Sequence models

Lecture 11 — CS 577 Deep Learning

Instructor: Yutong Wang

Computer Science
Illinois Institute of Technology

October 30, 2024

Sequence models

Lecture 11 — CS 577 Deep Learning

Instructor: Yutong Wang

Computer Science
Illinois Institute of Technology

October 30, 2024

- Seq model
- Flash Attention

Systems (ML + optimizing for memory bandwidth)

memory-bound

moving things from memory to compute (GPU) is the bottleneck

- Andre Bauer / time series for AI

  Minxuan Zhou Hardware topics
  Tailored hardware
  In-memory compute

- Matt Smith
  CV for ecology understanding insects biodiversity

  review for final

Topics

- Temporal convolutional networks
- Recurrent neural networks
- Transformers

Sequences

dependency across time

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

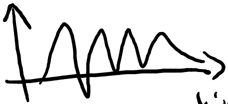
- Example: Time series (acoustic signal, weather, stock price,...)

music

audio

Sunny

Rainy



time

Sequences

$\mathbb{R}^{d_{emb}}$

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

the fox jump

- 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) clean up audio

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?)

document classif.

↳ generative
next word prediction

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?)

Sentiment analysis $0 \to 100$

Causal models

put "causal rail guard" in training loop

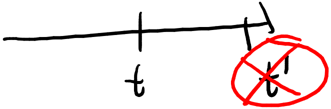
$$f(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}; \theta) = \mathbf{y}^{(t)}$$

implicitly make the model causal

Causality assumption

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

interest in current time



no dependency on the future

Prediction/forecasting/generative models

input space = output space

when is the not true ?

- Let $\hat{\mathbf{x}}^{(t+1)} := \mathbf{y}^{(t)}$
-

$$\underbrace{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(C)}}_{\text{observed}} \xrightarrow{f} \underbrace{\hat{\mathbf{x}}^{(C+1)}, \hat{\mathbf{x}}^{(C+2)}, \ldots}_{\text{predicted/generated}}$$

- Next: for simplicity, let's consider $x^{(t)} \in \mathbb{R}$. (So drop the bold fontface.)

Context = max time observed up to

Prediction/forecasting/generative models

- Let $\hat{\mathbf{x}}^{(t+1)} := \mathbf{y}^{(t)}$
-

$$\underbrace{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(C)}}_{\text{observed}} \xrightarrow{f} \underbrace{\hat{\mathbf{x}}^{(C+1)}, \hat{\mathbf{x}}^{(C+2)}, \ldots}_{\text{predicted/generated}}$$

- Next: for simplicity, let's consider $x^{(t)} \in \mathbb{R}$. (So drop the bold fontface.)

scalar

simplest non-trivial

Later: how this leads to RNNs

Autoregressive model

AR(5)

s := filter_size

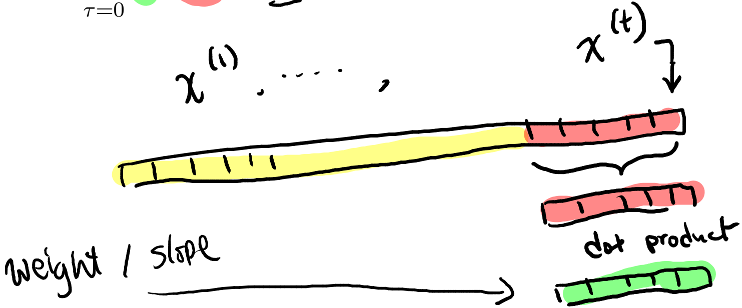
- For $t \geq s$

s = 5

$$\hat{x}^{(t+1)} = y^{(t)} = f(x^{(1)}, \ldots, x^{(t)}; w_0, \ldots, w_{s-1}, b)$$

$$= \sum_{\tau=0}^{s-1} w_\tau x^{(t-\tau)} + b$$

Goal



Autoregressive model

- For $t \geq s$

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

- Loss

MSE

$$\frac{1}{C} \sum_{t=1}^{C} (\underbrace{x^{(t+1)}}_{\text{next time data arrives}} - \underbrace{f(x^{(1)}, \dots, x^{(t)}; w_0, \dots, w_{s-1}, b)}_{\text{predict / forecast}})^2$$

