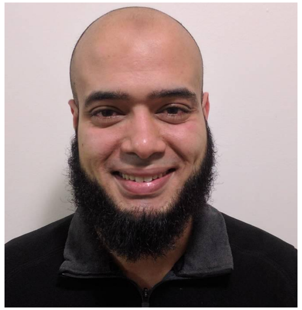
Machine Learning
More on RNN

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

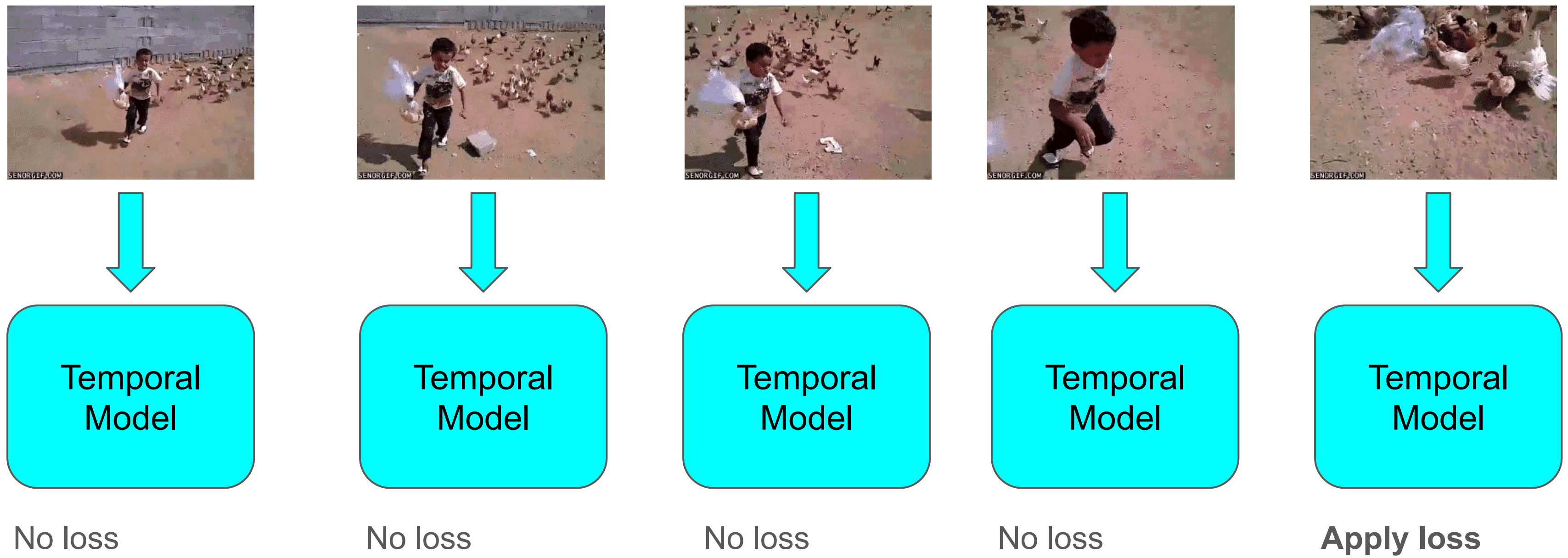


RNN Interesting Properties

- Input: As you see, RNN process a series of vectors
    - The length of the input sequence can vary
- Output: We get series of outputs one per a sequence elements
    - For example for input string “xyz” we will get 3 output vectors
    - **Many to Many processing**
        - This is where you feed the whole input sequence and care about every output
            - Example, generation like next-char generation
    - **Many to one processing**
        - This is where you want a judgment over the whole sequence
            - Example, video classification
            - We simply classify the last output step only, which includes all states
            - *Tip: we can model as many to many if we have interest (e.g. labeling all frames)*

Many to One Processing

- out, hidden = self.rnn(x)
- return self.fc(out[:, -1, :]) apply loss only on the last output
    - All batches, last sequence element, all features of this sequence



One to Many Processing

- Image captioning is a one-to-many sequence modeling task.
  - CNN extracts the image features (flatten the vector)
    - I think we use this one as the initial hidden state?!
  - Use a symbol to start RNN and another to end it (e.g. $ and #)

Img src

Many to Many Processing

- There are 2 cases
- 1) Every input has a corresponding output
    - This is a synced sequence input and output
    - Example: next word prediction
- 2) After the whole input sequence is processed, the output sequence starts
    - Example: text translation
    - First, the model absorb the whole input statement (e.g. in English)
    - Second, the model starts to generate the output sequence (e.g. in Arabic)
        - Stops at EOF symbol

