NLP: Sequence modelling

Sebastian Hofstätter

sebastian.hofstaetter@tuwien.ac.at
/s_hofstaetter

Today

NLP: Sequence modelling

1 Using Convolution (CNNs)

- N-gram modelling
- Character embedding

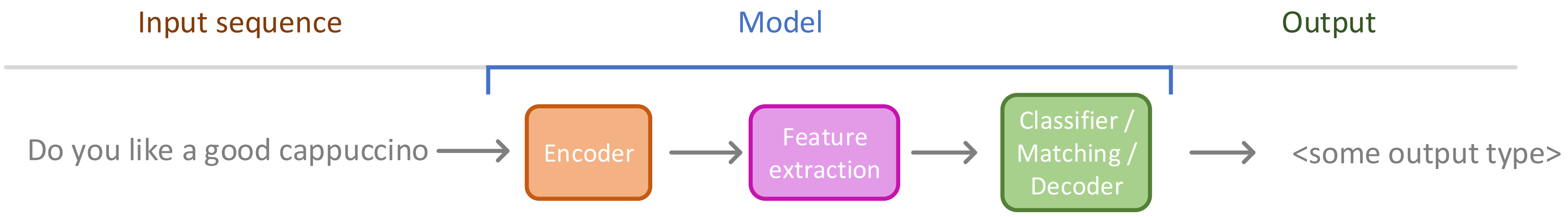
2 Using Recurrence (RNNs)

- Simple RNN & LSTM
- Encoder – Decoder architecture
- Attention

Based on: *Neural Network Methods for Natural Language Processing* by Yoav Goldberg https://twitter.com/yoavgo

Neural Networks are like LEGO blocks

- Around 2014, switch to nonlinear neural networks with dense input
- Neural Networks are like LEGO blocks
  - As soon as the tensor shapes match – you can combine modules!
  - The modules are then trained together (end-to-end)
  - Simplified example for text processing:



Representation Learning

Or how to use CNNs for text processing …

Representation Learning: Word N-Grams

“it was not good, it was actually quite bad”

== or !=

“it was not bad, it was actually quite good”

- The ordering & local context is important: “not good” vs. “not bad”
- Looking at N words at a time is called N-gram
- Creating bi-gram (2) or tri-gram (3) embeddings is not feasible
    - Sparsity problem
    - Not enough training data: no connection between “quite good” and “very good”

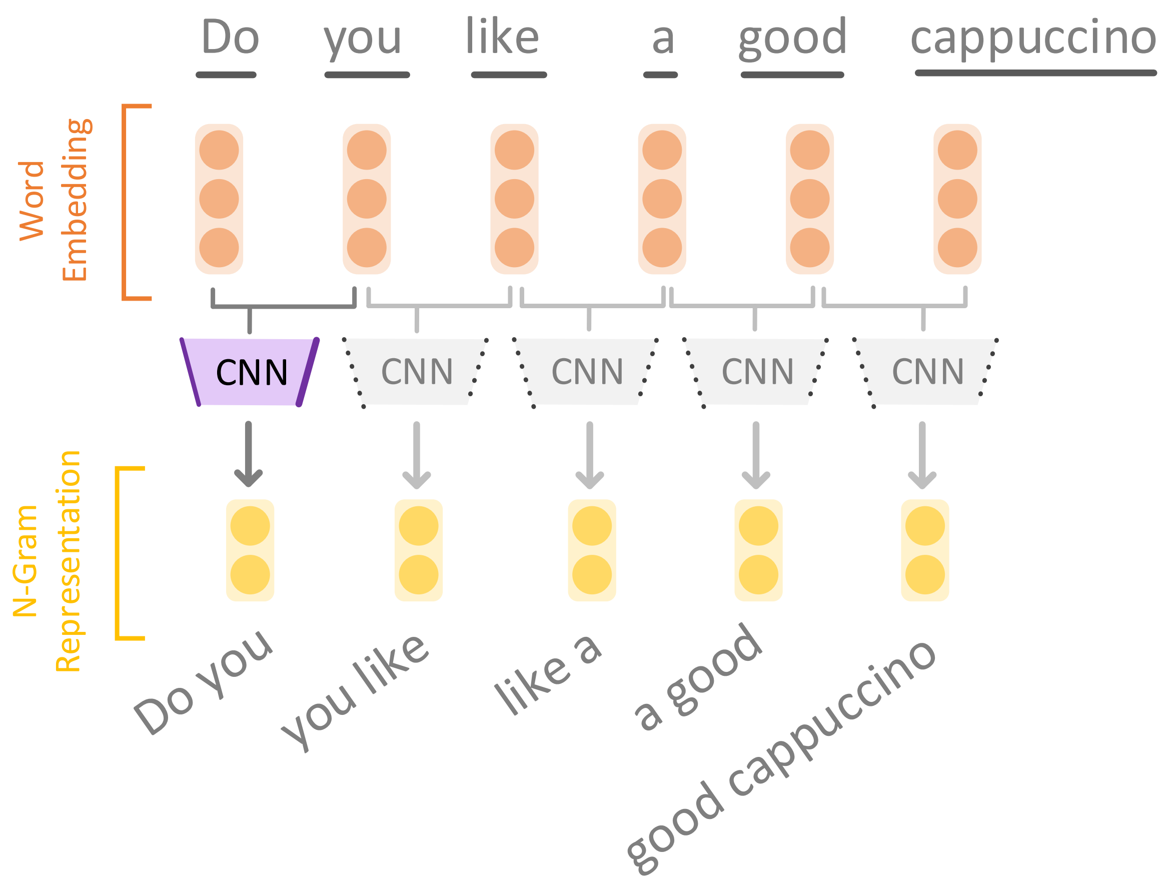
1D CNN

- 2D CNNs are ubiquitous in computer vision

- What are CNNs doing?
  - Applying a filter with a sliding window over the input data
  - Values in the filter region are merged into an output value -> guided by the filter
  - The filter parameters are learned during the training
  - Typically multiple filters are applied in parallel (made for GPU multiprocessing)

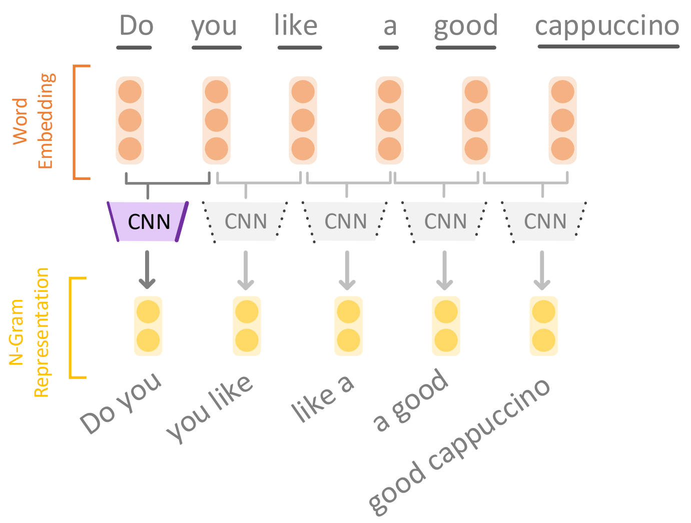
- 1D CNNs have a 1 dimensional filter and operate on 1 dimensional input

Modelling Word N-Grams with 1D CNNs



- Apply a 1D CNN on a sequence of word vectors
- N of N-grams = filter size
  - In this example N=2
- Output is a sequence of N-gram representations
  - Further used in other network components
- WE & CNN can be trained end-to-end

1D CNNs in PyTorch



- Definition:

```
conv = nn.Sequential(
          nn.ConstantPad1d((0,2 - 1), 0),
          nn.Conv1d(kernel_size=2,
                    in_channels=300,
                    out_channels=200),
          nn.ReLU())
```

