Recurrent Neural Networks (RNNs)

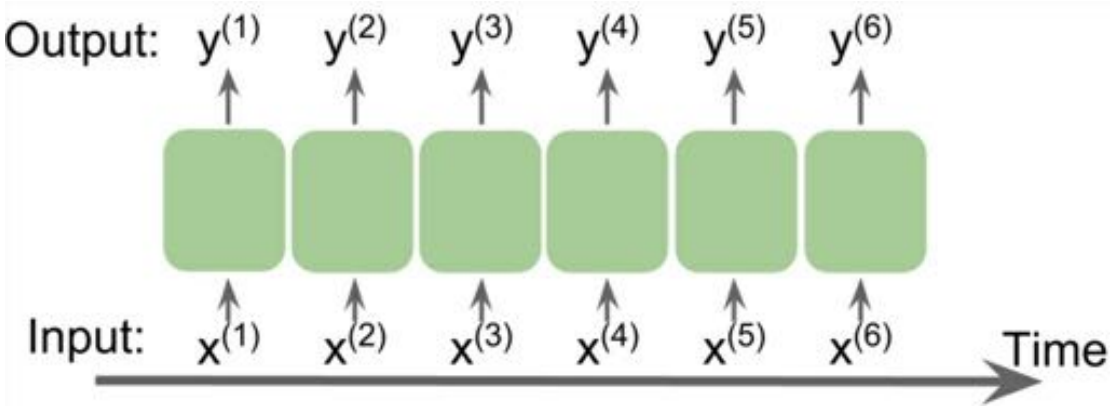
Roi Yehoshua
3/9/2018

Agenda

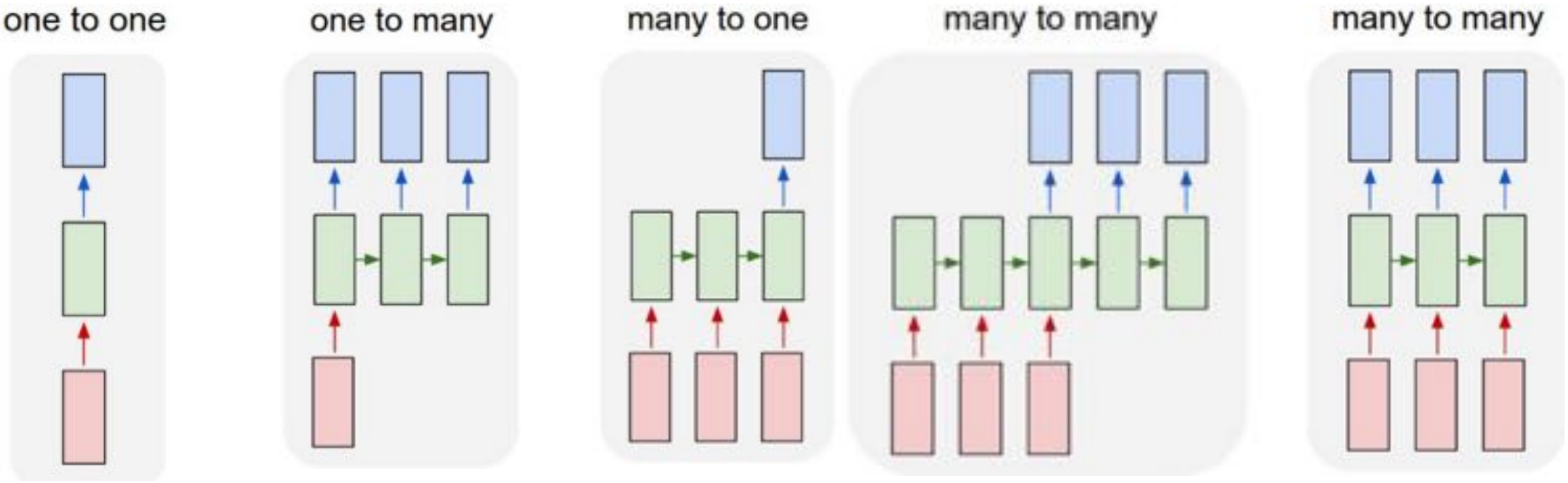
- Representing Sequences
- Structure of RNNs
- Computing activations in RNNs
- Training RNNs (Backpropagation Through Time)
- The problem of vanishing gradients
- LSTMs
- Practical Applications of RNNs

Recurrent Neural Networks (RNNs)

- Standard NN models (MLPs, CNNs) are not able to handle sequences of data
  - They accept a fixed-sized vector as input and produce a fixed-sized vector as output
  - The weights are updated independent of the order the samples are processed
- RNNs are designed for modeling **sequences**
  - Sequences in the input, in the output or in both
  - They are capable of remembering past information



Different Categories of Sequence Modeling



Vanilla mode without RNN
e.g. image classification

Sequence output
e.g., image captioning

Sequence input
e.g., sentiment analysis

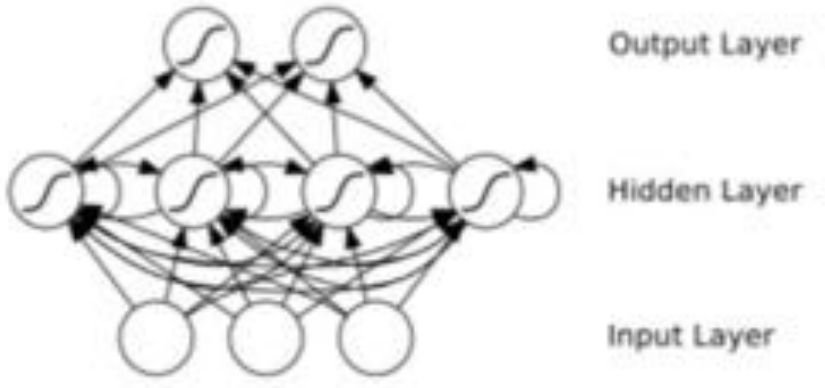
Sequence input and output
e.g., machine translation

