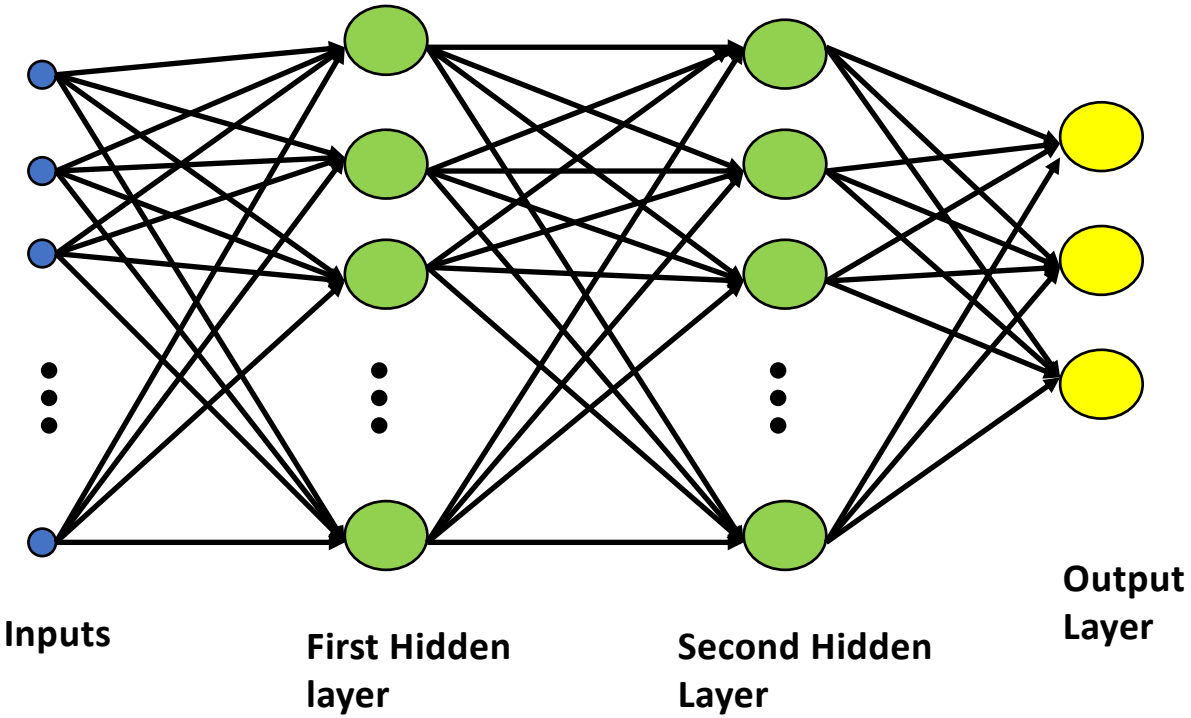
Recurrent Neural Networks (RNN)

John Tan Chong Min

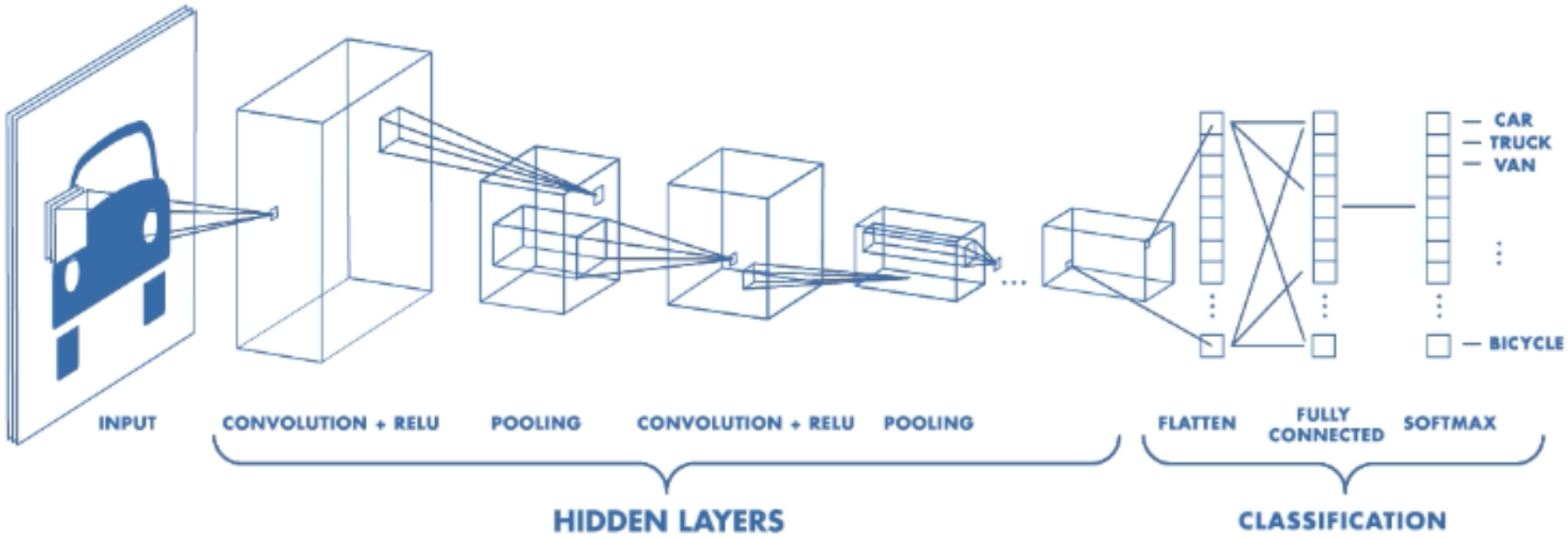
What to focus on

- Focus more on the concepts, less on the math
- No need to worry about the math equations
- Ask questions anytime: All questions are good. Questions help with understanding
- Every week: Theory first, then coding practical

