Notes for Networks of Memory

TANG Zhiyun, CSLT

LSTM(1)

-- decaying error backflow

-- gradient based

-- constant error flow through constant error carousels

-- as for rnn, short-term memory, store representations of recent input events in the form of activations; long-term, embodied by slowly changing weights

-- traditional rnn, error signals explode or vanish exponentially depending on the weights

-- previous work to bridge long time lags

LSTM(1)

-- exponentially decaying error, analysis

//

-- input gate to protect the memory contents from perturbation by irrelevant inputs

-- output gate to protect other units from perturbation by currently irrelevant memory contents

LSTM(1)

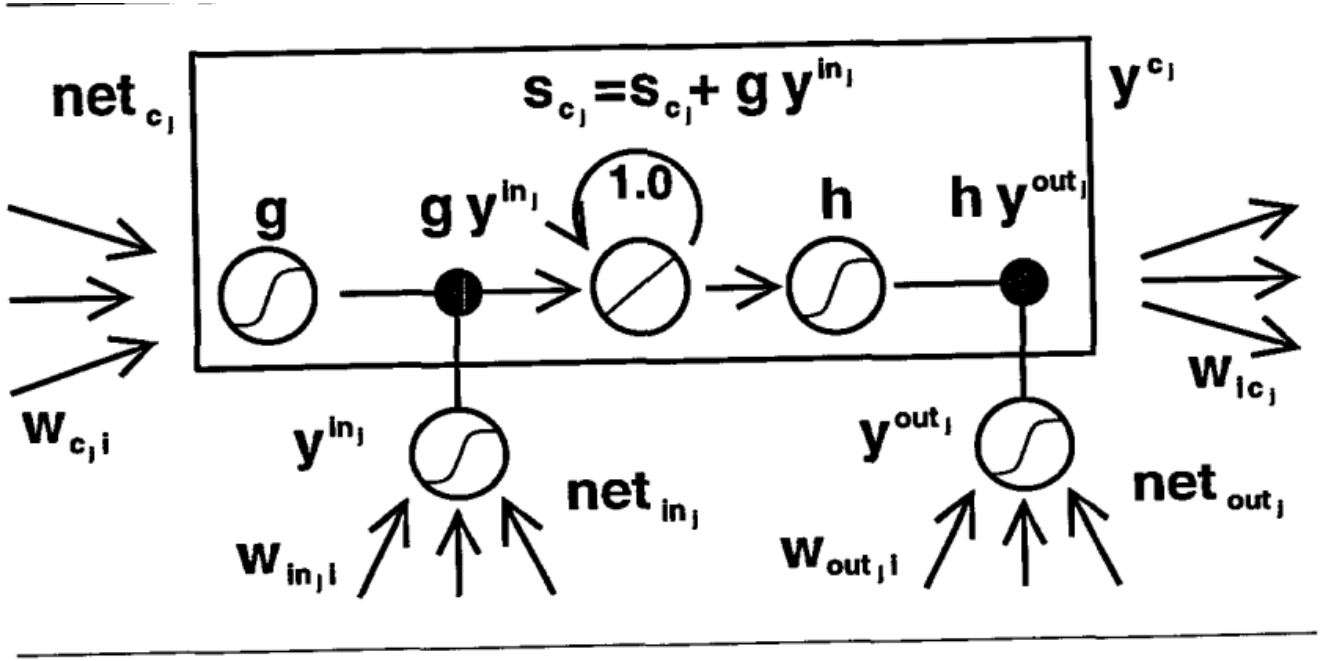


Figure 1: Architecture of memory cell c_j (the box) and its gate units in_j, out_j. The self-recurrent connection (with weight 1.0) indicates feedback with a delay of one time step. It builds the basis of the CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.

LSTM(1)

-- input gate(output gate) may use inputs form other memory cells to decide whether to store (access) certain information in its memory cell

-- (why gate units) To avoid input weight conflicts, input weight controls the error flow to memory cell's input connections. To circumvent cell's output weight conflicts, output gate controls the error flow from output connections.(BP) That is, the net can use input gate to decide when to keep or override information in memory cell and output gate to decide when to access memory cell and when to prevent other units from being perturbed.(Propagate)

-- by scaling the errors, open/close access to constant error flow

LSTM(1)

-- out gate sometimes prevent the net’s attempts at storing long-time-lag memories(hard to learn) from perturbing activations representing easily learnable short-time-lag memories