Autoregressive model

- For $t \geq s$

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

- 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 $[w_0, \ldots, w_{s-1}, b]$

$\hookrightarrow$ np. array

Autoregressive model

- For $t \geq s$

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

- 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`

im2col

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

1 1 2 3 5 8

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

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.])
```

most recent

least

w

b

Another way to look at autoregressive models

to RNN (recurrent neural network)

- Drop the bias term for simplicity

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

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 $t-1$)

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

def !!

current input + weight @ hidden_state

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 $t-1$)

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

← hidden state

Instead of modeling $y(t)$

- Next: directly model the hidden state

Recurrent unit (with linear activation)

- Recurrent unit

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

Recurrent unit (with linear activation)

- Recurrent unit

d_h

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

- $\mathbf{h}^{(t)} \in \mathbb{R}^{d_{\texttt{h}}}$ (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 $\mathbf{w}_{\mathtt{out}} \in \mathbb{R}^{d_{\mathtt{h}}}$

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

final layer of your model

regression "head"

classification "head"

hidden state

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?

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

Want this

$$h^{(t)} := \begin{bmatrix} x^{(t)} \\ x^{(t-1)} \\ x^{(t-2)} \\ x^{(t-3)} \end{bmatrix}$$

$$\mathbf{w}_{\mathtt{in}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

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?

Want $h^{(t)} = \begin{bmatrix} x^{(t)} \\ x^{(t-1)} \\ x^{(t-2)} \\ x^{(t-3)} \end{bmatrix}$

