Lecture 25

[RNN: backprop through time]

Consider $\begin{cases} \vec{h}_t = W_{hx}\vec{x}_t + W_{hh}\vec{h}_{t-1}, \\ \vec{o}_t = W_{ho}\vec{h}_t \end{cases}$ no biases for simplicity

Loss function: $L = \frac{1}{T}\sum_{t=1}^{T} \ell(y_t, \vec{o}_t)$ (y_t: true target labels)

Need to compute

$$\frac{\partial L}{\partial W_{hx}}, \quad \frac{\partial L}{\partial W_{hh}}, \quad \frac{\partial L}{\partial W_{ho}}$$ matrix derivatives

$$\frac{\partial L}{\partial W_{ho}} = \frac{1}{T}\sum_t \frac{\partial \ell(y_t, \vec{o}_t)}{\partial W_{ho}} = \frac{1}{T}\sum_t \underbrace{\frac{\partial \ell(y_t, \vec{o}_t)}{\partial \vec{o}_t}\frac{\partial \vec{o}_t}{\partial W_{ho}}}_{\text{local in time}}$$

Next, define $\vec{w}_h = \{W_{hh}, W_{hx}\}$, $\vec{w}_o = \{W_{ho}\}$ flattened weights

Then $\begin{cases} \vec{h}_t = \vec{f}(\vec{x}_t, \vec{h}_{t-1}, \vec{w}_h), \\ \vec{o}_t = \vec{g}(\vec{h}_t, \vec{w}_o) \end{cases}$ give

$$\frac{\partial L}{\partial \vec{w}_h} = \frac{1}{T}\sum_t \frac{\partial \ell(y_t, \vec{o}_t)}{\partial \vec{o}_t}\frac{\partial \vec{g}(\vec{h}_t, \vec{w}_o)}{\partial \vec{h}_t}\frac{\partial \vec{h}_t}{\partial \vec{w}_h}$$

Next, $$\frac{\partial \vec{h}_t}{\partial \vec{w}_h} = \frac{\partial \vec{f}(\vec{x}_t, \vec{h}_{t-1}, \vec{w}_h)}{\partial \vec{w}_h} +$$

$$+ \frac{\partial \vec{f}(\vec{x}_t, \vec{h}_{t-1}, \vec{w}_h)}{\partial \vec{h}_{t-1}} \frac{\partial \vec{h}_{t-1}}{\partial \vec{w}_h} \quad \leftarrow \text{recursive update}$$

This yields

$$\frac{\partial \vec{h}_t}{\partial \vec{w}_h} = \frac{\partial \vec{f}(\vec{x}_t, \vec{h}_{t-1}, \vec{w}_h)}{\partial \vec{w}_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial \vec{f}(\vec{x}_j, \vec{h}_{j-1}, \vec{w}_h)}{\partial \vec{h}_{j-1}} \right) \times$$

$$\times \frac{\partial f(\vec{x}_i, \vec{h}_{i-1}, \vec{w}_h)}{\partial \vec{w}_h} .$$

$O(T^2)$ complexity

Thus, the sum is truncated to the most recent K terms (K may be chosen adaptively)

[Gating and long-term memory]

RNNs "forget" inputs from the past due to vanishing gradients. Solutions: GRU + LSTM.

gated recurrent units (GRU)

Consider $X_t = N \times D$ data matrix

N: batch size; D: vocab. size (input data dim'n)

$H_t = N \times H$

↑ # hidden states (i.e., units)

Compute ($D \times H$, $H \times H$, $N \times H$ with the bias vector b_r in each row)

reset gate: $R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$ $N \times H$

update gate: $Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$ $N \times H$

$R_t, Z_t \in [0,1]$

Next, compute (element-wise $\otimes$)

$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \otimes H_{t-1}) W_{hh} + b_h)$ $N \times H$

$\tilde{H}_t \in [-1, 1]$

If all entries in R_t are close to 1, $\tilde{H}_t$ is approx. like H_t from the standard RNN.

If all entries in R_t are close to 0, $\tilde{H}_t$ 'forgets' H_{t-1} and acts as a simple MLP. Thus, reset gate captures <u>short-term</u> dependencies.

