Recurrent Neural Networks (RNNs)

Konda Reddy Mopuri

Recap

- MP neuron, Perceptron, MLP, and CNNs
- In other words, we have seen feedforward neural nets
  - No loops in the computational graphs
  - Input or output is not sequential

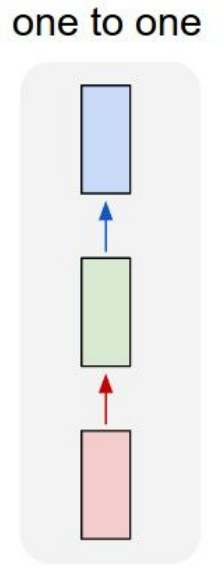


Figure Credit Andrej Karpathy

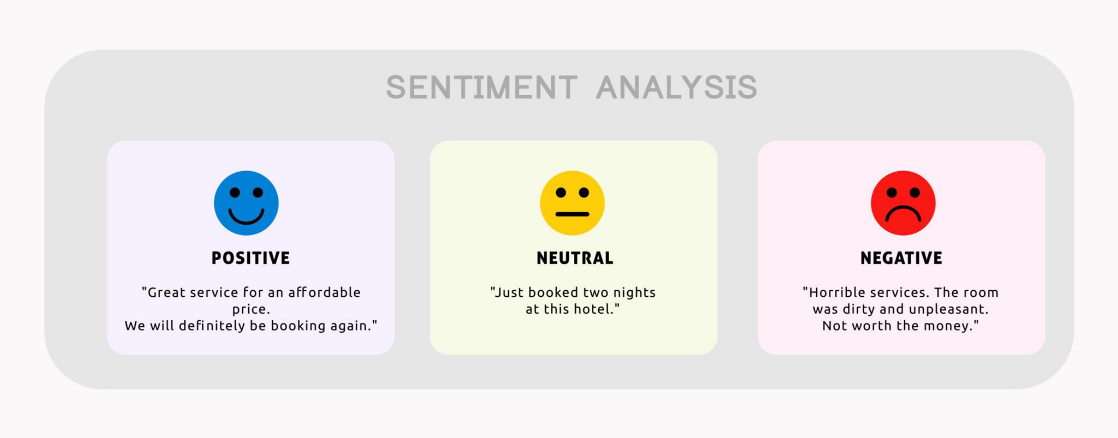
Sequential data

- Many real-world problems have to process data with sequential nature

Sequential data

- Many real-world problems have to process data with sequential nature

  - Sentiment analysis
  - Action recognition
  - DNA sequence classification



Sequence classification

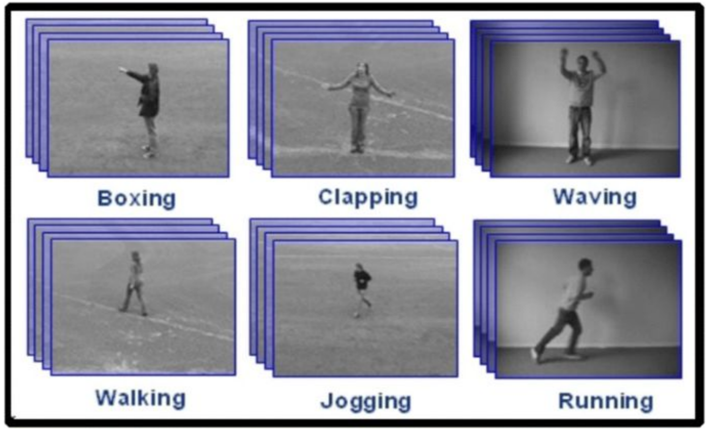
[Source]

