RNN extensions, Memory Addressing and Attention

Eilif Solberg

TEK5040/TEK9040

Outline

Composing RNNs
- Bidirectional RNNs
- Encoder-decoder framework

Recursive neural networks

RNN Memory extensions
- External memory
- Attending to previous states

Attention
- Content-based
- Self-attention
- Location-based

Composing RNNs

Bidirectional RNNs

Motivation:

- Want to include future context

Bidirectional RNNs

Motivation:

- Want to include future context
- Could solve with time-delay for predictions, though need to specify fixed context.

Bidirectional RNNs

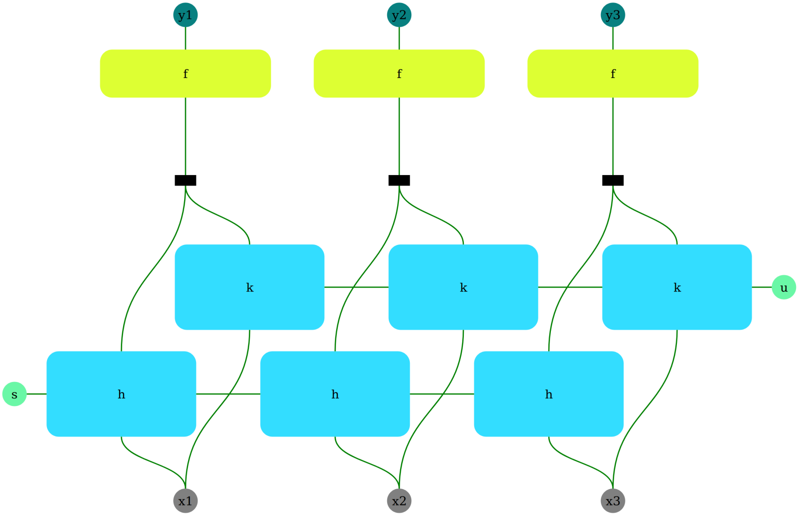
Motivation:

- Want to include future context
- Could solve with time-delay for predictions, though need to specify fixed context.

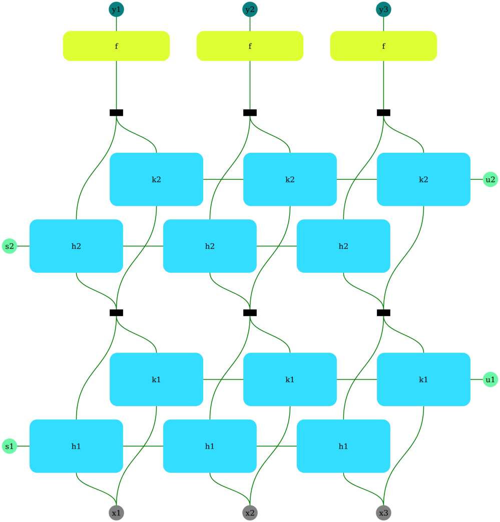
Assumes tight coupling between prediction at time t and input at time t.

- e.g. speech-to-text, text-to-speech

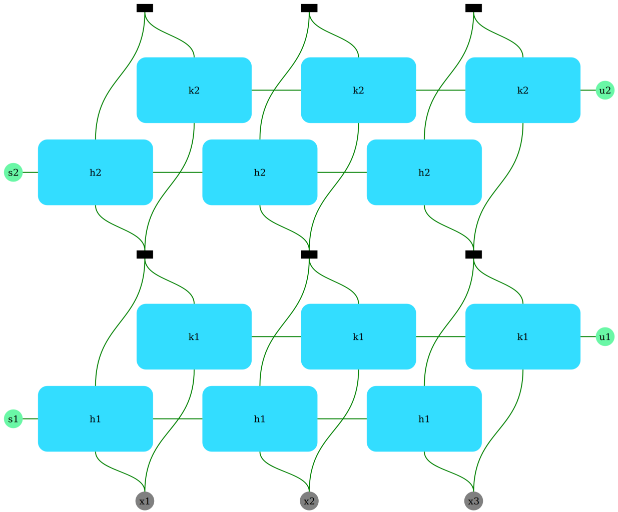
Bidirectional RNN - single layer



Bidirectional RNN - two layers



Bidirectional RNN - feature extraction



Bidirectional RNN in TensorFlow

Is there a `BidirectionalLSTMCell` in TensorFlow?

