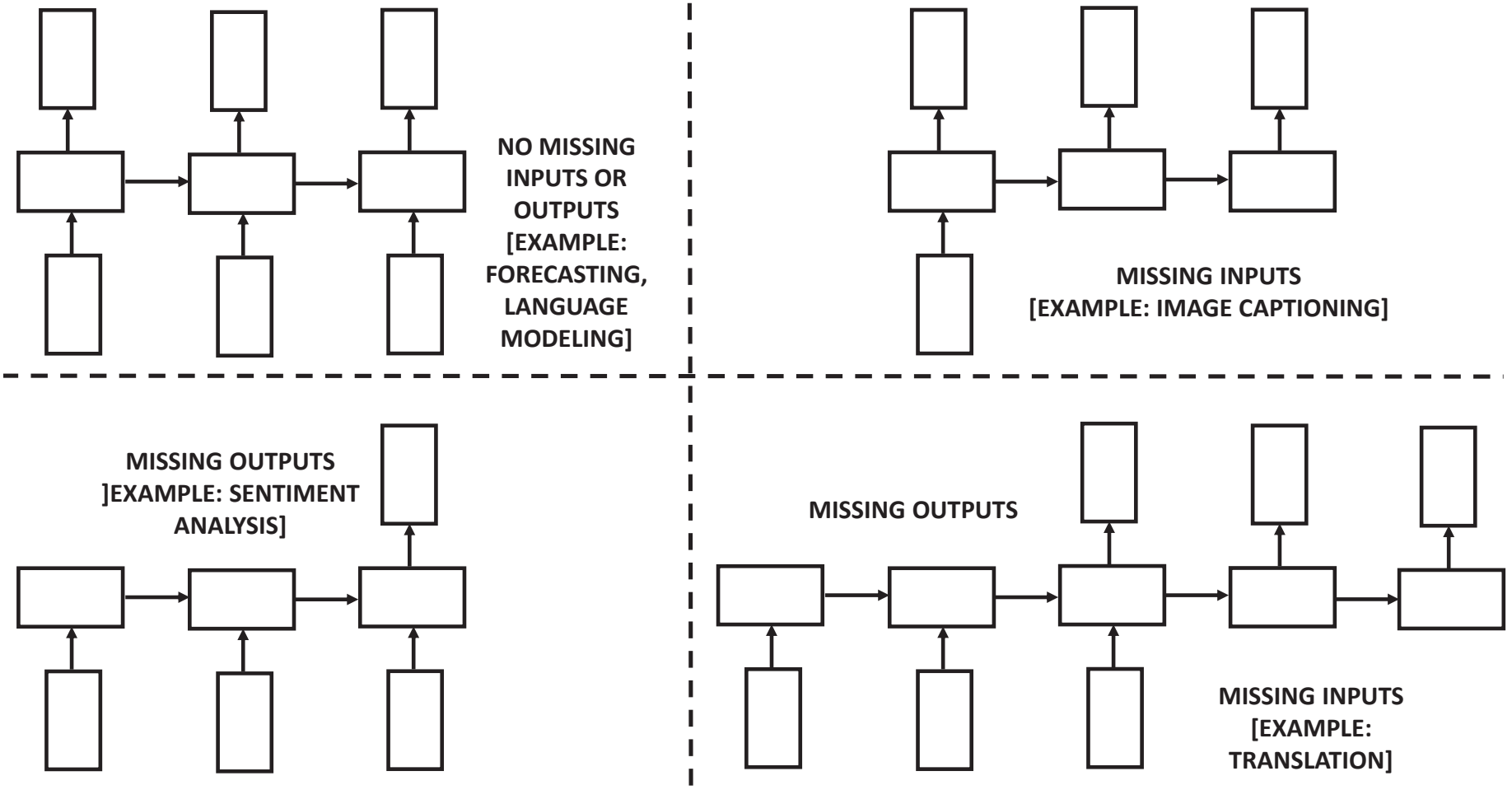
A Time-Layered Recurrent Network [Predicting Next Word]



- Note that the weight matrices W_{xh}, W_{hh}, and W_{hy} are shared across layers (vector architecture).

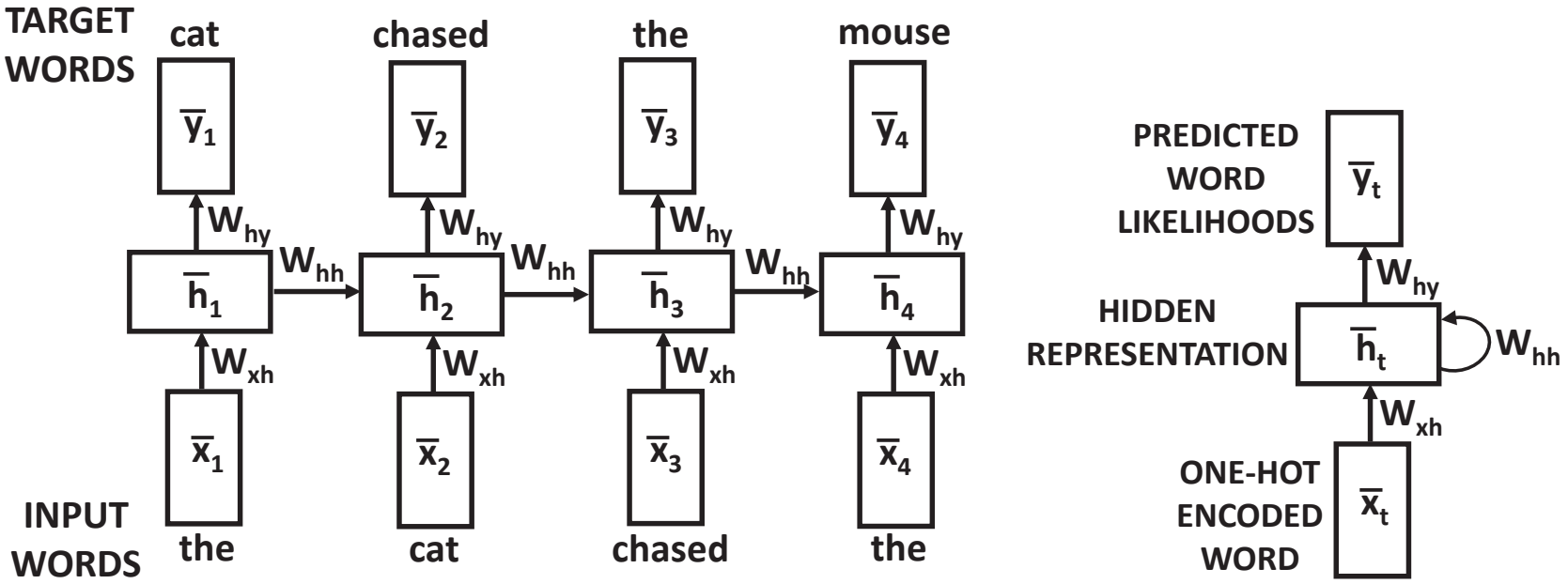
- An input $\overline{x}_t$ directly encounters the hidden state constructed from all inputs before it.