-- may abuse cell

-- tasks to demonstrate the quality of a novel long-time-lag algorithm: minimal time lags; complex enough

LSTM(2)

-- instantiate dynamics

-- LSTM(1) not for very long or continual time series that are not a prior segmented, causing the internal values of the cells to grow without bound

-- any training procedure for rnns which is powerful enough to span long time lags must also address the issue of forgetting in short term memory(unit activations)

-- forgetting may occur rhythmically or in an input-dependent fashion

-- when in/out are closed(activation around zero), irrelevant inputs and noise don not enter the cell , and the cell state does not perturb the remainder of the network

LSTM(2)

-- LSTM(1) may have saturation of output squashing function, cell states are explicitly reset to zero, not “forever” memory

-- adaptive “forget gates”

LSTM(2)

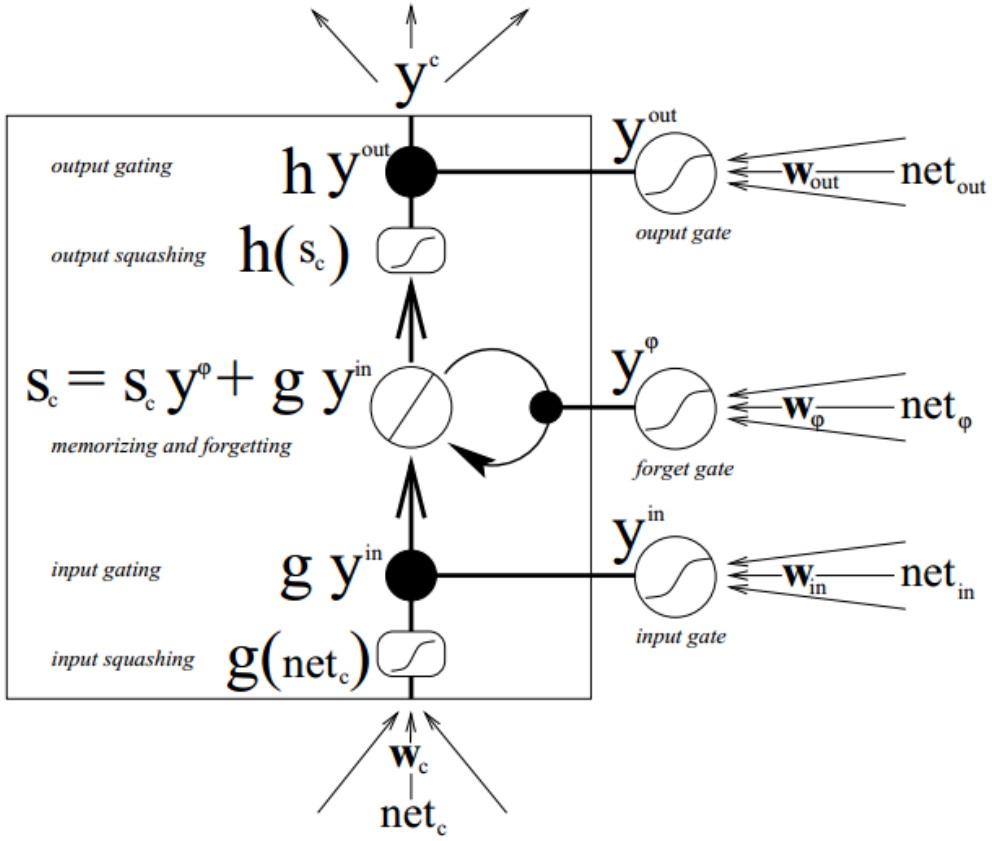


Figure 2: Memory block with only one cell for the extended LSTM. A multiplicative forget gate can reset the cell's inner state s_c.

LSTM(2)

-- the forget gate's activation ranges between 0 and 1, "gradually restes"

-- bias weights for gates are initialized with negative values for input and output gates positive for forget. Implies in the beginning, forget gate activation will be almost 1.0 and the entire cell will be behave like a standard LSTM cell. Forget anything until it has learned to forget.

-- backward pass, fusion of bp through time(BPTT) and real time recurrent learning(RTRL)

LSTM(3)

-- cell provides short-term memory storage for extended time periods

-- input, forget and output gate can be trained to learn, respectively, what information to store in the memory ,how long to store it and when to read it out.

