BIO-INSPIRED COMPUTING: APPLICATIONS AND INTERFACES

SEQUENCE LEARNING PART I

Slides created by Jo Plested

LECTURE OUTLINE

- Recurrent Neural Networks
  - motivation for RNNs
  - what they look like compared to standard NNs
  - how does the backpropagation algorithm change for an RNN
- Long Short-Term Memory Units
  - how do they differ from RNNs
  - current research

MOTIVATION FOR RNNS

Feed Forward Neural Networks can only be applied to problems whose inputs and targets can be encoded with vectors of fixed dimensionality

For example:

- how do we learn to translate a sentence of unknown length from one language to another?
- how do we learn to generate text of unknown length e.g. speech to text, image/video captioning?
- how do we learn to recognise actions in videos of unknown length?

MOTIVATION FOR RNNS

- Neural Networks are memoryless, current output only depends on current input:
  - but sometimes context is important
  - for example language translation. To translate "Little John was looking for his toy box. Finally he found it. The box was in the pen." context is required to work out that pen means enclosure rather than writing device

VECTOR NOTATION REFRESHER

Dot product

$$\langle \boldsymbol{x}, \boldsymbol{w} \rangle = \boldsymbol{x}^T \mathbf{w} = \sum_{i=1}^{n} x_i w_i$$

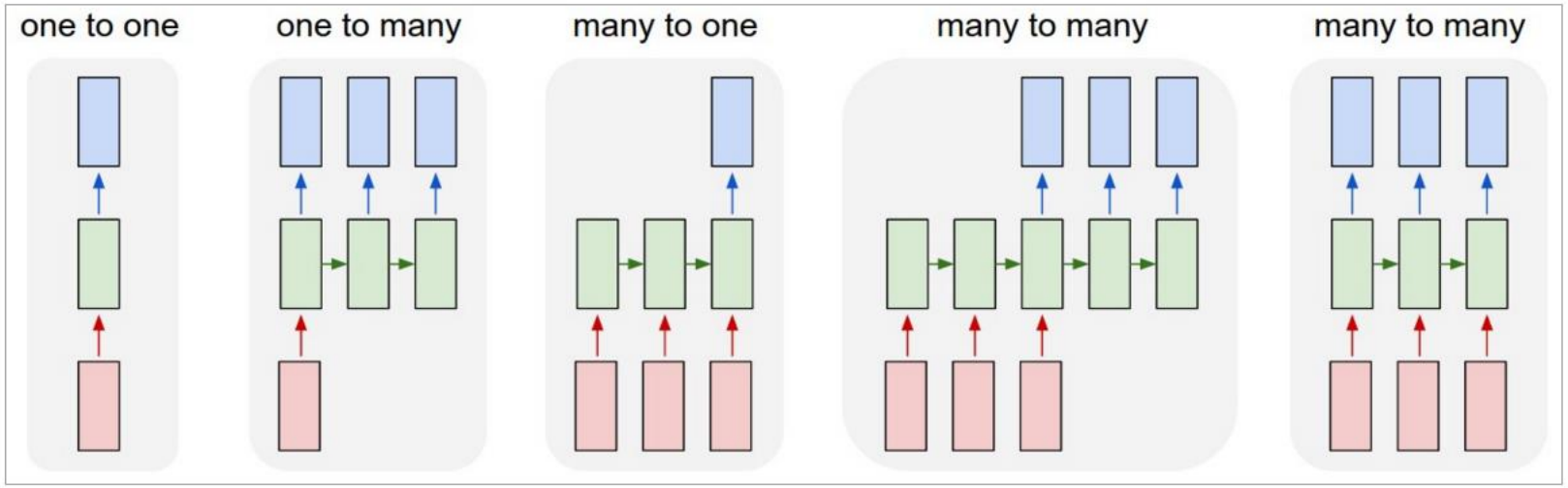
Matrix multiplication.

To multiply matrices they must have the same neighbouring dimensions e.g. For the matrices $A \in \mathbb{R}^{n \times k}$ and $B \in \mathbb{R}^{k \times m}$ the product $A \times B = C \in \mathbb{R}^{n \times m}$ exists. To compute the element c_{ij} of C we multiply the elements of the ith row of A with the jth column of B and sum them. The product $B \times A$ does not exist as the neighbouring dimensions m and n are not the same.

WHAT DOES AN RNN LOOK LIKE?

RNNs can model sequences of arbitrary length

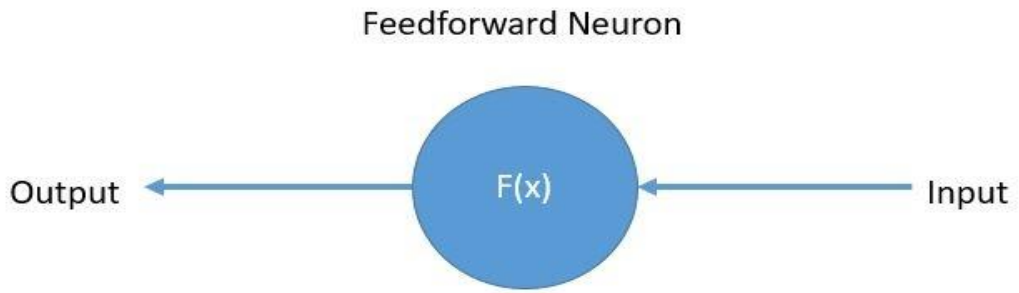
Inputs or outputs can be sequential, or both. Basically, anything that can be modelled as a sequence through time can be modelled by an RNN