Bidirectional RNN in TensorFlow

Is there a `BidirectionalLSTMCell` in TensorFlow?

```
#  Create LSTM forward and backward layers with 10 state neurons each
forward_layer = layers.LSTM(10, return_sequences=True)
backward_layer = layers.LSTM(10, activation='relu', return_sequences=True, go_backwards=True)
bidirectional_layer = layers.Bidirectional(
     forward_layer,
     backward_layer=backward_layer,
     merge_mode='concat')
```

Encoder-decoder

For sequence-to-sequence problems with loose coupling between sequences

- prediction at time t not directly related to input at time t.

Encoder-decoder

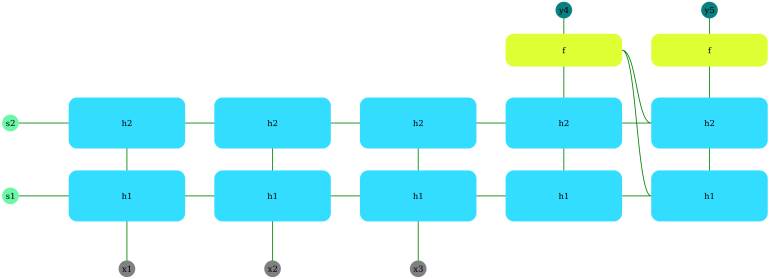
For sequence-to-sequence problems with loose coupling between sequences

- prediction at time t not directly related to input at time t.

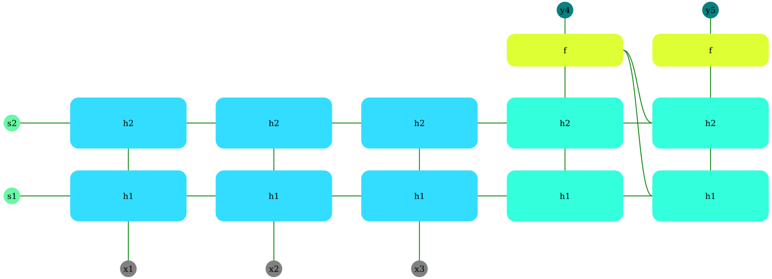
Example: sentence translation

1. Encode the "meaning" of sentence in source language into intermediate representation
2. Decode the "meaning" of the sentence into a representation in the target language

Encoder-decoder, shared RNN



Encoder-decoder, separate RNN



Encoder-decoder in TensorFlow I

Defined encoder, decoder and encode

```
# [batch_dim, time_dim, input_dim] == [1, 2, 3]
input_sequence = tf.constant([[[0.1, 0.3, 0.2], [-1.2, 0.4, -0.3]]])
batch_size = tf.shape(input_sequence)[0]
encoder_cell = layers.LSTMCell(10)
decoder_cell = layers.LSTMCell(10)

state = encoder_cell.get_initial_state(batch_size=batch_size, dtype=tf.float32)
# transpose time and batch axis before iterating
for x in tf.transpose(input_sequence, (1, 0, 2)):
  output, state = encoder_cell(x, state)

final_encoder_state = state
```

Encoder-decoder in TensorFlow I

Decoding

```
state = final_encoder_state
# assume 'init_symbol', 'f' and 'is_end_symbol' implemented
↪  somewhere.
ys = []
y = [init_symbol]
while True:
  output, state = decoder_cell(y, state)
  y = f(output)
  ys.append(y)
  # we assume batch size of 1 here
  if is_end_symbol(y[0]):
    break
```

Encoder-decoder in TensorFlow II

Often don't have to use `Cell` version in encoder.

```
input_sequence = tf.constant([[[0.1, 0.3, 0.2], [-1.2, 0.4,
↪  -0.3]]])
encoder = layers.LSTM(10, return_state=True)
decoder_cell = layers.LSTMCell(10)

out = encoder(input_sequence)
output, final_encoder_state = out[0], out[1:]

state = final_encoder_state
ys = []
y = [init_symbol]
while True:
  output, state = decoder_cell(y, state)
  y = f(output)
  ys.append(y)
  # we assume batch size of 1 here
  if is_end_symbol(y[0]):
    break
```

Encoder-decoder conclusion

Assume N source and N target languages.