Variations



- One does not need to have inputs and outputs at each time stamp!

Recurrent Network: Basic Computations



- $\overline{h}_t = f(\overline{h}_{t-1}, \overline{x}_t) = \tanh(W_{xh}\overline{x}_t + W_{hh}\overline{h}_{t-1})$ [Typical hidden-to-hidden]

- $\overline{y}_t = W_{hy}\overline{h}_t$ [Real-valued hidden-to-output $\Rightarrow$ Optional and depends on output layer]

Flexibility for Variable-Length Inputs

- We define the function for $\overline{h}_t$ in terms of t inputs.

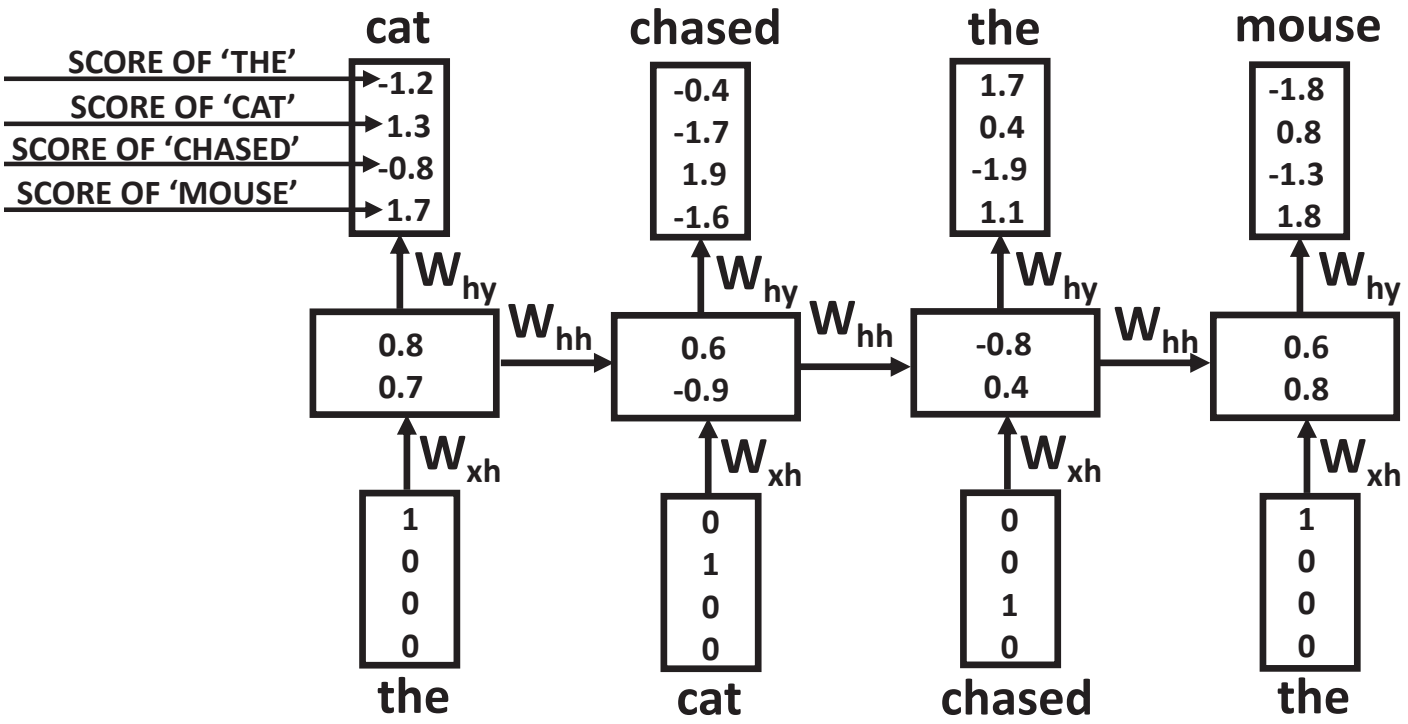
- We have $\overline{h}_1 = f(\overline{h}_0, \overline{x}_1)$ and $\overline{h}_2 = f(f(\overline{h}_0, \overline{x}_1), \overline{x}_2)$.

  - The vector $\overline{h}_1$ is a function of only $\overline{x}_1$, whereas $\overline{h}_2$ is a function of both $\overline{x}_1$ and $\overline{x}_2$.

- The vector $\overline{h}_t$ is a function of $\overline{x}_1 \ldots \overline{x}_t$.

  - Can provide an output based on entire history.

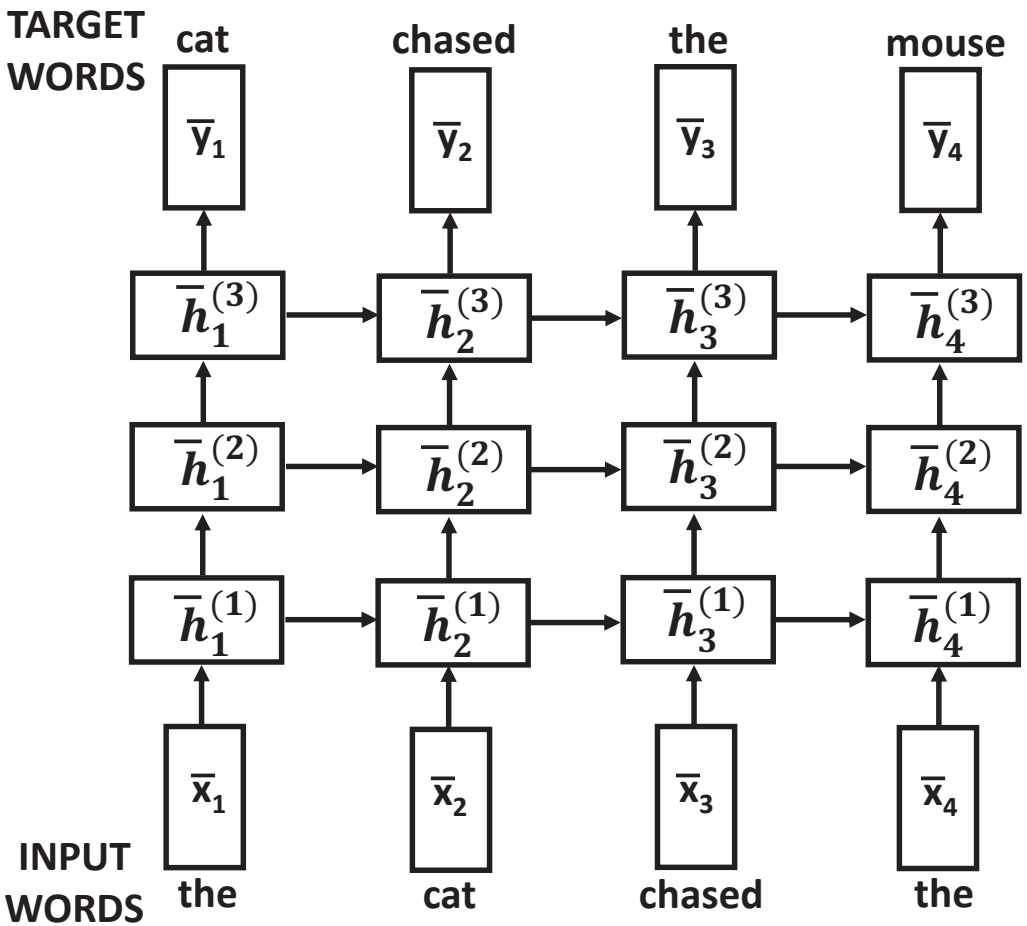
Language Modeling Example: Predicting the Next Word



- Predicting the next word for the sentence "*The cat chased the mouse.*"

- Lexicon of size four.

