Technische Hochschule Rosenheim

Applications of & Introduction to Artificial Intelligence (A2I2)

Recurrent Neural Networks and Applications of Artificial Intelligence

Technische Hochschule Rosenheim

Sommer 2020

Prof. Dr. M. Tilly

Agenda

- Analyse Contents of images with Azure, Google and AWS
- Services for custom vision models and exporting for various frameworks
- Detect and identify people, emotions and age
- Extract text, key-value pairs, and tables from documents
- Recognize digital ink and handwriting

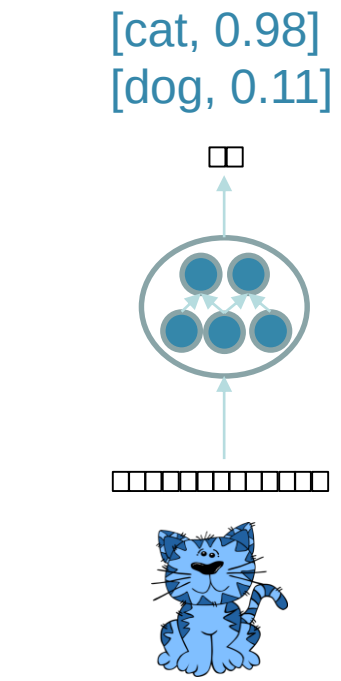
CNNs

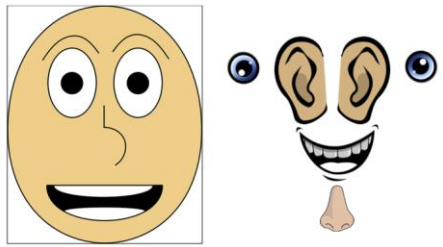
Main focus of Convolutional Neural Networks (CNNs):

- Uses Convolution and MaxPooling
- Classify everything which can be expressed as an image
- A CNN makes predictions by looking at an image and then checking to see if certain components are present in that image or not. If they are, then it classies that image accordingly
- There are limitations of CNNs
  - Fixed sized input and a fixed size output
  - Rely on the assumption of independence among the (training and test) examples.
  - After each data point is processed, the entire state of the network is lost
- Traditional neural networks can’t characterize temporal dependencies
  - e.g. classify what is happening at every point in a movie
  - How a neural network can inform later events about the previous ones?

Limitations of CNNs

- Main focus of CNNs:
  - Classify everything which can be expressed as an “**image**”
  - A CNN makes predictions by looking at an image and then checking to see if certain components are present in that image or not. If they are, then it classifies that image accordingly
- There are limitations of CNNs
  - Fixed sized input and a fixed size output
  - Rely on the assumption of independence among the (training and test) examples.
  - After each data point is processed, the entire state of the network is lost
- Traditional neural networks can’t characterize temporal dependencies
  - e.g. classify what is happening at every point in a movie
  - How a neural network can inform later events about the previous ones





What happens here?

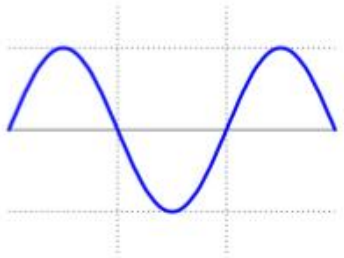
- Sequence modeling: "This morning I took the dog for a …."
  - Fixed window could use for example last 2 words for prediction!
- Long-term dependencies: "In France, I had a great time and I learnt some of the … language."
  - We need information from the past (and maybe future) to guess.
- Use entire sequence as a set of count: Bag of words!
  - But "The food was good, not bad at all." vs. "The food was bad, not good at all."
- References as parameter: "I like ~the~ book. Normally I do not like thriller but this one is great".

What is a Sequence?

- Sentence

"I took the dog fo a walk this morning."

- Function



- Waveform

Motivation for Seqeunce-Models

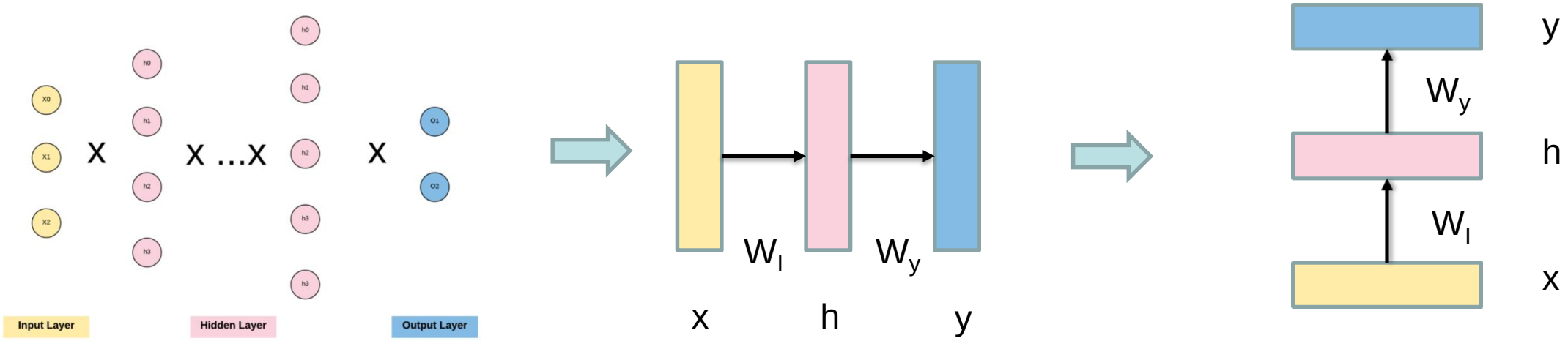
- Not all problems can be converted into one with fixed-length inputs and outputs!
- Problems such as Speech Recognition or Time-Series Prediction require a system to store and use context information!
- Simple case:
  - Output YES if the number of 1s is even, else NO!
    - 1000010101… –> YES
    - 1000000011.. –> NO
- Hard/Impossible to choose a fixed context window!
- There can always be a new sample longer than anything seen

Sequence models

- Sequence models are great if we need
  - … to deal with **variable-length** in sequences
  - … to maintain **sequence order**
  - … to keep track of **long-term dependencies**
  - …. to **share parameters** across the sequence

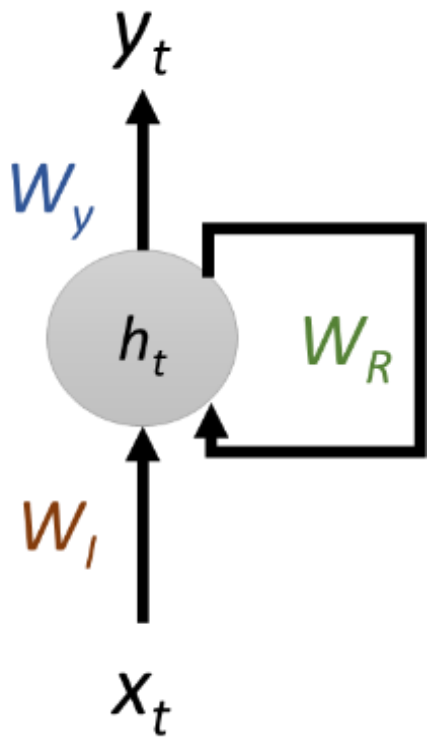
What are Recurrent Neural Networks (RNN)?

- Main idea is to make use of sequential information
- Perform the same task for every element of a sequence (that's what recurrent stands for)
- Output depends on the previous computations
- Another way of interpretation: RNNs have a "memory"
- Keep the basic idea of a **multi layer neural network**!



What are RNNs? – more formal -

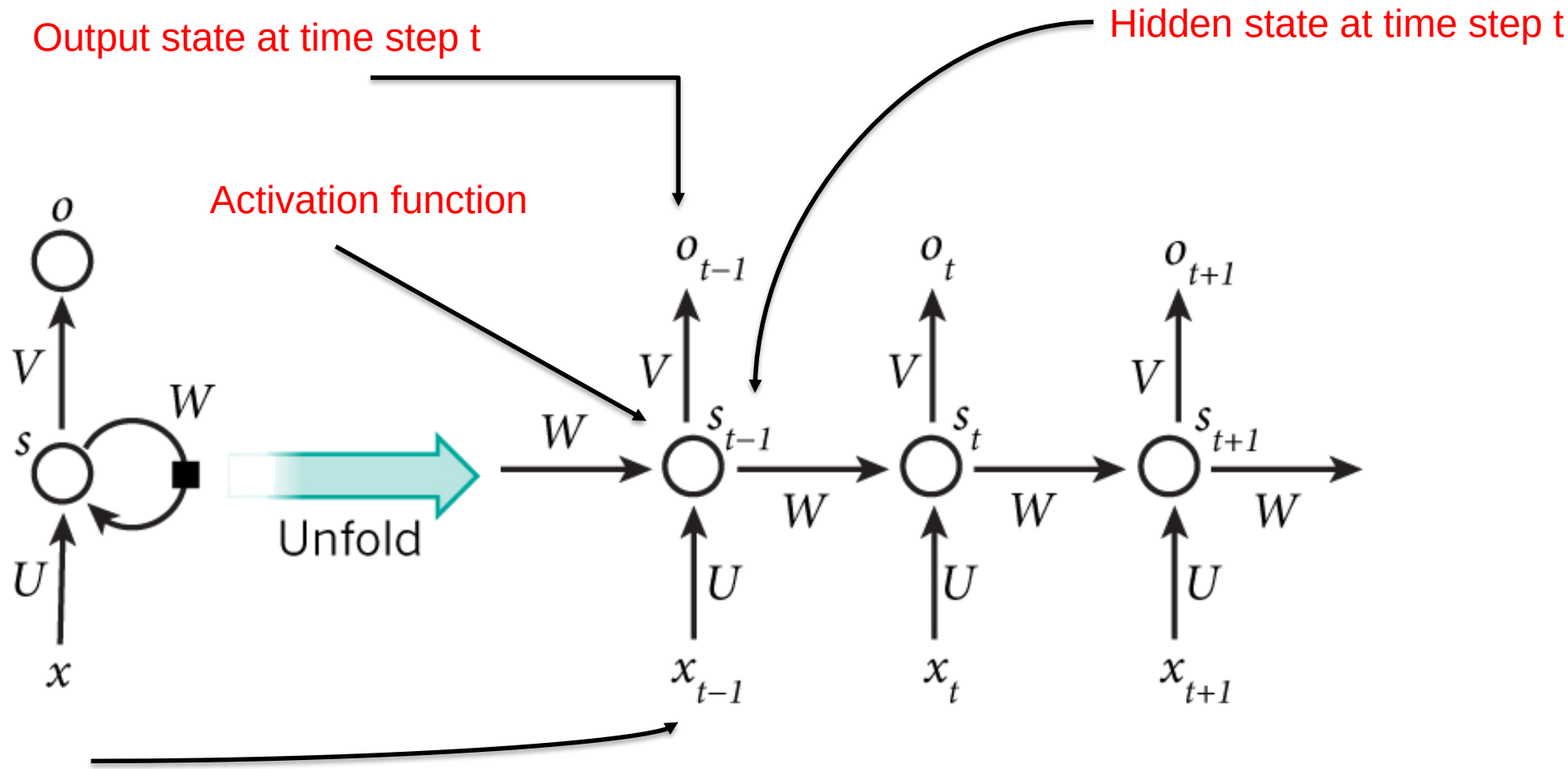
- Recurrent neural networks (RNNs) are connected models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time.



$$h_t = g_h(W_I x_t + W_R h_{t-1} + b_h)$$

$$y_t = g_y(W_y h_t + b_y)$$

(Vanilla) Recurrent Network

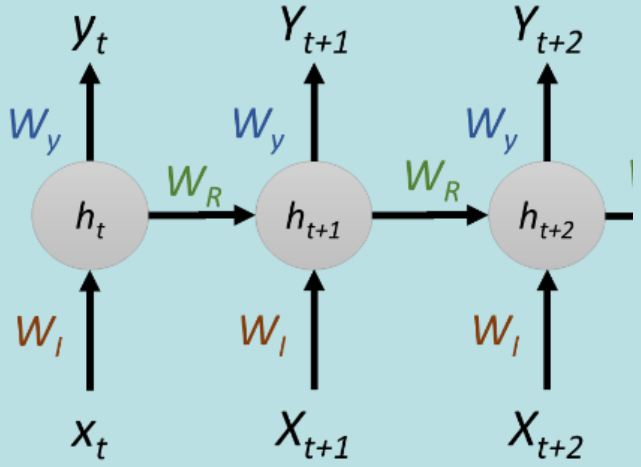


Unrolling a recurrent network

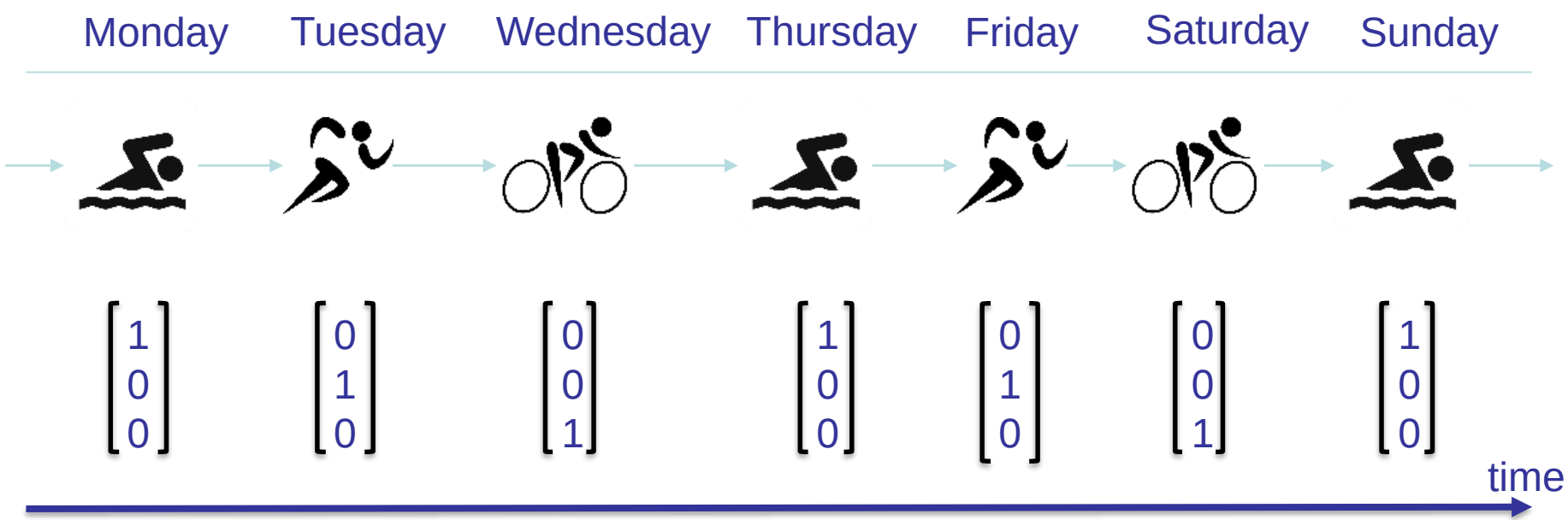
```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y

rnn1 = RNN()
rnn2 = RNN()

x =  np.array([1,0,0])
yt = rnn1.step(x)
yt_1 = rnn2.step(yt)
```