Sequential data

- Many real-world problems have to process data with sequential nature

  - Sentiment analysis
  - Action recognition
  - DNA sequence classification

Sequence classification

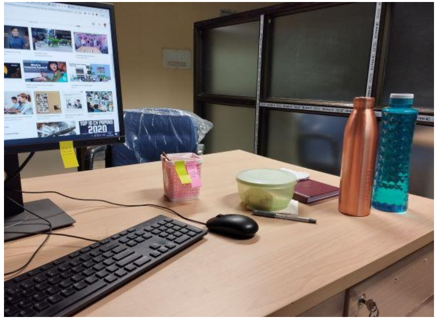


Source

Sequential data

- Many real-world problems have to process data with sequential nature

  - Image Captioning
  - Music/Art generation



Caption generator

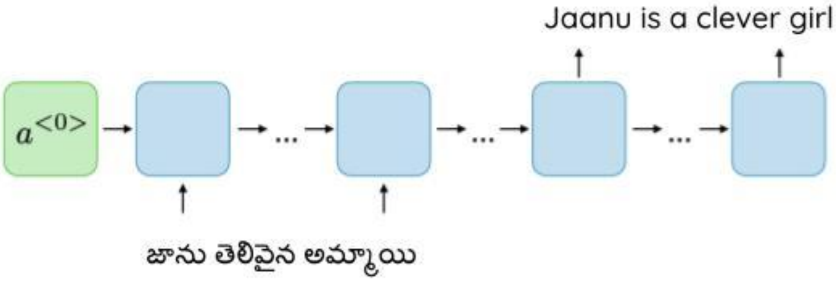
```
a laptop computer sitting on top of a desk
a desktop computer sitting on top of a desk
a laptop computer sitting on top of a wooden desk
a desktop computer sitting on top of a wooden desk
a laptop computer sitting on top of a table
a desk with a keyboard and a monitor
a desk with a laptop and a monitor
a laptop computer sitting on top of a wooden table
a laptop computer sitting on top of a desk next to a desktop computer
a laptop computer sitting on top of a desk next to a computer
```

Sequence Synthesis

Sequential data

- Many real-world problems have to process data with sequential nature

  - Machine translation
  - PoS tagging



Sequence-to-sequence prediction

Formally

Given a set $\mathcal{X}$, and if $S(\mathcal{X})$ is the set of sequences of elements from $\mathcal{X}$

$$S(\mathcal{X}) = \bigcup_{t=1}^{\infty} \mathcal{X}^t$$

Formally

Given a set $\mathcal{X}$, and if $S(\mathcal{X})$ is the set of sequences of elements from $\mathcal{X}$

$$S(\mathcal{X}) = \bigcup_{t=1}^{\infty} \mathcal{X}^t$$

$f : S(\mathcal{X}) \rightarrow \{1, \ldots, C\}$ Sequence classification

$f : \mathcal{R}^D \rightarrow S(\mathcal{X})$ Sequence Synthesis

$f : S(\mathcal{X}) \rightarrow S(\mathcal{Y})$ Sequence-to-sequence prediction

Can we use the known techniques to process the ‘sequential’ data?

Temporal Convolution

Temporal Convolutional Networks (TCN)

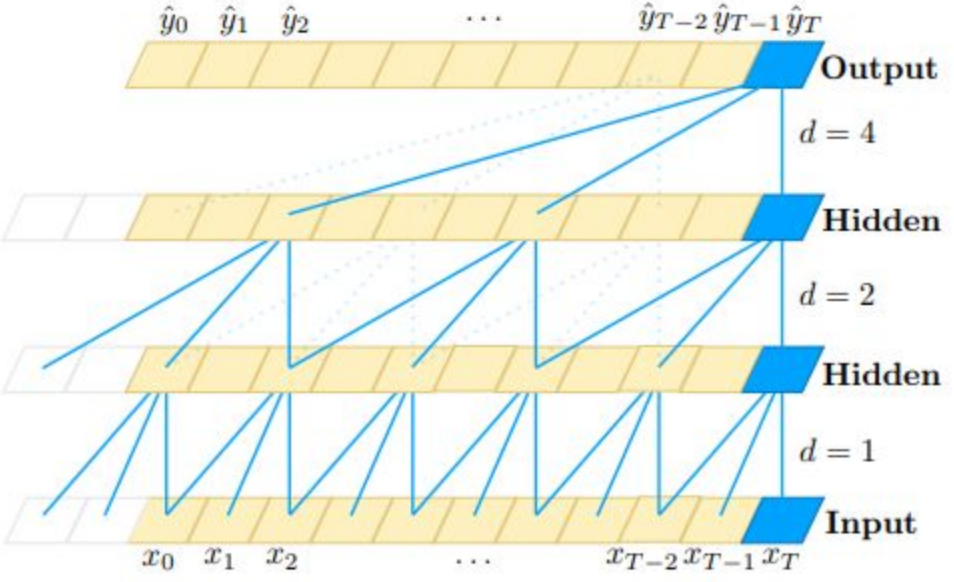
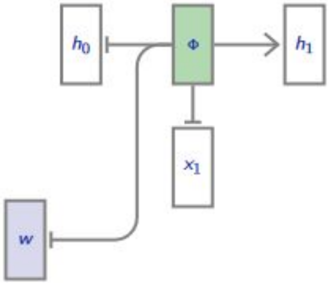


Figure credits: Bai et al.

