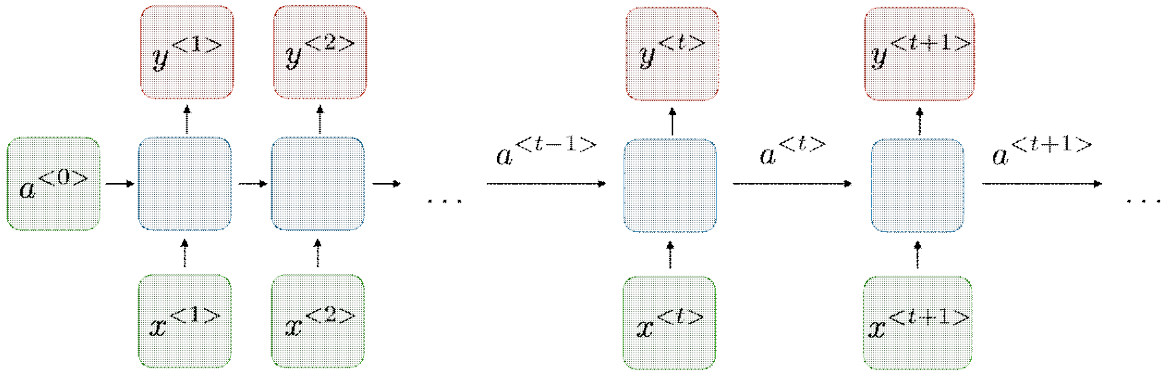
RNN

2024年9月20日 21:16

Introduction to RNN:

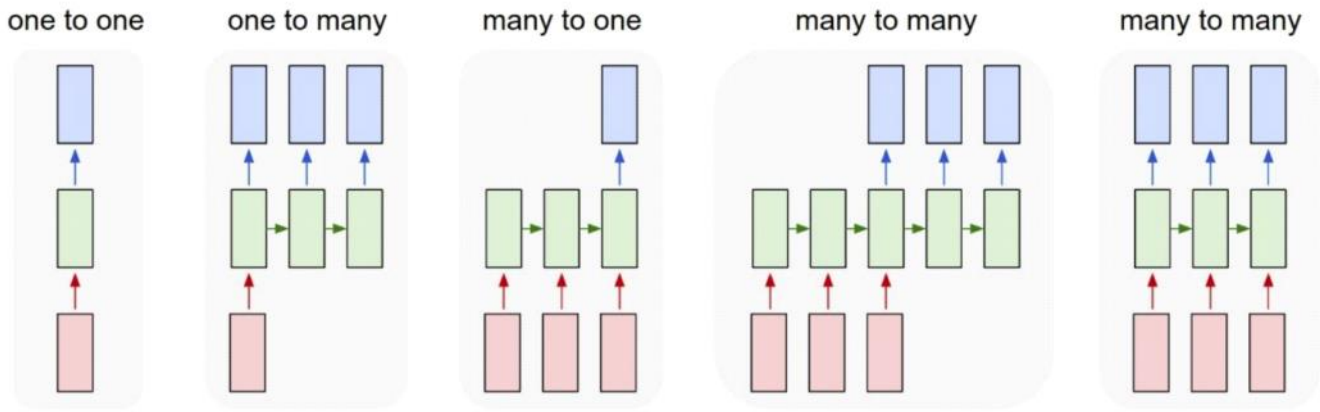
RNN (Recurrent Neural Networks):

- A type of Neural Network designed to handle sequential data, like time series or sentences
- RNNs have loops that allow information to be passed from one step of the sequence to the next.
  - Suitable for tasks where order of data matters
  - Eg: Text generation, Music generation, Speech recognition
- CNN can only take in inputs with a fixed size of width and height and cannot generalize over inputs with different sizes.



