CS689: Computational Linguistics for Indian Languages
Sequence Based Models

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

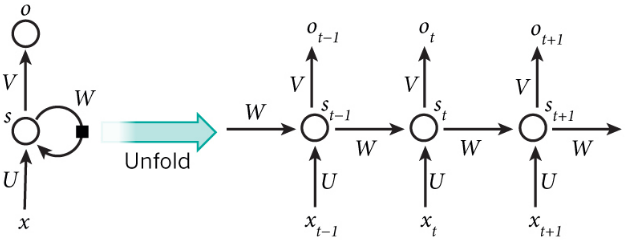
Computer Science and Engineering,
Indian Institute of Technology, Kanpur
http://web.cse.iitk.ac.in/~cs689/

2nd semester, 2023-24

Tue 10:30–11:45, Thu 12:00–13:15 at RM101/KD102

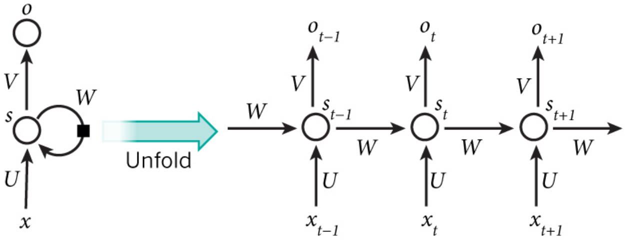
Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) model recurring patterns
  - Same task is repeated for every element of a sequence
- Hidden nodes are *not* independent of each other



- Output depends on previous steps, i.e., it uses "memory"
- "Unrolling" or "unfolding" produces the layers
- If a 3-length *context* is needed, RNN is unfolded to 3 layers
- Same parameters U, V, W are shared across the layers
  - General deep networks are not constrained by this

Components of an RNN



- $\vec{x}$ at each step is the *one-hot* vector (i.e., only 1 element is on)
- $\vec{s_t}$ is "memory" as it captures everything previous

$$\vec{s_t} = f(U \cdot \vec{x_t} + W \cdot \vec{s_{t-1}} + \vec{b_s})$$

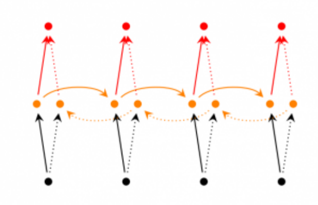
  - f is a non-linear function such as sigmoid or hyperbolic tangent
- $\vec{o_t}$ is *output* at step t

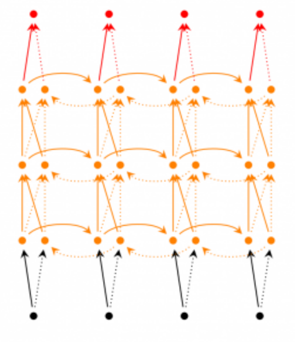
$$\vec{o_t} = g(V \cdot \vec{s_t} + \vec{b_o})$$

  - Generally, g is the *softmax* function to produce distributions

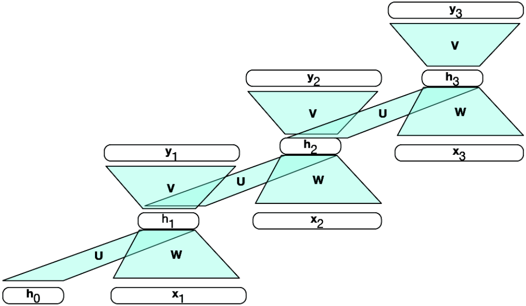
Types of RNNs

- Bi-directional RNNs use future as well as past to model present
- Deep/stacked bi-directional RNNs use multiple layers per time step





Training RNNs



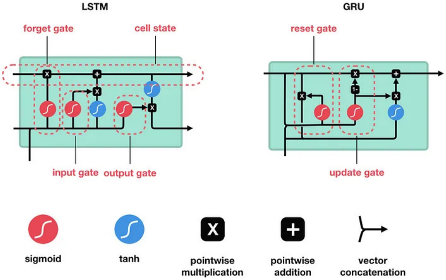
- Simple *backpropagation* does not work since there are loops
- Unfolding removes loops
- Backpropagation is then adopted as backpropagation through time (BPTT)
- Suffers from *vanishing/exploding gradients* problem for long chains

