3. Sequence Modeling

Neural networks basics

Key idea in neural nets: feature/representation learning

Building blocks:

- Input layer: raw features
- Hidden layer: perceptron + nonlinear activation function
- Output layer: linear (+ transformation, e.g. softmax)

Optimization:

- Optimize by SGD (implemented by back-propogation)
- Objective is non-convex, may not reach a global minimum

Recurrent neural networks

Problem setup: given an input sequence, come up with a model that outputs a representation of the sequence for downstream tasks (e.g., classification)

Key challenge: how to model interaction among words?

Approach:

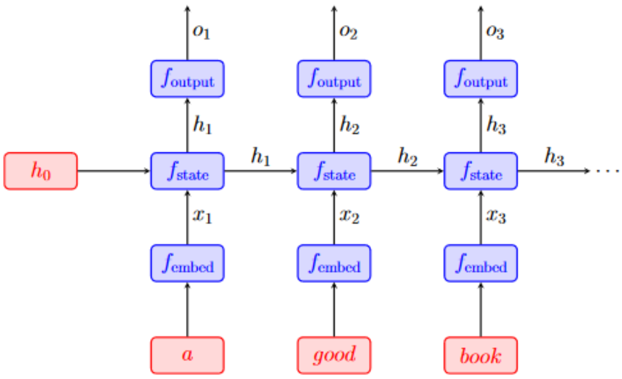
- Aggregation / pooling word embeddings
- Recurrence
- Self-attention

Goal: represent a sequence of symbols of **varying lengths**

Idea: combine new symbols with previous symbols recurrently by modeling the **temporal dynamics**$h_t = f(h_{t-1}, x_t)$

Forward pass

Use o_t's as features



$$x_t = f_{\text{embed}}(s_t)$$
$$= W_e \phi_{\text{one-hot}}(s_t)$$

$$h_t = f_{\text{state}}(x_t, h_{t-1})$$
$$= \sigma(W_{hh}h_{t-1} + W_{ih}x_t + b_h)$$

$$o_t = f_{\text{output}}(h_t)$$
$$= W_{ho}h_t + b_o$$

A deep neural network with shared weights in each layer

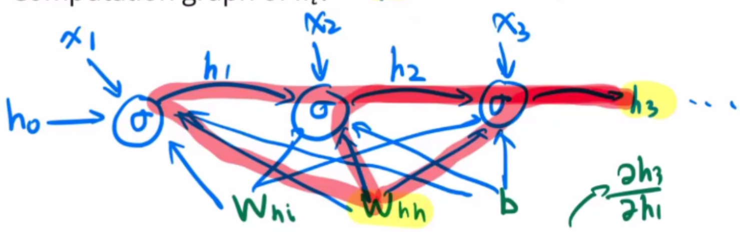
Which computation can be parallelized?

Backward pass

Given the loss ℓ, compute the gradient with respect to W_{hh}.

$$\frac{\partial \ell}{\partial W_{hh}} = \frac{\partial \ell}{\partial o_t}\frac{\partial o_t}{\partial h_t}\frac{\partial h_t}{\partial W_{hh}}$$

Computation graph of h_t: $h_t = \sigma(W_{hh}h_{t-1} + W_{hi}x_t + b)$



$$\frac{\partial h_3}{\partial W_{hh}} = \frac{\partial h_3}{\partial W_{hh}} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{hh}} + \underbrace{\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}}\frac{\partial h_1}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial W_{hh}} = \sum_{i=1}^{t} \boxed{\frac{\partial h_t}{\partial h_i}} \frac{\partial h_i}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial h_i} = \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_{i+1}}{\partial h_i} = (\sigma' W_{hh})^{t-i}$$

$$W_{hh}^k = Q \Lambda^k Q^T$$

Problem with standard backpropagation:

- Gradient involves repeated multiplication of W_{hh}
- Gradient will vanish / explode (depending on the eigenvalues of W_{hh})

Quick fixes:

- Reduce the number of repeated multiplication: truncate after k steps (h_{t-k} has no influence on h_t)
- Limit the norm of the gradient in each step: gradient clipping (can only mitigate explosion)

LSTM

Vanilla RNN: always update the hidden state

LSTM: learn when to update or reset a hidden state, no gradient vanishing and explosion problem

$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$ update with the new input x_t

c_{t-1} no update with x_t

Choose between $\tilde{c}_t$ (update) and c_{t-1} (no update): ($\odot$ means elementwise multiplication):

memory cell $c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}$

- i_t: proportion of the new state
- f_t: proportion of the old state

Input gate and forget gate (each between 0 and 1) depends on data:

- $i_t = \mathrm{sigmoid}(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$
- $f_t = \mathrm{sigmoid}(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$

How much should the memory cell state influence the rest of the network:

- $o_t = \mathrm{sigmoid}(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$ Output Gate
- $h_t = o_t \odot c_t$

Intuition: gating allows the network to learn to control how much gradient should vanish