Recap: Multilayer Perceptron

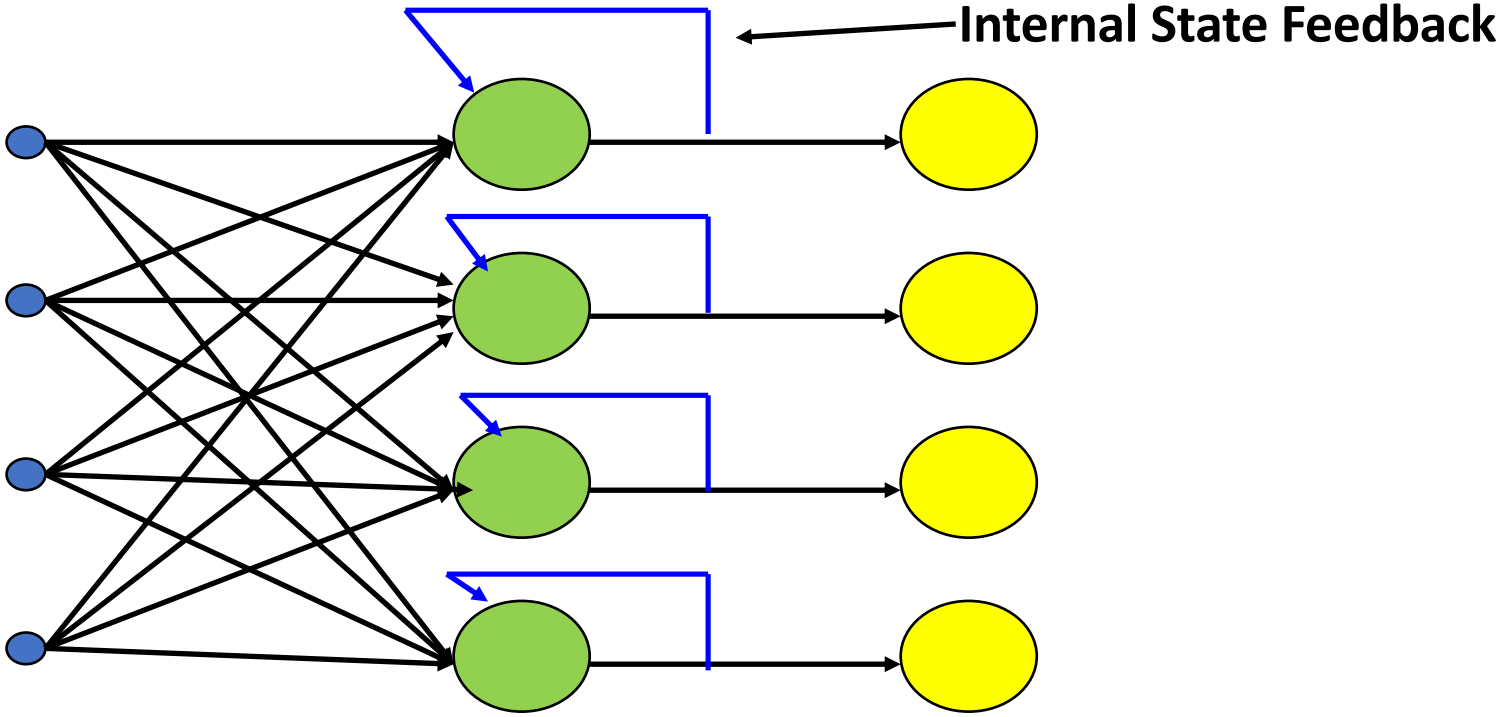


Recap: Convolutional Neural Networks



Invariance in filter weights

This Week: Recurrent Neural Network



Invariance in weights of node in hidden layer through time

Next Week: Transformers

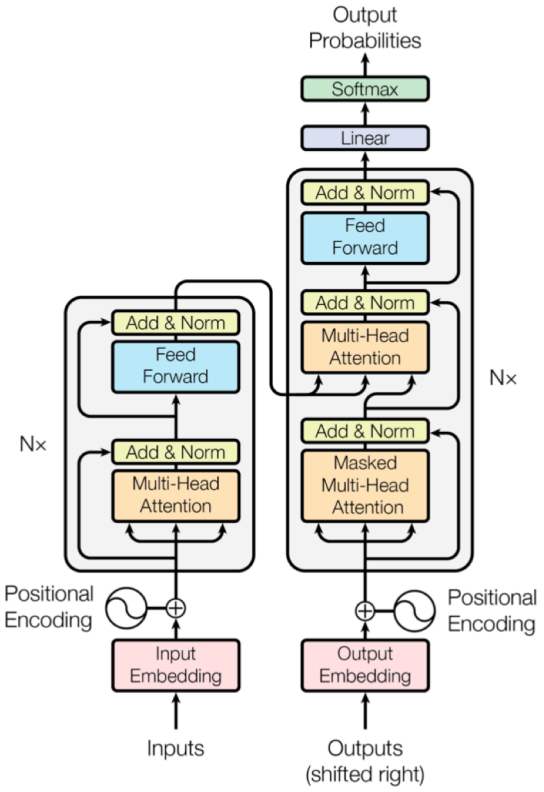
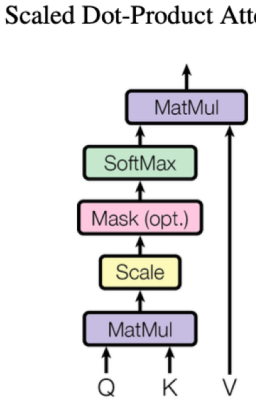


Figure 1: The Transformer - model architecture.