Assume this is achieved up to time t-1

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

then $\mathbf{W}_{\texttt{rec}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$, $\mathbf{h}^{(t-1)} = \begin{bmatrix} x^{(t-1)} \\ x^{(t-2)} \\ x^{(t-3)} \\ \cancel{x^{(t-4)}} \end{bmatrix}$

drop this

Recovering the autoregressive model

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

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

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

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

choose

$$\mathbf{W}_{\text{rec}} = \begin{bmatrix} & 0 \\ 1 & \\ 0 \; 1 & \end{bmatrix} \qquad \mathbf{w}_{\text{in}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$W_{out} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$\underbrace{W_{out}^\top h(t)}_{\text{recovers autoreg}}$

upshot $\Rightarrow$ ensure

$$h(t) = \begin{bmatrix} x^{(t)} \\ x^{(t-1)} \\ \vdots \\ x^{(t-2)} \end{bmatrix}$$

Recovering the autoregressive model

a single recurrent unit is already expressive

- 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?

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

AR(4) can be implemented as a Rec Unit with $d_h = 4$

Recurrent unit (with linear activation)

- Recurrent unit

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

can be vectorial

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

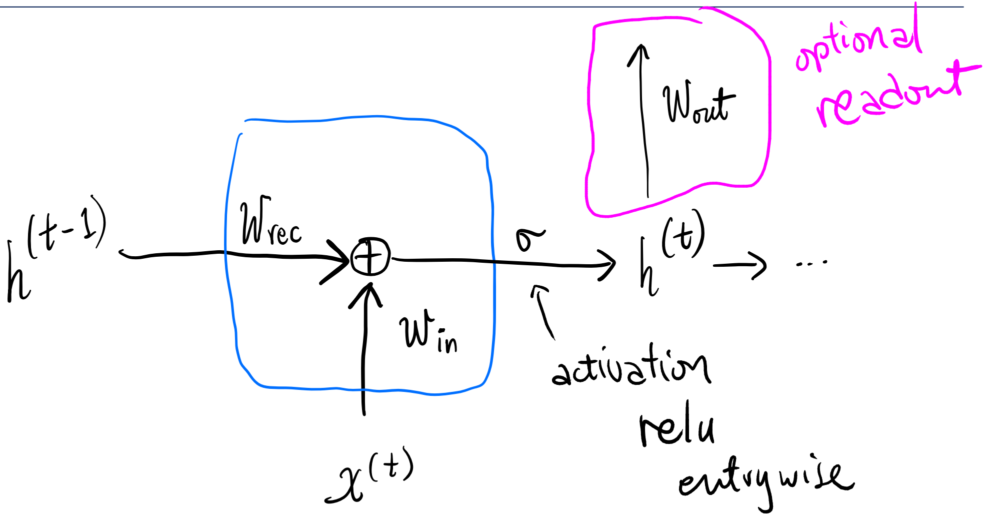
Recurrent unit (with linear activation)

- Recurrent unit

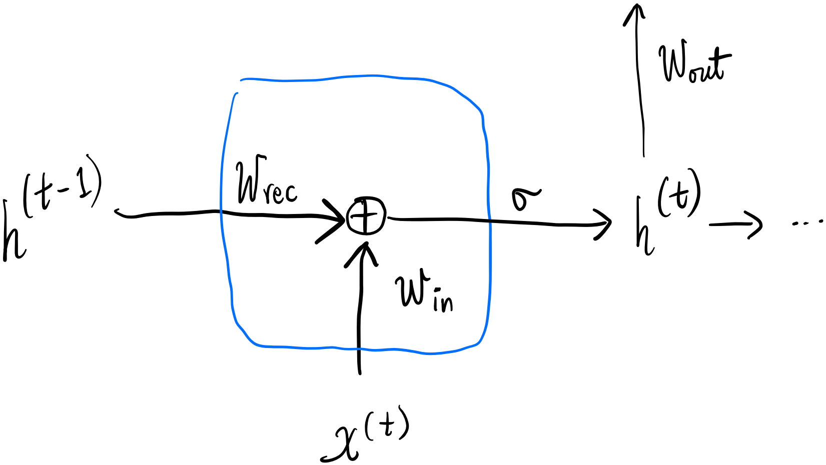
$$\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \sigma(\mathbf{W}_{\texttt{rec}}\mathbf{h}^{(t-1)} + \mathbf{W}_{\texttt{in}}\mathbf{x}^{(t)})$$

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

Recurrent unit



Recurrent unit



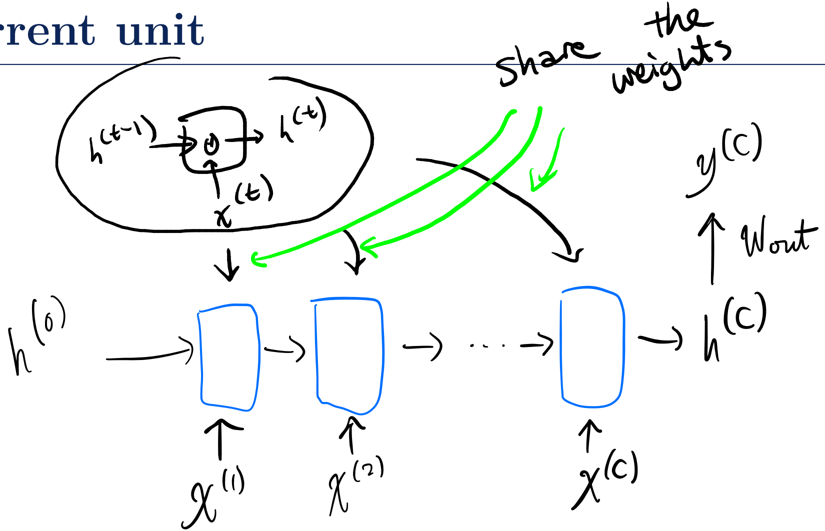
Recurrent unit (with linear activation)

- Recurrent unit

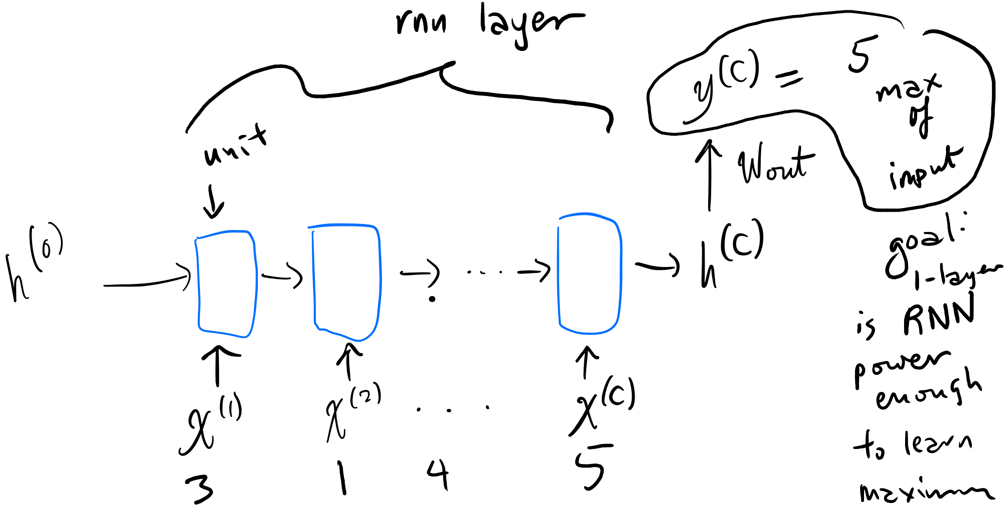
$$\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}; \boldsymbol{\theta}) = \sigma(\mathbf{W}_{\texttt{rec}}\mathbf{h}^{(t-1)} + \mathbf{W}_{\texttt{in}}\mathbf{x}^{(t)})$$