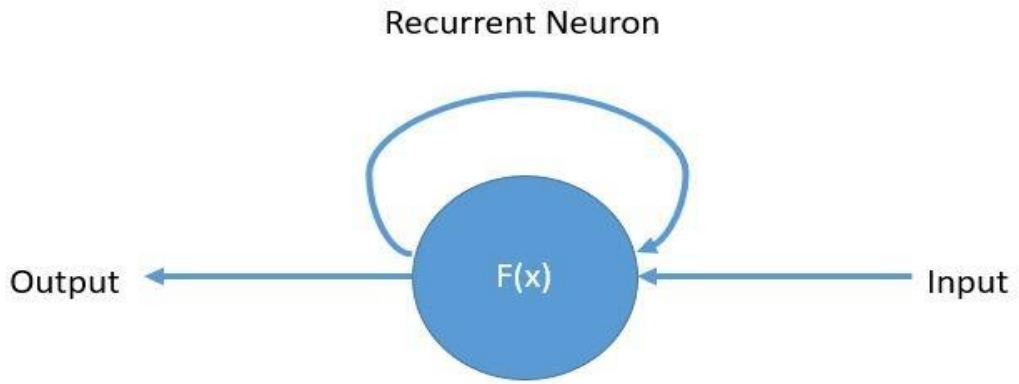


Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized 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.

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

WHAT DOES AN RNN LOOK LIKE?





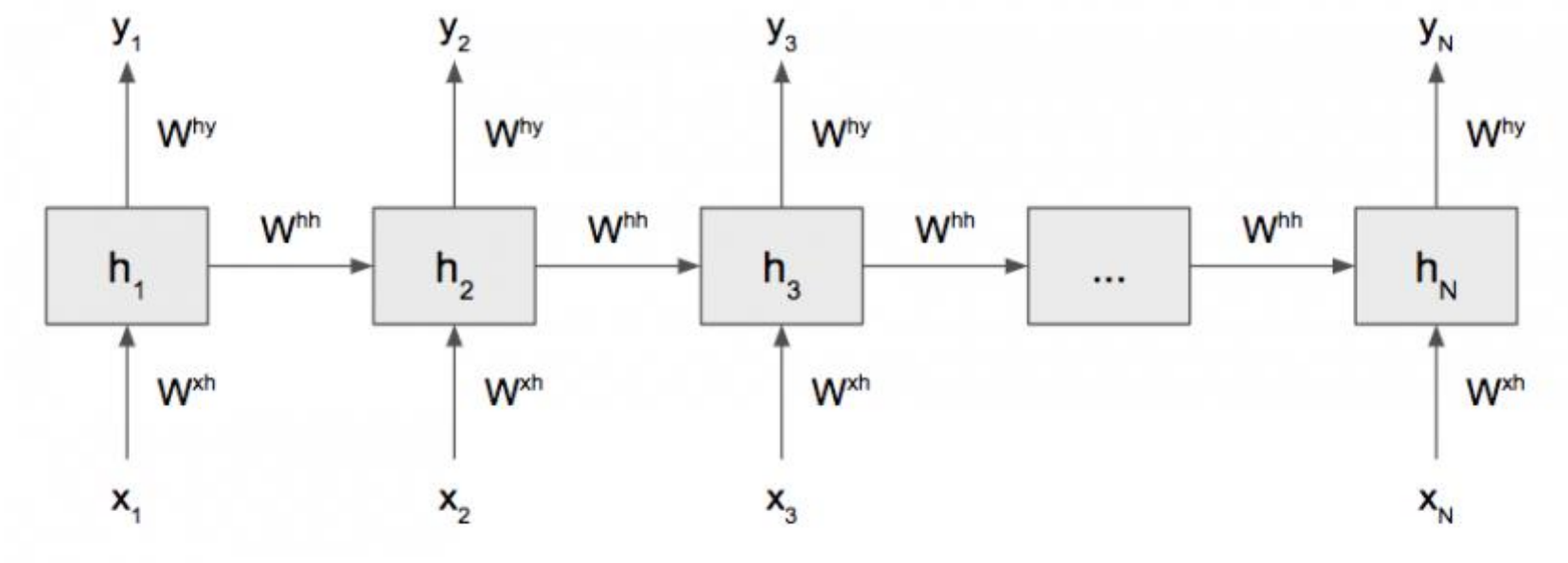
A **feedforward** neural network can't have any backwards or recurrent connections i.e. all connections must go from a neuron in one layer to a subsequent layer

A **recurrent** neural network has a feedback connection from its output back to it's input - so it can cycle through the same activation and weight connections for each time step (step of the sequence)

https://wiki.tum.de/display/lfdv/Recurrent+Neural+Networks+-+Combination+of+RNN+and+CNN

WHAT DOES AN RNN LOOK LIKE?

An RNN can be unrolled through time to create a standard NN with arbitrary input size with the condition that the weights W at each time step (step of the sequence, not learning step) have the same value



WHAT DOES AN RNN LOOK LIKE?

- How does the hidden state differ from a standard feed forward NN?
- The hidden state now holds the memory of the network as well as a representation of the current input
- The hidden state h at time t now becomes:
  - $\mathbf{h}_t = f(\mathbf{w}_{hh}\mathbf{h}_{t-1} + \mathbf{w}_{xh}\mathbf{x}_t + \mathbf{b}_h)$
  - where $\mathbf{w}_{hh}$ are the hidden to hidden weights, $\mathbf{w}_{xh}$ are the input to hidden weights and $\mathbf{b}_h$ is a standard bias vector.

WHAT DOES AN RNN LOOK LIKE?

The hidden state at time 0 (h_0) needs to be initialised for each new sequence - often done as a random initialisation, but can also be learned, or transferred from the last time step of the previous sequence if sequences are in order

WHAT DOES AN RNN LOOK LIKE?