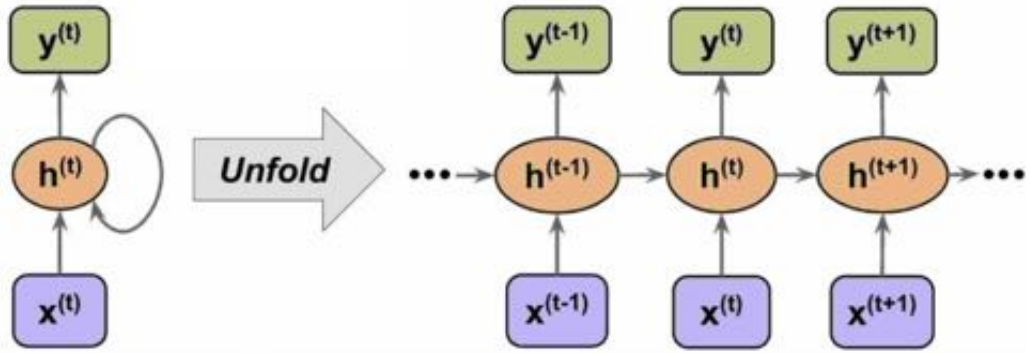
Synced sequence input and output
e.g., video classification

Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Structure of RNNs

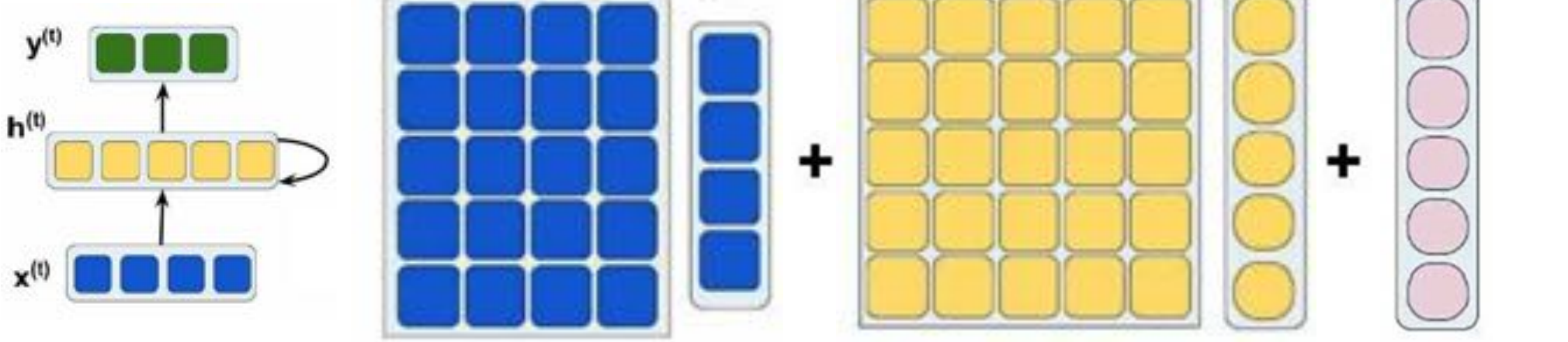
- In RNN the hidden layers are **recurrent layers**, where every neuron is connected to every other neuron in the layer
- The hidden layer gets its input from both the input layer $x^{(t)}$ and the hidden layer from the previous time step $h^{(t-1)}$





Computing Activations in RNN

- New hidden state: $\boldsymbol{h}^{(t)} = \phi_h(\boldsymbol{W}_{hx}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$



The weight matrix between the input and the hidden layer

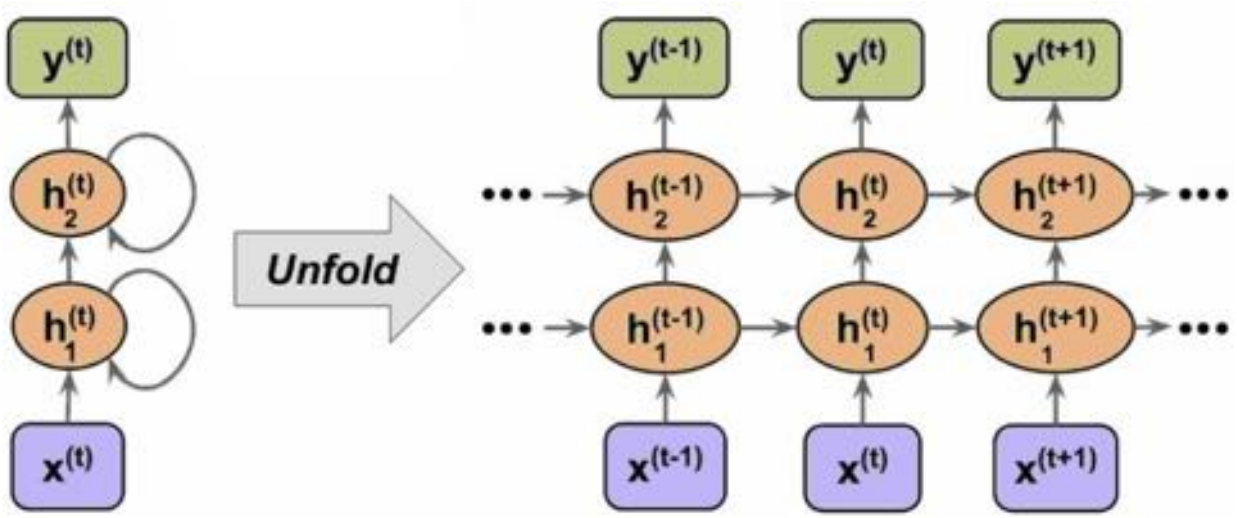
The weight matrix associated with the recurrent edges