-- each gate receives connections from the input units and the outputs of all cells, no direct connection from the CEC it is supposed to control

LSTM(3)

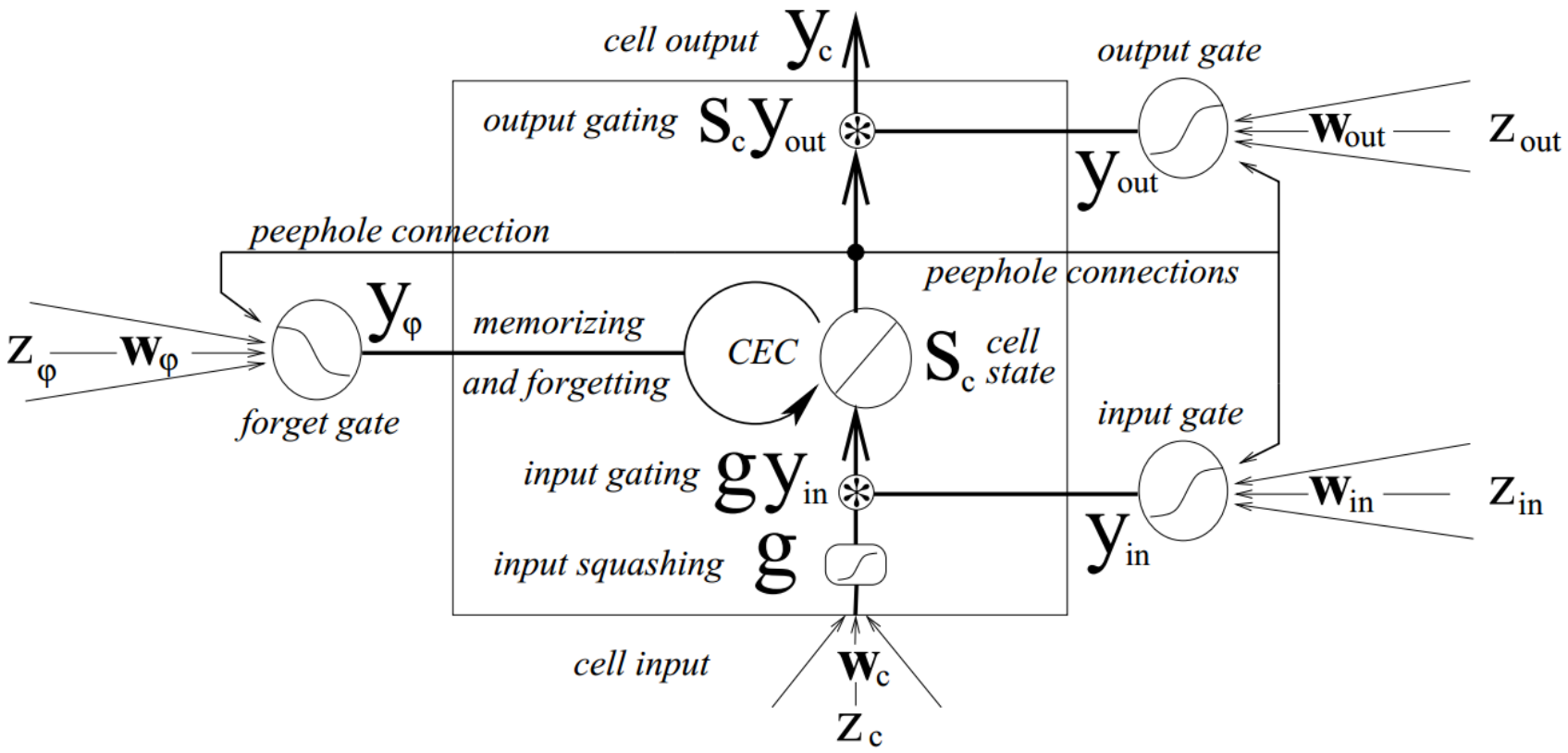


Figure 2: LSTM memory block with peephole connections from the CEC to the gates.