An example with **Numpy**:

```
class RNN:
  # …
  # loop through for each step in sequence
 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) + b)
     # compute the output vector
     y = np.dot(self.W_hy, self.h)
     return y
```

In **pytorch** RNN layers are found under recurrent layers in torch.nn

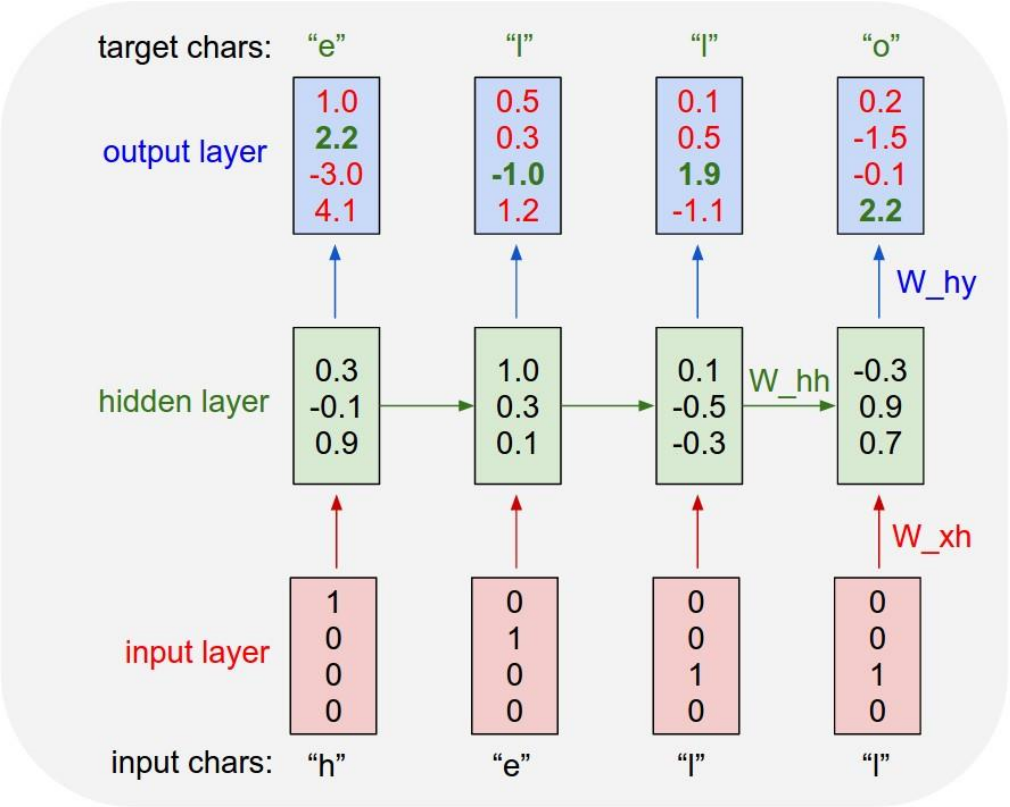
```
rnn = nn.RNN(10, 20, 2)
input = Variable(torch.randn(5, 3, 10))
h0 = Variable(torch.randn(2, 3, 20))
output, hn = rnn(input, h0)
```

BACKPROPAGATION THROUGH TIME

Backpropagation proceeds as usual, but with the weights $\mathbf{w}_{hh}$, $\mathbf{w}_{xh}$, $\mathbf{w}_{hy}$ and biases constrained to be the same value at each time step (step through the sequence, not learning step)

To constrain $w_1 = w_2$
Initialise them to the same weights then make sure $\Delta w_1 = \Delta w_2$
Use the learning rate multiplied by $\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$
to update both w_1 and w_2

A SIMPLE RNN EXAMPLE



An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"). We want green numbers to be high and red numbers to be low. The output numbers are raw scores and a softmax activation function would generally be used next to produce probabilities of each character.

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

DIFFICULTIES IN TRAINING RNNS

- With just one state vector and weight vector learning the relationship over time and being constantly overwritten, long term dependencies are difficult so the range of context that can be learned is limited
- In theory the backpropagation through time algorithm allows us to train RNNs using gradient descent
- In practice RNNs are unstable and when backpropagating gradients through long time windows the gradients can explode or vanish

VANISHING AND EXPLODING GRADIENTS

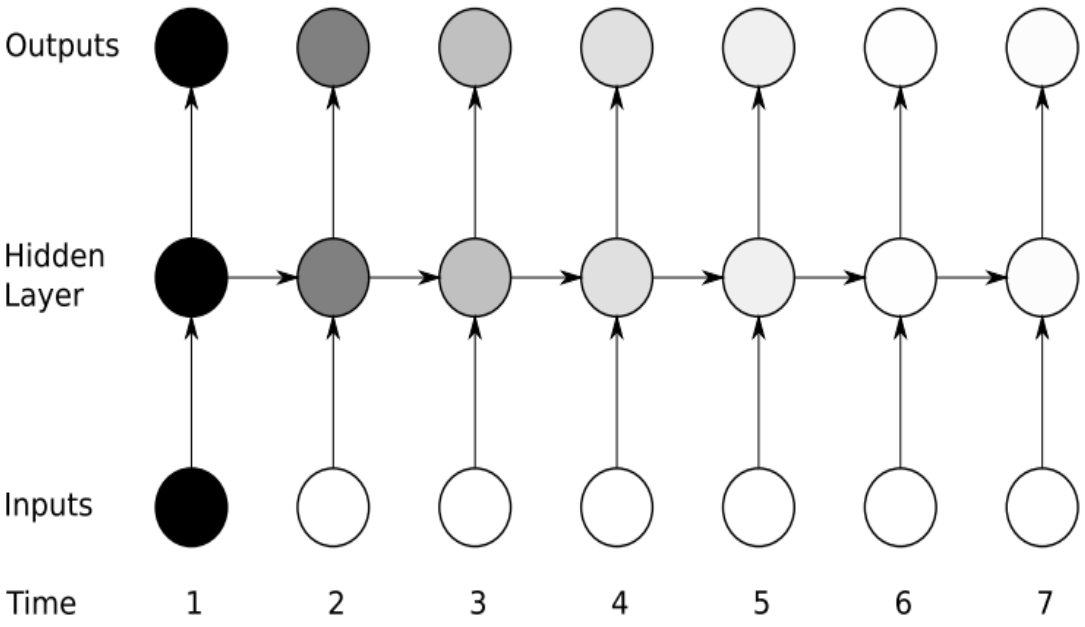
- Consider a simple RNN *without* an activation function and no biases or inputs.