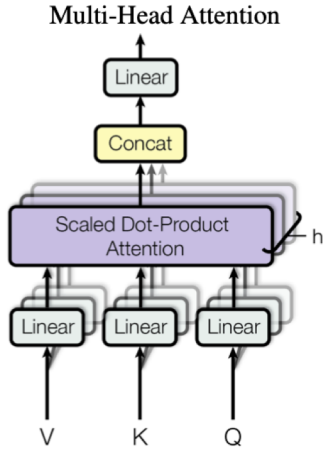


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

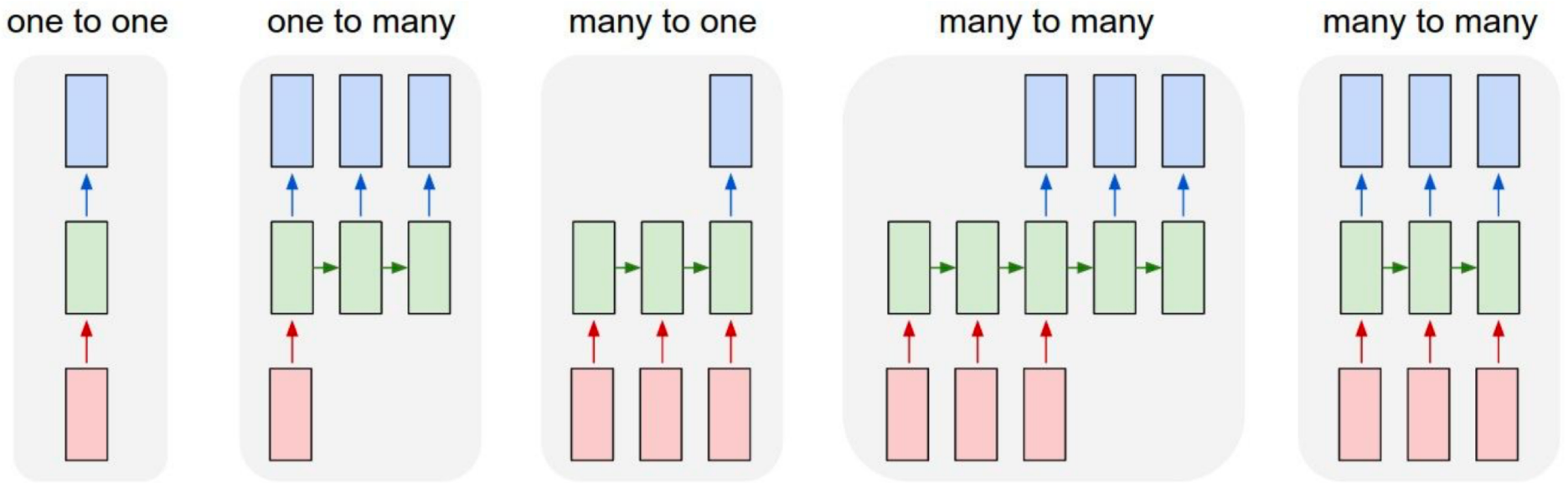
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Taken from: Attention is all you need (Vaswani et al, 2017)

When to use RNNs

- Sequential data
- Output to model is only dependent on earlier data and not later
- The way to process the data is the same through the sequence

Types of RNNs



Normal Neural Network

Image Captioning

Sentiment Classification

Language Translation

Video Classification (frame level)

Taken from: Stanford CS231n 2017 Lect 10 (Fei Fei Lin, Justin Johnson, Serena Yeung)

Basic RNN Equation

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

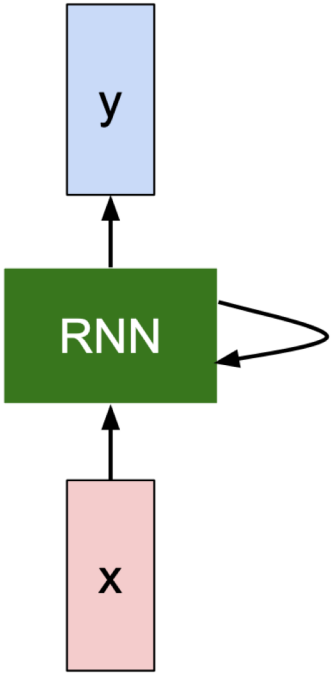
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