Finally, compute ($N \times H$ matrix of ones)

$H_t = Z_t \otimes H_{t-1} + (1 - Z_t) \otimes \tilde{H}_t$

when $Z_{ij} \approx 1 \Rightarrow H_{t,ij} \approx H_{t-1,ij}$, hidden state passed unchanged [i.e., X_t is ignored]

when $Z_{ij} \approx 0 \Rightarrow H_{t,ij} \approx \tilde{H}_{t,ij}$

Overall, the update gate captures long-term dependencies

Long short term memory (LSTM)

Compute

$$\begin{cases} O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o), & \text{output gate} \\ I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i), & \text{input gate} \\ F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) & \text{(not) forget gate} \end{cases}$$

Candidate cell state:

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$

Cell state:

$$C_t = F_t \otimes C_{t-1} + I_t \otimes \tilde{C}_t$$

↑ C_{t-1} included if F_t is 'on'

↑ $\tilde{C}_t$ included if I_t is 'on'

$F_t \approx 1$ & $I_t \approx 0$ over many timesteps ~ long-term memories, stored in cells C_t

Finally, $H_t = \underbrace{O_t \otimes \tanh(C_t)}$

short-term memory influenced by long-term memory stored in C_t

Attention

Consider m feature vectors of length v referenced by m keys of length k:

$$K \in R^{m \times k} \Rightarrow V \in R^{m \times v}$$

The model will produce a weighted combination of m feature vectors based on the input query vector $\vec{q} \in R^q$.

'Soft' (differentiable) lookup:

$$\text{Attn}(\vec{q}; (\vec{k}_1, \vec{v}_1) \dots (\vec{k}_m, \vec{v}_m)) = \qquad \dim = v$$

$$= \text{Attn}(\vec{q}; \vec{k}_{1:m}, \vec{v}_{1:m}) = \sum_{i=1}^{m} \underbrace{\alpha_i(\vec{q}, \vec{k}_{1:m})}_{\text{attention weights}} \vec{v}_i$$

$$0 \leq \alpha_i \leq 1 \; ; \quad \sum_{i=1}^{m} \alpha_i = 1 .$$

$$\text{Explicitly}, \quad \alpha_i(\vec{q}, \vec{k}_{1:m}) = \frac{e^{a(\vec{q}, \vec{k}_i)}}{\sum_{j=1}^{m} e^{a(\vec{q}, \vec{k}_j)}},$$

where $a(\vec{q}, \vec{k}_i) \in R$ = attention score

Parametric attention: assume that

$d \equiv q = k$, compute

$$a(\vec{q}, \vec{k}) = \frac{\vec{q} \cdot \vec{k}}{\underbrace{\sqrt{d}}_{\text{to scale out the variance}}}$$

In practice, we deal with n query vectors at the same time:

$$Q \in \mathbb{R}^{n \times d}, \quad K \in \mathbb{R}^{m \times d}, \quad V \in \mathbb{R}^{m \times v}$$

Then $$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{\overbrace{Q K^T}^{n \times m}}{\sqrt{d}}\right) \overbrace{V}^{m \times v} \in \mathbb{R}^{n \times v}$$

softmax applied row-wise

seq2seq with attention (RNN)

Recall that in RNN decoder,

$$\vec{h}_t^d = \vec{f}_d(\vec{h}_{t-1}^d, \vec{y}_{t-1}, \vec{c})$$

$\vec{c}$: context vector, encodes (entire) input $\vec{x}_{1:T}$

Typically, $\vec{c} = \vec{h}_T^e$ ⇐ final hidden state of RNN encoder

Idea: replace $\vec{c}$ with (feature i)

$$\vec{c}_t = \sum_{i=1}^{T} \alpha_i(\vec{h}_{t-1}^d, \vec{h}_{1:T}^e)\, \vec{h}_i^e$$

"decoder attends to encoder"

$\vec{h}_{t-1}^d$: query; $\vec{h}_{1:T}^e$: set of T keys

Then $$\vec{h}_t^d = \vec{f}_d(\vec{h}_{t-1}^d, \vec{y}_{t-1}, \vec{c}_t)$$

Transformers:

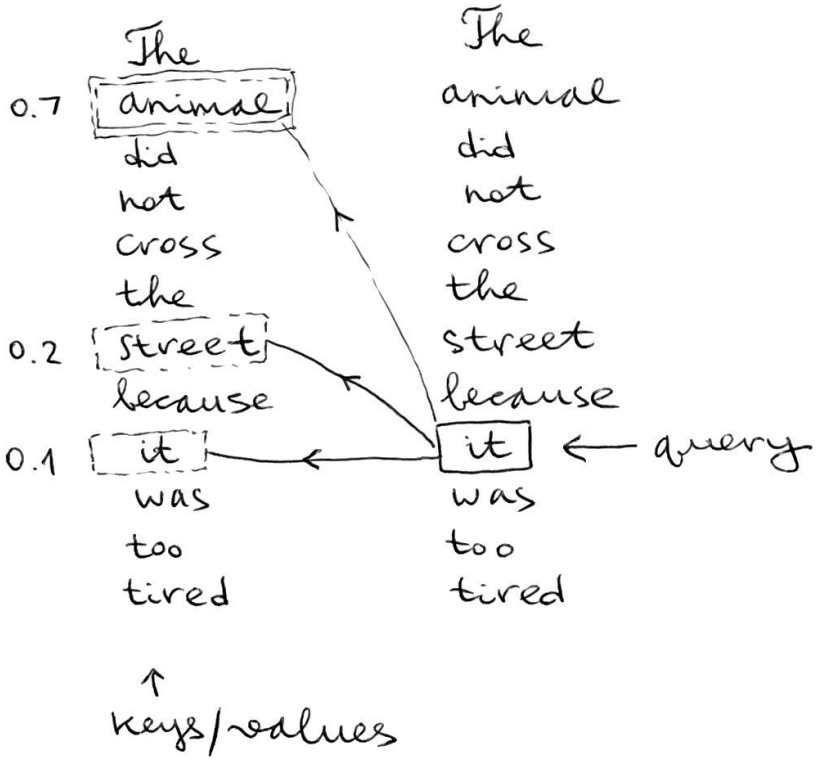
seq2seq model which uses attention in both encoder and decoder.