$$h_t = W_{hh} W_{hh} \dots W_{hh} h_0 = W_{hh}^t h_0$$

$$\frac{dh_t}{dh_0} = W_{hh}^t$$

- The derivative of the hidden state w.r.t to the state t time steps ago grows/decays exponentially if the weights in W_{hh}^t are greater/less then one.

VANISHING GRADIENT



A. Graves. 2012. “Supervised Sequence Labelling with Recurrent Neural Networks”, Chapter 4.

The shading of the nodes indicates the sensitivity over time of the network nodes to the input at time one (the darker the shade, the greater the sensitivity)

The sensitivity decays exponentially over time as new inputs overwrite the activation of hidden unit and the network ‘forgets’ the first input

TRICKS TO KEEP RNNS STABLE

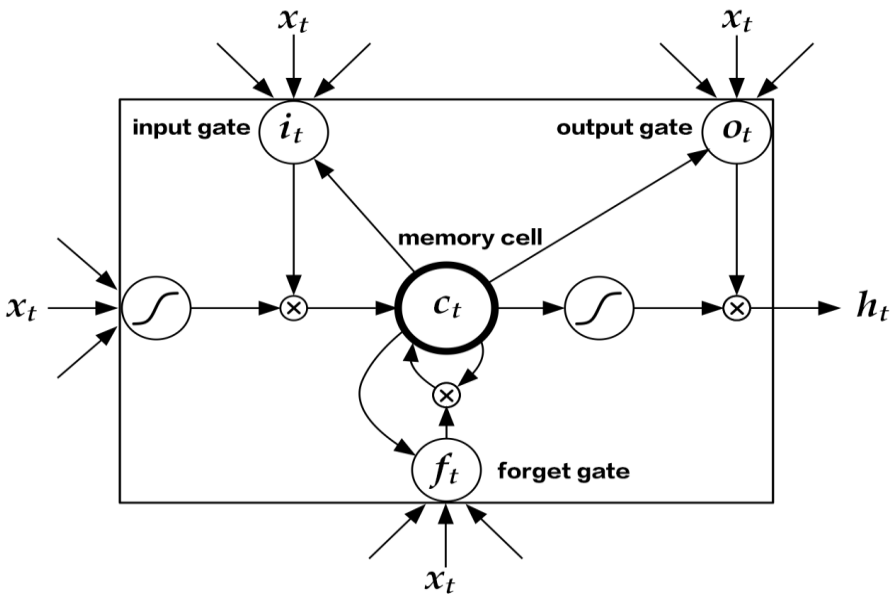
There are a number of tricks used to keep RNNs stable:

1. Gradient clipping: Prevents gradients from exploding by rescaling them so their norm is at most a particular value
2. Regularisation: Using an L1 or L2 penalty on the recurrent weights can help prevent exploding gradients
3. Input reversal: Reduces the distance between inputs e.g. in language translation to deal with context an RNN learns the entire representation of a sentence before outputting words of the target language. Learning the input in both directions creates shorter distances between input and target words
4. Long Short-Term Memory RNNs (LSTMs)

LSTMS

LSTMS follow the same basic structure as RNNS except that the nonlinear units in the hidden layer are replaced by memory blocks

Each memory block contains one or more memory cells along with input, output and forget gates which control flow of information into and out of the memory cell



$$
\begin{aligned}
i_t &= \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hc}h_{(t-1)} + b_{hg}) \\
o_t &= \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
$$

https://pytorch.org/docs/stable/nn.html#lstm

LSTMS

The basic idea of LSTMs is that the memory cell stores information until it's relevant:

- the input (i) gate controls what portion of the new input goes into the current hidden unit based on the previous content of the memory cell and hidden unit and the current input
- the output gate (o) controls what comes out of the memory cell into the current hidden unit
- the forget gate (f) controls when the contents of the memory cell are forgotten
- the input modulation gate (g) modulates the information that goes into the memory cell allowing for faster convergence (not always included but good practice)

LSTMS

- If you've missed what someone said and asked them to repeat it, but, by the time they do you've worked it out from context when they kept speaking, that's a similar process to an LSTM
- To go back to our earlier example of translating this sentence: "Little John was looking for his toy box. Finally he found it. The box was in the pen." An LSTM is much more suited to this task than a vanilla RNN as it can store the information that the box was a toy box until it needs it to work out that the pen is a play pen not a writing pen
- Handwriting recognition example: https://www.youtube.com/watch?v=mLxsbWAYIpw

