Applied Deep Learning

Day 3: Recurrent Neural Neworks

Motivation and context

- Recurrent neural networks are popular for sequence processing
  - Differ from feed forward neural networks, which are characterized by having some input produce a fixed-sized output by traveling through the hidden layers
  - Sequences have properties that make this feed forward framework unnatural
    - Arbitrary length
    - Desired output can be a single value or another sequence
    - Temporal dependencies in data rather than spatial

Motivation and context

- An RNN can be trained to generate an accurate caption for an image

a group of people standing around a room with remotes
logprob: -9.17

a young boy is holding a baseball bat
logprob: -7.61

a cow is standing in the middle of a street
logprob: -8.84

https://cs.stanford.edu/people/karpathy/sfmltalk.pdf

Motivation and context

- An RNN can be trained to generate an ok caption for an image

a man standing next to a clock on a wall
`logprob: -10.08`

a young boy is holding a baseball bat
`logprob: -7.65`

a cat is sitting on a couch with a remote control
`logprob: -12.45`

https://cs.stanford.edu/people/karpathy/sfmltalk.pdf

Motivation and context

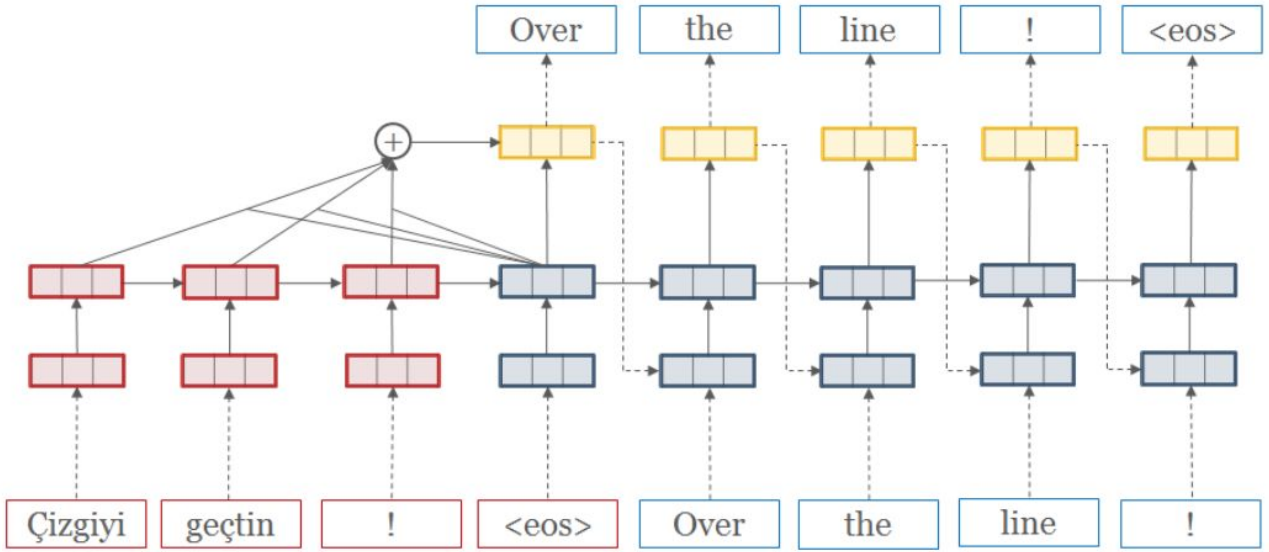
- An RNN can be trained to generate a bad caption for an image

a woman holding a teddy bear in front of a mirror
logprob: -9.65

https://cs.stanford.edu/people/karpathy/sfmltalk.pdf

Motivation and context

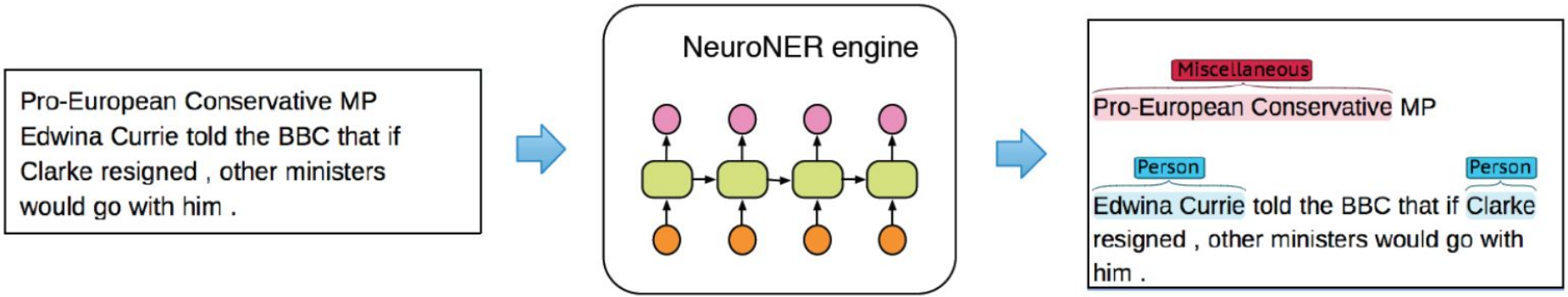
- An RNN can be trained to translate text between two languages