- Want to be able to translate between any two of them

Encoder-decoder conclusion

Assume N source and N target languages.

- Want to be able to translate between any two of them
- Possible to share encoder and decoder?

Encoder-decoder conclusion

Assume N source and N target languages.

- Want to be able to translate between any two of them
- Possible to share encoder and decoder?

Newer models include attention

- The input to the decoder may then include the whole state sequence of the encoder. This allows for e.g using bidirectional RNN in *encoder*.

Google's Neural Machine Translation system (2016)

- Encoder-decoder framework, RNNs for both.
- Bottom encoder layer is bidirectional, decoder uses attention.
- Plenty of residual connections in both encoder and decoder.
- Design choices influenced by production needs.

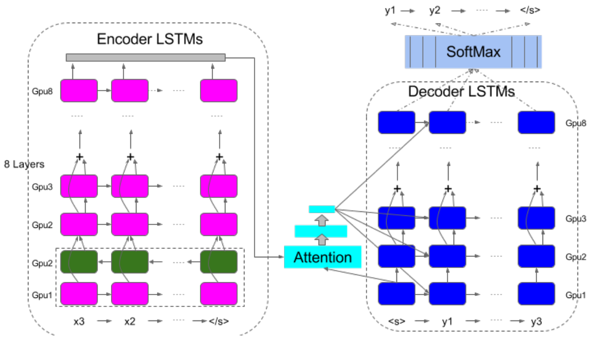


Figure: Illustration from Wu, Yonghui, et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016)

Recursive neural networks

RNN Memory extensions

Addressing: location vs content-based

Assume we have memory M with memory cells $M_1, \ldots, M_J$.

- E.g. $M_j \in \mathbb{R}^n$

Addressing: location vs content-based

Assume we have memory M with memory cells $M_1, \ldots, M_J$.

- E.g. $M_j \in \mathbb{R}^n$

How do we *address* memory?

Addressing: location vs content-based

Assume we have memory M with memory cells $M_1, \ldots, M_J$.

- E.g. $M_j \in \mathbb{R}^n$

How do we *address* memory?

Location

- Specify *where* to get information, e.g. index $j \in \{1, \ldots, J\}$
  - "Give me the content at memory cell 4"

Addressing: location vs content-based

Assume we have memory M with memory cells $M_1, \ldots, M_J$.

- E.g. $M_j \in \mathbb{R}^n$

How do we *address* memory?

Location

- Specify *where* to get information, e.g. index $j \in \{1, \ldots, J\}$
  - "Give me the content at memory cell 4"
- *Direct* addressing

Addressing: location vs content-based

Assume we have memory M with memory cells $M_1, \ldots, M_J$.

- E.g. $M_j \in \mathbb{R}^n$

How do we *address* memory?

Location

- Specify *where* to get information, e.g. index $j \in \{1, \ldots, J\}$
  - "Give me the content at memory cell 4"
- *Direct* addressing

Content

- Specify what kind of information through a *query* q
  - "When did the french revolution start?"

Addressing: location vs content-based

Assume we have memory M with memory cells $M_1, \ldots, M_J$.

- E.g. $M_j \in \mathbb{R}^n$

How do we *address* memory?

Location

- Specify *where* to get information, e.g. index $j \in \{1, \ldots, J\}$
  - "Give me the content at memory cell 4"
- *Direct* addressing

Content

- Specify what kind of information through a *query* q
  - "When did the french revolution start?"
- *Indirect* addressing

Content-based addressing

- Memory M with memory cells $M_1, \ldots, M_J$.

Content-based addressing

- Memory M with memory cells $M_1, \ldots, M_J$.
- query $q \in \mathbb{R}^d$

Content-based addressing

- Memory M with memory cells $M_1, \ldots, M_J$.
- query $q \in \mathbb{R}^d$
- key function K, e.g. $K : \mathbb{R}^n \rightarrow \mathbb{R}^d$

Content-based addressing

- Memory M with memory cells $M_1, \ldots, M_J$.
- query $q \in \mathbb{R}^d$
- key function K, e.g. $K\colon \mathbb{R}^n \to \mathbb{R}^d$
- matching function f, e.g. inner product function

$$\alpha_j = f(q, K(M_j))$$

Content-based addressing

- Memory M with memory cells $M_1, \ldots, M_J$.
- query $q \in \mathbb{R}^d$
- key function K, e.g. $K : \mathbb{R}^n \to \mathbb{R}^d$
- matching function f, e.g. inner product function

$$\alpha_j = f(q, K(M_j))$$

What is the returned result of our query?