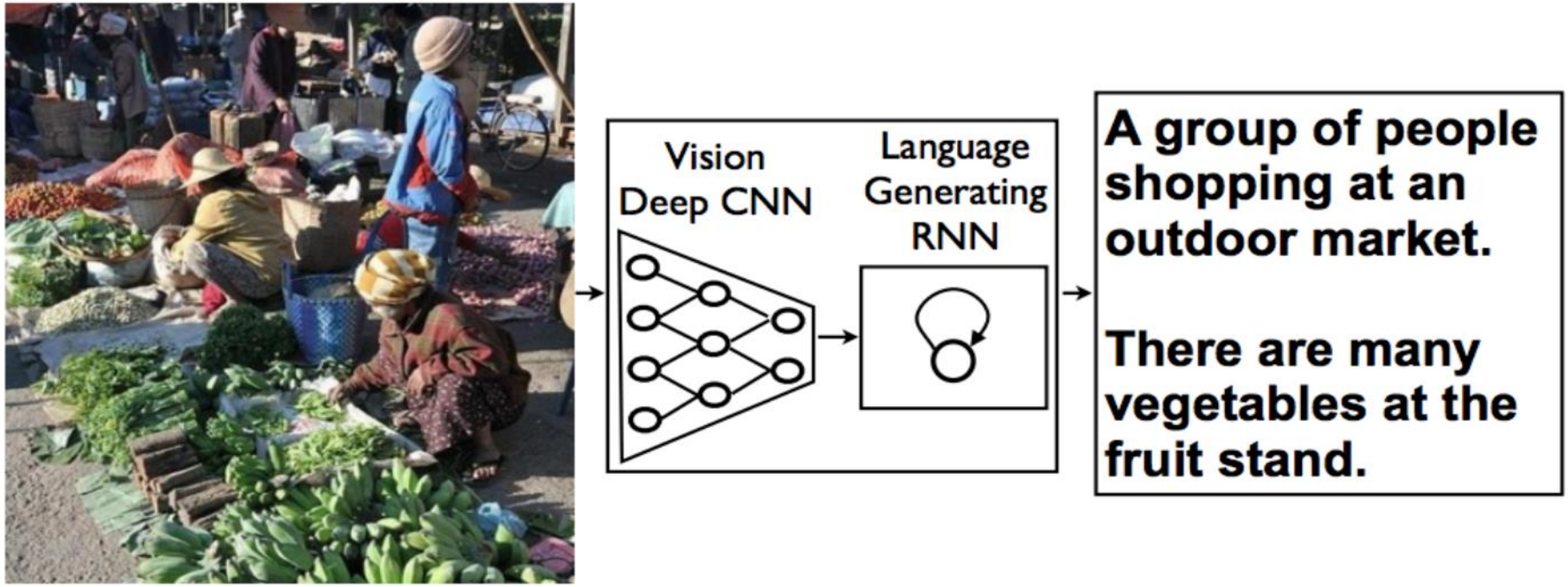
LSTMS TRICKS AND CONSIDERATIONS

- In practice when people talk about RNNs they generally mean LSTMs as they work better in most cases
- Still somewhat unstable though, so tricks for RNNs - such as gradient clipping and reversing inputs are still sometimes used in LSTMs
- Deep LSTMs can be built in the usual way by stacking LSTM layers on top of each other with the input for a deeper layer being the output from the previous layer

WORKING WITH TEXT - EMBEDDINGS

- In order to apply RNNs to text, we need to represent the text a sequence of fixed-length vectors.
- A very common approach is to just treat each word as a categorical variable and one-hot encode them.
- The one-hot encoded vectors are then fed through a linear layer – but if the vocabulary size is very large this requires a large matrix multiplication.
- Pytorch provides an nn.Embedding module which performs an equivalent operation by just looking up the specified row in the weight matrix.

LSTMS EXAMPLES – IMAGE CAPTIONING



https://www.cv-foundation.org/openaccess/content_cvpr_2015/app/2A_101.pdf

LSTMS EXAMPLES – IMAGE CAPTIONING

Image captioning: Uses a deep CNN for object detection and an RNN for generating text one word at a time

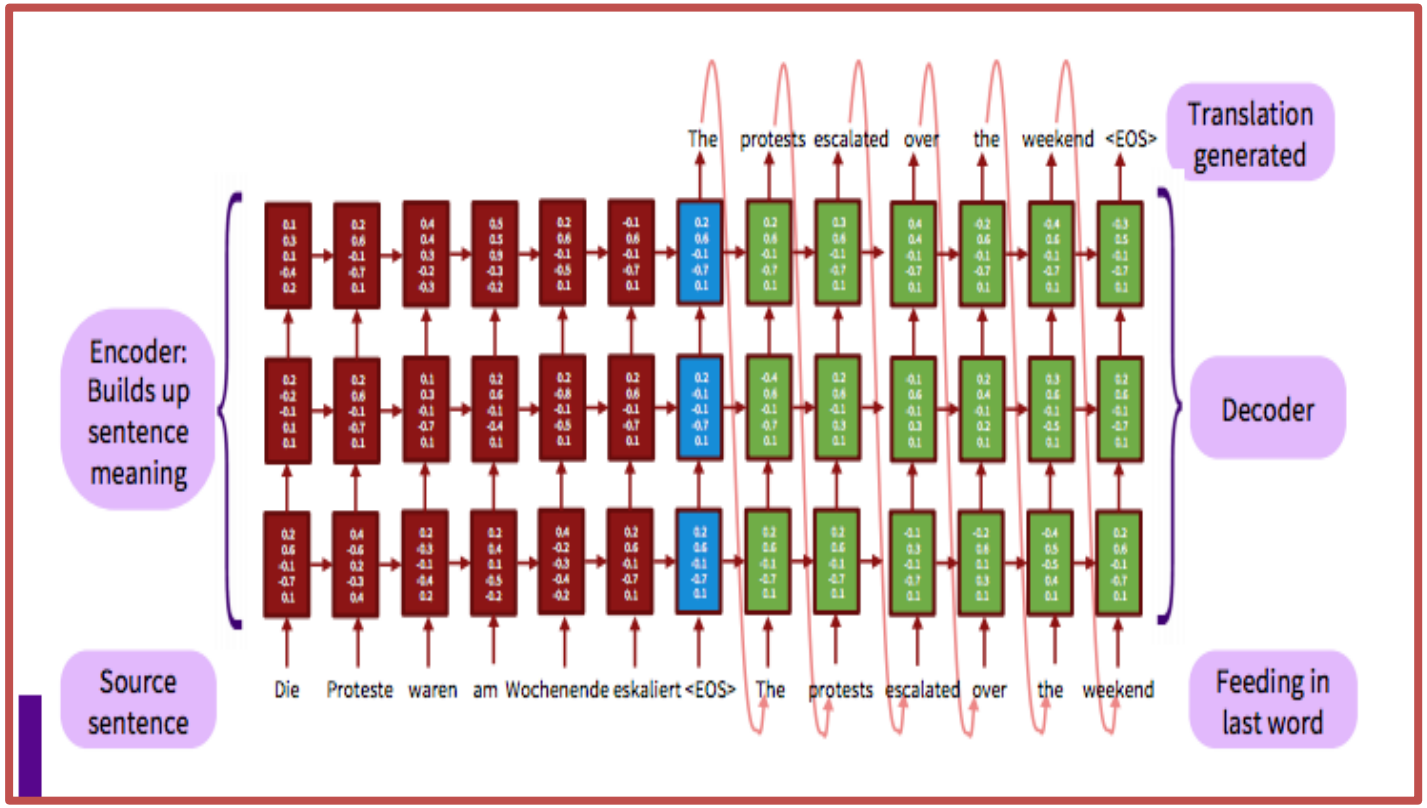
Papers:

- Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on. IEEE, 2015
- Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015

Implementation: Both papers are implemented in Python (not PyTorch) at https://github.com/karpathy/neuraltalk

LSTMS EXAMPLES – LANGUAGE TRANSLATION

Basic LSTM language translation model. One LSTM to learn sentence meaning, another to learn to generate a sentence in the target language

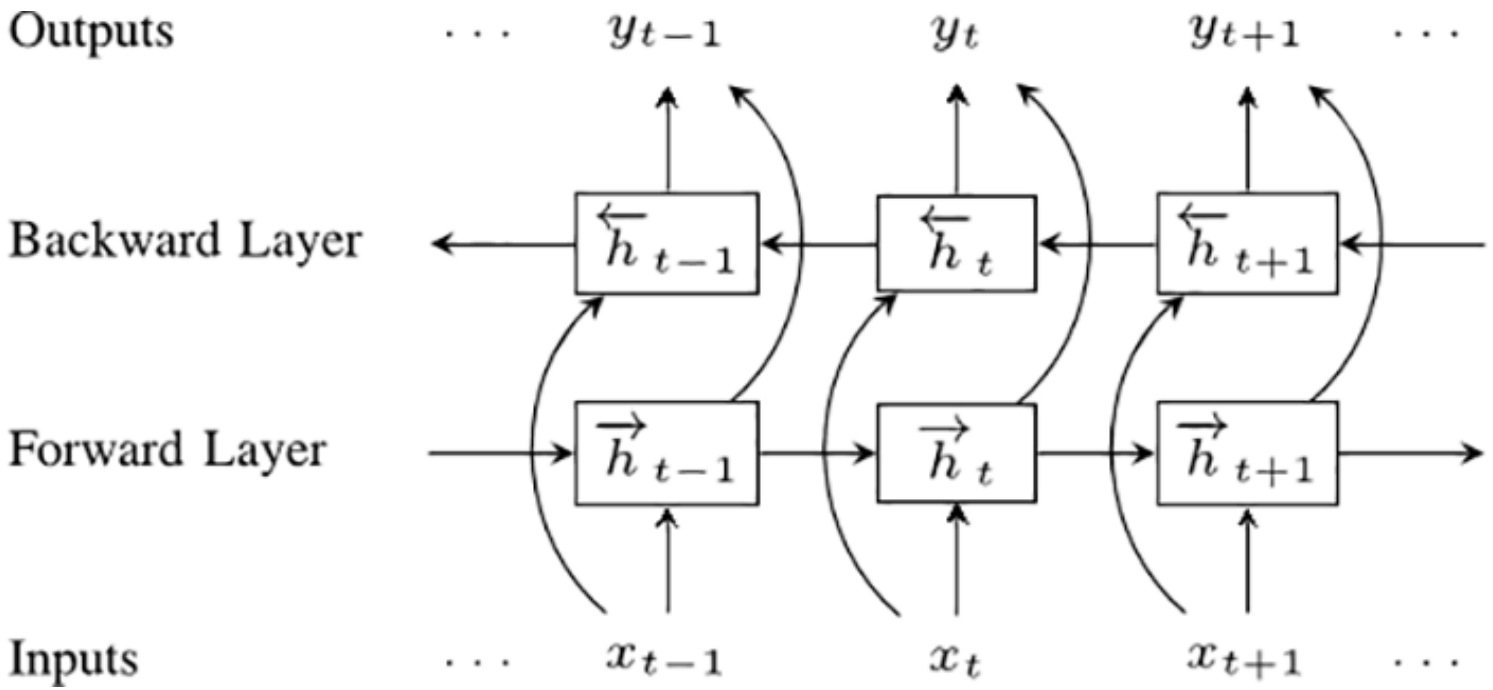


Paper: Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014

https://medium.com/@Synced/representations-for-language-from-word-embeddings-to-sentence-meanings-3fae81b2379a