Problems of RNNs

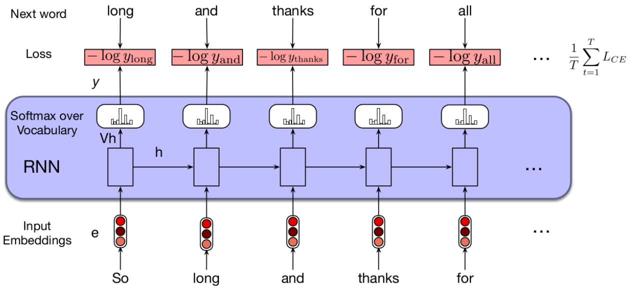
- Suffers from *short-term memory*
  - May leave out important information at the beginning of a sentence
- Vanishing gradients
  - Gradients are multiplied by the chain rule
  - Each gradient is a number between 0 to 1, or -1 to 1
  - Thus, for long sequences, gradients become too small
  - No learning
- *Exploding gradients*
  - For gradients that are greater than 1 or less than -1

Variants of RNNs

- Learn to keep or ignore information in a long sequence
- Neural network structure instead of a single activation function
- Most famous variant is LSTM (Long Short-Term Memory)
  - *Forget gate* that keeps/ignores information
  - *Input gate* to decide which information from current state to process
  - *Cell state* holds current information
  - *Output gate* to pass on information to the next layer
- A simpler variant is GRU (Gated Recurrent Unit)
  - *Update gate* decides what information to process
  - *Reset gate* decides how much of the past information to forget



Language Modeling using RNN

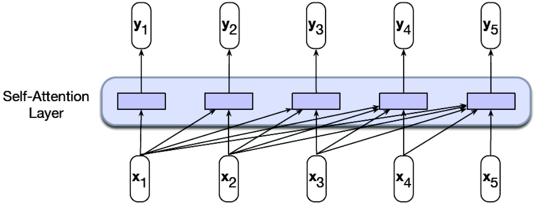


- Keep generating the next word for every time t
- Loss is average *cross-entropy loss*

$$L_{\text{cross-entropy}}(\hat{y}, y) = -\sum_i y_i \ln \hat{y}_i$$

- Word embedding vectors can be one-hot or global (e.g., Word2Vec)
- Teacher forcing sets word at $t-1$ to the *actual* word
- This is passed back to the unit at time t

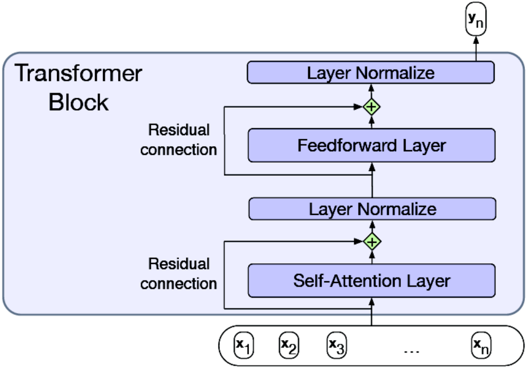
Transformers



- Transformers are *self-attention networks*
- Each unit i uses a *weighted* version of its previous units j as additional input
- This is called attention
- Weight depends on similarity between these two units

$$\alpha_{ij} = \text{softmax}(\vec{x_i} \cdot \vec{x_j}) \ \forall j \leq i = \frac{exp(\vec{x_i} \cdot \vec{x_j})}{\sum_{\forall j} exp(\vec{x_i} \cdot \vec{x_j})} \ \forall j \leq i$$

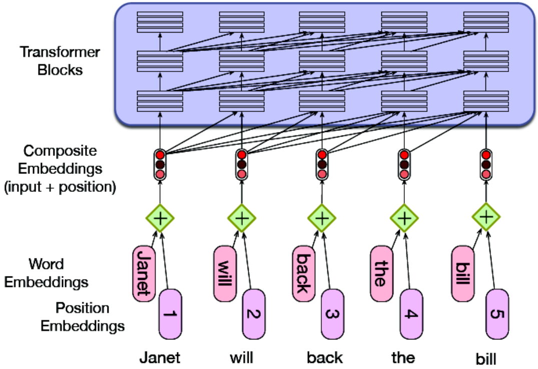
Transformer Block



- Stacked layers form a transformer block
- *Residual connections* short-circuit information by bypassing a layer
- *Layer normalization* constraints outputs to a range

