Sequence models

Lecture 11 — CS 577 Deep Learning

Instructor: Yutong Wang

Computer Science
Illinois Institute of Technology

October 30, 2024

Topics

- Temporal convolutional networks
- Recurrent neural networks
- Transformers

Sequences

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$$

- Example: Time series (acoustic signal, weather, stock price,...)

Sequences

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$$

- Example: Time series (acoustic signal, weather, stock price,...)
- Example: Sequence of word/token embeddings

Sequences

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$$

- Example: Time series (acoustic signal, weather, stock price,...)
- Example: Sequence of word/token embeddings

- Task: denoising (make a signal less noisy)

Sequences

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$$

- Example: Time series (acoustic signal, weather, stock price,...)
- Example: Sequence of word/token embeddings

- Task: denoising (make a signal less noisy)
- Task: forecasting (what is the weather like next week?)

Sequences

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$$

- Example: Time series (acoustic signal, weather, stock price,...)
- Example: Sequence of word/token embeddings

- Task: denoising (make a signal less noisy)
- Task: forecasting (what is the weather like next week?)
- Task: classification (is this news article about sport or about technology?)

Sequences

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$$

- Example: Time series (acoustic signal, weather, stock price,...)
- Example: Sequence of word/token embeddings

- Task: denoising (make a signal less noisy)
- Task: forecasting (what is the weather like next week?)
- Task: classification (is this news article about sport or about technology?)
- Task: regression (how favorable is this product review?)

Causal models

$$f(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}; \theta) = \mathbf{y}^{(t)}$$

Causality assumption

The output $\mathbf{y}^{(t)}$ does not depend on future values of $\mathbf{x}^{(t')}$ where $t' > t$

Prediction/forecasting/generative models

- Let $\hat{\mathbf{x}}^{(t+1)} := \mathbf{y}^{(t)}$
-
$$\underbrace{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(C)}}_{\text{observed}} \quad \overset{f}{\to} \quad \underbrace{\hat{\mathbf{x}}^{(C+1)}, \hat{\mathbf{x}}^{(C+2)}, \ldots}_{\text{predicted/generated}}$$
- Next: for simplicity, let's consider $x^{(t)} \in \mathbb{R}$. (So drop the bold fontface.)

Autoregressive model

- For $t \geq s$

$$
\begin{aligned}
y^{(t)} &= f(x^{(1)}, \ldots, x^{(t)}; w_0, \ldots, w_{s-1}, b) \\
&= \sum_{\tau=0}^{s-1} w_\tau x^{(t-\tau)} + b
\end{aligned}
$$

Autoregressive model

- For $t \geq s$

$$\begin{aligned} y^{(t)} &= f(x^{(1)}, \ldots, x^{(t)}; w_0, \ldots, w_{s-1}, b) \\ &= \sum_{\tau=0}^{s-1} w_\tau x^{(t-\tau)} + b \end{aligned}$$

- Loss

$$\frac{1}{C} \sum_{t=1}^{C} (x^{(t+1)} - f(x^{(1)}, \ldots, x^{(t)}; w_0, \ldots, w_{s-1}, b))^2$$

Autoregressive model

- For $t \geq s$

$$
\begin{aligned}
y^{(t)} &= f(x^{(1)}, \ldots, x^{(t)}; w_0, \ldots, w_{s-1}, b) \\
&= \sum_{\tau=0}^{s-1} w_\tau x^{(t-\tau)} + b
\end{aligned}
$$

- Loss

$$
\frac{1}{C} \sum_{t=1}^{C} (x^{(t+1)} - f(x^{(1)}, \ldots, x^{(t)}; w_0, \ldots, w_{s-1}, b))^2
$$

- Solve for $w_0, \ldots, w_{s-1}, b$

Autoregressive model

- For $t \geq s$

$$\begin{aligned} y^{(t)} &= f(x^{(1)}, \ldots, x^{(t)}; w_0, \ldots, w_{s-1}, b) \\ &= \sum_{\tau=0}^{s-1} w_\tau x^{(t-\tau)} + b \end{aligned}$$

- Loss

$$\frac{1}{C} \sum_{t=1}^{C} (x^{(t+1)} - f(x^{(1)}, \ldots, x^{(t)}; w_0, \ldots, w_{s-1}, b))^2$$