Applications: machine translation, music generation, text summarization, image generation (treat image as seqn of pixels), etc.

Self-attention: encoder attends to itself

given input tokens $\vec{x}_1, \ldots, \vec{x}_n$ $\vec{x}_i \in \mathbb{R}^d$,

generate $\underbrace{\vec{y}_i}_{\text{dim} = d} = \text{Attn}(\underbrace{\vec{x}_i}_{\text{query}}; (\underbrace{\vec{x}_1}_{\text{key}}, \underbrace{\vec{x}_1}_{\text{value}}) \ldots (\vec{x}_n, \vec{x}_n))$



Multi-headed attention (MHA): caption several notions of similarity at once.

Consider $\vec{q}_q \in R^{d_q}$, $\vec{k}_j \in R^{d_k}$, $\vec{v}_j \in R^{d_v}$

Define embeddings: $\underbrace{\vec{q}'_i}_{p} = \underbrace{W_i^{(q)}}_{p \times d_q} \underbrace{\vec{q}_q}_{d_q}$; $i = 1, \ldots, h$

$$\underbrace{\vec{k}'_{i,j}}_{p} = \underbrace{W_i^{(k)}}_{p \times d_k} \underbrace{\vec{k}_j}_{d_k}; \quad \underbrace{\vec{v}'_{i,j}}_{p_v} = \underbrace{W_i^{(v)}}_{p_v \times d_v} \underbrace{\vec{v}_j}_{d_v}$$

Compute $\underbrace{\vec{h}_i}_{p_v} = \text{Attn}(\vec{q}'_i; (\vec{k}'_{i,1}\ \vec{v}'_{i,1})(\vec{k}'_{i,2}\vec{v}'_{i,2})\ldots)$

Finally, $\vec{h} = \text{MHA}(\vec{q}; (\vec{k}_1 \vec{v}_1)(\vec{k}_2 \vec{v}_2)\ldots) =$

$$= \underbrace{W_o}_{p_o \times (h p_v)} \begin{pmatrix} \vec{h}_1 \\ \vdots \\ \vec{h}_h \end{pmatrix} \in R^{p_o}.$$

Positional embedding: use a set of basis functions to encode word positions in the sequence.

Consider a sequence of tokens labeled by $i = 1, \ldots, n$.

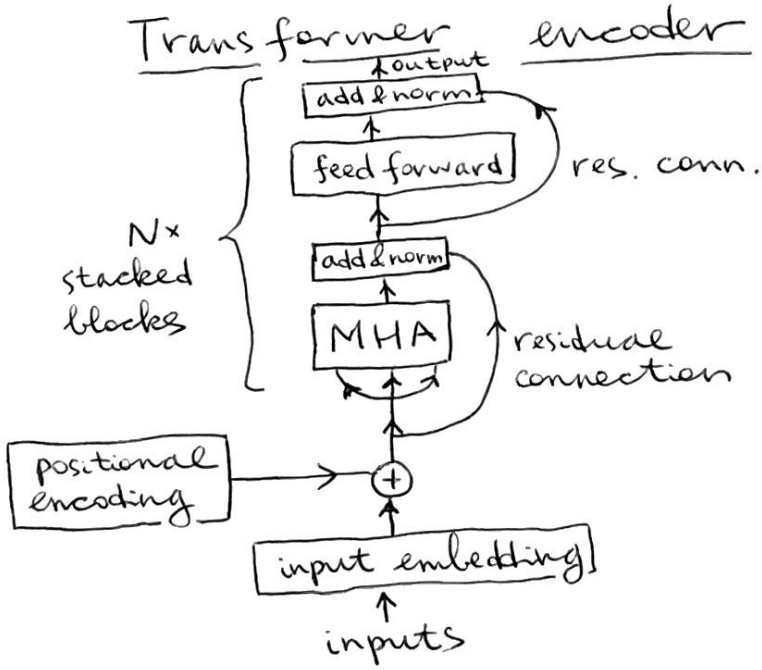
Compute $\begin{cases} p_{i,2j} = \sin\left(\frac{i}{C^{2j/d}}\right), \\ p_{i,2j+1} = \cos\left(\frac{i}{C^{2j/d}}\right). \end{cases}$

Here, $C = 10^4$ = max sequence length;
d = dim of embedding;
$j = 1, \ldots, d$.