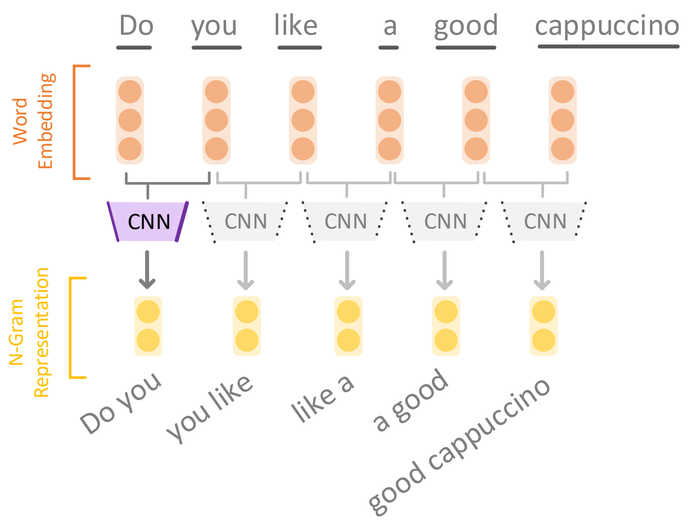
PyTorch Helper
Pad for same out dim.
The size of the window
Word embedding dim.
N-gram output dim
Activation function

- Forward:

```
embeddings_t = embeddings.transpose(1, 2)
n_grams = conv(embeddings_t).transpose(1, 2)
```

Documentation: https://pytorch.org/docs/stable/nn.html#convolution-layers

1D CNNs in PyTorch



- Forward:

Embeddings shape:
[batch, sequence_length, emb_dim]

```
embeddings_t = embeddings.transpose(1, 2)
n_grams = conv(embeddings_t).transpose(1, 2)
```

Transpose to nn.Conv1d required shape

nn.Conv1d requires tensor in shape:
[batch, emb_dim, sequence_length]

Optional transpose back to original layout:
[batch, sequence_length, conv1d_out_channels]

Documentation: https://pytorch.org/docs/stable/nn.html#convolution-layers

Pooling

- **Max-Pooling:** Retain only the strongest feature per region
- **Average-Pooling:** Retain the average over each feature per region

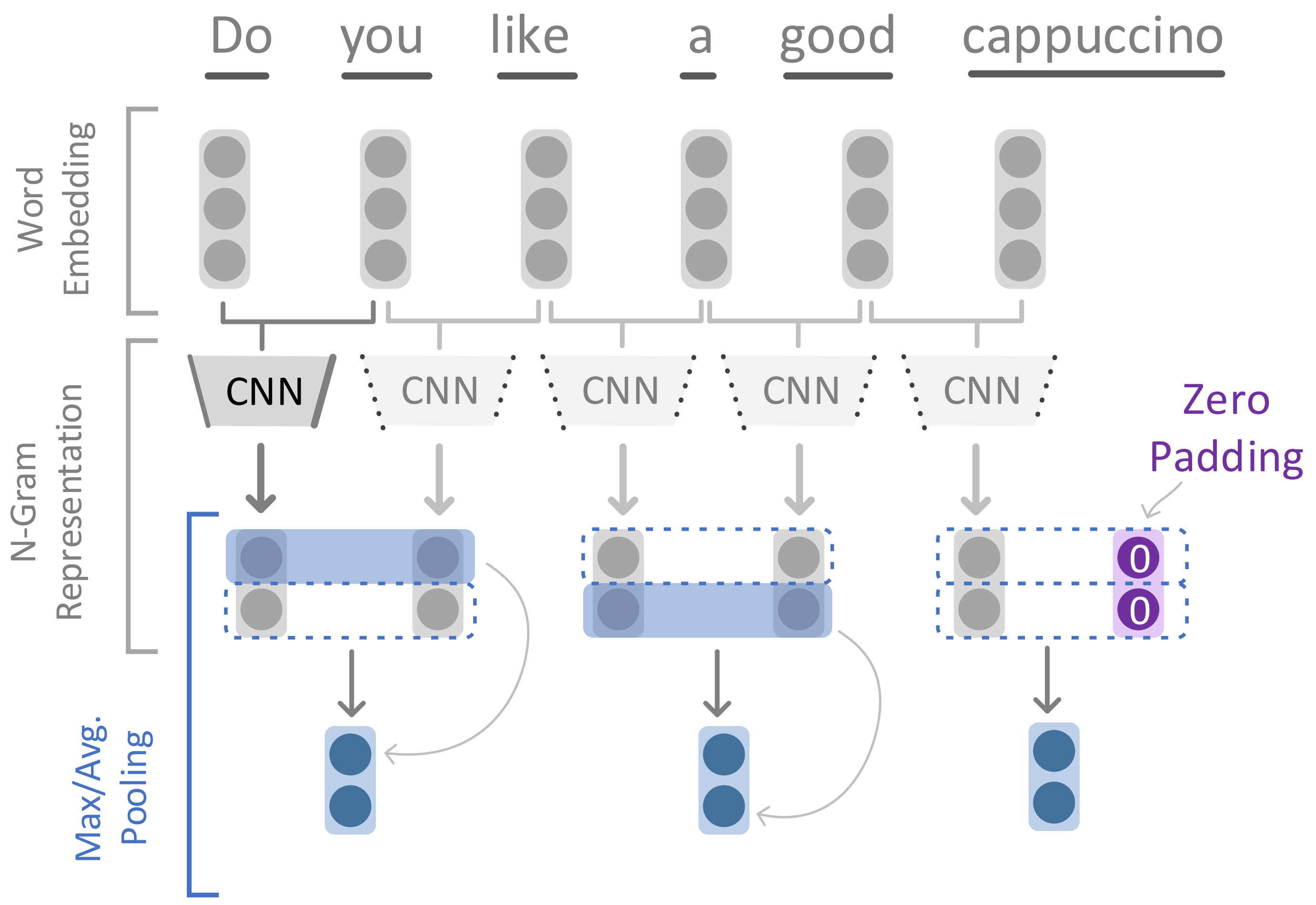
- Applied as sliding window over input sequence

- Does not train additional weights

- Commonly applied after a CNN

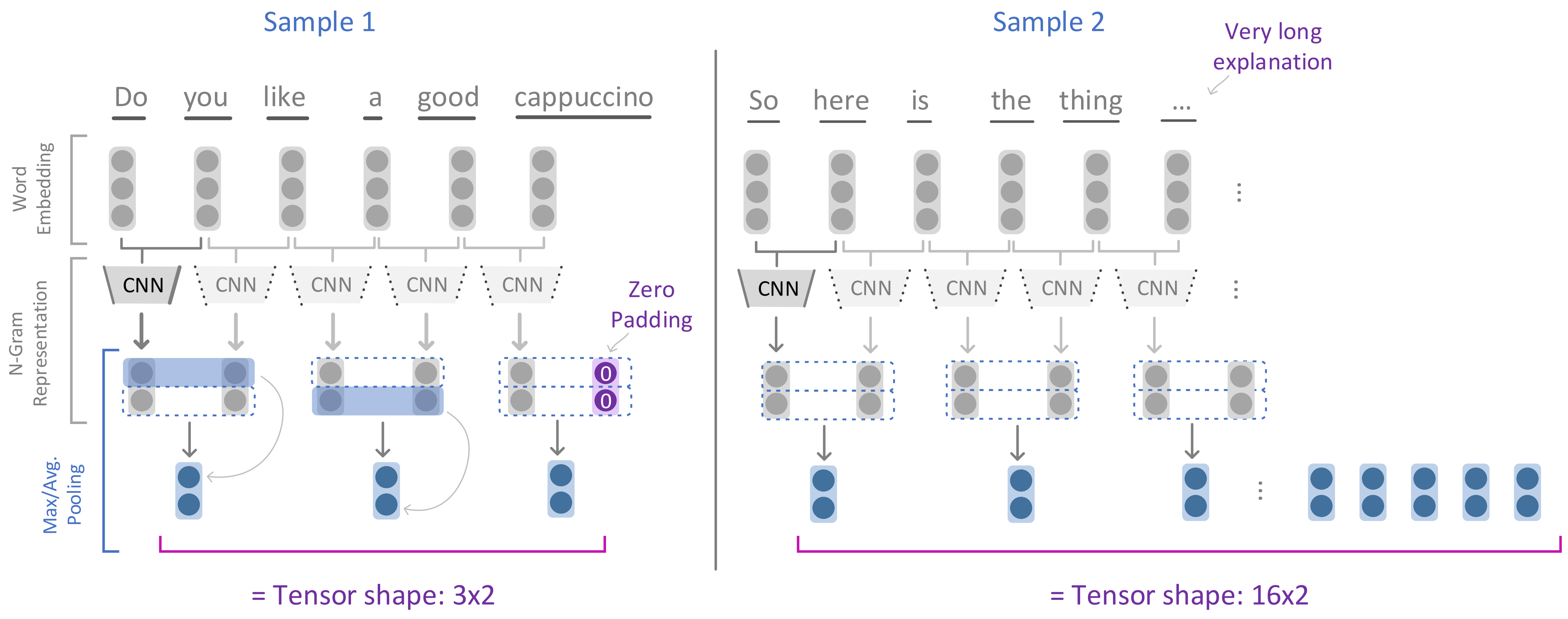
PyTorch documentation: https://pytorch.org/docs/stable/nn.html#maxpool1d