Different types of RNN architectures: [More details]

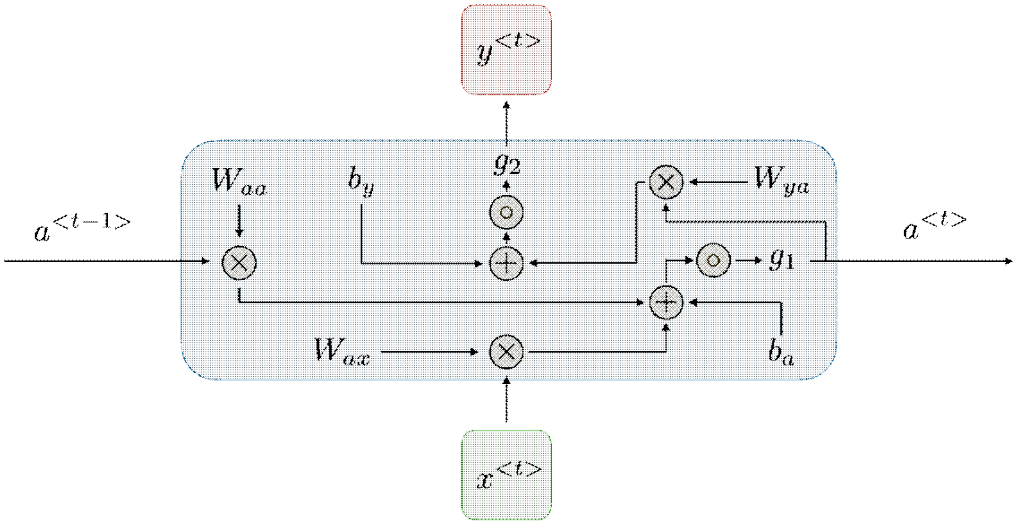
- **One to one**
  - Like a simple Neural Network, also known as Vanilla Neural Network
- **One to many**
  - Eg: Image captioning, input a fixed sized image and produce a sequence of words that describe the content of that image
- **Many to one**
  - Eg: Action prediction, input a sequence of video frames instead of a single image and produce a label of what action was happening in the video
  - Eg2: Sentiment classification, input a sequence of words of a sentence, classify what sentiment (e.g. positive or negative) that sentence is
- **Many to many (2nd last)**
  - Eg: Video-captioning, input a sequence of video frames and the output is caption that describes what was in the video
  - Eg2: Machine translation, input a sequence of words of a sentence in English, and produce a sequence of words of a sentence in French
- **Many to many (last)**
  - Eg: The model generates an output at every timestep (can be name entity recognition)



Recurrent Neural Network

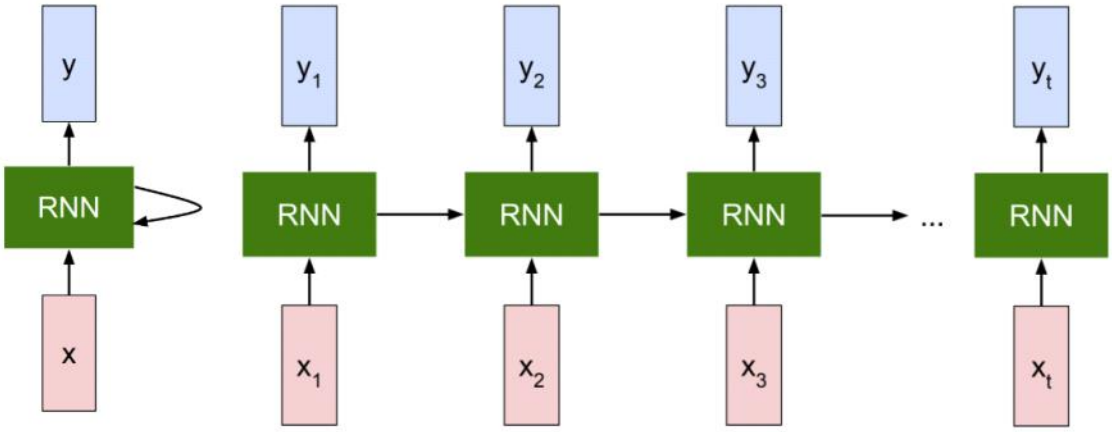
RNN as a blackbox:

- **Internal State:** Think of an RNN as having a memory, this internal state gets updated as it processes each part of a sequence.
- **Input and State Update:** At each step, you feed an input (like a video frame) into the RNN. The RNN updates its internal state based on this input.
- **Output Generation:** The RNN can also produce an output at each step, which is based on its current state.



