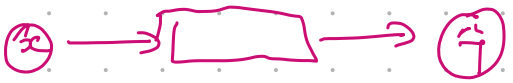
Sequences & Modeling

Applications:
1. Biological sequences (nucleic acids)
2. Video / motion
3. stock market prices
4.

→ Process a sequence of input and produce an output.

Types of Sequence Modeling Applications

① One to one (Binary) → 

eg will student pass

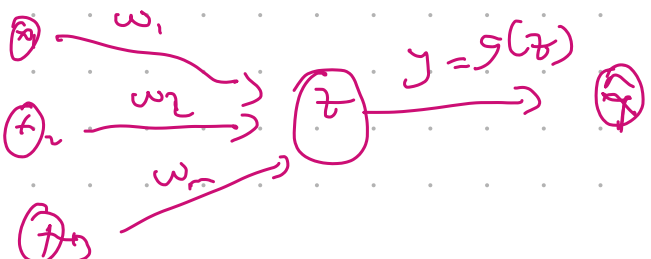
② Many to one eg Sentiment Classification } many inputs / sequences to get one output

③ One to many eg Image Captioning: eg given one input (image) generate caption (many sequence of words)

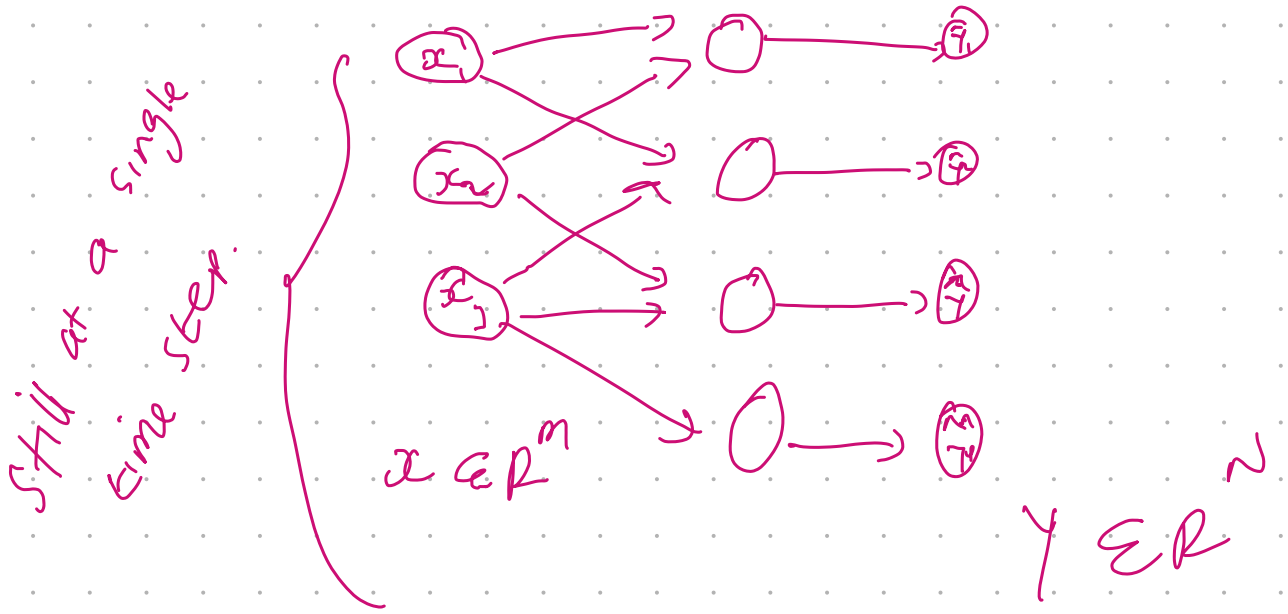
④ Many to Many eg Machine Translation: many sequences in one language to many in another.