Bias vector for the hidden units

- Output: $\boldsymbol{y}^{(t)} = \phi_y(\boldsymbol{W}_{yh}\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$

Multilayer RNN

- The second hidden layer gets its input from the hidden units from the layer below at the current time step $h_1^{(t)}$ and its own hidden values from previous time step $h_2^{(t-1)}$

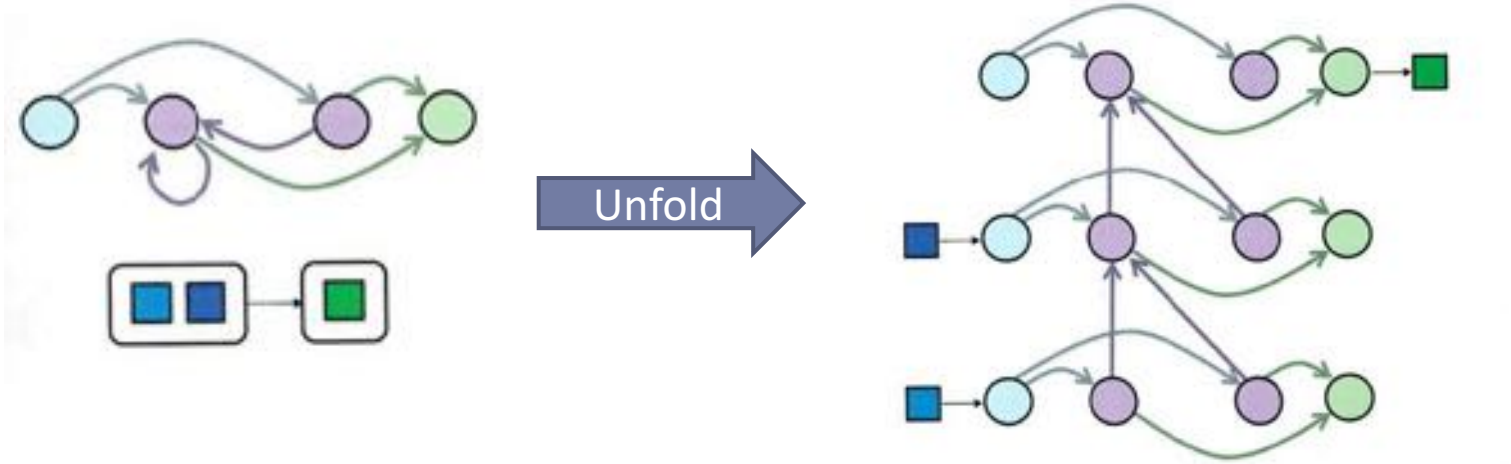


Representational Power of RNNs

- Regular feed-forward neural networks are not Turing-Complete
  - They can represent any single mathematical function but don't have any ability to perform looping or other control flow operations
- RNNs are Turing-Complete [Killan and Siegelmann, 1996]
  - They can simulate arbitrary programs
  - They can represent any mapping between input and output sequences

Training RNNs

- RNNs are trained by unfolding them into deep feedforward networks, where a new layer is created for each time step of an input sequence processed by the network



- Then Backpropagation is used to train the unfolded version of the network
  - Using the chain rule, as in feedforward networks
  - Paul Werbos, Backpropagation Through Time: What It Does and How to Do It, Proceedings of IEEE, 78(10):1550-1560, 1990

Backpropagation Through Time (BPTT)

- The overall loss L is the sum of all the loss functions at times $t = 1$ to $t = T$:

$$L = \sum_{t=1}^{T} L^{(t)}$$

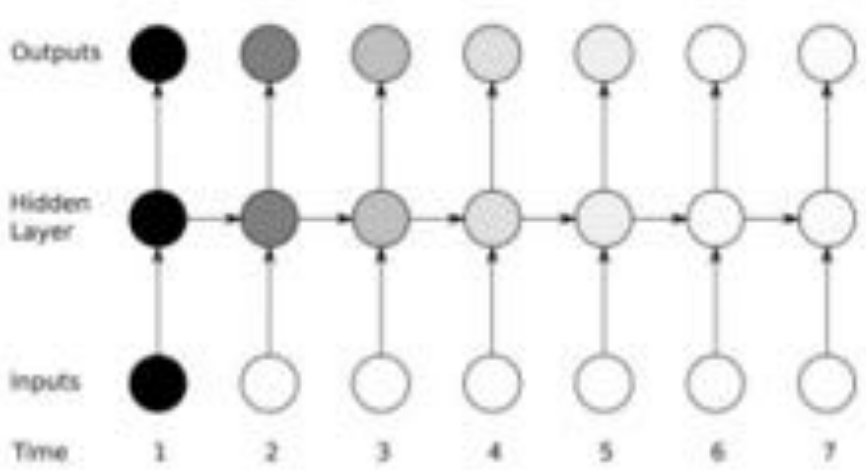
- The loss at time t depends on the hidden units at all previous time steps (1..t)