Pooling



- Max/Avg. is applied to every highlighted region
  - The output is a combination per feature (aka vector index)
- Without padding PyTorch ignores last element(s)
- Output size depend on the input

Pooling – Variable input length

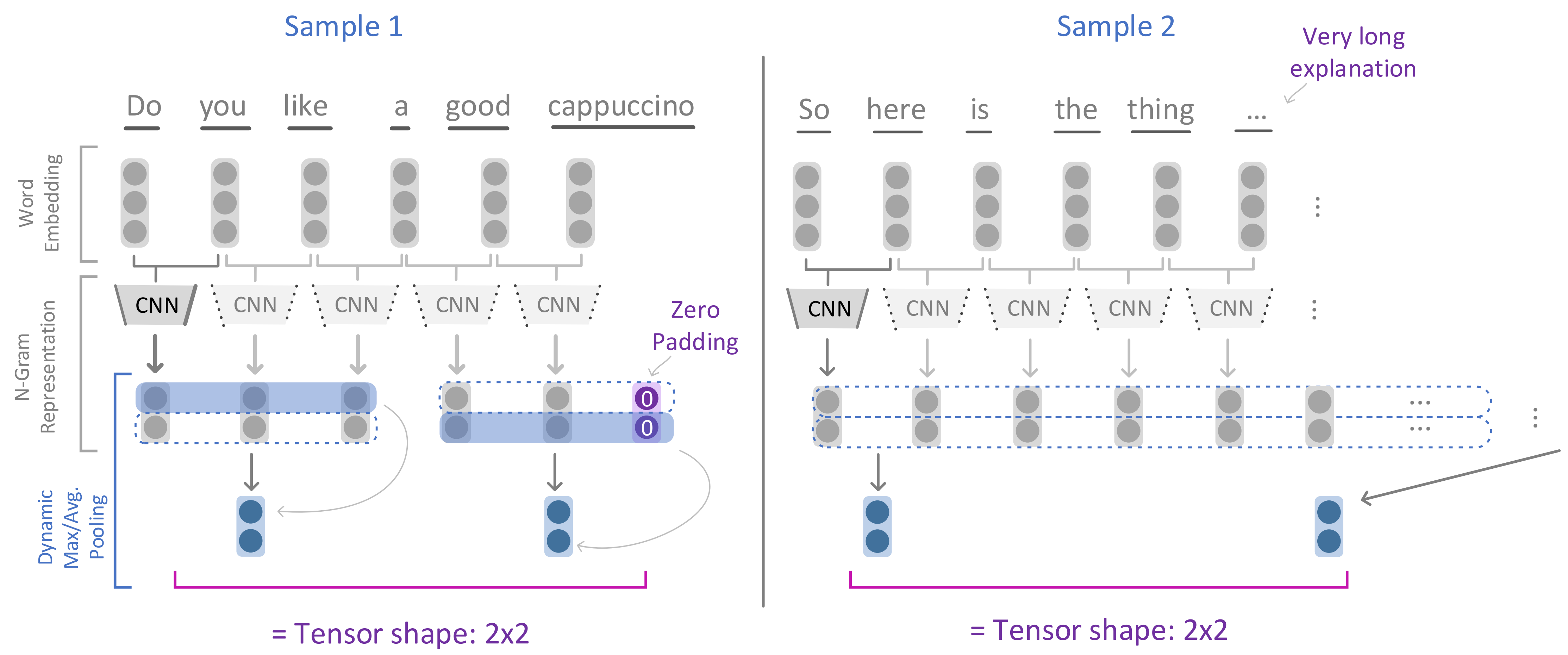


Adaptive (Dynamic) Pooling

- At some point in the network – switch from variable length to fixed length
  - Adaptive pooling is one option to achieve that
- The kernel size is a function of the input size, so that result is a fixed output size
- Available in PyTorch in Max/Avg. variants

Documentation: https://pytorch.org/docs/stable/nn.html#adaptivemaxpool1d

Adaptive (Dynamic) Pooling



Hierarchical CNNs

- Similar to computer vision models: stack multiple CNN-layers
- The output of the previous layer is the input of the next
- Can cover a broad range quickly
  - By increasing stride (sliding window step size)
  - Working with dilation (spacing between input points)

Cool tutorial & animations for different convolutions: https://github.com/vdumoulin/conv_arithmetic

Multiple CNNs in PyTorch (with a lot of flexibility)

- Definition:

```
self.convolutions = []
for i in range(1, max_conv):
  self.convolutions.append(
    nn.Sequential(
      nn.ConstantPad1d((0,i - 1), 0), # optional to keep the same dimensions
      nn.Conv1d(kernel_size=i, ...), # for example: change the kernel size
      # activation function (relu/tanh) and or pooling
  ))
# register conv as part of the model
self.convolutions = nn.ModuleList(self.convolutions)
```

- Forward:

```
for conv in self.convolutions:
  out = conv(in)
  # do something in between convolutions
```

Representation Learning: Character Embeddings

- Beyond Word2vec & word embeddings – many possibilities of encodings
  - “Word embedding” – index by word id & fixed vocabulary
  - Problem what to do with words not contained in the vocabulary (oov)

- Character embeddings: indexed by character id (one hot encoding)
  - CNNs produce character n-gram representations
    - Which can then be used in the rest of the network (just as a word embedding)
  - Might be used with or without tokenization (word boundaries)
  - Actual network implementation looks surprisingly similar to word embeddings

Zhang, X., Zhao, J. and LeCun, Y., 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*

Sequence Modelling with RNNs

Or the dawn of the leaderboard challenges

Recurrent Neural Networks

- Model global patterns in sequence data
  - Allow for arbitrarily sized inputs
- Trainable with backpropagation & gradient descent
  - Recursion gets unrolled to form a standard computation graph
- Ability to condition the next output on an entire sentence history
  - Allows for conditional generation models
- Of course not only useable with text data – but now it is our focus

Neural Network Methods in NLP, Chapter 14 (source of notations presented here)

Simple RNN

- The simplest RNN, which takes ordering of elements into account is:

$$s_i = R_{SRNN}(x_i, s_{i-1}) = g(s_{i-1}W^s + x_iW^x + b)$$

- s_i is the state of the RNN at position *i*
  - And it depends on the previous state s_{i-1} (recursive)
- W^s, W^x , b are trainable parameters

