Sequence Modelling

RNN, biRNN, BPTT, LSTM, RecNN

Fifth Machine Learning in High Energy Physics Summer School,
MLHEP 2019

Ekaterina Artemova

National Research University Higher School of Economics

Sequence modelling

Recurrent neural network

- Definition
- Training

Gated architectures

RNN generators

The Transformer

Implementation in PyTorch

Take-home message

Bonus: Recursive NN

Sequence modelling

Sequential data

1. Time series
   - Financial data analysis: stock market, commodities, Forex
   - Healthcare: pulse rate, sugar level (from medical equipment and wearables)
2. Text and speech: speech understanding, text generation
3. Spatiotemporal data
   - Self-driving and object tracking
   - Plate tectonic activity
4. Physics: jet identification
5. etc.

Sequence modelling I

Sequence classification

1. $\boldsymbol{x} = x_1, x_2, \ldots, x_n, x_i \in V$ - objects
2. $y \in \{1, \ldots, L\}$ - labels
3. $\{(\boldsymbol{x}^{(1)}, y_1), (\boldsymbol{x}^{(2)}, y_2), \ldots, (\boldsymbol{x}^{(m)}, y_m)\}$ – training data

Classification problem: $\gamma : \boldsymbol{x} \to y$

1. Activity recognition: x – pulse rate, y – activity (walking, running, peace)
2. Opinion mining: x – sentence, y – sentiment (positive, negative)
3. Trading: x – stock market, y – action (sell, buy, do nothing)

Sequence modelling II

Sequence labelling

1. $\boldsymbol{x} = x_1, x_2, \ldots, x_n, x_i \in V$ - objects
2. $\boldsymbol{y} = y_1, y_2, \ldots, y_n, y_i \in \{1, \ldots, L\}$ - labels
3. $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$ – training data
4. exponential number of possible solutions : if $\texttt{length}(x) = n$, there are L^n possible solutions

Classification problem: $\gamma : \boldsymbol{x} \to \boldsymbol{y}$

1. Part of speech tagging: x – word, y – part of speech (verb, noun, etc.)
2. Genome annotation: x – DNA, y – genes
3. HEP tracking: x - a set of hits with backgrounds, y – hit classification

Sequence modelling III

Sequence transduction / transformation

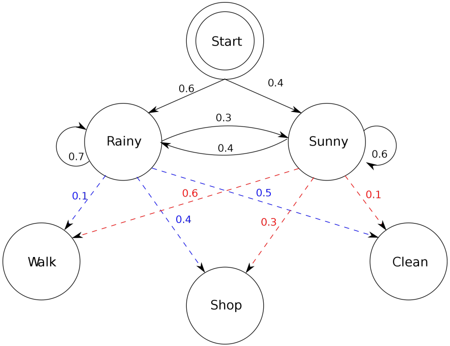
1. $\boldsymbol{x} = x_1, x_2, \ldots, x_n, x_i \in V_{source}$ - objects
2. $\boldsymbol{y} = y_1, y_2, \ldots, y_n, y_i \in V_{target}$ - objects
3. $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$ – training data
4. $\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}$ are of different length

Transduction problem: $\boldsymbol{x}_{source} \rightarrow \boldsymbol{y}_{target}$

1. Machine translation: x – sentence in German, y – sentence in English
2. Speech recognition: x – spoken language, y – text
3. Chat bots: x – question, y – answer

Traditional ML approaches to sequence modeling

- Hidden Markov Models (HMM)
- Conditional Random Fields (CRF)
- Local classifier: for each x define features, based on x_{-1}, x_{+1}, etc, and perform classification n times

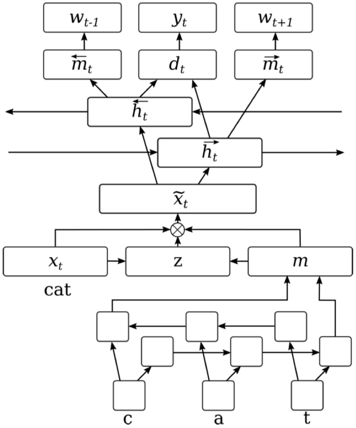


Problems:

1. Markov assumption: fixed length history
2. Computation complexity

DL approaches to sequence modeling

- Recurrent neural network and its modifications: LSTM, GRU, Highway
- Transformer
- 2D Convolutional Neural Network
- Pointer network



Problems:

