Faculty of Sciences of Tetouan

Introduction to Deep Learning

7- Architectures (Part 2): Recurrent Neural Networks (RNNs)

Prof. Monir EL ANNAS

Networks for sequential data

Sequential Data:

- Most of data are sequential: Speech, Text, Image, ...
- There are 3 problem types: Predict the next step, classify, and generate a sequence

Problems with feedforward neural networks:

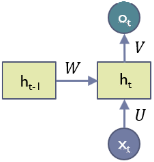
- They assume that each data point is independent and non-sequential.
- They lack the capability to retain any memory of previous inputs, treating each new input in isolation.

Deep Learnings for Sequential Data:

- Convolutional Neural Networks : Try to find local features from a sequence
- Recurrent Neural Networks : Try to capture the feature of the past

Recurrent neural networks

Recurrent neural networks: Structure



$$h_t = f(Ux_t + Wh_{t-1})$$
$$o_t = g(Vh_t)$$

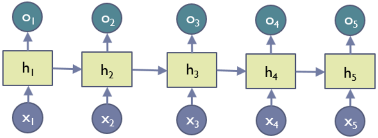
X_t: t^{th} input (sequential: $X_1, X_2, \ldots, X_t$)

h_t: t^{th} hidden state

f: activation function

U, V, W: Network parameters

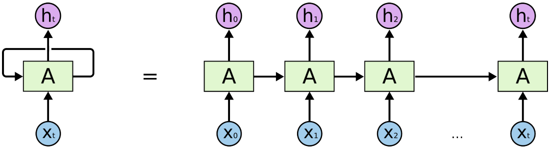
g: Activation function of the output layer

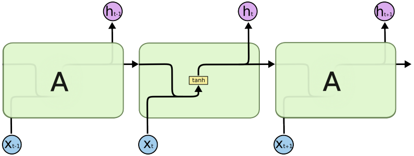


Recurrent neural networks

Recurrent neural networks: Structure

- **Sequential Data Processing:** Each network pass handles one time step of input, influencing the next.
- **Shared Parameters:** RNNs reuse the same weights at each step.
- **Backpropagation Through Time:** Unfolding the network through each time step, allow gradient calculations over sequences.



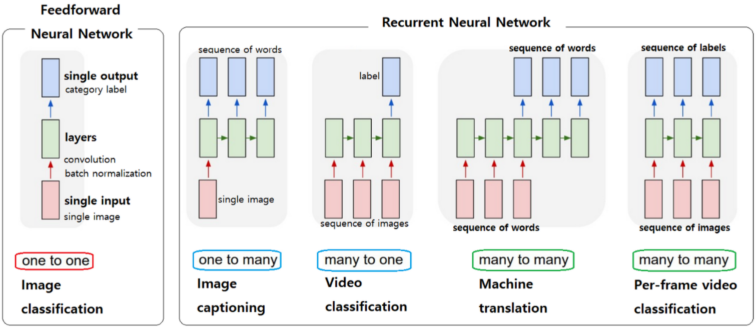


$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

Recurrent neural networks

Recurrent neural networks: Different combinations

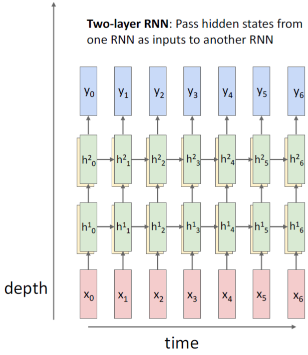
- Different applications will give rise to different ways in which we use RNNs



Recurrent neural networks

Recurrent Neural Networks: Multi-layer RNN

- Several layers of RNN units (cells) can be stacked vertically. Each layer processes the sequence output from the previous layer.
- The layers usually use different weight matrices.



Recurrent neural networks

Recurrent Neural Networks: PyTorch

CLASS torch.nn.RNN(*self*, *input_size*, *hidden_size*, *num_layers=1*, *nonlinearity='tanh'*, *bias=True*, *batch_first=False*, *dropout=0.0*, *bidirectional=False*, *device=None*, *dtype=None*) [SOURCE]

Apply a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence. For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

where h_t is the hidden state at time *t*, x_t is the input at time *t*, and $h_{(t-1)}$ is the hidden state of the previous layer at time *t-1* or the initial hidden state at time *0*. If `nonlinearity` is `'relu'`, then ReLU is used instead of $\tanh$.