Elman, J.L., 1990. Finding structure in time. Cognitive science

s_i	RNN state / output at *i*
x_i	Input vector at *i* (from $x_{1:n}$)
b	Bias vector
W^s, W^x	Weight matrices
g	Nonlinear activation function (tanh, relu)

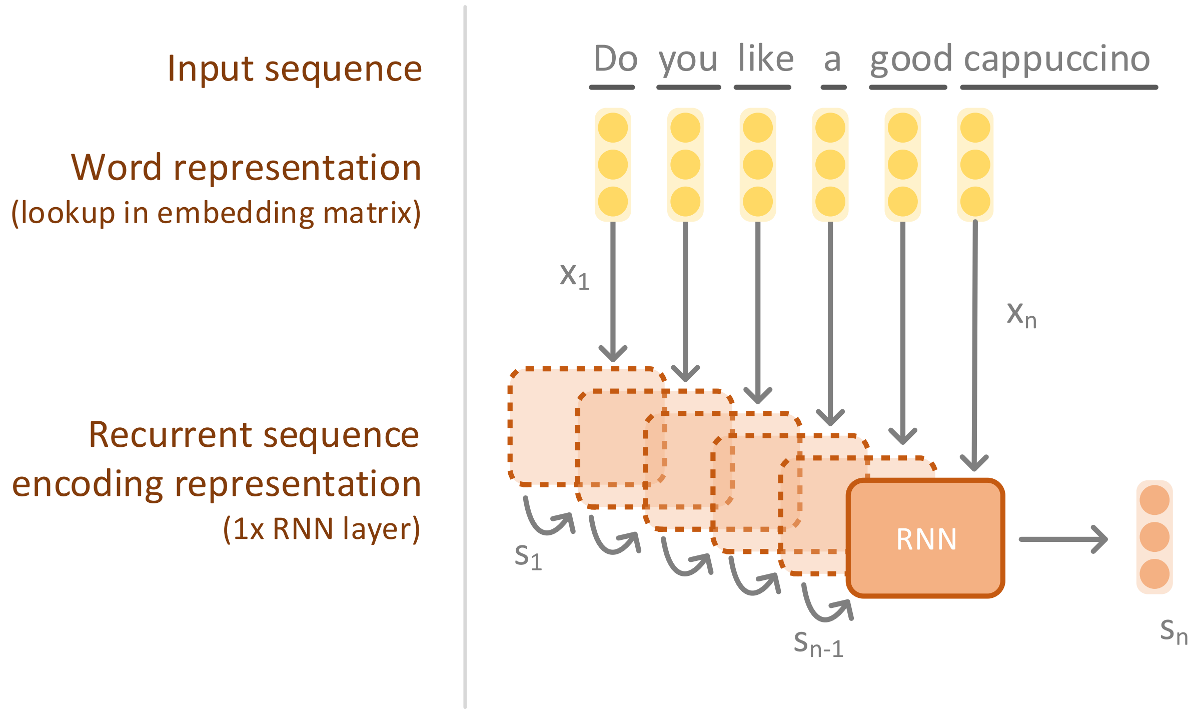
$s_i, b \in \mathbb{R}^{d_s}$

$x_i \in \mathbb{R}^{d_x}$

$W^s \in \mathbb{R}^{d_s \times d_s}$

$W^x \in \mathbb{R}^{d_x \times d_s}$

RNN as Encoder


Input sequence
Word representation
(lookup in embedding matrix)
Recurrent sequence encoding representation
(1x RNN layer)
Do you like a good cappuccino
x_1
x_n
RNN
s_1
s_{n-1}
s_n

- Sequence as the input
  single vector (last state) as output
  - Part of a larger network
- Unrolled computation graph visualized by shifted overlapping dotted boxes
  - The RNN is still one set of learnable parameters
- Optimally, s_n represents the full sequence*

*Well, that would be too easy, wouldn't it?

RNN flavor: LSTM

- Simple RNN suffers from the vanishing gradient problem
  - It “forgets” information from a few dozen steps ago
  - At each step of the computation, the entire memory state is read and written

- Long Short-Term Memory (LSTM) mitigates the vanishing gradient problem
- LSTM introduces gated memory access
  - LSTM splits the state vector s_i in two: “memory cell” & working memory
  - Still differentiable

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation

LSTM – Gating Mechanism

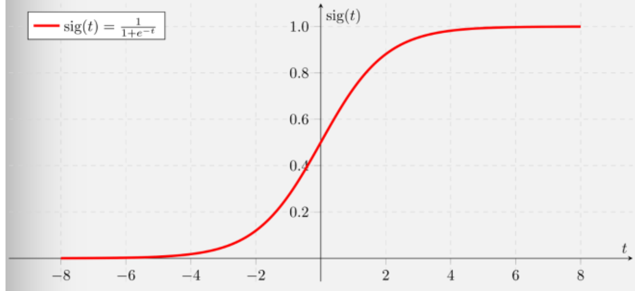
- Element-wise control of writing new values

$$s' = g \odot x + (1 - g) \odot s$$

$$\begin{matrix} 8 \\ 11 \\ 3 \end{matrix} = \begin{matrix} 0 \\ 1 \\ 0 \end{matrix} \odot \begin{matrix} 10 \\ 11 \\ 12 \end{matrix} + \qquad \begin{matrix} 1 \\ 0 \\ 1 \end{matrix} \odot \begin{matrix} 8 \\ 9 \\ 3 \end{matrix}$$

- Binary gate is not differentiable
- Make it differentiable: use $g' \in \mathbb{R}^n$ and bound & transform real numbers to sigmoid $\sigma(g')$
  - Range = (0, 1)
  - Most values at the borders

s	Memory (read)
s'	Memory (write)
g	Binary gate
x	Input (update)
$\odot$	Hadamard-product (element-wise multiplication) $a = b \odot c$ $a_{[i]} = b_{[i]} * c_{[i]}$
σ	Sigmoid activation



LSTM (As defined in Neural Network Methods in NLP)