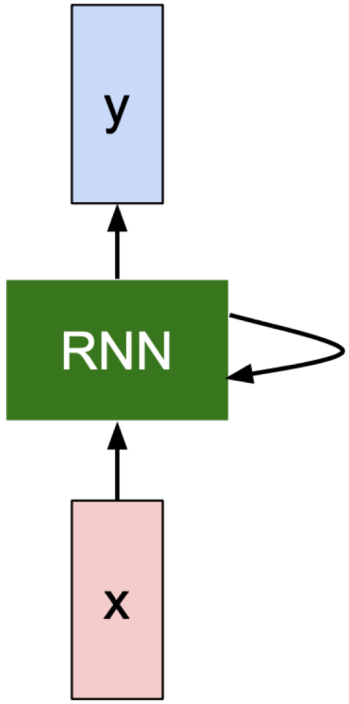
input vector at some time step

some function with parameters W



Taken from: Stanford CS321n

Possible RNN Equation



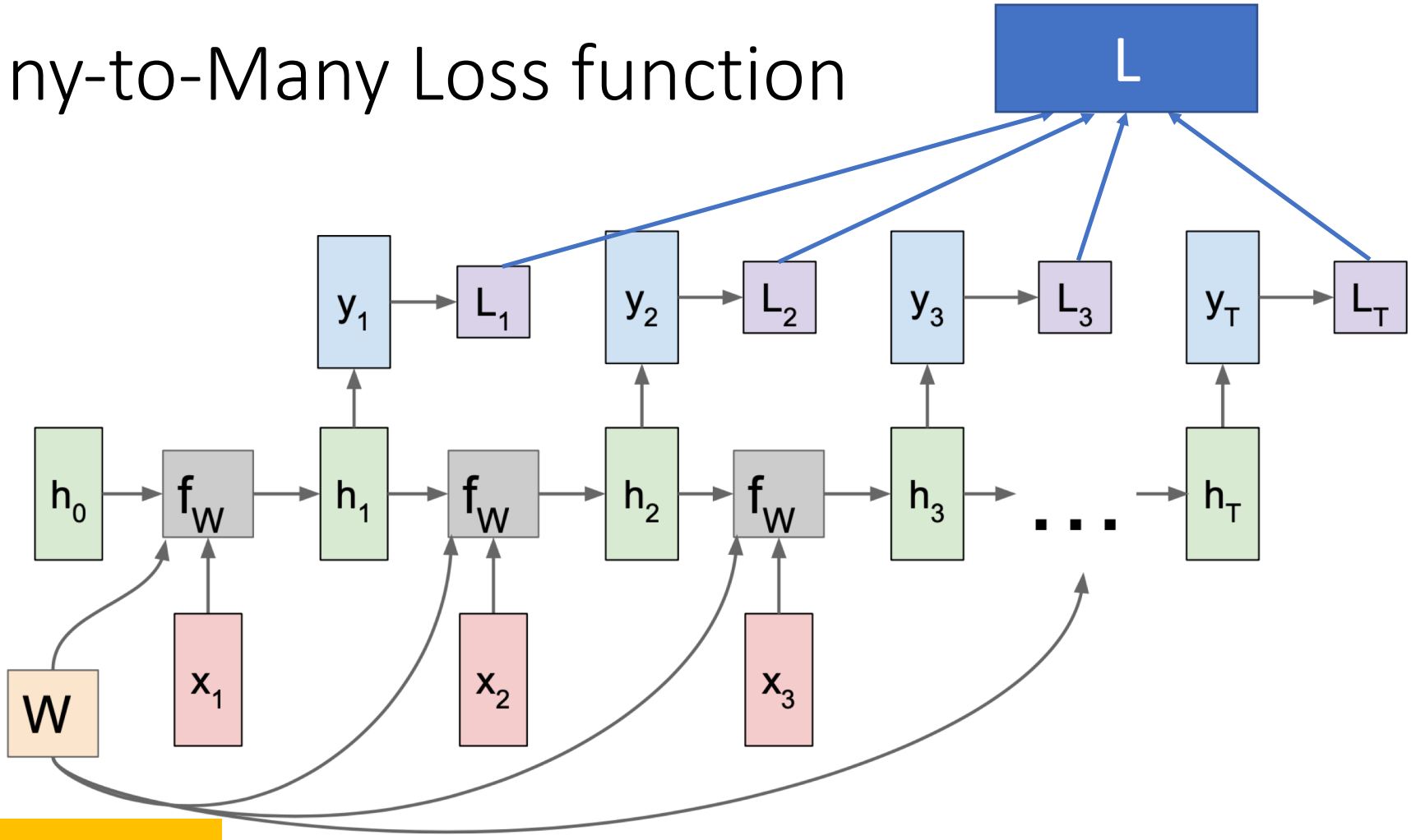
$$h_t = f_W(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$

Taken from: Stanford CS321n

Many-to-Many Loss function



Note: Weights
are shared at each time step

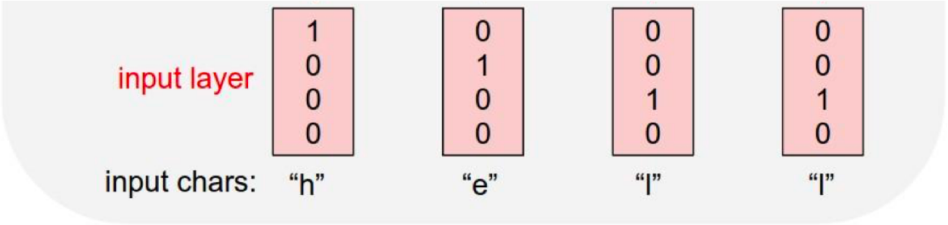
Taken from: Stanford CS321n

Worked Example

Example:
Character-level
Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



Taken from: Stanford CS321n

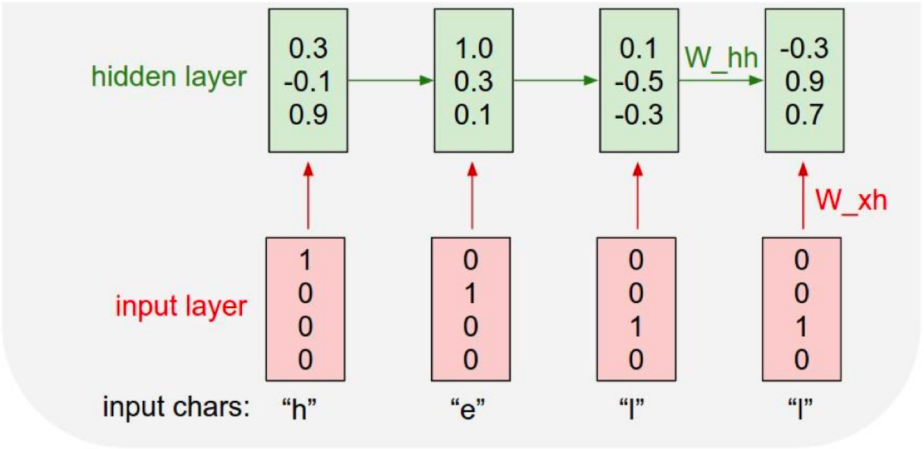
Worked Example

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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