Multilayer Recurrent Networks



$$\overline{h}_t^{(k)} = \tanh\ W^{(k)} \begin{bmatrix} \overline{h}_t^{(k-1)} \\ \overline{h}_{t-1}^{(k)} \end{bmatrix}$$

Training a Recurrent Network

- Main difference from traditional backpropagation is the issue of shared parameters:
  - Pretend that the weights are not shared and apply normal backpropagation to compute the gradients with respect to each copy of the weights.
  - Add up the gradients over the various copies of each weight.
  - Perform the gradient-descent update.

- Algorithm is referred to as "*backpropagation through time.*"

Truncated Backpropagation through Time

- The number of layers in a recurrent network depends on the the length of the sequence.

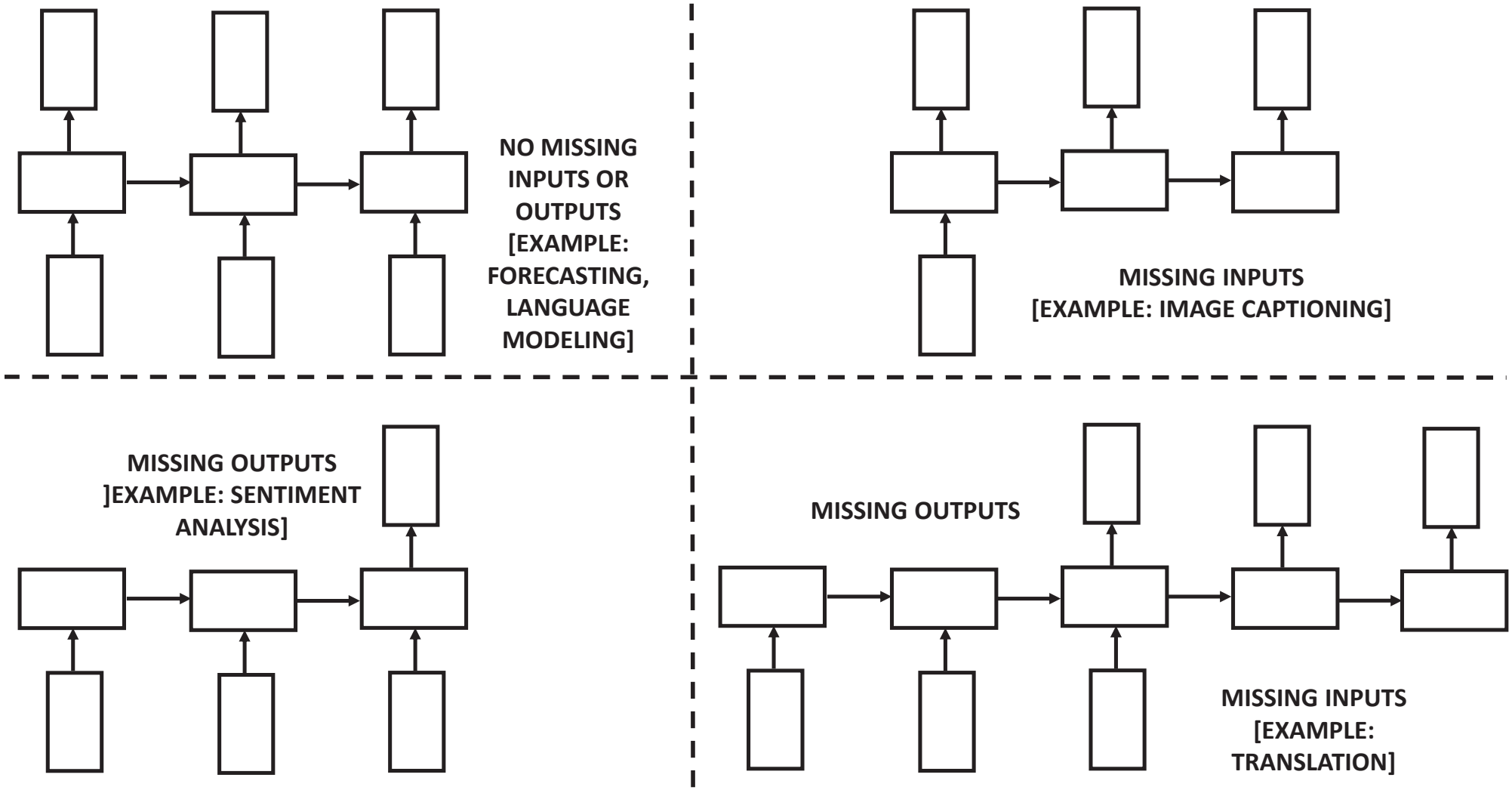
- Causes problems in memory, speed, and convergence.
  - Process chunk by chunk (around 100 sequence elements/layers).
  - Compute forward states exactly using the final state of previous chunk.
  - Compute loss backpropagation only over current chunk and update parameters $\Rightarrow$ Analogous to stochastic gradient descent.

Charu C. Aggarwal
IBM T J Watson Research Center
Yorktown Heights, NY

Applications of Recurrent Networks

Neural Networks and Deep Learning, Springer, 2018
Chapter 7.7

Recurrent Neural Network Applications are Architecture Sensitive!