1. Training time
2. Amount of training data

Recurrent neural network

Definition

Recurrent neural network

- Input: sequence of vectors
- $x_{1:n} = x_1, x_2, \ldots, x_n, x_i \in \mathbb{R}^{d_{in}}$
- Output: a single vector
  $y_n = RNN(x_{1:n}), y_n \in \mathbb{R}^{d_{out}}$
- For each prefix $x_{i:j}$ define an output vector y_i:
  $y_i = RNN(x_{1:i})$
- RNN^* is a function returning this sequence for input sequence $x_{1:n}$:
  $y_{1:n} = RNN^*(x_{1:n}), y_i \in \mathbb{R}^{d_{out}}$

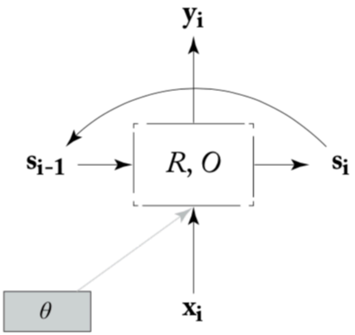


Figure: Goldberg, Yoav. Neural network methods for natural language processing

Sequence modelling with RNN

1. Sequence classification
   Put a dense layer on top of RNN to predict the desired class of the sequence after the whole sequence is processed

$$p(l_j|\boldsymbol{x}_{1:n}) = \texttt{softmax}(RNN(\boldsymbol{x}_{1:n}) \times W + b)_{[j]}$$

2. Sequence labelling
   Produce an output y_i for each input RNN reads in. Put a dense layer on top of each output to predict the desired class of the input

$$p(l_j|\boldsymbol{x}_j) = \texttt{softmax}(RNN(\boldsymbol{x}_{1:j}) \times W + b)_{[j]}$$

More details on RNN

- $RNN^*(x_{1:n}, s_0) = y_{1:n}$
- $y_i = O(s_i)$ – simple activation function
- $s_i = R(s_{i-1,x_i})$, where R is a recursive function, s_i is a state vector
- s_0 is initialized randomly or is a zero vector
- $x_i \in \mathbb{R}^{d_{in}}$, $y_i \in \mathbb{R}^{d_{out}}$, $s_i \in \mathbb{R}^{f(d_{out})}$
- θ – shared weights

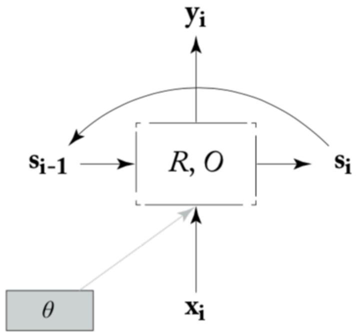


Figure: Goldberg, Yoav. Neural network methods for natural language processing

More details on RNN

- $s_i = R(x_i, s_{i-1}) = g(s_{i-1}W^s + x_iW^x + b)$
- $y_i = O(s_i) = s_i$
- $y_i, s_i, b \in \mathbb{R}^{d_{out}}, x_i \in \mathbb{R}^{d_{in}}$
- $W^x \in \mathbb{R}^{d_{in} \times d_{out}}, W^s \in \mathbb{R}^{d_{out} \times d_{out}}$

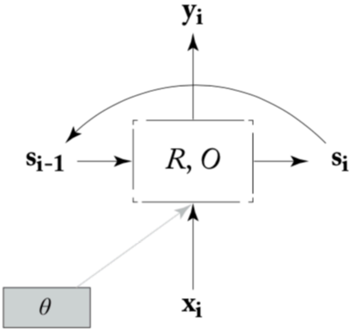
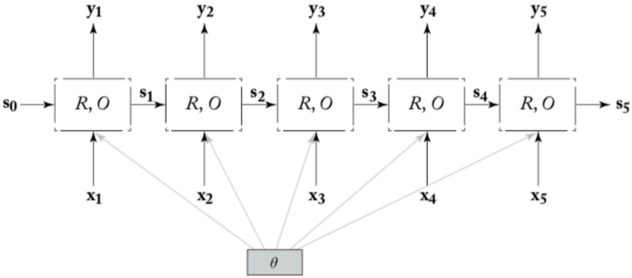