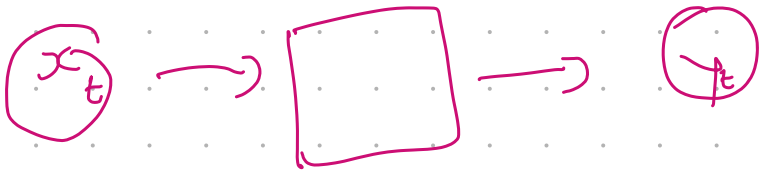
[A] Neurons with Recurrence

ⓐ Perceptron Revisited

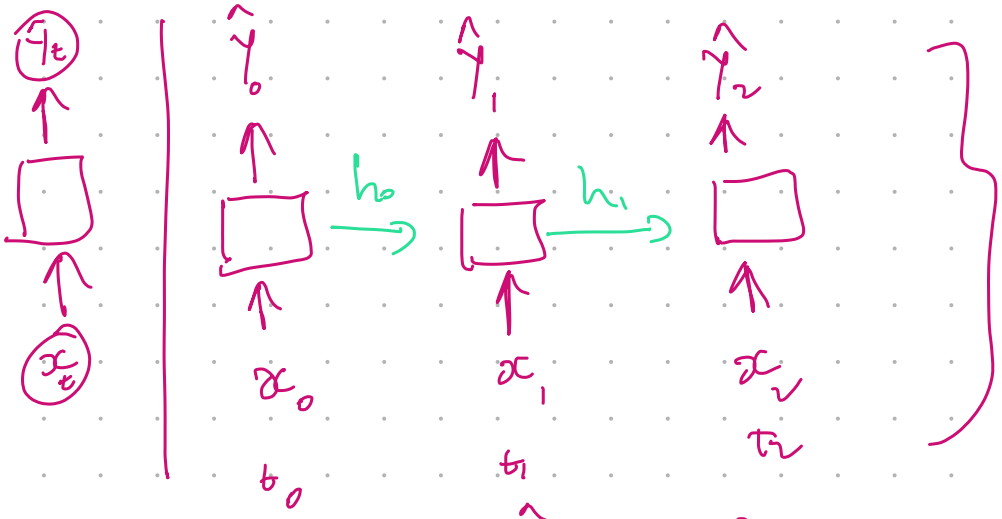


(B) Feed Forward Networks





Handle individual time steps



$$\hat{y}(t) = f(x_t)$$

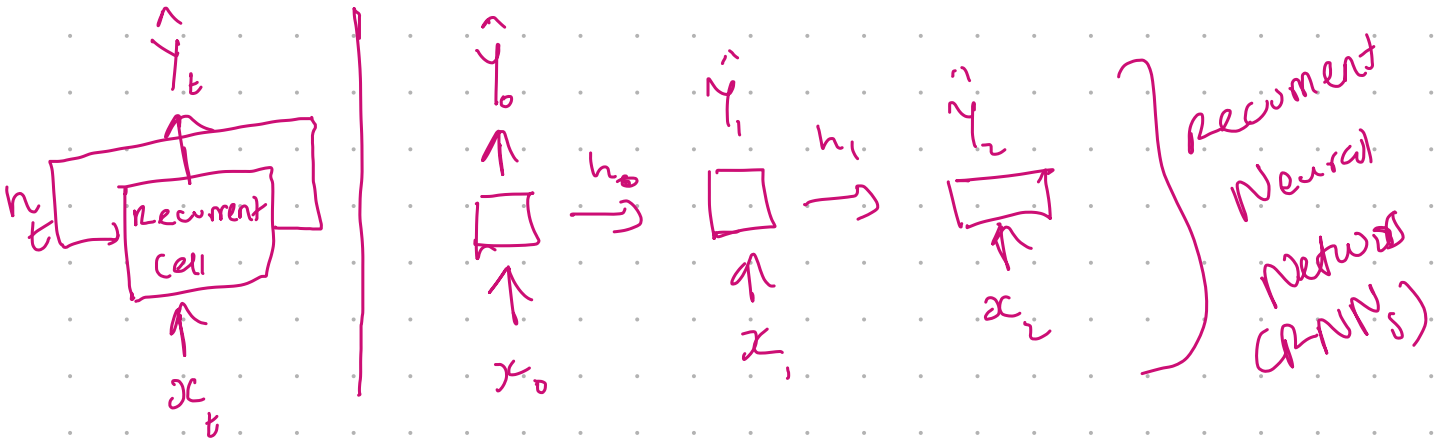
How to relate time steps to prior history