$$s_j = R_{LSTM}(x_j, s_{j-1}) = [c_j; h_j]$$

$$c_j = f \odot c_{j-1} + i \odot z$$

$$h_j = o \odot tanh(c_j)$$

$$i = \sigma(x_j W^{xi} + h_{j-1} W^{hi})$$

$$f = \sigma(x_j W^{xf} + h_{j-1} W^{hf})$$

$$o = \sigma(x_j W^{xo} + h_{j-1} W^{hz})$$

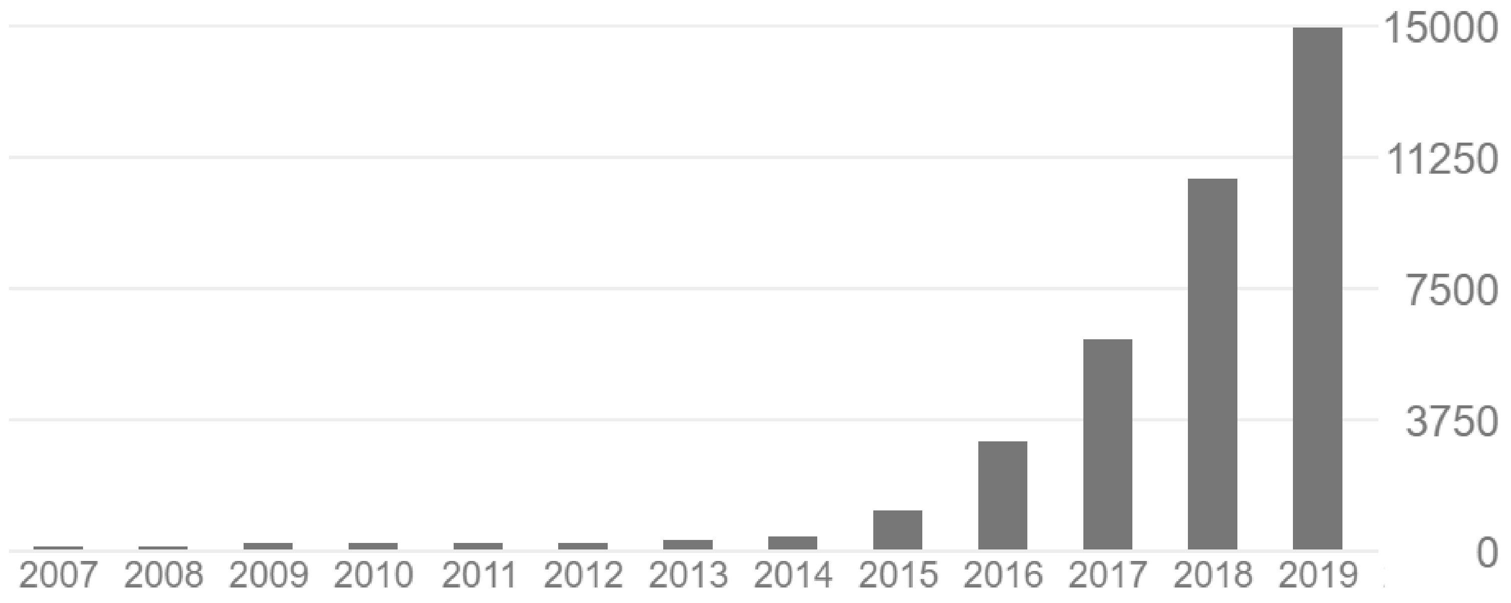
$$z = tanh(x_j W^{xz} + h_{j-1} W^{hz})$$

- Gates: ***i*** input, ***f*** forget, ***o*** output
- Memory: $\boldsymbol{c_j}$ Hidden state: $\boldsymbol{h_j}$

s_j	LSTM output at j
x_j	Input vector at j (from $x_{1:n}$)
z	Update candidate
$\odot$	Hadamard-product (element-wise multiplication)
W^*	Weight matrices
σ	Sigmoid activation
$[a; b]$	Vector concatenation

$s_j \in \mathbb{R}^{2d_h}$
$x_j \in \mathbb{R}^{d_x}$
$c_j, h_j, i, f, o, z \in \mathbb{R}^{d_h}$
$W^{h*} \in \mathbb{R}^{d_h \times d_h}$
$W^{x*} \in \mathbb{R}^{d_x \times d_h}$

LSTM Citations per Year



From Google Scholar: https://scholar.google.com/citations?user=tvUH3WMAAAAJ

Recurrent Neural Networks

- BiRNN – bidirectional RNN
  - Duplicate sequence & reverse 2nd & compute 2x + concatenate output per word
  - Useable with all RNN flavors (including LSTM)
  - In practice: bidirectional=true
- Multi-layer RNN – or Stacked RNN
  - Multiple RNNs: Use the output of the first as the input for the next
  - Can be combined with BiRNN
- Practical problems: A lot of details, lots of boilerplate stuff
  - Solution: AllenNLP library for PyTorch https://github.com/allenai/allennlp

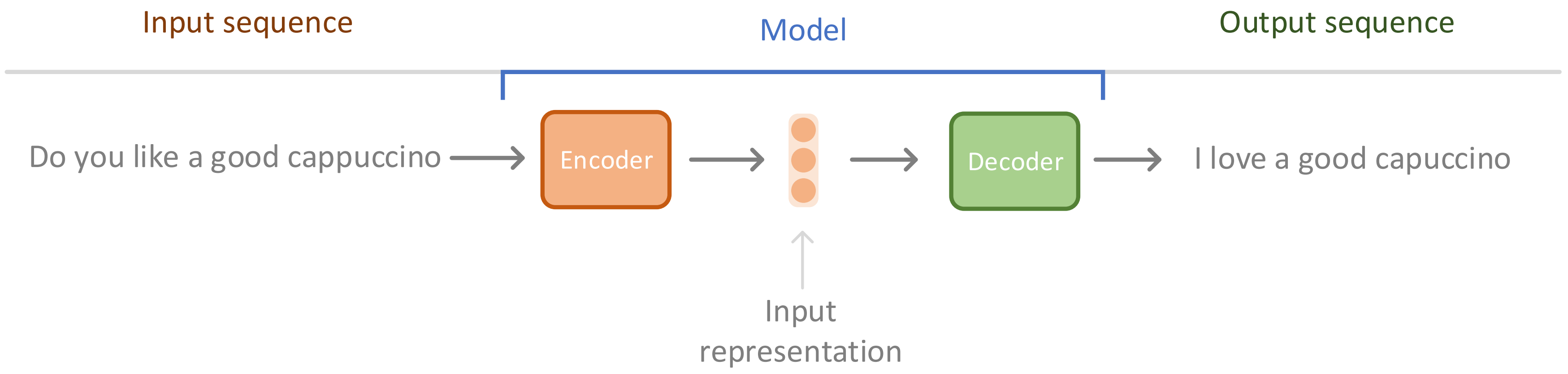
Sequence In + Out Tasks

- Translation
- Question Answering
  - SQuAD - https://rajpurkar.github.io/SQuAD-explorer/
- Summarization
- Email auto response
- Chatbots describing how much they like cappuccino

- Solutions for the tasks influence each other

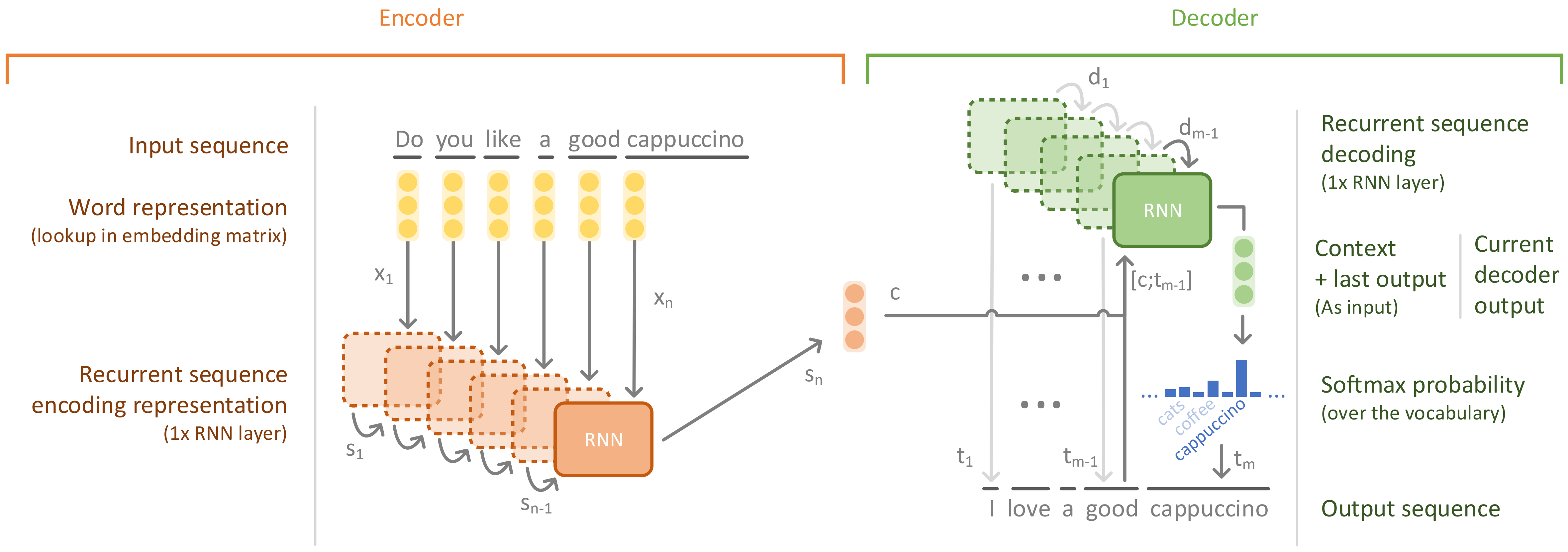
Encoder – Decoder Architecture

- Versatile architecture supporting sequence input & output
  - Based on the training data (and some tweaks) useable for different tasks