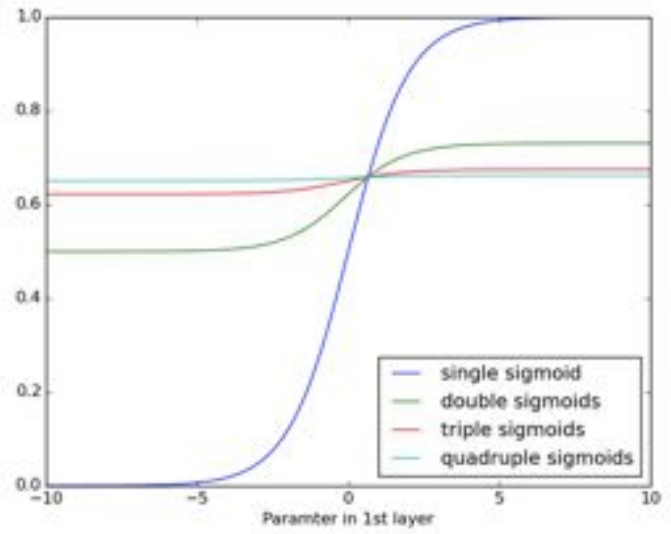
$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{y}^{(t)}} \cdot \frac{\partial \boldsymbol{y}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \cdot \left[\sum_{k=1}^{t} \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(k)}} \cdot \frac{\partial \boldsymbol{h}^{(k)}}{\partial \boldsymbol{W}} \right) \right] \qquad \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{h}^{(i-1)}}$$

  - $\boldsymbol{W}$ can be either $\boldsymbol{W}_{hh}$ or $\boldsymbol{W}_{hx}$
- BPTT can be computationally expensive as the number of time steps increases

The Problem of Vanishing Gradients

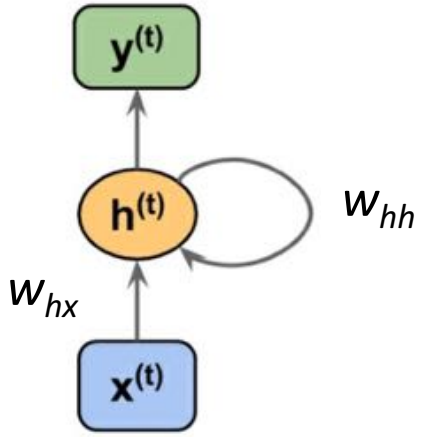
- Backpropagation computes gradients by the chain rule
- The gradient decreases exponentially with the number of layers in the network
  - or the length of the sequence in the case of RNNs
- This causes the front layers to train very slowly
  - Thus, vanilla RNNs are unable to capture long-term dependencies





Class Exercise

- Assume a single neuron, fully-connected recurrent layer:



- Prove: $\frac{\partial h^{(t)}}{\partial x^{(t-k)}} \leq \left| w_{hh} \right|^{k} \cdot w_{hx}$

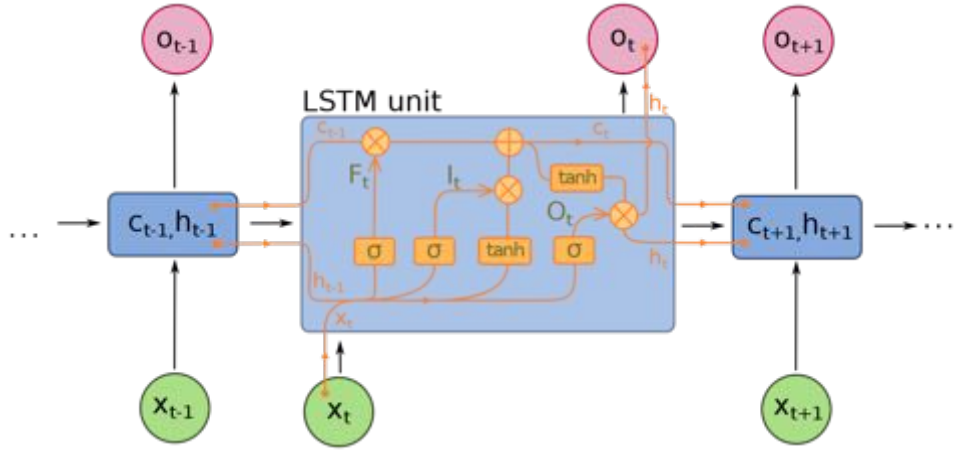
- Activation function may be either sigmoid, tanh or RelU

Truncated BPTT (TBPTT)

- A modified version of BPTT suggested by Ilya Sutskever in 2013:
  1. Present a sequence of k_1 time steps of input and output pairs to the network
  2. Perform an BPTT update back for k_2 time steps
  3. Repeat
- Example: Split a 1,000-long sequence into 50 sequences each of length 20 and treat each sequence of length 20 as a separate training case
- k_2 should be large enough to capture the temporal structure in the problem
  - but small enough to avoid vanishing gradients
- Problem: the network is blind to dependencies that span more than k_2 time steps

LSTM (Long Short Term Memory)

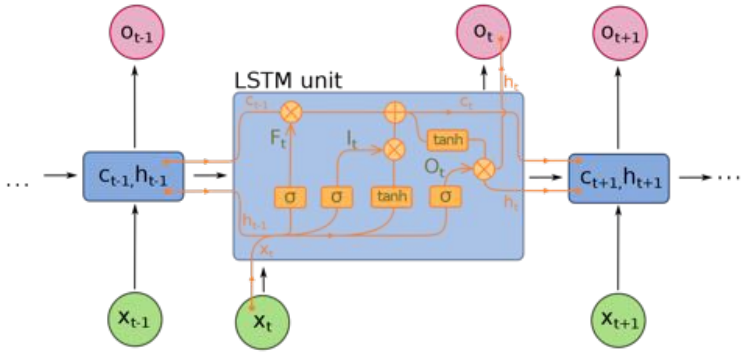
- Suggested in 1997 by Hochreiter and Schmidhuber as a solution to the vanishing gradient problem
- An LSTM cell stores a value (state) for either long or short time periods
- It contains three gates:
  - **Forget gate** - controls the extent to which a value remains in the cell
  - **Input gate** - controls the extent to which a new value flows into the cell
  - **Output gate** - controls the extent to which the value in the cell is used to compute the output