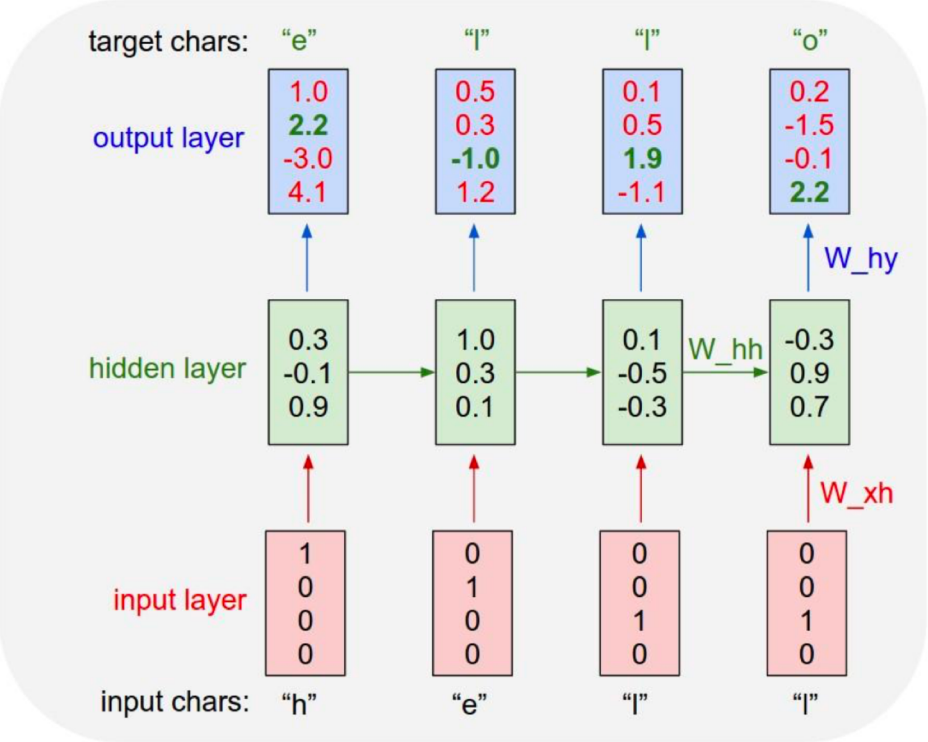
Taken from: Stanford CS321n

Worked Example

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



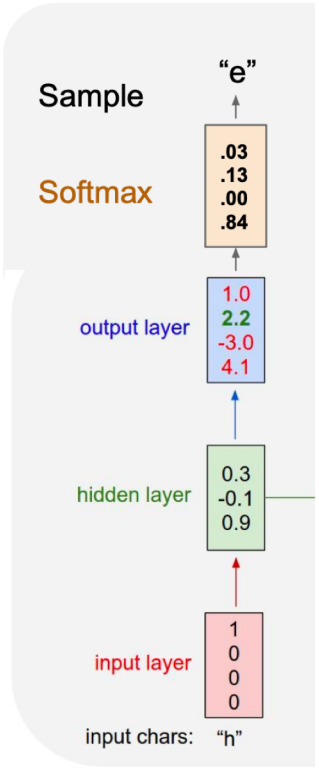
Taken from: Stanford CS321n

Worked Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



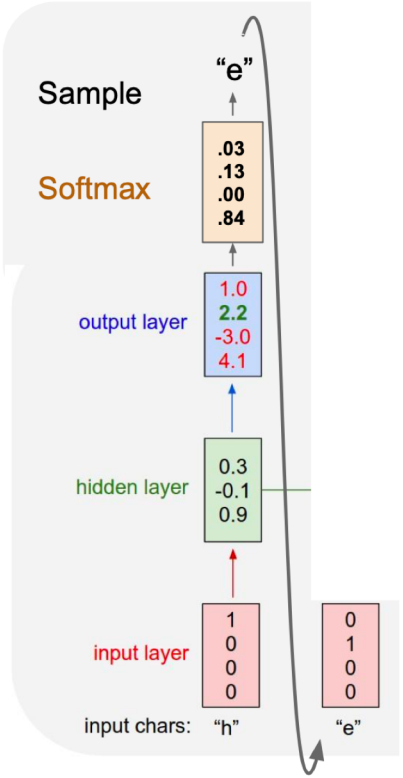
Taken from: Stanford CS321n

Worked Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



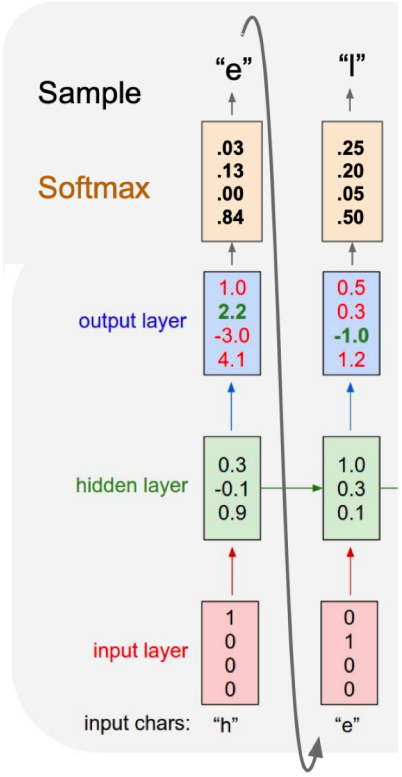
Taken from: Stanford CS321n

Worked Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