Sutskever, I., Vinyals, O. and Le, Q.V., 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*

Encoder – Decoder Architecture



Encoder – Decoder

- Encoder:

$$c = RNN_{Enc}(x_{1:n})$$

- Decoder:

$$p(t_v | t_{1:v-1}) = softmax(RNN_{Dec}([t_{1:v-1}; c]))$$

- Goal is best output sequence: $\arg\max_{T} p(T | x_{1:n})$

More details in Neural Network Methods in NLP, Chapter 17

c	Encoded input representation
$x_{1:n}$	Input sequence
$T, t_{1:m}$	Output sequence
t_v	Output at position *v*
$p(a\|b)$	Probability of a given b
$softmax$	Parameterized; normalized exponential function
$[a; b]$	Vector concatenation

Problems with the Plain Encoder – Decoder

- Sutskever et al. (translation) found: reversing the input sequence is best
  - *“While we do not have a complete explanation to this phenomenon, we believe that it is caused by the introduction of many short term dependencies to the dataset”*
- Simple greedy search for best output word does not maximize output sequence
  - ❶ Beam search
- The fixed-length context vector is a bottleneck
  - ❷ Attention
- What happens to out-of-vocabulary words (often names)?
  - ❸ Pointer generator

Generating the Output: Beam Search

- Each output word depends on the previous one
  - Taking more than one word into consideration: build up a tree
- General purpose heuristic graph search algorithm
- Only expands the most probable paths (threshold: beam-width/size)
  - Decoder – softmax assigns a probability to words (use that as heuristic)
  - Multiple runs of the next word, interactive
- Sutskever et al. on the performance-effectiveness tradeoff
  - *"Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12" (almost same effectiveness)*

https://en.wikipedia.org/wiki/Beam_search

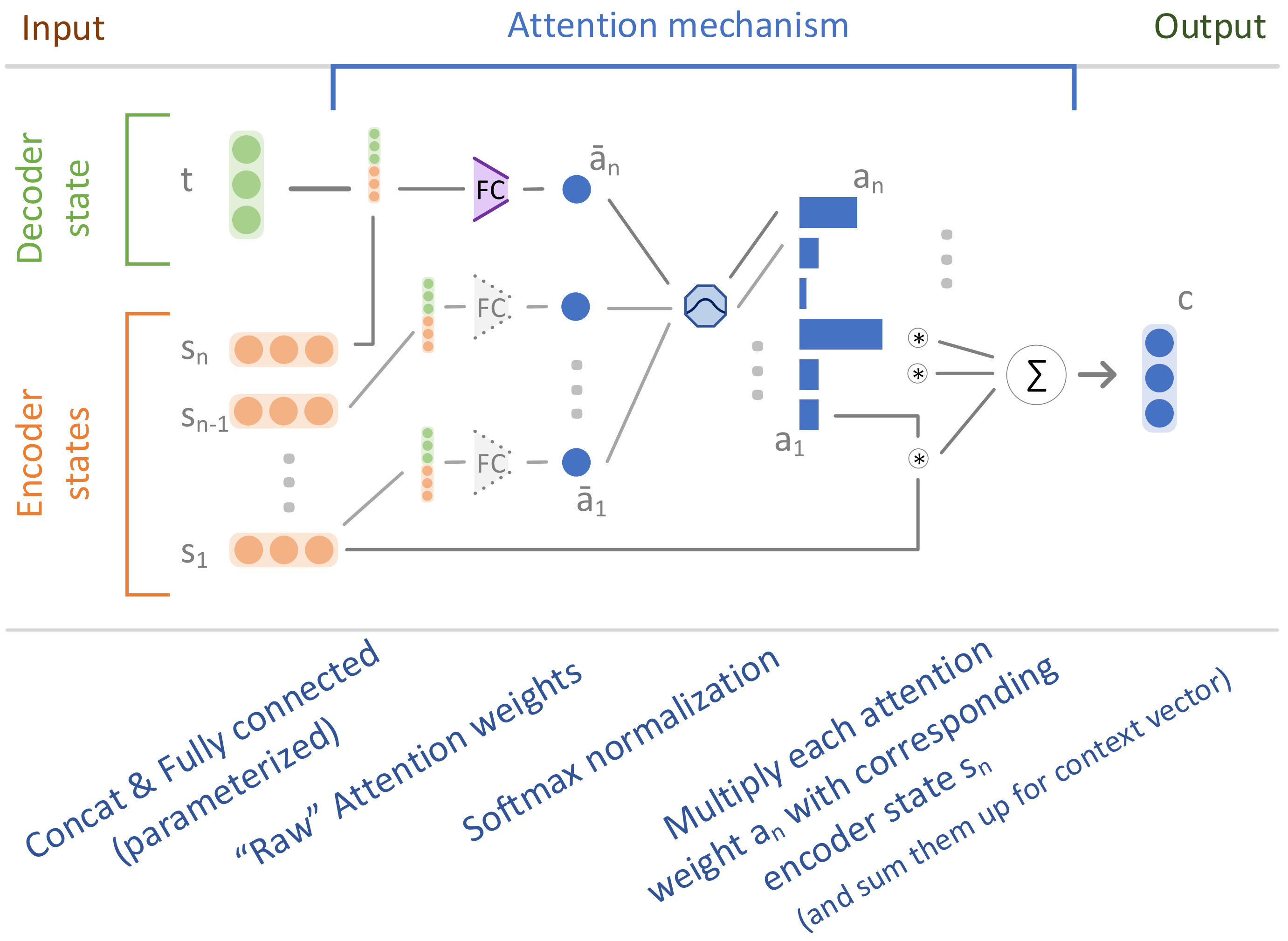
Implemented in AllenNLP: https://github.com/allenai/allennlp/tree/master/allennlp/state_machines

Attention

- Attention mechanism allows to search for relevant parts of the input
  - It creates a weighted average context vector
  - Weights are based on a softmax -> sum up to 1
  - Attention is parameterized & trained end-to-end with the model
- Attention is very effective and versatile
  - Now, there is a jungle of different versions and purposes
- Attention provides some interpretability
  - At least one can show which words have more impact for which output

Bahdanau, D., Cho, K. and Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate. In ICLR

Attention mechanism



- Each new decoder state produces a new context vector
  - Here, we show only one
- The encoder states are read-only memory
- Fully connected layer contains learned parameters & non-linear activation