RNNs and backprop through time

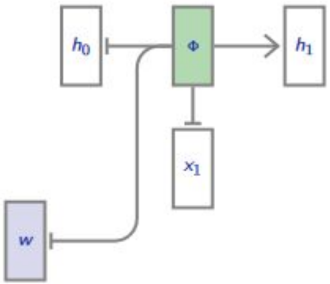
Key is the recurrent state

- Maintains a recurrent state updated at each time step



Key is the recurrent state

- Maintains a recurrent state updated at each time step

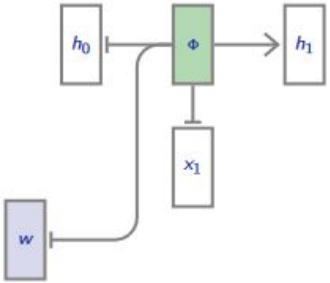


$$\text{With } \mathcal{X} = \mathcal{R}^D, \text{ and}$$

$$\text{given, } \phi(\cdot; w) : \mathcal{R}^D \times \mathcal{R}^Q \to \mathcal{R}^Q,$$

Key is the recurrent state

- Maintains a recurrent state updated at each time step



With $\mathcal{X} = \mathcal{R}^D$, and

given, $\phi(\cdot; w) : \mathcal{R}^D \times \mathcal{R}^Q \rightarrow \mathcal{R}^Q$,

an input sequence $x \in \mathcal{S}(\mathcal{R}^D)$,

an initial recurrent state $h_0 \in \mathcal{R}^Q$,

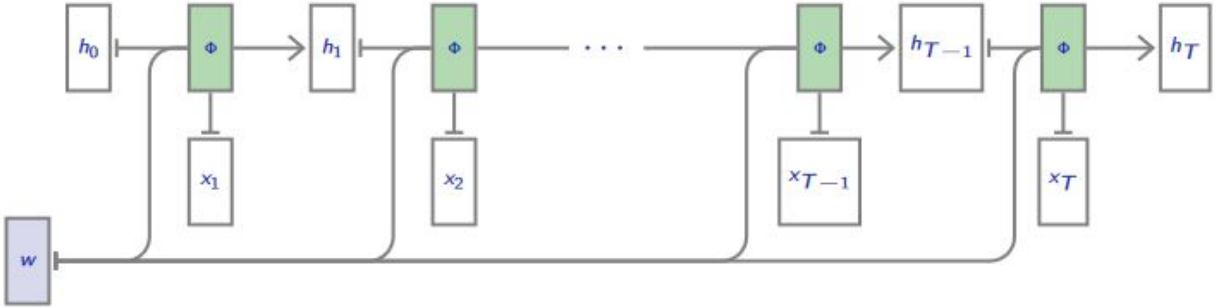
Key is the recurrent state

- Maintains a recurrent state updated at each time step

$$\text{With } \mathcal{X} = \mathcal{R}^D, \text{and}$$
$$\text{given, } \phi(\cdot; w) : \mathcal{R}^D \times \mathcal{R}^Q \rightarrow \mathcal{R}^Q,$$
$$\text{an input sequence } x \in \mathcal{S}(\mathcal{R}^D),$$
$$\text{an initial recurrent state } h_0 \in \mathcal{R}^Q,$$

$$\text{model computes sequence of recurrent states iteratively}$$
$$\forall t = 1, \ldots, T(x), h_t = \phi(x_t, h_{t-1}; w)$$