Figure: Goldberg, Yoav. Neural network methods for natural language processing

RNN unrolled

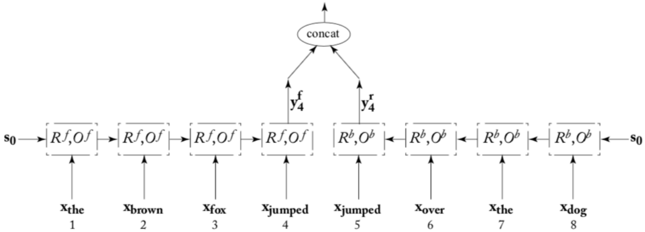


$$s_4 = R(s_3, x_4) = R(R(s_2, x_3), x_4) = R(R(R(s_1, x_2), x_3), x_4) =$$
$$= R(R(R(R(s_0, x1), x_2), x_3), x_4)$$

Figure: Goldberg, Yoav. Neural network methods for natural language processing

Bidirectional RNN (Bi-RNN)

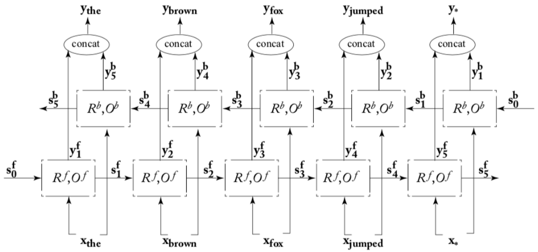
The input sequence can be read from left to right and from right to left. Which direction is better?



$$biRNN(x_{1:n}, i) = y_i = [RNN^f(x_{1:i}); RNN^r(x_{n:i})]$$

Figure: Goldberg, Yoav. Neural network methods for natural language processing

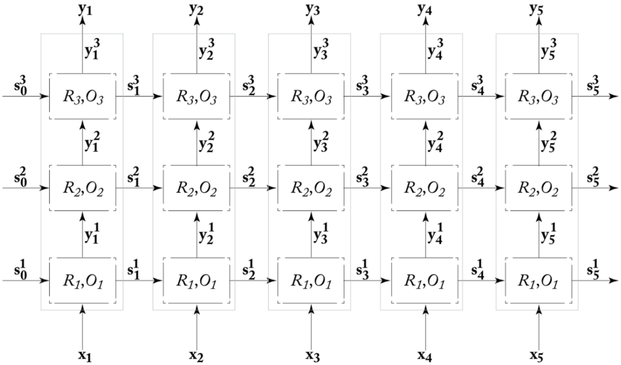
Bi-RNN



$$biRNN^*(x_{1:n}, i) = y_{1:n} = biRNN(x_{1:n}, 1) \dots biRNN(x_{1:n}, n)$$

Figure: Goldberg, Yoav. Neural network methods for natural language processing

Multilayer RNN



Connections between different layers are possible too: $y_1^2 = \texttt{concat}(x_1, y_1^1)$

Figure: Goldberg, Yoav. Neural network methods for natural language processing

Training

Sequence classification

- $\hat{y_n} = O(s_n)$
- `prediction` $= MLP(\hat{y_n})$
- Loss: $L(\hat{y_n}, y_n)$
- L can take any form: cross entropy, hinge, margin, etc.

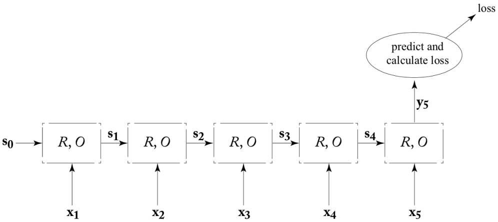


Figure: Goldberg, Yoav. Neural network methods for natural language processing

Sequence labelling

- Output $\hat{t_i}$ for each input $x_{1,i}$
- Local loss: $L_{local}(\hat{t_i}, t_i)$
- Global loss:
  $L(\hat{t_n}, t_n) = \sum_i L_{local}(\hat{t_i}, t_i)$
- L can take any form: cross entropy, hinge, margin, etc.