Unrolling an RNN:

- **Inputs at Different Timesteps:** Imagine you have a sequence of inputs (like video frames) labeled as $(x_1, x_2, x_3 \dots x_t)$
- **Two Inputs at Each Timestep:**
  - Current Input: The input at the current timestep x_i
  - Previous State: The RNN's memory of what it has seen so far (its history)
- **Output and State Update:**
  - Output: The RNN generates an output y_i, based on the current input and its previous state
  - State Update: The RNN updates its internal state, which will be used in the next timestep.



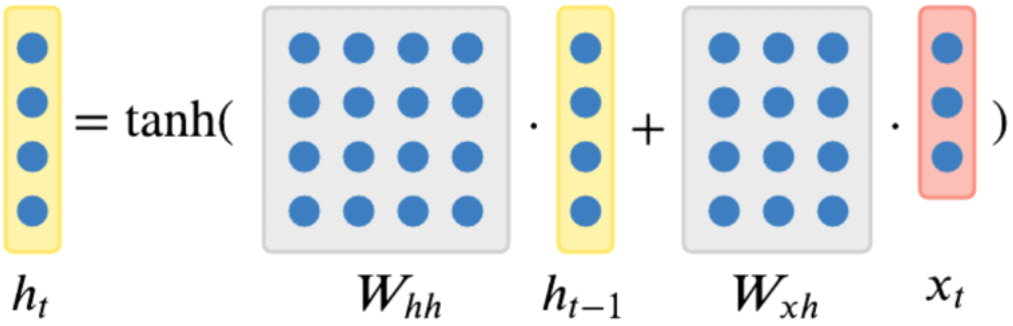
Recurrence Formula

RNN Recurrence:

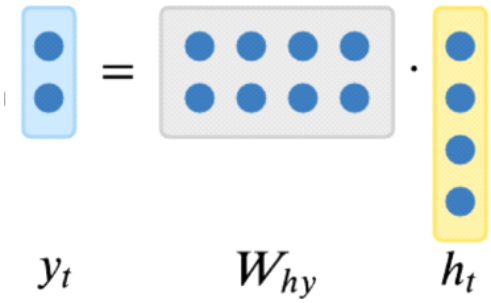
- **Internal State Update:**
  - Formula: $h_t = f_w(h_{t-1}, x_t)$
  - Steps:
    - At each timestep t, it takes previous state h_{t-1} and current input x_t to produce the new state h_t
    - Function f_w with parameters W determines how this update happens
- **Fixed Function:**
  - The same function f_w is applied every timestep, regardless of input length (it can handle any length, by repeatly apply same update rule)

Vanilla RNN:

- **Hidden State Update:**
  - Formula: $\tanh(W_{hh}h_{t-1} + W_{xh}x_t)$
  - Where:
    - W_{hh}: Weight matrix for the previous hidden state h_{t-1}
    - W_{xh}: Weight matrix for the current input x_t
    - tanh: squishes the summed values to keep them within a certain range, ensuring stability



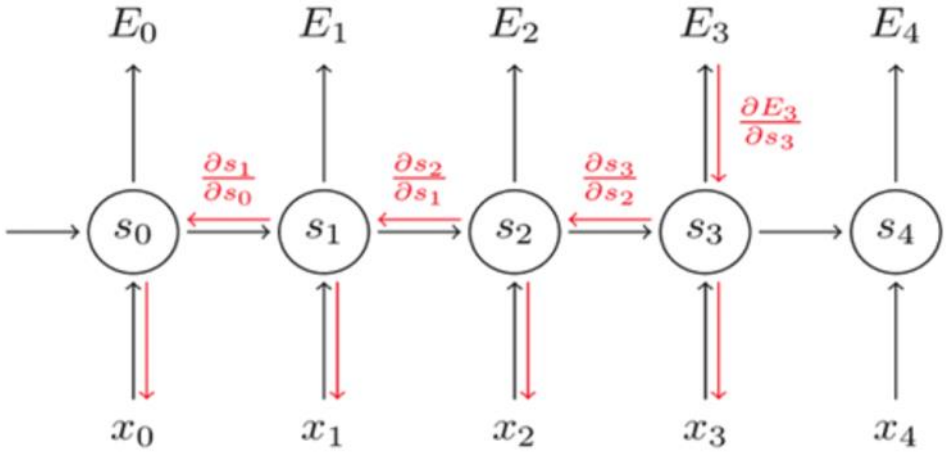
- **Prediction Based on Hidden State:**
  - Formula: $y_t = W_{hy}h_t$
  - The output y_t is generated by applying another weight matrix W_{hy} to the current hidden state h_t