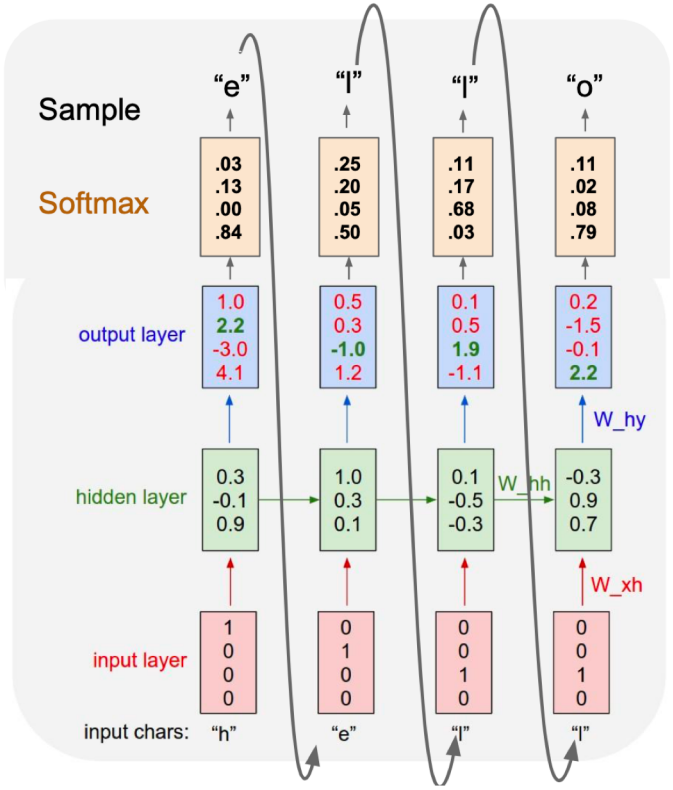
Taken from: Stanford CS321n

Worked Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



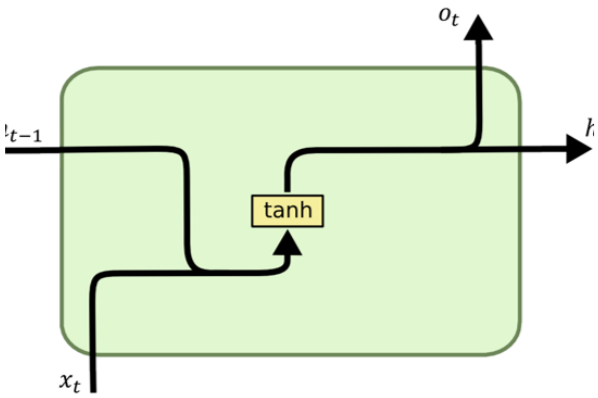
Taken from: Stanford CS321n

Extra details of how RNN works

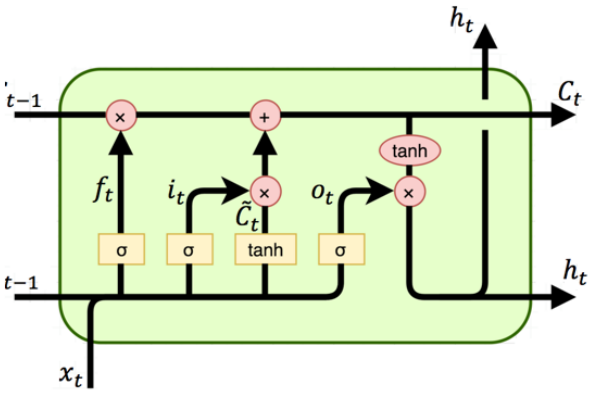
Some parts can be technical, but just focus on the idea

Types of RNN architecture

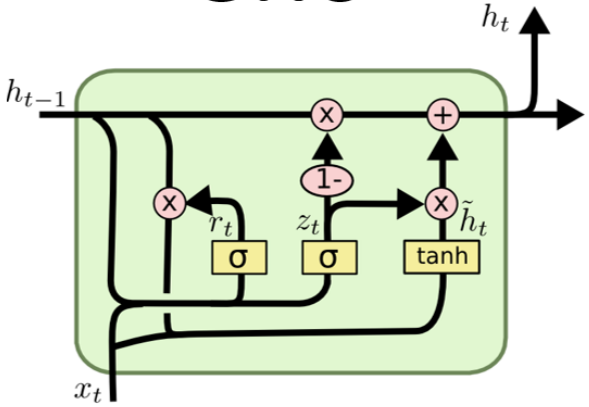
RNN



LSTM



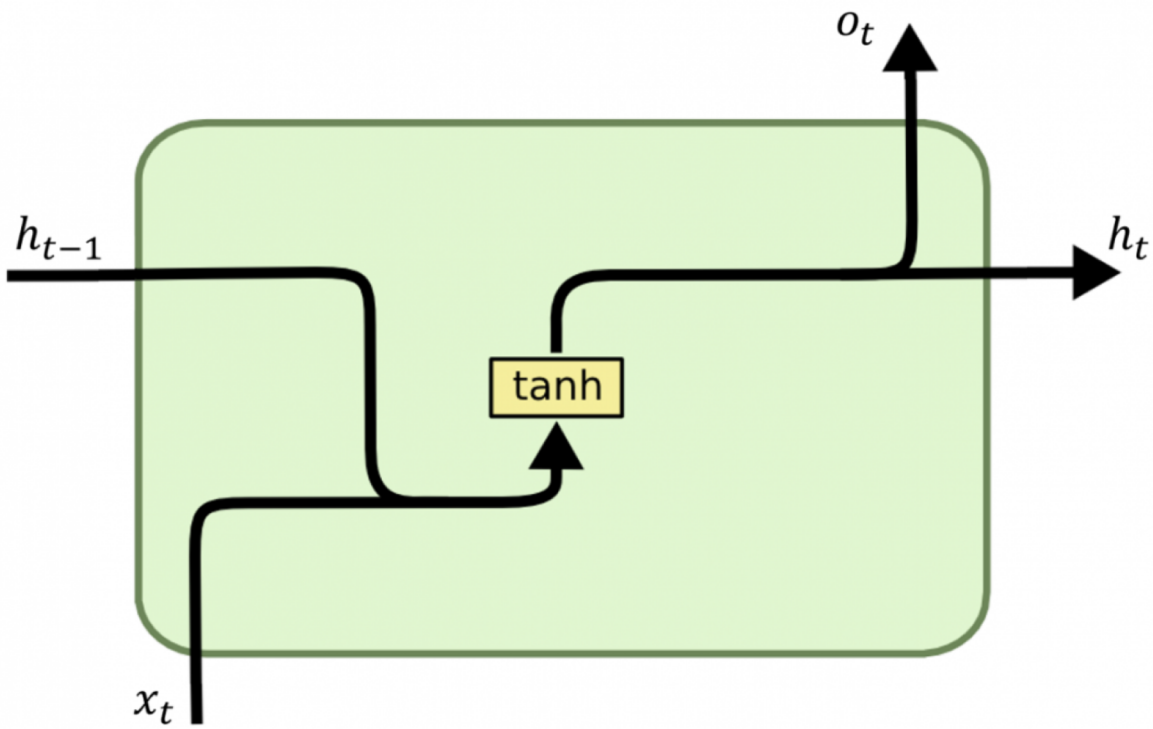
GRU



Has the “Vanishing / Exploding Gradient Problem”