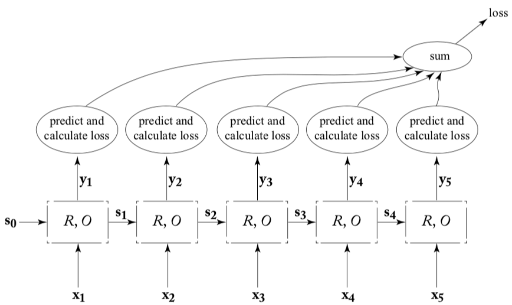


Figure: Goldberg, Yoav. Neural network methods for natural language processing

Backpropogation through time

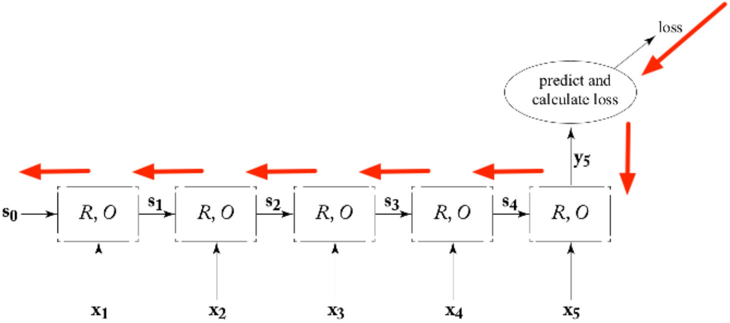


Figure: Goldberg, Yoav. Neural network methods for natural language processing

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

Chain rule: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial p(\hat{y_5})} \frac{\partial p(\hat{y_5})}{\partial s_4} (\frac{\partial s_4}{\partial w} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial w} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_w} + \ldots)$

Vanishing gradient problem

Chain rule: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial p(\hat{y_5})} \frac{\partial p(\hat{y_5})}{\partial s_4} (\frac{\partial s_4}{\partial w} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial w} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_w} + \ldots)$
g – sigmoid

1. Many sigmoids near 0 and 1
   - Gradients $\to$ 0
   - Not training for long term dependencies
2. Many sigmoids > 1
   - Gradients $\to + \inf$
   - Not training again

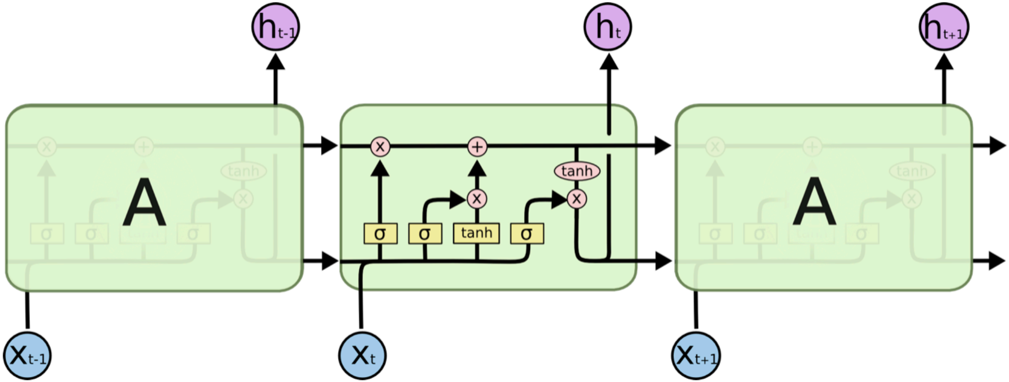
Solution: gated architectures (LSTM and GRU)

Gated architectures

Controlled memory access

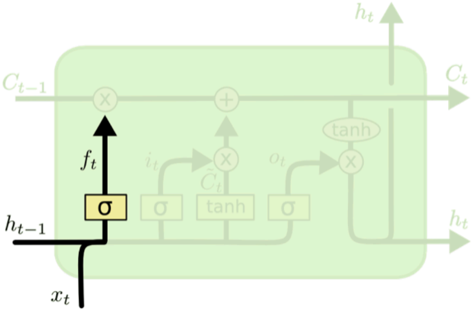
- Entire memory vector is changed: $s_{i+1} = R(x_i, s_i)$
- Controlled memory access: $s_{i+1} = g \odot R(x_i, s_i) + (1 - g)s_i$
  $g \in [0, 1]^d, s, x \in \mathbb{R}^d$
- Differential gates: $\sigma(g), g' \in \mathbb{R}^d$
- This controllable gating mechanism is the basis of the LSTM and the GRU architectures