Recurrence in a graph



State computes the output

- Prediction can be computed at any time step using the recurrent state

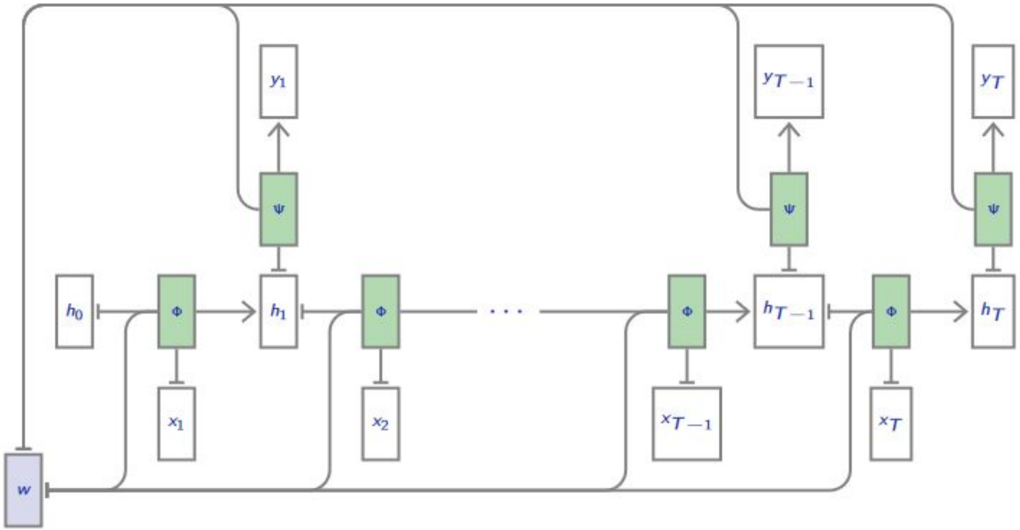
State computes the output

- Prediction can be computed at any time step using the recurrent state

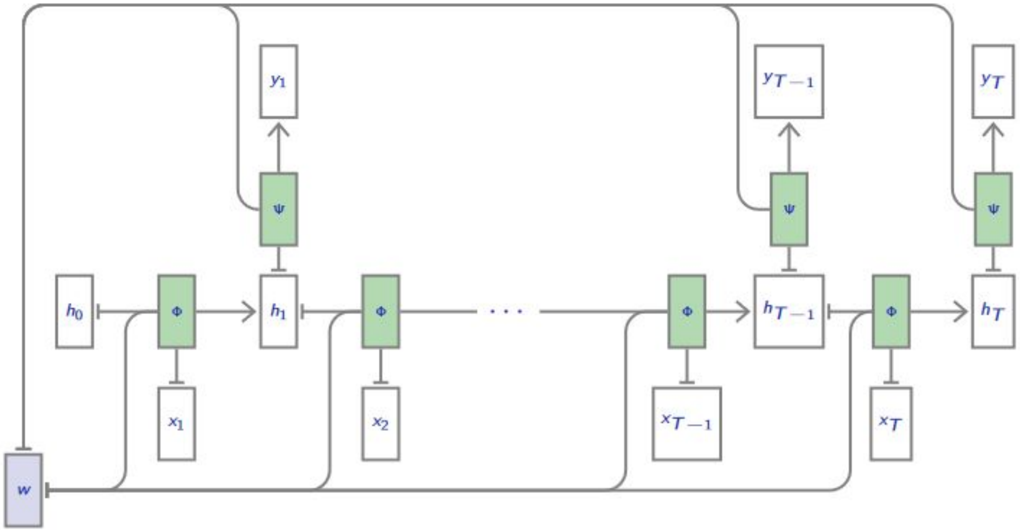
$$y_t = \psi(h_t; w)$$

$$\psi(\cdot; w) : \mathcal{R}^Q \rightarrow \mathcal{R}^C$$

Recurrence in a graph



Backprop in time



Number of steps is equal to the length of sequence T. The rest is similar to the DAGs we know, and autograd can handle.

Different types of RNNs and a sample problem

Elman Network (Elman, 1990)

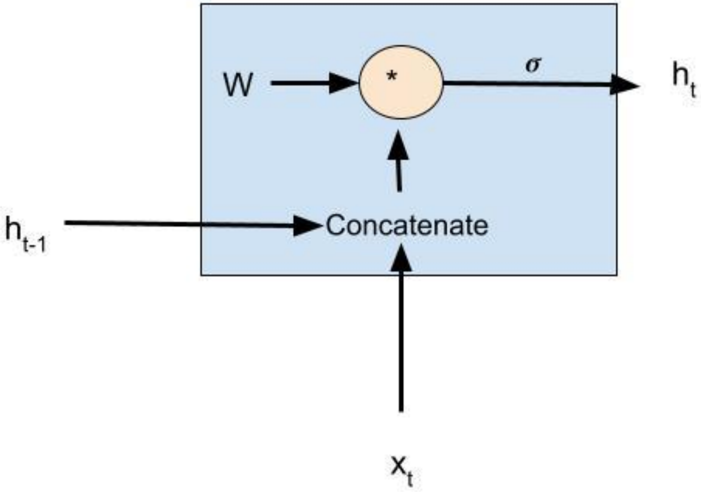
$$h_0 = 0$$

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

$$h_t = tanh(\begin{bmatrix} w_{hh} w_{xh} \end{bmatrix} \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_h)$$

$$y_t = w_{hy}h_t + b_y$$

Elman Network (Elman, 1990)



Sequence classification

Class 1 : sequence is concatenation of two identical halves

Class 0: otherwise

Sequence classification

Class 1 : sequence is concatenation of two identical halves

Class 0: otherwise

$$
\begin{aligned}
x &\rightarrow y \\
(1,2,3,4,5,6) &\rightarrow 0 \\
(3,9,9,3) &\rightarrow 0 \\
(7,4,4,7,5,4) &\rightarrow 0 \\
(7,7) &\rightarrow 1 \\
(1,2,3,1,2,3) &\rightarrow 1 \\
(5,1,1,2,5,1,1,2) &\rightarrow 1
\end{aligned}
$$

Implementation

While unfolding in time

- What is the depth of the model?