$$h_t = g_h(W_I x_t + W_R h_{t-1} + b_h)$$

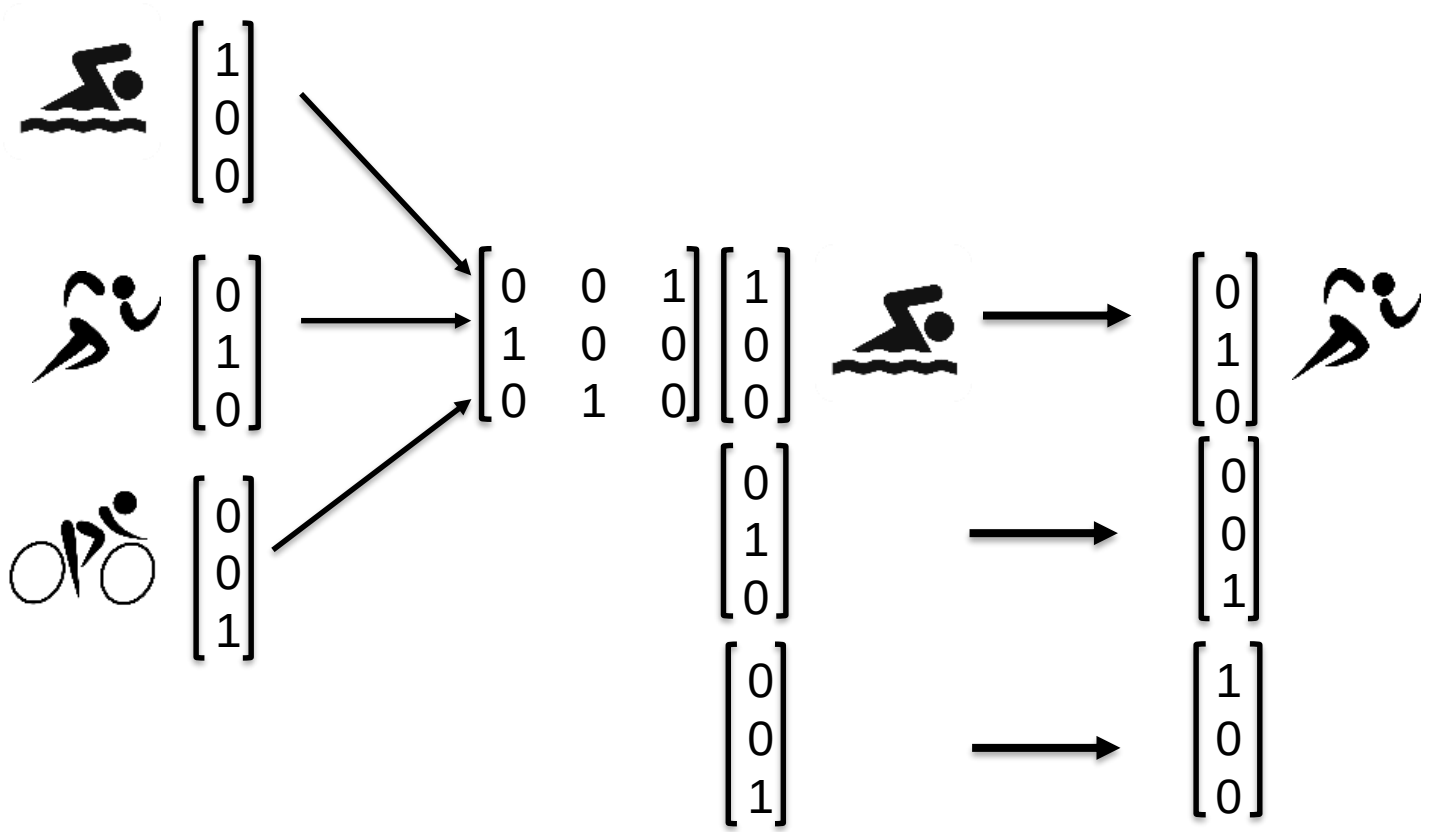
$$y_t = g_y(W_y h_t + b_y)$$



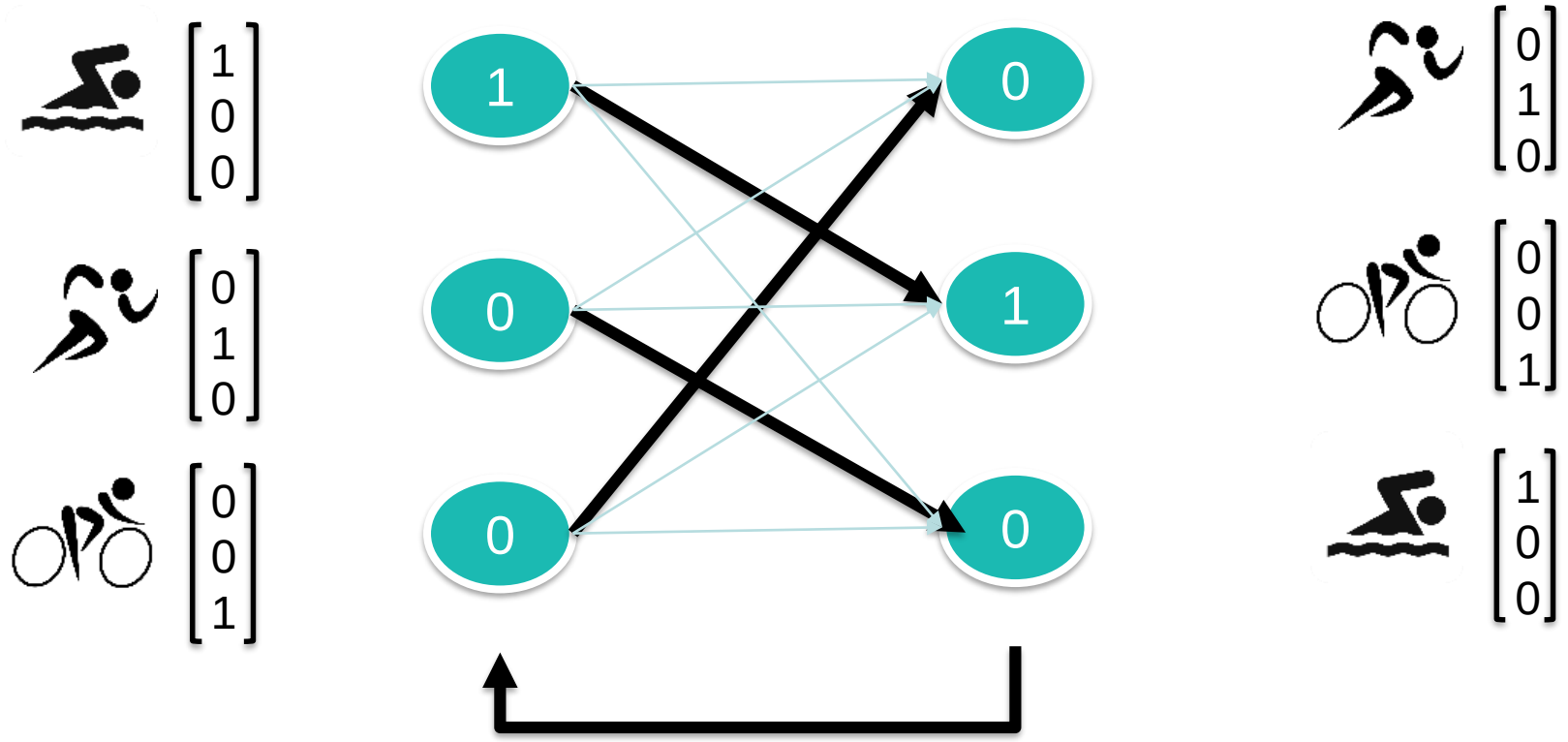
Motivation: Planning of training



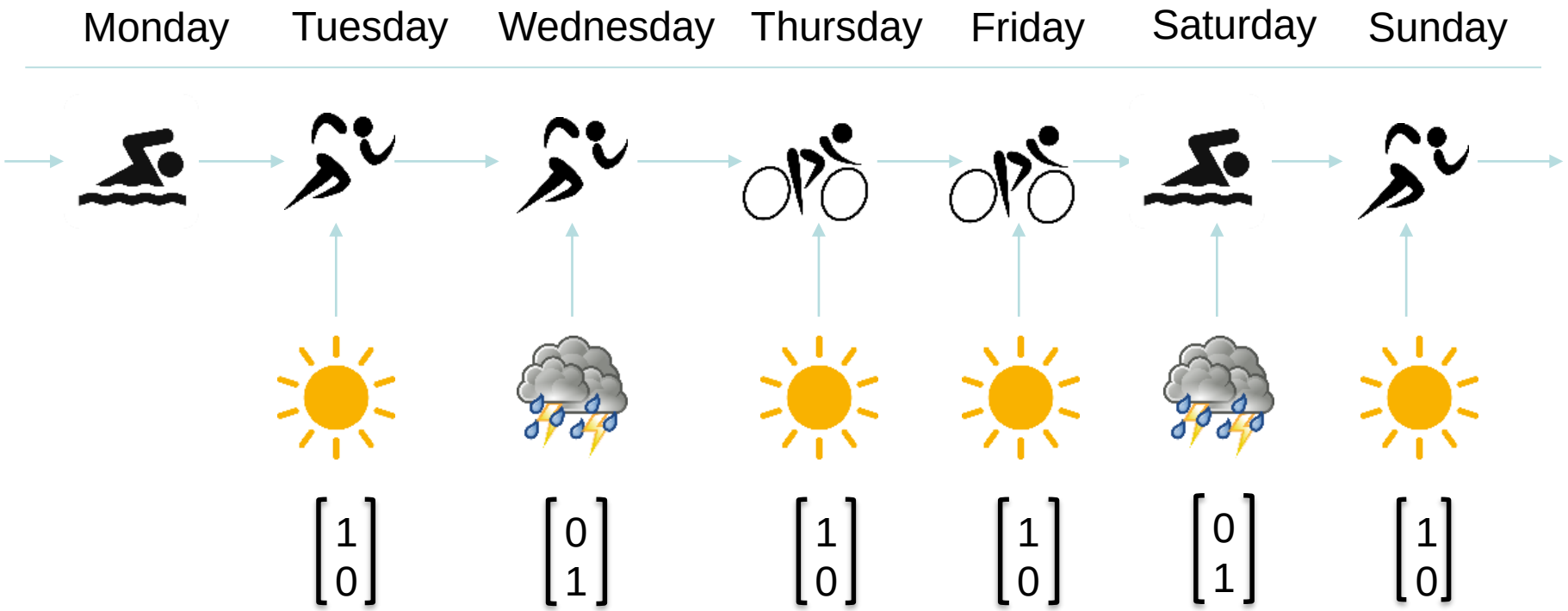
Towards a simple (recurrent) Neural Network