- Solve for $w_0, \ldots, w_{s-1}, b$
- Next: `seq2col`

Seq2col: in class exercise 1

In class exercise

- Complete the `seq2col` function
- Discuss with your neighbors regarding questions in the “Fibonacci” block

Seq2col: in class exercise 1 discussion

$$x^{(t+1)} = x^{(t)} + x^{(t-1)}$$

How do you interpret the coefficients when we set `filter_size = 3`?

```
filter_size = 3
def fibonacci(n):
#    [...]

fibonnaci_seq = list(fibonacci(20))

X, y = seq2col(fibonnaci_seq[:10], filter_size)

X_tilde = np.hstack([X, np.ones((X.shape[0], 1))])
w = np.linalg.pinv(X_tilde) @ y
np.round(w,5)
# array([-0.,  1.,  1.,  0.])
```

Another way to look at autoregressive models

- Drop the bias term for simplicity

$$y^{(t)} = w_0 x^{(t)} + \sum_{\tau=1}^{s-1} w_\tau x^{(t-\tau)} = w_0 x^{(t)} + \begin{bmatrix} w_1 \\ \vdots \\ w_{s-1} \end{bmatrix}^\top \begin{bmatrix} x^{(t-1)} \\ \vdots \\ x^{(t-(s-1))} \end{bmatrix}$$

Another way to look at autoregressive models

- Drop the bias term for simplicity

$$y^{(t)} = w_0 x^{(t)} + \sum_{\tau=1}^{s-1} w_\tau x^{(t-\tau)} = w_0 x^{(t)} + \begin{bmatrix} w_1 \\ \vdots \\ w_{s-1} \end{bmatrix}^\top \begin{bmatrix} x^{(t-1)} \\ \vdots \\ x^{(t-(s-1))} \end{bmatrix}$$

- History up to time $t-1$ (aka hidden state at time $t-1$)

$$y^{(t)} = w_0 x^{(t)} + \begin{bmatrix} w_1 \\ \vdots \\ w_{s-1} \end{bmatrix}^\top \tilde{\mathbf{h}}^{(t-1)}$$

Another way to look at autoregressive models

- Drop the bias term for simplicity

$$y^{(t)} = w_0 x^{(t)} + \sum_{\tau=1}^{s-1} w_\tau x^{(t-\tau)} = w_0 x^{(t)} + \begin{bmatrix} w_1 \\ \vdots \\ w_{s-1} \end{bmatrix}^\top \begin{bmatrix} x^{(t-1)} \\ \vdots \\ x^{(t-(s-1))} \end{bmatrix}$$

- History up to time $t-1$ (aka hidden state at time $t-1$)

$$y^{(t)} = w_0 x^{(t)} + \begin{bmatrix} w_1 \\ \vdots \\ w_{s-1} \end{bmatrix}^\top \tilde{\mathbf{h}}^{(t-1)}$$

- Next: directly model the hidden state

Recurrent unit (with linear activation)

- Recurrent unit

$$\mathbf{h}^{(t)} = f(x^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \mathbf{W}_{\mathtt{rec}}\mathbf{h}^{(t-1)} + \mathbf{w}_{\mathtt{in}}x^{(t)}$$

Recurrent unit (with linear activation)

- Recurrent unit

$$\mathbf{h}^{(t)} = f(x^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \mathbf{W}_{\texttt{rec}}\mathbf{h}^{(t-1)} + \mathbf{w}_{\texttt{in}}x^{(t)}$$

- $\mathbf{h}^{(t)} \in \mathbb{R}^{d_{\texttt{h}}}$ (hidden) states at time t

Recurrent unit (with linear activation)

- Recurrent unit

$$\mathbf{h}^{(t)} = f(x^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \mathbf{W}_{\mathtt{rec}}\mathbf{h}^{(t-1)} + \mathbf{w}_{\mathtt{in}}x^{(t)}$$

- $\mathbf{h}^{(t)} \in \mathbb{R}^{d_{\mathtt{h}}}$ (hidden) states at time t
- $d_{\mathtt{h}}$ hidden dimension

Recurrent unit (with linear activation)