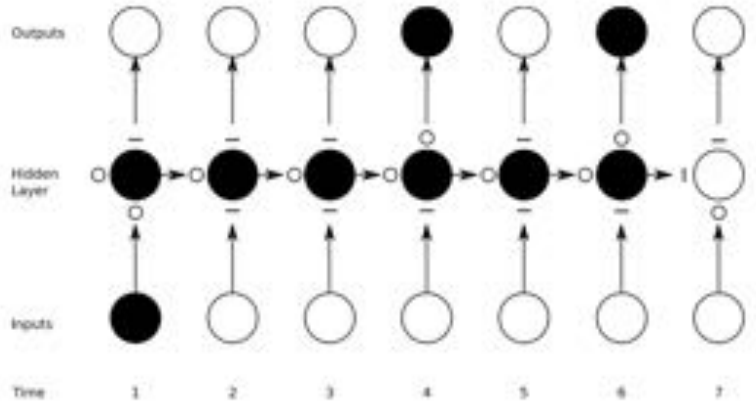
Source: Wikipedia

LSTM Forward Pass

- Forget gate: $\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{W}_{fx}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{fh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f)$
- Input gate: $\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_{ix}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{ih}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i)$
- Output gate: $\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{W}_{ox}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{oh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o)$
- New cell state: $\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \boldsymbol{c}^{(t-1)} \tanh(\boldsymbol{W}_{cx}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{ch}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_c)$
- Unit's output: $\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh(\boldsymbol{c}^{(t)})$



σ is the sigmoid function
$\circ$ refers to element-wise product



Training LSTMs

- An LSTM network (an RNN composed of LSTM units) is trained with BPTT
- The subsequent cell state is a **sum** of the current state with new input

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \boldsymbol{c}^{(t-1)} \tanh(\boldsymbol{W}_{cx}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{ch}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_c)$$

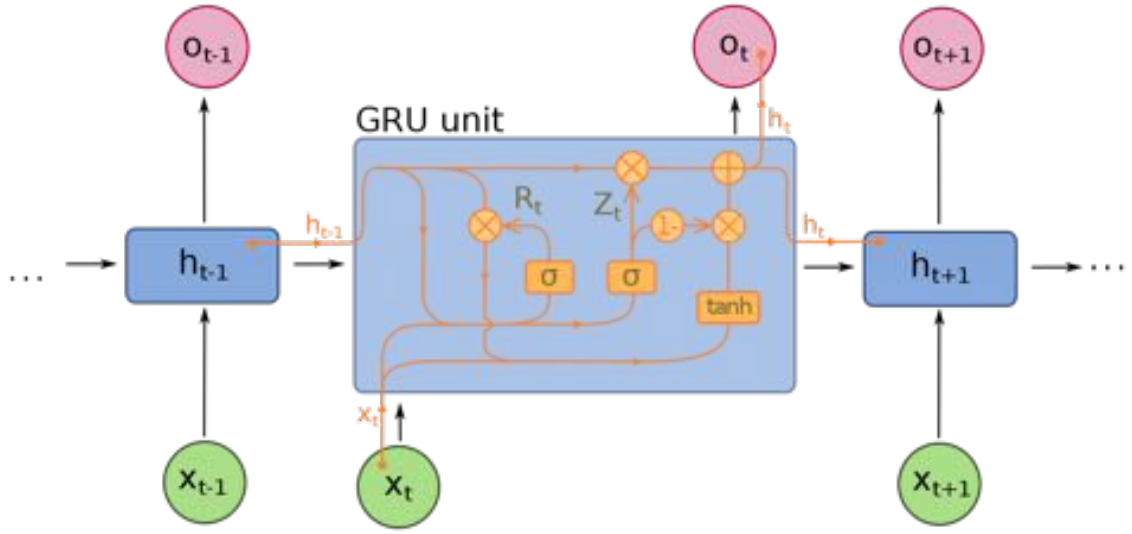
- This helps LSTMs preserve a constant error when it is backpropagated at depth

$$\frac{\partial \boldsymbol{c}^{(t)}}{\partial \boldsymbol{c}^{(t-1)}} = \boldsymbol{f}^{(t)} + \ldots$$

- The cells learn when to allow data to enter, leave or be deleted through the iterative process of backpropagating error and adjusting weights via gradient descent

Gated Recurrent Unit (GRU)

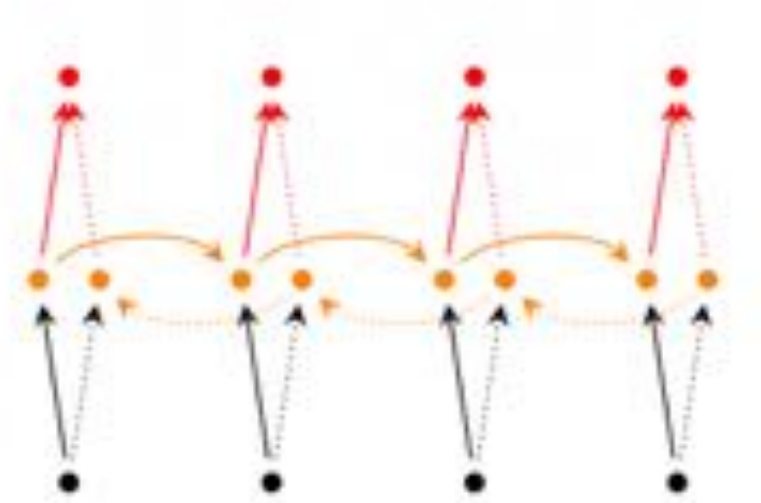
- Similar performance as LSTM with less computation
- They have fewer parameters than LSTM, as they lack an output gate



Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", arXiv:1406.1078

Bidirectional RNNs

- Output at time *t* may not only depend on the previous elements in the sequence, but also future elements.
  - e.g., to predict a missing word in a sequence we'd like to look at both left and right context
- Bidirectional RNNs are just two RNNs stacked on top of each other
- The output is then computed based on the hidden state of both RNNs