http://opennmt.net

Motivation and context

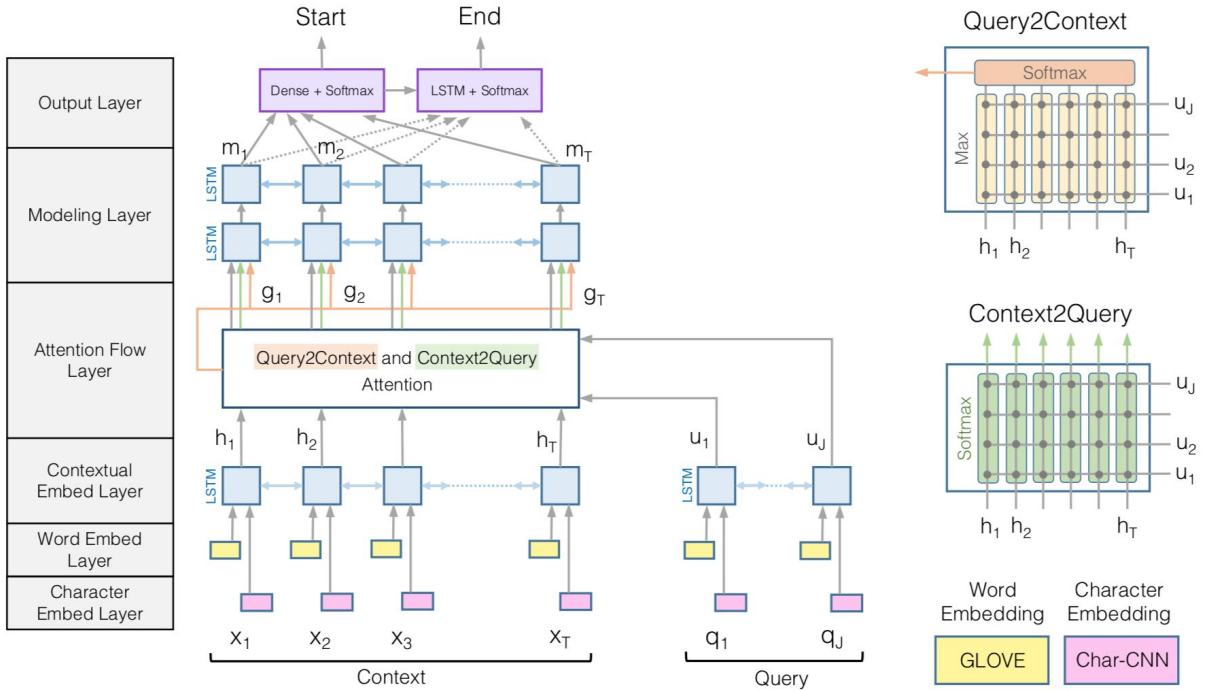
- An RNN can be trained to tag some text with the parts of speech or named entities



http://neuroner.com

Motivation and context

- An RNN can be trained to look up the answer to a question in a piece of text



Seo et al., Bi-Direction Attention Flow for Machine Comprehension, https://arxiv.org/pdf/1611.01603.pdf

Recurrent Neural Network Structure

Sequential data

- With the sequential data we want to be sensitive to dependencies in the data
  - Dependent data aren't necessarily close spatially
  - Relevant data could be sparse w.r.t the entire sequence
    - Only a few key points could be relevant out of the total sequence
- Sequences can be of varying length
- May want an output from every discrete time step in the sequence
- May want output only after the entire sequence is processed
- May want a sequence as an output after processing some fixed sized input

Sequential data

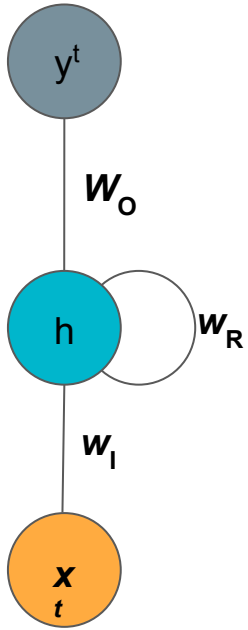
- Convolutional network can handle these types of data but they require some tweaking
  - The size of the kernel of a convolution layer will determine how many time steps to process at each stride
  - Feature map will become some condensed representation of the sequence based on the kernel operations

Sequential data

- We design a neural network with the assumption that inputs are not independent from each other
  - Need to have some mechanism to capture the information in a sequence
  - Use an internal state to create meaningful representations of the data as the sequence is processed
    - Maintains representations of past data to influence representations of current data

Recurrent Neural Networks

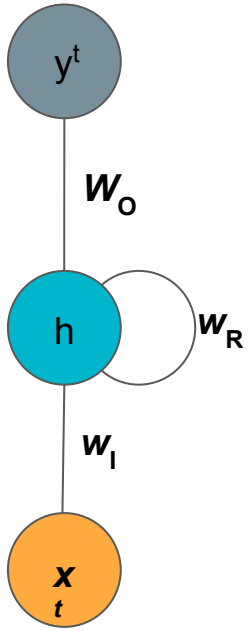
- Input does not need to be fixed size
- Can create output at arbitrary places along the sequence
  - Output is dependent only on the hidden state
- Maintains three weight matrices