- Recurrent unit

$$\mathbf{h}^{(t)} = f(x^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \mathbf{W}_{\mathtt{rec}}\mathbf{h}^{(t-1)} + \mathbf{w}_{\mathtt{in}}x^{(t)}$$

- $\mathbf{h}^{(t)} \in \mathbb{R}^{d_{\mathtt{h}}}$ (hidden) states at time t
- $d_{\mathtt{h}}$ hidden dimension
- Read out $\mathbf{w}_{\mathtt{out}} \in \mathbb{R}^{d_{\mathtt{h}}}$

$$y^{(t)} = \mathbf{w}_{\mathtt{out}}^{\top}\mathbf{h}^{(t)}$$

Recurrent unit (with linear activation)

- Recurrent unit

$$\mathbf{h}^{(t)} = f(x^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \mathbf{W}_{\texttt{rec}}\mathbf{h}^{(t-1)} + \mathbf{w}_{\texttt{in}}x^{(t)}$$

- $\mathbf{h}^{(t)} \in \mathbb{R}^{d_{\texttt{h}}}$ (hidden) states at time t
- $d_{\texttt{h}}$ hidden dimension
- Read out $\mathbf{w}_{\texttt{out}} \in \mathbb{R}^{d_{\texttt{h}}}$

$$y^{(t)} = \mathbf{w}_{\texttt{out}}^{\top}\mathbf{h}^{(t)}$$

- $\mathbf{W}_{\texttt{rec}} \in \mathbb{R}^{d_{\texttt{h}} \times d_{\texttt{h}}}$, $\mathbf{w}_{\texttt{in}} \in \mathbb{R}^{d_{\texttt{h}}}$ and $\mathbf{w}_{\texttt{out}} \in \mathbb{R}^{d_{\texttt{h}}}$ are parameters

Recovering the autoregressive model

- Autoregressive (AR) model (with filter size 4)

$$y^{(t)} = w_0 x^{(t)} + \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}^\top \tilde{\mathbf{h}}^{(t-1)} \quad \text{where} \quad \tilde{\mathbf{h}}^{(t-1)} = \begin{bmatrix} x^{(t-1)} \\ x^{(t-2)} \\ x^{(t-3)} \end{bmatrix}$$

- Recurrent unit: how can we choose the parameters to recover the AR model?

$$\mathbf{h}^{(t)} = f(x^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \mathbf{W}_{\texttt{rec}} \mathbf{h}^{(t-1)} + \mathbf{w}_{\texttt{in}} x^{(t)}$$

Recurrent unit (with linear activation)

- Recurrent unit

$$\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \mathbf{W}_{\texttt{rec}}\mathbf{h}^{(t-1)} + \mathbf{W}_{\texttt{in}}\mathbf{x}^{(t)}$$

- $\mathbf{W}_{\texttt{rec}}$ and $\mathbf{W}_{\texttt{in}}$ are parameters
- $\mathbf{h}^{(t)}$ (hidden) states at time t

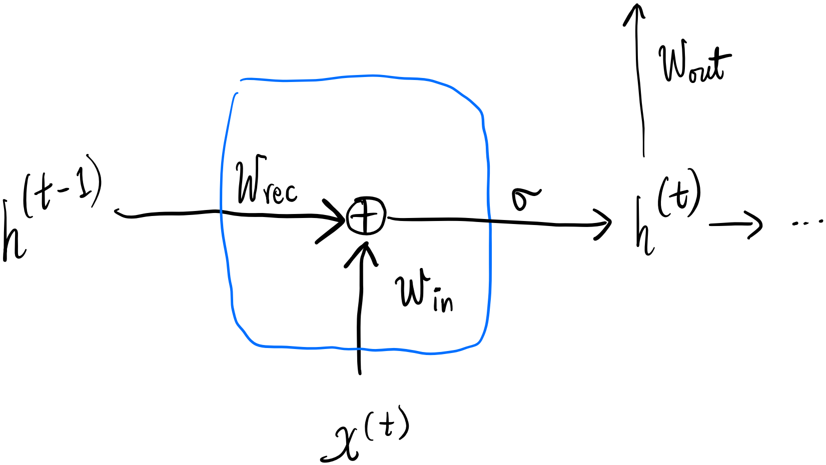
Recurrent unit (with linear activation)