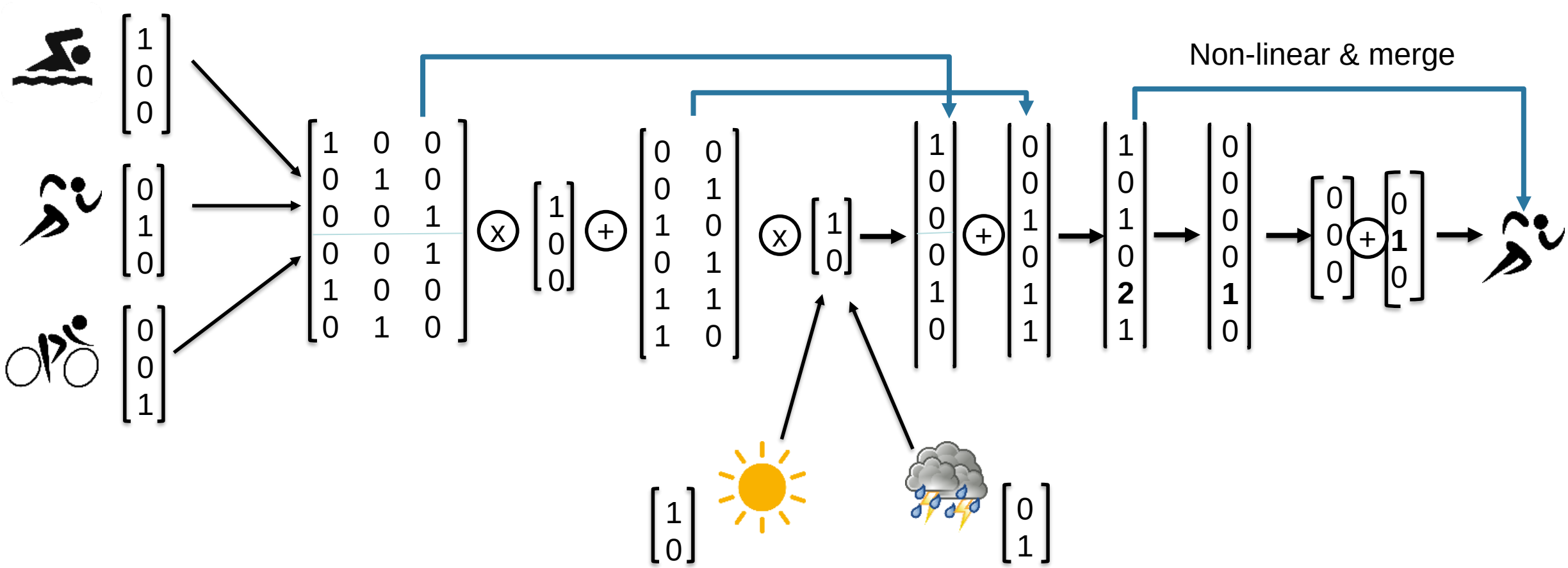
Simple (Recurrent) Neural Network



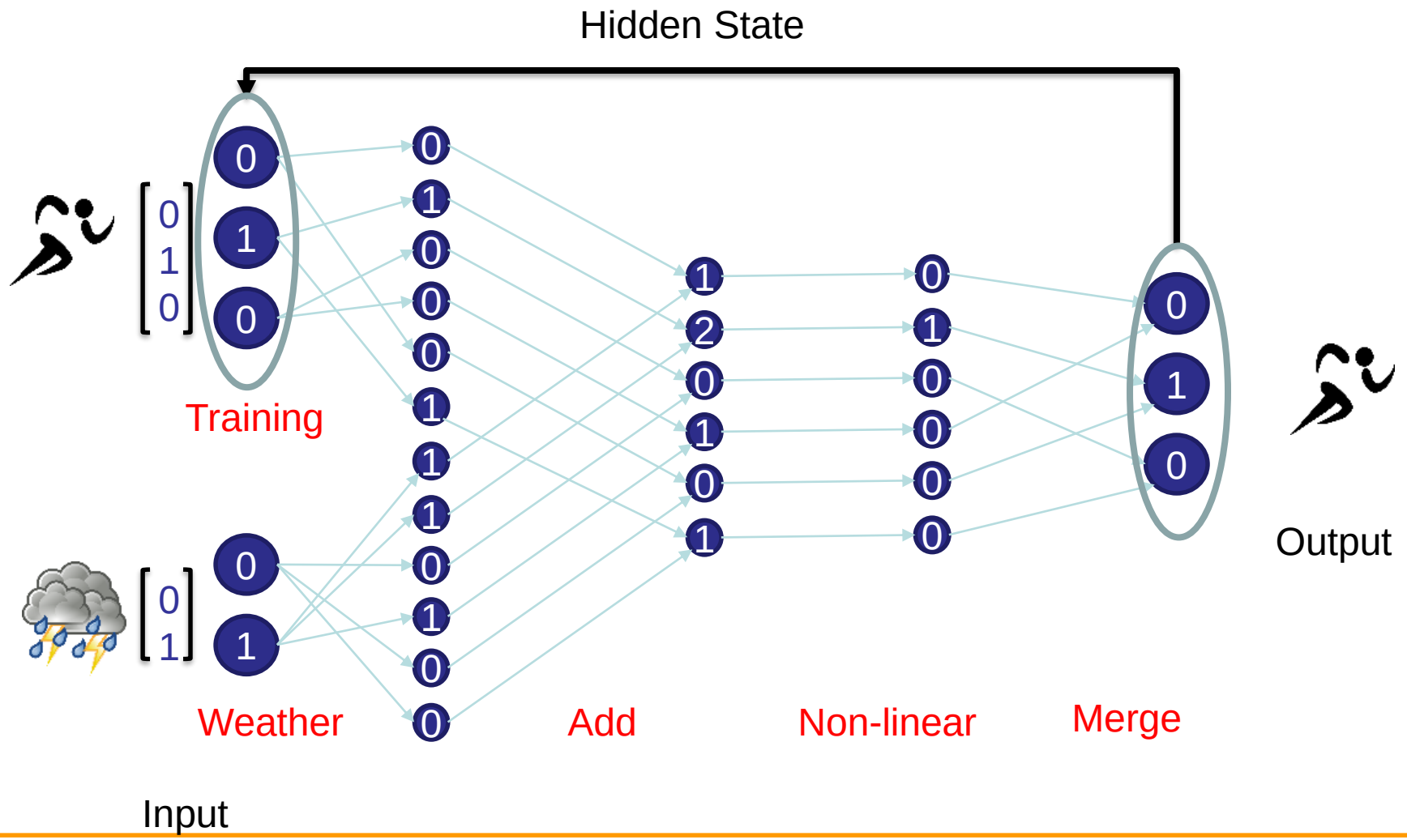
Motivation: Planning (incl. weather)



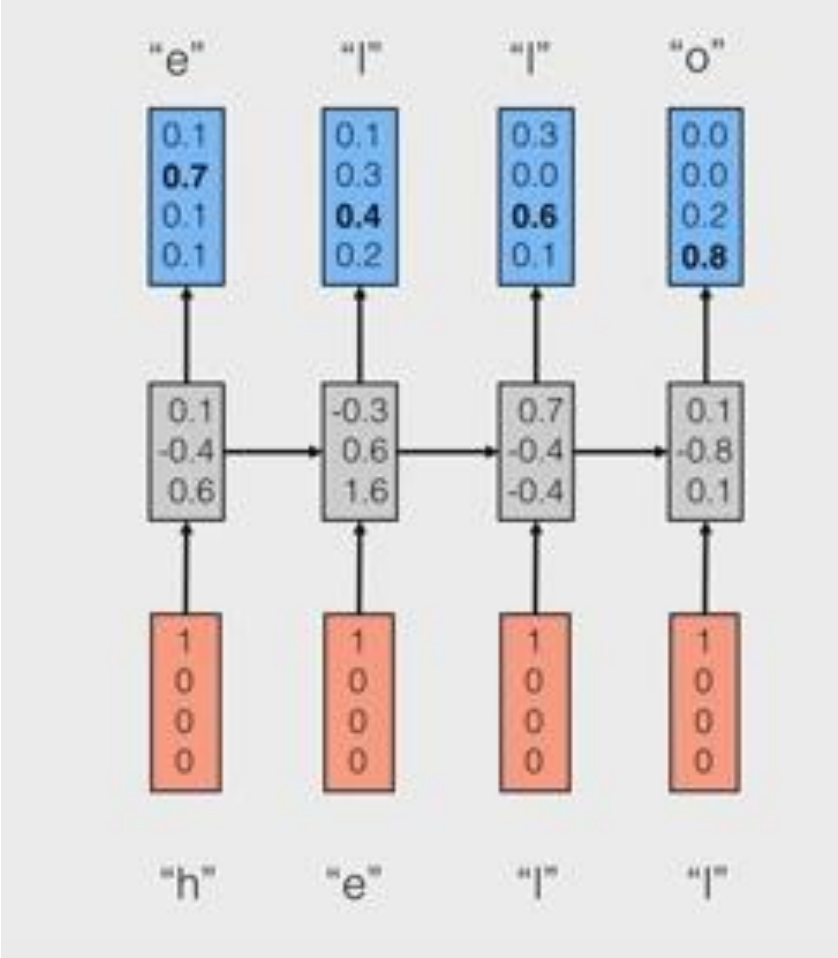
Towards a more complex RNN



Recurrent Neural Network *for my Training*



RNN in practice



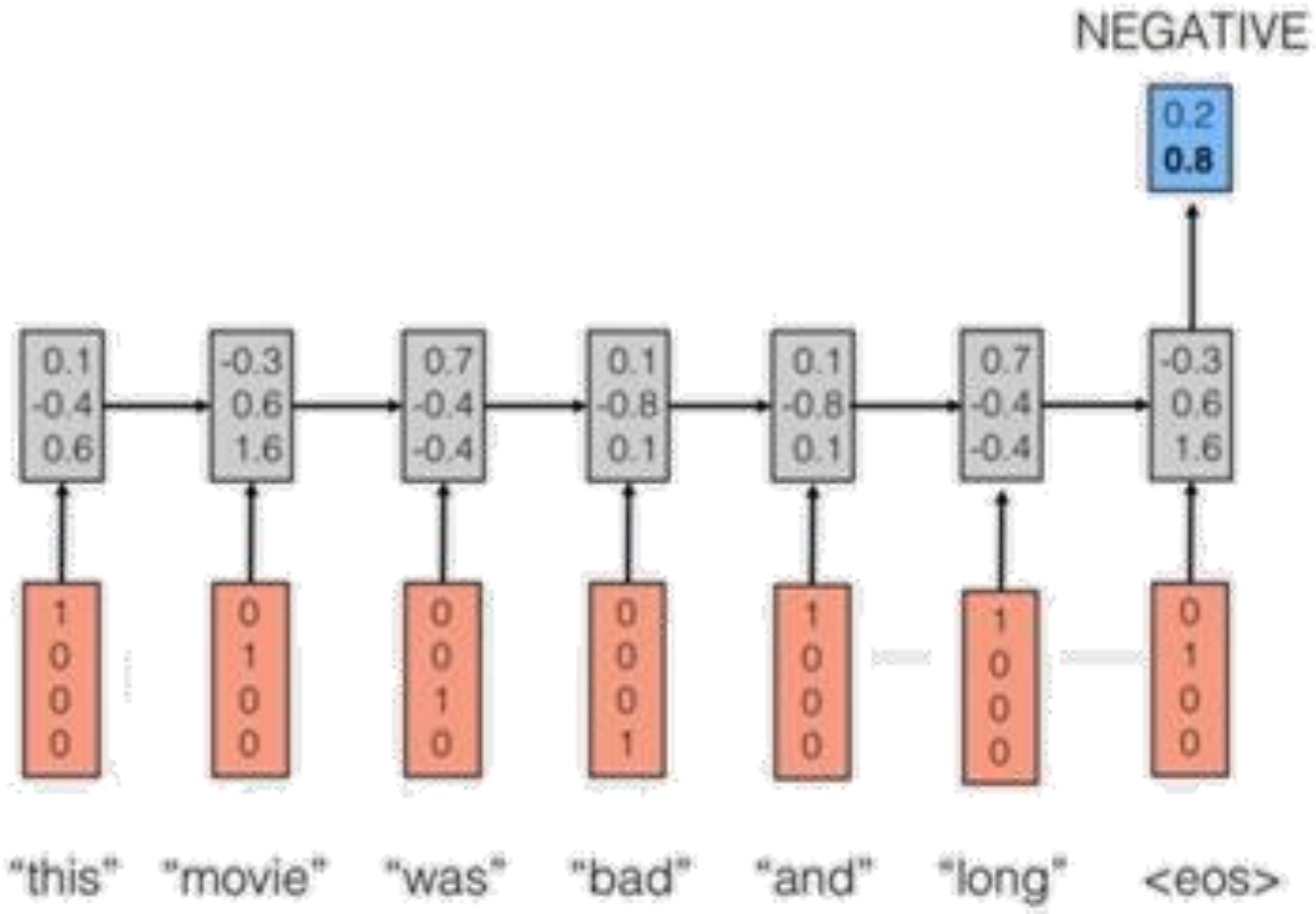
Input Alphabet

h 1 0 0 0 e 0 1 0 0 l 0 0 1 0 o 0 0 0 1

Learned language model

$$P(c_t | \{c_{t-1}, c_{t-2}, \ldots, c_0\})$$

RNN in practice



Computer Vision API

Content of Image:

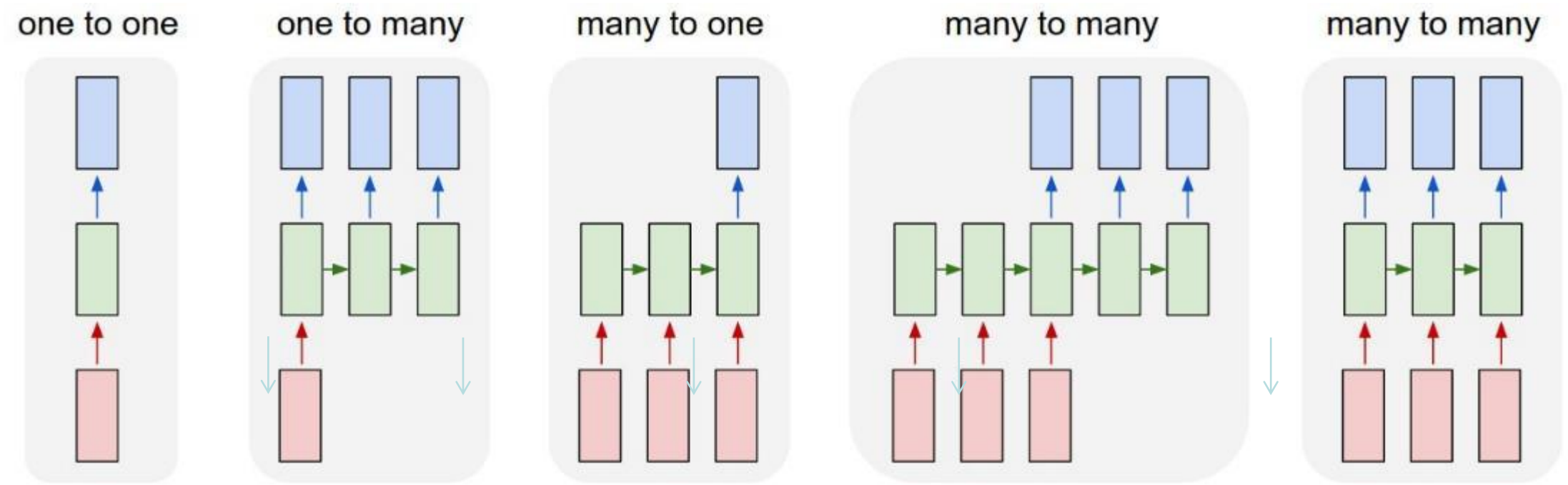
[{ "name": "grass", "confidence": 0.9999992847442627 },
{ "name": "outdoor", "confidence": 0.9999072551727295 },
{ "name": "cow", "confidence": 0.99954754114151 },
{ "name": "field", "confidence": 0.9976195693016052 },
{ "name": "brown", "confidence": 0.988935649394989 },
{ "name": "animal", "confidence": 0.97904372215271 },
{ "name": "standing", "confidence": 0.9632768630981445 },
{ "name": "mammal", "confidence": 0.9366017580032349, "hint": "animal" },
{ "name": "wire", "confidence": 0.8946959376335144 },
{ "name": "green", "confidence": 0.8844101428985596 },
{ "name": "pasture", "confidence": 0.8332059383392334 },
{ "name": "bovine", "confidence": 0.5618471503257751, "hint": "animal" },
{ "name": "grassy", "confidence": 0.48627158999443054 },
{ "name": "lush", "confidence": 0.1874018907546997 },
{ "name": "staring", "confidence": 0.165890634059906 }]