Long short term memory



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

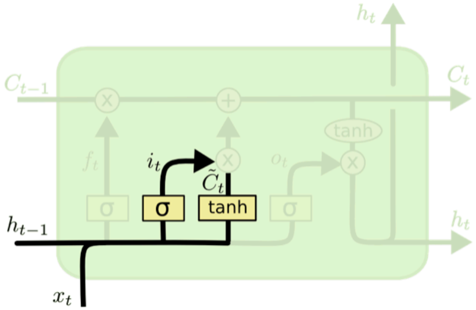
Long short term memory



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short term memory

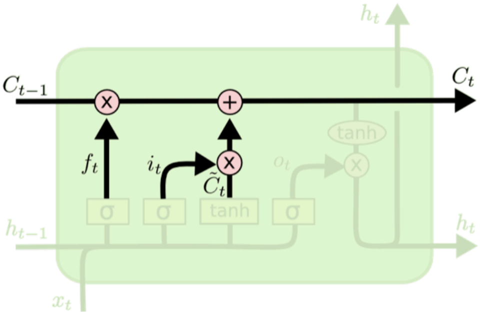


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

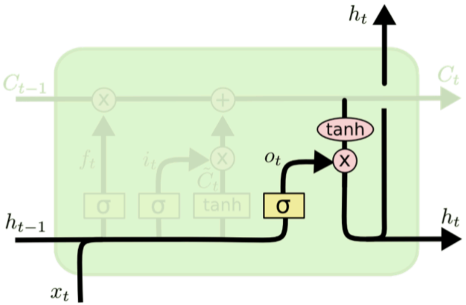
Long short term memory



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

```
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
```

Long short term memory

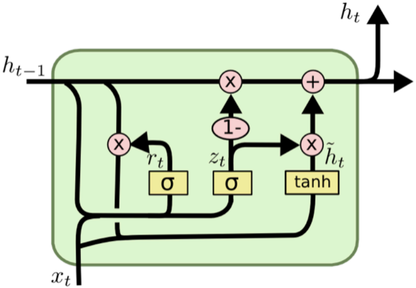


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated recurrent unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

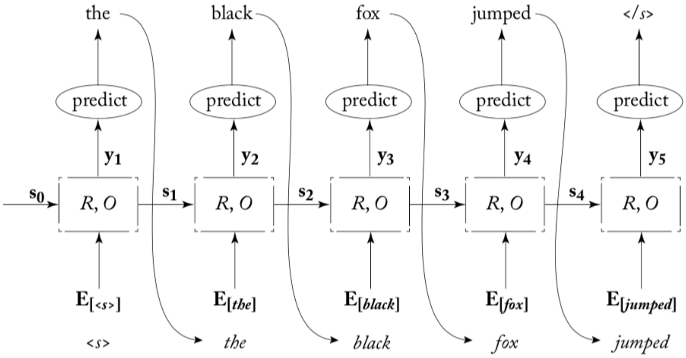
RNN generators

Sequence generation

Teacher forcing: $x :=< s > x, y := x < /s?$

$x :< s > x_1 x_2 \dots x_n$

$y : x_1 x_2 \dots x_n < /s >$



Sequence generation

- Examples of generated texts: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`
- Examples of generated MIDI music: `https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-6878`

Pros and cons of RNNs

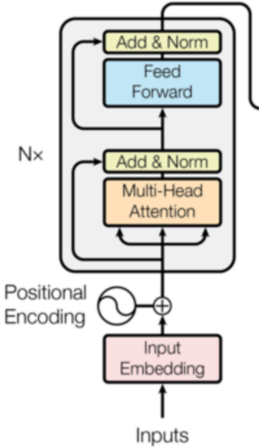
1. Advantages:
   - RNNs are popular and successful for variable-length sequences
   - The gating models such as LSTM are suited for long-range error propagation
2. Problems:
   - The sequentiality prohibits parallelization within instances
   - Long-range dependencies still tricky, despite gating

The Transformer

The Transformer

An alternative architecture to RNN which allows of parallel and faster training

- Several layers of identical modules
- Each module consists of Multi-Head Attention and Feed Forward layers
- Input: embeddings. To get embeddings for numerical input, apply any dense layer
- Positional embeddings to make use of the order of the sequence



Scaled Dot-Product Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors:

$$Attention(Q, K, V) = \texttt{softmax}(\frac{QK^T}{\sqrt{d_k}})V,$$

where the input consists of queries Q and keys K of dimension d_k and values V of dimension d_v

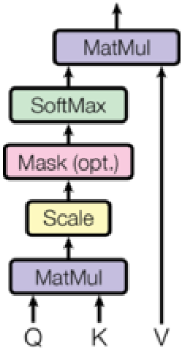
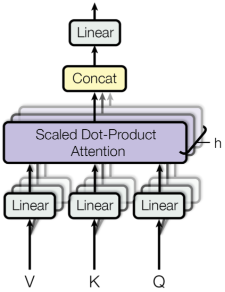


Figure: Vaswani, Ashish, et al. “Attention is all you need.” Advances in neural information processing systems. 2017.