- Recurrent unit

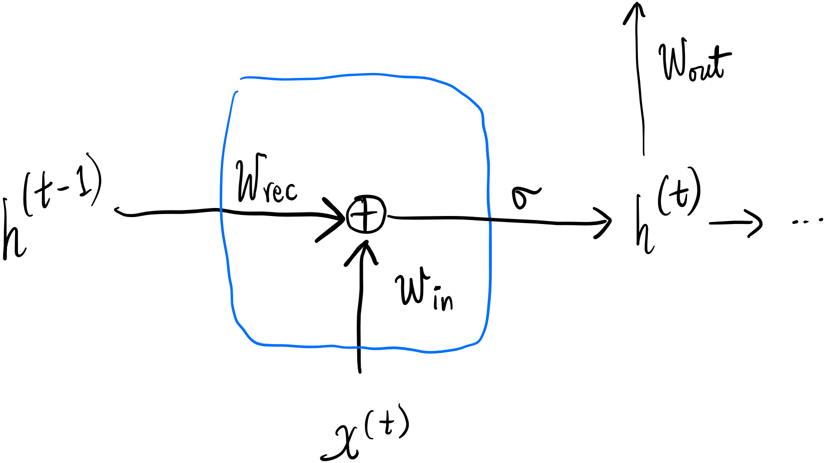
$$\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \sigma(\mathbf{W}_{\mathtt{rec}}\mathbf{h}^{(t-1)} + \mathbf{W}_{\mathtt{in}}\mathbf{x}^{(t)})$$

- $\mathbf{W}_{\mathtt{rec}}$ and $\mathbf{W}_{\mathtt{in}}$ are parameters
- $\mathbf{h}^{(t)}$ (hidden) states at time t
- σ can be the relu, hyperbolic tangent, or anything

Recurrent unit



Recurrent unit



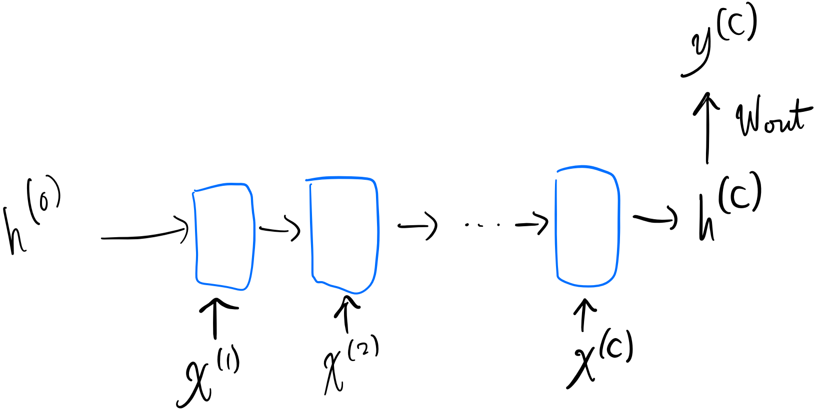
Recurrent unit (with linear activation)

- Recurrent unit

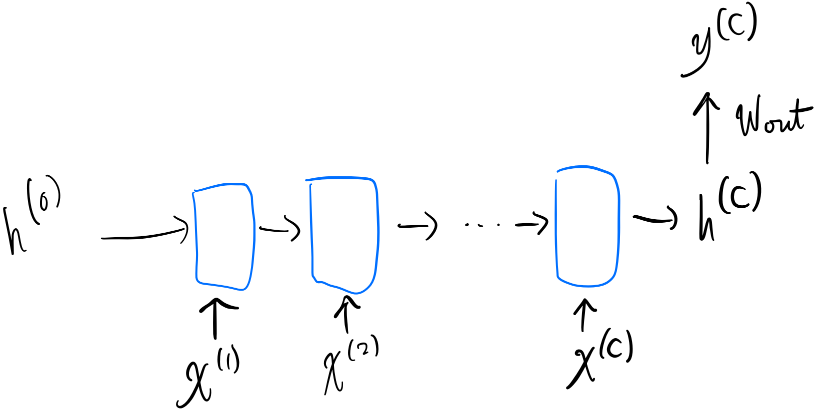
$$\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \sigma(\mathbf{W}_{\mathtt{rec}}\mathbf{h}^{(t-1)} + \mathbf{W}_{\mathtt{in}}\mathbf{x}^{(t)})$$

```
dim_hidden = 6
dim_input = 1
dim_output = 1

Wrec = torch.randn(dim_hidden, dim_hidden, requires_grad=True)
Win  = torch.randn(dim_hidden, dim_input, requires_grad=True)
Wout = torch.randn(dim_hidden, dim_output, requires_grad=True)

def recurrent_unit(x, h_prev):
    # YOUR CODE HERE
    raise NotImplementedError
```