LSTM(4)

$$i_t = \sigma(W_{ix}x_t + W_{ir}r_{t-1} + W_{ic}c_{t-1} + b_i)$$
$$f_t = \sigma(W_{fx}x_t + W_{rf}r_{t-1} + W_{cf}c_{t-1} + b_f)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cr}r_{t-1} + b_c)$$
$$o_t = \sigma(W_{ox}x_t + W_{or}r_{t-1} + W_{oc}c_t + b_o)$$
$$m_t = o_t \odot h(c_t)$$
$$r_t = W_{rm}m_t$$
$$p_t = W_{pm}m_t$$
$$y_t = W_{yr}r_t + W_{yp}p_t + b_y$$

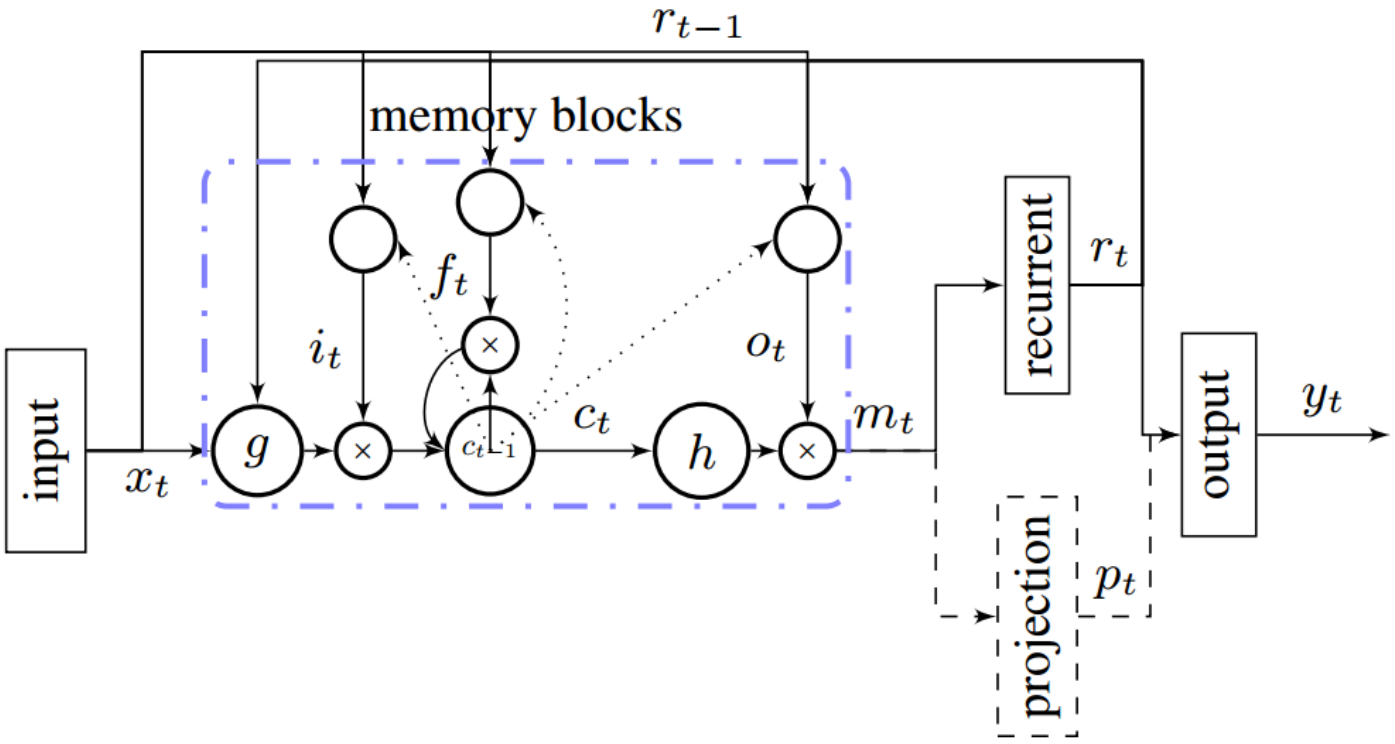


Fig. 1. LSTM based RNN architectures with a recurrent projection layer and an optional non-recurrent projection layer. A single memory block is shown for clarity.

Encoder-decoder(1)

-- two rnns, one encodes a sequence of symbols into a fixed-length vector representation and the other decodes the representation into another sequence of symbols.

-- jointly trained to maximize the conditional probability of a target sequence given a source sequence.