Content-based addressing

- Memory M with memory cells $M_1, \ldots, M_J$.
- query $q \in \mathbb{R}^d$
- key function K, e.g. $K : \mathbb{R}^n \rightarrow \mathbb{R}^d$
- matching function f, e.g. inner product function

$$\alpha_j = f(q, K(M_j))$$

What is the returned result of our query?

$$p = \text{softmax}(\alpha)$$

$$v(q, M) = M_j \text{ with probability } p_j \qquad \text{hard addressing}$$

$$v(q, M) = \sum_{j=1}^{J} p_j M_j \qquad \text{soft addressing}$$

Content-based addressing

- Memory M with memory cells $M_1, \ldots, M_J$.
- query $q \in \mathbb{R}^d$
- key function K, e.g. $K : \mathbb{R}^n \rightarrow \mathbb{R}^d$
- matching function f, e.g. inner product function

$$\alpha_j = f(q, K(M_j))$$

What is the returned result of our query?

$$p = \text{softmax}(\alpha)$$

$$v(q, M) = M_j \text{ with probability } p_j \qquad \text{hard addressing}$$

$$v(q, M) = \sum_{j=1}^{J} p_j M_j \qquad \text{soft addressing}$$

Where does the query vector q come from?

RNN example with external memory - read operation

RNN example with external memory - read operation

Perform query based on current state

$$q^t = Q^{(r)}(s^t)$$

RNN example with external memory - read operation

Perform query based on current state

$$q^t = Q^{(r)}(s^t)$$

Extract key for each memory cell

$$k_j^t = K^{(r)}(M_j^{t-1})$$

RNN example with external memory - read operation

Perform query based on current state

$$q^t = Q^{(r)}(s^t)$$

Extract key for each memory cell

$$k_j^t = K^{(r)}(M_j^{t-1})$$

Calculate how well memory cell match query

$$\alpha_j^t = f(q^t, k_j^t)$$

RNN example with external memory - read operation

Perform query based on current state

$$q^t = Q^{(r)}(s^t)$$

Extract key for each memory cell

$$k_j^t = K^{(r)}(M_j^{t-1})$$

Calculate how well memory cell match query

$$\alpha_j^t = f(q^t, k_j^t)$$

Get resulting vector r^t by

$$p^t = \text{softmax}(\alpha^t)$$

$$r^t = v(q^t, M^{t-1}) = \sum_{j=1}^{J} p_j^t M_j^{t-1}$$

RNN example with external memory - write operation

RNN example with external memory - write operation

Need to decide *what* to write in addition to *where*

RNN example with external memory - write operation

Need to decide *what* to write in addition to *where*

- *Where* can be decided as with read operation
  - Separate functions $Q^{(w)}$ and $K^{(w)}$.

RNN example with external memory - write operation

Need to decide *what* to write in addition to *where*

- *Where* can be decided as with read operation
  - Separate functions $Q^{(w)}$ and $K^{(w)}$.
- *What*: e.g. function W

$$w^t = W(s^t)$$

RNN example with external memory - write operation

Need to decide *what* to write in addition to *where*

- *Where* can be decided as with read operation
  - Separate functions $Q^{(w)}$ and $K^{(w)}$.
- *What*: e.g. function W

$$w^t = W(s^t)$$

How to make update?