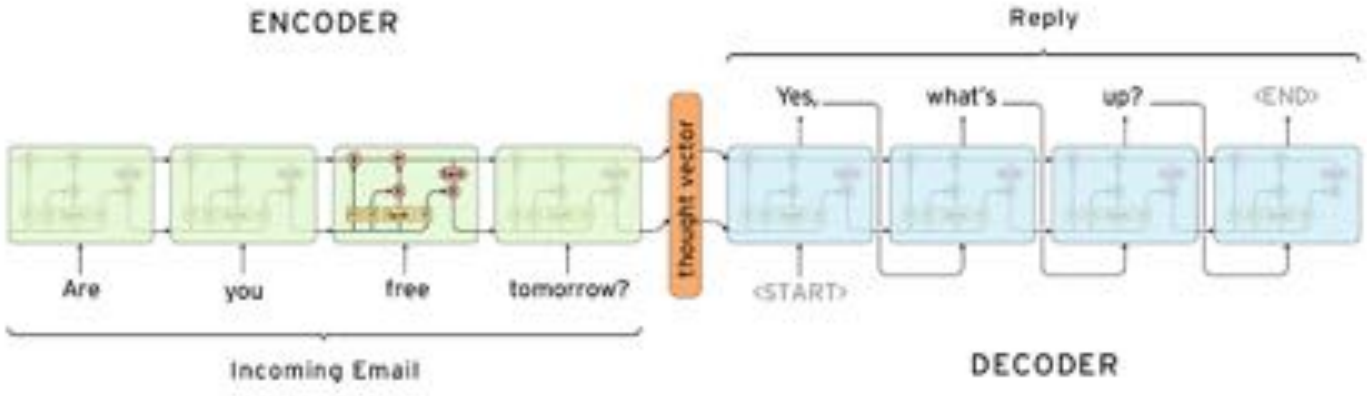


Practical Applications of RNNs

- Machine translation
  - Google uses LSTMs for Google Translate
- Question Answering
  - Apple uses LSTM for Siri, Amazon uses LSTM for Alexa
- Various NLP tasks
  - Part-of-speech tagging, named-entity recognition, sentiment analysis, etc.
- Speech recognition
  - Android's speech recognizer is based on LSTM RNNs (since 2012)
- Generating image descriptions
- Generating text
  - iOS QuickType auto-completion uses LSTM
- Handwriting recognition
  - LSTMs won the ICDAR handwriting competition (2009)

SEQ2SEQ (Sequence-To-Sequence)

- SEQ2SEQ has become a popular model for sequence generation
- Consists of two LSTMs: an Encoder and a Decoder
- The encoder takes a sequence (sentence) and converts it into a fixed-size vector
  - This ‘thought’ vector encodes the important information in the sentence
- The decoder ‘decodes’ this representation into a response, one word at a time
  - At each time step, the decoder is influenced by the context and the previously generated symbols.