While unfolding in time

- What is the depth of the model?
  - Length of the input

While unfolding in time

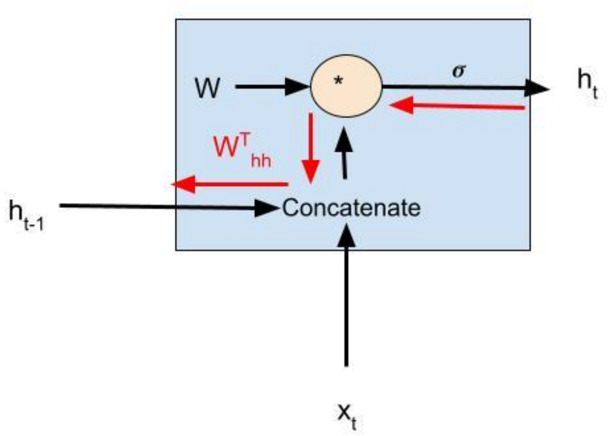
- What is the depth of the model?
    - Length of the input
- → vanishing or exploding gradient issue

$$h_0 = 0$$

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

While unfolding in time

- → vanishing or exploding gradient issue



While unfolding in time

- → vanishing or exploding gradient issue
- Gradient clipping is employed (to handle explosion)
- Introduce a ‘pass-through’
    - recurrent state does not go repeatedly through a squashing nonlinearity

Pass-through

- Recurrent state update can be weighted avg. of previous value and current full update

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

$$\text{where, } \bar{h}_t = \phi(x_t, h_{t-1}) \text{ and}$$
$$\text{weight } z_t = f(x_t, h_{t-1})$$

Pass-through

- Recurrent state update can be weighted avg. of previous value and current full update

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

where, $\bar{h}_t = \phi(x_t, h_{t-1})$ and

weight $z_t = f(x_t, h_{t-1})$

Acts as a 'forget' gate

Gating

- Update equations will now become

$$
\begin{aligned}
h_0 &= 0 \\
\bar{h}_t &= \tanh\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right) \text{ (full update)} \\
z_t &= sigm(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \text{ (forget gate)} \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t \text{ (recurrent state)} \\
y_t &= W_{hy}h_t + b_y \text{ (output)}
\end{aligned}
$$

LSTM

Work to do!

- Improve the sample problem with the updated model

Long-Short Term Memory unit

- Hochreiter and Schmidhuber (1997)
- Later improved by a forget gate (Gers, et al 2000)

Long-Short Term Memory unit

- Hochreiter and Schmidhuber (1997)
- Later improved by a forget gate (Gers, et al 2000)

It uses the structure founded on the short-term processes to create a long-term memory

Long-Short Term Memory unit

- Consists of a cell state (c_t) and an output state (h_t)
- Gates
  - f_t - if the cell state should be forgotten
  - i_t - if the new update should be taken into account
  - o_f - if the output state should be reset

Long-Short Term Memory unit

- Consists of a cell state (c_t) and an output state (h_t)
- Gates
  - f_t - if the cell state should be forgotten
  - i_t - if the new update should be taken into account
  - o_f - if the output state should be reset

$$f_t = sigm(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$i_t = sigm(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$o_t = sigm(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

$$g_t = tanh(W_{xc}X_t + W_{hc}h_{t-1} + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot tanh(c_t)$$