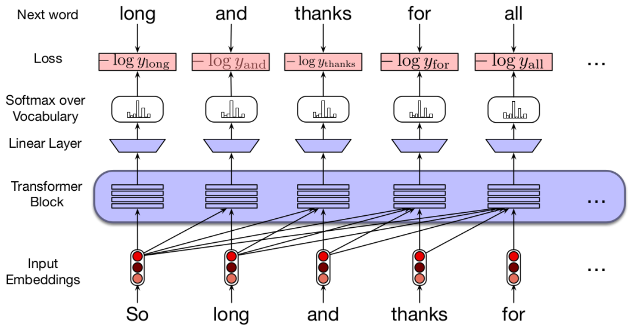
Sequence of Words

- So far, previous time steps act like a bag of words



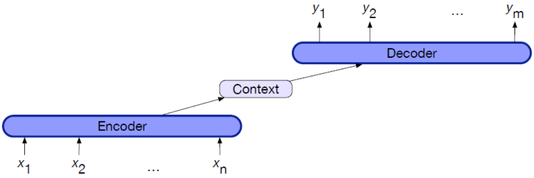
- To encode a sequence, *position vectors* are used
- Embeddings for positions are also learnt

Language Modeling using Transformers



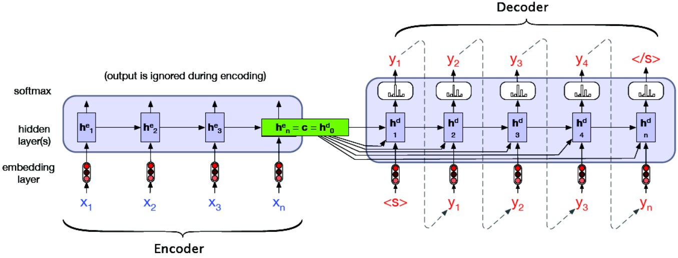
- RNN is replaced by transformer block

Encoder-Decoder Model



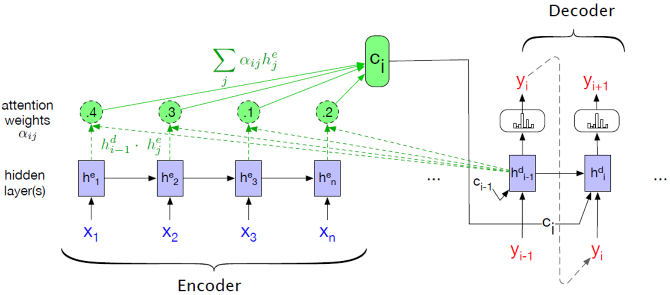
- Encoder accepts an input seqence and generates a sequence of contextualized representations
- Context vector is a function of the last contextual representation
  - This represents the entire sequence
- Decoder generates an arbitrary length sequence of output states

Encoder-Decoder using RNNs



- Starts with a sentence beginning marker <s>
- Keeps generating till a sentence end marker </s> is produced
- Teacher forcing is used
- Only the last encoder state matters
  - *Information bottleneck*
  - Everything about the input sequence must be captured by it
- Can use attention mechanism to resolve

Encoder-Decoder with Attention

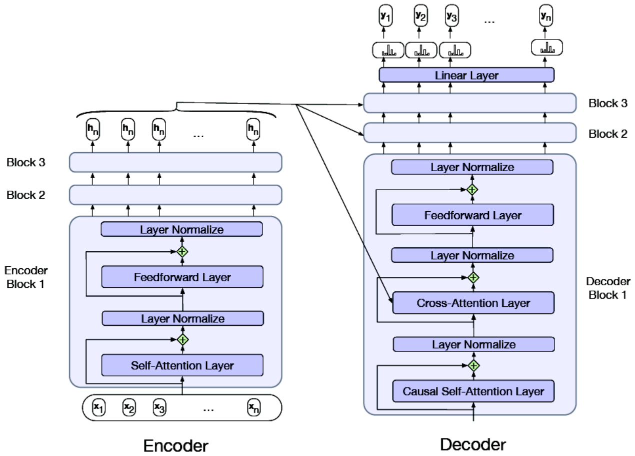


- Each decoder state gets an *attention* from every encoder state

$$\alpha_{ij} = \text{softmax}(\vec{h^d_{i-1}} \cdot \vec{h^e_j}) \; \forall j \in E$$