RNN example with external memory - write operation

Need to decide *what* to write in addition to *where*

- *Where* can be decided as with read operation
  - Separate functions $Q^{(w)}$ and $K^{(w)}$.
- *What*: e.g. function W

$$w^t = W(s^t)$$

How to make update?

$$M_j^t = (1 - p_j^w) M_j^{t-1} + p_j^w w^t \qquad \text{overwrite}$$

$$M_j^t = M_j^{t-1} + p_j^w w^t \qquad \text{residual update}$$

Exists corresponding *hard* update rules

RNN example with external memory - how to use it

Update function:

$$s^t = h(x^t, s^{t-1}, y^{t-1}, r^{t-1})$$

RNN example with external memory - how to use it

Update function:

$$s^t = h(x^t, s^{t-1}, y^{t-1}, r^{t-1})$$

Could also add directly to output function

$$y^t = f(s^t, r^t)$$

External memory - multiple read/write *heads*

- E.g. define N query, key function pairs $(Q_1^{(r)}, K_1^{(r)}), \ldots, (Q_N^{(r)}, K_N^{(r)})$
  - Concatenate all of the retrieved vectors, $r^t = (r_1^t, \ldots, r_n^t)$.

External memory - multiple read/write *heads*

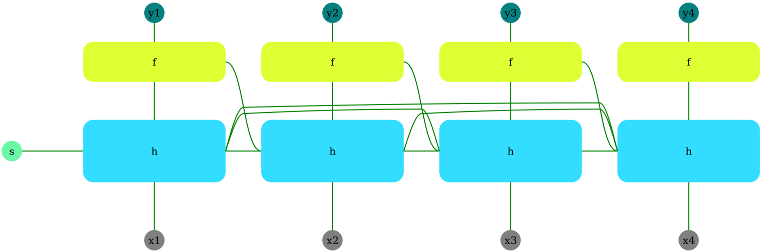
- E.g. define N query, key function pairs $(Q_1^{(r)}, K_1^{(r)}), \ldots, (Q_N^{(r)}, K_N^{(r)})$
  - Concatenate all of the retrieved vectors, $r^t = (r_1^t, \ldots, r_n^t)$.
- Write operations, need to resolve possible conflicts in updates

External memory - multiple read/write *heads*

- E.g. define N query, key function pairs $(Q_1^{(r)}, K_1^{(r)}), \ldots, (Q_N^{(r)}, K_N^{(r)})$
  - Concatenate all of the retrieved vectors, $r^t = (r_1^t, \ldots, r_n^t)$.
- Write operations, need to resolve possible conflicts in updates
- May use same matching function

Attending to previous states I

Attending to previous states I



Attending to previous states II

$$s^t = h(x^t, (s^1, \ldots, s^{t-1}), y^{t-1})$$

Attending to previous states II

$$s^t = h(x^t, (s^1, \dots, s^{t-1}), y^{t-1})$$

Do query with respect to "memory cells" $(s^1, \dots, s^{t-1})$.

$$\alpha_i^t = f(Q(s^{t-1}, x^t), K(s^i))$$
$$p^t = \text{softmax}(\alpha^t)$$
$$\tilde{s}^{t-1} = \sum_{i=1}^{t-1} p_i^t s^i$$

Attending to previous states II

$$s^t = h(x^t, (s^1, \ldots, s^{t-1}), y^{t-1})$$

Do query with respect to "memory cells" $(s^1, \ldots, s^{t-1})$.

$$\alpha_i^t = f(Q(s^{t-1}, x^t), K(s^i))$$

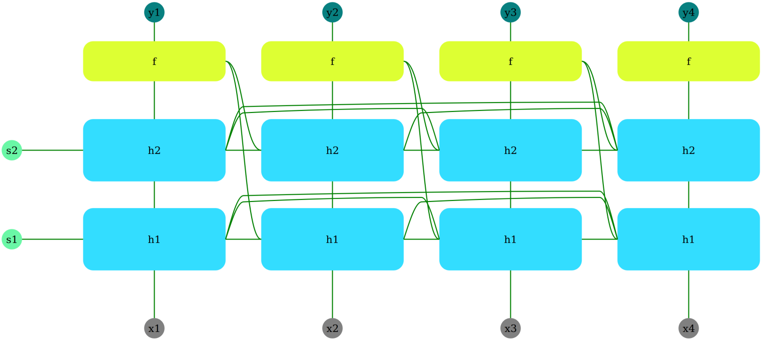
$$p^t = \mathrm{softmax}(\alpha^t)$$

$$\tilde{s}^{t-1} = \sum_{i=1}^{t-1} p_i^t s^i$$

Then proceed with "previous state" $\tilde{s}^{t-1}$

$$s^t = h(x^t, \tilde{s}^{t-1}, y^{t-1})$$

Attending to previous states III



Attention

Motivation for attention

- Don't get disctracted by irrelevant part of the input
- Use computational resources wisely