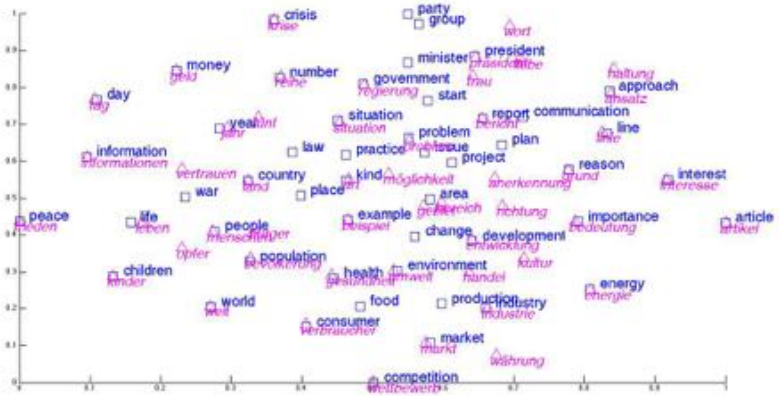


Source: https://github.com/farizrahman4u/seq2seq

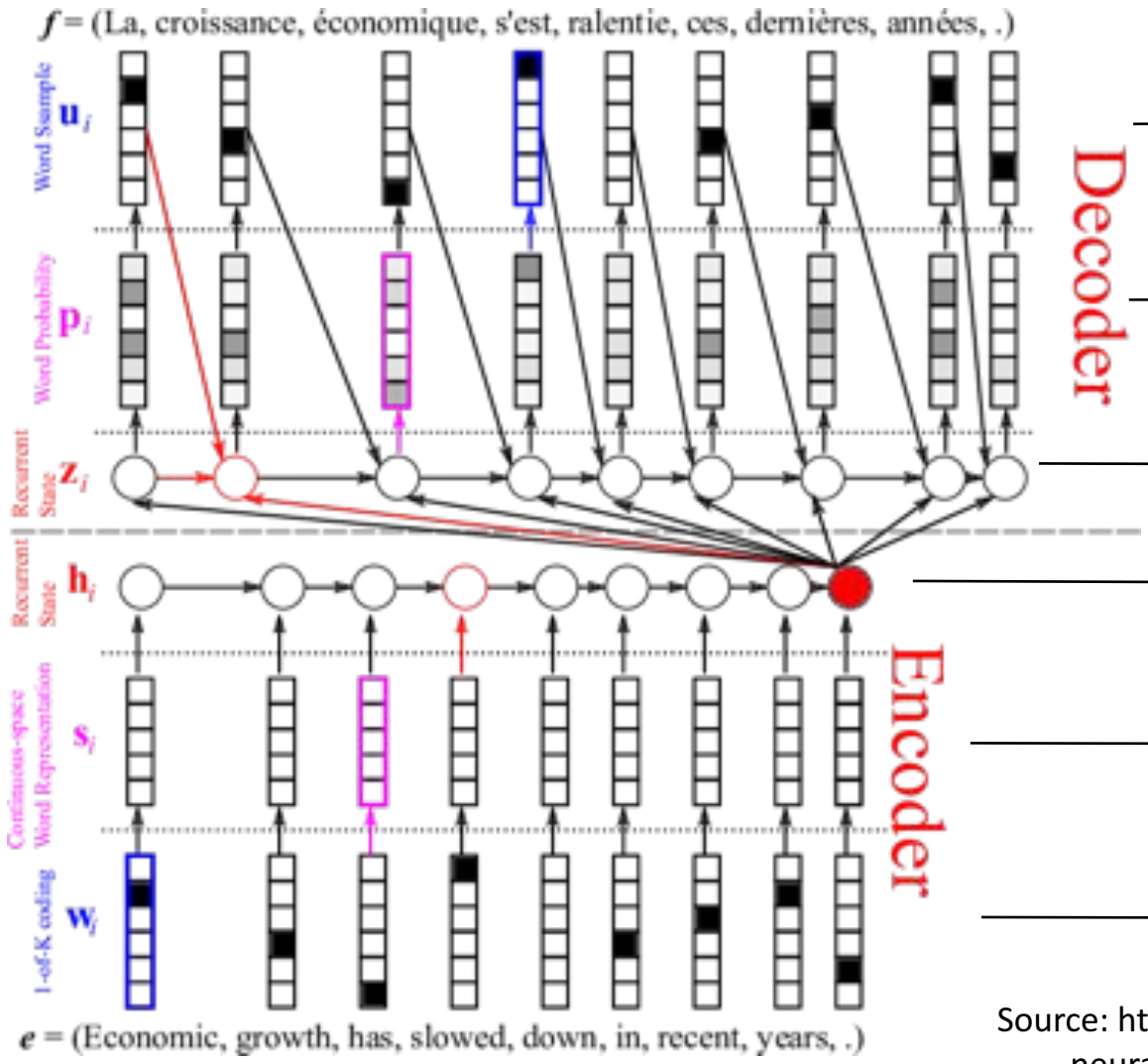
Word Embeddings

- Represent words using vectors of dimension *d*
  - *d* is typically in the range 100-500
- Solves the problem of sparsity in one-hot encodings
- Captures semantic relations between words
- The embedding layer is typically the first layer of the network
  - jointly trained with the other layers
  - often initialized with pre-trained embeddings such as word2vec

	x1	x2	x3	x4
the	0.51	0.009	0.43	0.44
cat	0.58	-0.53	0.25	-0.51
ate	0.54	-0.64	0.32	0.22
food	0.42	0.258	0.26	0.25



Machine Translation



Select a word by sampling the distribution

A softmax layer the computes the probability of each word given the hidden state z_i

The internal state of the decoder is based on the summary vector h_T, the previous word u_{i-1} and the previous internal state z_{i-1}

After the last word s_T is read, the encoder's hidden state h_T represents a summary of the whole sentence

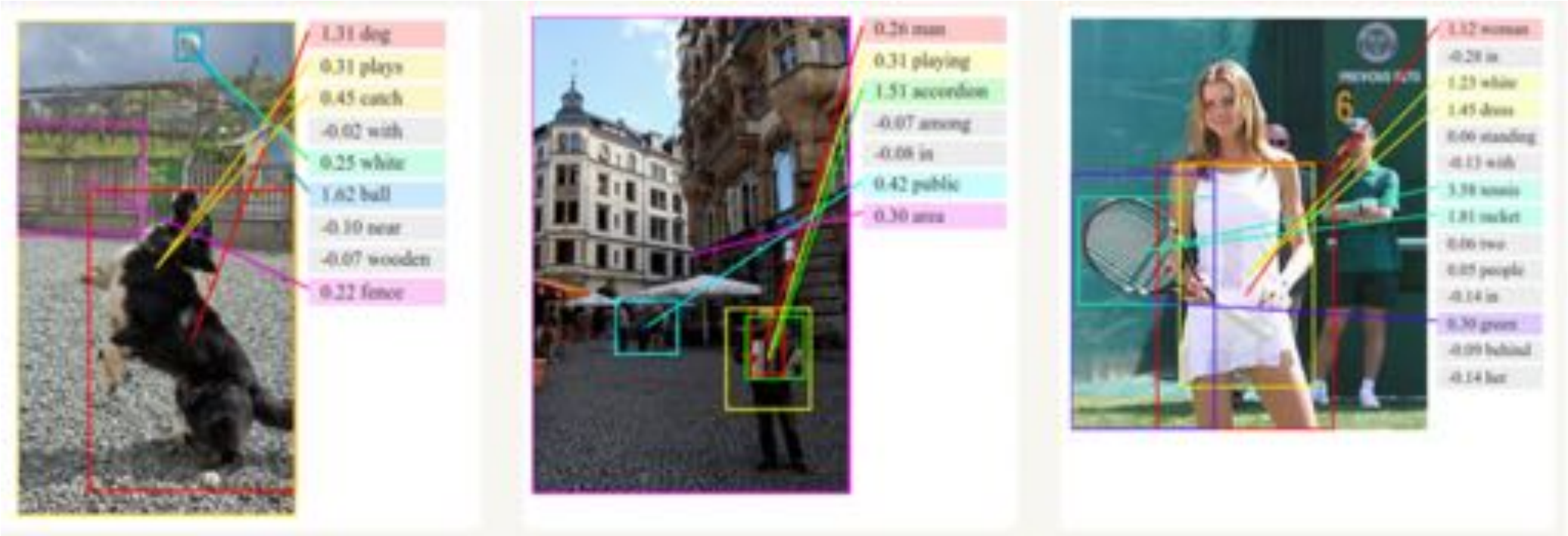
A one-hot vector to continuous space vector (the embedding layer)