Backpropagation Through Time (BPTT)

Backpropagation is done at each point in time. At timestep T, the derivative of the loss $\mathcal{L}$ with respect to weight matrix W is expressed as follows:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial W}\Big|_{(t)}$$



Vanilla RNN Gradient Flow & Vanishing/Exploding Gradient

Gradient Flow:

- **RNN Update Rule:**
  - Formula: h_t = tanh$(W_{hh}h_{t-1} + W_{xh}x_t)$
- **Backpropagation:**
  - Use backpropagation (BP) to train RNN
  - BP to update weights (W_{hh}, W_{xh}), by calculating the derivatives (eg: to update W_{hh}, calc derivative of L_t with respect to W_{hh})

Vanishing Gradient Problem:

- **Gradient Calculation:**
  - Formula:
    $\frac{\partial h_t}{\partial h_{t-1}}$ = tanh′$\left(W_{hh}h_{t-1} + W_{xh}x_t\right)W_{hh}$
  - The gradient of the hidden state h_t with respect to the previous hidden state h_{t-1} involves the derivative of the tanh function and the weight matrix W_{hh}
- **Vanishing Gradient**
  - The derivative of the tanh function is always less than 1.

- As we multiply these small values over many timesteps, the gradient becomes very small (close to zero).
- This means that the influence of earlier timesteps diminishes, making it hard for the RNN to learn long-term dependencies.

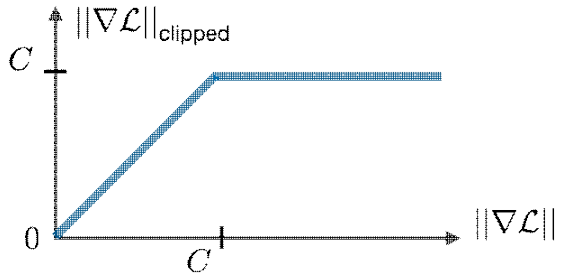
Exploding Gradient Problem:

- **Removing non-linearity:**
  - If we remove the tanh function to avoid the vanishing gradient problem, we get:

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial h_t}\left(\prod_{t=2}^{T} W_{hh}^{T-1}\right)\frac{\partial h_1}{\partial W}$$

  - Exploding Gradient: If the largest singular value of W_{hh} is greater than 1, the gradients can become very large, leading to instability (exploding gradients).

- **Gradient Clipping:**
  - We can clip the gradients to a maximum threshold, preventing them from becoming too large



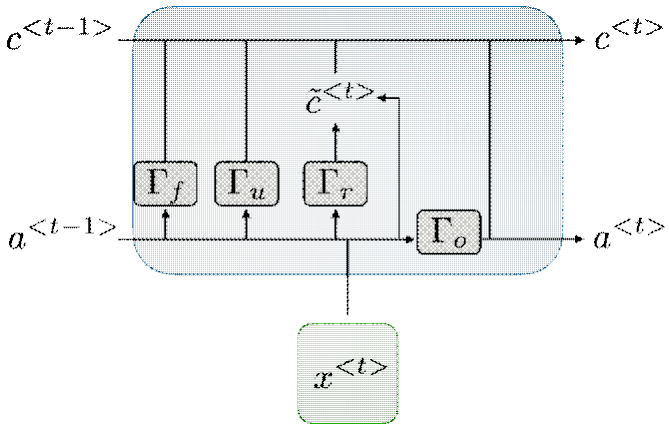
LSTM Formulation

LSTM (Long-Short Term Memory) is a type of RNN designed to handle long-term dependencies more effectively.