Image captioning with RNN and content-based attention

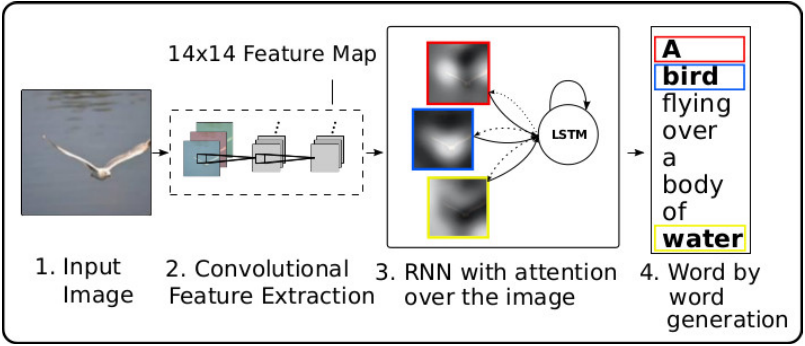


Figure: Illustration from "Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." International conference on machine learning. 2015."

Image captioning with RNN and content-based attention

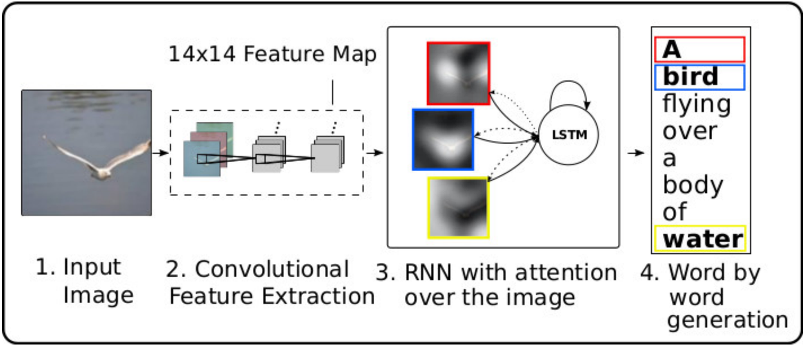


Figure: Illustration from "Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." International conference on machine learning. 2015."

- Content based addressing with 14×14 conv features as "memory"

Image captioning with RNN and content-based attention

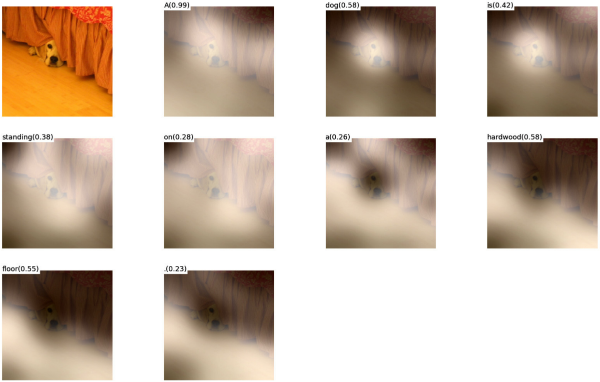


Figure: Illustration from "Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." International conference on machine learning. 2015."

Self-attention setup

- Assume we have data $x^1, x^2, \ldots, x^L \in \mathbb{R}^n$
- Define
    - query function $Q\colon \mathbb{R}^n \to \mathbb{R}^d$
    - key function $K : \mathbb{R}^n \to \mathbb{R}^m$
    - value function $V : \mathbb{R}^n \to \mathbb{R}^k$
    - matching function g that scores matches between queries and keys

Single-head attention

For *each* x^i

1. Perform a query against each x^j and get attention scores by
$$\alpha_{i,j} = g(Q(x^i), K(x^j))$$
2. Apply e.g. softmax to get probabilities
$$p_{i,j} = \frac{e^{\alpha_{i,j}}}{\sum_k e^{\alpha_{i,k}}}$$
3. With *soft attention* take a weighted average
$$z^i = \sum_{j=1}^{L} p_{i,j} x^j$$
4. Apply a function h such that to obtain $\tilde{x}^i$ as
$$\tilde{x}^i = h(z^i, x^i)$$