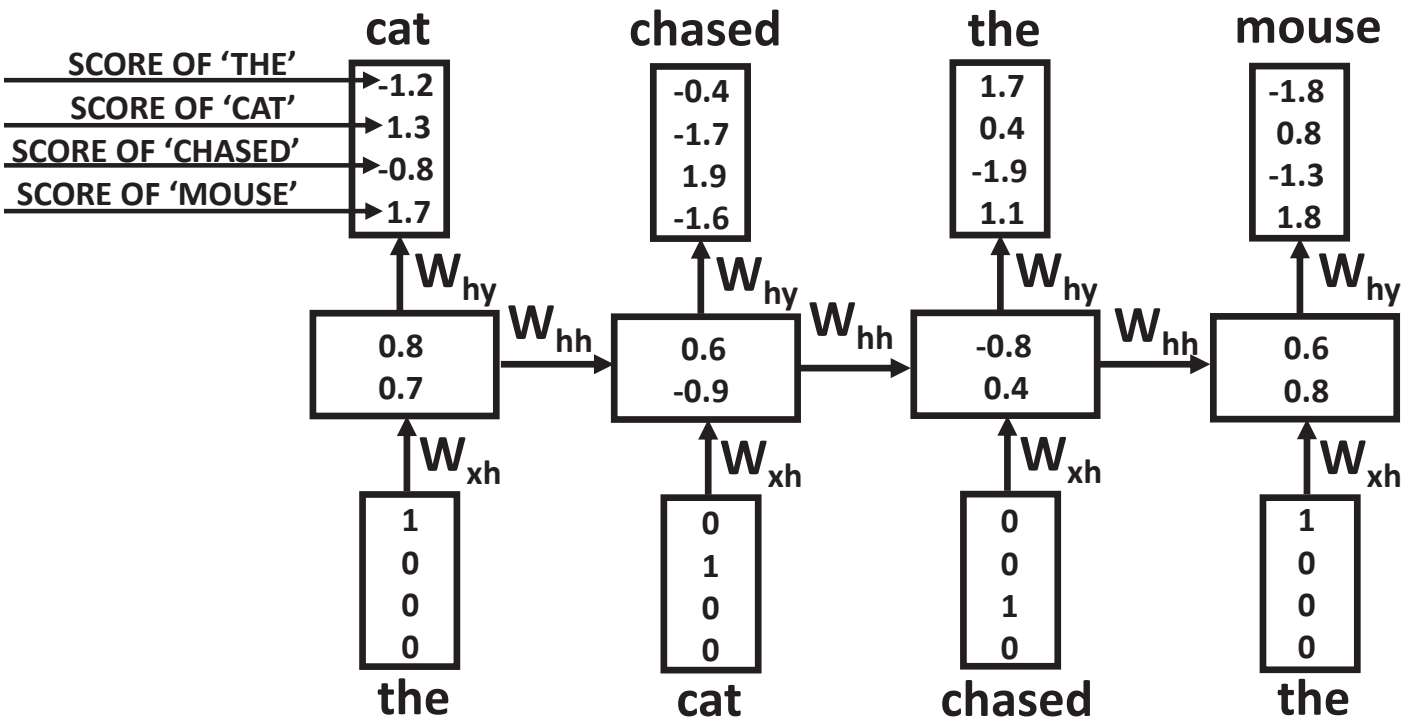
Missing Inputs

- Missing inputs represent a bigger challenge than missing outputs.
  - Missing inputs often occur at inference when output is sequence
  - Language modeling/Image captioning/machine translation
  - How to generate the second word in caption without knowing the first?
- Simply setting missing inputs to random values will result in bias
- Missing inputs are sequentially sampled from outputs at inference time [discussed a few slides later]

Observations

- Most of the applications use advanced variants like LSTMs and bidirectional recurrent networks.
  - Advanced variants covered in later lectures.
- Describe general principles using a simple recurrent network.
- Principles generalize easily to any type of architecture.

Language Modeling: Predicting the Next Word



- Predicting the next word for the sentence "*The cat chased the mouse.*"

- Can be used for time-series prediction.

Generating a Language Sample

- Predicting next word is straightforward (all inputs available).

- Predicting a language sample runs into the problem of missing inputs.
  - Sample a token generated at each time-stamp, and input to next time-stamp.
  - To improve accuracy, use beam search to expand on the most likely possibilities by always keeping track of the b best sequence prefixes of any particular length.
  - One can generate an arbitrary sequence of text that reflects the training data set.

Tiny Shakespeare Character-Level RNN: Karpathy, Johnson, Fei-Fei

- Executed code from `https://github.com/karpathy/char-rnn`

- After 5 iterations:

  KING RICHARD II:
  Do cantant,-'for neight here be with hand her,-
  Eptar the home that Valy is thee.

  NORONCES:
  Most ma-wrow, let himself my hispeasures;
  An exmorbackion, gault, do we to do you comforr,
  Laughter's leave: mire sucintracce shall have theref-Helt.

Tiny Shakespeare Character-Level RNN: Karpathy, Johnson, Fei-Fei

- After 50 iterations:

KING RICHARD II:
Though they good extremit if you damed;
Made it all their fripts and look of love;
Prince of forces to uncertained in conserve
To thou his power kindless. A brives my knees
In penitence and till away with redoom.

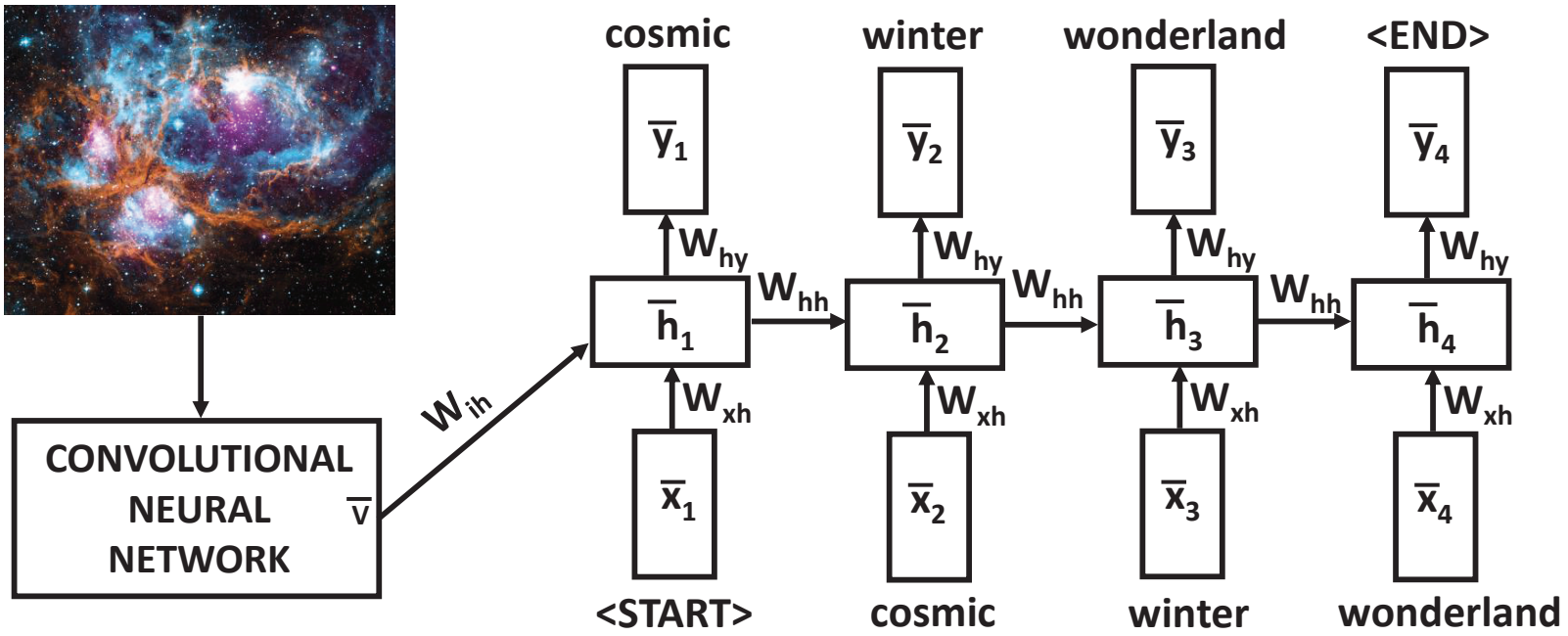
GLOUCESTER:
Between I must abide.

