Introduction to Program Synthesis (SS 25)

Chapter 4 - Advanced Methodologies

Dr. rer. nat. Roman Kalkreuth

Chair for AI Methodology (**AIM**), Department of Computer Science,
RWTH Aachen University, Germany

Advanced Methodologies

Neural Program Synthesis: Recurrent Neural Networks

- Feed-forward Neural Networks → uni directional dataflow
  - ⇝ Each input is treated independently

  Recurrent Neural Networks (RNN) → processing of sequential data via feedback loops
  - ⇝ Sequence modelling of time series, speech, text, code, music, ...
  - ⇝ Loop-like architecture
  - ⇝ Memoisation of past input states

Advanced Methodologies

Neural Program Synthesis: Recurrent Neural Networks

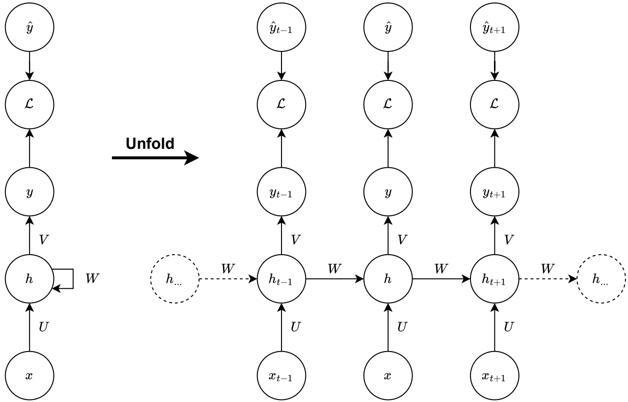

Figure: Recurrent Neural Network (RNN)

Advanced Methodologies
Neural Program Synthesis: Recurrent Neural Networks

- A RNN can be considered a function $f(x_t, h_t, \theta) \mapsto (y_t, h_{t+1})$
  - $\leadsto$ x_t : input vector
  - $\leadsto$ h_t : hidden vector
  - $\leadsto$ y_t : output vector
  - $\leadsto$ θ : hyperparameters
- The input vector x_t is mapped into an output y_t
  - $\leadsto$ hidden vector h_t serves as an *memory*
- *Transformation* of an input to an output at each step t
  - $\leadsto$ U, V and W $\rightarrow$ weight matrices
  - $\leadsto$ b and c $\rightarrow$ bias vectors
  - $\leadsto$ step-wise update $\rightarrow$ back-propagation through time (BPTT)

- Update equations are applied from $t = 1$ to $t = \tau$
  - $\leadsto$ $a^t = b + Wh^{t-1} + Ux^t$
  - $\leadsto$ $h^t = tanh(a^t)$
  - $\leadsto$ $y^t = c + Vh^t$

Advanced Methodologies
Neural Program Synthesis: Long-short term memory (LSTM)

- RNNs are prone to the vanishing gradient problem
  - $\leadsto$ Long-term gradients that are back-propagated through time can *vanish*
- **Long-short term memory (LSTM)** $\rightarrow$ learning when to remember and when to forget information
  - $\leadsto$ Ability to decide when inputs should be remembered or ignored in the hidden state

Advanced Methodologies
Neural Program Synthesis: Long-short term memory (LSTM)

- LSTM Architecture $\rightarrow$ gated memory cell
  - $\rightsquigarrow$ **Input gate** I_t $\rightarrow$ Decides when information is added to the cell
  - $\rightsquigarrow$ **Forget gate** F_t $\rightarrow$ Resets the content of the cell
  - $\rightsquigarrow$ **Output gate** O_t $\rightarrow$ Determines the output of the cell
- Hidden state $\rightarrow$ replaced with a memory cell

Advanced Methodologies

Neural Program Synthesis: Long-short term memory (LSTM)

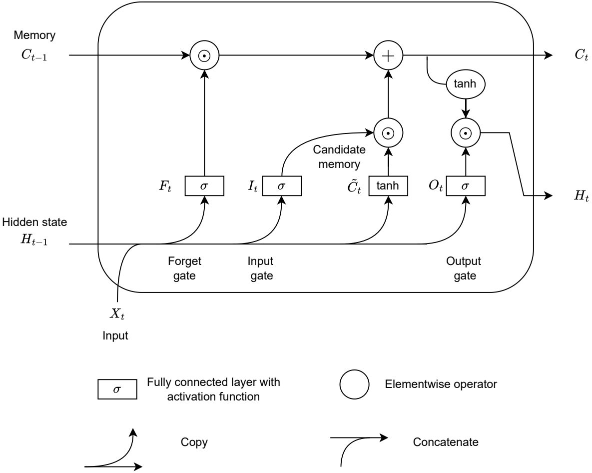

Figure: Long-short term memory (LSTM)