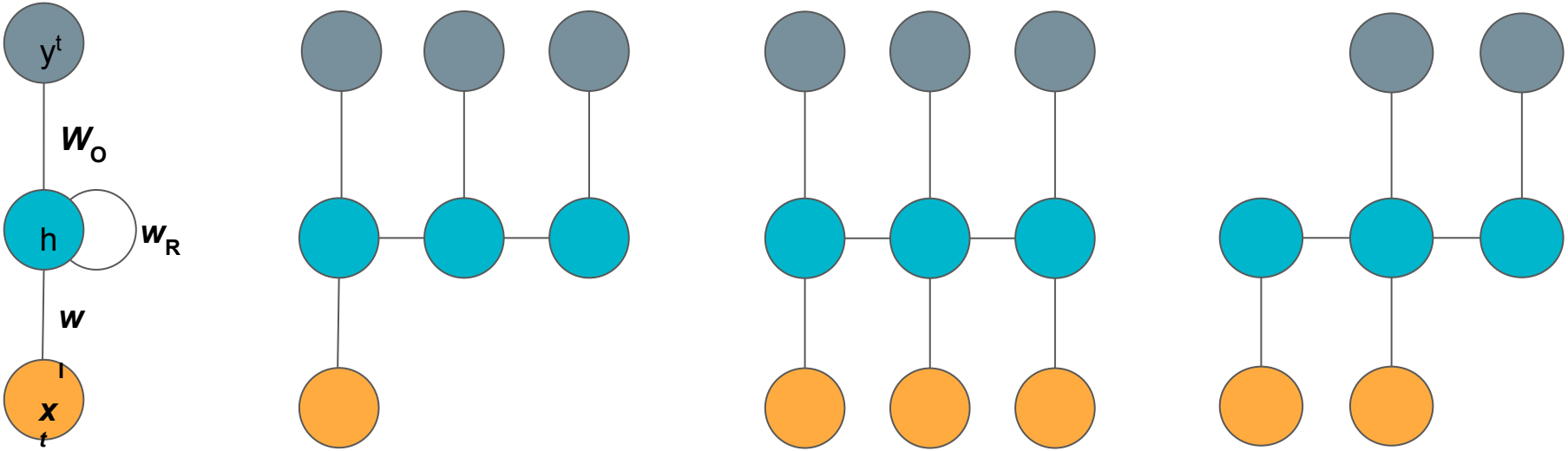
Recurrent Neural Networks

$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

$$y^t = a_y(W_o h^t + b_y)$$



Recurrent Neural Networks

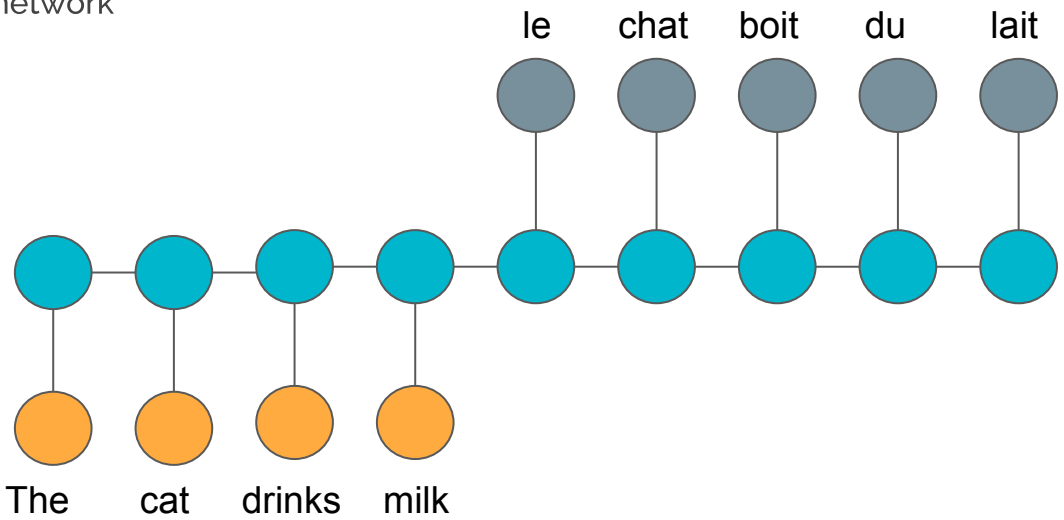


Representation Learning

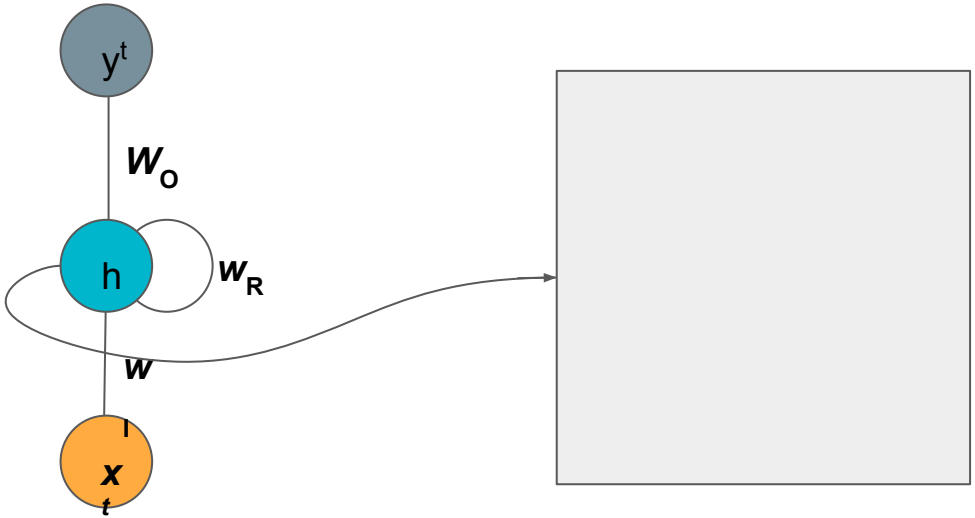
- In the context of NLP the hidden state of the RNN will encode a representation of the semantic meaning and context within some piece of text
  - An RNN designed to generate sentences will learn grammar rules
  - An RNN designed to generate words will learn appropriate uses of consonants and vowels

Representation Learning - Encoder Decoder Network

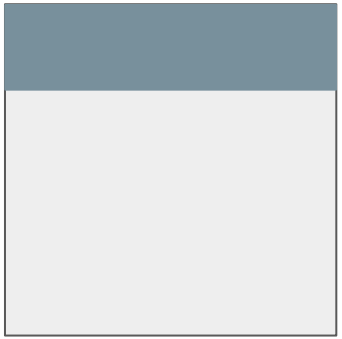
- A common RNN framework for language translation is an encoder-decoder network
- Highlights the efficacy of using an RNN for language representation learning
  - One RNN encodes a sentence in one language into a vector representation
  - A second RNN creates a new sentence in another language based on the encoded representation of the first network