What Good is Language Modeling?

- Generating a language sample might seem like an exercise in futility.
  - Samples are syntactically correct but not semantically meaningful.

- Samples are often useful when conditioned on some other data:
  - **Conditioning on an image:** Image captioning
  - **Conditioning on a sequence:** Machine translation and sequence-to-sequence learning

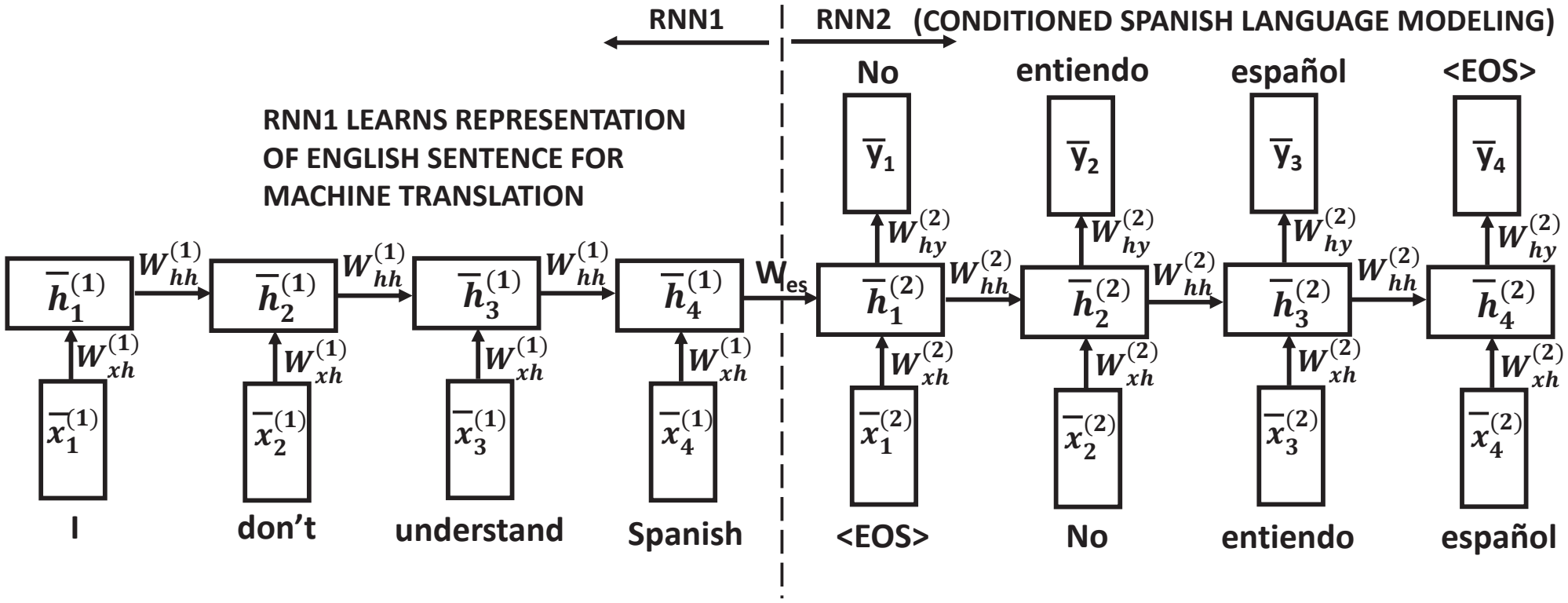
Image Captioning



- Language sample conditioned on an image $\Rightarrow$ Provides meaningful descriptions of images