① define internal state h_{t-1} that is propagated through time.

eqn now

$$\underset{\text{output}}{\hat{y}_t} = f(\underset{\text{input}}{x_t}, \underset{\text{past memory}}{h_{t-1}})$$

© Neurons with Recurrence



Recurrent Neural Networks (RNN)

same image

Apply recurrence relation at every time step to process sequence;

$$h_t = f_W(x_t, h_{t-1})$$

- h_t: cell state
- f_W: function with weights W
- x_t: input
- h_{t-1}: prior

same set and parameters used each time

RNN State Update and Output

Input vector x_t

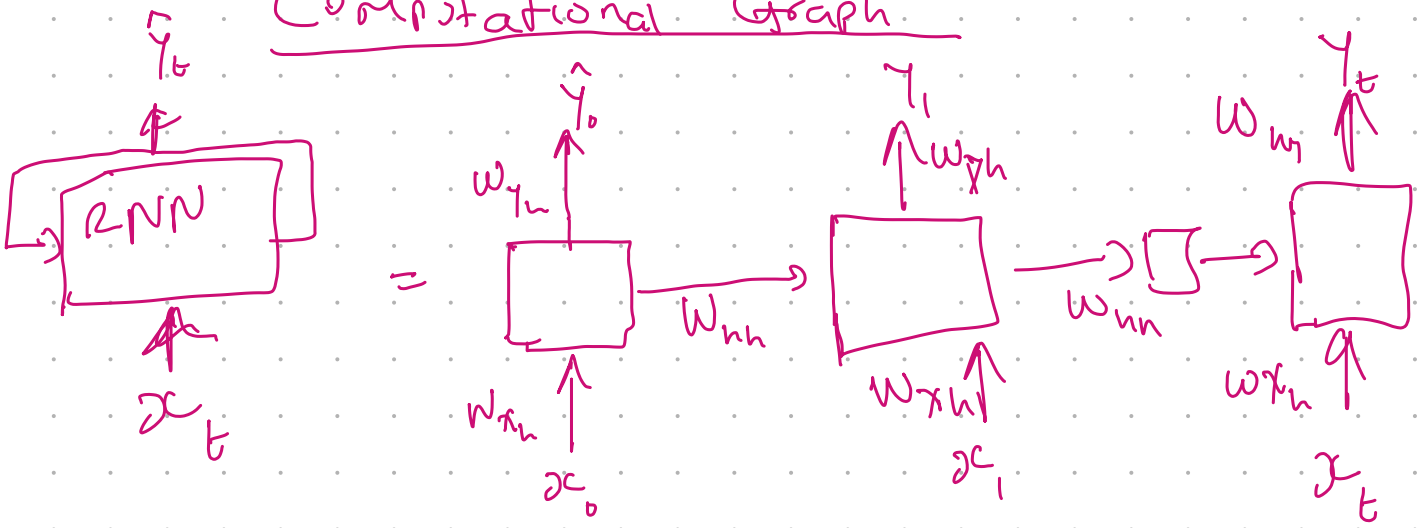
update hidden state

$$h_t = \tanh\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right)$$

Output:

$$\hat{y}_t = W_{hy}^T h_t$$

Computational Graph

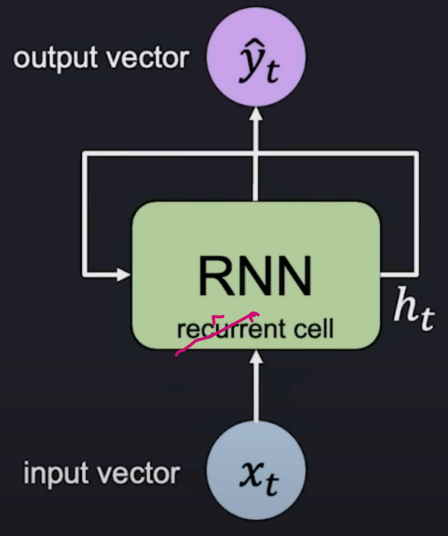


W_{xh}
W_{hh}
W_{hy}
[Reuse Same weight matrices at each step

Compute Building from Scratch.