Taken from: http://dprogrammer.org/rnn-lstm-gru

Vanilla RNN



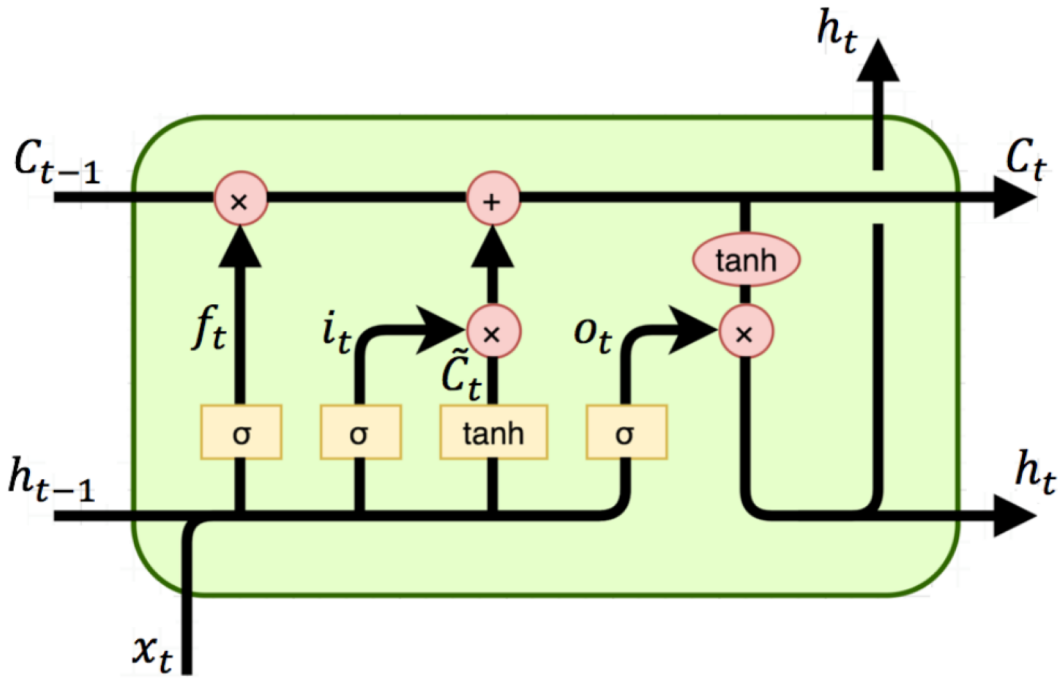
x_t : input vector $(m \times 1)$.
h_t : hidden layer vector $(n \times 1)$.
o_t : output vector $(n \times 1)$.
b_h : bias vector $(n \times 1)$.
U, W : parameter matrices $(n \times m)$.
V : parameter matrix $(n \times n)$.
σ_h, σ_y : activation functions.

$$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_h)$$

Taken from: http://dprogrammer.org/rnn-lstm-gru

Long-Short Term Memory (LSTM)



h_t , C_t : hidden layer vectors.
x_t : input vector.
b_f , b_i , b_c , b_o : bias vector.
W_f , W_i , W_c , W_o : parameter matrices.
σ , tanh : activation functions.

Forget gate $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

Input gate $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

Output gate $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

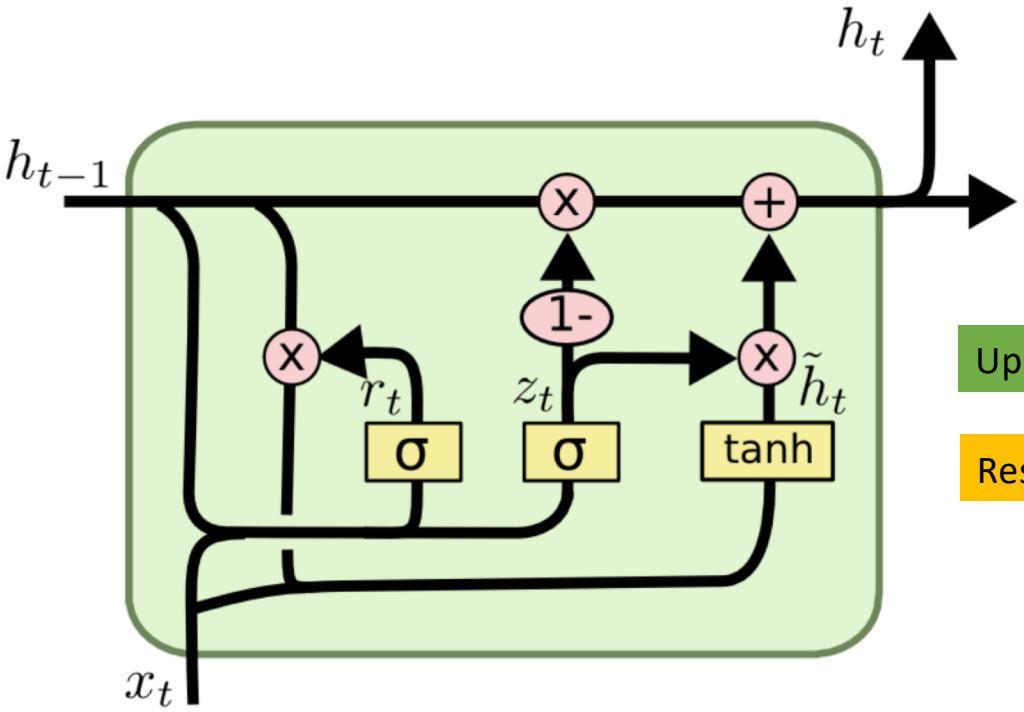
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$

Taken from: http://dprogrammer.org/rnn-lstm-gru

Gated Recurrent Units (GRU)



h_t : hidden layer vectors.
x_t : input vector.
b_z , b_r , b_h : bias vector.
W_z , W_r , W_h : parameter matrices.
σ , tanh : activation functions.