```python
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



goal: 1-layer is RNN power enough to learn maximum

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()
```

"seq" = xs = [6, 2, · · ·]

max(seq)

yhat

(yhat - max_xs) ** 2

In class exercise 2

torch.relu

given

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

```
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
```

Wrec, Win

$\boxed{\text{task 1}}$

initialize h(0)

Vectorize the minibatch?

triple for loop

```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()
```

256

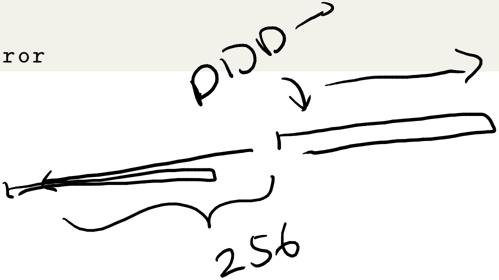
vectorized the minibatch

In class exercise 3

```
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
```

1

Possible to achieve zero error (challenge)



In class exercise 3

Expressivity

↳ universal

↳ approximate

gap

achievable via SGD adam

```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
```

there exists a set of weight (W_{rec}, W_{in}, W_{out}) s.t. we can get 0 error (forever)

Upshot: RNNs are very expressive "universal approximation"

An annoying loop

```
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)
    # [...]
```

Vectorize over time?

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

An annoying loop

"(# of param square)"

not SGD
not adam
need 2nd deriv
Newton's

```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)
    # [...]
```

Catch

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

$\sigma \neq$ identity

An annoying loop

$\frac{\partial f}{\partial \theta_i}$ first order

$i \in \{$all params$\}$

```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)
    # [...]
```

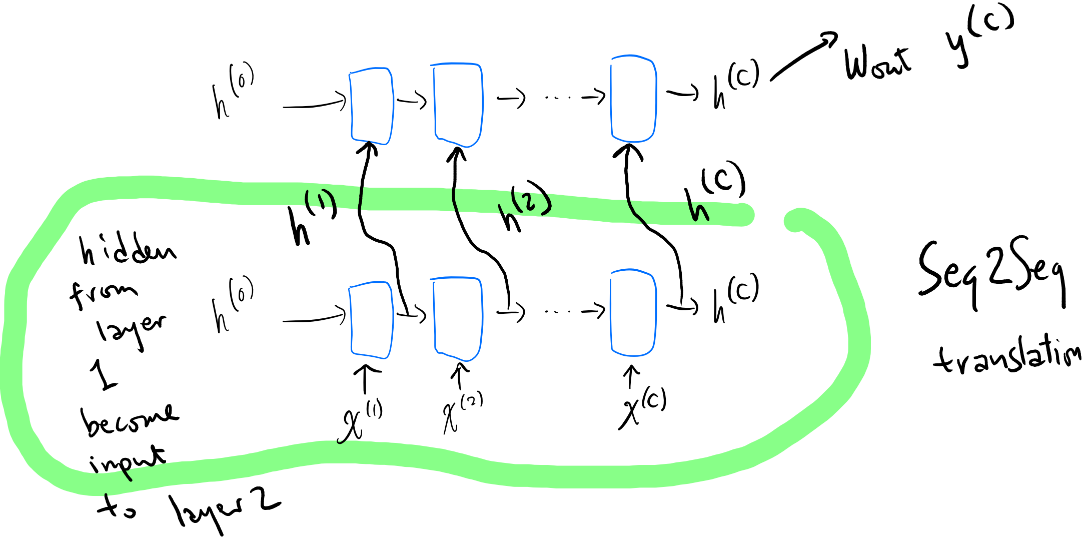
$(\#\text{param})^2 = \frac{\partial^2 f}{\partial \theta_i \partial \theta_j}$

$i,j \in \{$all params$\}$

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

↳ Neural ODE apply DL to physics

Recurrent Neural Network with multiple layers



Implementation