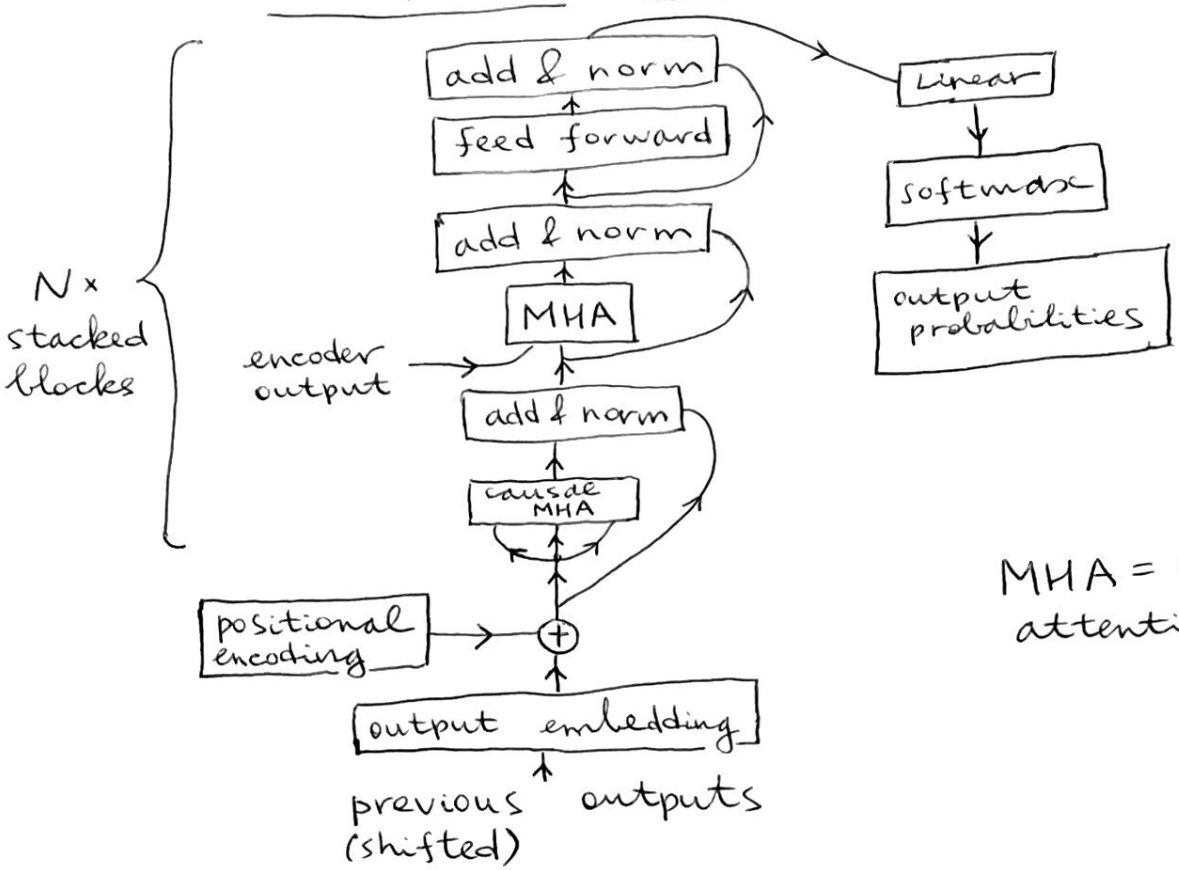
For example, if $d = 4$, token i is encoded as $\vec{p}_i = \left(\sin\frac{i}{C^{0/4}}, \cos\frac{i}{C^{0/4}}, \sin\frac{i}{C^{2/4}}, \cos\frac{i}{C^{2/4}}\right)$.

If a token is represented by a (learned) embedding $\vec{x}_i$ of length d, we compute $\vec{p}_i$ of length d and add them:

$$\text{Pos}(\text{Embed}(\text{token}_i)) = \vec{x}_i + \vec{p}_i .$$



Transformer decoder



MHA = multi-headed attention

Causal (or masked) MHA : future tokens masked / unavailable