RNNs from Scratch in TensorFlow

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



Software Lab

=> Many to Many Sequencing

Sequence Modeling : Design Criteria

Variable length → some sequences can be really short or really long.

Long term dependences → for some, something at the sequence beginning matters a lot to v later stages.

order : maintain order.

Share parameters across sequence.

Sequence Modeling Problem : Predict the next word

eg This morning I took my cat for a _____

Process

① Break sequence into words

② Vectorize text (Encoding)

Encoding Language For a Neural Network

need vocabulary] all words we could encounter.

① Indexp : convert each word to an index

Embedding

"cat" = [0, 1, 0, 1, 00]

one hot embeddings

② Handle Variable Sequence length

short : Food was good

long : Paragraph

③ Model long term dependencies :

→ need info from past for future.

Back propagation through Time

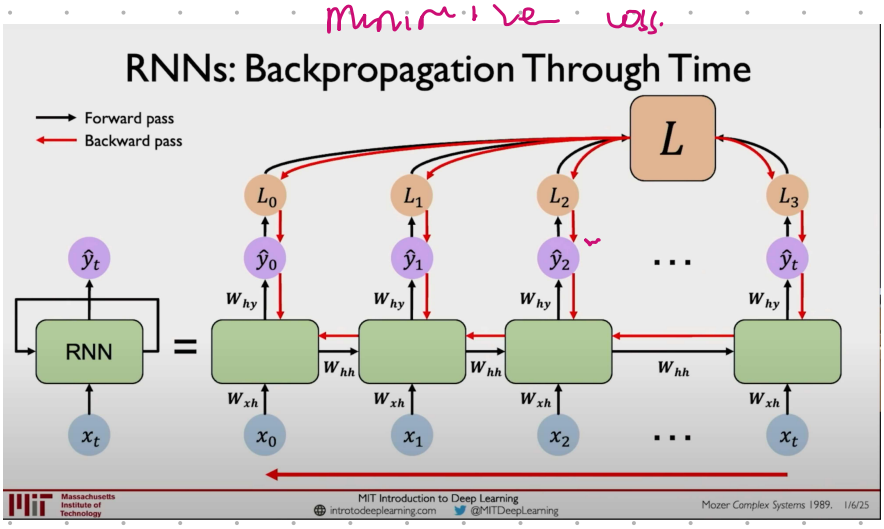
→ handling loss.

Back propagation Algorithm:

① Take the derivative (gradient) of loss with respect to each parameter.

② Shift parameters in order to minimize loss.

Errors from future time steps flow through to beginning



Challenges

computing gradient wrt h_0 involves many factors of W_{hh} + repeated gradient computation

- Many values make it harder to track long term dependencies
- Some vanish (gradients) others explode.

How to fix

① Big parameters

Gating Mechanisms in Neurons

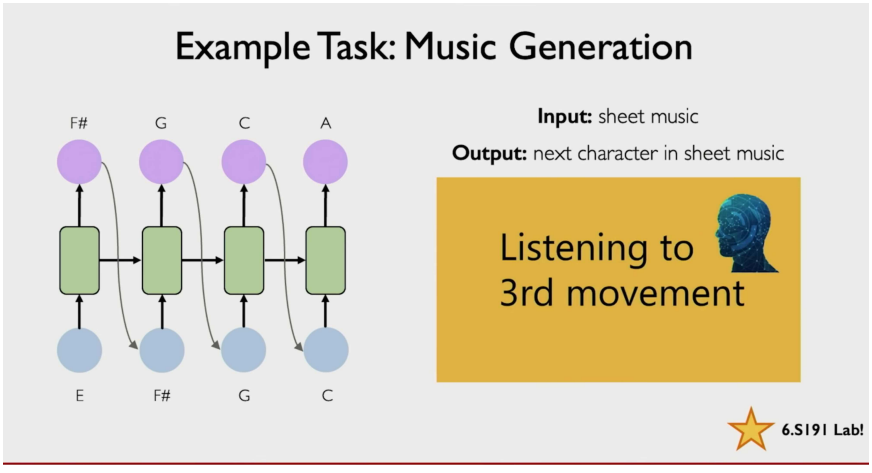
Minimize vanishing gradients

→ Use gates to selectively add or remove information within each recurrent unit with pointwise multiplication?