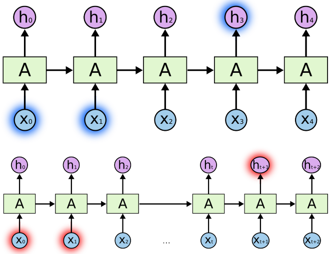
Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

Recurrent neural networks

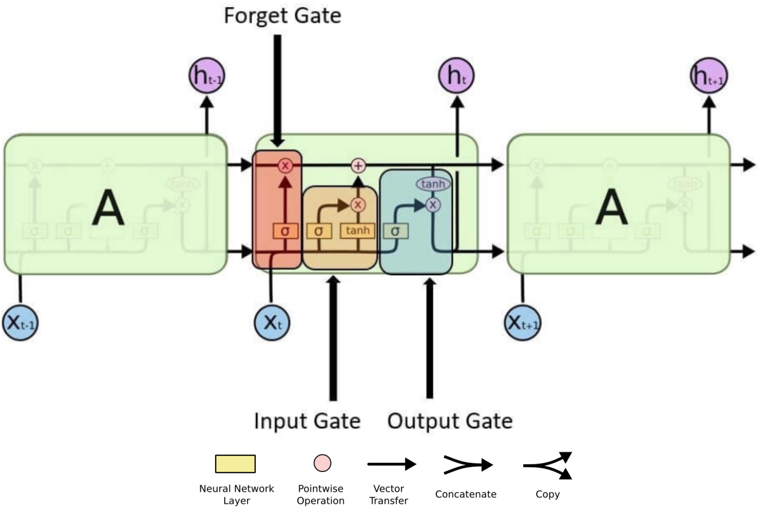
Recurrent Neural Networks: Problem

- Long-term dependency problem :
    - Must remember all states at any given time
    - Computationally expensive
    - Only store states within a time window
- Sensitive to changes in their parameters
- Vanishing Gradient
- Exploding Gradient



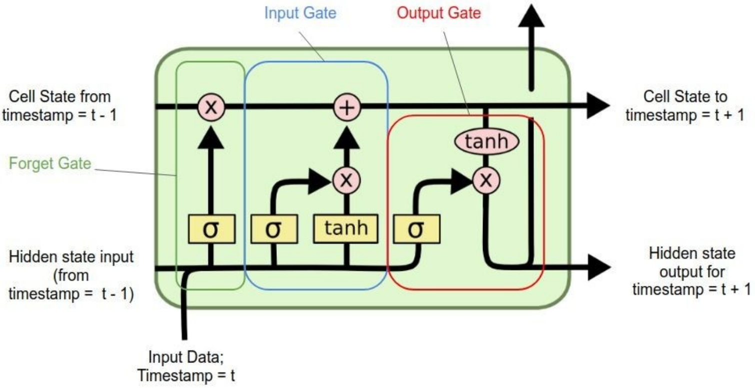
RNNs variants

Long Short-Term Memory (LSTM)



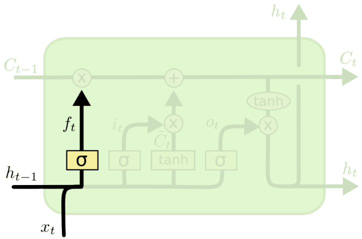
RNNs variants

Long Short-Term Memory (LSTM)



RNNs variants

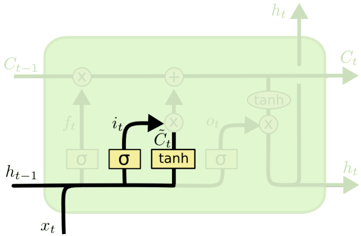
Long Short-Term Memory (LSTM)



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

RNNs variants

Long Short-Term Memory (LSTM)

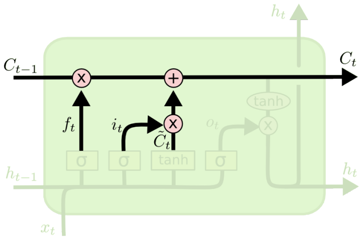


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

RNNs variants

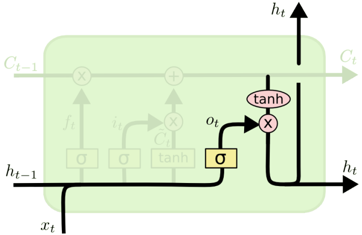
Long Short-Term Memory (LSTM)



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

RNNs variants

Long Short-Term Memory (LSTM)

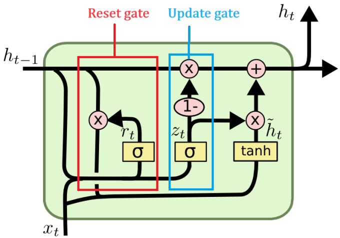


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

RNNs variants

Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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