Additive Attention

- The additive attention mechanism is defined as:

$$attend(s_{1:n}, t_j) = c^j$$

$$c^j = \sum_{i=1}^{n} a^j_{[i]} \cdot s_i$$

Sum up to 1

$$a^j = softmax\left(\bar{a}^j_{[1]}, \ldots, \bar{a}^j_{[n]}\right)$$

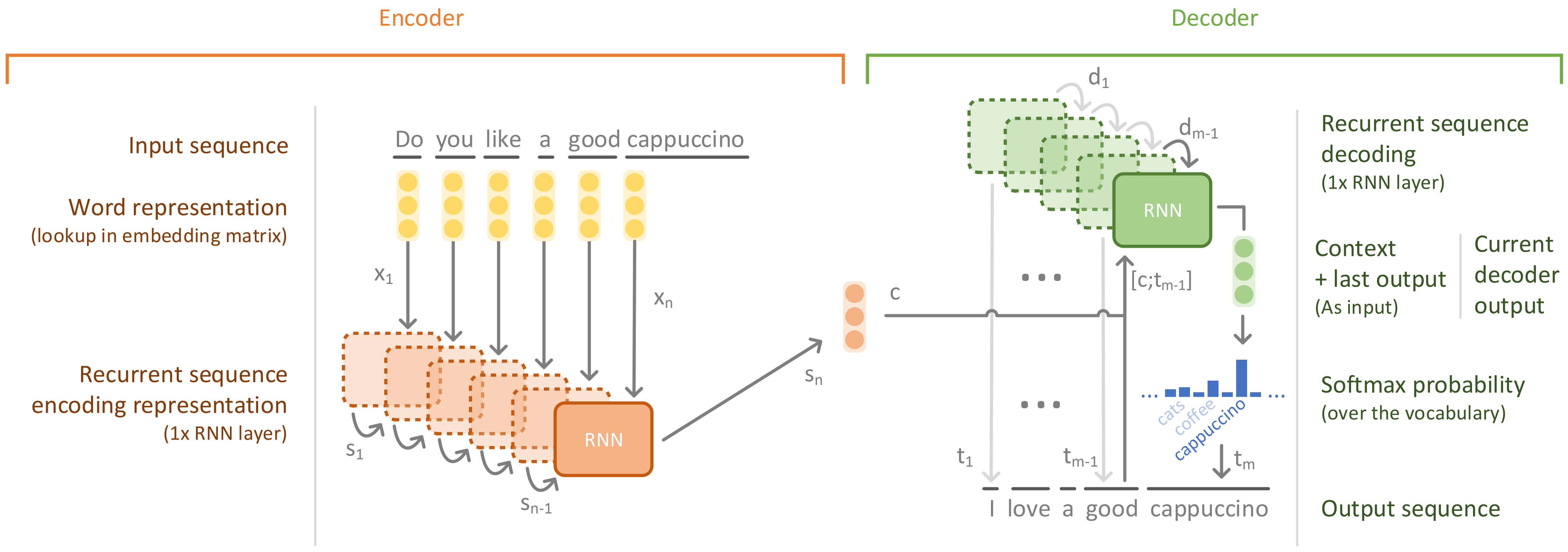
$$\bar{a}^j_{[i]} = \underbrace{v\, tanh([t_j; s_i]U + b)}_{\text{A single dense / fully connected / feed forward / MLP layer}}$$

- Many other attention varieties for $\bar{a}^j_{[i]}$ exist

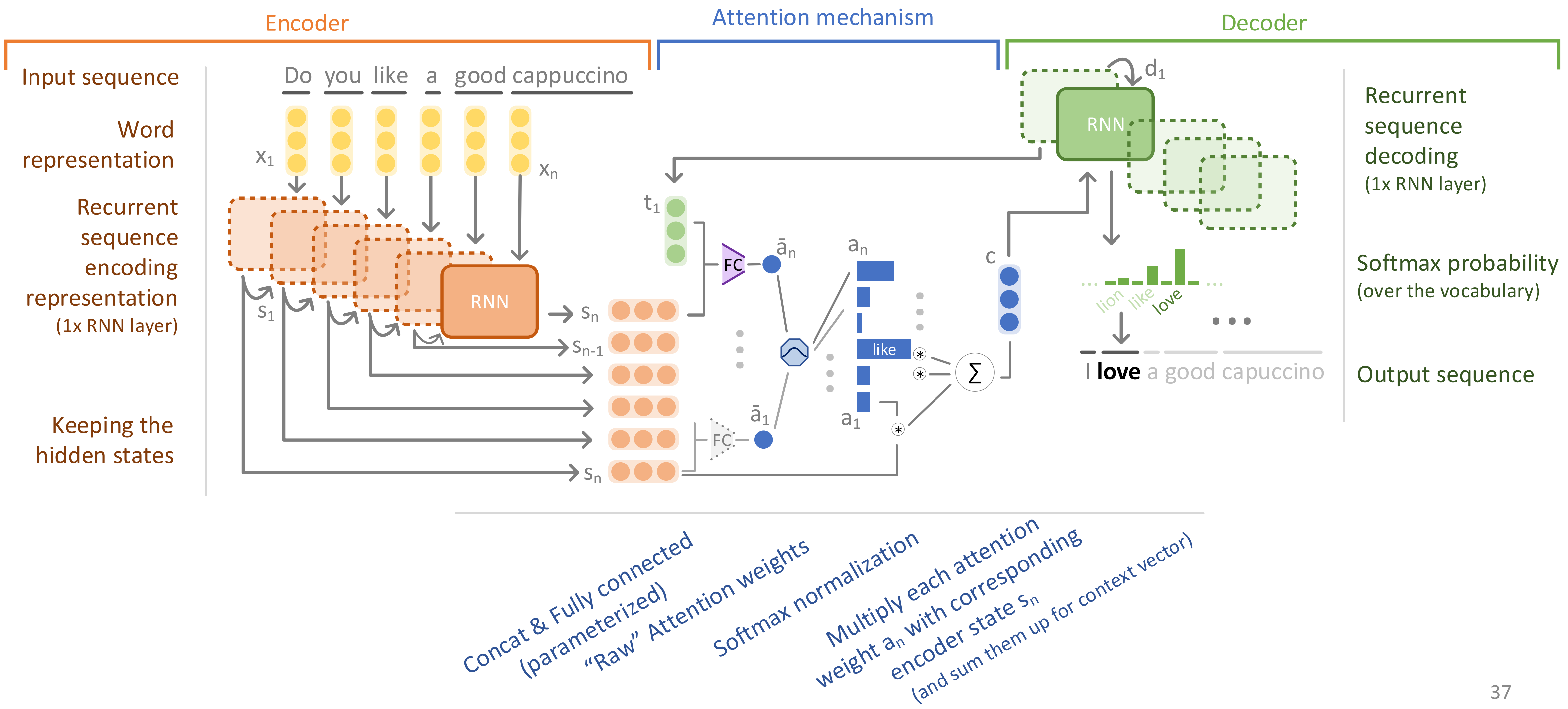
More details in Neural Network Methods in NLP, Chapter 17

$s_{1:n}$	Encoded input representation
c^j	Context vector
t_j	Decoder state at position *j*
$a_{[i]}$	Value at index I of vector a
v, U, b	Learnable parameters
$softmax$	Parameterized; normalized exponential function
$[a; b]$	Vector concatenation

Recall the Encoder – Decoder Architecture



Encoder – Decoder & Attention



Encoder – Decoder & Attention

- Encoder:

$$s_{1:n} = RNN_{Enc}(x_{1:n})$$

- Decoder:

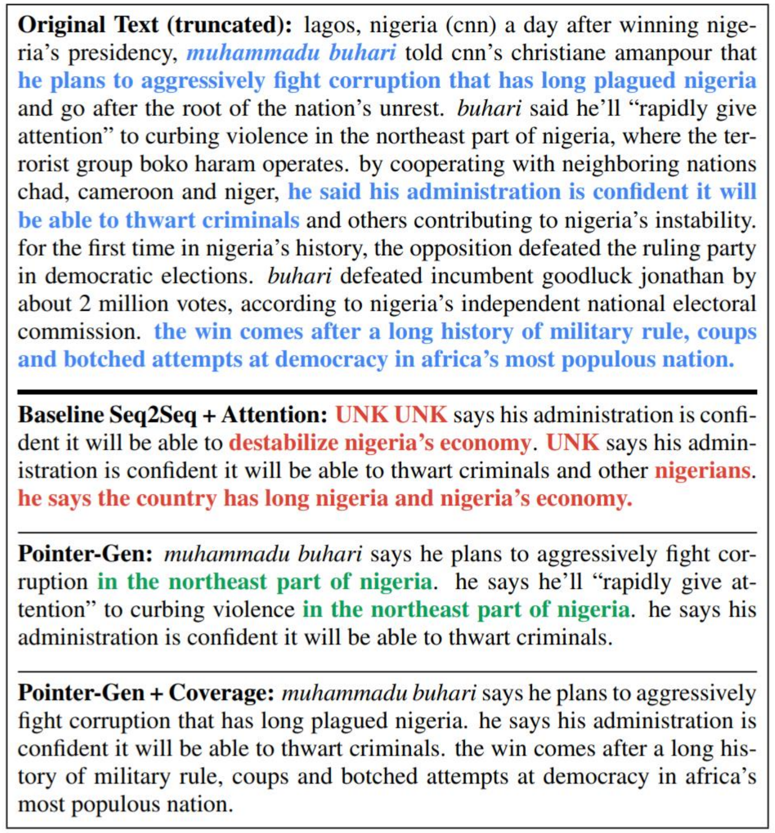
$$p(t_v | t_{1:v-1}) = softmax(RNN_{Dec}(attend(t_{v-1}, s_{1:n})))$$