Content of Image:

Describe

0.975 "a brown cow standing on top of a lush green field“
0.974 “a cow standing on top of a lush green field”
0.965 “a large brown cow standing on top of a lush green field”

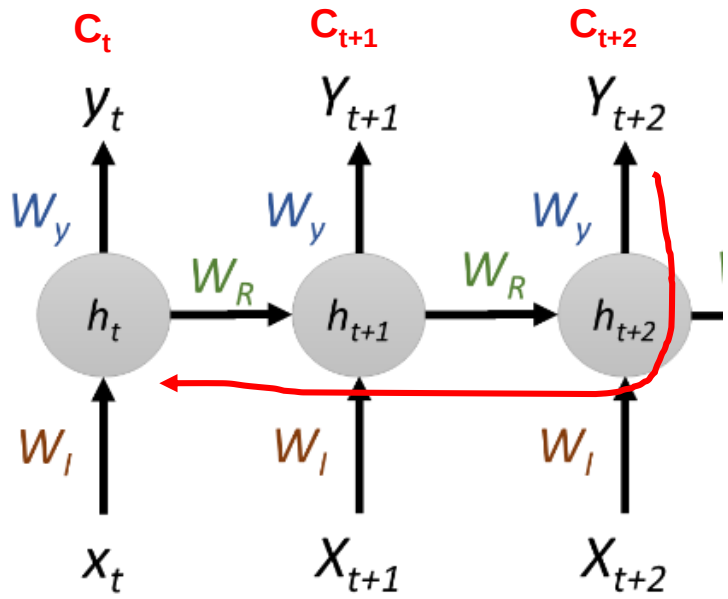
Recurrent Network Models



How to train a RNN?
Backpropagation Through Time (BPTT)

$$h_t = g_h(W_I x_t + W_R h_{t-1} + b_h)$$

$$y_t = g_y(W_y h_t + b_y)$$



Objective is to update the weight matrix

$$W_R \rightarrow W_R - \eta \frac{\partial C}{\partial W_R}$$

The gradients of the error with respect to our parameters
Just like we sum up the errors, we also sum up the gradients at each time step for one training example. For parameter W, the gradient is

$$\frac{\partial C}{\partial W_R} = \sum_t \frac{\partial C_t}{\partial W_R}$$

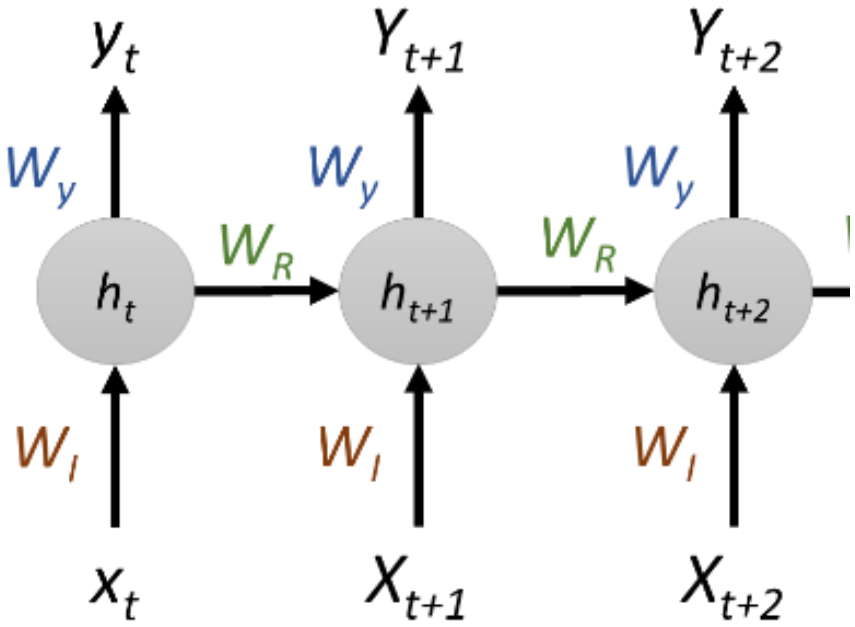
Example gradient (for t=2) (using Chain Rule):

$$\frac{\partial C_2}{\partial W_R} = \frac{\partial C_2}{\partial Y_2}\frac{\partial Y_2}{\partial h_2}\frac{\partial h_2}{\partial g}\frac{\partial g}{\partial a}\frac{\partial a}{\partial W_R}$$

$$a = (W_1 x_2 + W_R h_1 + b_h)$$

Depends on W_R too!

Vanishing/Exploding gradients

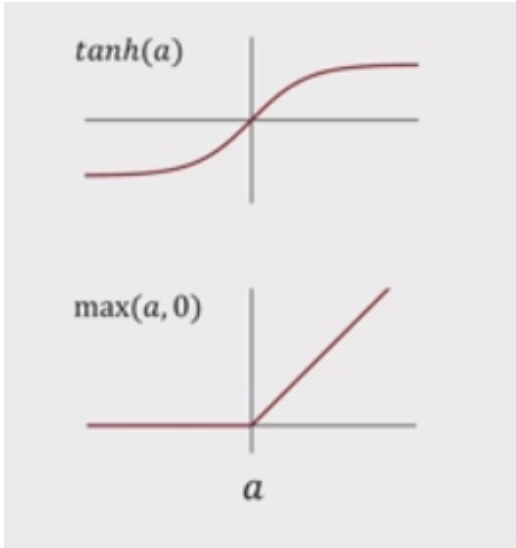


$$\frac{\partial C_{100}}{\partial W} = \frac{\partial C_{100}}{\partial Y_{100}} \dots W_R \frac{\partial g_{100}}{\partial a_{100}} \dots W_R \frac{\partial g_{99}}{\partial a_{99}} \dots . W_R \frac{\partial g_{98}}{\partial a_{98}} \dots .$$

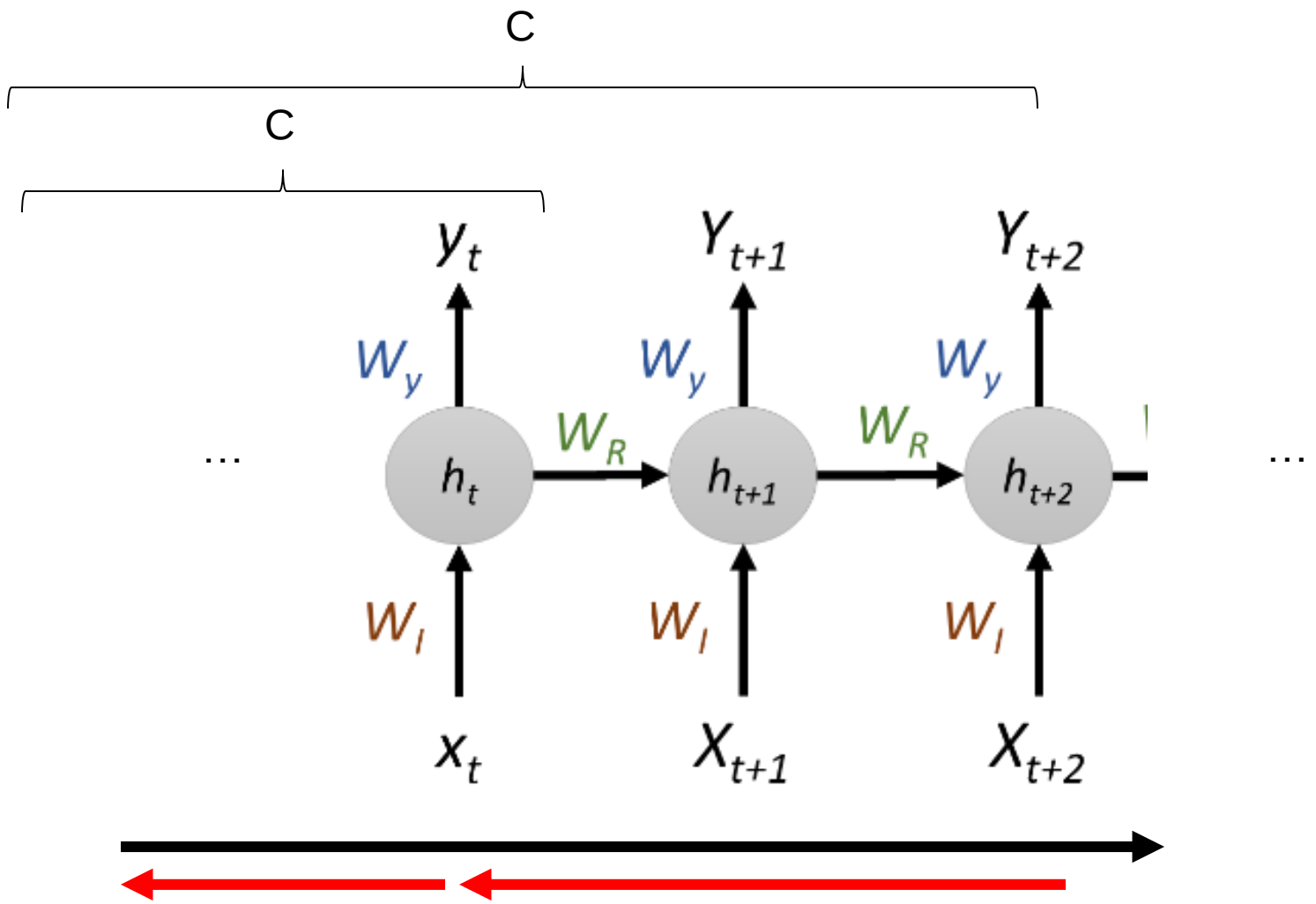
$$\frac{\partial C_T}{\partial W} \propto |W_R|^T \left|\frac{\partial g}{\partial a}\right|^T$$

Vanishing for W<1

Exploding for W>1



Truncated Backpropagation Through Time



Vanishing/exploring gradients

- Exploding gradients
  - Truncated BPTT
  - Clip gradients at threshold

- Vanishing gradients
  - Harder to detect
  - Weight initialization
  - ReLu activation functions
  - LSTM (Long-Short Term Memory)
  - GRUs (= Gated Recurrent Units)

Long Short-term Memory (LSTM)

- LSTM is like a conveyor belt
- Adds or remove information to
  - Forget: Keep state or forget state
  - Input: Which values to update?
  - Output: Which parts go out?

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

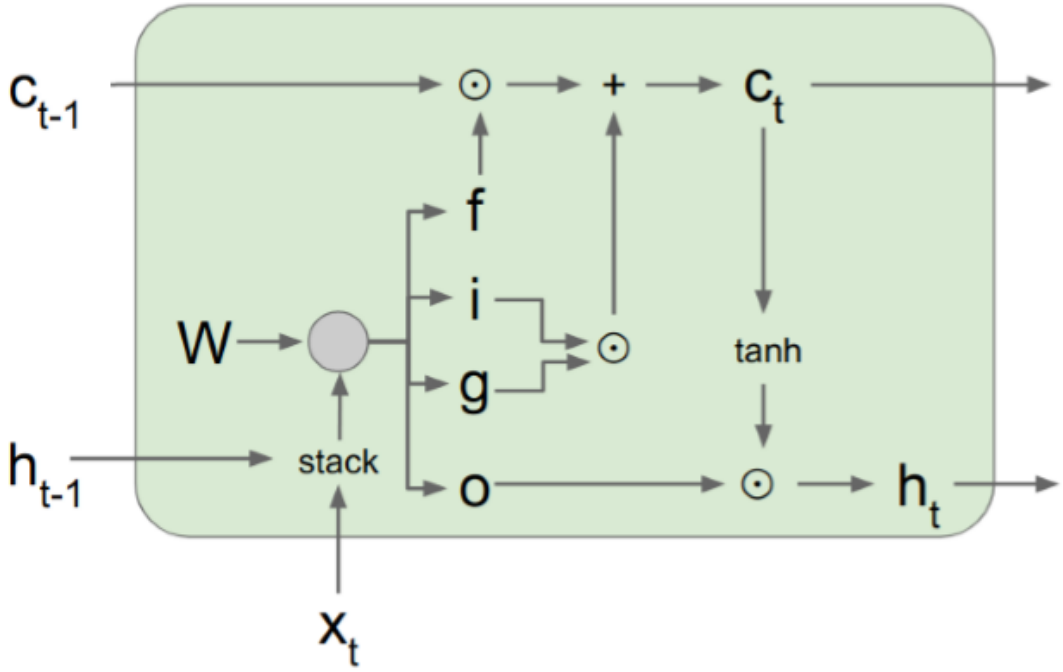
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

f: Forget Gate, i: Input Gate, g: Gate, o: Output Gate

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

The LSTM Cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

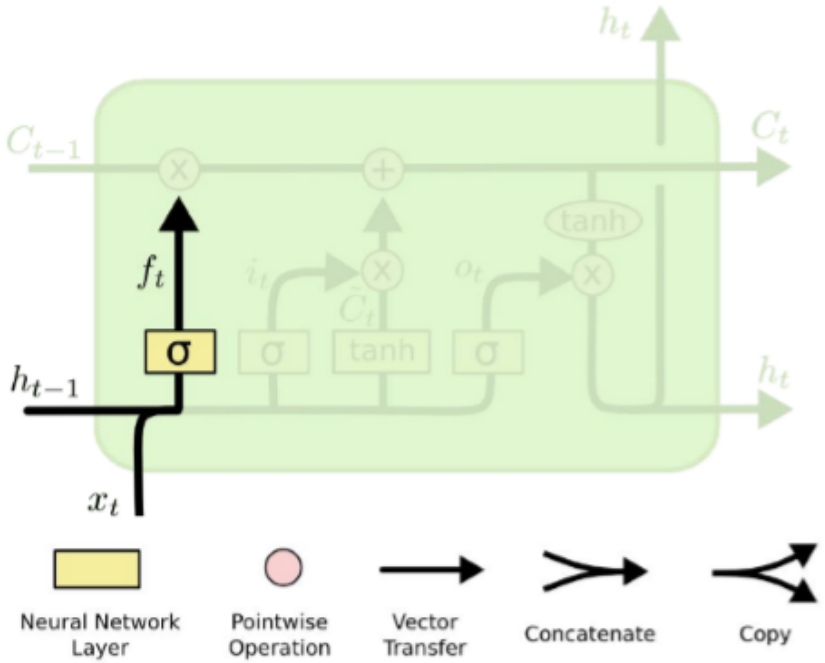
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