-- rnn to predict the next symbol in a sequence, to compute the probabililty of the sequence, the sample a new sequence by iteratively sampling a symbol at each time step

Encoder-decoder(1)

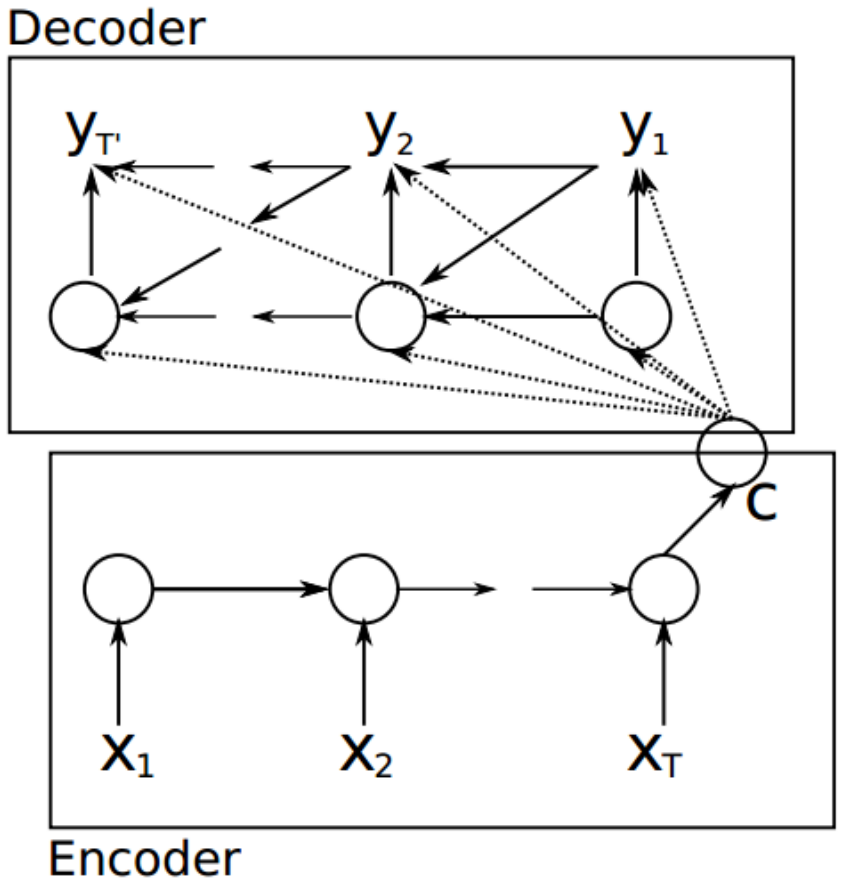


Figure 1: An illustration of the proposed RNN Encoder–Decoder.

Encoder-decoder(2)

-- a fixed-length context vector is problematic for long sentences

-- sequence to sequence, automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly
-- with attention

Encoder-decoder(2)

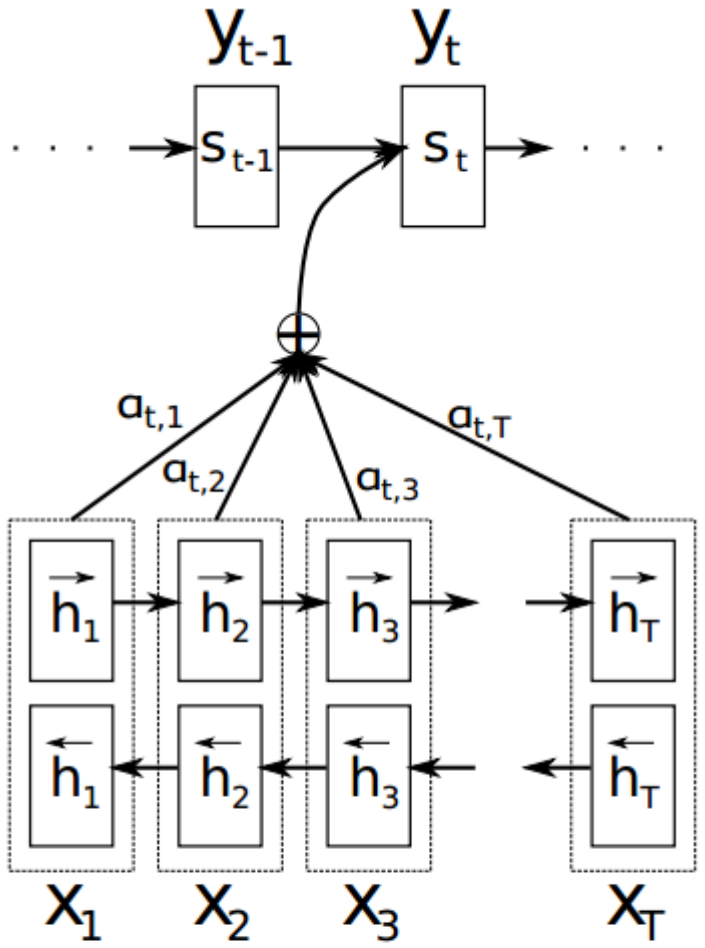


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Attention based(1)