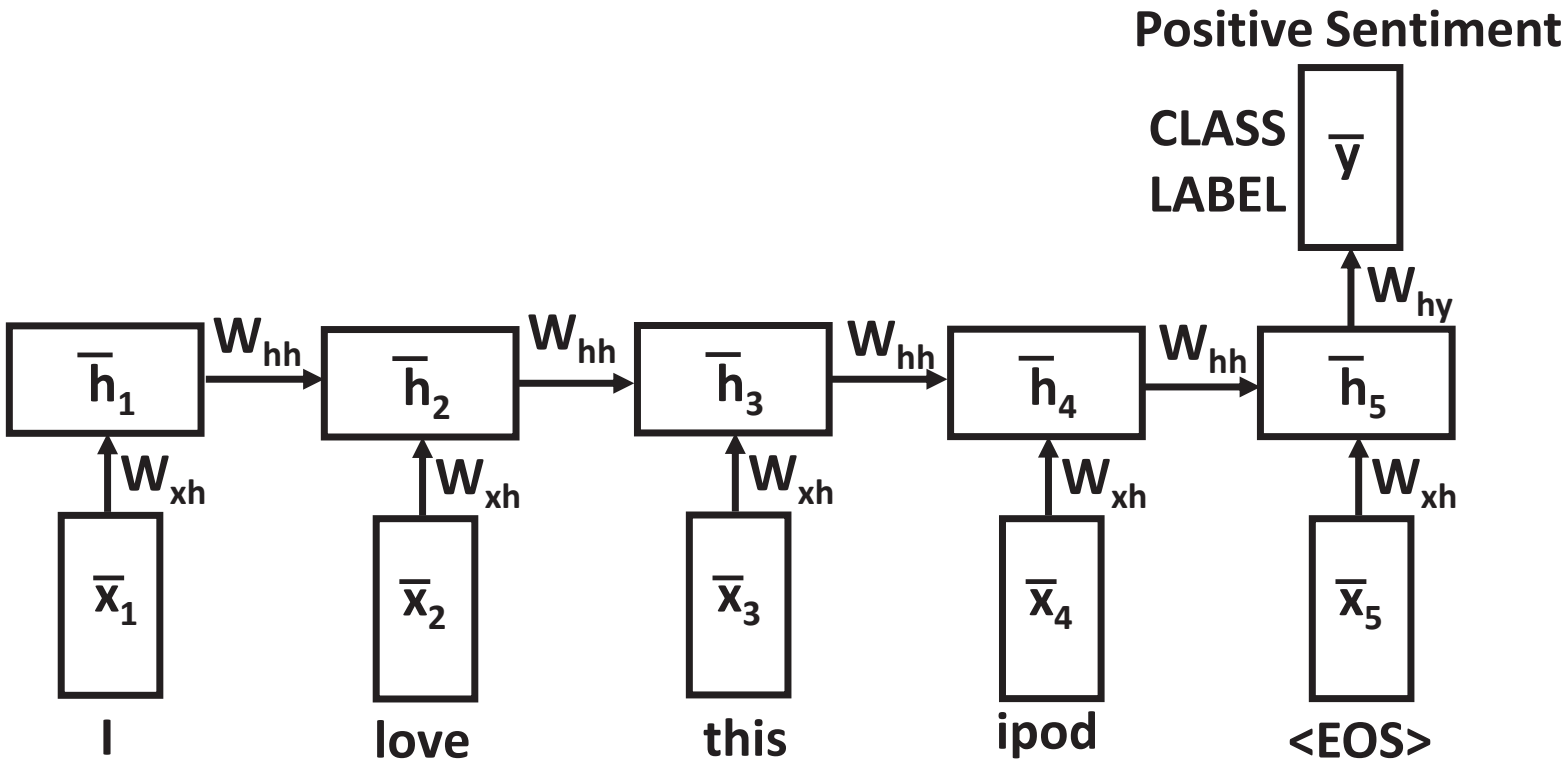
- Need START and END tokens

Machine Translation and Sequence-to-Sequence Learning



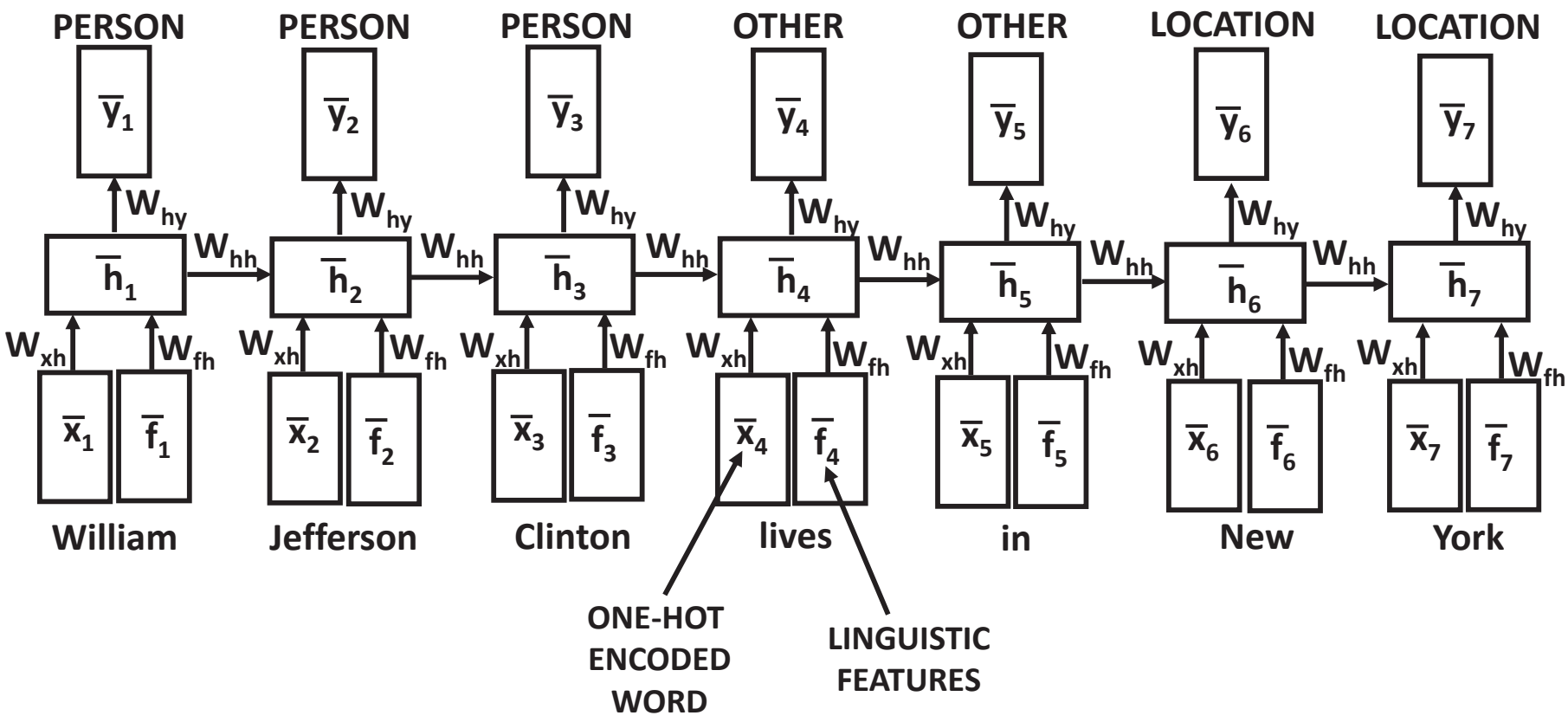
- Can be used for any sequence-to-sequence application including self-copies (recurrent autoencoders)

Sentence-Level Classification



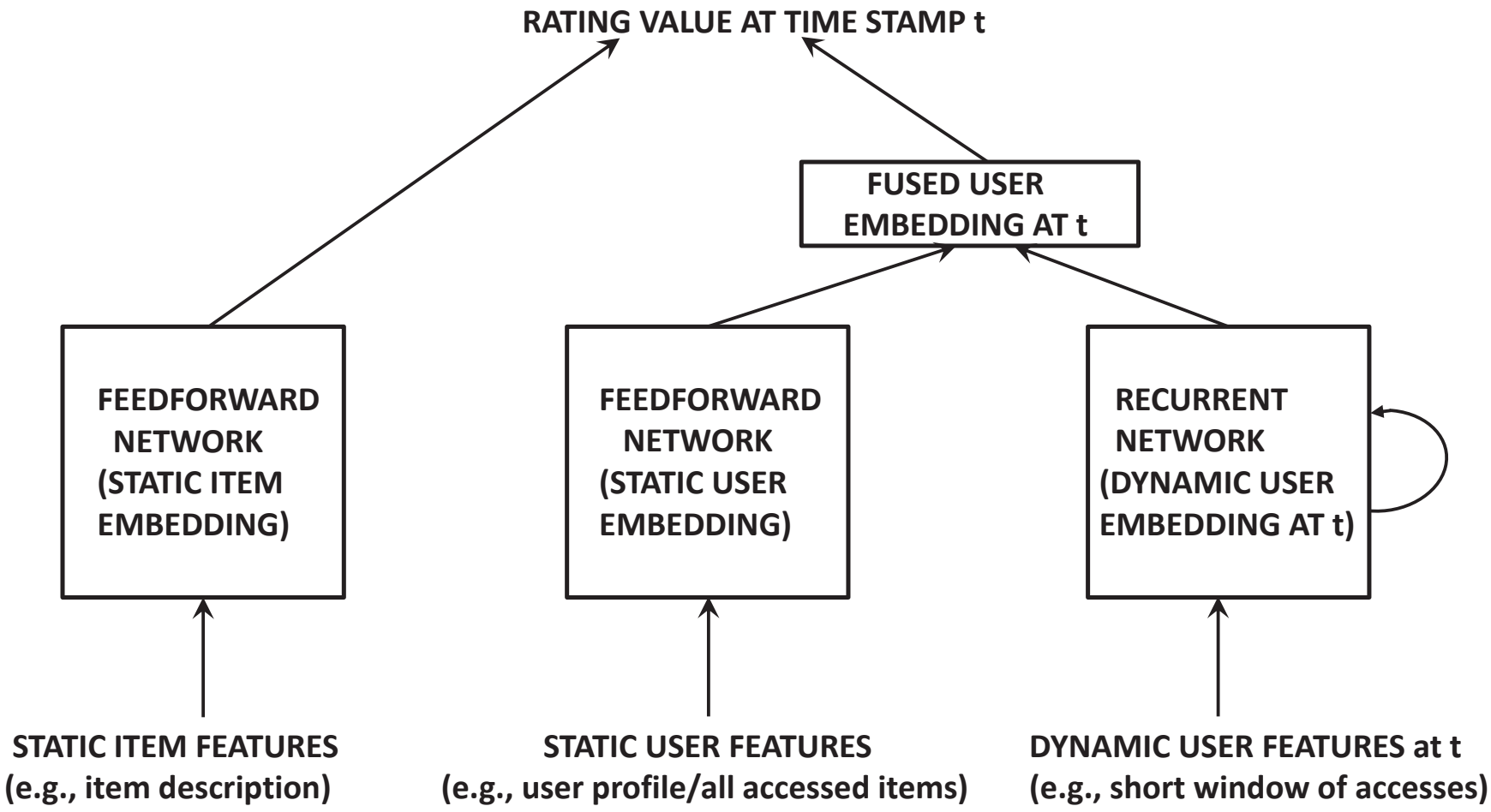
- Single target at the very end of the sentence.