Single-head attention, with value function

For *each* x^i

1. Perform a query against each x^j and get attention scores by

$$\alpha_{i,j} = g(Q(x^i), K(x^j))$$

2. Apply e.g. softmax to get probabilities

$$p_{i,j} = \frac{e^{\alpha_{i,j}}}{\sum_k e^{\alpha_{i,k}}}$$

3. With *soft attention* take a weighted average

$$z^i = \sum_{j=1}^{L} p_{i,j} V(x^j)$$

4. Apply a function h such that to obtain $\tilde{x}^i$ as

$$\tilde{x}^i = h(z^i, x^i)$$

Multi-head attention, with value function

For m in $1, \dots, M$

1. Perform a query against each x^j and get attention scores by

$$\alpha^m_{i,j} = g(Q_m(x^i), K_m(x^j))$$

2. Apply e.g. softmax to get probabilities

$$p^m_{i,j} = \frac{e^{\alpha^m_{i,j}}}{\sum_k e^{\alpha^m_{i,k}}}$$

3. With *soft attention* take a weighted average

$$z^i_m = \sum_{j=1}^{L} p^m_{i,j} V_m(x^j)$$

Let $z^i = (z^i_1, \dots, z^i_M)$, then let $\tilde{x}^i = h(z^i, x^i)$

Self-attention remarks

- May repeat self-attention transformation (with separate query, key and value functions for each repetition) to create deeper transformation.
- Processing is independent of ordering of input elements.
  - If order of input data matters, we need to add positional encoding

$$x^i_{\textbf{pos}} = f(x^i, i)$$

    e.g.

$$x^i_{\textbf{pos}} = (x^i, p(i))$$

    for some positional encoding function p and $x^1_{\text{pos}}, \ldots, x^L_{\text{pos}}$ are our positionally encoded input data.
- Computations scale quadratically with number of inputs

Image classification with RNN and location-based attention

Pseudoalgorithm:

[1] A *glimpse* is here defined as a crop of the image

Image classification with RNN and location-based attention

Pseudoalgorithm:

- Start with center/random glimpse[1] with center l^0

[1] A *glimpse* is here defined as a crop of the image

Image classification with RNN and location-based attention

Pseudoalgorithm:

- Start with center/random glimpse[1] with center l^0
- For $t = 1, \ldots, \tau$
  1. Extract glimpse with center at l^{t-1}.
  2. Extract features for location, e.g. with convnet
  3. Update state of RNN
  4. - if $t < \tau$: predict next glimpse center l^t
     - else: Make prediction/classification based on l^τ

[1] A *glimpse* is here defined as a crop of the image

Image classification with RNN and location-based attention

Pseudoalgorithm:

- Start with center/random glimpse[1] with center l^0
- For $t = 1, \ldots, \tau$
  1. Extract glimpse with center at l^{t-1}.
  2. Extract features for location, e.g. with convnet
  3. Update state of RNN
  4. - if $t < \tau$: predict next glimpse center l^t
     - else: Make prediction/classification based on l^τ
- How to encode l^t?

[1] A *glimpse* is here defined as a crop of the image

Image classification with RNN and location-based attention

Pseudoalgorithm:

- Start with center/random glimpse[1] with center l^0
- For $t = 1, \ldots, \tau$
  1. Extract glimpse with center at l^{t-1}.
  2. Extract features for location, e.g. with convnet
  3. Update state of RNN
  4. - if $t < \tau$: predict next glimpse center l^t
     - else: Make prediction/classification based on l^τ
- How to encode l^t?
- Glimpse policy trained with reinforcement learning (policy gradient)!

[1] A *glimpse* is here defined as a crop of the image

Image classification with RNN and location-based attention

Pseudoalgorithm:

- Start with center/random glimpse[1] with center l^0
- For $t = 1, \ldots, \tau$
  1. Extract glimpse with center at l^{t-1}.
  2. Extract features for location, e.g. with convnet
  3. Update state of RNN
  4. - if $t < \tau$: predict next glimpse center l^t
     - else: Make prediction/classification based on l^τ
- How to encode l^t?
- Glimpse policy trained with reinforcement learning (policy gradient)!

Usually extract some lower resolution crops as well.

[1] A *glimpse* is here defined as a crop of the image