LSTMS EXAMPLES – LANGUAGE TRANSLATION

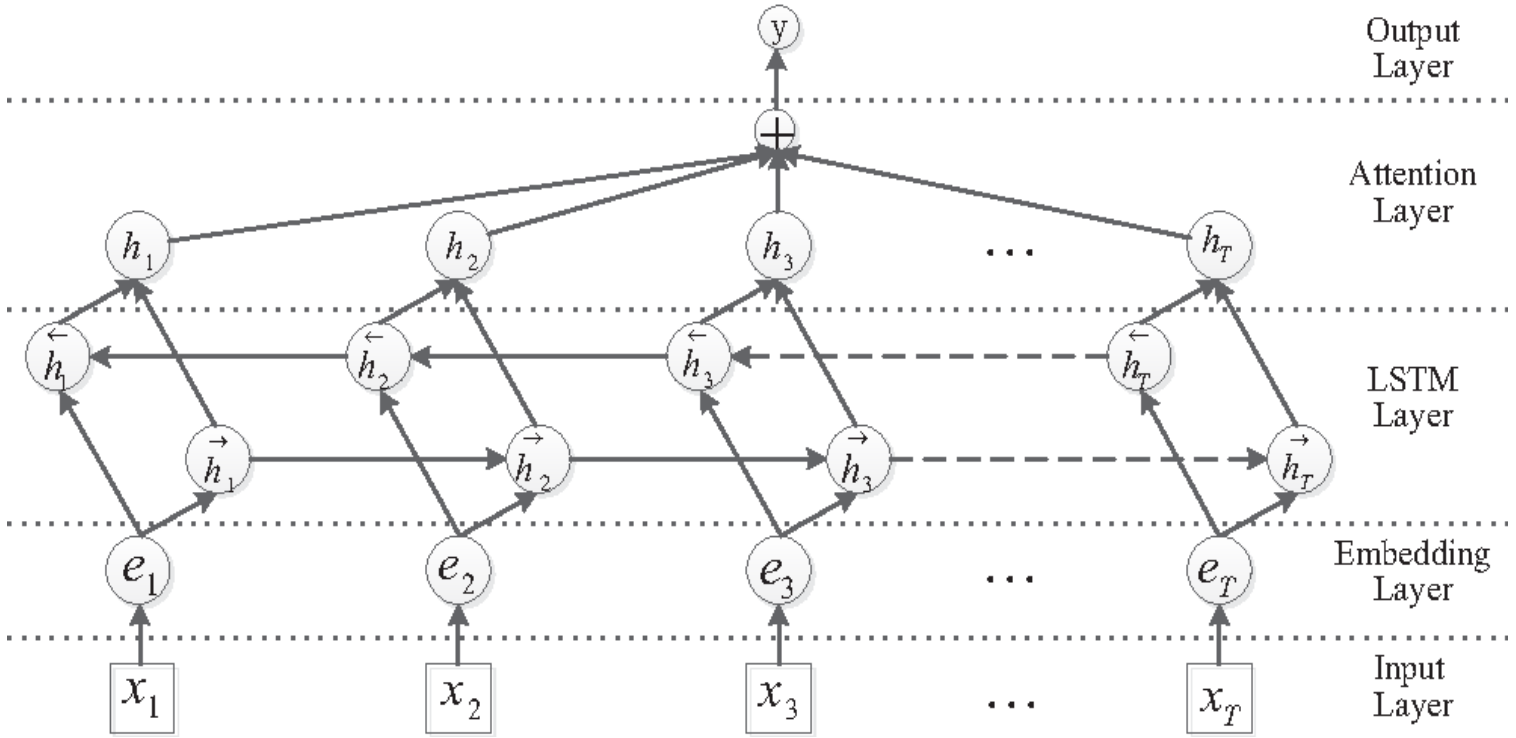
Bidirectional LSTM. Two LSTM layers, one going forward through the sentence and the other going backwards. Solves the problem of the model forgetting the tokens at the beginning of the sentence for longer sentences



https://www.semanticscholar.org/paper/Hybrid-speech-recognition-with-Deep-Bidirectional-Graves-Jaitly/1149888d75af4ed5dffc25731b875651c3ccdeb2