Representation Learning



Representation Learning



$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

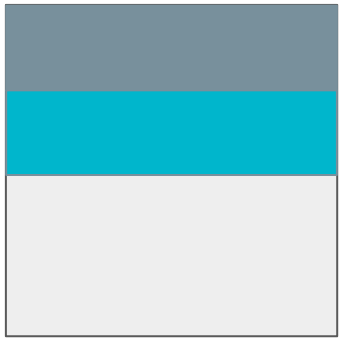
The

cat

drinks

milk

Representation Learning



$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

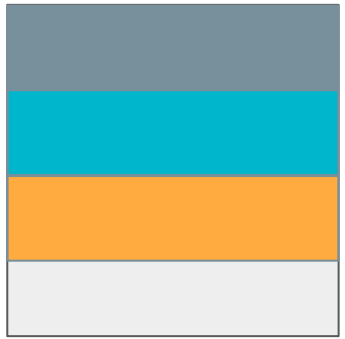
The

cat

drinks

milk

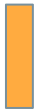
Representation Learning



$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

The

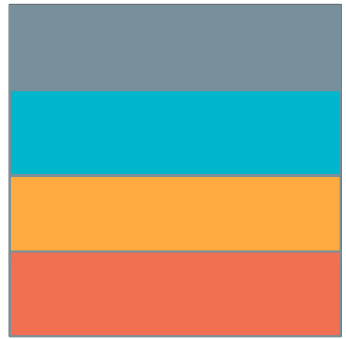
cat



drinks

milk

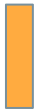
Representation Learning



$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

The

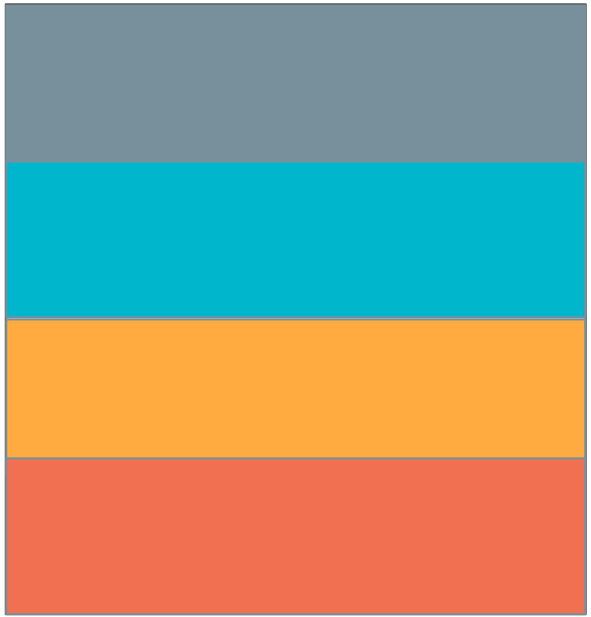
cat



drinks

milk

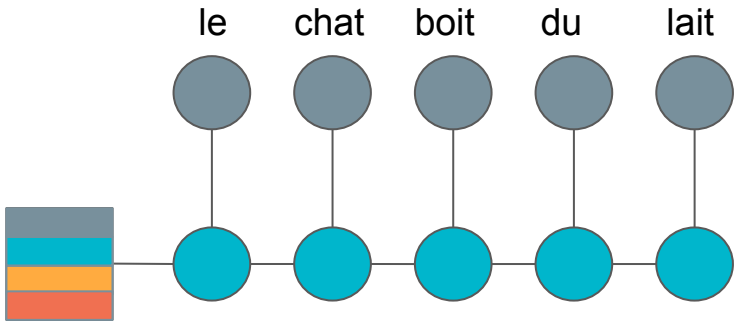
Representation Learning



- This hidden state is now a dense representation of the sentences after it's been processed by the RNN
  - This representation contains both semantic and contextual information

Representation Learning

- The hidden state representation of the sentence is then used as the seed hidden state for the decoder RNN to generate the same sentence in another language



Representation Learning

- The two weight matrices involved in creating the hidden state of the RNN will update so that the representations being stored are meaningful for the given problem
    - Much like we saw in ConvNets with the kernels becoming image feature extractors

$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

Hands on TensorFlow: Create a RNN from scratch

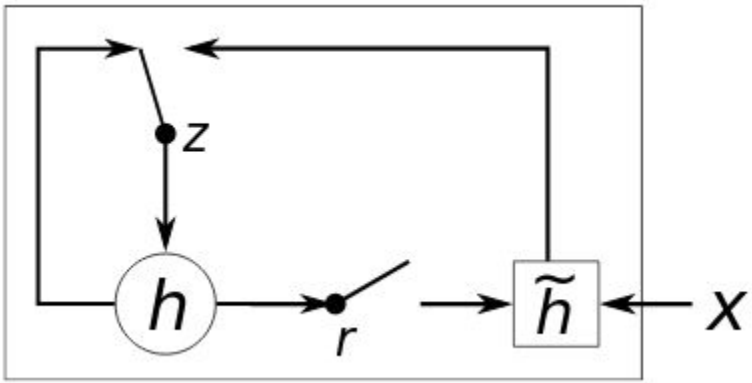
RNN Shortcomings

- There is an obvious shortcoming to the structure of a vanilla RNN
  - The hidden state is always updated every time we process input data
    - What if the current data is not useful?
    - What if we've already processed all the useful data and updating the weights will disturb our meaningful representations?

$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

Gated recurrent unit

- A better way to update the hidden state would be through the use of ‘gates’
  - Gates can be open or closed depending on whether or not we want to store representations of the current data
  - Gates can also dictate how much hidden state information is considered on each timestep
- A Gated Recurrent Unit (GRU) is one example of an architecture that solves these issues



Cho et al., Learning Phrase Representations using RNN Encoder-Decode for Statistical Machine Translation, https://arxiv.org/pdf/1406.1078.pdf

Gated recurrent unit

$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

Gated recurrent unit

$$h^t = a_h(W_I x^t + W_R h^{t-1} + b_h)$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

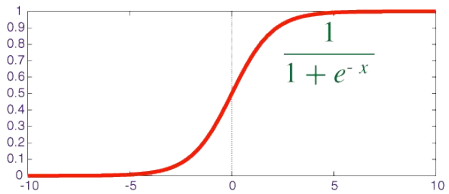
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

$$\tilde{h}_t = tanh(W_h x_t + U_h(r_t * h_{t-1}) + b_h)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Gated recurrent unit

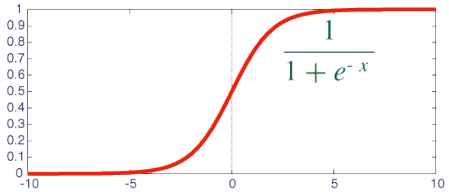
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$



Sigmoid activation

Gated recurrent unit

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

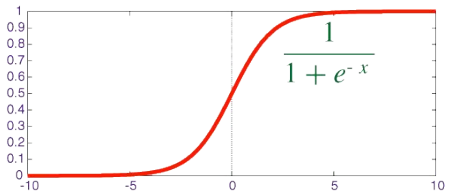


Sigmoid activation

Weights for input data [input_size, hidden_size]

Gated recurrent unit

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$



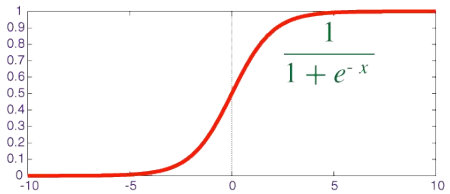
Sigmoid activation

Weights for input data [input_size, hidden_size]

Weights for previous hidden state [hidden_size, hidden_size]

Gated recurrent unit

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$



Sigmoid activation

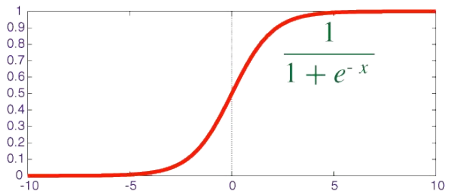
Weights for input data [input_size, hidden_size]

Weights for previous hidden state [hidden_size, hidden_size]

Bias vector [hidden_size]

Gated recurrent unit

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$



Sigmoid activation

Weights for input data [input_size, hidden_size]

Weights for previous hidden state [hidden_size, hidden_size]

Bias vector [hidden_size]

Produces a tensor of the same shape as the hidden size of floats between 0 and 1.