-- maximize the likelihood of the target description sentence given the training image

-- not attend

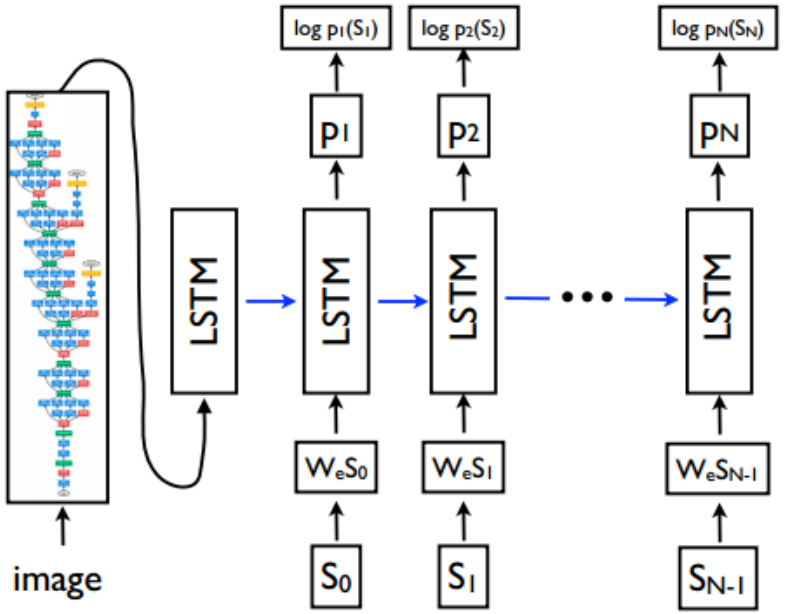
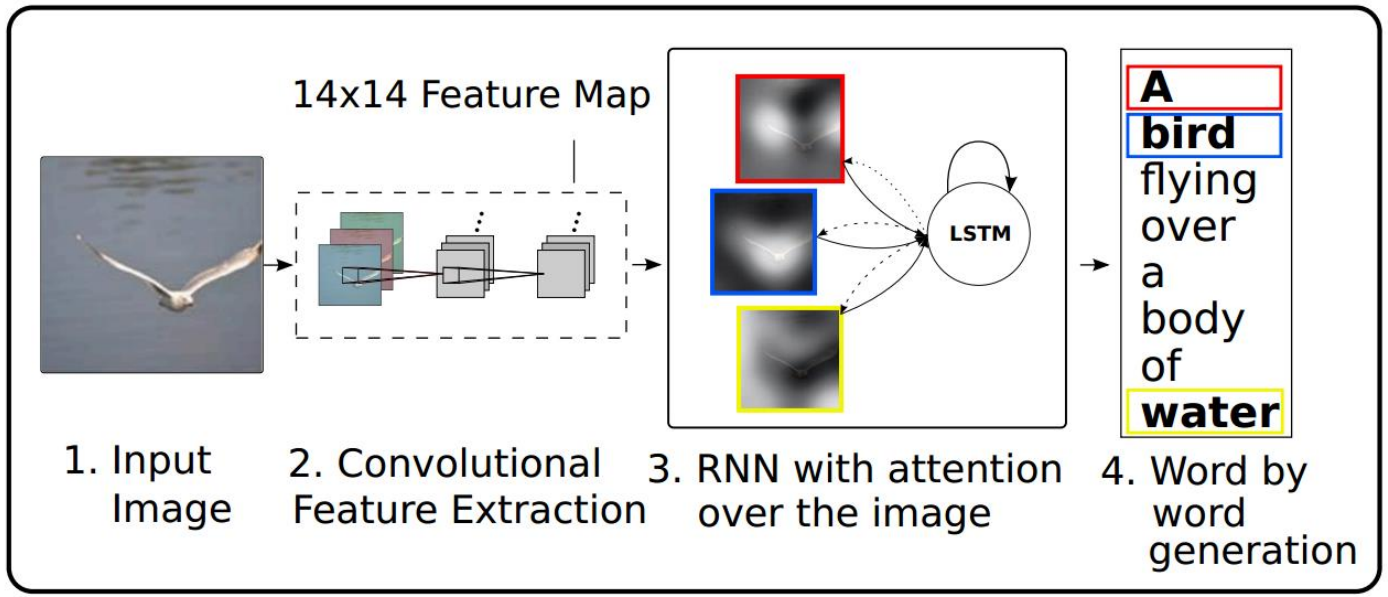