The LSTM Cell- Forget Gate

Forget Gate: ***e.g. forget the gender pronoun of previous subject sentence.***



Forget Gate:

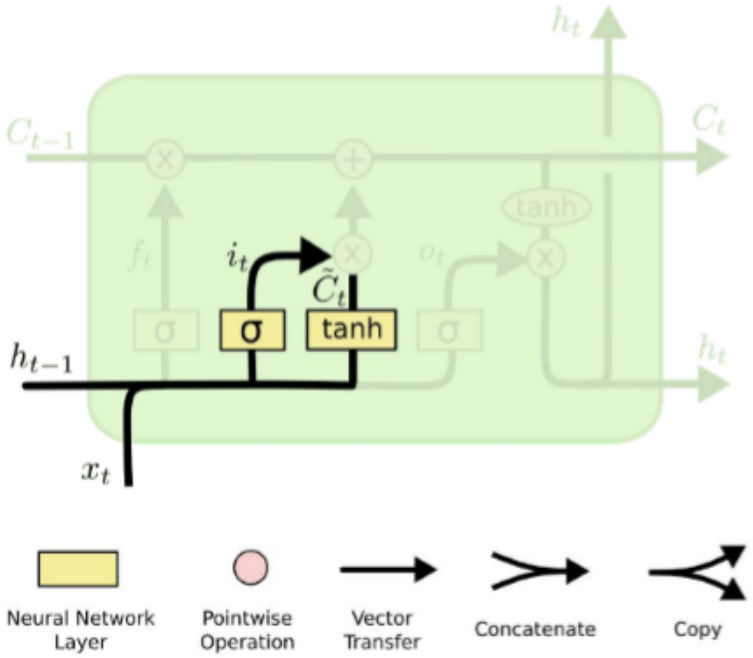
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Concatenate

Cristopher Olah, "Understanding LSTM Networks" (2015)

The LSTM Cell- Input Gate

Input Gate: ***e.g. add gender of new subject to replace that of old subject***



Input Gate Layer

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

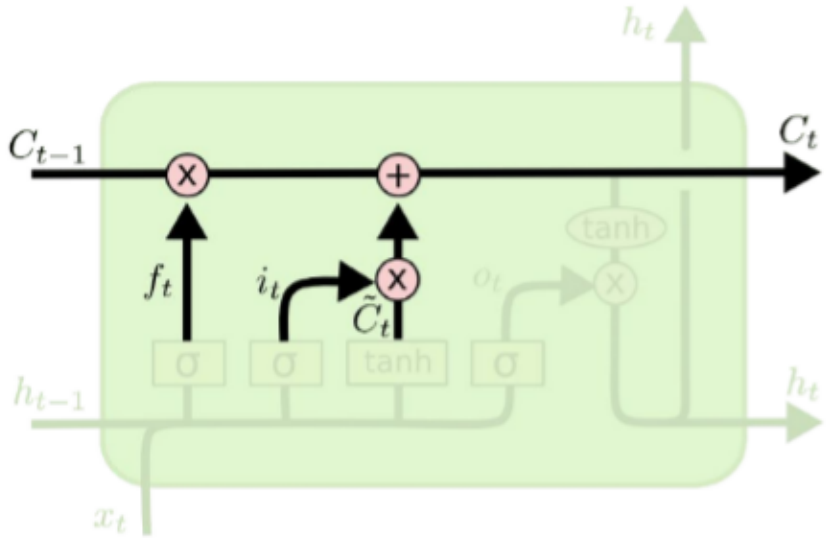
New contribution to cell state

$$\tilde{C}_t = \underbrace{\tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)}_{\text{Classic neuron}}$$

Cristopher Olah, "Understanding LSTM Networks" (2015)

The LSTM Cell- Update Gate

Update: ***drop old information and add new information about subject's gender.***



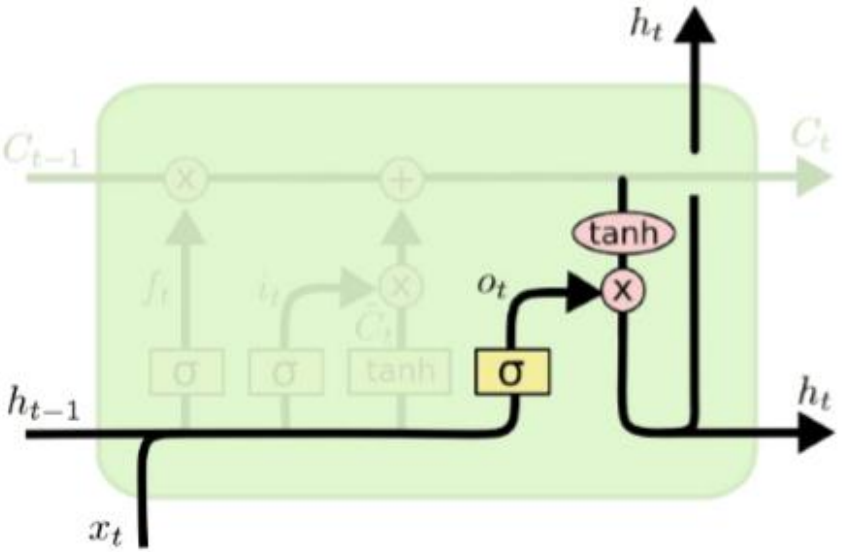
Update Cell State (memory):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Cristopher Olah, "Understanding LSTM Networks" (2015)

The LSTM Cell- Output Gate

Output: ***Having seen a subject, output may relates to a verb.***



Output Gate Layer

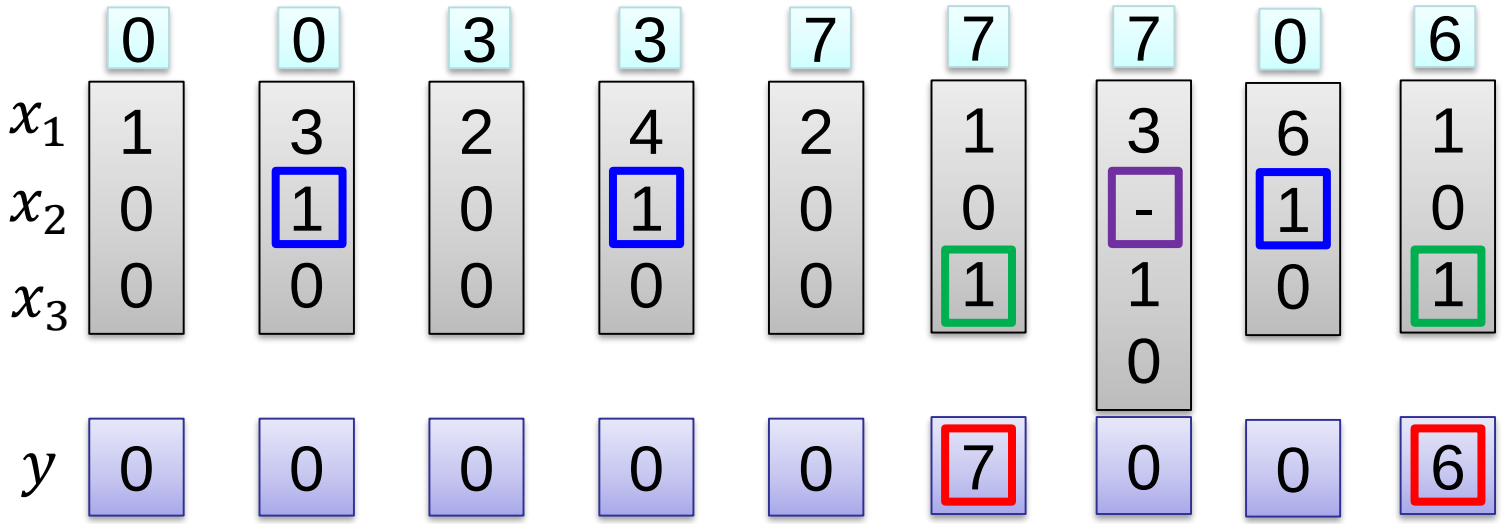
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

Output to next layer

$$h_t = o_t * \tanh\left(C_t\right)$$

Cristopher Olah, "Understanding LSTM Networks" (2015)

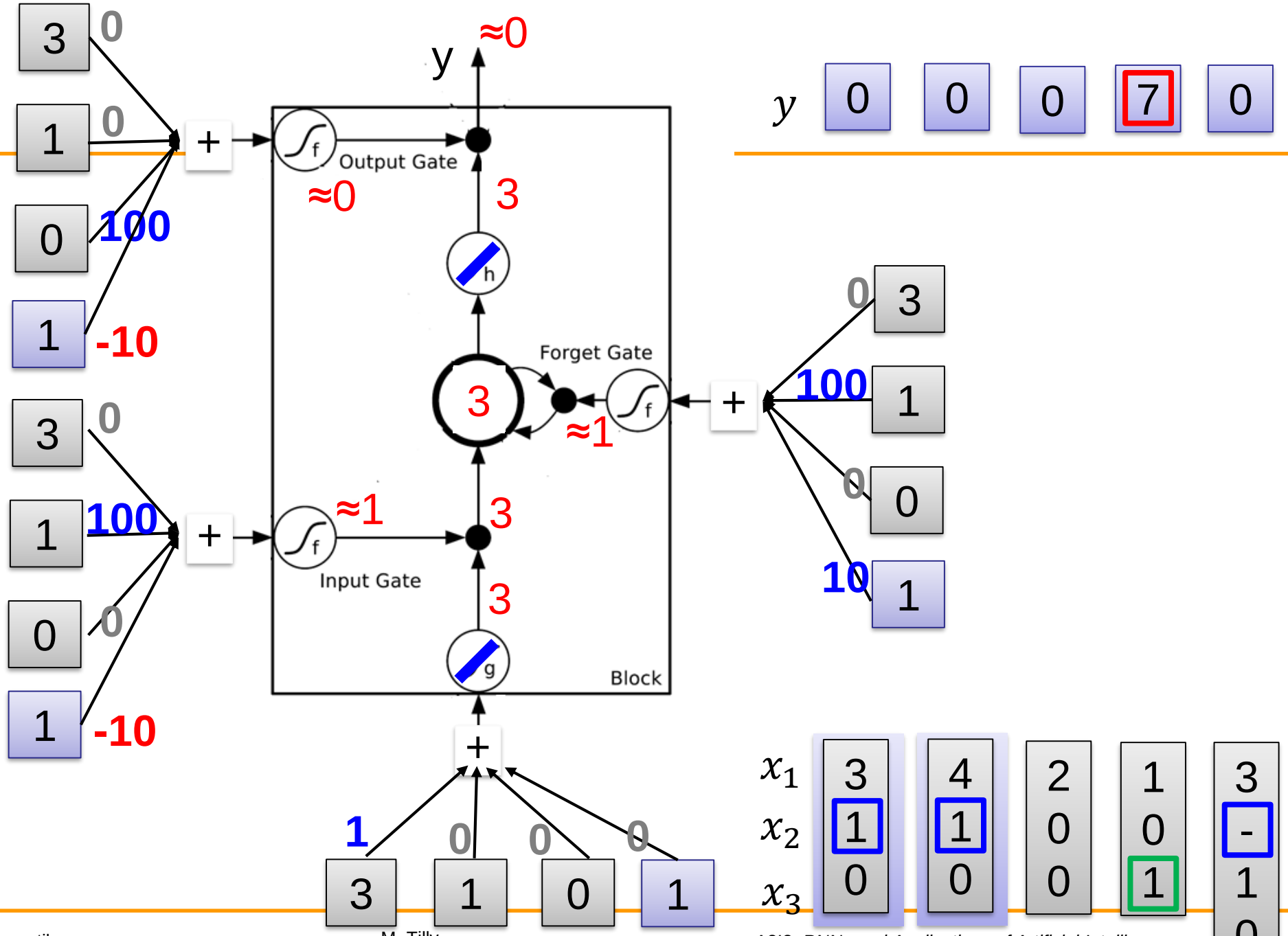
LSTM - Example



If $x_2 = 1$, add the numbers of x_1 into the memory (input gate)