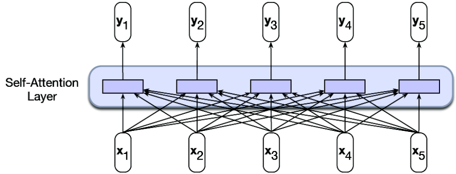
$$\vec{c_i} = \sum_{\forall j} \alpha_{ij} \vec{h^e_j}$$

Encoder-Decoder using Transformer Blocks



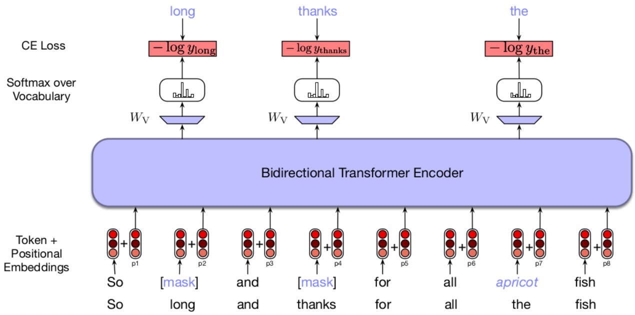
- Enormous number of parameters

Bi-directional Transformer Encoder



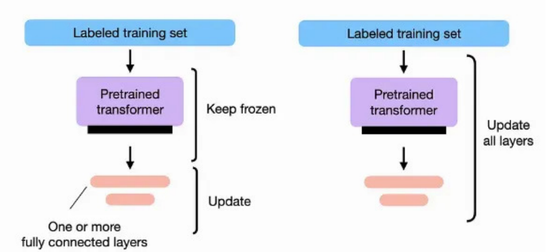
- BERT stands for *Bidirectional Encoder Representations from Transformer*
- Uses self-attention from both past and future
- Simple stacking of layers or using transformer blocks allows a time step to indirectly see itself
- Masking to resolve that

Masked Language Model



- Words are *masked*, i.e., they are replaced by special `[MASK]` tokens
- Sometimes they are replaced deliberately by *unrelated* words
- BERT uses subword tokens instead of actual words
- Produces contextual embeddings, i.e., embeddings of a word in context of the sequence

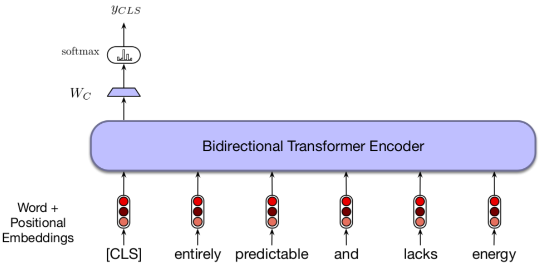
Fine-tuning



Fine-tuning Full training

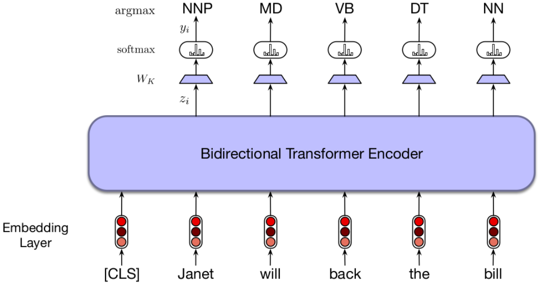
- Fine-tuning allows contextual *pre-trained* word vectors to be used for downstream CL/NLP/IR tasks
- Task-specific training data is needed

Fine-tuning using BERT



- For word tasks, the contextual embedding vector is used
- For sentence tasks, a special `[CLS]` token is prepended
- Vector of `[CLS]` is used for tasks with *task-specific training data*
  - Sentiment classification
  - Sentence entailment