Figure 3. LSTM model combined with a CNN image embedder (as defined in [12]) and word embeddings. The unrolled connections between the LSTM memories are in blue and they correspond to the recurrent connections in Figure 2. All LSTMs share the same parameters.

Attention based(2)

-- cnn to extract a set of feature vectors which we refer to as annotation vectors

Figure 1. Our model learns a words/image alignment. The visualized attentional maps (3) are explained in section 3.1 & 5.4



Neural Turing Machine

-- computer program: elementary operations, logical flow control, external memory (write to and read from).

-- rnn is turing complete

-- controller, memory bank

-- read: memory matrix, with a vector of weightings over the locations

-- write: taking inspiration from input and forget gates in LSTM, an erase followed by an add

-- controller as registers in the processor

-- dnn controller mimics rnn

Neural Turing Machine

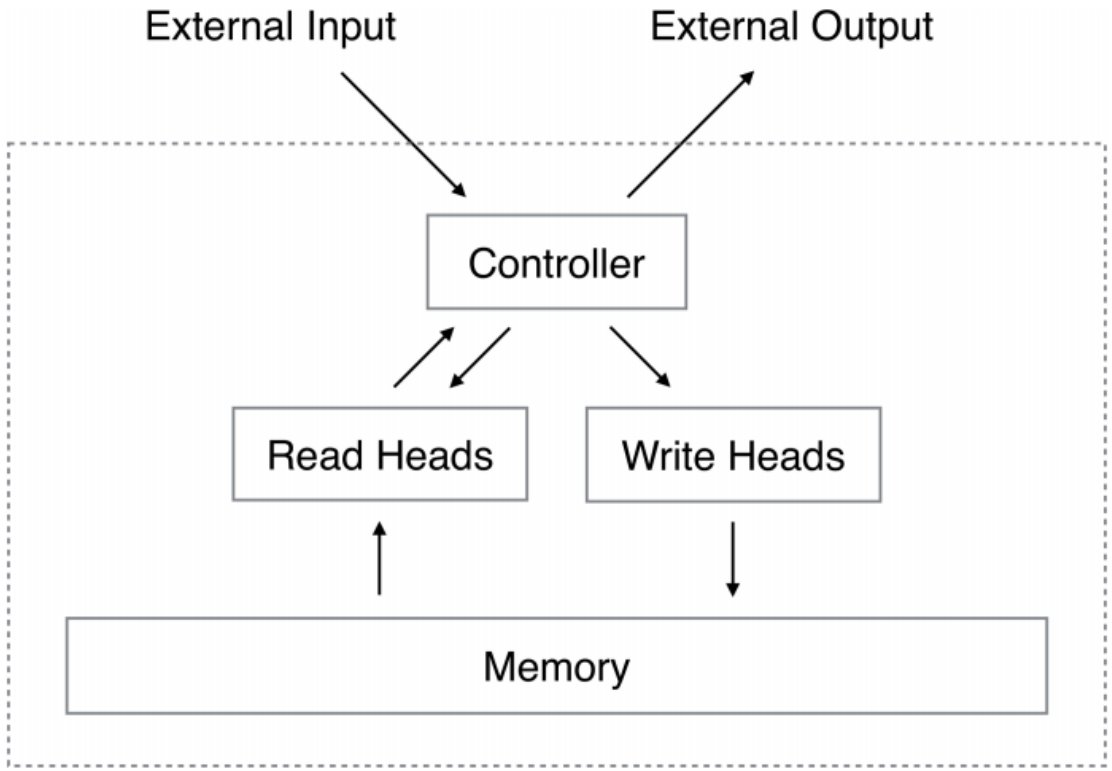


Figure 1: Neural Turing Machine Architecture. During each update cycle, the controller network receives inputs from an external environment and emits outputs in response. It also reads to and writes from a memory matrix via a set of parallel read and write heads. The dashed line indicates the division between the NTM circuit and the outside world.