Gated recurrent unit

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

- The *r* gate is set up exactly like the *z* gate
  - Make sure that the *r* gate has all it's own weights
    - The gates don't reuse or share weights

Gated recurrent unit

$$\widetilde{h}_t = tanh(W_h x_t + U_h(r_t * h_{t-1}) + b_h)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h}_t$$

- Modulating the hidden state requires creating an intermediate hidden state
  - The output of the reset gate will allow the network to drop any information from the previous hidden state when processing the current input
  - The update gate modulates how much of the previous hidden state carries over into the new hidden state

LSTM

- What about LSTM (long short term memory)?
  - LSTM and GRU have similar goals and are very similar
  - LSTM has one additional gate
    - Understanding GRU makes understanding LSTM very easy

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

https://en.wikipedia.org/wiki/Long_short-term_memory

Hands on TensorFlow: Create a GRU from scratch

Word Embeddings

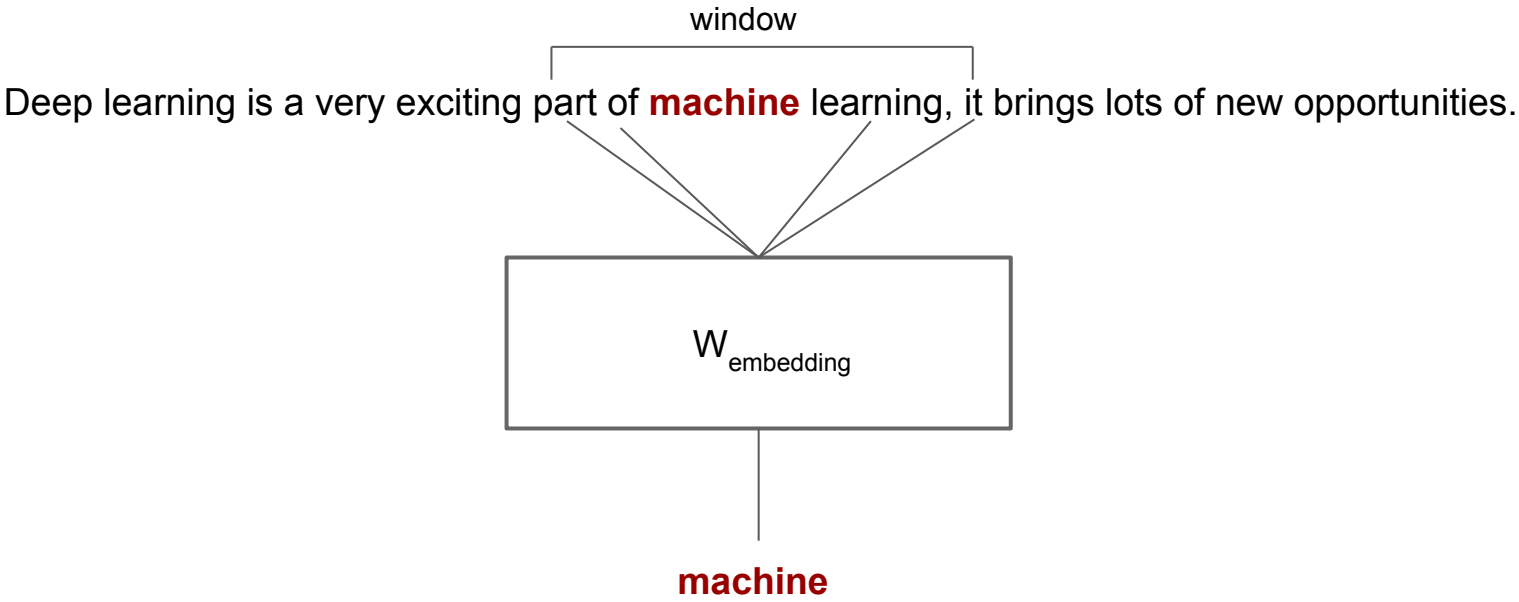
Word Embeddings

- Word embeddings are the building block of most neural network based NLP systems
- A representation learning system designed to represent semantic information as dense vectors
  - Easily handle; synonyms, categories (names, vehicles, animals, …), antonyms, and others
  - Multiple ways to learn representations
    - Shallow neural network with discrete words(word2vec)
      - https://arxiv.org/abs/1301.3781
    - Shallow neural network with subword components (fasttext)
      - https://arxiv.org/pdf/1607.04606.pdf
    - Factorize a co-occurrence matrix (GloVe)
      - https://nlp.stanford.edu/pubs/glove.pdf