Long-Short Term Memory unit

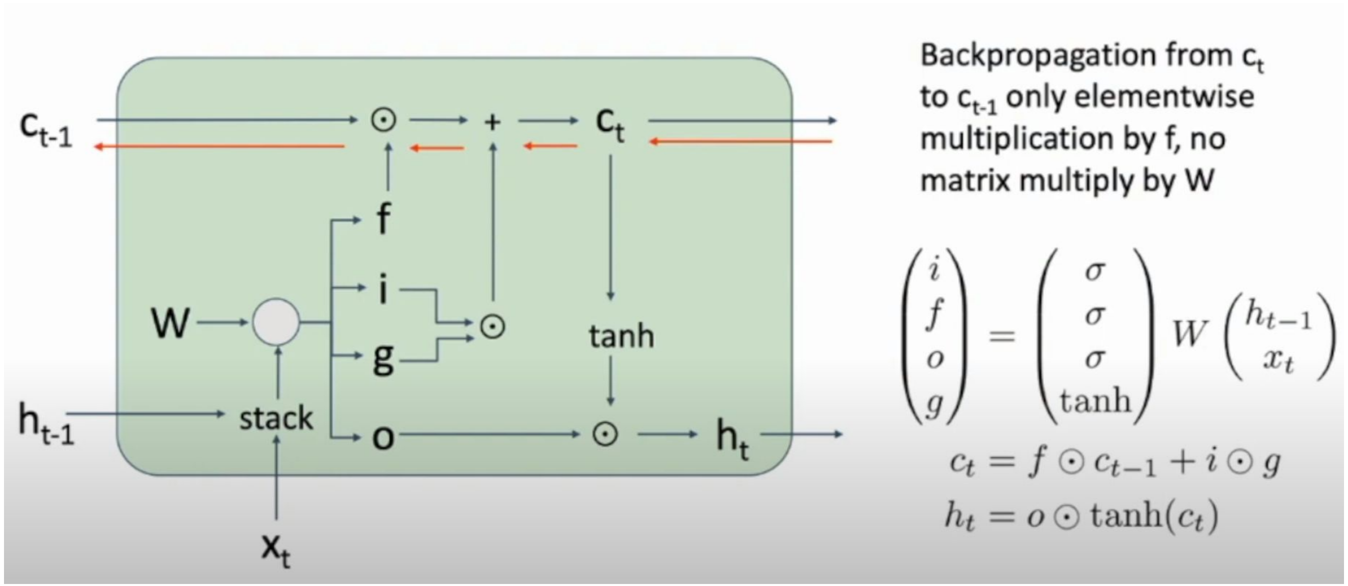
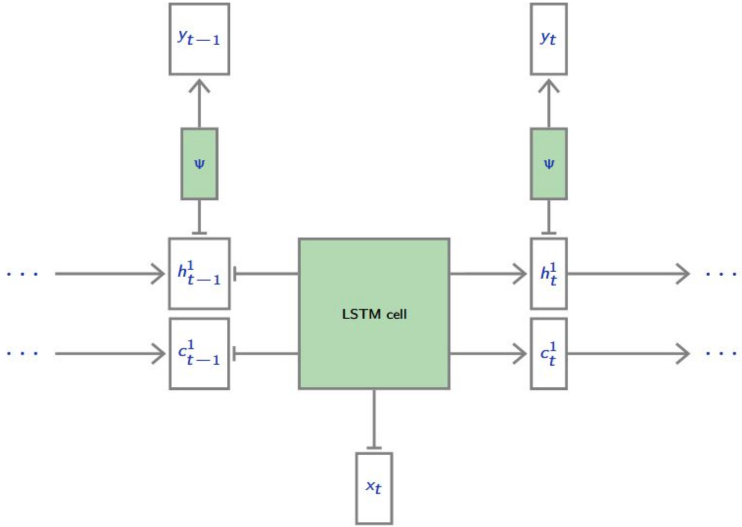
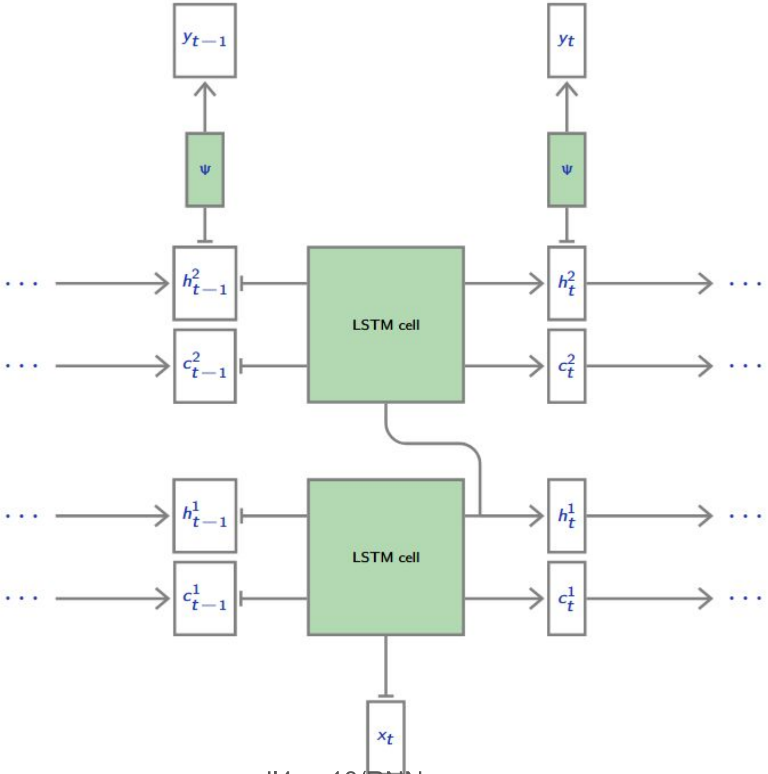


Figure credits: Dr Justin Johnson, U Michigan

LSTM unit



LSTM layers



y_{t-1}
y_t
Ψ
Ψ
h^2_{t-1}
c^2_{t-1}
LSTM cell
h^2_t
c^2_t
h^1_{t-1}
c^1_{t-1}
LSTM cell
h^1_t
c^1_t
x_t

torch.nn.LSTM

- Layers D
- Processes sequence of length T and outputs
- Outputs for all the layers at the last time step T: $h_T^1, h_T^2, \ldots.\ h_T^D$
- Outputs for the last layer at all the time steps: $h_1^D, h_2^D, \ldots.\ h_T^D$