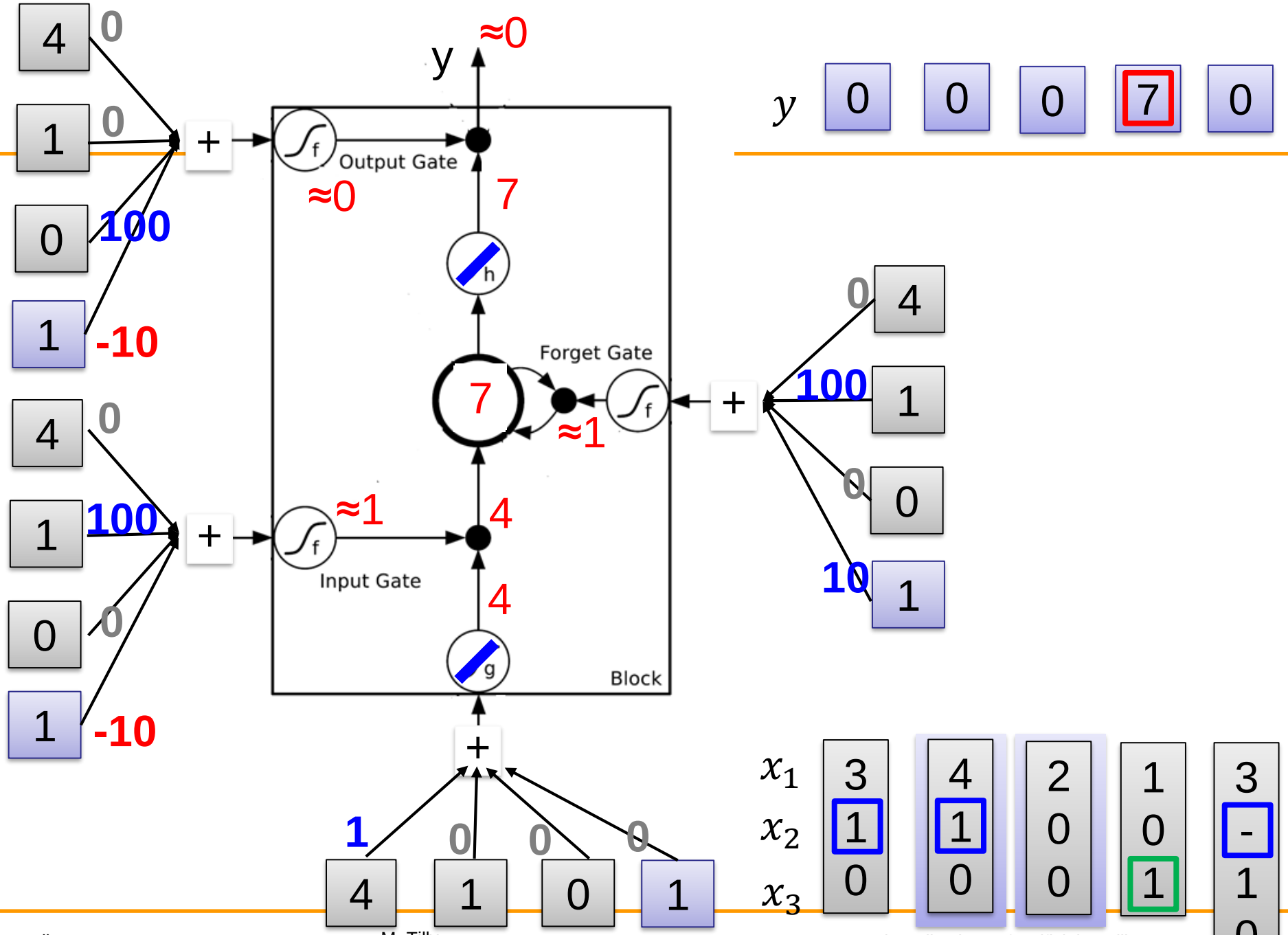
If $x_2 = -1$, reset the memory (forgot gate)

If $x_3 = 1$, output the number from memory (output gate)



y 0 0 0 7 0

x_1	3	4	2	1	3
x_2	1	1	0	0	-
x_3	0	0	0	1	1
					0

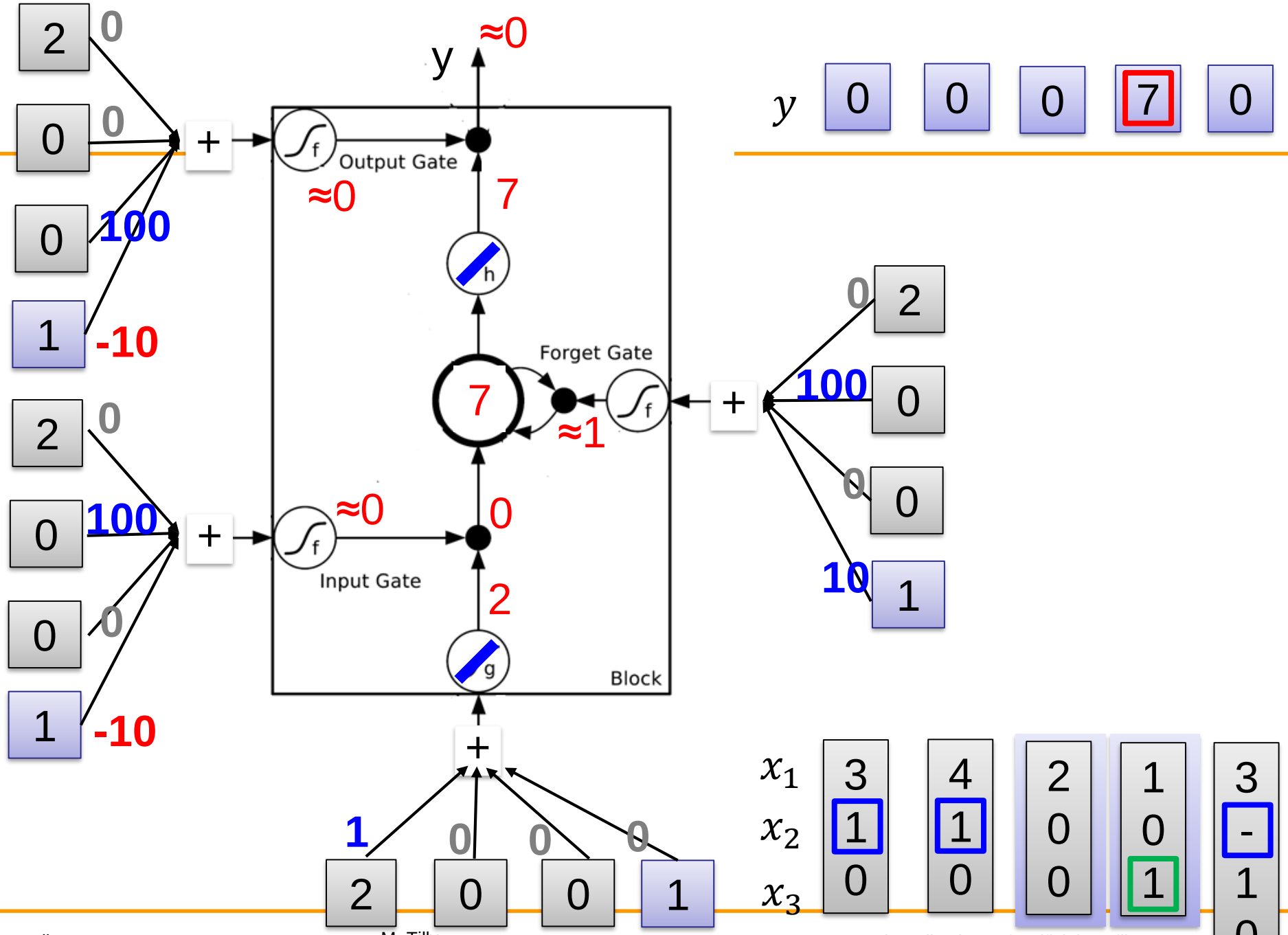


y 0 0 0 7 0

x_1 3 4 2 1 3

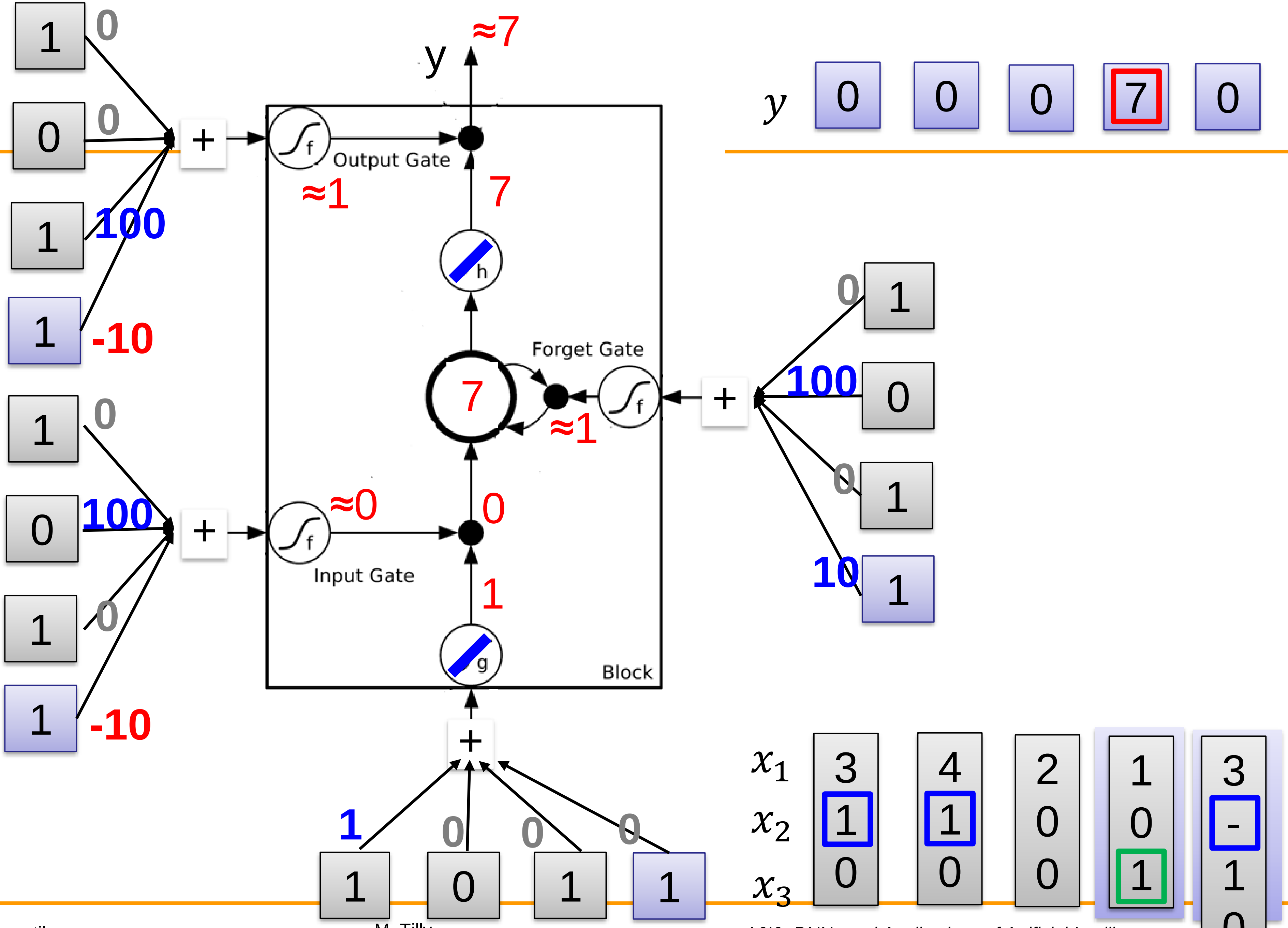
x_2 1 1 0 0 -

x_3 0 0 0 1 0

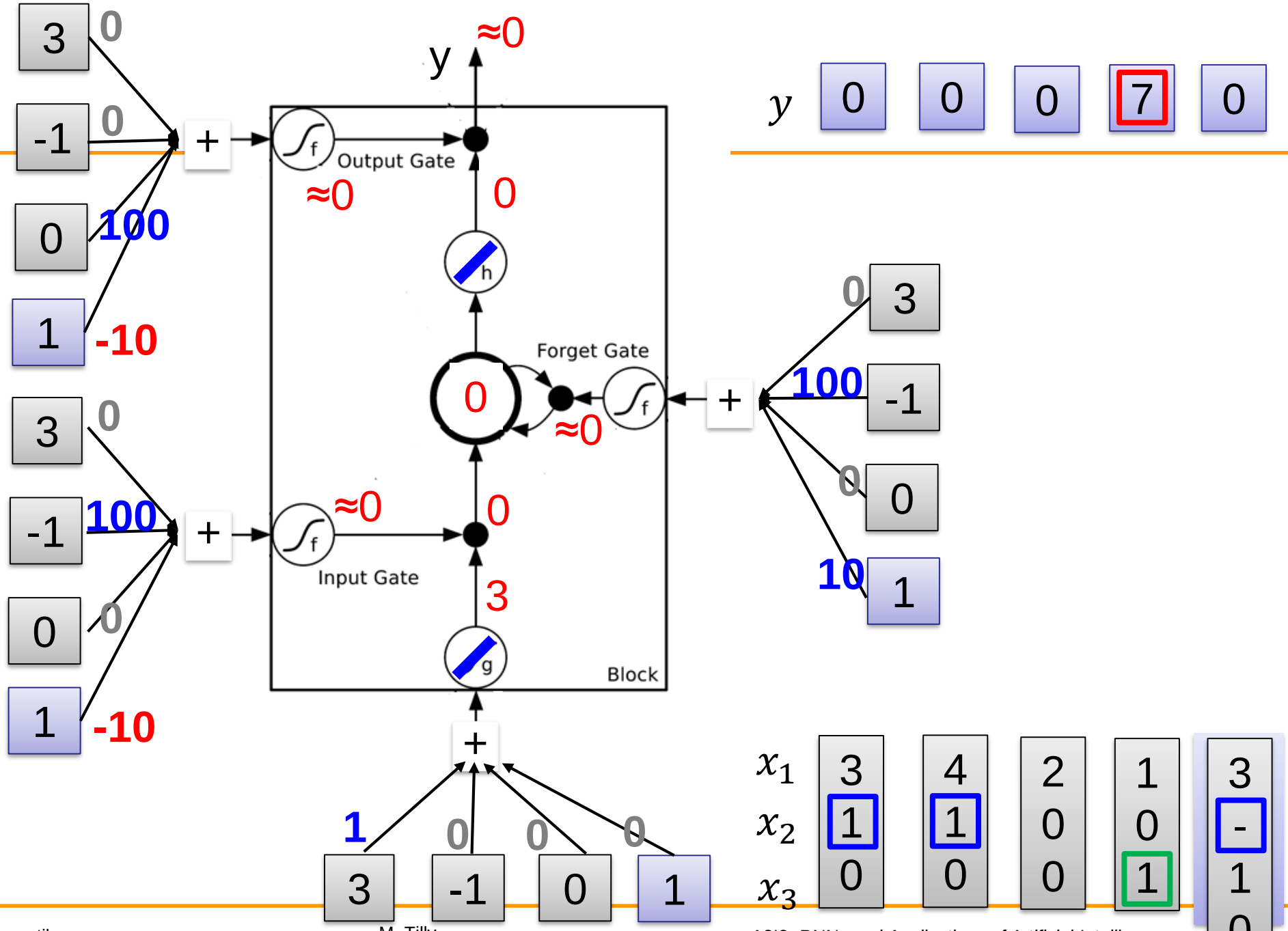


y	0	0	0	7	0

x_1	3	4	2	1	3
x_2	1	1	0	0	-
x_3	0	0	0	1	1
					0



y
≈7
Output Gate
≈1
7
h
Forget Gate
7
≈1
≈0
0
Input Gate
1
g
Block
+
0
0
100
-10
100
10
1
y
0 0 0 7 0
x_1
x_2
x_3
3 1 0
4 1 0
2 0 0
1 0 1
3 - 1 0