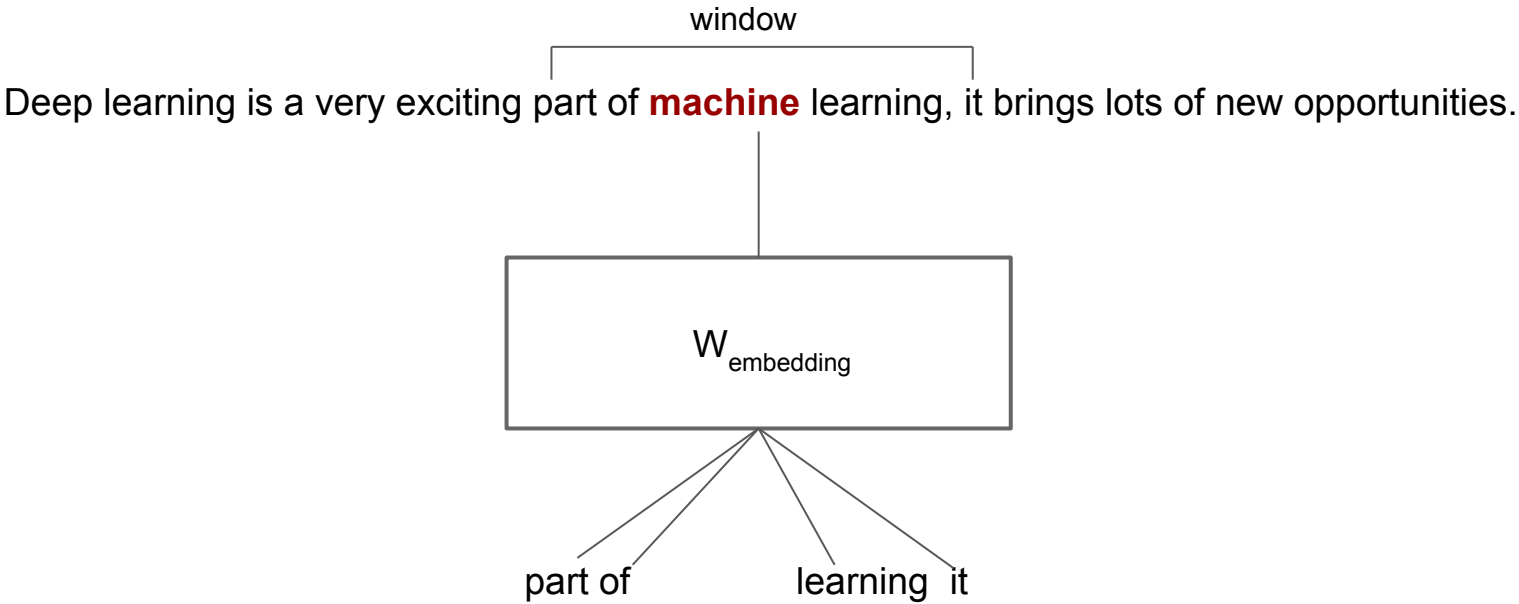
word2vec

- The insight that made these techniques possible
  - Words that appear near each other or within the same sentences are often related in some way
  - Their relationships can be exploited by setting up a problem where we try to predict some word based on its surrounding context
- Two ways to achieve this
  - Continuous bag of words
  - Skip-gram

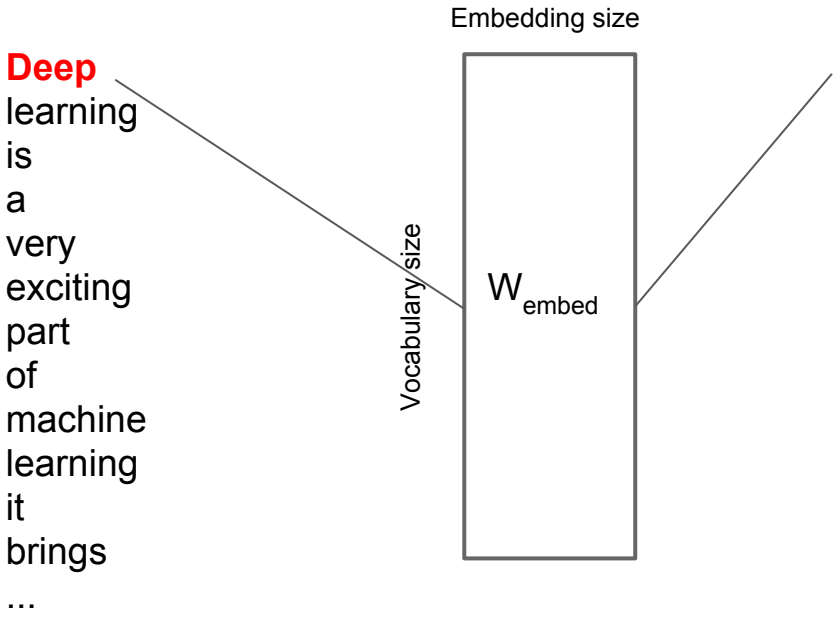
Word2vec - continuous bag of words



Word2vec - skip-gram



Word2vec - retrieving embeddings



Deep = [0.7 0.2 0.8 0.4 ...]

Once the model is trained, the $W_{embedding}$ matrix turns into a lookup table.

To retrieve the embeddings we just take each row vector that corresponds to each word in the embedding matrix.

The shape of the embedding matrix is [n_vocab, embed_size].

Word2vec - finding similar words

```
>>> model.most_similar('car')
    [(u'vehicle', 0.7821096181869507),
     (u'cars', 0.7423830032348633),
     (u'SUV', 0.7160962820053101),
     (u'minivan', 0.6907036304473877),
     (u'truck', 0.6735789775848389)]
```

$$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

By using the cosine similarity between the vectors representing each word we can query the model for the most similar words.

If we search for the words that are most similar to car we get; vehicle, cars, SUV, minivan and truck.

This type of thing is impossible if we represent words only as discrete objects.

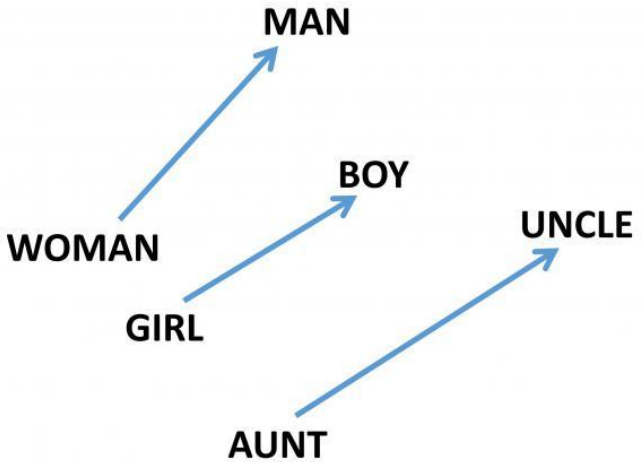
Word2vec - vector space arithmetic

These embeddings end up encoding some really useful information.

The difference between ‘woman’ - ‘man’, ‘girl’ - ‘boy’, and ‘aunt’ - ‘uncle’ are all vectors with roughly the same direction.

This tells us that the gender information is implicitly encoded in the word embeddings.

The same is true for many other comparisons that we can make between words.