```
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

```python
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

$h^{(1)} \quad \ldots \quad h^{(C)}$

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

translation

Causality assumption

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

transformer → self-attention is a seq2seq model

→ multilayer

Does not suffer from non-parallelization in time / seq length

Sequences

linear algebra-ification

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(C)} \end{bmatrix} \in \mathbb{R}^{d \times C}$$

where

$X^{(i)}$

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

Self-attention

X Y Z

self X = Y = Z

can't plug in different things

- $$\text{self-attention}(\mathbf{X};\theta) := \mathbf{W}^{(V)\top}\mathbf{X}^{\cancel{(x)}}\text{softmax}\left(\underbrace{\overbrace{\mathbf{X}^{\top}\mathbf{W}^{(K)\top}}^{K}}_{(C,d)}\underbrace{\overbrace{\mathbf{W}^{(Q)}\mathbf{X}}^{Q}}_{(d,C)}\right) \in \mathbb{R}^{d\times C}.$$

  $(C,d)\ (d,C) \to (C,C)$
- 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?

$\underbrace{\text{softmax}(K^\top Q)}_{(C,C)}$ – column stochastic

col sum to 1

Self-attention

- $$\texttt{attention}(\mathbf{X};\theta) := \overbrace{\mathbf{W}^{(V)\top}}^{(d,d)} \overbrace{\mathbf{X}^{(i)}}^{(d,C)} \overbrace{\text{softmax}\left(\mathbf{X}^\top \mathbf{W}^{(K)\top} \mathbf{W}^{(Q)} \mathbf{X}\right)}^{(C,C)} \in \mathbb{R}^{d\times C}.$$

  (d, 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?

input.shape = output.shape

Is this causal? $\boxed{\exp K Q}$

start here my rec

softmax $(K^T Q)$

$C = 4$

attention$(X) =$

$$W_v \underbrace{\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \end{bmatrix}}_{X}$$

force it zero

Σ to 1

In class exercise 4

$$\mathrm{att}(X)[0,2] = W_v(p_0 x_0 + p_1 x_1 + p_2 x_2 + \cancel{p_3 x_3})$$

2nd context

p_0, p_1, p_2, p_3

$\neq 0$

Is this causal?

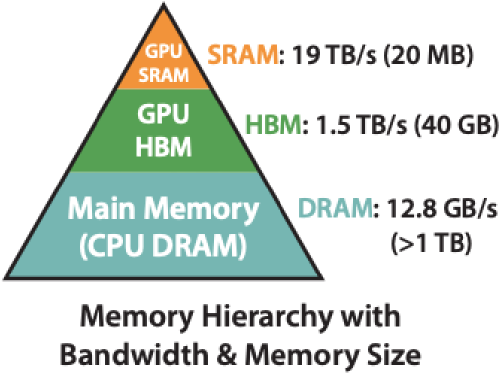
In class exercise 4

$$W_V \, X \left(\text{softmax}\,(KQ) * M \right)$$

not quite right but on right track

Causal mask

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