Update gate $z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$

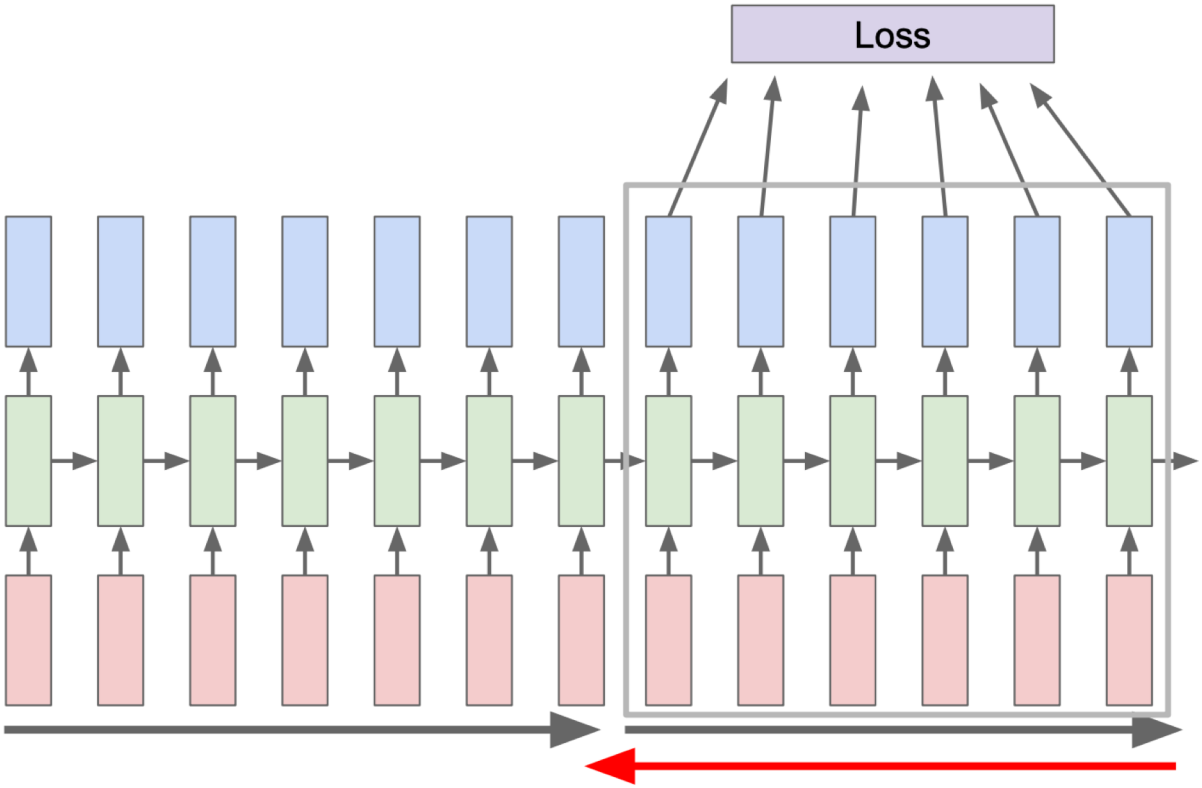
Reset gate $r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Taken from: http://dprogrammer.org/rnn-lstm-gru

How RNNs are trained:
(Truncated) Backpropagation through time



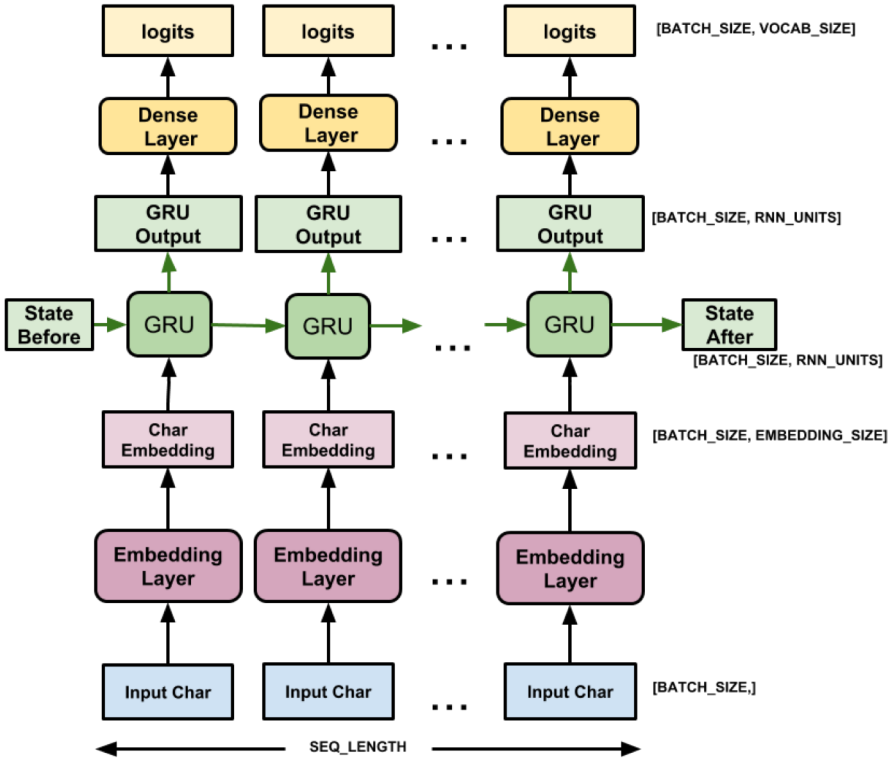
Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

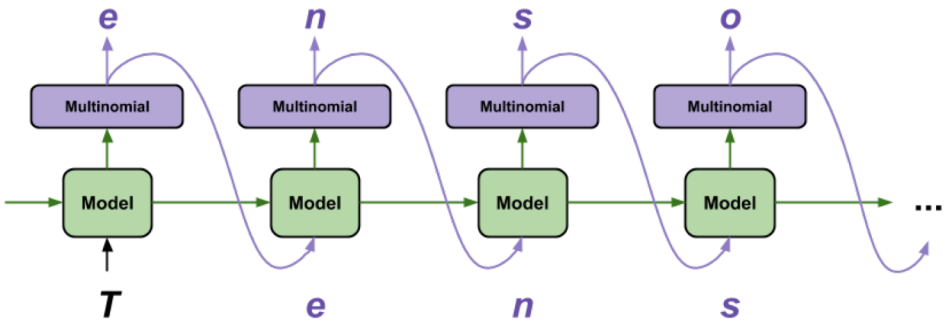
Taken from: Stanford CS321n

Generate Text

With RNN in TensorFlow

Use a GRU to generate Shakespeare Text!





TensorFlow Practice

- Let us run the model!

Questions?