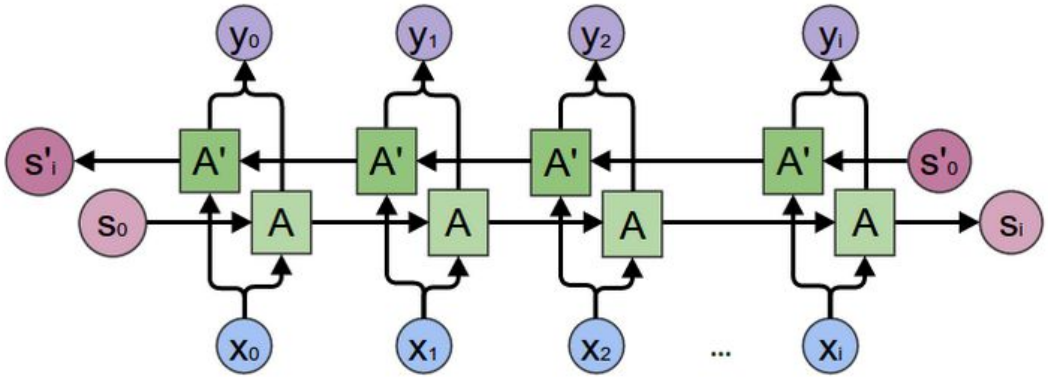

```
>>> model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
    [(u'queen', 0.7118192911148071)]
```

Hands on TensorFlow: Use TF helper functions for machine translation

RNN Enhancements

Bi-Directional RNNs

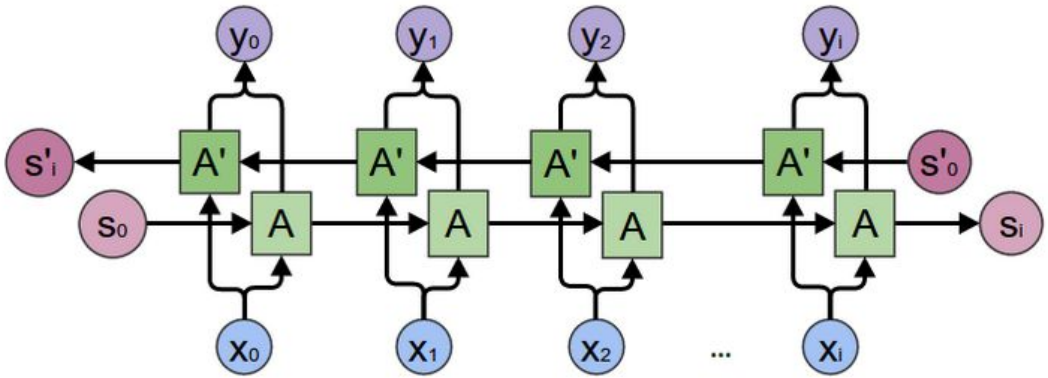
- The intuitive way to process a sequence is to go from start to finish
- With NLP applications especially, it often improves performance to also process the text from finish to start, reading it backwards.
- The representations created from going forwards and backwards can be concatenated



Olah et al., Neural Networks, Types, and Functional Programming, http://colah.github.io/posts/2015-09-NN-Types-FP/

Bi-Directional RNNs

- A sentence can often have a more sophisticated dependency structure than a simple start to finish reading
  - Increasing the flexibility of the RNN can help capture these dependencies



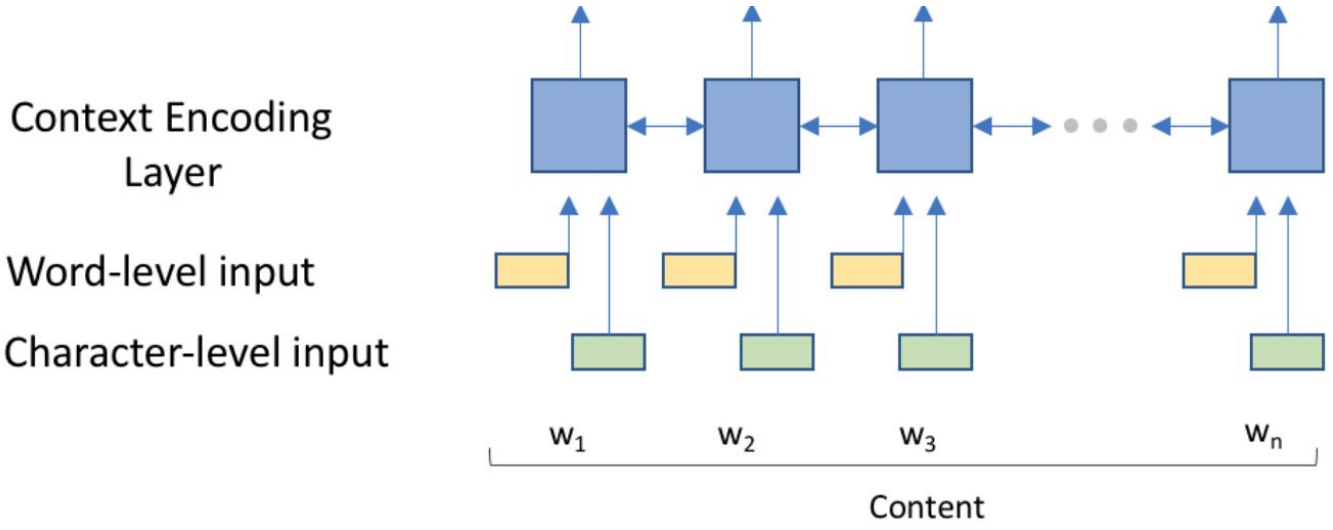
Olah et al., Neural Networks, Types, and Functional Programming, http://colah.github.io/posts/2015-09-NN-Types-FP/

Augmenting Embeddings

- Word embedding are an effective way of representing text to a ML model by capturing semantic meaning into dense vector representations
- However, word embeddings often don't make use of capitalization.
- Nor can they handle spelling errors