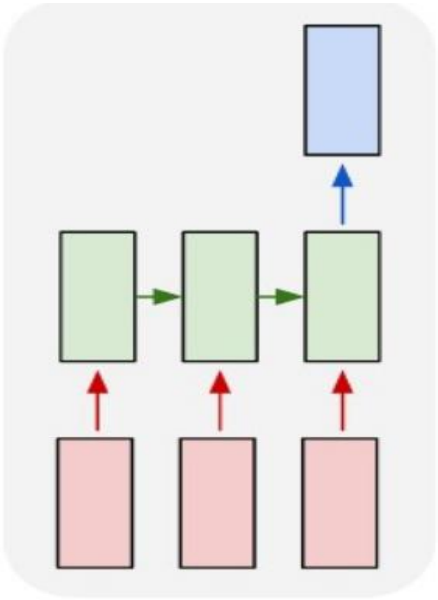


y	0	0	0	7	0

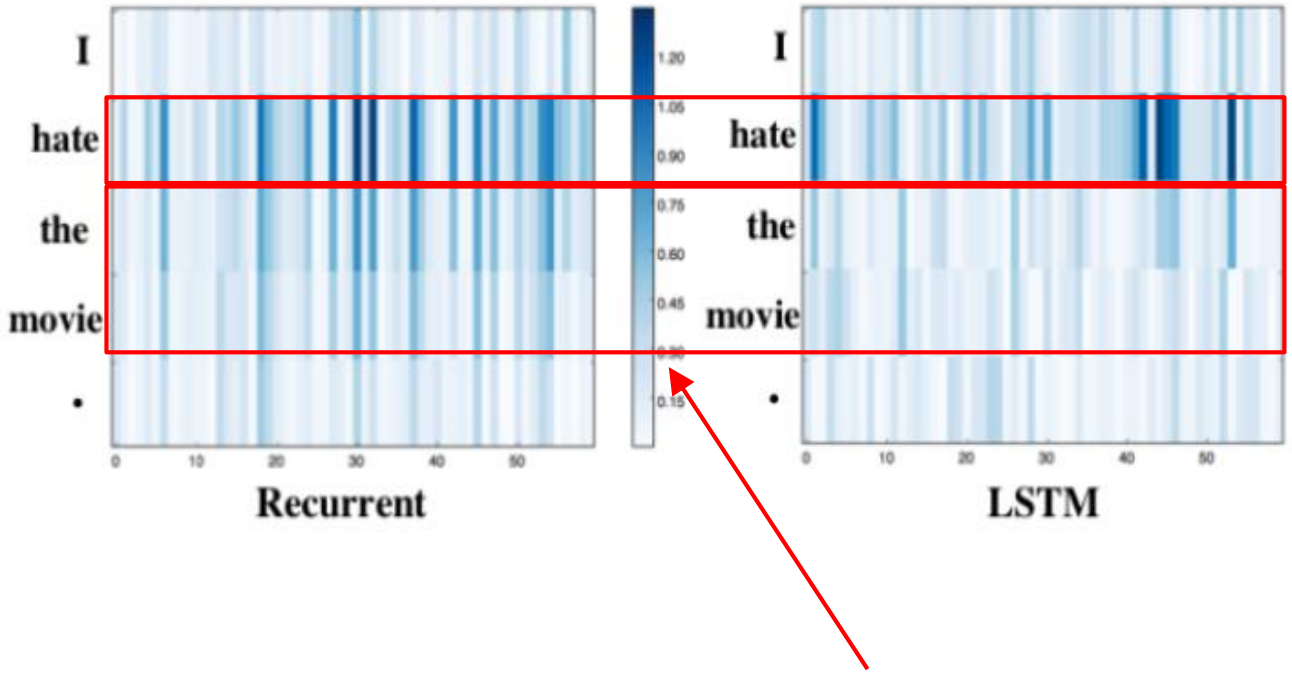
x_1	3	4	2	1	3
x_2	1	1	0	0	-
x_3	0	0	0	1	1
					0

Long Term Dependencies with LSTM

Sentiment Analysis



Many-one network

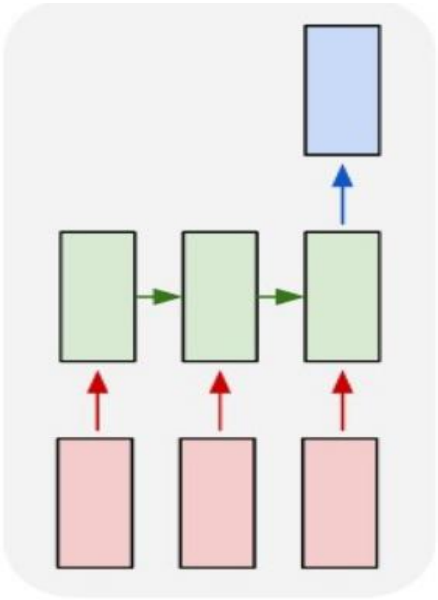


Recent words more salient

"Jiwei LI et al, "Visualizing and Understanding Neural Models in NLP"

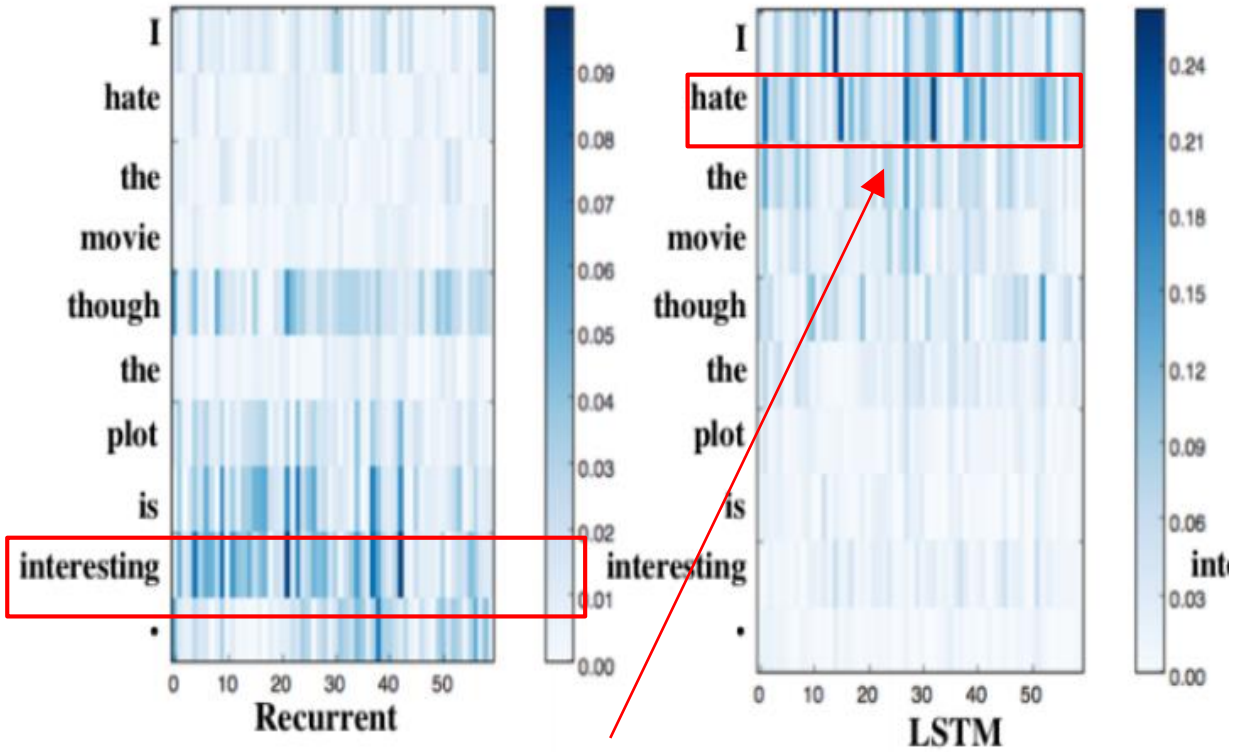
Long Term Dependencies with LSTM

Sentiment Analysis



Many-one network

Saliency Heatmap

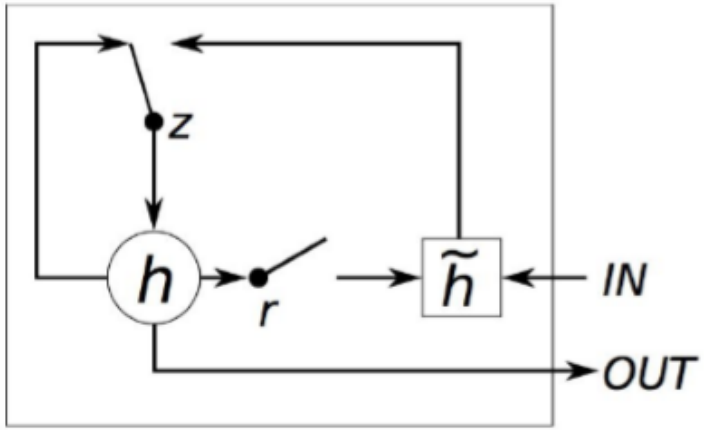


LSTM captures long term dependencies

"Jiwei LI et al, "Visualizing and Understanding Neural Models in NLP"

Gated Recurrent Unit (GRUs)

Similar performance as LSTM with less computation.



$$u_i = \sigma\left(W^{(u)}x_i + U^{(u)}h_{i-1} + b^{(u)}\right) \quad (1)$$

$$r_i = \sigma\left(W^{(r)}x_i + U^{(r)}h_{i-1} + b^{(r)}\right) \quad (2)$$

$$\tilde{h}_i = \tanh\left(Wx_i + r_i \circ Uh_{i-1} + b^{(h)}\right) \quad (3)$$

$$h_i = u_i \circ \tilde{h}_i + (1 - u_i) \circ h_{i-1} \quad (4)$$

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder forstatistical machine translation." arXiv preprint arXiv:1406.1078 (2014)

Comparing GRU and LSTM

- Both **GRU and LSTM better than RNN with tanh** on music and speech modeling
- GRU performs comparably to LSTM
- No clear consensus between GRU and LSTM

Source: Empirical evaluation of GRUs on sequence modeling, 2014

Wrap-up

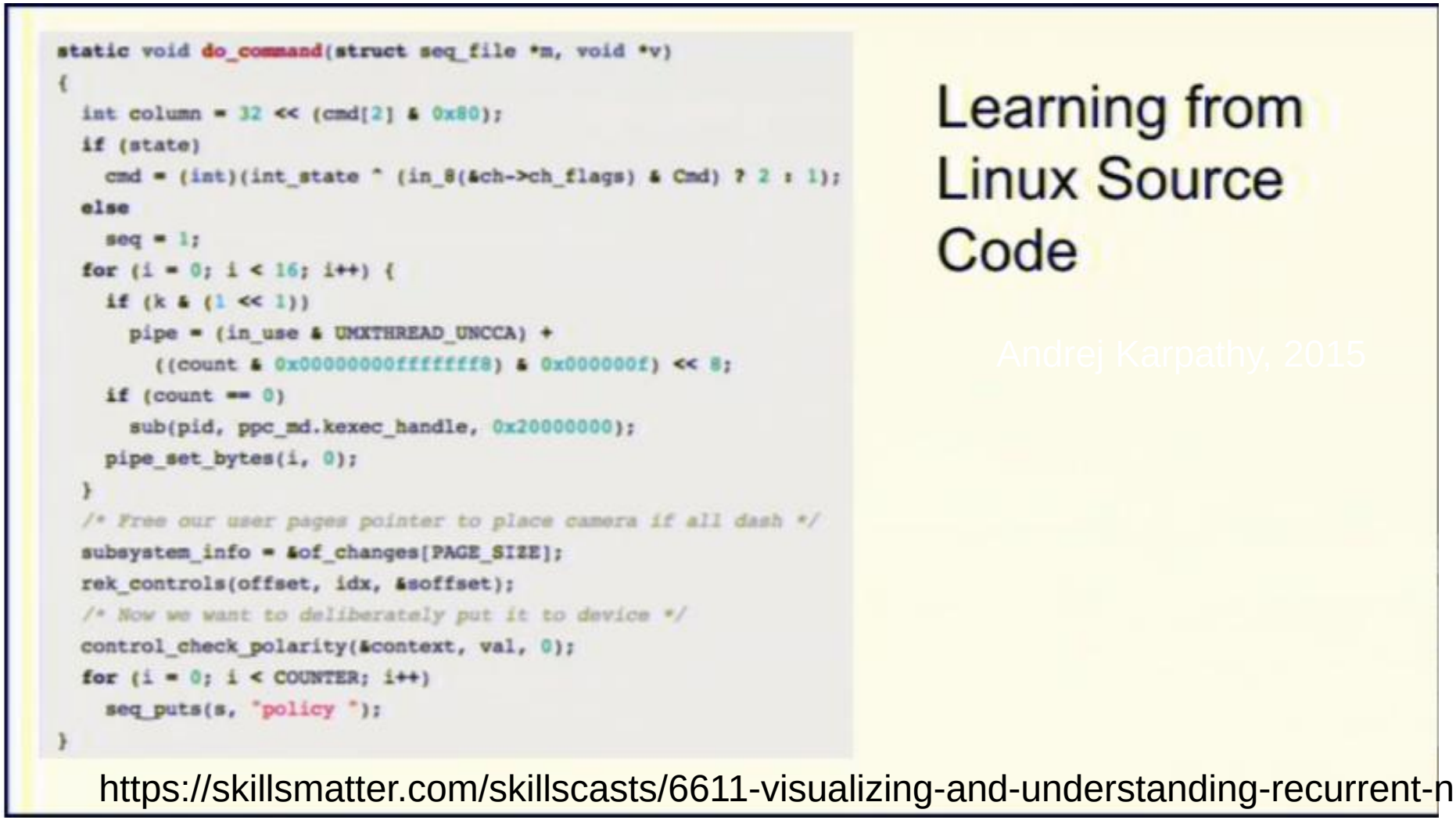
- RNNs - self connected networks
  - Text, speech, audio, video, …., anything embedded in time (almost everything)
  - Allows us to measure how likely a sentence is
  - Important input for Machine Translation
  - Can generate new text!!!!!
- RNN were introduced in the late 80's
- Hochreiter et. al. discovers the 'vanishing gradients' problem in 1991
  - Long Short Term Memory (LSTM) published in 1997
- LSTM - solves the vanishing gradient and the long memory limitation problem

Additional Material

- RNNs/ LSTMs with Keras: https://www.tensorow.org/guide/keras/rnn
- Recurrent neural networks and LSTM tutorial in Python and TensorFlow: https://adventuresinmachinelearning.com/recurrent-neural-networks-lstmtutorial-tensorow/
- An Intro Tutorial for Implementing Long Short-Term Memory Networks (LSTM): https://heartbeat.fritz.ai/a-beginners-guide-to-implementing-long-short-termmemory-networks-lstm-eb7a2ff09a27
- TimeSeries Forecast: https://medium.com/datadriveninvestor/multivariate-timeseries-using-rnn-with-keras-7f78f4488679

What can I build from all this AI things?