Advanced Methodologies

Neural Program Synthesis: Long-short term memory (LSTM)

- With h hidden units, d inputs and batch size is b
  - $\rightsquigarrow$ $X_t \in \mathbb{R}^{n \times d}$
  - $\rightsquigarrow$ $H_{t-1} \in \mathbb{R}^{n \times h}$
  - $\rightsquigarrow$ $I_t \in \mathbb{R}^{n \times h}$
  - $\rightsquigarrow$ $F_t \in \mathbb{R}^{n \times h}$
  - $\rightsquigarrow$ $T_t \in \mathbb{R}^{n \times h}$
- The gates are calculated as follows
  - $\rightsquigarrow$ $I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$
  - $\rightsquigarrow$ $F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$
  - $\rightsquigarrow$ $T_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$
- Weights and biases are defined as follows
  - $\rightsquigarrow$ $W_{xi}, W_{xf}, W_{xo} \in \mathbb{R}^{d \times h}$
  - $\rightsquigarrow$ $W_{hi}, W_{hf}, W_{ho} \in \mathbb{R}^{h \times h}$
  - $\rightsquigarrow$ $b_i, b_f, b_o \in \mathbb{R}^{1 \times h}$

Advanced Methodologies

Neural Program Synthesis: Long-short term memory (LSTM)

- Candidate memory cell $\tilde{C}_t \in \mathbb{R}^{n \times h}$
  - $\rightsquigarrow$ $\tanh(X_t W_{xc} + H_{t-1} Whc + b_c)$
- Memory cell C_t
  - $\rightsquigarrow$ $C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$
- Hiden state H_t
  - $\rightsquigarrow$ $H_t = O_t \odot \tanh(C_t)$
  - $\rightsquigarrow$ Values of H_t are then in the interval $(-1, 1)$

Advanced Methodologies

Neural Program Synthesis: Autoencoders

- Type of neural network that are trained to copy given input to ihe corresponding output
  - ⇝ General idea → Map an input x to an output (namely reconstruction) via a latent representation or code h
- A hidden layer h represents the code
  - ⇝ Latent space reprensetation of the input
- Mainly consists of two parts:
  - ⇝ Encoder function $h = f(x)$
  - ⇝ Decoder that does reconstruction $r = g(h)$

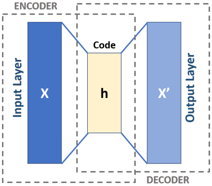

Figure: Autoencoder (Source: Wikimedia (Creator: Michela Massi))

Advanced Methodologies
Neural Program Synthesis: Transformer

- Successor of RNN's and LSTM for automated natural language processing (NLP)
  - $\rightsquigarrow$ Extension via attention mechanisms
  - $\rightsquigarrow$ Transformers neglect recurrent structures $\rightarrow$ focus on attention mechanism
  - $\rightsquigarrow$ Imitation of human cognitive attention
- Transformer calculate a weighting for each word in the context of the embedding
  - $\rightsquigarrow$ Embedding is based on encoder-decoder architecture
  - $\rightsquigarrow$ Embedding layer weights are adjusted during training
  - $\rightsquigarrow$ Transformation $\rightarrow$ **Word2Vec**

Advanced Methodologies

Neural Program Synthesis: Transformer

- **Tokenizer** → preparation of the inputs for a model
- Word2Vec → vector representations of words
  - Words in similar contexts → mapping to vectors
  - Distance is measured with **cosine similarity**
  - *Text-to-token* → Tokenizer
- **Positional encodin**g → sequential order of the words is respected
- **Attention mechanism** → Weighting tokens based on their importance
  - Self attention → Capturing of long-range dependencies without sequential processing
  - Multi-headed attention → Enabling focus on various aspects of the input data simultaneously

Advanced Methodologies

Neural Program Synthesis: Transformer

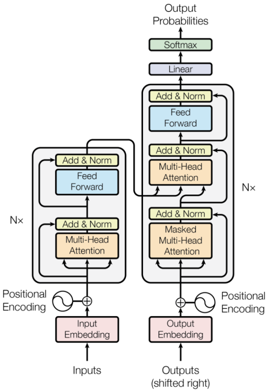

Figure: Transformer (Source: Vaswani et al. (2017))