Examples: GRU, LSTM

LSTMs networks use gated cells to track information throughout many time steps.

RNN Applications & Limitations



Limitations:

→ Encoding bottleneck
→ State of RNN is fixed vector i.e. how much RNN can hold

→ Sequence dependencies mean no parallelism

→ Not long term memory (fixed length)

How to handle limitations

① Can we eliminate recurrence

Smash everything together → ignore time steps, clump everything together.
→ Not Scalable
→ No order or memory

② Idea: Identify and Attend to important parts (Attention is All you need)

ATTENTION IS ALL YOU NEED

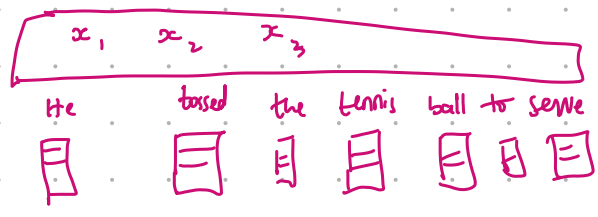
① Identify what to attend to } example: search to find matches

1. Compute attention mask (how similar to query)
2. Extract values

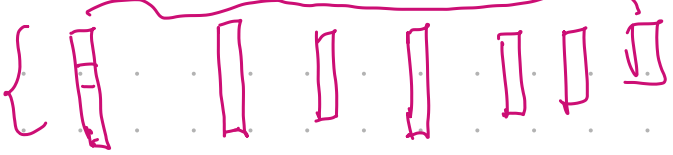
② ① Learning Self-Attention with Neural Networks

Goal: identify & attend to most important features in input

Feed data all at once



position query encoding/embedding



(I) Encode position information

3 sets of matrices

(II) Extract key, query, ~~value~~ value for search

→ query
→ key
→ value.

same positional embedding multiplied by different linear layers (neural network terms) to yield outputs.
outputs capturing different information

(III) Compute attention weighting

→ Attention score: Pairwise similarity between query & key. (dot products → cosine similarity)
use Softmax to squish matrix proximity to between 0 and 1

$$\text{Softmax}\left(\frac{Q \cdot K^T}{\text{Scaling}}\right)$$

Attention weighting

→ meaning

(IV) Extract features with high attention.

Attention weighting (from III) × Value = Output

$$\text{Softmax}\left(\frac{Q \cdot K^T}{\text{Scaling}}\right) \cdot V = A(Q, K, V)$$

unclear how Scaling works.

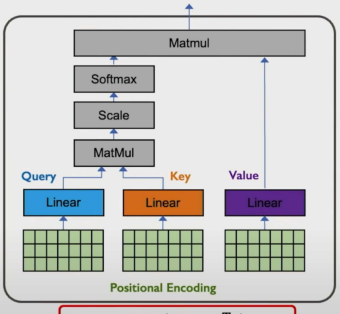
$$\left(\sqrt{\text{something}}\right)$$

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

These operations form a self-attention head that can plug into a larger network. Each head attends to a different part of input.