Token-Level Classification



- Label output for each token.

Temporal Recommender Systems



- Label output for each token.

Protein Structure Prediction

- The elements of the sequence are the symbols representing one of the 20 amino acids.

- The 20 possible amino acids are akin to the vocabulary used in the text setting.

- Each position is associated with a class label: *alpha-helix*, *beta-sheet*, or *coil*.

  – The problem can be reduced to token-level classification.

Speech and Handwriting Recognition

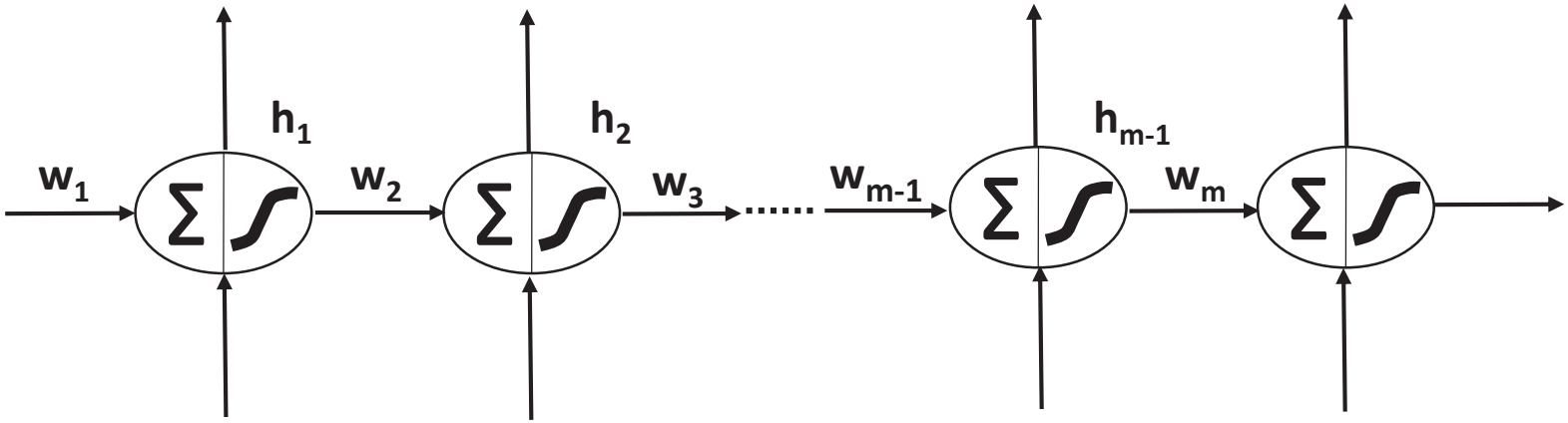
- Speech and handwriting recognition are sequential applications.
  - Frame representation of audios is transcribed into character sequence.
  - Convert sequence of strokes into character sequence.
  - Details and references in book.

Charu C. Aggarwal
IBM T J Watson Research Center
Yorktown Heights, NY

LSTMs and GRUs

Neural Networks and Deep Learning, Springer, 2018
Chapters 7.5 and 7.6

Vanishing and Exploding Gradient Problems



- Neural network with one node per layer.
- Backpropagated partial derivative get multiplied by weights and activation function derivatives.
- Unless the values are exactly one, the partial derivatives will either continuously increase (explode) or decrease (vanish).

Generalization to Multi-Node Layers

- Vectorwise back-propagation is done by using the Jacobian J.

$$\overline{g}_t = J^T \overline{g}_{t+1} \tag{1}$$

- The (i, j)th entry of J is the (i, j)th entry of the square hidden-to-hidden matrix W_{hh} multiplied with the derivative of the tanh function at the current value in node j.

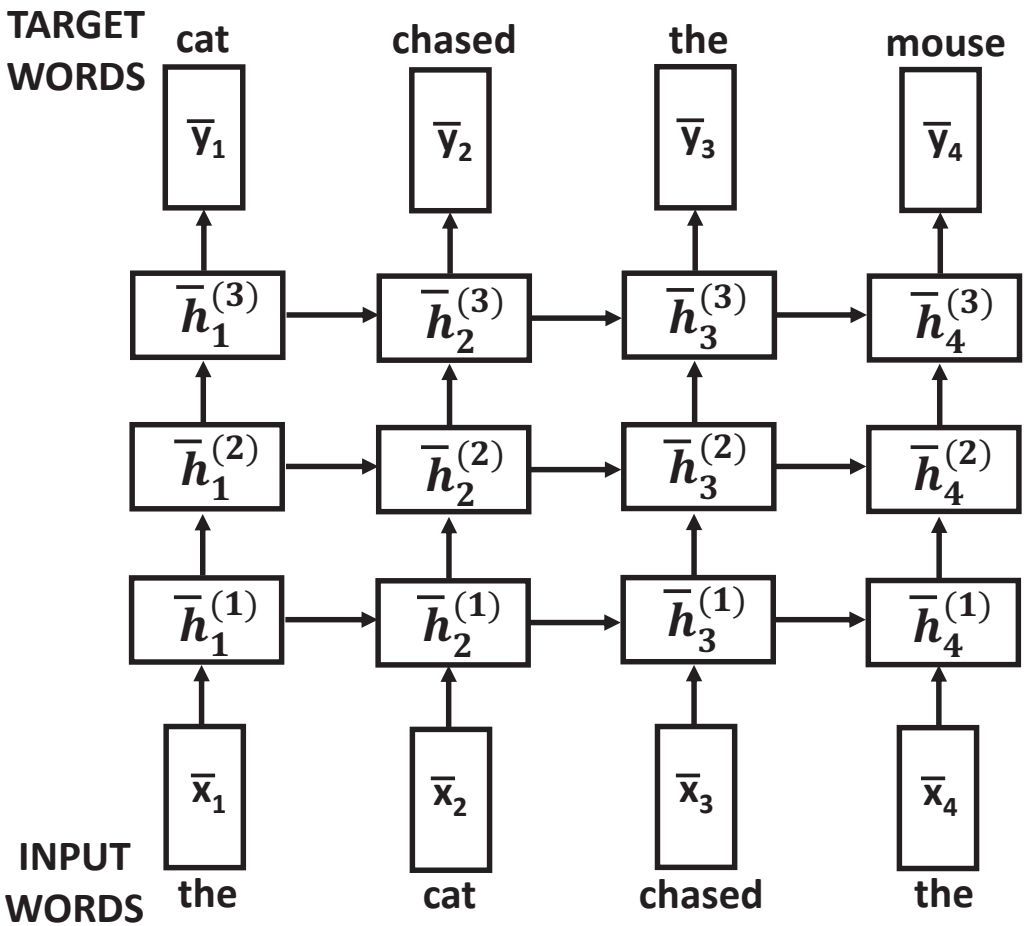
- Gradients cause successive multiplication with transposed Jacobian $J^T = P \Delta P^{-1}$.