Word Tasks using BERT



- Each individual *pre-trained* word vector is fine-tuned using a separate network for specific tasks
  - POS tagging
  - NER identification
- Since subwords may not be exactly aligned with words, a word is assigned the class to which its first subword belongs to
- Training assigns the golden class to all subwords

Discussion

- BERT is highly successful in many NLP tasks
- Employs a humongous number of parameters (10 crores+)
  - Subword vocabulary of size 30,000
  - Hidden layers of size 768
  - 12 layers of transformer blocks
  - 12 multi-head attention layers in each transformer block
- GPT-3 has 175 billion parameters
- Hence, requires a very large training corpus
- Downstream tasks require smaller sized training corpora
- Quality of corpus is very important since pre-trained vectors are *contextual*

XLNet

- BERT uses auto-encoder (AE) language modeling
- XLNet uses auto-regressive (AR) permutation language modeling
- It generates permutations of words in a sentence
- Consider a 4-length word sequence $x_1 x_2 x_3 x_4$
- It has $4! = 24$ permutations
- In a permutation, let position of word to be predicted is i
  - For example, position of x_3 is 2 in permutation $x_2 x_3 x_4 x_1$
- Only words *before* it, i.e., $x_{p_1}, x_{p_2}, \dots, x_{p_{i-1}}$, in a permutation are considered
  - Only x_2 is considered to predict x_3 in permutation $x_2 x_3 x_4 x_1$
  - All of x_2, x_4, x_1 are considered to predict x_3 in permutation $x_2 x_4 x_1 x_3$
- Lots of such permutations are used as training data

Large Language Model (LLM)

- Pre-training
  - Using the *language modeling* task of predicting the next token
  - Can be automatically generated from a corpus
- Supervised fine-tuning (SFT)
  - Needs a *task-specific* training data
  - A set of prompts and their corresponding responses
    - Prompt: Tell me about Sukumar Ray
    - Response: ⟨written by a human expert⟩
  - Uses the same next token prediction for *generating* the response
- Reward Model Training (RMT)
- Reinforcement Learning from Human Feedback (RLHF)
- The last steps are part of what is called instruction fine-tuning

RMT (Reward Model Training)

- Uses the SFT model as initial base model
- Given a prompt and a response, the aim is to output a *score*
- Given a prompt, *multiple* responses are generated
  - Llama uses beam-width
- Human expert *ranks* these responses
  - Does *not* generate absolute scores
  - May use majority ranking if there are multiple human experts
- RM uses pairwise rankings
- Generates *embeddings* of responses
- Distance from higher ranking to lower ranking should be *maximized*
- If $R_2 \succ R_1$, then

$$\max[\sigma(E_{RM}(P \cdot R_2), E_{RM}(P \cdot R_1))]$$

- $E_{RM}(P \cdot R)$ for prompt P and response R is made a scalar $RM_\theta(P \cdot R)$ by multiplying with a weight vector
- $RM_\theta(P \cdot R)$ is the score

$$\min[-\sigma(|RM_\theta(P \cdot R_2) - RM_\theta(P \cdot R_1)|)]$$

RLHF (Reinforcement Learning from Human Feedback)

- Human feedback is simulated by using the RM model
- Essentially, the SFT model is further fine-tuned using the RM model
- For a prompt, SFT model is encouraged to generate a response a high score
- Find the best model Π_{RL} generated by RL from SFT

$$\max_{\Pi_{RL}} RM(\Pi_{RL}(Response|Prompt))$$

- However, RM may give high scores even for garbage
- Response should not deviate too much from the original SFT model
- Penalty term to capture deviation
- If new model is Π_{RL} and original model is Π_{SFT}

$$\max_{\Pi_{RL}} [RM(\Pi_{RL}(Response|Prompt)) - \beta \cdot KL(\Pi_{RL}, \Pi_{SFT})]$$

- *Input* is made part of prompt while *output* is response
  - If prompt is same, can be made system prompt
  - Special token to separate system prompt from input

Efficiency in LLMs

- Parameter Efficient Finetuning (PEFT)
  - Only certain layers are trained
- Low Ranked Adaptation (LoRA)
  - Weight updates through two smaller matrices
  - Low-rank decomposition