Recurrent unit



Computing the maximum of a sequence



Computing the maximum of a sequence

```
minibatch_size = 2**8
seq_length = 5
for epoch in range(num_epochs):
    loss = 0
    for _ in range(minibatch_size):
        xs = torch.tensor(np.random.randn(seq_length, 1), dtype=torch.
    float32)
        max_xs = torch.max(xs).item()
        yhat = forward(xs, Wrec, Win, Wout)
        loss += torch.abs(yhat - max_xs) # try the MSE loss!
    loss = loss / minibatch_size

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

In class exercise 2

$$\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \sigma(\mathbf{W}_{\texttt{rec}}\mathbf{h}^{(t-1)} + \mathbf{W}_{\texttt{in}}\mathbf{x}^{(t)})$$

```python
def recurrent_unit(x, h_prev): # returns h_next
    # YOUR CODE HERE
    raise NotImplementedError


def forward(seq, Wrec, Win, Wout): # returns yhat
    # YOUR CODE HERE
    # hint: use ‘for x in seq: ...‘
    raise NotImplementedError
```

Vectorize the minibatch?

```python
minibatch_size = 2**8
seq_length = 5
for epoch in range(num_epochs):
    loss = 0
    # CAN WE GET RID OF THE FOLLOWING FOR loop?
    for _ in range(minibatch_size):
        xs = torch.tensor(np.random.randn(seq_length, 1), dtype=torch.float32)
        max_xs = torch.max(xs).item()
        yhat = forward(xs, Wrec, Win, Wout)
        loss += torch.abs(yhat - max_xs)
    loss = loss / minibatch_size

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

In class exercise 3

```python
def forward(seqs,Wrec, Win, Wout):
    """
    INPUT
    seqs - a (minibatch_size, seq_length, dim_input) tensor

    RETURN
    yhat - a (minibatch_size, ) tensor
    """
    # YOUR CODE HERE
    raise NotImplementedError
```

An annoying loop

```python
def forward(seqs,Wrec, Win, Wout):
    """
    INPUT
    seqs - a (minibatch_size, seq_length, dim_input) tensor

    RETURN
    yhat - a (minibatch_size, ) tensor
    """
    h = torch.zeros(seqs.shape[0], dim_hidden)

    for x in seqs.transpose(0, 1): # CAN WE GET RID OF THIS?
        h = recurrent_unit(x, h, Wrec,Win)
    # [...]
```

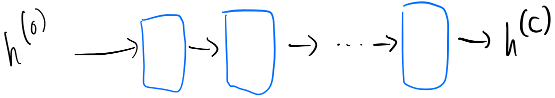
- No, at least not yet (active research area)

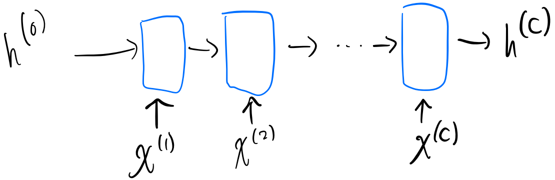
An annoying loop

```python
def forward(seqs,Wrec, Win, Wout):
    """
    INPUT
    seqs - a (minibatch_size, seq_length, dim_input) tensor

    RETURN
    yhat - a (minibatch_size, ) tensor
    """
    h = torch.zeros(seqs.shape[0], dim_hidden)

    for x in seqs.transpose(0, 1): # CAN WE GET RID OF THIS?
        h = recurrent_unit(x, h, Wrec,Win)
    # [...]
```