- Goal is best output sequence: $\arg\max_{T} p(T | x_{1:n})$

More details in Neural Network Methods in NLP, Chapter 17

$s_{1:n}$	Encoded input representation
$x_{1:n}$	Input sequence
T	Output sequence
t_v	Output at position *v*
$p(a\|b)$	Probability of a given b
$softmax$	Parameterized; normalized exponential function
$[a; b]$	Vector concatenation

Pointer Generator (Summarization)



Original Text (truncated): lagos, nigeria (cnn) a day after winning nigeria's presidency, *muhammadu buhari* told cnn's christiane amanpour that **he plans to aggressively fight corruption that has long plagued nigeria** and go after the root of the nation's unrest. *buhari* said he'll "rapidly give attention" to curbing violence in the northeast part of nigeria, where the terrorist group boko haram operates. by cooperating with neighboring nations chad, cameroon and niger, **he said his administration is confident it will be able to thwart criminals** and others contributing to nigeria's instability. for the first time in nigeria's history, the opposition defeated the ruling party in democratic elections. *buhari* defeated incumbent goodluck jonathan by about 2 million votes, according to nigeria's independent national electoral commission. **the win comes after a long history of military rule, coups and botched attempts at democracy in africa's most populous nation.**

Baseline Seq2Seq + Attention: UNK UNK says his administration is confident it will be able to **destabilize nigeria's economy**. UNK says his administration is confident it will be able to thwart criminals and other **nigerians**. **he says the country has long nigeria and nigeria's economy.**

Pointer-Gen: *muhammadu buhari* says he plans to aggressively fight corruption **in the northeast part of nigeria**. he says he'll "rapidly give attention" to curbing violence **in the northeast part of nigeria**. he says his administration is confident it will be able to thwart criminals.

Pointer-Gen + Coverage: *muhammadu buhari* says he plans to aggressively fight corruption that has long plagued nigeria. he says his administration is confident it will be able to thwart criminals. the win comes after a long history of military rule, coups and botched attempts at democracy in africa's most populous nation.

- What about words that are out of the vocabulary?
  - In architectures shown today (depending on a fixed word embedding vocabulary) out-of-vocabulary words are replaced by "UNK"
  - Kind of prevents real world application of translation and summarization, etc …
- Pointer generator: Allow to pick words that are copied from the input
  - Network learns to when to pick

See, A., Liu, P. J., & Manning, C. D. (2017). Get To The Point: Summarization with Pointer-Generator Networks. In Proc. of *ACL*

Pointer Generator

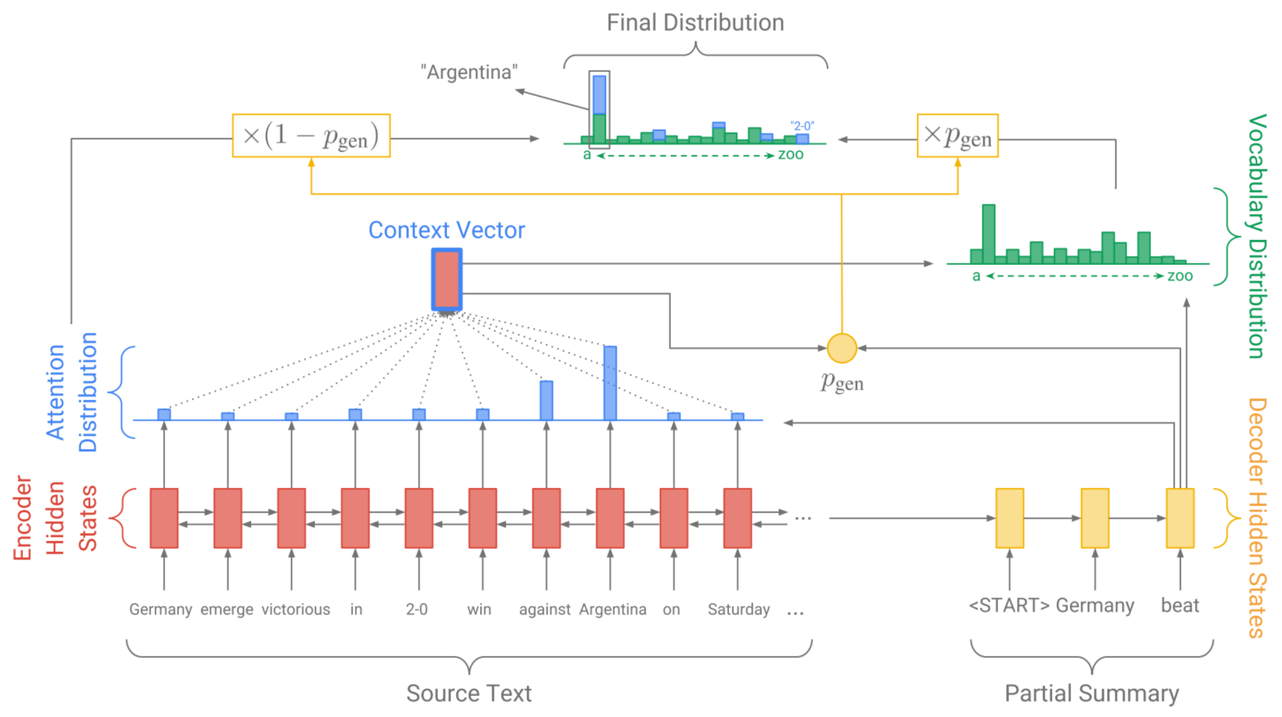


Figure 3: Pointer-generator model. For each decoder timestep a generation probability $p_{\text{gen}} \in [0,1]$ is calculated, which weights the probability of *generating* words from the vocabulary, versus *copying* words from the source text. The vocabulary distribution and the attention distribution are weighted and summed to obtain the final distribution, from which we make our prediction. Note that out-of-vocabulary article words such as *2-0* are included in the final distribution. Best viewed in color.

- The output vocabulary is the base word embedding + input tokens
- The "pointer" is a soft switch
  - Allows to copy or generate new words not in the input
- Trained end-to-end

Interested? Here is more …

- 2019 Stanford NLP with Deep Learning Course
  https://www.youtube.com/watch?v=8rXD5-xhemo&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z
- Distill – interactive, high quality articles:
  https://distill.pub/2016/augmented-rnns/ & https://distill.pub/2019/memorization-in-rnns/
- Neural Network Methods in Natural Language Processing
  by Yoav Goldberg (google for the pdf)

- Some cool & interesting NLP people to follow on twitter:
  nlpmattg, IAugenstein, honnibal, RadimRehurek, Tim_Dettmers,
  DynamicWebPaige, yoavgo

Summary: Neural Networks for NLP

1. Neural Networks are versatile – techniques are shared between tasks
2. CNNs can be utilized for n-gram representation learning
3. RNNs model sequences, for input and output

1. Neural Networks are versatile – techniques are shared between tasks
2. CNNs can be utilized for n-gram representation learning
3. RNNs model sequences, for input and output

Thank You