A word to a one-hot vector

Source: https://devblogs.nvidia.com/introduction-neural-machine-translation-gpus-part-2/

Generating Image Descriptions

- Together with Convolutional Neural Networks, RNNs have been used to generate descriptions for unlabeled images

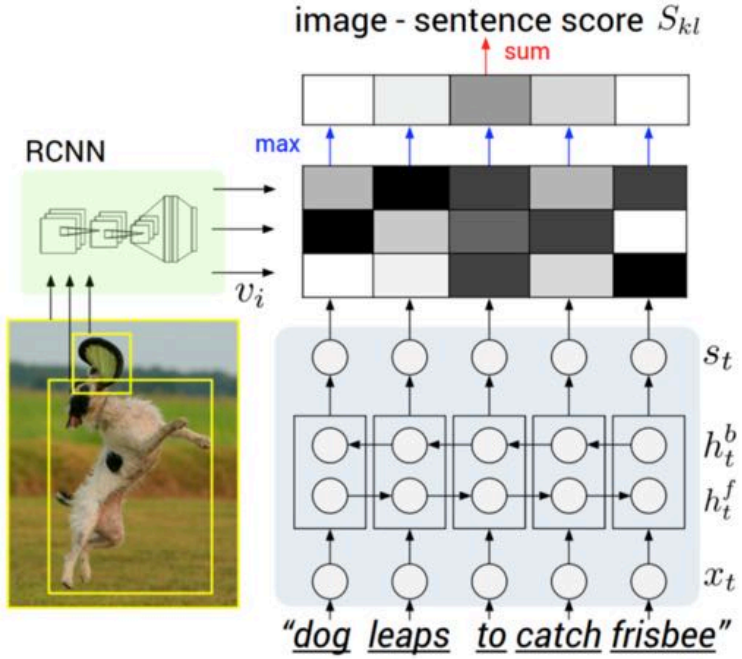


Karpathy and Fei-Fei, Deep Visual-Semantic Alignments for Generating Image Descriptions (2015)
Source: http://cs.stanford.edu/people/karpathy/deepimagesent/

Generating Image Descriptions – Step 1

- The network is first trained to align image regions with word snippets of the descriptions
- Regional Convoluatioal Neural Network (RCNN) is pre-trained on ImageNet to detect objects in images
  - Images are represented as a set of *h*-dimensional vectors v_i
- A Bidirectional Recurrent Neural Network (BRNN) is trained on text to compute word embeddings
  - Words are also represented as *h*-dimensional vectors s_t
- The dot product $v_i^T s_t$ reflects the similarity between region v_i and word s_t
- The objective function is to get the best alignment

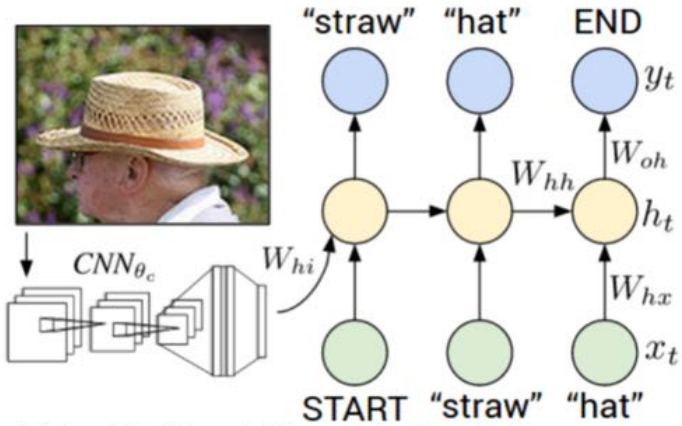
$$S_{kl} = \sum_{t \in g_l} max_{i \in g_k} v_i^T s_t$$



Ref: Deep Visual-Semantic Alignments for Generating Image Descriptions; http://cs.stanford.edu/people/karpathy/deepimagesent/

Generating Image Descriptions – Step 2

- The RNN takes a series of input words (e.g, START, “straw”, “hat”...), and a series of output words (e.g., “straw”, “hat”, END)
- The RNN is trained to combine a word (x_t) and the previous context (h_{t-1}) to predict the next word (y_t)
- The image representation is used to initialize the hidden state
- The cost function is to maximize the log probability of the assigned labels (i.e. a Softmax classifier)
- To predict a sentence:
  - The image representation b_v is computed
  - h_0 is set to 0 and x_1 is set to the START vector
  - The network computes the distribution over the first word y_1
  - A word is sampled from the distribution and its embedding vector is set as x_2
  - This process is repeated until the END token is generated



Ref: Deep Visual-Semantic Alignments for Generating Image Descriptions; http://cs.stanford.edu/people/karpathy/deepimagesent/

$$b_v = W_{hi}[CNN_{\theta_c}(I)]$$
$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h + \mathbb{1}(t=1) \odot b_v)$$
$$y_t = softmax(W_{oh}h_t + b_o).$$

Summary

- RNNs are neural networks that deal with sequence data
- Training recurrent nets is optimization over programs, not functions
- RNNs are becoming a pervasive and critical component to intelligent systems
  - with many practical applications
- Many variants
  - LSTM, GRU, Bi-Directional LSTM, Deep RNNs
- Further readings
  - The Unreasonable Effectiveness of Recurrent Neural Networks
    http://karpathy.github.io/2015/05/21/rnn-effectiveness/
  - Understanding LSTM Networks
    https://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - Training a Neural Machine Translation network in TensorFlow
    https://www.tensorflow.org/tutorials/seq2seq

http://www.cs.biu.ac.il/~yehoshr1/