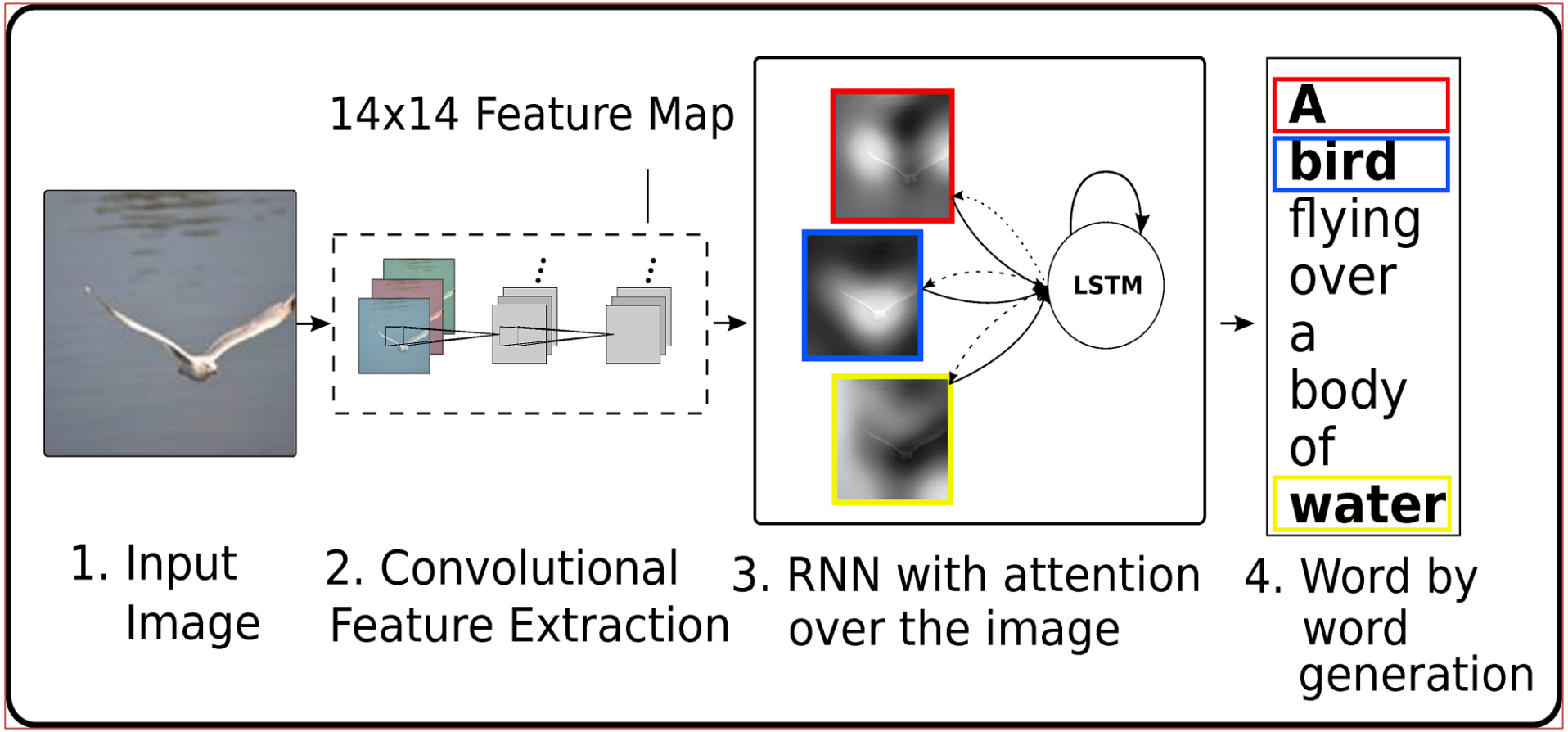
LSTMS EXAMPLES – LANGUAGE TRANSLATION

Bidirectional LSTM with attention. Adds an attention mechanism that weights the importance of each final hidden layer output in producing each word output. Considered cutting edge for NLP tasks.



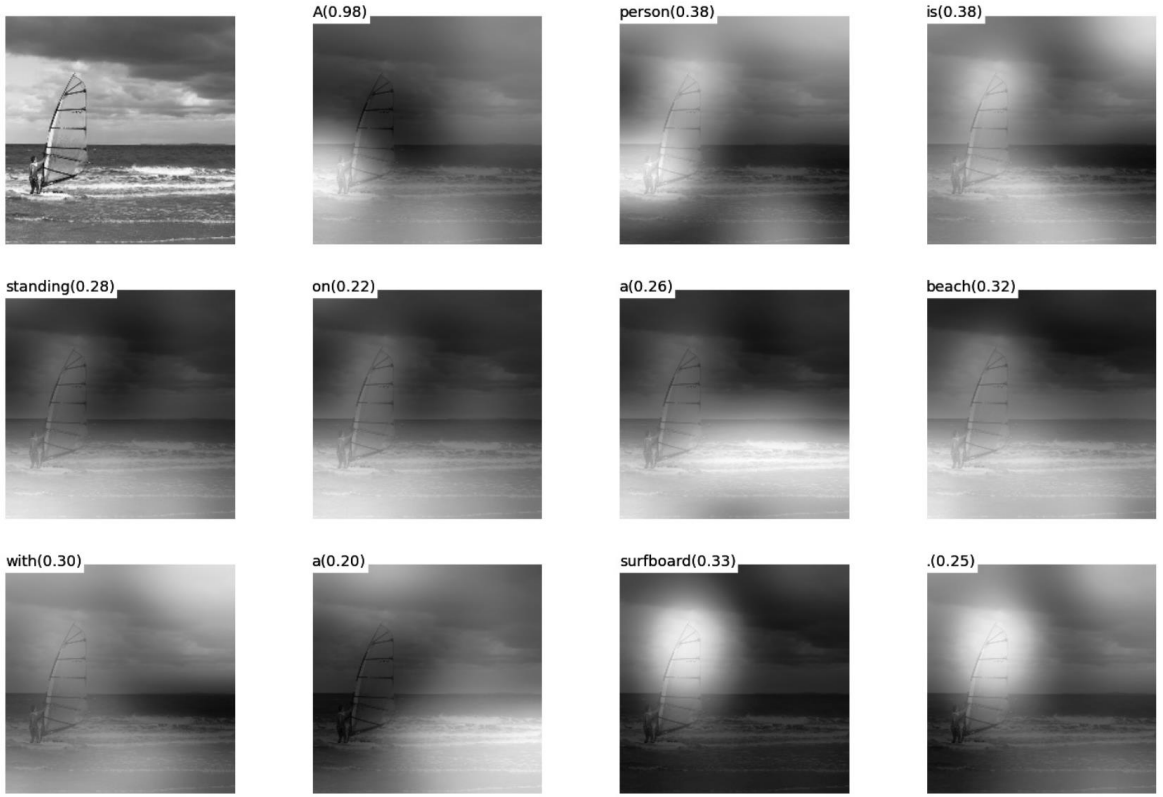
https://arxiv.org/pdf/1609.08144.pdf

LSTMS EXAMPLES – IMAGE CAPTIONING WITH ATTENTION



http://www.jmlr.org/proceedings/papers/v37/xuc15.pdf

LSTMS EXAMPLES – IMAGE CAPTIONING WITH ATTENTION



(b) A person is standing on a beach with a surfboard.

http://www.jmlr.org/proceedings/papers/v37/xuc15.pdf

SUNSPRING | A SCI-FI SHORT FILM

A Science Fiction short film starring Thomas Middleditch the entire script written by an LTSM Recurrent Neural Network predicting one character at a time ….