- Multiplying m times leads to $(J^T)^m = P \Delta^m P^{-1} \Rightarrow$ Largest eigenvalue decides everything!

Other Issues with Recurrent Networks

- Hard to retain the information in a hidden state with successive matrix multiplications.
  - Hidden states of recurrent networks are inherently short-term.
  - No mechanism for fine-grained control of what information to retain from hidden states.
- The LSTM uses *analog gates* to control the flow of information.

Recap: Multilayer Recurrent Networks



$$\overline{h}_t^{(k)} = \tanh\ W^{(k)} \begin{bmatrix} \overline{h}_t^{(k-1)} \\ \overline{h}_{t-1}^{(k)} \end{bmatrix}$$

Long-Term vs Short Term Memory

- A recurrent neural network only carries forward a hidden state $\overline{h}_t^{(k)}$ across time layers.

- An LSTM carries forward both a hidden state $\overline{h}_t^{(k)}$ and a cell state $\overline{c}_t^{(k)}$.
  - The hidden state is like short-term memory (updated aggressively).
  - The cell state is like long-term memory (updated gently).
  - Gates used to control updates from layer to layer.
  - Leaking between short-term and long-term memory allowed.

Setting Up Intermediate Variables and Gates

- Assume that hidden state and cell state are p-dimensional vectors.

- The matrix $W^{(k)}$ is of size $4p \times 2p$ [$4p \times (p + d)$ for $k = 1$].

- The intermediate variables are p-dimensional vector variables $\overline{i}$, $\overline{f}$, $\overline{o}$, and $\overline{c}$ that can be stacked to create a $4p$-dimensional vector:

$$\begin{matrix}\text{Input Gate:}\\ \text{Forget Gate:}\\ \text{Output Gate:}\\ \text{Cell Increment:}\end{matrix} \begin{bmatrix} \overline{i} \\ \overline{f} \\ \overline{o} \\ \overline{c} \end{bmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^{(k)} \begin{bmatrix} \overline{h}_t^{(k-1)} \\ \overline{h}_{t-1}^{(k)} \end{bmatrix} \quad (2)$$

- Right-hand side of update equation looks similar to that of a multilayer recurrent network, except that we are setting up 4 intermediate variable vectors.

Updating Cell States and Hidden States

- Selectively forget and/or add to long-term memory

$$\overline{c}_t^{(k)} = \underbrace{\overline{f} \odot \overline{c}_{t-1}^{(k)}}_{\text{Reset?}} + \underbrace{\overline{i} \odot \overline{c}}_{\text{Increment?}} \tag{3}$$

  - Long-term memory can be thought of as *iterative feature engineering* rather than *hierarchical feature engineering* $\Rightarrow$ Principle used in ResNet

- Selectively leak long-term memory to hidden state

$$\overline{h}_t^{(k)} = \overline{o} \odot \tanh(\overline{c}_t^{(k)}) \tag{4}$$

  - In some variations, the tanh function is not used in Equation 4, and the raw cell state is used.

Intuition for LSTM

- Examine the LSTM with single dimension:

$$c_t = c_{t-1} * f + i * c \tag{5}$$

- Partial derivative of c_t with respect to c_{t-1} is $f \Rightarrow$ multiply gradient flow with f!

- Element wise multiplication ensures that the result is also true for p-dimensional states.

- Each time-stamp has a different values of $f \Rightarrow$ less likely to cause vanishing gradients

- Gradient flow for c_t inherited by h_t because $h_t = o * \tanh(c_t)$

Gated Recurrent Unit

- Simplification of LSTM but not a special case.
  - Does not use explicit cell states.
  - Controls updates carefully.

GRU Updates

- Use two matrices $W^{(k)}$ and $V^{(k)}$ of sizes $2p \times 2p$ and $p \times 2p$, respectively.
  - In the first layer, the matrices are of sizes $2p \times (p+d)$ and $p \times (p+d)$.

- intermediate, p-dimensional vector variables $\overline{z}_t$ and $\overline{r}_t$, respectively $\Rightarrow$ Update and reset gates

$$\begin{array}{l}\text{Update Gate:} \\ \text{Reset Gate:}\end{array} \begin{bmatrix} \overline{z} \\ \overline{r} \end{bmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \end{pmatrix} W^{(k)} \begin{bmatrix} \overline{h}_t^{(k-1)} \\ \overline{h}_{t-1}^{(k)} \end{bmatrix} \tag{6}$$

$$\overline{h}_t^{(k)} = \overline{z} \odot \overline{h}_{t-1}^{(k)} + (1 - \overline{z}) \odot \tanh\, V^{(k)} \begin{bmatrix} \overline{h}_t^{(k-1)} \\ \overline{r} \odot \overline{h}_{t-1}^{(k)} \end{bmatrix} \tag{7}$$

Explanation of Updates

- The reset gate $\overline{r}$ decides how much of the hidden state to carry over from the previous time-stamp for a matrix-based transformation.

- The update gate $\overline{z}$ decides the *relative* strength of the matrix-based update and a more direct copy from previous time stamp.

- The direct copy from the previous time stamp stabilizes the gradient flows.

Intuition for GRU

- Consider a 1-dimensional and single-layer GRU update:

$$h_t = z \cdot h_{t-1} + (1 - z) \cdot \tanh[v_1 \cdot x_t + v_2 \cdot r \cdot h_{t-1}] \quad (8)$$

- One can compute the derivative as follows:

$$\frac{\partial h_t}{\partial h_{t-1}} = z + (\text{Additive Terms}) \quad (9)$$

- The extra term $z \in (0, 1)$ helps in passing *unimpeded* gradient flow and makes computations more stable.
  - Additive terms heavily depend on $(1 - z)$.
  - Gradient factors are *different* for each time stamp.

- These networks are sometimes referred to as *highway networks*.

Comparisons of LSTM and GRU

- K. Greff et al. LSTM: A search space odyssey. *IEEE TNNLS*, 2016.
  - Many variants of LSTM equations.
- J. Chung et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*, 2014.
- R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. *International Conference on Machine Learning*, pp. 2342–2350, 2015.
- Main advantage of GRU is simplicity.
- None of the LSTM variants could perform better than it in a reliable way.