Neural Turing Machine

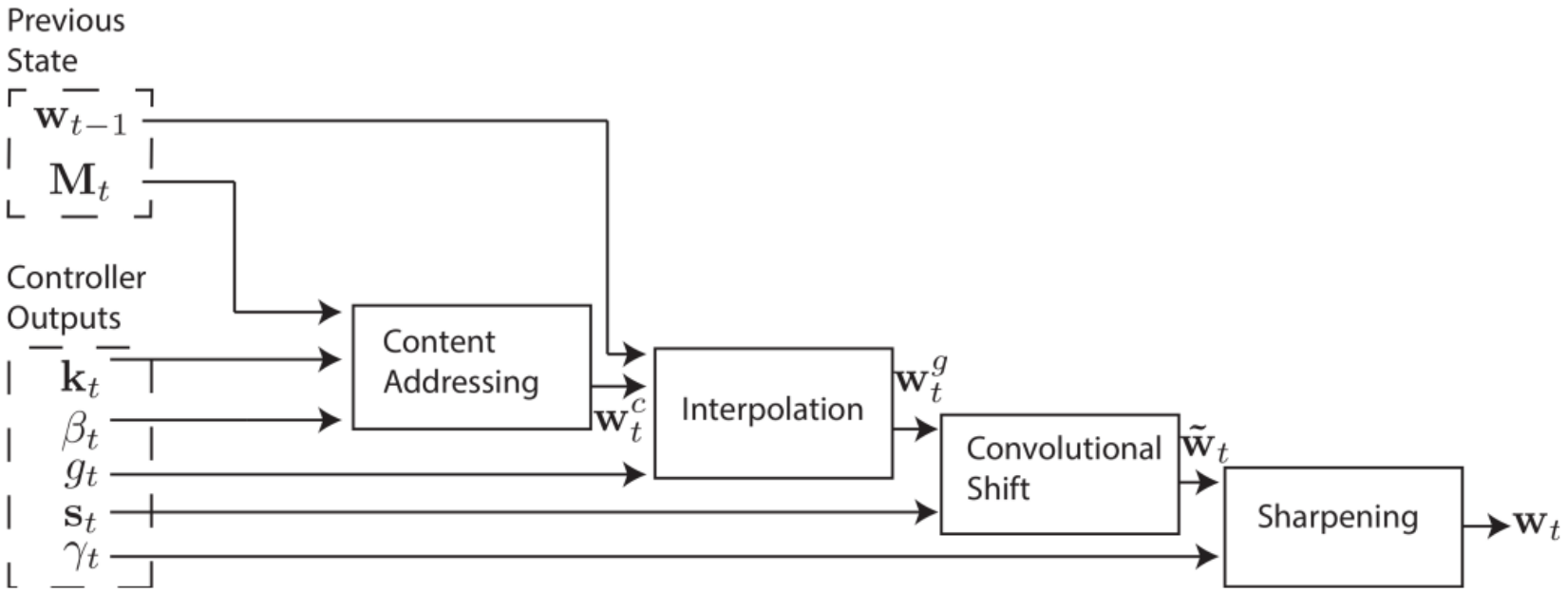


Figure 2: Flow Diagram of the Addressing Mechanism. The *key vector*, $\mathbf{k}_t$, and *key strength*, β_t, are used to perform content-based addressing of the memory matrix, $\mathbf{M}_t$. The resulting content-based weighting is interpolated with the weighting from the previous time step based on the value of the *interpolation gate*, g_t. The *shift weighting*, $\mathbf{s}_t$, determines whether and by how much the weighting is rotated. Finally, depending on γ_t, the weighting is sharpened and used for memory access.

Memory Network

-- I: input feature map

-- G: generalization

-- O: output feature map

-- R: response

-- they can potentially use any existing ideas from ML literature, e.g., SUM, dnn, decision tree.

Memory Network

Given an input x (e.g., an input character, word or sentence depending on the granularity chosen, an image or an audio signal) the flow of the model is as follows:

1. Convert x to an internal feature representation $I(x)$.
2. Update memories $\mathbf{m}_i$ given the new input: $\mathbf{m}_i = G(\mathbf{m}_i, I(x), \mathbf{m}), \ \forall i.$
3. Compute output features o given the new input and the memory: $o = O(I(x), \mathbf{m})$.
4. Finally, decode output features o to give the final response: $r = R(o)$.