How it works:

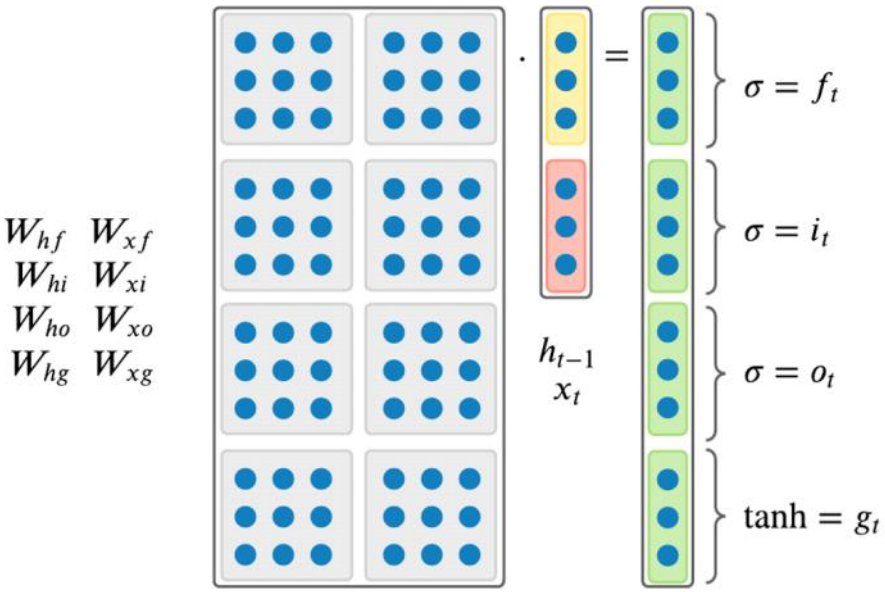
- **Hidden State and Cell State:**
  - Hidden State h_t: Similar to the hidden state in a Vanilla RNN
  - Cell State c_t: An additional state to store long-term info (A memory that can carry information across many timesteps)

- **Gates:**
  - **Input Gate $\boldsymbol{i_t}$**: Controls how much of the new information from the current input and previous hidden state should be added to the cell state
  - **Forget Gate $\boldsymbol{f_t}$**: Decides how much of the previous cell state should be kept or forgotten.
  - **Output Gate $\boldsymbol{o_t}$**: Determines how much of the cell state should be used to compute the hidden state and the output.

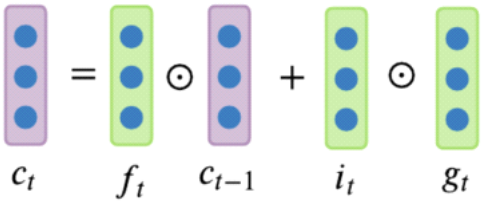


LSTM Equations:

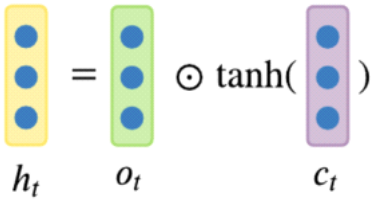
- **Calculate Gates:**
  - Forget Gate: $f_t = \sigma(W_{hf}h_{t-1} + W_{xf}x_t)$
  - Input Gate: $i_t = \sigma(W_{hi}h_{t-1} + W_{xi}x_t)$
  - Output Gate: $o_t = \sigma(W_{ho}h_{t-1} + W_{xo}x_t)$
  - Candidate Cell State: $g_t = \tanh(W_{hg}h_{t-1} + W_{xg}x_t)$