- No, at least not yet (active research area)
- "Parallelizing non-linear sequential models over the sequence length" (Lim et al., 2024 ICLR)

Recurrent Neural Network with multiple layers





Implementation

```python
class recurrent_cell(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        # [...]
        self.hidden_dim = hidden_dim
        self.input_to_hidden = nn.Linear(input_dim, hidden_dim)
        self.hidden_to_hidden = nn.Linear(hidden_dim, hidden_dim)

    def forward(self, x, h_prev):
        h_next = torch.relu(self.input_to_hidden(x) + self.hidden_to_hidden(h_prev))
        return h_next
```

Implementation

```python
class RNN_layer(nn.Module):
    def __init__(self, embed_dim, hidden_dim):
        # [...]
        self.rnn_layer = recurrent_cell(embed_dim, hidden_dim)

    def forward(self, x_seq, h):
        outputs = []
        for t in range(x_seq.size(1)):
            h = self.rnn_layer(x_seq[:, t, :], h)
            outputs.append(h)

        # Stack outputs
        x_transformed = torch.stack(outputs, dim=1)
        return x_transformed
```

Implementation

```
class RNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers):
        # [...]
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.layers = nn.ModuleList([RNN_layer(embed_dim, hidden_dim) for _ in range(num_layers)])
        self.classification_head = nn.Linear(embed_dim, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        h = torch.zeros(x.size(0), self.layers[0].rnn_layer.hidden_dim, device=x.device)  # Initial hidden state

        for rnn_layer in self.layers:
            x = rnn_layer(x, h)

        logits = self.classification_head(x)
        return logits
```

Seq-2-“one” Causal models

$$f(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}; \theta) = \mathbf{y}^{(t)}$$

Causality assumption

The output $\mathbf{y}^{(t)}$ does not depend on future values of $\mathbf{x}^{(t')}$ where $t' > t$

Seq-2-seq causal models

$$f(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(C)}; \theta) = \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(C)}$$

Causality assumption

The output $\mathbf{y}^{(t)}$ does not depend on future values of $\mathbf{x}^{(t')}$ where $C \geq t' > t \geq 1$

Sequences

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(C)} \end{bmatrix}$$

where

- $\mathbf{x}^{(t)} \in \mathbb{R}^d$
- $t \in \{1, \ldots, C\}$,
- we should really write $\mathbf{X}^{(i)}$ where $i \in \{1, \ldots, N\}$ but hide from notation for convenience

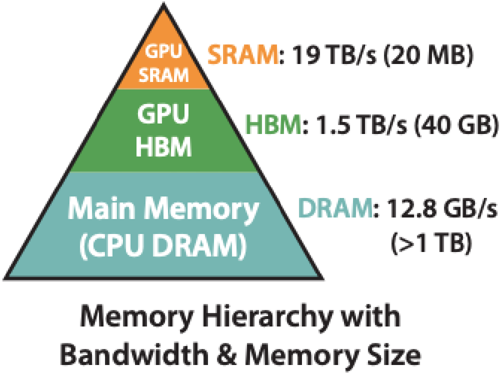
Self-attention

- $$\texttt{attention}(\mathbf{X};\theta) := \mathbf{W}^{(V)\top}\mathbf{X}^{(i)}\mathrm{softmax}\left(\mathbf{X}^\top\mathbf{W}^{(K)\top}\mathbf{W}^{(Q)}\mathbf{X}\right) \in \mathbb{R}^{d\times C}.$$
- parameters
$$\theta^{(\texttt{att})} = [\mathbf{W}^{(Q)}, \mathbf{W}^{(K)}, \mathbf{W}^{(V)}]$$
- (“Q, K, V” stands for “query”, “key”, “value”, respectively)
where
$$\mathbf{W}^{(Q)} \text{ and } \mathbf{W}^{(K)} \text{ and } \mathbf{W}^{(V)} \in \mathbb{R}^{d\times d}.$$
- “**seq-2-seq**”: maps the sequence $\mathbf{X}$ to another sequence $\texttt{attention}(\mathbf{X};\theta)$:
- **Problem**: is this causal?

Is this causal?

In class exercise 4

Flash attention



From Dao et al 2022 (Flash Attention)

Flash attention

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

From Dao et al 2022 (Flash Attention)

References I