Self-attention

Idea: model interaction between words in parallel using a ”soft” database

- Input: map each symbol to a query, a key, and a value (embeddings)
- Attend: each word (as a query) interacts with all words (keys)
- Output: contextualized representation of each word (weighted sum of values)

Scaled dot-product attention: $a(q, k) = \frac{q \cdot k}{\sqrt{d_k}}$

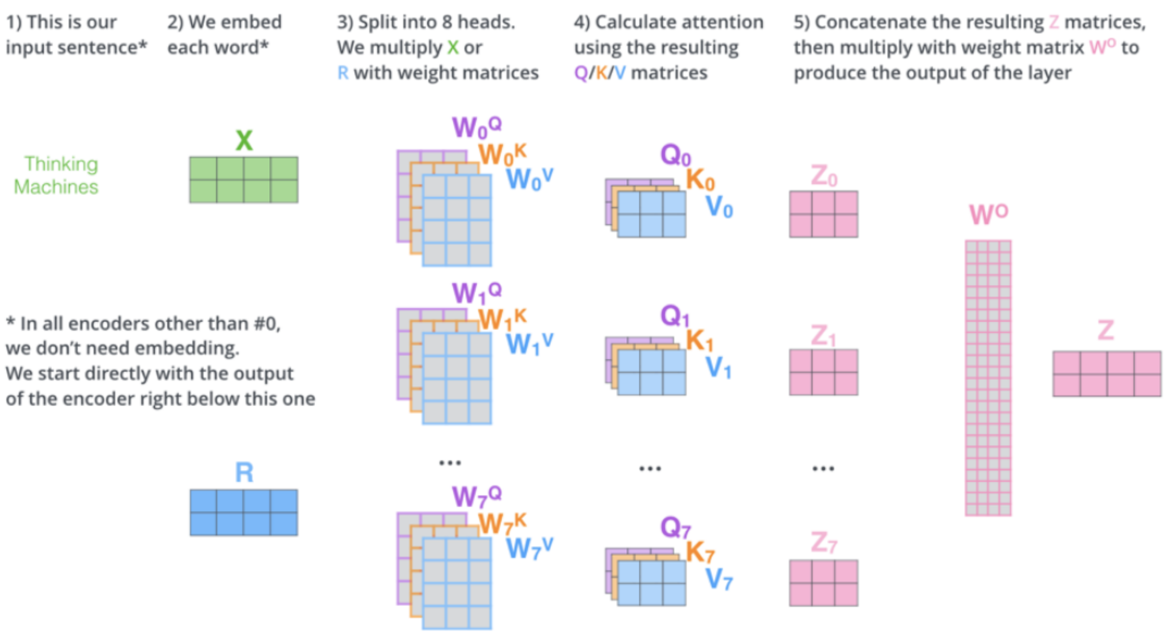
Multi-head attention

- Multiple attention modules: same architecture, different parameters
- A head: one set of attention outputs
- Concatenate all heads (increased output dimension)
- Linear projection to produce the final output

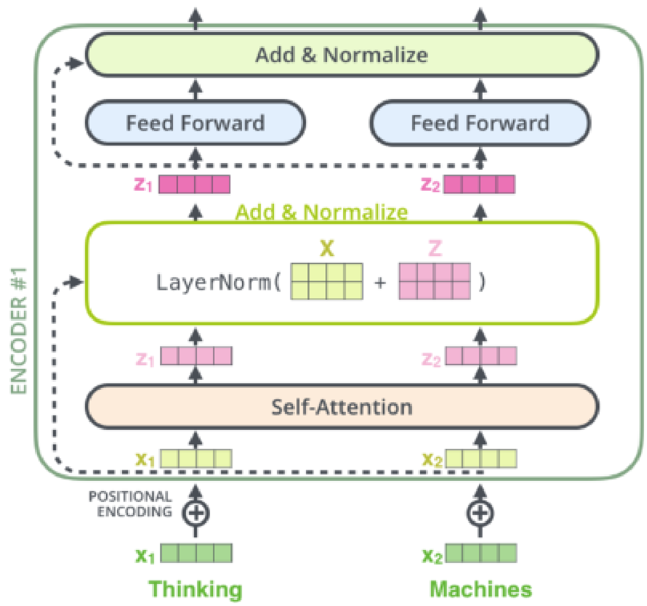
Input → embedding x → maps to query q, key k, value v → score z

$XW^Q = Q, XW^K = K, XW^V = V$

$\mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V = Z$



Tranformer



- Multi-head self-attention: capture dependence among input symbols
- Positional encoding: capture the order of symbols
  - Positional embeddings are added to the vector representations of the input tokens
  - Sinusoidal position embedding or Learned absolute position embeddings (most used now)
- Residual connection and layer normalization: more efficient and better optimization
  - Residual connection [He et al., 2015]
  - Layer normalization [Ba et al., 2016]: normalize (zero mean, unit variance) across features

$$\mathrm{LayerNorm(x)} = \frac{x-\mu}{\sigma}$$