Try LSTM on the toy task

```
class LSTMNet(nn.Module):
  def __init__(self, dim_input, dim_recurrent, num_layers, dim_output):
      super().__init__()
      self.lstm = nn.LSTM(input_size = dim_input,hidden_size = dim_recurrent,num_layers = 
num_layers)
      self.fc_o2y = nn.Linear(dim_recurrent, dim_output)
  def forward(self, input):
      # Get the last layer's last time step activation
      output, _ = self.lstm(input.permute(1, 0, 2))
      output = output[-1]
      return self.fc_o2y(F.relu(output))
```

Gated Recurrent Unit (GRU)

- LSTM was simplified by Cho et al. (2014)
- Has a gating for recurrent state
- Also has a reset gate

Gated Recurrent Unit (GRU)

$$r_t = sigm(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \quad \text{(reset gate)}$$

$$z_t = sigm(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \quad \text{(forget gate)}$$

$$\bar{h}_t = tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \quad \text{(full update)}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t \quad \text{(hidden update)}$$

Different sequence tasks

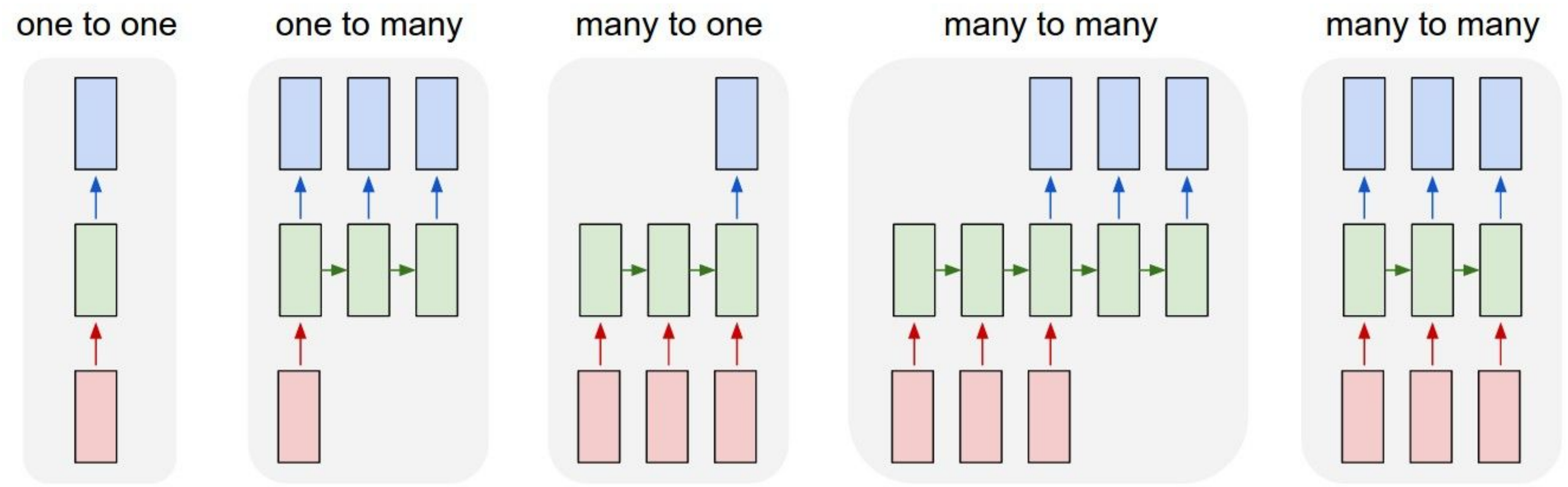
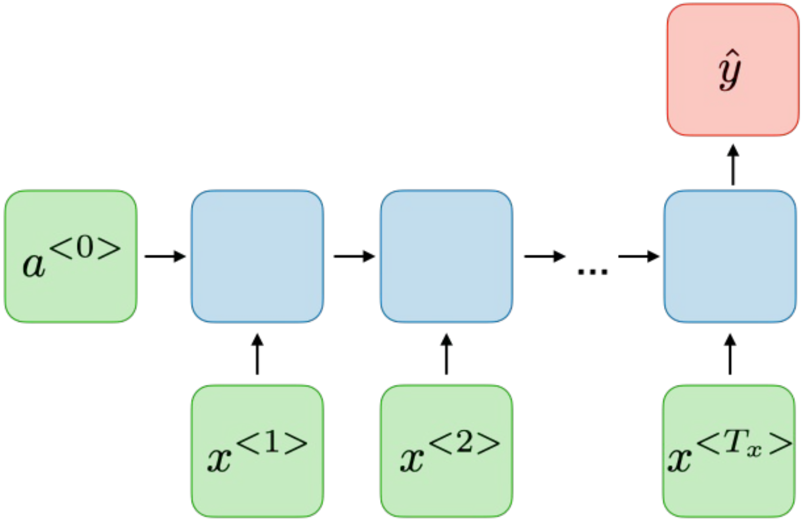


Figure Credit Andrej Karpathy

Many-to-One



Sentiment classification, etc.