Multi-head Attention

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions



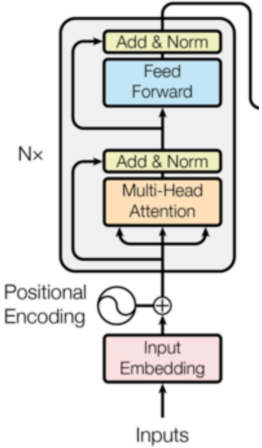
$$MultiHead(Q, K, V) = \texttt{concat}(head_1, ..., head_h)W^O,$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$
and W are projection matrices.

Figure: Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

The Transformer

Bringing it all together:



- LayerNorm: $\frac{x-\mu}{\sigma}$
- Residual connection: LayerNorm(x+Sublayer(x))
- Position-wise Feed-Forward Networks: $FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$

Positional Encoding

We need to inject some information about the relative or absolute position of x_{pos} in the sequence:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Positional encoding: $x = x + PE(x)$

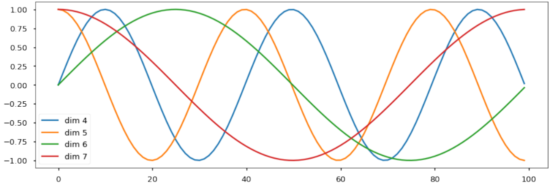


Figure: Vaswani, Ashish, et al. “Attention is all you need.” Advances in neural information processing systems. 2017.

Implementation in PyTorch

Recurrent neural networks

```
torch.nn.RNN(input_size, hidden_size,
    num_layers, nonlinearity, bias,
    batch_first, dropout, bidirectional)

torch.nn.LSTM(input_size, hidden_size,
    num_layers, bias,
    batch_first, dropout, bidirectional)
```

Transformers

```
from fairseq.models import transformer
```

Take-home message

Take-home message

- There is a lot of sequential data around us
- Before DL: HMM, MEMM
- Mid 2010 DL: RNN, LSTM, etc
- Late 2010 DL: the Transformer
- 2020: stack more transformer blocks (Trasformer XL)

Bonus: Recursive NN

Modeling trees with Recursive NN

- Input: $x_1, x_2, \dots, x_n$
- A binary tree T can be represented as a unique set of triplets (i, k, j), s.t. $i < k < j$, $x_{i:j}$ is parent of $x_{i:k}, i_{k+1,j}$
- RecNN takes as an input a binary tree and returns as output a corresponding set of inside state vectors $\boldsymbol{s}^{\boldsymbol{A}}_{\boldsymbol{i:j}} \in \mathbb{R}^d$
- Each state vector $\boldsymbol{s}^{\boldsymbol{A}}_{\boldsymbol{i:j}}$ represents the corresponding tree node $q^A_{i:j}$ and encodes the entire structure rooted at that node

RecNN

- Input: $x_1, x_2, \ldots, x_n$ and a binary tree T
- $RecNN(x_1, x_2, \ldots, x_n, T) = \{\boldsymbol{s^A_{i:j}} \in \mathbb{R}^d | q^A_{i:j} \in T\}$
- $\boldsymbol{s^A_{i:i}} = v(x_i)$
- $\boldsymbol{s^A_{i:j}} = R(A, B, C, \boldsymbol{s^B_{i:k}}, \boldsymbol{s^C_{k+1:j}})\boldsymbol{,}\ q^B_{i:k} \in T, q^C_{k+1:j} \in T$
- $R(A, B, C, \boldsymbol{s^B_{i:k}}, \boldsymbol{s^C_{k+1:j}}) = g([\boldsymbol{s^B_{i:k}}, \boldsymbol{s^C_{k+1:j}}]W)$

RecNN

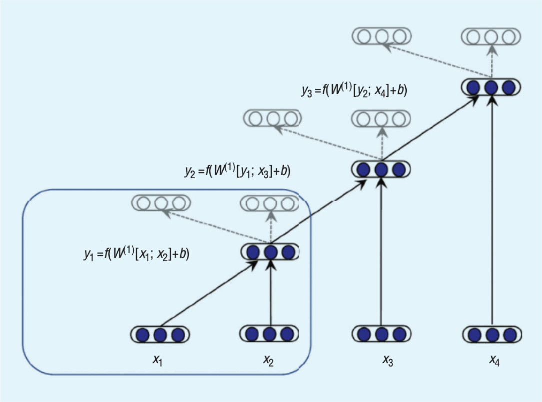


Figure: Zhang, Jiajun & Zong, Chengqing. (2015). Deep Neural Networks in Machine Translation: An Overview