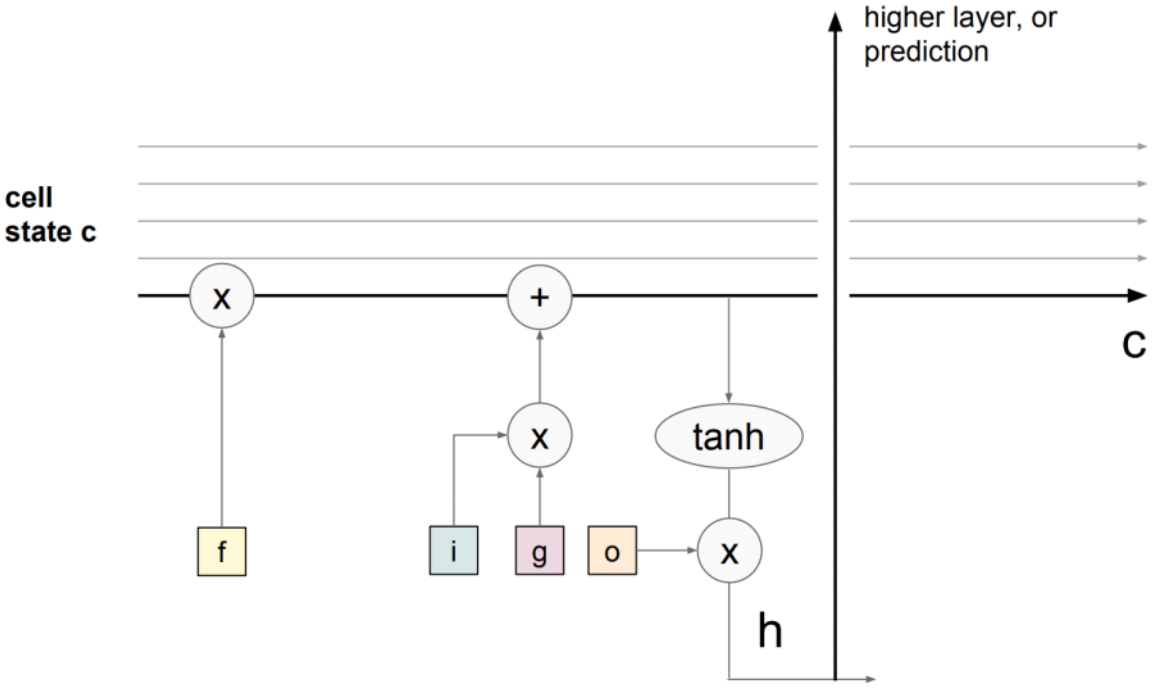
- **Update Cell State:**
  $c_t = f_t \odot c_{t-1} + i \odot g_t$



- **Update Hidden State:**
  $h_t = o_t \odot \tanh(c_t)$



LSTM cell state highway:
The cell state acts like a highway, allowing information to flow through many timesteps with minimal changes. This helps in maintaining long-term dependencies.



Does LSTM solve the vanishing gradient problem?
LSTMs do not guarantee that there is no vanishing/exploding gradient problems, but it does provide an easier way for the model to learn long-distance dependencies.

LSTM Key Concepts:

1. Maintain a separate cell state from what is outputted
2. Use gates to control the flow of information
   - Forget gate gets rid of irrelevant information
   - Selectively update cell state
   - Output gate returns a filtered version of the cell state
3. Backpropagation from C_t to C_{t-1} doesn't require matrix multiplication: uninterrupted gradient flow

Case Studies:

Classic RNN:

- RNN (Rumelhart et al., 1986a)

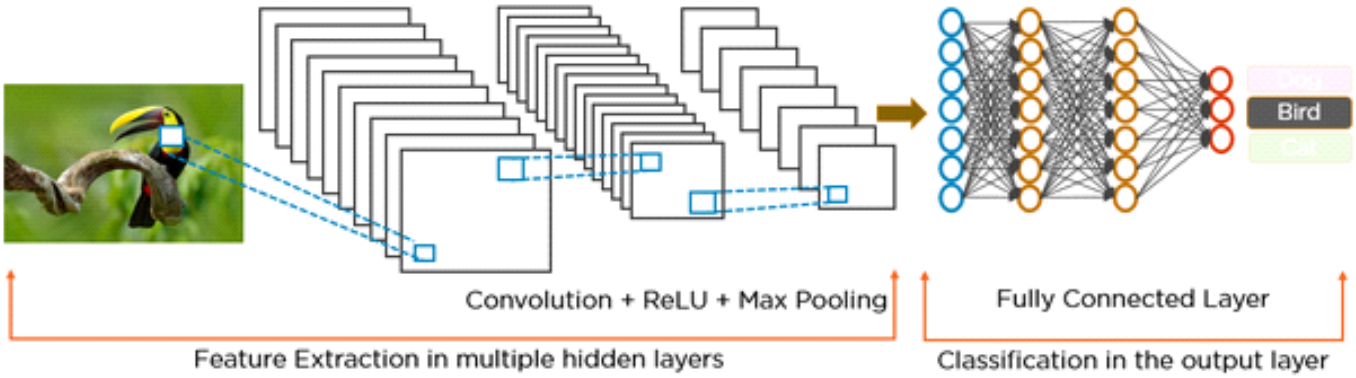
Modern RNN:

- LSTM (Long Short Term Memory)
- BRNN (Bidirectional RNN)
- DRNN (Deep RNN)

CNN vs RNN:

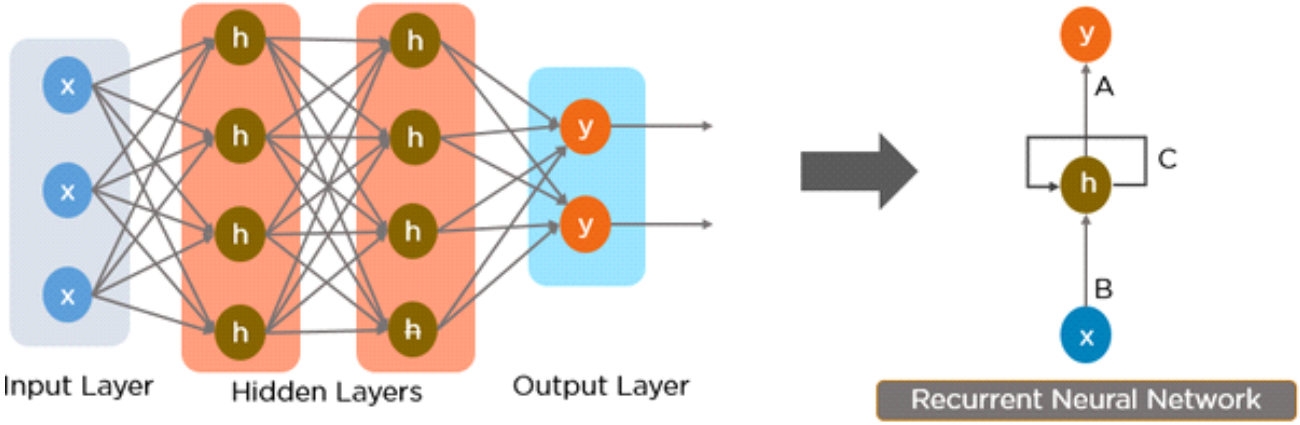
CNN:

- Looking for the same patterns on all the different subfields of the image
- CNN is looking for the same patterns over different regions of space.
- Ideal for images and videos processing.



RNN:

- In the simplest case, feeding the hidden layers from the previous step as an additional input into the next step
- RNN builds up memory in this process, it is not looking for the same patterns over different slices of time
- Ideal for text and speech analysis



More detailed comparisons:

	CNN	RNN
Data Type and Structure	Good for understanding pictures because they have a grid-like structure	Perfect for understanding things that happen one after the other
Memory and Context	Not very good at remembering things from a long time ago and has trouble understanding long sequences	Good at remembering things from earlier and understanding how they relate to what's happening now (especially in order)
Parallelization	Can do lots of things at once (doesn't need to wait for other parts to finish)	Can't do as many things at the same time (has to wait for one thing to finish before starting the next)
Training Dynamics	Needs lots of labeled pictures to learn well, especially for things like telling what's in a picture.	Can be hard to teach because sometimes it forgets important stuff, especially when a lot is happening in a long story or list.
Interpretability	Makes maps that help understand what's important in pictures.	It can be hard to understand sometimes, especially when it is doing complicated things, but there are ways to make it easier to see what's going on, especially when it's working with

		lists or stories.

References

RNN Links:

RNN Stanford CS231n Lecture Note	CS231n RNN
RNN Stanford CS230 Lecture Note	CS230 RNN
Geeks for geeks RNN	RNN Geeksforgeeks