Figure Credit CS321N, Stanford

One-to-Many

Music generation, image captioning, etc.

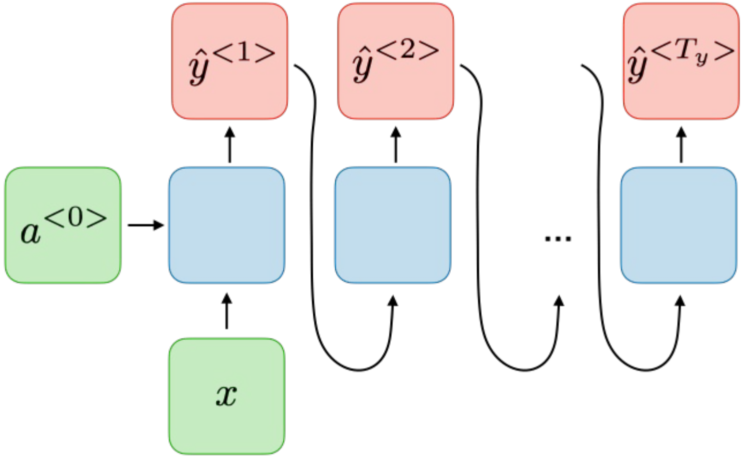
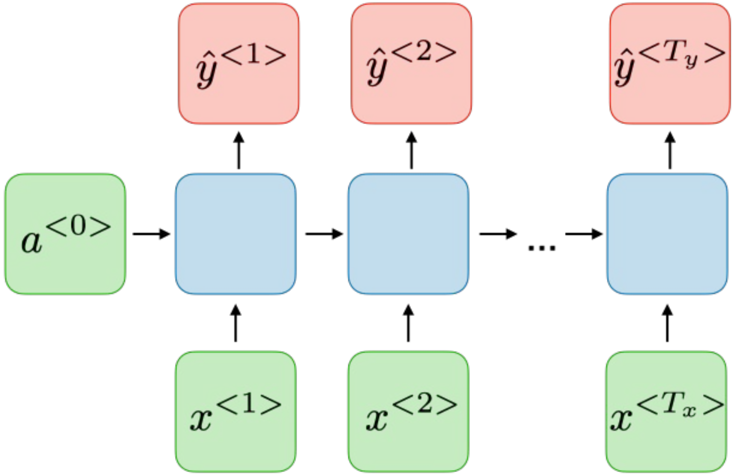


Figure Credit CS321N, Stanford

Many-to-Many



PoS tagging, etc.

Figure Credit CS321N, Stanford

Many-to-Many

Machine Translation, etc.

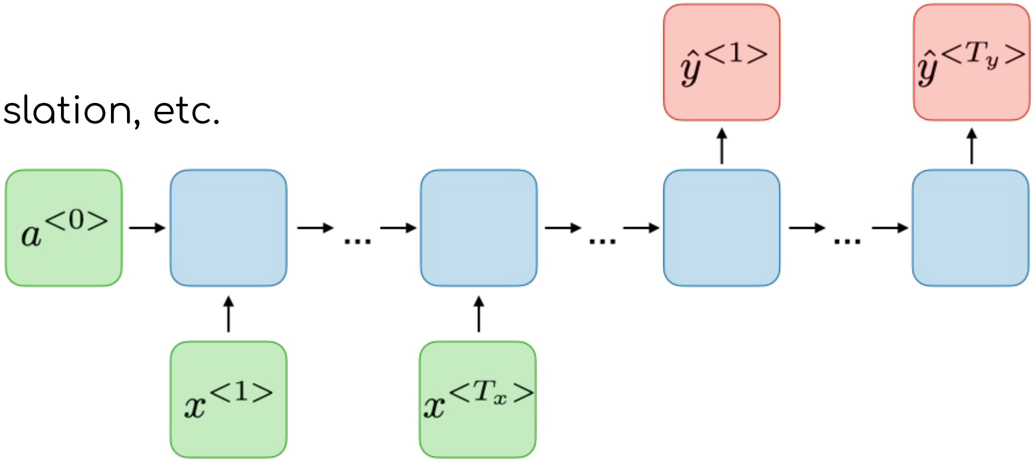


Figure Credit CS321N, Stanford

Thank You