$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V$$

Q and K matrix
If well associated
V has high
value/weight

Additional Notes

Query : What we are looking for

Key : What we have available to match against

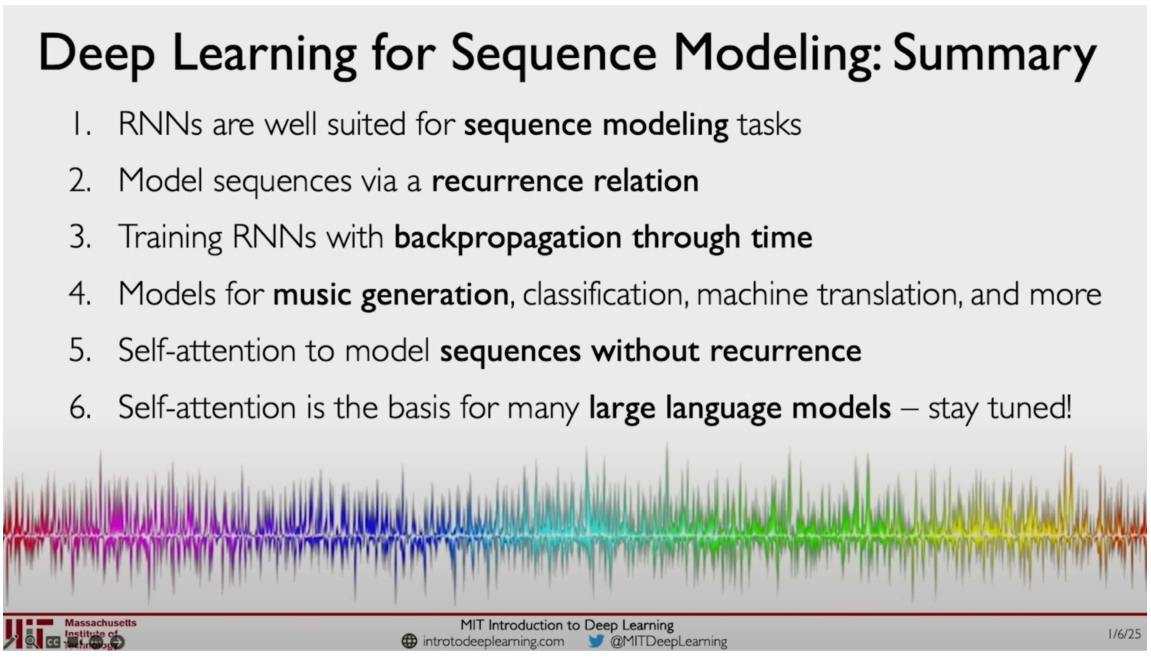
Value : Actual content info we want.

Applications of Self Attention

Language Processing (Bert, GPT etc) - Transformers

Biological Sequences: (protein structure models)

Computer Vision : Vision Transformers.



Premium

Search

1. Introduction to Sequence Modeling:
- Sequence modeling involves predicting outcomes based on sequential data.
- Real-world examples include time series data, audio signals, text sequences, and more.

2. Fundamentals of Neural Networks:
- Basics of neural networks, including feedforward models and backpropagation.
- Importance of understanding neural networks for building sequence models.

3. Recurrent Neural Networks (RNNs):
- RNNs are designed to handle sequential data by maintaining an internal state to capture information from previous time steps.
- Key concepts include:
- Internal State (h): Represents memory of previous inputs.
- Recurrence Relation: Defines how the internal state is updated over time steps.
- Forward Pass: Predicts outputs for each time step using the current input and previous state.

4. Training RNNs:
- Loss function: Measures how well predictions match actual outcomes.
- Backpropagation Through Time (BPTT): Adapts the backpropagation algorithm for RNNs to handle sequential data across multiple time steps.

5. Limitations of RNNs:
- Difficulty in training due to issues like vanishing and exploding gradients.
- Fixed-length internal states can limit the amount of information captured.

6. Long Short-Term Memory (LSTM) Networks:
- An advanced type of RNN designed to overcome the limitations of standard RNNs.
- Uses gates to control the flow of information and maintain long-term memory.

7. Attention Mechanism:
- Introduced as a way to improve upon RNNs by allowing models to selectively focus on important parts of the input.
- Core components include:
- Query, Key, Value: Matrices used to compute attention scores.
- Self-Attention: A mechanism that relates different parts of the sequence to one another, allowing for parallel processing.

8. Transformers:
- A neural network architecture that relies on the attention mechanism and eliminates the need for recurrence.
- Capable of processing entire sequences simultaneously, making it more efficient for long sequences.
- Applications in natural language processing, image processing (Vision Transformers), and more.