Create new Content



https://skillsmatter.com/skillscasts/6611-visualizing-and-understanding-recurrent-networks

MuseNet

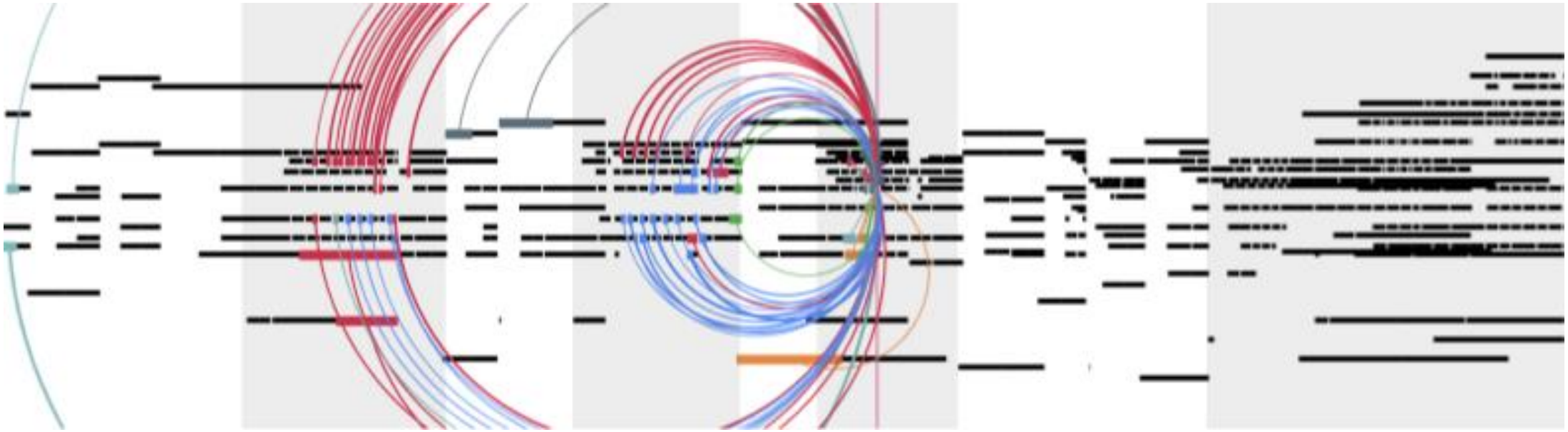
MuseNet

We've created MuseNet, a deep neural network that can generate 4-minute musical compositions with 10 different instruments, and can combine styles from country to Mozart to the Beatles. MuseNet was not explicitly programmed with our understanding of music, but instead discovered patterns of harmony, rhythm, and style by learning to predict the next token in hundreds of thousands of MIDI files. MuseNet uses the same general-purpose unsupervised technology as GPT-2, a large-scale transformer model trained to predict the next token in a sequence, whether audio or text.

https://openai.com/blog/musenet/

Music Transformer: Generating Music with Long-Term Structure

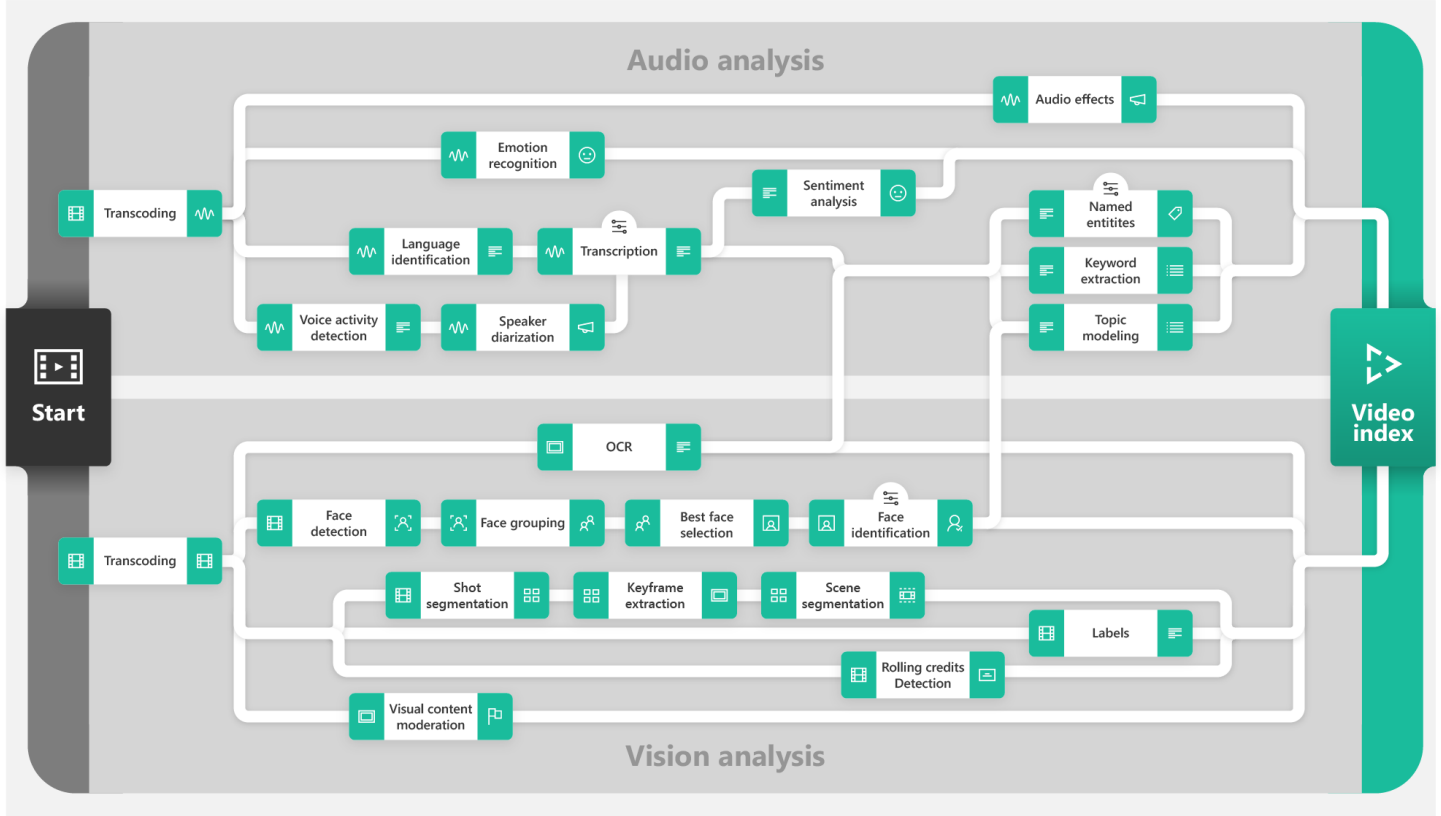
https://magenta.tensorflow.org/music-transformer



Seeing AI

https://www.microsoft.com/en-us/ai/seeing-ai

VideoIndexer.ai

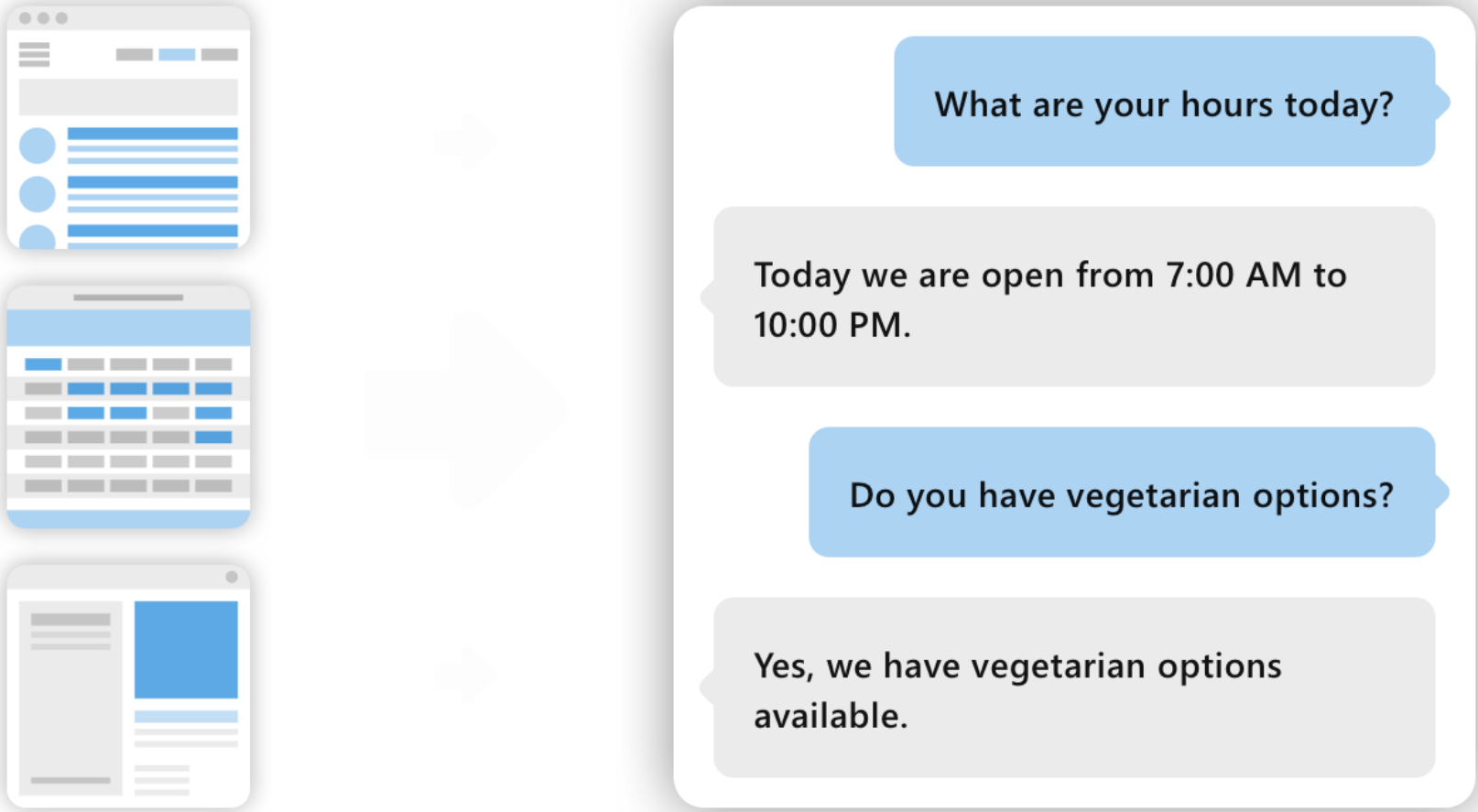


WebSite: https://www.videoindexer.ai/

Tutorial: https://docs.microsoft.com/en-us/azure/media-services/video-indexer/video-indexer-use-apis

QnA Maker

- https://azure.microsoft.com/en-us/services/cognitive-services/qna-maker/



Summary

- RNNs and LSTMs for sequence modelling
- Applications
  - Seeing AI
  - Video Indexer
  - QnA Maker

References

- Hochreiter, S., & Schmidhuber, J. (1997), „*Long short-term memory. Neural computation*“, 9(8), 1735-1780
- Gers, F . A., Schmidhuber, J., & Cummins, F . (2000), „*Learning to forget: Continual prediction with LSTM*“, Neural computation, 12(10), 2451-2471
- Graves, A. (2012), „*Supervised sequence labelling with recurrent neural networks*“ (Vol. 385). Springer
- Hochreiter, S., Bengio, Y., Frasconi, P ., & Schmidhuber, J. (2001), „*Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*“
- Bengio, Y., Simard, P ., & Frasconi, P . (1994), „*Learning long-term dependencies with gradient descent is difficult*“, Neural Networks, IEEE Transactions on, 5(2), 157166.
- Maas, A. L., Daly, R. E., Pham, P . T., Huang, D., Ng, A. Y., & Potts, C. (2011, June), „*Learning word vectors for sentiment analysis*“, In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1 (pp. 142-150). Association for Computational Linguistics.