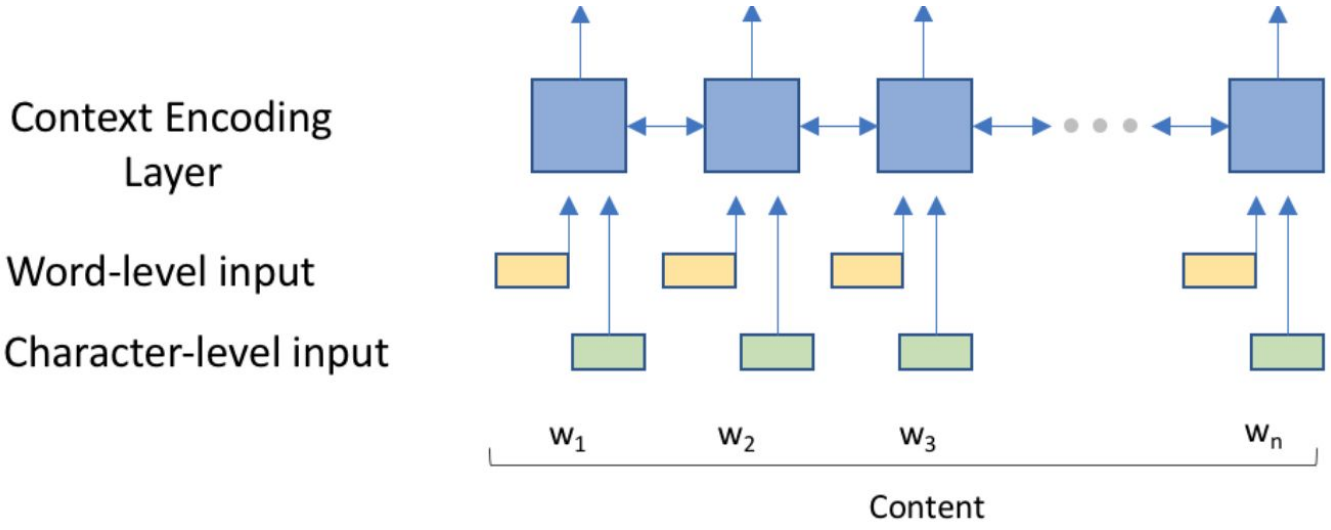
Augmenting Embeddings

- Adding a character level input to an RNN can improve performance by alleviating some of these issues



Augmenting Embeddings

- Character level input can be character level RNN or CNN that create a vector for each word
  - Resulting vector can be concatenated onto the word embeddings
  - Or passed through a sigmoid gate then added to the word embedding

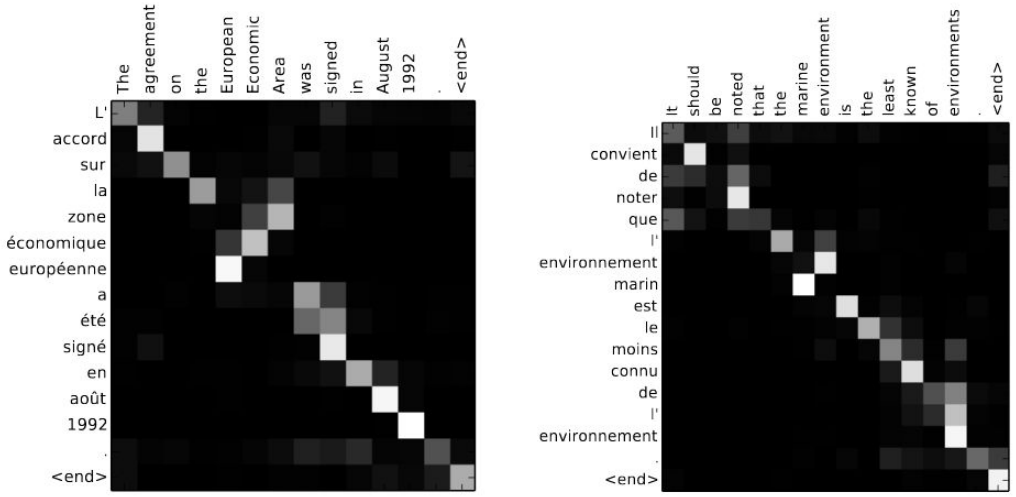


Attention

- Attention is the primary tool used to improve sequence-to-sequence encoder-decoder type networks
  - In out seq2seq model we initialized the decoder network with the last hidden state of the encoder network
  - Although we used LSTM networks that last state was still biased by the last letter of whatever word was encoded
  - We generated our pig-latin word one letter at a time
    - Starting with the first letter and by using a hidden state biased by the last letter of the encoded word

Attention

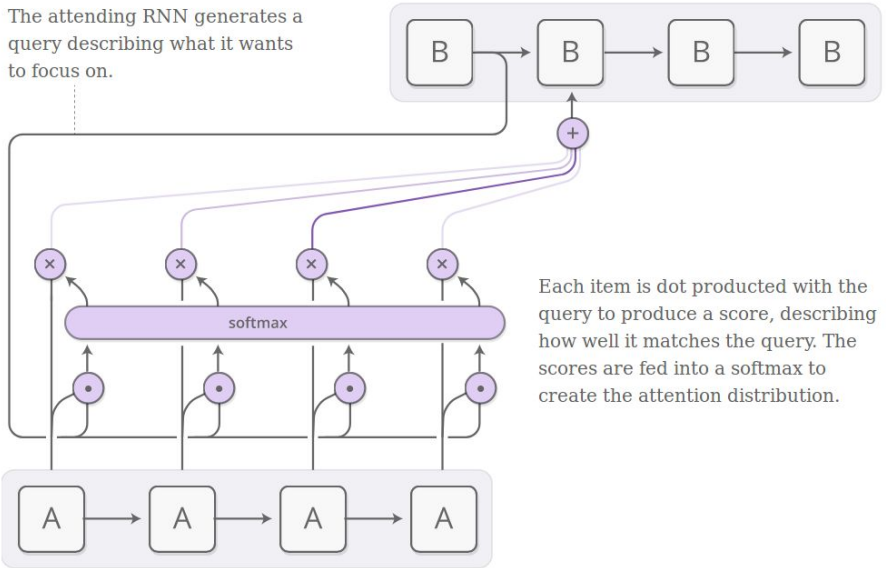
- Instead, what if we had a way to pay special attention to the part of the input sequence that is most relevant to whatever letter/word we are currently generating?



Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate, https://arxiv.org/pdf/1409.0473.pdf

Attention

- Instead of taking the last hidden state, we take a weighted sum of all of the hidden states based on the current word/letter that we want to generate



Olah et al., Attention and Augmented Recurrent Neural Networks, https://distill.pub/2016/augmented-rnns/

Hands on TensorFlow: Improve Machine Translation with Bi-LSTM

GRU/LSTM References

Chung et al., Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, https://arxiv.org/pdf/1412.3555.pdf

Christopher Olah, Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/