RNN Use Cases

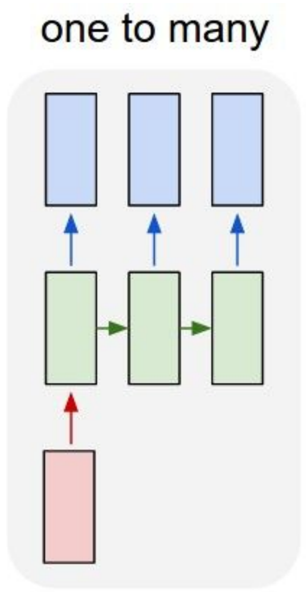
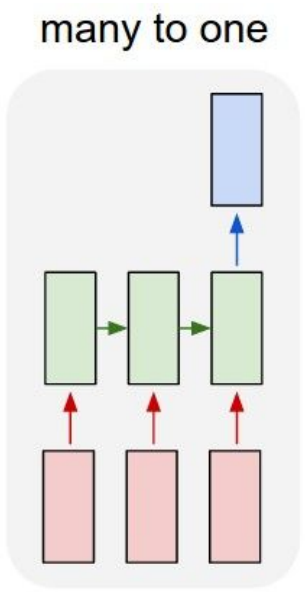
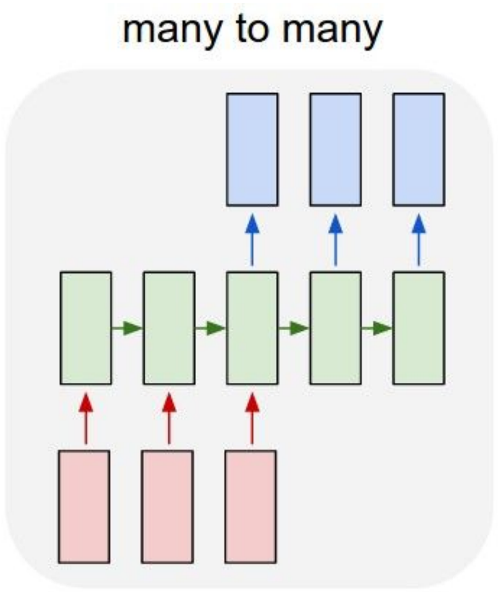


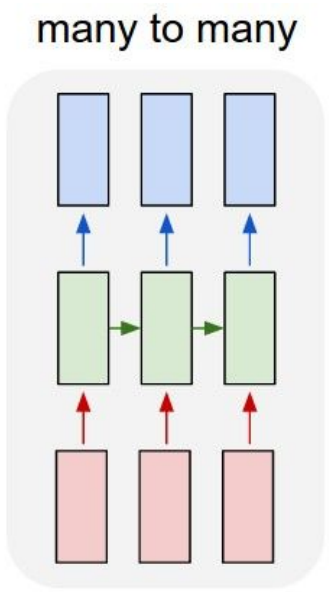
Image Caption



Sentiment Classification



Text Translation



Text Generation

Img src

Backpropagation in RNN

- **Unfolding** in Time
  - An RNN can be thought of as a deep neural network when unfolded over time.
  - Each time step of the RNN is like a layer in a deep network,
    sharing the same weights and biases across all steps.
- The Backpropagation is done on this unfolded network, similar to the normal Backpropagation (It's called Backpropagation **Through Time** (BPTT))
  - Propagate this gradient backward through the network, one time step at a time.
  - At each time step, compute the gradients with respect to the weights and biases, taking into account both the gradient propagated from the future time step and the gradient with respect to the output at the current time step.
    - In other words, keep summing the gradients from each iteration
  - Improved version: Truncated BPTT

Issues with RNN

- Vanishing Gradients: one of the most significant problems in training RNNs
  - Think in a 100 steps sequence creating a 100 normal hidden layers
  - In both feedforward and backpropagation, such long sequence will cause problems
- From feedforward perspective
  - The naive accumulations of all steps like you want to memorize every single information
    - But intuitively, many information are not that useful (e.g. duplicate clost frames)!
- As a result, we use it with shorter sequences and hard to capture long dependencies
  - Minor concerns: requires more time and memory
- LSTMs is one major way to tackle these limitations!

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”